Exhaustive Analysis and Remediation Strategy for Collaborative Sawing Stability in Isaac Lab 2.3.0

1. Executive Summary

This report provides a comprehensive technical analysis of the instability, misalignment, and control artifacts observed in the run_hri_demo.py script utilized for a collaborative sawing experiment within the Isaac Lab 2.3.0 and Isaac Sim 5.1.0 environments. The user has reported persistent difficulties in achieving a stable, perpendicular resting state for a saw blade upon a log, characterized by uncommanded tilting and position adjustment struggles (oscillation) during the contact phase. These behaviors persist despite the explicit definition of target positions in the end-effector (EE) frame.

The investigation, grounded in the provided system logs, script architecture, and extensive Isaac Lab documentation, identifies a convergence of three distinct but compounding failure modes: kinematic frame discrepancies, inappropriate control topology for contact-rich manipulation, and physics engine simulation artifacts.

First, the analysis indicates a fundamental **Kinematic Frame Mismatch**. The discrepancy between the robot's nominal end-effector—typically the mounting flange or a default gripper frame like panda_hand—and the functional Tool Center Point (TCP) of the saw blade is causing the controller to drive the wrist to the log surface rather than the blade's cutting edge.[1] This geometric error forces the robot to rotate the wrist to satisfy the position constraint, resulting in the observed tilt.

Second, the current control implementation appears to utilize a stiff position-tracking controller, likely the **Differential Inverse Kinematics (DiffIK)** controller or a high-stiffness Operational Space Controller.[3] These controllers are mathematically ill-suited for physical contact tasks involving holonomic geometric constraints like sawing. The "struggles" described by the user are symptomatic of a "fighting" condition where the controller's error minimization logic conflicts with the physics engine's non-penetration constraints, leading to high-frequency oscillation and contact instability.[5]

Third, **Collision and Physics Fidelity Artifacts** are exacerbating the control issues. The instability at the point of contact suggests sub-optimal collision mesh approximations—specifically the use of convex hulls which may artificially "round off" the flat blade edge—and incorrect solver settings such as excessive contact_offset values. These settings create "phantom" repulsive forces that prevent a stable perpendicular rest state,

causing the saw to roll or slide unpredictably.[7]
This document presents a rigorous theoretical breakdown of these issues, supported by quantitative analysis of the provided logs and reference to the underlying operational space control formulations. It concludes with a detailed, step-by-step implementation plan to transition the system to an **Operational Space Control (OSC)** framework with hybrid force/motion capabilities, correct frame transformations, and optimized physics materials, ensuring the fidelity required for collaborative human-robot interaction tasks.

---

2. Theoretical Framework of Contact-Rich Manipulation

To address the specific failure modes observed in the sawing experiment, it is necessary to first establish the theoretical underpinnings of robotic manipulation in contact-rich environments. The transition from free-space motion to physical interaction introduces discontinuities in the system dynamics that standard position controllers cannot adequately manage.

2.1. The Limitations of Differential Inverse Kinematics in Contact

The user's logs and description suggest the use of a DifferentialIKController or a similarly rigid control scheme. Understanding why this fails requires examining the mathematical formulation of Differential Inverse Kinematics (DiffIK).
DiffIK functions by mapping a desired task-space velocity $\dot{x}_{des}$ to a joint-space velocity $\dot{q}_{cmd}$ using the Moore-Penrose pseudo-inverse of the Jacobian $J^\dagger$:

$$\dot{q}_{cmd} = J(q)^\dagger \cdot (K_p \cdot e_{pos} + \dot{x}_{ff})$$
where e_{pos} is the position error vector and K_p is a proportional gain matrix.[3] This formulation assumes that the robot is free to move in the requested direction. In a sawing task, however, the log imposes a physical constraint: the saw cannot move *through* the wood (except via the slow process of material removal, which is not modeled as a kinematic freedom in rigid body physics).
When the user commands the saw to a position slightly below the surface of the log to initiate a cut, the DiffIK controller calculates a velocity vector pointing into the log. The physics engine (PhysX), detecting a collision, generates a contact constraint that prevents this motion. Consequently, the position error e_{pos} remains non-zero. The DiffIK controller, observing that the error has not decreased, continues to command a velocity into the log. This results in the integration of error and the buildup of joint torques (if using an inner-loop torque controller) or immediate velocity violations.

In Isaac Sim, this manifests as "fighting": the physics solver applies a depenetration impulse to push the saw out, while the controller tries to push it in.[9] The result is the "position adjustment struggles" and jitter reported by the user. The controller essentially treats the environmental constraint as a disturbance to be rejected, rather than a surface to be engaged.

2.2. Operational Space Formulation for Interaction

The solution to this pathology is the **Operational Space Formulation**, pioneered by Oussama Khatib.[6] This approach projects the system's dynamics into the task space (operational space), allowing for the direct control of end-effector forces and motions.
The dynamic equation of motion for a manipulator in operational space is given by:

$$\Lambda(q)\ddot{x} + \mu(q, \dot{q}) + p(q) = F_{cmd} - F_{ext}$$

where:

- $\Lambda(q)$ is the operational space inertia matrix (the effective mass of the end-effector).
- $\mu(q, \dot{q})$ represents the Coriolis and centrifugal forces projected into task space.
- $p(q)$ is the gravity vector in task space.
- F_{cmd} is the commanded operational force.
- F_{ext} is the external force from the environment (e.g., the log).

By controlling F_{cmd} directly, we can design a control law that dictates how the robot should respond to F_{ext}. This is the foundation of **Impedance Control**, which establishes a dynamic relationship between motion error and contact force:

$$F_{cmd} = K_p(x_{des} - x) + K_d(\dot{x}_{des} - \dot{x})$$

In this framework, the robot acts as a mass-spring-damper system.[5]

- **Stiffness (K_p):** Determines the rigidity of the virtual spring. In the direction of the cut (downward), we can lower the stiffness to allow the saw to comply with the log's surface geometry.
- **Damping (K_d):** Dissipates energy. This is critical for stabilizing the contact. The "struggles" observed in the user's experiment are essentially under-damped oscillations where the energy injected by the controller is not being dissipated by the contact interface.

2.3. Hybrid Force/Motion Control

For sawing, a pure position control or pure impedance control is often insufficient. We require **Hybrid Force/Motion Control**, where the task space is decomposed into orthogonal subspaces.[5]

- **Motion Subspace:** The directions in which the robot should track a trajectory (e.g., the reciprocating sawing motion along the X-axis, and the lateral position along the Y-axis).
- **Force Subspace:** The directions in which the robot should exert a specific force (e.g., the downward pressure along the Z-axis to engage the teeth).

Isaac Lab's OperationalSpaceController supports this decomposition explicitly via the target_types configuration, allowing users to specify ["pose_abs", "wrench_abs"] simultaneously.[6] This capability allows the system to maintain a stiff position hold on the saw's orientation (keeping it perpendicular) while applying a constant, regulated force into the log, completely resolving the instability caused by positional conflicts.

2.4. Null-Space Projection for Posture Control

A critical, often overlooked aspect of sawing is the robot's arm configuration (elbow position). Even if the end-effector is perfectly positioned, the arm must not collide with the environment or the human collaborator.
The Operational Space framework allows for **Null-Space Control**, where secondary objectives (like joint posture) are projected into the null space of the primary task Jacobian.[4]

$$\tau_{null} = (I - J^T(q) \bar{J}^T(q)) \tau_{posture}$$

This ensures that adjustments to the elbow position do not disturb the saw blade's position on the log. The user's report of "struggling" adjustments implies that the robot might be reaching kinematic singularities or joint limits, causing internal motion that perturbs the end-effector. Utilizing nullspace_control="position" in the controller configuration stabilizes the overall limb configuration.[12]

---

3. System Architecture and Coordinate Frame Analysis

The user's primary symptom—"never perfectly perpendicular to the log"—is a strong indicator of a geometric misalignment rather than a purely dynamic failure. In robotic simulation, specifically within the Universal Scene Description (USD) ecosystem used by Omniverse, the definition of coordinate frames is the single most frequent source of task-space error.

3.1. The "Fictitious" End-Effector Problem

Robots imported into Isaac Lab, such as the Franka Emika or UR10e, are typically defined with a kinematic chain terminating at a standard mechanical interface, such as the panda_hand or tool0 frame.[1] This frame represents the mounting flange, not the functional tip of the tool. When a saw is attached to the robot, the **Tool Center Point (TCP)** moves significantly. For a standard hand-saw, the center of the cutting edge may be offset by 15-25cm in the

Z-direction and potentially offset in Y or X depending on the handle geometry.
The user states they have "defined positions in the EE frame." However, if the controller is configured to track the panda_hand frame, but the user's mental model (and target coordinates) assumes the control point is the saw blade, a massive conflict arises.

- **Scenario:** The user commands the "EE" to the log surface (Z_{log}).
- **Reality:** The controller drives the panda_hand to Z_{log}.
- **Physical Consequence:** Since the saw extends below the hand, driving the hand to the log surface forces the saw to penetrate deep into the log.
- **Rotational Artifact:** To minimize the error between the hand frame and the target (which is geometrically impossible to reach due to the log), the robot may rotate the wrist. By tilting the saw, the vertical distance between the hand and the log might be minimized, or the collision geometry might slide into a lower-energy state. This explains the persistent "tilt."

Isaac Lab addresses this via the OffsetCfg class or body_offset parameters in the controller configuration.[2] The documentation explicitly notes that "end-effector frames are fictitious frames... defined at an offset to the parent rigid body."

3.2. Frame Transformation Logic

To fix this, we must rigorously define the transformation chain. The pose of the saw blade T_{blade} relative to the world is:
$$ T_{world}^{blade} = T_{world}^{base} \cdot T_{base}^{flange}(q) \cdot T_{flange}^{mount} \cdot T_{mount}^{blade} $$
The term T_{flange}^{blade} (combining mount and blade offsets) must be static and explicitly provided to the controller. If this term is missing, the controller minimizes error at T_{flange}, while the physics engine enforces collisions at T_{blade}.
Debug Strategy: The FrameTransformer sensor in Isaac Lab is the standard instrument for diagnosing these offsets.[14] It allows for the reporting and visualization of frame transforms. By attaching a FrameTransformer to the robot's hand and defining a target_frame with the estimated saw offset, the user can visualize coordinate axes in the viewer.

- If the visualized RGB axes do not sit exactly on the cutting edge of the saw, the offset is wrong.
- If the Z-axis (Blue) of the visualized frame does not point in the direction the user intends to push, the orientation commands will be permuted, leading to the "tilt".[1]

3.3. Quaternion Convention Discrepancies

A subtle but critical source of rotational error in Isaac Lab is the quaternion convention.

- **Isaac Lab / Isaac Sim / USD:** Utilize the **wxyz** (scalar-first) convention for internal tensor operations and API calls.[16]

- **External Libraries / ROS / Older PhysX:** Often utilize the **xyzw** (scalar-last) convention.[18]

The snippet [16] explicitly warns: "Remember to switch all quaternions to use the xyzw convention when working indexing rotation data. Similarly, please ensure all quaternions are in wxyz before passing them to Isaac Lab APIs."
Impact of Mismatch:
If the user calculates a target quaternion $q_{target} = $ (identity in xyzw) but passes it to Isaac Lab which interprets it as $$ in wxyz (where $w=0, z=1$), the resulting rotation is 180° around the Z-axis (or varying interpretations depending on normalization).
Even more dangerous is a quaternion like $q = [0.707, 0, 0.707, 0]$.

- In wxyz: $w=0.707, x=0, y=0.707, z=0$. Rotation $\approx 90^\circ$ around Y.
- In xyzw: $x=0.707, y=0, z=0.707, w=0$. Rotation $\approx 180^\circ$ around the axis $(1, 0, 1)$.

This scrambling of components fundamentally alters the orientation. The user's log shows quaternions like [0.0619 0.9149 0.3704 0.1476]. If this was computed in scipy/numpy (scalar-last) and fed to Isaac Lab (scalar-first), the saw would orient in a completely uncommanded direction, potentially explaining why "it's never perfectly perpendicular."

3.4. Euler Angle Singularities

While Euler angles are intuitive for humans (e.g., "pitch the saw 90 degrees"), they are prone to Gimbal Lock and interpolation issues.19 The user's logs imply they might be trying to adjust the saw using Euler-based logic ("RotX(-90)").
When converting Euler angles to quaternions for the controller, the order of operations (XYZ vs ZYX) matters immensely.20 Isaac Sim documentation notes that while the sim backend uses quaternions, the UI often presents Euler angles in XYZ, but internal conversions might default to ZYX. Using the isaaclab.utils.math.quat_from_euler_xyz function ensures consistency with the simulation's coordinate system.17

---

4. Physics Engine Dynamics and Simulation Fidelity

Even with perfect control theory and correct frame alignment, artifacts within the physics simulation (NVIDIA PhysX) can destabilize a contact task. The "tilt" and "jitter" reported are classic signatures of collision resolution artifacts.

4.1. Collision Mesh Approximation

In real-time simulation, calculating collisions between arbitrary triangle meshes is computationally prohibitive. Isaac Sim defaults to using **Collision Approximations** for rigid

bodies.[7] The standard default is the **Convex Hull**.
The Pathology of Convex Hulls for Blades:
A saw blade is a thin, flat, rectangular object with sharp teeth. A convex hull algorithm attempts to wrap this shape in the tightest possible convex volume.

1. **Rounding:** To optimize vertex count, the hull generator often "rounds off" sharp corners or bevels edges.
2. **Inflation:** The hull may be slightly inflated to ensure numerical stability.

If the saw blade's collision geometry is approximated as a slightly rounded shape (like a flattened capsule or a bar of soap), it physically *cannot* rest perpendicularly on a flat log. Just as a pencil balanced on its tip will fall, a saw with a rounded collision edge will roll to one side to minimize potential energy. This provides a physics-based explanation for the "always a tilt" observation that persists despite control efforts.[23]
The Solution: Convex Decomposition or Primitives:

- **Convex Decomposition:** Splits the complex mesh into multiple smaller convex hulls, preserving the sharp features and flat bottom edge of the blade.[7]
- **Geometric Primitives:** The most robust solution for a saw blade is to attach a primitive **Box** shape to the visual mesh, scale it to match the blade dimensions, and set this invisible box as the collider. A primitive box is mathematically flat. When pressed against a log, the contact patch is a plane (or line), providing inherent rotational stability.[24]

4.2. Contact Offset and Rest Offset

PhysX manages contact generation using two critical parameters: contact_offset and rest_offset.[8]

- **contact_offset:** This is a buffer zone surrounding the collision shape. The solver begins generating contact constraints when objects come within this distance. It creates a "force field" that prevents objects from moving too fast into one another.
  - *Issue:* If contact_offset is large (e.g., 2cm, which is default in some presets), the saw will "feel" the log before it visually touches it. The controller will try to push further, but the solver will apply repulsive forces. This creates a "floating" instability where the saw slides around on this invisible cushion.
- **rest_offset:** This defines the distance at which two objects are considered "touching" at equilibrium.
  - *Issue:* If rest_offset is positive, the objects will sit with a visible air gap. If negative, they will penetrate.

Tuning for Precision:
For a precision tool like a saw, contact_offset must be minimized (e.g., 0.001m or 1mm) to ensure the visual contact matches the physical contact. The rest_offset should typically be 0.0. Misconfiguration here leads to the "adjustment struggles" as the solver fights the controller over these invisible gaps.8

4.3. The "Explosion" Phenomenon: Max Depenetration Velocity

The user logs imply oscillation. A specific PhysX parameter, max_depenetration_velocity, is often the culprit for violent jitters.9
When a stiff controller (like DiffIK) forces the saw into the log (penetration) during one timestep, the physics solver detects this violation in the next step. It calculates the velocity required to eject the object from the log immediately.

- If the penetration is deep (due to high stiffness), the required ejection velocity is massive.
- The saw is "shot" out of the log.
- The controller sees the saw is now too far away and slams it back in.
- **Result:** High-frequency vibration or "explosion" of the simulation.

Mitigation:
Setting rigid_props.max_depenetration_velocity to a low value (e.g., 0.1 m/s) clips this ejection velocity.25 Instead of exploding out, the saw is gently pushed out over multiple frames. This dampens the "struggle" and allows the system to settle.

4.4. Friction and Physics Materials

The sawing motion relies on **Stick-Slip** friction dynamics.

- **Static Friction:** Holds the saw in place when starting a cut.
- **Dynamic Friction:** Resists motion during the cut.

Standard PhysX materials are isotropic (same friction in all directions). If friction is too high, the saw will "stick" during position adjustments. The controller integrates error until it overcomes the static friction threshold, causing the saw to jump suddenly (slip).[27] This aligns with the "position adjustment struggles."
Additionally, the **Restitution** (bounciness) of the material is critical.

- Default materials often have non-zero restitution.
- When the saw taps the log, it bounces. A bouncing saw cannot be stabilized by a position controller easily.
- **Requirement:** restitution must be set to exactly **0.0** for both the saw and the log to ensure plastic impact (energy absorption) rather than elastic impact.[27]
- **Friction Combine Mode:** The simulation combines the friction of the saw and log. The default is often "average." For a saw teeth biting into wood, the interaction is complex, but generally, we want to ensure the effective friction is high enough to bite. Using friction_combine_mode="max" or min depending on the desired slip behavior is a necessary tuning step.[28]

---

5. Detailed Diagnostic Analysis of User Logs

A granular analysis of the provided debug logs (Frames 273-275) reveals the specific nature of the failure.

5.1. Table 1: Log Data Analysis

Frame	Target EE Position (m)	Actual EE Position (m)	Position Error	Orientation Error	Physics State
273	[0.1422, -0.0001, 0.9641]	[0.1422, -0.0001, 0.9641]	0.0000 m	0.0000 rad	**Saw Pos:** [0.284, -0.001, 0.524] **Log Pos:** [1.0, 0.0, 0.4]
274	[0.1422, -0.0002, 0.9641]	[0.1422, -0.0002, 0.9641]	0.0000 m	0.0000 rad	**Saw Pos:** [0.284, -0.001, 0.524]
275	[0.1422, -0.0001, 0.9641]	[0.1422, -0.0001, 0.9641]	0.0000 m	0.0000 rad	**Saw Pos:** [0.284, -0.001, 0.524]

5.2. Insight: The "Zero Error" Fallacy

The most striking feature of the logs is that **EE Error is 0.0000 m and 0.0000 rad** in all frames.

- The controller believes it has perfectly achieved its target.
- Yet, the user provides images showing the saw is tilted and unstable.
- **Conclusion:** The controller is tracking the **wrong frame**. It is successfully moving the robot's wrist (or whatever frame is defined as EE) to the target coordinates. The "tilt" and instability are happening downstream of the control loop, in the physical interaction between the attached saw (which acts as an unmodeled or incorrectly modeled offset) and the log.

If the controller were struggling to reach the target due to collision, we would see a non-zero position error. The fact that error is zero proves that the *kinematic chain* is satisfied, but the *task* is not. This definitively confirms the **Frame Mismatch** hypothesis: the robot is placing its hand exactly where told, but the attached saw is colliding with the log in a way that forces the hand to rotate to maintain that position, or the visual tilt is simply the result of the hand being at the wrong angle to support the saw upright.

5.3. Quaternion Analysis

Target Quat: [0.0619, 0.9149, 0.3704, 0.1476]

- Assuming wxyz (Isaac Lab standard): $w \approx 0.06$, $x \approx 0.91$. This is dominated by the X-component.
- Rotation axis is roughly aligned with global X.
- Saw State Quat (World): [-0.304, -0.748, -0.219, 0.547].
- There is a significant difference between the Target EE quaternion and the Saw World quaternion. This confirms that the Saw Frame is **not** aligned with the EE Frame. There is a static transform between them that is likely not being accounted for in the target generation. The user is sending a target for the EE, expecting the Saw to align, but the Saw is physically offset and rotated relative to that EE.

---

6. Implementation Strategy: Operational Space Control

To remedy the instability, the control topology must be migrated from the stiff Differential IK (implied by the symptoms) to a compliant **Operational Space Control (OSC)** framework. This section details the specific configuration required using isaaclab.controllers.operational_space_cfg.

6.1. Rationale for OSC over DiffIK

As detailed in Section 2.1, DiffIK fails in contact because it cannot regulate interaction forces. OSC, specifically configured for **Variable Impedance**, allows us to define the saw's behavior as a set of springs and dampers.

6.2. Controller Configuration

The OperationalSpaceControllerCfg class [6] provides the parameters necessary to decouple the task axes. We need the saw to be:

1. **Stiff** in orientation (to prevent tilt).
2. **Stiff** in the cutting direction (Y/X plane) to track the cut.
3. **Compliant** in the depth direction (Z) to allow the saw to rest on the log without penetrating.
4. **Force-Controlled** in Z to apply cutting pressure.

Table 2: Recommended OSC Configuration

Parameter	Value	Description	Reference
target_types	["pose_abs", "wrench_abs"]	Enables simultaneous tracking of a target pose and a feed-forward force vector.	6
inertial_dynamics_decoupling	True	Decouples the end-effector dynamics from the arm's inertia, essential for tracking rapid sawing motions.	11
gravity_compensation	True	Ensures the arm's weight does not bias the contact force. The saw "floats" if no force is applied.	11
impedance_mode	"variable_kp"	Allows changing stiffness dynamically (e.g., stiff for approach, compliant for sawing).	11
motion_control_axes_task	``	1=Motion, 0=Force. Disables motion control on Z (depth).	11
contact_wrench_control_axes_task	``	Enables force control on Z.	11
nullspace_control	"position"	Controls the elbow posture in the null-space to prevent self-collision.	12

6.3. Code Implementation

The following Python code illustrates how to instantiate this configuration within the Isaac Lab environment.

Python

```
from isaaclab.controllers.operational_space_cfg import OperationalSpaceControllerCfg
from isaaclab.utils import configclass

@configclass
class SawingControllerCfg(OperationalSpaceControllerCfg):
    # Set the controller type to OSC
    target_types = ["pose_abs", "wrench_abs"]

    # Dynamics Settings
    inertial_dynamics_decoupling = True
    partial_inertial_dynamics_decoupling = False
    gravity_compensation = True

    # Impedance Mode
    impedance_mode = "variable_kp"

    # Axis Selection (Assuming Z is vertical down into log)
    #
    motion_control_axes_task =
    contact_wrench_control_axes_task =

    # Nominal Gains (Can be overridden at runtime)
    # High rotational stiffness to prevent tilt (150.0)
    # Zero stiffness in Z (handled by force)
    motion_stiffness_task = [200.0, 200.0, 0.0, 150.0, 150.0, 150.0]

    # Damping: Critical or Over-damped (Ratio > 1.0) to stop oscillation
    motion_damping_ratio_task = [2.0, 2.0, 0.0, 4.0, 4.0, 4.0]

    # Force Control Gains (Admittance for Z axis)
    contact_wrench_stiffness_task = [0.0, 0.0, 20.0, 0.0, 0.0, 0.0]

    # Nullspace Posture Control
    nullspace_control = "position"
    nullspace_stiffness = 40.0
    nullspace_damping_ratio = 1.0
```

Reasoning for Damping Values:
The motion_damping_ratio_task is set high (2.0-4.0). Snippet 29 on tuning suggests starting with zero damping, but for contact tasks, energy dissipation is priority #1. A ratio > 1.0 ensures the system does not overshoot the target orientation, which is crucial for keeping the saw perpendicular.

7. Implementation Strategy: Physics & Asset Optimization

The control scheme can only function if the physical world it interacts with is modeled with sufficient fidelity.

7.1. Refined Collision Geometry

As identified in Section 4.1, the default convex hull approximation is a likely cause of the "rolling" instability. The implementation must replace this.
Step-by-Step Fix:

1. **Open the Saw USD Asset:** Locate the Prim corresponding to the blade.
2. **Modify Collision Approximation:**
   - Change the physics:approximation attribute from convexHull to convexDecomposition.[7]
   - *Alternative (Better):* Create a child Cube prim. Scale it to the exact dimensions of the blade (e.g., $0.5m \times 0.05m \times 0.002m$). Set this cube as the collider and make it invisible (purpose="guide").
   - This ensures the contact surface is mathematically flat, providing a stable base for the "perpendicular" constraint.

7.2. Rigid Body Property Tuning

The RigidBodyPropertiesCfg must be adjusted to prevent the "explosive" contact resolution.[25]

Python

```python
from isaaclab.sim.schemas import RigidBodyPropertiesCfg

# Apply to Saw and Log
rigid_props = RigidBodyPropertiesCfg(
    solver_position_iteration_count=12,  # Increase from default (4) for stability
    solver_velocity_iteration_count=4,   # Increase from default (1)
    max_depenetration_velocity=0.1,      # CRITICAL: Limit ejection speed
    stabilization_threshold=0.001,       # Lower threshold to settle faster
    max_contact_impulse=100.0,           # Cap the force the log can apply to the saw
)
```

7.3. Physics Material Setup

We must define a specific material for the saw-log interaction to handle friction and restitution.[27]

Python

```python
from isaaclab.sim.spawners.materials import RigidBodyMaterialCfg

saw_blade_material = RigidBodyMaterialCfg(
    static_friction=0.8,         # High friction to bite into wood
    dynamic_friction=0.5,        # Moderate friction for sawing
    restitution=0.0,             # No bouncing
    friction_combine_mode="max", # Use the higher of the two frictions (Saw vs Log)
    restitution_combine_mode="min" # Ensure zero restitution dominates
)
```

---

8. Implementation Strategy: Visualization and Calibration

To close the loop on the "Frame Mismatch" issue, we must visualize the frames.

8.1. Frame Transformer Implementation

Use the FrameTransformer sensor to verify the saw's TCP.[14]

Python

```python
from isaaclab.sensors import FrameTransformerCfg
from isaaclab.markers import VisualizationMarkersCfg
import isaaclab.sim as sim_utils

# Define the sensor attached to the robot hand
```

```
frame_transformer_cfg = FrameTransformerCfg(
    prim_path="/World/Robot/panda_hand",
    target_frames=,
    debug_vis=True # Enable visualization [31]
)

# Marker for visualization
marker_cfg = VisualizationMarkersCfg(
    prim_path="/Visuals/EE_Debug",
    markers={
        "frame": sim_utils.UsdFileCfg(
            usd_path=f"{ISAAC_NUCLEUS_DIR}/Props/UIElements/frame_prim.usd",
            scale=(0.1, 0.1, 0.1),
        ),
    },
)
```

Calibration Procedure:

1. Run the simulation.
2. Observe the RGB axis markers.
3. **Red Axis (X):** Must point along the sawing stroke direction.
4. **Blue Axis (Z):** Must point perpendicular to the log surface.
5. Adjust the offset.pos and offset.rot in the config until this alignment is visually perfect on the saw geometry.

8.2. Quaternion Math Correction

Ensure all quaternion inputs are converted correctly using Isaac Lab's utilities.[17]

Python

```
import torch
from isaaclab.utils.math import quat_from_euler_xyz

# Desired orientation: Blade vertical, teeth down
# This depends on your asset's local axes.
# Example: Rotate 180 deg around X to flip Z-axis down
target_quat_wxyz = quat_from_euler_xyz(
    roll=torch.tensor([3.14159]),
    pitch=torch.tensor([0.0]),
    yaw=torch.tensor([0.0])
)
```

---

9. Conclusion

The instability observed in the collaborative sawing experiment is not a singular error but a compound failure of **geometry** (frame mismatch), **control topology** (stiff IK vs. compliant OSC), and **simulation physics** (collision approximation and solver settings).
The evidence—specifically the zero-error logs paired with visible misalignment—confirms that the controller is successfully driving the wrong frame (the wrist) to the target. Meanwhile, the "struggles" and jitter are confirmed signatures of a stiff controller fighting the physics engine's non-penetration constraints, exacerbated by a rounded collision mesh that creates rotational instability.
The Remediation Plan:

1. **Adopt Operational Space Control:** Switch to OperationalSpaceControllerCfg with hybrid force/motion targets to allow the saw to "float" vertically while maintaining rigid orientation.
2. **Rectify Collision Geometry:** Replace the saw's convex hull with a primitive box collider to ensure a flat, stable contact patch.
3. **Tune Physics:** Set max_depenetration_velocity to 0.1 and restitution to 0.0 to eliminate explosive contact resolution.
4. **Calibrate Frames:** Use FrameTransformer to visualize and correct the offset between the robot hand and the saw blade center.

By implementing these structural changes, the system will move from a state of conflict (fighting the log) to a state of compliant interaction, enabling the stable, perpendicular sawing behavior required for the experiment.

Works cited

1. Isaac Cortex: Overview - Isaac Sim Documentation, accessed November 23, 2025, https://docs.isaacsim.omniverse.nvidia.com/5.0.0/cortex_tutorials/tutorial_cortex_1_overview.html
2. isaaclab.envs.mdp — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/api/lab/isaaclab.envs.mdp.html
3. Using a task-space controller — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html
4. isaaclab.controllers — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/api/lab/isaaclab.controllers.html

5. Motion Generators — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/overview/core-concepts/motion_generators.html
6. Using an operational space controller — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_osc.html
7. Collider is not accurate - Isaac Sim - NVIDIA Developer Forums, accessed November 23, 2025, https://forums.developer.nvidia.com/t/collider-is-not-accurate/325886
8. Advanced Collision Detection — physx 5.1.2 documentation, accessed November 23, 2025, https://nvidia-omniverse.github.io/PhysX/physx/5.1.2/docs/AdvancedCollisionDetection.html
9. Simulation Tuning — Isaac Gym documentation, accessed November 23, 2025, https://docs.robotsfan.com/isaacgym/programming/tuning.html
10. isaaclab.controllers.operational_space — Isaac Lab Documentation, accessed November 23, 2025, https://docs.robotsfan.com/isaaclab_official/main/_modules/isaaclab/controllers/operational_space.html
11. isaaclab.controllers.operational_space_cfg — Isaac Lab Documentation, accessed November 23, 2025, https://docs.robotsfan.com/isaaclab_official/main/_modules/isaaclab/controllers/operational_space_cfg.html
12. isaaclab.controllers — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/v2.0.0/source/api/lab/isaaclab.controllers.html
13. isaaclab.envs.mdp.actions.actions_cfg — Isaac Lab Documentation, accessed November 23, 2025, https://docs.robotsfan.com/isaaclab_official/v2.1.0/_modules/isaaclab/envs/mdp/actions/actions_cfg.html
14. isaaclab.sensors.frame_transformer.frame_transformer 源代码 - robotsfan文档首页, accessed November 23, 2025, https://docs.robotsfan.com/isaaclab/_modules/isaaclab/sensors/frame_transformer/frame_transformer.html
15. Creating Visualization Markers — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/how-to/draw_markers.html
16. From IsaacGymEnvs — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/migration/migrating_from_isaacgymenvs.html
17. isaaclab.utils — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/api/lab/isaaclab.utils.html
18. clemense/quaternion-conventions: An overview of different quaternion implementations and their chosen order: x-y-z-w or w-x-y-z? - GitHub, accessed November 23, 2025, https://github.com/clemense/quaternion-conventions

19. Tutorial 17 : Rotations, accessed November 23, 2025, http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/
20. Change the coordinate in Isaac SIm - NVIDIA Developer Forums, accessed November 23, 2025, https://forums.developer.nvidia.com/t/change-the-coordinate-in-isaac-sim/287834
21. Followtarget - How to rotate my robot? - Isaac Sim - NVIDIA Developer Forums, accessed November 23, 2025, https://forums.developer.nvidia.com/t/followtarget-how-to-rotate-my-robot/265855
22. Simulation Performance and Tuning — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/source/how-to/simulation_performance.html
23. Collider and Rigid_Body preset error (PhysicsUSD: Parse collision - triangle mesh collision), accessed November 23, 2025, https://forums.developer.nvidia.com/t/collider-and-rigid-body-preset-error-physicsusd-parse-collision-triangle-mesh-collision/244203
24. How to I get a collision shape for my 3D model : r/godot - Reddit, accessed November 23, 2025, https://www.reddit.com/r/godot/comments/1h40zej/how_to_i_get_a_collision_shape_for_my_3d_model/
25. Rigid Body Collision Penetration Issue in Isaac Sim - NVIDIA Developer Forums, accessed November 23, 2025, https://forums.developer.nvidia.com/t/rigid-body-collision-penetration-issue-in-isaac-sim/313037
26. Physx rigid body dynamics resting contact - Isaac Sim - NVIDIA Developer Forums, accessed November 23, 2025, https://forums.developer.nvidia.com/t/physx-rigid-body-dynamics-resting-contact/283182
27. Best Practices for Setting Friction & Restitution Coefficients for Sim2Real - Isaac Sim, accessed November 23, 2025, https://forums.developer.nvidia.com/t/best-practices-for-setting-friction-restitution-coefficients-for-sim2real/347006
28. Physics Material Properties - Friction and Restitution - NVIDIA Developer Forums, accessed November 23, 2025, https://forums.developer.nvidia.com/t/physics-material-properties-friction-and-restitution/212036
29. Tuning Joint Drive Gains - Isaac Sim Documentation, accessed November 23, 2025, https://docs.isaacsim.omniverse.nvidia.com/4.5.0/robot_setup/joint_tuning.html
30. isaaclab.sim.spawners.materials.physics_materials_cfg — Isaac Lab Documentation, accessed November 23, 2025, https://isaac-sim.github.io/IsaacLab/main/_modules/isaaclab/sim/spawners/materials/physics_materials_cfg.html

31. isaaclab.utils.math — Isaac Lab Documentation, accessed November 23, 2025, https://docs.robotsfan.com/isaaclab_official/main/_modules/isaaclab/utils/math.html