Internship Report

Ingvild Kvalsvik | s2237121

Master Human Factors & Engineering Psychology

19.07.2023

Internship organisation: University of Twente, CODE department

Duration of internship: 15.05.2023 - 19.07.2023 (280 hours)

Supervisor: Martin Schmettow

Summary

The goal of the internship was to develop a screen tracker to be used in a research project for children. In the project, there is a need to know whether a child is looking at a tablet or not, so the idea was to create a tablet tracker that can determine whether the child's head is directed at the tablet screen or not. To make the tracker, two parts were needed: creating a headset with a camera fixated to it that the child can wear and creating a program that can determine the position of the tablet in space.

To create the headset, the plastic insert of a child-size helmet was used. This headset can be easily adjusted to fit the head of a child by using the rotating mechanism in the back and is comfortable to wear. A camera mount for a 7mm endoscope camera was 3D printed and glued to the headset. The program can track a tablet in space using an AruCo marker (small markers similar to a QR code) in each corner of the tablet. Based on the position of the tablet in space, it is possible to infer whether a person is looking at the screen or not.

Introduction

Two researchers at the CODE department developed an app for children as part of their research, and they need to verify how much children are actually looking at the tablet screen. To do so, they tried using the Tobii eye-tracking glasses. However, these glasses are not made for children (the glasses have the dimensions of 153 × 168 × 51 mm (Tobii, 2022), making them too large for a five-year-old head) and so this approach did not work as intended. Therefore, an alternative solution was needed. The idea was to create a screen tracker that uses a custom algorithm to track whether the child's head is in line with the tablet, with the assumption that if the tablet is skewed or rotated, the child is not looking at the tablet. Furthermore, to complete the project, a headset that a camera could be mounted on was needed.

Methods

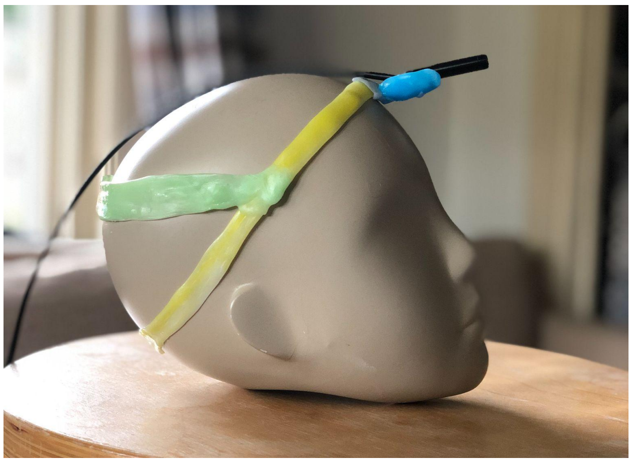
Headset

The initial plan was to 3D model a headset in Fusion 360 that could be 3D printed flat. To facilitate making the headset, the head of a five-year-old child was 3D scanned using the Creality CR-Scan Ferret. To ensure the scan would be more accurate, the child was wearing a textile swimming cap as the scanner has some trouble scanning soft materials, such as hair (Creality, 2023). The final 3D scan resulted in a 3D model of the head that could be used for making a model of the headband that fit the dimensions of a five-year-old's head.

A prototype was created using moldable polymorph plastic moulded on a small mannequin head to visualise the form of the headband to make it easier to 3D model the headband (see Fig 1).

Figure 1

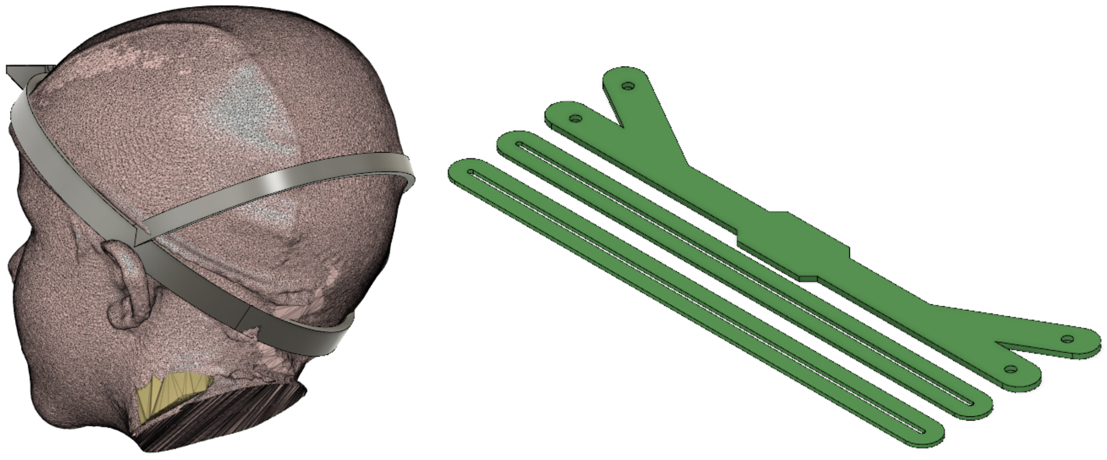
Prototype of Headband made of Polymorph Plastic



Using the 3D model of the child’s head, it was possible to use the mesh intersection tools in Autodesk Fusion 360 to create a simple prototype based on the curvature of the head (see Fig. 2). Using this prototype, a flat model was created. The initial idea was to create a headband that could close using snaps such as the ones found on baseball caps. The model of the headband was divided into three parts of approximately 30 cm each to account for the dimensions of a 3D printer. However, if using a regular 3D printer, there is a risk that the snaps could break over time due to the layers being printed horizontally. Therefore, the adjusted idea was to make a headset with screws for inserts that could be soldered in place to the headset using a soldering iron (see Fig. 2).

Figure 2

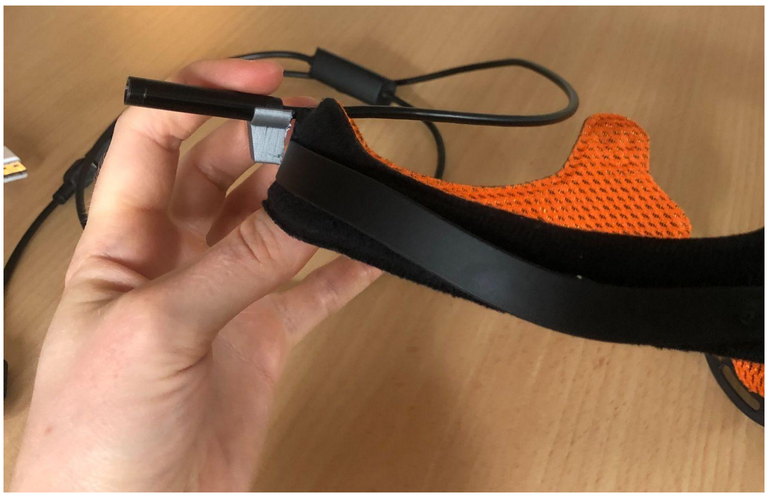
3D model of Headset Prototype using 3D Scan and Flat Model of Headband



However, the 3D model of the headset did not end up being 3D printed as a simpler solution was found, namely making the final headband out of the plastic insert of a child-sized helmet. The plastic insert was pulled out of the helmet, and some protrusions were cut off for aesthetic purposes. A simple camera mount for a 7mm endoscope camera was first 3D modelled in Fusion 360 and then 3D printed and mounted to the headset using superglue (see Fig. 3).

Figure 3

3D-printed Camera Mount

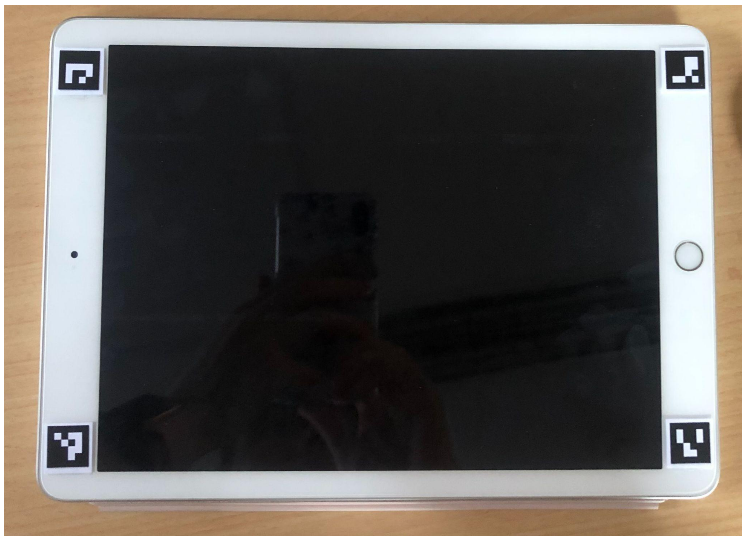


Programming

The program was created based on code sourced from GitHub and using ChatGPT to get information on existing libraries and functions to facilitate the programming process. The program runs in Python, version 3.10.9 using PyCharm 2023.1.2. Specifically, the code by GitHub user Datigrezzi (2019) was modified and expanded on to create the final program. The code by Datigrezzi is a program that can track an object with AruCo markers using video input and a reference image with AruCo markers situated on an object. For the tablet tracking program, four AruCo markers were printed on regular paper and taped to a tablet using double-sided tape, with a small white border on all sides of the markers for better detection.

Figure 4

Position of AruCo Markers on Tablet



The output of the program is a video stream with a rectangle drawn between the four AruCo markers situated in each corner of the tablet that tracks the AruCo markers. However, this program was still missing several functions that the tablet tracker program needed. Therefore, the following functionalities were implemented (see Appendix I for the entire code):

1. More filtering was added to improve the detection of the markers (blurring, thresholding, morphological operations)
2. Calculate the angles of each corner of the rectangle to be able to determine when the tablet is skewed
3. Calculate the rotation angle of the tablet to be able to tell if the tablet is being rotated
4. The program had an issue that would crash the program if it could not solve the homography matrix. Therefore, a feature was implemented to have the program keep running if this is encountered
5. Display text of alignment status (tablet aligned / tablet not aligned) based on the rotation angle and the angle of the four corners
6. Replace the parser arguments that the original code used (instead, features are embedded directly into the code)
7. Replace the custom markers that the original code used with a pre-defined 4x4 AruCo marker dictionary
8. Save all data (time stamp, alignment status, rotation angle, angle of each of the four corners) to a CSV file once per second
9. Save the video output in mp4 format

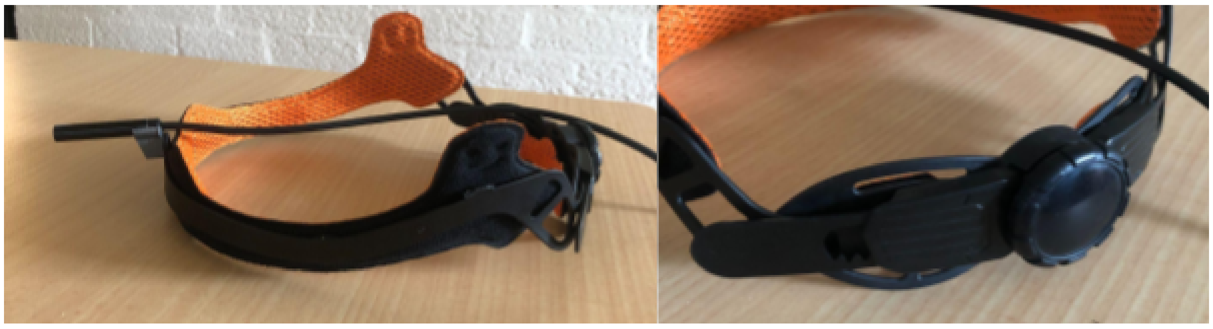
Results

Headset

The final headset is made of the plastic insert of a helmet with a camera mount superglued to the front (see Fig. 5). The headset can easily be adjusted and tightened accordingly using the rotating mechanism in the back (see Fig. 5). This allows the headset to be used for different head types and sizes. The camera is a 7mm endoscope camera that is placed in the camera mount. The camera connects via USB to the computer where the program is run.

Figure 5

Headset and Rotating Mechanism



Program

The final program can track a tablet in space using four AruCo markers, one situated in each corner of the tablet (outside of the screen see Fig. 4). The program uses a reference picture with the AruCo markers to detect them. The program draws a rectangle between the four markers and calculates the angle of each corner of the rectangle as well as the rotation angle of the tablet. While the program is running, a video stream is shown with the following information overlaid (see Fig. 6 for an example):

1. The detection of the markers (each marker will be outlined and have an ID displayed if they are detected, or in the case of no marker being detected, the text "No AruCo markers detected" will be displayed)
2. The angle of each corner of the tablet (1 = Upper Left corner, 2 = Upper Right, 3 = Lower Right, 4 = Lower Left)
3. The rotation angle of the tablet

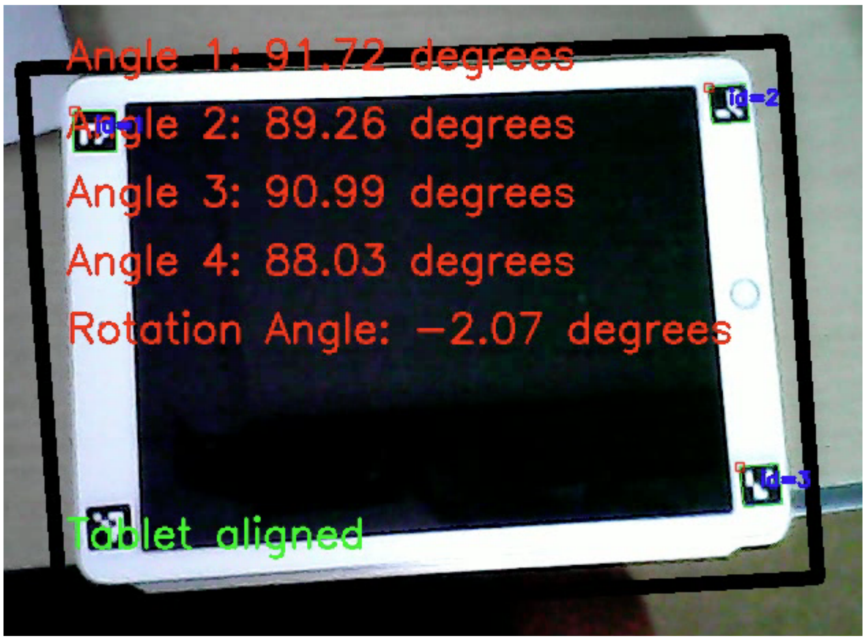
Currently, the program displays the text "Tablet is aligned" in green if all angles are less than 100 degrees and the rotation angle is between -30 and 30 degrees. However, these values were chosen based on intuition and might need to be adjusted based on what is actually considered to be aligned. If these values are not in the desired range, the text "Tablet is not aligned" is displayed in red.

Additionally, the program saves the video output as an mp4 format (this video has the overlaid text on it, the same as can be seen when the program is running) and a CSV file with the following information:

1. A timestamp (Y/M/D_H/M/S) which is updated every second
2. The alignment status (*tablet is aligned* or *tablet is not aligned*, or in the case of no markers being detected, *No AruCo markers detected*)
3. The rotation angle of the tablet
4. The angles of each of the four corners of the tablet

Figure 6

Example of Video Output from the Program



```
Angle 1: 91.72 degrees
Angle 2: 89.26 degrees
Angle 3: 90.99 degrees
Angle 4: 88.03 degrees
Rotation Angle: -2.07 degrees
Tablet aligned
```

Discussion

Headset

The final headset is made of the plastic insert of a helmet and can be adjusted to fit. The current headset is made for children but can be adjusted to fit a small adult size head, making it possible to use it for other purposes (e.g. BCI). For future iterations, 3D printing a headset could be considered, although it is important to keep in mind that 3D printing comes with its challenges such as having to print with supports or keeping in mind the dimensions of a 3D printer. Furthermore, the adjustment mechanism on the helmet insert is very easy to adjust compared to some of the design ideas for the 3D-printed version of a headset. Alternatively, another solution could be to use straps or textiles to create a headset that would be comfortable to wear and adjustable.

3D scanning

The 3D scanning method was found to work rather well and could, in the future, be used for creating customisable products or devices. However, it was found that scanning more complex objects worked significantly better than objects with little texture or dimensions (e.g. a coffee cup). Furthermore, scanning soft and deformable materials such as hair was challenging and did not grant a very good result. To mitigate this issue, using a textile swimming cap was found to be a good solution that can be recommended when 3D scanning a head.

Program and camera

The program can track a tablet in space using AruCo markers and determine whether it is aligned or not based on the angles of the four corners. It should be noted that the program is based on the assumption that if the tablet is not skewed or rotated, the person is looking at the screen. However, this is not a guarantee that the participant is actually looking at the screen. In reality, a participant could have their head pointed straight towards the tablet screen but move their eyes to another part of the room, which will not be caught by the program. Furthermore, the program does not determine at which part of the screen someone is looking. For this purpose, an eye tracker would be needed to be used.

There are some further limitations to the program and the hardware that is used. One limitation is that the program cannot accurately detect the AruCo markers if there is quick motion from the camera. Furthermore, the program is somewhat sensitive to lighting conditions and glare from the tablet screen. In harsh lighting conditions, it was experienced that the AruCo markers were not always detected. This issue was not experienced when using a simple sheet of paper with four markers on it, so it is likely due to the reflective material of the tablet or the glare from the screen.

Additionally, noise in the background might at times distort the detection as the program might incorrectly identify background noise as an AruCo marker. However, this issue was not frequently encountered in testing.

There are four AruCo markers, but it should be noted that the program does not always detect all of them correctly. However, even with only two AruCo markers being detected (this also applies if a marker is being covered up by a hand), the program can still determine the position of the tablet.

Additionally, the endoscope camera that is currently used comes with some limitations. The camera offers a relatively low resolution, which in turn affects the quality of the detection of the markers, especially if the markers are situated far away from the camera. Furthermore, there is no possibility of adjusting the exposure (as far as I am aware), which can be an issue if the glare of the screen disturbs the image, again leading to the markers not being detected correctly. An additional disadvantage is that the camera is connected to a computer using a USB wire, which limits the possibility of movement for the participant. However, this could also be partially solved by using a longer wire.

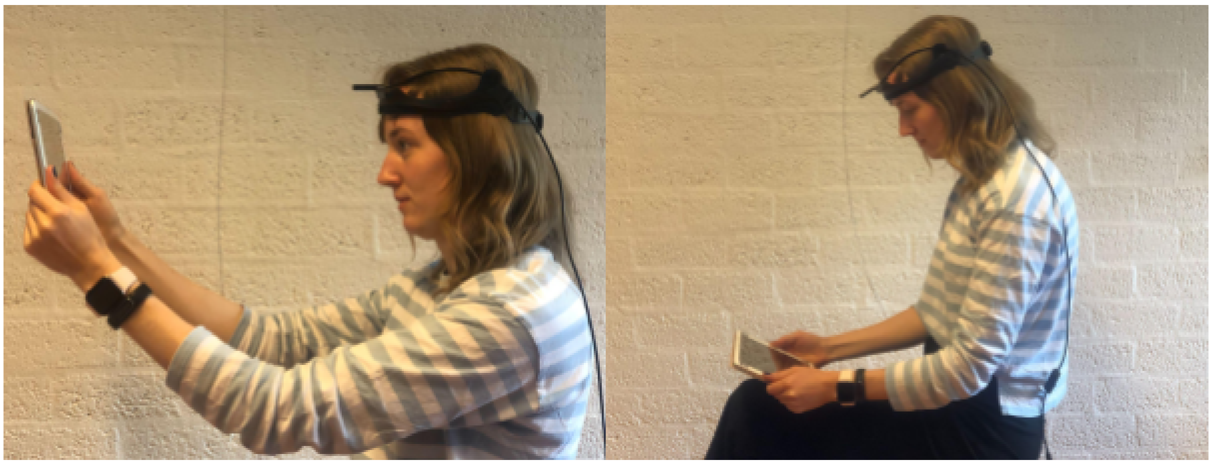
Future recommendations

One recommendation is to look into the possibility of using a different camera for the headset as the endoscope camera comes with several disadvantages. The main advantages of the endoscope camera are that it is cheap (one can be bought for around 20 euros) compared to other commercially available cameras, and it is small. However, an alternative solution could be to use a small "spy" camera with storage on an SD card as some of these cameras offer better resolution and no wire (albeit at a steeper price than the endoscope camera). However, a disadvantage of such a camera is that the video output would have to be run through the program after the experiment, meaning that there would be no possibility of seeing what is happening in real time.

Another recommendation for future iterations of the headset is to make a camera mount that can be adjusted on the vertical axis. Now, the camera faces straight forward from the head which is not an issue if you are looking straight at a screen. However, if you are holding the tablet in, for example, your lap, it might not be in line with the camera (see Fig. 7). This issue could be solved by having the camera fixated to a joint that could be used to adjust the camera based on how the participant is holding the tablet. Alternatively, instructing the participants to only hold the tablet in such a way that the camera is directed at the tablet could be another solution.

Figure 7

Demonstration of the Angle of the Camera



Note. In both instances, the gaze was on the tablet. However, only in the instance on the left would the camera be directed at the tablet screen.

A way the program could be improved is to create a menu screen to make the program easier to navigate for people who are not familiar with programming. In the current code, some adjustments need to be made to ensure all information is stored correctly which needs to be adjusted directly in the code. Having a menu pop up when you start the program could make it easier to use. However, this is only a minor suggestion and is not necessary for the functionality of the program.

Another recommendation is to include a calibration state in the program. At the moment, the angles of the corner and the rotation angle are set to change the alignment state (tablet aligned/tablet not aligned) if any angle is above 100 degrees and the rotation angle is between -30 and 30 degrees. A suggestion is to have a calibration state where the participant could hold the tablet in their desired position, and then the program could calculate, based on this position, what the threshold values should be.

It is also recommended to conduct a pilot test with a couple of participants to determine what the optimal threshold values are. One suggestion is to have the participants look at the screen for some time, and then instruct them to divert their attention to somewhere else in the room to determine how they hold the tablet when they are not looking at it. By completing such a pilot test it would be possible to better determine an optimal threshold value.

Conclusion

The created headset and program is a solution to determining when a person is looking at a tablet or not by determining the position of a tablet in space relative to a head. There are some limitations to this solution, such as the low resolution of the camera and the angle of the camera only allowing the tablet to be tracked as long as it is directly in front of the head. Furthermore, the functionality of the found solution relies on the assumption that as long as the tablet is not skewed or rotated, the person is looking at the screen, whereas in reality, this might not always be the case. However, the headset and the program are, overall, a simple, low-cost solution to the problem.

References

Creality. (2023). CR-Scan Ferret 3D Scanner User Manual (Version July 3, 2023). Retrieved from: https://img.staticdj.com/ea8c32866614771e6967bee8d020ef8f.pdf?spm=..page_2747139.download_support_1.1&spm_prev=..product_e5b9e2b0-d48d-4aca-9010-4414919e7070.nav_link_store_1.1

Datigrezzi. (2019). Python AruCo Highlight Tracking. *GitHub.* Retrieved from: https://github.com/datigrezzi/PythonAruco/blob/master/aruco_highlight_tracking.py

OpenCV. (n.d.). Detection of AruCo Markers. Retrieved from: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

Tobii. (2022). Tobii Pro Glasses 3. Retrieved from: https://www.tobii.com/products/eye-trackers/wearables/tobii-pro-glasses-3#field

Appendix 1

Code

```
import numpy as np
import cv2
import datetime
import csv
import math
from collections import deque

def which(x, values):
indices = []
for ii in list(values):
if ii in x:
indices.append(list(x).index(ii))
return indices

# Important!
# Some lines of code need to be updated so the output data is saved where you
want it to be
# therefore, check and adjust line 29, 69, and 241 before you run the code


# load video file
cap = cv2.VideoCapture(0)

# Generate a timestamp string
timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")

# Specify the output filename with the timestamp
#  In  the  following  format  (might  need  to  be  changed  depending  on  OS  of
computer):
# f"/Users/yourname/nameoffolder/videoname_{timestamp}.mp4"
output_filename                                                              =
f"/Users/Ingvild/screentracker/output_video_{timestamp}.mp4"
fourcc = cv2.VideoWriter_fourcc(*'mp4v')

# Get the webcam stream dimensions
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

```
# Create the VideoWriter object to save the video
video_out  =  cv2.VideoWriter(output_filename,  fourcc,  10,  (frame_width,
frame_height))

# Create a list to store the alignment data
alignment_data = []

# Variables to keep track of the previous second and alignment status
previous_alignment_status = None
previous_second = None

# min_marker_perimeter = 0.03
# max_marker_perimeter = 0.05

# setting up dictionary of aruco markers and adding parameters for detection
aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_250)
parameters = cv2.aruco.DetectorParameters()
# parameters.minMarkerPerimeterRate = min_marker_perimeter
# parameters.minMarkerPerimeterRate = max_marker_perimeter

# refines the parameters for better detection
parameters.cornerRefinementMethod = 3
parameters.errorCorrectionRate = 0.5

# Preprocessing parameters
resize_width = 640 # Adjust the desired width
blur_kernel_size = (7, 7) # Adjust the kernel size for blurring
threshold_value = 150 # Adjust the threshold value
morph_kernel_size  =  (7,  7)  #  Adjust  the  kernel  size  for  morphological
operations

# load reference image
# remember to use reference image of actual tablet, it makes a difference!
# upload reference image to a folder and specify the filepath like this:
#  "/Users/yourname/nameoffolder (if image is in a subfolder)/imagename.jpeg"
(or .jpg, please check!)
refImage        =        cv2.cvtColor(cv2.imread("/Users/Ingvild/ipad6.jpeg"),
cv2.COLOR_BGR2GRAY)

# detect markers in reference image
```

```
refCorners,    refIds,    refRejected    =    cv2.aruco.detectMarkers(refImage,
aruco_dict, parameters=parameters)
# create bounding box from reference image dimensions
rect = np.array([[[0, 0],
[refImage.shape[1], 0],
[refImage.shape[1], refImage.shape[0]],
[0, refImage.shape[0]]]], dtype="float32")

# noise reduction
h_array = deque(maxlen=5)

while True:
# read next frame from VideoCapture
ret, frame = cap.read()
if frame is not None:
# convert frame to gray scale
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Apply Gaussian blur to reduce noise
gray = cv2.GaussianBlur(gray, blur_kernel_size, 0)

# Apply thresholding to convert the image to binary format
_, threshold = cv2.threshold(gray, threshold_value, 255, cv2.THRESH_BINARY)

# Apply morphological operations to refine the marker regions
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, morph_kernel_size)
threshold = cv2.morphologyEx(threshold, cv2.MORPH_CLOSE, kernel)

# detect aruco markers in gray frame
res_corners,    res_ids,    _    =    cv2.aruco.detectMarkers(gray,    aruco_dict,
parameters=parameters)

# if markers were detected
if res_ids is not None and len(res_ids) > 0:
# find which markers in frame match those in reference image
idx = which(refIds, res_ids)
alignment_status = "Tablet aligned"

# Make an empty list for the rotation angle and corner angles
rotation_angle = 0.0
angles = [0.0, 0.0, 0.0, 0.0]
```

```
# if any detected marker in frame is also in the reference image
if len(idx) > 0:
# flatten the array of corners in the frame and reference image
these_res_corners = np.concatenate(res_corners, axis=1)
these_ref_corners = np.concatenate([refCorners[x] for x in idx], axis=1)

# Estimate homography matrix
if these_res_corners.shape[0] == these_ref_corners.shape[0]:
try:
h,  s  =  cv2.findHomography(these_ref_corners,  these_res_corners,  cv2.RANSAC,
5.0)

# in case the program cannot calculate the homography matrix
except cv2.error as e:
print("Homography calculation failed:", str(e))
alignment_status = "Homography calculation failed"

else:
# for smoothing
h_array.append(h)
this_h = np.mean(h_array, axis=0)

# transform the rectangle using the homography matrix
newRect    =    cv2.perspectiveTransform(rect,    this_h,    (gray.shape[1],
gray.shape[0]))

# draw the rectangle on the frame
frame = cv2.polylines(frame, np.int32(newRect), True, (0, 0, 0), 10)

# Calculate the rotation angle
top_vector = newRect[0][1] - newRect[0][0]
rotation_angle = math.degrees(math.atan2(top_vector[1], top_vector[0]))

# Display the rotation angle
cv2.putText(frame,  f"Rotation  Angle:  {rotation_angle:.2f}  degrees",  (50,
250),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

# calculate the angles at each corner of the quadrilateral
rect_points = np.squeeze(newRect)
```

```
# calculate the angles between the lines
angles = []
for i in range(4):
pt1 = rect_points[i]
pt2 = rect_points[(i + 1) % 4]
pt3 = rect_points[(i + 2) % 4]

# calculate the vectors of the lines
v1 = pt1 - pt2
v2 = pt3 - pt2

# calculate the angle between the lines using dot product
angle = np.arccos(np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2)))
angle_deg = np.degrees(angle)
angles.append(angle_deg)

# check if any angle is above 100 degrees and if rotation angle is between
-30 and 30 degrees
# These values need to be updated depending on what values are optimal
if any(angle > 100 for angle in angles) or rotation_angle > 30 or
rotation_angle < -30:
cv2.putText(frame, "Tablet not aligned", (50, 400), cv2.FONT_HERSHEY_SIMPLEX,
1,
(0, 0, 255), 2)
alignment_status = "Tablet not aligned"

# if angles are not above 100 deg and rotation angle is okay, tablet is
aligned
else:
cv2.putText(frame, "Tablet aligned", (50, 400), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 255, 0), 2)
alignment_status = "Tablet aligned"

# display the angles on the screen
for i, angle in enumerate(angles):
angle_text = f"Angle {i + 1}: {angle:.2f} degrees"
cv2.putText(frame, angle_text, (50, 50 + 50 * i), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 0, 255), 2)

else:
```

```
# No ArUco markers found
# Display text on screen
cv2.putText(frame,    "No    ArUco    markers    detected",    (50,    50),
cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 0, 255), 2)

# No ArUco markers detected
# Update alignment status
alignment_status = "No ArUco markers detected"
continue

# Updating alignment status once every second
if alignment_status != previous_alignment_status:
current_time = datetime.datetime.now()
current_second = current_time.second

# Making sure information is only appended to the list once every second
if current_second != previous_second or previous_second is None:
alignment_data.append(
[current_time.strftime("%Y-%m-%d         %H:%M:%S"),         alignment_status,
rotation_angle] + angles)
previous_second = current_second

else:
# No ArUco markers found
cv2.putText(frame,    "No    ArUco    markers    detected",    (50,    50),
cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 0, 255), 2)

# Update alignment atatus to No ArUco markers detected
alignment_status = "No ArUco markers detected"

# Get the current timestamp
current_time = datetime.datetime.now()
current_second = current_time.second

# Check if the current second is different from the previous second
if current_second != previous_second or previous_second is None:
# Append the alignment data for the current second
alignment_data.append([current_time.strftime("%Y-%m-%d            %H:%M:%S"),
alignment_status])
```

```
# Update the previous second and alignment status
previous_second = current_second
previous_alignment_status = alignment_status

# draw detected markers in frame with their ids
cv2.aruco.drawDetectedMarkers(frame, res_corners, res_ids)

# Write the processed frame to the output video
video_out.write(frame)

# Display the resulting frame
cv2.imshow('frame', frame)

# exit if q is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
break

# Save the data to a CSV file
# again, update filepath to where you want to save the csv data
tracker_data_csv_filename                                                    =
f"/Users/Ingvild/screentracker/tracker_data_{timestamp}.csv"
with open(tracker_data_csv_filename, mode='w', newline='') as file:
writer = csv.writer(file)
writer.writerow(["Timestamp", "Alignment Status", "Rotation Angle", "Upper
Left Angle", "Upper Right Angle",
"Lower Right Angle", "Lower Left Angle"])
writer.writerows(alignment_data)

# When everything is done, release the capture and video output
if video_out is not None:
video_out.release()
cap.release()

# close cv2 window
cv2.destroyAllWindows()
```