Autonomous Object Detection and Retrieval with Stretch Robot

Nikhil Gutlapalli
M.S. in Robotics
Northeastern University
Boston, MA, USA
gutlapalli.n@northeastern.edu

Ruchik Jani
M.S. in Robotics
Northeastern University
Boston, MA, USA
jani.ru@northeastern.edu

Shubhankar Katekari
M.S. in Robotics
Northeastern University
Boston, MA, USA
katekari.s@northeastern.edu

Febin Wilson
M.S. in Robotics
Northeastern University
Boston, MA, USA
wilson.fe@northeastern.edu

***Abstract*—This project presents the development of an autonomous object retrieval system using the Stretch RE3 mobile manipulator. The system integrates key robotic capabilities, such as navigation, perception, and manipulation, within a unified pipeline to detect, locate, and grasp a target object, specifically a tennis ball. Using ROS 2 as the middleware, the robot performs SLAM-based navigation for autonomous exploration, employs RGB-D camera data and point cloud processing for object detection and localization, and executes a motion planning algorithm for arm trajectory generation and grasp execution. Upon successful grasp, the manipulator retracts to a predefined configuration. This work demonstrates the potential of mobile manipulators in service robotics applications, particularly in domestic and office environments where autonomous object retrieval is essential.**

Link to the code: here

Link to the demonstration video: here

***Index Terms*—Navigation, Detection, Tracking, Manipulation, Grasping, YOLOv5**

I. Introduction

The project begins with autonomous environment mapping using the SLAM Toolbox in ROS 2, leveraging the LiDAR and RGB-D sensors onboard the Stretch RE3 robot. Once the map is generated, the ROS 2 Navigation Stack is launched to localize the robot and guide it to a predefined goal location using a combination of global and local planners. Upon arrival, the robot performs a full 360-degree rotation to scan the environment and identify the target object, a tennis ball, using RGB-D image streams processed through OpenCV and point cloud clustering with the DBSCAN algorithm.

Following object detection, the robot autonomously navigates closer to the ball and uses its 6-DOF manipulator to grasp. Trajectory planning for arm motion is executed using the joint_trajectory_controller and position feedback from the robot's encoders. Once the object is securely grasped, the arm is retracted to a predefined resting pose.

This task is particularly challenging due to the need for real-time integration of multiple subsystems, such as Simultaneous Localization and Mapping (SLAM), Visual Object Detection, Motion Planning, and Manipulation. Each subsystem presents its own complexities and is an area of ongoing research within the fields of robotics and artificial intelligence. The successful integration of these components demonstrates the potential of mobile manipulators in service robotics applications such as object retrieval in household or office environments.

II. System Architecture and Implementation

A. Hardware and Software Setup

The primary platform used in this project is the **Stretch RE3 Mobile Manipulator Robot**. The following onboard sensors were utilized:

1) **Intel RealSense D435i RGB-D Camera** — mounted on the pan-tilt head, used for environment perception and object tracking.
2) **Intel RealSense D405 Depth Camera** — embedded within the gripper, used for fine-grained visual servoing.
3) **Slamtec RPLIDAR A1 360° 2D LiDAR** — located on the mobile base, used for SLAM and navigation.

The software development was carried out using **ROS 2** (Humble) and **Python**. The following packages and frameworks were employed:

1) **Mapping:** RTAB-Map (Real-Time Appearance-Based Mapping)
2) **Navigation:** ROS 2 Navigation Stack (Nav2 Planner)
3) **Object Detection:** Modified YOLOv5-L model
4) **Manipulation:** Visual Servoing implemented using the Stretch Python API

B. System Workflow

The complete system workflow is orchestrated through a multiphase automation pipeline that integrates Stretch's core APIs with custom modules.

Phase 1 – Initialization and Perception: The robot initializes its ROS 2 drivers and launches the object detection pipeline using the RealSense D435i camera mounted on the head. YOLOv5-L, adapted for tennis ball detection, processes the incoming image stream. The robot then rotates 360° to scan and detect the object in its surroundings.

Phase 2 – Object Tracking and Approach: Upon detection, the robot aligns itself with the object and navigates toward it. Custom control scripts handle primitive actions such as base translation, head pan-tilt adjustments, and arm positioning.

Phase 3 – Visual Servoing and Grasping: Once the ball is in close proximity, the system transitions to visual servoing. Using feedback from both the head camera (D435i) and the gripper camera (D405), precise alignment is achieved. The grasp is executed using Stretch's servoing logic, with customized joint control.

Phase 4 – Post-Grasp Retraction: After a successful grasp, the manipulator executes a predefined sequence to retract the arm to a resting pose. This includes coordinated control of joints such as `joint_lift`, `wrist_yaw`, and `wrist_pitch`, managed through scripts developed entirely by our team.

This modular pipeline ensures robust execution by combining Stretch's base capabilities with custom-built intelligence, resulting in a seamless autonomous object retrieval system.

III. Description of the Demonstration

This project presents a semi-autonomous robotic system built on the Stretch mobile manipulator. It begins with system calibration and progresses to autonomous navigation using RTAB-Map for visual SLAM and AMCL for probabilistic localization. The robot constructs a 2D occupancy grid map from RealSense D435i point clouds and odometry and navigates using the Nav2 stack. This navigation part and the initial mapping of the environment before it are where user interference is required. After the navigation part, the rest of the implementation runs autonomously. For object detection and tracking, the system employs YOLOv5-L for real-time ball detection, initiating a 360° scan followed by precise base and head orientation. The 3D coordinates of the detected ball are then passed to a custom manipulation planner based on the stretch visual servoing algorithm to calculate the trajectory of the arm. Finally, fine alignment and grasping are achieved using the visual servoing controller that adapts to live perception feedback for precise closed-loop manipulation.

A. Mapping and Navigation

To enable autonomous navigation for our Stretch robot, we began by mapping the environment using the stretch_rtabmap package. This utilizes RTAB-Map (Real-Time Appearance-Based Mapping), a graph-based Visual SLAM algorithm that creates a 3D representation of the environment using RGB-D data from the Intel RealSense D435i camera. RTAB-Map leverages rich visual features, making it ideal for cluttered or visually complex indoor spaces, making it suitable for our application. The 2D lidar may miss critical vertical obstacles, which is why we use it for navigation and not mapping. The RTAB-Map algorithm constructs a pose graph where each node represents a keyframe and is connected through visual similarities (loop closures), enabling robust, drift-resistant mapping over time. The robot was manually teleoperated using the joystick/keyboard to explore its surroundings, ensuring smooth movement and loop closures for better map optimization. Once the environment was sufficiently covered, the map was automatically saved to a default directory.

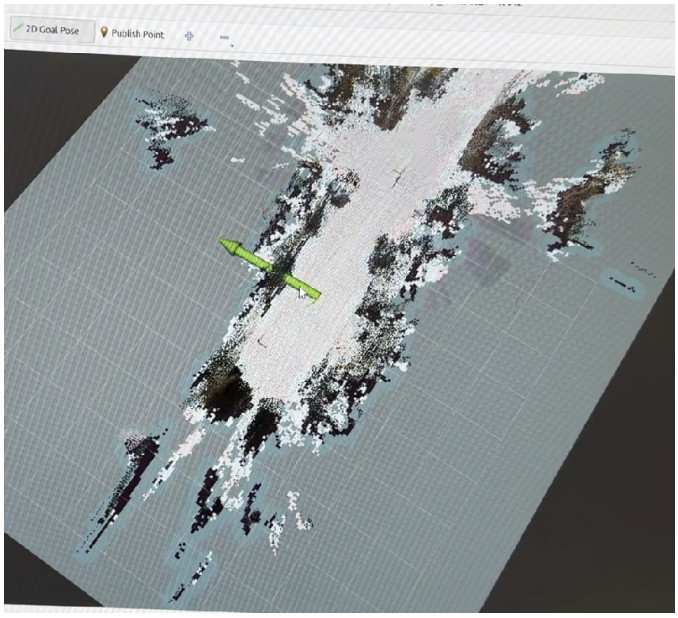

Fig. 1. Providing the Goal Pose to the robot for navigation in Rviz.

After mapping, we launched the autonomous navigation node, which reused RTAB-Map in localization mode, and initialized the Nav2 stack using the stretch_nav2 package. This loads an occupancy grid map in Rviz. To perform localization and path planning, we use the ROS 2 Navigation Stack (Nav2), which includes components such as AMCL (Adaptive Monte Carlo Localization), NavFn for global planning, and DWBLocalPlanner for local trajectory generation. AMCL is a probabilistic localization algorithm that uses a particle filter to estimate the robot pose on a known map. It continuously updates its belief based on laser scan data from the 2D LIDAR on the robot and wheel odometry, converging to the most likely position. For planning, NavFn (based on Dijkstra's algorithm) calculates a globally optimal path, while DWB (Dynamic Window Approach) selects feasible velocity commands to follow the path while avoiding obstacles.

Before navigation begins, we provide an initial pose estimate to the robot using the '2D Pose Estimate' option in Rviz. Although the AMCL can recover from poor initial

estimates using its particle filter approach, its convergence is heavily dependent on the initial distribution of particles. Without a rough initial pose, AMCL would have to rely on global localization, which is computationally expensive and less reliable. By specifying a rough starting location, we bootstrap AMCL with an informed guess, enabling rapid and accurate localization against the visual map generated earlier.

The next step is to provide a goal pose (Fig. 1) to the robot so that it safely navigates to a place from where it can scan its surroundings to look for the target object. a tennis ball. Throughout the navigation process, the robot used sensor data from its RGB-D camera and lidar to continuously update its position and plan safe paths. The local and global costmaps were configured to include layers for obstacles, inflation zones, and voxel representations, enhancing obstacle avoidance capabilities. Additionally, the robot's path is smoothed using a simple path smoother, and recovery behaviors like spinning or backing up are triggered when the planner fails. This robust integration of RTAB-Map with Nav2 made it possible for Stretch to operate autonomously in complex indoor spaces, combining appearance-based loop closure with probabilistic localization and dynamic path planning. This setup balances spatial awareness with responsiveness, ensuring safe and intelligent navigation.

B. Object Detection and Tracking

The object detection and tracking phase begins once the robot has successfully reached the navigation goal set in Rviz. This phase is powered by the stretch_deep_perception package, which launches the RGB-D camera and an object detection node that processes incoming image data to identify and localize objects in the environment. For object detection, the YOLOv5-L model is deployed. Using visual information, the detection node continuously publishes marker data corresponding to detected objects, enabling downstream modules to interact with the scene. This package also provides a preconfigured Rviz setup to visualize detected objects, helping the user confirm detections in real time.

To localize the object relative to the robot, the system relies on the marker-based detection from the previously mentioned perception node. The detected object's 3D pose—specifically its x, y, and z coordinates—is continuously updated through a dedicated topic, and the robot subscribes to this stream for tracking purposes. The robot first rotates in place slowly to ensure that the camera scans the surrounding environment. This rotational scan is performed at a fixed angular speed, completing up to a 360-degree sweep if necessary. If the object is detected before completing a full scan, the mobile base stops. If the base rotated a bit more than required, we have ensured in our code that it corrects itself to face towards the target ball.

Upon successful detection of the marker (Fig. 2), the robot initiates a series of fine alignment procedures. First, it adjusts its head pan and tilt joints to center the object within its field of view along the horizontal (x) and vertical (y) axes. It then uses depth information (z-axis) to approach the object

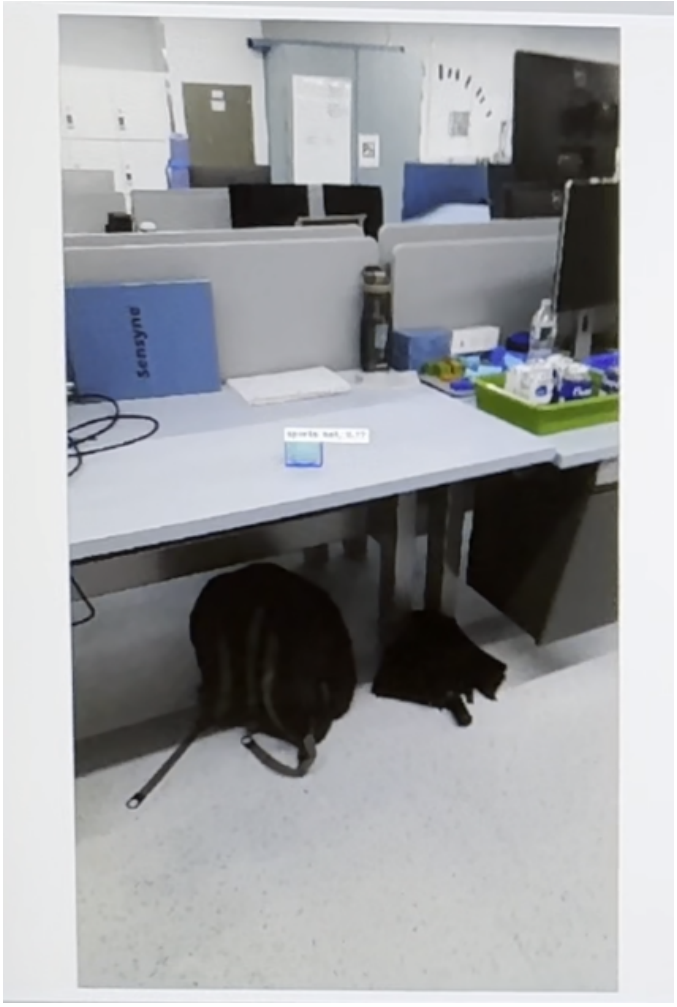

Fig. 2. Target object is detected using YOLO and the robot tries to keep the object in the centre of the frame by positioning the head camera.

to a predefined distance threshold. The control logic includes corrective movements of the robot base and head, allowing precise alignment with the object while maintaining stable perception. The feedback control loop ensures real-time pose correction by constantly checking the offset between the detected object's position and the desired reference values. We have set the predefined distance threshold as 75 cm. This is the distance of the mobile base from the target tennis ball along the Z-axis. This distance is to ensure that the robot does not collide with the table/elevated surface on which the ball is kept when it rotates at 90 degrees in the next part.

After aligning itself to the object, the robot performs a final reorientation to optimize its arm's reachability. This involves rotating the base by 90 degrees to the left and panning the head 90 degrees to the right, thereby positioning the end-effector to face the object head-on. This orientation ensures that the arm is directly aligned with the object and ready to perform manipulation tasks such as gripping. This transition marks the completion of the detection and tracking phase and seamlessly prepares the robot for the subsequent manip-

ulation and grasping operations. The architecture supporting this phase was developed to efficiently integrate perception, motion control, and spatial reasoning through ROS 2 nodes and communication interfaces.

C. Grasping and Manipulation

The grasping and manipulation phase of the project is initiated once the robot has visually localized and aligned itself with the target object using the head camera and YOLO-based object detection. This phase leverages the visual servoing algorithm provided by Hello Robot, which integrates image-based feedback from the D405 gripper camera to continuously refine the pose of the robot's arm and end effector in real time. The robot receives object position and fingertip data over ZeroMQ from a remote image processing module and then uses this information to calculate velocity commands for its joints. This ensures real-time responsiveness to changes in the object's position, even during approach.

The visual servoing behavior utilizes a modular behavior structure based on state machines. It switches between various states like 'reach,' 'retract,' 'celebrate,' and 'disappointed' based on the success of the grasping attempt. A closed-loop visual feedback controller is used for visual servoing, which compares the 3D position of the object (obtained via YOLO and depth sensing) against the estimated position of the gripper fingertips (inferred through ArUco markers on the gripper). Using control strategies such as normalized velocity control and a set of gain-scaled joint commands, the robot aligns its wrist, arm, lift, and mobile base to center its gripper on the object. Once the object is within grasping range and the error in positioning drops below a threshold, the robot closes its gripper using preset force thresholds to detect successful object acquisition.

After the robot successfully grasps the object, it enters the 'retract' state where it performs a sequence of joint movements to safely lift and stow the object. Although the visual servoing handles the grasp initiation, our team developed custom scripts to perform the final repositioning of the robot's joints for manipulation. These include repositioning the wrist yaw and pitch, lowering the lift, and retracting the arm using the Stretch Python API. Each joint is handled separately in modular Python scripts, allowing for fine-tuned control and easy debugging. This modularity also provides flexibility for future adaptation to different object sizes or stowing configurations.

Fig. 3. During Visual servoing, the robot uses the gripper camera to detect and grasp the ball by minimising the position error between the gripper and the ball.

Fig. 4. After successfully grasping the ball, the robot will now retract itself to a custom position.

IV. Challenges faced in Project Execution

During the course of our project development with the Stretch robot, we encountered several technical challenges that required thoughtful analysis and iterative experimentation to resolve.

- **YOLOv5 Model Selection:** One of the key issues arose during the object detection and tracking phase, where we initially experimented with different YOLO model variants. Although the YOLOv5-small model offered computational efficiency, it consistently misclassified the target object—a toy ball—as a basketball. This misclassification reduced the accuracy of downstream tasks such as alignment and grasping. After empirical evaluation, we selected YOLOv5-large, which provided a more accurate detection result at the cost of moderately increased computational load. We consciously avoided YOLOv8 due to its heavier architecture, as it could introduce latency during real-time detection and servoing tasks on our platform. The selected YOLOv5-large model achieved a practical balance between precision and runtime perfor-

mance.

- **Object Tracking within Camera Field of View:** Another significant challenge involved the camera's field of view during the visual servoing process. Specifically, we observed that as the robot approached the object, the target would frequently move out of the camera frame, disrupting tracking and alignment. This was primarily due to the fixed mounting angle of the Intel D405 depth camera located on the robot's wrist. To address this, we implemented a continuous adjustment of the robot's head tilt joint during motion. By dynamically tilting the camera downward as the robot moved forward, we ensured the object remained centered in the field of view. This solution significantly improved the robustness of the visual tracking system and allowed the grasping algorithm to receive consistent 3D localization data of the object throughout the approach phase.
- **Limitations in Visual Servoing Execution:** We utilized the official visual servoing framework provided by Hello Robot to perform precise object alignment and grasping. However, during testing, we observed that the original implementation halted the robot's motion upon reaching the target object without executing a complete grasp. Although the manipulator arm correctly aligned with the tennis ball, the gripper remained open, resulting in an unsuccessful pick action.

  To resolve this, we modified the servoing pipeline to explicitly issue grasp commands upon reaching the desired pose. Furthermore, we extended the code to include a post-grasp action that commands the robot arm to move to a predefined home configuration. This involved sequencing multiple joint-level commands, including adjustments to `wrist_yaw`, `joint_lift`, and `arm` extension values. These enhancements ensured not only successful object acquisition but also a smooth transition to a resting pose, thus improving the overall reliability and autonomy of the manipulation subsystem.
- **FUNMAP Navigation:** FUNMAP is a mapping and planning framework designed specifically for the Stretch robot, offering integrated navigation and manipulation using a heightmap-based environment representation. We initially chose it for its efficiency and tight integration with the robot's kinematics, allowing fast local planning from onboard RGB-D data.

  While running the navigation pipeline using FUNMAP on the Stretch robot, we encountered a critical issue where the robot failed to reach its specified goal pose. Upon closer inspection of the 3D map generated by FUNMAP, we identified that certain regions of the environment were erroneously marked as occupied, even though no physical obstacles were present. We traced this anomaly to the placement and motion of the head-mounted depth camera on the robot's mast. Specifically, during the panoramic head scan phase (as implemented in the mapping part of FUNMAP), the camera likely captured transient parts of the robot's own structure—particularly the mast—as static environmental features. Since the Stretch robot performs mapping while the base and mast are potentially in motion, the camera may have logged these dynamic parts as persistent obstacles. These falsely mapped points, once committed to the Max Height Image (MHI) representation, incorrectly populated the cost maps used in the navigation components of FUNMAP, thereby leading to planning failures due to perceived occlusions.

  This issue stems from the fact that FUNMAP relies heavily on MHI representations built from depth scans, where every pixel stores the highest observed point in a given (x, y) planar location. Without robust self-filtering (i.e., removing robot body parts from point clouds), the mast and possibly even the arm or gripper can be misinterpreted as part of the environment, especially during overlapping scans. Although this code includes post-processing steps to prevent these, these may not be sufficient if the dynamic motion introduces new, unfiltered artifacts before or after these calls. To overcome these limitations, we transitioned to RTAB-Map for SLAM. RTAB-Map provided real-time loop closure detection, accurate scan integration, and effective self-filtering, resulting in a clean and consistent environment map without phantom obstacles. Its robust 3D graph optimization and keyframe-based approach handled the Stretch's camera motion more gracefully, allowing the navigation stack to successfully compute viable and obstacle-free paths to the goal.

V. Future Scope

There are many limitations we faced while working on this project, and we can overcome them systematically if we have another few months of time. We initially aimed to use CLIP for semantic-level object grounding, which would have enabled the robot to localize objects using natural language descriptions. However, CLIP's inference requires substantial GPU resources and efficient integration with a ROS2 system. Our hardware lacked a suitable onboard GPU, and offloading to an external machine with an RTX 4070 or better GPU required networking, image streaming, and model serving infrastructure, which proved time-prohibitive within our 10–13 day access window. Instead, we selected YOLOv5-Large for a balance between detection accuracy and real-time performance.

The current system is calibrated specifically for grasping a tennis ball, with fixed joint tolerances, grasp thresholds, and servoing parameters optimized for its geometry. Testing on additional household objects like cups, bottles, or boxes was deferred due to time constraints. These objects differ in shape, reflectivity, and grasp affordances, all of which affect YOLO detection reliability and grasp planning. During integration, we adjusted thresholds such as $\texttt{target\_error} < 2\,\text{cm}$ for grasp initiation and tuned servoing velocities assuming a spherical object with low center offset. Introducing new objects (cups, bottles) would affect grasp center estimation, orientation assumptions, and depth feedback. To generalize,

we propose using a meta-parameter tuning framework (e.g., via YAML config per object class) and introducing a classification submodule that selects the appropriate servoing parameters at runtime.

Moreover, the system's performance is sensitive to environmental factors. Inconsistent lighting conditions—particularly high contrast or brightness—can reduce object detection reliability, especially when using the RGBD camera. This would require some image preprocessing to resolve. The visual servoing control also assumes a relatively flat and known surface height; changes in the table height or object elevation can affect 3D localization accuracy and the success of the grasping behavior. To correct for this, we can incorporate the IMU sensor on the mobile base to correct the pose estimation.

ACKNOWLEDGMENT

We thank Prof. Zhi Tan for providing us an opportunity to work on the Stretch Robot. The resources and guidance he provided helped us familiarize ourselves quickly with the Stretch ecosystem.

REFERENCES

[1] https://github.com/hello-robot/stretch_ros2
[2] https://docs.hello-robot.com/0.3/
[3] https://forum.hello-robot.com/c/knowledge-base/13
[4] https://forum.hello-robot.com/t/error-during-first-time-setup/446