Robotic Science and Systems - Project Final Report

Name	Email ID
Advaith Kandiraju	kandiraju.v@northeastern.edu
Poorna Chandra Vemula	vemula.p@northeastern.edu
Sanjay Prabhakar	prabhakar.sa@northeastern.edu
Samavedam Manikhanta Praphul	samavedam.m@northeastern.edu

Introduction:

Robots are meant to reduce human effort by aiding them in some daily chores or mundane tasks. It is quite often that someone could misplace or forget the pinpoint location of a particular equipment/tool/object in a given closed space or at home. Here is where iSPy comes into the picture. This robot essentially assists humans to help them find things that are difficult to find or inaccessible to reach.

To address this problem, the robot should necessarily have at least one perception module like a camera or LiDAR. It must be mobile to freely work in the workspace. It must have a manipulator to help the Master fetch equipment. It should not consist of a lot of machinery and should not occupy much space. And it must be technically well-equipped to work autonomously. Keeping these things in mind, we landed on Locobot WX250s which is a one-stop solution that precisely aspires to tackle the problem.

System/Implementation details:

The overall approach we could come up with can be best represented by the below skeleton.

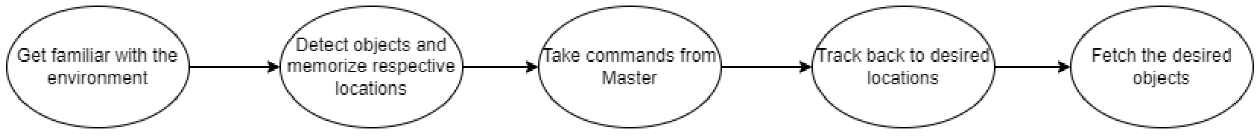

Get familiar with the environment
Detect objects and memorize respective locations
Take commands from Master
Track back to desired locations
Fetch the desired objects

Figure 1: Overall pipeline

Perception: To become acquainted with the environment, our robot iSPy is initially designed to move freely and become comfortable with it. Yolov5 is used for object detection, and Intel's RealSense Depth Camera D435 is used to determine the depth of the identified objects, which is then stored in the database. This database consists of all the essential information like visual features, depth measurements, robot poses, and other metadata for the reconstruction of the environment that helps the robot for desirable navigation. For localization and mapping, we use Locobot's built-in Simultaneous Localization and Mapping (SLAM) function, which internally employs the RTAB-Map. Real-Time Appearance-Based Mapping (RTAB-Map) is an RGB-D and Lidar Graph-Based SLAM technique that uses an incremental appearance-based loop closure detector. The RTAB-map consists of a 2D occupancy grid map for basic navigation and obstacle avoidance, a 3D point cloud for a detailed representation of the environment, and a graph of poses that connects various views of the robot within the environment for precise localization and relocalization.

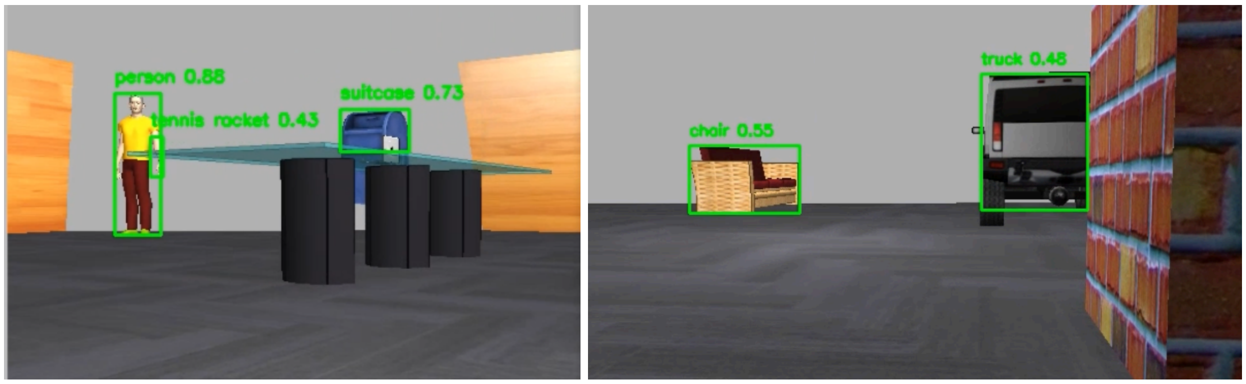
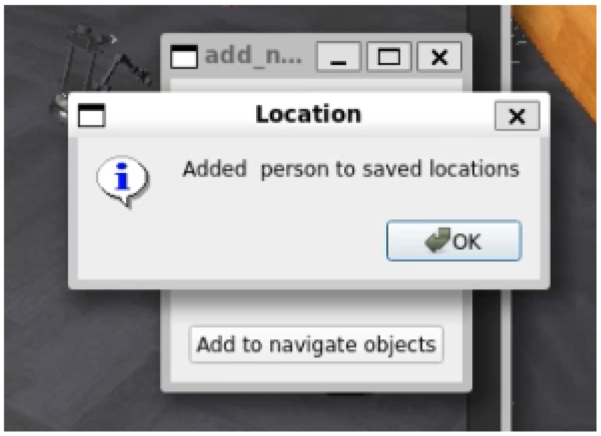


Figure 2: Object Detection via YOLO in Gazebo

Human Robot interaction: As part of the human-robot interactions, we created a GUI drop-down menu using pyQT, allowing the user to navigate to a specific object while also adding new objects to the database as soon as the robot encounters a previously unknown object. The interface also lets users upload music files into the system to which the robot may tap its legs(dance) while it is idle, keeping the user engaged. As part of the user experience, the robot can also collect objects thrown from a distance by keeping track of the tossed elements.



Added person to saved locations

Figure 3: Addition of new object locations

Planning and Navigation: Once we have this mapping information, we use it to determine the desired object position depending on the object picked by the user to navigate. We navigate the robot to the required object location by utilizing the Locobot's move_base package (a built-in navigation package) from the Locobot's navigation stack. During the navigation process, the robot's camera continuously feeds things into a separate window, allowing the operator to add them to the list of recognized objects and their locations. This feature enables the user to create new items that can be evolved via software updates. The reason for using Yolov5 for object detection is that it can offer inference on the identified item as well as the detected bounding boxes in real-time, as opposed to Masked R-CNNs, which have a longer processing time. When the robot's camera catches sight of the target object and the robot comes to a standstill, we consider the robot to have arrived at its destination. Now, the user can choose whether to approach the user or direct the robot to another object.

User Experience: We've also introduced a feature that allows the robot to dance to music picked by the user based on the tempo and beats. The user-selected song is analyzed using Librosa to determine the rhythm and tempo. Once we get the beats and tempo of the song, we assign random poses to the robot at each major beat and bucket the tempo in units of 25, which reduces the robot's jerk motions. In this method, the robot dances to the tune of the song chosen by the user.

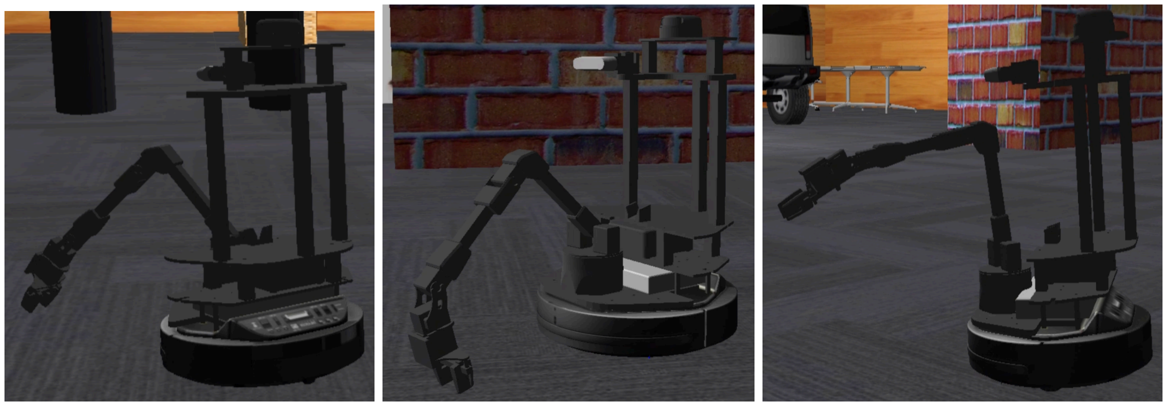


Figure 4: iSPy dance movements for audio input

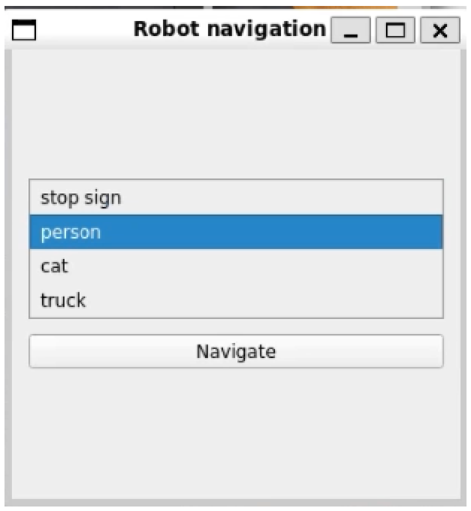


Figure 5: Drop-down Menu for accessing saved locations

Description of the demonstration/experiment:

We first intended to collaborate with the robot. We had to use the Gazebo simulation environment to demonstrate our approach because the robot was not available. The Gazebo simulation platform allows for the rapid iteration of different physical designs in realistic worlds/environments with high-fidelity sensor streams.

We designed a realistic setting with several obstacles and items that are commonly recognized by the YOLO object detection algorithm. Furthermore, we manually manipulated the robot in the environment to move about and map the world with objects and obstacles using the 'rtabmap_ros' software, which is optimized for an Interbotix robot.

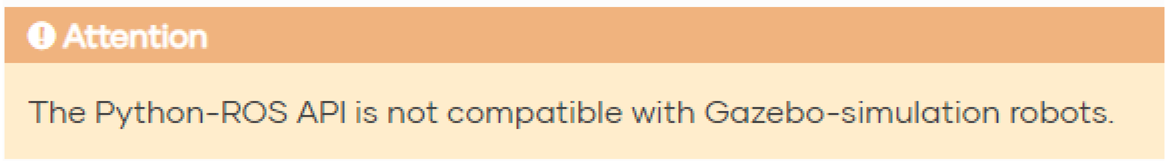
After mapping the world, we now have an occupancy grid that shows where the objects and obstacles are placed. The user may now choose a specific thing for the robot to find, and the robot will go to that object in the environment. We showed that when the robot detects a new thing, the user may add it as well.

Challenges faced:

The first challenge we encountered was with the package restrictions of Interbotix Locobot. Interbotix Locobot, like other robotics platforms, has certain packages and libraries of feedback control systems, sensor drivers, and motor controls that are tailored specifically for the physical robot, leveraging its unique hardware capabilities. These packages were quite challenging to replicate accurately in gazebo or rviz due to their reliance on real-world physics, hardware-specific interactions, and sensory feedback.

The next hurdle we faced was setting up the ROS and Gazebo simulation environment for Interbotix Locobot using Python. We looked at VMWare, Oracle VirtualBox, and WSL for putting up a Linux base to install ROS Noetic and Gazebo. We couldn't run ROS on Mac M1, thus we were limited to only two machines for the duration of our project, slowing our progress.

The next challenge was the limited objects that are available that are recognizable by the YOLO model such as STOP Sign, Man, etc., so we had to hunt for additional YOLO recognizable simulation model files, which were usually licensed ones and free available options did not have all the required files such as mesh files along with sdf/URDF (Spatial Data File/ Universal Robot Description File), either of these files were missing to properly load the free models into the simulator. In the process of deciding to use only the restricted compatible objects, we discovered that Locobot did not support the Python-ROS API in the Gazebo simulation environment, as advertised on their official website.


Attention

The Python-ROS API is not compatible with Gazebo-simulation robots.

Figure 6: Compatibility Warning

This was a major roadblock for us, as we had figured out a way to have the Locobot in the Gazebo environment, as we had created the complete environment in the Gazebo simulation.

Another difficulty was that, in order to move forward with our project, we needed to understand the novel framework approach that the ROS uses—publisher, receiver architecture. We had originally considered chaining the Python scripts together, but we had to modify our code to make use of the ROS publisher and receiver architecture after finding out about it. Learning about this design and how it gets around some of the problems with our method was a really worthwhile experience.

It was also challenging for us to achieve manipulation in the gazebo simulation as the Robot state publisher, Joint state publisher, and Moveit! Packages were tailored for physical applications, and it was difficult to set up the grasping pipeline and simulation of contacts and forces because sensor simulation and contact force handling required manipulating ROS URDF/Xacro files within Moveit Launch files, which frequently resulted in roscore issues that affected the functionality of other publishers and packages.

Limitations and future scope:

Among the objects positioned in the surroundings, the Yolov5 model that was pre-trained achieved an accuracy of 57% in object detection. Occasionally, the model was misidentifying items, mistaking a bench for an airplane, a cat for a dog, and so forth. To enhance the model's performance in our environment, future work may involve using more advanced models, such Yolov8 or a higher version, and fine-tuning them for the items in the environment in various ways, as done in assignment 4.

The environment can only contain a single sort of object. This is another restriction. That is to say, our robot is now unable to discriminate between two things of the same type in the environment and move to the matching object that has been selected. The ability to distinguish between the two things and maintain them as distinct entities is what the future scope entails.

Another limitation was the robot faced difficulty moving through the textured floor. We had overcome this in our project by removing the textured floor space from the environment. For future work, we need to work to overcome this issue on the textured floor. This is the limitation that we have observed.

One of the limitations is that the current implementation doesn't have an interface where the users can interact or give instructions directly through speech. In the future, we would like to implement a capability where when a user gives an instruction, we convert the speech signal to text using Automatic speech recognition(ASR) like Whisper and then process the text using a Large language model to breakdown into predefined instructions of the robot to extend the interfacing with speech input to the robot.

At the moment, our robot can only dance in response to an audio file input; it cannot dance to live music. As part of our method, we used Librosa to process the audio file for beats and tempo. In order for the robot to groove to live music, we will need to filter the music for tempo and beats in future work. This will likely include identifying the song online using the audio that the robot has recorded (a microphone will be needed).

Simulation Folder Videos link:
https://drive.google.com/drive/folders/1F7-lQBuZQ2cW6pcZ-Iaw3ZyKH9VQvSAG?usp=sharing