4TN4 Assignment 6

Adam Bujak (400113347)

March 26, 2022

1 Theory

1.1 Hough Transform

Given the following image where each 1 represents an edge, we can calculate the Hough Transform by looping through each pixel and calculating the perpendicular distance between each line that satisfies that pixel.

0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0

using the following formulas we can calculate the perpendicular distance and then cast votes for each line:

$$d = x \cdot cos(\theta) + y \cdot sin(\theta)$$

Then we intuitively know that the maximum value in the accumulator should be 4, since when that is the largest number of pixels on the same line in this image (see Figure 1).

Therefore the corresponding (ρ, θ) value could be related to any of these, but I will describe the one related to the main diagonal line.

Which would be (0, $\frac{3\Pi}{4}$).

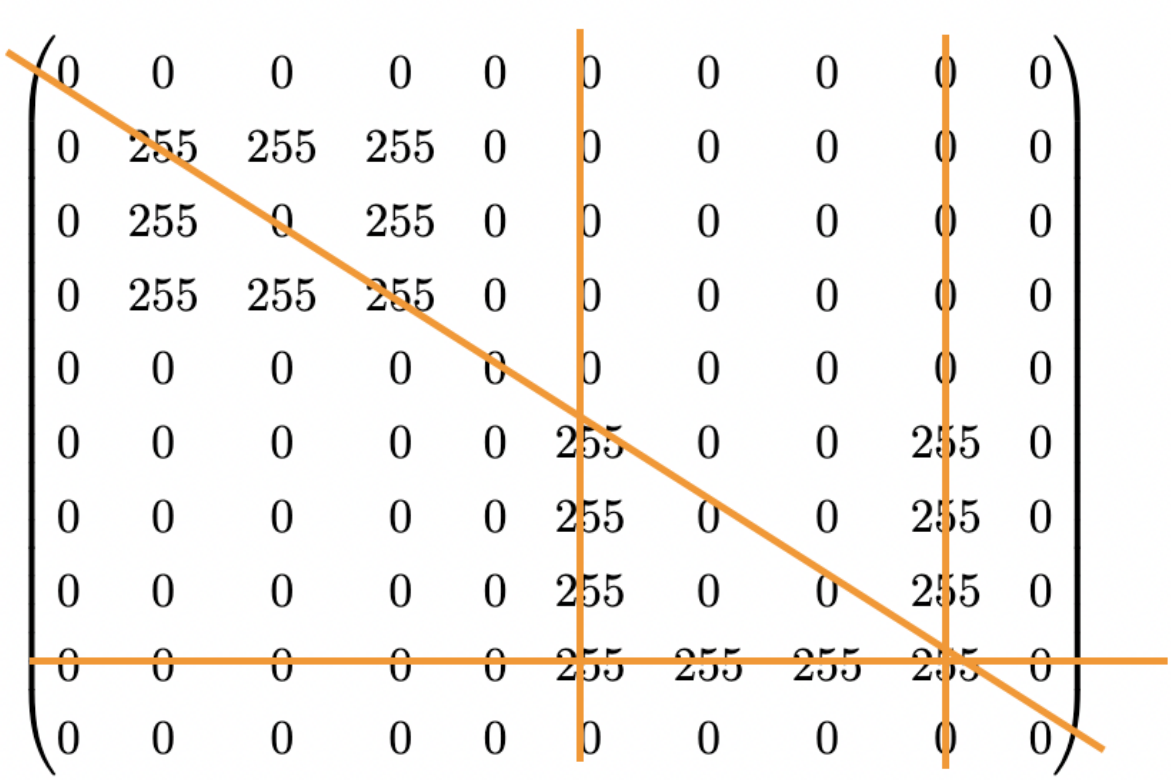

Figure 1: Best lines in image

1.2 Optical Flow

1.3 Orientation Detection

To detect the orientation of this image I would get the Hough Transform and take the 4 most prominent lines from it. Then, once I have that, I can discard lines with similar angles, so that I'm left with only two angles that should be orthogonal. Once I have the two orthogonal angles, I have my basis vector, and know the orientation of this object.

2 Implementation

2.1 Tracking

Using the following code I got the following result for optical flow tracking:

```python
import numpy as np
import cv2 as cv

cap = cv.VideoCapture("video1.mp4")
# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 40,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )
# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (15, 15),
                  maxLevel = 2,
                  criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03))
# Create some random colors
color = np.random.randint(0, 255, (100, 3))
# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
p0 = cv.goodFeaturesToTrack(old_gray, mask = None, **feature_params)
# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)
thickness = 30
while(1):
    ret, frame = cap.read()
    if not ret:
        print('No frames grabbed!')
        break
    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)
    # Select good points
    if p1 is not None:
        good_new = p1[st==1]
        good_old = p0[st==1]
    # draw the tracks
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        a, b = new.ravel()
        c, d = old.ravel()
        mask = cv.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].tolist(), int(thickness))
        thickness -= .1
        thickness = max(thickness, 1)
        frame = cv.circle(frame, (int(a), int(b)), 5, color[i].tolist(), -1)
    img = cv.add(frame, mask)
    cv.imshow('frame', img)
    k = cv.waitKey(300) & 0xff
    if k == 27:
        break
    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)

cv.destroyAllWindows()
```

Figure 2: Optical flow code

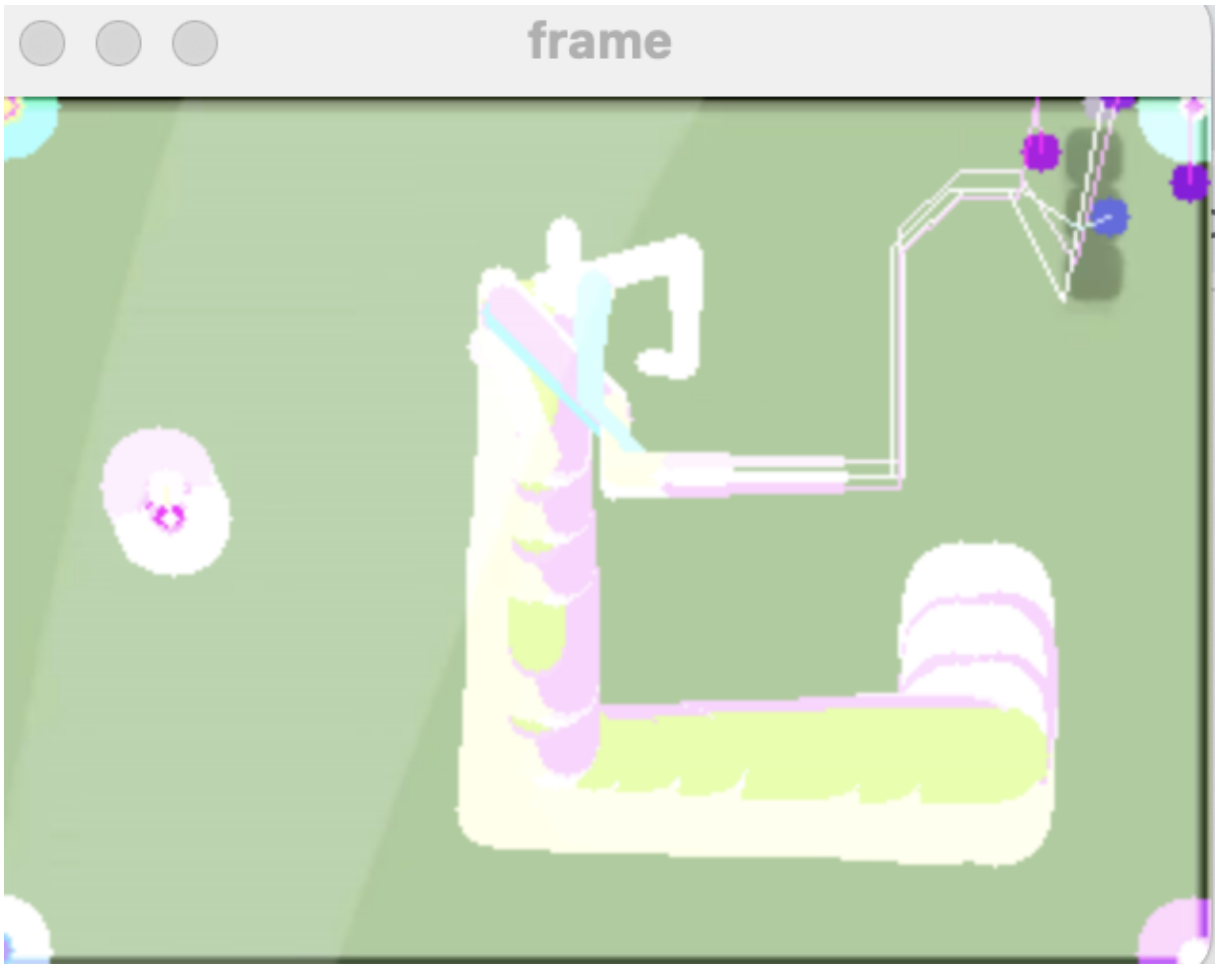

Figure 3: Optical flow results