Group7 Final Project
Smart and Accurate Path Planning with the Mycobot Pro 600: Solving Mazes

Andy Tsai
ctsai67@asu.edu
1233653410

Michael Bogale
mabogale@asu.edu
1235712207

Lalgudi Shivakumar Ganesh Rathnam
glalgudi@asu.edu
1233693450

October 27, 2025

Abstract

This project builds on the curved path planning work from Lab 4, but now focuses on solving mazes. We developed an image-based A* search pipeline that detects the shortest path through a maze and translates it into real-world coordinates. The MyCobot Pro 600 is then controlled through inverse kinematics and TCP commands to follow this path. Compared to Lab 4, the major change is replacing curve-following interpolation with a full maze-solving algorithm wrapped in a ROS 2 node.

Introduction

In Lab 4, we implemented a system to follow curved paths using image processing and kinematic control. For this final project, we extended that system to enable the robot to solve and follow a maze autonomously. This required implementing a pathfinding algorithm (A*) and modifying the binary maze map to push the resulting path away from walls for safer execution. The camera still provides the visual input, and pixel-to-world coordinate transformation is done through homography.

We reused the same digital twin and ROS 2 infrastructure as before, and the robot's downward-facing constraint was still enforced via an IK constraint: $\theta_2 + \theta_3 + \theta_4 = -\frac{\pi}{2}$.

Method

1. URDF and RViz Setup

We used the same URDF and RViz setup from Lab 4. No changes were made to the digital twin or robot model.

2. Image Processing and Preprocessing

A top-down image of the maze was captured and cropped. OpenCV was used to:

- Detect red and green markers (start and goal)
- Binarize the maze image and skeletonize it
- Apply morphological dilation to thicken walls, making the path stay clear of them

3. Path Planning (A* Search)

We implemented a standard A* algorithm using a distance transform-based cost map. This ensures that the chosen path stays near the center of the maze corridors. Only the shortest path was selected.

4. Coordinate Transformation

We used four reference points (maze corners) to compute a homography matrix and map all path points from pixel space to world space.

5. Inverse Kinematics

Each world coordinate is converted into a 6-DOF joint configuration using our previous IK solver with the $\theta_2 + \theta_3 + \theta_4 = -\frac{\pi}{2}$ constraint.

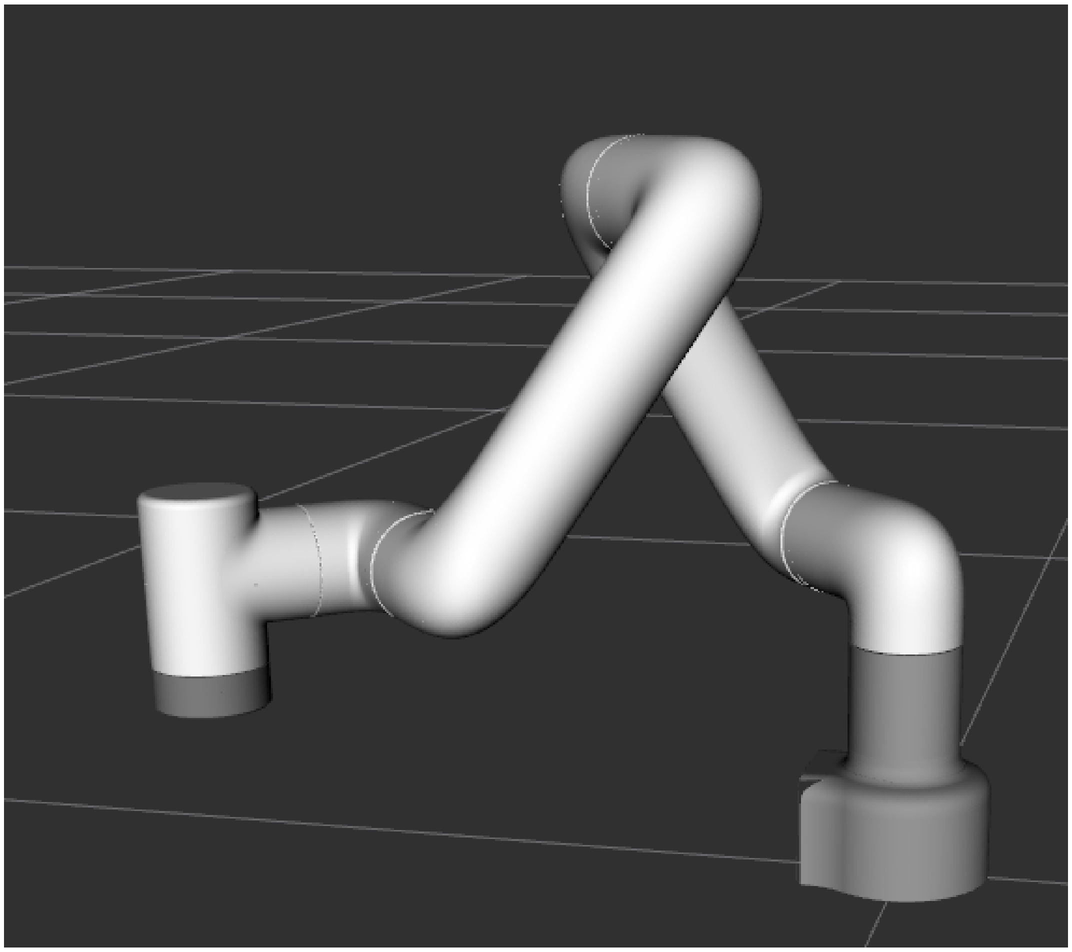
6. ROS 2 Integration and Actuation

The full pipeline is wrapped inside a ROS 2 node called `maze_solver_publisher.py`, which:

- Publishes joint angles to `/joint_states`
- Skips points that create sudden jumps in joint space
- Sends joint commands to the real robot with time delays

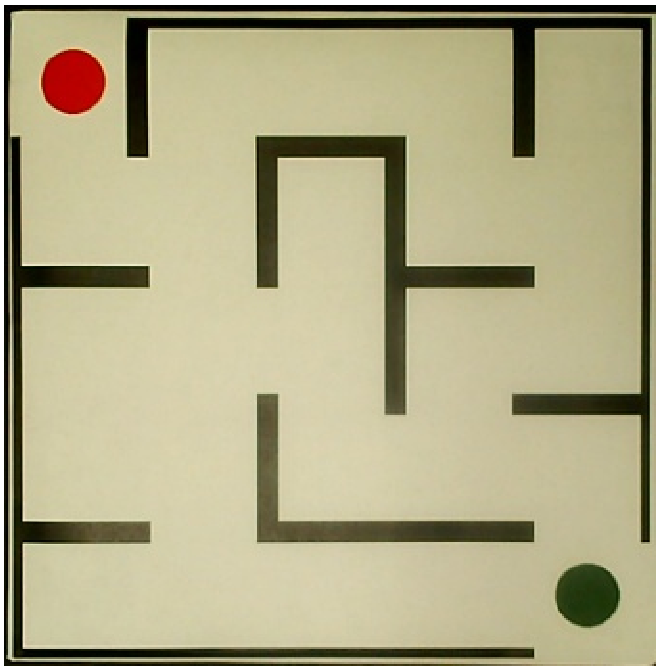
Results

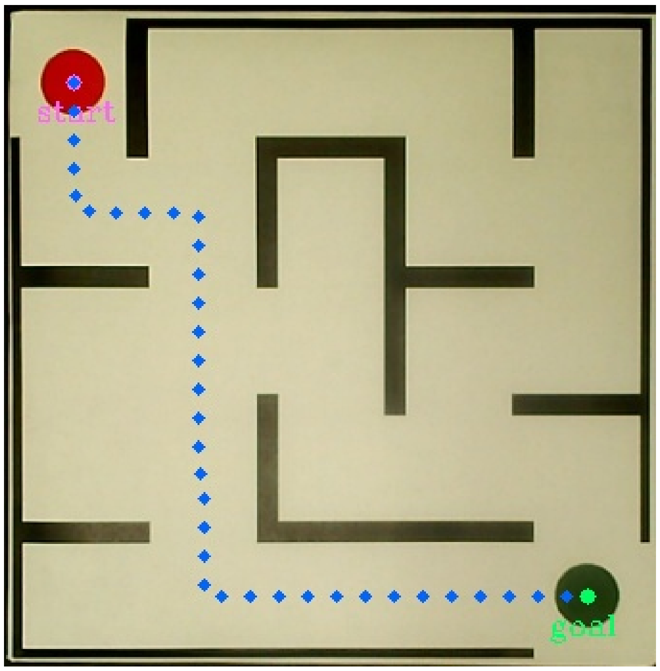
1. **The Cobot Model (Simulated in RViz2)**

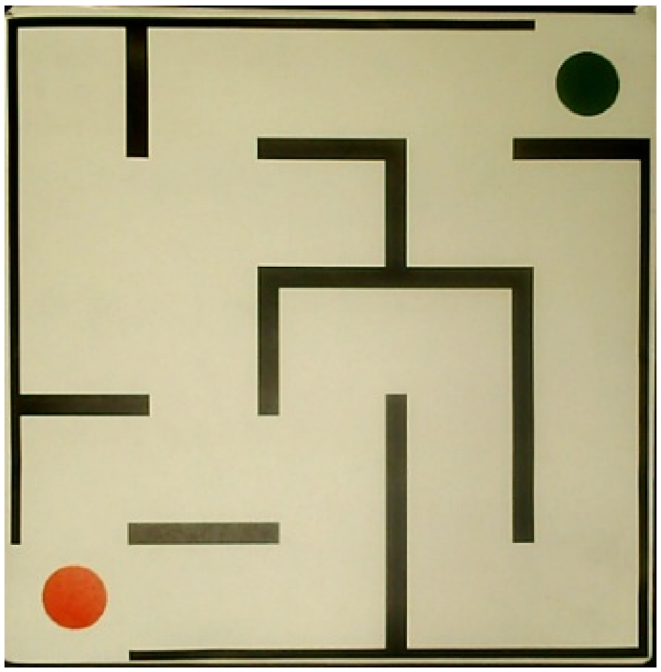


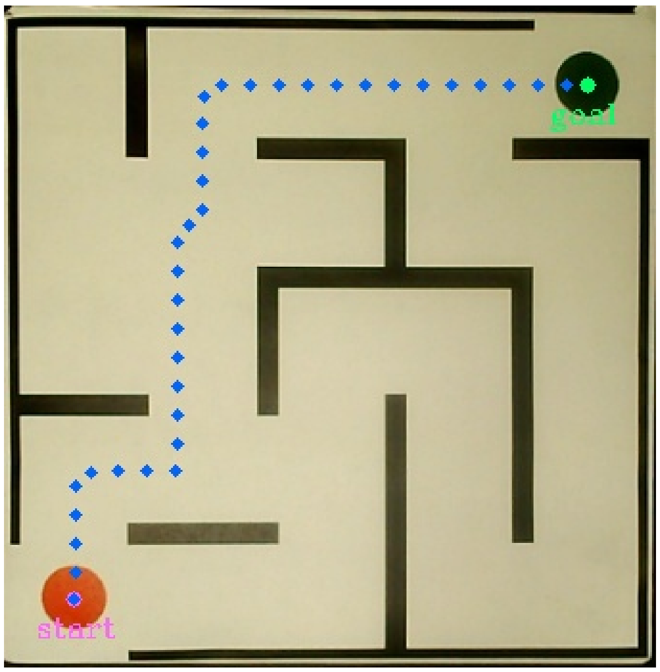
2. **Path Extraction**
   Left image: Original maze image with red start and green goal markers.

   Right image: Shortest path generated by the A* algorithm, overlaid as a blue dotted line from start to goal.

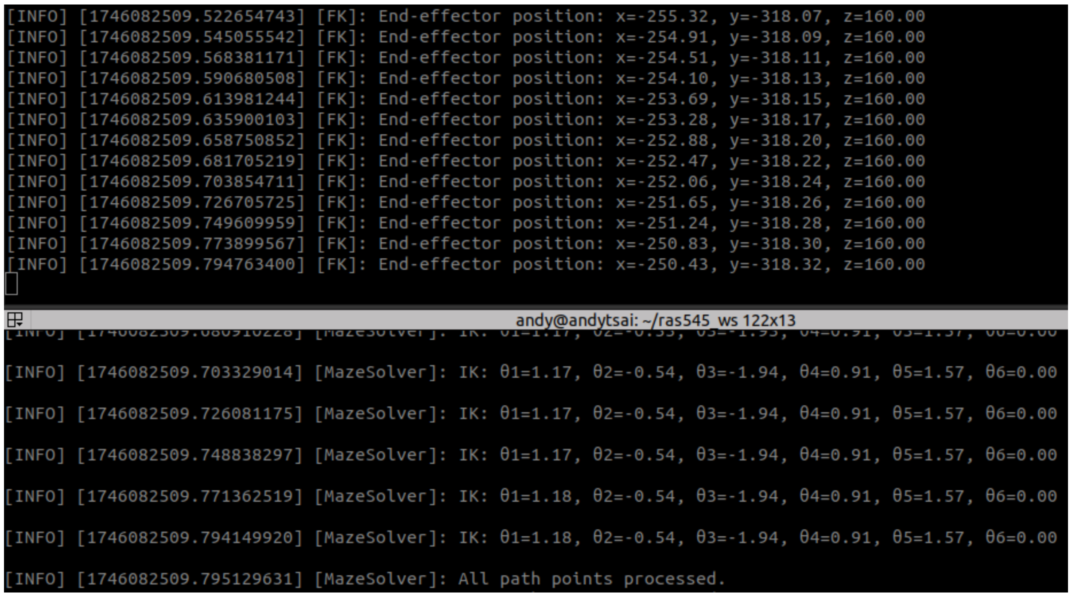








3. **Inverse Kinematics & Forward Kinematics Validation**



```
[INFO] [1746082509.522654743] [FK]: End-effector position: x=-255.32, y=-318.07, z=160.00
[INFO] [1746082509.545055542] [FK]: End-effector position: x=-254.91, y=-318.09, z=160.00
[INFO] [1746082509.568381171] [FK]: End-effector position: x=-254.51, y=-318.11, z=160.00
[INFO] [1746082509.590680508] [FK]: End-effector position: x=-254.10, y=-318.13, z=160.00
[INFO] [1746082509.613981244] [FK]: End-effector position: x=-253.69, y=-318.15, z=160.00
[INFO] [1746082509.635900103] [FK]: End-effector position: x=-253.28, y=-318.17, z=160.00
[INFO] [1746082509.658750852] [FK]: End-effector position: x=-252.88, y=-318.20, z=160.00
[INFO] [1746082509.681705219] [FK]: End-effector position: x=-252.47, y=-318.22, z=160.00
[INFO] [1746082509.703854711] [FK]: End-effector position: x=-252.06, y=-318.24, z=160.00
[INFO] [1746082509.726705725] [FK]: End-effector position: x=-251.65, y=-318.26, z=160.00
[INFO] [1746082509.749609959] [FK]: End-effector position: x=-251.24, y=-318.28, z=160.00
[INFO] [1746082509.773899567] [FK]: End-effector position: x=-250.83, y=-318.30, z=160.00
[INFO] [1746082509.794763400] [FK]: End-effector position: x=-250.43, y=-318.32, z=160.00
```

```
andy@andytsai: ~/ras545_ws 122x13
```

```
[INFO] [1746082509.703329014] [MazeSolver]: IK: θ1=1.17, θ2=-0.54, θ3=-1.94, θ4=0.91, θ5=1.57, θ6=0.00

[INFO] [1746082509.726081175] [MazeSolver]: IK: θ1=1.17, θ2=-0.54, θ3=-1.94, θ4=0.91, θ5=1.57, θ6=0.00

[INFO] [1746082509.748838297] [MazeSolver]: IK: θ1=1.17, θ2=-0.54, θ3=-1.94, θ4=0.91, θ5=1.57, θ6=0.00

[INFO] [1746082509.771362519] [MazeSolver]: IK: θ1=1.18, θ2=-0.54, θ3=-1.94, θ4=0.91, θ5=1.57, θ6=0.00

[INFO] [1746082509.794149920] [MazeSolver]: IK: θ1=1.18, θ2=-0.54, θ3=-1.94, θ4=0.91, θ5=1.57, θ6=0.00

[INFO] [1746082509.795129631] [MazeSolver]: All path points processed.
```

Figure 1: 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**

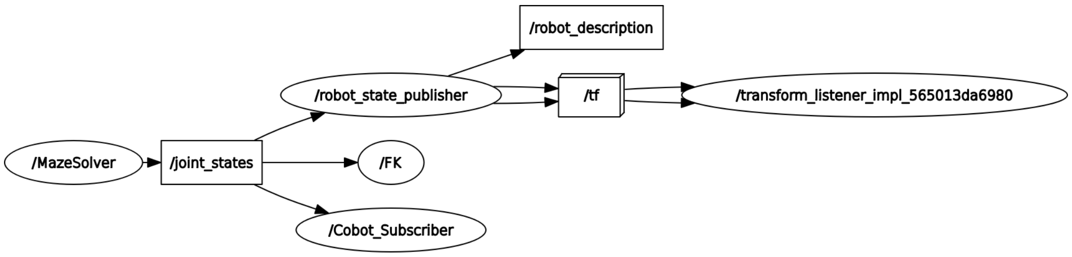


Figure 2: ROS 2 node graph showing data flow between the image processing, inverse kinematics, and actuation nodes. The `maze_solver_publisher` node handles perception and planning, while joint angles are published to `/joint_states` and sent to the real robot.

Conclusion

This final project successfully demonstrated autonomous maze solving using image processing and A* search. The robot was able to follow the shortest path safely and smoothly. By building on Lab 4, we avoided redundant work and focused on integrating planning intelligence into the system. The biggest takeaway is how useful map dilation and cost maps can be in real-world robot navigation.

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