Design of a Robotic Computer Vision System for Autonomous Operations

Ryan Coman, Mitanshu Reshamwala, Akshay Balasubramaniyan, Jordan Zhang,
Dallin Yost, Steve Qiu, Amarnath Shinde
SeHee Jung, Lokesh Kondapaneni, Pranav Vishnu Anandakumar, Nithin Joseph Madapally Abraham,
Sumukha Manjunath, Bingyi Su, Muskaan Bhargava, Akshay Vijay Khanna

***Abstract*—Construction sites often need to verify that sections have been constructed properly. Factors such as safety and manpower can sometimes restrict workers' ability to manually inspect an assembled item. This paper discusses two separate implementations of an autonomous object inspection robot designed to solve this problem. The first implementation uses a RealSense RGB-D camera and a visual-SLAM (Simultaneous Localization And Mapping) algorithm to capture the object. The second utilizes a different sensor input and uses a FARO laser scanner to scan the object of interest. Both implementations share a Clearpath Jackal as the base vehicle implementation that performs the autonomous navigation to and around the object to be scanned. The UGV team's focus is on localization, mapping, and path planning to ensure the successful implementation of the autonomous robotic system. The findings suggest that utilizing autonomous technology in the construction industry can lead to smarter, safer, and more efficient operations, ultimately accelerating construction projects in a cost-effective manner. The project highlights the role of autonomous robots in the construction industry and contributes to the scope of integration of robotics in traditional industries.**

***Keywords*—ROS, Kinova, Object Inspection, Autonomy, Visual-SLAM, Adaptive Monte Carlo Localization, Navigation**

I. Introduction

This project aims to develop a vision-based autonomous navigation system for a Clearpath robot in a construction environment. The system involves integrating machine vision algorithms such as Simultaneous Localization and Mapping (SLAM) and object recognition/scene segmentation with the robot to enable it to navigate and operate autonomously. The project also includes the design of a simulation environment to test the system's performance and train the robot for different scenarios. Additionally, feedback and motion planning strategies have been developed to ensure the robot can move around objects to be inspected while avoiding obstacles. We have achieved a nearly fully automated and autonomous inspection robot for construction and fabrication inspection. The RGB-D/UGV-B teams have paired together to create an inspector robot that uses a vision-based approach to inspection so we create our output from the data collected through our RGB-D camera and their work is covered in Chapter A. Chapter B covers the implementation of the model using the same UGV model with an alternative mapping methodology integrated with a FARO Laser System that scans the mapped layout.

II. Chapter A: RGBD/UGV-B

A. System Overview

Hardware

1) Dell Precision 5540 laptop
   Intel Xenon(R) E-2276M Processor
   Intel UHD Graphics 630 (CFL GT2) Graphics Card
2) 6-DOF, 3-finger Jaco 1 arm
3) Intel Realsense D435
4) Clearpath Jackal
5) NVIDIA Jetson Nano

Software

1) Ubuntu 18.04
2) ROS Melodic
3) Python3
4) Python2

The team used the Clearpath Jackal robot and PC devices to develop a vision-based autonomous navigation system for a construction environment. To ensure all devices are on the same network, the PC creates a local WiFi hotspot that the Jackal and other computing devices (laptop for RGB-D and Jetson Nano [7] for UGV-B) connect to. This ensures that all devices are able to communicate to each other, the Jackal serves as the ROS master for the system. This has some disadvantages since the ROS master connection is lost every time that the Jackal needs to restart, but presented a reliable solution since both teams shared the Jackal.

1) Jaco Arm: By using the Jaco arm, the team was able to move the RGB-D camera to inspect objects in the construction environment.

In addition to controlling the Jaco arm, the team utilized the RTabMap algorithm for SLAM in the demo. This algorithm enabled the detection of features from the frames captured by the RGB-D camera and facilitated the registration of all camera trajectories and keyframe trajectories. The system is tested in a simulation environment that has been designed by the team. The long-term goal is to achieve fully automated and autonomous inspection for construction and fabrication.

Figure 1 shows our Kinova Jaco arm running in our ROS environment displayed in Rviz. While we did not use Rviz to

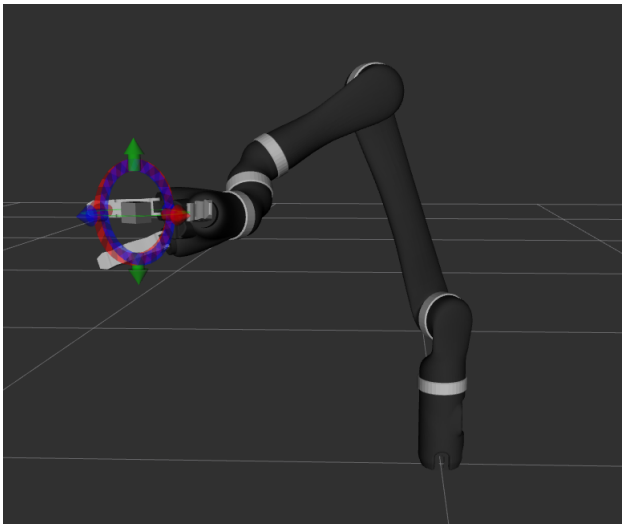

Fig. 1. Jaco Arm in Rviz

control the robot during our demonstration, it was used for development purposes. Rviz allowed us to graphically send commands to the robot and see the idealized robot move and compare it to our actual system in real-time.

Kinova-ROS API: Kinova Robotics provides ROS packages that can be used inside a ROS system to control the arm and access arm outputs such as its current pose [2]. Some of the driver code needed to be modified to run in our system as intended, but the provided URDF files and inverse kinematic calculator worked without modifications. This was very helpful since it allowed us to simply send the desired pose in a KinovaPose message and the arm could automatically calculate the joint angles necessary to achieve the desired pose. The provided GitHub contains the necessary installation and use instructions as well as examples that show how the API can be used.

2) RTAB Mapping for object scanning: The Real-Time Appearance-Based Mapping (RTAB-Map) [9] method is a type of Graph-Based SLAM that utilizes RGB-D, Stereo, and Lidar technologies. It uses an incremental loop closure detector that employs a bag-of-words technique to determine whether a new image belongs to a previous or new location. Once a loop closure hypothesis is verified, a new constraint is added to the map's graph, and a graph optimizer works to minimize any errors. To manage memory and respect real-time constraints, only a limited number of locations are used for loop closure detection and graph optimization, especially for large-scale environments. We use this algorithm to create a 3D RGB-D point cloud of the object of interest using Realsense D435 camera [8].

As you can see in Figure 4 RTABMap will compare the 2 images and create the loop closure hypothesis. This hypothesis is either accepted, rejected or maybe. Based on these the map graph is updated.

3) Clearpath Jackal: The UGV used for navigation tasks in this project is Clearpath Jackal mobile robot, which is an entry-

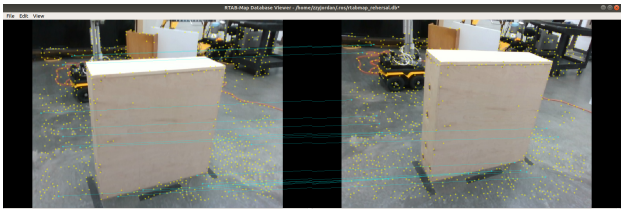

Fig. 2. Loop Closure in images captured

level robotics research platform. It is a four-wheeled platform that has been optimized for use in a variety of environments, from indoor labs to outdoor fields. In addition, the Jackal mobile robot is equipped with an onboard computer, GPS, and IMU. Its rugged design and customizable platform make it an ideal choice for researchers and developers looking to experiment with robotics in real-world environments.

One of the key advantages of the Clearpath Jackal is its support for the ROS (Robot Operating System) framework, which is a widely used open-source platform for developing and controlling robotic systems. This allows researchers and developers to quickly and easily program and control the robot using a range of programming languages, making it an ideal platform for rapid prototyping and experimentation. The Jackal can be seen in the following picture.

Fig. 3. Clearpath Jackal Mobile Robot

4) ZED Camera: The ZED camera [10] is a stereo camera system designed for depth sensing and 3D mapping. The camera uses a combination of passive stereo and active depth sensing technologies to capture high-quality images and depth maps. It has a field of view of 120 degrees, a maximum resolution of 2K (2560x1440) for color images, and a depth range of up to 40 m. The ZED camera also has an adjustable baseline distance that allows users to optimize depth sensing performance for their specific applications.

One of the key features of the ZED camera is its real-time 3D mapping capability. Using the depth information captured by the camera, the ZED SDK can create a 3D point cloud of the environment in real time. In addition, the ZED camera

also has an integrated IMU (Inertial Measurement Unit), which provides additional motion tracking data to improve the accuracy of the depth maps and 3D point clouds. The ZED camera can be seen in Figure 4.

In the process of creating the map, all environment information is sensed based on the location of the ZED camera (zed_base_link). While navigating the Jackal mobile robot in the given map using move_base, the localization is based on the position of Jackal (base_link). Since the ZED camera and Jackal mobile robot are from different nodes, there is no information to define the link relationship between zed_base_link and base_link. Therefore, before creating the map using the ZED camera, it is necessary to add zed_base_link to the URDF description file of the Jackal. The added zed_base_link_joint connects the parent link base_link of Jackal and the child link zed_base_link of the ZED camera. Also, the relative position between these two links if specified as $x : 0.21$, $y : 0$, and $z : 0.35$. The detailed information added into the $jackal.urdf.xacro$ file is as follows.

$< link\ name = "zed\_base\_link" >$
$< inertial >$
$< mass\ value = "0.2" / >$
$< origin\ xyz = "0.21\ 0\ 0.35"\ rpy = "0\ 0\ 0" / >$
$< inertial\ ixx = "dummy\_inertial"\ ixy = "0.0"\ ixz = "0.0"\ iyy = "dummy\_inertial"\ iyz = "0.0"\ izz = "dummy\_inertial" / >$
$< /inertial >$
$< /link >$

$< joint\ name = "zed\_base\_link\_joint"\ type = "fixed" >$
$< origin\ xyz = "0.21\ 0\ 0.35"\ rpy = "0\ 0\ 0" / >$
$< parent\ link = "base\_link" / >$
$< child\ link = "zed\_base\_link" / >$
$< /joint >$

Fig. 4. ZED camera

5) RTAB Mapping for environment mapping : RTAB-Map is a popular and well-documented SLAM approach that is widely used in robotics applications. One of the key benefits of RTAB-Map is that it provides a series of tutorials for a wide range of sensors and mounting scenarios, allowing users to build fully functional SLAM systems that are tailored to their specific needs. These tutorials are easy-to-follow and provide a comprehensive guide for implementing RTAB-Map in various robotics applications. One of the tutorials provided by RTAB-Map is called "handheld mapping," which involves using a handheld RGB-D camera to create a 3D map of an indoor environment. We have adapted the same tutorials to use it with the jackal. The RTABMAP node subscribes to the odometry, RGB data stream, depth stream, and camera and depth camera-related information of the ZED stereo camera through ROS topics. We launch the RTABMAP node using the following command:

roslaunch zed_rtabmap_example zed_rtabmap.launch

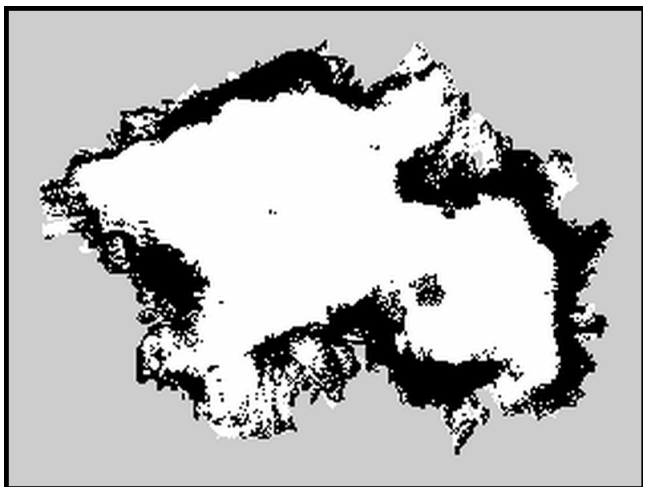

Fig. 5. Saved map of the lab using RTABMAP with Zed Camera

We navigate the jackal around the environment by publishing geometric messages to the jackal cmd_vel topic. We observe the loop closure detection window in the rviz window of RTABMAP and save the map accordingly using the map_server package. Since we are using the map generated using the sensor data from ZED Camera, we can save the map using the following command:

rosrun map_server map_server -f map /map:=/zed/map

This command saves the file in Portable Gray Map format and also saves an associated yaml file containing details regarding the Portable Gray Map file in the working directory.

6) Complete System: Each of the parts described above come together to form a complete system that is capable of localizing itself in a predefined map, move to the initial scanning position, and scan each side of the object, moving to the appropriate side between each scan. A flowchart of our final demo can be seen in Figure 6.

The Demo has 2 major phases, before arriving at the initial scanning position, and after. Before the UGV-B team has complete control over the Jackal and they are responsible for arriving precisely at the desired initial position. After the Jackal has arrived, the RGB-D team takes control and starts the scanning pattern. Once the scanning has been completed for one side, we move the jackal to a position where we can scan the second s in by rotating the Jackal Clockwise by 90 degrees.

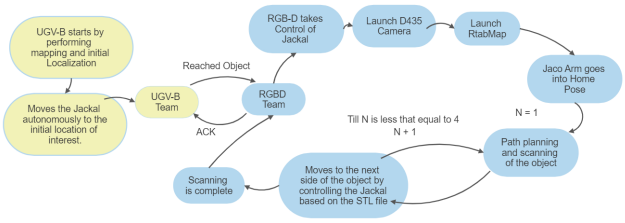

Fig. 6. Final Demo flowchart

B. System Performance

In this section, we will discuss each module of our solution.

1) Autonomous localization and navigation: Once we have generated the map of the environment using RTABMap, we utilize the 2D occupancy map generated for localization and planning required for the navigation to the desired location.

- Adaptive Monte Carlo Localization (AMCL) [11] is a localization algorithm used in robotics and autonomous systems to estimate the position and orientation of a robot within an environment. It is a probabilistic technique that utilizes a particle filter-based approach, also known as a Monte Carlo localization (MCL) algorithm, with the additional capability of adaptively adjusting the number of particles based on the current situation and sensor measurements.
  The basic idea behind AMCL is to represent the belief about the robot's pose (position and orientation) as a probability distribution using a set of particles. Each particle represents a hypothetical pose of the robot in the environment. The particles are sampled from the distribution based on the robot's motion model and updated using sensor measurements, such as odometry, laser scans, or other sensors, to refine the estimate of the robot's pose.
  One of the requirements of the AMCL package in ROS is that it expects laser data as one of the inputs. Since we do not have a lidar, we need a way to transform the depth information from the ZED Camera to the laser format. We utilize zed_laserscan_nodelet.launch file which starts ZEDWrapperNodelet and the DepthImage-ToLaserScanNodelet in the same nodelet manager. We obtain this conversion using the following command:

  roslaunch zed_nodelet_example zed_laserscan_nodelet.launch camera_model:=zed

  The depth-converted laser data is published to the topic /zed/scan. Apart from the laser data, the AMCL node also expects odometry data and a saved map (in .pgm format) as prerequisites. For the odometry, we use the $/zed/zed\_nodelet/odom$ topic publishing the odometry data of the jackal. With the prerequisites satisfied, we launch the AMCL node using the following command:

  roslaunch jackal_navigation modified_amcl.launch

  We launch rviz for the purpose of visualization. Initially, we have to guide the jackal to learn the initial pose and location. We use the "2D Pose Estimate" option in the rviz.

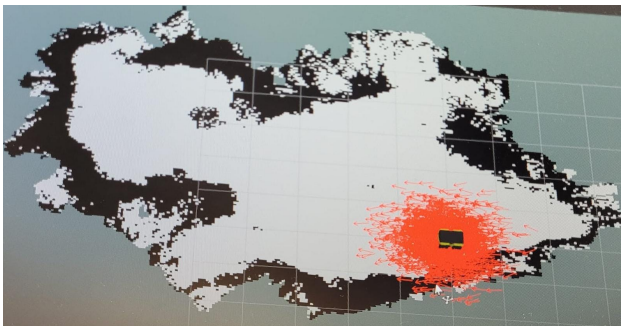

Fig. 7. Inaccurate localization of the jackal in the saved map using AMCL

One issue during localization using AMCL is that the Jackal robot keeps jumping from the initial and estimated positions. Several measures were taken to solve this problem, which are described as follows:

1. Remove redundant joint_state_publisher node

One possible reason for the issue is that two different nodes are publishing joint state information to the $/joint\_states$ topic, while the information from these two sources conflicts with each other. Therefore, the rqt_graph of the navigation system was examined. It is found that both the $/joint\_state\_publisher$ node and $/jackal\_node$ node are publishing $/joint\_states$ information to the $/robot\_state\_publisher$ node, which can be seen in Fig. 5. The measure we took was removing the $/joint\_state\_publihser$, and the result rqt_graph can be seen in Fig 6. The problem was alleviated to some extent, but the issue still remained.

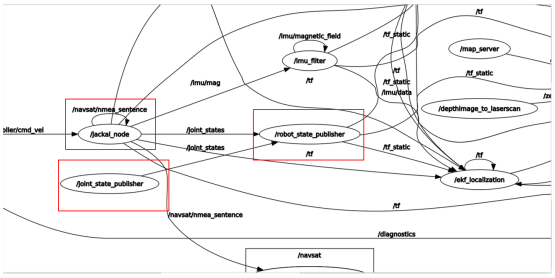

Fig. 8. Redundant nodes for $/joint\_states$ topic

2. Re-calibrate the ZED camera and create a new map

Since localization highly relies on the sensor data and the generated map, the next step taken was to re-calibrate the ZED camera and to create a new map of the lab. The calibration was done by using the ZED Calibration tool following this tutorial $https://support.stereolabs.com/hc/en-us/articles/360011828773-How-do-I-recalibrate-my-ZED-stereo-camera-$. After the re-calibration, another map creation using RTAB

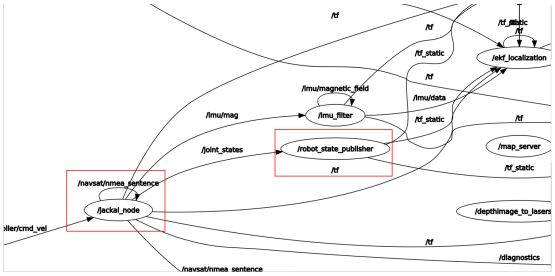

Fig. 9. rqt_graph after removing redundant node

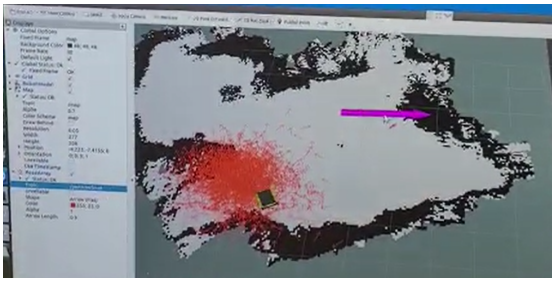

Fig. 10. Providing a target location in rviz to navigate using move base

Map was performed. However, the problem was still not completely solved after this step.

3. Changing Odom information from $/odometry/filtered$ *to* $/zed/zed\_nodelet/odom$

Another important information used in localization is odometry. When the Odom information of Jackal was plotted in RViz, it can be seen that it was not stable and accurate. On the other side, the Odom information from the ZED camera was more stale and very accurate. Therefore, we changed the Odom information from $/odometry/filtered$ to $/zed/zed\_nodelet/odom$. So far, the robot's position on the map is more stable, while the localization accuracy is still sensitive to the initial position. This issue is the limitation of the present work and should be improved in the future.

- We navigate the jackal in the saved map using the "move_base" package available in ROS. The "move_base" package [11] is designed to work with a robot's sensor inputs, such as odometry, laser or RGB-D sensor data, and a costmap representation of the environment, to generate safe and optimal trajectories for the robot to follow. It consists of several components, including a global planner, a local planner, and a costmap, which work together to enable the robot to navigate autonomously.

  The global planner is responsible for generating a high-level plan or path from the robot's current location to the goal location in the global coordinate frame. It uses algorithms such as Dijkstra's algorithm, A* algorithm, or Trajectory Rollout to find an optimal or near-optimal path while avoiding obstacles.

  The local planner is responsible for generating a low-level plan or trajectory that the robot can follow in real-time to avoid dynamic obstacles and reach the goal. It uses algorithms such as the Dynamic Window Approach or the Timed Elastic Band to generate local trajectories that take into account the robot's dynamic constraints, sensor data, and environmental information.

  The costmap is a representation of the environment that provides information about obstacles, free space, and other relevant information for navigation. It is used by both the global and local planners to plan paths that avoid obstacles and ensure safe navigation.

  "move_base" also provides interfaces for controlling the robot's motion, such as sending velocity commands to a robot's actuators to follow the planned trajectory, and monitoring the robot's state, such as its position, velocity, and sensor readings.

  "move_base" node is launched using the following command:

  roslaunch jackal_navigation move_base.launch

  We fine-tuned certain parameters with respect to the local planner and costmap parameters. The observation was that increasing the controller frequency improves the smoothness of the movement of the jackal. Reducing the planar frequency accordingly improves the smoothness of the movement of the jackal. We also fine-tuned the parameters "obstacle_range" and "raytrace_range" which impact how close the jackal can move toward an obstacle before changing the local plan.

  The demonstration of the navigation system can be seen in the following video, which can be accessed via the link $https://drive.google.com/file/d/1nK5Zt9yueamZvRrsrZ-imyUjoBWARp5F/view?usp=share\_link$

2) Object scanning: Figure 11 shows our completed system performing the scanning of the first side of the object during our final demo.

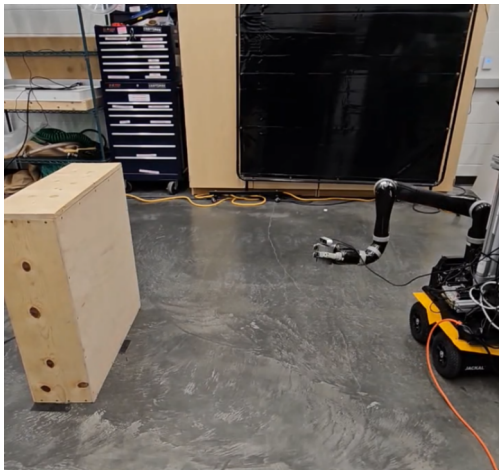

Fig. 11. Live Demo During Scanning

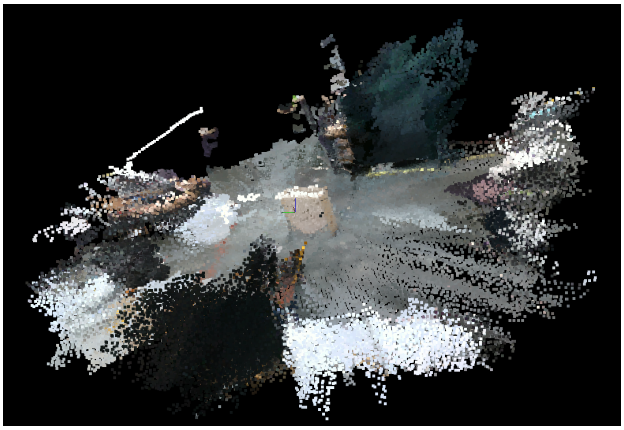

Fig. 12. RGB Odometry

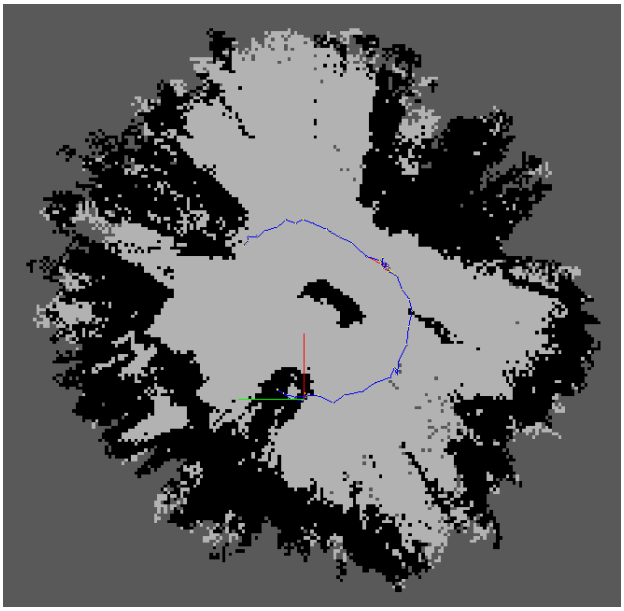

Fig. 13. Camera Trajectory

The map is used to then place the RGB images captured from the camera to the points that each frame captures to create a RGB Odometry as shown in Fig 12. We can see the Odometry data as trajectories around the object on the Occupancy map in Fig 13

As we see, the camera's odometry tracking successfully represents the path that the Jackal took, roughly a Counter-clockwise circle. We should note that this is not a direct measure of the Jackal motion since it is also sensitive to the motion of the arm. However, it is proof that the camera is correctly localizing. However, this was one of the demo's primary weaknesses. Since autonomous motion of our system often results in severe shaking of the system and is not always reliable, we decided to hardcode the motion of the Jackal around the object to be scanned. This "hardcoding" is still semi-dynamic because it reads the size of the object from an STL file and can adjust the length of the movements based on this information. The shape and motion pattern are fixed, and this information only serves to scale the path. This leaves the system vulnerable to errors in the position of the jackal relative to the object when the scan begins. If the Jackal does not start in the correct desired position, there is currently no way to know that and adjust automatically. As we saw in our demo, the Jackal arrived at a location that was slightly different than the desired and we were forced to adjust the location of the object during our demo to account for this. A more robust solution would utilize the information from the RGB-D camera to create a closed-loop system. This would require an object detection algorithm implementation that could identify an object as the desired scanning object. This object's location would then have to be compared to the estimated location of the Jackal using the odometry from the RTABMapping. This would have allowed us to dynamically determine the location of the Jackal and adjust for errors in the Jackal's location at the start of the scan.

We did not perform a specific analysis of the Jaco arm's movements since it is inherently done by the Kinova-ROS system. In order to move to the next desired point, the arm must have satisfactorily arrived at the previous point. While we do not know the exact criteria for arriving "satisfactorily", it is not important to our system since the RTAB performs its own local odometry and can localize where it is on its own. As long as the arm moves the camera in the desired pattern discussed above, which can be visually verified, our requirements are satisfied.

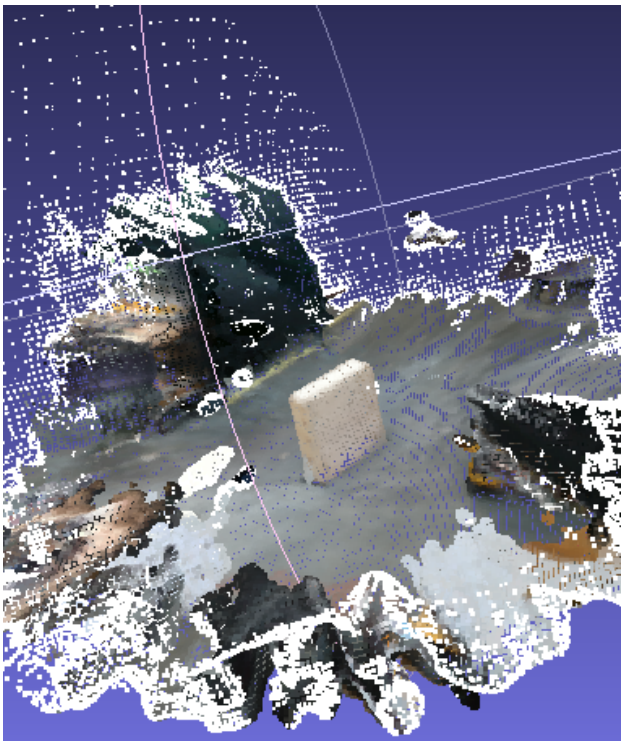

Fig. 14. Generated Point Cloud

In figure 14 we see the final point cloud generated from the

RGB-D team's scan of our demonstration object. We can see the desired object clearly in the center of the image. Around the edges, the point cloud has more noise, but that is to be expected since the camera is almost always facing the inside of the room toward the object of interest. We successfully demonstrate the ability to create a visual scanner that can create a point cloud of the scanned object. Future work could identify the desired object in this point cloud and post-process the data to exclude the surrounding environment data and only include the point cloud of the desired object.

C. Conclusion

We have tried to build a project that aims to develop a vision-based autonomous navigation system for a Clearpath robot in a construction environment.

UGV-B, is responsible for simultaneous localization and mapping, as well as navigation using a ZED Stereo camera on the Jackal robot while working closely with the RGB-D team to track the object of interest and generate a point cloud of the object. We have successfully tested and demonstrated the integration of autonomous navigation with the RGB-D team on the Jackal by mapping the unknown environment, updating the known map for changes, and localizing the robot in the map using the SLAM and AMCL algorithm on the ZED Stereo camera. Our implementation of the Vision SLAM solution on Nvidia Jetson Nano has enabled us to generate a real-time map of the environment and navigate in it, visualizing a simulated Jackal model combined with a Zed camera model in RVIZ, which simulates the actions of the Jackal in the real world. We have been able to monitor the navigation status and communicate it to the RGB-D team in real-time.

The system for object scanning includes integrating machine vision algorithms such as SLAM and object recognition/scene segmentation with the robot to enable it to navigate and operate autonomously. We also tried the development of a simulation environment to test the system's performance and train the robot for different scenarios but the simulation did not come out very well so we switched from simulation to the real components. In the final demo, our vehicle is able to move from the initial scanning position to the next three different positions by following the predicted positions information given. The Kinova Jaco Arm is able to initiate the scanning job and send the message back to the laptop after it finished the scanning path. It was able to make a path for the Realsense camera to scan and create a point cloud of the object. And in the end, our team is able to use Meshlab to generate the output that we got from the RTAB-Map SLAM algorithm. The long-term goal is to achieve fully automated and autonomous inspection for construction and fabrication using an RGB-D camera. Overall, the report provides insight into the design and implementation process of the RGB-D team towards achieving the project's goals, including the approaches, challenges, failures, and successes throughout the semester.

III. Chapter B: LASER/UGV-A

Modern construction methods can create potentially dangerous environments for a worker to complete object/site inspection tasks [5]. On the other hand, we see more robotic applications in construction and manufacturing that save workers from potentially harming the project and improve efficiency on the job [6]. We therefore propose a construction inspection robot that is capable of automatically navigating to and around the object of interest, scanning the object, and automatically obtaining a 3-D point cloud for this target object. The work is distributed in two teams, the laser team and the UGV team.

The laser team is in charge of controlling a FARO Laser System to obtain detailed point cloud information. We also handle communication between two Jetson Nano's (Nano) and a Windows-based PC that sends commands to the laser.

A. Laser Scanning and Best Scanning Pattern

The goal of the Laser Scanning part is to obtain a complete 3-D point cloud of the target object. To obtain a complete point cloud, we take multiple scans from the FARO Focus-150 Scanner and register the scanned results. The registration process matches the common backgrounds within different scans and registers them into complete 3-D point clouds. One could think of scanning and point cloud registration as an experienced photographer taking multiple panoramic images, finding the common backgrounds of these images and merging them into one 3-D space to obtain details of the target object. For example, four panoramic photos at the front, back, left, and right of a car will provide 360-degree information about the car.

The FARO Focus-150 Scanner is capable of emitting pulses of laser and after the emitted laser hits an object it can receive the reflection. By computing the time difference and wavelength difference between the emitted laser and the reflection, we could detect the distance and color information of each object surrounding the laser scanner. It is important to note that a single 360 degree scan does not provide complete point cloud information for our target object. We, therefore, need multiple scans at different directions of the target object. The complete 3-D point cloud is obtained from matching common backgrounds from different scans.We experimented and obtained a scanning pattern that, a) allows automatic registration for the FARO software without human intervention, b) covers all the geometry details for the target object and c) comparatively takes less time to finish.

Fig 16 shows the registered image including where the scanner was relative to the object of interest, a wooden box in this case, for the final demonstration.

As shown in Fig 15 above, Placing the target object at the center of a circle, we propose using 4 scans in 3,6,9,12 o'clock directions. Each scan is 360 degrees horizontal and 150 degrees vertical, with a quality index of 1/4. Experiments showed promising automatic point cloud registration results with this scanning configuration. Figure 17 illustrates the scanning of the target object and communication between the

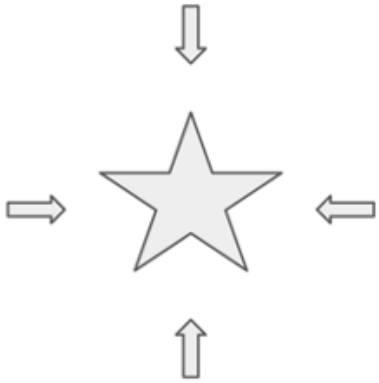

Fig. 15. Final Scan Pattern

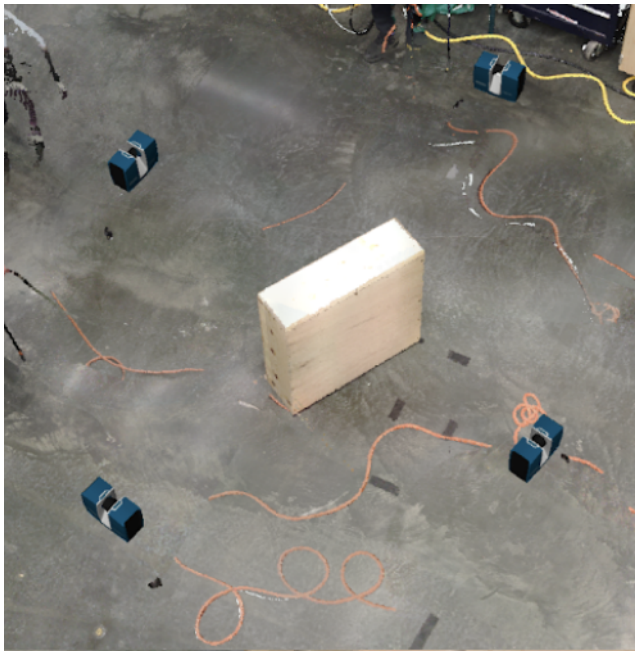

Fig. 16. Scan Pattern at Demonstration

jackal and the laser team Nano. After the scans, we obtain a complete point cloud of the target object in 15 minutes.

B. System Simulation on Gazebo

We simulated the whole planned sequence on Gazebo so that we could make changes to the urdf file as well as test amcl localization and path planning before implementing the actual model. Affixed below is the approach we took. We used gmapping to get a yaml and pgm map file obtained using the map_server topic. Amcl takes in information from the odometry data, laser scanner and an existing map to estimate the robot's pose.Before navigating, you need to initialize the localization system by setting the pose of the robot in the map. This can be done using 2D Pose Estimate in rviz or by setting the amcl initial_pose parameters.

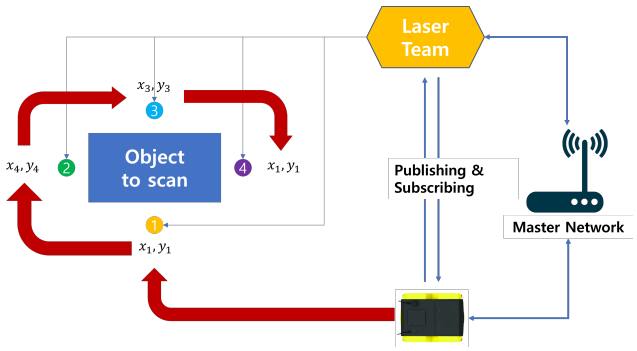

Fig. 17. Overview of the scanning and communication

The "robot_localization" is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. It contains two state estimation nodes, ekf localization node and ukf localization node. In addition, robot_localization provides navsat transform node, which aids in the integration of GPS data.

Figure 18 below is an illustration of our approach on a gazebo simulation environment.

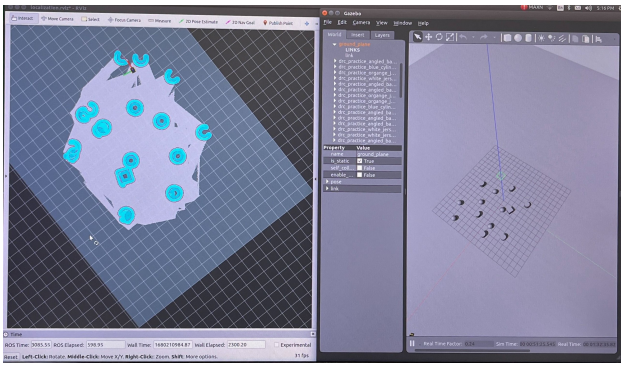

Fig. 18. amcl localization and path planning in gazebo simulation environment

C. Mapping of Lab Layout

Mapping of the lab layout was done using the Jackal UGV and zed camera. Our approach used the laser scanner data obtained from the depth image of the stereo zed camera using the depthimage_to_laser node. It takes a depth image (float encoded meters) and generates a 2D laser scan based on the provided parameters. It uses lazy subscribing and will not subscribe to an image or camera_info until there is a subscriber for scan, which in our case is the stereo imager.

The depthimage_to_laserscan is invoked from the zed nodelet package. It uses the feed from the zed_left_camera_frame which is subscribed to the sensor_imgs/Image topic which provides the depth information which is mapped to the /zed/scan topic which works as an alias for the /front/scan topic set for SICK LMS111 laser scanner.

Hence when using the gmapping launch file, it is necessary to mention the scan topic as /zed/scan as it is set to /front/scan

by default. The scan topic can vary with model used and can be obtained from the rostopic list.

The jackal is slowly moved around the lab layout to build the map. As obstacles come into view of the laser scanner (zed alias), they are added into the map which can be monitored in the rviz visualization tool. The mapping can be done either manually using a bluetooth operated PS controller connected using bluetoothctl or semi-autonomously by sending nav goals to specific locations in the layout. The map marks free space in grey and obstacles in black.

Once we were satisfied, the recorded map was saved using the ros_melodic map_server function which allows dynamically generated maps to be saved as a .yaml file and its corresponding .pgm image file. The illustration of the map visualized on rviz is shown below with the robot_model at its origin post mapping.

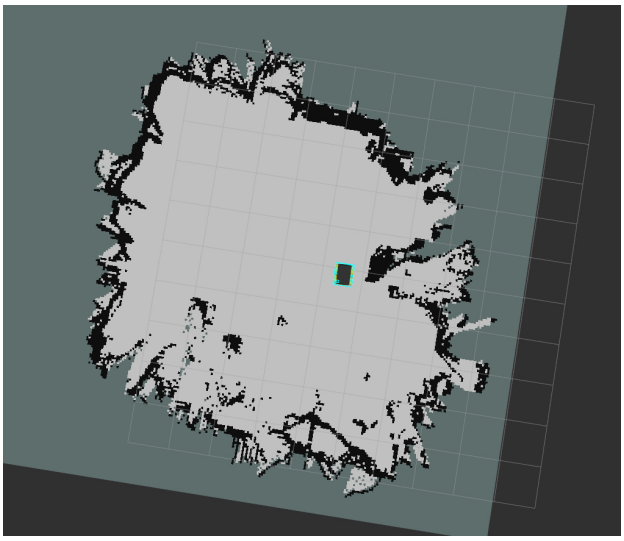

Fig. 19. Map layout of lab using gmapping

D. Localization

The UGV is able to localize itself in the mapped environment using Adaptive Monte Carlo Localization (AMCL).

It is available as an inbuilt package in ros-melodic as amcl. It is a probabilistic localization system for a robot moving in 2D. It implements adaptive KLD sampling, which uses a particle filter to track the pose of a robot in a known map, which in our case, is fed as the mapped .yaml file along with the launch file.

To achieve accurate localization, we used amcl along with the robot_localization package which is a non-linear state estimator for a robot moving in 3D space. Robot_localization uses extended Kalman Filtering and the odometry of the laser scanner (imager) to accurately estimate the pose and direction of movement of the robot model.

Both amcl and robot_localization require odometry data, which by default is assumed to come from the /front/scan topic. However, as our model uses a zed camera, it is essential that the base_link be transformed to zed_camera_center, as the robot model mounts the zed_camera and hence localizing the camera would imply the model is placed accurately in the map. Changes are made to the base_frame_id node in the jackal description files (.urdf.xacro) present in the jackal repository as well as the amcl launch file. The amcl launch file must be edited to include the odom_model_type as "diff".

The two packages now work in tandem to provide an accurate localized robot model in the map. The robot_model localized in the cost map is shown below:

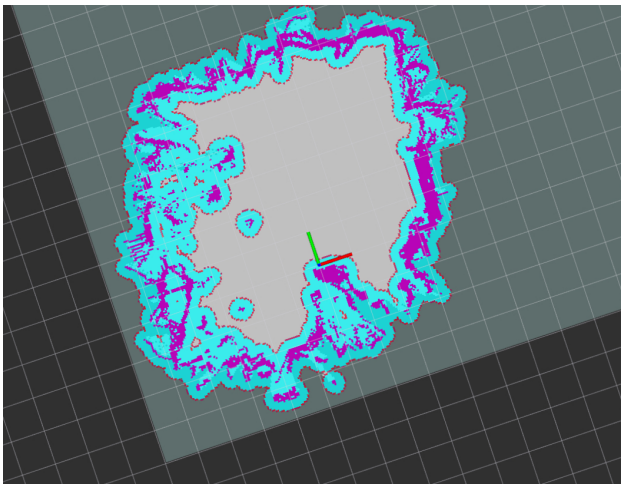

Fig. 20. amcl localized output visualized on rviz

E. Device to Device Communication

After the map of the unseen environment is generated and saved following the process so far, this map is used to visualize the movement of the robot and track where the robot is traversing to understand the location of the object of interest in the map.

Communication is established between the UGV-A teams Nano and Laser teams Nano through the Jackal as ROS-Master. All devices are connected through a wireless network established in the lab. The main script on the UGV-A Nano listens for the coordinates by subscribing on the /scan_coordinates or /scan_test topic and is obtained as shown in Figure 21

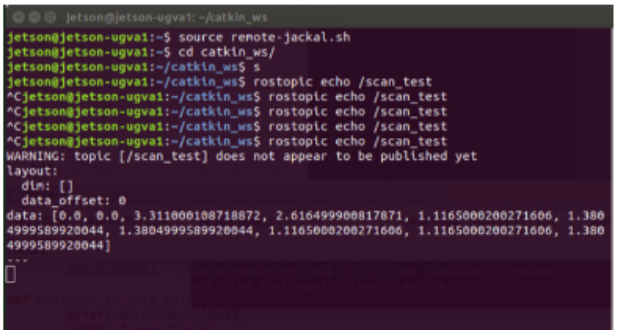

Fig. 21. Planned Path Coordinates received from FARO Nano

Additionally, ROS provides a useful tool called *rqt_graph* that allows us to visualize the ROS infrastructure the project uses to communicate between devices. In figure 22 the /ugv_dummy node is when the laser teams Nano hands control over to the UGV-A team to navigate to the next coordinate. For reference, the rectangles represent the topics messages can be published on, while the ovals represent the nodes within the program.

Fig. 22. rqt_graph showing the basic ROS Infrastructure of the program

The message type used in our case is a Float32MultiArray as it can be used to send multiple float values at the same time without disrupting the order of the message. Once the topic publishes the coordinates, these coordinates are processed and saved locally, and the script waits for the 'go' command, a boolean true value, sent by the Laser teams Nano. The topic used for this communication, /laser_to_ugv, helps UGV to understand if it should traverse to the next location. After receiving the signal from the laser teams nano, the movement function is called to move the jackal move the robot to the first given coordinates within the generated map. The robot traverses to the location and communicates that it has reached its given destination by publishing a boolean True value over the topic, /ugv_to_laser. This message signals to the laser team that the laser is in the location desired and it can start scanning at that point. The image shows the received coordinates outputs, navigation information, publishing and receiving messages over topics mentioned.

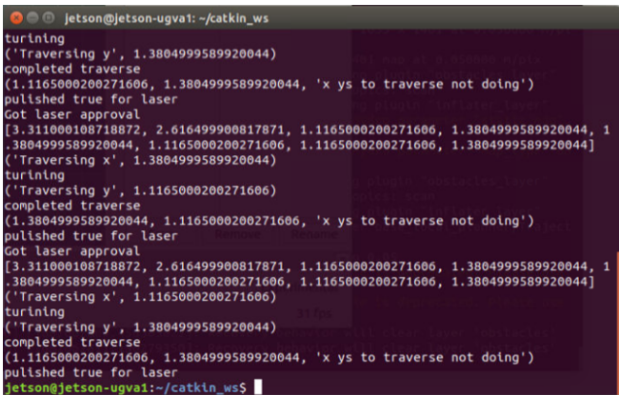

Fig. 23. Log information of communication on UGV

The robot then remains stationary for the laser to finish scanning and waits for a boolean True signal on the topic /laser_to_ugv to move to the next location in the given coordinates. This loop continues until a scan has been completed at each coordinate location, after which the script shuts down all the initialized nodes and exits the program.

Throughout this process, RViz is initialized which gives us real-time images/video feed and shows where the robot is currently in the generated map including the trajectory followed to reach the said location. This helps in understanding the unknown environment, precisely locate the location of the robot with respect to the environment and understanding the object of interest's location within the map. All the processes have little to no human interaction which makes it thoroughly an autonomous process. Figure 24 below shows the localized robot as well as the trajectory followed by the robot to scan the target.

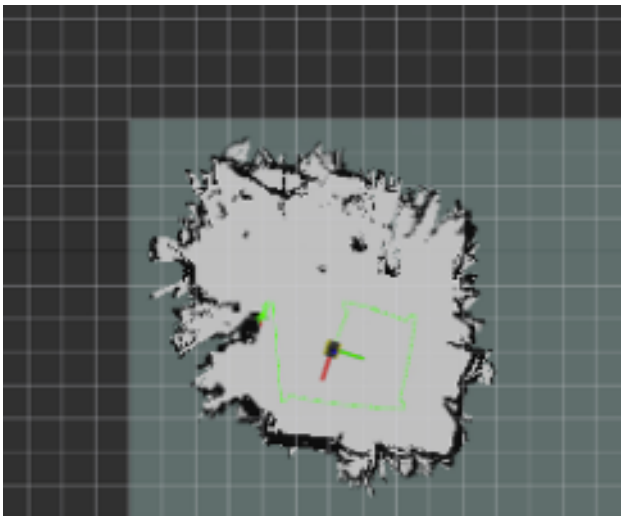

Fig. 24. Trajectory of robot model around target

Before moving on, a few quick notes regarding the scanning process for this project. First, the FARO laser scanner did not have Linux support, so we had to use a Windows machine to communicate with the scanner. To connect the laser teams nano and the windows machine, a simple socket script was written to handle controlling when signals were sent to or received from the laser scanner. Second, the FARO laser scanner we used, does not have an API. So, when demonstrating the project we had to manually walk up to the laser after receiving a signal from the laser teams nano and manually start the scan.

F. Limitations and Future Work

A constant problem that did hinder development was achieving localization using the robot model odometry from the base_frame.

The output was at best jittery, and at its worst, did not allow accurate navigation as the NavGoal is done with respect to the robot model and the base frame odometry was inaccurate. Hence, causing the robot model to move in an erratic manner. This challenge was overcome in our case by using the odom information coming from the zed camera as described above.

The behavior is documented well in online ros forums and while there are workarounds similar to the one we used, an ultimate fix has not been proposed so far. However, there is room to explore if the robot model odometry can be tuned hence not requiring accessory odom information for accurate localization. The problem described can be seen clearly in the illustration in Figure 25

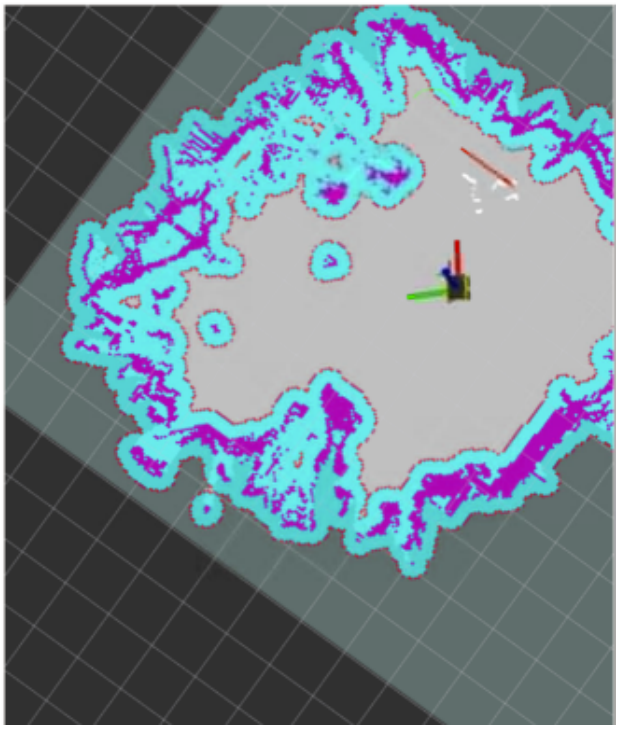

Fig. 25. Jittery localization of robot model without accessory odom information

In order to go around this issue with base odometry from the jackal, we have re-mapped all the odometry data to be received from ZED camera and its alias laser scan output. This included changing the frame when mapping and after the mapping as well. This helped in localizing precisely and reach the nav goal destination using the move_base packages with utmost accuracy. We can either use the visualization platform RViz to give the robot a destination or we can also send the jackal its destination coordinates directly through the script.

IV. CONCLUSION

At the completion of the project, a vision-based autonomous navigation system for a Clearpath robot was built. It has been demonstrated that bot the Kinova Jaco Arm and the FARO Laser scanner could produce accurate scans of a desired object. Figure 26 shows the final scan from the FARO laser.

However, for it to work in a construction environment, there is more work that needs to be done. The first limitation is that the Clearpath Jackal needs to be pre-programmed for it to get into the right positions for scanning. The second limitation is that the Kinova Jaco Arm moves quite slowly and took time to finish the scanning task. The third limitation is that the Realsense camera is not able to adjust all the parameters to make the Kinova Jaco Arm follow a good scanning path. The fourth limitation of this system is that the Kinova Jaco Arm needs to have an extension power cord for it and it has to be programmed on a laptop since the Kinova API is not compatible with the ARM-based Jetson Nano Microprocessor. At the same time, the Realsense camera also needs to be

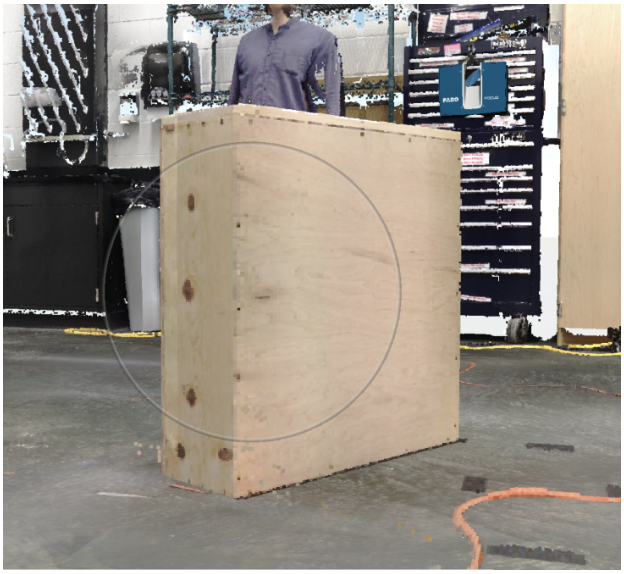

Fig. 26. FARO Laser Final Demo Scan Results

connected to the laptop to run the SLAM algorithm to scan the object. These limitations gave us problems since cords sometimes blocked the ZED camera which is attached to the Clearpath Jackal vehicle when the vehicle is creating the map. The Clearpath Jackal vehicle can be smarter so that it can fully autonomously pick up the right position for the Kinova Jaco Arm and Realsense camera to scan the object. The other work that can be done better is that the Kinova Jaco Arm can be programmed better so it can move faster. It took about two minutes to scan one surface of the object right now, if it can be programmed to move faster, it can save a lot of time to scan the whole object. At the same time, the power that has been used for Kinova Jaco Arm is from an extension cord of the power outlet, if the power of the Kinova Jaco Arm can be setup by using the battery of the Clearpath Jacal vehicle, it can be more flexible to fit in different environments.

After the Jackal Arm part has been discussed, the camera also has some parts that could be done better in the future. Overall, the camera is able to use the RTAB-Map SLAM algorithm to capture the object and create a point cloud of the object. But there is more work that could be done to the Realsense camera. First, the SLAM algorithm can be developed to make the camera able to find the object for the Clearpath Jackal vehicle so that the position of the object for scanning does not have to be pre-coded on the vehicle. This would avoid the need for the object to manually be moved to avoid conflict with the UGV during the final demo. It is better if the Realsense camera can be programmed to do the position estimation and give all the parameters that are needed for the Clearpath Jackal vehicle's autonomous navigation. If that function is able to be applied to the Clearpath Jackal vehicle, the vehicle should be able to get into the right position and stop itself from moving over there.

In regards to the Laser scanning portion, the FARO scanner

and software worked incredibly well when producing a useful, clear image of the object. However, the fact that there is no API for the scanner we used, it would be good to use a scanner that either has an API available, or has Linux support.

Overall, the system achieved the basic goal that it needed to achieve but there is still some work that could be done to make it even more efficient and autonomous.

REFERENCES

[1] ”Robotic arm.” Kinova Assistive. [Online]. Available: https://assistive.kinovarobotics.com/product/jaco-robotic-arm. [Accessed: 17-Apr-2023].

[2] ”Kinova-ROS,” GitHub. [Online]. Available: https://github.com/Kinovarobotics/kinova-ros. [Accessed: 14-Apr-2023]

[3] ”ros”, author = Stanford Artificial Intelligence Laboratory et al., title = Robotic Operating System, Available: https://www.ros.org, version: ROS Melodic Morenia, date: 2018-05-23

[4] ”clearpath-jackal-docs”, title: Clearpath Jackal Documentation, organization: Clearpath Robotics, url: https://clearpathrobotics.com/jackal-simulation/, date: 2023-04-28

[5] A. Albert, B. Pandit, and Y. Patil, “Focus on the fatal-four: Implications for construction hazard recognition,” Safety science, vol. 128, p. 104774, 2020.

[6] M. Pan, T. Linner, W. Pan, H. Cheng, and T. Bock, “A framework of indicators for assessing construction automation and robotics in the sustainability context,” Journal of Cleaner Production, vol. 182, pp. 82–95, 2018

[7] Jetson Nano. Available: https://developer.nvidia.com/embedded/jetsonnano

[8] Intel© RealSense™D435. Available: intelrealsense.com/depth-camerad435/

[9] ”RTAB-Map: Real-Time Appearance-Based Mapping” Available: http://introlab.github.io/rtabmap/

[10] Zed Stereo Camera. Available: www.stereolabs.com/docs/tutorials/depth-sensing/

[11] ”Dellaert, F., Fox, D., Burgard, W. and Thrun, S., 1999, May. Monte Carlo localization for mobile robots. In Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C) (Vol. 2, pp. 1322-1328). IEEE.”

[12] Navigation with Move-base using ROS- http://wiki.ros.org/move_base