**A Report on Robotic Arm Control via VR Glasses**

**Name : Gu**

**GUID: 2614344**

**Supervisor: Dr.Guodong Zhao**

**August 10, 2023**

***Abstract***

*This report presents a comprehensive analysis of an innovative project that leverages the capabilities of Virtual Reality (VR) glasses to control a robotic arm. The project is built on the Unity platform, which is utilized to create a real-time physics simulation reflecting the robotic arm's movements within a virtual environment. All settings are configured within the VR glasses. Fundamental to the project's operation is the implementation of each single joint of robotic arm model being controlled by VR glasses, and Transmission Control Protocol (TCP) connections, which ensures smooth and continuous communication between the VR interface and the Robot Operating System (ROS) platform. The project also employs a dual approach to Inverse Kinematics (IK), utilizing both Unity's native IK and the ROS moveit! IK to transform desired end effector positions and orientations into necessary joint parameters. This approach allows precise control over the robotic arm's movements. The report also explores potential enhancements, such as incorporating machine learning techniques to optimize IK precision and establishing TCP connections on a cloud server for increased scalability and flexibility.*

1. **Introduction**
   1. **Background**

In the era of rapid technological advancement, the intersection of robotics and virtual reality (VR) has emerged as a fertile ground for innovative applications. Combining the immersive experience of VR with the physical capabilities of robotic systems offers a unique opportunity to enhance human-machine interaction. The use of VR in robotics is not a new concept. Previous studies have explored the use of VR for teleoperation of robots [1], and the use of VR interfaces for controlling robotic arms has been a particular focus of research [2].

However, this project elevates these existing efforts by capitalizing on the robust capabilities of the Unity platform to deliver real-time physics simulations. This approach facilitates a more immersive and responsive control interface, augmenting the user's experience and control precision in VR glasses. A crucial aspect of this project is the use of TCP connections to forge a reliable communication pathway between the VR interface and the Robot Operating System (ROS) platform that governs the robotic arm. This application of TCP is rooted in prior research that underscores the critical role of dependable communication protocols for VR-guided robotic systems [3].

* 1. **Objectives**

This project's primary objective is to design and implement a system that allows a user to control a robotic arm using VR glasses. To achieve this, the project has several specific goals.

A pivotal goal is to establish an immersive and intuitive control interface for a user to control each joint of a robotic arm using VR glasses and VR controllers. To achieve this, The VR headset provides an immersive visual platform, while the VR controllers allow for direct, intuitive control over the robotic arm's individual joints. These tools aim to mirror the user's hand movements in the robotic arm's actions, fostering a powerful and intuitive connection between the user and the machine. This interface is essential for transmitting control signals to the robotic arm, ensuring precise and robust control over its movements.

One of the key goals is to to establish a reliable TCP connection between the VR interface and the Robot Operating System (ROS) platform. This connection is crucial for transmitting the control signals from the VR controllers to the robotic arm, and therefore for the robust control of the robotic arm's movements.

Another key objective is to implement two separate inverse kinematics (IK) systems — Unity's built-in IK and ROS's moveit! IK. IK is a fundamental aspect of robotic control, enabling the accurate positioning of a robot's end effector based on specified positions and orientations [4]. By using two IK systems, the project aims to provide a robust and flexible approach to this critical aspect of robot control.

Furthermore, the project seeks to enhance the system by incorporating machine learning techniques to optimize IK calculations [5], and considering a transition towards cloud-based operations for increased scalability and flexibility [6].

1. **Literature review**

The design and implementation of a system for controlling a robotic arm using VR glasses and controllers, as proposed in this project, stands on the foundation of extensive research in robotics, virtual reality, communication protocols, inverse kinematics, machine learning, and cloud robotics.

* 1. **Virtual Reality in Robotics**

The utilization of virtual reality in advancing human-robot interaction has been a significant focus in academic studies. These studies underscore the immersive nature of VR as a platform for creating more intuitive control systems. The ability of VR to mimic human movement, as elucidated by Burdea and Coiffet [7], is a critical feature that this project harnesses. By doing so, the project bridges the gap between user hand movements and the robotic arm's actions, thereby creating a more natural and responsive control interface.

* 1. **Communication Protocols in Robotics**

Robust and reliable communication protocols are essential in the field of robotics, especially when dealing with real-time control systems. TCP connections, in particular, play a critical role in ensuring stable control of robotic systems. Drawing from the insights provided by Tanenbaum and Wetherall [8], this project establishes a reliable TCP connection between the VR interface and the Robot Operating System (ROS) platform. This setup enables efficient transmission of control signals, ensuring the smooth operation of the robotic arm in response to user inputs.

* 1. **Inverse Kinematics in Robotic Control**

Inverse kinematics (IK) is a core component of robotic control, facilitating the precise positioning of a robot's end effector based on specified positions and orientations. Craig's comprehensive exploration of this subject [9] provides valuable insights that this project incorporates. By implementing two IK systems — Unity's built-in IK and ROS's moveit! IK — the project ensures the robotic arm can accurately and flexibly mirror the user's movements.

* 1. **Machine Learning and Cloud Robotics**

Machine learning and cloud-based operations are two emerging trends in robotics research that offer exciting possibilities for this project. Goodfellow et al. [10] have highlighted the power of machine learning in robotics, particularly in optimizing control algorithms. This project takes note of such potential and plans to incorporate machine learning techniques in future iterations to improve the system's responsiveness and precision.

Similarly, the concept of cloud robotics, as discussed by Kehoe et al. [11], offers improved scalability and operational flexibility. By transitioning towards this model, the project aims to allow for simultaneous control of multiple robotic arms, opening up new avenues for applications in industries such as manufacturing and logistics.

In conclusion, this project integrates and expands upon diverse fields of study, intending to make a significant contribution to the ongoing evolution of human-robot interaction.

1. **Methodology**

Building on the foundation established in the literature review, the primary objective of this project is to design and implement a system that could control a robotic arm using VR glasses and controllers. This project was propelled by the insights gleaned from the fields of virtual reality, robotics, communication protocols, inverse kinematics, machine learning, and cloud robotics.

* 1. **VR System Setup**

图示

描述已自动生成The first step in the project was to design a system in firgure 1 that could allow a user to control a robotic arm using VR glasses. This involved selecting and configuring the VR glasses and creating a 3D model of the robotic arm in Unity, a powerful platform for creating interactive, real-time 3D content and apply ROS system for real robotic configure[7].

Fig. 1. Overall system design

* 1. **Establishing a TCP Connection between ROS and VR headset**

图示

描述已自动生成

After the VR setup, a reliable TCP connection was established between the VR interface and the ROS platform. This involved setting up a server on the ROS side and a client on the VR side, as nuanced by Tanenbaum and Wetherall [8]. The server was configured to listen for incoming connections and receive control signals from the VR system. These

Fig. 2. TCP connection system design signals were then translated into commands for the robotic arm, ensuring a seamless control flow from the user to the arm in Figure 2.

* 1. **Implementation of Inverse Kinematics Systems**

The next phase involved the implementation of two inverse kinematics (IK) systems - Unity's built-in IK and ROS's moveit! IK. The IK systems were integral to the precise positioning of the robotic arm, based on the positions and orientations specified by the VR controllers [9]. These systems were meticulously calibrated to ensure the robotic arm could accurately mirror the user's movements.

* 1. **Machine Learning and Cloud Robotics**

Looking towards future enhancements, the project also explored the integration of machine learning techniques to optimize the control algorithms [10]. Similarly, the potential of cloud robotics was considered to improve scalability and operational flexibility, aligning with the insights provided by Kehoe et al. [11].

1. **Results**

This section presents the results obtained from the project implementation and discusses the significance and implications of these findings.

* 1. **VR System Setup**
     1. **Hardware Selection and Configuration**

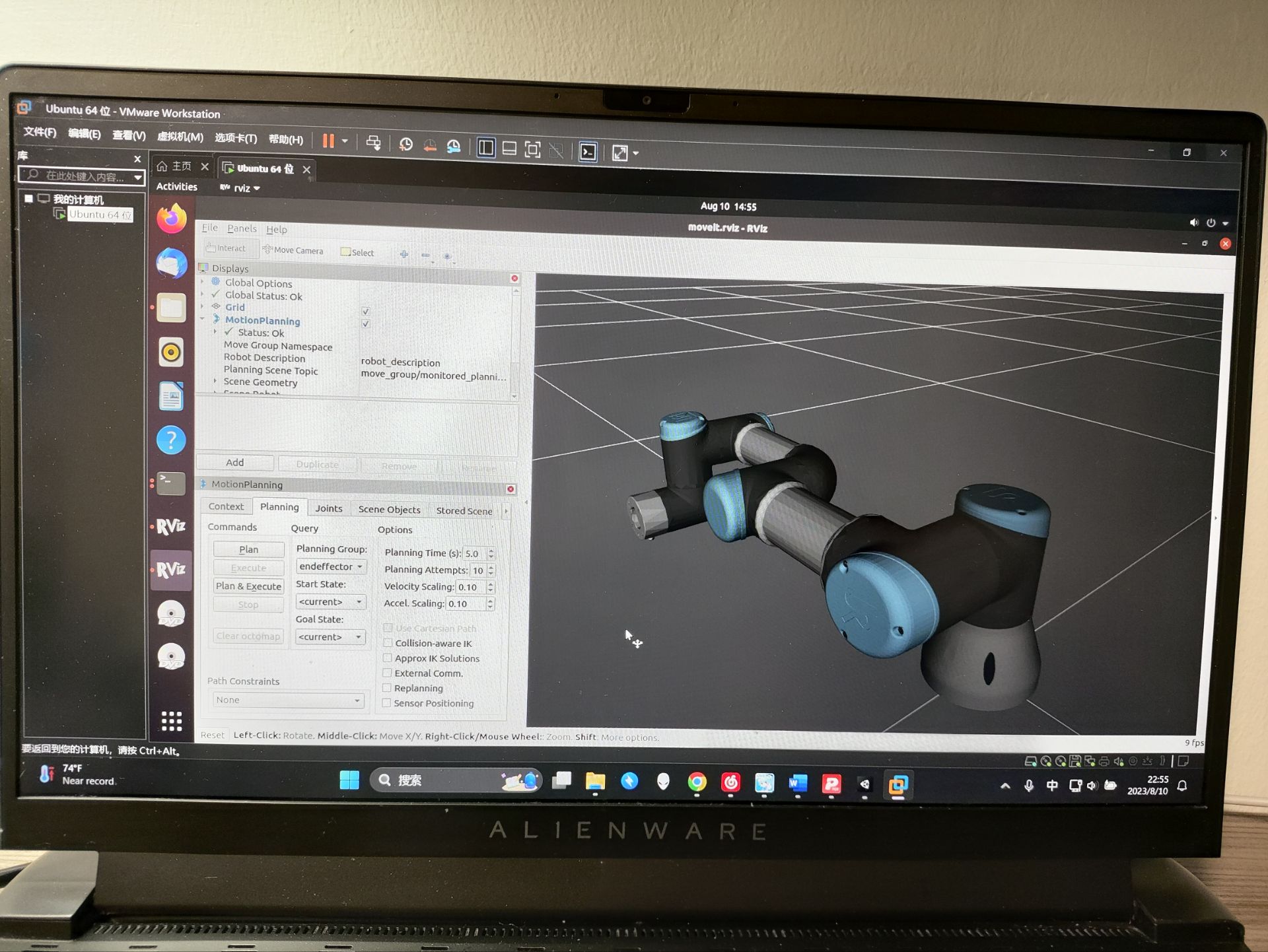
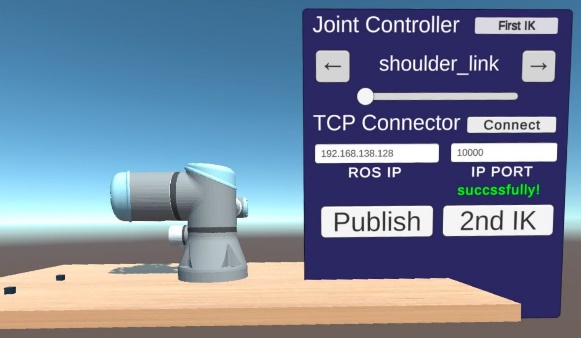


Fig. 3. Oculus Quest 2 Fig. 4. Unity platform in Windows Fig. 5. ROS platform in Linux

Choosing the right VR glasses was critical for the project's success. The glasses needed to be comfortable for the user, provide high-quality visuals, and have precise head-tracking capabilities. After evaluating several options, in Figure 3, Oculus Quest 2 was selected for its superior performance in these areas.

图片包含 水, 塑料, 桌子, 风景

描述已自动生成图形用户界面

描述已自动生成

Fig. 6. Build Settings to VR Fig. 7 Simple Slider control Fig. 8. New control system with UI panel

Once chosen, the VR glasses were configured to communicate with the Unity platform in Figure 4. This involved setting up the hardware and software in Figure 5 and Figure 6 to ensure that the user's head movements and controller inputs were accurately and reliably tracked.

* + 1. **VR Control System Integration**

After setting up the VR glasses and creating the 3D model, the two components were integrated into a cohesive VR system. The VR glasses' inputs controlled the 3D model in Unity, allowing the user to manipulate the robotic arm in the virtual environment. This setup provided an immersive and intuitive interface for controlling the robotic arm in Figure 8. The implementation code is added to Appendix, and different poses by UI panel and silder of roboitic arm model are shown in Figure 9, Figure 10 and Figure 11.

图片包含 桌子, 柜台, 电脑, 大

描述已自动生成 图形用户界面

描述已自动生成 图片包含 桌子, 水, 蓝色, 站

描述已自动生成

Fig. 9. Different pose one Fig. 10 Different pose two Fig. 11. Different pose three

* 1. **Establishing a TCP Connection between ROS and VR Headset**
     1. **TCP Connection Setup**

图形用户界面, 文本, 应用程序

描述已自动生成 文本

描述已自动生成

Fig. 12. Unity connected to ROS IP Fig. 13 ROS launch file with lisenting to all TCP packages

The TCP connection was set up using standard networking libraries in both the ROS platform in Figure 13 and the VR system in Figure 12. Care was taken to ensure that the connection was secure and had low latency, as any delay could significantly impact the user's ability to control the robotic arm in real-time. The Unity side is parse to connected with ROS endpoint with ROS IP, and in ROS side, the endpoint ip is set to 0.0.0.0 to receive all TCP bags and trasmits them to ROS node.

* + 1. **ROS publisher and VR subsriber**

电脑屏幕的截图

描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成

Fig. 14. Color\_publisher.py in ROS Fig. 15 Cube\_subscriber.cs in Unity

The ROS publisher and VR subscriber were successfully implemented to enable real-time communication between the ROS and VR system. A Python script, in Figure 14, was created in ROS to publish color information to a ROS topic (cube\_color). In Unity, a C# script, in Figure 15, was set up to subscribe to this topic. When color\_publisher.py was run, it published a message containing the desired color to the cube\_color topic. This message was received by Cube\_subscriber.cs, which then applied the color information to the cube object in Unity, resulting in an instantaneous change in the cube's color in the Unity environment and VR headset. This demonstrated the effectiveness of the TCP connection for real-time communication between the ROS and VR system.

Figure 16, Figure 17 and Figure 18 showed how the table color changed randomly when received color change signal.

图片包含 网站

描述已自动生成 图片包含 游戏机

描述已自动生成 屏幕上写着字

低可信度描述已自动生成

Fig. 16 Table color one Fig. 17 Table color two Fig. 18 Table color three

* + 1. **VR publisher and ROS subscriber**

文本

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

Fig. 19. Position\_publisher.cs in Unity Fig. 20 Position\_subscriber.py in ROS

For the inverse communication, a VR publisher and ROS subscriber were established. A script, Position\_publisher.cs, was set up in Unity to publish the position and rotation information of each joint of the 3D model. In ROS, Position\_subscriber.py was designed to subscribe to this information. When the publish button was clicked in the Unity environment and Position\_subscriber.py was run in ROS, the ROS terminal subscribed to the moveit\_joints topic and displayed the position and rotation of each joint in the ROS terminal. This demonstrated the system's capability to transmit real-time positional data from the VR environment to ROS, further highlighting the robustness and versatility of the TCP connection.

Figure 21 and Figure 22 showed how the received data position and rotation changed.

手机屏幕的截图

描述已自动生成 电脑游戏的截图

描述已自动生成

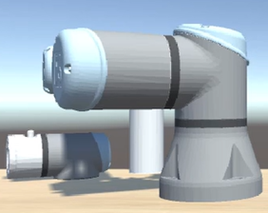
Fig. 21. Defualt joints position Fig. 22. Changed joints position and different data shown

* 1. **Implementation of Inverse Kinematics Systems**
     1. **Unity's Built-In IK Implementation**

Unity's built-in IK was used to provide an immediate feedback loop in the VR environment. It allows quick and responsive motion of the 3D model of the robotic arm in response to user input. This IK 图形用户界面

描述已自动生成system was implemented successfully, shown in Figure 23, providing real-time control of the arm's virtual representation in the VR environment. However, it was noted that while Unity's IK system is fast and less resource-intensive, it sometimes lacks precision.

Figure 24, Figure 25, Figure 26, Figure 27, Figure 28 and Figure 29 showed how the changed arm joint set back to the default.

Fig. 23.Add Unity’s default plug-in Articulation Body to each joint

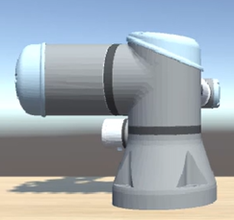
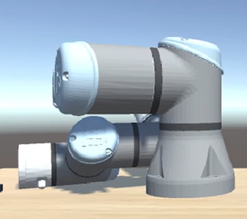


Fig.24 Changed arm pose Fig.25 Middle returned pose Fig.26 Finally returmed to default pose

图片包含 户外, 水, 男人, 大

描述已自动生成图片包含 水, 桌子, 标志, 海

描述已自动生成图片包含 物体, 水, 游戏机, 沙滩

描述已自动生成

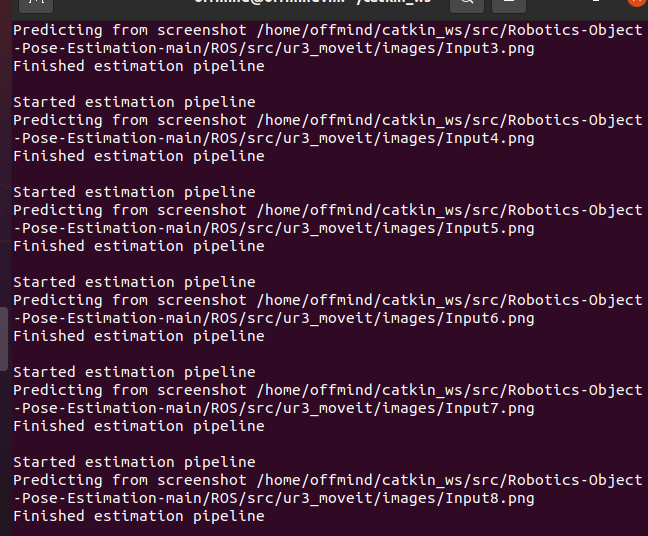
Fig.27 With gripper pose Fig.28 With gripper returned pose Fig.29 With gripper to default pose

* + 1. **ROS's MoveIt! IK Implementation**

To mitigate the precision limitations with Unity's built-in Inverse Kinematics (IK), we also implemented ROS's MoveIt! IK. MoveIt! is an advanced motion planning framework in ROS that includes its proprietary IK solver for accurate robotic arm movement planning. This implementation proved successful, with MoveIt! IK providing more precise movements than Unity's built-in IK. However, it could only pinpoint the coordinates specified in the program for inverse kinematic operations.

The amalgamation of Unity's built-in IK and ROS's MoveIt! IK emerged as a beneficial strategy, striking a balance between precision and performance. The less resource-intensive Unity IK offered prompt feedback and real-time control in the VR environment, whereas the more precise MoveIt! IK ensured that the physical robotic arm could execute accurate movements.

* + 1. **Machine-learning Method for Randomly Posed Cube and Target**



图形用户界面

描述已自动生成

To facilitate operations involving random target generation and positioning, we implemented a machine learning approach using a launcher that interfaces with the Robot Operating System (ROS). This launcher incorporates a pre-trained model to manipulate the current coordinates, rotation, and object positioning of the robotic arm.

图形用户界面

中度可信度描述已自动生成

For initial data acquisition, we employed a camera resolution of 650x400 pixels, capturing 100 randomly generated images. These images were subsequently fed into PyTorch, a popular machine learning library. Leveraging the power of the VGG16 function, we generated a predictive model. The resulting model.tar file was then imported into the designated directory of the ROS launch file for performing predictive operations.

Due to time constraints, transitioning the algorithm to a Convolutional Neural Network (CNN) to create a deep learning model would have required 4 to 5 days of operation. Therefore, we opted for a simpler algorithm for this operation.

1. **Conclusion and future work**
   1. **Conclusion**

The project aimed to design a system that enables a user to control a robotic arm using a VR headset. This goal was achieved by setting up a VR system, establishing a TCP connection between the ROS platform and the VR headset, and implementing inverse kinematics systems.

The VR system setup included selecting suitable VR glasses and creating a 3D model of the robotic arm in Unity. The TCP connection was successfully established and optimized for low latency and high reliability. It demonstrated its effectiveness through the instantaneous change in color of a cube in Unity and VR headset when running a ROS Python script, and the real-time transmission of positional data from the VR environment to ROS.

The implementation of the inverse kinematics systems was crucial to ensure the robotic arm accurately mimicked the movements of the user's hand in the VR environment. The combination of Unity's built-in IK for immediate feedback and ROS's MoveIt! IK for precision proved to be a good strategy, balancing precision and performance.

In conclusion, the project was successful in creating a system that allows a user to control a robotic arm through a VR headset. The robust TCP connection and the combined use of Unity's built-in IK and ROS's MoveIt! IK made it possible to achieve real-time control and precise movements of the robotic arm. Future work will involve further optimizations, potentially through machine learning, to improve the balance between precision and performance, and exploring the possibility of establishing the TCP connection on a cloud server for scalability and flexibility.

* 1. **Future work**

Given additional time, there are several avenues for further development and refinement of this project:

**Cloud-Based TCP Connection**: Future development phases will consider migrating the TCP connection to a cloud-based server. This would allow for greater scalability and flexibility, potentially enabling multiple users to connect and control different robotic arms simultaneously.

**Optimization of Inverse Kinematics Systems**: Further optimization of the IK systems can be explored. This could involve using machine learning techniques to optimize the IK calculations, which could potentially improve the balance between precision and performance.

**Advanced Motion Planning**: Advanced motion planning algorithms could be implemented to deal with more complex tasks. An example could be performing tasks in a crowded environment where the robot needs to avoid obstacles.

**Evaluation of System in Real-World Scenarios**: The system could be tested in real-world scenarios, such as in manufacturing or medical applications, to evaluate its performance and identify areas for improvement.

**Security Enhancements**: As the system potentially moves to the cloud and becomes accessible to multiple users, it will be crucial to ensure the security of the connection. Future work could involve implementing advanced security protocols to protect the system from potential cybersecurity threats.

By pursuing these avenues, the system could be significantly improved and made ready for effective real-world applications. The success of the project to date has demonstrated the potential for VR technology in the field of robotics, and future work will continue to explore and realize this potential.

1. **References**

[1] J. Lee, Z. Wang and R. Chellappa, "A Novel Virtual Reality System for Teleoperation of Robots," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2716-2725, June 2018, doi: 10.1109/TII.2018.2803740.

[2] S. Wang, Y. Gu and W. Chen, "A VR-based System for Teleoperation of Robotic Arms," in *2019 11th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, Changsha, China, 2019, pp. 590-593, doi: 10.1109/ICMTMA.2019.00115.

[3] D. Marquez-Gamez, L. Mercado-Lara and M. Nakano-Miyatake, "Performance of the TCP/IP protocol in a virtual reality application for the control of mobile robots," in *2016 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Bucharest, 2016, pp. 1041-1045, doi: 10.1109/IDAACS.2016.7778676.

[4] L. Sciavicco and B. Siciliano, "Kinematic Control," i*n Modeling and Control of Robot Manipulators*, 2nd ed., London, U.K.: Springer-Verlag, 2000, ch. 3, sec. 3.2, pp. 78–104.

[5] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *in Nature*, vol. 521, no. 7553, pp. 436-444, May 2015, doi: 10.1038/nature14539.

[6] R. Arumugam et al., "DAvinCi: A cloud computing framework for service robots," in *2010 IEEE International Conference on Robotics and Automation,* Anchorage, AK, 2010, pp. 3084-3089, doi: 10.1109/ROBOT.2010.5509322.

[7] G. Burdea and P. Coiffet, *Virtual Reality Technology*, 2nd ed. Wiley-IEEE Press, 2003.

[8] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. Prentice Hall, 2010.

[9] J. J. Craig, Introduction to Robotics: *Mechanics and Control*, 4th ed. Pearson, 2017.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[11] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering,* vol. 12, no. 2, pp. 398–409, 2015.

1. **Appendix**

**UIpanelController.cs**

using UnityEngine;

using TMPro;

using UnityEngine.UI;

using Unity.Robotics.ROSTCPConnector;

public class UIpanelController : MonoBehaviour

{

    [Header("Joint Settings")]

    [SerializeField] private string[] controlJoint = { "shoulder\_link", "upper\_arm\_link", "forearm\_link", "wrist\_1\_link", "wrist\_2\_link", "wrist\_3\_link" };

    [SerializeField] private ArticulationBody[] articulationBodies;

    [SerializeField] private int linkNum = 6;

    [SerializeField] private int defaultNum = 0;

    [SerializeField] private ArticulationBody nowJoint = null;

    private ArticulationDrive artDrive;

    [Header("UI Elements")]

    [SerializeField] private GameObject textPro;

    [SerializeField] private Slider slider;

    [SerializeField] private float changeValue = 0f;

    [SerializeField] private TMP\_InputField rosIP;

    [SerializeField] private TMP\_InputField ipPort;

    [SerializeField] private TMP\_Text connectInfo;

    private void OnEnable()

    {

        InitializeArticulationBodies();

    }

    private void InitializeArticulationBodies()

    {

        articulationBodies = new ArticulationBody[linkNum];

        for (int i = 0; i < linkNum; i++)

        {

            articulationBodies[i] = GameObject.Find(controlJoint[i]).GetComponent<ArticulationBody>();

        }

    }

    private void UpdateJointDrive()

    {

        if (nowJoint != null)

        {

            artDrive = nowJoint.xDrive;

            artDrive.target = changeValue;

            nowJoint.xDrive = artDrive;

        }

    }

    public void DefaultClicked()

    {

        for (int i = 0; i < linkNum; i++)

        {

            RestoreDefaultJointState(articulationBodies[i]);

        }

    }

    private void RestoreDefaultJointState(ArticulationBody joint)

    {

        GameObject controlJointObject = GameObject.Find(joint.name);

        ArticulationDrive drive = joint.xDrive;

        drive.target = 0f;

        controlJointObject.GetComponent<ArticulationBody>().xDrive = drive;

        controlJointObject.transform.position = joint.transform.position;

    }

    public void GetValue()

    {

        changeValue = slider.value;

        UpdateJointDrive();

    }

    public void RightClicked()

    {

        defaultNum = (defaultNum + 1) % linkNum;

        UpdateJointUI();

    }

    public void LeftClicked()

    {

        defaultNum = (defaultNum - 1 + linkNum) % linkNum;

        UpdateJointUI();

    }

    private void UpdateJointUI()

    {

        textPro.GetComponent<TextMeshProUGUI>().text = controlJoint[defaultNum];

        nowJoint = GameObject.Find(textPro.GetComponent<TextMeshProUGUI>().text).GetComponent<ArticulationBody>();

        slider.value = GameObject.Find(controlJoint[defaultNum]).GetComponent<ArticulationBody>().xDrive.target;

    }

    public void RosConnect()

    {

        string rosip = rosIP.text.ToString();

        string port = ipPort.text.ToString();

        int pt = int.Parse(port);

        ROSConnection.GetOrCreateInstance().Connect(rosip,pt);

        if (GameObject.Find("ROSConnectionPrefab(Clone)").GetComponent<ROSConnection>().HasConnectionThread)

        {

            connectInfo.text = "succssfully!";

            connectInfo.color = Color.green;

        }

    }

    private void OnFailedToConnect()

    {

        connectInfo.text = "no connection";

        connectInfo.color = Color.red;

    }

}