Final Report

EECE.5800 Robotics, Automation and Machine Intelligence

Date submitted: 12/19/20201

Date due: 12/19/2021

Project Team Members:

Alden Chad Daniels
Ernest Richard Pellegrino
Pablo Ruiz

Purpose:

Guide on how to solve the coffee delivery problem written to be understood by people coming out of EECE.5560.

1. Problem Statement

This report outlines the process of completing a task autonomously using a robot equipped with a variety of sensors and actuators. The task at hand consists of picking up and delivering a cup of coffee from one indoor location to another. Such a task seems simple but can become very complex when being performed in a dynamic environment such as a college building. To efficiently complete this task, it was subdivided into the following sub-tasks which should all be relatively simple to complete on their own:

Sub-Goals:

- Commence with a pre-made map of the environment the robot will be expected to navigate (necessary to be able to set waypoints).
- Have robot drive to a predefined navigation goal defined as a coordinate with an orientation relative to the map frame. Avoid dynamic obstacles (i.e. students) and update the map as necessary if the environment has changed.
- Use environmental features (could be AR tags) to precisely localize the robot, relative to its goal, once it is close to its destination.
- Employ the robot's depth camera to identify the cluster representing the object to be picked up.
- Use the known location of the object to position the gripper in an advantageous position.
- Close the gripper, pick up the object, and bring the arm to rest close to the center of gravity of the robot to avoid hindering its mobility.
- Have robot drive to a second predefined navigation goal.
- Use environmental features (again probably AR tags) to locate the precise location where to place the object.
- Position the gripper at the correct location with the object slightly above the surface where it will be placed.
- Open the gripper and back away the arm avoiding a collision with the object.

2. Background Information

Resources:

- ROS Wiki:
    - http://wiki.ros.org/
- Interbotix Locobot Documentation:
    - https://trossenrobotics.com/docs/interbotix_xslocobots/index.html
- Interbotix Youtube Channel:
    - https://www.youtube.com/@InterbotixTrossenRobotics
- Interbotix Support Contact:
    - trsupport@trossenrobotics.com
- Interbotix Github:
    - https://github.com/Interbotix/interbotix_ros_rovers

Hardware:

- Lidar (RPLIDAR A2M8 360° Laser Range Scanner)
    - The robot uses a 360-degree 2D Lidar centrally fixed at its highest point. When activated this device spins in a 2D plane while emitting a laser in all directions and scans the reflected laser to acquire 2D point cloud data which can be used for mapping and localization purposes. It spins at 10Hz, has a 0.45° resolution and a 0.2m-12m range.
- Camera (Intel Realsense Depth Camera D435)
    - The D435 combines a stereo camera with an infrared point projector which allows it to measure depth. It has a range of 10m and a depth resolution of 1280x720. Its primary purpose is to detect objects in close proximity to the robot usually for manipulation tasks. It can also be used in some mapping and localization packages (such as RTAB-Map), but it’s not recommended.

- Robotic Arm (WidowX 250s)
  - The robotic arm has 6 degrees of freedom, a reach of 650mm and can carry a 250g payload. It is currently equipped with a set of custom grippers designed for grasping conical objects, but it can also be equipped with shorter flat fingers. It is used by the robot to interact with its environment and perform manipulation tasks.
- Base (iRobot Create3 Base)
  - The Locobot is designed to work with the Kobuki and Create3 bases. The configuration used for this demonstration uses the Create3 robotic base.

ROS Support:

Currently the robot is controlled by a mix of ROS and ROS2 packages. The Create3 base is controlled using a ROS2 package while everything else is controlled using ROS1 packages. You can have a ROS1 release and a ROS2 release installed on your computer at the same time however you should not source both of their setup.bash files in the same shell session. When the setup.bash file for the Interbotix software is sourced it will also automatically source the setup.bash for the ROS release that you specified when you installed their software so no further sourcing should be necessary (as long as this line is present in your .bashrc as outlined above). In order to get the Create3 ROS2 package to communicate with the rest of the ROS packages there is a ROS to ROS2 "bridge" implemented by Interbotix in the background. You do not need to interact with this as it mostly works. However, you should be aware that the ROS topics relating to the base that you see when you run "rostopic list" are not the real topics for the base, but simply stand-ins to work with the bridge. The real topics for the base are behind the bridge in ROS2 and cannot be visualized.

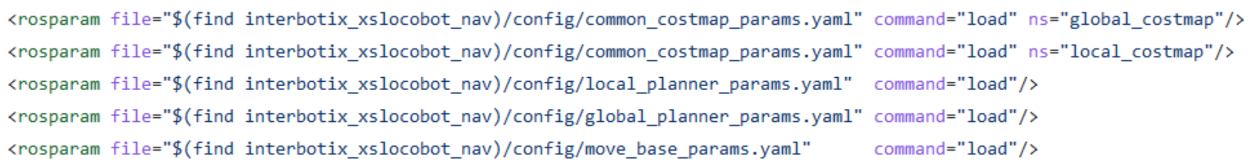
All of the above is subject to change as Interbotix has a plan to port all of their packages to ROS2 in the future.

Launch Files:

The launch files for each package are at the core of the Interbotix software. For the most part all the launch files do is launch standard ROS packages (whose documentation can be found in the ROS wiki) with the right arguments to work with the Locobot hardware and topic namespaces. It is recommended that you read and understand these launch files and default to them instead of the Interbotix documentation as they offer more clarity into what is really going on "under the hood".

YAML Configuration Files:

Most of the packages started in the launch files have their arguments declared and set directly in the launch files. However, there are some packages, like the move_base package which is part of the navigation launch file, which import their arguments from YAML files. The path to these YAML files will be listed in the launch files.



```
<rosparam file="$(find interbotix_xslocobot_nav)/config/common_costmap_params.yaml" command="load" ns="global_costmap"/>
<rosparam file="$(find interbotix_xslocobot_nav)/config/common_costmap_params.yaml" command="load" ns="local_costmap"/>
<rosparam file="$(find interbotix_xslocobot_nav)/config/local_planner_params.yaml"  command="load"/>
<rosparam file="$(find interbotix_xslocobot_nav)/config/global_planner_params.yaml" command="load"/>
<rosparam file="$(find interbotix_xslocobot_nav)/config/move_base_params.yaml"      command="load"/>
```

Figure 1. Parameters for the Move_Base package being loaded from YAML files.

Robot Description URDF:

The 3D description of the robot used for the visualization in RVIZ, for the simulations in Gazebo and for the manipulation in MoveIt is defined by the URDF files inside the "Interbotix_xslocobot_description" package. The URDF is a XML style document which defines a visual representation and collision footprint for each component using 3D mesh files (generally .stl format). Additionally, each component also has a defined center of gravity and moment of inertia which is used for the physics simulations in Gazebo. The interactions between elements can also be defined by using by using "joints" i.e., how the gripper moves relative to the rest of the robotic arm. More details on the specific syntax can be found at http://wiki.ros.org/urdf/XML.

A new gripper design was created for the specific task of picking up a conical object like a 4oz plastic cup and implemented into the robot URDF. This was necessary for MoveIt to be able to visualize and perform manipulation tasks with the new gripper. Even though the position and movement of the new gripper relative to the arm was defined accurately, the moment of inertia and center of gravity were left untouched and as such still correspond to the original Interbotix “fingers”.

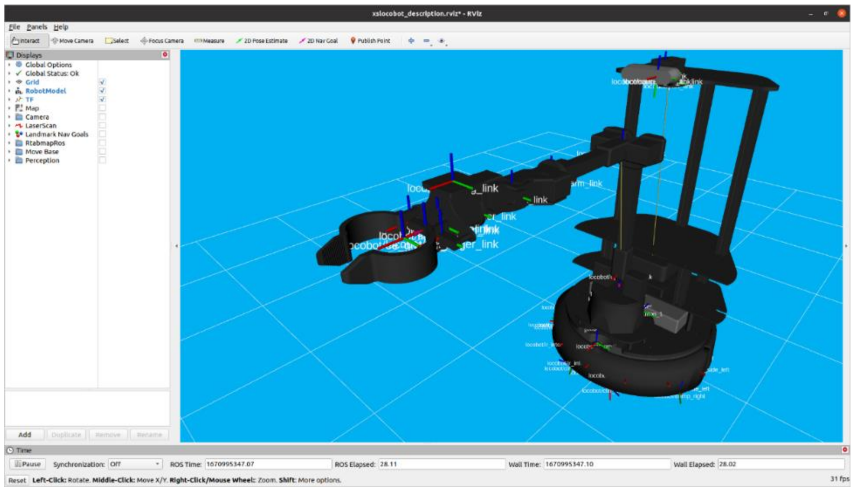


Figure 2. Locobot URDF visualized in RVIZ.

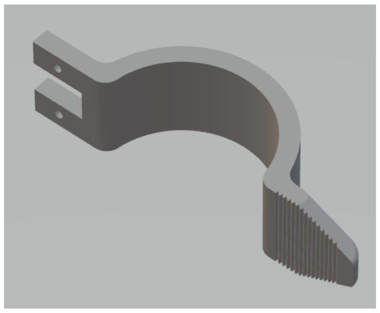


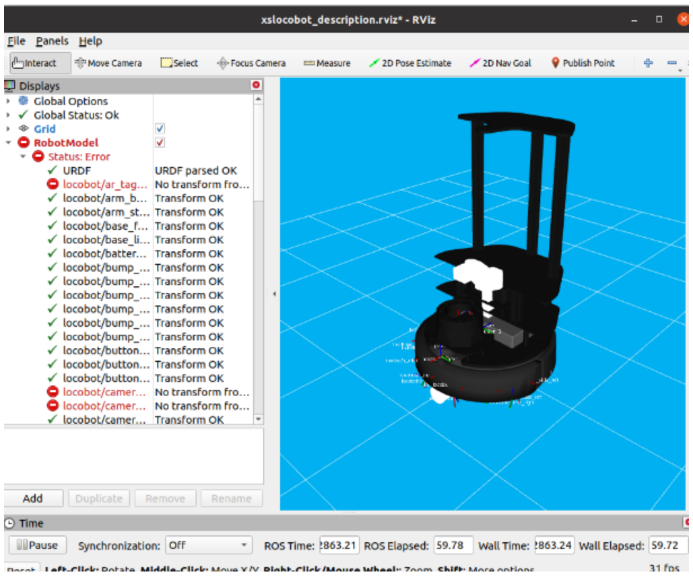
Figure 3. 3D view of the new gripper STL file.

Remote Gazebo Simulation:

The Locobot can be simulated in a robotics simulator called Gazebo which is packaged with the full installation of a ROS release. There are several launch files that can start Gazebo depending on what the goal is during the simulation. For example, there is a launch file exclusively for Gazebo in the “Interbotix_xslocobot_nav” package called “xslocobot_nav_sim.launch” which will start a Gazebo simulation with all of the navigation packages and nodes running for SLAM. Additionally, there is a dedicated “Interbotix_xslocobot_gazebo” package which contains a “xslocobot_gazebo.launch” file which launches only the basic simulation components. More information on these topics can be found at https://trossenrobotics.com/docs/interbotix_xslocobots/ros_packages/gazebo_simulation_configuration.html.

Transform Trees (TF trees):

Many of the functionalities of the robot require the use of a set of transforms that specify how the different components of the robot are spatially oriented relative to one another . These transforms are published on the ‘tf’ and ‘tf_static’ topics (within the Locobot’s namespace). These transforms form a tree which is rooted at a specific transform representing the map's location. Occasionally an issue may be encountered involving a broken link in the tree, and this will cause problems involved with modeling the robot or planning its motion. In Rvis this could appear something like the following:



```
Global Options
Global Status: Ok
Grid
RobotModel
  Status: Error
    URDF                 URDF parsed OK
    locobot/ar_tag...    No transform fro...
    locobot/arm_b...     Transform OK
    locobot/arm_st...    Transform OK
    locobot/base_f...    Transform OK
    locobot/base_li...   Transform OK
    locobot/batter...    Transform OK
    locobot/bump_...     Transform OK
    locobot/bump_...     Transform OK
    locobot/bump_...     Transform OK
    locobot/bump_...     Transform OK
    locobot/bump_...     Transform OK
    locobot/bump_...     Transform OK
    locobot/button...    Transform OK
    locobot/button...    Transform OK
    locobot/button...    Transform OK
    locobot/camer...     No transform fro...
    locobot/camer...     No transform fro...
    locobot/camer...     Transform OK

Synchronization: Off   ROS Time: 2863.21   ROS Elapsed: 59.78   Wall Time: 2863.24   Wall Elapsed: 59.72
31 fps
```

Figure 4. Robot URDF being visualized using RVIZ with a broken Transform (TF) tree.

These breaks can be easily visualized by using a publicly available tool tf2_tool. This can be done by calling the following commands while the robot is running and transforms are being published to the 'tf', and 'tf_static' topics:

- Rosrun tf2_tools view_frames.py
- Evince frames.pdf

This will launch a program that listens to all the transforms which appear on the topics for a few seconds before generating a pdf which displays the transforms and their connections in a graph. A pdf displaying a broken transform tree will appear something like what follows:

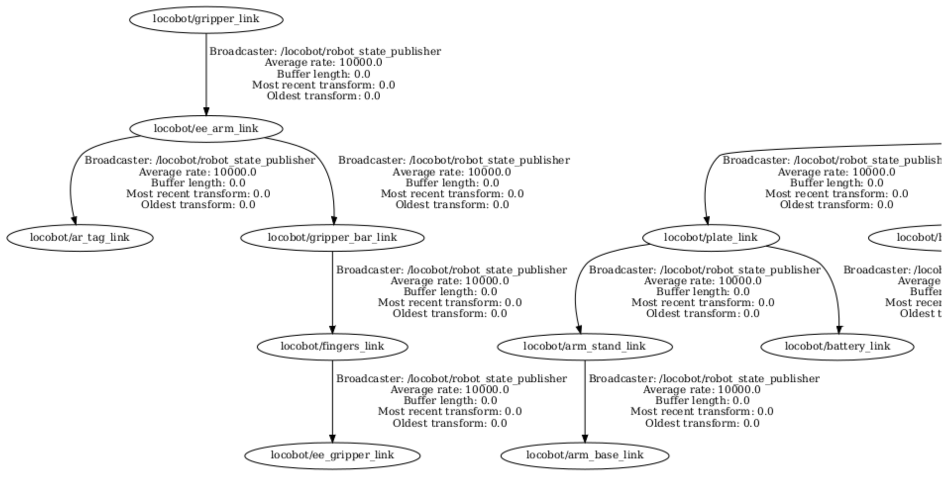


Figure 5. Example of a broken transform tree.

A vulnerable place where such transform tree issues are likely to occur is in the communication between the main locobot computer and the Create3 base, in such an event a likely fix can be found by consulting the Create3 base setup description documentation page found above.

Bashrc:

The bashrc file in linux bash terminal is the file that contains your configuration commands for your terminal. It gets run every time you start a new terminal in order to set up your environment. It is located in your home directory (~/) and it is hidden so to view it you need to use the "*ls -a*" command to show all files in the current directory. Additionally, if you or another program has made changes to the bashrc file you can rerun and update the shell environment by typing "*source ~/.bashrc*" into the terminal.

3. Approach

Mapping & Localization (SLAM):

The navigation package which comes bundled with the locobot uses a depth camera-based package “RtabMap” to perform localization and mapping. Rtabmap takes data from the depth camera together with odometry and lidar data and uses it to build a map which is then hosted on the ‘map’ topic. A transform is published to ‘tf’ which describes the robots position relative to the map and can thus be used for localization. Using RVIZ the map can be displayed, and loop closures can be observed in real time. When it is finished generating the map should appear something like the following:

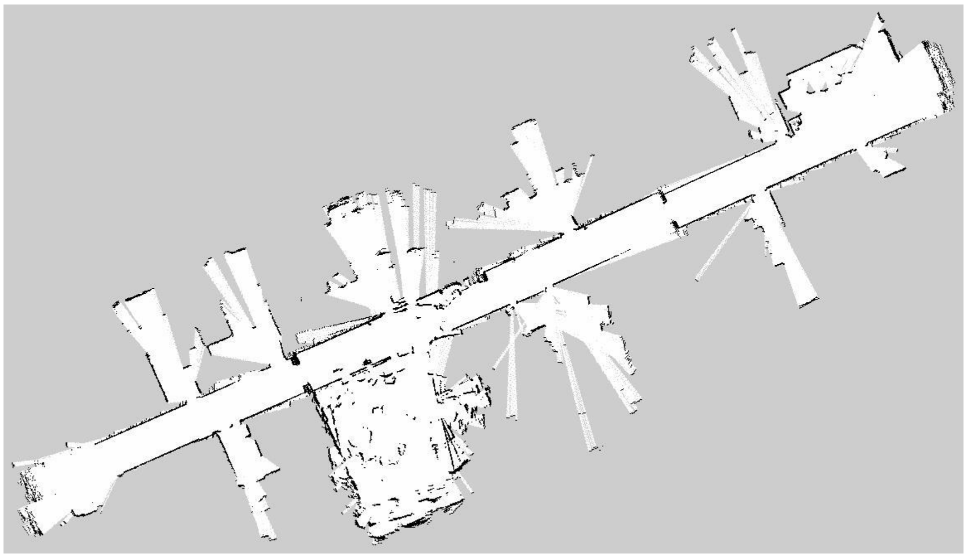


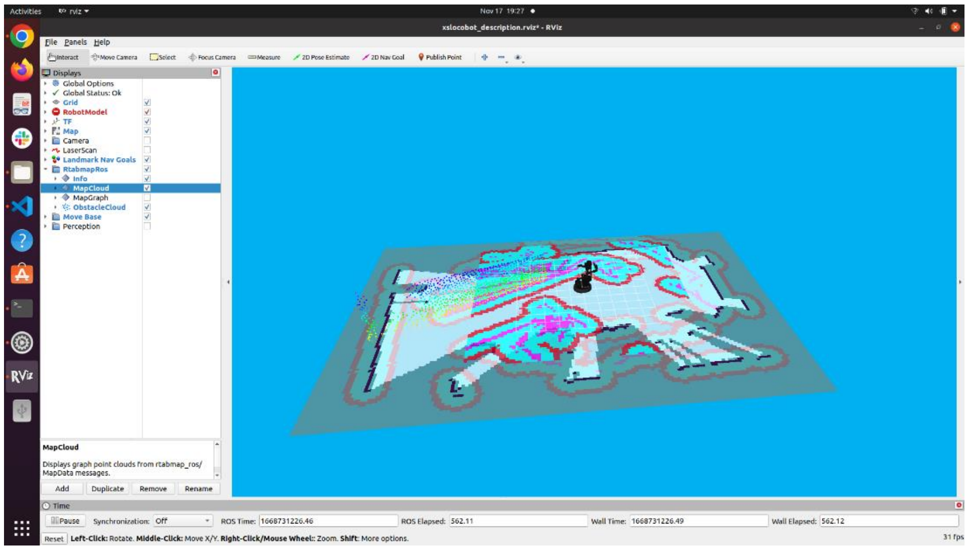
Figure 6. Example of a 2D map of DAN407 and its surroundings.

Further instructions for how to use the navigation stack for both SLAM and general navigation tasks can be found at:

- https://www.trossenrobotics.com/docs/interbotix_xslocobots/ros_packages/navigation_stack_configuration.html#navigation-stack-configuration

Navigation:

The navigation stack uses a number of open source packages together with the above mentioned RTAB-Map to perform navigation tasks. When run together with RVIZ its operations can be visualized as in the following display.



MapCloud

Displays graph point clouds from rtabmap_ros/MapData messages.

Synchronization: Off ROS Time: 1668731226.46 ROS Elapsed: 562.11 Wall Time: 1668731226.49 Wall Elapsed: 562.12

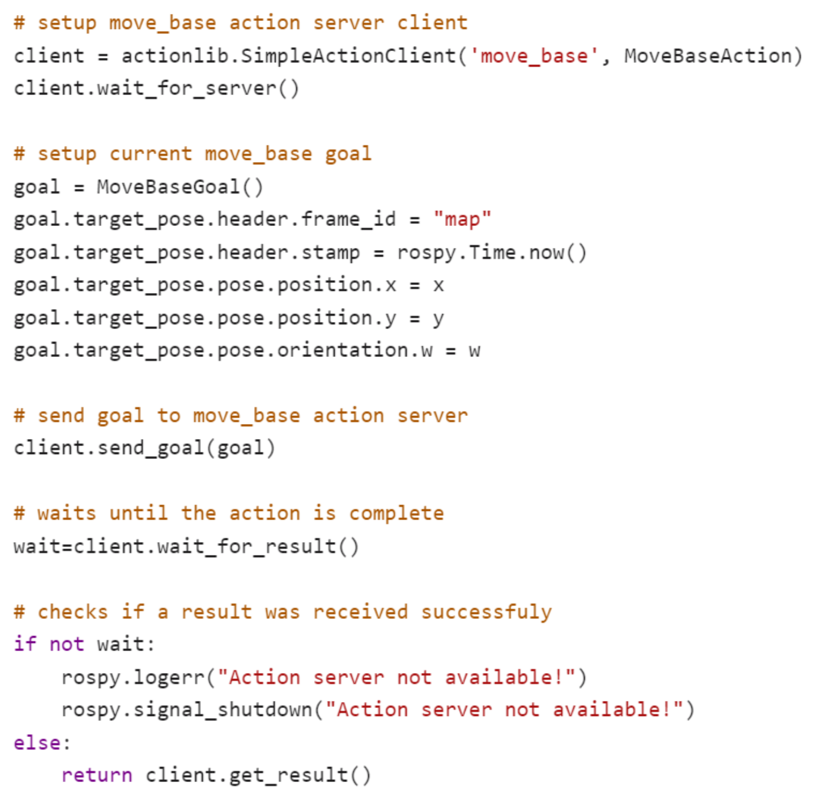
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

Figure 7. Cost map RVIZ view inside DAN407.

With RVIZ, navigation goals can be set at any point on the map and the Move_Base package is then used to plan a path to the goal and publish pose messages to guide the robot to the desired location. The Move_Base package uses both a local and global 'cost map' to represent obstacles in the environment and associate costs with them in order to optimize the selected path to the goal. Using RVIZ these can be visualized as point clouds around the robot. Parameters can be adjusted in the respective YAML files linked in the navigation launch file to alter the size of the cost map point clouds and other sensitivity settings.

Additionally, the Move_Base package uses an implementation of a simple action server to receive goal messages from other nodes. This allows you to send goals programmatically through a C++/python ROS node. To accomplish this first a simple action client must be declared as shown in Figure 8. Next a goal message must be declared and sent to the simple action server through the simple action client. The script sending the goal should pause until

Move_Base either completes or fails the task and returns a result message letting the client know if the action was successful.



```python
# setup move_base action server client
client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
client.wait_for_server()

# setup current move_base goal
goal = MoveBaseGoal()
goal.target_pose.header.frame_id = "map"
goal.target_pose.header.stamp = rospy.Time.now()
goal.target_pose.pose.position.x = x
goal.target_pose.pose.position.y = y
goal.target_pose.pose.orientation.w = w

# send goal to move_base action server
client.send_goal(goal)

# waits until the action is complete
wait=client.wait_for_result()

# checks if a result was received successfuly
if not wait:
    rospy.logerr("Action server not available!")
    rospy.signal_shutdown("Action server not available!")
else:
    return client.get_result()
```

Figure 8. Example of a simple action client and goal message in use.

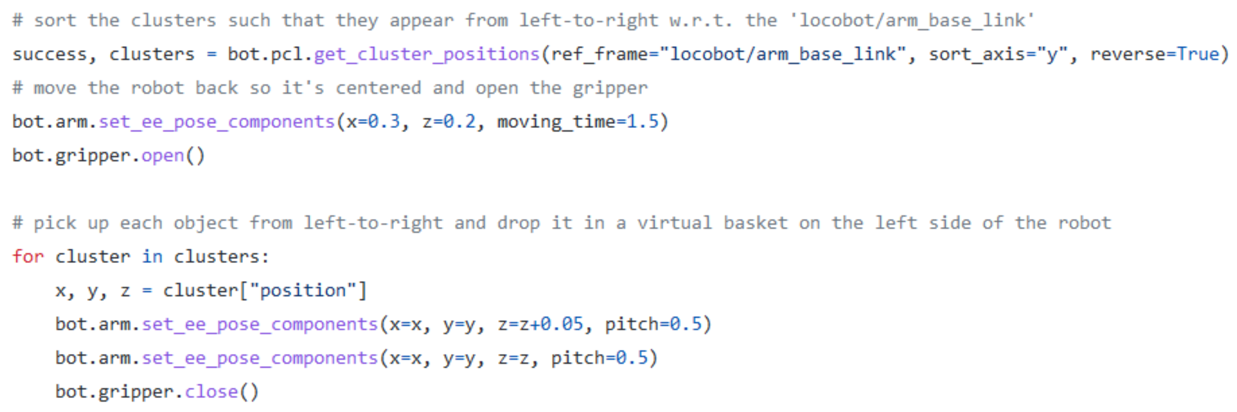
Manipulation:

Manipulation refers to using the attached WidowX-250 arm to interact with the environment, usually performing pick and place routines. Perception is key in manipulation because it does not matter how capable an arm or gripper is if it does not have a precise occupancy map of its environment and its target. There are currently two approaches to manipulation with the locobot. First, the Interbotix API can be used to move the arm to specific pose using inverse kinematics and open or close the gripper at will. More information on the available API calls and their arguments can be found at:

- https://trossenrobotics.com/docs/interbotix_xsarms/python_ros_interface/index.html#functions.

The second approach is to use the MoveIt graphical interface or API to plan and execute pick and place routines. The first approach is simple but is not aware of its surroundings, meaning it is up to the programmer to prevent it from colliding with its environment and making sure it successfully picks or places an object. Effectively, this could be considered “open loop mode”. MoveIt on the other hand is a very robust and full-fledged manipulation API and GUI which can create an occupancy map of its surroundings and prevent collisions with the environment. Additionally, it can implement several different grasping algorithms which will make it much easier to pick and place oddly shaped objects. The downside to MoveIt is that it has a much steeper learning curve and that Interbotix has only implemented basic functionality into their MoveIt API, this means that many of MoveIt’s more advanced features will have to be implemented by the user. More information can be found at https://trossenrobotics.com/docs/interbotix_xslocobots/ros_packages/moveit_interface_and_api.html.

Due to time limitations imposed by technical problems with the locobot hardware which plagued the project throughout most of the semester, a decision was made to use the Interbotix API rather than MoveIt. To locate objects the depth camera was used to detect clusters of points. Once the locations of the clusters were known relative to the robot frame the arm was moved to a position that facilitated picking. This position might be different depending on the object being picked up. In the case of a conical cup the gripper was positioned above the cup so it could descend upon it and grasp it. Once the gripper was in place around the cup, the gripper was closed around the cup and the picking task was complete. Template code for using the Interbotix API is provided in Figure 9.



```python
# sort the clusters such that they appear from left-to-right w.r.t. the 'locobot/arm_base_link'
success, clusters = bot.pcl.get_cluster_positions(ref_frame="locobot/arm_base_link", sort_axis="y", reverse=True)
# move the robot back so it's centered and open the gripper
bot.arm.set_ee_pose_components(x=0.3, z=0.2, moving_time=1.5)
bot.gripper.open()

# pick up each object from left-to-right and drop it in a virtual basket on the left side of the robot
for cluster in clusters:
    x, y, z = cluster["position"]
    bot.arm.set_ee_pose_components(x=x, y=y, z=z+0.05, pitch=0.5)
    bot.arm.set_ee_pose_components(x=x, y=y, z=z, pitch=0.5)
    bot.gripper.close()
```

Figure 9. Template code for grasping using Interbotix python API.

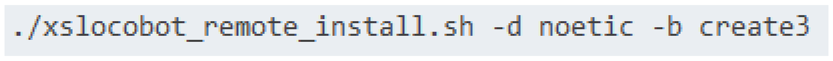
4. Step-by-step Instructions for Installing and Running

Interbotix installer shell script

Interbotix has two shell script (.sh file) that will install all of their packages mentioned in their documentation and add the necessary lines to the .bashrc file for you. Follow the instructions at:

- AMD64 (NUC computer on the robot):
  - https://trossenrobotics.com/docs/interbotix_xslocobots/ros_interface/software_setup.html#amd64-architecture
- Remote install (your computer used to control the robot remotely):
  - https://trossenrobotics.com/docs/interbotix_xslocobots/ros_interface/software_setup.html#remote-install

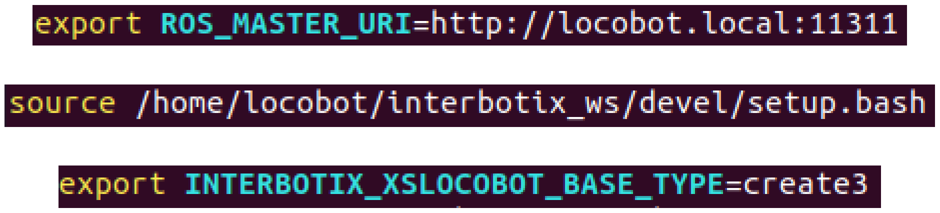
Make sure to set the right ROS version and base type when running the shell script, as seen in Figure 10.



```
./xslocobot_remote_install.sh -d noetic -b create3
```

Figure 10. Template code for grasping using Interbotix python API.

There are three important lines that the installation of the Interbotix packages adds to the .bashrc file as shown in Figure 11.



```
export ROS_MASTER_URI=http://locobot.local:11311
source /home/locobot/interbotix_ws/devel/setup.bash
export INTERBOTIX_XSLOCOBOT_BASE_TYPE=create3
```

Figure 11. Template code for grasping using Interbotix python API.

The first sets the ROS master address to that of the locobot, this way the ROS instance on the remote computer will be connected directly to the ROS master on the locobot meaning all the topics and messages will be shared between both systems. This line should be commented out when not working remotely with the physical robot since it will prevent ROS from working locally.

The second sources the setup file for the Interbotix software which sets up the ROS and Interbotix environments.

The final line sets the base type environmental variable so the Interbotix packages and launch files will know which base is being used with the locobot.

Remote Connection to the Robot:

If the locobot has been properly configured, the robot can be connected to remotely by opening a terminal and running the command:

- ssh locobot@locobot.local

Alternatively connect to the router managing the network the robot is connected to and find the robot's IP address. Additionally, if the IP address is dynamic, configure it to be static so that it will never change. To connect using the IP address use the following command:

- ssh locobot@<IP ADDRESS>.local

Setting Up the Create3 Base:

Care must be taken to ensure that the Create3 base is properly configured so that the Locobot can properly communicate with it. Importantly the namespace on the Create3 base must be set to match the default used by the locobot. Instructions can be found at: https://www.trossenrobotics.com/docs/interbotix_xslocobots/getting_started/create3_configuration.html#create-3-configuration

If errors are encountered during SLAM where transforms require a significant extrapolation into the past in order for the robot to localize, it may be an indication that the robot's clock is out of sync with the base. A fix for this can be found at: https://www.trossenrobotics.com/docs/interbotix_xslocobots/troubleshooting/index.html#less-common-issues

Running Packages on the Robot:

To run a package on the robot first make sure the remote pc and the robot are connected to the same network. Next connect to the robot through ssh as described previously. Once a terminal into the robot is established run any package just like you would on a remote PC.

For your remote pc ROS instance to communicate with the robot's ROS instance as its running (in order to for example visualize the robot in RVIZ) make sure that the line declaring the address of the "ROS_MASTER_URI" in your .bashrc file is not commented out and resource if needed. This will make your ROS instance defer to the ROS master node running on the locobot effectively syncing the topics and messages on both systems.

Visualizing the Robot in RVIZ While Its performing Tasks:

The ROS tool "RVIZ" can be used to visualize the robot while it performs a variety of tasks. To render a 3D display of the robot and its environment the tool consumes a fair number of resources so it should usually be run on a remote computer for performance reasons. This can be done using the command:

- roslaunch interbotix_xslocobot_descriptions remote_view.launch

Additional Instructions/Comments on setting up Personal Computer and Robot Computer

We download additional files from the Trossen Robotics GitHub to our personal computers interbotix_ws and then removed most of the Catkin_Ignore files inside the folders that were downloaded. If a package was seen twice then the Catkin_Ignore was not deleted in one of the packages.

- interbotix_ros_toolboxes: https://github.com/Interbotix/interbotix_ros_toolboxes
- Interbotix_ros_core: https://github.com/Interbotix/interbotix_ros_core
- Interbotix_ros_manipulators: https://github.com/Interbotix/interbotix_ros_manipulators
- Interotix_ros_arms: https://github.com/Interbotix/interbotix_ros_arms

The main link to the interbotix GitHub is provided below:

- Interbotix GitHub: https://github.com/orgs/Interbotix/repositories?type=all

5. Results

Due to technical issues, primarily concerning communication with the Create3 base across the ROS to ROS2 bridge, which encompassed a substantial percentage of the semester, the goals stated previously were not fully achieved. Most of the sub-tasks outlined above were completed in isolation, however, it was not possible to complete them all sequentially due to time constraints.

Mapping, localization, and navigation were achieved using the built in ROS navigation stack outlined above with a high degree of repeatability when using RVIZ to set goals on the map. The performance of RTAB-Map was lackluster and better results were obtained when using another mapping and localization package called SLAM-Toolbox on recorded bag files. Unfortunately, SLAM-Toolbox could not be implemented in real time on the robot by the end of the semester. On the other hand, MoveBase was successful both at navigating, and at reacting to and adjusting its planned path relative to dynamic obstacles like people. Additionally, the robot was repeatably able to detect, pick, and place a brightly colored 4oz cup used for testing. The reliability of this procedure, however, decreased dramatically when a partially translucent cup was used. Furthermore, the cup was always picked and placed on the floor partly due to the limited reach of the arm and partly because the detection of clusters is much more effective with a uniform flat background which the floor provides. Overall, the exercise was a partial success and a solid foundation for future work.

6. Discussion (Lessons Learned)

The many issues encountered in the initial stages of the project forced a confrontation with the messy underpinnings of the transform tree which allows the Locobot to orient its components in space. Though this slowed down progress considerably it did encourage a deeper exploration of how the transforms were published and how to modify them. An early effort to resolve the issue revealed that the “static_transform_publisher” node could be used to introduce transforms that effectively stitch together the disconnected pieces of the tree. Though this effort ultimately proved inadequate to fully resolve the issues, it did nevertheless yield a deeper understanding of the subject.

Investigating multiple methods for SLAM also yielded a better appreciation for the diverse ways in which sensor data can be used to perform localization and mapping. The RTAB-map package which relied heavily on camera data proved less accurate than the SLAM-Toolbox which only used the lidar, and this yielded some insight into how these methods differ in their data utilization. Additionally, The RTAB-Map approach was much more susceptible to errors involving the disconnects with the transform-tree which were encountered early on, while SLAM-Toolbox proved fairly resilient and unlike RTAB-Map was able to function after stitching the transform-tree together with static transforms. This provided insight into how the transform-tree plays a crucial role in gathering data from the robot's sensors since the transforms corresponding with various components of the robot must be properly oriented with one another to successfully perform localization and mapping.

Early attempts at navigating resulted in issues where the robot would have difficulty passing through doorways. In an attempt at mitigating this issue we were led to tweak the parameters used by the robot in generating its cost-maps for obstacle avoidance with RTAB-Map. These parameters were found available in a YAML file called "common_costmap_params.yaml" located in the "interbotix_xslocobot_nav" package under the "config" directory. By artificially lowering the robot's radius in this file, we improved the robot's ability to pass through the doorway.

Finally, attempting to bring all the components to get them to function in unison to perform a sequential task helped bring about a deeper appreciation for the complexity of coordinating the various procedures. Even after various sub-tasks had been successfully demonstrated it proved exceedingly difficult to bring them together in sequence. With the robots' successful navigation to goals, for instance, it proved difficult to accurately predict its resulting orientation, making it difficult to transition to the picking and placing place. This is something that could have been addressed by introducing counter measures given more time but remained an issue in the final phases of implementation.

7. Conclusion

A launch file was successfully implemented which initiates the Locobot navigation stack and uses a custom ROS node to set navigation goals. With SLAM performed ahead of time, a prior generated map is loaded by the navigation stack using RtabMap, and the ROS node can assign navigation goals at predefined points using types from the MoveBase package. The navigation stack then successfully plans a path and executes movement commands to guide the robot to these set points.

Separately, picking and placing functions were implemented in the custom ROS node which uses the Interbotix API to send commands to the robot arm directing it to fixed locations in space in space relative to the robot's arm location. These commands were based off the Locobot “pick and place demo’ which can be found at: https://www.trossenrobotics.com/docs/interbotix_xslocobots/ros_packages/perception_pipeline_configuration.html#id1

With more time the custom launch file and node that was implemented could be extended to both utilize the locobot perception module and properly perform a sequence of tasks. First the MoveBase package would be used to direct the robot to a predefined point at which the cup (ideally coffee) would be located. Then the node would use the perception module to identify point clusters in the immediate vicinity (assuming the cup would register as the most prominent cluster). The node would then utilize the picking function to direct the arm so that the gripper would be centered on the cluster corresponding with the cup before proceeding to trigger the gripping action and direct the arm back up to lift the cup into the air. The node would then again utilize MoveBase to direct the robot back to the starting location. Finally, the ‘placing’ function would be used to direct the arm to place the cup on the ground (or table) and release the gripper.

With even more time this same procedure could be modified and extended with the addition of preset navigation points and arm actions, until a routine was in place that could effectively direct the robot to perform the initial coffee acquisition plane. Additional orientation correction could be made by introducing April tag detection logic utilizing the locobot perception packages.

Also SLAM-toolbox could be thoroughly investigated as a replacement for RTAB-Map. Early experiments with SLAM-toolbox suggested that it provides significantly more accurate mapping by relying more heavily on the lidar data as opposed to the camera data which RTAB-Map uses so heavily. With enough time work could be done to replace RTAB-Map with SLAM-toolbox in the above-described procedure. This in turn might help reduce orientation errors which naturally emerge during movement to navigation points since SLAM-Toolbox would likely do a better job at providing precise navigation toward the specified coordinates.

8. References

[1] “Interbotix X-series LoCoBotsℑ,” *Interbotix X-Series LoCoBots - Interbotix X-Series LoCoBot Documentation*. [Online]. Available: https://docs.trossenrobotics.com/interbotix_xslocobots_docs/. [Accessed: 10-Dec-2022].

[2] “Trossen Robotics,” *YouTube*. [Online]. Available: https://www.youtube.com/@InterbotixTrossenRobotics. [Accessed: 12-Dec-2022].

[3] “Trossen Robotics (Interbotix),” *GitHub*. [Online]. Available: https://github.com/Interbotix. [Accessed: 19-Dec-2022].

[4] “Getting started” *Getting Started - moveit_tutorials Noetic documentation*. [Online]. Available: https://ros-planning.github.io/moveit_tutorials/doc/getting_started/getting_started.html. [Accessed: 19-Dec-2022].

9. Appendix

Video 1. LoCoBot in Gazebo:

https://www.youtube.com/watch?v=ZsSag2rg26g

Video 2. SLAM Running:

https://youtu.be/p1fKRDKZbWc

Video 3. Bartending Demo:

https://youtube.com/shorts/dtcC1WRhwxU?feature=share

Video 4. Pick and Place Demo Adjusted for Cup:

https://youtube.com/shorts/5GdlfPOB93E?feature=share

Video 5. Pick and Place Demo with Cubes and not Fingers:

https://youtube.com/shorts/G4ucDjc80tY?feature=share

Video 6. LoCobot Autonomous Moving:

https://www.youtube.com/watch?v=vFcMZ8oWO9A