Team Falcon – Liquibot (Bartender Robot)

Yogi Shah
EECE Department
MS in Robotics
shah.yo@northeastern.edu

Sushma Aare
MIE Department
MS in Robotics
aare.s@northeastern.edu

Srinidhi Pattala
EECE Department
MS in Robotics
pattala.s@northeastern.edu

Prathima Anand Kumar
EECE Department
MS in Robotics
anandakumar.p@northeastern.edu

***Abstract*: Liquibot is an autonomous bartending robot designed to replicate the complete drink preparation process, from syrup handling to serving. Mounted on a mecanum-wheeled base capable of omnidirectional motion, Liquibot features a robotic arm that performs complex bartending tasks such as pouring various liquids, adding ice cubes, stirring or shaking drinks, and accurately dispensing water into a glass. The robot is designed to operate in a structured environment using visual markers and pre-defined trajectories for precise localization and manipulation. By integrating advanced control systems, ROS-based automation, and custom mechanical design, Liquibot aims to deliver interactive and efficient robotic bartending experience.**

I. Introduction

Liquibot is inspired by the infamous Makr Shakr bartending robot, known for its ability to mix and serve drinks using two 6+ DOF robotic arms. We were intrigued by the idea and wanted to explore whether similar levels of slosh control and drink-making precision could be achieved on a mobile platform. The thought of having a bartending system on wheels opened up exciting possibilities, especially in terms of increasing reach and accessibility in a larger workspace.

Liquibot is designed as an autonomous robotic bartending system capable of navigating, detecting, and serving beverages in a semi-structured environment. The robot features a 5-DOF robotic arm mounted on a mecanum-wheeled base, enabling omnidirectional movement in eight directions across the planar surface. This setup allows the robot to move freely around a space, align itself accurately with drink stations, and carry out a variety of bartending tasks.

Our goal is to have Liquibot autonomously localize a cup, navigate to its location, and successfully pick it up using real-time perception and inverse kinematics (IK). By combining mobility and manipulation, we aim to bring a new level of autonomy and adaptability to robotic bartending.

II. Implementation

A. Hardware

We built a mecanum-wheeled mobile base using four JGB37-520 motors controlled by two Cytron motor drivers, all managed by an ESP32 Devkit V1 on a custom chassis. Mounted on top is the ReactorX-200 robotic arm from Trossen Robotics, connected via USB to a laptop that also powers the system. For object manipulation, we designed a custom gripper capable of handling two cup sizes smaller shot glasses for syrups and larger cups for drink preparation. A Realsense D455 camera is mounted on the robotic arm, providing real-time perception and depth data, also powered and controlled through the laptop.

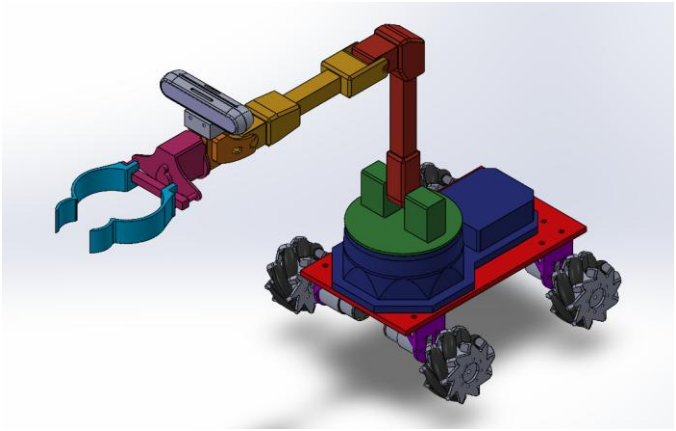

Fig. 1. CAD Assembly of the complete robot with the robotic arm.

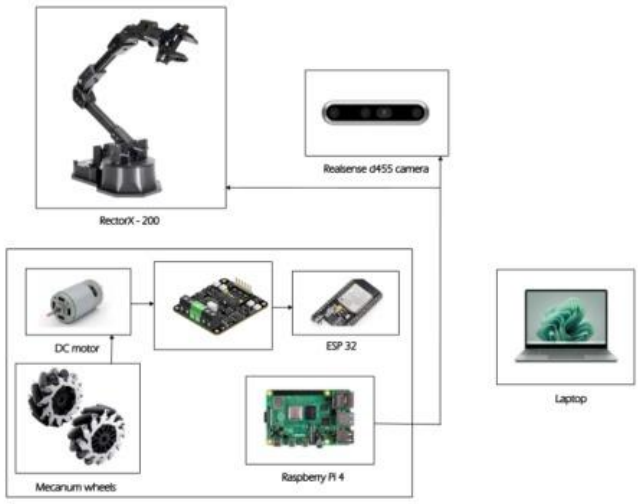

Fig. 2. Hardware flowchart.

B. Perception

Cup detection using the Realsense camera begins with a training phase where an automated script captures multiple images of various cups. These images are then labeled using RoboFlow AI, and the annotated dataset is used to train a YOLOv8 model locally. The dataset is split into training and validation sets to ensure effective model performance. Since the Realsense D455 camera inherently provides depth estimation, no additional depth algorithms were required.

Identify applicable funding agency here. If none, delete this text box.

For localization and mapping, the robot employs RTAB-Map, a graph-based SLAM framework. RTAB-Map fuses RGB-D data from the Realsense and wheel odometry to generate a real-time 3D occupancy grid of the environment, enabling effective navigation and spatial awareness.

C. Planning

As part of Liquibot's path and motion planning setup, we've created a simple but effective system using ROS nodes that talk to each other to handle everything from detecting cups to moving the robot and making the drink. The main brain here is the planner_node, which subscribes to the cup_detection_node this node sends out a list of all the cups it sees, along with their poses and IDs. The planner then checks if the main cup (the one we need for the drink) is within the workspace of the robotic arm. For our setup, we set this workspace as anything within 15 cm depth from the cup.

If the cup is outside this range, the planner hands things off to the navigation_node. This node oversees localization and SLAM using RTAB-Map, which uses the point cloud from the Realsense camera and odometry data to build a map of the environment. It plans a path to the cup using the A* algorithm, making sure to avoid any obstacles and giving us a real-time updated, optimized trajectory. The navigation node sends velocity commands VX and VY for x and y movement. These values are sent over serial to the ESP32, which then converts them into power and strafe commands for our mecanum wheels, letting the robot move smoothly in any direction.

Now, if the cup is already within reach, the planner skips the navigation and directly runs the controller_node. This is where the actual drink-making happens. It takes cup poses from the cup_data topic and runs a set sequence of actions calculating IK for the robotic arm to go and grab each cup, pouring the ingredients, and doing whatever action the drink needs, like stirring or shaking. All of this is automated and handled in a clean flow.

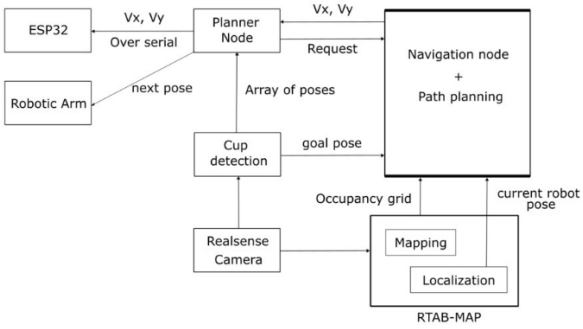

Fig. 3. Flow of the ROS Network.

D. Drink making Sequence

Step 1: open the gripper and move to cup location
Step 2: grasp the cup
Step 3: check the gripper pressure and lift up
Step 4: move to pour location with a offset relative to main cup
Step 5: perform pour action by roll motion
Step 6: return to original cup location
Step 7: release the cup
Step 8: Lift up and go to home pos
This is repeated for the syrup and the water to make the drink.

III. Experiments and Demonstrations

At the start of our experiments, we began by breaking down the architecture and testing each part individually to make sure things worked before putting it all together. We first got navigation and path planning running on one system, while planning logic was developed on a separate setup. The mecanum car and robotic arm were initially connected to a Raspberry Pi along with the camera, all wirelessly communicating with each other. Later on, we decided to remove the Raspberry Pi setup which we'll go into more in the challenges section.

One of our main experiments was integrating the planner_node and navigation_node on a single system and connecting the car via serial to verify end-to-end functionality. This test checked if the planner could correctly subscribe to the cup detection node, request navigation when the cup was out of reach, and ensure the navigation node published the right VX and VY values for the wheels. The goal was to see if all nodes worked in sync during runtime.

For our live demo, we showed how the robot behaves when a detected cup is within the 15 cm arm workspace. With object detection running, the planner triggers the drink-making sequence. However, during this test, we ran into an issue with the IK solver. It kept entering an error state, which caused the arm to execute faulty or incomplete poses while trying to reach the cup basically messing up the action sequence.

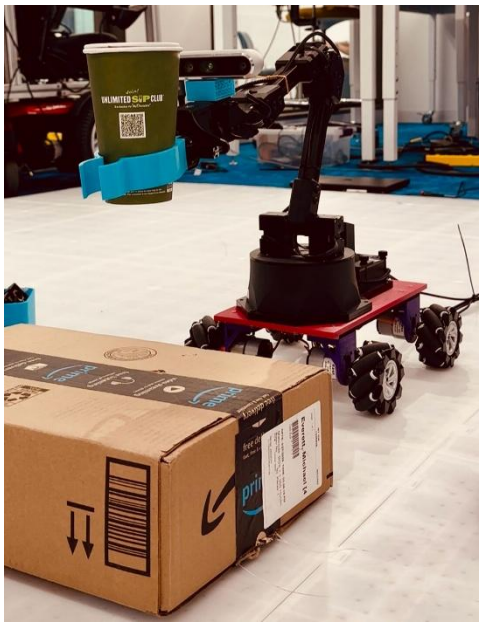

Fig. 4. Complete pipeline testing with human replacement for IK graping Issue.

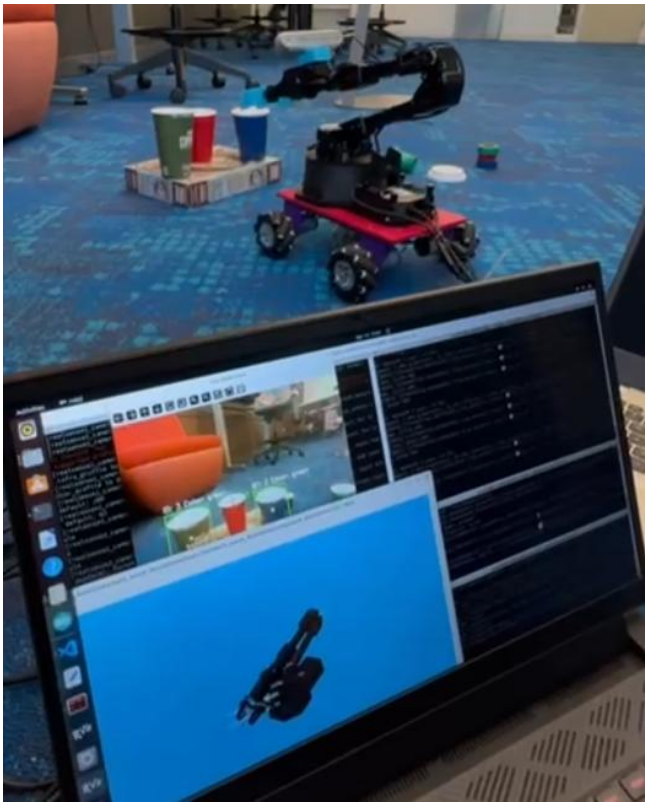

Fig. 5. Preliminary testing of grasping with object detection.

IV. Challenges

Throughout the development of Liquibot, we encountered several technical and integration challenges across both hardware and software components. Below are some of the key issues we faced and how we addressed them.

A. AprilTag Detection Accuracy

We initially explored AprilTag detection for object localization, but ran into accuracy issues. To fix this, we performed intrinsic camera calibration using a standard checkerboard pattern and OpenCV's calibration toolbox. This gave us accurate intrinsic parameters (focal lengths, principal point, and distortion coefficients), which we used to undistort image frames. This significantly improved pose estimation and localization during AprilTag detection.

B. Cup Detection and Dataset Training

One of the bigger challenges was creating a high-quality, labeled dataset to train our YOLOv8 model to detect cups. We used a Logitech monocular webcam to capture images of cups under varying lighting, angles, and colors. The annotation process was fully manual using LabelImg, where we assigned class names like red_cup and blue_cup for each bounding box. We tried auto-labeling tools initially but found them inaccurate for our custom objects. Organizing the data into YOLO format (train/val split for both images and labels) was tricky due to mismatched file names and annotation issues. To streamline this, we wrote helper scripts to validate dataset integrity. In the end, this manual approach helped us train a well-performing custom cup detection model.

C. Cup Detection with Logitech Webcam

Using a basic Logitech webcam for cup detection presented several issues. It lacks depth sensing, so estimating 3D cup position was tough. We tried monocular depth estimation models like MiDaS and Depth Anything, but they were resource-heavy and not accurate in real time. The webcam also struggled with poor lighting and low resolution, leading to inconsistent detection. We optimized the pipeline by reducing image resolution and calibrating the camera, but ultimately decided to switch to the Realsense D455, which gave us reliable depth data out of the box and improved detection accuracy significantly.

D. Depth Estimation with MiDaS

Working with the MiDaS model for depth estimation was one of the more frustrating parts. With just RGB input, depth values had ±15 cm error, which was unusable for precise manipulation. Integration was difficult too estimating cup pose required combining YOLO detection, depth estimation, and camera intrinsics. Even close-up cups were reported at 5 meters sometimes, while distant ones returned less depth. After many tests and inconsistent results, we switched to the Realsense D455, which solved most of our depth issues with its native RGB-D output.

E. Navigation and Integration

Navigation was a tough challenge, mainly due to integrating multiple systems SLAM, object detection, and motion planning into a synchronized pipeline. Converting the cup's pose into the global frame, planning a safe path using A*, and calculating proper wheel velocities for omnidirectional motion all had to work in real time. Getting this to trigger the arm at the right time added another layer of complexity. These interdependencies made the system delicate but also helped us learn a lot about end-to-end robotics integration.

F. Raspberry Pi & ESP32 Setup

In our early setup, we ran the robotic arm, mecanum car, and camera through a Raspberry Pi that communicated wirelessly with the other systems. While it worked in smaller tests, syncing it with the ESP32 and the rest of the ROS ecosystem created latency and reliability issues. We later removed the Raspberry Pi and moved the control directly to a laptop using USB connections, which made the setup more stable and responsive.

G. Micro-ROS Setup with PlatformIO

To integrate low-level motor control, we used Micro-ROS to connect an ESP32 DevKit V1 to our main ROS 2 Humble system. The ESP32 was programmed via PlatformIO and acted as a Micro-ROS node to control motors and sensors. Meanwhile, tasks like cup detection, depth estimation, and planning were handled on the local system, which published commands via serial. The Raspberry Pi was initially used to relay messages but was later phased out for a more direct

connection, improving communication latency and system responsiveness.

H. Configuration Space & Singularity Issues

A key challenge we faced was with the RX-200's configuration space. The 3D plot revealed many target poses that were unreachable, which we only discovered through trial and error. The arm also encountered singularities some poses could be reached by moving in X and Z together, but failed when moved in sequence. These limitations stem from the robot's 5-DOF design, which lacks the flexibility needed for certain orientations or edge-case poses. A 6-DOF arm would likely solve most of these issues, offering greater maneuverability and eliminating many of the kinematic constraints we encountered during motion planning.

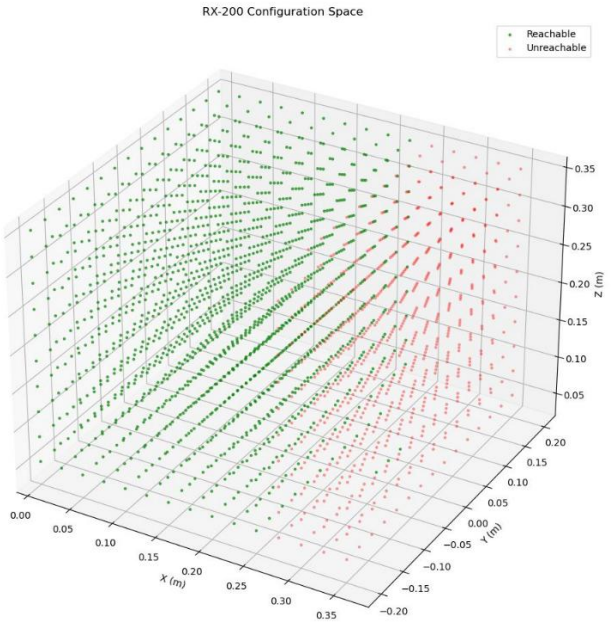

Fig. 6. Reachable and unreachable pose in the configuration Space of 5DOF ReactorX-200 ARM.

I. ORB-SLAM3 Mapping

Working with ORB-SLAM3 came with a few significant hurdles. First, the mapping process heavily depends on having enough visual features in the environment. If it doesn't detect sufficient features from earlier frames, it tends to start a new map altogether. To work around this, we had to place multiple AprilTags in the scene to improve feature consistency. Second, localization and loop closure were slower and more unreliable compared to RTAB-Map. ORB-SLAM3 would take noticeable time to re-align with the map, especially after movement. Lastly, ORB-SLAM3 doesn't provide a 2D occupancy grid directly we had to convert 3D point clouds manually, which introduced scaling issues. In contrast, RTAB-Map gives a clean 2D grid, simplifying path planning considerably.

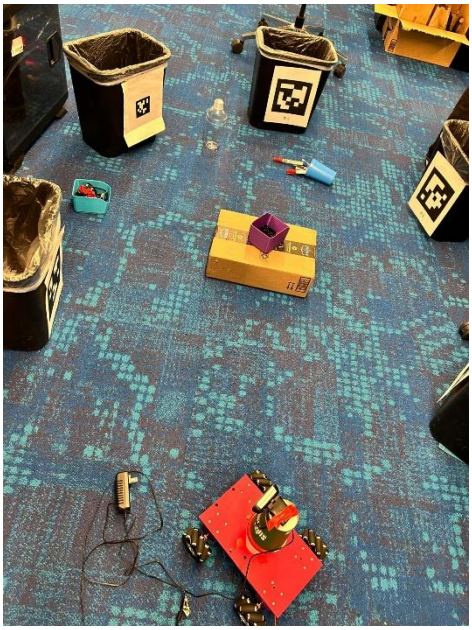

Fig. 7. Mapping enviornment for the ORBSLAM3 testing..

V. Future Scope

If we get access to a 6-DOF robotic arm with a higher payload, that would be one of the first upgrades we'd go for. It would make inverse kinematics much more reliable and eliminate a lot of the pose-not-found errors we currently face, as we'd have enough degrees of freedom to reach more complex poses detected by the camera. With a stronger arm, we could also experiment with heavier tools like a 150g shaker, allowing us to actually shake and mix real drinks.

Another key area for improvement is adding encoders to the DC motors for better position control, along with an IMU to assist with camera synchronization and pose verification. For the object detection pipeline, we'd like to go beyond YOLO and explore advanced vision models potentially those based on vision transformers (ViTs) which are more robust to varying lighting, background clutter, and distance. These models would also reduce the amount of manual pre-training and dataset tuning currently needed.

We also plan to further refine the planning pipeline, add a user-friendly interface for drink selection, and extend the robot's role to include drink delivery after preparation. Lastly, we want to demonstrate realistic slosh-controlled motion and achieve accurate, timed shaker movements to complete the full robotic bartending experience.

Acknowledgment

We would like to sincerely thank our Teaching Assistant and Professor Zhi Tan for their continuous support, guidance, and encouragement throughout the course of this project. Their insights and feedback were invaluable in helping us navigate the challenges and bring Liquibot to life.

REFERENCES

[1] Makr Shakr, “Robotic Bartending System.” [Online]. Available: https://www.makrshakr.com

[2] Trossen Robotics, “ReactorX-200 Robot Arm.” [Online]. Available: https://www.trossenrobotics.com/reactorx-200-robot-arm.aspx

[3] Ultralytics, “YOLOv8 Object Detection,” GitHub repository. [Online]. Available: https://github.com/ultralytics/ultralytics

[4] Roboflow, “Dataset Management and Annotation Tool.” [Online]. Available: https://roboflow.com

[5] Intel Corporation, “Intel RealSense Depth Camera D455.” [Online]. Available: https://www.intelrealsense.com/depth-camera-d455

[6] IntRoLab, “RTAB-Map: Real-Time Appearance-Based Mapping,” GitHub repository. [Online]. Available: https://github.com/introlab/rtabmap

[7] UZ-SLAMLab, “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM,” GitHub repository. [Online]. Available: https://github.com/UZ-SLAMLab/ORB_SLAM3

[8] ISL-ORG, “MiDaS: Monocular Depth Estimation,” GitHub repository. [Online]. Available: https://github.com/isl-org/MiDaS

[9] ISL-ORG, “Depth Anything,” GitHub repository. [Online]. Available: https://github.com/isl-org/Depth-Anything

[10] Micro-ROS, “Micro-ROS on ESP32,” [Online]. Available: https://micro.ros.org/docs/tutorials/advanced/esp32/

[11] Open Robotics, “ROS 2 Documentation (Humble Hawksbill),” [Online].