Labb2

Elias Berglin

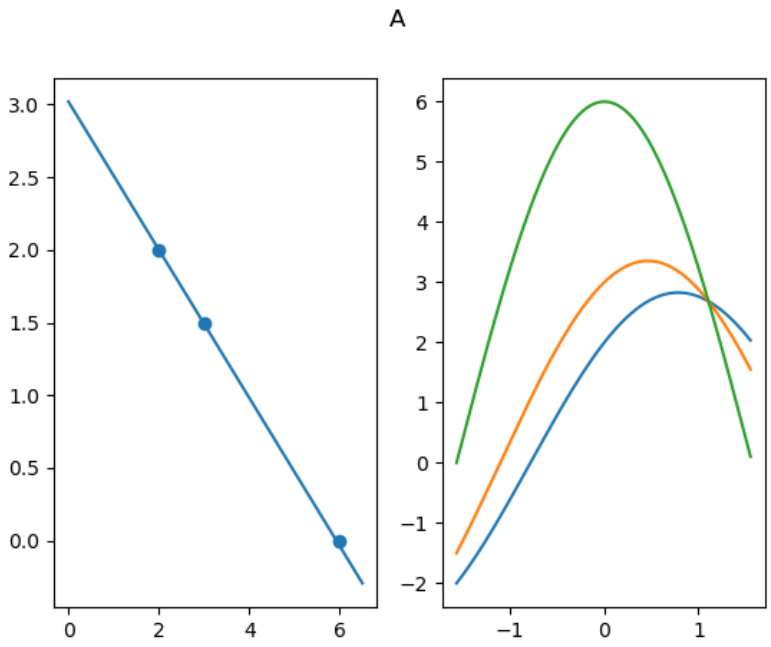
29th November 2021

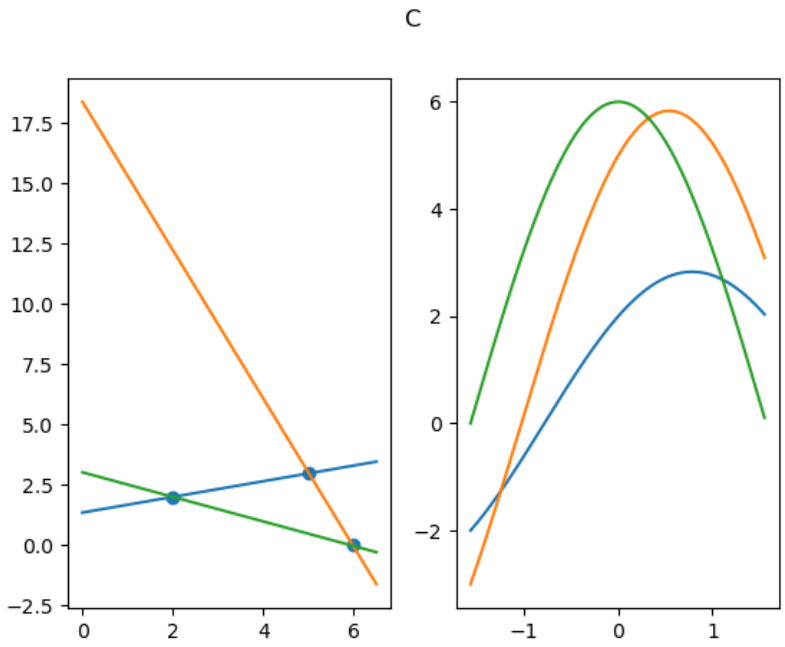
1 Week 3

1.1 Assignment

1.1.1 Hough transform

The following graphs was created with the points given and with Hough transform applied:





A quick note, Question b was not plotted due it being the same question as a

1.1.2 Feature descriptors

I have read 2 comparison papers between different feature descriptors. SIFT seems to always be the most acurate. But it is also not as fast. Considoring we have an AR application we need to have a somewhat faster algorithm to make it easier to track features. These papers say that ORB is almost as good as SIFT but much faster so I would go with ORB for my AR application. [1] [2]

1.1.3 Feature detection (and matching)

Python code used for the assignment can be found in appendix A. Following images was generated:

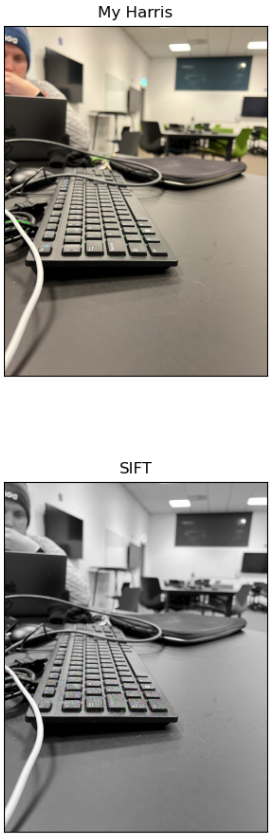
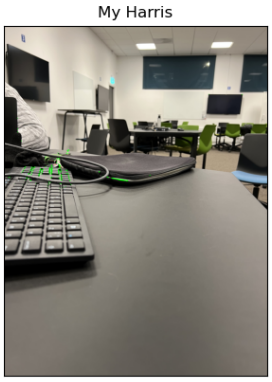
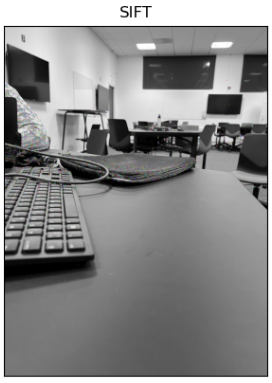


Figure 1: First image with Harris and SIFT



(a)



(b)

Figure 2: Second image with Harris and SIFT

1.2 Reflection

This week was interesting. I thought that feature detection/descriptors was a really interesting subject. The most interesting part was reading the research papers where they benchmarked different algorithms for finding features. I had a hard time grasping the concept of Hough transform but after watching the YouTube course I quickly understood how it worked. This has been the fact for all of the weeks. If I don't fully understand something during the lecture the YouTube course have always cleard it up for me.

My Harris detector did not performe well compared to SIFT. But when I compared it to the OpenCV implementation it was not far off. From reading the papers [1] [2] I now understand that SIFT is really good at finding features

and thus it is understandable that Harris does not perform ass well

2 Week 4

2.1 Assingment

2.1.1 2D Transformations

The question was for a similarity transform but we are not doing any scaling so we are really doing an euclidian transformation so the folling matrix was created based on the lecture slides:

$$\begin{pmatrix} cos(15 * \frac{\pi}{180}) & -sin(15 * \frac{\pi}{180}) & 3 \\ sin(15 * \frac{\pi}{180}) & cos(15 * \frac{\pi}{180}) & -2 \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

To calculate the transfomration matrix (homography matrix) the code in appendix B was used and the following matrix was calculated:

$$\begin{pmatrix} 1.8 & 0.67 & 1 \\ 0.6 & 0.67 & 1 \\ 0.2 & 0 & 1 \end{pmatrix} \tag{2}$$

2.1.2 Stereo estimation

I had ha hard time understanding the website interface so I just took the first algorithm with a paper connected to it. So I chose RAFT-Sterio [3]. This algorithm is based on RAFT [4] and extends the concepts of GRU

2.1.3 Plane Sweep

2.1.4 Stitching

2.2 Reflection

References

[1] BR Kavitha, G Ramya, and G Priya. "Performance comparison of various feature descriptors in object category detection application using SVM classifier". In: (2019).

[2] Shaharyar Ahmed Khan Tareen and Zahra Saleem. "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk". In: *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*. IEEE. 2018, pp. 1–10.

[3] Lahav Lipson, Zachary Teed, and Jia Deng. "RAFT-Stereo: Multilevel Recurrent Field Transforms for Stereo Matching". In: *arXiv preprint arXiv:2109.07547* (2021).

[4] Zachary Teed and Jia Deng. "Raft: Recurrent all-pairs field transforms for optical flow". In: *European conference on computer vision*. Springer. 2020, pp. 402–419.

Appendix A Harris implementation

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def harris(img_name, window_size, k, threshold):
    img = cv2.imread(img_name)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_gauss = cv2.GaussianBlur(gray, (3, 3), 0)
    h = img.shape[0]
    w = img.shape[1]

    matrix_r = np.zeros((h, w))

    dx = cv2.Sobel(img_gauss, cv2.CV_64F,1,0, ksize=3)
    dy = cv2.Sobel(img_gauss, cv2.CV_64F, 0, 1, ksize=3)

    dx2 = np.square(dx)
    dy2 = np.square(dy)
    dxy = dx * dy

    offset = int(window_size/2)
    print("Finding corners...")
    for y in range(offset, h-offset):
        for x in range(offset, w-offset):
            sx2 = np.sum(dx2[y-offset:y+1+offset, x-offset:x+1+
                offset])
            sy2 = np.sum(dy2[y-offset:y+1+offset, x-offset:x+1+
                offset])
            sxy = np.sum(dxy[y-offset:y+1+offset, x-offset:x+1+
                offset])

            H = np.array([[sx2, sxy], [sxy, sy2]])
            det = np.linalg.det(H)
            tr = np.matrix.trace(H)
            R = det - k * (tr ** 2)
            matrix_r[y - offset, x - offset] = R

    cv2.normalize(matrix_r, matrix_r, 0, 1, cv2.NORM_MINMAX)
    for y in range(offset, h - offset):
        for x in range(offset, w - offset):
            value = matrix_r[y, x]
            if value > threshold:
                cv2.circle(img, (x, y), 3, (0, 255, 0))

    plt.figure("Harris detector")
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title("My
         Harris")
    plt.xticks([]), plt.yticks([])
    plt.show()

def cv2Harris(img_name):
    img = cv2.imread(img_name)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

```

```
    harris = cv2.cornerHarris(gray, 2, 3, 0.04)
    harris = cv2.dilate(harris, None)
    img[harris > 0.01 * harris.max()] = [0, 0, 255]

    plt.figure("Harris detector")
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title("
        CV2 Harris")
    plt.xticks([]), plt.yticks([])
    plt.show()

def cv2Sift(img_name):
    img = cv2.imread(img_name)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    sift = cv2.SIFT_create()
    kp = sift.detect(gray, None)

    img = cv2.drawKeypoints(gray, kp, img)
    plt.figure("SIFT Detector")
    plt.imshow(img), plt.title("SIFT")
    plt.xticks([]), plt.yticks([])
    plt.show()

img_name = "start2.jpeg"

harris(img_name, 5, 0.04, 0.30)
cv2Harris(img_name)
cv2Sift(img_name)
```

Appendix B Python code for calculating the homography matrix

```
import numpy as np

np.set_printoptions(suppress=True)

def calcHomo(p1, p2):
    A = []
    for i in range(0, len(p1)):
        x, y = p1[i][0], p1[i][1]
        u, v = p2[i][0], p2[i][1]
        A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
        A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
    A = np.asarray(A)
    U, S, Vh = np.linalg.svd(A)
    L = Vh[-1,:] / Vh[-1,-1]
    H = L.reshape(3, 3)
    return H

def calcNewPoints(src, H):
    newPoints = []
    for x,y in src:
        vec = np.asarray([x,y,1])
        mult = H.dot(vec)
        newPoint = mult/mult[-1]
```

```
        newPoints.append([newPoint[0], newPoint[1]])


    return np.asarray(newPoints)

src = np.asarray([[0,0],[0,3],[5,3],[5,0]])
dst = np.asarray([[1,1],[3,3],[6,3],[5,2]])

H = calcHomo(src, dst)

print(H)

print(calcNewPoints(src, H))
```

Appendix C Python code for Plane Sweep

```
import cv2
import numpy as np

def depth(img1, img2, distance):
    height, width = img1.shape
    disparity = 0
    depthArray = np.zeros((height, width))
    maxDepth = 255/distance
    for y in range(0,height):
        for x in range(0, width):
            prev_min_val = np.inf
            for d in range(distance):
                temp_min_value = float(img1[y][x]) - float(img2[y][
                    x-d])
                min_value = temp_min_value * temp_min_value
                if min_value < prev_min_val:
                    prev_min_val = min_value
                    disparity = d
            depthArray[y][x] = disparity * maxDepth
    cv2.imwrite("sd.png", depthArray)




img1 = cv2.imread("ps1.ppm", cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread("ps2.ppm", cv2.IMREAD_GRAYSCALE)

stereo = cv2.StereoSGBM_create(numDisparities=16, blockSize=5)
disp = stereo.compute(img1,img2)
cv2.imwrite("cv2Sereo.png", disp)

depth(img1, img2, 16)
```

Appendix D Python code for Image Stitching

```python
import cv2
import numpy as np
import random
np.set_printoptions(suppress=True)

def calcHomo(p1, p2):
    A = []
    for i in range(0, len(p1)):
        x, y = p1[i][0][0], p1[i][0][1]
        u, v = p2[i][0][0], p2[i][0][1]
        A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
        A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
    A = np.asarray(A)
    U, S, Vh = np.linalg.svd(A)
    L = Vh[-1,:] / Vh[-1,-1]
    H = L.reshape(3, 3)
    return H

def geoDistance(p1, p2, h):
    p1 = p1[0]
    p2 = p2[0]
    p1 = np.append(p1, 1)
    p2 = np.append(p2, 1)
    estimatep2 = np.dot(h, p1)
    estimatep2 = estimatep2/estimatep2[-1]
    error = p2 - estimatep2

    return np.linalg.norm(error)


def ransac(src, dst, threshold):
    maxInliers = []
    finalH = None
    meme = 0
    for i in range(1000):
        meme = i
        p1 = []
        p2 = []
        #First cord:
        index = random.randrange(0, len(src))
        p1.append(src[index])
        p2.append(dst[index])
        #Second cord:
        index = random.randrange(0, len(src))
        p1.append(src[index])
        p2.append(dst[index])
        #Third cord:
        index = random.randrange(0, len(src))
        p1.append(src[index])
        p2.append(dst[index])
        #Fourth cord:
        index = random.randrange(0, len(src))
        p1.append(src[index])
        p2.append(dst[index])

        h = calcHomo(p1, p2)
        inlliers = []
```

```
58
59          for i in range(len(src)):
60              d = geoDistance(src[i], dst[i], h)
61              if d<5:
62                  inlliers.append([src[i], dst[i]])
63
64          if len(inlliers) > len(maxInliers):
65              maxInliers = inlliers
66              finalH = h
67          if len(maxInliers) > (len(src) * threshold):
68              break
69      print("Num iter:",meme)
70      return finalH
71
72
73
74  img1 = cv2.imread("right.jpg")
75  img2 = cv2.imread("left.jpg")
76
77  img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
78  img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
79
80  detector = cv2.ORB_create(nfeatures=2000)
81
82  keypoints1, descriptors1 = detector.detectAndCompute(img1, None)
83  keypoints2, descriptors2 = detector.detectAndCompute(img2, None)
84
85  bf = cv2.BFMatcher()
86  matches = bf.knnMatch(descriptors1,descriptors2, k=2)
87
88  # Ratio test
89  good_matches = []
90  for m, n in matches:
91      if m.distance < 0.6 * n.distance:
92          good_matches.append(m)
93
94
95  src_pts = np.float32([keypoints1[m.queryIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
96  dst_pts = np.float32([keypoints2[m.trainIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
97
98
99  H2, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
100 H = ransac(src_pts, dst_pts, 0.74)
101
102 print(H)
103 print(H2)
104
105 result = cv2.warpPerspective(img1, H,(img1.shape[1] + img2.shape
        [1], img1.shape[0]))
106 result[0:img2.shape[0], 0:img2.shape[1]] = img2
107 cv2.imshow("Result", result)
108 result2 = cv2.warpPerspective(img1, H2,(img1.shape[1] + img2.shape
        [1], img1.shape[0]))
109
110 result2[0:img2.shape[0], 0:img2.shape[1]] = img2
```

```
111 cv2.imwrite("output.jpg", result)
112 cv2.waitKey()
```