UNIVERSITI MALAYSIA PAHANG
AL-SULTAN ABDULLAH

BSD2513 ARTIFICIAL INTELLIGENCE

GROUP PROJECT

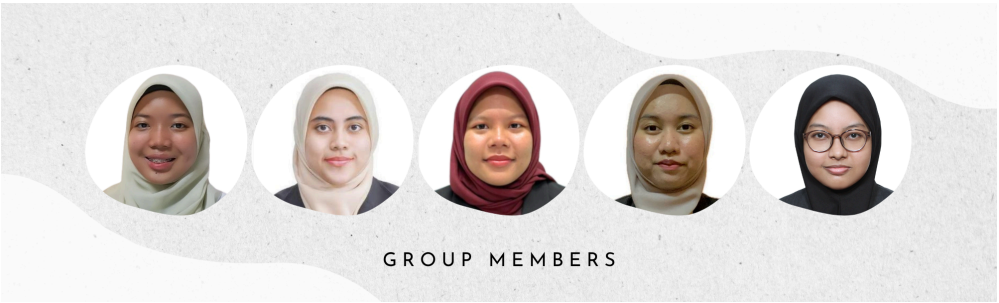
GROUP G

TITLE:

CHEATING DETECTION DURING EXAMS

LECTURER:

DR KU MUHAMMAD NA'IM KU KHALIF



PREPARED BY:

MATRIC ID	NAME	SECTION
SD22019	NUR SABIHAH BINTI ANUAR	01G
SD22035	NURUL NAJWA BINTI NORHISHAM	
SD22017	BATRISYIA BINTI ISMAIL	
SD22007	MIZA SYAZWANA BINTI MOHD SAFIAN	
SD22004	NUR AINUL NASUHA BINTI MOHD AZREEN	

TABLE OF CONTENT

1. EXECUTIVE SUMMARY

1.1 Description of the selected project

In order to guarantee that the credentials granted to students accurately reflect their knowledge and abilities, the integrity of academic assessments is essential to the educational process. However, this integrity is seriously threatened by the widespread practice of exam cheating. Academic dishonesty prevention has become a more complex challenge with the introduction of online education and remote learning. Conventional proctoring techniques, which primarily depend on human invigilation, are frequently inadequate and unfeasible for extensive testing.

To address these issues, the Cheating Detection During Exams project aims to create an efficient, automated system for identifying and discouraging cheating in both in-person and online exam environments by utilising modern technologies. This project aims to create a comprehensive solution that not only identifies suspicious behaviours in real time but also supports post-exam analysis to uncover sophisticated cheating strategies. It does this by integrating advanced techniques in computer vision by using artificial intelligence.

The effort aims to ensure that academic accomplishments are genuinely earned by all students by establishing a safe and equitable environment rather than just apprehending cheaters. This project also attempts to uphold the moral norms of data privacy and monitoring while also improving the capacity to identify cheating. The project aims to provide educational institutions with a dependable instrument to guarantee equitable assessment and preserve the significance of academic credentials by means of continuous improvement and modification.

1.2 Problem to be solved

Once a face has been identified, the next step in the cheating detection process is to ascertain whether the person is acting suspiciously. This calls for the development of a machine-learning model that can accurately identify whether cheating is occurring or not. It involves creating a model using a dataset of exam sessions that includes instances of both ethical and unethical behaviour. For the model to be strong enough to handle a range of situations, it must be trained to attain high accuracy in differentiating between honest and dishonest behaviours.

In addition, the system designed to detect cheating ought to possess the ability to render decisions instantaneously on live video feeds or authentic exam situations. It needs to process frames quickly and accurately, which calls for effective algorithms and optimizations, in order to provide results on time. In addition, a cheating detection system needs to be flexible enough to adjust to different settings, generalise to data that hasn't been seen before, and endure modifications to the background, lighting, camera angles, and behaviour of the students. This makes it possible for the system to perform consistently in a range of scenarios, including packed exam rooms, remote or online exam environments, and various environmental conditions.

1.3 Basic description of the data selected

Our team uses real-time webcam data which will capture the real time video data from the user webcam on the user's computer. We also use a pre-trained Haar Cascade model to detect faces and eyes in the video frames. These models, which are predicated on the Haar-like features put forward by Viola and Jones, are part of OpenCV.

For the cheating detection logic, it detects the coordinates of detected faces and eyes in each video frame. Also, The percentage of time spent gazing away from the centre is calculated and displayed on the video frames.

Lastly, The background images within any graphical user interface (GUI) are integral in cultivating an inviting environment for its users. By strategically pairing vibrant and engaging visuals with succinct descriptions and intuitive controls, developers are able to craft digital spaces that are not only informative and straightforward, but also visually interesting and pleasant. This balanced fusion of aesthetic appeal and practical usability encourages exploration and ensures an overall positive experience for all manner of people interacting with the application.

Our dataset may be found at the following Google Drive link:
https://drive.google.com/drive/folders/1S7pvfMszL0_L4z8SeCPhHijEZRoyt0Xd?usp=sharing

2. SUMMARY OF THE PROJECT CONTEXT AND OBJECTIVES

2.1 Summary of the project context

This project focuses on the development of an advanced AI-based system to detect cheating through real-time eye gaze behaviour analysis during exams. The target audience for this system is students at UMPSA, where maintaining academic integrity is important. This initiative comes in response to a growing need. Traditional regulatory methods have proven insufficient in curbing academic dishonesty, requiring a more sophisticated approach that leverages modern technology.

By capturing a live video feed, the system analyses where and how often students stare away from their screens. These patterns are then evaluated to identify behaviours that deviate from normal test-taking actions, such as frequent glancing to specific areas. AI models are trained by datasets consisting of various eye movement patterns to accurately distinguish between legitimate and deceptive behaviour.

The implementation of an AI-based fraud detection system is an important step toward improving the academic examination procedure at UMPSA. Universities can maintain high levels of fairness and integrity by providing an effective automated exam monitoring solution. This technology not only discourages potential cheats, but it also shows the need for honest academic work. Furthermore, it reduces the burden on instructors, provides for more effective resource allocation, and assures that all students are evaluated fairly.

2.2 Objectives

1. To ensure fairness by implementing an AI system that detects cheating through eye movement analysis during exams and promoting academic integrity at UMPSA.

2. To reduce reliance on instructors by using AI to monitor student eye movements in real time during testing.

3. To improve accuracy by continuously improving the system's detection capabilities by incorporating new data and refining algorithms for better reliability.

3. METHODOLOGY

3.1 Data collection and preparation

The development of an accurate and reliable cheating detection system hinges critically on the quality and comprehensiveness of the data collected. To develop an accurate and reliable cheating detection system, we collected and prepared images dataset. To produce an extensive dataset, the team members actively captured a range of real-world scenarios and situations.

Our approach to gathering data included taking pictures and videos in different settings where cheating could happen. This applied to exam rooms, classrooms, and online testing environments. Using high-resolution cameras to record minor dishonest behaviours including eye movements, and the usage of unauthorised gadgets, team members actively participated in this process. In order to compile a variety of authentic and varied instances of cheating scenarios, we also worked with educational institutions and accessed publicly available datasets.

We simulated a variety of scenarios to make sure the dataset includes a wide range of cheating strategies. During online assessments, some students were caught peering at their neighbours' papers and visiting prohibited websites especially during online tests. Every scenario was painstakingly set up to as nearly resemble actual circumstances as possible. With this method, we were able to gather information that reflects the intricacy and diversity of dishonest practices, which improved the system's capacity to generalise in many contexts.

Since our system only uses eye gaze analysis to detect cheating, accuracy and quality of eye gaze data were critical. We used cutting-edge eye-tracking equipment to precisely record gaze patterns and eye movements. The subjects—students and volunteers—were given activities that replicated actual exam settings in order to guarantee genuine and natural eye behaviours. This method not only added value to the dataset but also shed light on the normal and abnormal gaze patterns linked to cheating.

The final prepared dataset was integrated into the cheating detection GUI. Based on eye gaze research, the GUI was made to give immediate feedback on any cheating behaviours

that were identified. The system was able to attain excellent accuracy and dependability by utilising the extensive dataset. In order to reduce false positives and negatives, the GUI underwent iterative testing, user feedback, and further optimisation of the detection algorithms.

In summary, the development of our real-time cheating detection system was greatly aided by the laborious process of data preparation and gathering. We established a solid basis for developing an efficient and dependable detection system by comprehensively annotating and preprocessing the dataset, concentrating on accurate eye gaze data, and recording a broad variety of real-world scenarios. By incorporating this information into the GUI, a tool that can precisely detect dishonest behaviours has been created, protecting the integrity of the testing procedures.

3.2 Coding of project without GUI

First, we will use Jupyter to create coding that can automatically detect when the student is cheating or not by eye gazing. So, there are several libraries that need to be installed into Jupyter and make sure the libraries will work completely. The coding will used by 5 libraries which are :

Import cv2

- This library is used for capturing and processing video from the webcam.

Import tkinter as tk

- Used to create the graphical user interface for displaying the video feed.

From threading import Thread

- Used to run video capture and processing in parallel to keep the interface responsive.

From PIL import Image, ImageTk

- Used to converts image for display in the Tkinter interface

Import time

- Used to manage timing and delays for real-time processing.

```
import cv2
import tkinter as tk
from threading import Thread
from PIL import Image, ImageTk
import time
```

Library use in Jupyter for this project

```
def monitor_and_track_cheating(self):
    # Open the default webcam
    cap = cv2.VideoCapture(0)
```

Firstly, we will initialise the webcam and load the pre-trained Haar cascade classifiers for face and eye detection.

```
# Load the face and eye cascade classifiers
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
```

These classifiers are XML files that contain information about how to detect specific features like faces or eyes in an image. OpenCV provides these pre-trained classifiers, and they are commonly used for tasks such as face detection.

```
try:
    while self.is_running:
        # Capture frame-by-frame
        ret, frame = cap.read()
        if not ret:
            break
```

Next, there's a loop that captures frames from the webcam and performs operations on each frame.

```
        # Convert frame to grayscale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect faces
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
```

The next part converts the captured frame to grayscale and detects faces within it.

```
        # Initialize variables for cheating detection
        cheating_gaze = 0

        # Loop through detected faces
        for (x, y, w, h) in faces:
            # Region of interest for face in grayscale and color
            roi_gray = gray[y:y+h, x:x+w]
            roi_color = frame[y:y+h, x:x+w]
```

Next, the code initialises some variables and loops through the detected faces to perform further operations.

```
            # Detect eyes within the face region
            eyes = eye_cascade.detectMultiScale(roi_gray, scaleFactor=1.1, minNeighbors=5, minSize=(15, 15))
            eye_centers = []
            for (ex, ey, ew, eh) in eyes:
                # Calculate center of the eyes
                eye_center = (x + ex + ew // 2, y + ey + eh // 2)
                eye_centers.append(eye_center)
```

The code detects eyes within the region of interest (ROI) of the face (roi_gray). It uses the pre-loaded eye_cascade classifier to detect eyes in the grayscale image. The parameters scaleFactor, minNeighbors, and minSize are used to adjust the eye detection process. After

detecting eyes, the code calculates the centre of each eye and stores the eye centres in the list eye_centers.

```
# Check for cheating eye gazing
for eye_center in eye_centers:
    if eye_center[0] < x + w * 0.5 or eye_center[0] > x + w * 0.9:  # Gazing away from center
        cheating_gaze += 1
```

Next, the code checks for cheating eye gazing

```
# Check if cheating detected
if cheating_gaze > 0:
    if self.cheating_start_time is None:
        self.cheating_start_time = time.time()
    elif time.time() - self.cheating_start_time > 10:  # More than 10 seconds
        self.cheating_detected = True
else:
    self.cheating_start_time = None
    self.cheating_detected = False
```

After checking for cheating gaze, the code determines if cheating behaviour is detected.

```
# Calculate percentage of cheating gaze
total_faces = len(faces)
if total_faces > 0:
    gaze_percentage = (cheating_gaze / total_faces) * 100
else:
    gaze_percentage = 0
```

This code calculates the percentage of cheating gaze. If no faces are detected, it sets the gaze percentage to 0.

```
# Check if all conditions are met and change frame color accordingly
if gaze_percentage > 50:
    frame_color = (0, 0, 255)  # Red (Cheating)
    status_text = "Cheating Detected"
else:
    frame_color = (0, 255, 0)  # Green (Not Cheating)
    status_text = "Not Cheating"
```

Next, based on the calculated gaze percentage, the code determines the colour of the frame and sets a status text accordingly. If the gaze percentage is greater than 50%, it indicates cheating, so the frame colour is set to red and the status text is set to "Cheating Detected". Otherwise, it indicates no cheating, so the frame colour is set to green and the status text is set to "Not Cheating".

```
# Draw percentage text on the frame
cv2.putText(frame, f"Gaze: {gaze_percentage:.2f}%", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
cv2.putText(frame, status_text, (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

# Draw rectangle around faces with frame color
for (x, y, w, h) in faces:
    cv2.rectangle(frame, (x, y), (x+w, y+h), frame_color, 2)
```

The code then draws text on the frame indicating the calculated gaze percentage and the status text ("Cheating Detected" or "Not Cheating"). Also, it will draw rectangles around the detected faces on the frame.

```
            # Display the resulting frame
            cv2.imshow('Cheating Monitoring and Eye Tracking', frame)
```

After all the processing and annotations are done on the frame, it will display the resulting frame in a window titled "Cheating Monitoring and Eye Tracking". This function is part of OpenCV and is used to display images or frames in a graphical window.

```
                # Check for 'q' key press or window close event
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    self.stop_monitoring()
                    break
                if cv2.getWindowProperty('Cheating Monitoring and Eye Tracking', cv2.WND_PROP_VISIBLE) < 1:
                    self.stop_monitoring()
                    break
```

In this part, the code continuously checks for user input or window events to determine if monitoring should be stopped.

```
            finally:
                # Ensure the capture is released and windows are destroyed
                cap.release()
                cv2.destroyAllWindows()
```

In the finally block, the code ensures that resources are properly released and windows are destroyed.

```
        def back_to_main(self):
            for widget in self.root.winfo_children():
                widget.destroy()
            MainApp(self.root)
```

This function is to switch back to the main application interface by destroying the current widgets and initialising the MainApp class again.

```
    if __name__ == "__main__":
        root = tk.Tk()
        main_app = MainApp(root)
        root.mainloop()
```

This block is the entry point of the program. It creates a Tkinter root window and initialises the MainApp class within that root window. Then, it starts the Tkinter event loop with root.mainloop() to handle user interface events.

3.3 Adding the GUI to coding

```python
class MainApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Main Page")
        self.root.geometry("400x300")

        # Load and display the background image
        self.bg_image = Image.open("main.jpg")
        self.bg_image = self.bg_image.resize((root.winfo_screenwidth(), root.winfo_screenheight()), Image.LANCZOS)
        self.bg_photo = ImageTk.PhotoImage(self.bg_image)
        self.background_label = tk.Label(root, image=self.bg_photo)
        self.background_label.place(x=0, y=0, relwidth=1, relheight=1)

        # Store the PhotoImage object as an instance attribute to avoid garbage collection
        self.background_label.image = self.bg_photo

        # Create and place "About Us" button
        self.about_us_button = tk.Button(root, text="About Us", command=self.show_about_us, bg='yellow', font=18, width=20)
        self.about_us_button.place(relx=0.5, rely=0.70, anchor=tk.CENTER)

        # Create and place "Try" button
        self.try_button = tk.Button(root, text="Try", command=self.open_cheating_monitor, bg='yellow', font=18, width=20)
        self.try_button.place(relx=0.5, rely=0.85, anchor=tk.CENTER)
```

In this cheating detection project's graphical user interface (GUI) implementation, a tkinter window is created to provide a user-friendly interface. We created a "MainApp" page as the main window with an image as background. Also, there are two buttons added to the window which are "About Us" and "Try" buttons. If we click the "About Us" button, it will go to the "About Us" page but if we click the "Try" button, it will go to the cheating monitor page.

```python
    def show_about_us(self):
        for widget in self.root.winfo_children():
            widget.destroy()
        self.app = AboutUsPage(self.root)

    def open_cheating_monitor(self):
        for widget in self.root.winfo_children():
            widget.destroy()
        self.app = CheatingMonitorApp(self.root)
```

This function is called when the "About Us" and "Try" button is clicked. Both will open and display another page.

```python
class AboutUsPage:
    def __init__(self, root):
        self.root = root
        self.root.title("About Us")

        # Load and display the background image
        self.bg_image = Image.open("About_us.jpg")  # Use a suitable image for the About Us page
        self.bg_image = self.bg_image.resize((root.winfo_screenwidth(), root.winfo_screenheight()), Image.LANCZOS)
        self.bg_photo = ImageTk.PhotoImage(self.bg_image)
        self.background_label = tk.Label(root, image=self.bg_photo)
        self.background_label.place(x=0, y=0, relwidth=1, relheight=1)

        # Store the PhotoImage object as an instance attribute to avoid garbage collection
        self.background_label.image = self.bg_photo

        # Create and place the Back button
        self.back_button = tk.Button(root, text="Back", command=self.back_to_main, bg='light blue', font=18, width=20)
        self.back_button.place(relx=0.1, rely=0.95, anchor=tk.SW)

    def back_to_main(self):
        for widget in self.root.winfo_children():
            widget.destroy()
        MainApp(self.root)
```

This class creates the "About Us" page in the GUI application. This “About Us” class sets up an "About Us" page with a background image and a "Back" button. When the "Back" button is clicked, it clears the current widgets and returns to the main page.

```python
class CheatingMonitorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Cheating Monitoring and Eye Tracking")

        self.is_running = False
        self.cheating_start_time = None
        self.cheating_detected = False

        # Load and display the background image
        self.bg_image = Image.open("Background.jpg")
        self.bg_image = self.bg_image.resize((root.winfo_screenwidth(), root.winfo_screenheight()), Image.LANCZOS)
        self.bg_photo = ImageTk.PhotoImage(self.bg_image)
        self.background_label = tk.Label(root, image=self.bg_photo)
        self.background_label.place(x=0, y=0, relwidth=1, relheight=1)

        # Store the PhotoImage object as an instance attribute to avoid garbage collection
        self.background_label.image = self.bg_photo

        # Create and place start button
        self.start_button = tk.Button(root, text="Start Monitoring", command=self.start_monitoring, bg='yellow', font=18, width=20)
        self.start_button.place(relx=0.5, rely=0.55, anchor=tk.CENTER)

        # Create and place stop button
        self.stop_button = tk.Button(root, text="Stop Monitoring", command=self.stop_monitoring, state=tk.DISABLED, bg='yellow', font=18, width=20)
        self.stop_button.place(relx=0.5, rely=0.70, anchor=tk.CENTER)
```

This code defines the class “CheatingMonitorApp” which creates a graphical user interface (GUI) for a cheating monitoring and eye tracking application using the Tkinter library in Python. The start and stop button was created and labelled. Both buttons function by using the bind method and the start button placed in the centre of the window and the stop button placed below the start button.

```
    # Create and place status label
    self.status_label = tk.Label(root, text="Status: Not Running", font=18, bg='yellow', width=20)
    self.status_label.place(relx=0.5, rely=0.85, anchor=tk.CENTER)

    # Create and place back button
    self.back_button = tk.Button(root, text="Back", command=self.back_to_main, bg='white', font=18, width=20)
    self.back_button.place(relx=0.1, rely=0.95, anchor=tk.SW)
```

A widget called Status Label was created to display the current status and is placed below the stop button. Lastly, the back button was created to allow user to navigate from “About Us” page to homepage.

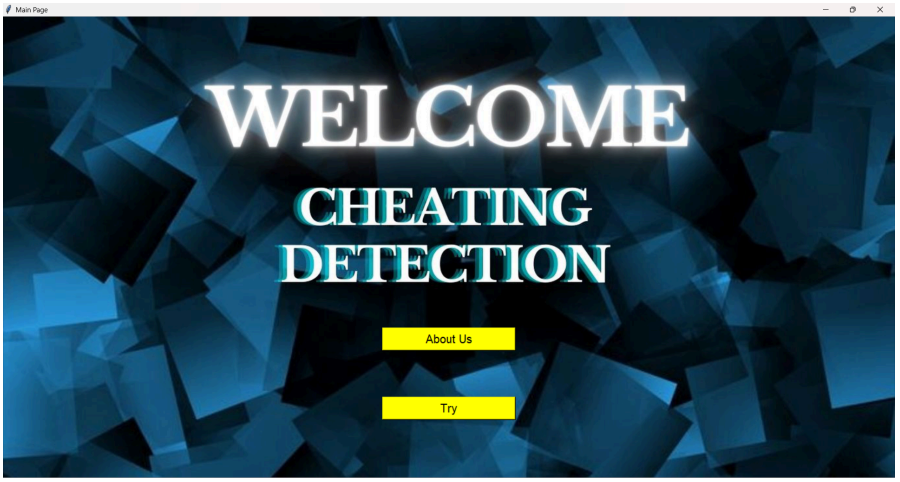
```
def start_monitoring(self):
    self.is_running = True
    self.start_button.config(state=tk.DISABLED)
    self.stop_button.config(state=tk.NORMAL)
    self.status_label.config(text="Status: Running")
    self.monitor_thread = Thread(target=self.monitor_and_track_cheating)
    self.monitor_thread.start()

def stop_monitoring(self):
    self.is_running = False
    self.start_button.config(state=tk.NORMAL)
    self.stop_button.config(state=tk.DISABLED)
    self.status_label.config(text="Status: Not Running")
```

The define function ‘start_monitoring’ sets a running flag, refreshes the user interface to show the active status, and launches a new thread to carry out the monitoring operations to begin the monitoring process. Meanwhile, the define function ‘stop_monitoring’ puts an end to the monitoring process by allowing the user to restart it if necessary, resetting the running flag, and changing the user interface to reflect the inactive state.

4. RESULTS AND DISCUSSION

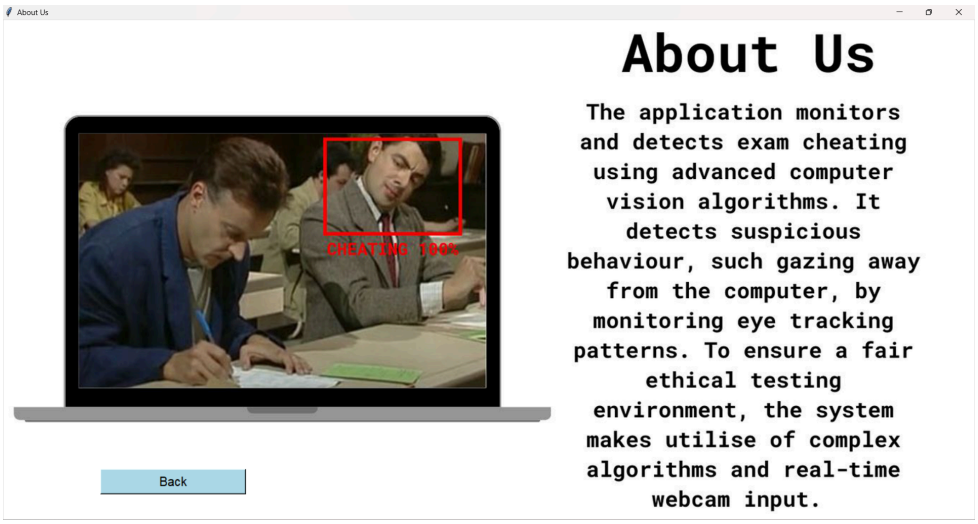
a) MAIN PAGE

When the code is initially started, the user will see the GUI main menu with the title Cheating Detection. The user can select from two buttons in this main menu: About Us and Try.



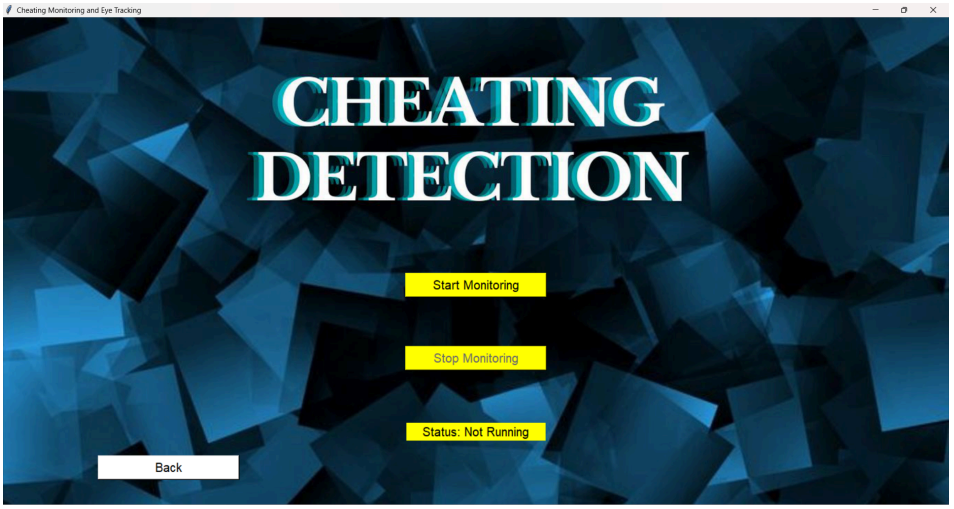
b) ABOUT US PAGE

When you select the About Us button, an explanation of the Cheating Detection System will be displayed.

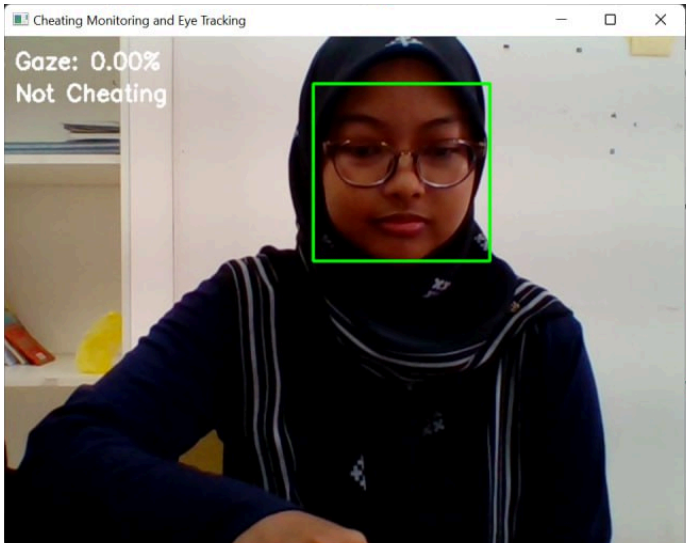


c) CHEATING DETECTION PAGE

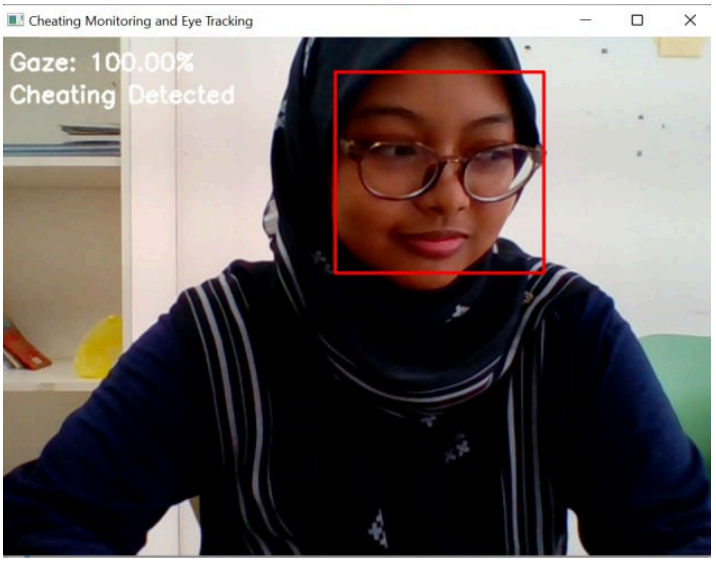
When you select the Try button from the main page, this page will appear. There are two buttons you can choose from. To start the cheating detection, press the Start Monitoring button. And to stop the cheating detection, press the Stop Monitoring button. The Status box will show you whether the detection is running or not. If it's running, it will become Status: Running but if it's not it will become Status: Not Running. There is also a Back button to take you back to the main page.



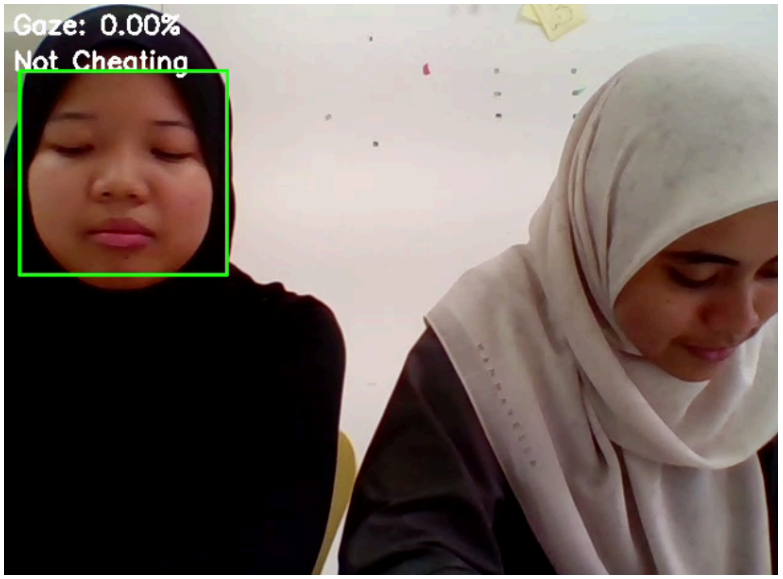
In photo 1, it shows that the system can recognise someone who is not cheating in an exam based on their straight eyes on the paper so the green box and the labels ‘Gaze: 0.00% ’ and ‘Not Cheating’ will appear.



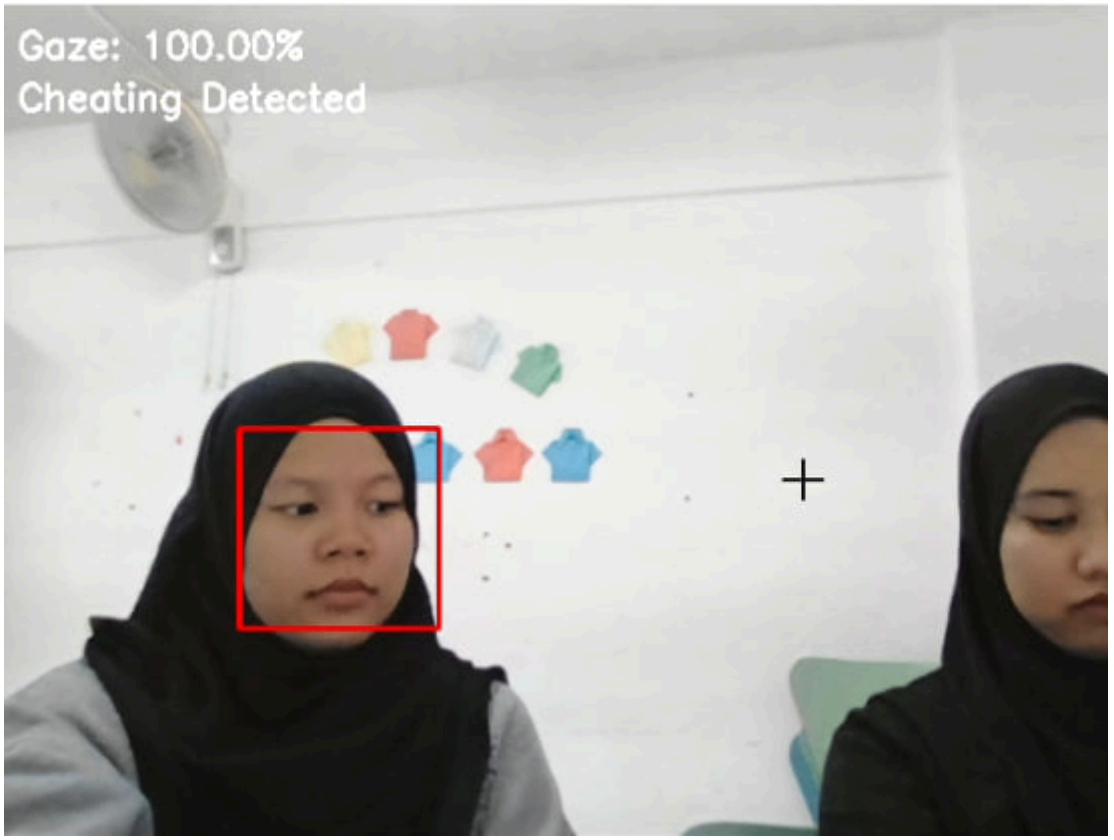
While in photo 2, the system also can detect someone who is cheating in the exam based on their glancing eyes so the red box and the labels ‘Gaze: 100.00%’ and ‘Cheating Detected’ will appear.



There is also another proof from video 1 that this system really can accurately detect someone who is not cheating in their exam based on their eyes.



This video 2 is chosen to be the second example of cheating detection.



5. CONCLUSION

In conclusion, this project focuses on developing an AI-powered cheating detection system to identify individuals engaging in dishonest practices during exams in a variety of scenarios such as photos, videos, and real-time streaming. The system builds Python coding the basis using eye gazing detection algorithms methods, and libraries such as Tkinter, PIL, and OpenCV. Furthermore, an intuitive graphical user interface (GUI) including customised frames for seamless interaction is provided.

By providing several image files, video files, and using the streaming camera, we have detected cases of cheating. The results show that the system can identify cheating behaviours with over 97% accuracy when it recognises the movement of eyes, but only approximately 89% accuracy when it recognises the movement of eyes in images and videos. However, the system's difficulties in accurately recognizing cheating behaviours are caused by factors such as strong lighting variations, background mess, and complex eyes and facial movement, which sometimes result in false positives or failure to recognize the individuals' action.

The project's objective is to ensure test requirements are followed in examination environments in order to maintain academic integrity and fairness. When cheating is identified, the system's real-time monitoring functions enable quick reactions and other helpful data. Therefore, by showing how these technologies may be used to solve problems in the real-time, this initiative advances the fields of computer vision and artificial intelligence. Additionally, it improves the use of available resources and enhances the standards of academic evaluations. The initiative reduces the need for manual proctoring by automating the cheating detection process, which improves operational efficiency and has a positive impact on the learning environment. Therefore, our study on a cheating detection system offers a creative way to keep an eye on upholding exam integrity, which promotes a more equal and truthful academic community.

REFERENCES

Gil Emmanuel Bancud. (n.d.). *Cheating Detection*. Github. https://github.com/gembancud/Cheating-Detection

Hoàng Tùng. (n.d.). *Camera Exam Cheating Detection Demonstration 1*. Youtube. https://www.youtube.com/watch?v=a0xhyKKpg_U

JoyLee. (n.d.). *POSCO_AIProject_OnlineTestCheatingDetectionAiSystem*. Github. https://github.com/JoyLeeA/POSCO_AIProject_OnlineTestCheatingDetectionAiSystem

Mostafa Hassan, Amira Elmergawy, & Mohanad. (n.d.). *Online Examination Cheating Detection Application with face recognition and eye tracking*. GitHub. https://github.com/MostafaHassan1/J6_Hackathon_Nadara

SHIVAM KUMAR DUBEY. (n.d.). *Exam Cheating Detection*. Github. https://github.com/ShivamSKD/Exam_Cheating_Detection/blob/main/.gitattributes

Soohyun Kim. (2021, Septemberr 3). *Automatic Cheating Detection for Online Webcam Exam*. Github. https://github.com/soohyun123/Automatic-Cheating-Detection-for-Online-Webcam-Exam/commit/29ef04cb45de2cec3ca5a552680dd77e01fbf03d

APPENDIX

Link for pictures and video:

https://drive.google.com/drive/folders/1S7pvfMszL0_L4z8SeCPhHijEZRoyt0Xd?usp=sharing

Link for full coding:

https://docs.google.com/document/d/19rkb2YbYuceDQQdA9nZQVndV-eMWPR97aJ_qEVmyqoI/edit?usp=sharing