VR Robot Arm Teleoperation Control Scheme

Core Principle

This teleoperation method is based on a **relative, incremental control scheme**, inspired by how we use a computer mouse and mouse pad (we lift up and put down the mouse to disengage and engage with the pointer's movements).

- The user can "lift" the VR controller off the **virtual mouse pad** (by releasing a button) to reposition their physical hand without affecting the robot arm.
- When the controller is "placed" back down (by pressing the button), only the **relative changes** in the controller's position and orientation are mapped proportionally to the robot's end-effector.
- This design ensures smooth, safe, and continuous motion, avoiding the instability and sudden jumps that occur with absolute positional mapping.

Purpose of the Design

The relative control method is crucial for **safety and stability**:

- **Avoids discontinuity:** Direct, one-to-one mapping of absolute positions would cause the robot arm to instantly jump to the user's hand position upon engagement. Such behavior is unpredictable and potentially dangerous.
- **Ensures smoothness:** By mapping only incremental changes, the robot arm moves continuously and predictably, maintaining operator control and system stability.

Step-by-Step Workflow

1. **Initialization**
    - The robot arm and VR controller begin in their own independent positions.
    - No control link is active at this stage.
2. **Engagement**
    - When the user presses and holds a designated button on the VR controller, the control system **engages**.
    - A baseline is established for the controller's current position and orientation.
3. **Relative Movement Mapping**
    - After engagement, any movement of the VR controller (translation or rotation) is treated as a **relative increment**.
    - These increments are mapped directly to the robot arm's end-effector.

    - Example: If the VR controller moves 5 cm to the right, the robot's end-effector also moves 5 cm to the right. The same principle applies to orientation changes (pitch, yaw, roll).

4. **Disengagement**

    - Releasing the button **disengages** the system.
    - The robot arm holds its current position and orientation.
    - The user can freely reposition their hand and controller without affecting the robot.
    - This allows the operator to "re-center" comfortably, just like repositioning a mouse on a mouse pad, without causing unintended jumps in the robot's movement.

Practical convenience for efficient teloperation: This incremental, relative control scheme provides a safe, intuitive, and stable way to teleoperate a robot arm in VR, combining precision with user comfort.

Pseudocode

The pseudocode below implements a relative teleoperation control scheme using the variables hand_entry_pose and robot_entry_pose to record the initial positions and orientations of the VR controller and robot end-effector at the moment of engagement (button_pressed). While the system is engaged (is_engaged == True), it continuously calculates the relative movement increments—position_increment and orientation_increment—by subtracting the entry pose from the current hand pose (hand_current_pose). These increments are scaled using position_scale and orientation_scale, then added to the robot's entry pose to compute the new target pose (target_robot_pose). This target is sent to the robot via send_robot_command() to update its motion. When the button is released (button_released), the system disengages and the robot holds its position, allowing the user to reposition their hand freely.

```
# Data structures
Pose:
   position: Vector3  # (x, y, z)
   orientation: Rotation3D  # (pitch, yaw, roll) or quaternion

# State variables
is_engaged = False
hand_entry_pose = Pose()
robot_entry_pose = Pose()

# Scaling coefficients
position_scale = 1.0      # e.g., 0.5 for half-speed movement
orientation_scale = 1.0    # e.g., 0.8 for damped rotation

# Main loop
while system_is_running:
   hand_current_pose = get_hand_pose()     # from VR controller
   robot_current_pose = get_robot_pose()    # from robot sensors

   if button_pressed and not is_engaged:
      # Engagement: record entry poses
      hand_entry_pose = hand_current_pose
      robot_entry_pose = robot_current_pose
      is_engaged = True

   elif button_released and is_engaged:
      # Disengagement: freeze robot
      is_engaged = False

   if is_engaged:
      # Compute relative increments
      position_increment = hand_current_pose.position - hand_entry_pose.position
      orientation_increment = hand_current_pose.orientation - hand_entry_pose.orientation

      # Apply scaling
      scaled_position = position_increment * position_scale
      scaled_orientation = orientation_increment * orientation_scale

      # Map to robot target pose
      target_robot_pose.position = robot_entry_pose.position + scaled_position
      target_robot_pose.orientation = robot_entry_pose.orientation + scaled_orientation

      # Send command to robot
      send_robot_command(target_robot_pose)
```