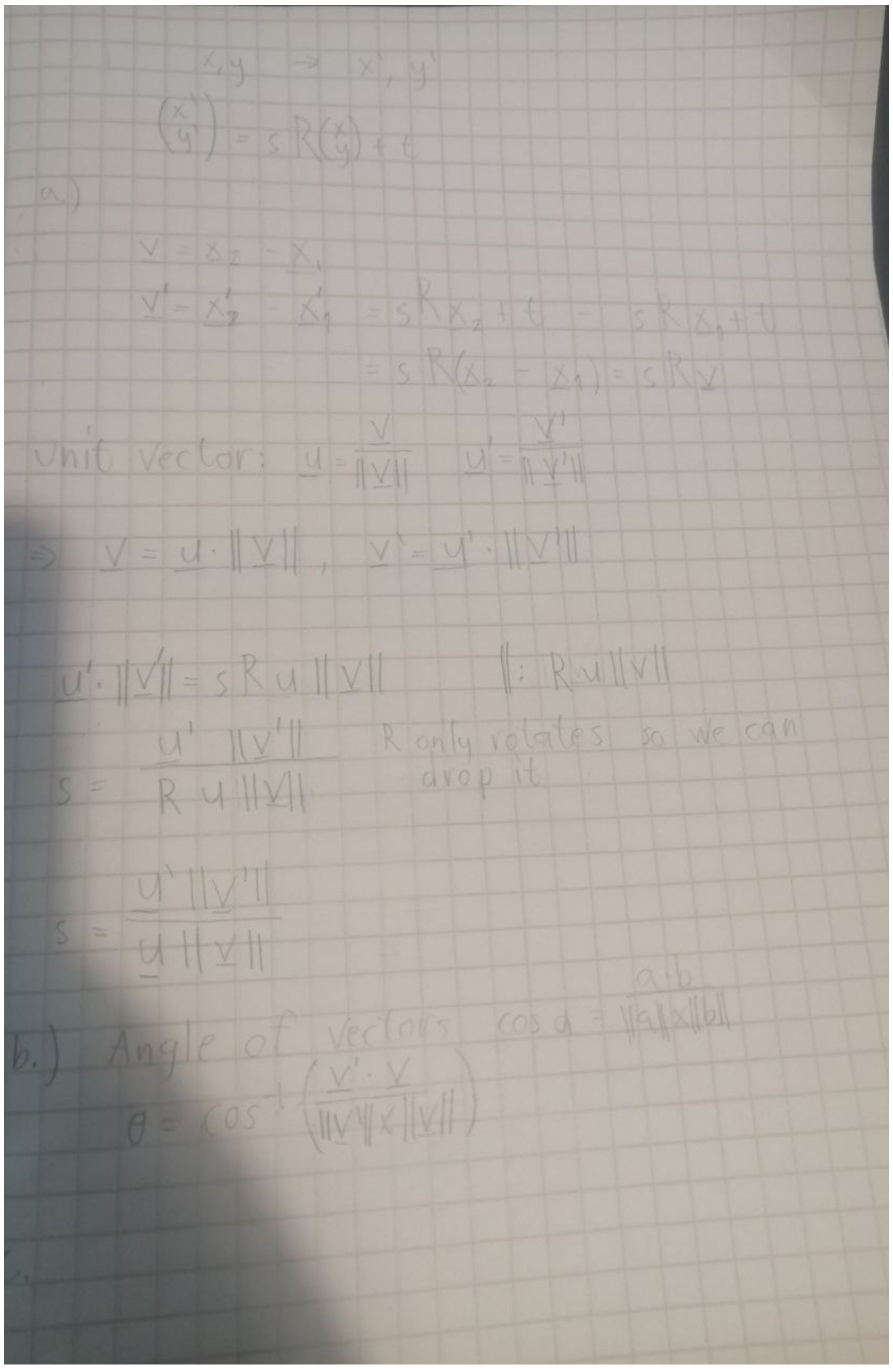
DATA.ML.300 Computer Vision Exercise 5

Miska Romppainen
H274426

February 2021

1 Task: Similarity transformation from two point correspondences.



$x, y \rightarrow x', y'$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = s R \begin{pmatrix} x \\ y \end{pmatrix} + t$$

a.)

$$\underline{v} = x_2 - \underline{x}_1$$

$$\underline{v}' = \underline{x}_2' - \underline{x}_1' = s R x_2 + t - s R x_1 + t$$

$$= s R (x_2 - x_1) = s R \underline{v}$$

Unit vector: $\underline{u} = \frac{\underline{v}}{\|\underline{v}\|}$ $\underline{u}' = \frac{\underline{v}'}{\|\underline{v}'\|}$

$\Rightarrow$ $\underline{v} = \underline{u} \cdot \|\underline{v}\|$, $\underline{v}' = \underline{u}' \cdot \|\underline{v}'\|$

$\underline{u}' \cdot \|\underline{v}'\| = s R u \|\underline{v}\|$ $\quad \| : R u \|v\|$

$$s = \frac{\underline{u}' \ \|\underline{v}'\|}{R u \|\underline{v}\|}$$ R only rotates so we can drop it

$$s = \frac{\underline{u}' \|\underline{v}'\|}{\underline{u} \|\underline{v}\|}$$

b.) Angle of vectors $\cos \alpha = \frac{a \cdot b}{\|a\| \times \|b\|}$

$$\theta = \cos^{-1} \left(\frac{\underline{v}' \cdot \underline{v}}{\|\underline{v}'\| \times \|\underline{v}\|} \right)$$

Figure 1: Task1

2 Task: Homography using SIFT

Here is my solution to task 2. Homography using SIFT. Idea of the task was to implement a code that finds a reference image from a test image. The goal of the code was to find reference image (figure 2) from an image (figure 3).

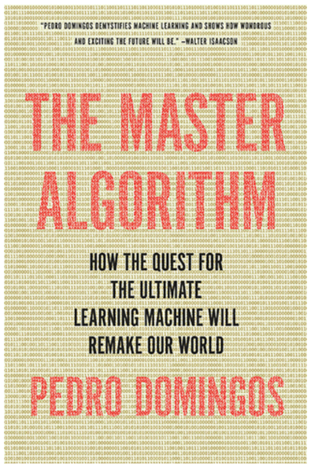

"PEDRO DOMINGOS DEMYSTIFIES MACHINE LEARNING AND SHOWS HOW WONDROUS AND EXCITING THE FUTURE WILL BE." —WALTER ISAACSON
THE MASTER ALGORITHM
HOW THE QUEST FOR THE ULTIMATE LEARNING MACHINE WILL REMAKE OUR WORLD
PEDRO DOMINGOS

Figure 2: Reference image

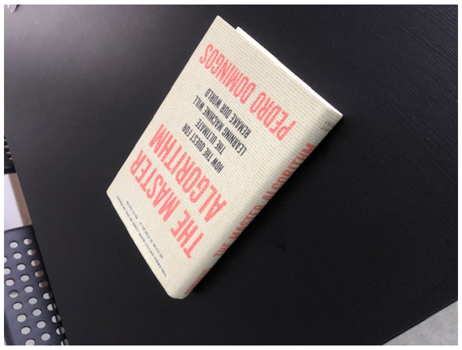


Figure 3: Original image

To find the reference image from the original first we gray-scale the reference image (figure 4), after which we can determine and calculate key-points and descriptors with the SIFT features calculated from the reference image, after which we search for the matches from reference images using declared Fast Library for Approximate Nearest Neighbors(Flann) matcher finding the matches as k-Nearest neighbors (where k=2). After finding the points which are included inside our threshold of error we can find homography matrix using

$cv2.findHomography$. The "good matches" are visualized in the figure 5. Using homography matrix we can draw the lines around the image (figure 6. To warp the referenced image from the original image we use $cv2.warpPerspective$, and we get 7.

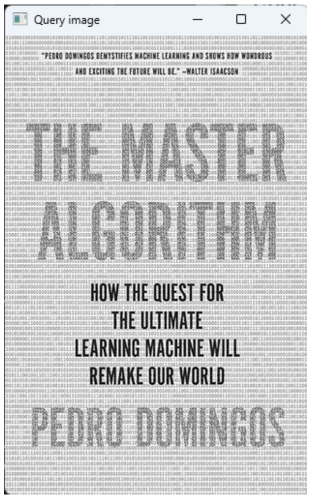

Query image
"PEDRO DOMINGOS DEMYSTIFIES MACHINE LEARNING AND SHOWS HOW WONDROUS AND EXCITING THE FUTURE WILL BE." —WALTER ISAACSON
THE MASTER ALGORITHM
HOW THE QUEST FOR THE ULTIMATE LEARNING MACHINE WILL REMAKE OUR WORLD
PEDRO DOMINGOS

Figure 4: Gray-Scaled reference image

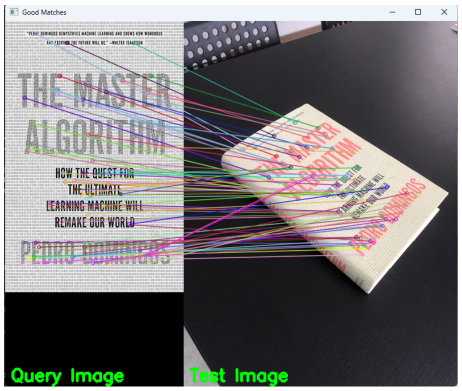


Figure 5: Good matches

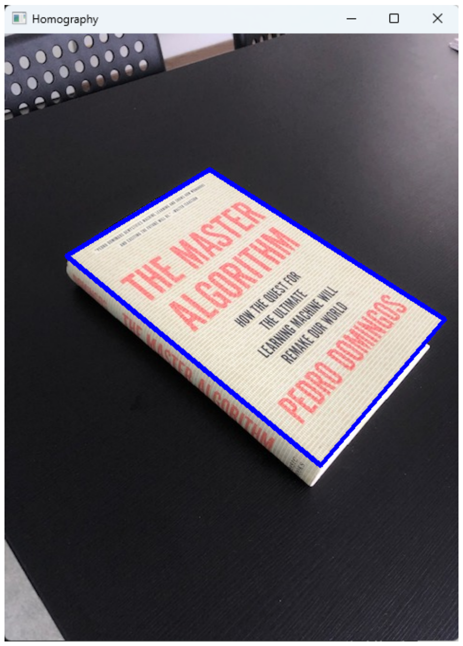


Figure 6: Homography

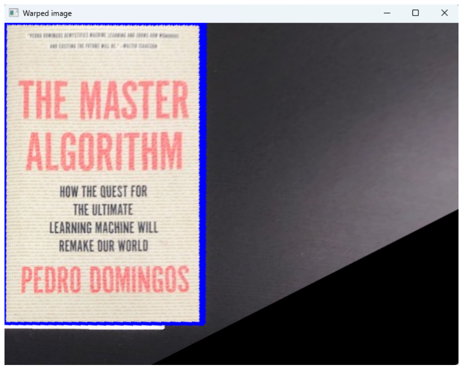


Figure 7: Wrapped image

CODE SNIPPET FROM: *homography.py*

```
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 = cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY) # replaced
keyp_grayframe, descrip_keyframe = sift.detectAndCompute(grayframe, None) # replaced
matches = flann.knnMatch(descrip_image, descrip_keyframe, k=2) # replaced
##--your-code-ends-here--##

good_points = []
thresh = 0.6

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

# Visualize matches # Uncomment to see matches visualized.
# Might not work depending on how you calculated the matches.
img_matches = np.empty((max(img.shape[0], test_img.shape[0]),
                        img.shape[1]+test_img.shape[1],
                        3),
                        dtype=np.uint8)
cv2.drawMatches(img,
                keyp_image,
                test_img,
                keyp_grayframe,
                good_points,
                img_matches,
                flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

 # Label left image as query image and
 # right image as test image in the lower left corner
cv2.putText(img_matches, "Query Image",
            (10, img_matches.shape[0]-10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0,
            (0, 255, 0), 3)
cv2.putText(img_matches, "Test Image",
            (img.shape[1] + 10, img_matches.shape[0]-10),
            cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 255, 0), 3)
cv2.imshow('Good Matches', img_matches)

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, _ = cv2.findHomography(query_pts,
                                       test_pts,
                                       cv2.RANSAC,5.0)  # replace me
        ##--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 = cv2.warpPerspective(test_img,
                                       np.linalg.inv(matrix),
                                       (720, 540)) # replace me
        ##--your-code-ends-here--##
        cv2.imshow("Warped image", im_warped)

    else:
        cv2.imshow("Homography", grayframe)

    cv2.waitKey(0)
```

3 Task: Real-time face point tracking

Here is my solution to task 3. Real-time face point tracking. Real-time face point tracking is done using KLT-tracker to track points detected from a face. To detect a face, a classifier is created using cascade classifier from a frontal-face default file. First each frame of the video-feed is gray-scaled. If a face is detected in a frame a ROI is extracted from a gray-scaled frame. The 'calcOpticalFlowPyrLK' function calculates the optical flow between two frames, in this case, the previous frame and the current frame. The ROI's image coordinates are extracted and a box is drawn around the coordinates. The outcome images can be seen in figures 8 and 9. As we can see in the figure 9 the tracking is not perfect, there are some false positives on the wall. These false-positives could be avoided by for example increasing the detection threshold. Increasing the detection threshold can reduce the number of false positive detections, as it only considers detections with a high confidence score. Also the false-positives could be avoided by data augmentation by expanding the training dataset to increase the diversity of examples. This can help the model to generalize better and reduce false positive detections.

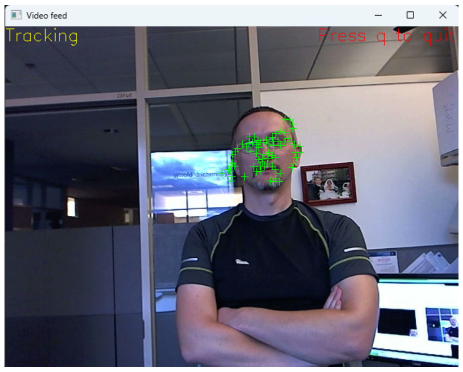


Figure 8: Tracking outcome example 1

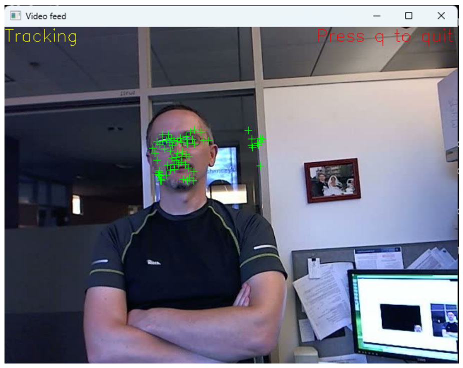


Figure 9: Tracking outcome example 2

CODE SNIPPET FROM: *face_tracking.py*

```
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-##
                    (x, y, w, h) = faces[0]
                    roi_gray = gray[y : y+h, x :  x+w]  # replaced
                    ##-your-code-ends-here-##

                    # 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-##
                    for i in range(len(p0)):
                        p0[i][0] += x
                        p0[i][1] += y
                    ##-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 |
                                                            flags=cv2.OPTFLOW_LK_GET_MIN_EIGEN
                                                            minEigThreshold=0.00025)

                # Select good points. Use isFound to select valid found points from p1
                ##-your-code-starts-here-#
                points = p1[isFound[:, 0] == 1, :]
                # Convert float tuples to float int
                good_points = [(int(element[0]), int(element[1])) for element in points]
                ##-your-code-starts-here-##

                # Draw points using e.g. cv2.drawMarker
                ##-your-code-starts-here-##
                for point in good_points:
                    cv2.drawMarker(img,
                                point,
                                color=(0, 255, 0),
                                markerType=cv2.MARKER_CROSS,
                                markerSize=10)
                ##-your-code-ends-here-##q

                # Update p0 (which points should be kept?) and gray_prev for
                # next iteration
                ##-your-code-starts-here-##
                p0 = points
                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()
```