Teleoperation Control Architecture (Joint Space Teleop)

1. Overview

This document describes the software architecture of a simple yet scalable teleoperation system for a 6-DOF robotic manipulator using ROS 2. The system is designed with clear separation of concerns, allowing easy extension to task-space control, inverse kinematics, force control, or real hardware integration.

The architecture is composed of two primary custom nodes:

1. **KeyboardTeleop** – Human input interface
2. **JointSpaceController** – Joint-level motion controller

Supporting nodes include:

- robot_state_publisher
- RViz

---

2. High-Level Architecture

Human Operator
    |
   V
KeyboardTeleop Node
    |
   V **(/teleop/pos_delta : geometry_msgs/Twist)**
JointSpaceController Node
    |
   V **(/joint_states : sensor_msgs/JointState)**
robot_state_publisher
    |
   V **(/tf)**
RViz Visualization

---

3. Node 1: KeyboardTeleop

3.1 Purpose

The **KeyboardTeleop** node captures human keyboard input and converts it into incremental motion commands.

3.2 Responsibilities

- Read keyboard input in non-blocking mode
- Map key presses to incremental motion commands
- Publish motion deltas as a ROS message

3.3 Interfaces

Publisher

- Topic: /teleop/pos_delta
- Type: geometry_msgs/Twist

3.4 Message Semantics

Generally Twist is typically used for velocities, in this system it represents *incremental joint commands*:

Twist Field	Meaning
linear.x	Δ Joint 1
linear.y	Δ Joint 2
linear.z	Δ Joint 3
angular.x	Δ Joint 4
angular.y	Δ Joint 5
angular.z	Δ Joint 6

3.5 Internal Flow

```
Keyboard Press
    |
   V
Key Mapping
    |
   V
Create Twist Message
    |
   V
Publish /teleop/pos_delta
```

3.6 Design Characteristics

Easily replaceable with joystick, haptic device, SpaceMouse and other custom made devices(Axis mapping matters)

---

4. Node 2: JointSpaceController

4.1 Purpose

The JointSpaceController node converts incremental teleoperation commands into valid joint states and publishes them for visualization and downstream control.

4.2 Responsibilities

- Maintain the robot's joint configuration
- Initialize the robot to a safe home pose
- Integrate incremental joint commands
- Enforce safety limits
- Publish joint states

4.3 Internal State

The controller maintains an internal joint vector:

q = [q1, q2, q3, q4, q5, q6]

- Units: Degrees
- Initial pose: [90°, 0°, 90°, 0°, 90°, 0°]

4.4 Interfaces

Subscriber

- Topic: /teleop/pos_delta
- Type: geometry_msgs/Twist

Publisher

- Topic: /joint_states
- Type: sensor_msgs/JointState

4.5 Control Loop Logic

```
Receive Δ Command
   |
  V
Map to Joint Increments
   |
  V
Integrate with Current Joint State
   |
  V
Apply Joint Limits (Safety limits)
   |
  V
Publish /joint_states
```

4.6 Key Design Decisions

Incremental Control

- Enables smooth teleoperation
- Suitable for Cartesian and force control extensions

Separation of Concerns

- Input handling and robot control are fully decoupled

5. Supporting Node: robot_state_publisher

5.1 Role

- Reads the robot URDF
- Subscribes to /joint_states
- Publishes TF frames

5.2 Important Notes

- Contains **no control logic**
- Purely a kinematic transformation node

---

6. Visualization: RViz

6.1 Function

- Visualizes robot geometry
- Displays joint motion using TF

6.2 Data Dependency

/joint_states → robot_state_publisher → /tf → RViz

---

7. Layered System View

```
Human Interface Layer
(KeyboardTeleop)
          |
          | Δ commands
         V
 Control Layer
 (JointSpaceController)
         |
         | JointState
        V
```

```
Kinematic Layer
(robot_state_publisher)
         |
         | TF
        V
 Visualization Layer
      (RViz)
```

---

8. Extensibility

This architecture is intentionally designed to scale:

- **Joint-space to Task-space**: Replace JointSpaceController with a Cartesian IK controller
- **Keyboard to Haptic device or Joystick**: Replace KeyboardTeleop only
- **Simulation to Real hardware**: Replace joint publisher with ros2_control hardware interface
- **Position to Force control**: Add impedance/admittance logic inside the controller layer

---

9. Summary

The presented architecture follows best practices used in both industrial and medical robotic systems:

- Modular and maintainable
- Deterministic control flow
- Safe initialization and ownership of joint states
- Ready for real-time, task-space, and force-compliant extensions

This design forms a strong foundation for advanced teleoperation and medical robotics applications.