Mini-Project Report On

Quirky Hand Combat

A modified version of the traditional game Rock Paper Scissors Lizard Spock

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

By

Elizabath Gigi (U2003077)
Govind Prasad (U2003089)
Manu Jaison (U2003126)
Naina Fathima K (U2003140)

Under the guidance of
Dr. Jisha G

Department of Computer Science & Engineering
Rajagiri School of Engineering and Technology (Autonomous)
Rajagiri Valley, Kakkanad, Kochi, 682039

July 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
RAJAGIRI VALLEY, KAKKANAD, KOCHI, 682039

CERTIFICATE

This is to certify that the mini-project report entitled ***"Quirky Hand Combat (A modified version of the traditional game Rock Paper Scissors Lizard Spock)"*** *is a bonafide work done by* ***Ms.Elizabath Gigi(U2003077), Mr.Govind Prasad(U2003089), Mr.Manu Jaison(U2003126), Ms. Naina Fathima K(U2003140)****, submitted to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2022-2023.*

Dr. Preetha K. G.	**Mr. Uday Babu P.**	**Dr. Jisha G**
Head of Department	Mini-Project Coordinator	Mini-Project Guide
Dept. of CSE	Asst. Professor	Asst. Professor
RSET	Dept. of CSE	Dept. of CSE
	RSET	RSET

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude towards **Dr. P. S. Sreejith**, Principal of RSET, and **Dr. Preetha K. G.**, Head of Department of Computer Science and Engineering for providing us with the opportunity to undertake our mini-project, "Quirky Hand Combat".

We are highly indebted to our mini-project coordinators, **Mr. Uday Babu P**, Assistant Professor, Department of Computer Science and Engineering, **Ms. Tripti C**, Assistant Professor, Department of Computer Science and Engineering

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our mini-project guide **Dr. Jisha G**, for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Elizabath Gigi

Govind Prasad

Manu Jaison

Naina Fathima K

ABSTRACT

This project focuses on developing an immersive and interactive game called "Quirky Hand Combat" using hand gesture recognition technology. The primary objective is to provide an entertaining gaming experience that engages players in strategic decision-making. By making hand gestures corresponding to different moves, players can compete against each other in real-time or challenge a computer opponent. The game's rules and logic will be implemented to determine the winner based on the combinations of gestures, while a user interface will display the game state and provide visual feedback on the outcome.

To achieve this, the project will utilize OpenCV and Mediapipe for real-time hand detection and tracking. Hand gestures will be recognized using TensorFlow, which will be trained on a dataset of preprocessed and augmented hand gesture images. The game will offer seamless multiplayer functionality, enabling players to connect and compete with each other. Additionally, a challenging computer opponent will be developed for engaging single-player mode. Through this game, players can enhance their strategic thinking, decision-making, and pattern recognition skills while enjoying a social and interactive gaming experience.

Overall, the project aims to create a dynamic game that not only entertains players but also encourages social interaction and skill development. By combining hand gesture recognition technology with engaging gameplay, the "Rock Paper Scissors Lizard Spock" game provides an immersive and entertaining experience for players to enjoy.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Quirky Hand Combat is a variation of the classic game Rock Paper Scissors, popularized by the television show "The Big Bang Theory." The game was invented by Sam Kass and Karen Bryla as an expansion to the traditional three-option game, adding two additional options to increase the complexity and strategy involved.

The original Rock Paper Scissors game revolves around three hand signs representing rock, paper, and scissors, where rock beats scissors, scissors beats paper, and paper beats rock. Stone Paper Scissors Lizard Spock takes this concept further by introducing two new hand signs, lizard and Spock, creating a five-option game. The following is how the game operates:

- Rock crushes Scissors: The rock hand sign wins against the scissors hand sign.
- Scissors cuts Paper: The scissors hand sign wins against the paper hand sign.
- Paper covers Rock: The paper hand sign wins against the rock hand sign.
- Rock crushes Lizard: The rock hand sign wins against the lizard hand sign.
- Lizard poisons Spock: The lizard hand sign wins against the Spock hand sign.
- Spock smashes Scissors: The Spock hand sign wins against the scissors hand sign.
- Scissors decapitates Lizard: The scissors hand sign wins against the lizard hand sign.
- Lizard eats Paper: The lizard hand sign wins against the paper hand sign.
- Paper disproves Spock: The paper hand sign wins against the Spock hand sign.

- Spock vaporizes Rock: The Spock hand sign wins against the rock hand sign.

In Stone Paper Scissors Lizard Spock, each hand sign has strengths and weaknesses against the other hand signs, creating a more intricate and strategic game. The expanded options increase the possibilities for outcomes, making the game more challenging and engaging for players.

Stone Paper Scissors Lizard Spock is often played as a friendly competition or as a means of decision-making when a simple coin toss or other random selection methods are desired. It has gained popularity due to its inclusion in popular culture and its ability to add complexity to the original game while maintaining its simplicity.

1.2 Existing System

Verbal Announcements: Before revealing their hand gestures, players verbally announce their choice, saying "rock," "paper," "scissors," "lizard," or "Spock." Hand Gestures: Players use their hands to represent the five elements: rock, paper, scissors. Each element is associated with a specific gesture is used to play against computer. Visual Representations: In digital versions of the game, players select their choices by clicking on graphical representations of the elements on a screen. This method eliminates the need for physical hand gestures.

1.3 Problem Statement

To develop a software application that implements the game rock paper scissors lizard spock using hand gesture recognition technology.

1.4 Objectives

- Multiplayer gaming allows individuals from various locations to engage in gameplay against each other.
- The score is calculated and presented in the form of a Leaderboard chart.
- In addition to multiplayer mode, players have the option to enjoy the game in single-player mode by competing against the computer for recreational purposes

1.5 Scope

The scope of the online multiplayer game "Rock Paper Scissors Spock Lizard" encompasses various aspects that contribute to its engaging and dynamic gameplay experience.

Multiplayer Interactions: The game allows players from different locations to connect and compete against each other in real-time. This fosters social interaction, friendly competition, and the opportunity to test one's strategic thinking against opponents with different playing styles.

Game Mechanics: The game incorporates the classic "Rock Paper Scissors" framework and expands it with the addition of two extra options, "Spock" and "Lizard." This introduces a fresh twist to the traditional game, providing players with a broader range of choices and strategic possibilities.

Rules and Strategy: The game follows specific rules, where each gesture (rock, paper, scissors, Spock, or Lizard) has strengths and weaknesses against other gestures. Players must analyze their opponents' patterns and strategically choose their moves to gain an advantage and outsmart their opponents.

Skill Development: "Rock Paper Scissors Spock Lizard" offers players the opportunity to refine their decision-making skills, adaptability, and strategic thinking. With each match, players can learn from their victories and defeats, honing their abilities to anticipate and counter their opponents' moves effectively.

Online Leaderboards and Rankings: The game often incorporates a system to track and display players' scores, creating a competitive environment. Online leaderboards and rankings allow players to measure their progress and strive to climb the ladder, increasing the replay value and encouraging continuous engagement.

Accessibility and User-Friendliness: Online multiplayer games like "Rock Paper Scissors Spock Lizard" typically aim to be easily accessible across various platforms, offering a user-friendly interface that allows players to quickly connect, find opponents, and engage in matches without complications.

Overall, the scope of "Rock Paper Scissors Spock Lizard" as an online multiplayer game spans interactive gameplay, strategic decision-making, skill development, competitive rankings, and a user-friendly experience, providing players with an entertaining and engaging platform to test their wits and enjoy multiplayer gaming.

Chapter 2

Literature Review

2.1 Recognition of Hand Gestures Using Mediapipe Hands

The paper explains that MediaPipe is an open-source framework provided by Google for processing perceptual data from various modalities, such as video and audio. The framework includes solutions for tasks like posture estimation and face recognition. For their project, they utilize MediaPipe Hands for hand tracking, which uses regression analysis to calculate finger coordinates from the detected palm of the hand. This approach reduces computation and improves accuracy compared to other models like OpenPose.

The paper mentions that the hand tracking model detects initial hand locations using a single shot detector optimized for real-time mobile applications. To tackle the challenges of detecting hands with varying sizes and occlusions, they train a palm detector instead of a hand detector. The model focuses on estimating bounding boxes for palms and fists, which are simpler than detecting hands with articulated fingers. Additionally, they use an encoder-decoder feature extractor for better scene-context awareness, even for small objects like palms.

After palm detection, the subsequent hand landmark model performs precise localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression. This means they predict the coordinates directly. The model is designed to handle partially visible and occluded hands, making it robust in various real-world scenarios. They gathered ground truth data by manually annotating 30K real-world images with 21 3D coordinates. To cover different hand poses and provide additional supervision, they also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.

The paper explains the process of building a hand landmarks detection model using the MediaPipe library as the base library and the CV2 library for computer vision pre-

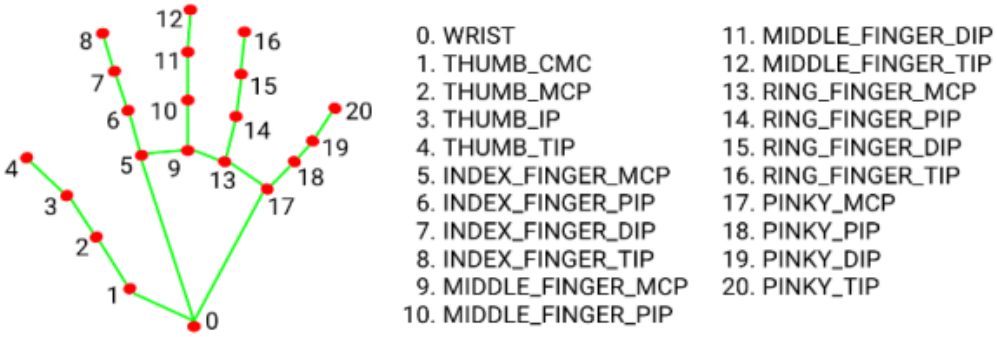

Figure 2.1: Hand Landmark in MediaPipe

processing. The MediaPipe library offers a wide range of human body detection and tracking models, trained on a diverse dataset by Google. These models use key points or landmarks to track different body parts, and all coordinate points are normalized to three dimensions (X, Y, Z). The library employs a graph-based dataflow structure with nodes and edges, enabling efficient information flow.

The hand tracking solution consists of two models: the Palm Detection Model and the Hand Landmark Model. The Palm Detection Model accurately detects the palm and provides a cropped palm image to the Hand Landmark Model. This approach reduces the use of data augmentation and simplifies the hand detection process. The Hand Landmark Model then precisely localizes 21 3D hand-knuckle coordinates inside the detected hand regions, even for partially visible or occluded hands.

To build the model, they used a dataset of various sign languages, where each image was passed through the detection model to collect the 21 landmark points, which were stored in a CSV file. Data cleaning and normalization were performed, removing null entries caused by blurry images, and normalizing the coordinates to fit the system. The dataset was then split into training and validation sets.

For predictive analysis of different sign languages, machine learning algorithms were used, and Support Vector Machine (SVM) outperformed other algorithms. SVM is effective in high-dimensional spaces and works well when there is a clear margin of separation between classes. Performance metrics such as accuracy, precision, recall, and F1 score were used to evaluate the model's performance.

The results were analyzed quantitatively using these metrics and the confusion matrix to understand the model's performance and types of errors made by the classifier. SVM

was found to be effective for classifying multiple classes of sign language alphabets and numeric gestures.

Overall, the paper presents a comprehensive approach to hand landmarks detection using MediaPipe and machine learning techniques, demonstrating its potential applications in various domains like virtual reality and real-time gaming experiences.

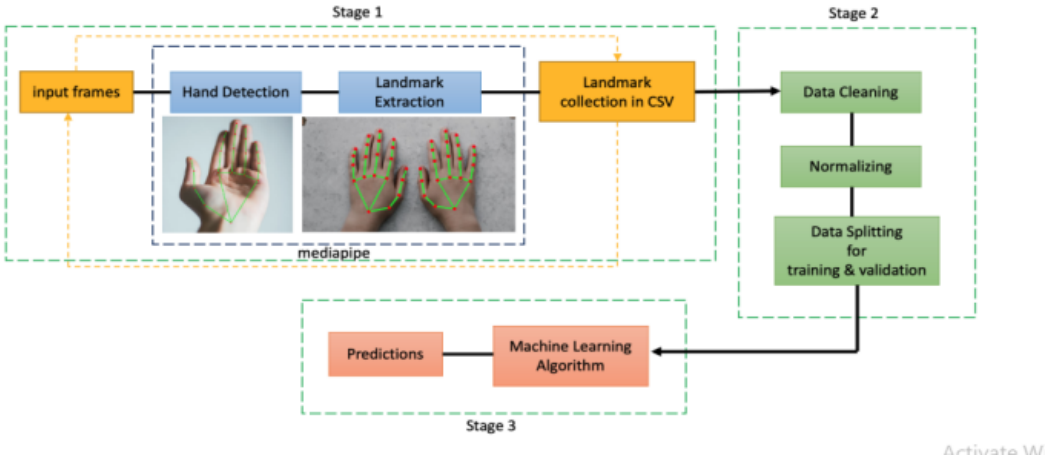

Figure 2.2: Proposed architecture to detect handgestures and predict sign languages finger spellings

2.2 Mediapipe and CNNs for Real-Time ASL Gesture Recognition

The research article emphasizes the significance of sign language as a crucial form of communication for the hearing and speech-impaired community. To promote accessibility and facilitate communication between sign language users and those without hearing loss, the authors stress the importance of developing automatic sign language recognition (SLR) systems. These systems analyze multimodal data, including hand and body motions, facial expressions, and overall movement, to create assistive technology for the deaf and hard of hearing.

The article discusses three main subproblems in SLR: static, isolated, and continuous recognition. Static recognition focuses on identifying finger-spelled gestures representing alphabets and digits. Isolated recognition involves recognizing dynamic gestures corresponding to individual words, while continuous recognition tackles the challenge of recognizing dynamic gesture sequences that encompass phrases combined with non-sign portions.

Various approaches exist for SLR, depending on the employed techniques. Two commonly used methods are glove-based and vision-based approaches. Vision-based approaches rely on image and signal processing techniques, analyzing pictures or videos as input to determine the relevant sign. Glove-based methods utilize data gloves to capture hand movements and positions. Each approach presents its own advantages and limitations, including computational costs, accuracy standards, and equipment requirements.

The authors review previous work in the field of SLR, highlighting studies that have used different techniques and datasets to recognize sign language gestures accurately. Some studies employed deep learning models like convolutional neural networks (CNNs), achieving high accuracy rates for both static and dynamic sign language recognition. Others focused on specific sign languages, such as Indian Sign Language (ISL), using supervised learning approaches and obtaining impressive accuracy levels.

The proposed architecture for continuous sign language recognition centers around a CNN and data augmentation approaches. The authors aim to create a more precise and effective SLR system by leveraging a large ASL dataset and state-of-the-art techniques. The architecture involves image frame acquisition, hand tracking using the Mediapipe module, feature extraction, and classification. The CNN is employed to analyze the features extracted from the sign language gestures and predict the corresponding sign language letter between 'A' and 'Z'.

The research findings demonstrate the effectiveness of the proposed SLR model. The authors used a substantial ASL dataset with 4500 photos per class for training, resulting in a model that achieved an accuracy score of 100percent on the test set. The classification report revealed 100percent precision, recall, and F1-score values for all classes, indicating that the model correctly identified each sign language gesture without making any errors.

The article concludes by underlining the importance of SLR in enhancing communication for the deaf community and promoting inclusivity. It calls for continued research and development to create more robust and efficient SLR systems that can handle various sign language motions and real-world variations. The authors envision the potential application of SLR technology in bidirectional communication applications, which would further improve accessibility and communication for sign language users, especially when

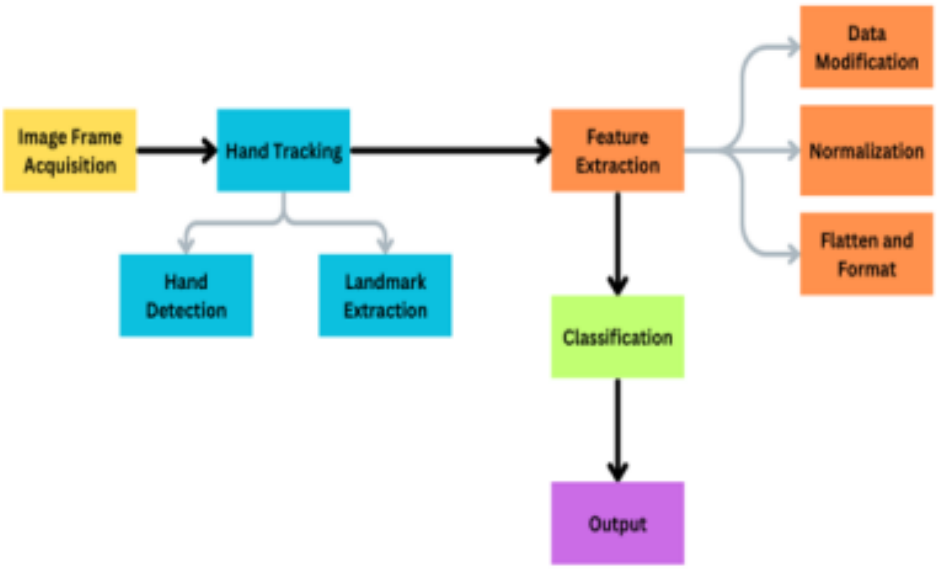

Figure 2.3: Workflow Method Research.

interacting with non-sign language users.

In summary, the research article provides a comprehensive overview of sign language recognition and its significance in enabling communication for the hearing and speech-impaired community. It highlights the evolution of SLR systems, various approaches used in previous studies, and the proposed architecture for continuous ASL recognition. The research demonstrates the high accuracy achieved by the CNN-based model and emphasizes the need for ongoing research to enhance accessibility and inclusivity for sign language users on a global scale.

Comparison

The [1] research article introduced a novel approach of gesture recognition using Mediapipe which is successfully implemented .In [2], the researchers seamlessly integrated the Mediapipe hand tracking module with CNNs, leveraging the strengths of both technologies. The CNN architecture effectively processed and analyzed the feature vectors extracted from the hand landmarks, allowing for accurate and rapid recognition of American Sign Language gestures. We modified [2] into game implementation of the traditional game Rock Paper Scisors into the modified version Rock Paper Scissos Spock Lizard named 'Quirky Hand Combat'. The addition of peer-to-peer communication using TCP tech-

nology extended the system's capabilities, enabling multiplayer gaming experiences and highlighting its potential for interactive and inclusive applications.

Chapter 3

System Analysis

3.1 Expected System Requirements

The system of user which is a laptop is expected to have the following features:

- A stable internet connection is necessary to access the online game and engage in multiplayer matches
- Depending on the platform hosting the game, you may need a compatible web browser
- The game should be compatible with your device's screen resolution.

3.2 Feasibility Analysis

3.2.1 Technical Feasibility

The project is technically feasible since majority of the population are in possession of smartphone or laptops. The web application only requires minimum requirements to run on a smartphone .

3.2.2 Operational Feasibility

It relies on reliable gesture recognition, stable real-time networking, platform compatibility, user-friendly interface, scalability, security, ongoing maintenance, and resource management.

3.2.3 Economic Feasibility

The development of the app is also zero budget as it was built using free resources.

3.3 Hardware Requirements

The following are the system requirements to develop the game.

- Processor: Intel Core i5
- Hard Disk: Minimum 100GB
- RAM: Minimum 8GB
- OS:Windows 10

3.4 Software Requirements

The following are the softwares used in the development of the game. Operating System:Windows or Linux

3.4.1 Visual Studio Code

Visual Studio is a powerful integrated development environment (IDE) used by developers to create, edit, and debug software applications. It supports various programming languages, including C, C++, Visual Basic, Python, and more. With its rich set of features, Visual Studio helps developers efficiently build applications for web, desktop, mobile, cloud, and gaming platforms. It also provides tools for version control, testing, and deployment, making it a comprehensive solution for software development projects.

3.4.2 Mediapipe 0.8.1

MediaPipe 0.8.1 is a software framework developed by Google that offers a comprehensive set of tools and pre-built models for building real-time multimedia applications. It facilitates tasks like hand, face, and pose recognition, as well as object tracking, body language analysis, and more, making it valuable for a wide range of computer vision and machine learning projects.

3.4.3 OpenCV 3.4.2

OpenCV 3.4.2 is a widely-used open-source computer vision library. It provides a collection of tools and functions for image and video processing, object detection, feature

extraction, and more. With its broad support for various programming languages like C++, Python, and Java, OpenCV is a valuable resource for developers working on computer vision applications and research.

3.4.4 Tensorflow 2.3.0

TensorFlow 2.3.0 is a popular open-source deep learning framework developed by Google. It enables developers and researchers to create and train machine learning models for various tasks, such as image and speech recognition, natural language processing, and more. TensorFlow 2.3.0 includes improvements in performance, ease of use, and integration with Keras, making it a versatile and powerful tool for building artificial intelligence applications.

3.4.5 tf-nightly 2.5.0.dev

tf-nightly 2.5.0.dev is the nightly development version of TensorFlow 2.5.0, containing the latest features and updates but may not be stable for production use. Use official releases for production work.

Chapter 4

Methodology

4.1 Proposed Method

- Hand Detection and Tracking: Utilize OpenCV and Mediapipe to detect and track the player's hand in real-time.
- Gesture Recognition: Employ TensorFlow to train a deep learning model on a dataset of hand gesture images. Preprocess and augment the dataset to improve model performance.Train the model to classify hand gestures corresponding to the moves: rock, paper, scissors, Spock, and lizard.
- Game Logic and Interaction: Implement the game's rules and logic, mapping the recognized gestures to the appropriate moves.Develop a user interface to display the game state, player's move.Determine the winner based on the rules of the "Rock Paper Scissors Spock Lizard" game.Enable player-player interaction through gestures and provide visual feedback on the game outcome.

4.1.1 Conceptualization

In the conceptualization phase, the game's concept, rules, and mechanics are defined. The goal is to enhance the traditional "Rock, Paper, Scissors" game by incorporating the extended elements of "Lizard" and "Spock." This involves determining how each gesture interacts with others, creating a more complex gameplay dynamic. For example, "Rock" crushes "Scissors," "Scissors" cuts "Paper," "Paper" covers "Rock," "Rock" crushes "Lizard," "Lizard" poisons "Spock," "Spock" smashes "Scissors," "Scissors" decapitates "Lizard," "Lizard" eats "Paper," and "Paper" disproves "Spock." This results in a total of ten possible interactions between gestures.Each interaction is generalised as "gesture 1" beats "gesture 2" in our game.

4.1.2 Technology Selection

In the technology selection phase, appropriate technologies for gesture recognition and real-time multiplayer networking are chosen. The gesture recognition system should be robust and accurate, able to interpret hand gestures from players in real-time. Various options include using computer vision libraries like OpenCV for image processing or machine learning models trained on hand gesture datasets. For real-time multiplayer networking, technologies like WebSockets or WebRTC are considered, allowing players to interact and compete with low latency and seamless communication.

4.1.3 Game Design

The game design phase focuses on creating an intuitive and enjoyable user interface (UI) and user experience (UX) for the online game. The UI should allow players to perform hand gestures easily, and the UX should provide clear feedback on the game's progress. Visual cues or animations may be incorporated to enhance the gesture recognition experience and engagement.

4.1.4 Gesture Recognition

In the gesture recognition phase, the chosen technology is implemented to interpret hand gestures from players in real-time. Video input from players' cameras is processed, and the gestures are mapped to the corresponding elements of the game (Rock, Paper, Scissors, Lizard, Spock). The system should accurately recognize the gestures, enabling smooth gameplay interactions.

4.1.5 Networking and Backend

Setting up a reliable server infrastructure is crucial for handling player connections, managing game sessions, and synchronizing game data between players in real-time. This infrastructure ensures seamless interactions between players during multiplayer matches, enhancing the overall gaming experience.

4.1.6 Multiplayer Functionality

The multiplayer functionality is developed to enable players to connect with each other, interact through hand gestures, and compete in real-time matches. Players can join game rooms or match lobbies, and the server facilitates the communication and synchronization of game data between players.

4.1.7 Gameplay Logic

The gameplay logic is implemented to determine the winner based on the hand gestures provided by the players. The rules for comparing gestures are programmed into the game's core logic, enabling the system to identify the winning gesture combination during a match.

4.1.8 Testing and Debugging

Extensive testing and debugging are conducted to ensure the game functions correctly under various scenarios. This includes testing different combinations of gestures, multiplayer interactions, and edge cases to identify and address potential issues.

4.1.9 Deployment

The online game is deployed to a server or cloud-based platform, making it accessible to players through web browsers, mobile apps, or other targeted platforms. This step involves setting up the hosting environment and configuring the necessary services for player access.

4.1.10 Monitoring and Maintenance

After deployment, the game's performance server load, and user feedback are continuously monitored. This data helps in identifying and resolving any reported issues and provides insights for ongoing maintenance and updates. Additionally, security and privacy considerations are prioritized to protect players' data and comply with legal and regulatory requirements.

Throughout the development process, gathering feedback from players during beta testing helps in fine-tuning the game's mechanics and improving the overall user experi-

ence. User feedback is valuable for addressing any potential shortcomings and enhancing the game's playability and enjoyment.

Chapter 5

System Design

5.1 Architecture Diagram

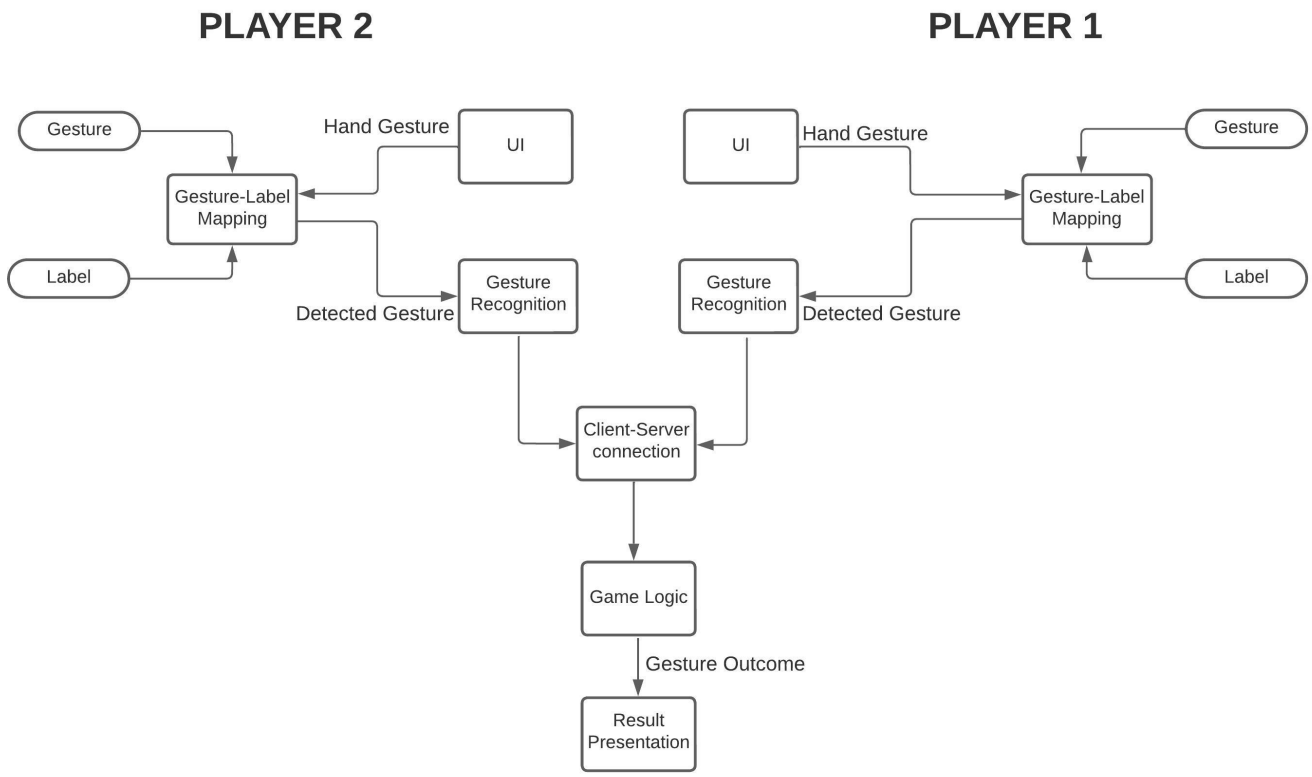

Figure 5.1: Architecture diagram

5.2 Usecase diagram

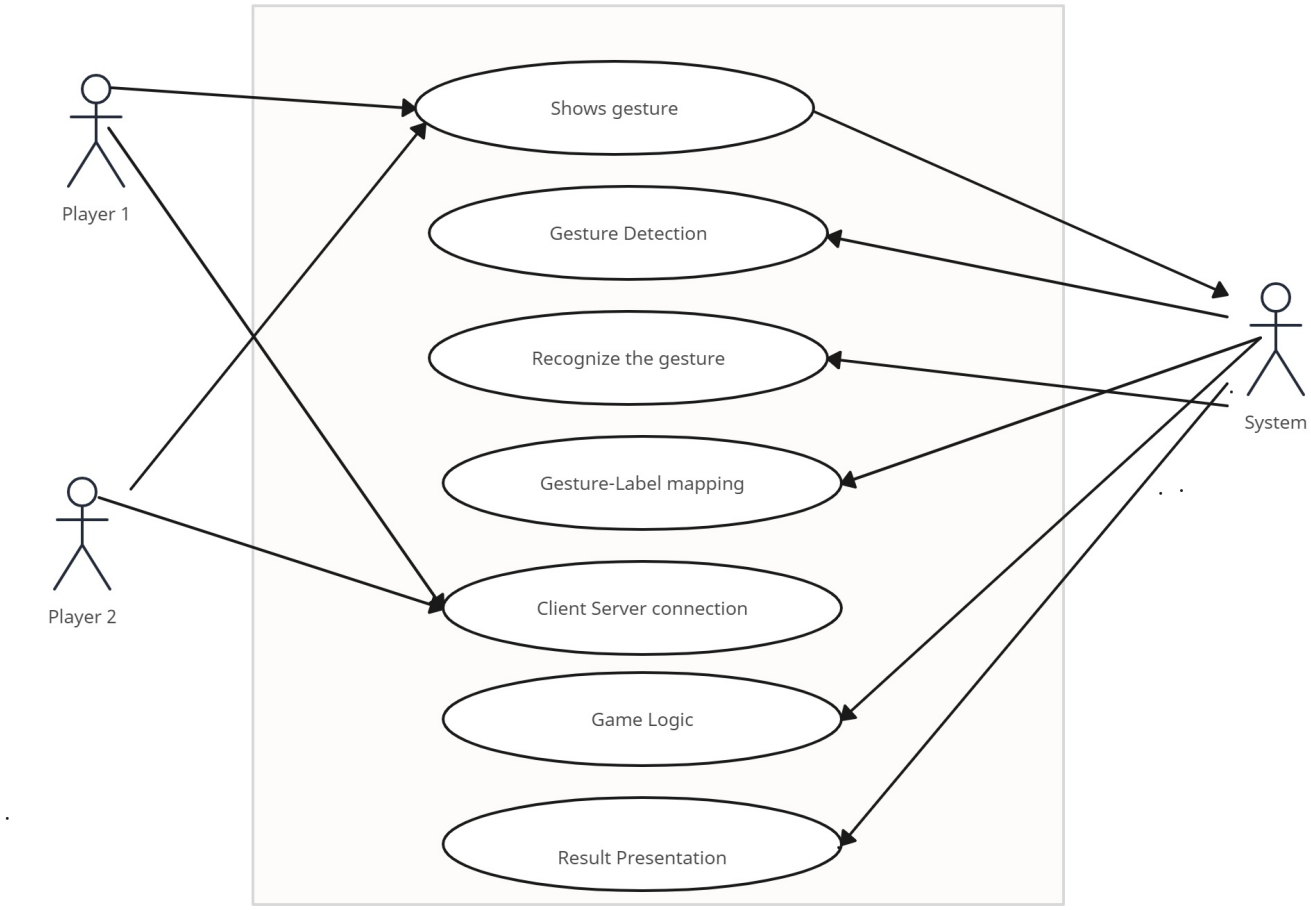

Figure 5.2: Usecase diagram

Chapter 6

System Implementation

6.1 Gesture Recognition

6.1.1 Data Loading and Preprocessing

The code loads gesture data from a CSV file and splits it into input features ($X\_dataset$) and corresponding labels ($y\_dataset$). The data is then split into training and testing sets using the $train\_test\_split$ function from $scikit-learn$.

6.1.2 Model Creation and Training

A sequential neural network model is created using tf.keras.layers to classify the gestures. The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss, and accuracy metrics. The model is trained on the training data using model.fit, with early stopping to prevent overfitting. The model weights are saved using ModelCheckpoint callbacks.

6.1.3 Model Evaluation

The trained model is evaluated on the test set to obtain the validation loss and accuracy.

6.1.4 Confusion Matrix and Classification Report

A confusion matrix is generated using $scikit-learn's\ confusion\_matrix$ function, and a classification report is printed to assess the model's performance.

6.1.5 Model Export and Conversion to TFLite

The trained model is saved as an HDF5 file. The model is then converted to a quantized TensorFlow Lite model (TFLite) using tf.lite.TFLiteConverter. The TFLite model is

saved to a file for deployment on resource-constrained devices.

6.1.6 Gesture Recognition with TFLite Model

The saved TFLite model is loaded using tf.lite.Interpreter. An input gesture sample from the test set is fed to the interpreter, and the gesture's class label is predicted. The result is printed as the output class label index and the corresponding gesture class. We are implementing a hand gesture recognition system using the Mediapipe library for hand tracking and a custom-trained classifier for recognizing gestures.

Import Libraries:

Import necessary libraries like copy, argparse, itertools, Counter, deque, cv2 (OpenCV), numpy, mediapipe, and user-defined modules utils and model.

Argument Parsing:

Define the $get\_args()$ function to parse $command-line$ arguments such as device, width, height, confidence thresholds, and static image mode.

Main Function:

- Parse command-line arguments using $get\_args()$.
- Initialize the camera and 'mediapipe.Hands' object.
- Load the $pre-trained$ KeyPointClassifier and PointHistoryClassifier models.
- Read the gesture labels from CSV files.

Camera and Gesture Processing Loop:

Enter a continuous loop to process camera frames and recognize hand gestures in $real-time$. Within the loop:

- Capture frames from the camera.
- Flip the image horizontally to mirror the display.
- Perform hand detection using 'mediapipe.Hands'.
- Calculate bounding rectangles around the detected hand(s).
- Extract landmark coordinates for each detected hand.

- $Pre-process$ the landmark data for classification.
- Perform hand sign classification using the KeyPointClassifier.
- Update the finger gesture history and calculate the most common finger gesture ID.
- Draw bounding rectangles and landmarks on the frame.
- Display information about the detected gestures, including hand sign and finger gesture.
- Continue the loop until the ESC key is pressed.

Pre-processing Functions:

Define functions to $pre-process$ the landmark and point history data before classification. Convert landmark coordinates to relative and normalized values for better classification results. Drawing Functions: Define functions to draw bounding rectangles, landmarks, and information text on the image frame. Display information about the detected gestures, hand sign, finger gesture, and mode on the frame. Select Mode Function: Define a function $select\_mode(key, mode)$ to interpret user input and update the current mode (logging key points or point history). Logging CSV Function: Define a function to write gesture data (landmark and point history) to CSV files based on the selected mode and number. Helper Functions: Define other helper functions for calculating bounding rectangles, converting coordinates, and displaying information. Main Function Call: Finally, call the main() function to start the gesture recognition system.

Chapter 7

Testing

7.1 Unit Testing

Tested individual components and logic within the game implementation to ensure proper functioning of the game mechanics. First, the game was tested by feeding text inputs (paper,scissors etc) on individual systems and then implemented in the other system.Provided test cases to validate different scenarios within the game, such as winning, losing, and ties.

Both the client and server codes were first run in the same system, ensured proper working, applied to the other system and ensured proper communication.Tested the communication and data exchange between the client and server components of the client-server architecture. Verified data synchronization, error handling, and responsiveness. Documented every connection issues and data inconsistencies and the steps taken to address and solve them.

After gesture recognition model is trained,its performance is evaluated on the test dataset using the evaluate method, obtaining the validation loss and accuracy. The code generates a confusion matrix and a classification report to assess the model's performance on the test data. The confusion matrix provides insights into the model's predictive performance for each class, while the classification report includes precision, recall, F1-score, and support metrics for each class.A confusion matrix, you compare the model's predictions against the ground truth labels of the dataset. Each row of the matrix represents the instances in an actual class,while each column represents the instances in a predicted class. Fig 7.1 shows that the mode correctly predicted 404 Stone(0th label) from 407 stone data.A classification report is a summary of various performance metrics for a classification model. It provides a detailed evaluation of the model's performance for each

class in the dataset.

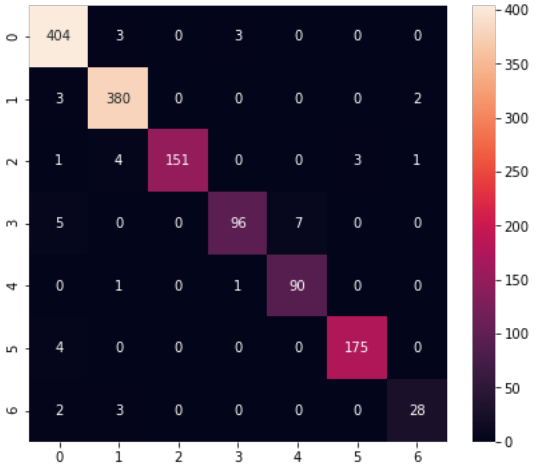

Figure 7.1: Confusion Matrix

Classification Report

	precision	recall	f1-score	support
0	0.96	0.99	0.97	410
1	0.97	0.99	0.98	385
2	1.00	0.94	0.97	160
3	0.96	0.89	0.92	108
4	0.93	0.98	0.95	92
5	0.98	0.98	0.98	179
6	0.90	0.85	0.88	33
accuracy			0.97	1367
macro avg	0.96	0.94	0.95	1367
weighted avg	0.97	0.97	0.97	1367

Figure 7.2: Classification Report

7.2 Integration Testing

Integrated the client-server connection and game implementation modules and tested their interactions. Tested various game scenarios, including multiplayer interactions, and ensured smooth data exchange. Then, integrated the gesture recognition module into the game and tested its interaction with the overall application. Evaluated how well the game responds to gestures recognized by the module. Ensured that the correct gestures trigger the appropriate game actions. .

7.3 System Testing

System testing evaluates the entire software application as a whole. Presented the comprehensive test cases that assess the game's functionality and user experience as a whole, considering the integrated components - gesture recognition, client-server, and game implementation. Included scenarios covering different gestures, game outcomes, client-server interactions, and edge cases. The system testing test cases was designed to cover all aspects of the game, ensuring it functions correctly from end to end. It considered the following scenarios: Basic Game Play: Tested the basic game rules, ensuring that the correct gestures from the players are recognized, and the game outcome is determined accurately. Winning and Losing Scenarios: Tested the different winning and losing combinations for Rock-Paper-Scissors-Lizard-Spock and verify that the game correctly identifies the winner. Tie Scenarios: Tested tie scenarios to ensure that the game correctly detects when both players have made the same gesture. Client-Server Interaction: Tested the communication between the client and server to ensure smooth data exchange and responsiveness. Error Handling: Verified how the system handles unexpected errors or edge cases, such as network disruptions or invalid inputs.

Chapter 8

Results

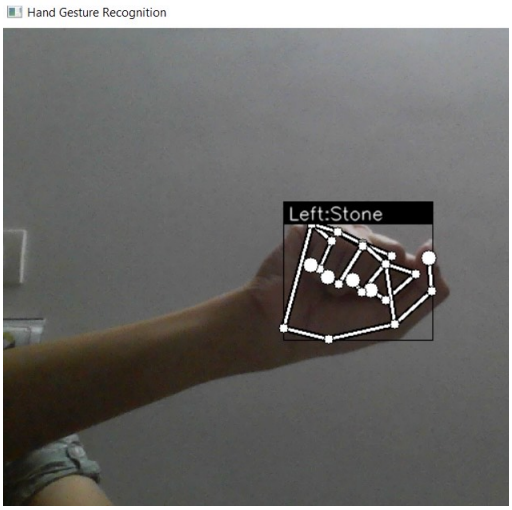

Figure 8.1: Stone

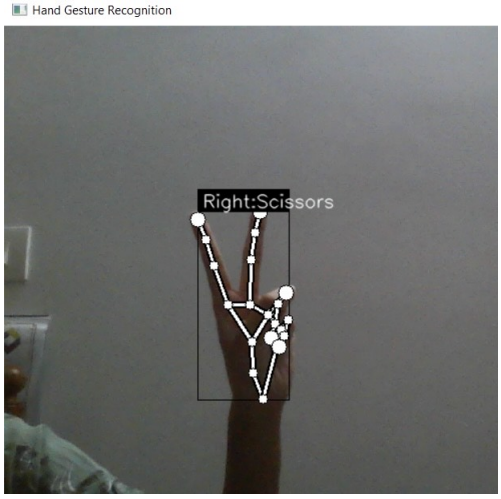

Figure 8.2: Scissors

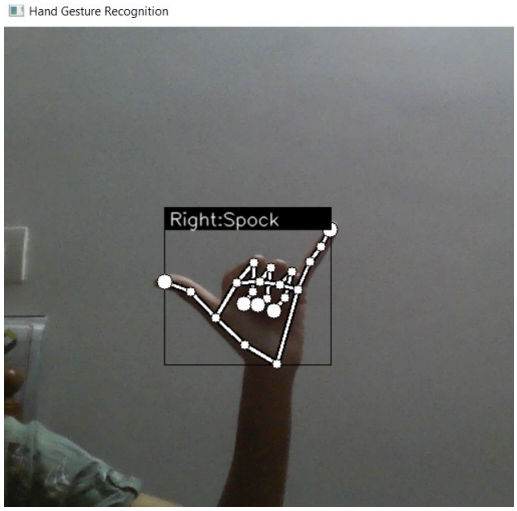

Figure 8.3: Spock

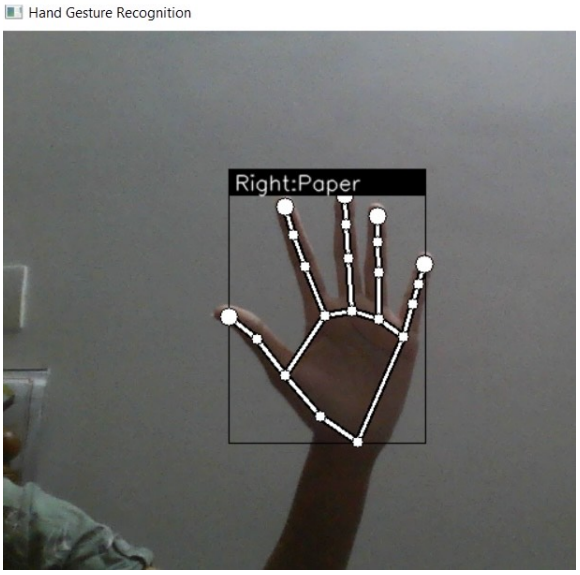

Figure 8.4: Paper

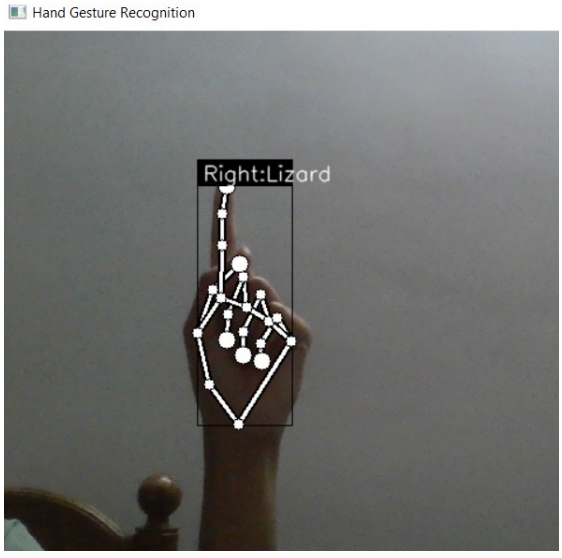

Figure 8.5: Lizard

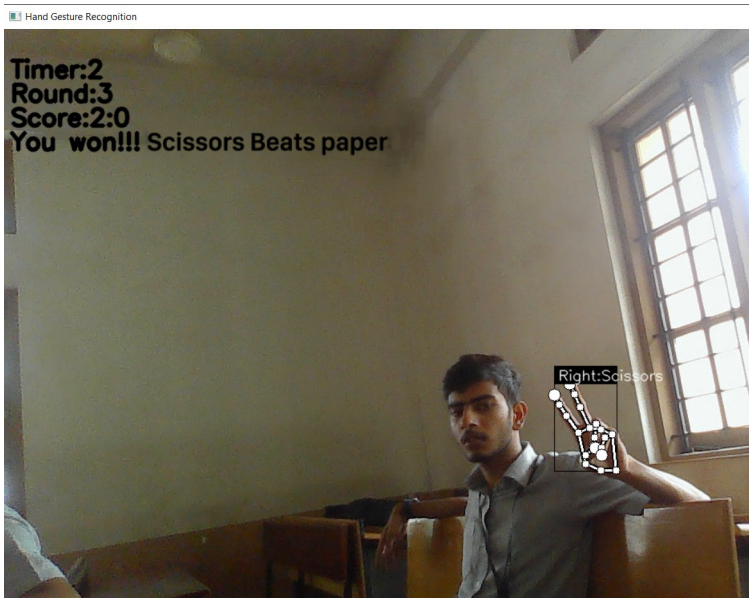

Figure 8.6

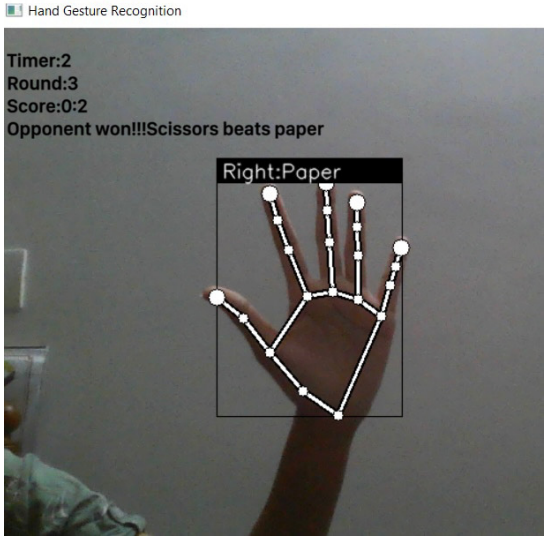

Figure 8.7:

Figure 8.6 and 8.7 shows two players playing Quirky Hand Combat simultaneously in two different systems.

Chapter 9

Risks and Challenges

1. Data Collection: Collecting a diverse and representative dataset of hand gestures is crucial for training the gesture recognition models. Ensuring that the dataset includes a wide range of hand shapes, sizes, and orientations, as well as different lighting conditions and backgrounds, is important to improve the model's accuracy and generalization.

2. Model Training:Training accurate and robust gesture recognition models requires careful selection of the architecture, hyperparameters, and optimization techniques. Proper model evaluation and validation are essential to avoid overfitting and underfitting. Fine-tuning and transfer learning might be necessary to adapt pre-trained models to the specific hand gestures of the Rock-Paper-Scissors-Lizard-Spock game.

3. Variability in Hand Gestures: People might perform hand gestures differently, leading to variability in the captured hand landmarks. Ensuring that the models can handle this variability and accurately recognize gestures from different users is a challenge.

4. Lighting and Background Noise: Hand gesture recognition heavily relies on the quality of the input images. Variations in lighting conditions and background noise can affect the model's performance. Pre-processing techniques such as image enhancement and background subtraction might be used to reduce the impact of these factors.

5. User Experience and Interaction: Providing clear instructions and visual feedback to users about their detected hand gestures can enhance the overall gaming experience. Ensuring that the interaction with the game feels responsive and smooth is also important for user satisfaction.

Chapter 10

Conclusion

Conclusion + Future scope

In this project, we created an interactive Rock-Paper-Scissors-Lizard-Spock game using hand gesture recognition. Players competed against each other with hand gestures detected using the MediaPipe library's pre-trained model. Python implemented the game logic with a client-server architecture for player communication.

The webcam captured hand gestures, which were classified into game choices using a keypoint classifier. Real-time feedback determined the winner based on the classic game rules. The project showcased the potential of computer vision and machine learning for interactive applications and demonstrated the versatility of MediaPipe for hand gesture recognition.

To improve the project, we can enhance hand gesture recognition accuracy through advanced machine learning techniques and refine the user interface for a more immersive experience. Overall, the Rock-Paper-Scissors-Lizard-Spock game successfully showcased hand gesture recognition and holds promise for future applications in human-computer interaction.

References

[1] Kavana KM, Suma NR.,Volume:04/Issue:06/June-2022 .Recognition of Hand Gestures using Mediapipe Hands.International Research Journal of Modernization in Engineering Technology and Science e-ISSN: 2582-5208

[2] Rupesh Kumar, Ashutosh Bajpai, Ayush Sinha, S.K Singh.,Mediapipe and CNNs for Real-Time ASL Gesture Recognition.Department of CSE Galgotias College of Engineering and Technology,Greater Noida, India

[3] Indriani, Mohammad Harris Ali, Suryaperdana Agoes1.,Advances in Engineering Research, volume 207.Applying Hand Gesture Recognition for User Guide Application Using MediaPipe.Proceedings of the 2nd International Seminar of Science and Applied Technology (ISSAT 2021)

[4] Smith, J., Johnson, A.,Lee, C.,Real-time Hand Gesture to Voice Conversion using Deep Learning.IEEE Transactions on Human-Machine Systems, 2020.

[5] Chen, L., Wang, S., Zhang, M.,Hand Gesture Recognition for Assisting Speech-Impaired Individuals.Proceedings of the ACM Conference on Assistive Technologies, 2019.

[6] Gupta, R., Sharma, A., Singh, A.,A Survey of Hand Gesture Recognition Techniques for Human-Computer Interaction. International Journal of Human-Computer Interaction, 2018.

Appendix B: Sample Code

Main

```
import csv import copy import
argparse import itertools from
collections import Counter from
collections import deque

import cv2 as cv import
numpy as np import
mediapipe as mp

#from tkinter import *
#from PIL import Image, ImageTk

from utils import CvFpsCalc from model
import KeyPointClassifier from model
import PointHistoryClassifier



def get_args():
    parser = argparse.ArgumentParser()

    parser.add_argument("--device", type=int, default=0)     parser.add_argument("--
width", help='cap width', type=int, default=960)     parser.add_argument("--height",
help='cap height', type=int, default=540)

    parser.add_argument('--use_static_image_mode', action='store_true')
parser.add_argument("--min_detection_confidence",
                    help='min_detection_confidence',
                    type=float,
default=0.7)
```

```
    parser.add_argument("--min_tracking_confidence",
                        help='min_tracking_confidence',
                        type=int,
default=0.5)

    args = parser.parse_args()

    return args

def main():
    # Argument parsing
#################################################################
    args = get_args()     """win=Tk()     win.title("Quirky Hand Combat")     #
Set the size of the window     win.geometry("1920x1080")# Create a Label
to capture the Video frames
    label =Label(win)     label.place(x=10, y=10)     button =
Button(win, text="Score",bg="green",fg="white")
button.place(x=1000,y=700)
score_label=Label(win,text="")     score_label.place(x=900,
y=750)     # Add the button to the window     button.pack()
score_label.pack()"""

    cap_device = args.device
    cap_width = args.width cap_height
    = args.height

    use_static_image_mode = args.use_static_image_mode
min_detection_confidence = args.min_detection_confidence
min_tracking_confidence = args.min_tracking_confidence

    use_brect = True
```

```
    # Camera preparation
##################################################################
    cap             =             cv.VideoCapture(cap_device)
cap.set(cv.CAP_PROP_FRAME_WIDTH,      cap_width)
cap.set(cv.CAP_PROP_FRAME_HEIGHT, cap_height)

    # Model load #############################################################
mp_hands = mp.solutions.hands     hands = mp_hands.Hands(
static_image_mode=use_static_image_mode,        max_num_hands=1,
min_detection_confidence=min_detection_confidence,
min_tracking_confidence=min_tracking_confidence,
    )

    keypoint_classifier = KeyPointClassifier()

    point_history_classifier = PointHistoryClassifier()
    # Create an instance of TKinter Window or frame
    """win= Tk("Quirky Hand Combat")
    # Set the size of the window win.geometry("1024x950")# Create a Label
    to capture the Video frames label =Label(win)
    label.place(x=0, y=0)"""

    # Read labels ###########################################################
    with open('model/keypoint_classifier/keypoint_classifier_label.csv',
              encoding='utf-8-sig') as f:
keypoint_classifier_labels = csv.reader(f)
keypoint_classifier_labels = [            row[0] for
row in keypoint_classifier_labels
        ]
    with open(
```

```
        'model/point_history_classifier/point_history_classifier_label.csv',
encoding='utf-8-sig') as f:
        point_history_classifier_labels = csv.reader(f)
point_history_classifier_labels = [            row[0] for
row in point_history_classifier_labels
        ]

    # FPS Measurement ########################################################
cvFpsCalc = CvFpsCalc(buffer_len=10)

    # Coordinate history
#################################################################
    history_length = 16     point_history =
deque(maxlen=history_length)

    # Finger gesture history ################################################
finger_gesture_history = deque(maxlen=history_length)

    #  ########################################################################
    #global mode
    mode=0

    while True:
        fps = cvFpsCalc.get()

        # Process Key (ESC: end) #################################################
        key = cv.waitKey(10)
if key == 27:  # ESC
            break
        number, mode = select_mode(key, mode)
```

```
    # Camera capture #####################################################
    ret, image = cap.read()
    if not ret:            break        image =
cv.flip(image, 1)  # Mirror display
    debug_image = copy.deepcopy(image)

    # Detection implementation
#############################################################
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

    image.flags.writeable = False
results = hands.process(image)
image.flags.writeable = True

    #  ####################################################################
    if results.multi_hand_landmarks is not None:
      for hand_landmarks, handedness in zip(results.multi_hand_landmarks,
                                            results.multi_handedness):
        # Bounding box calculation
        brect = calc_bounding_rect(debug_image, hand_landmarks)
        # Landmark calculation
        landmark_list = calc_landmark_list(debug_image, hand_landmarks)

        # Conversion to relative coordinates / normalized coordinates
pre_processed_landmark_list = pre_process_landmark(
            landmark_list)            pre_processed_point_history_list =
pre_process_point_history(                debug_image, point_history)
# Write to the dataset file            logging_csv(number, mode,
pre_processed_landmark_list,
                    pre_processed_point_history_list)
```

```
            # Hand sign classification
            hand_sign_id = keypoint_classifier(pre_processed_landmark_list)
if hand_sign_id == '':  # Point gesture
point_history.append(landmark_list[8])
            else:
                point_history.append([0, 0])

            # Finger gesture classification               finger_gesture_id =
0               point_history_len =
len(pre_processed_point_history_list)               if
point_history_len == (history_length * 2):
finger_gesture_id = point_history_classifier(
pre_processed_point_history_list)

            # Calculates the gesture IDs in the latest detection
finger_gesture_history.append(finger_gesture_id)               most_common_fg_id
= Counter(                  finger_gesture_history).most_common()

            # Drawing part
            debug_image = draw_bounding_rect(use_brect, debug_image, brect)
debug_image = draw_landmarks(debug_image, landmark_list)
debug_image = draw_info_text(                  debug_image,
                brect,                  handedness,
keypoint_classifier_labels[hand_sign_id],
point_history_classifier_labels[most_common_fg_id[0][0]],
            )
        else:
            point_history.append([0, 0])

        debug_image = draw_point_history(debug_image, point_history)
debug_image = draw_info(debug_image, fps, mode, number)
```

```
    # Screen reflection #############################################################
cv.imshow('Hand Gesture Recognition', debug_image)
    """debug_image = cv.cvtColor(debug_image,cv.COLOR_RGB2BGR)
img = Image.fromarray(debug_image)        imgtk =
ImageTk.PhotoImage(image = img)        label.imgtk = imgtk
label.configure(image=imgtk)        win.after(50,fun1)"""
  #fun1()    #mainloop()
cap.release()
cv.destroyAllWindows()
def select_mode(key,
mode):    number = -1
if 48 <= key <= 57:  # 0 ~ 9
number = key - 48    if
key == 110:  # n
mode = 0    if key == 107:
# k        mode = 1    if key
== 104:  # h        mode =
2    return number, mode


def calc_bounding_rect(image, landmarks):
  image_width, image_height = image.shape[1], image.shape[0]

  landmark_array = np.empty((0, 2), int)

  for _, landmark in enumerate(landmarks.landmark):
    landmark_x = min(int(landmark.x * image_width), image_width - 1)
landmark_y = min(int(landmark.y * image_height), image_height - 1)

    landmark_point = [np.array((landmark_x, landmark_y))]
```

```
        landmark_array = np.append(landmark_array, landmark_point, axis=0)

    x, y, w, h = cv.boundingRect(landmark_array)

    return [x, y, x + w, y + h]

def calc_landmark_list(image, landmarks):
    image_width, image_height = image.shape[1], image.shape[0]

    landmark_point = []

    # Keypoint     for _, landmark in
enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width), image_width - 1)
landmark_y = min(int(landmark.y * image_height), image_height - 1)
        # landmark_z = landmark.z

        landmark_point.append([landmark_x, landmark_y])

    return landmark_point



def pre_process_landmark(landmark_list):
    temp_landmark_list = copy.deepcopy(landmark_list)

    # Convert to relative coordinates     base_x, base_y = 0, 0     for
index, landmark_point in enumerate(temp_landmark_list):
        if index == 0:
            base_x, base_y = landmark_point[0], landmark_point[1]
```

```
        temp_landmark_list[index][0] = temp_landmark_list[index][0] - base_x
temp_landmark_list[index][1] = temp_landmark_list[index][1] - base_y

    # Convert to a one-dimensional list
temp_landmark_list = list(
itertools.chain.from_iterable(temp_landmark_list))

    # Normalization     max_value = max(list(map(abs,
temp_landmark_list)))

    def normalize_(n):
return n / max_value

    temp_landmark_list = list(map(normalize_, temp_landmark_list))

    return temp_landmark_list


def pre_process_point_history(image, point_history):
    image_width, image_height = image.shape[1], image.shape[0]

    temp_point_history = copy.deepcopy(point_history)

    # Convert to relative coordinates     base_x, base_y
= 0, 0     for index, point in
enumerate(temp_point_history):
        if index == 0:
            base_x, base_y = point[0], point[1]

        temp_point_history[index][0] = (temp_point_history[index][0] -
```

```
                                base_x) / image_width
        temp_point_history[index][1] = (temp_point_history[index][1] -
                                base_y) / image_height

    # Convert to a one-dimensional list
temp_point_history = list(
itertools.chain.from_iterable(temp_point_history))

    return temp_point_history


def logging_csv(number, mode, landmark_list, point_history_list):
if mode == 0:
        pass
    if mode == 1 and (0 <= number <= 9):
        csv_path = 'model/keypoint_classifier/keypoint.csv'
with open(csv_path, 'a', newline="") as f:            writer
= csv.writer(f)
            writer.writerow([number, *landmark_list])
if mode == 2 and (0 <= number <= 9):
        csv_path = 'model/point_history_classifier/point_history.csv'
with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *point_history_list])
return


def draw_landmarks(image, landmark_point):
    if len(landmark_point) > 0:
```

```
        # Thumb        cv.line(image, tuple(landmark_point[2]),
tuple(landmark_point[3]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[3]),
                (255, 255, 255), 2)        cv.line(image,
tuple(landmark_point[3]), tuple(landmark_point[4]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[3]), tuple(landmark_point[4]),
                                                        (255, 255, 255), 2)

        # Index finger
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]),
                (255, 255, 255), 2)        cv.line(image,
tuple(landmark_point[7]), tuple(landmark_point[8]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[7]), tuple(landmark_point[8]),
                (255, 255, 255), 2)

        # Middle finger
        cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[10]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[10]),
                (255, 255, 255), 2)        cv.line(image, tuple(landmark_point[10]),
tuple(landmark_point[11]),
                (0, 0, 0), 6)
```

```
        cv.line(image, tuple(landmark_point[10]), tuple(landmark_point[11]),
                (255, 255, 255), 2)        cv.line(image, tuple(landmark_point[11]),
tuple(landmark_point[12]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[11]), tuple(landmark_point[12]),
                (255, 255, 255), 2)

        # Ring finger
        cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[14]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[14]),
                (255, 255, 255), 2)        cv.line(image, tuple(landmark_point[14]),
tuple(landmark_point[15]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[14]), tuple(landmark_point[15]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[15]), tuple(landmark_point[16]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[15]), tuple(landmark_point[16]),
                (255, 255, 255), 2)

        # Little finger
        cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[18]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[18]),
                (255, 255, 255), 2)        cv.line(image, tuple(landmark_point[18]),
tuple(landmark_point[19]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[18]), tuple(landmark_point[19]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[19]), tuple(landmark_point[20]),
```

```
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[19]), tuple(landmark_point[20]),
            (255, 255, 255), 2)

        # Palm        cv.line(image, tuple(landmark_point[0]),
tuple(landmark_point[1]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[0]), tuple(landmark_point[1]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]),
            (255, 255, 255), 2)        cv.line(image,
tuple(landmark_point[5]), tuple(landmark_point[9]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[9]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[13]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[13]),
            (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[17]),
            (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[17]),
            (255, 255, 255), 2)        cv.line(image,
tuple(landmark_point[17]), tuple(landmark_point[0]),
            (0, 0, 0), 6)
```

```
        cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[0]),
                (255, 255, 255), 2)

    # Key Points
    for index, landmark in enumerate(landmark_point):
        if index == 0:              cv.circle(image, (landmark[0], landmark[1]),
5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 1:              cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 2:      cv.circle(image, (landmark[0], landmark[1]), 5, (255,
255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 3:              cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 4:          cv.circle(image, (landmark[0], landmark[1]), 8,
(255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
        if index == 5:
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                    -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 6:
```

```
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 7:
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 8:              cv.circle(image, (landmark[0], landmark[1]), 8,
(255, 255, 255),                    -1)
            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
if index == 9:          cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 10:             cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 11:             cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 12:             cv.circle(image, (landmark[0], landmark[1]), 8,
(255, 255, 255),
                      -1)
            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
if index == 13:              cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                      -1)
```

```
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 14:             cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 15:             cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 16:

        cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
                -1)
        cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
if index == 17:       cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 18:           cv.circle(image, (landmark[0], landmark[1]), 5,
(255, 255, 255),
                -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 19:  cv.circle(image, (landmark[0], landmark[1]), 5, (255,
255, 255),
                -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 20:  cv.circle(image, (landmark[0], landmark[1]), 8, (255,
255, 255),
                -1)
        cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
```

```
    return image


def draw_bounding_rect(use_brect, image, brect):
    if use_brect:         # Outer rectangle         cv.rectangle(image,
(brect[0], brect[1]), (brect[2], brect[3]),
                  (0, 0, 0), 1)

    return image


def draw_info_text(image, brect, handedness, hand_sign_text,
                   finger_gesture_text):
    cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[1] - 22),
                 (0, 0, 0), -1)

    info_text = handedness.classification[0].label[0:]
if hand_sign_text != "":
        info_text = info_text + ':' + hand_sign_text     cv.putText(image,
info_text, (brect[0] + 5, brect[1] - 4),
cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1, cv.LINE_AA)

    #if finger_gesture_text != "":
        #cv.putText(image, "timer" + i/10, (10, 60),
                 # cv.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 0), 4, cv.LINE_AA)
        #cv.putText(image, "Finger Gesture:" + finger_gesture_text, (10, 60),
                  #cv.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 2,
                  #cv.LINE_AA)

    return image
```

```
def draw_point_history(image, point_history):
for index, point in enumerate(point_history):
        if point[0] != 0 and point[1] != 0:
            cv.circle(image, (point[0], point[1]), 1 + int(index / 2),
                      (152, 251, 152), 2)

    return image


def draw_info(image, fps, mode, number):
    """"""
    cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT_HERSHEY_SIMPLEX,
               1.0, (0, 0, 0), 4, cv.LINE_AA)     cv.putText(image, "FPS:" + str(fps),
(10, 30), cv.FONT_HERSHEY_SIMPLEX,
               1.0, (255, 255, 255), 2, cv.LINE_AA)"""

    mode_string = ['Logging Key Point', 'Logging Point History']
if 1 <= mode <= 2:
        cv.putText(image, "MODE:" + mode_string[mode - 1], (10, 90),
cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1,
                   cv.LINE_AA)
        if 0 <= number <= 9:
            cv.putText(image, "NUM:" + str(number), (10, 110),
cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1,
                       cv.LINE_AA)
    return image


if __name__ == '__main__':
    main()
```

Training

```
import csv import numpy as np import tensorflow as
tf from sklearn.model_selection import
train_test_split RANDOM_SEED = 42
dataset = 'model/keypoint_classifier/keypoint.csv' model_save_path
= 'model/keypoint_classifier/keypoint_classifier.hdf
NUM_CLASSES = 5
X_dataset = np.loadtxt(dataset, delimiter=',', dtype='float32', usecols=list(range(1, (21 * 2) + 1)))
y_dataset = np.loadtxt(dataset, delimiter=',', dtype='int32', usecols=(0))
X_train, X_test, y_train, y_test = train_test_split(X_dataset, y_dataset, train_size=0.75,
random_state=RANDOM_SEED)
model = tf.keras.models.Sequential([
   tf.keras.layers.Input((21 * 2, )),
tf.keras.layers.Dropout(0.2),    tf.keras.layers.Dense(20,
activation='relu'),    tf.keras.layers.Dropout(0.4),
tf.keras.layers.Dense(10, activation='relu'),
tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
odel.summary()  # tf.keras.utils.plot_model(model, show_shapes=True)
cp_callback = tf.keras.callbacks.ModelCheckpoint(    model_save_path,
verbose=1, save_weights_only=False) es_callback =
tf.keras.callbacks.EarlyStopping(patience=20, verbose=1)
model.compile(    optimizer='adam',
loss='sparse_categorical_crossentropy',    metrics=['accuracy']
)
model.fit(
X_train,    y_train,
   epochs=1000,    batch_size=128,
validation_data=(X_test, y_test),
callbacks=[cp_callback, es_callback]
```

```
)
val_loss, val_acc = model.evaluate(X_test, y_test, batch_size=128) model
= tf.keras.models.load_model(model_save_path) predict_result =
model.predict(np.array([X_test[0]])) print(np.squeeze(predict_result))
print(np.argmax(np.squeeze(predict_result))) import pandas as pd
import seaborn as sns import matplotlib.pyplot as plt from
sklearn.metrics import confusion_matrix, classification_report def
print_confusion_matrix(y_true, y_pred, report=True):     labels =
sorted(list(set(y_true)))     cmx_data = confusion_matrix(y_true, y_pred,
labels=labels)     df_cmx = pd.DataFrame(cmx_data, index=labels,
columns=labels)     fig, ax = plt.subplots(figsize=(7, 6))
   sns.heatmap(df_cmx, annot=True, fmt='g' ,square=False)
ax.set_ylim(len(set(y_true)), 0)     plt.show()     if report:
       print('Classification Report')
print(classification_report(y_test, y_pred)) Y_pred =
model.predict(X_test) y_pred = np.argmax(Y_pred,
axis=1) print_confusion_matrix(y_test, y_pred)
model.save(model_save_path, include_optimizer=False)
tflite_save_path = 'model/keypoint_classifier/keypoint_classifier.tflite'
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quantized_model = converter.convert() open(tflite_save_path,
'wb').write(tflite_quantized_model) interpreter =
tf.lite.Interpreter(model_path=tflite_save_path)
interpreter.allocate_tensors() input_details =
interpreter.get_input_details() output_details =
interpreter.get_output_details()
interpreter.set_tensor(input_details[0]['index'], np.array([X_test[0]]))

%%time
```

```
interpreter.invoke() tflite_results =
interpreter.get_tensor(output_details[0]['index'])
print(np.squeeze(tflite_results))
print(np.argmax(np.squeeze(tflite_results)))
```

Appendix C: CO-PO And CO-PSO Mapping

COURSE OUTCOMES:

After completion of the course the student will be able to

SL. NO	DESCRIPTION	Blooms' Taxonomy Level
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools & advanced programming techniques (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)	Level 3: Apply

CO-PO AND CO-PSO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO3
CO1	3	3	3	3		2	2	3	2	2	2	3	2	2	2
CO2	3	3	3	3	3	2		3	2	3	2	3	2	2	2
CO3	3	3	3	3	3	2	2	3	2	2	2	3			2
CO4	2	3	2	2	2			3	3	3	2	3	2	2	2
CO5	3	3	3	2	2	2	2	3	2		2	3	2	2	2

3/2/1: high/medium/low

JUSTIFICATIONS FOR CO-PO MAPPING

MAPPING	LOW/ MEDIUM/ HIGH	JUSTIFICATION
100003/CS622T.1-PO1	**HIGH**	Identify technically and economically feasible problems by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.1-PO2	**HIGH**	Identify technically and economically feasible problems by analysing complex engineering problems reaching substantiated conclusions using first principles of mathematics.
100003/CS622T.1-PO3	**HIGH**	Design solutions for complex engineering problems by identifying technically and economically feasible problems.
100003/CS622T.1-PO4	**HIGH**	Identify technically and economically feasible problems by analysis and interpretation of data.
100003/CS622T.1-PO6	**MEDIUM**	Responsibilities relevant to the professional engineering practice by identifying the problem.
100003/CS622T.1-PO7	**MEDIUM**	Identify technically and economically feasible problems by understanding the impact of the professional engineering solutions.
100003/CS622T.1-PO8	**HIGH**	Apply ethical principles and commit to professional ethics to identify technically and economically feasible problems.
100003/CS622T.1-PO9	**MEDIUM**	Identify technically and economically feasible problems by working as a team.
100003/CS622T.1-PO10	**MEDIUM**	Communicate effectively with the engineering community by identifying technically and economically feasible problems.
100003/CS622T.1-P011	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles by selecting the technically and economically feasible problems.
100003/CS622T.1-PO12	**HIGH**	Identify technically and economically feasible problems for long term learning.
100003/CS622T.1-PSO1	**MEDIUM**	Ability to identify, analyze and design solutions to identify technically and economically feasible problems.
100003/CS622T.1-PSO2	**MEDIUM**	By designing algorithms and applying standard practices in software project development and Identifying technically and economically feasible problems.
100003/CS622T.1-PSO3	**MEDIUM**	Fundamentals of computer science in competitive research can be applied to Identify technically and economically feasible problems.
100003/CS622T.2-PO1	**HIGH**	Identify and survey the relevant by applying the knowledge of mathematics, science, engineering fundamentals.

100003/CS622T.2-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems get familiarized with software development processes.
100003/CS622T.2-PO3	**HIGH**	Design solutions for complex engineering problems and design based on the relevant literature.
100003/CS622T.2-PO4	**HIGH**	Use research-based knowledge including design of experiments based on relevant literature.
100003/CS622T.2-PO5	**HIGH**	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes by using modern tools.
100003/CS622T.2-PO6	**MEDIUM**	Create, select, and apply appropriate techniques, resources, by identifying and surveying the relevant literature.
100003/CS622T.2-PO8	**HIGH**	Apply ethical principles and commit to professional ethics based on the relevant literature.
100003/CS622T.2-PO9	**MEDIUM**	Identify and survey the relevant literature as a team.
100003/CS622T.2-PO10	**HIGH**	Identify and survey the relevant literature for a good communication to the engineering fraternity.
100003/CS622T.2-PO11	**MEDIUM**	Identify and survey the relevant literature to demonstrate knowledge and understanding of engineering and management principles.
100003/CS622T.2-PO12	**HIGH**	Identify and survey the relevant literature for independent and lifelong learning.
100003/CS622T.2-PSO1	**MEDIUM**	Design solutions for complex engineering problems by Identifying and survey the relevant literature.
100003/CS622T.2-PSO2	**MEDIUM**	Identify and survey the relevant literature for acquiring programming efficiency by designing algorithms and applying standard practices.
100003/CS622T.2-PSO3	**MEDIUM**	Identify and survey the relevant literature to apply the fundamentals of computer science in competitive research.
100003/CS622T.3-PO1	**HIGH**	Perform requirement analysis, identify design methodologies by using modern tools & advanced programming techniques and by applying the knowledge of mathematics, science, engineering fundamentals.
100003/CS622T.3-PO2	**HIGH**	Identify, formulate, review research literature for requirement analysis, identify design methodologies and develop adaptable & reusable solutions.

100003/CS622T.3-PO3	**HIGH**	Design solutions for complex engineering problems and perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO4	**HIGH**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.3-PO5	**HIGH**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools.
100003/CS622T.3-PO6	**MEDIUM**	Perform requirement analysis, identify design methodologies and assess societal, health, safety, legal, and cultural issues.
100003/CS622T.3-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts and Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PO8	**HIGH**	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions by applying ethical principles and commit to professional ethics.
100003/CS622T.3-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.3-PO10	**MEDIUM**	Communicate effectively with the engineering community and with society at large to perform requirement analysis, identify design methodologies.
100003/CS622T.3-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering requirement analysis by identifying design methodologies.
100003/CS622T.3-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS622T.3-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and prior to that perform requirement analysis, identify design methodologies.
100003/CS622T.4-PO1	**MEDIUM**	Prepare technical report and deliver presentation by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.4-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by preparing technical report and deliver presentation.

100003/CS622T.4-PO3	**MEDIUM**	Prepare Design solutions for complex engineering problems and create technical report and deliver presentation.
100003/CS622T.4-PO4	**MEDIUM**	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions and prepare technical report and deliver presentation.
100003/CS622T.4-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and Prepare technical report and deliver presentation.
100003/CS622T.4-PO8	**HIGH**	Prepare technical report and deliver presentation by applying ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
100003/CS622T.4-PO9	**HIGH**	Prepare technical report and deliver presentation effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS622T.4-PO10	**HIGH**	Communicate effectively with the engineering community and with society at large by prepare technical report and deliver presentation.
100003/CS622T.4-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work by prepare technical report and deliver presentation.
100003/CS622T.4-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by prepare technical report and deliver presentation.
100003/CS622T.4-PSO1	**MEDIUM**	Prepare a technical report and deliver presentation to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas.
100003/CS622T.4-PSO2	**MEDIUM**	To acquire programming efficiency by designing algorithms and applying standard practices in software project development and to prepare technical report and deliver presentation.
100003/CS622T.4-PSO3	**MEDIUM**	To apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs by preparing technical report and deliver presentation.
100003/CS622T.5-PO1	**HIGH**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS622T.5-PO2	**HIGH**	Identify, formulate, review research literature, and analyze complex engineering problems by applying engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PO3	**HIGH**	Apply engineering and management principles to achieve the goal of the project and to design solutions for complex engineering problems and design system components or processes that meet the specified needs.
100003/CS622T.5-PO4	**MEDIUM**	Apply engineering and management principles to achieve the goal of the project and use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS622T.5-PO5	**MEDIUM**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO6	**MEDIUM**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities by applying engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO7	**MEDIUM**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO8	**HIGH**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice and to use the engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO9	**MEDIUM**	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO11	**MEDIUM**	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PO12	**HIGH**	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO1	**MEDIUM**	The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas. Apply engineering and management principles to achieve the goal of the project.

100003/CS622T.5-PSO2	**MEDIUM**	The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry and to apply engineering and management principles to achieve the goal of the project.
100003/CS622T.5-PSO3	**MEDIUM**	The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur and apply engineering and management principles to achieve the goal of the project.