IRIS and PUPIL Ratio Determination in Real time

Internship report on

"IRIS PUPIL RATIO DETECTION in REAL TIME"

Submitted by:

Name	USN
NORIN ROBY ISSAC	1RF23EC051
RAJESH G.D.	1RF23EC055
VARUN ACHAR	1RF24EC404

Internship Details

Start Date	06/09/2024
End Date	04/01/2025
Total Period	120 Days

Department of Electronics and Communication Engineering

With invaluable guidance from,

Dr. Prashant P Patavardhan
Professor of ECE,
RVITM, Bangalore
and
Dr. Kavitha N
Professor of ECE,
RVITM, Bangalore

ACKNOWLEDGEMENT

We express our profound gratitude to **Dr. Nagashettappa Biradar, Principal, RVITM, Bangalore,** for providing the necessary facilities and an ambient environment to work.

We are grateful to **Dr. Manjunatha Prasad. R, Head of Department, Electronics and Communication Engineering, RVITM, Bangalore,** for his valuable suggestions and advice throughout our work period.

We are grateful to **Dr. Prashant P Patavardhan** and **Dr. Kavitha N.** for their invaluable suggestions and advice throughout our work period.

We would like to thank all the staff members of the **Department of Electronics and Communication Engineering** for their support and encouragement during this internship.

Signature
Dr. Prashant P Patavardhan
Professor of ECE,
RVITM
Bangalore

Signature
Dr. Kavitha N.
Professor of ECE,
RVITM
Bangalore

Name	USN	Signature	Semester
Rajesh G.D.	1RF23EC055		III
Norin Roby Issac	1RF23EC051		III
Varun Achar	1RF24EC404		III

PROJECT GOALS

The project focuses on the **detection and analysis of the iris and pupil** from an image of an individual's eye, which could be a standard photograph or a medical imaging scan. This task is essential for extracting meaningful biometric and health-related data, offering insights into potential medical conditions or overall health status. Here's a detailed breakdown of the goals:

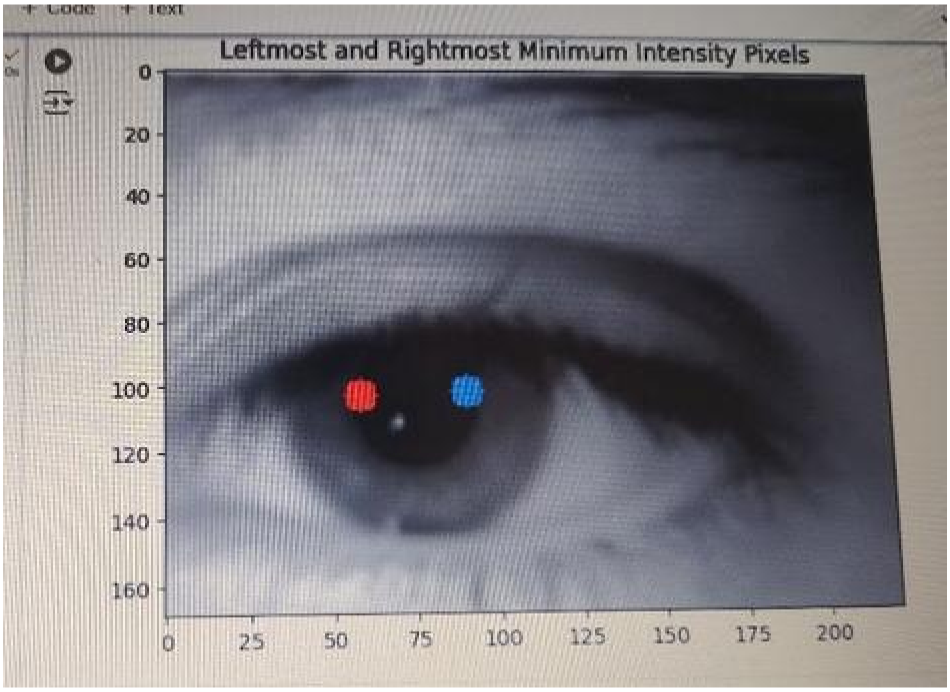
1. **Detection of Iris and Pupil**:
    - The primary task is to **accurately locate and segment the iris and pupil** within the eye image.
    - This involves distinguishing these regions from the surrounding sclera and other ocular structures.
2. **Radius Calculation**:
    - After segmentation, the **radii of the iris and pupil** are measured.
    - These measurements are crucial for determining the **ratio of the radii**, a metric that has implications for diagnostic assessments.
3. **Health Correlation via Ratios**:
    - Variations in the **iris-to-pupil ratio** may indicate medical conditions such as glaucoma, anisocoria, or other ocular diseases.
    - Ratios may also reflect physiological responses to stimuli, aiding in assessing neurological health.
4. **Statistical Analysis**:
    - Perform **additional statistical computations** on the extracted features, such as:
        - Shape and size variation of the pupil and iris.
        - Deviation from normal ranges based on medical benchmarks.
        - Temporal changes (if multiple images are provided).
5. **Health Diagnosis Potential**:
    - Extracted data can support **diagnoses of diseases** like diabetes (through iris patterns) or neurological disorders (pupil dynamics).

    - Patterns in the sclera, iris, and pupil may also indicate blood pressure abnormalities or systemic illnesses.
6. **Medical Applications**:
    - Insights from this project could be integrated into **ophthalmological diagnostic systems**.
    - Data may also contribute to research on **biometrics** or **health monitoring systems** for early disease detection.
7. **Automation and Accuracy**:
    - The focus is on creating an **automated and robust solution** using KMeans++ for segmentation, mediapipe for detection ensuring high accuracy in identifying boundaries.
8. **Visual and Analytical Outputs**:
    - Visual representation of segmented regions.
    - Comprehensive analysis reports summarizing radius ratios, statistical findings, and health interpretations.
9. **Broader Impact**:

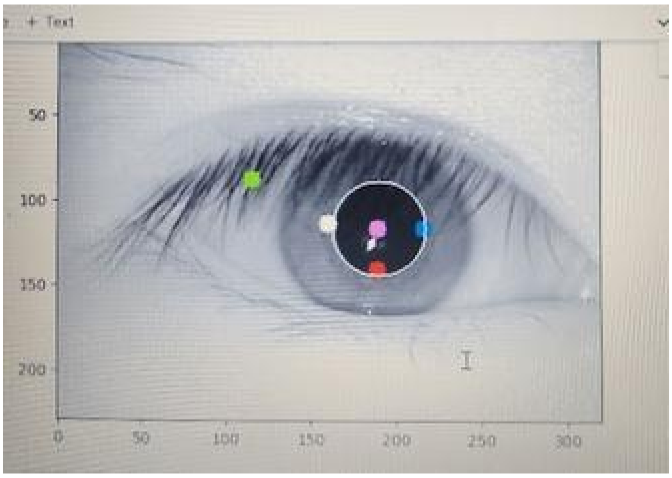
- The project bridges **machine learning, image processing, and medical diagnostics**, demonstrating how technology can support healthcare advancements and improve individual well-being.

PROJECT DEVELOPMENT TIMELINE

In order to identify methods and algorithms that could be used to achieve the stated project objectives, heuristic methods were experimented with, making use of the MMU Iris dataset for testing. The initial attempts involved attempting to place markers on the edge of the pupil by locating the leftmost and rightmost points with the lowest intensity in the image, since analysis of the CSV datasheets for a few randomly selected images indicated a marked drop in intensity around the pupil.



However batch processing showed that this method was prone to errors arising as a result of rogue dark spots or overlapping eyelashes yielding false results. An improvement on this two point method, the four point dark spot method, was therefore tried.



While this method yielded promising results, being more adept at compensating for false positives arising from eyebrow or rogue spots, batch processing showed its accuracy to be unsatisfactory. Therefore, the first attempts at more advanced methods involving basic data science or machine learning techniques were made. Initial attempts used the Hough transform and other algorithms to try and group pixels on the basis of similarity, or find circles within the image, which would correspond to the iris and pupil.

In the above image, the horizontal gradient of the image is taken along and low intensity pixels are enhanced with the dilate and erode functions, which then allows us to run the Hough transform to look for prominent circles. Above, the transform with some parameter adjustments successfully detects the iris.

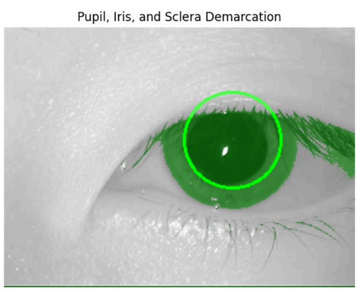
```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Load the image in grayscale
image = cv2.imread("eye.jpg", cv2.IMREAD_GRAYSCALE)
image2 = image.copy()

# Get image dimensions
height, width = image.shape

# Compute horizontal gradient
for i in range(height):
      for j in range(width - 1):
      image2[i][j] = abs(int(image[i][j + 1]) - int(image[i][j]))

# Erosion
kernel = np.ones((3, 3), np.uint8)
eroded_image = cv2.erode(image2, kernel, iterations=1)

# Dilation
kernel = np.ones((3,3), np.uint8)
dilated_image = cv2.dilate(eroded_image, kernel, iterations=1)

# Divide the image into 3x3 segments
segment_size = 3
threshold = 1.0000000000000001  # Set your threshold for low pixel density
output_image = dilated_image.copy()

for i in range(0, height, segment_size):
      for j in range(0, width, segment_size):
      # Define the segment
      segment = dilated_image[i:i + segment_size, j:j + segment_size]
      diff_arr=
      # Calculate the average intensity of the segment
      avg_intensity = np.mean(segment)

      # Make the segment black if the average intensity is below the threshold
      if avg_intensity < threshold:
            output_image[i:i + segment_size, j:j + segment_size] = 0

# Display the results
plt.figure(figsize=(12, 6))
plt.subplot(1, 4, 1)
```

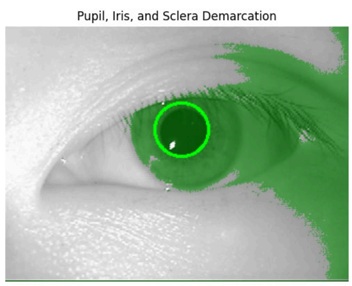
```
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 2)
plt.title('Horizontal Gradient')
plt.imshow(image2, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 3)
plt.title('Dilated Image')
plt.imshow(dilated_image, cmap='gray')
plt.axis('off')

plt.subplot(1, 4, 4)
plt.title('Segmented Output')
plt.imshow(output_image, cmap='gray')
plt.axis('off')

plt.show()
```

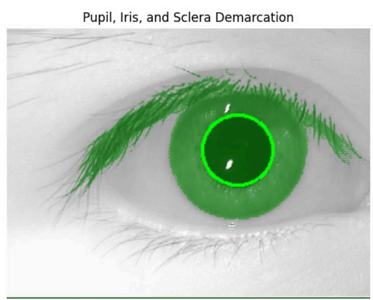
KMeans++ for Iris, Pupil, and Sclera Segmentation

1. **Objective**: Segment the eye image into three key regions: iris, pupil, and sclera using KMeans++.
2. **Preprocessing**: Convert the eye image to grayscale for better feature representation.
3. **Why KMeans++**: It initializes centroids more effectively, improving clustering accuracy and speed.
4. **Pixel Features**: Use intensity, as features for clustering.
5. **Cluster Count**: Set the number of clusters to 3 (pupil, iris, and sclera).
6. **Centroid Initialization**: KMeans++ selects centroids to maximize inter-cluster distance.
7. **Clustering**: Assign each pixel to the closest centroid based on feature similarity.
8. **Segmentation Map**: Generate a labeled map identifying pixels belonging to the pupil, iris, or sclera.
9. **Post-Processing**: Refine segmentation using morphological operations (e.g., dilation, erosion).

<u>Results</u>







How does Mediapipe Work for Iris Detection?

Mediapipe is a framework by Google for building multimodal, cross-platform machine learning pipelines.

For iris detection, Mediapipe's Face Mesh uses 468 3D landmarks to map facial features, including eyes.

Key Features of Mediapipe for Iris Detection:

Accurate real-time face and eye landmark detection.

***1.)** Refinement of eye landmarks with an additional 5-point iris Model.*

***2.)** Works on multiple platforms, including web, mobile, and Desktop.*

***3.)** Mediapipe is known for its efficiency and high performance with minimal computational resources.*

CODE

```
# Make sure to remove iris_pupil_data.csv if doing detection for a fresh
candidate.
import cv2
import mediapipe as mp
import numpy as np
import csv
import os
import time

# Remove the CSV file if it already exists
csv_file = 'iris_pupil_data.csv'
if os.path.exists(csv_file):
      os.remove(csv_file)

# Initialize Mediapipe Face Mesh
mp_face_mesh = mp.solutions.face_mesh

# Define the indices for the iris, pupil, and distance measurement (eyes)
LEFT_IRIS = [474, 475, 476, 477]
```

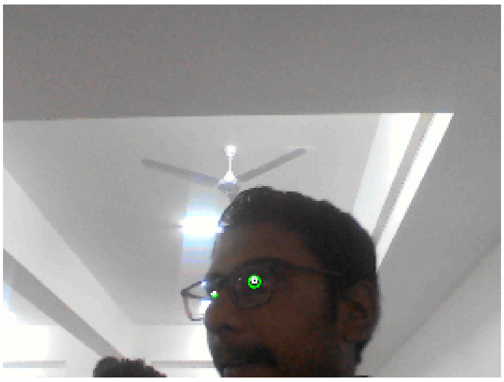
```
RIGHT_IRIS = [469, 470, 471, 472]
LEFT_PUPIL = [468, 469, 470, 471]
RIGHT_PUPIL = [473, 474, 475, 476]
LEFT_EYE_OUTER_CORNER = 33  # Landmark for outer corner of the left eye
RIGHT_EYE_OUTER_CORNER = 263  # Landmark for outer corner of the right eye

# Set up the webcam
cap = cv2.VideoCapture(0)

# Set the distance threshold range (in cm) and the corresponding pixel distance
range
distance_threshold_min = 59  # Minimum distance in cm
distance_threshold_max = 60  # Maximum distance in cm

# You can adjust this scale based on your camera calibration, assuming that
59-60 cm corresponds to some distance in pixels
pixel_distance_min = 100  # Minimum pixel distance between the eyes for 59 cm
pixel_distance_max = 110  # Maximum pixel distance between the eyes for 60 cm

# Open the CSV file in write mode
with open(csv_file, mode='w', newline='') as file:
       writer = csv.writer(file)
       # Write headers
       writer.writerow(['Frame',
                       'Left Iris Center X', 'Left Iris Center Y', 'Left Iris
Radius',
                       'Right Iris Center X', 'Right Iris Center Y', 'Right Iris
Radius',
                       'Left Pupil Center X', 'Left Pupil Center Y', 'Left Pupil
Radius',
                       'Right Pupil Center X', 'Right Pupil Center Y', 'Right Pupil
Radius'])

       # Mediapipe FaceMesh initialization
       with mp_face_mesh.FaceMesh(
       max_num_faces=1,
       refine_landmarks=True,
       min_detection_confidence=0.5,
       min_tracking_confidence=0.5) as face_mesh:

       frame_number = 0
       max_frames = 500  # Capture only 500 frames
       process_start_frame = 50  # Start processing after first 50 frames
       process_end_frame = max_frames - 50  # Stop processing before last 50
frames

       while cap.isOpened() and frame_number < max_frames:
              success, image = cap.read()
              if not success:
              print("Ignoring empty camera frame.")
              continue
```

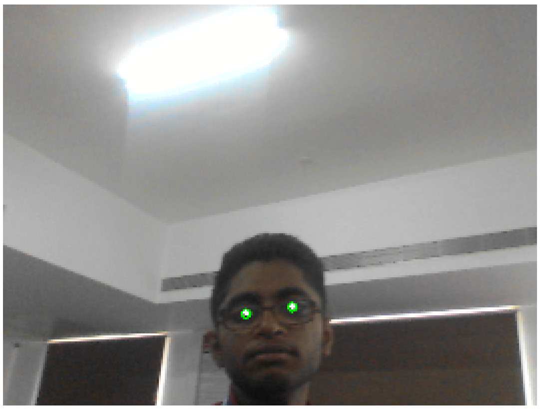
```
        # Flip and convert the image to RGB
        image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Process the image and detect face landmarks
        results = face_mesh.process(image)

        # Convert the image color back for rendering
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Skip the first 50 frames and the last 50 frames
        if process_start_frame <= frame_number < process_end_frame:
        # Initialize data to write for the current frame
        data_to_write = [frame_number, None, None, None, None, None, None,
None, None, None, None, None, None]

        if results.multi_face_landmarks:
            for face_landmarks in results.multi_face_landmarks:

                # Function to calculate the distance between two
landmarks
                def calculate_distance(landmark1, landmark2):
                x1, y1 = face_landmarks.landmark[landmark1].x,
face_landmarks.landmark[landmark1].y
                x2, y2 = face_landmarks.landmark[landmark2].x,
face_landmarks.landmark[landmark2].y
                pixel_distance = np.sqrt((x1 - x2) ** 2 + (y1 - y2)
** 2) * np.array([image.shape[1], image.shape[0]])
                return np.linalg.norm(pixel_distance)

                # Calculate the distance between the outer corners of
the eyes
                eye_distance =
calculate_distance(LEFT_EYE_OUTER_CORNER, RIGHT_EYE_OUTER_CORNER)

                # Check if the person is within the 59-60 cm range
                if pixel_distance_min <= eye_distance <=
pixel_distance_max:

                # Function to calculate center and radius for given
indices
                def calculate_center_and_radius(indices):
                    x = np.mean([face_landmarks.landmark[i].x for i
in indices])
                    y = np.mean([face_landmarks.landmark[i].y for i
in indices])
                    center = (int(x * image.shape[1]), int(y *
image.shape[0]))
                    radius = int(np.linalg.norm(np.array([
```

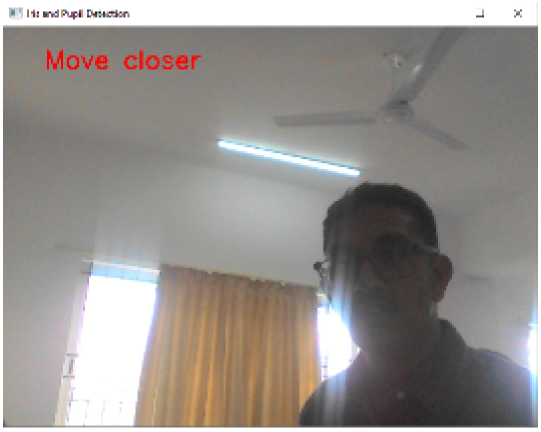
```
                                face_landmarks.landmark[indices[0]].x -
face_landmarks.landmark[indices[2]].x,
                                face_landmarks.landmark[indices[0]].y -
face_landmarks.landmark[indices[2]].y
                                ]) * np.array([image.shape[1],
image.shape[0]])) / 2)
                                return center, radius

                          # Calculate and draw circles for left iris
                          left_iris_center, left_iris_radius =
calculate_center_and_radius(LEFT_IRIS)
                          cv2.circle(image, left_iris_center, left_iris_radius,
(0, 255, 0), 2)  # Green color for iris
                          data_to_write[1:4] = [left_iris_center[0],
left_iris_center[1], left_iris_radius]

                          # Calculate and draw circles for right iris
                          right_iris_center, right_iris_radius =
calculate_center_and_radius(RIGHT_IRIS)
                          cv2.circle(image, right_iris_center,
right_iris_radius, (0, 255, 0), 2)  # Green color for iris
                          data_to_write[4:7] = [right_iris_center[0],
right_iris_center[1], right_iris_radius]

                          # Calculate and draw circles for left pupil
                          left_pupil_center, left_pupil_radius =
calculate_center_and_radius(LEFT_PUPIL)
                          if left_pupil_radius > 0:  # Ensure valid radius
before drawing
                                cv2.circle(image, left_pupil_center,
left_pupil_radius, (255, 0, 0), 2)  # Blue color for pupil
                                data_to_write[7:10] = [left_pupil_center[0],
left_pupil_center[1], left_pupil_radius]
                          else:
                                # Set default values if the radius is not valid
                                data_to_write[7:10] = ['NA', 'NA', 'NA']

                          # Calculate and draw circles for right pupil
                          right_pupil_center, right_pupil_radius =
calculate_center_and_radius(RIGHT_PUPIL)
                          if right_pupil_radius > 0:  # Ensure valid radius
before drawing
                                cv2.circle(image, right_pupil_center,
right_pupil_radius, (255, 0, 0), 2)  # Blue color for pupil
                                data_to_write[10:13] = [right_pupil_center[0],
right_pupil_center[1], right_pupil_radius]
                          else:
                                # Set default values if the radius is not valid
                                data_to_write[10:13] = ['NA', 'NA', 'NA']

                          # Write data for the current frame to the CSV file
```

```
                                        writer.writerow(data_to_write)

                                        else:
                                        # Indicate whether the person is too close or too far
                                        if eye_distance < pixel_distance_min:
                                                  cv2.putText(image, "Move closer", (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
                                        elif eye_distance > pixel_distance_max:
                                                  cv2.putText(image, "Go back", (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

                    # Display the image with detected iris and pupil
                    cv2.imshow('Iris and Pupil Detection', image)
                    frame_number += 1

                    # Exit if 'Esc' is pressed
                    if cv2.waitKey(5) & 0xFF == 27:  # 'Esc' key to exit
                    break

          cap.release()
          cv2.destroyAllWindows()
```

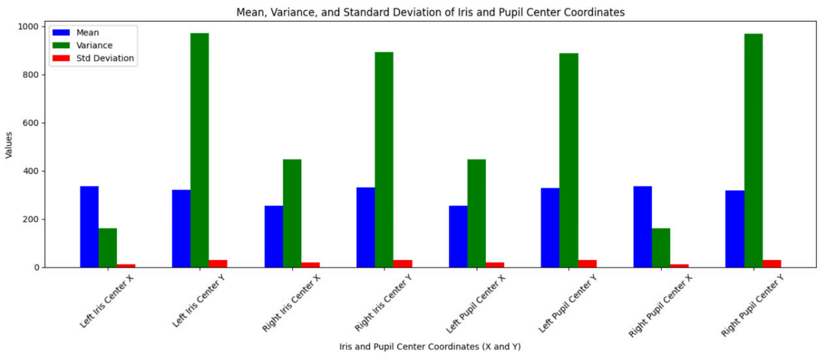
RESULTS

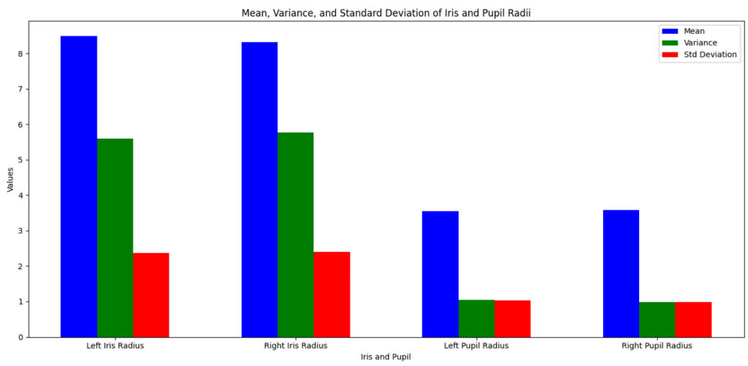






Frame	Timestamp	Left Iris Center X	Left Iris Center Y	Left Iris Radius	Right Iris Center X	Right Iris Center Y	Right Iris Radius	Left Pupil Center X	Left Pupil Center Y	Left Pupil Radius	Right Pupil Center X	Right Pupil Center Y	Right Pupil Radius	Left Pupil/Iris Ratio	Right Pupil/Iris Ratio
303	16:04:37	356	392	6	301	410	5	300	409	2	356	390	2	0.3333	0.4
304	16:04:37	355	393	5	300	412	5	299	411	2	355	392	2	0.4000	0.4
306	16:04:38	354	396	5	300	413	5	300	412	2	354	394	2	0.4000	0.4
307	16:04:38	355	396	5	299	414	5	299	413	2	355	395	2	0.4000	0.4
308	16:04:38	355	396	5	298	414	5	298	412	2	354	395	2	0.4000	0.4
309	16:04:38	354	398	6	298	415	5	298	414	2	353	397	2	0.3333	0.4
310	16:04:38	353	399	6	299	415	5	299	414	2	353	397	2	0.3333	0.4
311	16:04:38	354	399	6	299	415	5	299	414	2	354	398	2	0.3333	0.4
312	16:04:38	358	388	5	302	403	5	302	401	2	358	387	2	0.4000	0.4
313	16:04:38	360	385	5	304	402	5	304	401	2	360	384	2	0.4000	0.4
	Average	355	394	5	300	411	5	300	410	2	355	393	2	0.3733	0.4





CONCLUSION

Experimentation with mediapipe and analysis of various methods has allowed us to conclude that mediapipe, with suitable modifications is the most viable method to achieving the stated project goals. Other methods utilised fail to give accurate and precise results, or need specific parameter adjustments to achieve accurate results. We therefore recommend mediapipe or or a similar **neural network** to perform the task of iris segmentation.