ROS System Architecture Guide

Understanding the PAROL6 ROS 2 Control Pipeline

This guide explains how commands flow from RViz to the ESP32 through the ROS system.

📚 Table of Contents

🎯 System Overview

The Complete Pipeline

```
┌──────────┐    ┌──────────┐    ┌──────────┐    ┌──────────┐    ┌──────────┐
│  RViz    │───▶│  MoveIt  │───▶│  Driver  │───▶│  Serial  │───▶│  ESP32   │
│  (GUI)   │    │(Planner) │    │ (Python) │    │  Port    │    │(Motors)  │
└──────────┘    └──────────┘    └──────────┘    └──────────┘    └──────────┘
   User          Motion         Real Robot       /dev/          Hardware
 Interface       Planning       Controller       ttyUSB0
```

What Each Component Does

Component	Role	Input	Output
RViz	User interface	Mouse/keyboard	Goal pose
MoveIt	Motion planner	Start/goal poses	Trajectory
Driver	Hardware interface	Trajectory	Serial commands
Serial	Communication	Commands	Bytes to ESP32
ESP32	Motor controller	Serial commands	Motor signals

🔧 Component Details

1. RViz (Visualization & Interaction)

File: ROS 2 built-in package
What it is: 3D visualization tool for ROS

Responsibilities:

- Display robot model
- Show interactive markers (orange sphere + arrows)
- Send goal poses to MoveIt
- Visualize planned trajectories

Key Features:

- **Interactive Markers:** Drag to set goal pose
- **Motion Planning Panel:** Plan and Execute buttons
- **Displays:** RobotModel, TF frames, planning scene

RViz Configuration:

```
# parol6_moveit_config/rviz/moveit.rviz
MotionPlanning:
  Planning Request:
    Query Goal State: true        # Enable interactive markers
    Interactive Marker Size: 0.2  # Marker size
    Planning Group: parol6_arm    # Which joints to plan for
```

2. MoveIt (Motion Planning)

Package: `moveit_ros_move_group`
Node: `move_group`

Responsibilities:

- Path planning (obstacle avoidance)
- Inverse kinematics (pose → joint angles)
- Trajectory generation (smooth motion)
- Collision checking

Key Concepts:

Planning Group:

```
<!-- parol6_moveit_config/config/parol6.srdf -->
<group name="parol6_arm">
  <joint name="base_joint"/>
  <joint name="shoulder_joint"/>
  <joint name="elbow_joint"/>
  <joint name="wrist_pitch_joint"/>
  <joint name="wrist_roll_joint"/>
  <joint name="gripper_joint"/>
</group>
```

Controller Configuration:

```yaml
# parol6_moveit_config/config/moveit_controllers.yaml
controller_names:
  - parol6_arm_controller

parol6_arm_controller:
  type: FollowJointTrajectory
  action_ns: follow_joint_trajectory
  joints:
    - base_joint
    - shoulder_joint
    # ... etc
```

What MoveIt Outputs:

A `JointTrajectory` message containing:

```
trajectory:
  points:
    - positions: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
      velocities: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
      accelerations: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
      time_from_start: {sec: 0, nanosec: 0}
    - positions: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
      velocities: [0.5, 0.3, 0.2, 0.1, 0.4, 0.1]
      accelerations: [0.05, 0.03, 0.02, 0.01, 0.04, 0.01]
      time_from_start: {sec: 0, nanosec: 50000000}  # 50ms
    # ... many more points
```

3. Robot Driver (Hardware Interface)

File: `parol6_driver/parol6_driver/real_robot_driver.py`
Node: `real_robot_driver`

Responsibilities:

- Receive trajectories from MoveIt
- Format commands for ESP32
- Send via serial port
- Log all commands to CSV
- Handle acknowledgments

Class Structure:

```python
class RealRobotDriver(Node):
    def __init__(self):
        # Setup serial connection
```

```
        self.ser = serial.Serial('/dev/ttyUSB0', 115200)

        # Setup action server (receives from MoveIt)
        self._action_server = ActionServer(
            self,
            FollowJointTrajectory,
            'parol6_arm_controller/follow_joint_trajectory',
            self.execute_callback
        )

        # Setup logging
        self.log_writer = csv.writer(...)
        self.seq_counter = 0
```

Action Server Pattern:

```
def execute_callback(self, goal_handle):
    # 1. Get trajectory from MoveIt
    trajectory = goal_handle.request.trajectory

    # 2. Execute each point
    for point in trajectory.points:
        positions = point.positions      # [J1, J2, J3, J4, J5, J6]
        velocities = point.velocities    # (we log but don't send)
        accelerations = point.accelerations  # (we log but don't send)

        # 3. Format command for ESP32
        cmd = f"<{self.seq_counter},{','.join([f'{p:.4f}' for p in 
positions])}>\n"

        # 4. Send to ESP32
        self.ser.write(cmd.encode())

        # 5. Log to CSV (with vel/acc)
        self.log_writer.writerow([
            self.seq_counter,
            timestamp,
            *positions,
            *velocities,
            *accelerations,
            cmd.strip()
        ])

        self.seq_counter += 1

        # 6. Wait for next point
        time.sleep(0.05)  # 50ms

    # 7. Report success to MoveIt
    goal_handle.succeed()
    return FollowJointTrajectory.Result()
```

4. Serial Communication

Port: `/dev/ttyUSB0` (USB-to-Serial adapter)
Baud Rate: 115200
Protocol: Text-based, newline-terminated

Message Format:

```
TX (PC → ESP32): <SEQ,J1,J2,J3,J4,J5,J6>\n
RX (ESP32 → PC): <ACK,SEQ,TIMESTAMP_US>\n
```

5. ESP32 Firmware

File: `esp32_benchmark_idf/main/benchmark_main.c`
See: DEVELOPER_GUIDE.md for details

What it does:

- Parse incoming commands
- Detect packet loss
- Send ACKs
- Control motors (to be implemented)

🔄 Data Flow Walkthrough

Example: Moving the Robot

Step 1: User Interaction

User drags interactive marker in RViz to new position:

```
Goal Pose:
  position: {x: 0.3, y: 0.2, z: 0.5}
  orientation: {x: 0, y: 0, z: 0, w: 1}
```

Step 2: User Clicks "Plan"

RViz sends goal to MoveIt via ROS action:

```
/move_group/goal [moveit_msgs/action/MoveGroup]
  request:
    goal_constraints:
      position_constraints: [...]
      orientation_constraints: [...]
```

Step 3: MoveIt Plans Path

MoveIt:

1. Runs inverse kinematics: `pose → joint angles`
2. Plans collision-free path
3. Generates smooth trajectory with vel/acc profiles
4. Returns trajectory preview to RViz

Step 4: User Clicks "Execute"

RViz tells MoveIt to execute → MoveIt sends to driver:

```
/parol6_arm_controller/follow_joint_trajectory/goal
  goal:
    trajectory:
      joint_names: [base_joint, shoulder_joint, ...]
      points:
        - positions: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
          time_from_start: 0s
        - positions: [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]
          velocities: [0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
          time_from_start: 0.05s
        # ... ~20-30 points total
```

Step 5: Driver Formats & Sends

For each trajectory point:

```
# Point 0:
positions = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
cmd = "<0,0.0000,0.0000,0.0000,0.0000,0.0000,0.0000>\n"
serial.write(cmd)  # → ESP32

# Point 1 (50ms later):
positions = [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]
cmd = "<1,0.0500,0.1000,0.1500,0.2000,0.2500,0.3000>\n"
serial.write(cmd)  # → ESP32

# etc...
```

Step 6: ESP32 Receives & Processes

```
// ESP32 parses:
int seq = 1;
float joints[6] = {0.05, 0.10, 0.15, 0.20, 0.25, 0.30};

// Sends ACK:
```

```
printf("<ACK,1,12345678>\n");  // → PC

// Controls motors:
move_motors(joints);  // (to be implemented)
```

Step 7: Driver Logs

Simultaneously, driver logs to CSV:

```
1,1736781234567890,2026-01-
13T02:30:00,0.05,0.10,0.15,0.20,0.25,0.30,0.5,0.5,0.5,0.5,0.5,0.5,0.05,0.05
,0.05,0.05,0.05,0.05,"<1,0.05,0.10,0.15,0.20,0.25,0.30>"
```

📡 ROS 2 Concepts Used

1. Topics (Publish/Subscribe)

What: One-to-many communication streams

Used For:

- Joint states: `/joint_states`
- Robot description: `/robot_description`
- Planning scene: `/monitored_planning_scene`

Example - Publishing Joint States:

```python
from sensor_msgs.msg import JointState

joint_pub = self.create_publisher(JointState, '/joint_states', 10)

msg = JointState()
msg.header.stamp = self.get_clock().now().to_msg()
msg.name = ['base_joint', 'shoulder_joint', ...]
msg.position = [0.1, 0.2, 0.3, ...]

joint_pub.publish(msg)
```

2. Actions (Goal-Status-Result)

What: Long-running tasks with feedback

Used For:

- MoveIt motion execution
- Trajectory following

Action Structure:

```
Goal:     "Execute this trajectory"
          ↓
Feedback: "Currently at point 5 of 20"
          ↓
Result:   "SUCCESS" or "ABORTED"
```

Action Server (in driver):

```python
from rclpy.action import ActionServer
from control_msgs.action import FollowJointTrajectory

self._action_server = ActionServer(
    self,
    FollowJointTrajectory,
    'parol6_arm_controller/follow_joint_trajectory',
    self.execute_callback  # Called when goal received
)
```

Action Client (MoveIt uses internally):

```python
from rclpy.action import ActionClient

client = ActionClient(self, FollowJointTrajectory,
                      'parol6_arm_controller/follow_joint_trajectory')

# Send goal
client.send_goal_async(goal_msg)
```

3. Parameters

What: Configuration values

Example - In driver:

```python
self.declare_parameter('enable_logging', True)
self.declare_parameter('log_dir', '/workspace/logs')

self.enable_logging = self.get_parameter('enable_logging').value
```

Set from launch file:

```
```

```
Node(
    package='parol6_driver',
    executable='real_robot_driver',
    parameters=[{
        'enable_logging': True,
        'log_dir': '/workspace/logs'
    }]
)
```

🛠️ Interacting with the System

Monitor Topics

```
# List all topics
ros2 topic list

# See topic info
ros2 topic info /joint_states

# Monitor messages
ros2 topic echo /joint_states

# Check message rate
ros2 topic hz /joint_states
```

Send Test Commands

Method 1: Command Line

```
# Publish joint state
ros2 topic pub /joint_states sensor_msgs/msg/JointState \
  "{name: ['base_joint'], position: [0.5]}"
```

Method 2: Python Script

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState

class TestPublisher(Node):
    def __init__(self):
        super().__init__('test_pub')
        self.pub = self.create_publisher(JointState, '/joint_states', 10)

    def send_position(self, angles):
```

```
        msg = JointState()
        msg.name = ['base_joint', 'shoulder_joint', ...]
        msg.position = angles
        self.pub.publish(msg)

# Usage:
rclpy.init()
node = TestPublisher()
node.send_position([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
```

Inspect Running Nodes

```
# List nodes
ros2 node list

# Node info
ros2 node info /real_robot_driver

# Check connections
ros2 node info /move_group
```

Debug Actions

```
# List actions
ros2 action list

# Action info
ros2 action info /parol6_arm_controller/follow_joint_trajectory

# Send test goal
ros2 action send_goal /parol6_arm_controller/follow_joint_trajectory \
  control_msgs/action/FollowJointTrajectory "{...}"
```

🚀 Adding Custom Functionality

Example 1: Add Custom Motion Planner

Create new node that sends to driver:

```
from rclpy.action import ActionClient
from control_msgs.action import FollowJointTrajectory
from trajectory_msgs.msg import JointTrajectoryPoint

class CustomPlanner(Node):
    def __init__(self):
```

```
        super().__init__('custom_planner')

        # Connect to driver
        self.client = ActionClient(
            self,
            FollowJointTrajectory,
            'parol6_arm_controller/follow_joint_trajectory'
        )

    def execute_custom_motion(self):
        # Create trajectory
        goal = FollowJointTrajectory.Goal()
        goal.trajectory.joint_names = ['base_joint', 'shoulder_joint', ...]

        # Add points
        point1 = JointTrajectoryPoint()
        point1.positions = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        point1.time_from_start.sec = 0

        point2 = JointTrajectoryPoint()
        point2.positions = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
        point2.time_from_start.sec = 1

        goal.trajectory.points = [point1, point2]

        # Send to driver
        self.client.send_goal(goal)
```

Example 2: Add Feedback Monitoring

Monitor what driver is executing:

```
from rclpy.action import ActionClient

class FeedbackMonitor(Node):
    def __init__(self):
        super().__init__('feedback_monitor')

        self.client = ActionClient(
            self,
            FollowJointTrajectory,
            'parol6_arm_controller/follow_joint_trajectory'
        )

    def send_goal_with_feedback(self, goal_msg):
        future = self.client.send_goal_async(
            goal_msg,
            feedback_callback=self.feedback_callback
        )

    def feedback_callback(self, feedback_msg):
```

```
        # Called periodically during execution
        actual_pos = feedback_msg.feedback.actual.positions
        print(f"Current position: {actual_pos}")
```

Example 3: Bypass MoveIt (Direct Control)

Send commands directly to driver:

```
class DirectController(Node):
    def __init__(self):
        super().__init__('direct_controller')
        self.client = ActionClient(...)

    def move_to_position(self, positions):
        """Skip MoveIt, send position directly to driver"""
        goal = FollowJointTrajectory.Goal()
        goal.trajectory.joint_names = [...]

        # Single point trajectory
        point = JointTrajectoryPoint()
        point.positions = positions
        point.time_from_start.sec = 1

        goal.trajectory.points = [point]

        # Bypasses MoveIt planning entirely!
        self.client.send_goal(goal)
```

Example 4: Add Vision Integration

Connect camera to motion planning:

```
from sensor_msgs.msg import Image

class VisionController(Node):
    def __init__(self):
        super().__init__('vision_controller')

        # Subscribe to camera
        self.camera_sub = self.create_subscription(
            Image,
            '/camera/image_raw',
            self.image_callback,
            10
        )

        # Connect to driver
        self.motion_client = ActionClient(...)
```

```
    def image_callback(self, img_msg):
        # Process image (detect object)
        target_pose = self.detect_target(img_msg)

        # Plan path to target
        # (Use MoveIt or custom planner)

        # Execute
        self.move_to_pose(target_pose)
```

📊 System Diagram (Detailed)

```
                         ROS 2 System Architecture

   ┌───────────────────────────────────────────────────────────────┐
   │                           RViz (GUI)                          │
   │  ┌─────────────────────┐      ┌───────────────────────────┐   │
   │  │ Interactive Marker  │─────▶│ Motion Planning Panel     │   │
   │  └─────────────────────┘      └───────────────────────────┘   │
   └───────────────────────────────────────────────────────────────┘
                                    │ /move_group/goal (Action)
                                    ▼
   ┌───────────────────────────────────────────────────────────────┐
   │                      move_group (MoveIt)                      │
   │  ┌──────────────┐  ┌───────────────┐  ┌──────────────────┐    │
   │  │ IK Solver    │─▶│ Path Planner  │─▶│ Trajectory Gen   │    │
   │  │ (KDL)        │  │(OMPL)         │  │(Time Optimal)    │    │
   │  └──────────────┘  └───────────────┘  └──────────────────┘    │
   └───────────────────────────────────────────────────────────────┘
                                    │ follow_joint_trajectory (Action)
                                    ▼
   ┌───────────────────────────────────────────────────────────────┐
   │            real_robot_driver (Python Node)                    │
   │  ┌───────────────────┐    ┌─────────────────┐                 │
   │  │ Action Server     │───▶│ Serial Writer   │─────┐           │
   │  └───────────────────┘    └─────────────────┘     │           │
   │  ┌───────────────────┐                            │           │
   │  │ CSV Logger        │────────────────────────────┘           │
   │  └───────────────────┘                                        │
   └───────────────────────────────────────────────────────────────┘
                                    │ Serial: <SEQ,J1,...,J6>
                                    ▼
                          /dev/ttyUSB0 (115200 baud)
                                    │
                                    ▼
   ┌───────────────────────────────────────────────────────────────┐
   │                 ESP32 (C Firmware)                            │
   │  ┌──────────────┐  ┌───────────────┐  ┌──────────────────┐    │
   │  │ U ART RX     │─▶│ Parser        │─▶│ Motor Control    │    │
   │  └──────────────┘  └───────────────┘  └──────────────────┘    │
```

```
│    ┌──────────────┐                                            │
│    │ UART TX(ACK) │◄───────────────────────────────────────────┘
│    └──────────────┘                                            │
└──────────────────────────────────┬───────────────────────────────┘
                                   │
                                   ▼
                            Physical Motors
```

📚 Further Reading

ROS 2 Tutorials:

- https://docs.ros.org/en/humble/Tutorials.html

MoveIt 2 Documentation:

- https://moveit.ros.org/

Action Servers/Clients:

- https://docs.ros.org/en/humble/Tutorials/Intermediate/Writing-an-Action-Server-Client/Cpp.html

Our Project Docs:

- ESP32 DEVELOPER_GUIDE.md - ESP32 firmware details
- TESTING_WITH_ROS.md - Full pipeline testing
- GET_STARTED.md - Quick setup for new teammates

Questions? Ask the team or check ROS 2 documentation!