Lab 9: Sim2Real

Preparing for Hardware Deployment

EE 3002 - JDS Robotics

Lahore University of Management Sciences (LUMS)

Spring Semester 2026

1 Introduction: The Sim2Real Gap

In the simulation phase of your project, you developed a mathematically perfect drawing pipeline. However, as you prepare to deploy this exact software onto the physical **WidowX 250s** hardware, you must account for real-world physics.

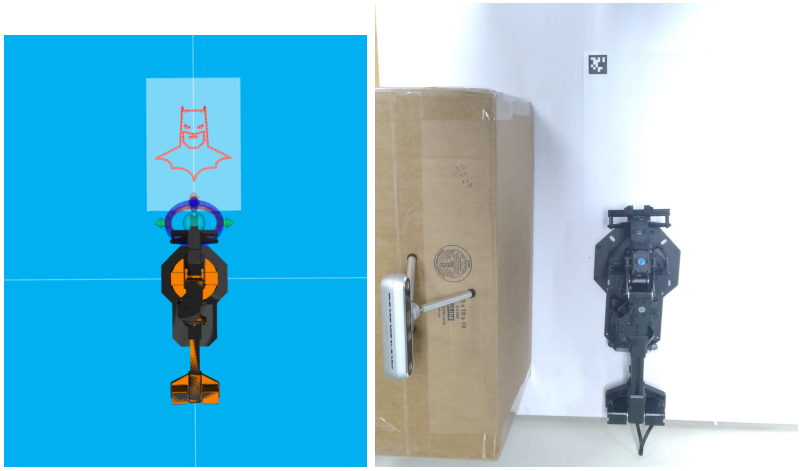

Figure 1: Visualizing the Sim2Real Gap: Gazebo Simulation (left) vs. Physical Hardware Setup (right).

Moving from simulation to reality introduces friction, lens distortion, and uneven surfaces, a phenomenon known in robotics as the **Sim2Real Gap**. Moving from simulation to reality introduces friction, lens distortion, and uneven surfaces, a phenomenon known in robotics as the **Sim2Real Gap**.

Hardware Demonstration Video

To truly understand the Sim2Real gap and see the final pipeline in action, watch the dynamic tracking demonstration. In this video, you will see the WidowX 250s autonomously adjust its hover position as the A4 paper is moved across the workspace.

→ Click Here to Watch the Video ←

2 Dynamic Vision Integration

You will no longer type the paper's location into the terminal as a static transform.

3 Dynamic Vision Integration

You will no longer type the paper's location into the terminal as a static transform. An overhead RealSense camera will continuously scan the physical workspace.

- **Camera Calibration:** Real-world lenses are curved, causing "fisheye" radial distortion. If uncorrected, the robot will draw curved lines instead of straight ones. You will compute an Intrinsic Matrix (K) using a checkerboard pattern to mathematically flatten the incoming video feed.
- **Fiducial Tracking:** You will utilize **AprilTags** (specifically tag 0) printed on the corner of the A4 paper. Unlike standard QR codes, AprilTags are designed for high-precision robotics; they provide a full **6-DOF Pose** (Position and Orientation).

  By tracking tag 0, the camera calculates the distance (X, Y, Z) and the tilt (Roll, Pitch, Yaw) of the paper relative to the lens. This data is broadcast to the `tf2` network at 30Hz, allowing the `paper_frame` to move dynamically in the robot's coordinate system, even if the paper is moved across the desk.

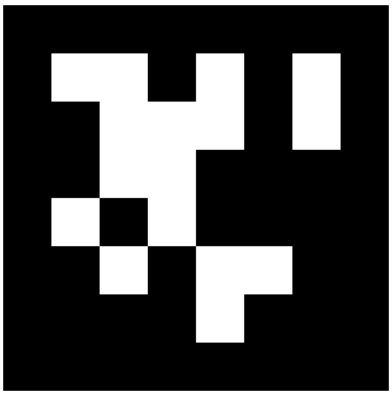

Figure 2: The AprilTag tag 0 defines a local coordinate system where the center of the tag is $(0, 0, 0)$. The robot uses this "anchor" to calculate the paper's center.

4 Environment Setup: Installing Dependencies

Before hardware communication, install the necessary drivers. Ensure your VM has **USB 3.1** compatibility enabled.

4.1 1. Intel RealSense SDK & ROS 2 Wrapper

```
sudo apt install ros-humble-librealsense2*
sudo apt install ros-humble-realsense2-camera
```

4.2 2. AprilTag Detection Suite

```
sudo apt update
sudo apt install ros-humble-apriltag-msgs
sudo apt install ros-humble-apriltag
```

5 Mechanical Compliance

- **Friction Management:** Wrap the pen in foam to allow the gripper to securely hold the cylindrical shape without twisting.
- **The Ink Mechanism:** Use a felt-tip Sharpie. These require zero downward force compared to ballpoint pens.

6 Safety Protocol: The "Air Draw"

Mandatory Safety Rule: The Air Draw

Before physical ink touches the paper, run your script with an intentional $+5cm$ $(0.05m)$ **Z-axis offset**. Only after verification will you be cleared to remove the offset.

7 Phase 1: Mechanical Preparation

1. **Lock it Down:** Tape the target A4 sheet firmly over the workplace.
2. **The Gripper Hack:** Wrap the center of a Sharpie in foam and command the WidowX gripper to clamp it securely.

8 Phase 2: The Vision System

3. **Mount the Camera:** Place the RealSense camera overhead, looking down at the workspace.
4. **Calibrate the Lens:** Run the calibration node to generate `camera_info.yaml` (the Intrinsic Matrix, K).
5. **Launch Tag Tracking:** Start the AprilTag detection node to output the 3D location of the printed tag.

9 Phase 3: Building the TF Tree

6. **Link 1 (Base → Camera):** Measure the distance from the robot's base to the camera lens. Run a static TF publisher with these X, Y, Z and RPY values.
7. **Link 2 (Camera → Tag):** This dynamic link is broadcasted automatically by the `apriltag_ros` node.
8. **Link 3 (Tag → Paper Center):** Run a second static TF to shift the frame from the tag corner to the paper center ($+10.5cm$ X, $-14.85cm$ Y).

10 Phase 4: Execution & Safety

9. **The Offset:** Intentionally add $+5cm$ to the Z-axis height in your TF publisher.
10. **The "Air Draw" Test:** Run `main.py`. The robot should trace the design in the air.
11. **The Final Run:** If safe, remove the offset and run `main.py` at table level.

11 Phase 5: High-Precision Camera Calibration (720p)

11.1 Step 1: The High-Resolution Stream

```
ros2 launch realsense2_camera rs_launch.py \
    enable_color:=true \
    color_width:=1280 \
    color_height:=720 \
    color_fps:=6.0 \
    enable_depth:=false
```

11.2 Step 2: Checkerboard Calibration

```
ros2 run camera_calibration cameracalibrator \
    --size 8x6 \
    --square 0.025 \
    image:=/camera/color/image_raw \
    camera:=/camera/color
```

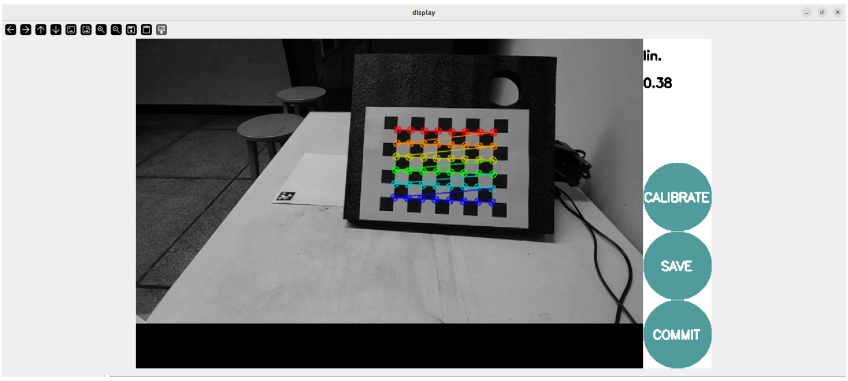

Figure 3: Checkerboard calibration GUI: Ensure all bars (X, Y, Skew, Size) turn green.

Calibration Checklist

- **Center Focus:** Position the board in the center to lock the Principal Point (c_x, c_y).
- **Edge Case:** Move the board to the extreme corners of the 1280×720 frame.
- **Wait for Math:** Wait until the terminal shows the K and D matrices before clicking **COMMIT**.

11.3 Step 3: "Touch-Point" Extrinsic Calibration

1. Place the AprilTag on the table.
2. **Manual Touch:** Manually move the robot until the gripper tip touches the center of the tag.
3. **Truth Check:**

```
ros2 run tf2_ros tf2_echo wx250s/base_link tag_0
```

4. **Adjustment:** If the Echo says $X = 0.315$ but the tag is at 0.300: Subtract 0.015 from the `--x` in Terminal 3.

12 Phase 6: The Final Deployment

This phase represents the "Sim2Real" integration. We will launch six concurrent terminals to bridge the gap between raw pixels and physical motor torque. Ensure your camera is plugged into a **USB 3.1** port before starting.

12.1 Terminal 1: The Camera Driver

We launch the RealSense node with a specific resolution and frame rate (15 FPS) to maintain a stable USB stream without overwhelming the Virtual Machine's CPU.

```
ros2 launch realsense2_camera rs_launch.py \
    enable_color:=true \
    enable_depth:=false \
    color_width:=1280 \
    color_height:=720 \
    color_fps:=15.0 \
    device_type:=d455
```

12.2 Terminal 2: AprilTag Detection

This node performs the heavy lifting of Computer Vision. It scans the incoming $720p$ frames for the specific black-and-white patterns of the AprilTag and calculates its 3D pose relative to the camera lens.

```
ros2 launch apriltag_ros continuous_detection.launch.xml \
    camera_name:=/camera/color \
    image_topic:=image_raw
```

12.3 Terminal 3: The Static Bridge (Extrinsics)

This command creates the mathematical link between the Robot's physical base and the Camera's lens. **Note:** Use the calibrated values $(x, y, z, pitch)$ derived from your Step 3 Touch-Point calibration.

```
ros2 run tf2_ros static_transform_publisher \
    --x [add your value] --y [add your value] --z [add your value] \
    --yaw [add your value] --pitch [add your value] --roll [add your value] \
    --frame-id wx250s/base_link --child-frame-id camera_link
```

12.4 Terminal 4: The "Truth Check" (Echo)

Before enabling the motors, we must verify the TF Tree. This terminal acts as your digital ruler. Ensure the output X and Y coordinates match your physical table measurements within ± 1 cm.

```
ros2 run tf2_ros tf2_echo wx250s/base_link tag_0
```

12.5 Terminal 5: MoveIt & Hardware Interface

This node wakes up the Dynamixel motors. It handles the Inverse Kinematics (IK) and ensures that the arm follows a collision-free path. **Warning:** The robot will "twitch" slightly upon launch as the torque is applied.

```
ros2 launch interbotix_xsarm_moveit xsarm_moveit.launch.py \
    robot_model:=wx250s \
    hardware_type:=actual
```

12.6 Terminal 6: Execution Script

Finally, run your custom logic. The script will pull the target coordinates from Terminal 4 and command Terminal 5 to move the arm.

```
python3 main.py
```

13 Appendix A: Dynamic TF-Tree Visualization

A healthy Sim2Real tree must link `base_link` → `camera_link` → `optical_frame` → `tag_0`.

14 Appendix B: Vision-to-Motion CodeWalk (`main.py`)

1. Vision Bridge: Finding `tag_0`

The script calculates the vector between the robot's base and the AprilTag:

```
trans = self.tf_buffer.lookup_transform('wx250s/base_link', 'tag_0', rclpy.time
    .Time())
```

2. The Paper Transform

The script transforms the point into the robot's base frame:

$$P_{\text{target}} = P_{\text{base\_link}} - (R_{\text{camera}} \times P_{\text{tag\_0}}) \tag{1}$$

3. The World Z-Clamp

To ensure the robot never crashes, the script forces its own safe Z-heights:

- **Approach** ($18cm$)**:** The "Safety Ceiling."
- **Hover** ($8cm$)**:** The final "Paper Level" height.