**ECE 661 Homework 9**

**Weichen Xu**

**xu1363@purdue.edu**

**Theory Question**

No, you cannot see in a camera image, because the pixels of the absolute conic are imaginary. Since is positive definite, but the conic , cannot be positive definite due to its definition .

Since is only decided by the intrinsic parameters of the camera, with. So, we can estimate which contains the intrinsic parameters from corner correspondences. In camera calibration, intrinsic parameters are totally decided by the characteristics of the camera, comparing to the extrinsic parameters which are decided by the rotation, and translation.

**Corner Detection**

The first step is to convert the color image to the grayscale image, then we can implement the Canny detection algorithm with OpenCV library, here the threshold is obtained by trial and error is 100 and 500. After obtaining the image’s edges, we can implement the Hough line also with OpenCV library to fit the edges with straight lines. It is noted that typically implementing Hough line will return more straight lines fitting the edges than expected number of lines.

The next step is to filter all returned Hough lines. For given images, the desired numbers for straight lines are 10 for horizontal lines and 8 for vertical lines. For lines with the absolute degree no more than are included in the horizontal lines, otherwise vertical lines. Based on this ,we can implement the filtering method for valid lines. Starting from the top lines in vertical direction (most left lines in horizontal direction), search following lines, include it to the candidate lines only if the distances between this line and all other lines in candidate lines are no smaller than certain preset threshold. If the returned candidate lines’ total number is not desired number, then change the threshold for minimal line distance and run above process again until the final number for this direction is as desired. It has been tested such implementation can result in lines of relatively high accuracy.

Now we have obtained 10 horizontal lines labeled from up to down sand 8 vertical lines labeled from left to right. Thus, we can get the intersection points from each horizontal line and vertical line, resulting 80 intersections in total, the left top one is labeled with 1, while the right bottom is labeled with 80.

**Experiment Results**

A close up of a keyboard

Description automatically generated

Figure 1: Canny edge detection of Pic\_1.jpg in Dataset1

A picture containing letter

Description automatically generated

Figure 2: Filtered Hough lines of Pic\_1.jpg in Dataset1

Calendar

Description automatically generated

Figure 3: Detected corners of Pic\_1.jpg in Dataset1

A close up of a logo

Description automatically generated

Figure 4: Canny edge detection of Pic\_4.jpg in Dataset1

Diagram

Description automatically generated

Figure 5: Filtered Hough lines of Pic\_4.jpg in Dataset1

A picture containing calendar

Description automatically generated

Figure 6: Detected corners of Pic\_4.jpg in Dataset1

**Code**

*#!/usr/bin/env python  
# coding: utf-8*import numpy as np  
import cv2  
import os  
import matplotlib.pyplot as plt  
  
  
def select\_line\_hor(lines):  
 *# sort all horizontal lines and select lines with mutual distance larger than threshold* threshold = 15  
 N\_goal = 10  
 N\_lines = 0  
 while N\_lines != N\_goal:  
 candidates = []  
 candidates.append(lines[0])  
 for i in range(1, len(lines)):  
 rho = lines[i][0]  
 theta = lines[i][1]  
 L = rho \* np.sin(theta)  
 Ls = []  
   
 for j in range(len(candidates)):  
 rhos = candidates[j][0]  
 thetas = candidates[j][1]  
 Ls.append(rhos \* np.sin(thetas))   
   
 *# obtain the distance with all other candidates* distance = np.min(abs(Ls - L))  
 if distance > threshold:  
 candidates.append(lines[i])  
   
 N\_lines = len(candidates)  
   
 *# change threshold value when returned number doesn't match* if N\_lines > N\_goal:  
 threshold += 0.5  
 if N\_lines < N\_goal:  
 threshold -= 0.5  
  
 return candidates  
  
def select\_line\_ver(lines):  
 *# sort all vertical lines and select lines with mutual distance larger than threshold* threshold = 15  
 N\_goal = 8  
 N\_lines = 0  
 while N\_lines != N\_goal:  
 candidates = []  
 candidates.append(lines[0])  
 for i in range(1, len(lines)):  
 rho = lines[i][0]  
 theta = lines[i][1]  
 L = rho \* np.cos(theta)  
  
 Ls = []  
 for j in range(len(candidates)):  
 rhos = candidates[j][0]  
 thetas = candidates[j][1]  
 Ls.append(rhos \* np.cos(thetas))   
  
 distance = np.min(abs(Ls - L))  
 if distance > threshold:  
 candidates.append(lines[i])  
   
 N\_lines = len(candidates)  
 if N\_lines > N\_goal:  
 threshold += 0.5  
 if N\_lines < N\_goal:  
 threshold -= 0.5  
   
 return candidates  
  
  
  
  
def get\_intersection\_from\_lines(img, lines):  
 img\_plot = img.copy()  
 N = lines.shape[0]  
 thetas = lines[:,1].copy()  
 thetas -= np.pi/2  
 idx\_hor = []  
 idx\_ver = []  
   
 *# horizontal line with angle in (-pi/4, pi/4)  
 # vertical line else* for i in range(N):  
 if abs(thetas[i]) < np.pi/4:  
 idx\_hor.append(i)  
 else:  
 idx\_ver.append(i)  
   
 lines\_hor = lines[idx\_hor,:]  
 lines\_ver = lines[idx\_ver,:]  
   
 *# sort horizontal lines from top to bottom, vertical lines from left to right* lines\_hor = sorted(lines\_hor, key = lambda x: x[0]\*np.sin(x[1]), reverse = False)  
 lines\_ver = sorted(lines\_ver, key = lambda x: x[0]\*np.cos(x[1]), reverse = False)  
   
 lines\_hor = np.array(select\_line\_hor(lines\_hor))  
 lines\_ver = np.array(select\_line\_ver(lines\_ver))  
   
   
 *# get the intersection point coordiantes* intersections = np.zeros((10, 8, 2))  
 for i in range(10):  
 for j in range(8):  
 line\_hor = get\_HC(lines\_hor[i])  
 line\_ver = get\_HC(lines\_ver[j])  
 intersection = np.cross(line\_hor, line\_ver)  
 intersection = intersection / intersection[2]  
 intersections[i,j,:] = [intersection[0], intersection[1]]   
   
 return intersections  
  
  
  
  
def get\_HC(line):  
 rho = line[0]  
 theta = line[1]  
   
 pt1 = np.array([rho\*np.cos(theta), rho\*np.sin(theta), 1])  
 pt2 = np.array([rho\*np.cos(theta) + np.sin(theta), rho\*np.sin(theta) - np.cos(theta), 1])  
   
 HC = np.cross(pt1, pt2)  
 HC = HC / HC[2]  
  
 return HC  
  
  
  
  
def plot\_edges(img, corners, name):  
 cv2.imwrite(**'/home/xu1363/Documents/ECE 661/hw9/Dataset1\_plot/'** + **'edges\_'** + name, corners)  
  
def plot\_intersections(img, intersections, name):  
 img\_plot = img.copy()  
 num = 0  
 for i in range(10):  
 for j in range(8):  
 num += 1  
 loc = tuple([int(intersections[i,j,0]), int(intersections[i,j,1])])  
 cv2.circle(img\_plot, loc, 2, (0,0,255), 2)  
 cv2.putText(img\_plot, str(num),loc, cv2.FONT\_HERSHEY\_COMPLEX, 0.5, (0,0,255), 1)  
   
 *#plt.imshow(cv2.cvtColor(img\_plot, cv2.COLOR\_BGR2RGB))  
 #plt.show()* cv2.imwrite(**'/home/xu1363/Documents/ECE 661/hw9/Dataset1\_plot/'** + **'points\_'** + name, img\_plot)  
  
   
def plot\_lines(img, lines, name):  
 img\_plot = img.copy()  
   
 N = lines.shape[0]  
 thetas = lines[:,1].copy()  
 thetas -= np.pi/2  
 idx\_hor = []  
 idx\_ver = []  
 for i in range(N):  
 if abs(thetas[i]) < np.pi/4:  
 idx\_hor.append(i)  
 else:  
 idx\_ver.append(i)  
   
 lines\_hor = lines[idx\_hor,:]  
 lines\_ver = lines[idx\_ver,:]  
 lines\_hor = sorted(lines\_hor, key = lambda x: x[0]\*np.sin(x[1]),   
 reverse = False)  
 lines\_ver = sorted(lines\_ver, key = lambda x: x[0]\*np.cos(x[1]),   
 reverse = False)  
   
 lines\_hor = np.array(select\_line\_hor(lines\_hor))  
 lines\_ver = np.array(select\_line\_ver(lines\_ver))  
   
 for rho,theta in lines\_hor:  
 a = np.cos(theta)  
 b = np.sin(theta)  
   
 x0 = a\*rho  
 y0 = b\*rho  
 x1 = int(x0 + 1000\*(-b))  
 y1 = int(y0 + 1000\*(a))  
 x2 = int(x0 - 1000\*(-b))  
 y2 = int(y0 - 1000\*(a))  
 cv2.line(img\_plot,(x1,y1),(x2,y2),(0,0,255),2)  
   
 for rho,theta in lines\_ver:  
 a = np.cos(theta)  
 b = np.sin(theta)  
   
 x0 = a\*rho  
 y0 = b\*rho  
 x1 = int(x0 + 1000\*(-b))  
 y1 = int(y0 + 1000\*(a))  
 x2 = int(x0 - 1000\*(-b))  
 y2 = int(y0 - 1000\*(a))  
 cv2.line(img\_plot,(x1,y1),(x2,y2),(0,0,255),2)  
  
   
 *#plt.imshow(cv2.cvtColor(img\_plot, cv2.COLOR\_BGR2RGB))  
 #plt.show()* cv2.imwrite(**'/home/xu1363/Documents/ECE 661/hw9/Dataset1\_plot/'** + **'lines\_'** + name, img\_plot)  
  
  
  
  
path = **'/home/xu1363/Documents/ECE 661/hw9/Dataset1/'**for name in os.listdir(path):  
 img = cv2.imread(path + name)  
 img\_plot = img.copy()  
 corners = cv2.Canny(img, 100, 500, apertureSize=3)  
 lines = cv2.HoughLines(corners, 1, np.pi/180, 40)  
 lines = lines.reshape(lines.shape[0], lines.shape[-1])  
 for rho,theta in lines:  
 a = np.cos(theta)  
 b = np.sin(theta)  
 x0 = a\*rho  
 y0 = b\*rho  
 x1 = int(x0 + 1000\*(-b))  
 y1 = int(y0 + 1000\*(a))  
 x2 = int(x0 - 1000\*(-b))  
 y2 = int(y0 - 1000\*(a))  
 cv2.line(img\_plot,(x1,y1),(x2,y2),(0,0,255),2)  
   
  
 intersections = get\_intersection\_from\_lines(img, lines)  
 plot\_edges(img, corners, name)  
 plot\_intersections(img, intersections, name)  
 plot\_lines(img, lines, name)