Lab 9: Vision & Kinematics

Building Perception-Action Pipelines in ROS 2

EE 3002 - JDS Robotics

Lahore University of Management Sciences (LUMS)

Spring Semester 2026

Contents

1 Introduction & Objectives

In this lab, you will transition from basic motor control to high-level autonomy. We will use the **Interbotix WidowX 250s** (6-DOF) to bridge the gap between computer vision (OpenCV) and motion planning (MoveIt 2).

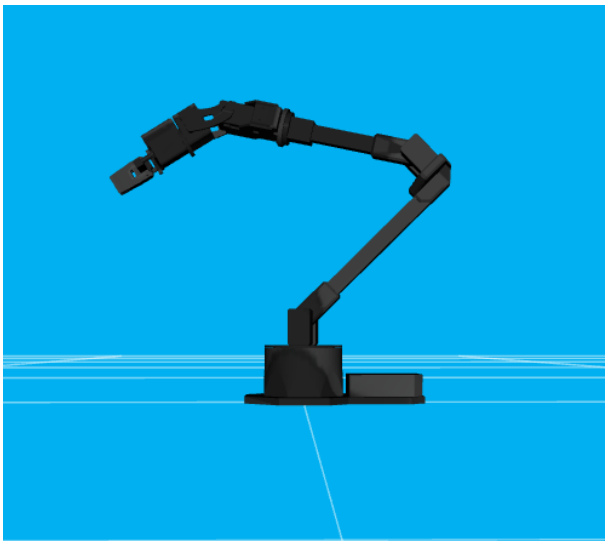

Figure 1: Interbotix WidowX 250s 6-DOF Manipulator Workspace.

1.1 Learning Objectives

By the end of this lab, students will be able to:

- Implement **Spatial Scaling** to map pixel coordinates to metric workspaces.
- Utilize **TF2 (Transform Library)** to handle dynamic frame relationships.
- Execute **Cartesian Path Planning** for linear end-effector movement.
- Integrate disparate ROS 2 nodes into a functional "Sense-Plan-Act" pipeline.

2 Theoretical Background

2.1 Coordinate Frames and TF2

Robots do not operate in a vacuum. Every component exists in a specific *Coordinate Frame*. To move the pen to a point on the paper, we must compute the transform T_{base}^{paper}.

2.2 Cartesian vs. Joint Space

Standard motion planning often results in curved paths. For drawing, we require **Cartesian Paths**, where the end-effector follows a straight line. This requires solving Inverse Kinematics (IK) along the trajectory.

3 Task 0: Environment Setup

3.1 Dependencies

Ensure your environment is running ROS 2 Galactic on Ubuntu 20.04. Run the following commands to install the necessary hardware abstraction packages:

```
# Install Interbotix XS Arms Suite
sudo apt install curl
curl 'https://raw.githubusercontent.com/Interbotix/interbotix_ros_manipulators/main/interbotix_ros_xsarms/install/amd64/xsarm_amd64_install.sh' > xsarm_amd64_install.sh
chmod +x xsarm_amd64_install.sh
./xsarm_amd64_install.sh -d galactic
```

3.2 Directory Structure

Create a package named `ee_vision_lab9` inside your workspace. Your directory must be organized as follows to ensure the build system recognizes your modules:

Required Directory Tree

```
interbotix_ws/src/ee_vision_lab9/
vision_processor.py   (Module 1:  Image Scaling math)
robot_drawer.py     (Module 2:  TF2 and MoveIt commands)
main.py           (Integration script)
paper_visualiser.py   (Visualization utility)
```

4 Documentation & MoveIt 2 Concepts

Before writing any code, you must familiarize yourself with the robot's kinematics and the MoveIt 2 motion planning framework.

4.1 Official Documentation

Read the official Interbotix MoveIt 2 documentation here:
MoveIt 2 Motion Planning Configuration — Interbotix X-Series Documentation

4.2 Key Concepts to Research

Using the link above and the MoveIt 2 tutorials, ensure you can answer the following:

- **Planning Groups:** Why is our group named `interbotix_arm`?
- **Planning Scene:** How does MoveIt represent the "floor" or "table" to avoid collisions?
- **Cartesian Planning:** Why do we use `compute_cartesian_path` instead of standard joint-space goals for drawing?

5 Task 1: Perception & Trajectory Generation (20 Marks)

Goal: Extract visual contours from an image and map them to a metric 2D plane. Do not attempt to move the robot in this task.

5.1 Step A: OpenCV Extraction

In `vision_processor.py`, implement thresholding and `cv2.findContours`. You must **downsample** your contours (e.g., take every 5th point via `contour[::5]`) to prevent overloading the trajectory planner later.

5.2 Step B: Spatial Scaling

Complete **TODO 1** and **TODO 2**. Map your pixel coordinates (u, v) to the A4 metric dimensions ($0.297m \times 0.210m$). Ensure the origin $(0, 0)$ is at the center of the paper:

$$x_p = -(y_{norm} - 0.5) \cdot 0.297, \quad y_p = -(x_{norm} - 0.5) \cdot 0.210 \tag{1}$$

5.3 Step C: Path Verification via RViz

Before dealing with robot kinematics, verify your perception logic visually. Broadcast your scaled points as a `visualization_msgs/Marker` on the `/drawing_ink` topic. Open RViz, set your Fixed Frame to `paper_frame`, and add the Marker display to see the exact contour you extracted drawn in the landscape. Do not move the arm yet.

5.4 Deliverable 1

1. Run your standalone script and **print out the first 5 coordinates of your path.** If your output displays raw pixel values (e.g., $X = 250, Y = 140$), you will fail this step. You must demonstrate proper metric scaling (e.g., $X = 0.12, Y = -0.05$).

2. **Provide a screenshot** of the isolated contours rendering correctly as "Digital Ink" in the RViz landscape based on your `/drawing_ink` marker.

6 Task 2: Kinematic Trees & Transformations (20 Marks)

Goal: Prove your mathematical understanding of TF2, kinematic chains, and homogeneous transformation matrices.

6.1 Step A: The TF Tree

Launch the ROS 2 environment and the static transform publisher (see commands in Task 3). Open a new terminal and generate your frame tree using the `tf2_tools` utility:

```
ros2 run tf2_tools view_frames
```

6.2 Step B: Manual Matrix Calculation

Pick **one** 3D metric point from your Task 1 output (local paper coordinates). Write down the homogeneous transformation matrix that represents the relationship between `paper_frame` and `wx250s/base_link`. Manually multiply your point by this matrix to find its world coordinate.

6.3 Step C: Code Integration

In `robot_drawer.py`, complete **TODO 3** and **TODO 4**. Use `lookup_transform` and `do_transform_pose` to automate the math you just performed manually.

6.4 Deliverable 2

1. Include the `frames.pdf` generated by TF2 in your report.
2. Show your manual matrix multiplication for the single point.
3. Provide a print statement from your Python script proving the code outputs the exact same transformed coordinate as your hand calculation.

7 Task 3: Cartesian Path Planning & Generalization (20 Marks)

Goal: Introduce MoveIt Commander, apply orientation constraints, and execute the final path.

7.1 Step A: Orientation Constraints

If you use standard `go_to_pose` functions, joints will interpolate in arcs, lifting the pen off the paper. We must use `compute_cartesian_path`. Complete **TODO 5** by defining the end-effector orientation. Use the quaternion $[x : 0.0, y : 0.7071, z : 0.0, w : 0.7071]$ to force the IK solver to keep the pen pointed 90-degrees downward.

7.2 Step B: System Execution

Ensure your RViz is configured to display a **Marker** on the `/drawing_ink` topic. Run the pipeline:

Step-by-Step Execution

1. **Terminal 1 (MoveIt):**
   `ros2 launch interbotix_xsarm_moveit xsarm_moveit.launch.py robot_model:=wx250s hardware_type:=fake`
2. **Terminal 2 (Static TF):** Define paper position:
   `ros2 run tf2_ros static_transform_publisher --x 0.3 --y 0.0 --z 0.062 --yaw 0 --pitch 0 --roll 0 --frame-id wx250s/base_link --child-frame-id paper_frame`
3. **Terminal 3 (Visualizer):**
   `python3 paper_visualiser.py`
4. **Terminal 4 (Main Script):** Run your drawing pipeline:
   `python3 main.py --image batman.png`

7.3 Step C: The Triple-Move Challenge

A robust robotic pipeline should not require code changes if the target moves. **Restart Terminal 2** with the following coordinates to verify your system adapts dynamically:

Test	X-offset	Y-offset	Z-offset	Result
1 (Center)	0.30m	0.00m	0.062m	Success/Fail
2 (Left)	0.25m	0.08m	0.062m	Success/Fail
3 (Right)	0.28m	-0.05m	0.062m	Success/Fail

Critical Note

If the planner returns a "Fraction < 1.0", the paper is outside the reachable workspace.

7.4 Deliverable 3

Show screenshots of RViz successfully rendering the Batman logo as `/drawing_ink` alongside the robot arm in **all three** locations from the table above.

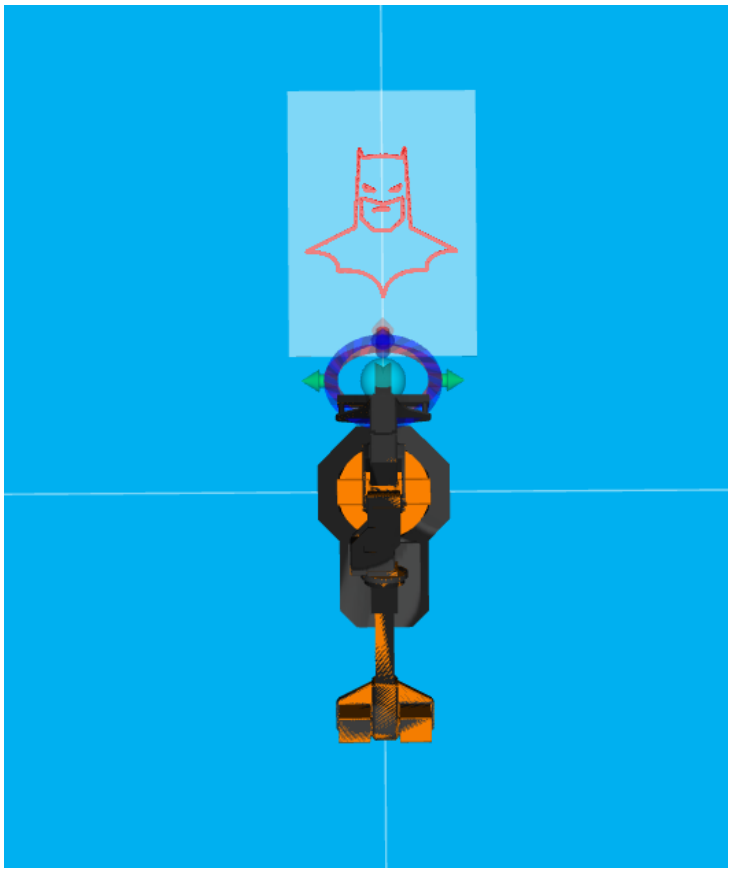

Figure 2: Expected Batman logo rendered as digital ink in RViz.

A Troubleshooting Guide

- **Stuttering Motion:** Ensure you are downsampling your contours (e.g., `contour[::5]`). MoveIt will crash if you send thousands of dense points.
- **Mirrored Drawing:** Check the signs in your scaling math (TODO 2).
- **No Ink in RViz:** In RViz, click **Add -¿ Marker** and set the topic to `/drawing_ink`. Ensure your Fixed Frame matches the coordinate space you are publishing to.