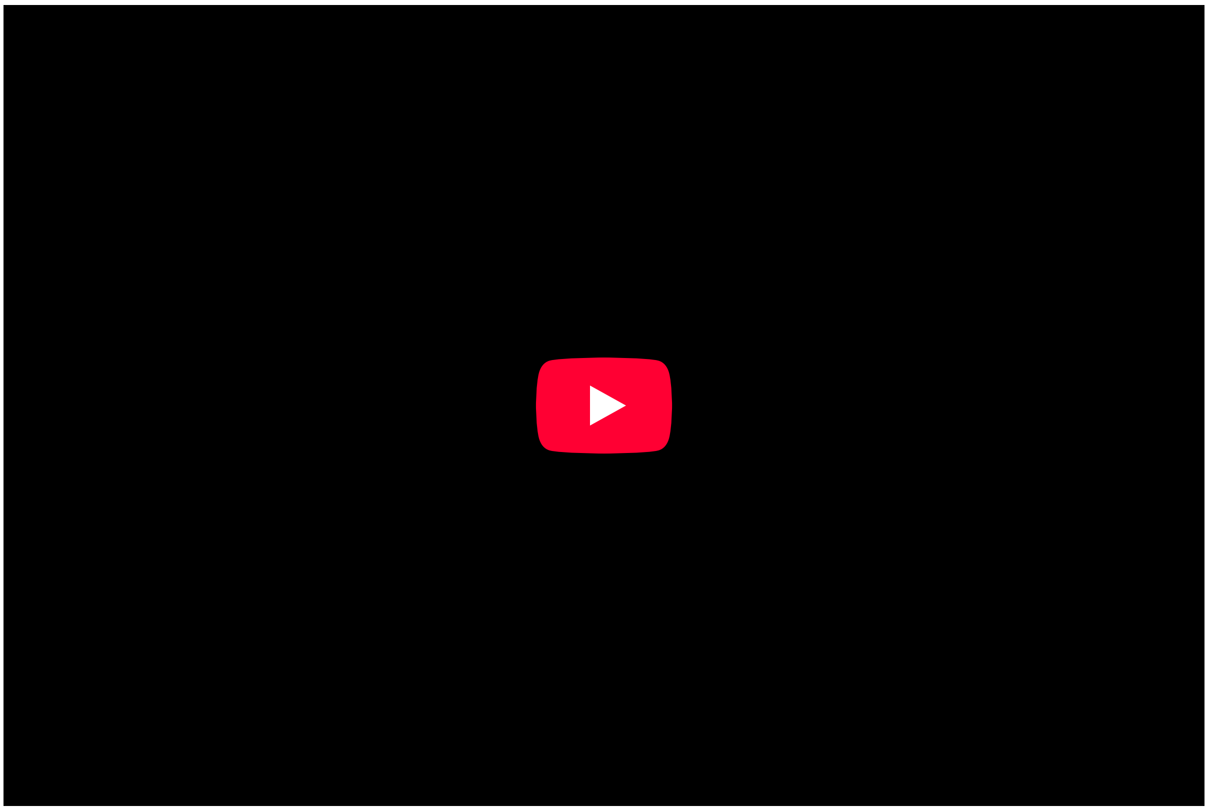
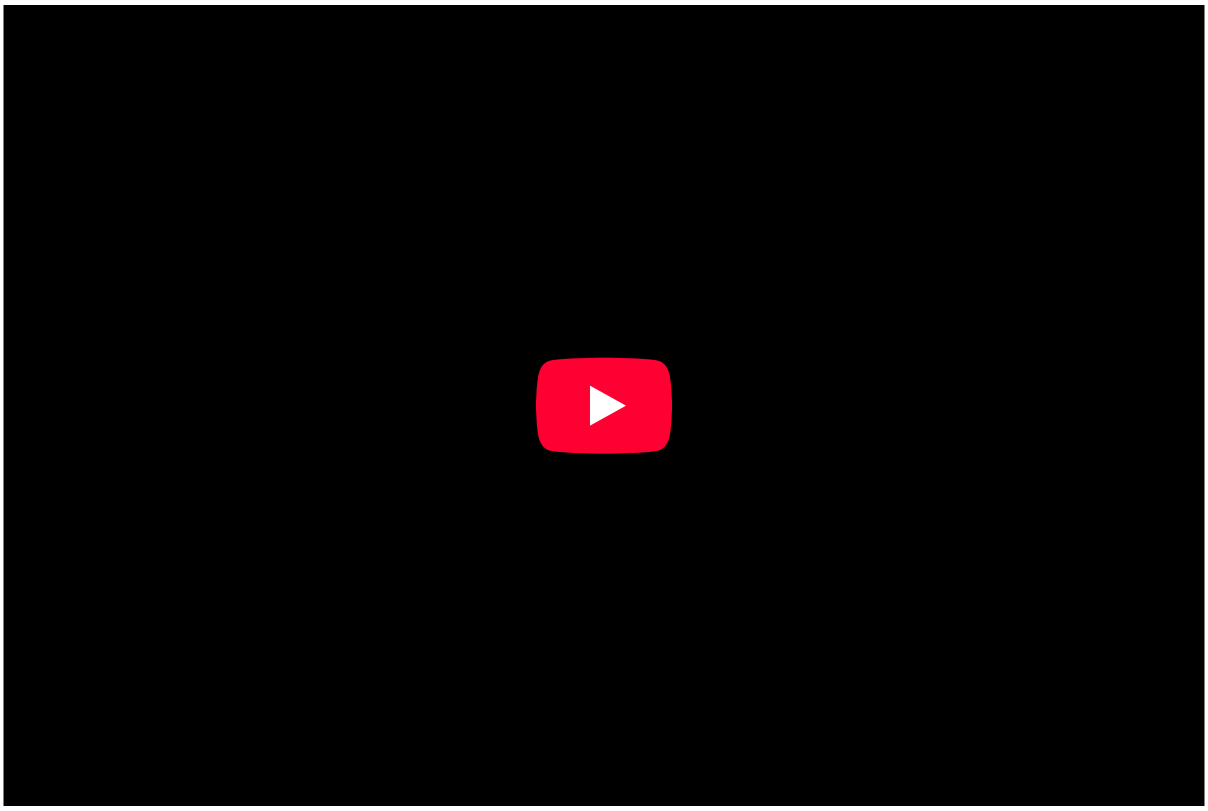
Teleoperating Quadrupeds with AR Glasses



Knocking Pins using hand gestures and feedback from the AR Glasses



Closeup of what the AR Glasses display

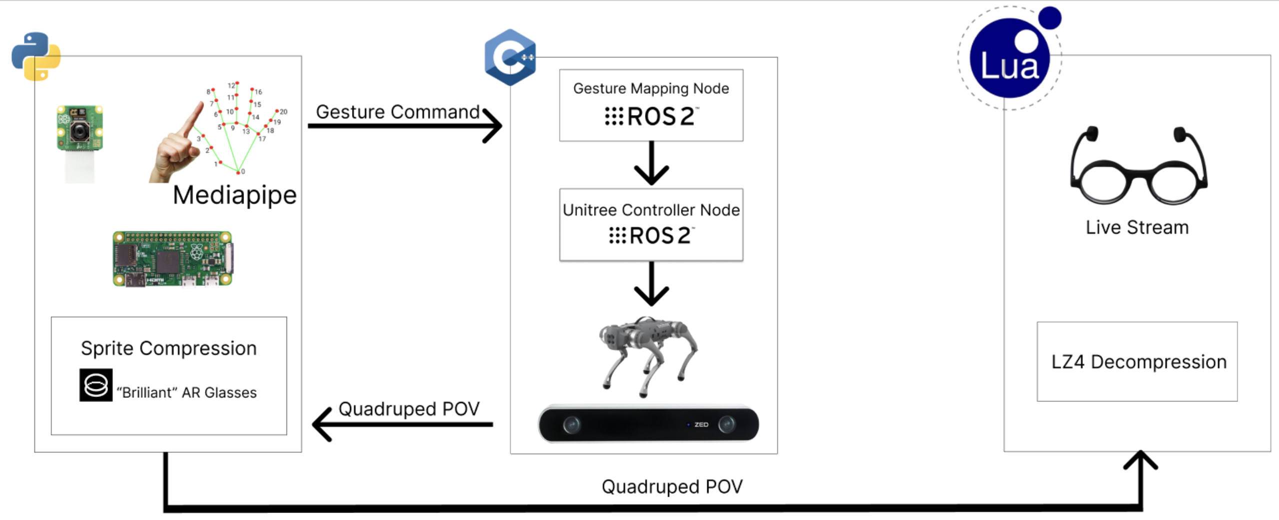
PROF

Github Repository is here

Project Overview

AR technology has spent the last decade on standby to be the next big breakthrough. This started in 2013 when Google released its consumer AR glasses, the "Google Glass", which ended up in their graveyard in 2023. This is a drop in the bucket of failed AR technologies, and it seems like AR will never receive mainstream adoption. Recently, Meta came out with the new consumer Ray Ban AR Glasses, claiming to have revolutionized the industry. However, they have a big problem: lack of developer support. While the Ray Ban Meta glasses can benefit people with low-vision, we as developers are not given an opportunity to develop applications for this technology.

In the sea of closed-source AR devices, Brilliant Labs is a company that attempts to restore faith in AR technology. Their frames are fully open-source and support Python, Flutter, and JavaScript SDKs, opening up many opportunities for applying AR technology in a meaningful way. The Brilliant Frames support a fully programmable 640x400 color OLED display and a fully capable 720p low power color camera. I found this to be adequate for applying mediapipe by using the glasses' camera (shown on the disassembled image later in this article). To maximize the frames per second (FPS) of the quadruped's point of view, and to minimize latency, I used central computer camera instead of the one on the glasses. It is possible to take and receive photos on the glasses simultaneously, but it limits the FPS and slows down the whole operation.



Block Diagram: Input from the camera and output on the display

Unitree GO1 Quadruped Robot

The GO1 robot is a quadruped robot developed by Unitree Robotics. It is controlled using a ROS2 C++ API that communicates with the robot through a UDP connection. The API provides functions to control the robot's joints, read sensor data, and get camera feed. However, I am using a camera feed. Instead a ZED camera is mounted on top of the robot and connected to a Jetson Nano, which runs the dog_camera_worker upon startup. The robot control pipeline is adapted from the repository unitree_legged_sdk. A new ROS2 node, `frames_open_palm`, was created to control the robot. The node has a variable that stores the current gesture.

```
std::string current_gesture_ = "";
```

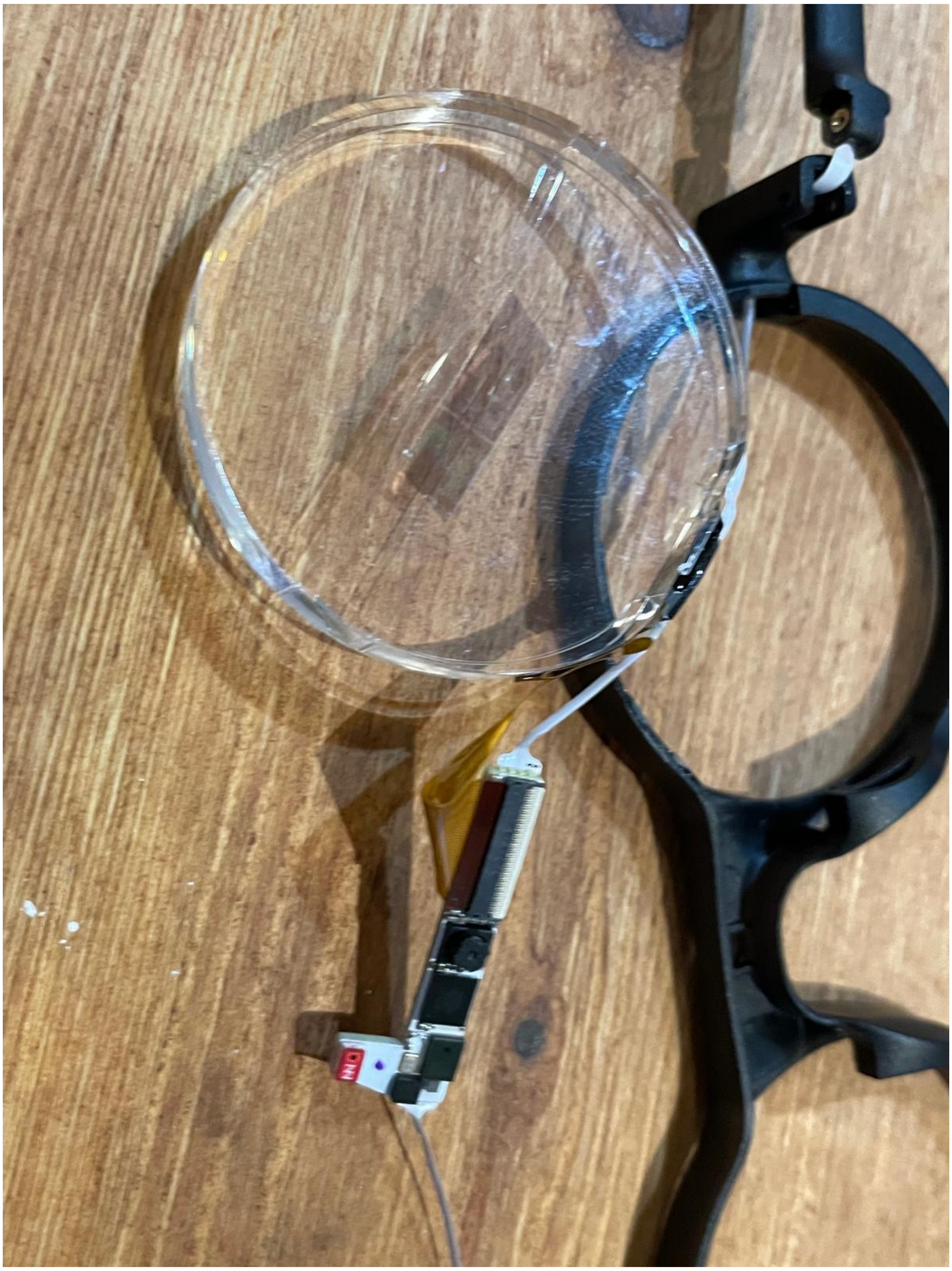
This is updated via a subscription to the `hgr_topic` topic. The subscriber callback receives strings representing gestures and sets the current gesture accordingly.

```
sub_ = this->create_subscription<std_msgs::msg::Int32>(
            "hgr_topic", 10,
            [this](std_msgs::msg::Int32::SharedPtr msg) {
                RCLCPP_INFO(this->get_logger(), "Received gesture code:
'%d'", msg->data);
                this->current_gesture_ = mapGestureCodeToString(msg-
>data);
            });
```

In the high-frequency control loop, the node evaluates the current gesture and formulates a corresponding command message. Depending on the gesture (such as "up," "down," "left," "right," "forward," "back," or "hand"), the node configures parameters like velocity, gait type, and motion mode. Then, it publishes a message to the "high_cmd" topic.

AR Glasses

PROF



AR Glasses Disassembled

PROF

The AR glasses are a central component for providing the operator with real-time visual feedback directly from the robot's onboard ZED camera. The **Brilliant Frames** are entirely open-source. For this project, I developed the graphics for it, which is documented here. Below are the key developments:

Displaying images on the glasses:

- I wrote a custom Python TxSprite class was written to convert images into a sprite format compatible with the glasses. Due to the availability of 4 bits per pixel, the images were quantized to 16 colors (huge shoutout to github user @CitizenOneX for his help in this process). The "Brilliant Frames" company promoted my code into a pip library for future developers to use and can be found here. My full contributed code can be seen here.
- I updated and flashed Lua scripts to ensure that bitmapped images (with necessary color quantization for a monochrome display) could be rendered on the glasses. For the purpose of this project, the following file is flashed in `camera_feed.py` in the frames_worker:

```
await frame.upload_file("lua/compressed_prog_sprite_frame_app.lua",
"frame_app.lua")
```

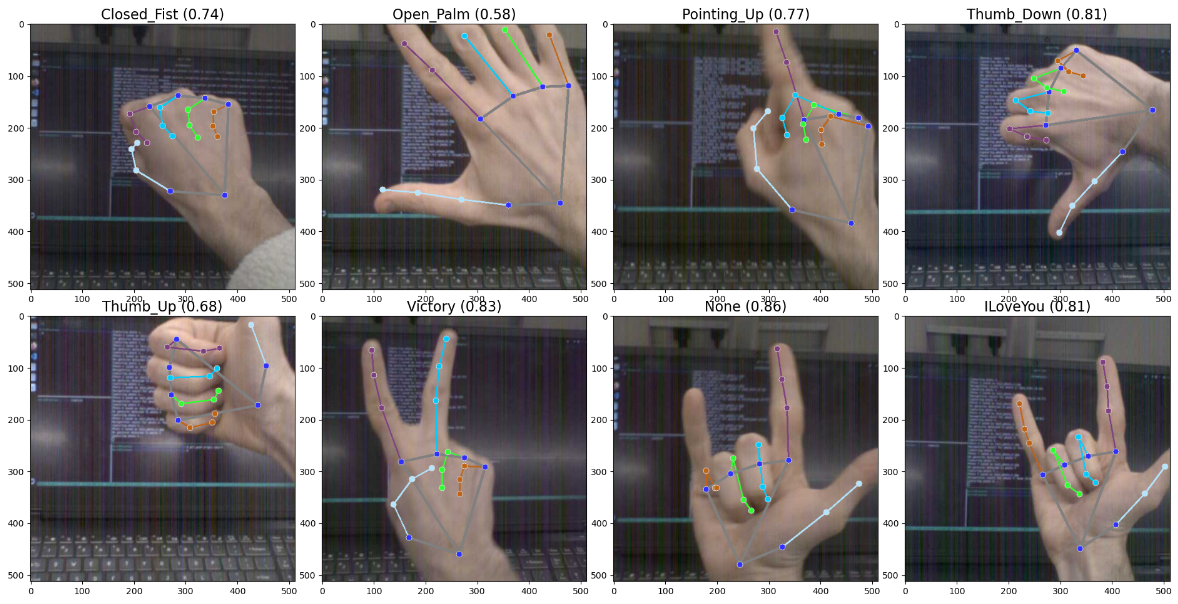
Image Feedback from the Robot:

- Early experiments streamed images using SCP for convenience. After rapid prototyping, the pipeline was changed to use HTTP transfer (yielding approximately 0.3-second latency) for improved reliability.
- Improvements included parallelization of the Mediapipe pipeline and the robot's live stream using asyncio, leading to a more robust closed-loop control demonstration.
- Experimental modifications with LZ4 compression, enabled by firmware release v25.031.0924 (which added an API for decompression), further accelerated the display. This firmware was published by Brilliant Frames on GitHub and can be found here.

Gesture Recognition

I made gesture recognition using the MediaPipe library. MediaPipe is a cross-platform framework for building multimodal applied machine learning pipelines. It was developed by Google and is used in a variety of applications, including gesture recognition. MediaPipe has a pre-trained model for hand gesture recognition that can recognize 8 different gestures. The model is based on a convolutional neural network (CNN) and is trained on a large dataset of hand gestures. In the original implementation, mediapipe was a standalone script. This created a massive control latency. Inspired from Ava Zahedi's mediapipe integration into ROS2, I managed to bring the latency to near zero, where it feels very smooth, living up the teleoperation name. Below is an image of the available gestures:

PROF

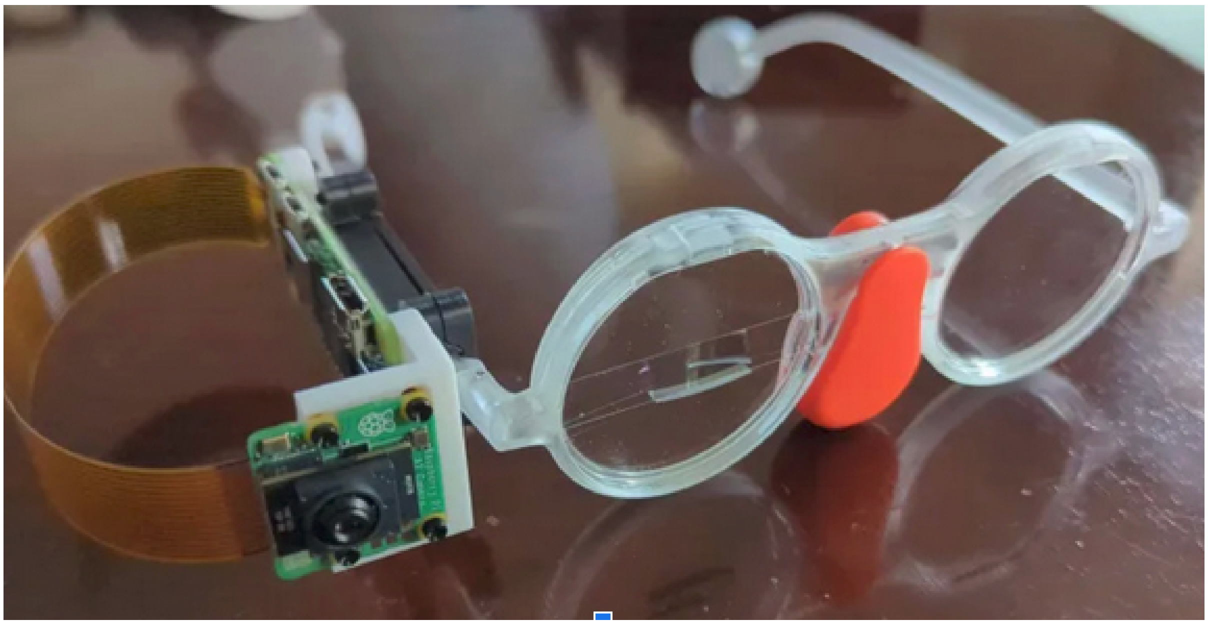


Initial gesture_recognizer.task model gestures

Integration, Advanced Gestures, and Future Modes

After the successful integration of gesture-based controls with visual feedback, my focus turned to advanced control modes. The Unitree GO1 successfully performed all required movements (left/right/up/down) and even executed a dance routine in response to an “I love you” gesture in ASL. A desired feature I have yet to implement is an IMU-based control mode. When teleoperating the robot, it is useful to turn the robot's head independently, without moving the body. Another enhancement would be developing a side mount and porting the entire program so it runs as a startup script on a raspberry pi. This will create a standalone hardware that is ready to go at the push of a button.

PROF



Side Mount of the AR Glasses. Concept Design Found on Brilliant Frames Discord

Conclusion

This project demonstrates the feasibility of closed-loop control of a quadruped robot, using high-level hand gestures and AR-enabled visual feedback. The integration of ROS2-based control, Mediapipe hand tracking, and AR glasses for real-time imaging forms a robust proof-of-concept for teleoperation in a resource constrained environment. This not only achieves closed-loop control but also opens the door to future enhancements such as IMU-based head movement control.

Sources and References

- Brilliant Frames
- MediaPipe
- Unitree SDK
- Unitree Documentation
- Mediapipe Integration with ROS2