ECE 661 Homework 9

Weichen Xu

xu1363@purdue.edu

Theory Question

No, you cannot see ω in a camera image, because the pixels of the absolute conic are imaginary. Since $\omega = K^{-T}K^{-1}$ is positive definite, but the conic C, cannot be positive definite due to its definition $\vec{x}^T C\vec{x} = 0$.

Since ω is only decided by the intrinsic parameters of the camera, with $\omega = K^{-T}K^{-1}$. So, we can estimate K 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 $\pi/4$ 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

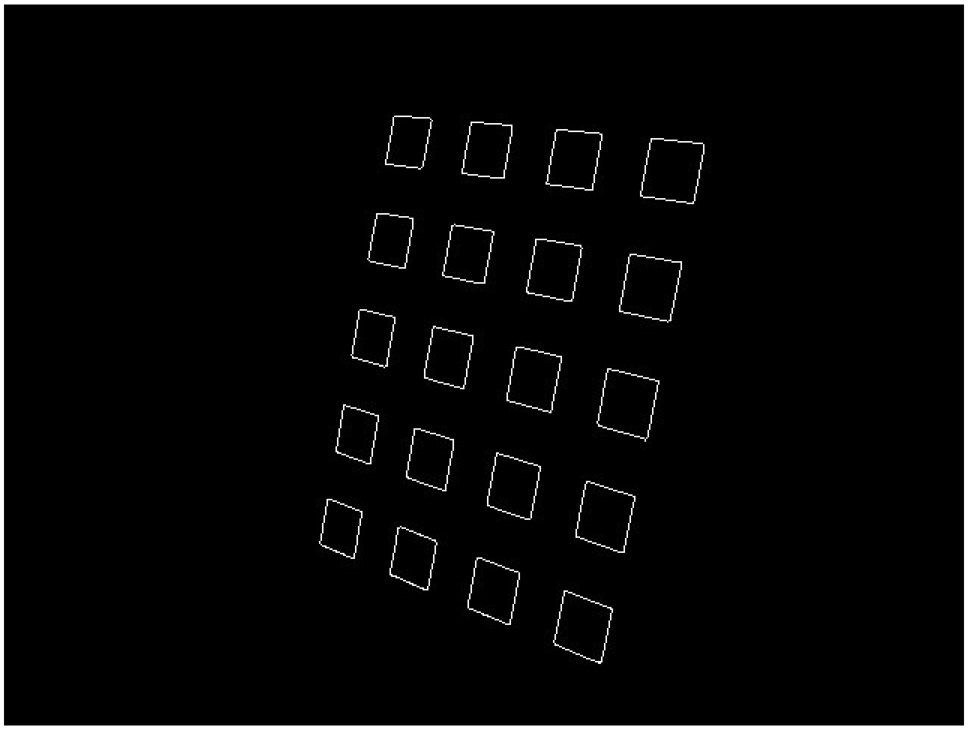

Figure 1: Canny edge detection of Pic_1.jpg in Dataset1

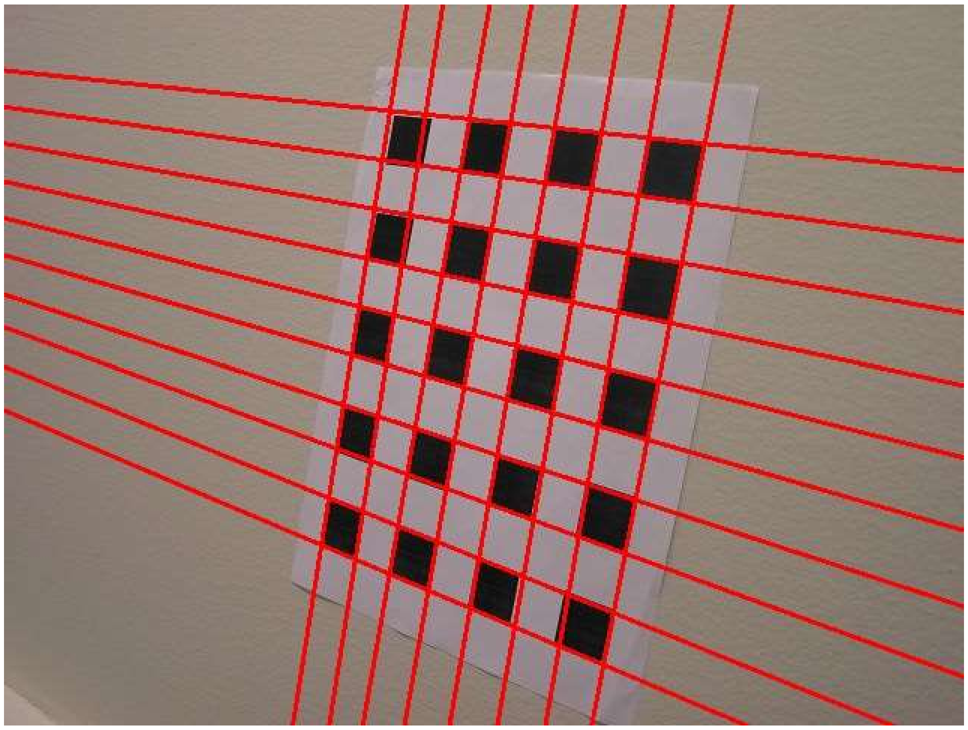

Figure 2: Filtered Hough lines of Pic_1.jpg in Dataset1

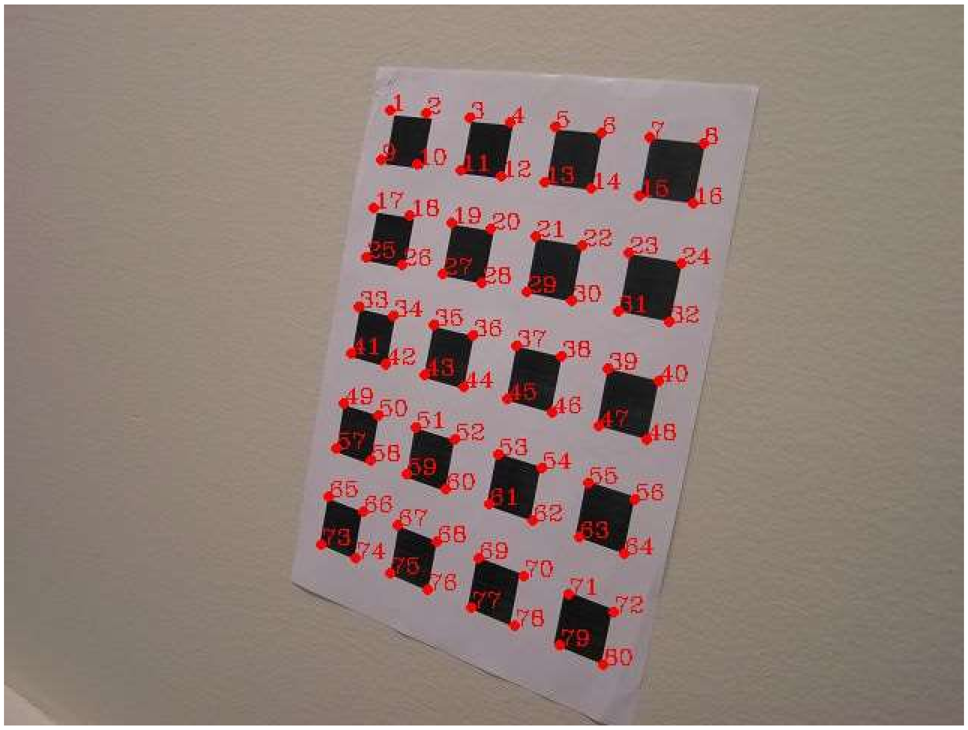

Figure 3: Detected corners of Pic_1.jpg in Dataset1

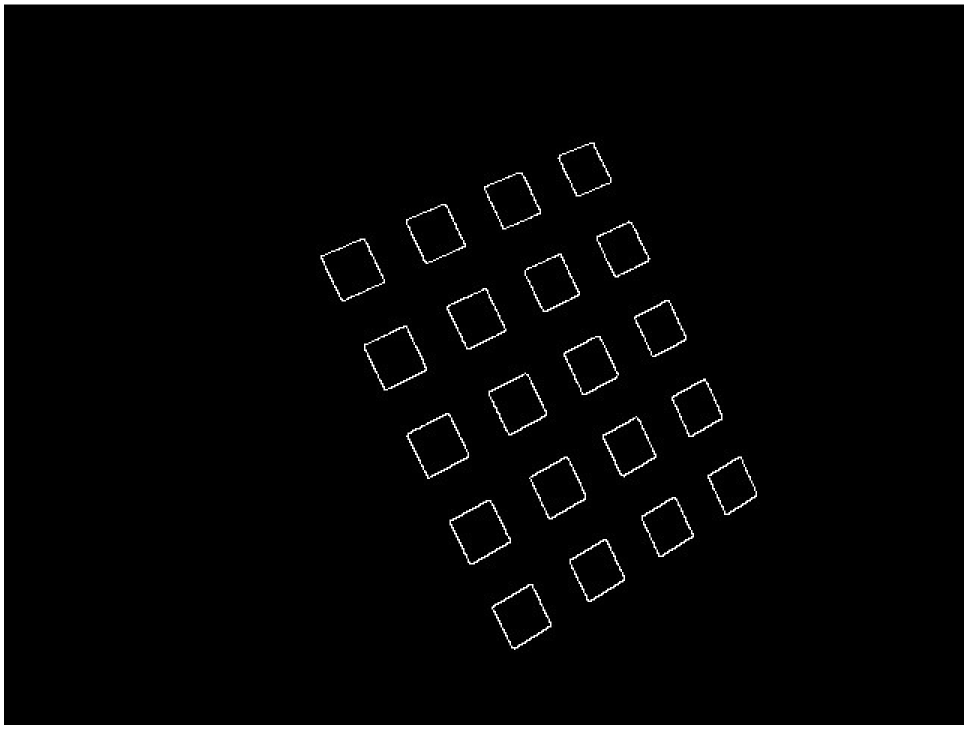
Figure 4: Canny edge detection of Pic_4.jpg in Dataset1

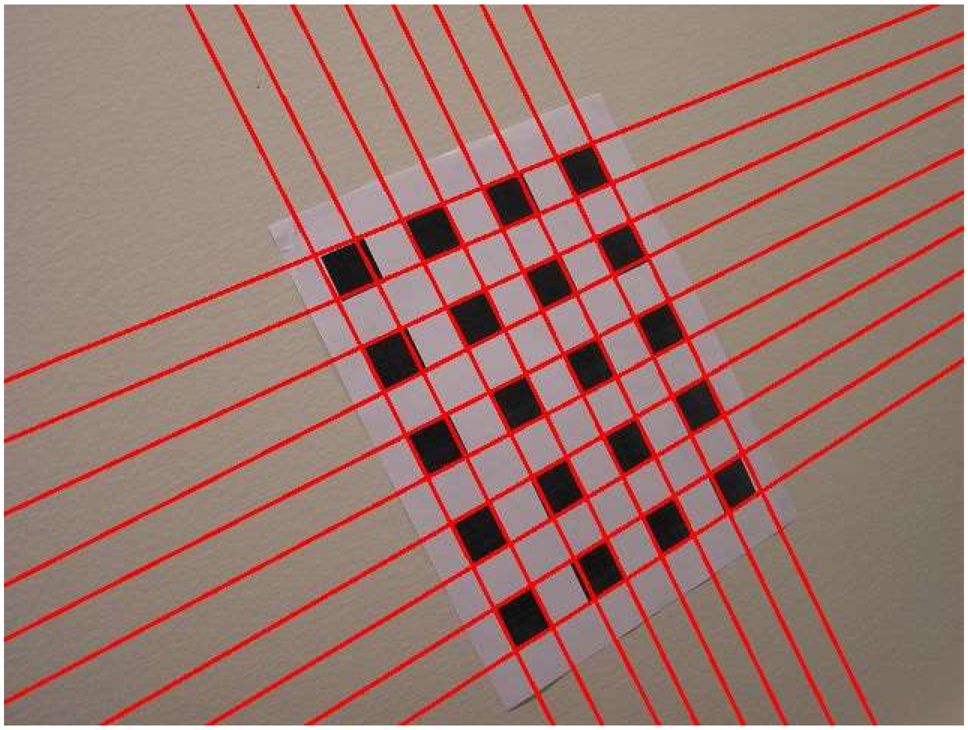
Figure 5: Filtered Hough lines of Pic_4.jpg in Dataset1

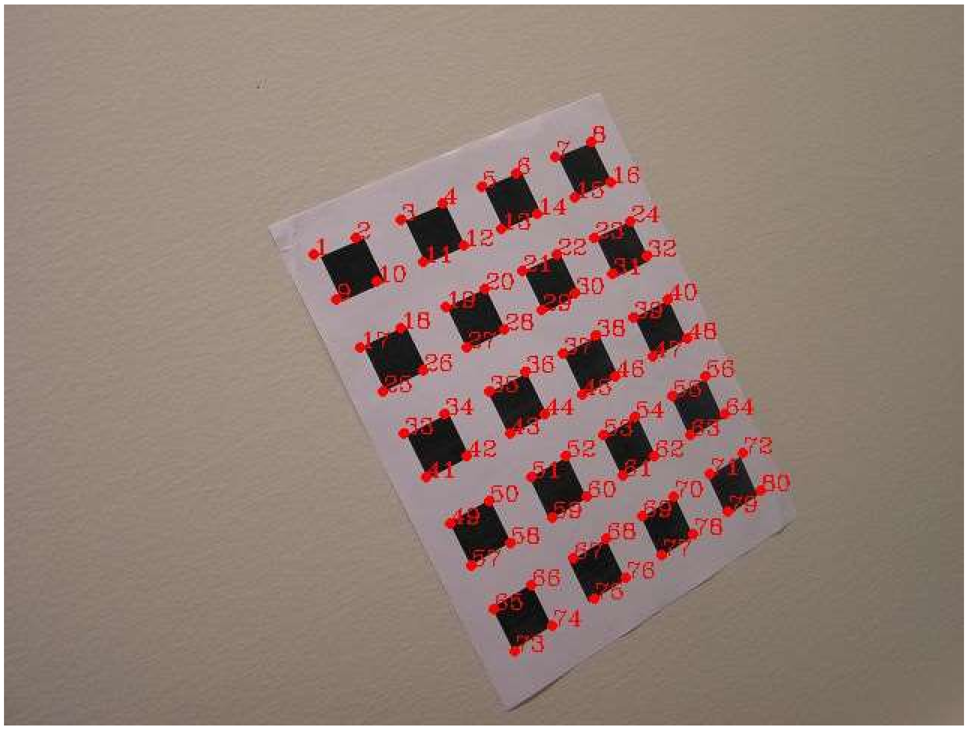

Figure 6: Detected corners of Pic_4.jpg in Dataset1

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)
```