Firmware & Hardware-in-the-Loop (HIL) Troubleshooting

This document is dedicated to solving deeply technical issues encountered when bridging the ROS 2 environment with the Teensy 4.1 hardware (or ESP32 simulation), specifically concerning the Hardware Abstraction Layer, Action Servers, and Real-Time control loop timing.

🛑 Controller / RViz Crashes

Issue 1: Controller Manager Segfaults on Launch (`exit code -11`)

Symptoms: When launching the hardware interface, the `ros2_control_node` crashes instantly, taking down the robot state publisher as well.

```
[ERROR] [ros2_control_node]: process has died [exit code -11]
```

Root Cause: In ROS 2, `rclcpp::Time` instances must be initialized with the same clock base before performing subtraction. Subtracting a default-initialized `rclcpp::Time` (which acts as `RCL_ROS_TIME`) from a monotonic `time` parameter passed into a hardware interface `read()` loop (`RCL_STEADY_TIME`) will throw an unhandled C++ exception that silently aborts the node.

Fix: Never directly subtract raw `rclcpp::Time` headers in hardware loops unless both are explicitly instantiated against `clock_.now()`. Instead, perform float arithmetic on their extracted seconds:

```
// Bad: Throws fatal exception
double dt = (time - last_rx_time_).seconds();

// Good: Safe explicit float math
double dt = std::abs(time.seconds() - last_rx_time_.seconds());
```

Issue 2: RViz Segfaults AFTER Successful Trajectory Execution

Symptoms: The terminal log shows that MoveIt executed the trajectory beautifully:

```
[move_group]: Completed trajectory execution with status SUCCEEDED
```

But RViz immediately throws an error and crashes the entire GUI:

```
[rviz2]: unknown result response, ignoring...
[ERROR] [rviz2]: process has died [exit code -11]
```

Root Cause: This is an Action Server namespace mismatch. In ROS 2 Humble, the Controller Manager automatically maps the result topic for the `joint_trajectory_controller`. If you explicitly declare `action_ns: follow_joint_trajectory` in your `moveit_controllers.yaml`, RViz gets double-nested (`/follow_joint_trajectory/follow_joint_trajectory`) and listens to a ghost pointer when trying to parse the success message.

Fix: Remove the redundant `action_ns` from `parol6_moveit_config/config/moveit_controllers.yaml`:

```
  parol6_arm_controller:
    type: FollowJointTrajectory
    # action_ns: follow_joint_trajectory  <-- DELETE THIS LINE
    default: true
```

Then re-build the config package:

```
colcon build --packages-select parol6_moveit_config
```

🕑 Timing & Trajectory Aborts

Issue 3: MoveIt Reports "TIMED_OUT" Despite Robot Moving

Symptoms: During Hardware-in-the-Loop (HIL) simulation or real execution with encoder noise, the RViz robot model animates perfectly, but the execution eventually aborts with:

```
[move_group]: Controller is taking too long to execute trajectory...
TIMED_OUT
```

Root Cause: MoveIt's execution monitor enforces mathematical state trajectory margins. If you comment out the `constraints` block in `ros2_controllers.yaml`, MoveIt defaults to its internal boundary (usually <0.01 rad). Even minuscule floating-point rounding errors caused by the UART feedback loop or serial parser will cause the monitor to instantly preempt the path for "violating constraints."

Fix: Re-instate the explicit constraints in `config/ros2_controllers.yaml`, but aggressively relax them to `999.0` to disable the preemption monitor altogether during spoofing validations:

```
    constraints:
      stopped_velocity_tolerance: 999.0
      goal_time: 0.0
      joint_L1: { trajectory: 999.0, goal: 999.0 }
      # ... repeat for all joints
```

Issue 4: HIL Spoofing Initialization Instability (`NaN` Jumps)

Symptoms: When running the robot offline (spoofing commands directly to state feedback to test planners), the arm jumps erratically or aborts instantly at the `start()` request.

Root Cause: MoveIt controller arrays initialize to `0.0`. If you aggressively spoof `hw_state_positions_[i] = hw_command_positions_[i]` continuously from boot, you will overwrite the actual starting URDF position of the arm with zeros, creating a massive start-point deviation that permanently condemns the trajectory tracking monitor.

Fix: Always explicitly filter out `NaN` and `0.0` initialization states before reflecting commands to state:

```
for (size_t i = 0; i < 6; ++i) {
  if (!std::isnan(hw_command_positions_[i])) {
    hw_state_positions_[i] = hw_command_positions_[i];
  }
}
```

Issue 5: RViz Throws "unknown goal response" and Segfaults Despite Robot Moving

Symptoms: You click "Plan and Execute" in RViz. The robot physically moves (or the simulation visualizer shows movement), but the MoveIt planner instantly says `MoveGroupInterface::move() failed or timeout reached` and you see:

```
[rviz2]: unknown goal response, ignoring...
[ERROR] [rviz2]: process has died [exit code -11]
```

Root Cause: Orphaned background processes! If you restart the HIL test or the Docker container without cleanly killing previous ROS 2 nodes, multiple `move_group` instances will run concurrently in the background. When RViz sends an Action Server request to execute a trajectory, *all of the orphaned `move_group` nodes* try to reply at the exact same millisecond. RViz receives multiple conflicting Action Server signatures, panics, immediately aborts the GUI request, and often segfaults.

Fix: Ensure your launch scripts forcefully purge all zombie `move_group`, `rviz2`, and `ros2_control_node` processes before spinning up a new instance:

```
# Add this to the top of your start_hil_test.sh script
echo "Cleaning up any orphaned background nodes..."
docker exec parol6_dev pkill -9 -f
"move_group|rviz2|ros2_control_node|robot_state_publisher" || true
```

⏱️ Hard Real-Time & Hardware Diagnostics

With the transition to the hard real-time Teensy 4.1 architecture (Phase 4), new physical and timing failure modes exist outside of the ROS domain.

Issue 6: Control Jitter Measuring > 1 µs

Symptoms: When profiling the 1 kHz `run_control_loop_isr` with an oscilloscope on the `ISR_PROFILER_PIN` or reading the DWT Cycle Counter telemetry, you notice the jitter (deviation from the strict 1000 µs interval) occasionally spikes to 15 µs or more.

Root Cause: If QuadTimers are used, jitter should be `< 1 µs`. A spike indicates an architectural violation. Common culprits:

1. **Rogue Software Interrupts**: A background library (e.g., USB Serial or a standard Arduino library) activated a higher-priority interrupt that preempted the `IntervalTimer`.
2. **Cache Misses**: You didn't place the critical control arrays (e.g., `AlphaBetaFilter` states) in Tightly Coupled Memory (TCM), causing the Cortex-M7 D-cache to stall the CPU while fetching data from slower RAM.
3. **ISR Printf**: Someone accidentally left a `Serial.print()` inside the control loop.

Fix: Ensure all arrays used inside the ISR are decorated with `EXTMEM` or explicitly routed to `DMAMEM`/TCM depending on the platform config. audit the `setup()` function to ensure no competing `IntervalTimers` or software interrupts are active.

Issue 7: SafetySupervisor Triggers "ISR Overrun" Fault

Symptoms: The robot abruptly halts, and the telemetry stream outputs a `FAULT_ISR_OVERRUN` error code. The system refuses to resume motion.

Root Cause: The `run_control_loop_isr()` execution time exceeded the 1000 µs tick window (less a safety margin). Based on our profiling, the math + safety checks should complete in `< 25 µs`. An overrun means a fatal mathematical hang or an infinite loop occurred inside `ControlLaw` or `AlphaBetaFilter`. A common trigger is accidentally introduced floating-point division by zero (e.g., dividing by `delta_t` when `delta_t` evaluates to `0.0f`), which can cause the FPU to stall or throw an exception that delays execution.

Fix:

1. **Remove Divisions**: Audit the filter and interpolator math. Replace `A / B` with `A * (1.0f / B)` pre-computed in constructors where possible.
2. **Check Unwrapping Logic**: Ensure the 360-degree `M_PI` encoder unwrap bounds cannot get stuck in an infinite `while()` loop if the sensor completely disconnects and sends garbage floating-point noise. Always use bounded `if()` statements for angle wrapping.