CS172 Computer Vision I:
Final Project Report
3D Board Game

Lu Yuzhou, Cheng Chen, Chen Yiming, Du Jiaying
2021533001 2020533145 2021533017 2021533037
luyzh chengchen chenym1 dujy2 @shanghaitech.edu.cn

Abstract

We developed a range of projects, including a 2D board game using OpenCV with full implementation of game logic and user interface, a 2D danmaku game using Unity, and a 3D drawboard also using Unity. With Stable Diffusion, we colorize the output of the drawboard. To enable gesture-based interactions, we abstracted hand joint positions, collected a hand dataset, and trained a DD-Net model for hand gesture classification, which we integrated as inputs for our games. Additionally, we applied knowledge distillation to enhance the model's robustness and speed. Through benchmark experiments, we concluded that the existing state-of-the-art (SOTA) skeleton-based gesture recognition models exhibit redundant architecture designs.

1. Introduction

This report presents the culmination of a final project for the Computer Vision I course, where we focused on the practical application of computer vision (CV) techniques to create engaging experiences for the public. Our primary objective was to explore the integration of CV techniques in game development, aiming to enhance user interactions and immersion. To achieve this, we embarked on three distinct projects, each highlighting the utilization of CV technologies in different game genres.

The first project involved developing a 2D board game using OpenCV, which featured a comprehensive implementation of game logic and a visually appealing user interface. Through computer vision, we incorporated gesture recognition capabilities, allowing players to interact with the game using hand movements and gestures. This added a new level of interactivity and excitement to the gameplay experience.

For the second project, we delved into the realm of 2D danmaku games using the Unity game engine. Leveraging computer vision techniques, we integrated gesture-based controls that enabled players to navigate through intense bullet hell scenarios by moving their hands and executing gestures. The fusion of CV and game development techniques resulted in an immersive and captivating gameplay experience.

In the third project, we ventured into the realm of 3D drawboards, again utilizing the Unity engine. With computer vision algorithms, we abstracted hand joint positions and employed them as a means of virtual drawing. This allowed users to sketch and create in a 3D virtual space simply by moving their hands, enhancing creativity and interaction in a unique way. Furthermore, with the output of the draw pattern, we used stable diffusion to colorize the pattern, which enabled the possibility for the public to draw by their hands then generate the good-looking images.

To enable the aforementioned gesture-based interactions, we collected a comprehensive hand dataset and trained a DDNet model specifically designed for hand gesture classification. Moreover, to further enhance the performance of the model, we applied knowledge distillation techniques, optimizing both its robustness and speed.

Through extensive benchmark experiments, we discovered that the existing state-of-the-art skeleton-based gesture recognition models exhibited redundant architecture designs. This finding not only emphasized the importance of efficient model design but also showcased the potential for making these models more streamlined and faster through knowledge distillation.

Overall, our project highlights the exciting possibilities of integrating computer vision techniques into game development, opening doors for innovative and immersive gaming experiences and discover the fact that the architecture design of SOTA skeleton-based gesture recognition model can be distilled into a more complicated model.

1.1. Aim

The CV technologies may be doing much more works in the background of our society. But the public may not be so aware about the significance of these game-changing techs. While implying cutting-edge knowledge into application, we hope to develop easy and enjoyable games related to CV to import these techs to public minds.

1.2. RoadMap

We use D455 camera to catch color and depth images. With the help of Mediapipe Model, we transfer them into 2D and 3D coordinates of the hand joints. Then we trained a network to classify the joints into 4 gestures. And we use web-socket to send the datapack to unity engine as game input.

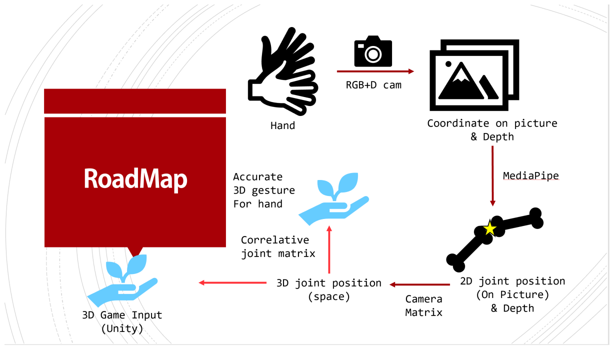
RoadMap
Hand
RGB+D cam
Coordinate on picture & Depth
MediaPipe
Accurate 3D gesture For hand
Correlative joint matrix
2D joint position (On Picture) & Depth
Camera Matrix
3D joint position (space)
3D Game Input (Unity)

Figure 1. RoadMap

2. Previous Researches

As a part of pose detection, Hand Pose Estimation has been one of the most popular problems in CV. Divided by the data source, the research before 2018 are mostly RGB+D or multiple RGB. The recent research are most about single RGB. Though for accuracy they are skyhigh, we want to use light-load solutions.

2.1. Models

- YOHA [2]
  Written in JavaScript and Tensorflow, YOHA is a light-load hand tracking engine with notable accuracy. But YOHA can only export 2D joint coordinates, and YOHA is out of maintenance. Our group members also have nothing to do with web SDK development.

- OpenPose [7]
  Small, Fast, great! But with limited joints placed on hand, and is a whole-body tracking model.

- MediaPipe [1]
  Maintained by google, Mediapipe is a light and easy-to-use sets of ML. Pick out hand lankmark recognition as our model to transfer RGB+D image to joint coordinates. It has much more better support than above two, and also maintains high accuracy.

2.2. Datasets

- RGB+D
  Existing RGB+D datasets, for example, *Ego3DHands* [4] and *MVHand* [10], are mostly focused on hand landmark extraction. They provided video frames and the position data of the physical hand in the frame. This function is done by Mediapipe.
  For gesture classification, *Domain and View point Agnostic Hand Action Recognition* [6] is an outstanding dataset with more than 200 sets of inputs. The input of the dataset is a siries of weights, and the recognised gesture is always binded with an object which is interacting with the hand. In our games, we do not want players to hold any specific object to satisfy a gesture.

- Our Own
  We try to collect a little dataset for classification. First, we decided four gestures for controlling. Then we asked our team members to record the motions. One person must record a gesture with both hands for 9 times, each time about 1 minute and fixed at 30 frames.

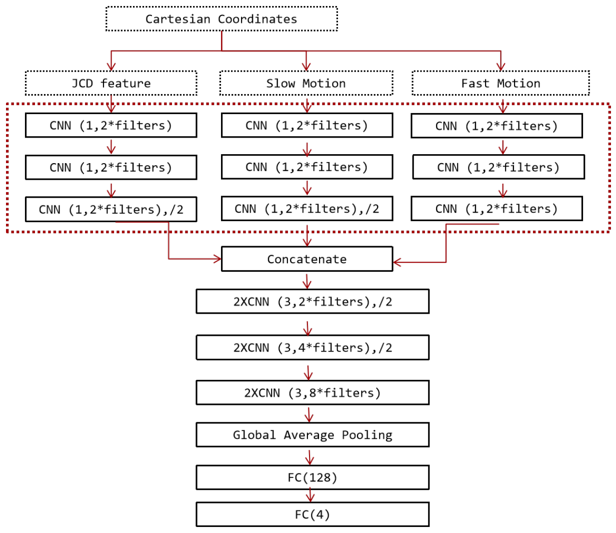

Figure 2. The structure of DD-Net is described as follows: ”2×CNN(3, 2*filters), /2” represents two 1D ConvNet layers with a kernel size of 3 and channels equal to 2 times the number of filters, followed by Maxpooling with a stride of 2. The remaining ConvNet layers follow the same format. FC represents Fully Connected Layers (or Dense Layers). The model size can be adjusted by modifying the number of filters.

3. Gesture Classification

Since we have recognized the skeleton of human hands with *MediaPipe* [1], we selected *Double-feature Double-motion Network (DD-Net)* [9] for skeleton-based action recognition. *DD-Net* which utilize skeleton sequence properties is a lightweight network structure, reaching a fast speed, as 5,500 FPS on a GTX 2080Ti GPU, or, 2,800 FPS on a Intel i7-107050H CPU.

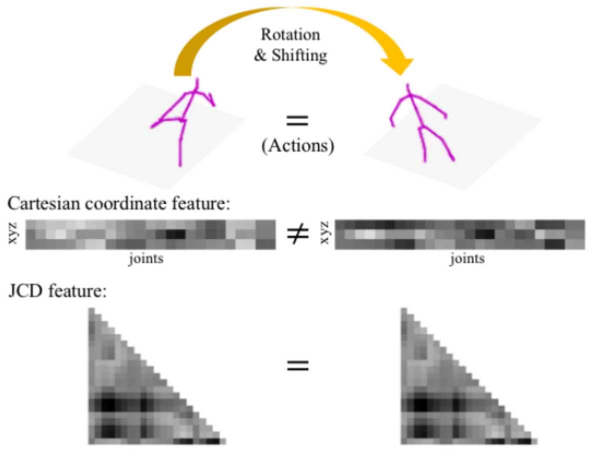

Figure 3. Cartesian coordinate is a location-viewpoint variable but JCD feature is a location-viewpoint invariant. This picture is from [9].

3.1. Network Architecture

Figure 2 illustrates the network structure of DD-Net. In the subsequent sections, we present the behind design principle for input features and network configurations in DD-Net.

3.1.1 Modeling Location-viewpoint Invariant Feature by Joint Collection Distances (JCD)

In the field of skeleton-based action recognition, two commonly utilized types of input features include the geometric feature (e.g. [11]) and the Cartesian coordinate feature (e.g. [8]). The Cartesian coordinate feature is susceptible to variations in locations and viewpoints, as depicted in Figure 3 , where rotations or shifts of skeletons can significantly alter the Cartesian coordinate feature. On the other hand, the geometric feature, such as angles or distances, is invariant to location and viewpoint, making it a longstanding choice for skeleton-based action recognition.

However, existing geometric features often require extensive redesigning when applied to different datasets ([18], [22]) or may contain redundant elements ([33]). To address these challenges, Yang et al. [9] elicited a novel feature called Joint Collection Distances (JCD). The JCD feature involves calculating the Euclidean distances between pairs of collective joints, resulting in a symmetric matrix. To reduce redundancy, they utilized only the lower triangular matrix, excluding the diagonal elements, as the JCD feature. Consequently.

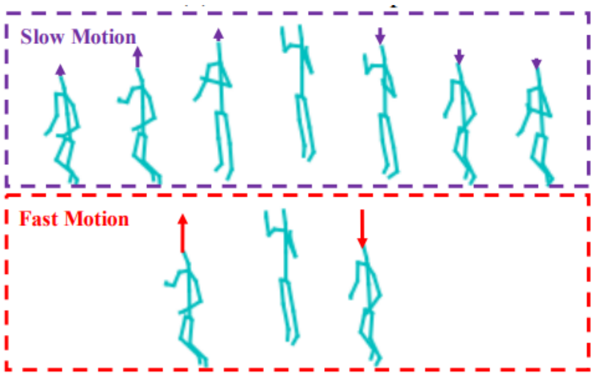

Figure 4. Motion scale variation. This picture is from [9].

3.1.2 Modeling Global Scale-invariant Motions by a Two-scale Motion Feature

While the Joint Collection Distances (JCD) feature, like other geometric features, remains invariant to location and viewpoint, it lacks global motion information. *DD-Net* takes a different approach by seamlessly integrating both.

To capture global motions that are location-invariant, we calculate the temporal differences (i.e., speed) of the Cartesian coordinate feature. However, it is worth noting that the scale of global motions may vary for the same action, with some being faster and others slower (as shown in Figure 4). To learn a robust global motion feature, it is crucial to consider both fast and slow motions.

Technically, the two-scale motions can be generated by the following equation:

$$M^k_{\text{slow}} = S^{k+1} - S^k, k \in \{1, 2, 3, \dots, K-1\};$$
$$M^k_{\text{fast}} = S^{k+2} - S^k, k \in \{1, 3, \dots, K-2\};$$

where $S^k = \{(x, y, z)^k_1, (x, y, z)^k_2, \dots, (x, y, z)^k_N\}$ is the sequence of N key points at frame k.

3.2. Dataset

3.2.1 Binary Classification Dataset

Before reaching the milestone, we assembled a dataset consisting of one gesture, where the positive sample represents a fist and the negative samples encompass any other gesture. Each sample consists of approximately 80 sequences. Specifically, we recorded a total of 72 samples for the left hand of a single individual.

3.2.2 Multi Classification Dataset

After reaching the milestone, we gathered a dataset comprising four gestures, which are illustrated in Figure 5. The dataset follows a hierarchical structure. For each gesture, every member of our group recorded the gesture nine times using both their left and right hands. Consequently, we obtained a total of 280 samples, with each sample containing approximately 60 to 120 sequences (i.e., frames).

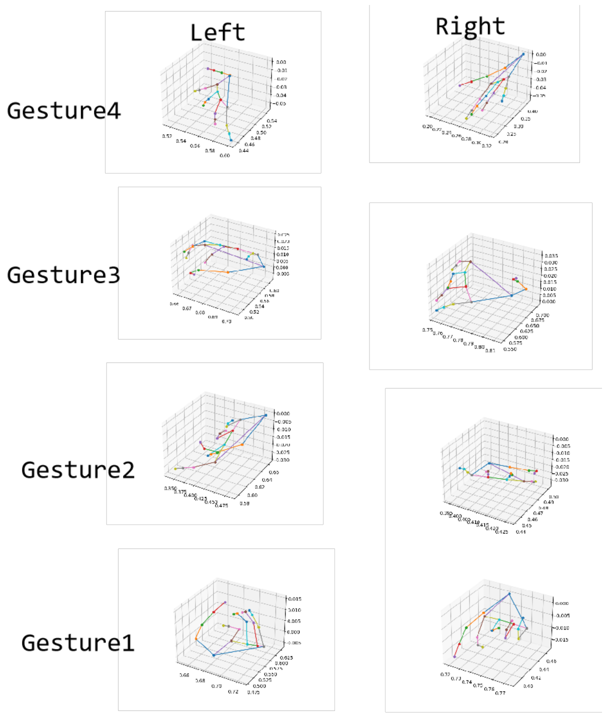

Figure 5. The four gestures in our multi-classification dataset

3.3. Experiment

3.3.1 Experiment Setup

Since the *DD-Net* is small, it is feasible to put all of the training sets into one batch on a single Intel i7-107050H CPU. We choose the default parameter setting in [9]. In all experiment, we split the dataste with the size of trainign set : the size of test set $= 1 : 1$

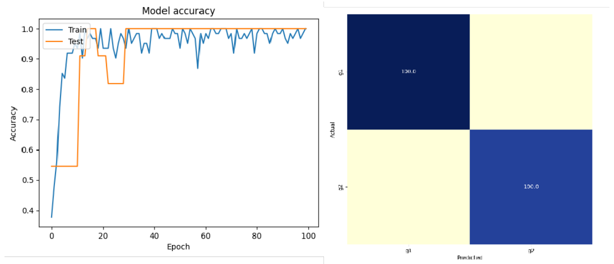

Figure 6. The training procedure and result of *DD-Net* on the binary classification dataset. The left figure is the model accuracy on the training set and test set. The right figure is the confusion matrix of the fist gesture and the none gesture.

3.3.2 Binary Classification

Before achieving the milestone, we trained the *DD-Net* on a binary classification dataset. The left figure in Figure 6 indicates that the model is overfitting due to the dataset's small size and the complexity of the model. Furthermore, when we implemented the model in our games, the real-time recognition performance was notably poor. Nevertheless, this was merely a trial run and served as a warm-up for the subsequent multi-classification task.

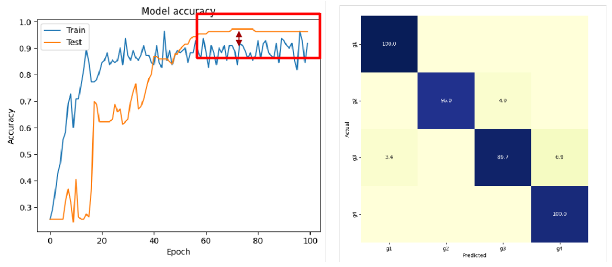

Figure 7. The training procedure and result of *DD-Net* on the multi classification dataset. The left figure is the model accuracy on the training set and test set. The right figure is the confusion matrix of four gestures.

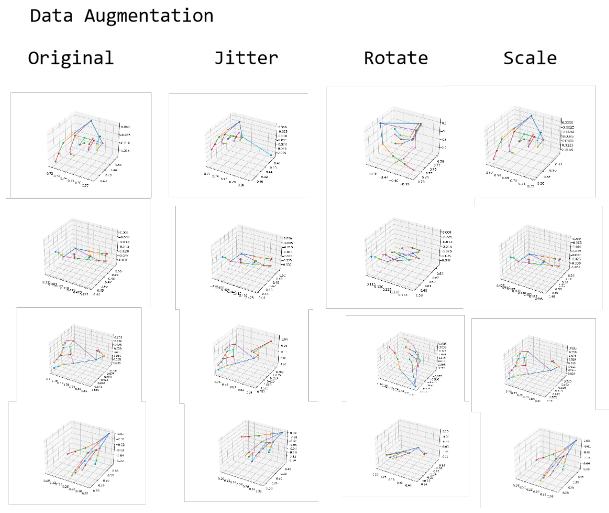

Figure 8. Three data augmentation methods on the training set in multi-classification dataset

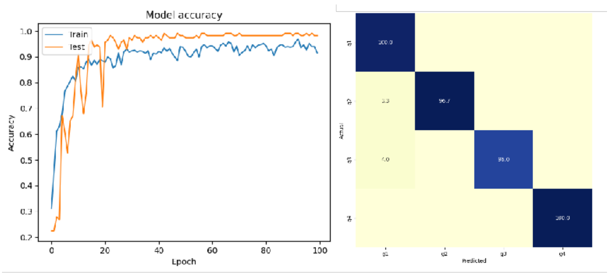

Figure 9. The training procedure and result of *DD-Net* on the augmented multi classification dataset. The left figure is the model accuracy on the augmented training set and test set. The right figure is the confusion matrix of four gestures.

3.3.3 Multi Classification

Before reaching the milestone, we trained the *DD-Net* on a binary classification dataset (as shown in Figure 7). We achieved an average accuracy of 96.2%. However, we noticed that the test accuracy was slightly higher than the training accuracy. This phenomenon was caused by a difference in the distribution of samples between the training set and the test set. In other words, our training set, consisting of only 144 samples, was still too small to effectively train the *DD-Net*.

To address this issue, we employed three methods to augment the training set, as illustrated in Figure 8. As a result, we increased the training samples to 576. With this enlarged dataset, we achieved an accuracy of 98.2%, showing a 2% improvement. Comparing Figure 7 and Figure 9, we can observe that the training curve became more stable, indicating that the model became more robust. Additionally, the gap between the training accuracy and the test accuracy reduced. When we deployed the model in our games, we found it performed satisfactorily in terms of both speed and accuracy.

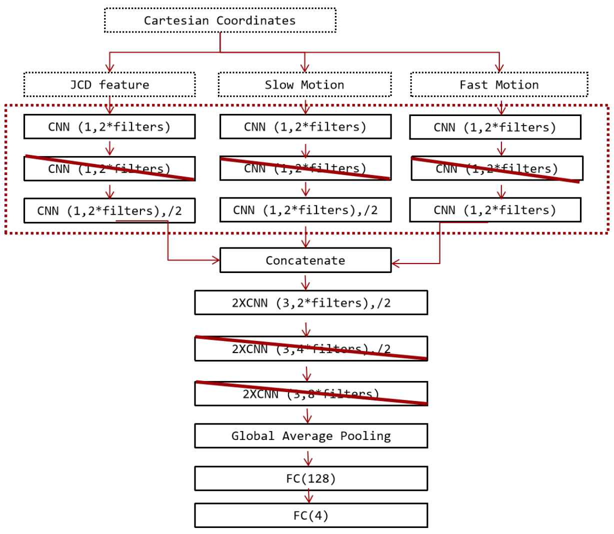

Figure 10. The architecture of the student model in knowledge distillation

3.3.4 Knowledge Distillation

There is still a gap between the training accuracy and test accuracy, indicating that the model is still too complex for the augmented dataset. Therefore, we attempted to reduce the model size. However, when we decreased the number of filters to achieve size reduction, we observed a rapid drop in model accuracy. Since most layers already had only 2 filters, there was limited room for further reduction. As a result, we decided to explore the knowledge distillation method, which yielded some interesting results.

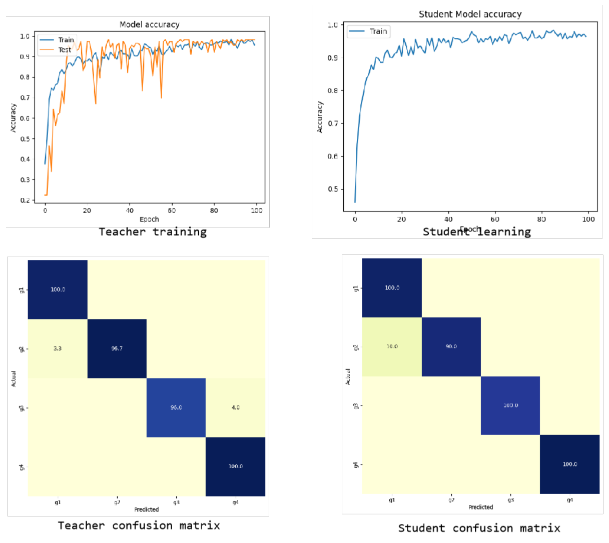

Figure 11. The knowledge distillation result on our augmented dataset

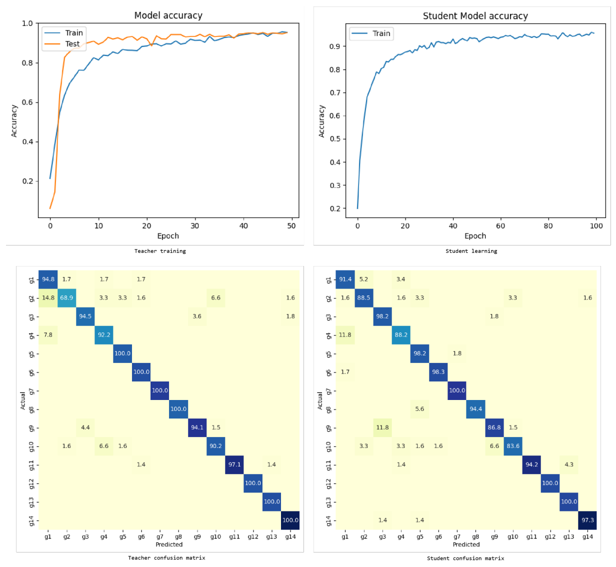

Figure 12. The knowledge distillation result on *SHREC17* [3]

We designed the student model architecture by removing five layers from the original architecture (as shown in Figure 10), resulting in a reduction of trainable parameters from 0.151M to 0.025M. For the distillation process, we first trained a teacher network using the original *DD-Net*. Next, we utilized the logits of the teacher's network as the target for training the student network. This size reduction also improved the inference speed by a factor of 2. The training results are shown in Figure 11. Despite the decreased size, the student model achieved a high performance with a minimal decrease in total accuracy of only 0.9%, going from 98.2% to 97.3%. This demonstrates that the student model maintained strong performance even after size reduction. **Therefore, we hypothesized that for existing skeleton-based gesture classification models, there might still be many redundant parameters.**

To test this hypothesis, we used the same student model architecture to distill a teacher model trained on a much larger dataset called *SHREC17* [3]. The *SHREC17* dataset consists of 1960 training samples and 840 testing samples, representing 14 genres of gestures. The training results are shown in Figure 12. Despite the reduced size, the student model achieved high performance with only a minimal decrease in total accuracy, dropping from 95.1% to 94.1%. **This provides strong evidence that the state-of-the-art skeleton-based gesture recognition models contain redundant parameters, and through knowledge distillation, we can make them more efficient and faster without compromising performance.**

4. Techniques

4.1. Python-Unity Interface

Once extracted the 3d coordinates in the space, we can use zmq to send TCP massages to port 5555 for every frame. These message can include 3d coordinates of hand landmarks, gesture, status code and more depending on the game need. Unity script fetches the buffer any time it want to avoid blocking the process. Then it decodes the package and provide the input to the game engine. See Figure 13.

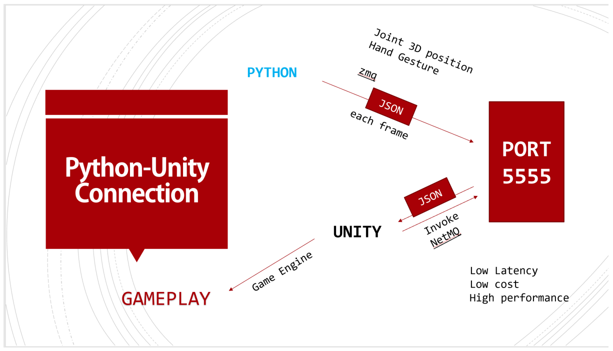

Figure 13. Python Unity Connection

5. Product

5.1. 2D board game

5.1.1 Description

2D board game is a simple game for two players to play, each player need to use one hand to control a ”board”, and use it to hit the ball. once your board hit the ball, you will get 1 score. Each player can use 4 gestures, open hand and you can control your board, make a fist and the game will restart, “OK” will make the ball moving fast and show your index finger to pause this game.

5.1.2 Image

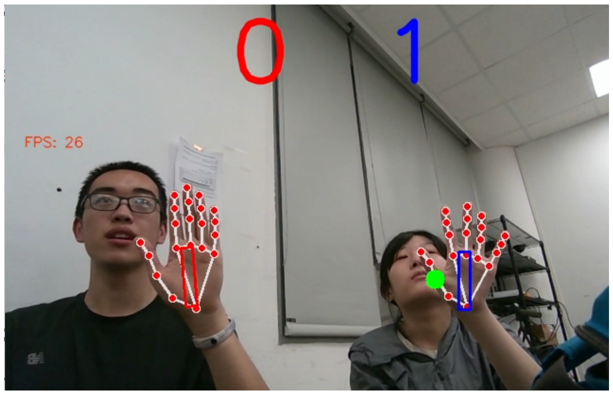

Figure 14. A screenshot for two player playing, more in demo videos.

As shown in Figure 14, this game is some simple shapes drown on image captured by camera. Player can see their hands and board, which makes it easy to control and play.

5.2. 2D damaku game

5.2.1 Description

Experience the thrill of a simple, yet engaging 2D ’Danmaku’ style game, where your primary objective is to vanquish the spawned enemies while ensuring your survival for as long as possible. This game utilizes your hand’s rotation, gesture, and position as inputs. The player-controlled character will fire bullets in the direction of your hand when your hand is posed in the ’1’ gesture. During other gestures, the character will remain idle. Be vigilant as collision with enemy bullets will cause damage, and direct contact with an enemy will result in a game over.

5.2.2 Image

As shown in Figure 15, the enemies are represented by green triangles, while you, the player, take the form of a red hexagon.

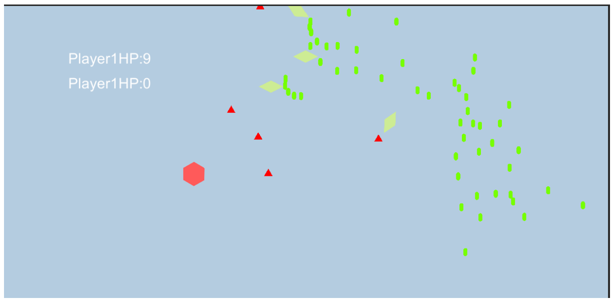

Figure 15. screenshot

5.3. 3D draw game

5.3.1 Description

The 3D Draw Game is a transformative adaptation of the traditional 2D drawing game by YOHA, enhanced by introducing a third dimension. The depth value is cleverly obtained by calculating the relative distance based on the size of the user's hand. Four specific gestures (0, 1, 3, 5) are uniquely associated with different functions within the game: '0' clears all the drawn lines, '1' activates the erase mode, '3' enables drawing, while '5' signifies an idle state.

The drawing function is implemented through the 'MonoBehaviour' class in the Unity package and a 'LineRenderer' component, which can be attached to any GameObject. In 'draw' mode, a new 'LineRenderer' is instantiated to track the movement of your hand, point by point. It then renders the line in real time, mirroring your hand's path. Once you switch to another gesture, the line ends, and all corresponding points are preserved in that 'LineRenderer'.

The erasing function is realized by utilizing Unity's collider functionality. Any 'LineRenderer' objects that intersect with your '1' (erase) gesture will be destroyed. The clear function, mapped to '0' gesture, empties all 'LineRenderer' objects stored in the array, effectively clearing the canvas.

5.3.2 Image

Figure 16 is a sample screenshot of the game in action:

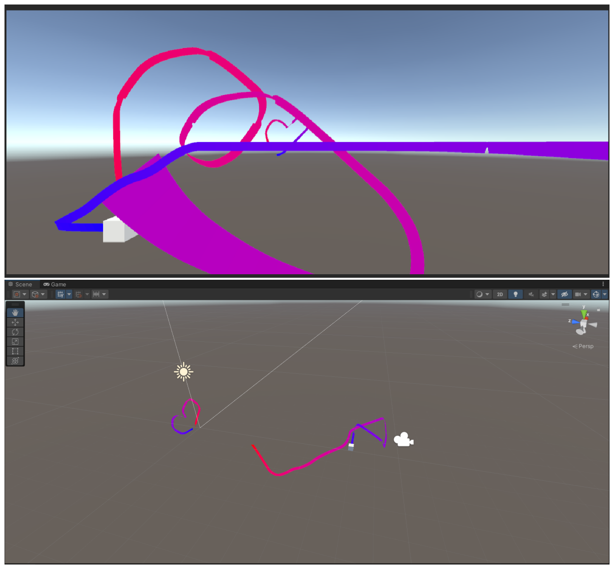

Figure 16. Up: screenshot. Bottom: In the scene It looks like

5.4. Auto Colorize

Once we catch the drawing pattern, we use stable diffusion [5] to automatically colorize it. We can easily get a colorful image by only draw the abstract lines of an object. See Figure 17 for an example of drawing a flower.

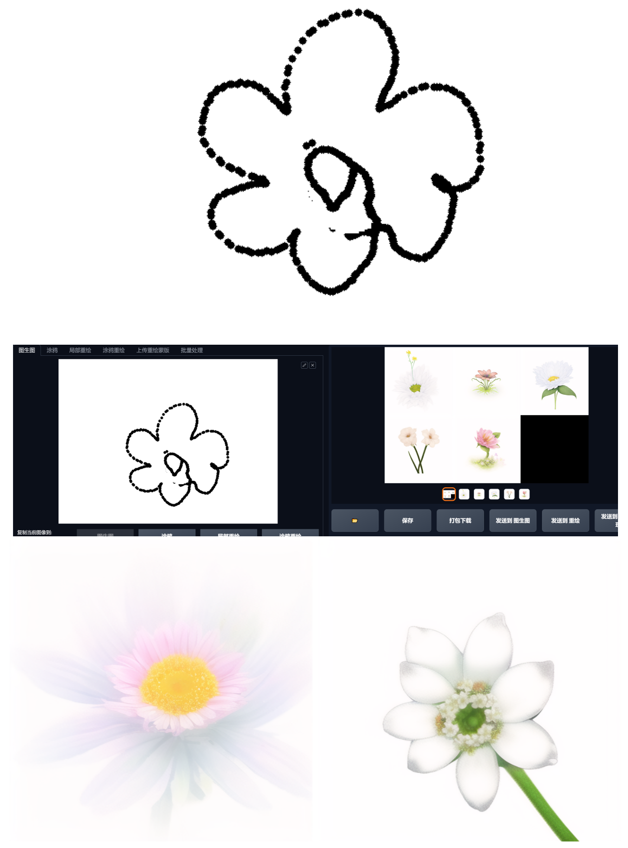

Figure 17. The Draw picture of flower and generated picture

6. Conclusion

The outcomes of this project highlight the successful implementation of CV techniques in game development, emphasizing the potential for creating engaging and interactive experiences for the public. The integration of OpenCV, Unity, RealSense D455 camera, MediaPipe, and deep learning methodologies demonstrates the practical application of CV techniques and lays the groundwork for further exploration and innovation in this field. This project serves as a valuable learning experience and paves the way for future advancements in the intersection of computer vision and game development.

7. Acknowledgements

I would like to express my sincere gratitude to Professor Jingya Wang for her guidance and support throughout this research. It was under her tutelage that we learned the fundamentals of computer vision. I am also grateful to our teaching assistants, Qianyang Wu and Jie Tian, for providing valuable insights and practical suggestions that directed our project towards success. Additionally, I would like to extend my thanks to the VRVC laboratory for providing us with the D455 camera, which greatly facilitated our experiments.

I would like to acknowledge the hard work and cooperation of my fellow classmates within the group. Their dedication and support have been invaluable throughout this journey. I am also thankful for the continuous advancements made by previous researchers in the field of computer vision, as well as all those who have contributed to our research by providing assistance and support.

Finally, I would like to express my best wishes for the ongoing progress and development of computer vision technology. May it continue to advance and contribute more wisdom and power to the development of human society.

References

[1] Website for mediapipe. `https://developers.google.com/mediapipe`.

[2] Website for yoha. `https://github.com/handtracking-io/yoha`.

[3] Quentin De Smedt, Hazem Wannous, Jean-Philippe Vandeborre, Joris Guerry, Bertrand Le Saux, and David Filliat. Shrec'17 track: 3d hand gesture recognition using a depth and skeletal dataset. In *3DOR-10th Eurographics Workshop on 3D Object Retrieval*, pages 1–6, 2017.

[4] Fanqing Lin, Connor Wilhelm, and Tony Martinez. Two-hand global 3d pose estimation using monocular rgb. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2373–2381, January 2021.

[5] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[6] Alberto Sabater, Iñigo Alonso, Luis Montesano, and Ana C Murillo. Domain and view-point agnostic hand action recognition. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[7] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017.

[8] Hongsong Wang and Liang Wang. Modeling temporal dynamics and spatial configurations of actions using two-stream recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 499–508, 2017.

[9] Fan Yang, Yang Wu, Sakriani Sakti, and Satoshi Nakamura. Make skeleton-based action recognition model smaller, faster and better. In *Proceedings of the ACM multimedia asia*, pages 1–6. 2019.

[10] Ziwei Yu, Linlin Yang, Shicheng Chen, and Angela Yao. Local and global point cloud reconstruction for 3d hand pose estimation. *arXiv preprint arXiv:2112.06389*, 2021.

[11] Songyang Zhang, Yang Yang, Jun Xiao, Xiaoming Liu, Yi Yang, Di Xie, and Yueting Zhuang. Fusing geometric features for skeleton-based action recognition using multi-layer lstm networks. *IEEE Transactions on Multimedia*, 20(9):2330–2343, 2018.