# UGV-SLAM: Visual Tracking and SLAM for Autonomous Navigation based on UGV

Wentao Liu[1,2], Khoa Do[1], Chinmay Mahendra Savadikar[1], Jatin Nimawat[1]
*Biomedical Engineering, Electrical and Computer Engineering, Electrical and Computer Engineering, Civil Engineering*
*[1]NC State University, [2]UNC Chapel Hill*

## I. Introduction and Objectives

Autonomous robotics has a wide range of applications in and beyond areas of science, technology, and engineering such as medical field, education, manufacturing warehouses, security surveillance, space exploration, and military. In recent years, autonomous driving/robotic technology applications in the construction industry has drawn a lot of attentions. Instead of heavily rely on human labor, autonomous robots will serve an important role to accelerate construction projects, do the heavy lifting unsuitable and unsustainable to human's physics in a "smarter" way on their own, and ensure safety across the board [1]. Big players in construction equipment like Caterpillar, Komatsu, and John Deere see the value of autonomous machines in this industry and are fueling an explosion of robotics being used on construction sites in the next several years.

Inspired by this trend, this project-oriented course aims to develop a robotic system capable of autonomous navigation and operation. This robotic system focuses on field welding. It is built around Clearpath's Husky ground vehicle platform and Kinova's Jaco manipulator. Hardware design includes the integration of an embedded system (NVIDIA Jetson Nano [2]), visual sensors (camera, LiDAR, and IMU), and networking. Software development includes the design of a simulation environment, integration of visual simultaneous localization and mapping (SLAM) algorithm, and the design of feedback and motion planning strategies.

The relevant files produced and used in this project can be found on GitHub at https://github.com/britwright/sp22Robot/tree/team-ugv-slam.

## II. Methodology

Our design uses the Clearpath Husky as the UGV platform, we also developed SLAM algorithm and navigation modules based on ROS and RTAB-Map. We run tested this setup on both Jetson Nano and a laptop. Our results show that the Husky is able to successfully generate map for the surrounding environment and navigate to specified locations.

### A. Platform Overview

This project is based on a Clearpath Huksy unmanned ground vehicle. It is a medium-sized wheeled robotic platform with large payload capacity. A Kinova robotic arm is mounted on the robot's front bumper while the RealSense camera is mounted a little further back as shown in Fig. 1. Fig. 2 is the photo of the environment which the Husky navigates around. The task for this project is to generate a map of this surrounding environment and realize real time localization and mapping.

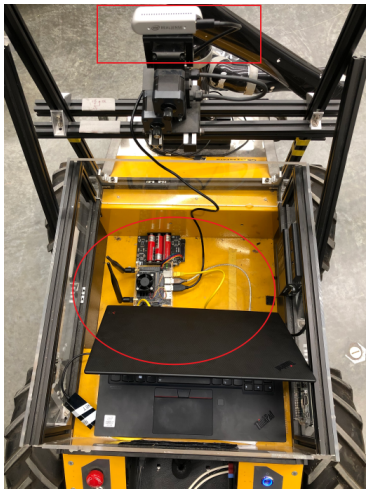

Fig. 1. The photo of husky platform. An Intel RealSense camera is mounted on the top of the UGV, with a Nvidia Jetson Nano that was planned to serve as the on-board computing platform. The Nano has been replaced by a laptop for the final run.

Clearpath provides a customized installation image of Ubuntu 18.04 "Melodic" which automatically pulls in all necessary dependencies for Husky software for installation on a laptop. Another way to setup the on-board controlling computer is based on Nvidia Jetson Nano with Jetpack installed (red circle shown in Fig. 1). It is powered by a rechargeable battery pack which allows about 2 hours of operations. Fig. 3 provides the overview of the Husky system. We would discuss different part of the system in details in the following sections.

### B. Platform development and ROS integration

*1) Intel RealSense:* RealSense D435 is a RGB-D camera which uses IR sensor for the calculation of depth and comes with an SDK provided by Intel [3]. This stereo camera offers quality depth for a variety of applications as well as a wide field of view. Fig. 4 is a demo showing the RGB image, depth

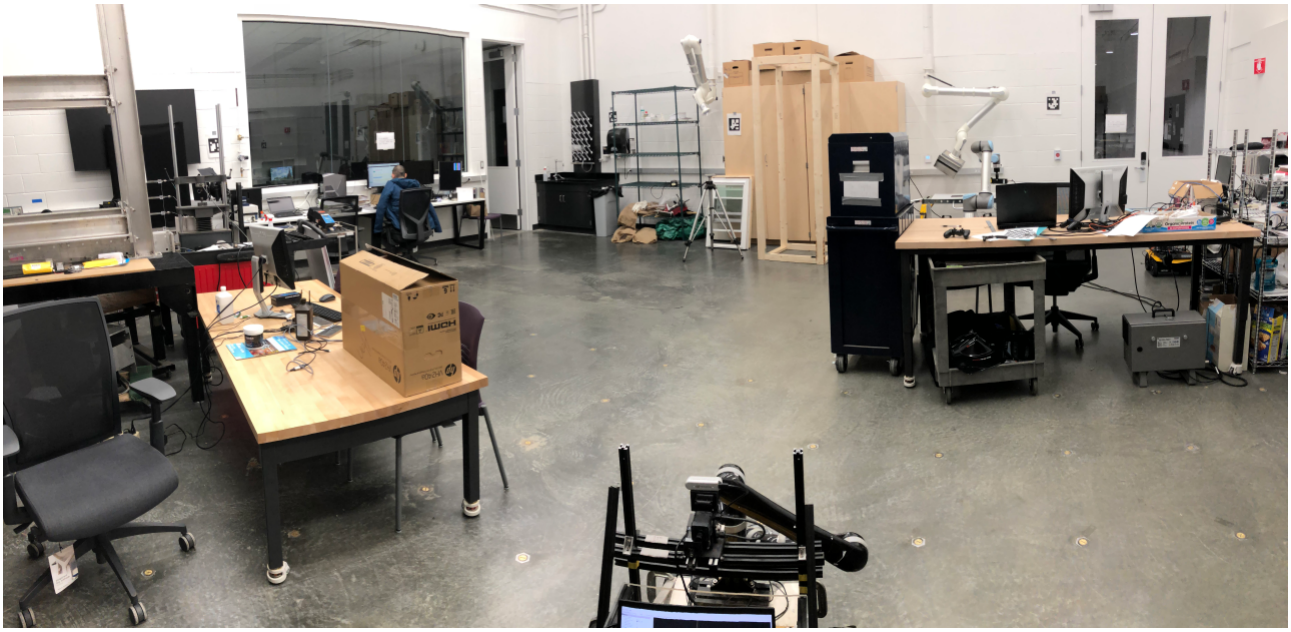

Fig. 2. The panoramic photo of lab environment.

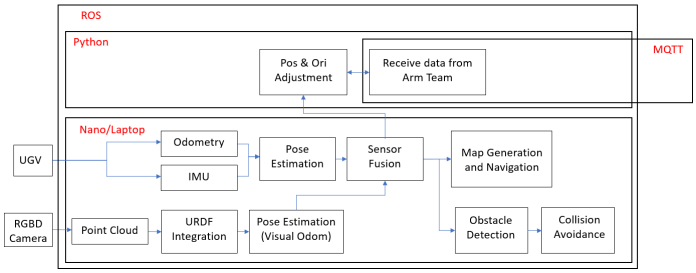


Fig. 3. Project outline of mapping and navigation based on RGBD camera, as well as communication with robotic arm.

image, as well as the camera pose through the intel realsense SDK.

*2) Husky Model and Interface:* For both Husky simulation and control, sensors integration, as well as the map generation, we develop the integrated model under Robot Operating System (ROS). ROS provides libraries and tools to help software developers create robot applications. It is an open source middleware that provides hardware abstraction, device drives, libraries, and visualizers [4].

To link the Husky and the RealSense camera in and establish a transformation between their individual coordinate systems, we integrate the Husky URDF model as well as the RealSense URDF descriptions together, which has been visualized in ROS Rviz in Fig. 5.

### C. Simultaneous Localization, Mapping, and Navigation

*1) ORB-SLAM2:* During the initial stages of the project, we experimented with ORB-SLAM2 [5] for performing Visual SLAM. ORB-SLAM2 is an extension of ORB-SLAM [6],

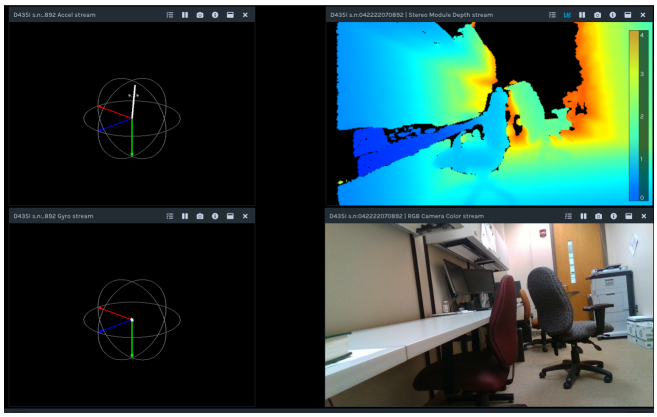

Fig. 4. Test on realsense. The figure shows the IMU and gyroscope information, RGB camera video stream, as well as depth information.

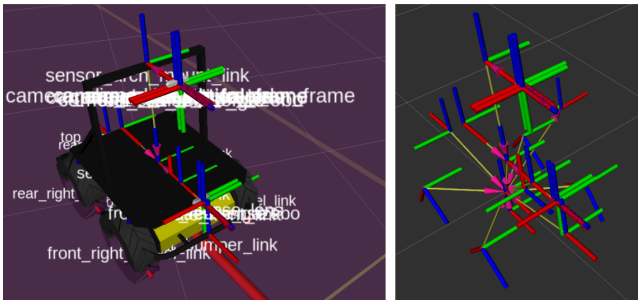


Fig. 5. Husky visualization with RealSense camera model mounted in a simulation world. Each red-green-blue shows the coordinate frame of different parts of the husky and the sensor (such as base, front bump, rear bump, camera, etc.). We used tf to maintain the relationship between coordinate frames in a tree structure buffered in time, which allows user transform points, vectors, etc between any two coordinate frames at any desired point in time.

and uses the ORB [7] image descriptors for image matching. It incorporates SLAM for RGB-D and stereo cameras and includes loop closure, relocalization, and map reuse. Moreover, it can be integrated with Intel RealSense through ROS. The RealSense camera publishes the RGB data on a ROS topic `/camera/color/image_raw`, and the depth data on `/camera/aligned_depth_to_color/image_raw`, which can be read by ORB-SLAM2. Fig 6 shows preliminary mapping results using ORB-SLAM2 and Fig. 7 shows the integration of ORB-SLAM2 with Husky's odometry in Rviz.

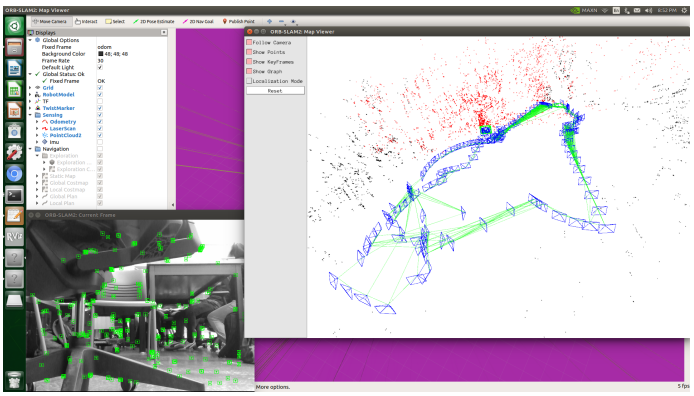

Fig. 6. ORB features visualized on the input image, along with the sparse pointcloud and the keyframe trajectory.

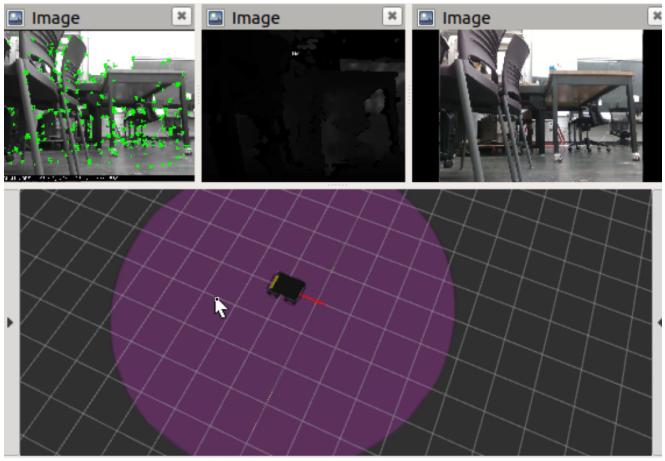


Fig. 7. The integration of the Husky, ORB-SLAM2, and Intel RealSense and their visualization in Rviz. The green markers on the image denote the ORB features detected by ORB-SLAM2.

We visualize the sparse point cloud and the keyframe trajectory using the Pangolin viewer [8]. To make the system work with the RealSense camera, we modify the default camera parameters (focal length, FPS, and depth map factor) to be specific to the camera. We use an FPS of 15, and images from the camera are downsized to $640 \times 320$.

After validating the SLAM algorithm with Realsense camera, we further proposed a solution for integrating SLAM algorithm, husky control, RGBD camera reading, map generation, and navigation based on ROS. We adopted RTAB-MAP as the map server.

*2) Husky SLAM:* RTAB-Map or Real-Time Appearance-Based Mapping [9] is a popular SLAM approach which is based on incremental appearance-based loop closure. This approach is compatible with RGB-D cameras, LiDAR sensor, as well as stereo cameras. In this approach, feature matching is performed on consecutive frames to track the motion and orientation of the camera/sensor and accordingly mapping & localization is done. RTAB-Map has an automatic loop closure detector which continuously evaluates loop closure. Whenever loop closure is detected, additional constraint gets added to the map.

RTAB-Map is well documented and provides a series of tutorials for a large variety of sensors and mounting scenarios for building fully functional SLAM systems. These tutorials are easy-to-follow and are varied enough to match most robotics arrangements. The tutorial called "handheld mapping" has been replicated in this study. However, we found that RTAB-Map was computationally heavy. We found that the NVIDIA Jetson Nano was not fast enough to run RTAB-Map in real time. Hence, we switched our computing platform to a laptop with Intel i5 9300H processor, Nvidia GTX 1650 graphics card, and 16GB of RAM. To make sure the sensor reading, calculation and control can keep up with the kinematics of the UGV, given the fact that Nano is only able to achieve a relatively low refreshing rate (around 5-10 frames/sec), we put limitations on the maximum speed of the Husky. Without the loss of generality, we present the following results based on the laptop platform. Fig. 8 shows the 2D occupancy map generated by RTAB-Map, which is used for navigation. This map is generated from the dense 3D point could, shown in Fig. 9.

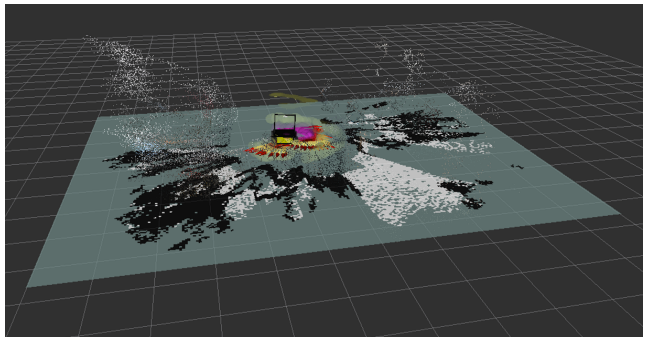

Fig. 8. The 2D occupancy grid (white and black pixles on the level of ground) of the surroundings generated by RTAB-Map. The white pixels are the space where the Husky can navigate and the black pixels are the boundaries (obstacles).

*3) Sensor Fusion:* We adopt an extended kalman filter (EKF) [10] for merging sensor measurements. EKF is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance. In this project, we use EKF to fuse wheel encoder odometry information, IMU sensor information, and visual odometry information to create a better estimate of where a robot is located in the envi-

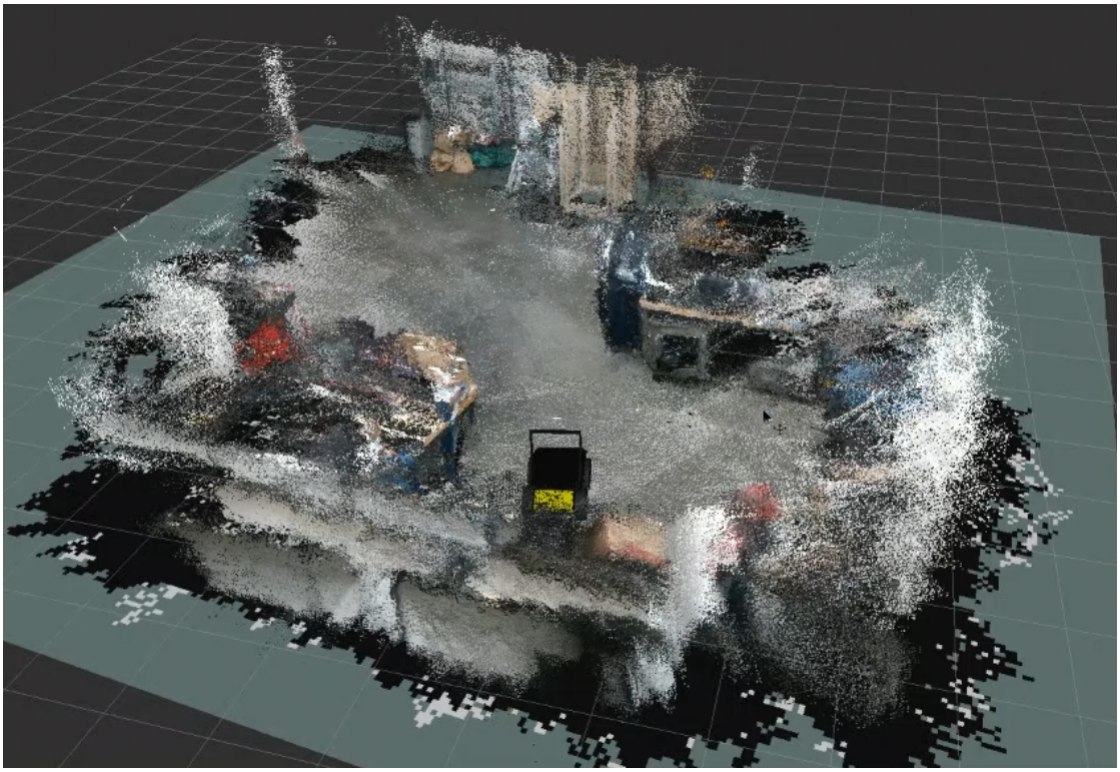

Fig. 9. 3D point cloud map generated by SLAM. Compared to Fig. 2, we can see that the 3D point cloud accurately reflects the real-life environment of the surrounding area.

ronment. The basic idea is to offer loosely coupled integration with different sensors, where sensor signals are received as ROS messages. All the sensor sources send information to the filter node within their own world reference frame. The node itself uses the relative pose differences of each sensor to update the extended Kalman filter. In this way, based on pose measurements coming from different sources, we successfully use SLAM as well as IMU and odomety measurements to estimate the 3D pose of a robot.

*4) Husky Navigation and Collision Avoidance:* We implemented the navigation module based on move base. `move_base` is a ROS package which enables a robot (base) to move to a desired goal based on an action. move_base makes use of the map generated from the navigation stack. It subscribes to the `/map` and the pointcloud topics, which are generated using RTAB-Map to generate a global and local costmaps. The global map is used for path planning to find the best path between the current location and the goal location of the robot. It also subscribes to the odometry information through the `/odom` topic to generate a local path planning, along with any transformation defined through `tf` between the odometry and the base link. We use the Husky's odometry for the local path planning, which is based on the odometry from the RealSense camera and the Husky's IMU. The final odometry is achieved through a sensor fusion of the two. Fig. 10 shows the trajectory followed by the Husky during navigation after specifying a goal by human.

We also tested the collision avoidance on husky-slam system. We added the obstacle in the route of husky by placing a cardboard box in the middle of the room. Refer to Fig. 11, we could see that algorithm could automatically detect the new object and update the map. The planned trajectory (green line) goes around the obstacle to avoid collision between husky and the box.

### *D. Cross Functional Team Collaboration*

This section describe the collaboration between our team and the ARM team for functional (to make the system work end-to-end), and algorithmic (where we collaborate to develop code which serves a common purpose).

For the end-to-end system to work autonomously, the arm needs to be alerted on the status of the navigation. This must be done for safety purposes to ensure that the arm does not move when the navigation is in progress. For this purpose, we track the status of the navigation throughout the process by monitoring the `move_base/status` topic where `move_base` publishes the real-time status of the navigation. We send a stream of 0's when the navigation is in progress (or aborted, paused, etc., essentially whenever the robotic arm is not safe to move), and 1's when the navigation is complete. The communication is achieved through the MQTT protocol, which is a low-overhead, asynchronous communication protocol popular in IoT applications. Instead of a client-server architecture, the the MQTT protocol consists of a broker, which is a device that acts as a central entity, and multiple

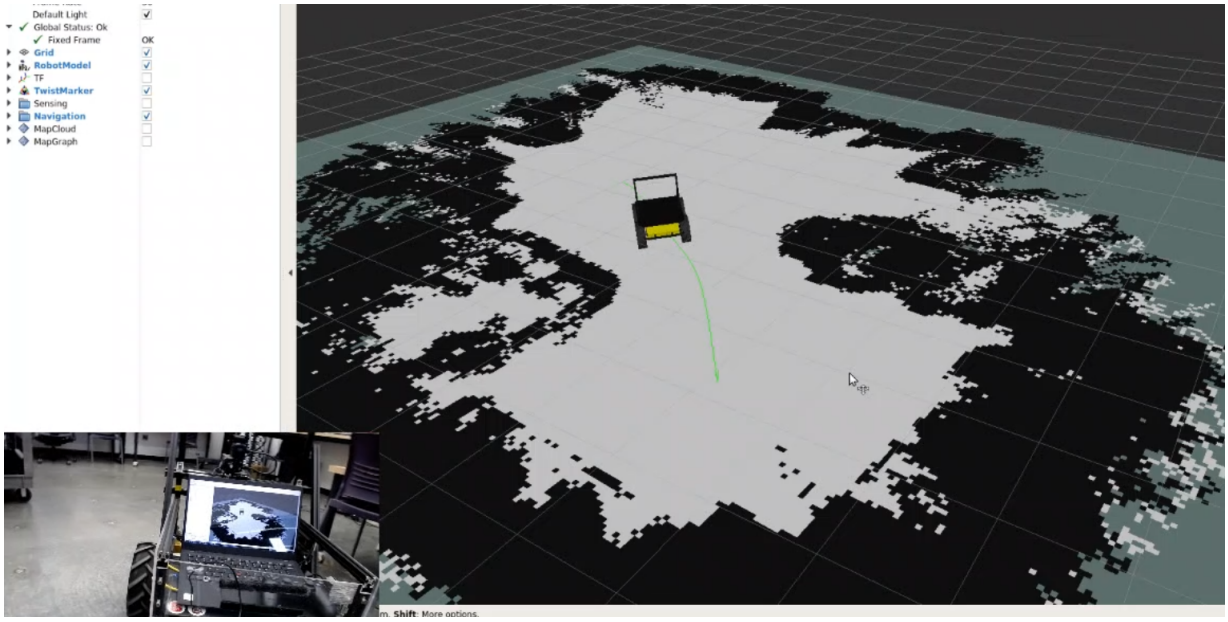


Fig. 10. The trajectory followed by the Husky during navigation. The white portion in the 2D occupancy map denotes the region where the Husky is safe to move, and the black portion is the area inaccessible to the Husky (meaning the boundary of the map due to obstacles). The bottom left corner shows the physical world around husky (captured by a webcam). We could see that the husky is facing towards the corner of the lab (where the Baxter stands), and the real-time mapping visualized by RVIZ shows the correct localization result. The green line in the figure shows the automatically planned trajectory for Husky.

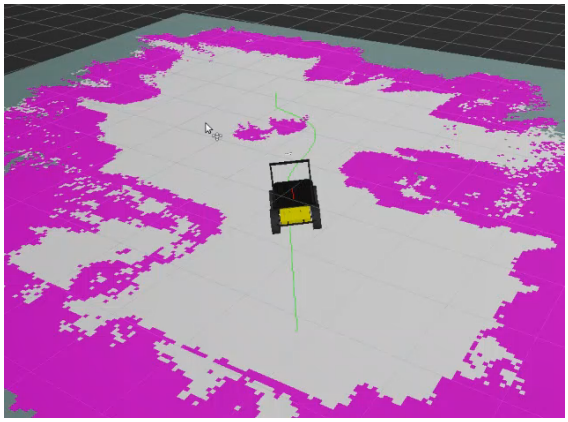

Fig. 11. The updated mapping and trajectory planning after putting an obstacle into the middle of path. The SLAM algorithm is able to detect and map this obstacle, as well as plan a route avoiding the collision.

devices which can send messages to the broker as well as receive messages. n our architecture, the Arm team runs a broker, which is used by the UGV teams and the Learning team to send and receive messages.

The status message is published on an MQTT topic named `arm_safe_to_move_ugv`. For this purpose, we develop a custom Python script, which integrates monitoring for the status (using `rospy`), and sending the message. We use the `paho_mqtt` package to achieve MQTT communication. The arm's coordinates with the Learning team to move only when a 1 is received, and hence, we do not need any direct communication with the Learning team's processor.

For cases where the welding joint is out of reach of the Kinova robotic arm, a functionality for fine movements during the welding operation is required. Since the arm blocks most of the field of view of the camera during the welding process, we have collaborated with the ARM team and created a script which uses odometry commands for these fine movements. The ARM team publishes a *3-dimentional* vector indicating the x, and y-coordinate of the final location with respect to its current location along with the final orientation of the Husky. The communication of this message is done over an MQTT topic named `movement_without_nav`. In this communication, we act as a subscriber and the Arm team as a publisher which ultimately relies on the Learning team for the coordinates.

Fig. 12 is the flow chart of our proposed algorithm for husky pose fine-tune. The distance to be travelled is then calculated using the distance formula and the target location is calculated using the arctangent function. One important thing to note here is that since the arctangent is applied on the ratio of $x$ and $y$, the information on which quadrant the target location lies may get lost. So, to combat this issue, the arctan function is applied on the absolute value of the $y$ over the absolute value of $x$ and then, we add the appropriate adjustments to the result for the quadrant information. For the first and fourth quadrant, no adjustments are required but for the second and third quadrant, we subtract the $arctan$ value from $\pi$. Once the required angle is known, a timed loop is executed giving the husky an angular velocity of 0.1 rad/s in either clockwise or counter-clockwise direction based on the quadrant of the target. Now, as the Husky is facing in the correct direction and is given a linear speed of 0.1 m/s for the appropriate duration calculated by dividing distance to be travelled with the linear speed.

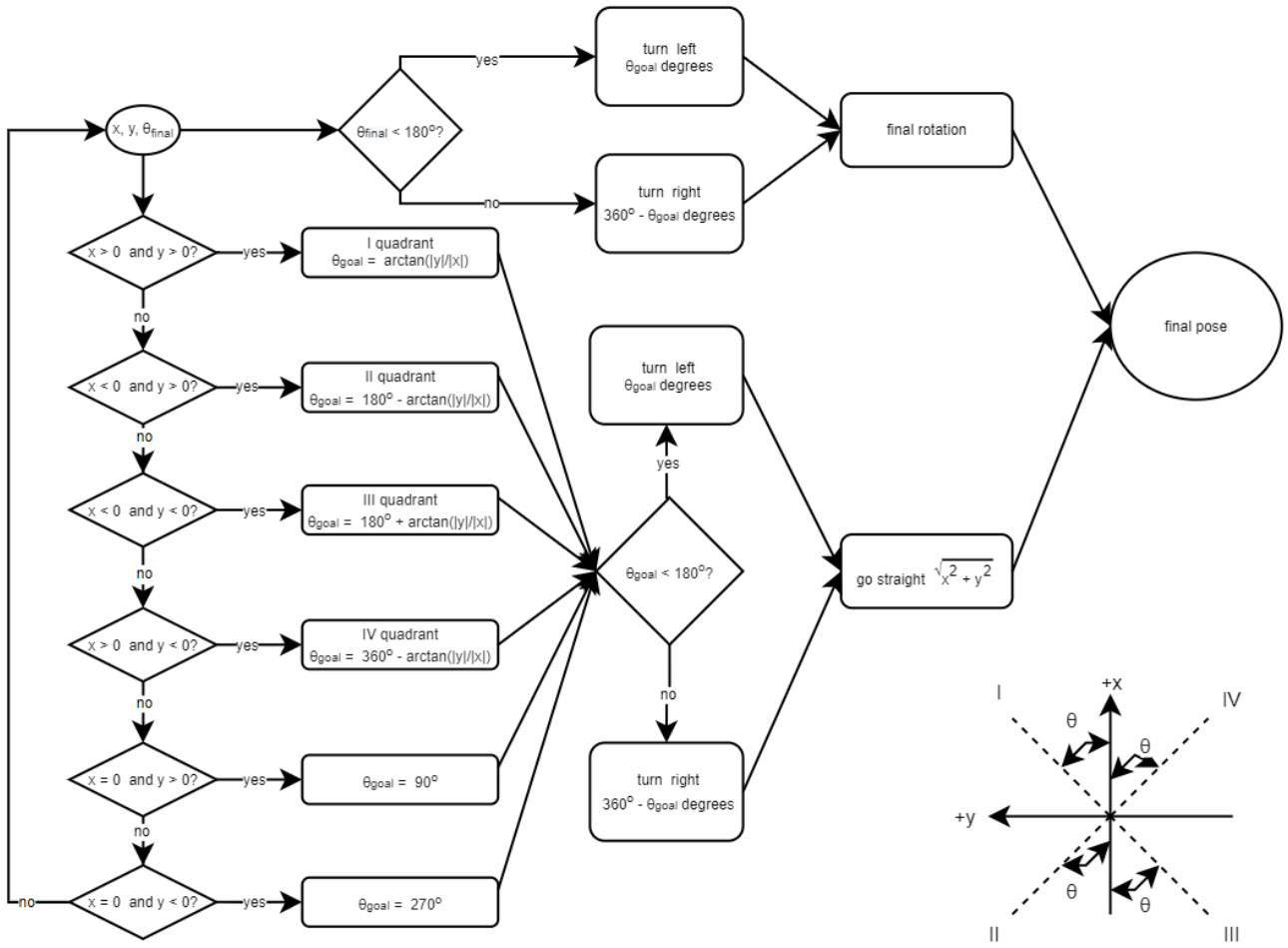


Fig. 12. Logic for moving the robot after receiving coordinates from the ARM team.

## III. Conclusion and Discussion

This unique project-based course requires a great effort from multidisciplinary teams working together. Each different team works on each different assigned task that is successfully integrated into a final goal - a fully functional robot that is capable of autonomously navigating through a construction site environment, looking for joints, and performing joint welding. Our team, UGV-SLAM, is tasked with performing simultaneous localization and mapping, as well as navigation using a depth camera on the Husky while working closely to the ARM team for monitoring the locations of the joints where the robot and the robotic arm needs to reach to. The autonomous navigation and integration with the Arm team are successfully tested and demonstrated on the Husky by mapping the unknown environment, updating the known map for changes, and localizing the robot in the map using SLAM algorithm on the Intel RealSense RGB-D camera.

We try to implement this RGB-D camera based SLAM solution on both laptop and Nivdia Jetson Nano. In both situation, we are able to successfully generate a map of the environment in real time and navigate in the generated map. We are able to visualize a simulated Husky model in RVIZ, which simulates the actions taken by the Husky in the real world. We are able to monitor the status of the navigation and communicate it to the ARM team in real time. We are also able to receive coordinates from the ARM team if the welding joint is out of the robotic arm's reach, so that we can move the robot to the desired location. We also find out with laptop as the computing platform, the refreshing rate is higher than Nano which allows a more precise and fast control of the UGV. All these works demonstrate a feasible solution for autonomous system capable of navigating in a construction environment, localizing a joint, and perform welding.

For future works, we think it might be worthwhile to run the visual-SLAM algorithm with CUDA to fully use the computing power of Jetson Nano as well as optimize the system performance. In this work, we briefly attempt to compile RTAB-Map with CUDA. In principle, this means compiling all the dependencies in CUDA, and linking those to ROS. This will allow us to use the Jetson Nano instead of the laptop used in our current setup. However, this turns out to be a time-consuming task. In the future, an embedded device such as the Jetson Nano can be used with all the libraries

compiled to use CUDA acceleration, instead of a laptop.

## REFERENCES


[1] A. Evers, “How autonomous robots are changing construction,” CNBC, 30-Nov-2020. [Online].

[2] Jetson Nano. Available: https://developer.nvidia.com/embedded/jetson-nano

[3] Intel© RealSense™D435. Available: intelrealsense.com/depth-camera-d435/

[4] “ROS Wiki” Open Robotics, 08-Aug-2018. [Online]. Available: ROS.org.

[5] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, ”ORB-SLAM: A Versatile and Accurate Monocular SLAM System,” in IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163, Oct. 2015, doi: 10.1109/TRO.2015.2463671.

[6] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, ”ORB-SLAM: A Versatile and Accurate Monocular SLAM System,” in IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163, Oct. 2015, doi: 10.1109/TRO.2015.2463671.

[7] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, ”ORB: An efficient alternative to SIFT or SURF,” 2011 International Conference on Computer Vision, 2011, pp. 2564-2571, doi: 10.1109/ICCV.2011.6126544.

[8] Pangolin View. Available: https://github.com/stevenlovegrove/Pangolin

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

[10] Ribeiro, Maria Isabel. ”Kalman and extended kalman filters: Concept, derivation and properties.” Institute for Systems and Robotics 43 (2004).