

Labb2

Elias Berglin

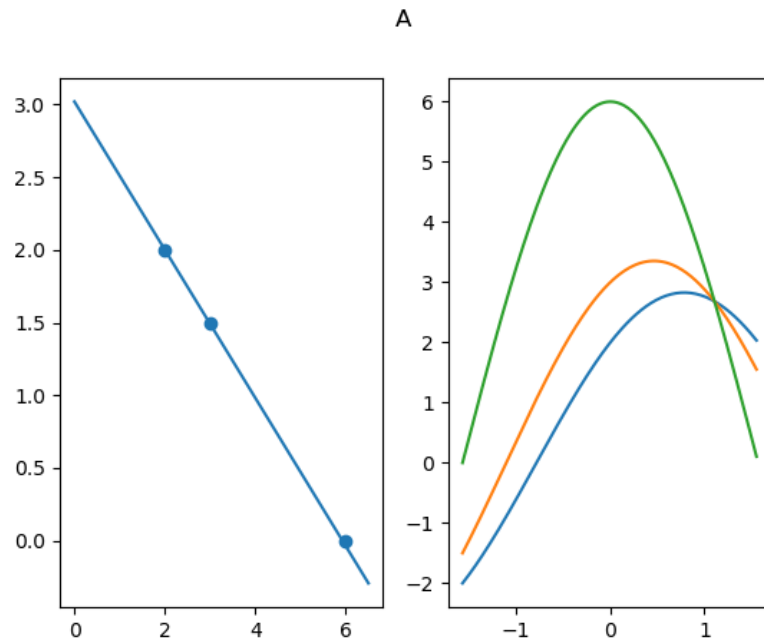
26<sup>th</sup> November 2021

# 1 Week 3

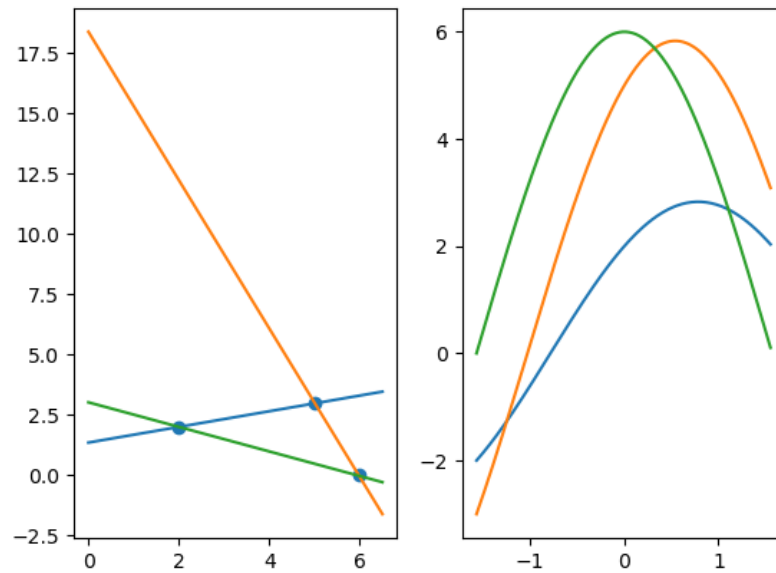
## 1.1 Assignment

### 1.1.1 Hough transform

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



C



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 accurate. But it is also not as fast. Considering 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:

My Harris



SIFT

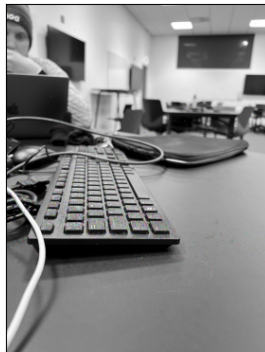
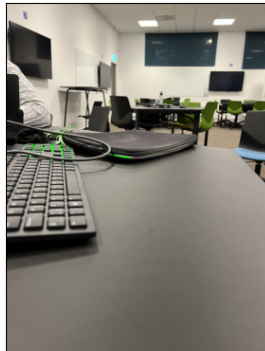


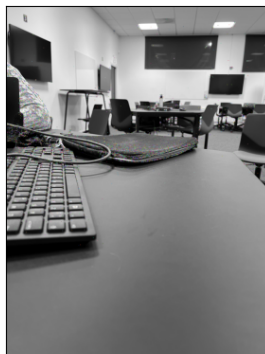
Figure 1: First image with Harris and SIFT

My Harris



(a)

SIFT



(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 cleared 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 as 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:

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

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

$$\begin{pmatrix} 1.8 & 0.67 & 1 \\ 0.6 & 0.67 & 1 \\ 0.2 & 0 & 1 \end{pmatrix} \quad (2)$$

#### 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

```

1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5
6 def harris(img_name, window_size, k, threshold):
7     img = cv2.imread(img_name)
8     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
9     img_gauss = cv2.GaussianBlur(gray, (3, 3), 0)
10    h = img.shape[0]
11    w = img.shape[1]
12
13    matrix_r = np.zeros((h, w))
14
15    dx = cv2.Sobel(img_gauss, cv2.CV_64F, 1, 0, ksize=3)
16    dy = cv2.Sobel(img_gauss, cv2.CV_64F, 0, 1, ksize=3)
17
18    dx2 = np.square(dx)
19    dy2 = np.square(dy)
20    dxy = dx * dy
21
22    offset = int(window_size/2)
23    print("Finding corners...")
24    for y in range(offset, h-offset):
25        for x in range(offset, w-offset):
26            sx2 = np.sum(dx2[y-offset:y+1+offset, x-offset:x+1+
27                           offset])
28            sy2 = np.sum(dy2[y-offset:y+1+offset, x-offset:x+1+
29                           offset])
30            sxy = np.sum(dxy[y-offset:y+1+offset, x-offset:x+1+
31                           offset])
32
33            H = np.array([[sx2, sxy], [sxy, sy2]])
34            det = np.linalg.det(H)
35            tr = np.matrix.trace(H)
36            R = det - k * (tr ** 2)
37            matrix_r[y - offset, x - offset] = R
38
39    cv2.normalize(matrix_r, matrix_r, 0, 1, cv2.NORM_MINMAX)
40    for y in range(offset, h - offset):
41        for x in range(offset, w - offset):
42            value = matrix_r[y, x]
43            if value > threshold:
44                cv2.circle(img, (x, y), 3, (0, 255, 0))
45
46    plt.figure("Harris detector")
47    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), plt.title("My
48    Harris"))
49    plt.xticks([], plt.yticks([]))
50    plt.show()
51
52 def cv2Harris(img_name):
53     img = cv2.imread(img_name)
54     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
55

```



```

52 harris = cv2.cornerHarris(gray, 2, 3, 0.04)
53 harris = cv2.dilate(harris, None)
54 img[harris > 0.01 * harris.max()] = [0, 0, 255]
55
56 plt.figure("Harris detector")
57 plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title("
    CV2 Harris")
58 plt.xticks([], plt.yticks([]))
59 plt.show()
60
61 def cv2Sift(img_name):
62     img = cv2.imread(img_name)
63     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
64     sift = cv2.SIFT_create()
65     kp = sift.detect(gray, None)
66
67     img = cv2.drawKeypoints(gray, kp, img)
68     plt.figure("SIFT Detector")
69     plt.imshow(img), plt.title("SIFT")
70     plt.xticks([], plt.yticks([]))
71     plt.show()
72
73 img_name = "start2.jpeg"
74
75 harris(img_name, 5, 0.04, 0.30)
76 cv2Harris(img_name)
77 cv2Sift(img_name)

```

## Appendix B Python code for calculating the homography matrix

```

1 import numpy as np
2
3 np.set_printoptions(suppress=True)
4
5 def calcHomo(p1, p2):
6     A = []
7     for i in range(0, len(p1)):
8         x, y = p1[i][0], p1[i][1]
9         u, v = p2[i][0], p2[i][1]
10        A.append([x, y, 1, 0, 0, 0, -u*x, -u*y, -u])
11        A.append([0, 0, 0, x, y, 1, -v*x, -v*y, -v])
12    A = np.asarray(A)
13    U, S, Vh = np.linalg.svd(A)
14    L = Vh[-1,:] / Vh[-1,-1]
15    H = L.reshape(3, 3)
16    return H
17
18 def calcNewPoints(src, H):
19     newPoints = []
20     for x,y in src:
21         vec = np.asarray([x,y,1])
22         mult = H.dot(vec)
23         newPoint = mult/mult[-1]

```

```
24     newPoints.append([newPoint[0], newPoint[1]])
25
26
27     return np.asarray(newPoints)
28
29 src = np.asarray([[0,0],[0,3],[5,3],[5,0]])
30 dst = np.asarray([[1,1],[3,3],[6,3],[5,2]])
31
32 H = calcHomo(src, dst)
33
34 print(H)
35
36 print(calcNewPoints(src, H))
```