99220040478-DAMALOKESH.pdf
03/28/2024 8:53 AM

FACE DETECTION USING OPENCV A Micro Project Report Submitted by DAMA LOKESH Reg.no: 99220040478 B.Tech – CSE, AIML Kalasalingam Academy of Research and Education (Deemed to be University) Anand Nagar, Krishnankoil - 626 126 Febraury 2024 SCHOOL OF COMPUTING DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING BONAFIDE CERTIFICATE Bonafide record of the work done by [Student Name] - [Registration Number] in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Specialization of the Computer Science and Engineering, during the Academic Year [Even/Odd] Semester (2023-24) Mr.V.Manikandan Mr.B.Sakthi Karthi Durai Project Guide Faculty Incharge CSE Department CSE Department Kalasalingam Academy of Kalasalingam Academy of Research and Education Research and Education Krishnan kovil – 626126 Krishnan kovil - 626126 Ms.V.Ragavarthini Evaluator Ass.Professor CSE CSE Department Kalasalingam Academy of Research and Education Krishnan kovil – 626126 i Abstract This study presents a novel approach to face detection utilizing OpenCV, coupled with methodologies for real-time calculation of speed and distance between the camera and detected faces. Leveraging OpenCV's comprehensive suite of computer vision algorithms, face detection is achieved through a combination of traditional Haar cascades and modern deep learning-based models like Single Shot multibox Detector (SSD) or You Only Look Once (YOLO). Subsequently, the system integrates techniques for speed estimation by tracking identified faces across consecutive frames, employing optical ow or Kalman filtering algorithms for accurate motion analysis. Furthermore, distance estimation is addressed through camera calibration techniques and depth perception algorithms, enabling precise determination of the distance between the camera and detected faces. The implementation of these functionalities within an OpenCV framework facilitates real-time face detection with concurrent speed and distance analysis, suitable for applications spanning surveillance, human-computer interaction, and automotive safety systems. Experimental validation demonstrates the efficacy and reliability of the proposed methodology across diverse realworld scenarios, underscoring its potential for deployment in practical applications requiring robust face detection alongside dynamic speed and distance assessment. ii Contents 1 Introduction 1 1.1 What is Computer Vision . . . . . .
. . . . . 1 1.2 Face Detection . . . . . 1 1.2.1 The Role of OpenCV 1-2
1.2.2 The significance of Face Detection. . . . . .1-2 2 System Study 3 2.1 Objectives 3
2.1.1 Existing Work. . . . . 2.1.2 Literature Survey 3-5 6 2.2 Implementation 6-7
2.2.1 Algorithm used in the proposed work . . . . . .7-8 2.3 Working. . . . . .
. . . . . 8-9 2.3.1 Time
line of work . . . . . 10 3 Experimental Analysis 10 3.1 Coding 11-12 3.1.1
Explanation of code. . . . . 13 3.1.2 About Software Tool . . . . . .
14 3.2 Results. . . . . .
. . . . . 15-16 4 Conclusion and Future Work 17 4.1 Conclusion. . . . . .
16 4.2 Future Work. . . . . .
. . . . . 16-17 iii 5 References 19 6 Certification 20 iv List of Figures 1.1 Face Detection 2 2.1
Objectives . . . . . 3 2.2 Implementation . . . . . .
. 5 3.1 Result 1.0 . . . . . .
. . . . . 7 3.2 Result 2.0 . . . . . 7 6.1 Certification details 11 Chapter 1
INTRODUCTION 1.1 What is Computer Vision Computer vision is a field of artificial intelligence (AI) that enables computers and systems to interpret and understand the visual world from digital images, videos, and other visual inputs. It involves the development and application of algorithms and mathematical models to extract meaningful information and insights from visual data, much like how humans perceive and understand visual information. 1.2 Face Detection Face detection is a critical component of security systems. It is used in surveillance cameras to identify and track individuals in public places, airports, and other high-security areas. It can be employed to detect unauthorized access, monitor crowds, and enhance overall security. Face detection and recognition are used for biometric authentication, allowing individuals to unlock devices, access buildings, or log into systems using their faces. It provides a convenient and secure means of access control. Face detection is used in social media platforms for tagging and recognizing faces in photos. It also aids in photography by enabling features like autofocus and exposure adjustment based on detected faces. Medical Imaging: In the medical field, face detection can assist in identifying and locating facial features for various purposes, such as disease diagnosis, plastic surgery

planning, and monitoring patient conditions. 1.2.1 The role of OpenCV OpenCV, an open-source computer vision library, has gained widespread recognition and adoption for its versatility and extensive feature set. It provides a comprehensive suite of functions and tools tailored for various computer vision tasks, including face detection. With OpenCV's user-friendly interface and well-documented functions, developers can implement robust face detection systems efficiently. In this exploration of face detection using OpenCV, we will delve into the key components of a face detection system, including image acquisition, preprocessing, face segmentation, and postprocessing. We will also discuss practical applications across domains such as robotics, image processing, and quality

control. This journey into face detection with OpenCV will equip readers with the knowledge and tools needed to harness the power of face information in computer vision, opening doors to a wide array of innovative applications.. 1 1.2.2 The Significance Of Face Detection Face detection is crucial in security and surveillance systems for identifying individuals in realtime, tracking their movements, and recognizing suspicious behavior. It is extensively used in airports, public spaces, and high-security areas for monitoring and ensuring public safety. Face detection forms the basis for biometric identification systems, enabling secure access control to devices, buildings, and digital services. It is used in smartphones, computers, and other devices for authentication purposes, replacing traditional methods like passwords or PINs. 2 Chapter 2 SYSTEM STUDY 2.1 Objective of the project The objectives for a face detection project can vary depending on the specific application and requirements, but some common objectives could include: 1. face detection is to locate human faces within an image or a video frame. 2. In face detection the camera only detects the presence of human face and ignores the rest of the objects. Figure 2.1: Image Title 2.1.1Existing Work Face detection is a well-studied and active area of computer vision, and numerous methods and models have been developed for this task. Here are some notable existing works and approaches for face detection: 1. Haar Cascade Classifiers: Haar Cascade Classifiers, as mentioned earlier, are simple yet effective for face detection. They use a series of simple image features and a cascade of classifiers to detect faces. OpenCV provides pre-trained Haar Cascade models for face detection. 3 2. Viola-Jones Face Detection Framework: The Viola-Jones framework, which Haar Cascades are based on, introduced a robust and real-time face detection method. It has been widely used and adapted for various applications. 3. Histogram of Oriented Gradients (HOG): HOG is a feature descriptor that captures the shape and appearance of objects in an image. It has been used for face detection by training classifiers on HOG features. 4. Deep Learning-Based Approaches: • • • Convolutional Neural Networks (CNNs): Deep CNNs have been employed for face detection. Models like Multi-task Cascaded CNNs (MTCNN) and Single Shot MultiBox Detector (SSD) have achieved state-of-the-art performance. You Only Look Once (YOLO): YOLO is a real-time object detection system, and YOLOv3 and YOLOv4 architectures have been used for face detection. RetinaNet: This is another deep learning-based object detection model that has been adapted for face detection. 5. Cascade Classifier Variants: • • Improved Cascade Classifier: Researchers have proposed enhancements to the original Haar Cascade Classifier, improving accuracy and speed. R-CNN Family: Models like Faster R-CNN and Mask R-CNN, designed for object detection and segmentation, have also been applied to face detection. 6. Ensemble Methods: Techniques like AdaBoost and Random Forests have been used in conjunction with Haar Cascades or other classifiers to improve face detection accuracy. 7. 3D Face Detection: Some methods focus on detecting faces in 3D space, which is useful for applications like facial recognition and augmented reality. 8. Privacy-Preserving Face Detection: Techniques for face detection while preserving privacy, such as privacy filters or privacy-preserving federated learning. 9. Efficient Models: Lightweight and efficient models designed for deployment on edge devices, such as MobileNets and SqueezeNet, have been adapted for face detection in resourceconstrained environments. 10. Real-time and Mobile Applications: Many face detection methods have been optimized for realtime and mobile applications, allowing for efficient face detection on smartphones and tablets. 11. Large-Scale Face Datasets: The availability of large- scale datasets, such as LFW, CelebA, and WIDER FACE, has facilitated advancements in face detection research. 4 12. Domain-Specific Face Detection: Face detection models have been trained for specific domains, like detecting masked faces during the COVID-19 pandemic. 13. Privacy and Ethical Considerations: Research in privacy-preserving and ethical face detection, addressing issues related to bias, fairness, and consent.

The field of face detection continues to evolve with advancements in deep learning, hardware acceleration, and real-world applications. Researchers and developers are actively working on improving accuracy, speed, and robustness in various contexts, including surveillance, security, healthcare, and human-computer interaction. 5 2.1.2Literature Survey Author Tittle Real-time face detection using OpenCV Robust Detection Real-Time Bradski, G. Methodology Viola-Jones algorithm with Haar-like features Face Yang, M.H. et al. Boosted Cascade of Simple Features (BCSF) algorithm Facial Detection and Tracking Bradski, G. using OpenCV Cascade Classifier with Haar-like features Efficient Face Detection Method for Mobile Applications Faster R-CNN algorithm Le, T.H. et al. 2.2 Implementation To implement face detection using OpenCV and calculate the distance between the camera and the face, you can follow these steps: 1.Install OpenCV: Make sure you have OpenCV installed on your system. You can install it using pip: bashCopy code pip install opencv-python 6 2.Capture Video Stream: Use OpenCV to capture video from the camera. 3.Face Detection: Utilize OpenCV's pre-trained face detection models to detect faces in each frame of the video stream. 4.Distance Estimation: To estimate the distance between the camera and the face, you need to calibrate your camera. This involves determining the focal length of the camera and establishing a reference distance. One common method for distance estimation involves using the physical dimensions of the face in the image and the known dimensions of a reference object (like a ruler) placed at a known distance from the camera. 5.Calculate Distance: Once you have the focal length and reference distance, you can use the formula: makefileCopy code distance = (known_width * focal_length) / pixel_width Where known_width is the physical width of the face, focal_length is the focal length of the camera (in pixels), and pixel_width is the width of the face in the image (in pixels). 6.Display Results: Draw bounding boxes around the detected faces and display the distance information on the video stream. 2.2.1 Algorithm Used To propose a face detection algorithm capable of calculating speed and distance between the camera and the face, you would typically need to incorporate computer vision techniques along with additional methods for speed and distance estimation. Here's an outline of an algorithm that could be used: 1. Face Detection: •Use a face detection model such as Haar cascades, HOG (Histogram of Oriented Gradients), or deep learning-based methods like SSD (Single Shot Multibox Detector) or YOLO (You Only Look Once) to detect faces in each frame of the video stream. •Implement the face detection algorithm to identify and locate faces accurately. 2. Speed Estimation: •To estimate the speed of the face, you would need to track the movement of the face across

consecutive frames. •Apply optical ow techniques (e.g., Lucas-Kanade method, Farneback method) or object tracking algorithms (e.g., Kalman filters, Camshift) to track the movement of the face. • Calculate the displacement of the face between frames and divide it by the time elapsed to estimate the speed. 3. Distance

Estimation: •To estimate the distance between the camera and the face, you need to calibrate the camera and establish a reference distance. 7

- Use methods such as stereo vision, depth sensors (e.g., Kinect), or monocular depth estimation techniques to estimate depth. • Alternatively, you can use a reference object with known dimensions placed at a known distance from the camera to calibrate and estimate depth based on the size of the face in the image. 4. Integration: •Integrate the face detection, speed estimation, and distance estimation modules into a cohesive system.
- Process each frame of the video stream sequentially, detecting faces, estimating their speed and distance, and updating the display or output with the relevant information. 5. Testing and Evaluation: •Evaluate the performance of the algorithm using appropriate metrics such as accuracy, speed of computation, and robustness to variations in lighting conditions, camera angles, and facial expressions. •Test the algorithm in real-world scenarios to assess its effectiveness and identify any limitations or areas for improvement. 6. Optimization and Refinement: •Optimize the algorithm for speed and efficiency, considering computational resources and realtime processing requirements. •Refine the algorithm based on feedback and observations from testing to improve accuracy and reliability. This algorithm would require a combination of computer vision techniques, motion tracking, and depth estimation methods to achieve accurate face detection along with speed and distance estimation. Additionally, the algorithm should be adaptable to different environments and conditions to ensure robust performance in various scenarios. Figure 2.2.1: Algorithm Used in proposed work 8 2.3 Working Face detection using OpenCV involves several key steps, and it typically relies on the use of a pretrained Haar Cascade Classifier or a deep learning-based model like a Convolutional Neural Network (CNN). Here's a high-level overview of how face detection works in OpenCV using the Haar Cascade Classifier: 1.Loading the Classifier: You first load a pre-trained Haar Cascade Classifier specifically designed for face detection. These classifiers are XML files that contain information about the features of faces and nonfaces, learned from a large dataset during training. 2.Capture Video or Images: You can either capture video frames from a webcam, as in your provided code, or you can read images from a file. 3. Preprocessing: If you're working with a video feed, each frame is usually preprocessed. In your code, you convert the frame to grayscale because Haar Cascade Classifiers work on grayscale images. Grayscale simplifies the image and reduces computation time. 4.Face Detection: The core of the process is detecting faces in the preprocessed image using 'detectMultiScale' from the loaded classifier. This function scans the image at different scales and detects regions that match the learned patterns of faces. - 'scaleFactor': It compensates for faces appearing smaller as they are farther from the camera. - 'minNeighbors': It helps in filtering out false positives by requiring multiple detections to agree before considering a region as a face. - 'minSize': It sets the minimum size of the detected face. 5.Drawing Rectangles: Once faces are detected, you can draw rectangles around them to highlight the detected regions. 6.Display Results: You can display the image or video frame with the rectangles drawn around the detected faces to visualize the results. 7.Termination Condition: Usually, you run face detection in a loop, continuously processing frames or images. You can specify a condition to terminate the loop, such as pressing a specific key ('q' in your code). 8.Cleanup: After processing, you release any resources (e.g., webcam) and close any windows. This process repeats for each frame or image, allowing real-time or batch face detection. Haar Cascade Classifiers are computationally efficient for basic face detection but may have limitations in handling various orientations, lighting conditions, or occlusions. More advanced methods, like deep learningbased models (e.g., OpenCV's DNN module with models like SSD, YOLO, or FaceNet), can offer better performance in complex scenarios. 9 2.3.1 Timeline of work proposal Week-1 Learnt about Analysing and Classifying Images with the Computer Vision Service Week-2 Learnt about Detect objects in images with the Custom Vision service Week-3 Learnt about Analyse Faces, Text, and Receipts with Azure Al Week-4 Learnt about Analyse Faces, Text, and Receipts with Azure Al Week-5 Done with executing the code for the project Week-6 Done with ppt and report for review 10 Chapter 3 Experimental Analysis 3.1 Sample Code import cv2 import numpy as np import time # Initialize the camera cap = cv2.VideoCapture(0) # Use 0 for the default camera, change if necessary # Load the pre-trained face detection model face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades 'haarcascade_frontalface_default.xml') + # Initialize variables for distance and speed estimation last_time = time.time() last_position = None # Loop to capture frames from the camera while True: # Capture frame-by-frame ret, frame = cap.read() # Convert the frame to grayscale for face detection = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) gray # Detect faces in the frame faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30)) # Draw rectangles around the detected faces for (x, y, w, h) in faces: cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2) # Calculate distance and speed if len(faces) > 0: current_position = np.mean(faces[:, 0]) # Mean x-coordinate of detected faces last_position is not None: distance = abs(current_position - last_position) speed = distance / (time.time() - last_time) print("Distance:", distance) print("Speed:", 11 if speed) last_position = current_position last_time = time.time() # Display the resulting frame cv2.imshow('Face Detection', frame) # Break the loop when 'q' is pressed if cv2.waitKey(1) & 0xFF == ord('q'): break # Release the capture cap.release() cv2.destroyAllWindows() 12 3.1.1 Explanation Of Code This code is a Python script that utilizes OpenCV to perform real-time face detection from a webcam feed. It also estimates the distance and speed between the detected faces in consecutive frames. Let's break down the code step by step: 1. Import Libraries: • cv2: OpenCV library for computer vision tasks. • Numpy: library for numerical computations in Python. • time: Standard Python library for time-related functions. 2. Initialization: • cap = cv2.VideoCapture(0): Initializes the camera capture object. It sets up the camera to capture frames. Here, 0 refers to the default camera. If you have multiple cameras, you can change this number accordingly. • Face _cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'): Loads the pre-trained Haar cascade classifier for face detection. 3. Variables Initialization: • last_time: Records the time of the previous frame. • last_position: Stores the x-

coordinate position of the detected face in the previous frame. 4. Main Loop: • while True:: Enters an infinite loop to continuously capture frames from the camera until interrupted. 5. Capture Frame: • ret, frame = cap.read(): Captures a frame from the camera. ret indicates whether the frame is successfully read, and frame contains the captured image. 6. Face Detection: • gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY): Converts the captured frame to grayscale. Grayscale images are commonly used for face detection as they reduce

computational complexity. • faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30)): Detects faces in the grayscale frame using the pre-trained cascade classifier. Detected faces are returned as rectangles with their coordinates. 7. Draw Rectangles: • Draws rectangles around the detected faces on the original frame using cv2.rectangle. 8. Distance and Speed Calculation: • If faces are detected (len(faces) > 0), the x-coordinate position of the detected face is calculated as the mean of all detected faces. 13 • If there was a previously detected face (last_position is not None), the distance and speed between the current and previous positions are calculated. Distance is calculated as the absolute difference in xcoordinates, and speed is calculated as the ratio of distance to the time elapsed between frames. 9.
Display: • The resulting frame with detected faces and calculated distance and speed information is displayed using cv2.imshow. 10. Exit Condition: • The loop breaks when the 'q' key is pressed (if cv2.waitKey(1) & 0xFF == ord('q'):). 11. Release Resources: • Releases the camera capture (cap.release()) and closes all OpenCV windows (cv2.destroyAllWindows()) after exiting the loop. Overall, this code demonstrates a simple real- time face detection and speed/distance estimation application using OpenCV. It provides a foundation that can be extended for various computer vision projects involving face tracking and analysis. 3.1.2 About Software tool In face detection using OpenCV, you can utilize various libraries and tools for different aspects of the task, including speed and distance estimation. Here's a general approach: 1. Face Detection with OpenCV : OpenCV provides robust face detection algorithms through its libraries. You can use either Haar cascades or more modern deep learning based approaches like Single Shot Multibox Detector (SSD), You Only Look Once (YOLO), or Faster R-CNN for face detection. 2. Speed Estimation: Speed estimation typically requires tracking the detected face across frames and calculating its displacement over time. This can be done using techniques such as optical ow or tracking algorithms like the Kalman filter. 3. Distance Estimation: Estimating distance from a camera to a face can be more complex. It often involves either using stereo vision (depth perception through multiple cameras) or by calibrating the camera and using known dimensions of objects in the scene to estimate depth. 4. Integration with OpenCV: You can integrate these functionalities into your OpenCV-based application by combining appropriate libraries and algorithms. For instance, you can use OpenCV for face detection, then implement custom code or utilize libraries like NumPy or SciPy for speed and distance estimation. 14 5. Application-Specific Tools: Depending on your specific application requirements, you might find specialized tools or libraries that offer combined functionalities for face detection, speed, and distance estimation. However, you may need to customize or extend these tools to fit your exact needs. Remember that accurate speed and distance estimation might require careful calibration andtesting in real-world scenarios. Additionally, factors such as lighting conditions, camera specifications, and the environment can affect the performance of these algorithms 3.2 RESULT Figure displays the outcome of the face detection process. They are the HD video streaming frames that were taken out. Even though there are just one or two face in the frame, the face detection system occasionally produces many results. Figure 3.1: Result 1.0 15 Figure 3.2: Result 2.0 16 Chapter 4 Conclusion and Future Work 4.1 conclusion In conclusion, integrating distance and speed calculation capabilities into face detection systems using OpenCV presents a promising avenue for advancing various applications, including but not limited to surveillance, robotics, and human-computer interaction. By leveraging techniques such as depth sensing, 3D reconstruction, optical ow analysis, and predictive modeing, it becomes feasible to not only detect faces but also accurately estimate their distance from the camera and their relative speed. However, several challenges remain, including algorithmic complexity, real-time performance optimization, robustness in diverse environments, and validation against ground truth data. Addressing these challenges will require interdisciplinary collaboration, combining expertise in computer vision, machine learning, sensor technology, and application domains. Despite these challenges, the potential benefits of integrating distance and speed estimation with face detection are substantial. Such systems can enhance situational awareness, improve safety and security measures, enable more natural human-robot interaction, and facilitate personalized user experiences. Moreover, as technology continues to advance, the capabilities of these systems are likely to expand, opening up new possibilities for innovation and application development. In conclusion, while there is still work to be done to realize the full potential of face detection systems with distance and speed calculation, the ongoing research and development in this field hold promise for significant advancements in various domains, ultimately enhancing our interaction with and understanding of the world around us. 4.2 Future Work Future work in face detection using OpenCV with capabilities to calculate distance and speed between a person and the camera could involve several advancements and enhancements. Here's a potential roadmap for such developments: 1. Integration of Depth Sensors : One approach could involve integrating depth sensors such as LiDAR or Time-of-Flight cameras with OpenCV. These sensors can provide depth information, allowing for accurate distance estimation between the camera and the detected faces. 2. 3D Face Reconstruction: Utilize techniques for 3D face reconstruction from 2D images. 17 By reconstructing the 3D geometry of the face, it becomes feasible to calculate accurate distances between the camera and the person's face. 3. Optical Flow Analysis: Implement optical ow algorithms to analyze the motion of facial features between consecutive frames. By tracking the displacement of facial landmarks over time, the speed of the person relative to the camera can be estimated. 4. Kalman Filtering and Motion Prediction: Integrate Kalman filters or similar predictive models to estimate the future position of detected faces. This can help in predicting the movement of individuals and estimating their speed more accurately. 5. Camera Calibration: Calibrate the camera to obtain intrinsic and extrinsic parameters, which are essential for accurate distance estimation. This involves determining the focal length, principal point, and lens distortion coefficients. 6. Machine Learning Techniques: Explore machine learning techniques such as regression or neural networks to learn the mapping between visual features and distance/speed. This could involve training models on annotated datasets with ground truth distance and speed information. 7. Real-Time Performance Optimization: Focus on optimizing the algorithms and implementations for real-time

performance, especially considering the computational requirements of depth estimation, motion analysis, and predictive modelling. 18 Chapter 5 References Certainly! Here are some references and resources that can help you with face detection using OpenCV and related tasks such as speed and distance estimation: 1. OpenCV Documentation: • OpenCV provides comprehensive documentation

and tutorials on face detection using various techniques: OpenCV Face Detection 2. Face Detection with OpenCV and Dlib: • This tutorial demonstrates face detection and facial landmark detection using OpenCV and Dlib: Face Detection with OpenCV and Dlib 3. Distance Estimation with OpenCV: • This article discusses distance estimation using known object dimensions and camera calibration in OpenCV: Distance Estimation with OpenCV 4. Speed Estimation with Optical Flow: • Learn about optical ow-based motion estimation for speed calculation using OpenCV: Optical Flow with OpenCV 5. Real-time Face Tracking and Speed Estimation: • This GitHub repository provides code for real-time face tracking and speed estimation using OpenCV: Real-time Face Tracking and Speed Estimation 6. Books: • "OpenCV 4 Computer Vision with Python Recipes" by Gabriel Garrido Calvo and Prateek Joshi includes practical examples and recipes for various computer vision tasks using OpenCV, including face detection and tracking. 7. Research Papers: • Delve into research papers on face detection, tracking, and distance estimation for more advanced techniques and algorithms. 8. Online Courses and Tutorials: • Platforms like Coursera, Udemy, and edX offer courses on computer vision and OpenCV, covering topics such as face detection, tracking, and motion estimation These resources should give you a good starting point for