Development of a Bowling Game in Unity Based on Wearable Data Gloves and Spatial Positioning

Author: Chongyang Rao
Instructor: *Yu He*

Abstract:
This document meticulously records the design and implementation process of the Unity bowling game. The project created a virtual reality bowling game using the Unity engine, Manus data gloves, and ZED cameras, achieving an interactive experience for players in a virtual environment. The report covers the preparation of software and hardware, the application of visual positioning technology, the setup of the Unity environment, the implementation of physics simulation, and the final comprehensive testing.

1. Acquisition of glove pose information and reconstruction in Unity

a. Software preparation

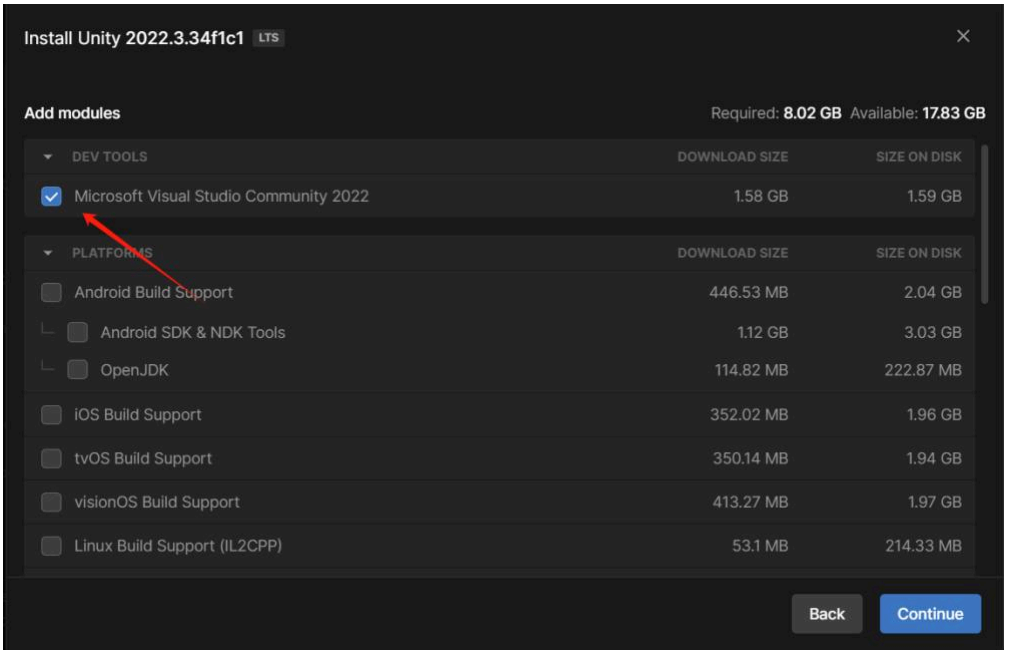
1. The latest version of the Manus Core program (regardless of the glove model, choose any glove model), download address

2. Manus Core plugins for Unity\

(Note: To download the program from the official website of Manus, you need "scientific Internet access" and register an account. The above website may be opened by not found without logging in to the account. If you use a domestic mailbox, you may fail to register. It is best to use a foreign mailbox.)

3. Unity Engine (you need to download Unity Hub first) download address

4. Visual Studio 2022 (You can choose to download the Unity engine as an Visu al Studio2022 when downloading it in Unity Hub) / or other available text editor


Install Unity 2022.3.34f1c1
LTS
Add modules
Required: 8.02 GB Available: 17.83 GB
DEV TOOLS
DOWNLOAD SIZE
SIZE ON DISK
Microsoft Visual Studio Community 2022
1.58 GB
1.59 GB
PLATFORMS
DOWNLOAD SIZE
SIZE ON DISK
Android Build Support
446.53 MB
2.04 GB
Android SDK & NDK Tools
1.12 GB
3.03 GB
OpenJDK
114.82 MB
222.87 MB
iOS Build Support
352.02 MB
1.96 GB
tvOS Build Support
350.14 MB
1.94 GB
visionOS Build Support
413.27 MB
1.97 GB
Linux Build Support (IL2CPP)
53.1 MB
214.33 MB
Back
Continue

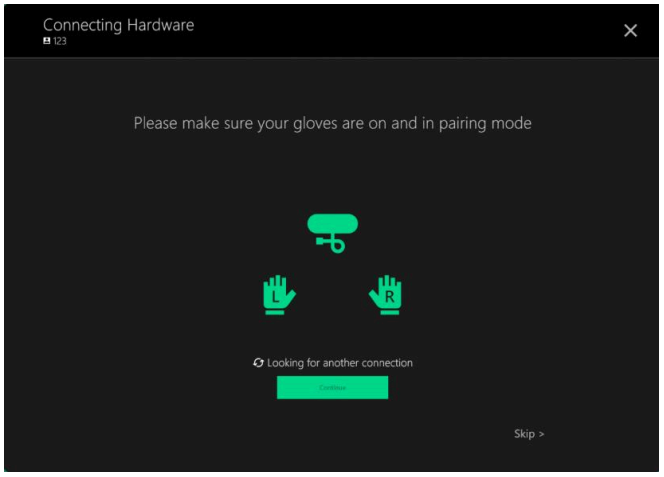
b. Hardware preparation

Manus data gloves and their encryption dongles

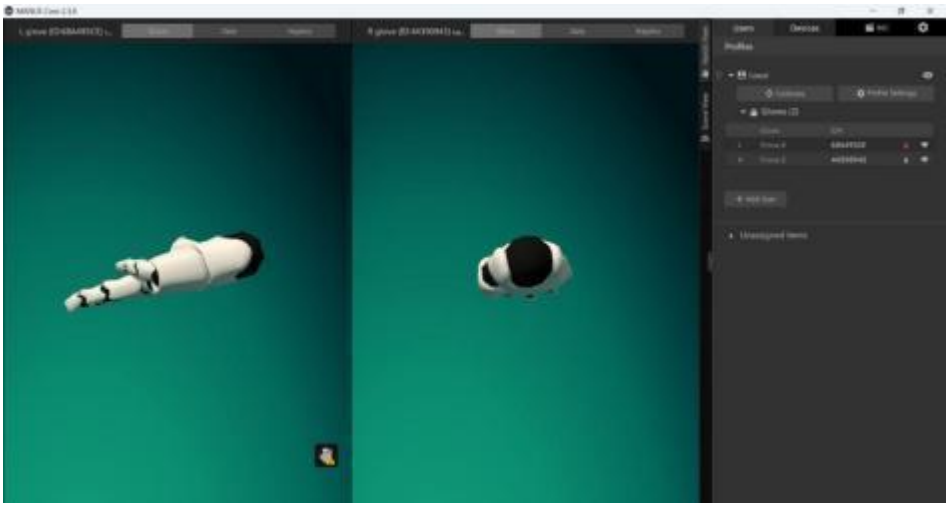
c. operating steps

1. Open Manus Core and name the project.

2. Insert Manus encryption dog into the usb port, and turn on the switch of Manus glove to the transceiver mode. At this time, the Manus Core page should be as shown in the figure below. Then follow the process to calibrate the glove and other operations.



3. After the process is calibrated (note that both hands should be calibrated), the following diagram page should appear, and the glove information can be transmitted to the Manus Core in real time.

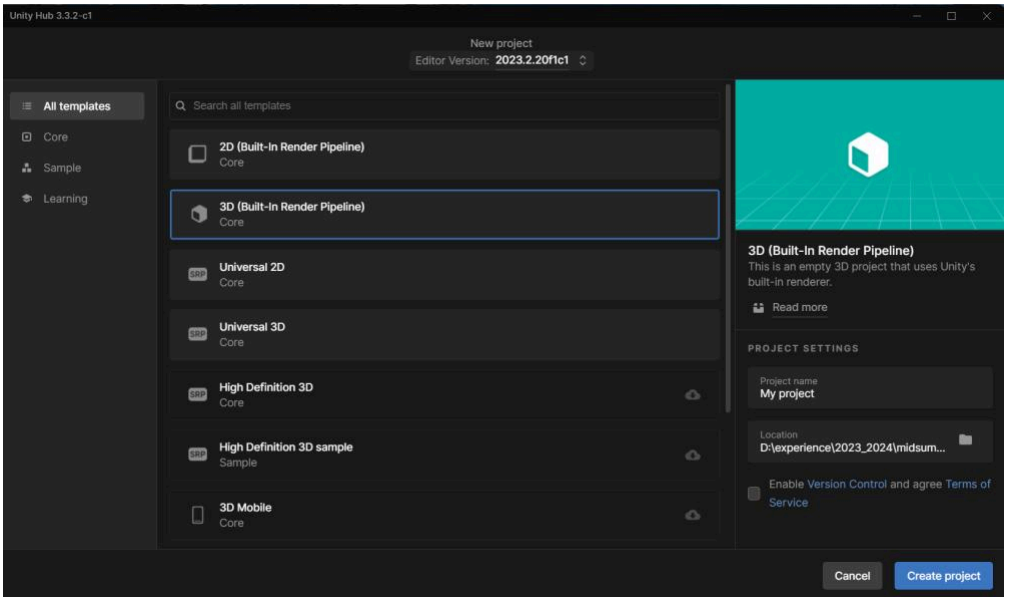


4. In the upper right corner of the page, find the word Device, click to enter this subpage and you can see the category name of the glove, ID
(SN code), link channel, glove firmware version, license level, etc. If the license version is Unlicensed, go to Step 5; if the license version is Core Partner, go to Step 6; if the license version is Core
Go to step 7, please.

5. If the license is Unlicensed and the glove is a Haptic tactile feedback glove, please find the CorePro information authorization file (file type: License) in the attachment, copy the file path, click on the folder icon to the right of the License license under the Manus Core→Device subdirectory, select the CorePro information authorization certificate, load it, and at this point, the certificate type should change to CorePro. If the certificate is invalid or if the glove is a regular Manus glove, please contact a technical support representative to obtain a new authorization file. Proceed to step 7.

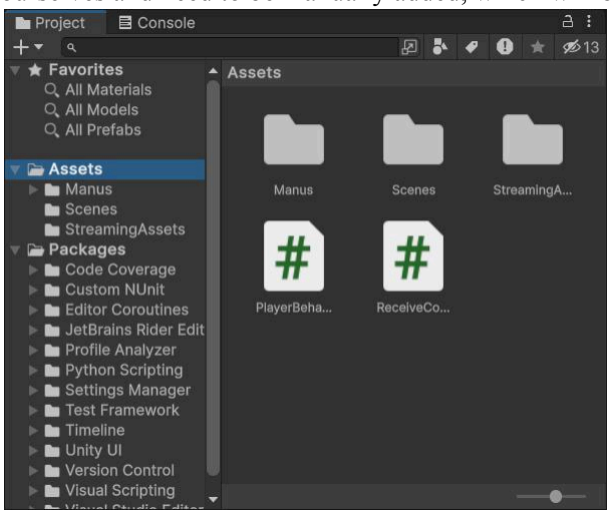
6. If the license is Core Partner, it indicates that the glove can only be read by the MVN Analyse software and transmitted to the Unity engine via data streaming.

7. If the license is Core Pro, this glove can directly transfer data to the Unity engine and open Unity Hub, Create a new project and select 3D Project. (If you need to install PlasticSCM Cloud disk, uncheck: Enable version management and agree to the policy terms. This step is only for some hosts that cannot install PlasticSCM Cloud disk. If you can install it normally, ignore the operation in parentheses.)

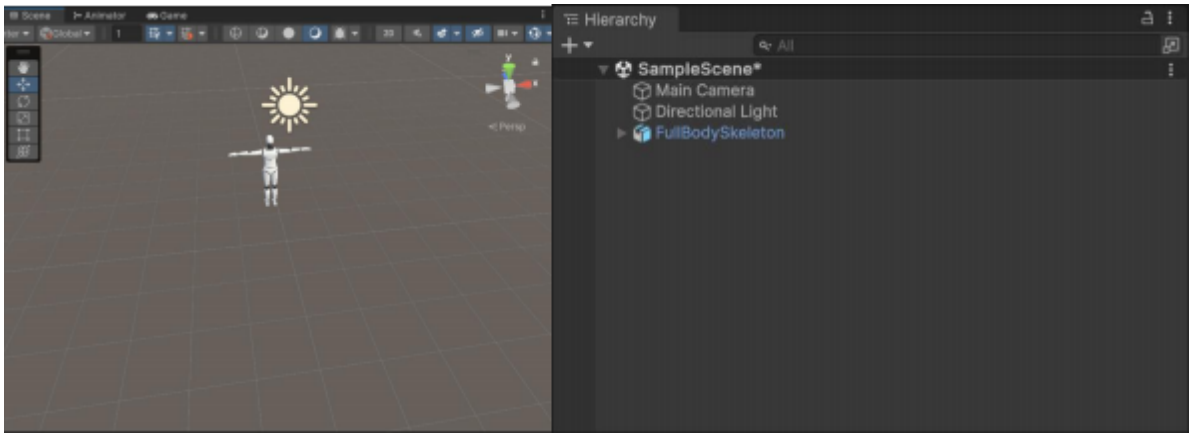


8. After entering the project editor page, find Asset s → Import Package → Custom Package and select it ManusCorePlugin File, open, click Import under the Unity page. After completion, you can go to the Assets file

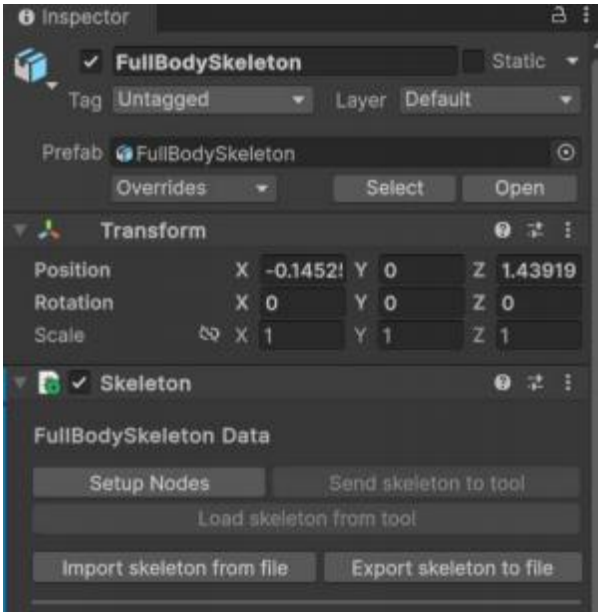
Below is the Manus folder, as shown in Figure (the two scripts in the figure are scripts written by ourselves and need to be manually added, which will be involved in the following steps).



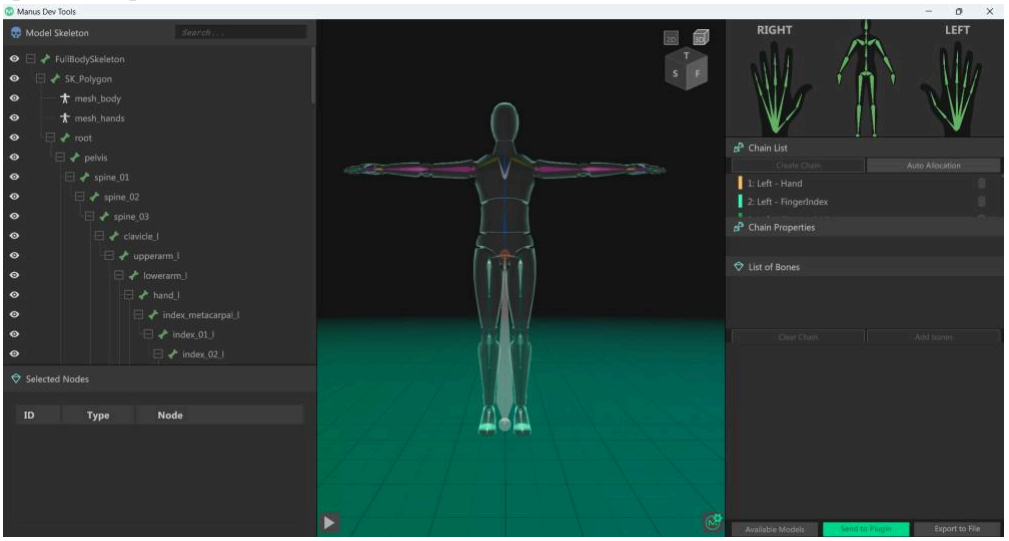
9. After completion, enter Manus→unity-plugin→Models in the folder (for example, ManusCorePlugin_v2.2.1, the location may be slightly different for different versions), drag the FullBodySkeleton model in the folder into the Unity scene, and FullBodySkeleton will appear in the Hierarchy after success.



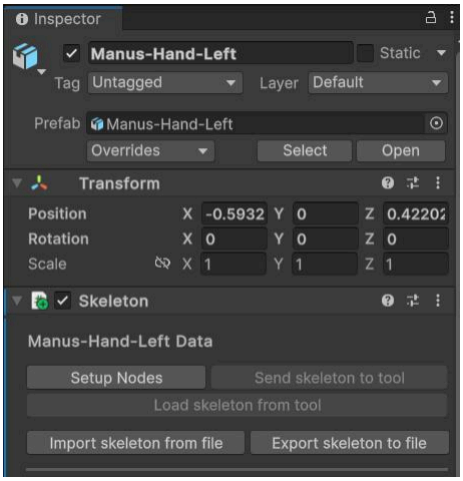
10. Click FullBodySkeleton in Hierarchy, add Add Component to the Inspector page, search for Skeleton, and left-click to add the script to the model.



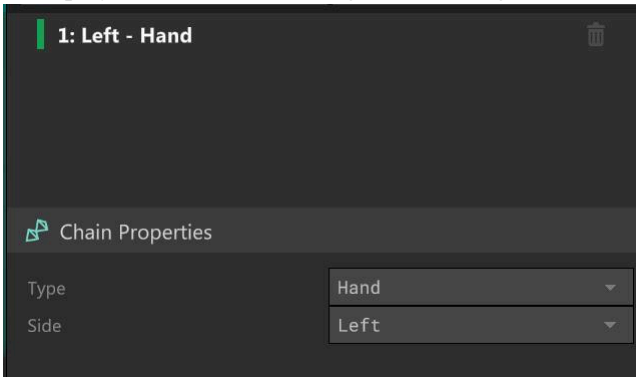
11. On the script operation page, click Setup Nodes→Send skeleton to tool to transfer to Manus Dev
The Tools page (if it does not open properly, you can find the Manus icon in the taskbar, right-click → More → Dev
Tools open), click the Available Models icon, select the recently imported model. After the skeleton model appears, click Auto Allocation. At this point, all hand nodes of the skeleton model should have been entered (you can also click Create Chain to create node information yourself). Then click Send to Plugin (the green button in the lower right corner of the image; due to resolution issues, you may need to drag the window or switch to full-screen display to find it). Back in Unity Editor, click Load skeleton from tool, then click Play. At this point, the hand information should be correctly transmitted to the Unity engine. (For detailed operations, please click here)

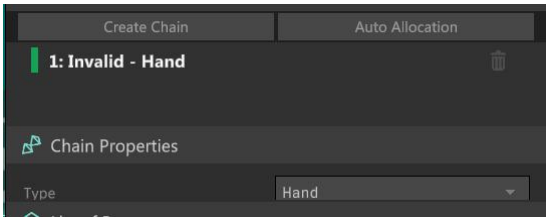


12. Since only the glove model is needed for this project, replace the human model with the glove model and repeat steps 10 and 11.
Enter Manus → unity-plugin → Models → ManusHand (with a period in the folder
Take ManusCorePlugin_v2.2.1 as an example, the location of different versions may be slightly different), select a glove model in the folder and drag it into the Unity scene (note the difference between left and right hands, take the left hand as an example).

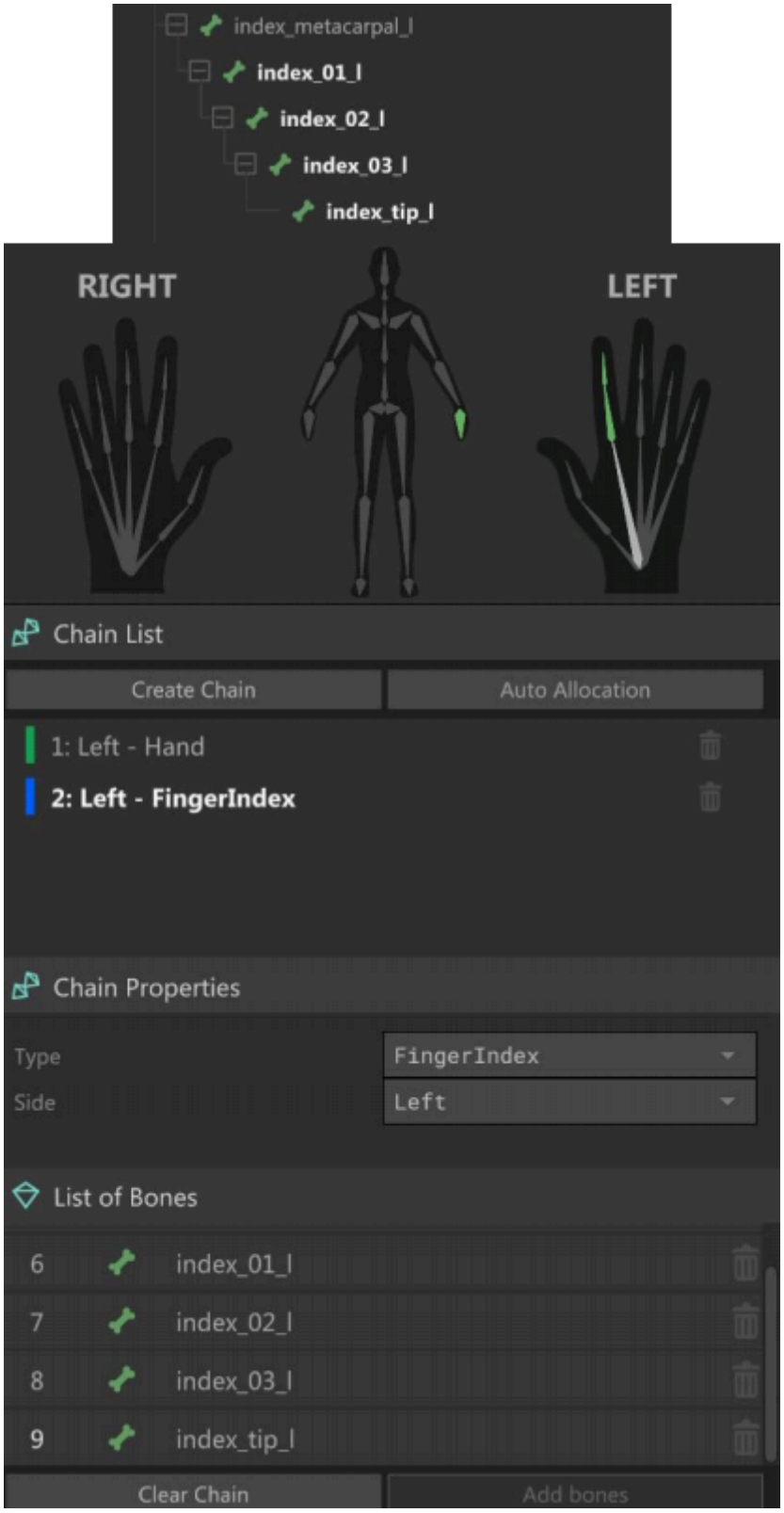


13. The automatic allocation (Auto Allocation) function cannot be used on the Manus Dev Tools page. You need to create your own node information and a node chain with one hand and five fingers. Click the hand_l node and select it in the Chain Lis t
Create Chain. Under Chain Prop erties, select type as hand and side as left. (It is possible that Side cannot be displayed as shown in the figure on the right. It is recommended to try another computer.)

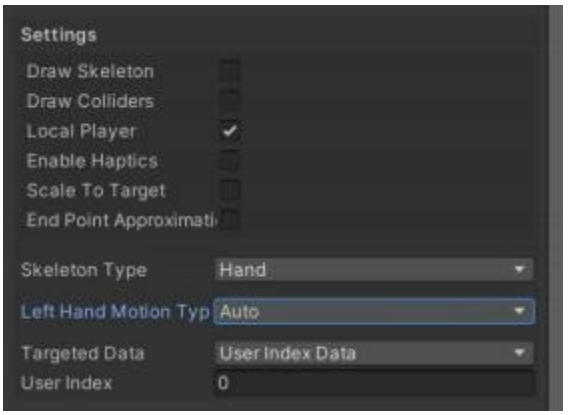




14. Click the index_metacarpal_left node, select Create Chain in Chain List, and Chain Properties Below, select type as FingerIndex and side as left. Select all remaining nodes under the modified node (you can start by clicking the first node and then hold down shift to click the last node), and click Add bones in List of Bones. The effect is shown in the figure, indicating that the left-hand index finger chain has been successfully created. Repeat this process with the remaining fingers
Build, and when its done, click Send to Plugin to go back to Unity, click Load skeleton from tool, and click play. Now you can control the Unity glove with a real glove.



The Settings under the inspector option may need to be adjusted as shown in the figure, or you can explore it yourself.


Settings
Draw Skeleton
Draw Colliders
Local Player
Enable Haptics
Scale To Target
End Point Approximati
Skeleton Type
Hand
Left Hand Motion Typ
Auto
Targeted Data
User Index Data
User Index
0

2. Visual positioning of the gloves spatial position

a. Software preparation

Official website link, csdn tutorial, it is recommended to do step by step according to the official website link, there will be no mistakes.

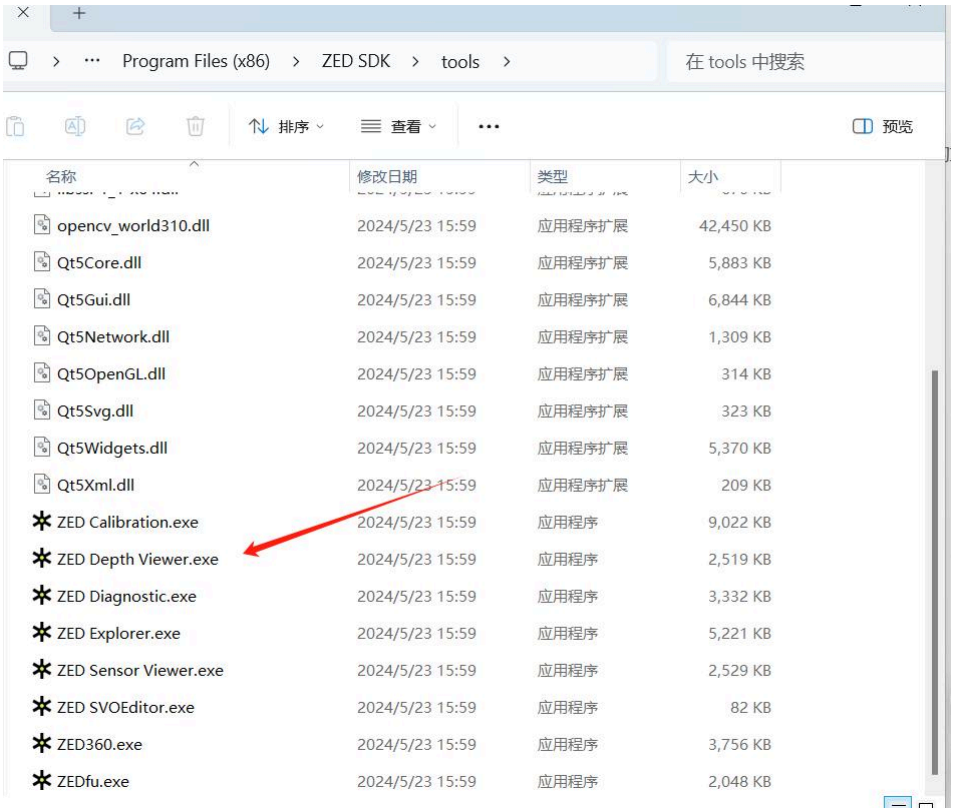
1. Cuda

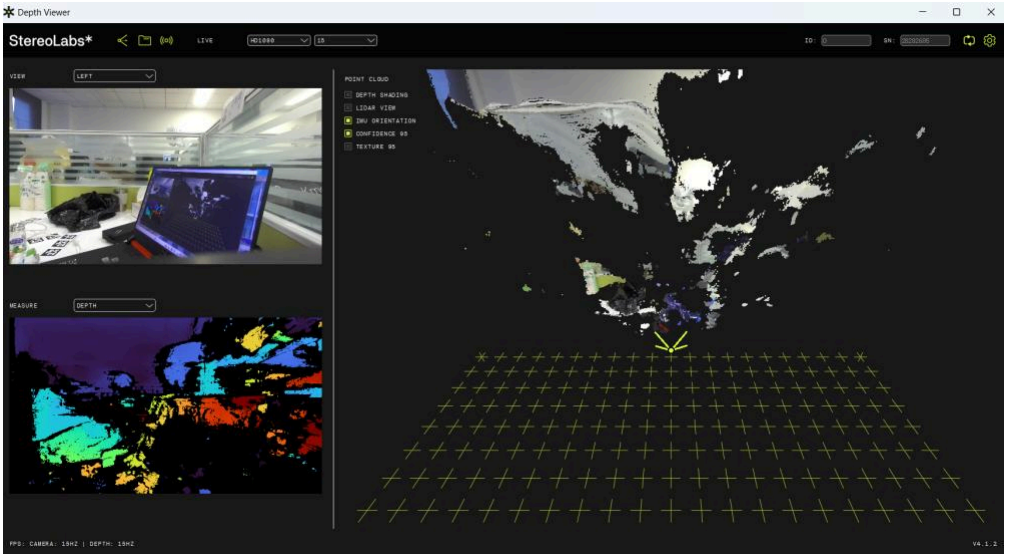
2. ZED SDK (note that it corresponds to the Cuda version)

3. ZED-Python-API（pyzed）

4. OpenCV

After downloading Cuda and ZED SDK, you can run the ZED built-in exe for testing, such as ZED Depth Viewer, exe, and it will be installed successfully if it can be opened normally.



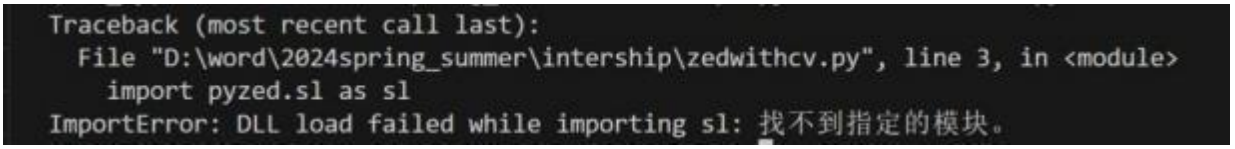


Possible difficulties

1.The computer does not have Nvidia and cannot drive Cuda. (Cuda has certain requirements for computer configuration, so light laptops may not be able to use it.)

The above error occurs when running Depth Viewer

2. Unable to find pyzed.sl



If the environment configuration is faulty, you need to run get_python_api.py and configure the corresponding environment in the virtual environment.

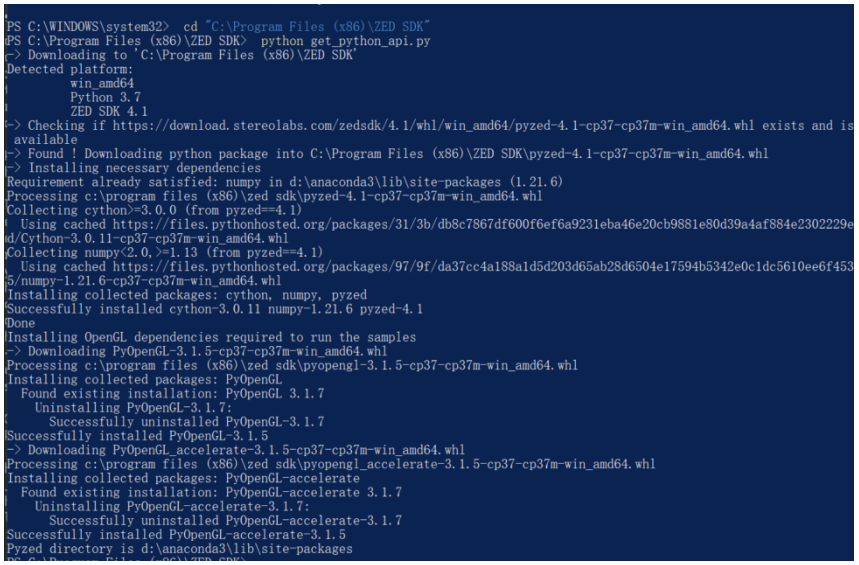
3. Under normal circumstances, the get_python_api.py script will automatically download the matching version of numpy. If the numpy version does not match the pyzed version, it will cause the problem that cv2 cannot recognize the image array.

4. It is also possible to find the library missing when running other scripts, often due to changes in the names of certain functions due to version updates. The authors environment is given here for reference.

```
#
# Name                    Version                   Build  Channel
ca-certificates           2024.7.2             haa95532_0    defaults
certifi                   2024.7.4                  pypi_0    pypi
charset-normalizer        3.3.2                     pypi_0    pypi
cython                    3.0.10                    pypi_0    pypi
idna                      3.7                       pypi_0    pypi
libffi                    3.4.4                hd77b12b_1    defaults
numpy                     1.24.4                    pypi_0    pypi
opencv-contrib-python     4.10.0.84                 pypi_0    pypi
opencv-python             4.10.0.84                 pypi_0    pypi
openssl                   3.0.14               h827c3e9_0    defaults
pip                       24.0              py38haa95532_0    defaults
pyopengl                  3.1.5                     pypi_0    pypi
pyopengl-accelerate       3.1.5                     pypi_0    pypi
python                    3.8.19               h1aa4202_0    defaults
pyzed                     4.1                       pypi_0    pypi
requests                  2.32.3                    pypi_0    pypi
setuptools                69.5.1            py38haa95532_0    defaults
sqlite                    3.45.3               h2bbff1b_0    defaults
urllib3                   2.2.2                     pypi_0    pypi
vc                        14.2                 h2eaa2aa_4    defaults
vs2015_runtime            14.29.30133          h43f2093_4    defaults
watchdog                  4.0.1                     pypi_0    pypi
wheel                     0.43.0            py38haa95532_0    defaults
```

5. Permission problem encountered when running get_python_api.py script

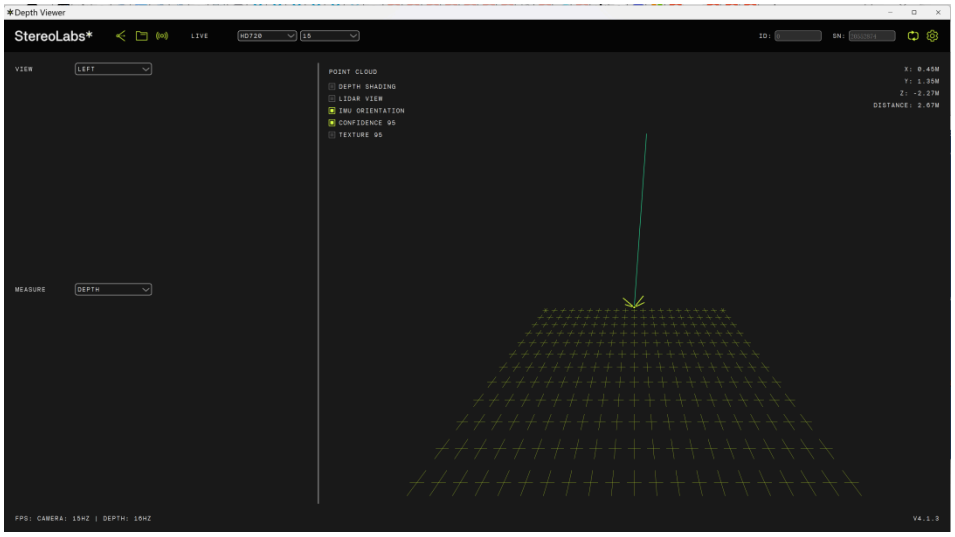
Run PowerShell as an administrator and install dependencies using the--user option



6. Compatibility issues were encountered when installing the ZED SDK Python binding

Make sure Python version is compatible with ZED SDK and use the correct whl file for installation.

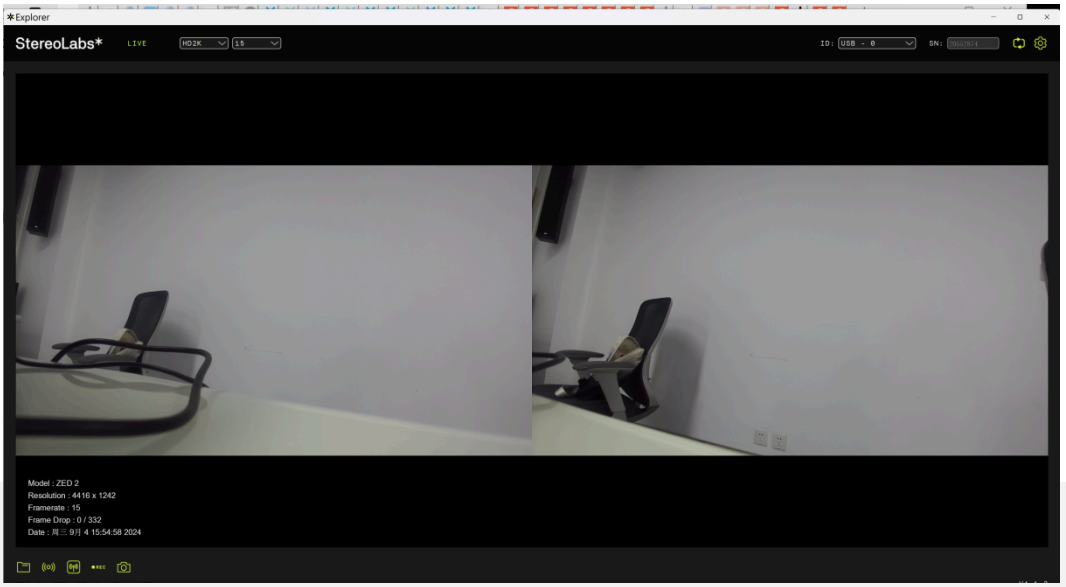
7.ZED cameras Depth Viewer cannot display depth images



Rx:

1. Check that the camera connection, USB port, driver and firmware are up to date, confirm that the SDK is installed correctly, and try to run Depth Viewer with administrator privileges.

2. Check ZED EXplorer, if the picture can be displayed normally, it indicates that the camera is running normally, as shown in the figure



Note: It is best to use the python environment of conda virtual, otherwise it is easy to encounter environmental configuration problems, especially pay attention to version matching problems. Use the statement conda activate zed_env to activate

b. Hardware preparation

1. Computers that can use Cuda are available.
2. The Zed2 camera.

Auxiliary: csdn tutorial, Aruco generator

c. Detection script

1.Connect the test script test.py

```
PS C:\Users\Lenovo\Documents\ZED> conda activate zed_env
PS C:\Users\Lenovo\Documents\ZED> python test.py
Hello! This is my serial number: 29458470
```

test.py

```
import pyzed.sl as sl

def main():
Create the camera
zed = sl.Camera()

Create the InitParameters object and set configuration parameters
init_params = sl.InitParameters()
init_params.sdk_verbose = False

Open the camera
err = zed.open(init_params)
if err != sl.ERROR_CODE.SUCCESS:
exit(1)
```

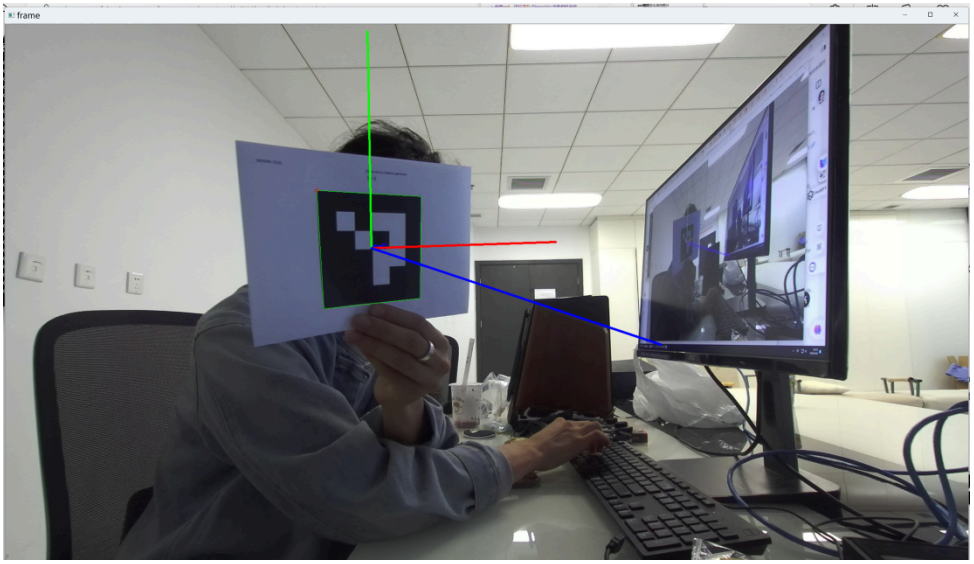
```
Get camera parameters (ZED serial number)
zed_serial = zed.get_camera_information().serial_number
print("Hello! This is my serial number: {0}".format(zed_serial))

# Turn off the camera
zed.close()

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

2. Test depth information

```
import cv2
import numpy as np
import pyzed.sl as sl

# Set ArUco dictionary and detection parameters
aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_6X6_250)
parameters = cv2.aruco.DetectorParameters()

# Camera calibration parameters (adjust according to actual setup)
camera_matrix = np.array([[1000, 0, 320], [0, 1000, 240], [0, 0, 1]], dtype=float)
dist_coeffs = np.zeros((5, 1))

# Initialize the ZED camera
zed = sl.Camera()
init_params = sl.InitParameters()
init_params.camera_resolution = sl.RESOLUTION.HD2K  # Set resolution
init_params.camera_fps = 30  # Set frame rate
init_params.coordinate_units = sl.UNIT.METER  # Set unit to meters

# Open the camera
if zed.open(init_params) != sl.ERROR_CODE.SUCCESS:
    print("Failed to open ZED camera")
    exit(1)

runtime_parameters = sl.RuntimeParameters()
image_zed = sl.Mat()

# Start capturing frames
while True:
    if zed.grab(runtime_parameters) == sl.ERROR_CODE.SUCCESS:
        # Retrieve left image
        zed.retrieve_image(image_zed, sl.VIEW.LEFT)
        frame = image_zed.get_data()

        # Check if frame is valid
        if frame is None or frame.size == 0:
            continue

        # Convert frame to grayscale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGRA2GRAY)

        # Detect ArUco markers
        corners, ids, rejectedImgPoints = cv2.aruco.detectMarkers(gray, aruco_dict,
```

```
parameters=parameters)

        if ids is not None and len(ids) > 0:
            print(f"Detected markers: {len(ids)}")
            print(f"Marker IDs: {ids}")
            print(f"Marker corners: {corners}")

            # Ensure frame is 3-channel (BGR) for drawing
            if frame.ndim == 3 and frame.shape[2] == 4:
                frame = cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)

            # Draw detected markers
            frame = cv2.aruco.drawDetectedMarkers(frame, corners, ids)

            # Estimate the pose of each marker
            rvecs, tvecs, _ = cv2.aruco.estimatePoseSingleMarkers(corners, 0.05, camera_matrix,
dist_coeffs)
            for rvec, tvec in zip(rvecs, tvecs):
                # Draw axis on each detected marker
                frame = cv2.drawFrameAxes(frame, camera_matrix, dist_coeffs, rvec, tvec, 0.1)

                # Print marker position (x, y, z)
                print(f"Marker Position: x={tvec[0][0]:.2f}, y={tvec[0][1]:.2f}, z={tvec[0][2]:.2f}")
        else:
            print("No ArUco markers detected")

        # Display the resulting frame
        cv2.imshow('frame', frame)

        # Exit the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Release resources
zed.close()
cv2.destroyAllWindows()
```

The effect is as follows:



```
PS C:\Users\Lenovo> cd "C:\Users\Lenovo\Documents\ZED"
PS C:\Users\Lenovo\Documents\ZED> conda activate zed_env
PS C:\Users\Lenovo\Documents\ZED> python send_coordinates.py
[2024-09-10 08:01:34 UTC][ZED][INFO] Logging level INFO
[2024-09-10 08:01:35 UTC][ZED][INFO] [Init]  Depth mode: PERFORMANCE
[2024-09-10 08:01:35 UTC][ZED][INFO] [Init]  Requested FPS (30fps) is too high for the current
resolution... Using 15 fps instead.
[2024-09-10 08:01:36 UTC][ZED][INFO] [Init]  Camera successfully opened.
[2024-09-10 08:01:36 UTC][ZED][INFO] [Init]  Camera FW version: 1523
[2024-09-10 08:01:36 UTC][ZED][INFO] [Init]  Video mode: HD2K@15
[2024-09-10 08:01:36 UTC][ZED][INFO] [Init]  Serial Number: S/N 29458470
Marker Position: x=0.41, y=0.13, z=0.43
Sent: 0.4073716646717943,0.12783142985813487,0.43296304078725156
Marker Position: x=0.41, y=0.13, z=0.44
Sent: 0.4119113034380357,0.12569641693950206,0.43571187710663783
Marker Position: x=0.42, y=0.12, z=0.44
Sent: 0.4173012710988372,0.12414453617177759,0.43946544467862897
Marker Position: x=4.27, y=2.92, z=3.18
Sent: 4.2669071101255955,2.9168010479339124,3.182659219636726
Marker Position: x=0.42, y=0.12, z=0.44
Sent: 0.41982809969805684,0.1216669967100769,0.44004273805628624
Marker Position: x=0.41, y=0.12, z=0.43
Sent: 0.4108820922772146,0.11729391881877324,0.4287512331993532
Marker Position: x=0.42, y=0.12, z=0.44
Sent: 0.4237535686845115,0.1200503166704485 8,0.4398745165429735
```

d. Test the communication script between Python and Unity

If the depth information detection script runs successfully, you can use SendToUnity.py to pass the depth information into Unity.
To receive messages from Python, you need to add the ReceiveCoordinates.cs script under the Unity glove model (or other Game Object). The script content can be found in Section 3. You must first click play in Unity and then run the Python script to establish a normal connection.

It is worth noting that when receiving depth camera data in Unity, there is a chance of connection failure using the TCP protocol. The error message is "[WinError 10061] cannot connect due to active rejection by the target computer." The solution is to change the communication protocol from TCP to UDP. When sending data, you can directly call the sendto method, without using the TcpListener to listen to the port and receive data. Improve the success rate.

send_to_Unity.py (Added adjustable monitoring window on the basis of send_ c oordinates.py script)

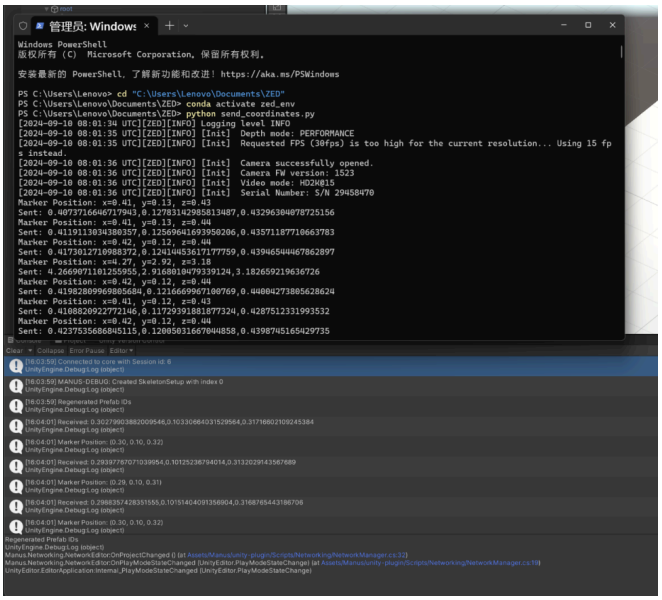
```
import cv2

import numpy as np

import pyzed.sl as sl

import socket

import time



def initialize_camera():

    zed = sl.Camera()

    init_params = sl.InitParameters()

    init_params.camera_resolution = sl.RESOLUTION.HD2K

    init_params.camera_fps = 30

    init_params.coordinate_units = sl.UNIT.METER
```

```
    status = zed.open(init_params)
    if status != sl.ERROR_CODE.SUCCESS:
Print ("ZED camera failed to open")
        exit(1)
    return zed

def main():
        zed = initialize_camera()
#6X6ARUCO dictionary
    aruco_dict = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_6X6_250)
    parameters = cv2.aruco.DetectorParameters_create()
    camera_matrix = np.array([[1000, 0, 320], [0, 1000, 240], [0, 0, 1]], dtype=float)
    dist_coeffs = np.zeros((5, 1))

    image_zed = sl.Mat()
    runtime_parameters = sl.RuntimeParameters()

Set up UDP communication
    host = localhost
    port = 49672
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

Create a resizable window
    cv2.namedWindow(frame, cv2.WINDOW_NORMAL)
Set the initial size of the window
    cv2.resizeWindow(frame, 1024, 768)

    try:
        while True:
            if zed.grab(runtime_parameters) == sl.ERROR_CODE.SUCCESS:
                zed.retrieve_image(image_zed, sl.VIEW.LEFT)
```

```
            frame = image_zed.get_data()
            if frame is None or frame.size == 0:
                continue
            gray = cv2.cvtColor(frame, cv2.COLOR_BGRA2GRAY)
            corners, ids, rejectedImgPoints = cv2.aruco.detectMarkers(gray, aruco_dict,
parameters=parameters)
            if ids is not None and len(ids) > 0:
                                if frame.ndim == 3 and frame.shape[2] == 4:
                    frame = cv2.cvtColor(frame, cv2.COLOR_BGRA2BGR)
                frame = cv2.aruco.drawDetectedMarkers(frame, corners, ids)
                rvecs, tvecs, _ = cv2.aruco.estimatePoseSingleMarkers(corners, 0.05, camera_matrix,
dist_coeffs)
                for rvec, tvec in zip(rvecs, tvecs):
                    frame = cv2.aruco.drawAxis(frame, camera_matrix, dist_coeffs, rvec, tvec, 0.1)
                    x, y, z = tvec[0][0], tvec[0][1], tvec[0][2]
                    print(f"Marker Position: x={x:.2f}, y={y:.2f}, z={z:.2f}")

Send coordinate information
                    message = f"{x},{y},{z}"
                    s.sendto(message.encode(), (host, port))
                    print(f"Sent: {message}")
            else:
Print ("No ArUco tag detected")
            cv2.imshow(frame, frame)
            if cv2.waitKey(1) & 0xFF == ord(q):
                break
    finally:
        zed.close()
        cv2.destroyAllWindows()
        s.close()

if __name__ == "__main__":
```

```
main()
```

The effect of successful information transmission is shown in the figure



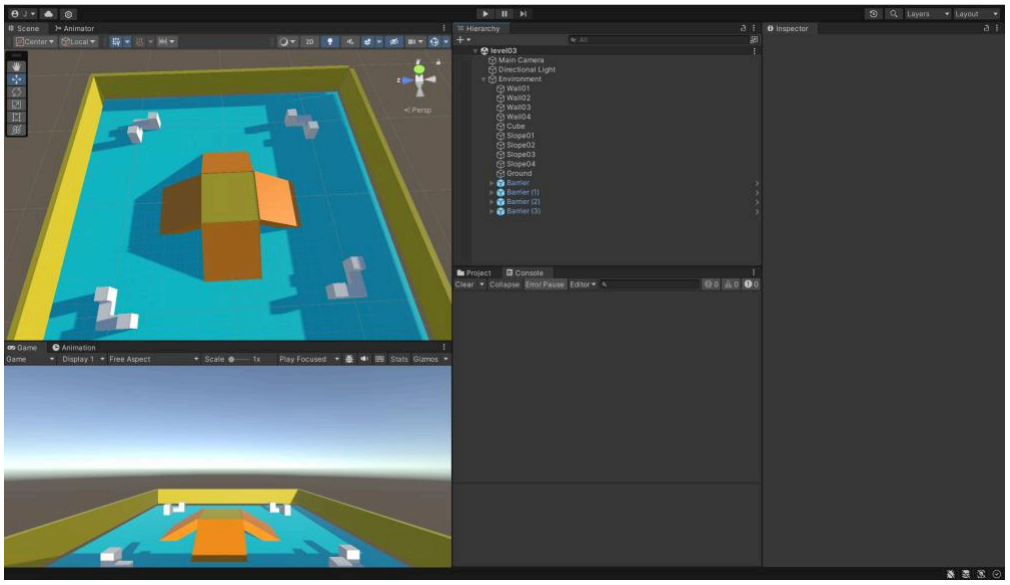
3. Comprehensive test and application scenarios

a.Visual positioning technology application

First, you need to open unity and create a simple scene. If you are not familiar with unity and C#, you can watch the video in the following link,
You can also search for relevant information on the Internet. [C# in Uni ty Zero-based Introduction]
0.0_course introduction_Bilibili_ bilibili

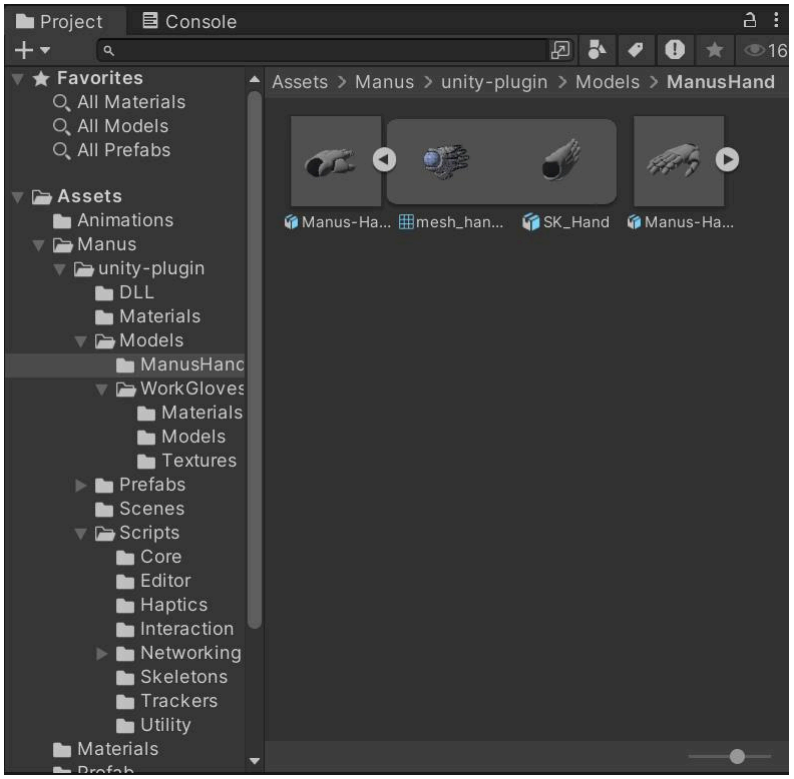
https://www.bilibili.com/video/BV1T14y177CN/?spm_id_from=333.788&vd_source=5 53abba2b0ba644b2d2afda00f707078

The scenario we built is as follows:

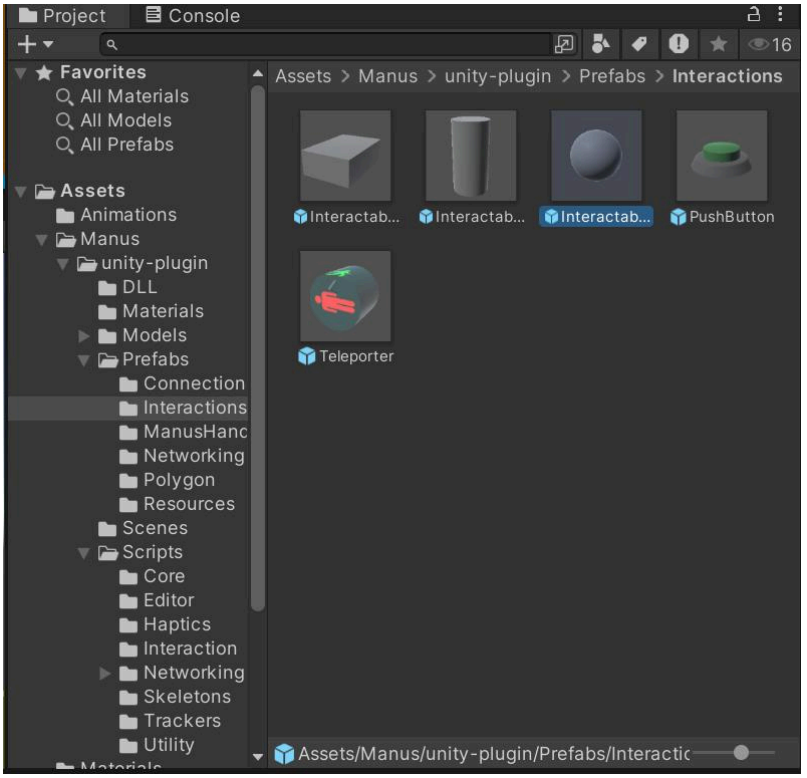


Then refer to section 1.c to import the model into unity so that the hand motion animation can appear in unity. The specific operation is as follows (firstly, you need to import Man usCorePlugin according to section 1.c):

1. First, find the individual hand model under the path in the picture and add it to the scene. Here we take Manus-Hand-Left as an example;



2. Similar to the previous step, import a ball for grabbing. Here we use InteractableSphere as an example, but you can also do it
Design by yourself;

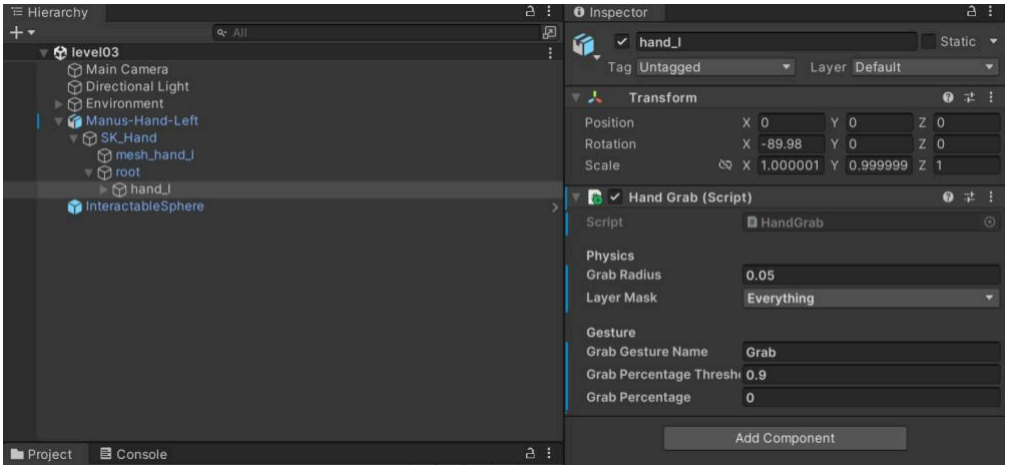


3. Select Manus-Hand-Left under the Hierarchy directory, and then select Add in the Inspector interface

Component Search Skeleton and Rigid Body, and add the two scripts into the model. Refer to section 1 for specific Settings;

4. Then, under the Hierarchy directory, expand the Manus-Hand-Left subdirectory and add a Hand Grab script in hand_I. After completing this step, we can click the play button to test whether the

glove can move and achieve the grabbing function in Unity; (since the camera has not yet been embedded, you need to manually move objects or small balls to achieve positioning ability)



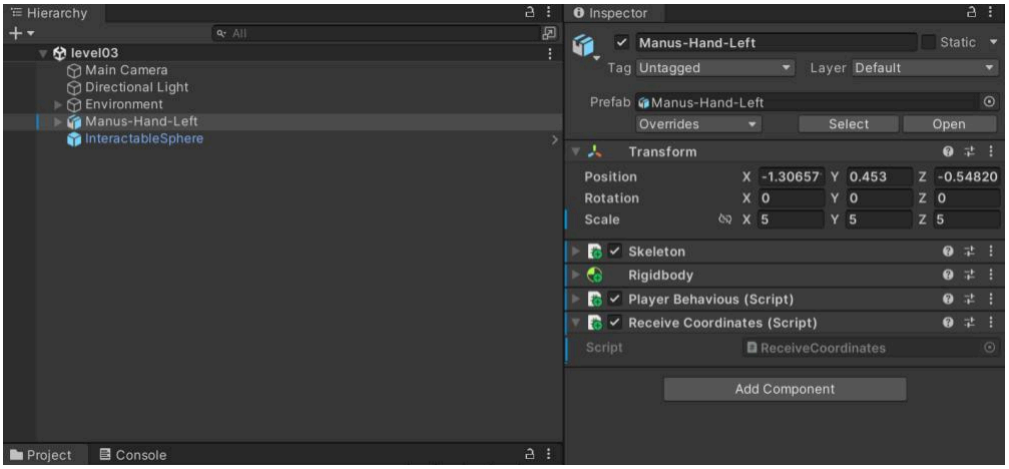
5.After achieving the aforementioned functions, we next consider using visual positioning to drive the gloves movement. We will attach ArUco codes to the gloves and use cameras to recognize these ArUco codes for positioning. The code for the camera to achieve distance measurement is detailed in Section Two. This script implements a simple UDP data receiver, which receives and processes data sent over the network in Unity, and uses this data to move the hand models Transform.

UDP Recever 1 .cs

```
using UnityEngine;
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Collections.Generic;

public class UDPReceiver : MonoBehaviour
{
    private UdpClient udpClient;
   private Thread receiveThread;
   public int port = 49672;
public Transform handTransform; // Used to specify the Transform of the hand model

   private Queue<Action> mainThreadQueue = new Queue<Action>();

   void Start()
   {
      udpClient = new UdpClient(port);
      receiveThread = new Thread(new ThreadStart(ReceiveData));
      receiveThread.IsBackground = true;
      receiveThread.Start();
   }

   void Update()
   {
      lock (mainThreadQueue)
      {
         while (mainThreadQueue.Count > 0)
         {
            mainThreadQueue.Dequeue().Invoke();
```

```
            }
        }
    }

    private void ReceiveData()
    {
        IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, port);
        try
        {
            while (true)
            {
                byte[] data = udpClient.Receive(ref remoteEndPoint);
                string message = Encoding.UTF8.GetString(data);
                Debug.Log("Received: " + message);
                ProcessData(message);
            }
        }
        catch (Exception e)
        {
            Debug.LogError(e.ToString());
        }
    }

        private void ProcessData(string message)
    {
        string[] coordinates = message.Split(,);
        if (coordinates.Length == 3)
        {
            if (float.TryParse(coordinates[0], out float x) &&
                float.TryParse(coordinates[1], out float y) &&
                float.TryParse(coordinates[2], out float z))
            {
                Vector3 position = new Vector3(x, y, z);
                Debug.Log("Marker Position: " + position);
                QueueMainThreadAction(() => MoveHand(position));
            }
        }
    }

    private void MoveHand(Vector3 position)
    {
// Here update the position of the hand model
        if (handTransform != null)
        {
// Coordinates can be scaled or transformed appropriately to fit the world size of Unity
            handTransform.position = position;
        }
    }

    private void QueueMainThreadAction(Action action)
    {
        lock (mainThreadQueue)
        {
            mainThreadQueue.Enqueue(action);
        }
    }

    void OnApplicationQuit()
    {
        if (receiveThread != null)
        {
            receiveThread.Abort();
        }
        udpClient.Close();
```

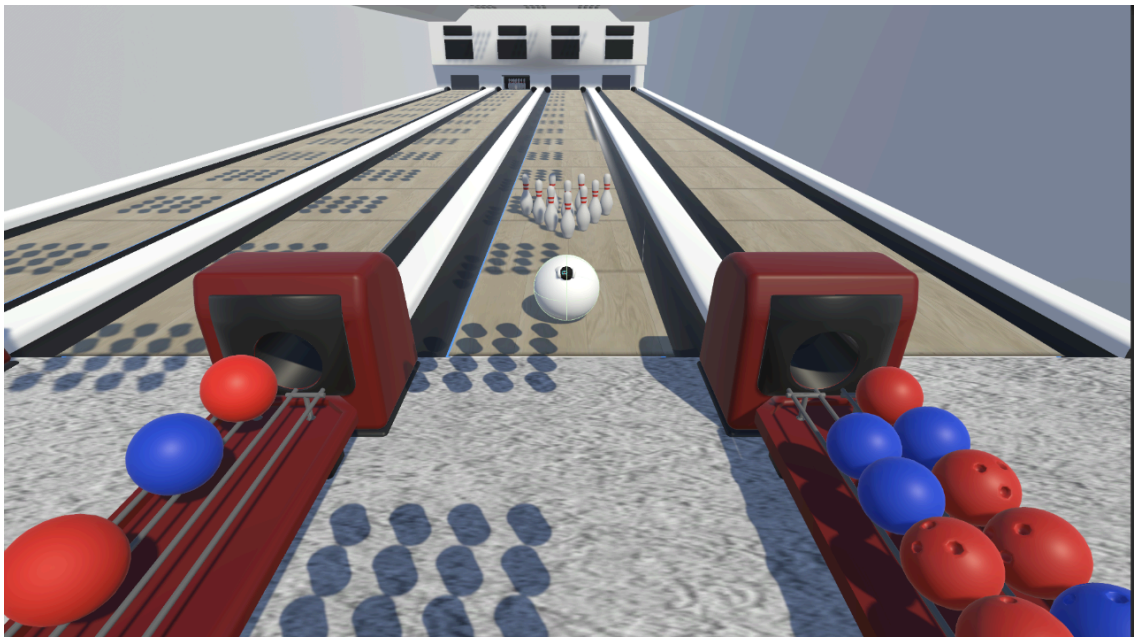
```
    }
}
```



6. (Optional) Add collision detection for fingers. You need to open the root folder under "Manus-Hand-Left." In the expanded branch, you should see the nodes of each finger joint. Then, you can add colliders to each phalanx (for specific operations on adding colliders, refer to the previous Unity tutorial link). However, this feature can easily cause the ball to bounce off when the grabbing function is not yet implemented, which needs further improvement.

7. Run Unity, then run SendToUnity.py, and you can see that when the glove moves, the ArUco code on the back of the hand is detected, so the hand in Unity also moves.

b.Unity interactive bowling simulation



[Final scene setup]

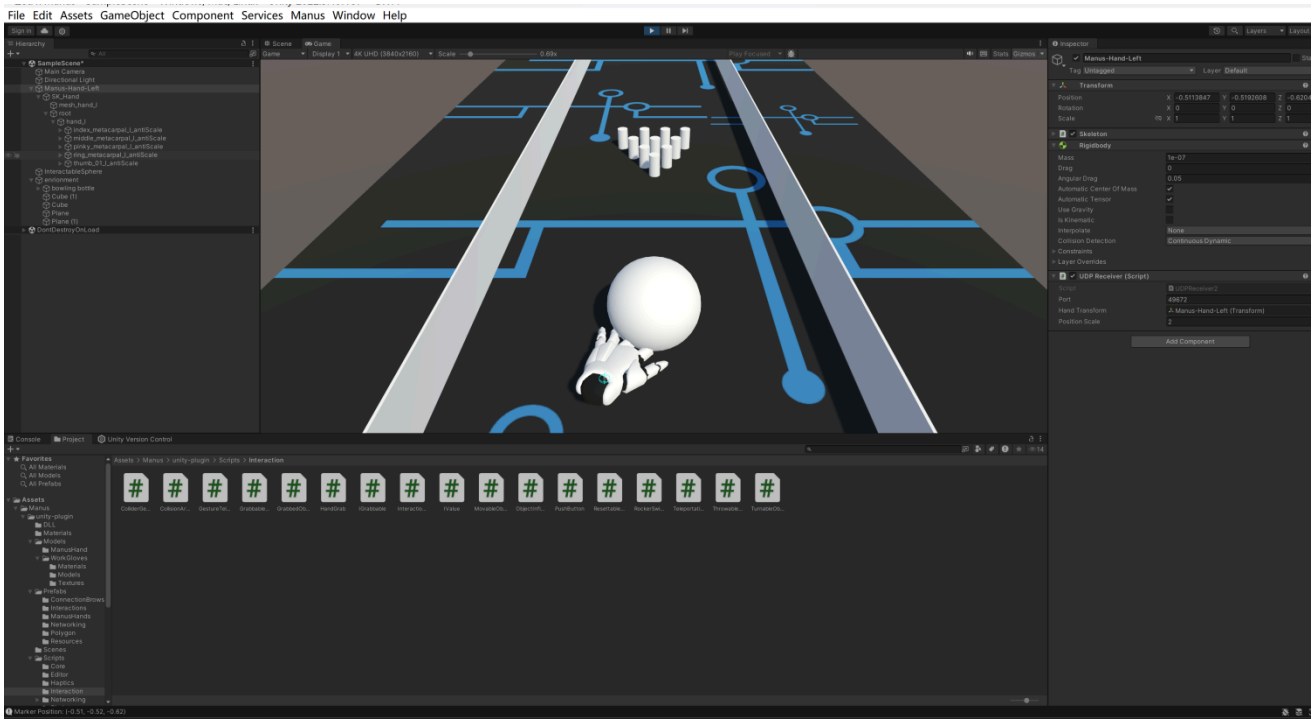
After the successful communication between unity and zed camera, we can use motion capture gloves and depth camera to complete a simple interactive bowling game on this basis

Some tutorials are referenced:

[Yang, PICO Application Development] Make a simple and rough VR bowling game!]
https://www.bilibili.com/video/BV1xd4y1F7RS/?share_source=copy_web&vd_source=07dc9757c50395fee3d2a54261ae1870

[Unity Zero Foundation Series-3.3 Bowling Game Project Development 2-Add Collision Detection to Implement Bonus Function]
https://www.bilibili.com/video/BV1KnsmewEWA/?share_source=copy_web&vd_source=07dc9757c50395fee3d2a54261ae1870

1. Create a simple bowling ball and lane model and a test scene for the pinball in Unity.



In Unity, create a plane as a ball path by GameObject-> 3D Object-> Plane and set its position
Similarly, create a sphere as a bowling ball by GameObject-> 3D Object-> Sphere and set its position
Create a cylinder as a ball pot by GameObject-> 3D Object-> cylinder, and copy it to place it

Add material (optional):
Add textures to the ball and bowling ball to beautify the scene. You can use existing resources placed in the projects Assets directory and drag them onto the corresponding game objects

2. The test was conducted using gesture control with the Manus data glove and visual positioning with the ZED camera.

According to the test results, it was found that due to the limited field of view of the ZED camera, the position of the hand model needed to be adjusted more flexibly. For this reason, in the updated script UDPReceiver3.cs, we introduced a speed amplification factor and a default offset. These parameters allow setting the initial position offset and motion speed amplification of the hand model in Unity, ensuring that the model starts closer to the front and can move more freely.

The updated script is based on UDPReceiver1.cs and adds the following features:

1. Speed amplification factor: used to adjust the speed of the received position data.
2. Initial position offset: Add it to the position data received through UDP to adjust the initial position of the model.

With these changes, users can easily set the initial position and movement speed of the hand model in the Unity editor to better adapt to the field of view of the ZED camera.

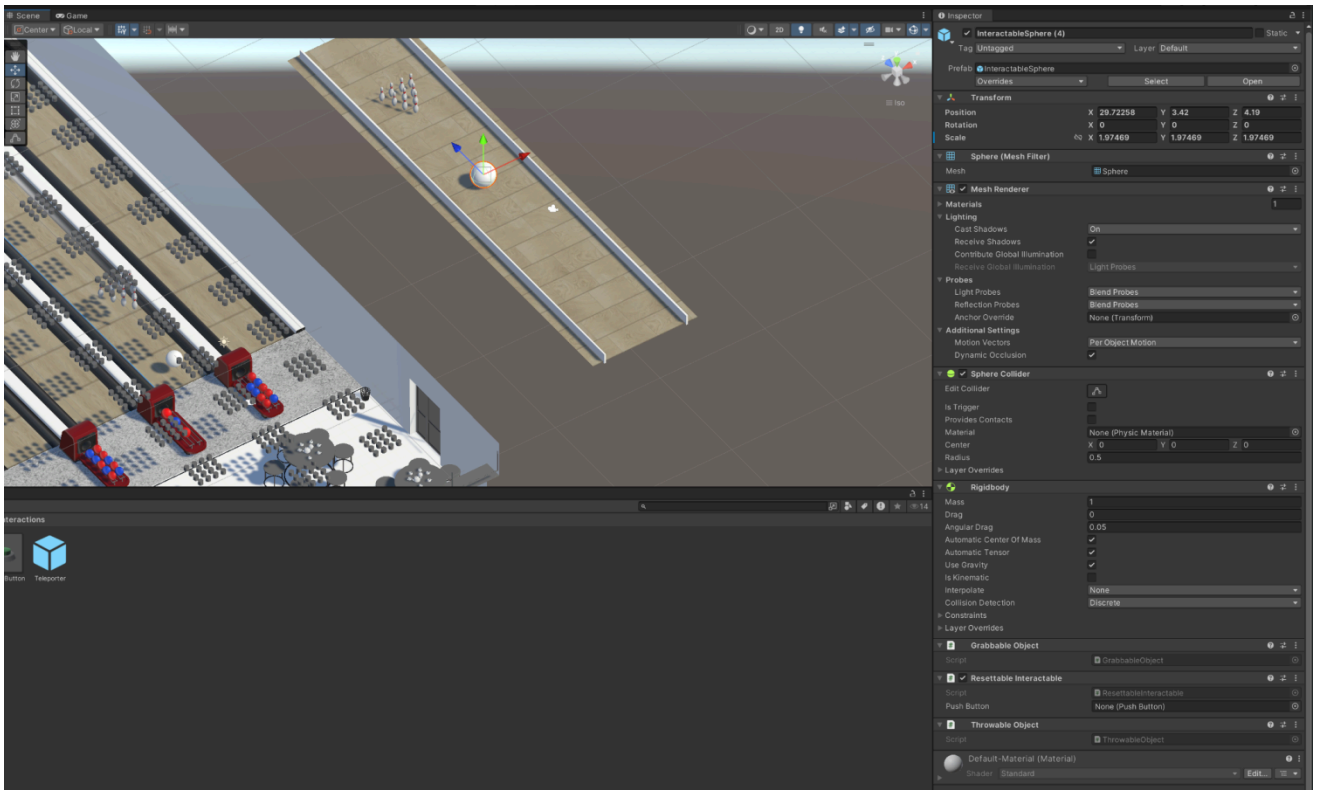
UDP Recever3.cs

```
using UnityEngine;
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Collections.Generic;

public class UDPReceiver : MonoBehaviour
{
   private UdpClient udpClient;
   private Thread receiveThread;
   public int port = 49672; // You can try changing this port
public Transform handTransform; // Used to specify the Transform of the hand model
public float positionScale = 2.0f; // The amplification factor used to adjust the motion speed
public Vector3 positionOffset = new Vector3(0,0,0.5f); // Initial position offset

   private Queue<Action> mainThreadQueue = new Queue<Action>();

   void Start()
   {
      if (handTransform == null)
      {
Debug.LogError ("handTransform is not bound. Please bind the hand models Transform in the Unity editor!")
         return;
      }

      try
      {
         udpClient = new UdpClient(port);
         receiveThread = new Thread(new ThreadStart(ReceiveData));
         receiveThread.IsBackground = true;
         receiveThread.Start();
      }
      catch (SocketException ex)
      {
         Debug.LogError($"SocketException: {ex.Message}");
         Debug.LogError($"Socket Error Code: {ex.SocketErrorCode}");
         Debug.LogError($"Stack Trace: {ex.StackTrace}");
      }
      catch (Exception ex)
```

```
        {
            Debug.LogError($"Exception: {ex.Message}");
                Debug.LogError($"Stack Trace: {ex.StackTrace}");
        }
    }

    void Update()
    {
        lock (mainThreadQueue)
        {
            while (mainThreadQueue.Count > 0)
            {
                mainThreadQueue.Dequeue().Invoke();
            }
        }
    }

    private void ReceiveData()
    {
        IPEndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, port);
        try
        {
            while (true)
            {
                byte[] data = udpClient.Receive(ref remoteEndPoint);
                string message = Encoding.UTF8.GetString(data);
                Debug.Log("Received: " + message);
                ProcessData(message);
            }
        }
        catch (Exception e)
        {
            Debug.LogError(e.ToString());
        }
    }

    private void ProcessData(string message)
    {
        string[] coordinates = message.Split(,);
        if (coordinates.Length == 3)
        {
            if (float.TryParse(coordinates[0], out float x) &&
                float.TryParse(coordinates[1], out float y) &&
                float.TryParse(coordinates[2], out float z))
            {
                                Vector3 position = new Vector3(-x * positionScale, -y * positionScale, -z * positionScale) +
positionOffset;
                Debug.Log($"Processed Position: {position} (Offset: {positionOffset})");
                QueueMainThreadAction(() => MoveHand(position));
            }
            else
            {
                Debug.LogError("Failed to parse coordinates from message: " + message);
            }
        }
        else
        {
            Debug.LogError("Received message with incorrect format: " + message);
        }
    }

    private void MoveHand(Vector3 position)
    {
        if (handTransform != null)
        {
            Debug.Log($"Moving hand to: {position}");
            handTransform.position = position;
        }
```

```
        else
        {
            Debug.LogError("handTransform is null. Cannot move hand.");
        }
    }

    private void QueueMainThreadAction(Action action)
    {
        lock (mainThreadQueue)
        {
            mainThreadQueue.Enqueue(action);
        }
    }

    void OnApplicationQuit()
    {
        if (receiveThread != null)
        {
            receiveThread.Abort();
        }
        if (udpClient != null)
            {
            udpClient.Close();
        }
    }
}
```

3. Configure the Rigidbody and Collider components of bowling ball, adjust the physical material parameters to simulate the real rolling and collision effects.



(Use InteractableSphere as an example)

1.**Add the Rigidbody component**
In the Hierarchy panel of the Unity editor, select the bowling ball object you created.
Add the Rigidbody component by navigating through the menu bar Component-> Physics-> Rigidbody.
This component allows the bowling ball to respond to physical forces, such as gravity and forces applied by the user.

Configuration Rigidbody: In the Inspector panel, you can see the settings of the Rigidbody component.
Make sure Use Gravity is checked so that the bowling ball is affected by gravity.
The parameters such as Mass (mass), Drag (resistance) and Angular Drag (angle resistance) should be adjusted according to the needs.

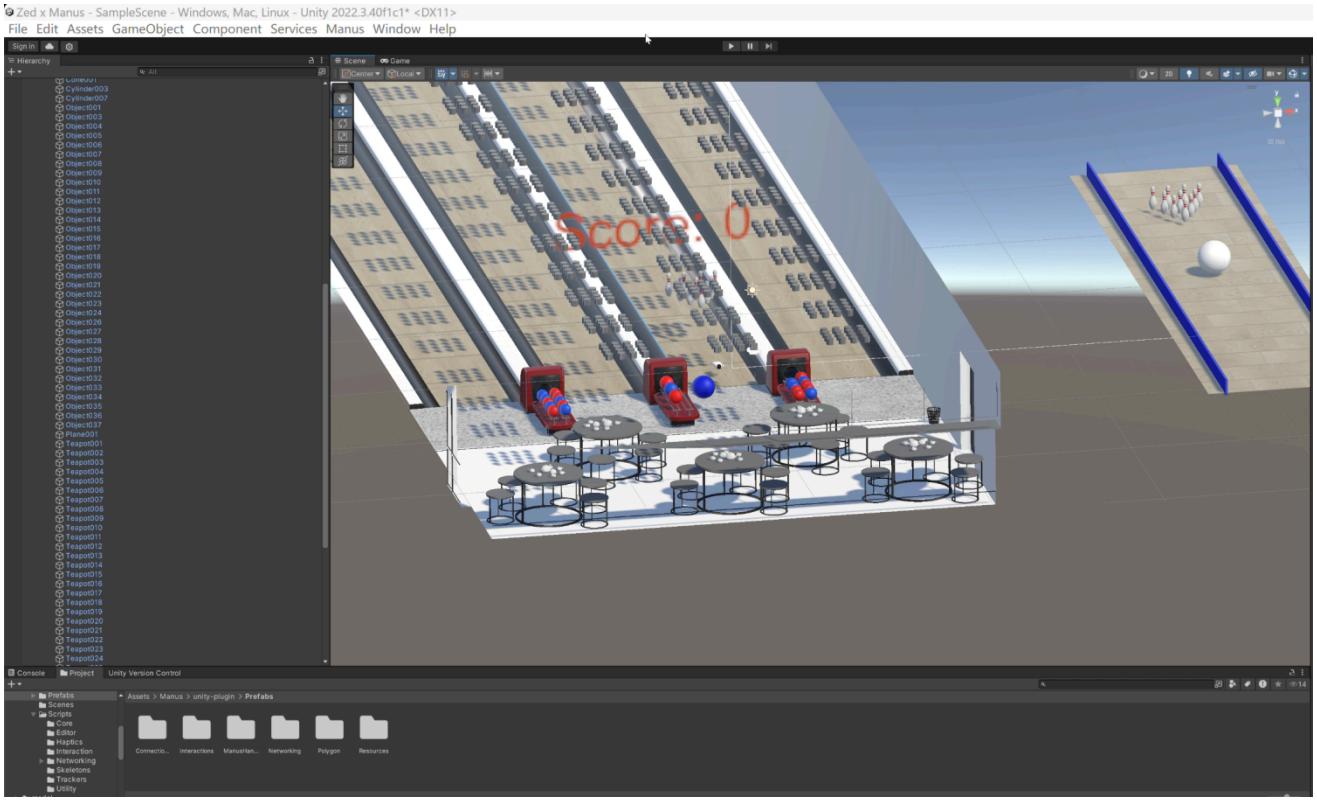
2.**Add Collider**
Select the lane object. Add a Box Collider component, which is suitable for flat or rectangular shaped lanes.
Select bowling ball object. Add the Sphere Collider component, which is suitable for spherical bowling balls.
Adjust the size and position of the collision bodies as needed to ensure that they fit the model correctly.
For the fairway, you may need to adjust the size of the Box Collider to match the size of the fairway.
For bowling, the size of the Sphere Collider should match the radius of the ball.
The same goes for the ball pot

3.**Configure physical materials (optional)**
Create a physical material: In the Project panel, right-click and select Create-> Physic Material.
Name the newly created physical materials, such as "BowlingLaneMaterial" and "BowlingBallMaterial".

Configure physical material parameters: In the Inspector panel, you can set parameters such as Dynamic Friction (dynamic friction), Static Friction (static friction), Bounciness (elasticity), and Friction Combine (friction combination).
For the fairway, you may want to set a lower friction to simulate the real fairway surface.
For bowling, you may need to adjust the elasticity to simulate the balls rebound behavior.

Apply physical material: Drag the created physical material onto the corresponding collision body (Collider) component of the lane and bowling ball.
This is how these physical material Settings are used when the bowling ball collides with the lane or other objects.

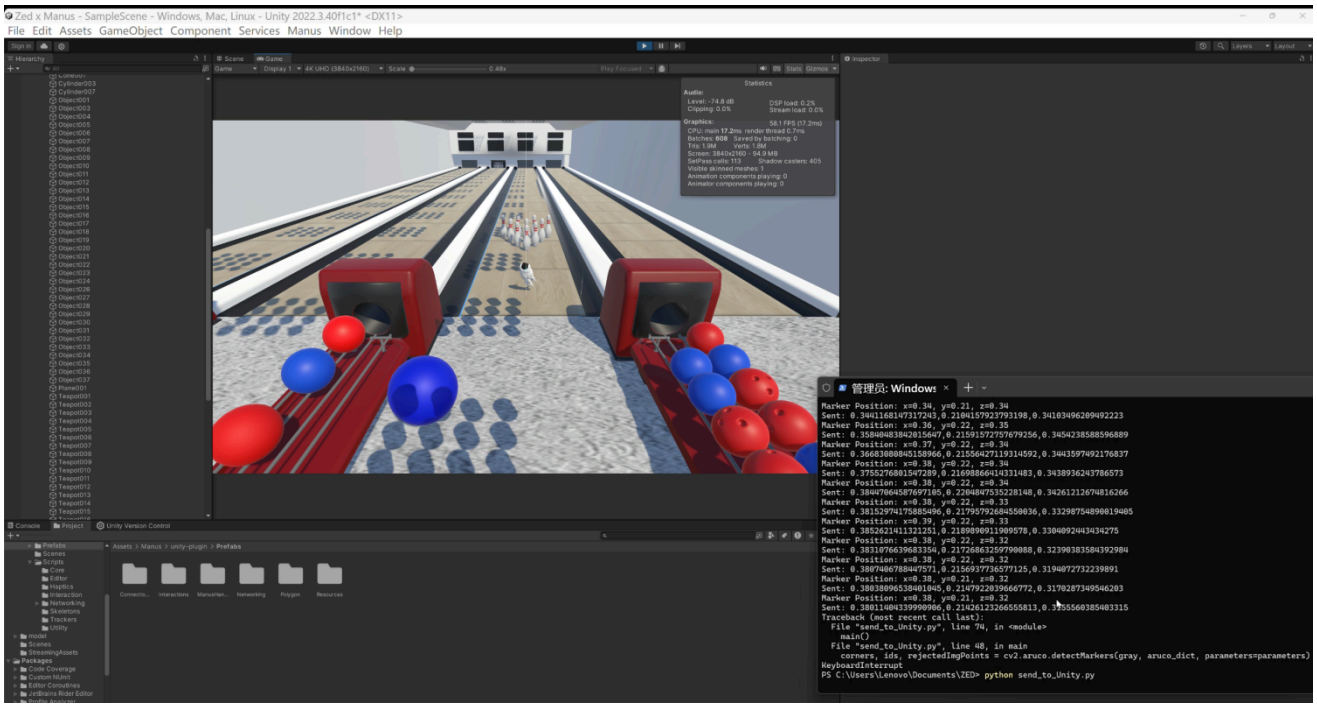
4. Unity improves the scene rendering by replacing objects with real bowling ball and pin models (pictured)



[3d model download website]
[Modeling software blender]

Import model: Import real bowling ball and pin models into the Unity project. These models are usually provided in FBX or OBJ format. You can download free models or build your own models in 3d software, and drag the model file into the Assets panel of Unity.
Replace object: Select the bowling ball and pin objects in the scene. Drag the imported model into the Inspector panel of these objects to replace the original model.

It is worth noting that if you want to add a bowling alley for environmental rendering, it is recommended to uncheck the models Mesh Collider/Mesh Renderer, otherwise it may affect certain physical performance and cause the ball not to be thrown properly.



Click play in Unity and run the SendToUnity.py. Test it several times, adjust the position and speed parameters according to the location feedback in Unity until you achieve a smooth game experience. You can add features or change scenes according to the tutorial.

C.Future Implementation

1.Improve the accuracy of location information (multiple ARuco for multi-point positioning)

scheme:
Multi-point positioning: Place multiple ARuco markers in the scene to ensure they are distributed in different locations. These markers will work together to improve the accuracy of positioning.
Pose estimation and fusion algorithm: Use a computer vision library (such as OpenCV) to identify and estimate the position of each ARuco tag. Then use a fusion algorithm (such as Kalman filter) to integrate this data and calculate the exact position of the object in space.
implementation step:
Install OpenCV and other necessary computer vision libraries.
Place multiple ARuco markers in the cameras field of view.
The script is written to identify and track these markers and attempt to use algorithms to fuse the location information of multiple markers.

2. Bowling pin collision detection (score display)

scheme:
Physics engine: Use the physics engine in Unity to add a collision body (Collider) component to the bowling ball and pin.
Collision detection script: Write a script to detect the collision between the bowling ball and the bottle, and update the score sheet.
UI system: Use the UI system of Unity to create a points display screen for real-time display of current points.
Text update: When the bowling ball hits the pin, update the text content in the UI element.
implementation step:
Add Collider components to the bowling ball and pin in Unity.
Write a script to listen for OnCollisionEnter events and detect collisions between bowling balls and pins.
Create a Canvas in Unity and add a Text or other UI element to the Canvas.
Write a script to update the content of the Text component when the score changes.

3. Three-finger grab recognition (simulating real bowling ball)

scheme:
Gesture recognition: Identify specific three-finger grab gestures through sensor data from the Manus data glove.
Gesture simulation: When the three-finger grab gesture is detected, the grab action of bowling ball is simulated.
implementation step:
Integrate with Manus data glove hardware to obtain sensor data.
Write a gesture recognition algorithm to detect three-finger grabbing gestures.
When a grab gesture is detected, the simulation of the bowling ball grab action is triggered.

4. Capture object vibration feedback (vibration function to be verified)

scheme:
Tactile feedback device: vibration motor is integrated into the data glove to realize vibration feedback.
Implementation steps: Write a script to send a signal to the vibration motor in the data glove when the gesture is recognized, triggering the vibration feedback.

References:

Manus Core Documentation (https://resources.manus-meta.com/)

ZED SDK and StereoLabs Documentation (https://www.stereolabs.com/)

Unity Official Documentation (https://unity.com/)

OpenCV ArUco Marker Documentation