A PROJECT REPORT ON

"Real Time Object Detection System"

SUBMITTED TO

Department of Computer Science & Engineering (Data Science) in fulfillment of Project Based Learning for the semester-IV of academic year 2022-2023

SUBMITTED BY

Sr.No	RollNo	Name of the Student
1	01	Bhakti Ayarekar
2	10	Ankita Yadav
3	21	Rohan Chopade
4	29	Soham Mangore

UNDER THE GUIDANCE OF

Mrs.N.C.Adnaik

Asst.Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SPECIALIZATION IN DATA SCIENCE

KIT'S COLLEGE OF ENGINEERING,

KOLHAPUR.

YEAR : 2022-2023

CERTIFICATE

KIT's COLLEGE OF ENGINEERING

This is to certify that, the project entitled "Real Time Object Detection System", has been satisfactorily completed by ,

Bhakti Ayarekar (1)
Ankita Yadav (10)
Rohan Chopade (21)
Soham Manogre (29)

the students of SY B.Tech, Department of Computer Science & Engineering, Specialization in Data Science in fulfillment of Mini Project for the semester-IV of academic year 2022-2023.

This project report is a record of student's own work carried by him/her under my supervision and guidance in satisfactory manner.

Date: 24/05/2023

Mrs . N. C. Adnaik	Dr. Uma Gurav	Dr. Mohan Vanarotti
Asst Professor	**HOD**	**Director**

Kolhapur Institute of Technology's
College of Engineering, Kolhapur.
Year 2022-2023

ACKNOWLEDGEMENT

We are highly grateful to the Dr. Uma Gurav, HOD CSE (AIML & DS), KIT's College of Engineering, Kolhapur, for providing this opportunity to carry out the Project Based Learning at CSE(AIML & DS) department. We would like to express our gratitude to other faculty members of department for providing academic inputs, guidance and encouragement throughout this period. We would like to express a deep sense of gratitude and thank to Dr. Uma Gurav and Mrs. Nitila Adnaik without whose permission, wise counsel and able guidance, it would have not been possible to carry out our project in this manner.

We express our indebtedness to all who have directly or indirectly contributed to the successful completion of Project Based Learning.

Date: 24/05/2023

Place: KIT, Kolhapur

Sincerely by,

Name	Roll No
Bhakti Ayarekar	DS01
Ankita Yadav	DS10
Rohan Chopade	DS21
Soham Manogre	DS29

Project Overview

This project involves creating a server application that listens for connections from multiple clients. The clients send real-time video streams to the server, which uses the YOLOv8 object detection model to analyze the frames and detect objects. The server then annotates the video frames with bounding boxes and labels for the detected objects and presents the annotated video in real-time on the server screen. Socket programming is utilized for communication between the clients and the server.

INDEX

INTRODUCTION

"The Real Time Object Detection System is a client-server application developed using socket programming in Python. The client captures live video frames and sends them to the server, where the YOLOv8 model performs object detection. This system enables real-time video streaming with seamless transmission and accurate object detection, making it suitable for applications such as surveillance and video analytics."

PROBLEM STATEMENT

Develop a real-time video streaming application using Python socket programming.

PROJECT PURPOSE

The purpose of this project is to develop a Real Time Object Detection System using socket programming in Python. The system involves a client-server architecture where the client captures live video frames and sends them to the server. The server utilizes the YOLOv8 model for real-time object detection. The main objective is to enable seamless video streaming with accurate object detection, making the system suitable for applications like surveillance and video analytics.

REQUIREMENT ANALYSIS

- Real-Time Video Streaming: The system should enable seamless transmission of live video frames from the client to the server, ensuring real-time updates and minimizing latency.
- Accurate Object Detection: The server should utilize the YOLOv8 model to perform accurate and reliable object detection on the received video frames, providing precise annotations of identified objects.
- Efficiency and Performance: The system should be optimized for efficient video streaming and object detection, ensuring minimal delay and high-speed processing to handle real-time requirements.
- Flexibility for Different Applications: The system should be adaptable to various applications, such as surveillance systems, video analytics, and other scenarios requiring real-time object detection on a video stream.
- Scalability: The system should be designed to accommodate scalability, allowing multiple clients to connect simultaneously and stream their video feeds for object detection, ensuring that the system can handle increased demand and workload effectively.

NON-FUNCTIONAL REQUIRMENTS

- Performance: The system should be capable of processing video frames and performing object detection in real-time, with minimal latency and high throughput
- Maintainability: The system should be designed in a modular and well-structured manner, making it easier to maintain, update, and enhance in the future.
- Resource Efficiency: The system should be resource-efficient, optimizing memory usage, processing power, and network bandwidth to minimize resource consumption.
- Reliability: The system should be reliable, ensuring consistent and uninterrupted video streaming and object detection even in the presence of network issues

TECHNOLOGY

- Operating System: Windows
- Language: Python
- ML model: YOLOv8
- Deep Learning Framework: PyTorch

TECHNICAL IMPLEMENTATION

- Client-Server Architecture: The Real Time Object Detection System utilizes a client-server architecture. The client is responsible for capturing live video frames using a camera or a video source. These frames are then transmitted to the server for processing.

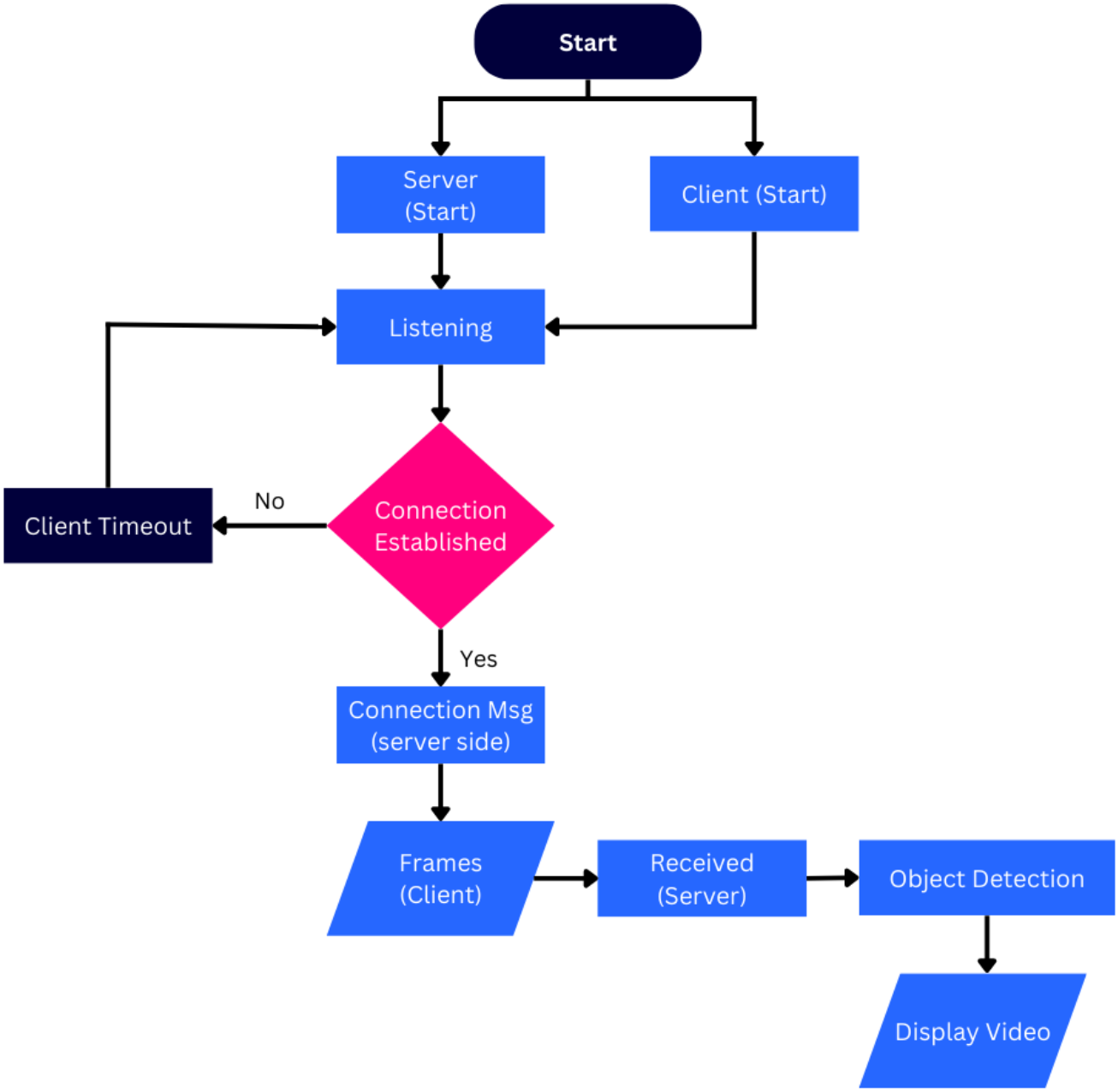
- Socket Programming: The client-server communication is established using socket programming in Python. Sockets enable bidirectional data transfer between the client and server, facilitating the real-time video streaming.

- Live Video Streaming: The client continuously captures video frames and sends them over the network to the server in real-time. This ensures a seamless transmission of video data, allowing for a smooth viewing experience.

- YOLOv8 Object Detection: The server utilizes the YOLOv8 model for object detection on the received video frames. YOLOv8 (You Only Look Once v8) is a popular real-time object detection algorithm known for its speed and accuracy. It processes the frames and identifies objects present in the video stream.

- Object Detection and Annotation: The YOLOv8 model performs object detection by dividing the input frames into a grid and predicting bounding boxes and class probabilities for objects within each grid cell. The detected objects are then annotated on the video frames, providing visual indications of their presence.

FLOWCHART


Start
Server (Start)
Client (Start)
Listening
Client Timeout
No
Connection Established
Yes
Connection Msg (server side)
Frames (Client)
Received (Server)
Object Detection
Display Video

PSEUDOCODE

Client-side pseudocode

1. Initialize the video capture from the webcam or video source.
2. Connect to the server using socket programming.
3. Start a loop to continuously capture and send video frames:
   3.1. Capture a frame from the video source.
   3.2. Convert the frame to a suitable format for transmission.
   3.3. Send the frame to the server using the established socket connection.

Server-side pseudocode

1. Initialize the socket and listen for incoming connections.
2. Accept client connections and start a loop to receive video frames:
   2.1. Receive a frame from the client over the socket connection.
   2.2. Process the received frame for object detection using the YOLOv8 model.
   2.3. Annotate the detected objects on the frame.
   2.4. Display the annotated frame on the server's screen.

3. Continue receiving and processing frames until the connection is closed.

SNAPSHOTS

Client :

client2.py > ...

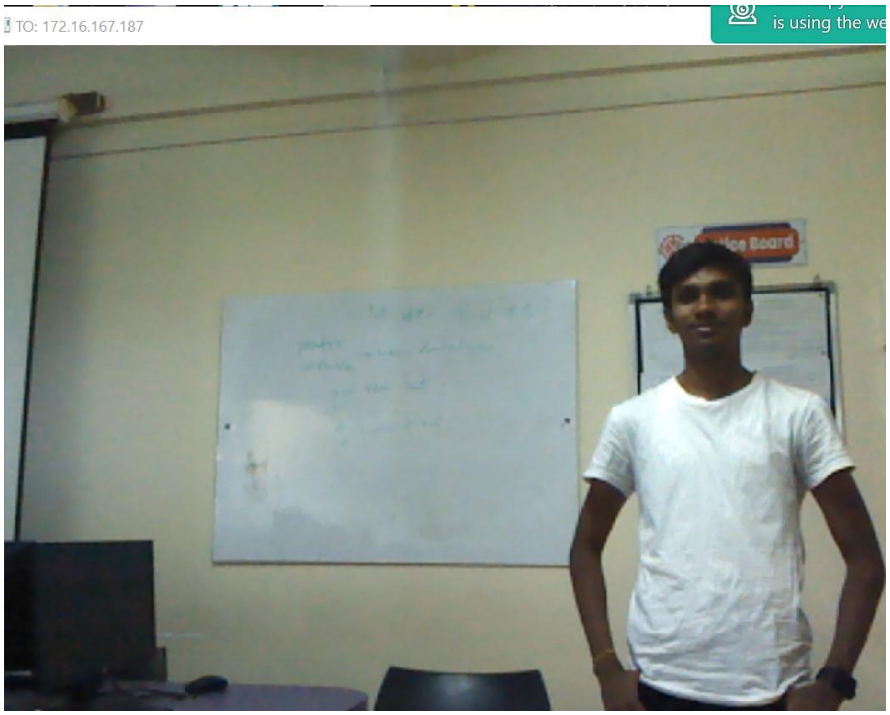
```python
import imutils # pip install imutils
camera = True
if camera == True:
    vid = cv2.VideoCapture(0)
else:
    vid = cv2.VideoCapture('videos/mario.mp4')
client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host_ip = '172.16.150.58' # Here according to your server ip write the address
#host=socket.gethostname()
#host_ip=socket.gethostbyname(host)
port = 9999
client_socket.connect((host_ip,port))

if client_socket:
    while (vid.isOpened()):
        try:
            img, frame = vid.read()
            #frame = imutils.resize(frame,width=380)

            frame = imutils.resize(frame,width=380)
            a = pickle.dumps(frame)
            message = struct.pack("Q",len(a))+a
            client_socket.sendall(message)
            cv2.imshow(f"TO: {host_ip}",frame)
            key = cv2.waitKey(1) & 0xFF
            if key == ord("q"):
                client_socket.close()
        except:
            print('VIDEO FINISHED!')
            break

```

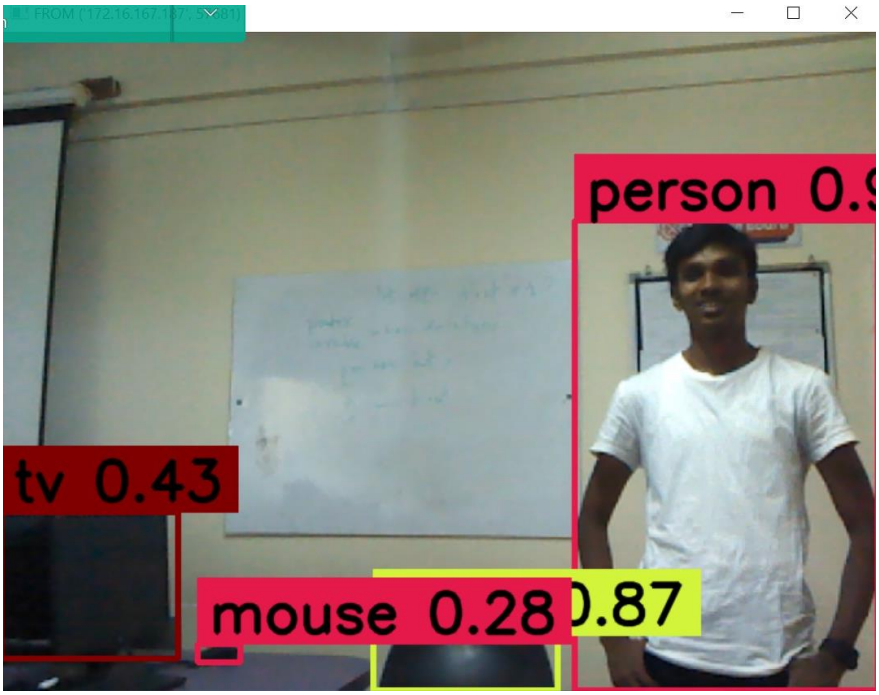
Server :

server2.py > show_client

```python
#-----------------------------------------------------------------------------------------------------
model = load_model()
class_names_dict = model.model.names
box_annotator = BoxAnnotator(color=ColorPalette(), thickness=3, text_thickness=3, text_scale=1.5)
font_scale=3
font=cv2.FONT_HERSHEY_PLAIN

server_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host_name  = socket.gethostname()
host_ip = socket.gethostbyname(host_name)
print('HOST IP:',host_ip)
port = 9999
socket_address = (host_ip,port)
server_socket.bind(socket_address)
server_socket.listen()
print("Listening at",socket_address)

def show_client(addr,client_socket):
    try:
        print('CLIENT {} CONNECTED!'.format(addr))
        if client_socket: # if a client socket exists
            data = b""
            payload_size = struct.calcsize("Q")
            while True:
                while len(data) < payload_size:
                    packet = client_socket.recv(4*1024) # 4K
                    if not packet: break
                    data+=packet
                packed_msg_size = data[:payload_size]
                data = data[payload_size:]
                msg_size = struct.unpack("Q",packed_msg_size)[0]

                while len(data) < msg_size:
                    data += client_socket.recv(6*1024)
                frame_data = data[:msg_size]
                data  = data[msg_size:]
                frame = pickle.loads(frame_data)

                results = predict(model, frame)
                frame=plot_bboxes(results, frame, class_names_dict, box_annotator)

                #text  =  f"CLIENT: {addr}"
                #frame =  ps.putBText(frame,text,10,10,vspace=10,hspace=1,font_scale=0.7, background_RGB=(255,0,0),text_RGB=(255,250,250))
                cv2.imshow(f"FROM {addr}",frame)
                key = cv2.waitKey(1) & 0xFF
                if key  == ord('q'):
                    break
            client_socket.close()
    except Exception as e:
        print(f"CLINET {addr} DISCONNECTED")
        pass

while True:
    client_socket,addr = server_socket.accept()
    thread = threading.Thread(target=show_client, args=(addr,client_socket))
    thread.start()
    print("TOTAL CLIENTS ",threading.activeCount() - 1)
```

Output:





SOFTWARE TESTING STRATEGIES

- Unit Testing: Focus on thoroughly testing critical functions, algorithms, and classes in isolation to ensure their correctness and reliability. Prioritize testing the object detection algorithm, socket communication, and other key components.
- Integration Testing: Verify the proper integration and interaction between the client and server components. Test the video streaming functionality, including capturing, transmitting, and receiving frames, to ensure seamless communication.
- Functional Testing: Conduct comprehensive tests to verify that the system meets the specified functional requirements. Test scenarios such as starting and stopping the video streaming, verifying accurate object detection, and validating the display of annotated frames.
- Performance Testing: Assess the system's performance under varying load conditions to ensure real-time requirements are met. Test the system's response time, video streaming efficiency, and object detection speed to identify and address performance bottlenecks.
- Usability Testing: Involve users or testers to evaluate the user interface and overall usability of the system. Gather feedback on the intuitiveness of video streaming controls, clarity of object annotations, and overall user experience to enhance usability.

FUTURE PLAN AND ACTION WORK

- Performance Optimization: Continuously optimize the system to improve its efficiency, reduce latency, and enhance video streaming performance. Explore techniques such as parallel processing, hardware acceleration, and network optimization to achieve faster and smoother real-time object detection and video streaming.

- Integration with Advanced Object Detection Models: Stay updated with the latest advancements in object detection algorithms and integrate more advanced models alongside YOLOv8. Experiment with models like EfficientDet, Cascade R-CNN, or Mask R-CNN to improve detection accuracy and handle complex scenarios.

- Real-time Analytics and Insights: Extend the system's capabilities to provide real-time analytics and insights based on the detected objects. Implement features such as object tracking, behavior analysis, or counting statistics to derive meaningful insights from the video stream, enabling more sophisticated surveillance and video analytics applications.

- Cloud-based Deployment and Scalability: Explore deploying the system in a cloud environment, leveraging services like AWS, Azure, or Google Cloud. This allows for scalability, high availability, and easy management of resources. Utilize cloud-based features such as auto-scaling to handle varying workloads efficiently.

- Mobile Application Integration: Develop a mobile application that can connect to the Real Time Object Detection System, enabling users to monitor live video streams, receive real-time alerts, and access system controls from their smartphones. This expands the accessibility and usability of the system, empowering users to stay connected and informed on the go.

CONCLUSION

In conclusion, the Real Time Object Detection System project aims to provide a robust and efficient solution for detecting objects in real-time video streams. By leveraging socket programming and the YOLOv8 model, the system enables clients to transmit their video frames to a server, where object detection is performed. The system offers the potential for a wide range of applications, including surveillance, video analytics, and real-time monitoring. With continuous improvement, optimization, and future enhancements, the Real Time Object Detection System can contribute to advancements in object detection technology, providing valuable insights and enhancing situational awareness in various domains.

REFERENCES

- www.google.com
- https://github.com/ultralytics/ultralytics
- https://youtu.be/7-O7yeO3hNQ