Ex 1 Daniel Kusnetsoff

a)

$$v = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$$

$$v' = \begin{pmatrix} x_2' - x_1' \\ y_2' - y_1' \end{pmatrix} = s \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$$

we need the θ angle

$$\cos\theta = \frac{v' \cdot v}{\|v'\| \cdot \|v\|} = \frac{(x_2'-x_1')(x_2-x_1) + (y_2'-y_1')(y_2-y_1)}{\sqrt{(x_2'-x_1')^2 + (y_2'-y_1')^2} \cdot \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}}$$

$$\theta = \cos^{-1}\left(\frac{(x_2'-x_1')(x_2-x_1) + (y_2'-y_1')(y_2-y_1)}{\sqrt{(x_2'-x_1')^2 + (y_2'-y_1')^2} \cdot \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}} \right)$$

b)

$$s = \frac{\|v'\|}{\|v\|} = \frac{\sqrt{(x_2'-x_1')^2 + (y_2'-y_1')^2}}{\sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}}$$

c)

$$x' = s\cos\theta x - s\sin\theta y + t_x$$

$$\Rightarrow t_x = x' - s \cdot \cos\theta x + s \cdot \sin\theta y$$

$$y' = s \cdot \sin\theta x + s\cos\theta y + t_y$$

$$\Rightarrow t_y = y' - s \cdot \sin\theta x - s \cdot \cos\theta y$$

Daniel Kusnetsoff

d) $v = \begin{pmatrix} -\frac{1}{2} \\ 1/2 \end{pmatrix} \nearrow \begin{pmatrix} x_2' - x_1' \\ y_2' - y_1' \end{pmatrix}$ $v' = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \nearrow \begin{pmatrix} x_2' - x_2' \\ y_2' - y_1' \end{pmatrix}$

From task a

$$\theta = \cos^{-1} \frac{(x_2' - x_1')(x_2 - x_1) + (y_2' - y_1')(y_2 - y_1)}{\sqrt{(x_2' - x_1')^2 + (y_2' - y_1')^2} \cdot \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

$$= \cos^{-1} \frac{(-1)(1/2) - (-1)(1/2)}{\text{---}\,u\,\text{---}}$$

$$\theta = \cos^{-1}(0) = \frac{\pi}{2}$$

From task b

$$S = \frac{\|v'\|}{\|v\|} = \frac{\sqrt{(x_2' - x_1')^2 + (y_2' - y_1')^2}}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

$$= \frac{\sqrt{(-1)^2 + (-1)^2}}{\sqrt{(-1/2)^2 + (1/2)^2}} = \frac{\sqrt{2}}{\sqrt{1/2}} = \sqrt{4} = 2$$

From task c)

$$t_x = x' - s \cdot \cos\theta x + s \sin\theta y = 0 - 2(0)(1/2) + 2(0)(1) = 0$$

$$t_y = y' - s \sin\theta x - s \cos\theta y = 0 - 2(1)(1/2) + 2(0)(0) = -1/2$$

$\Rightarrow$ Final equation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = 2 \begin{pmatrix} \cos(\pi/2) & -\sin(\pi/2) \\ \sin(\pi/2) & \cos(\pi/2) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ -1/2 \end{pmatrix}$$

Ex5tasks

February 13, 2022

```
[ ]: #Task 2
```

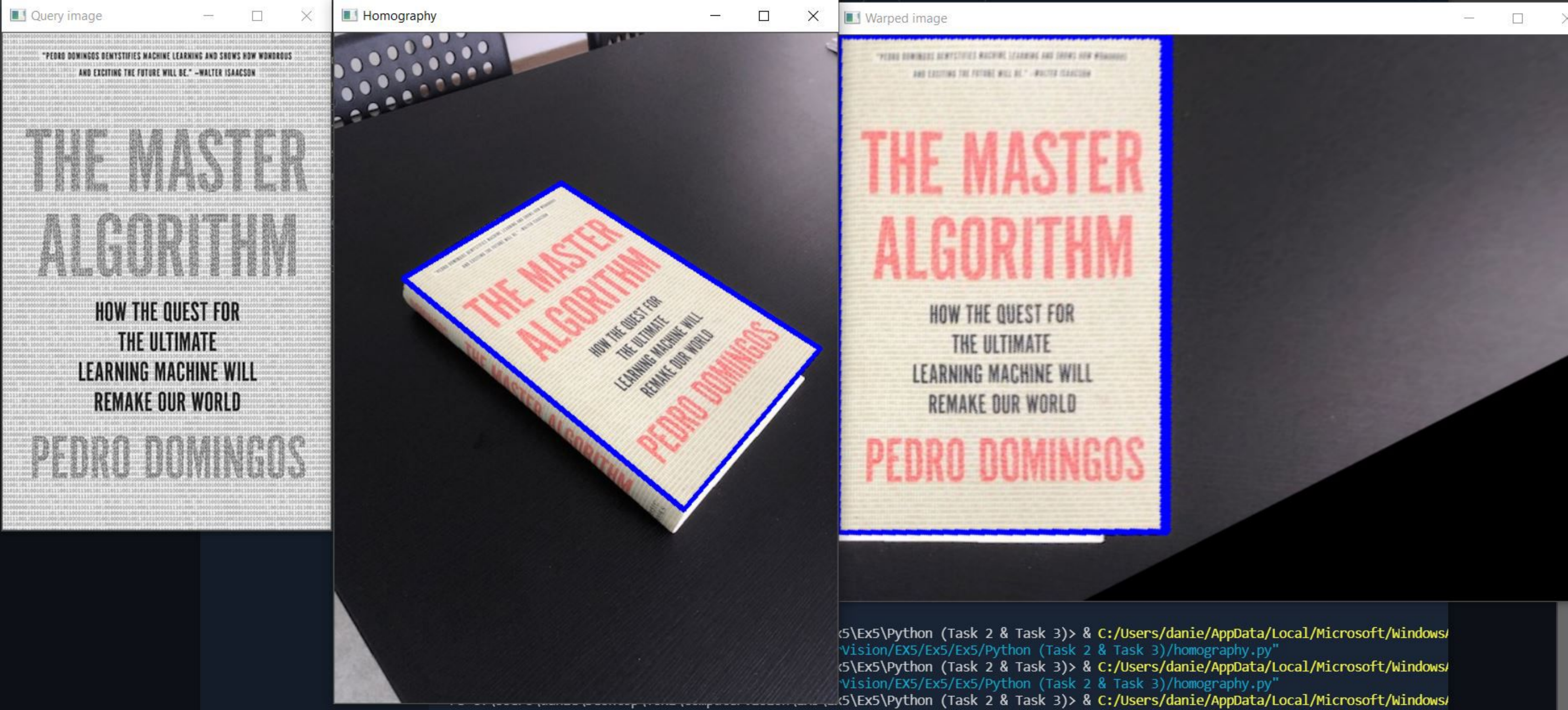
```
[ ]: import cv2
     import numpy as np

     img = cv2.imread("reference.jpg", cv2.IMREAD_GRAYSCALE)  # query image
     test_img = cv2.imread("image.jpg") # test image

     # Features
     sift = cv2.SIFT_create()
     keyp_image, descrip_image = sift.detectAndCompute(img, None)

     # Feature matching
     index_params = dict(algorithm=0, trees=5)
     search_params = dict()
     flann = cv2.FlannBasedMatcher(index_params, search_params)

     # Convert the  test image to grayscale using proper cv2-function.
     # After that calculate the keypoints and descriptors with SIFT.
     # Then calculate the matches between both query and test image descriptors
     # with already declared flann using knnMatch-function (k = 2).
     # Store the matches to "matches"-variable.

     ##--your-code-starts-here--##
     #grayframe = test_img #replace me
     grayframe=cv2.cvtColor(test_img,cv2.COLOR_BGR2GRAY)

     #keyp_grayframe, descrip_keyframe = 0, 0 # replace me
     keyp_grayframe, descrip_keyframe = keyp_grayframe, descrip_keyframe = sift.
      ↪detectAndCompute(grayframe, None)

     #matches = [] # replace me
     matches = flann.knnMatch(descrip_image, descrip_keyframe, k = 2)

     ##--your-code-ends-here--##

     good_points = []
```

```
thresh = 0.6

for m, n in matches:
    if m.distance < thresh * n.distance:
        good_points.append(m)

cv2.imshow("Query image", img)

if len(good_points) > 20:
    query_pts = np.float32([keyp_image[m.queryIdx].pt for m in good_points]).
 ↪reshape(-1, 1, 2)
    test_pts = np.float32([keyp_grayframe[m.trainIdx].pt for m in good_points]).
 ↪reshape(-1, 1, 2)

    # Calculate the homography using cv2.findHomography, look up the␣
 ↪documentation
    # (https://docs.opencv.org/master/d9/d0c/group__calib3d.html)
    # for the function to see what values it takes in. Store this homography␣
 ↪matrix to
    # variable "matrix". Note that the function returns the mask as well and
    # the code will throw an error if you don't store it anywhere.

    ##--your-code-starts-here--##

    #matrix = 0  # replace me
    matrix, mask = cv2.findHomography(query_pts, test_pts, cv2.RANSAC)

    ##--your-code-ends-here--##

    # Perspective transform
    h, w = img.shape
    pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
    dst = cv2.perspectiveTransform(pts, matrix)
    homography = cv2.polylines(test_img, [np.int32(dst)], True, (255, 0, 0), 3)
    cv2.imshow("Homography", homography)

    # Warp the image using cv2.warpPerspective and the homography matrix
    # so the target is in one to one correspondence to query image
    # in terms of perspective.
    # Use dsize = (720, 540)
    # HINT: In order to produce the inverse of what the homography does what
    # should you do with the homography matrix?

    ##--your-code-starts-here--##

    #im_warped = 0 # replace me
    dsize = (720, 540)
    im_warped = cv2.warpPerspective(test_img, np.linalg.inv(matrix), dsize)

    ##--your-code-ends-here--##
    cv2.imshow("Warped image", im_warped)
    ## added for viewing
    cv2.waitKey()
else:
    cv2.imshow("Homography", grayframe)
    ## added for viewing
    cv2.waitKey()
```

[ ]:
```
Task 3
```

[ ]:
```
import cv2
import time
import traceback
import numpy as np


def get_delay(start_time, fps=30):
    if (time.time() - start_time) > (1 / float(fps)):
        return 1
    else:
        return max(int((1 / float(fps)) * 1000 - (time.time() - start) * 1000),␣
 ↪1)


# Instantiate cascade classifiers for finding faces
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# Camera instance
cam = cv2.VideoCapture(0)
#cam = cv2.VideoCapture('visionface.avi')  # uncomment if you want to use a␣
 ↪video file instead

# Check if instantiation was successful
if not cam.isOpened():
    raise Exception("Could not open camera/file")


# USE OPENCV DOCUMENTATION TO FIND OUT HOW CERTAIN FUNCTIONS WORK.
# Your task is to implement real-time face point tracking.
# A few tips:
#  You should start by implementing the detection part first.
#  Try drawing the trackable points in the detection part without saving them
#  to p0 so you're able to see if the point coordinates are correct.
#  When finding the good points in the tracking part, use isFound as an index
#  for telling if the point is valid. (you may have to convert this to a␣
↪boolean array first).

gray_prev = None  # previous frame
p0 = []  # previous points

while True:
    start = time.time()
    try:
        # Get a single frame
        ret_val, img = cam.read()
        if not ret_val:
            cam.set(cv2.CAP_PROP_POS_FRAMES, 0)  # restart video
            gray_prev = None  # previous frame
            p0 = []  # previous points
            continue

        else:
            # Mirror
            img = cv2.flip(img, 1)

            # Grayscale copy
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            if len(p0) <= 10:
                # Detection
                img = cv2.putText(img, 'Detection', (0,20),
                                  cv2.FONT_HERSHEY_SIMPLEX, 0.8,␣
↪color=(0,255,255))

                # Detect faces
                faces = face_cascade.detectMultiScale(gray, 1.1, 5)

                # Take the first face and get trackable points.
                if len(faces) != 0:
                    # Extract ROI (face) from the grayscale frame
                    # Detections are in the form
                    # (x_upperleft, y_upperleft, width, height)
                    # You can also crop this ROI even more to make sure only
                    # the face area is considered in the tracking.

                    ##-your-code-starts-here-##
                    roi_gray = np.zeros_like(gray)  # replace me
                    ##-your-code-ends-here-##
                    for x_upperleft, y_upperleft, width, height in faces:
                        roi_gray = gray[y_upperleft:y_upperleft + height,␣
↪x_upperleft:x_upperleft + width]
```

```
                    # Get trackable points
                    p0 = cv2.goodFeaturesToTrack(roi_gray,
                                                 maxCorners=70,
                                                 qualityLevel=0.001,
                                                 minDistance=5)

                    # Convert points to form (point_id, coordinates)
                    p0 = p0[:, 0, :] if p0 is not None else []

                    # Convert from ROI to image coordinates
                    ##-your-code-starts-here-##
                    p0[:, 0] = p0[:, 0] + x_upperleft
                    p0[:, 1] = p0[:, 1] + y_upperleft
                    ##-your-code-ends-here-##

                # Save grayscale copy for next iteration
                gray_prev = gray.copy()

            else:
                # Tracking
                img = cv2.putText(img, 'Tracking', (0, 20), cv2.
↪FONT_HERSHEY_SIMPLEX, 0.8, color=(0, 255, 255))

                # Calculate optical flow using calcOpticalFlowPyrLK
                p1, isFound, err = cv2.calcOpticalFlowPyrLK(gray_prev, gray, p0,
                                                            None,
                                                            winSize=(31,31),
                                                            maxLevel=10,
                                                            criteria=(cv2.
↪TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 30, 0.03),
                                                            flags=cv2.
↪OPTFLOW_LK_GET_MIN_EIGENVALS,
                                                            minEigThreshold=0.
↪00025)

                # Select good points. Use isFound to select valid found points␣
↪from p1
                ##-your-code-starts-here-##
                p1 = [p1[k] for k in range(len(isFound)) if isFound[k] == 1]
                p1 = np.array(p1)
                ##-your-code-starts-here-##

                # Draw points using e.g. cv2.drawMarker
                ##-your-code-starts-here-##
                for p in p1:
                    img = cv2.circle(img, (p[0], p[1]), radius=5, color=(0,␣
↪255, 255))
                ##-your-code-ends-here-##

                # Update p0 (which points should be kept?) and gray_prev for
                # next iteration
                ##-your-code-starts-here-##
                p0 = p1
                gray_prev = gray.copy()
                ##-your-code-ends-here-##

            # Quit text
            img = cv2.putText(img, 'Press q to quit', (440, 20),
                              cv2.FONT_HERSHEY_SIMPLEX, 0.8, color=(0,0,255))
            cv2.imshow('Video feed', img)

        # Limit FPS to ~30 (if detector is fast enough)
        if cv2.waitKey(get_delay(start, fps=30)) & 0xFF == ord('q'):
            break  # q to quit

    # Catch exceptions in order to close camera and video feed window properly
    except:
        traceback.print_exc()  # display for user
        break

# Close camera and video feed window
cam.release()
cv2.destroyAllWindows()




#a.
# How it works.
# - Predefined face cascading classifiers
# - The classier is used to extract ROI from webcam footage
# - The accepted or successfull ROI will go trough another function
#   that is set to track the wanted features
# - Wanted feature points are marked in the image
#
#b. Problems with the tracking.
# - The biggest issue is the program can not track successfully if the face is␣
↪moving.
# - The possible solution could be that we add the number of feature points␣
↪required for the tracking.
# - This should lead to a situation where the detection is not reset as often␣
↪as now.
#
```


Query image
Homography
Warped image
"PEDRO DOMINGOS DEMYSTIFIES MACHINE LEARNING AND SHOWS HOW WONDROUS AND EXCITING THE FUTURE WILL BE." –WALTER ISAACSON
THE MASTER ALGORITHM
HOW THE QUEST FOR THE ULTIMATE LEARNING MACHINE WILL REMAKE OUR WORLD
PEDRO DOMINGOS

```
‹5\Ex5\Python (Task 2 & Task 3)> & C:/Users/danie/AppData/Local/Microsoft/Windows/
Vision/EX5/Ex5/Ex5/Python (Task 2 & Task 3)/homography.py"
‹5\Ex5\Python (Task 2 & Task 3)> & C:/Users/danie/AppData/Local/Microsoft/Windows/
Vision/EX5/Ex5/Ex5/Python (Task 2 & Task 3)/homography.py"
‹5\Ex5\Python (Task 2 & Task 3)> & C:/Users/danie/AppData/Local/Microsoft/Windows/
```