Contents

PAROL6 Project Report

This is a comprehensive technical description of the PAROL6 project, verifying all branches and configurations directly from the source code.

1. Project Overview

- **Robot**: PAROL6 (6-DOF Robotic Arm).
- **Simulation**: Ignition Gazebo (primary) & Gazebo Classic (legacy).
- **Control Stack**: ROS 2 Humble + MoveIt 2 + ros2_control.
- **Middleware**: Micro-ROS for ESP32.

2. Docker Architecture

The container content is defined by `Dockerfile` and built via `scripts/setup/rebuild_image.sh`. * **Image Name**: `parol6-ultimate:latest` * **Base Image**: `osrf/ros:humble-desktop` * **Key Components**: * **Gazebo**: Both Classic and Ignition (Fortress) are verified installed. * **MoveIt 2**: `ros-humble-moveit` and `ros-humble-moveit-ros-visualization`. * **Hardware Interface**: `ros-humble-gazebo-ros2-control`. * **Build Process**: The `rebuild_image.sh` script halts any running containers, rebuilds from `Dockerfile`, and tags it `parol6-ultimate:latest`. It does **not** rely on pulling from Docker Hub by default; it builds locally.

3. ROS 2 Architecture (Verified)

- **Package**: `parol6` (CMake)
- **Launch Flow**:
  1. **Starts**: `ignition.launch.py`.
  2. **Spawns**: Robot in Ignition Gazebo (`create` nodes).
  3. **Bridges**: `ros_ign_bridge` connects Ignition topics (`/clock`, `/cmd_vel`) to ROS 2.
  4. **Controllers**: Spawns `joint_state_broadcaster` and `parol6_arm_controller`.
- **MoveIt Integration**: `add_moveit.sh` is a separate script run *after* simulation start. It launches `Movit_RViz_launch.py` to bring up the MoveGroup node and RViz 2 independently, preventing race conditions.

4. Branch Description (Detailed)

The repository uses a branch-based feature workflow.

Branch	Verified Content & Purpose
`ros2_controller`	**(Active)** Main dev branch. Configured for `ign_ros2_control` with `JointTrajectoryController`.
`main`	Stable base. Contains core URDF and basic configs.

Branch	Verified Content & Purpose
`xbox-controller`	Contains `xbox_direct_control.py`. **Logic**: Subscribes to `/joy`, integrates velocity to position, and sends `FollowJointTrajectory` goals. It is a custom "servo" implementation.
`mobile-ros`	Contains `ros2_ws/src/mobile_bridge`. **Logic**: A Python bridge node (`mobile_bridge.py`) that likely uses WebSockets or TCP to communicate with a mobile app (Flutter/React Native).
`ESP32-main`	Contains `ros2_ws/src/serial_publisher`. **Logic**: A Python node (`serial_node.py`) designed to talk to the ESP32 via USB Serial, acting as a bridge for hardware that doesn't use micro-ROS directly.
`xbox_camera`	Integration for Kinect V2. Contains `install_kinect.sh` and `Dockerfile` updates for `libfreenect2`.
`microros_espidf`	Located in `microros_esp32` folder (in main). Structure confirms it is an **ESP-IDF Component**, meaning the firmware is built using Espressif's IDF tools, not Arduino.

5. File & Script Verification

- **`start_ignition.sh`**: The master entry point.
  - Mounts local directory to `/workspace` (hot-reloading code).
  - Enables GPU acceleration (`--gpus all` or `/dev/dri`).
  - Runs **`colcon build`** inside the container on **every run**.
- **`add_moveit.sh`**:
  - Checks for running container.
  - Launches `Movit_RViz_launch.py` inside the container.
- **`mobile_control/`**: In the current branch, this directory is largely empty/placeholder. The actual code resides in the `mobile-ros` branch.

6. Detailed ROS 2 Architecture

A. Node Graph & Data Flow

1. Core Simulation Nodes

- `ros_ign_bridge` (`parameter_bridge`):
  - **Function**: Bridges basic simulation data.
  - **Topics**: `/clock` (Ignition -> ROS), `/model/parol6/pose` (Ignition -> ROS).
- `ign_ros2_control` (Plugin inside Ignition):
  - **Function**: Acts as the "Hardware Interface". It allows the ROS `controller_manager` to see the Ignition model as if it were a real robot.
  - **Services**: Exposes the standard `controller_manager` services (list/load/switch controllers).
- `spawner` (controller_manager):
  - **Function**: Loads the specific controllers defined in `ros2_controllers.yaml`.
  - **Joint State Broadcaster**: Reads joint positions from Ignition and publishes `/joint_states`.
  - **Joint Trajectory Controller** (`parol6_arm_controller`): Listens for trajectory goals and commands the Ignition joints.

2. Motion Planning Nodes (MoveIt)

- `move_group`:
  - **Function**: The central motion planning brain.
  - **Inputs**: `/joint_states` (Current position), `/robot_description` (URDF), `/tf` (Transforms).
  - **Outputs**: `/parol6_arm_controller/follow_joint_trajectory/goal` (Sends planned paths to execution).
  - **Services**: Provides `/compute_ik`, `/plan_kinematic_path` for external nodes.
- `rviz2`:
  - **Function**: Visualization.
  - **Plugin**: `MotionPlanning` plugin connects directly to `move_group` to allow drag-and-drop target setting and visualizes the `/move_group/display_planned_path` topic.

3. Branch-Specific Nodes

- `xbox_direct_control` (Node: `xbox_direct_control`):
  - **Source**: `xbox-controller` branch.
  - **Subscriptions**: `/joy` (Joystick data), `/joint_states`.
  - **Actions**: Client for `/parol6_arm_controller/follow_joint_trajectory`.
  - **Logic**: Runs a 10Hz control loop that integrates joystick velocity into a position target and streams trajectory points.
- `serial_publisher` (Node: `serial_publisher`):

  - **Source**: `ESP32-main` branch.
  - **Publisher**: `/esp_serial` (`std_msgs/String`).
  - **Logic**: Reads raw text lines from `/dev/ttyUSB0` (default) and publishes them. This appears to be a uni-directional debug tool, NOT a full robot controller.
- `extract_from_bag`:
  - **Source**: `xbox_camera` branch (`vision_work/`).
  - **Logic**: Off-line processing tool to extract images/data from recorded ROS bags, likely for training vision models.

B. Connection Diagram

B. Connection Diagram

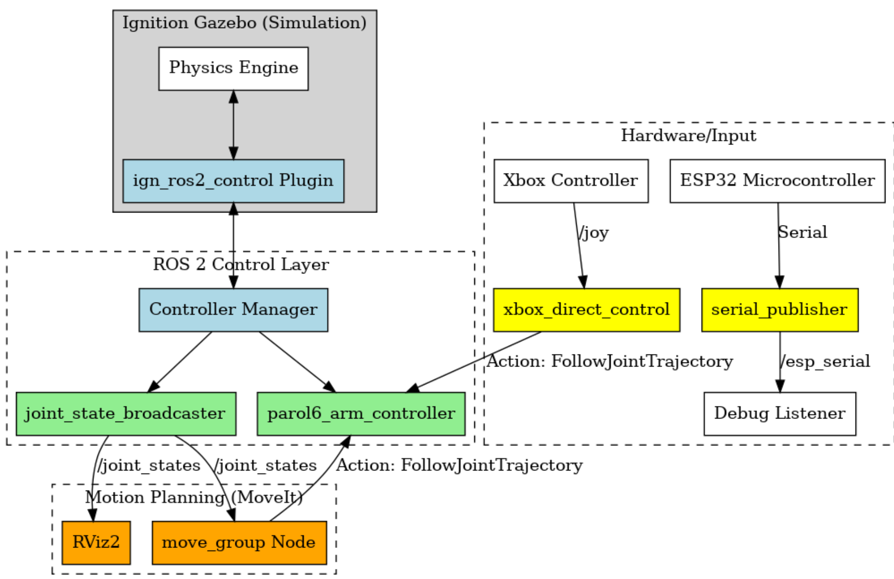

Figure 1: Connection Diagram

C. Key Packages & Configs

1. **`parol6` (The Description)**:
   - **`urdf/PAROL6.urdf`**: Defines the physical `link` and `joint` structure.
   - **Important**: Contains the `<ros2_control>` tag which loads the `ign_ros2_control/IgnitionSystem` plugin. This is the *critical link* between URDF and Ignition.
   - **`config/ros2_controllers.yaml`**: Defines names and types of controllers (`joint_trajectory_controller/JointTrajectoryController`).
2. **`parol6_moveit_config` (The Brain)**:

- **`config/parol6.srdf`**: Defines the "planning group" (`parol6_arm`) containing joints L1-L6.
- **`config/moveit_controllers.yaml`**: Tells MoveIt *how* to talk to the controllers defined in the `parol6` package. It maps the `parol6_arm` group to the `parol6_arm_controller` action namespace.

7. Topic & Network Analysis

A. Network Visualization (RQT Graph Style)

This diagram accurately represents the active ROS 2 node graph during simulation.

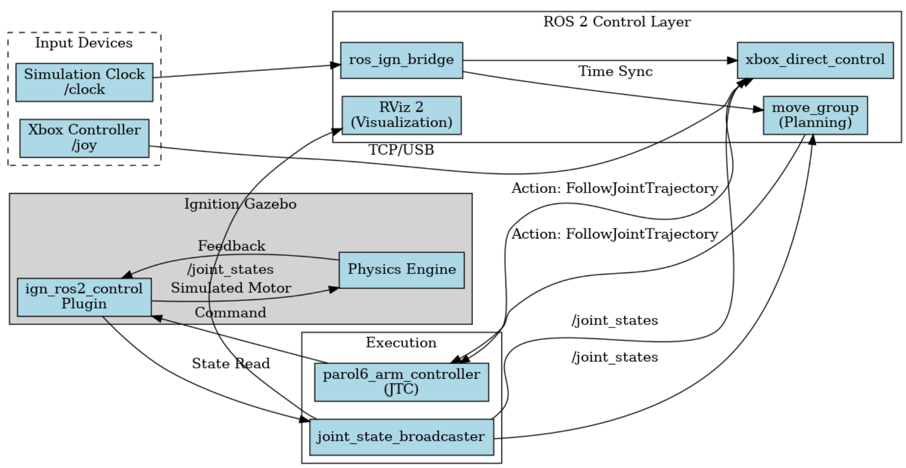

Figure 2: System Architecture Visual

B. System Topic Table

A complete list of active topics, their types, and connected nodes.

Topic Name	Message Type	Publishers	Subscribers	Description
`/joy`	`sensor_msgs/Joy`	`joy_node`	`xbox_direct_control`	Raw button/axis data from Gamepad.
`/joint_states`	`sensor_msgs/JointState`	`joint_state_broadcaster`	`move_group`, `rviz2`, `xbox_direct_control`, `robot_state_publisher`	Real-time angles of joints L1-L6.

Topic Name	Message Type	Publishers	Subscribers	Description
`/robot_description`	`std_msgs/String`	`robot_state_publisher`	`move_group`, `rviz2`	The XML URDF string (Static).
`/tf`	`tf2_msgs/TFMessage`	`robot_state_publisher`	`move_group`, `rviz2`	Dynamic transforms (Moving links).
`/tf_static`	`tf2_msgs/TFMessage`	`robot_state_publisher`	`move_group`, `rviz2`	Static transforms (Fixed base).
`/clock`	`rosgraph_msgs/Clock`	`ros_gz_bridge`	All Nodes (`use_sim_time=True`)	Simulation time sync.
`/parol6_arm_controller/follow_joint_trajectory/goal`	`FollowJointTrajectory`	`move_group`, `xbox_direct_control`	`parol6_arm_controller`	The command target for the robot.
`/monitored_planning_scene`	`moveit_msgs/PlanningScene`	`move_group`	`rviz2`	The 3D world representation (obstacles + robot).
`/move_group/display_planned_path`	`moveit_msgs/DisplayTrajectory`	`move_group`	`rviz2`	The "Ghost" robot trail you see in RViz.
`/esp_serial`	`std_msgs/String`	`serial_publisher`	(Debug Tools)	**(ESP32-main branch only)** Raw serial debug data.

C. Key Services & Actions

Service/Action	Type	Server	Client	Purpose
`/compute_ik`	`GetPositionIK`	`move_group`	External / RViz	Calculate joint angles from XYZ pose.
`/plan_kinematic_path`	`GetMotionPlan`	`move_group`	External	Calculate collision-free path.

Service/Action	Type	Server	Client	Purpose
`/parol6_arm_controller/follow_joint_trajectory`	`FollowJointTrajectory`	`parol6_arm_controller`	`move_group` / `xbox_direct_control`	The “Move” command.

8. Component Responsibility Reference

A. Nodes Reference Table

Node Name	Package	Type	Responsibility
`xbox_direct_control`	`(custom)`	Python Node	Reads `/joy` and calculates position commands for the arm (Teleop).
`move_group`	`moveit_ros_move_group`	C++ Node	Central MoveIt planner; handles collisions, kinematics (IK), and path planning.
`rviz2`	`rviz2`	C++ GUI	Visualization interface for the user and MoveIt interaction.
`ros_ign_bridge`	`ros_ign_bridge`	C++ Bridge	Transfers messages (Clock, TF, Cmd) between ROS 2 and Ignition Gazebo.
`spawner`	`controller_manager`	Python Script	Loads and configures `ros2_control` controllers into the manager.
`robot_state_publisher`	`robot_state_publisher`	C++ Node	Publishes static and dynamic TFs based on URDF and joint states.

Node Name	Package	Type	Responsibility
`serial_publisher`	`serial_publisher`	Python Node	**(ESP32 Branch)** Reads raw serial data from microcontroller for debugging.

B. Packages Reference Table

Package Name	Location in Repo	Type	Description
`parol6`	`/PAROL6`	CMake	**Core Description**. Contains URDF, Meshes (`/meshes`), and Gazebo Launch files (`/launch`).
`parol6_moveit_config`	`/parol6_moveit_config`	CMake	**MoveIt Config**. Generated SRDF, kinematics.yaml, and MoveIt launch files.
`serial_publisher`	`ros2_ws/src/...`	Python	**ESP32 Bridge**. Handles serial communication in the `ESP32-main` branch.
`microros_espidf`	`/microros_esp32`	ESP-IDF	**Firmware**. Micro-ROS agent component for the ESP32 (Main Branch).

C. Topic Responsibility Matrix

This table categorizes topics by which subsystem "owns" or produces them, and who consumes them.

Subsystem & Responsibility	Topic Name	Message Type	Consumed By (Subscribers)
IGNITION GAZEBO (Simulation)			
Publishes Sim Time	`/clock`	`rosgraph_msgs/Clock`	**All Nodes** (Sync)
Publishes Ground Truth	`/model/parol6/pose`	`geometry_msgs/Pose`	**ros_ign_bridge**
ROS 2 CONTROL (Hardware Interface)			
Motor Feedback	`/joint_states`	`sensor_msgs/JointState`	**move_group**, **rviz2**, **xbox_direct_control**
Dynamic Transforms	`/tf`	`tf2_msgs/TFMessage`	**move_group**, **rviz2**
MOVEIT 2 (Motion Planning)			
Visualizing Plans	`/move_group/display_planned_path`	`moveit_msgs/DisplayTrajectory`	**rviz2**
Scene Awareness	`/monitored_planning_scene`	`moveit_msgs/PlanningScene`	**rviz2**
INPUT / BRIDGE (User Commands)			
GamePad Input	`/joy`	`sensor_msgs/Joy`	**xbox_direct_control**
Command Target	`/parol6_arm_controller/follow_joint_trajectory/goal`	`FollowJointTrajectory`	**parol6_arm_controller**

D. Services & Actions Responsibility Matrix

Subsystem & Responsibility	Name	Type	Called By (Clients)
MOVEIT 2 (Planning)			
Inverse Kinematics	`/compute_ik`	`GetPositionIK`	**External Nodes**, **RViz**
Path Planning	`/plan_kinematic_path`	`GetMotionPlan`	**External Nodes**
Execute Plan	`/move_group`	`MoveGroupAction`	**RViz**, **External Scripts**
ROS 2 CONTROL (Execution)			

Subsystem & Responsibility	Name	Type	Called By (Clients)
Execute Trajectory	`/parol6_arm_controller/follow_joint_trajectory`	FollowJointTrajectory	`move_group`, **xbox__direct__control**
Manage Controllers	`/controller_manager/list_controllers`	ListControllers	**spawner**
Manage Controllers	`/controller_manager/list_controllers`	ListControllers	**spawner**

9. Major Component Interaction Guide

This table provides a "Standard Usage" guide for the primary system components, explaining how they are intended to be used logically.

Component	Role	Key Inputs (Subscribes)	Key Outputs (Publishes)	How to Interact (Standard Usage)
MoveIt 2 (`move_group`)	**Planner**. Calculates collision-free paths.	`/joint_states`, `/tf`, `/robot_description`	`/parol6_arm_controller/follow_joint_trajectory/goal`, `/move_group/display_planned_path`	**To Move Robot:** Use the MoveGroup C++/Python Interface to set a target pose (XYZ) and call `.go()`. **Do NOT** publish to topics directly; use the MoveGroup API.
ROS 2 Control (`parol6_arm_controller`)	**Executor**. Interpolates points and drives motors.	`/parol6_arm_controller/follow_joint_trajectory/goal` (Action)	`/joint_states` (via broadcaster)	**To Send Motion:** Send a `FollowJointTrajectory` Action Goal. **To Read Motion:** Subscribe to `/joint_states`.

Component	Role	Key Inputs (Subscribes)	Key Outputs (Publishes)	How to Interact (Standard Usage)
Ignition Gazebo (`ros_ign_bridge`)	**Hardware**. Simulates physics and motors.	`/model/parol6/cmd_vel` (if using Twist), Control Commands (internal plugin)	`/clock`, `/model/parol6/pose`	**Standard:** Do not interact directly. Use ROS 2 Control (above). **Debug:** You can subscribe to `/model/parol6/pose` for ground truth, but use `/joint_states` for control loops.
Xbox Control (`xbox_direct_control`)	**Teleop**. Human-in-the-loop control.	`/joy`, `/joint_states`	`/parol6_arm_controller/follow_joint_trajectory/goal`	**Usage:** Press buttons on controller. **Logic:** It bypasses MoveIt and talks directly to the Controller for low-latency response.
RViz 2	**Visualizer**. Shows what the robot thinks.	`/tf`, `/robot_description`, `/move_group/display_planned_path`	*(Interactive Markers)*	**Usage:** Use "MotionPlanning" plugin to drag the end-effector and visualize plans before executing.

10. Common Developer Workflows

A. How to Insert the Microcontroller (Micro-ROS Layer)

To transition from Simulation to Real Hardware, you replace the **Ignition/Bridge** layer with the **Micro-ROS Layer**. 1. **The Concept**: The ESP32 acts as a "Hardware Bridge". It must run a Micro-ROS node that talks to the PC via USB Serial (UART) or Wi-Fi (UDP). 2. **Data Flow**: * **PC -> ESP32**: The ESP32 subscribes to `/parol6_arm_controller/follow_joint_trajectory` (or an intermediate topic like `/joint_commands`). It receives trajectory points (Position/Velocity) and drives the stepper motors. * **ESP32 -> PC**: The ESP32 publishes to `/joint_states`. It reads encoders/steps and sends back the *real* angles [L1…L6] so MoveIt knows where the robot is. 3. **Implementation**: * Use the `microros_esp32` component (verified in `microros_espidf` branch). * Run the **Micro-ROS Agent** on the PC (inside Docker): `bash ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0 --baudrate 115200` * Flash the ESP32 to connect to this agent.

B. How to Tune PID Controllers

If the robot wobbles or reacts too slowly in simulation: 1. **Edit**: `PAROL6/config/ros2_controllers.yaml`. * Look for `gains:` under `parol6_arm_controller`. * Adjust `p` (Proportional) for stiffness, `d` (Derivative) for damping. 2. **Apply**: * Stop the simulation (`Ctrl+C` or `stop.sh`). * Rebuild: `colcon build`. * Restart: `./start_ignition.sh`.

C. How to Add a Sensor (e.g., Camera)

1. **Model It**: Add a `<link>` (camera body) and `<joint>` (mount) to `PAROL6/urdf/PAROL6.urdf`.
2. **Simulate It**: Add a `<sensor>` tag inside a `<gazebo>` plugin block in the URDF. Use `type="camera"` or `type="depth"`.
3. **Bridge It**: Update `PAROL6/launch/ignition.launch.py` to bridge the new sensor topic (e.g., `/camera/image_raw`) from Ignition to ROS 2.

D. Alternative: Direct Serial Bridge (Any MCU / No Micro-ROS)

If you cannot use Micro-ROS (e.g., using Arduino Uno/Nano, or prefer simpler code), you can use a **Custom Serial Bridge**. 1. **Architecture**: `[ROS 2 Node] <---> [USB Serial] <---> [MCU Firmware]` 2. **The ROS 2 Node (Python)**: * **Input**: Subscribes to `/joint_states` (or your custom command topic). * **Logic**: Converts the list of 6 floats into a formatted string (e.g., `<J1,J2,J3,J4,J5,J6>`). * **Output**: Writes this string to `/dev/ttyUSB0` using the `pyserial` library. * **Feedback**: Reads lines from Serial and publishes them back to ROS (e.g., `/real_joint_states`). * *Reference*: The `serial_publisher` node in the `ESP32-main` branch is a basic example of the "Reader" part of this bridge. 3. **The Firmware (C++)**: * Standard `void loop()` that listens for

Serial data. * Parses the string `<...>` and runs `stepper.moveTo()`. * **Pros**: Works on *any* board, easier to debug with Serial Monitor. * **Cons**: No guaranteed timing (latency), you must invent your own data protocol.

12. Camera Integration (Xbox Kinect V2)

Based on the `xbox_camera` branch and standard ROS 2 perception workflows.

A. Driver Stack

The Kinect V2 is not plug-and-play; it requires a specific userspace driver and a bridge node. 1. **`libfreenect2`**: The core driver. It communicates directly with the USB 3.0 controller to fetch raw RGB and Depth packets. * *Installation*: Handled by `scripts/install_kinect.sh`. 2. **`kinect2_bridge`** (or `kinect_ros2`): The ROS 2 Node. It links against `libfreenect2` and publishes standard ROS messages.

B. Key Topics

Topic Name	Message Type	Purpose	Consumers
`/kinect2/qhd/image_color`	`sensor_msgs/Image`	Raw RGB video stream (960x540).	**Computer Vision** (YOLO, OpenCV), **RViz**
`/kinect2/qhd/image_depth_rect`	`sensor_msgs/Image`	Rectified depth map (mm distance).	**Navigation**, **3D Reconstruction**
`/kinect2/qhd/points`	`sensor_msgs/PointCloud2`	3D Point Cloud (XYZRGB).	**MoveIt 2** (Octomap), **RViz**
`/kinect2/camera_info`	`sensor_msgs/CameraInfo`	Calibration intrinsics (K matrix).	**Image Proc**, **Rectification Nodes**

C. MoveIt 2 Integration (Perception)

To allow the robot to "see" and avoid obstacles dynamically: 1. **Update Config**: Edit `parol6_moveit_config/config/sensors_3d.yaml`. 2. **Add Sensor**: `yaml sensors: - sensor_plugin: occupancy_map_monitor/PointCloudOctomapUpdater point_cloud_topic: /kinect2/qhd/points max_range: 2.0` 3. **Result**: MoveIt will consume the PointCloud and build a dynamic **Octomap** (represented as voxels) in the planning scene, preventing the arm from hitting objects seen by the Kinect.

D. Common Approaches for Vision Tasks

1. **Object Detection**: Subscribe to `/kinect2/qhd/image_color`. Use a separate node (e.g., `yolov8_ros`) to find bounding boxes like "Cup".

2. **Grasping**:
    - Take the center pixel (u,v) of the bounding box.
    - Look up the depth (d) at (u,v) in `/kinect2/qhd/image_depth_rect`.
    - Deproject (u,v,d) using `/kinect2/camera_info` to get 3D coordinates (X,Y,Z).
    - Send this (X,Y,Z) to MoveIt as a target pose.

13. Troubleshooting & Optimization

A. GPU Acceleration & Docker

The `start_ignition.sh` script is configured for **Direct Rendering (DRI)**, which works for most Intel/AMD/NVIDIA setups on Linux. * **The Check**: Inside the container (or outside), run `glxinfo | grep "OpenGL renderer"`. You should see your actual GPU (e.g., "NVIDIA GeForce…"), not "llvmpipe" (software rendering). * **NVIDIA Specifics**: If `glxinfo` fails or simulation is slow on NVIDIA: 1. Install **NVIDIA Container Toolkit** on your host machine: `sudo apt install nvidia-container-toolkit`. 2. Edit `start_ignition.sh`: Change `--device /dev/dri` to `--gpus all`. 3. Restart.

B. Micro-ROS Connection Issues

If the Agent says "Waiting for agent…" or doesn't connect: 1. **Permissions**: Ensure your user owns the port: `sudo chmod 666 /dev/ttyUSB0`. 2. **Reset Sequence**: * Start the Agent on PC *first*. * Press the **EN (Reset)** button on the ESP32. * Watch for "Session Established" logs. 3. **Baud Rate**: Ensure the baud rate in the C++ firmware (`rmw_uros_set_custom_transport`) matches the agent command (default `115200`).

C. X11 / GUI Not Opening

If `start_ignition.sh` fails with "Can't open display": 1. **Unlock X11**: Run `xhost +local:docker` on your host machine. 2. **Check Display**: Ensure `echo $DISPLAY` returns something like `:0` or `:1`.

14. Daily Operation Cheat Sheet (TL;DR)

Copy and paste these commands.

Task	Command / Action
Start Everything	`./start_ignition.sh` (Wait for Gazebo) -> New Terminal -> `./add_moveit.sh`
Stop Everything	`Ctrl+C` in all terminals -> `./stop.sh`

Task	Command / Action
Reset Simulation	Press **Orange “Reset” Button** in Ignition GUI (bottom left).
Record Data	`ros2 bag record -o my_data /joint_states /kinect2/qhd/image_color /kinect2/qhd/points`
Play Data	`ros2 bag play my_data`
List Topics	`ros2 topic list`
Echo Topic	`ros2 topic echo /joint_states` (Ctrl+C to stop)
View Graph	`rqt_graph`

15. Extensibility Guide

A. How to Add a Gripper

1. **URDF**: Edit `PAROL6/urdf/PAROL6.urdf`. Add a `<joint>` connecting `link_6` to your gripper base.
2. **SRDF**: Edit `parol6_moveit_config/config/parol6.srdf`. Add a new `<group name="gripper">` and define its joints/links.
3. **Controllers**: Update `PAROL6/config/ros2_controllers.yaml`. Add a `gripper_controller` (type: `PositionJointInterface`).
4. **MoveIt Controllers**: Update `parol6_moveit_config/config/moveit_controllers.yaml`. Add `gripper_controller` to the list.
5. **Rebuild**: Run `colcon build` inside the container.

B. How to Change Motors / Gear Ratios

1. **Limits**: Edit `PAROL6/urdf/PAROL6.urdf`. Find the `<joint>` tags.
   - Update `velocity="..."` (Max rad/s).
   - Update `effort="..."` (Max Nm).
2. **Transmissions**: If using `ros2_control` hardware interface, update the `mechanicalReduction` in the `<transmission>` block (if implemented) or adjust the steps/rad ratio in your microcontroller firmware.

16. Maintenance & Deployment

A. Exporting Your Work (Docker)

To share your exact setup with a colleague who doesn’t want to rebuild: 1. **Commit**: `docker commit parol6_dev parol6-export:v1` 2. **Save**: `docker save parol6-export:v1 | gzip > parol6_v1.tar.gz` 3. **Load (Colleague)**: `docker load < parol6_v1.tar.gz`

B. Adding New Packages

If you need a new ROS package (e.g., `ros-humble-yolo`): 1. **Try Online**: Enter container, run `sudo apt install ros-humble-yolo`. 2. **Make Permanent**: Add the line `RUN apt-get install -y ros-humble-yolo` to your `Dockerfile`. All future builds will have it.