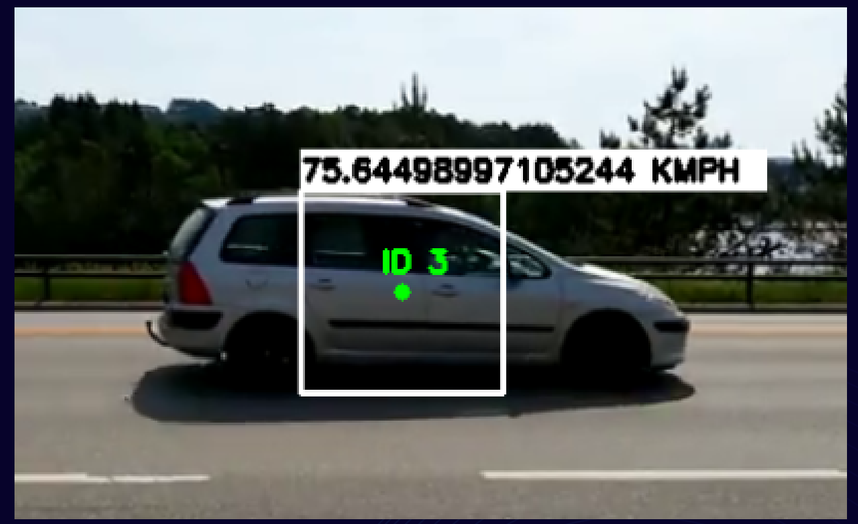
DISTANCE AND SPEED MEASUREMENT USING SINGLE CAMERA

PRESENTED BY: **RAJNISH KUMAR (2K21/IT/139)**
RITIK (2K21/IT/145)
RUGUNG DAIMARY (2K21/IT/151)

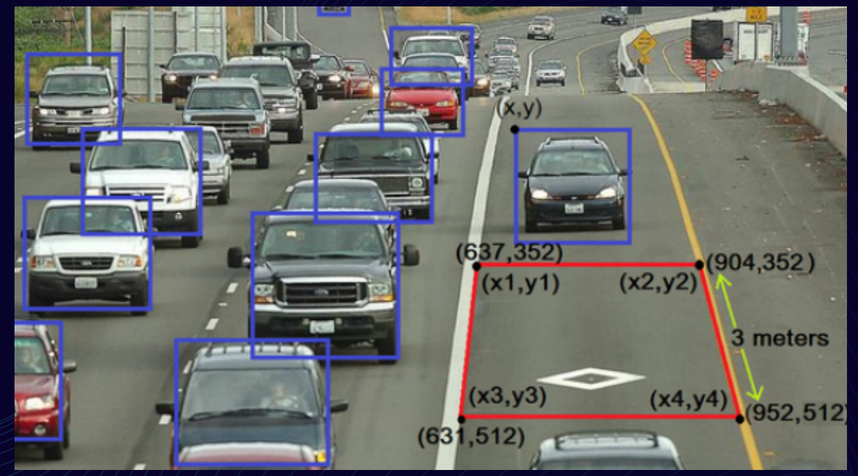
INTRODUCTION

In this project, we explore the exciting field of computer vision applied to the tasks of distance and speed measurement using a single camera setup. Computer vision plays a critical role in modern technology, enabling machines to understand and interpret visual information from their surroundings.



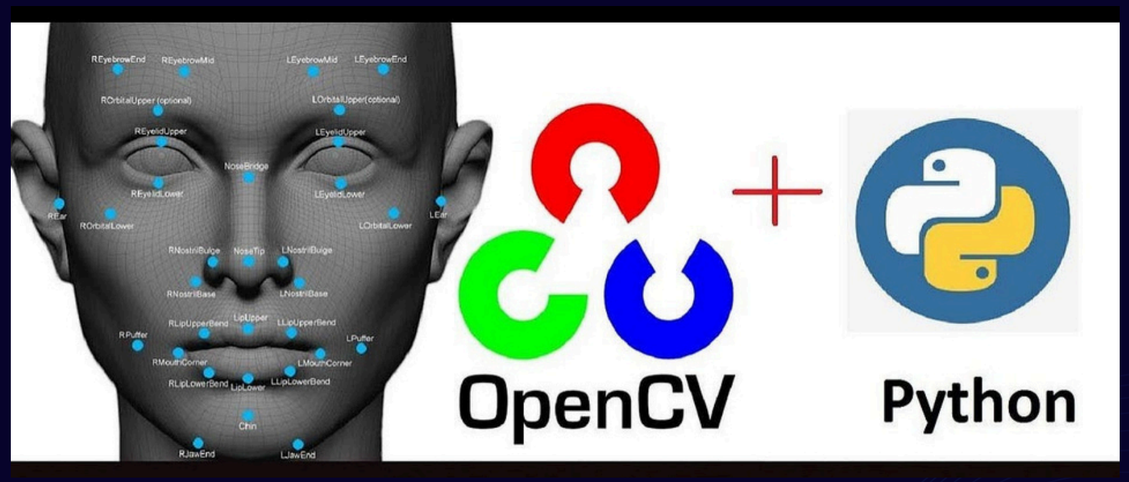
Importance of Distance and Speed Measurement

- Accurate distance and speed measurement are essential for applications such as:
  - Traffic monitoring and management
  - Object detection and tracking
  - Sports analytics and performance analysis
  - Robotics navigation and obstacle avoidance



Utilizing OpenCV for Distance and Speed Estimation

- Introduction to OpenCV
  - OpenCV (Open Source Computer Vision Library) is a powerful open-source library that provides tools and algorithms for image and video processing, including features for computer vision tasks.
- Benefits of OpenCV for Single Camera Measurements
  - OpenCV offers a comprehensive suite of functions to:
    - Capture and process video streams from cameras.
    - Implement algorithms for object detection, feature extraction, and motion analysis.
    - Calculate distances, speeds, and trajectories based on camera perspective and image analysis.



Objectives of this Presentation

- Key Objectives
  - Explore techniques and methodologies for estimating distances using single-camera setups.
  - Discuss algorithms and approaches for measuring object speeds based on visual data captured by a single camera.
  - Showcase practical examples and demonstrations using Python and OpenCV for real-time distance and speed measurement.

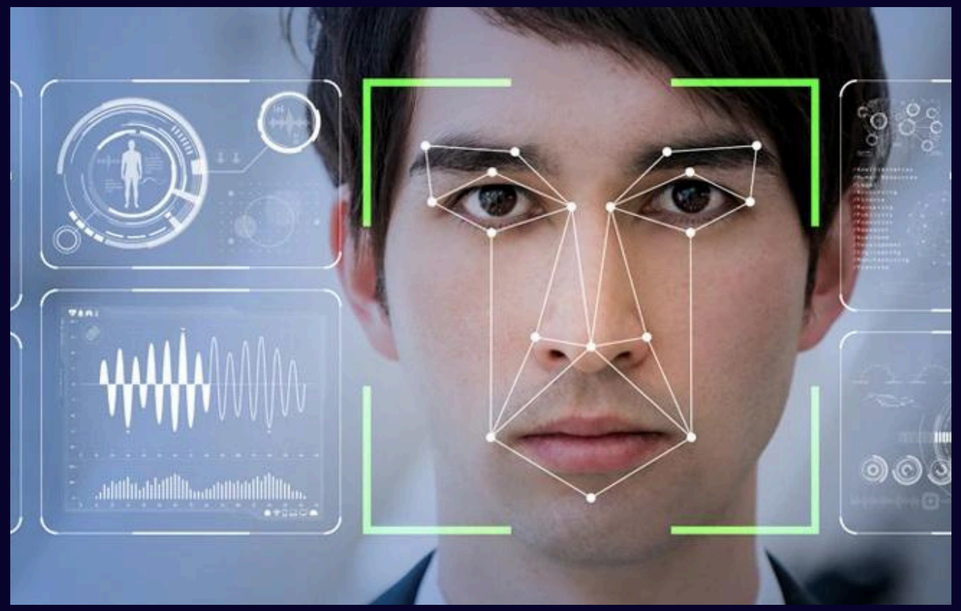
The project's implementation involves several key components

- **Face Detection**
- **Focal Length Calculation** : Calculate the focal length using a known distance and width of a reference object.
- **Distance Estimation** : Utilize the calculated focal length to estimate distances of objects from the camera.
- **Speed Calculation** : Measure the change in distance over time to calculate the speed of the object.
- **Visualization and Output**

Face Detection

- The system employs a pre-trained Haar Cascade classifier from OpenCV to detect human faces in each frame of the video feed.

- The face detection step helps the system identify and focus on the important areas (where the faces are) in each video frame, so it can analyze those specific regions for measuring distance and movement.



FACE DETECTION

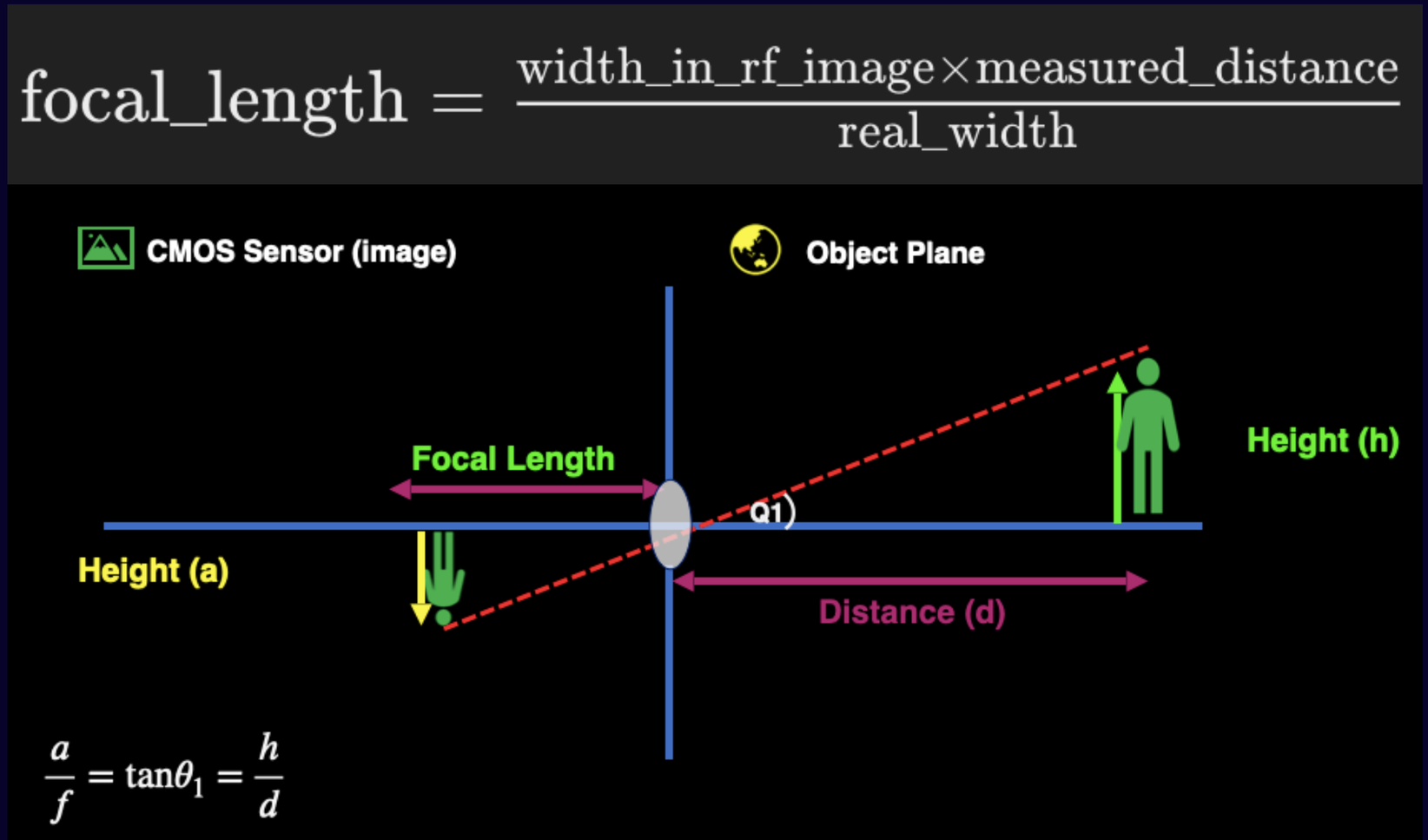
```
def face_data(image, CallOut, Distance_level):

    face_width = 0
    face_x, face_y = 0, 0
    face_center_x = 0
    face_center_y = 0
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray_image, 1.3, 5)
    for (x, y, h, w) in faces:
        line_thickness = 2
        LLV = int(h*0.12)

        cv2.line(image, (x, y+LLV), (x+w, y+LLV), (GREEN), line_thickness)
        cv2.line(image, (x, y+h), (x+w, y+h), (GREEN), line_thickness)
        cv2.line(image, (x, y+LLV), (x, y+LLV+LLV), (GREEN), line_thickness)
        cv2.line(image, (x+w, y+LLV), (x+w, y+LLV+LLV),
                 (GREEN), line_thickness)
        cv2.line(image, (x, y+h), (x, y+h-LLV), (GREEN), line_thickness)
        cv2.line(image, (x+w, y+h), (x+w, y+h-LLV), (GREEN), line_thickness)

        face_width = w
        face_center = []
        face_center_x = int(w/2)+x
        face_center_y = int(h/2)+y
        if Distance_level < 10:
            Distance_level = 10

        if CallOut == True:

            cv2.line(image, (x, y-11), (x+180, y-11), (ORANGE), 28)
            cv2.line(image, (x, y-11), (x+180, y-11), (YELLOW), 20)
            cv2.line(image, (x, y-11), (x+Distance_level, y-11), (GREEN), 18)

    return face_width, faces, face_center_x, face_center_y
```

Focal Length Calculation

Methodology:

- Measure the distance from the camera to an object of known width
- Capture a reference image
- Using the known distance and width, calculate the focal length using the formula

$$\text{focal\_length} = \frac{\text{width\_in\_rf\_image} \times \text{measured\_distance}}{\text{real\_width}}$$



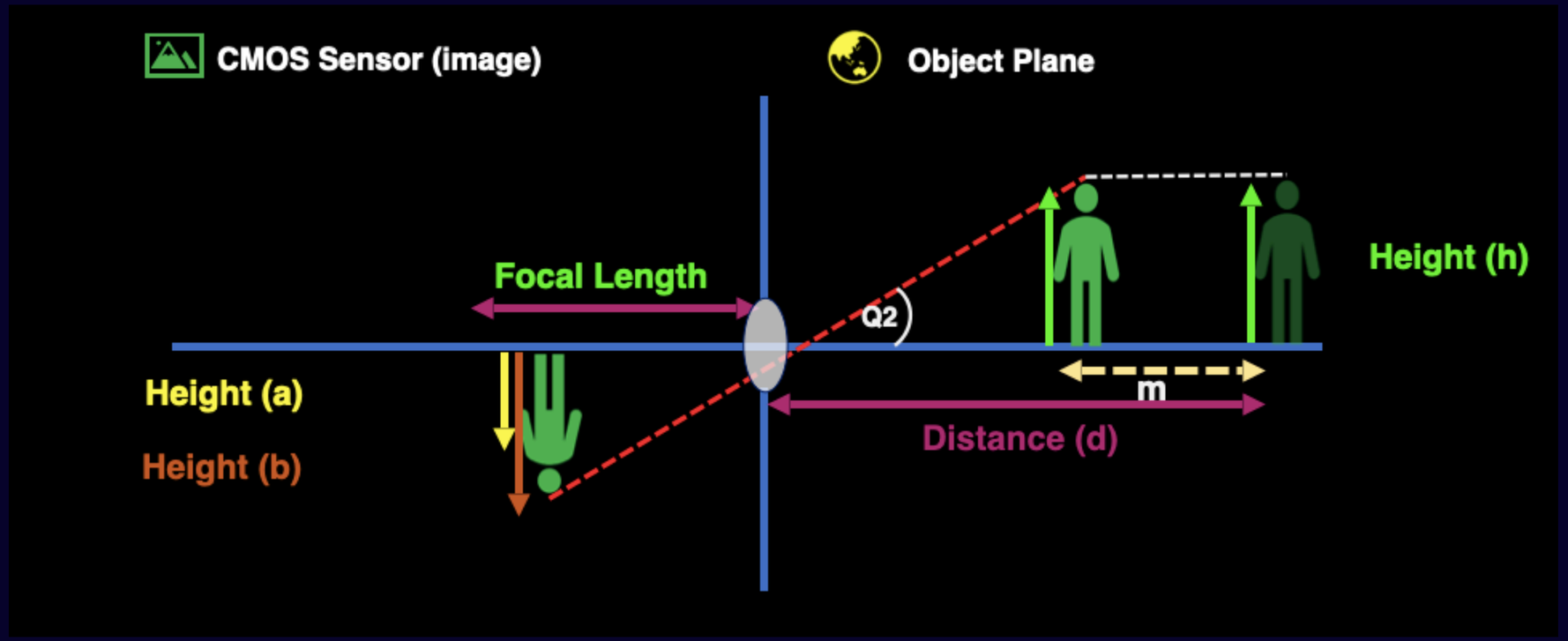
Distance Estimation

Methodology:

- Detect faces in the frame using Haar Cascade Classifier.
- Measure the width of the face in the frame.
- Using the calculated focal length, estimate the distance of the face from the camera using the formula

DISTANCE ESTIMATION

```
def Distance_finder(Focal_Length, real_face_width, face_width_in_frame):
    distance = (real_face_width * Focal_Length)/face_width_in_frame
    return distance
```



$$\frac{a}{f} = \tan\theta_1 = \frac{h}{d}$$ → Equation No: 1

$$\frac{b}{f} = \tan\theta_2 = \frac{h}{d-m}$$ → Equation No: 2

Divide Equation 1 with Equation 2 We get:

$$\frac{a}{b} = \frac{h}{d} \times \frac{d-m}{h}$$

$$\frac{a}{b} = \frac{d-m}{d} = 1 - \frac{m}{d}$$

$$\frac{m}{d} = 1 - \frac{a}{b}$$

$$d = \frac{m}{1 - \frac{a}{b}}$$

Speed Calculation

Methodology:

- Measure the change in distance over time.
- Calculate the speed of the object using the formula:

The averageFinder function computes the average of the last numberElements values in a list, facilitating the smoothing of data by providing a more stable representation of recent values. It aids in reducing noise or fluctuations in data analysis.

SPEED

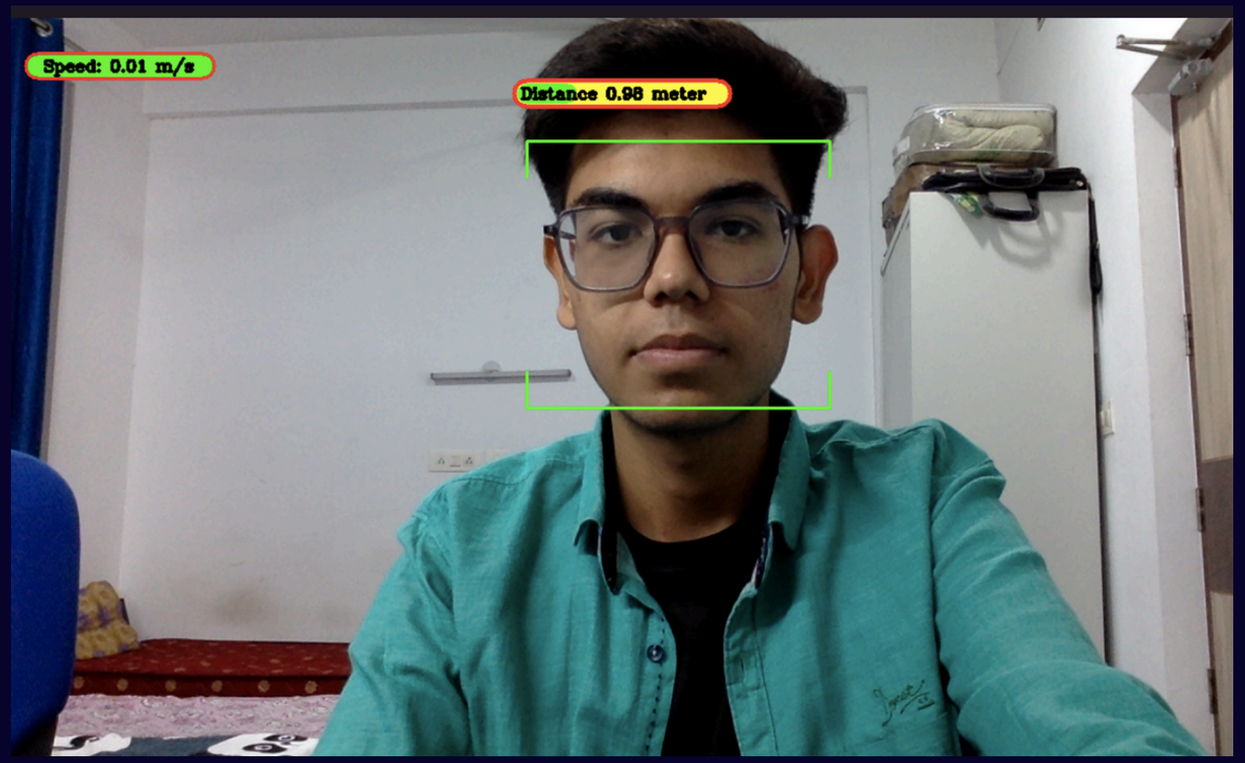
```
def speedFinder(distance, takenTime):
    speed = distance/takenTime
    return speed
```

AVERAGE FINDER

```
def averageFinder(valuesList, numberElements):
    sizeOfList = len(valuesList)
    lastMostElement = sizeOfList - numberElements
    lastPart = valuesList[lastMostElement:]
    average = sum(lastPart)/(len(lastPart))
    return average
```

Visualization and Output

The project provides real-time visualization of the detected face, overlaid with information about the estimated distance and speed. Additionally, the processed video feed, including the distance and speed measurements, can be saved as an output video file for further analysis or record-keeping purposes.



THANK YOU!!