Group7 Lab3
Path Planning with the Mycobot Pro 600

Andy Tsai
ctsai67@asu.edu
1233653410

April 11, 2025

Abstract

This project focuses on developing a vision-based path planning pipeline for the MyCobot Pro 600 robot. The goal was to extract a path from an image, convert it into real-world coordinates, and generate joint commands that allow the robot to follow the path accurately. I chose to use ROS 2 and RViz2 for simulation, testing, and integration with the physical robot. The workflow involved creating a digital twin from CAD files, applying image processing to skeletonize the path, mapping image coordinates to physical space, and solving inverse kinematics to get joint angles. The robot was able to follow the extracted path in both simulation and the real world, showing that the full planning pipeline worked as intended.

Introduction

In this lab, I used ROS 2 and RViz2 to simulate, test, and control the MyCobot Pro 600 robot for a vision-based path planning task. The goal was to get the robot to follow a path detected from an image — both in simulation and on the real robot. To do that, I first built a digital twin of the robot by converting the provided CAD file into a URDF using SolidWorks, which I then loaded into my ROS 2 workspace for visualization in RViz2.

Next, I worked on image processing to extract the path from a camera image. I used skeletonization to trace the centerline of the path, which makes the system more flexible because it can handle curves and not just straight lines. Then, I manually measured real-world coordinates of the camera's view — including the home position and three other points — to transform image points into physical coordinates.

Once I had the path in real-world coordinates, I used inverse kinematics to calculate the joint angles needed to reach each point. I published these angles to the `/joint_states` topic in ROS 2, which updated the robot's pose in both the RViz simulation and the physical robot through TCP commands. In the end, I was able to control the robot to follow the path captured by the camera — both virtually and in the real world.

Method

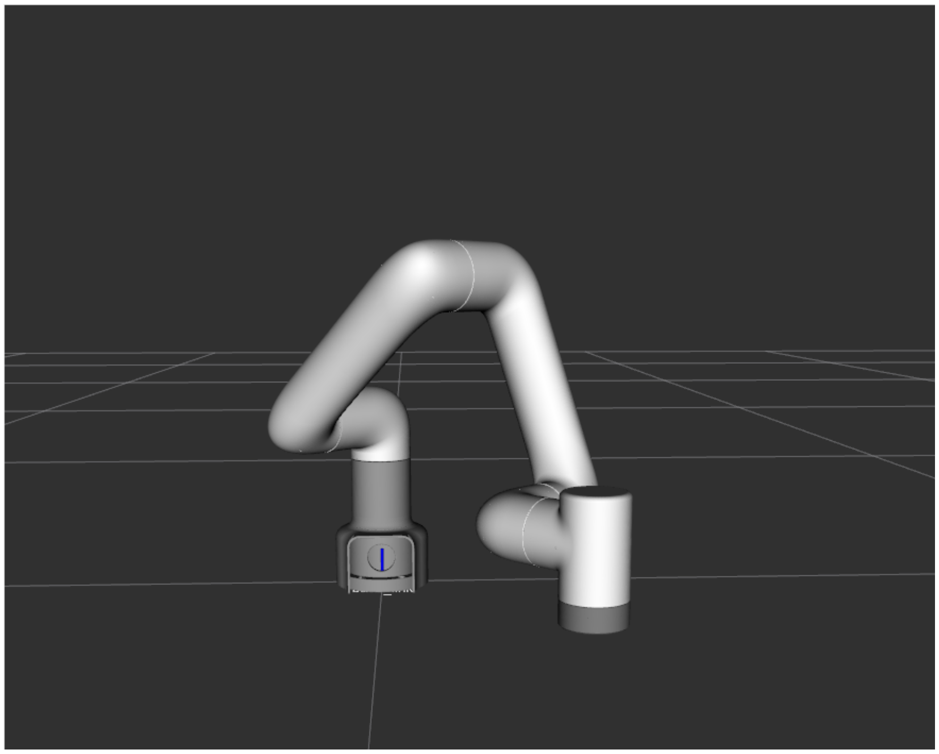
The workflow for implementing vision-based path planning with the MyCobot Pro 600 is as follows:

1. **URDF Creation**
   - Used SolidWorks to convert the provided CAD model into a URDF file.
   - Loaded the URDF into a ROS 2 workspace to visualize and simulate the robot in RViz2.
2. **Path Extraction from Image**
   - Captured a top-down image of the workpiece using a fixed camera.
   - Cropped the image to focus only on the usable workspace area, removing background and irrelevant regions.
   - Applied OpenCV-based image processing to isolate the drawn path.
   - Skeletonized the path to reduce it to a single-pixel-wide centerline.
   - Selected 10 evenly distributed points along the skeletonized path to simplify the planning and motion process.
3. **Image-to-World Coordinate Mapping**
   - Manually measured the physical coordinates of four key points in the camera's field of view (home corner + 3 reference vertices).
   - Computed a transformation to convert 2D image coordinates into real-world 3D coordinates.

4. **Inverse Kinematics (IK)**
   - Developed a custom IK solver to compute joint angles for each path point.
   - Introduced an angle offset constraint $\theta_2+\theta_3+\theta_4 = -\frac{\pi}{2}$ to ensure the end effector always points downward.
5. **Forward Kinematics (FK) Validation**
   - Used FK to validate the end-effector position from the solved joint angles.
   - Ensured positional consistency before sending joint commands to the robot.
6. **Robot Control and ROS 2 Integration**
   - Published joint angles to the `/joint_states` topic to drive both the RViz2 simulation and the real robot.
   - Used a ROS 2 timer to sequentially send joint commands.
   - Sent joint positions over TCP/IP to the physical robot using its API via Ethernet.
   - Verified that the motion of the physical robot matched the digital twin behavior.

Results

1. **The Cobot Model (Simulated in RViz2)**



2. **Path Extraction from Image**

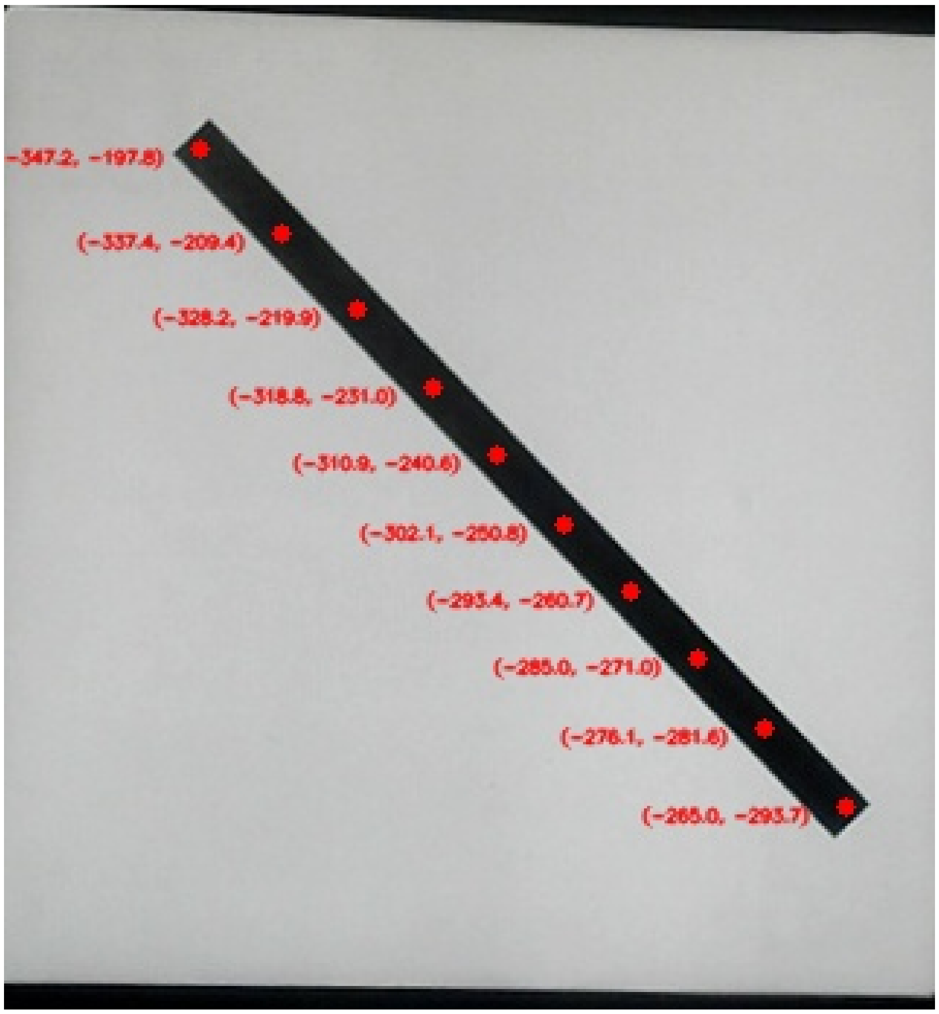
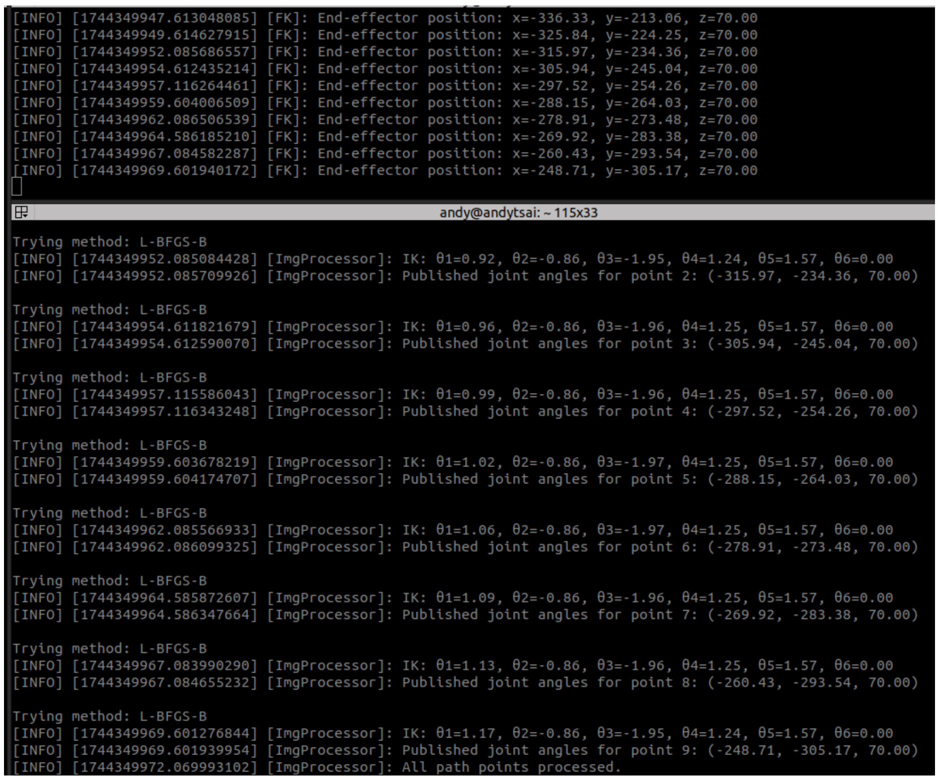


Figure 1: The extracted path alone with their coordinates correspondent to the real world coordinates.

3. **Inverse Kinematics & Forward Kinematics Validation**



```
[INFO] [1744349947.613048085] [FK]: End-effector position: x=-336.33, y=-213.06, z=70.00
[INFO] [1744349949.614627915] [FK]: End-effector position: x=-325.84, y=-224.25, z=70.00
[INFO] [1744349952.085686557] [FK]: End-effector position: x=-315.97, y=-234.36, z=70.00
[INFO] [1744349954.612435214] [FK]: End-effector position: x=-305.94, y=-245.04, z=70.00
[INFO] [1744349957.116264461] [FK]: End-effector position: x=-297.52, y=-254.26, z=70.00
[INFO] [1744349959.604006509] [FK]: End-effector position: x=-288.15, y=-264.03, z=70.00
[INFO] [1744349962.086506539] [FK]: End-effector position: x=-278.91, y=-273.48, z=70.00
[INFO] [1744349964.586185210] [FK]: End-effector position: x=-269.92, y=-283.38, z=70.00
[INFO] [1744349967.084582287] [FK]: End-effector position: x=-260.43, y=-293.54, z=70.00
[INFO] [1744349969.601940172] [FK]: End-effector position: x=-248.71, y=-305.17, z=70.00
```

andy@andytsai: ~ 115x33

```
Trying method: L-BFGS-B
[INFO] [1744349952.085084428] [ImgProcessor]: IK: θ1=0.92, θ2=-0.86, θ3=-1.95, θ4=1.24, θ5=1.57, θ6=0.00
[INFO] [1744349952.085709926] [ImgProcessor]: Published joint angles for point 2: (-315.97, -234.36, 70.00)

Trying method: L-BFGS-B
[INFO] [1744349954.611821679] [ImgProcessor]: IK: θ1=0.96, θ2=-0.86, θ3=-1.96, θ4=1.25, θ5=1.57, θ6=0.00
[INFO] [1744349954.612590070] [ImgProcessor]: Published joint angles for point 3: (-305.94, -245.04, 70.00)

Trying method: L-BFGS-B
[INFO] [1744349957.115586043] [ImgProcessor]: IK: θ1=0.99, θ2=-0.86, θ3=-1.96, θ4=1.25, θ5=1.57, θ6=0.00
[INFO] [1744349957.116343248] [ImgProcessor]: Published joint angles for point 4: (-297.52, -254.26, 70.00)

Trying method: L-BFGS-B
[INFO] [1744349959.603678219] [ImgProcessor]: IK: θ1=1.02, θ2=-0.86, θ3=-1.97, θ4=1.25, θ5=1.57, θ6=0.00
[INFO] [1744349959.604174707] [ImgProcessor]: Published joint angles for point 5: (-288.15, -264.03, 70.00)

Trying method: L-BFGS-B
[INFO] [1744349962.085566933] [ImgProcessor]: IK: θ1=1.06, θ2=-0.86, θ3=-1.97, θ4=1.25, θ5=1.57, θ6=0.00
[INFO] [1744349962.086099325] [ImgProcessor]: Published joint angles for point 6: (-278.91, -273.48, 70.00)

Trying method: L-BFGS-B
[INFO] [1744349964.585872607] [ImgProcessor]: IK: θ1=1.09, θ2=-0.86, θ3=-1.96, θ4=1.25, θ5=1.57, θ6=0.00
[INFO] [1744349964.586347664] [ImgProcessor]: Published joint angles for point 7: (-269.92, -283.38, 70.00)

Trying method: L-BFGS-B
[INFO] [1744349967.083990290] [ImgProcessor]: IK: θ1=1.13, θ2=-0.86, θ3=-1.96, θ4=1.25, θ5=1.57, θ6=0.00
[INFO] [1744349967.084655232] [ImgProcessor]: Published joint angles for point 8: (-260.43, -293.54, 70.00)

Trying method: L-BFGS-B
[INFO] [1744349969.601276844] [ImgProcessor]: IK: θ1=1.17, θ2=-0.86, θ3=-1.95, θ4=1.24, θ5=1.57, θ6=0.00
[INFO] [1744349969.601939954] [ImgProcessor]: Published joint angles for point 9: (-248.71, -305.17, 70.00)
[INFO] [1744349972.069993102] [ImgProcessor]: All path points processed.
```

Figure 2: The bottom window is the output messages of applying joints angles from inverse kinematics. The top window is the validation by forward kinematics.

4. **ROS 2 Nodes Graph**

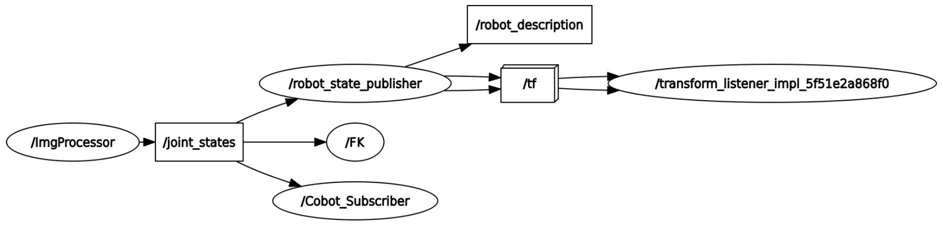


Figure 3: This is the ROS 2 node graph for the path planning pipeline. The `/ImgProcessor` node publishes joint angles to `/joint_states`, which are used by `/robot_state_publisher` for visualization, `/FK` for verification, and `/Cobot_Subscriber` to send commands to the real robot. Transforms are broadcast via `/tf` from the `/robot_description` model.

Conclusion

In this lab, I successfully built a full pipeline that allowed the MyCobot Pro 600 to follow a path extracted from a camera image. I created the robot's URDF model, set up the simulation in RViz2, and handled image processing to extract and skeletonize the path. After mapping image coordinates to real-world space, I used inverse kinematics with an end-effector constraint to compute joint angles, then used ROS 2 and TCP communication to control both the simulated and physical robot. The robot was able to follow the selected path accurately, and the motion in the real world closely matched the RViz2 simulation.
Although this was a group project, I completed all parts of the lab on my own, as my teammates did not contribute. Despite the added workload, this experience gave me a deeper understanding of how image processing, robot kinematics, and ROS 2 integration come together in a real-world robotics application.

References

[1] Course Materials

[2] ROS2 Tutorials - ROS2 Humble For Beginners

[3] MyCobot Pro 600-Official Website

[4] MoveIt2 and ROS2 Control: How to Configure & Fix Motion Planning Issues with Setup Assistant

[5] OpenCV Course - Full Tutorial with Python