ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ
КАФЕДРА ЕОМ

Кваліфікаційна робота
Другий рівень (магістр)

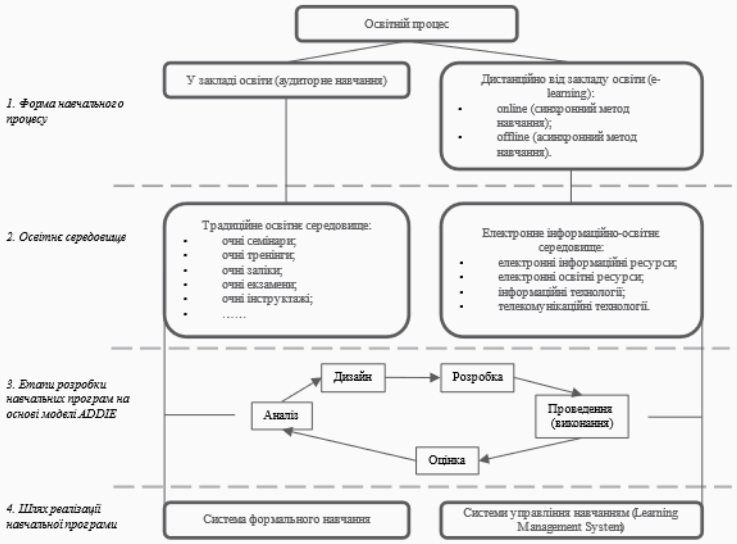
Методи оцінки напряму погляду людини в контексті LMS

Автор
Ляпін Я. А.
ст. гр. СПм-23-2

Керівник
Барковська О.Ю.
доц. каф. ЕОМ

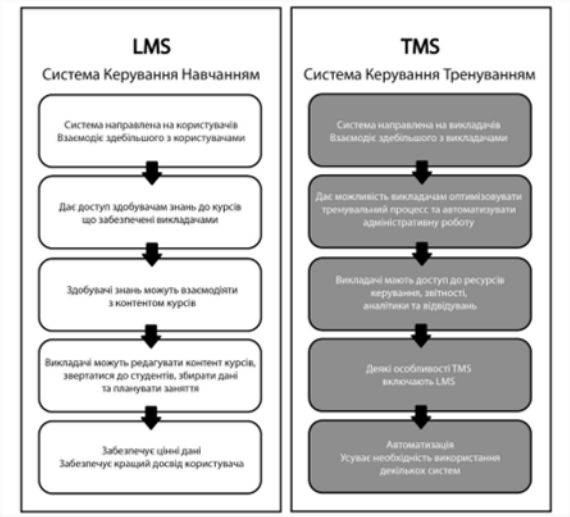
ОГЛЯД ПРОБЛЕМНОЇ ОБЛАСТІ

1. Аудиторні заняття
2. Дистанційне навчання



АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1. Trainual
2. Moodle
3. Google Classroom



МЕТА І ЗАВДАННЯ РОБОТИ

Мета роботи

Метою кваліфікаційної роботи є дослідження та вдосконалення методів оцінки напряму погляду людини в контексті створення адаптивної системи управління навчанням .

Завдання роботи

- ✓ визначення можливостей персоналізованого використання LMS;
- ✓ оцінка точності визначення напряму погляду на основі існуючих методів ;
- ✓ розробка моделі адаптивної системи управління навчанням на основі тривалості фокусування та напряму погляду ;
- ✓ тонке налаштування параметрів нейромережевих моделей для визначення напряму погляду ;
- ✓ оцінка впливу умов зйомки на точність визначення напряму погляду .

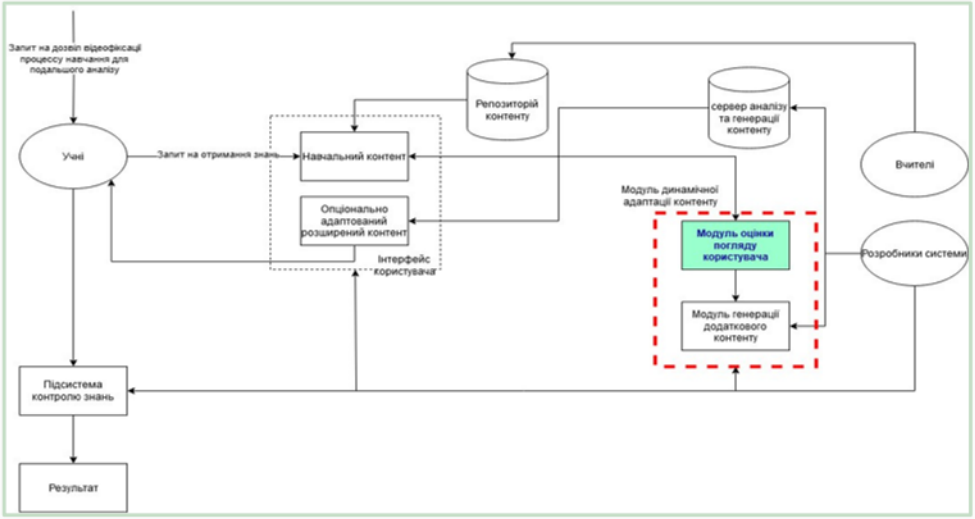
АКТУАЛЬНІСТЬ РОБОТИ

- складність викладення матеріалу ;
- одноманітність викладення матеріалу ;
- низька зацікавленість здобувачів знань у процесі ;
- відсутність персоналізації та обмежена інтерактивність ;
- низька мотивація користувачів ;
- низька ефективність для різнорівневих груп ;
- складність відстеження прогресу учнів /

Тому персоналізація та адаптація навчального матеріалу під учня є задачею актуальною!

5

ЗАПРОПОНОВАНА МОДЕЛЬ



6

ВИКОРИСТАННЯ ДАТАСЕТІВ

1. UnityEyes:
   - 1,04 ГБ;
   - 129285 зображень .
2. Власний підготовлений тестовий датасет :
   - 59 МБ;
   - 1163 зображень .

7

Дослідження нейромережевих архітектур конволюційного типу

1 LeNet

Кількість епох навчання	Тестова точність детектування напряму погляду	Значення функції втрат	Час навчання, секунд
1	0.5102	0.3634	226
5	0.6187	0.2431	1074
10	0.7894	0.1919	2175
15	0.7961	0.1872	3127
25	0.8072	0.1820	5621
50	0.8086	0.1812	11278

2 VGGNet

Кількість епох навчання	Тестова точність детектування напряму погляду	Значення функції втрат	Час навчання, секунд
1	0.6097	0.3152	327
5	0.7863	0.1940	1655
10	0.8149	0.1862	3221
15	0.8231	0.1823	4826
25	0.8316	0.1790	8284
50	0.8331	0.1779	16582

Точність нейромережевого детектування напряму погляду на основі CNN при розподілі датасету у співвідношенні тренувальний : валідаційний : тестовий набори = 70% : 15% : 15%

8

Дослідження нейромережевих архітектур конволюційного типу

3 AlexNet

Кількість епох навчання	Тестова точність детектування напряму погляду	Значення функції втрат	Час навчання, секунд
1	0.5914	0.3218	410
5	0.7122	0.2138	1978
10	0.7923	0.1928	4021
15	0.8059	0.1862	6031
25	0.8093	0.1829	10123
50	0.8116	0.1791	21091

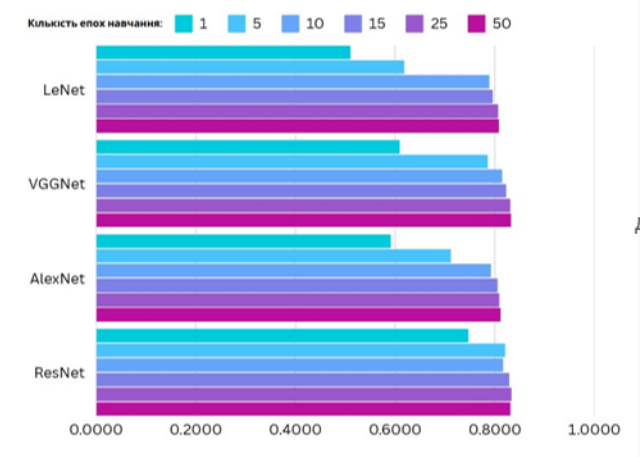
4 ResNet

Кількість епох навчання	Тестова точність детектування напряму погляду	Значення функції втрат	Час навчання, секунд
1	0.7472	0.1913	1086
5	0.8213	0.1912	5671
10	0.8168	0.1872	10791
15	0.8294	0.1863	16180
25	0.8337	0.1836	26291
50	0.8318	0.1747	50137

Точність нейромережевого детектування напряму погляду на основі CNN при розподілі датасету у співвідношенні тренувальний : валідаційний : тестовий набори = 70% : 15% : 15%

9

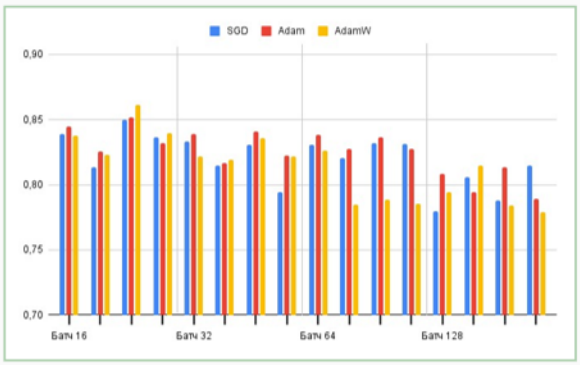
Визначення архітектур та параметрів для подальших тонких налаштувань



Точність нейромережевого детектування при розподілі датасету у співвідношенні 70% : 15% : 15%

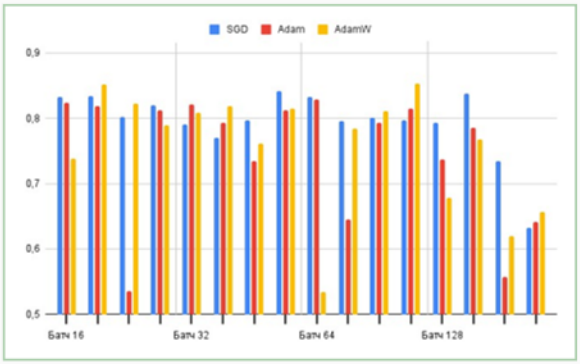
10

РЕЗУЛЬТАТИ ТОНКИХ НАЛАШТУВАНЬ АРХІТЕКТУРИ VGGNet ТА ResNet



1. ResNet

2. VGGNet



11

МЕТОДОЛОГІЯ ОЦІНКИ ВПЛИВУ ЗОВНІШНІХ УМОВ НА ТОЧНІСТЬ ВИЗНАЧЕННЯ НАПРЯМКУ ПОГЛЯДУ

Гарне освітлення (300 люмен)		
	Відстань від камери 30 см	Відстань від камери 100 см
Кут нахилу 0	Експеримент 1	Експеримент 4
Кут нахилу 15	Експеримент 2	Експеримент 5
Кут нахилу 30	Експеримент 3	Експеримент 6
Погане освітлення (30 люмен)		
Кут нахилу 0	Експеримент 7	Експеримент 10
Кут нахилу 15	Експеримент 8	Експеримент 11
Кут нахилу 30	Експеримент 9	Експеримент 12

Вплив на точність визначення напряму погляду

12

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПРИ ГАРНОМУ ОСВІТЛЕННІ (300 лм/м^2)



VGGNet	Відстань від камери = 30 см			Відстань від камери = 100 см		
	Кут:0°	Кут:15°	Кут:30°	Кут:0°	Кут:15°	Кут:30°
Точність, %	0.8151	0.8134	0.7913	0.7818	0.7282	0.7139
Значення функції втрат, %	0.1813	0.1884	0.2017	0.1961	0.2184	0.2131
Час оцінки, с	471	464	473	484	467	472

ResNet	Відстань від камери = 30 см			Відстань від камери = 100 см		
	Кут:0°	Кут:15°	Кут:30°	Кут:0°	Кут:15°	Кут:30°
Точність, %	0.8380	0.8264	0.8209	0.8063	0.7937	0.7962
Значення функції втрат, %	0.1726	0.1791	0.1786	0.1916	0.1986	0.1949
Час оцінки, с	532	561	568	541	545	559

13

РЕЗУЛЬТАТИ ТЕСТУВАННЯ ПРИ ПОГАНОМУ ОСВІТЛЕННІ (30 лм/м^2)

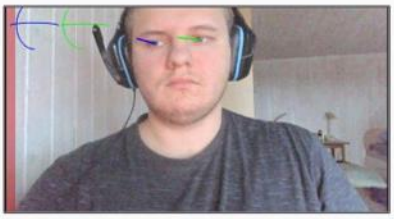


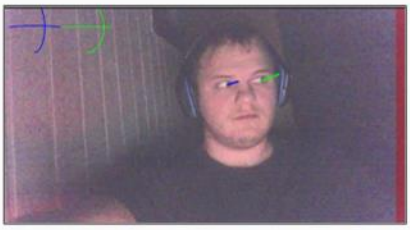
VGGNet	Відстань від камери = 30 см.			Відстань від камери = 100 см.		
	Кут:0°	Кут:15°	Кут:30°	Кут:0°	Кут:15°	Кут:30°
Точність, %	0.7951	0.7814	0.7847	0.6713	0.6927	0.6876
Значення функції втрат, %	0.2004	0.1916	0.1932	0.2532	0.2476	0.2580
Час оцінки, с	481	462	475	468	469	474

ResNet	Відстань від камери = 30 см.			Відстань від камери = 100 см.		
	Кут:0°	Кут:15°	Кут:30°	Кут:0°	Кут:15°	Кут:30°
Точність, %	0.8184	0.7931	0.7946	0.7631	0.7541	0.7591
Значення функції втрат, %	0.1801	0.1935	0.1985	0.2153	0.2236	0.2209
Час оцінки, с	553	548	561	543	575	550

14

ОТРИМАНІ РЕЗУЛЬТАТИ





15

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було досліджено та вдосконалено методи оцінки напряму погляду людини. Спираючись на результати проведених досліджень розроблено програмний модуль, який може бути інтегровано в системи управління навчанням, в системи медичної діагностики психоемоційного стану, в маркетингові системи тощо.

Вирішено наступні задачі :

- на основі аналізу функціональних можливостей сучасних LMS та TMS, визначено місце та роль методів штучного інтелекту для персоналізації навчального матеріалу ;
- проведена оцінка точності визначення напряму погляду на основі існуючих методів показала переваги використання нейромережевих архітектур VGGNet та ResNet, оскільки забезпечують базову точність визначення напряму погляду на 3% вище, ніж аналоги LeNet та Alex Net ;
- розроблена модель адаптивної системи управління навчанням на основі тривалості фокусування та напряму погляду, яка забезпечує персоналізацію навчального матеріалу та можливість поглиблення знань ;
- виконана оцінка впливу умов зйомки на точність визначення напряму погляду довела, що запропоноване тонке налаштування параметрів нейромережевих архітектур дають можливість детектування напряму погляду при гарному освітленні із точністю 83%, при поганому освітленні із точністю 81%. Також показано, що відстань від камери та кут нахилу голови знижують точність лише на 5%.

Подальшими кроками розвитку запропонованої системи є впровадження її у якості модулю адаптації контенту у LMS.

16

АПРОБАЦІЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Стаття у журналі, матеріали якого індексуються у наукометричній барі Web-Of-Science

Olesia Barkovska, **Yaroslav Liapin,** Tetiana Muzyka, Ihor Ryndyk, Pavlo Botnar, "GAZE DIRECTION MONITORING MODEL IN COMPUTER SYSTEM FOR ACADEMIC PERFORMANCE ASSESSMENT. CIVIL LAW ASPECT" , Information Technologies and Learning Tools, 2024, Vol 99, №1, 63-75pp DOI: 10.33407/itlt.v99i1.5503

Тези доповіді:

2. **Ляпін Я.А.,** Барковська О.Ю. Модель моніторингу напрямку погляду в системі комп'ютерного контролю знань. // Проблеми інформатизації. Тези доповідей одинадцятої міжнародної науково-технічної конференції. - ЧДТУ, ВА ЗС АР, УТіГН, НТУ "ХПІ", ХНУРЕ, "ПД ПКНДІ АП", 2023. – 16 – 17 листопада 2023 року. – с.18.

3. **Ляпін Я.А.,** Барковська О.Ю. Методи оцінки напряму погляду людини в контексті LMS. // Проблеми інформатизації. Тези доповідей дванадцятої міжнародної науково-технічної конференції. - ЧДТУ, ВА ЗС АР, УТіГН, НТУ "ХПІ", ХНУРЕ, "ПД ПКНДІ АП", 2024. - 21 – 22 листопада 2024 року. – с.68.

ДОДАТОК Б

Б.1 Файл GazeProcessor.py, що містить відповідний клас

```
import mediapipe as mp
import cv2
import time
from landmarks import *
from face_model import *
import keras
from AffineTransformer import AffineTransformer
import numpy as np

number=0
number2=0

model_path = "./face_landmarker.task"
model_analyze="./ResNet.h5"

model = keras.api.models.load_model("lenet_trained.h5",
compile=False)

model.compile(
    optimizer=keras.api.optimizers.SGD()

)

FaceLandmarker = mp.tasks.vision.FaceLandmarker


class AffineTransformer:

    def __init__(self, m1_points, m2_points, m1_hor_points,
m1_ver_points, m2_hor_points, m2_ver_points):

        self.scale_factor = self._get_scale_factor(
            np.array(m1_hor_points),
            np.array(m1_ver_points),
            np.array(m2_hor_points),
            np.array(m2_ver_points)
        )

        scaled_m2_points = m2_points * self.scale_factor

        retval, M, inliers = cv2.estimateAffine3D(m1_points,
scaled_m2_points)
        if retval:
            self.success = True
            self.transform_matrix = M
        else:
```

```
            self.success = False
            self.transform_matrix = None

    def _get_scale_factor(self, m1_hor_points, m1_ver_points,
m2_hor_points, m2_ver_points):

        m1_width = np.linalg.norm(m1_hor_points[0] -
m1_hor_points[1])
        m1_height = np.linalg.norm(m1_ver_points[0] -
m1_ver_points[1])
        m2_width = np.linalg.norm(m2_hor_points[0] -
m2_hor_points[1])
        m2_height = np.linalg.norm(m2_ver_points[0] -
m2_ver_points[1])
        scale_width = m1_width / m2_width
        scale_height = m1_height / m2_height
        return (scale_width + scale_height) / 2

    def to_m2(self, m1_point):

        if self.success:
            m1_point_homogeneous = np.append(m1_point, 1)  #
Convert to homogeneous coordinates
            return np.dot(self.transform_matrix,
m1_point_homogeneous) / self.scale_factor
        else:
            return None

    def to_m1(self, m2_point):

        if self.success:
            affine_transform_4x4 =
np.vstack([self.transform_matrix, [0, 0, 0, 1]])
            inverse_affine_transform =
np.linalg.inv(affine_transform_4x4)
            m2_point_homogeneous = np.append(m2_point *
self.scale_factor, 1)  # Convert to homogeneous coordinates
            m1_point_homogeneous =
np.dot(inverse_affine_transform, m2_point_homogeneous)

            # Convert back to non-homogeneous coordinates
            return (m1_point_homogeneous[:3] /
m1_point_homogeneous[3])
        else:
            return None


class GazeProcessor:

    def __init__(self, camera_idx=0, callback=None,
visualization_options=None):
        self.camera_idx = camera_idx
        self.callback = callback
```

```
        self.vis_options = visualization_options
        self.left_detector =
EyeballDetector(DEFAULT_LEFT_EYE_CENTER_MODEL)
        self.right_detector =
EyeballDetector(DEFAULT_RIGHT_EYE_CENTER_MODEL)
        self.options = mp.tasks.vision.FaceLandmarkerOptions(

base_options=mp.tasks.BaseOptions(model_asset_path=model_path),
            running_mode=mp.tasks.vision.RunningMode.VIDEO
        )

    async def start(self):
        with FaceLandmarker.create_from_options(self.options) as
landmarker:
            cap = cv2.VideoCapture(0)
            cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
            cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
            width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
            height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
            while cap.isOpened():
                success, frame = cap.read()
                frame = cv2.flip(frame, 1)
                if not success:
                    print("Ignoring empty camera frame.")
                    continue

                timestamp_ms = int(time.time() * 1000)
                mp_image =
mp.Image(image_format=mp.ImageFormat.SRGB, data=frame)

                face_landmarker_result =
landmarker.detect_for_video(mp_image, timestamp_ms)

                if face_landmarker_result.face_landmarks:
                    lms_s = np.array([[lm.x, lm.y, lm.z] for lm
in face_landmarker_result.face_landmarks[0]])
                    lms_2 = (lms_s[:, :2] * [frame.shape[1],
frame.shape[0]]).round().astype(int)

                    mp_hor_pts = [lms_s[i] for i in
OUTER_HEAD_POINTS]
                    mp_ver_pts = [lms_s[i] for i in
[NOSE_BRIDGE, NOSE_TIP]]
                    model_hor_pts = OUTER_HEAD_POINTS_MODEL
                    model_ver_pts = [NOSE_BRIDGE_MODEL,
NOSE_TIP_MODEL]

                    at =
AffineTransformer(lms_s[BASE_LANDMARKS,:], BASE_FACE_MODEL,
mp_hor_pts, mp_ver_pts, model_hor_pts, model_ver_pts)
                    indices_for_left_eye_center_detection =
LEFT_IRIS + ADJACENT_LEFT_EYELID_PART
                    left_eye_iris_points =
```

```
lms_s[indices_for_left_eye_center_detection, :]
                    left_eye_iris_points_in_model_space =
[at.to_m2(mpp) for mpp in left_eye_iris_points]

self.left_detector.update(left_eye_iris_points_in_model_space,
timestamp_ms)

                    indices_for_right_eye_center_detection =
RIGHT_IRIS + ADJACENT_RIGHT_EYELID_PART
                    right_eye_iris_points =
lms_s[indices_for_right_eye_center_detection, :]
                    right_eye_iris_points_in_model_space =
[at.to_m2(mpp) for mpp in right_eye_iris_points]

self.right_detector.update(right_eye_iris_points_in_model_space,
timestamp_ms)

                    crop_img1 = frame[right_eye_iris_points[1]-
36:right_eye_iris_points[1]+36, right_eye_iris_points[0]-
60:right_eye_iris_points[0]+60]
                    crop_img2 = frame[left_eye_iris_points[1]-
36:left_eye_iris_points[1]+36, left_eye_iris_points[0]-
60:left_eye_iris_points[0]+60]

                    left_gaze_vector, right_gaze_vector = None,
None

                    img = np.asarray(img)
                    img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    ex=np.asarray([crop_img1])
                    values = model.predict([ex])
                    right_gaze_vector=values

                    img = np.asarray(img)
                    img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                    ex=np.asarray([crop_img2])
                    values = model.predict([ex])
                    left_gaze_vector=values

                    if self.left_detector.center_detected: #and
self.right_detector.center_detected
                        left_eyeball_center =
at.to_m1(self.left_detector.eye_center)
                        left_pupil = lms_s[LEFT_PUPIL]
                        left_gaze_vector = left_pupil -
left_eyeball_center
                        left_proj_point = left_pupil +
left_gaze_vector*5.0

                    if self.right_detector.center_detected:
                        right_eyeball_center =
at.to_m1(self.right_detector.eye_center)
                        right_pupil = lms_s[RIGHT_PUPIL]
```

```
                        right_gaze_vector = right_pupil -
right_eyeball_center
                        right_proj_point = right_pupil +
right_gaze_vector*5.0

                    if self.callback and (left_gaze_vector is
not None or right_gaze_vector is not None):
                        await self.callback(left_gaze_vector,
right_gaze_vector)

                    if self.vis_options:
                        if self.left_detector.center_detected
and self.right_detector.center_detected:
                            p1 = relative(left_pupil[:2],
frame.shape)
                            p2 = relative(left_proj_point[:2],
frame.shape)
                            frame = cv2.line(frame, p1, p2,
(255, 0, 0), self.vis_options.line_thickness)
                            val=p2[1]-p1[1]
                            val=int(val/2)
                            val1=p2[0]-p1[0]

 (val1,val))
                            if val1<-30:
                                print("rec")
                                overlay = frame.copy()
                                cv2.rectangle(overlay, (0, 0),
(30, 720), (0, 0, 200), -1)
                                alpha = 0.4
                                frame = cv2.addWeighted(overlay,
alpha, frame, 1 - alpha, 0)
                            elif val1>30:
                                print("rec")
                                overlay = frame.copy()
                                cv2.rectangle(overlay, (1280,
720), (1250, 0), (0, 0, 200), -1)  # A filled rectangle
                                alpha = 0.4
                                frame = cv2.addWeighted(overlay,
alpha, frame, 1 - alpha, 0)
                            elif val<-10:
                                print("rec")
                                overlay = frame.copy()
                                cv2.rectangle(overlay, (0, 0),
(1280, 30), (0, 0, 200), -1)  # A filled rectangle
                                alpha = 0.4
                                frame = cv2.addWeighted(overlay,
alpha, frame, 1 - alpha, 0)
                            elif val>10:
                                print("rec")
                                overlay = frame.copy()
                                cv2.rectangle(overlay, (0, 690),
(1280, 720), (0, 0, 200), -1)  # A filled rectangle
```

```
                                alpha = 0.4
                                frame = cv2.addWeighted(overlay,
alpha, frame, 1 - alpha, 0)
                            else:
                                overlay = frame.copy()

cv2.circle(frame,(int(1280/2)+val1*15,int(720/2)+val*15+50),150,
(0,200,0), -1)
                                alpha = 0.8
                                frame = cv2.addWeighted(overlay,
alpha, frame, 1 - alpha, 0)
                            if val<0:

cv2.ellipse(frame,(94,94),(80,abs(val)),0,0,-180,( 255, 0, 0
),2)
                            else:

cv2.ellipse(frame,(94,94),(80,abs(val)),0,0,180,( 255, 0, 0 ),2)



                            if val1<0:

cv2.ellipse(frame,(94,94),(abs(val1),80),0,90,270,( 255, 0, 0
),2)
                            else:

cv2.ellipse(frame,(94,94),(abs(val1),80),0,270,450,( 255, 0, 0
),2)
                            p1 = relative(right_pupil[:2],
frame.shape)


                            p2 = relative(right_proj_point[:2],
frame.shape)
                            # crop_img2 = frame[p1[0]-
60:p1[0]+60, p1[1]-36:p1[1]+36]
                            # np.save("file"+str(number),
crop_img1)
                            # number = number +1
                            val=p2[1]-p1[1]


                            val2=p2[0]-p1[0]
                            val2=int(val/2)
                            np.save("file2-"+str(number2),
(val2,val))
                            number2 = number2 +1
                            frame = cv2.line(frame, p1, p2, (0,
255, 0), self.vis_options.line_thickness)

                            if val2<0:
```

```
cv2.ellipse(frame,(262,94),(80,abs(val2)),0,0,-180,(0, 255,
0),2)
                            else:

cv2.ellipse(frame,(262,94),(80,abs(val2)),0,0,180,(0, 255, 0),2)
                            val3=p2[0]-p1[0]
                            if val3<0:

cv2.ellipse(frame,(262,94),(abs(val3),80),0,90,270,(0, 255,
0),2)
                            else:

cv2.ellipse(frame,(262,94),(abs(val3),80),0,270,450,(0, 255,
0),2)


                        else:
                            text_location = (10, frame.shape[0]
- 10)
                            # cv2.putText(frame,
"Calibration...", text_location, cv2.FONT_HERSHEY_SIMPLEX, 0.5,
self.vis_options.color, 2)

                cv2.imshow('Amogus', frame)
                if cv2.waitKey(5) & 0xFF == 27:
                    break
```

Б.2 Файл main.py, що викликає програму

```
from GazeProcessor import GazeProcessor
from VisualizationOptions import VisualizationOptions
import asyncio

async def gaze_vectors_collected(left, right):
    print(f"left: {left}, right: {right}")

async def main():
    vo = VisualizationOptions()
    gp = GazeProcessor(visualization_options=vo,
callback=gaze_vectors_collected)
    await gp.start()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    try:
        loop.run_until_complete(main())
    finally:
        loop.close()
```