AR TUTOR

A PROJECT REPORT

Submitted by

AADITYA PRABU (2020115001)

VENU ARVIND A (2020115101)

In partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY

CHENNAI 600 025

MAY 2023

ABSTRACT

This project aims to develop a Unity AR application that leverages MediaPipe as the body tracking package in Python. The application incorporates buffer queues and multithreading to enhance the efficiency and real-time interaction of the system. The body tracking process begins with MediaPipe, which provides models for accurate body pose estimation. The tracked data undergoes processing and analysis in Python, and is broken into chunks and broadcasted with delimiters. On the receiving side, a UDP packet receiver thread, written in C# , receives the key point landmarks of the tracked human and stores the chunks in a buffer queue and assembles them and converts the assembled chunks back into keypoint-landmarks in a separate thread and forwards it to the keypoint mapper function which takes care of mapping the coordinates with 3d Character. The Unity AR Foundation Package coupled with AR Core library takes care of augmenting the character in the real environment where learners can observe and interact with a humanoid avatar representing the performers's movements. The integration of buffer queues and multithreading improves synchronization, data flow, and performance, creating an immersive and interactive learning experience. The proposed system addresses the limitations of traditional teaching methods by providing an engaging and realistic virtual learning environment. By utilizing MediaPipe for body tracking, Python for data processing, sockets for real-time communication, and Unity for AR application development, this project presents a comprehensive solution that enables learners to visualize and mimic the performer's movements in a dynamic and interactive manner.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In recent years, Augmented Reality (AR) has emerged as a powerful tool for enhancing educational experiences. The ability to create immersive virtual environments opens up new possibilities for interactive and engaging learning. In this context, we present a project that combines body tracking, Berkeley-sockets and Unity AR to create an innovative educational application.

The primary goal of this project is to develop a Unity AR application that leverages body tracking technology to simulate a performer's movements in a virtual environment. By utilizing MediaPipe as the body tracking package in Python, the application captures the user's body movements and sends the keypoint data through sockets to multiple users. The keypoint data is then used to construct a humanoid-shaped object that mimics the performer's movements which is then augmented to provide visual aids and explanations from the performer. Learners, using their smartphones, can immerse themselves in this virtual experience, observe the performer 's movements, and interact with the AR elements to deepen their understanding of the subject matter. Through this project, we aim to revolutionize traditional teaching methods by combining the power of body tracking, sockets, and AR technology to create an interactive and immersive learning experience. This innovative approach has the potential to enhance engagement, retention, and comprehension of complex concepts, making education more accessible and captivating for learners.

1.1 OBJECTIVE

The Objectives of this project are:

- Implement Plane tracking, to augment the avatar on horizontal surfaces.
- Implement buffer queues and multithreading to optimize data flow and ensure real-time interaction, enabling smooth synchronization between the body tracking data processing component and the Unity AR application.
- Enable real-time feedback and interaction by transmitting the processed body tracking data from the Python component to the Unity AR application through a berkeley-socket server, allowing learners to receive immediate responses to performer's movements and fostering a dynamic and interactive learning environment.

1.2 MOTIVATION

The motivation behind this project stems from the growing need for innovative educational tools that can captivate and engage learners in a digital age. Traditional teaching methods often struggle to convey complex concepts effectively, leading to reduced interest and limited retention. By leveraging the advancements in body tracking, berkeley-sockets, and Augmented Reality (AR), we aim to provide an immersive and interactive learning experience. This project seeks to empower educators by allowing them to virtually demonstrate movements and actions, creating a dynamic and engaging classroom environment. By combining body tracking technology with AR elements, we can enhance the visualization and understanding of subject matter, catering to diverse learning styles and fostering a deeper connection between performers and learners. The project's ultimate goal is to transform education by leveraging

cutting-edge technologies to create a captivating and effective learning experience.

1.3 PROBLEM STATEMENT

Traditional teaching methods often struggle to engage learners and effectively convey complex concepts. This project addresses the need for innovative educational tools that can enhance understanding and captivate students. The challenge lies in creating a virtual learning environment that accurately tracks and replicates a performer's movements, allowing learners to observe and interact with a realistic avatar. Additionally, seamlessly transmitting the keypoint data from the body tracking package through berkeley-sockets to multiple users poses technical complexities. The project aims to solve these challenges by integrating body tracking, sockets, and Unity AR to develop an application that simulates a performer's movements in a virtual environment. The goal is to create an immersive and interactive learning experience that fosters deeper engagement and comprehension.

1.4 OVERVIEW

The application will enable learners to observe the movements of a performer represented by a virtual avatar. The project incorporates buffer queues and multithreading to optimize data flow and ensure real-time interaction. Using MediaPipe, the application will accurately track the performer's body movements in real-time, capturing even subtle gestures and postures. The tracked data will undergo processing and analysis in Python, where it will be enqueued in a buffer queue. Multithreading will enable parallel execution and efficient data transfer.

The socket server, implemented in Python, will facilitate real-time communication between the body tracking data processing component and the

Unity AR application. The processed data will be transmitted to Unity Client, where a separate thread will consume it for real-time Augmenting and interaction. Overall, this project aims to revolutionize the traditional teaching methods by leveraging Body Tracking, AR technology, Networking to create an engaging and immersive learning platform.

1.5 LIBRARIES USED

- **MediaPipe:** MediaPipe is a Framework for building machine learning pipelines for processing time-series data like video, audio, etc. Implementing MediaPipe in projects nullifies most of the hassles we usually face while working on an ML project.

- **OpenCV:** OpenCV is an open-source software library for computer vision and machine learning. The OpenCV full form is Open Source Computer Vision Library. It was created to provide a shared infrastructure for applications for computer vision and to speed up the use of machine perception in consumer products.

- **Sockets:** Sockets is a library for building berkeley-socket servers and clients in Python with a focus on correctness, simplicity, robustness, and performance.

CHAPTER 2

LITERATURE SURVEY

This chapter deals with the existing work carried out using the Body tracking AI , XR in various domains. It gives an overview about challenges in developing the AR model in the following manner.

2.1 RESEARCH PUBLICATION

2.1.1 Pauzi, A. S. B., Mohd Nazri, F. B., Sani, S., Bataineh, A. M., Hisyam, M. N., Jaafar, M. H., ... & Mohamed, A. S. A. (2021).

The paper describes a system to track the body movement of a person from a video source while augmenting the labeled skeleton joints onto the body of the person. This work has endless applications in the real world especially in the physical-demanding working environment as well as in the sports industry. By implementing deep learning, the techniques can recognize the joints on a person’s body.

2.1.2 Patil, A., Rao, D., Utturwar, K., Shelke, T., & Sarda, E. (2022). Body Posture Detection and Motion Tracking using AI for Medical Exercises and Recommendation System.

A software-based motion tracker can keep track of all the exercises you've done and provide you feedback on your posture while you're working out. Through computing data and analysis, the exercise's beneficial efficiency will be increased. The MediaPipe framework could be utilized for this application; in this machine learning model, points are plotted at several joints of the human body posture, and movement is tracked, stored, and analyzed.

2.1.3 Yang, J., Chen, T., Qin, F., Lam, M. S., & Landay, J. A. (2022, April). HybridTrak: Adding Full-Body Tracking to AR Using an Off-the-Shelf Webcam.

Full-body tracking in virtual reality improves presence, allows interaction via body postures, and facilitates better social expression among users. However, full-body tracking systems today require a complex setup fixed to the environment (e.g., multiple lighthouses/cameras) and a laborious calibration process, which goes against the desire to make AR systems more portable and integrated.

2.1.4 Bradski, G. (2000). The openCV library. Dr. Dobb's Journal: Software Tools for the Professional Programmer, 25(11), 120-123.

Presents information on a study which focused on OpenCV, an open-source, computer-vision library for extracting and processing meaningful data from images. Overview of OpenCV; Example of gesture recognition for conducting music using the library; Routines and content of OpenCV; Conclusions.

2.2 CONCLUSION FROM LITERATURE SURVEY

The literature survey reveals a growing interest in the integration of body tracking, berkeley-sockets, and Augmented reality (AR) technologies in educational applications. The studies demonstrate the potential of these technologies to create immersive and interactive learning environments that enhance student engagement, understanding, and retention. Body tracking technology enables the replication of realistic movements and interactions, while berkeley-sockets facilitate real-time data transmission for collaborative

experiences. Augmented reality (AR) elements further enhance the visualization and explanation of educational content.

The findings indicate that the integration of these technologies holds promise for revolutionizing traditional teaching methods, providing educators with innovative tools to create captivating and effective learning experiences for students. Further research and development in this area can contribute to the advancement of educational technology and pedagogy.

CHAPTER 3

DESIGN OF THE APPLICATION

This chapter deals with the overall architecture of the application, the sender applications design and the receiver applications design.

3.1 ARCHITECTURE

The major components in the application are:

1. Body Tracking Module
2. Networking Module
3. Unity Application

1.Body Tracking Module:

Data Capture: This module captures the user's body movements using a single RGB camera, It tracks the positions and orientations of key body joints or keypoints, such as head, hands, and limbs, to accurately represent the user's movements.

Data Processing: The data processing module analyzes the captured body tracking data. It applies algorithms and techniques to extract relevant information, such as keypoint coordinates, representing the user's body pose and movements.

2.Networking Module:

Socket Server(Python):

Data Transmission: The processed data is broken into chunks because of the maximum transmission unit barrier and is broadcasted along with delimiters to differentiate between packets using a UDP Server for further distribution.

Socket Client(C#):

Data Reception: The data reception module receives the chunk transmitted from the UDP server. It decodes and parses the received data, extracting the relevant information such as keypoint coordinates. The module then enqueues the chunks into a buffer queue for further processing.

3.Unity Application:

In this project, Unity 3D Engine is utilized as the core development platform for creating the AR application. It provides a robust framework for designing and rendering the graphics, handling user interactions, and integrating various components such as Augmented Reality / Virtual Reality. With Unity's extensive library of assets, scripting capabilities, and cross-platform support, it enables the seamless integration of 3D models, animations, and interactive elements. The Unity AR application is developed, allowing for efficient communication with the UDP server to receive real-time body tracking data. Unity's flexibility and versatility empower the project to deliver an immersive and interactive learning experience, leveraging the capabilities of AR technology to enhance the performer's explanations.

The three major components involved in the client side include:

- Assembling
- Mapping
- Augmenting

Assembling:

This component assembles the chunks from the buffer queue. It parses the queue and based on the delimiters it takes different actions and finally converts the assembled chunks into keypoint-landmarks and stores them into a landmark queue.

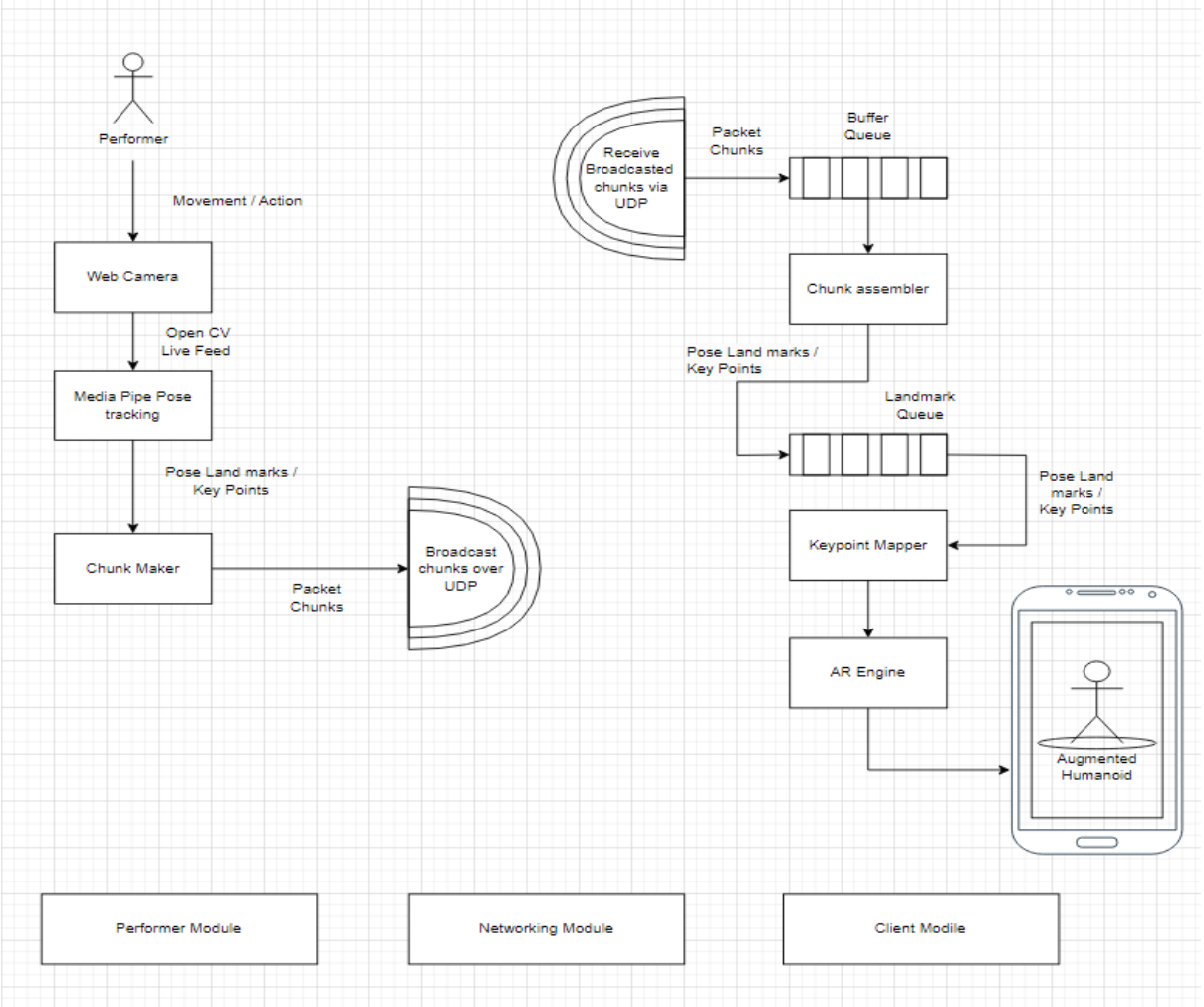
Mapping:

This component receives its input from the landmark queue, and maps the keypoint coordinates with each joint of the 3D avatar enabling the avatar to mimic the performer's movements.

Augmenting:

The mapped avatar is finally augmented on to the real world using the AR Foundation and AR core package. The tracked plane is used as a platform for the avatar to be spawned

3.2 ARCHITECTURE DIAGRAM



CHAPTER 4

RESULT AND DISCUSSION

This chapter shows the observation and result obtained from the Application that

4.1 PROBLEMS FACED:

1. **Technical Compatibility:** Ensuring compatibility between different hardware and software components, such as the AR Core support on Android Mobile, RGB Camera and the Unity 3D Engine. Compatibility issues arise due to different communication protocols, hardware specifications and software dependencies.

2. **Data Accuracy and Precision:** Achieving accurate and precise body tracking data is crucial for the project's success. However, limitations of the RGB cameras or environmental factors (e.g., lighting conditions) impact the quality of the captured data, leading to inaccuracies and jittery movements.

3. **Latency and Synchronization:** Real-time synchronization of the performer's movements with the humanoid avatar and real time Augmentation requires low latency. Network bottlenecks (MTU), delays and processing bottlenecks introduce latency, resulting in a noticeable delay between the performer's movements and their representation in the virtual environment.

4.2 CODE SCREENSHOTS

Client:

```
private Vector3 _current;
1 reference
private void KeypointMapper(Dictionary<int, Dictionary<string, double>> landmarkMap) {
    try {
        foreach (int keypoint in landmarkMap.Keys) {
            _current = _bodyKeypointTransformArray[keypoint].localPosition;
            float x = (float)landmarkMap[keypoint]["x"];
            float y = (float)landmarkMap[keypoint]["y"];
            float z = (float)landmarkMap[keypoint]["z"];
            _destination.Set(x, y, z);
            _destination *= ScaleFactor;

            Vector3 resultant = _destination - _current;
            _bodyKeypointTransformArray[keypoint].Translate(TranslateSpeed * Time.deltaTime * resultant);



        }
    } catch (Exception error) {

        Debug.Log(error);
    }

}
```

```
private void ChunkAssembler() {

    string landmarkString = "";
    while (_isApplicationOn) {
        while (_bufferQueue.Count != 0) {
            if (_landmarkQueue.Count < _landmarkQueueThreshold) {
                string chunk = _bufferQueue.Dequeue();
                if (chunk.Equals("START")) {
                    landmarkString = "";
                } else if (chunk.Equals("END")) {
                    Debug.Log(landmarkString);
                    try {

                        Dictionary<int, Dictionary<string, double>> landmarkMap
                            = JsonConvert.DeserializeObject
                                <
                                  Dictionary<int, Dictionary<string, double>>
                                >(landmarkString);

                        if (landmarkMap != null) {
                            _landmarkQueue.Enqueue(landmarkMap);
                        }
                        _logMessage2 = "";
                    } catch (Exception error) {
                        landmarkString = "";
                        _logMessage2 = error.Message.ToString();
                    }
                } else {
                    landmarkString += chunk;
                }
            }
        }
    }


}
```

```
private void ChunkReceiver() {

    byte[] chunkByteArray = new byte[1024 * 5];
    while (_isApplicationOn) {
        try {
            if (_bufferQueue.Count < _bufferQueueThreshold) {
                Array.Clear(chunkByteArray, 0, chunkByteArray.Length);
                int? bytesRead = _clientSocket?.ReceiveFrom(chunkByteArray, ref _socketEndPoint);
                if(bytesRead != 0) {
                    string chunkByteString = Encoding.UTF8.GetString(chunkByteArray, 0, (int)bytesRead);
                    _bufferQueue.Enqueue(chunkByteString);
                }
            }
        } catch (Exception error) {
            _logMessage2 = error.Message.ToString();

        }

    }

}
```

```
    }
    2 references
    private void FingerDown(EnhancedTouch.Finger finger) {
        if (finger == null || finger.index != 0) return;
        if (aRRaycastManager.Raycast(finger.currentTouch.screenPosition,hits,TrackableType.PlaneWithinPolygon)) {
            foreach(ARRaycastHit hit in hits) {
                Pose pose = hit.pose;
                Debug.Log("Parent pose: " + pose.position.ToString());
                Debug.Log("Child pose: " + BodyContainer.transform.GetChild(0).position.ToString());
                BodyContainer.transform.SetPositionAndRotation(pose.position, pose.rotation);
                BodyContainer.SetActive(true);

            }
        }
    }
}
```

Performer:

```python
class App:
    def __init__(self, port=8080, ip_address='<broadcast>'):
        self.video_capture = None
        self.mp_drawing = None
        self.mp_pose = None
        self.server_socket = None
        self.server_port = port
        self.server_ip_address = ip_address

    def setup_server(self):
        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
        self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        self.server_socket.settimeout(0.2)

    def setup_pose_estimator(self):
        self.video_capture = cv2.VideoCapture(0)
        self.mp_drawing = mp.solutions.drawing_utils
        self.mp_pose = mp.solutions.pose
```

```python
def capture_and_broadcast_pose(self):
    with self.mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
        while self.video_capture.isOpened():
            ret, frame = self.video_capture.read()

            # Recolor image to RGB
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            image.flags.writeable = False

            # Make detection
            results = pose.process(image)
            # Recolor back to BGR
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            # Render detections

            self.mp_drawing.draw_landmarks(image, results.pose_landmarks, self.mp_pose.POSE_CONNECTIONS)

            cv2.imshow('Camera Feed', image)

            try:
                normalized_pose_landmarks = results.pose_landmarks.landmark
                keypoints = dict()
                for pose_landmark in self.mp_pose.PoseLandmark:
                    extracted_pose_landmark = normalized_pose_landmarks[pose_landmark.value]
                    unity_x = (extracted_pose_landmark.x - 0.5) * 2
                    unity_y = (extracted_pose_landmark.y - 0.5) * -2
```

```
                unity_z = extracted_pose_landmark.z

                keypoints[pose_landmark.value] = {
                    'x': unity_x,
                    'y': unity_y,
                    'z': unity_z
                }

            print(keypoints)

            keypoints = json.dumps(keypoints)
            max_payload_size = len(keypoints) // 2

            self.server_socket.sendto("START".encode(), (self.server_ip_address, self.server_port))

            self.server_socket.sendto(keypoints[:max_payload_size].encode(),
                                      (self.server_ip_address, self.server_port))
            self.server_socket.sendto(keypoints[max_payload_size:].encode(),
                                      (self.server_ip_address, self.server_port))
            self.server_socket.sendto("END".encode(), (self.server_ip_address, self.server_port))



        except:
            print('Error in decoding landmarks or broadcasting landmarks!')
            pass
```

CHAPTER 5

CONCLUSION AND FUTURE WORK

This chapter deals with the final conclusion and the future development of the AR Tutor application.

5.1 CONCLUSION

From the ARTutor application developed using Unity, it can be concluded that it:

1. Utilizes Unity 3D Engine, Body tracking, and Sockets to create an immersive AR learning experience.
2. Maps the performer's movements onto a virtual humanoid object, synchronizing real-time movements for accurate representation.
3. Augments the humanoid avatar, enhancing explanations and providing visual context.
4. Promotes active learning through user interaction within Augmented Reality .
5. Enhances performer-student interactions and improves the understanding of complex concepts through an engaging and interactive AR platform.

5.2 FUTURE ENHANCEMENT

Customization and Personalization: Introduce customization options for clients/learners to personalize their virtual avatars and environments. This could include customizable appearances, preferences, and settings, enhancing the sense of ownership and personalization within the AR learning experience.

Advanced Interactivity: Expand the range of interactive elements within the virtual environment, such as virtual objects, simulations, or gamified elements. This would create more interactive and engaging learning scenarios, encouraging active participation and exploration.

Enhanced Feedback Mechanisms: Incorporate real-time feedback mechanisms that provide personalized guidance and assessment to students based on their interactions and progress within the AR learning environment. This could include adaptive learning algorithms, performance tracking, and personalized recommendations for further study.

Integration with Learning Management Systems: Integrate the AR application with existing learning management systems (LMS) or educational platforms. This would allow for seamless integration of AR experiences into existing educational workflows, including tracking student progress, assigning AR-based assignments, and incorporating AR content into existing curriculum structures.