Design of a Robotic Computer Vision System for an Autonomous Welding Robot

Alexander McGirt[1], Doyun Lee[1], Britanya Wright[1]
Chinmay Mahendra Savadikar[1] Jatin Nimawat[1], Khoa Do[1], Wentao Liu[1,2],
Nicholas Albergo[1], Gabriel Francis[1], Mahathi Lanka[1],
Juan Benavides[1], Derik Munoz Solis[1], Shilpa Kancharla[1], Stuti Garg[1]
,
Biomedical Engineering, Civil Engineering, Computer Science, Electrical and Computer Engineering
[1]NC State University, [2]UNC Chapel Hill

I. System Overview

A. Introduction and Course Objectives

The objective of this course is to develop a welding robotics systems that consists of a ground vehicle, manipulator, and visual sensors. The platform consist of a unmanned ground vehicle (clearpath Husky), a robotic arm (Kinova Jaco), and sensors (Intel Realsense D435 camera, LiDAR). Hardware design including the integration of embedded platforms, sensors and networking communication, based on NVIDIA Jetson Nano and laptops. Software development includes design of a simulation environment, integration of visual Simultaneous Localization and Mapping (SLAM) algorithms,design of feedback and motion planning strategies, and object detection models.

The goal of this project is to design an UGV to automatically navigate in a construction scene while building a map of the scene and following a trajectory generated automatically after receiving a user-specified target destination, detecting welding joints, and perform welding.

B. Overall System Function Descriptions

This project is broken down into multiple teams, as shown in Figure 1. Team one focuses on joint detection and trajectory planning using Deep Learning. Team two focuses on the movement of the robotic arm for joint tracking and welding. Team three focuses on 3D mapping using visual SLAM. Team four focuses on 3D mapping using LiDAR and wheel encoders. The photo of the lab is shown in Fig. 2, which is the environment where Husky needs to map, navigate and conduct welding.

1) Learning: The learning team is responsible for detecting welding joints and communicating a trajectory along the welding seam to the robotic arm. Using a YOLOv5 object detection model the learning team identifies a class (butt joint/ T-joint) and an approximate bounding box of the metal plates from the camera feed supplied by the Realsense camera. From this bounding box a welding seam is extracted using color thresholding and its 2D location is mapped to the depth frame to get 3D coordinates for a series of points along the seam. Finally, these coordinates are published to the arm team via MQTT.

Fig. 1: Overview of the entire system

2) Visual SLAM: The Visual SLAM component focuses on mapping and navigation of the construction environment. Visual SLAM uses images (video) for sensing the environment, and, on a high level, works on feature matching between adjacent frames. We use the Intel RealSense 435D as our visual sensor, which generates an RGB-D iamge used by the SLAM algorithm. Along with the RGB-D, we use the visual odometry and the Husky's IMU sensors (fused with an Extended Kalman Filter algorithm), for SLAM using by RTAB-Map, an open source Visual SLAM package. The ROS `navigation stack` uses the map generated by RTAB-Map and does path planning, and this information, along with the odometry, is used for moving the Husky in the environment using `move_base`. We make use of `tf` and URDF files to link the coordinate systems of the `base` (Husky) and the `camera` (RealSense). This setup makes it easy for `move_base` to perform navigation.

3) LiDAR: Although the Husky UGV system is quite robust, various types of sensors and actuators will need to be added to it before autonomous welding operations can be carried out. To make the robot move to the desired position, map generation using LiDAR is needed. The 2D Lidar system utilized is a current product on the market created by SlamTec, called the RPLidar A2 laser range scanner. This sensor generates light energy reflected from various surfaces where the LiDAR sensor measures the return energy. With

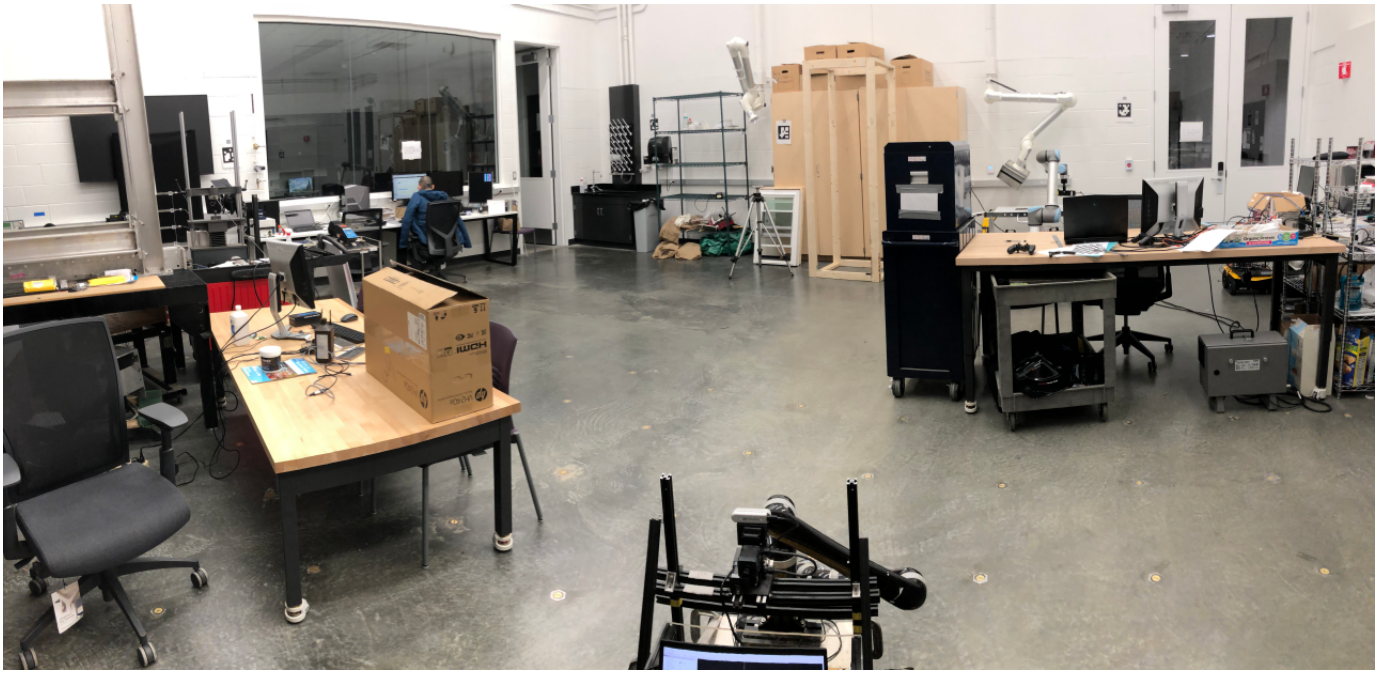


Fig. 2: The photo of lab environment.

the information on the direction and range of the laser beam, in conjunction with the device's position at which the sensor is attached, a 2D representation of the desired space can be formed. Providing a scan output of up to a 16-meter range radius, RPLidar A2 is to be utilized to generate a 2D rendering of the environment around it. Gmapping, one of the most widely used Simultaneous Localization and Mapping (SLAM) methods, is used in this research. It will be utilized to help navigate the Husky around the construction environment while also aiding in collision avoidance efforts when applicable.

4) Robotic Arm: Welding requires a programmable arm to follow the seam as required. To facilitate this we have made use of the Kinova Jaco 2. This is an arm originally made for use by the differently abled, but it also allows for programmable use via a USB connection to a computer with ROS along with the necessary Kinova packages installed on it.

The arm is able to receive both relative and absolute positional coordinates, and angular coordinates in the form of radians, degree or quaternions from the computer controlling it. The coordinates of end effector of the arm are relative to the location of the base of the arm. The angle controls work relative to the current position of the end effector.

The arm driver also allows for joint based control where every joint can be moved by a certain angle, but this feature was not used was Cartesian coordinates just made more sense.

C. System Integration

1) MQTT communication between UGV teams and ARM teams: Two methods are used to handle communication between teams: MQTT and ROS.

The UGV module monitors the status of navigation and sends the status to the Arm module via MQTT. The UGV teams will constantly communicate with the Arm team to give updates on when it is safe to move the arm. The UGV teams will move from the start location to the target location. During the duration of the UGV's travel, a 0 (or a series of zeros in the case of vSLAM team) is sent to the Arm team to signal "unsafe to move." Once the target location has been reached, a 1 is sent to the Arm team to signal "safe to move, proceed". All communication is done with MQTT, a messaging protocol for IoT devices.

Communication from the arm to the UGV is also possible where the arm can send a message to the UGV in the form of an x and y relative distance and an angle. These can be used to inform the UGV of where it should move to relative to its current position and what direction it should face.

Once the target location has been reached, the arm along with the learning team looks for the seam which has to be welded. In the event that the seam is not found, the arm tells the UGV to rotate by 30 degrees in the clockwise direction in order to get a better view of the environment around it. If a seam is found but it is beyond the reach of the arm, the UGV is told by the arm to move such that the UGV is only 0.3m from the seam and directly facing the it as well.

To send messages, the NVIDIA Jetson Nano for the UGV Lidar team (or a laptop computer for the UGV vSLAM team) is set as the publisher while the Arm team's computer is set as the subscriber, as shown in Figure 3.

2) Deep learning and robotic arm integration: The learning team communicates with the ARM team using MQTT when a seam has been detected. Before a seam has been detected, the

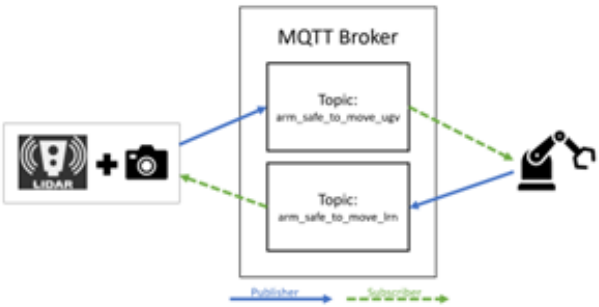


Fig. 3: MQTT broker and topics connecting the modules

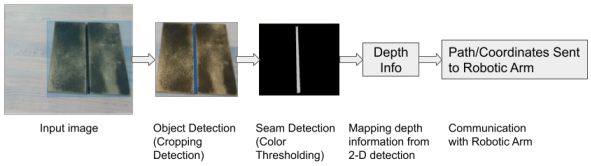
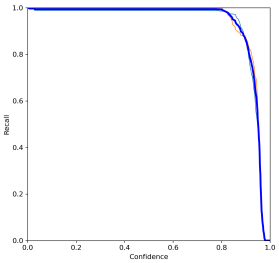
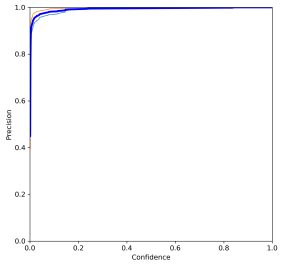


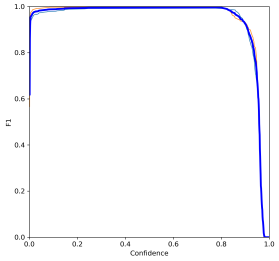
Fig. 4: Proposed workflow for seam detection.



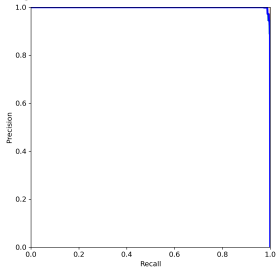
(a) Recall curve for training.



(b) Precision curve for training.



(c) F1 curve for training.



(d) Precision-recall curve for training.

Fig. 5: Metrics for training the YOLOv5 model from no pre-trained weights.

learning team continuously publishes ”0” to the joint_detected topic. Once a seam has been detected, a ”1” will be published to this topic and a list of 10 points along the seam will be published to a separate ”coordinates” topic in the form [[x,y,z],[x,y,z],...[x,y,z]]. These coordinates are normalized so that the center of the camera field of view is the point (0,0) and the upper right corner is (1,1). Using this information, along with the camera field of view angles, the arm team can localize the seam in the global frame of reference.

In certain scenarios the arm will be located too close to the welding seam to obtain an accurate depth reading, or too far as to be within reach. In these scenarios, the arm team uses the depth values published by the learning team to decide whether it needs to ask UGV to move in some direction to attain a better position. This eliminates the need for the learning to integrate directly with the UGV teams.

II. System Performance

A. Joint Detection (Learning)

The learning team is responsible for joint detection and identification, seam detection, and 3D profile generation. Specifically, we are tasked with determining if a joint is a butt joint or T-joint, and then we produce the path for the robotic arm to perform weaving. Our overall workflow is shown in Figure 4. Our model takes an input image and draws a bounding box around the object of interest, and then uses color thresholding to isolate the welding seam. The coordinates for the path are gathered this way and depth information from our camera is used. The coordinates and depth reading are passed to the robotic arm via an MQTT protocol.

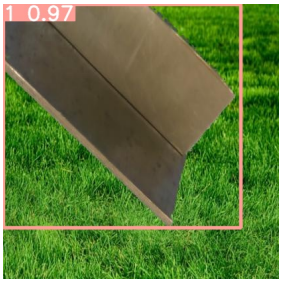
1) Object Detection: We trained a YOLOv5s model with no pre-trained weights. The small model was chosen in particular so that it would be lightweight and portable on the Jetson Nano. In order to create the object detection model, we collected data of both types of joints from far away distances, against different types of backgrounds, and introduced objects that were rectangular in shape like the joints into the view. This was done so that our model would not overfit or miss a detection if external factors such as another rectangular object or a surface that was not the table, for instance, were not in front of the camera. This variegated data was used to train the YOLOv5s model from scratch. The model produces a bounding box from which we use color thresholding to detect the seam.

The results from training the YOLOv5 model from no pre-trained weights are displayed in Figure 5. Figure 5d shows the precision-recall curve, which represents the trade-off between precision and recall. A high area under the curve represents both high recall and high precision. High recall, as shown in 5a, relates to a low false negative rate. High precision, as shown in 5b, relates to low false positives. High scores for both precision and recall demonstrate that our classifier is returning accurate results. These results show that our model was able to perform well on our training set. Figure 5c shows the F1-Score of the model while training. An F1-score of 1.0 means the model is able to predict every instance correctly, so a score close to that, as shown in Figure 5c, corroborates that our model is performing well.

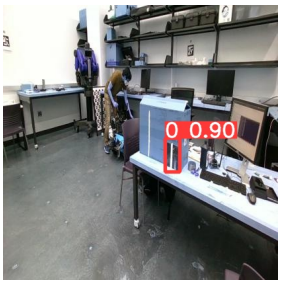
We also show a few examples of the predictions on our test set in Figure 6. Through 6a and 6b, we demonstrate that our model is able to predict the joint type with a high confidence level, at various different angles. In addition, we are able to detect joints from farther distances as well, as shown in Figure 6c. There are some incorrect predictions produced as displayed

in 6d, but these are very low confidence and seldom show up when we perform inference.

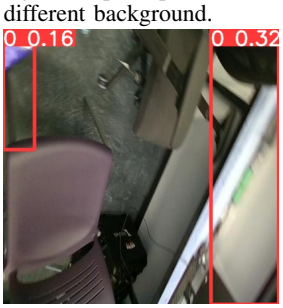
(a) High confidence prediction for a butt joint.



(b) High confidence for a T-joint superimposed on a different background.



(c) Fair confidence prediction for a butt joint from a farther distance.



(d) Incorrect predictions, but with very low confidence levels.

Fig. 6: Examples of predictions on our test set.

2) Seam Detection with Color Thresholding: Once the joints have been detected in the image from the camera, the next step is to locate the seam between the plates. We opted to use an approach inspired by the work of Gomez-Espinosa et. al. [1], in which a similar setup was employed for autonomous welding, and painted the seam such that the seam can be detected using a color thresholding pipeline.

Using color thresholding, the seam between the plates is painted and basic image processing using OpenCV is used to locate the seam. A mask is created using the threshold color by selecting a region in HSV color space that represents the seam color. This mask is used to detect the canny edges from the mask, and the edge mask is then used to detect the path of the seam using the `HoughlineP` function from OpenCV to detect the lines in edge image. The 2D coordinates for the seam are then mapped to the depth frame so that this information can be passed to the robotic arm for welding. The actual transformation into the 3D world coordinates was done on the robotic arm's end using the information sent by the learning team.

3) Inference on the Jetson Nano: For the final product, the Pytorch YOLO model was converted to an ONNX model and then this model was used to create a TensorRT engine the engine was built and serialized on the Nano using the python TensorRT and PyCuda modules. The model was built with FP16 precision down from the original FP32 used for the ONNX model. Using this TensorRT engine and building it with the same input image size that the original YOLO model was trained with (640x640) the results from the engine were almost identical to those from the OpenCV implementation and the original Pytorch, but the frame rate drastically increased from around 2 frames, on the OpenCV implementation that was used in the earlier stages of development, to around 6 frames per second.

4) 3D Profile Generation: Another area of interest explored by the learning team is 3D profiling, in particular the capabilities achieving this on a lightweight system such as the Jetson Nano with the use of the Intel Realsense RGB-D camera. We generate point clouds with the use the `pyrealsense` library. The results of this are displayed in Figure 7.

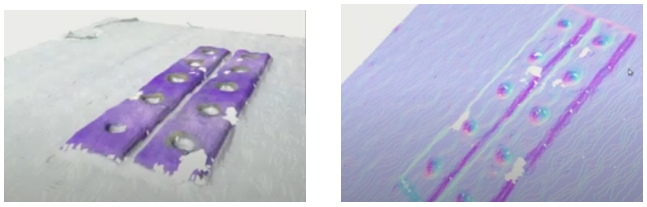


Fig. 7: 3D point clouds generated from the RGB and depth data for a butt joint.

Because the 3D point cloud is generated from the RGB and depth frames that we were already using to map the welding seam in 3D space, it is not necessarily providing new information, but serves as a different way to visualize the data. Currently, we are not generating point clouds during regular operation, as we can already map the welding seam in 3D space without this, so it will only slow down performance without much added benefit.

5) Issues Faced: List of past and ongoing issues:

- There were several issues with overfitting on certain types of data. For instance, the detection algorithm draws a bounding box around the surface the joint is placed. While running inference on a joint placed against a

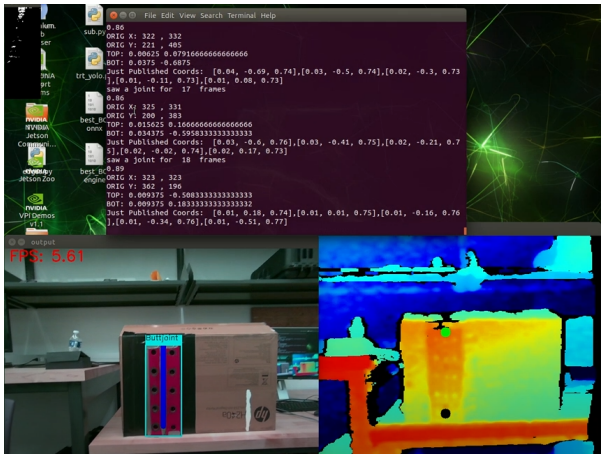


Fig. 8: Example Joint and Seam Detection from Learning Team

different background, the joint would not be recognized at times.
- If a joint was placed too far away from the view of the camera, it would not be recognized. We added data of the joint recorded from varying distances to account for this.
- At certain angles, T-joints were not recognized. In almost all instances, the butt joint was recognized.
- Jetson Nano was often slow in running segmentation models, which is why we opted to use an object detection model.
- If we place the Intel Realsense camera too close to the joint (or any surface), the depth cannot be detected.

B. Robotic Arm Operation (Arm)

The arm team is responsible for the direct operation of the arm based on the information provided by the UGV and learning teams. Based on the task at hand, the arm team is responsible for moving the camera in order to search for the seam, instruct the UGV team where to move and what direction to face, and to move along the seam with a weaving pattern in order to properly weld it.

Fig. 9: Jaco Arm attached to Clearpath Husky.

1) Arm Movement: Manipulation of the Jaco arm is performed using Kinova drivers inside ROS melodic using Ubuntu 18.04. The arm is run from a python program which controls the arm using the Cartesian control driver.

The driver is able to receive both relative and absolute positional coordinates, and angular coordinates in the form of radians, degree or quaternions. If these coordinates are within reach of the arm (generally less than 1m from the arm) and the arm is not in any kind of singularity, the arm moves to that position.

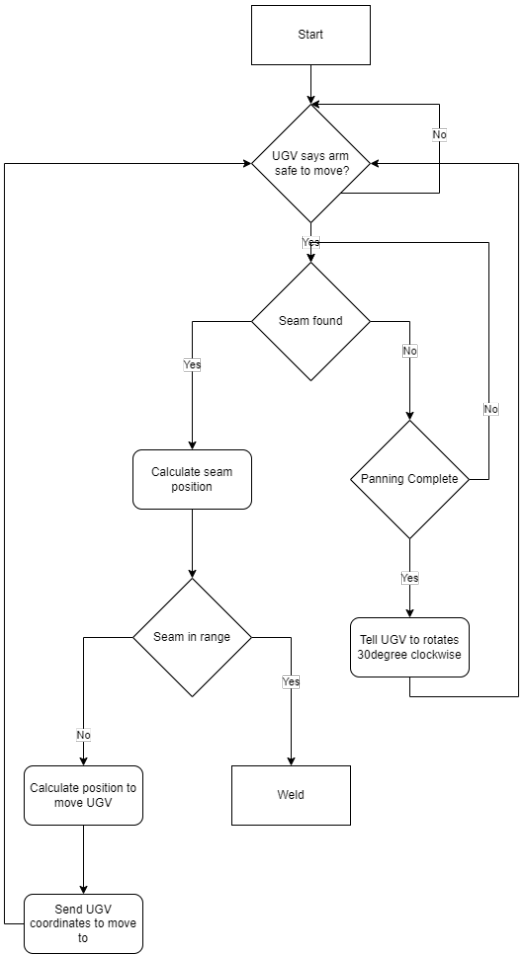


Fig. 10: Implemented arm process flow.

2) Scanning for a Joint: To ensure detection of the welding joint in the proximity of the Husky, we implement an arm sweeping motion to pan the camera around the scene where the UGV is located. This process steps the arm along the y-axis in small increments with a delay so that the camera and learning team can capture and process the image to check for welding joints. If the seam is found and within reach, the trajectory of the arm is then computed from the position of the joint and proceeds with the weaving pattern along the seam of the joint. If the seam is out of reach, the arm sends information to the UGV team telling it where to move. If none are found, the arm will tell the UGV to rotate by 30 degrees in the clockwise direction and then once again start its panning motion.

The stepping delay between points along the pan is large to allow for the limited frame-rate (¡6 fps) of the camera coupled with the learning team algorithm to process the frames. Work

was done to to ensure that this delay is minimized to allow faster working of the whole process.

3) Interpreting Positional Data from Learning Team: The program receives information from the learning team as two points x and y which are the positions of pixels on the screen from the camera relative to the center of the camera, and distance which is the distance of the object shown at pixel (x,y). The process to interpret this information goes in the following steps.

1) Calculate the angles the object is present at w.r.t. the direction the arm is currently facing. This is done by multiplying the x value with half the field of vision (FOV) of the camera in the horizontal space and multiplying the y value with half the FOV of the camera of in teh vertical axis. This gives us the rotation we need to perform along the z axis and y axis respectively in Cartesian space. To accommodate the conventional directions of rotation, the negative of the rotation along the z axis is taken.

$$rotation_z = -(x * (FOV_h)/2)$$

$$rotation_y = (y * (FOV_v)/2)$$

2) Trigonometry is used to calculate the distance the arm need to move in Cartesian space. The distance to be moved along the axes are $cos(rotation_z) * distance$ for the x axis, $sin(rotation_z) * distance$ for the y axis, and $cos(rotation_y) * distance$ for the z axis.

$$x_{distance} = distance * cos(rotation_z)$$

$$y_{distance} = distance * sin(rotation_z)$$

$$z_{distance} = distance * sin(rotation_y)$$

The arm can then be sent the command to move to these Cartesian by these distances relative to its current position.

4) Weaving: Proper welding technique is a key part of ensuring a weld will hold in the long run [2]. It is also important in controlling the flow of the filler into puddles based on the requirement of the joint and the environment and position the welding is taking place in.

We have successfully implemented 3 basic but rather universal weaving techniques known as the triangle weave,zig-zag weave and V-weave. These weaving technique can be used in both of the joints in consideration for this project, namely the butt joint and the T joint. Since, the Kinova arm does not have in-built libraries to regulate the end-effector motion in linear and circular paths, The code for this weave patterns was written from scratch in python and visualized using p5.js.

Current implementation of the weaving is such that the arm can only move in either the xy or yz planes and not in 3D space. Allowing weaving in 3D space will be attempted but may turn out to be unfeasible due to the high level of calculation and mathematics required to derive a formula for it.

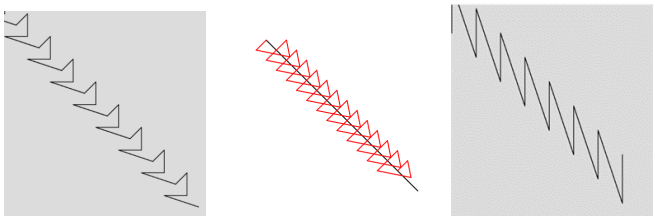


Fig. 11: Weaving patterns.

5) Issues Faced: List of past and ongoing issues:

- Hardware troubleshooting - Jetson Nano does not support Kinova Jaco drivers, Jaco arm requiring re-flashing and update of firmware, lack of precision in both actuators as well as sensors for detecting joint
- The arm cannot accept a trajectory and only accepts and works with a series of points.
- The somewhat unpredictable movement of the arm leads to spastic behavior at certain positions
- Arm rotation seems to vary based on end effector orientation or position, this makes it difficult to pan effectively and is largely why the panning consists of stepping in the y-axis rather than using angles.
- The above potentially be remedied by switching to a joint controller; however, as we use Cartesian control for the rest of the arm movement this would lead to physical delays in switching controllers and possible errors.
- All the implemented weaving patterns even though were working in terms of logic, we could not move the arm in the trajectory passed by the weaving pattern as it passed a large collection of points which was taking unusually longer to move in the path.
- The weaving patterns could only be implemented to take place in a single plane at a time and not in 3D space. This was due to the complexity of the vector math required and limited information from the learning team.

C. Visual Tracking and SLAM (vSLAM)

The UGV SLAM team is responsible for the autonomous navigation of the UGV using Visual SLAM. This is achieved using the Intel RealSense D435 camera mounted on top of the Clearpath Husky robot (refer to Fig. 12). We integrate the Husky and the RealSense camera in the ROS environment using URDF files, which define the transforms between the coordinate systems of the base (Husky) and the camera. The overall structure of the visual based SLAM solution for Husky can be summariez by the flowchart shows in Fig. 13. Following is a description of the Visual SLAM system.

1) Simultaneous Localization and Mapping: The Simultaneous Localization and Mapping (SLAM) module is responsible for generating a map of the environment and localize the robot with it. We use RTAB-Map, an open source software for this purpose. RTAB-Map is a computationally heavy system, because of which we use a laptop with Intel i5 9300H processor, Nvidia GTX 1650 graphics card, and 16GB of RAM as our computing platform. RTAB-Map also integrates very

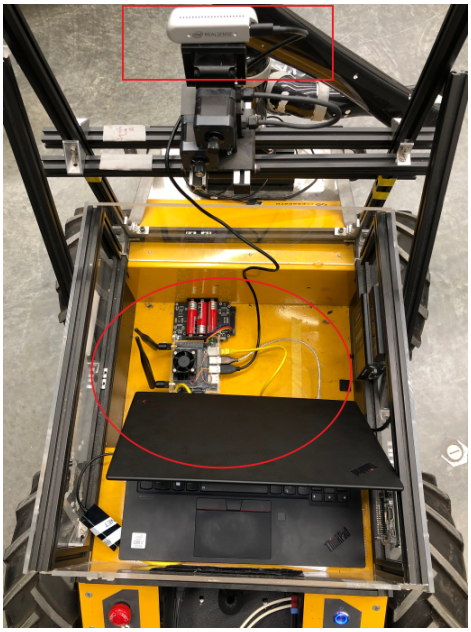


Fig. 12: The photo of husky platform. An Intel RealSense camera is mounted on the top of the UGV, with a Nvidia Jetson Nano that was planned to serve as the on-board computing platform. The Nano can be been replaced by a laptop to achieve better performance.

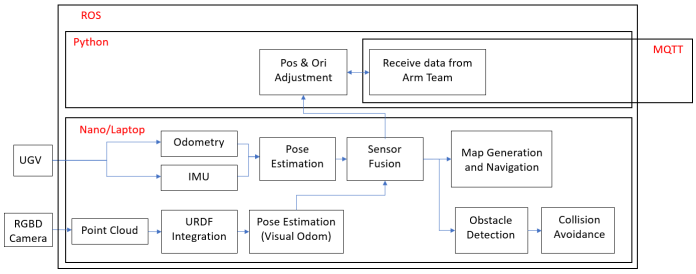


Fig. 13: Outline of vSLAM team proposed solution.

well with ROS and the navigation stack. Fig. 14 demonstrated the results of 3D and 2D mapping.

2) Sensor Fusion: We adopted extended kalman filter (EKF) [3] 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 environment.

3) Navigation: `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 map if the environment. It also uses the odometry information, which we generate by integrating the odometry from the IMU sensors of the Husky and the visual odometry

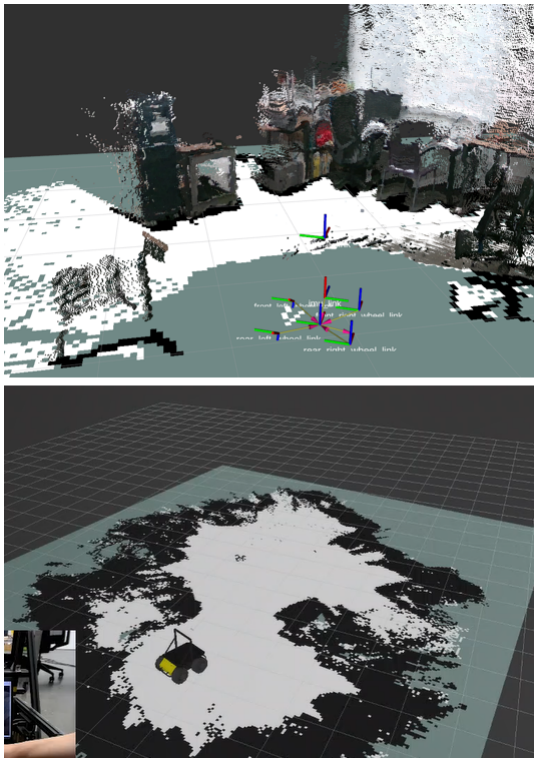


Fig. 14: Top: 3D mapping by point cloud. Bottom: Converted 2D mapping with binary pixles.

from the RealSense.

4) Position and Orientation Adjustment: For the cases where there are more than one welding sites in the vicinity, we have developed a script which specifically focuses on small adjustments in husky's position. This script is particularly important because during the welding operation, the robotic arm blocks the view of the Realsense camera. Hence, any movement during the welding operation relies on the arm's camera, which pans around for the nearby welding sites and the basic motion commands. The Arm with assistance from the learning team, calculate the coordinates of the target location and send it to UGV via MQTT communication. It consists of three numbers, the x, & the y-coordinate with respect to current position and the final orientation angle.

Once the husky receives the coordinates, we calculate the required angle and required distance to travel. Using the arctangent function with appropriate quadrant adjustments, and the distance formula, the required values are determined. Based on the calculated values, the husky first turns towards the target, and then, moves towards it.

5) Visual SLAM Performance Analysis and Discussion: We evaluate the performance of the Visual SLAM by validating the 3D dense map and the 2D Occupancy map. An accurate 3D map, and the localization of the Husky within it is essential for navigation. Figure 15 shows an accurate 3D dense point cloud generated by the Visual SLAM system. Figure 16 shows the 3D point cloud from the point of view of the Husky, which clearly shows the accurate point cloud and localization

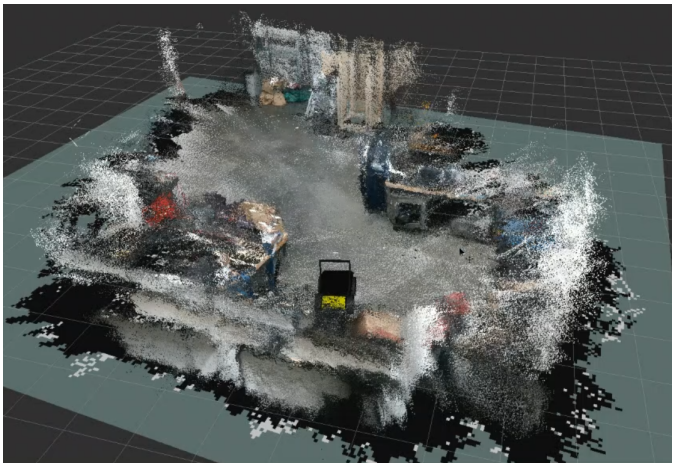


Fig. 15: 3D point cloud map generated by SLAM. Compared to Fig. 2, we could see that the 3D point cloud accurately reflect the real-life environment of the surrounding area.

Fig. 16: Results of the 3D map versus the real physical environment.

of the Husky within it. Figure 17 shows the 2D occupancy map, which is an accurate projection of the 3D dense point cloud, and the Husky navigation path within it. These tests validate the performance of the Visual SLAM algorithm with the Husky and Intel RealSense.

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. 18, 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. Mapping and Navigation Using a LiDAR (UGVLidar)

This section has a common purpose with the previous section, performing mapping and navigation of the UGV. However, this section focused on mapping using a LiDAR (Light Detection and Ranging), whereas the previous section used SLAM for mapping. To generate a map of the space, 2D RPLidar A2 was used in this section. Figure 19 shows the project outline of mapping, obstacle avoidance while navigation, and position and orientation adjustment after the navigation. In addition, Figure 20 depicts the overall diagram of the robot's objective destination and intended path.

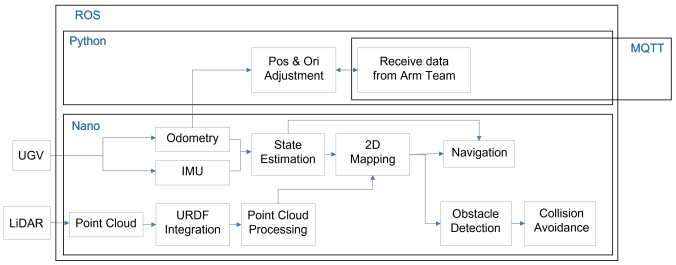


Fig. 19: Project outline of mapping and navigation using the LiDAR

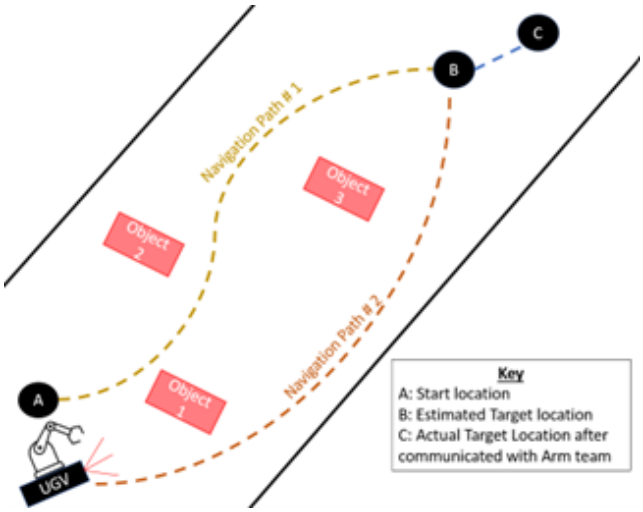


Fig. 20: Outline diagram of the overall objective during each stage of the robot's operation

1) 2D Map Generation with LiDAR: The environment in which the robot is to move autonomously must be mapped in a teleoperator mode first when the robot uses sensors to scan the environment. The robot then navigates to the specified waypoint, mainly according to the map. Since the LiDAR is attached to the Husky, both odometry and Inertial Measurement Units (IMU) data from the Husky are used for mapping with point clouds generated by the LiDAR, as shown in Figure 21.

2) Localization: After generating a map, the robot can be moved to any desired position in the generated map. First, Husky needs to be localized in the virtual environment on the generated 2D map. In this step, the viewing direction and position of Husky shown in RViz should be matched to the actual Husky machine. We assisted Husky in localization by giving it a rough idea of where it should be on the map using 2D pose estimate.

3) Navigation: Once the map has been imported into RViz and the Husky has been localized on the 2D map, it is ready

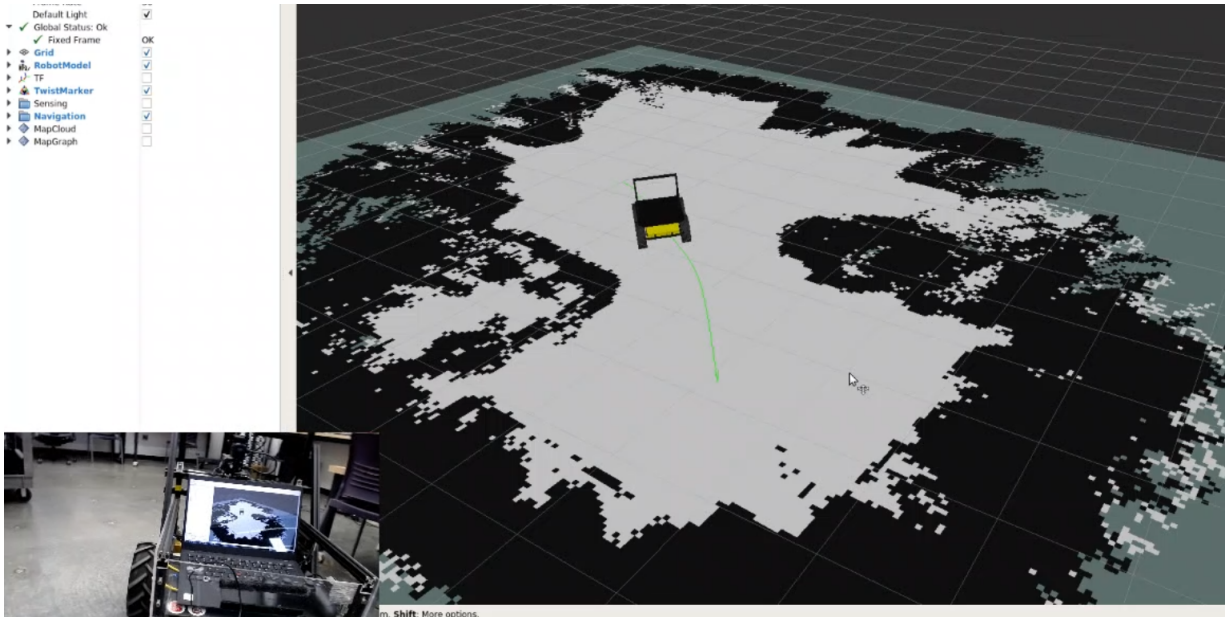


Fig. 17: The trajectory followed by the Husky during navigation based on visual-SLAM. The white portion in the 2D occupancy map denotes the region where the Husky is safe to move, and the black portion is the area inaccessible to the Husky (meaning the boundary of the map due to obstacles). The bottom 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.

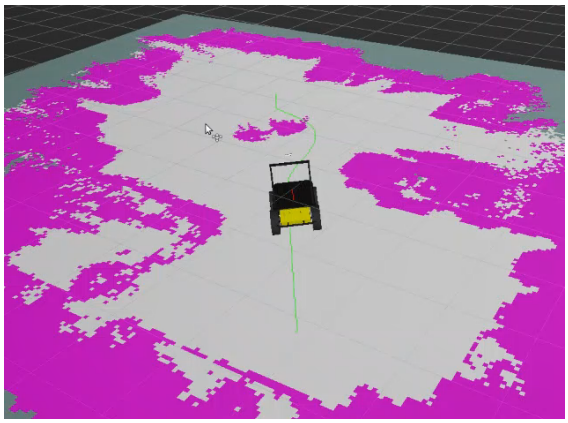


Fig. 18: Obstacle avoidance based on visual-SLAM. Figure shows the updated mapping and trajectory planning after putting an obstacle into the middle of path. The visual-SLAM algorithm is able to detect and map this obstacle, as well as plan a route avoiding the collision.

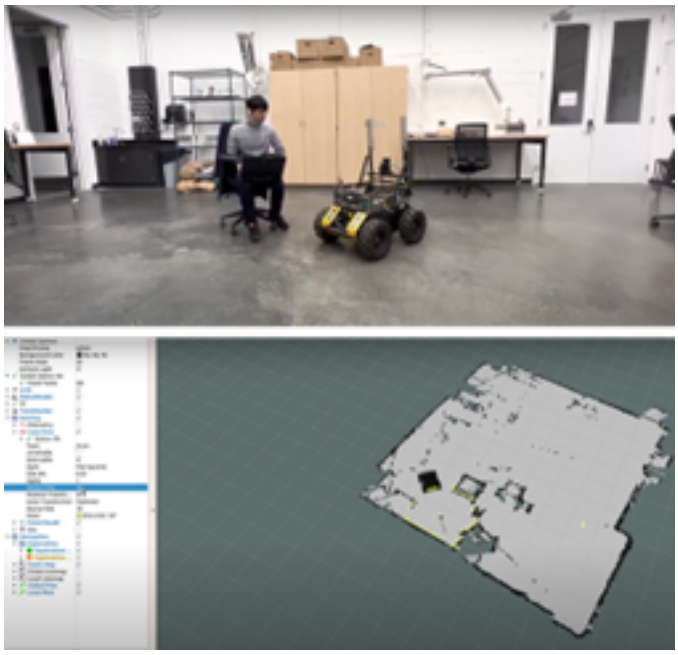


Fig. 21: 2D mapping using UGV and LiDAR

to perform navigation. The adaptive Monte Carlo localization (ACML), which uses a particle filter to track the pose of a robot against a known map, was implemented for automated navigation in our research. AMCL takes in information from odometry, laser scanner, and an existing map and estimates the robot's pose. During operation, AMCL estimates the transformation of the base frame with respect to the global frame but it only publishes the transformation between the global frame and the odometry frame. Once the goal pose and viewing direction are selected, Husky plans its trajectory and moves from the current pose to the selected goal pose using AMCL, as shown in Figure 22.

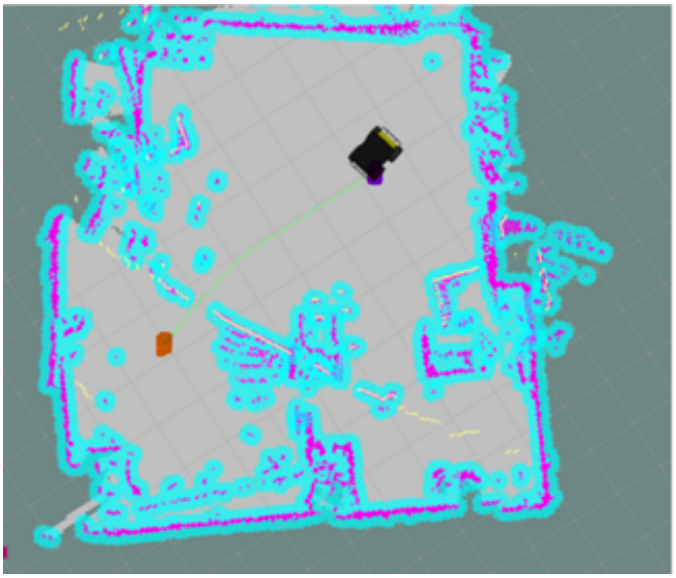


Fig. 22: The start point for the Husky is point A (denoted by the orange dot) and the end goal is point B (denoted by the purple dot). The Husky planned a path from point A to point B (green line) after receiving a 2D navigation goal

4) Obstacle Avoidance: Although the robot can be navigated to the specified goal point, it still needs to perceive its surroundings and in case of an incorrect location on the map or at the risk of collision with an obstacle. We used integrated reactive navigation for obstacle avoidance when the robot responds to conditions in the environment with its sensor system, including mapping data to increase safety. It is navigation using a local map of the environment with the lidar in this research. It detects an obstacle on both the local and global map which is a pre-generated map in the mapping procedure. When it detects the obstacle within a distance of 1.5 m on either a local or global map, it stops and rotates continuously at the same position to replan the route, bypass the obstacle, and continue according to the map to the specified destination. This process can involve several iterations if necessary. As can be seen in figure 23, successful test trials were carried out to test for both single obstacle avoidance and navigation.

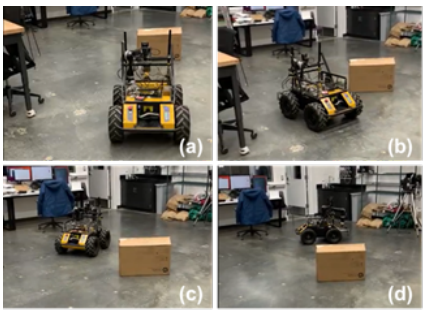


Fig. 23: Obstacle Avoidance with Sequential Steps from (a) to (d)

5) Position and Orientation Adjustment: Since the user needs to manually select the target position and orientation in RViz, there should be an error. Therefore, automatic position and orientation adjustment of the Husky system based on the Arm team's requirements is needed, as shown in Figure 24. First, desired target position (x, y) and orientation (θ) are sent by the arm team. Based on the received desired target position (x, y) data, the required angle as well as the distance to the goal are calculated. Then the Husky rotates based on the calculated angle at the same place, and goes straight based on the calculated distance. Finally, the angle to the desired orientation (θ) is calculated and the Husky rotates. Since there is no rostopic that we can directly use for position and orientation adjustment, we developed a new Python script using the rostopic for moving the robot at the desired speed to rotate a certain angle and reach the 2D coordinate (x, y). The limitation of the developed code is that it cannot go backward in our code. If the desired (x, y, θ) are (-1, 0, 0), the robot will rotate 180 degrees, go straight 1m, and rotate 180 degrees again. To add special cases, we need to spend more time on optimization.

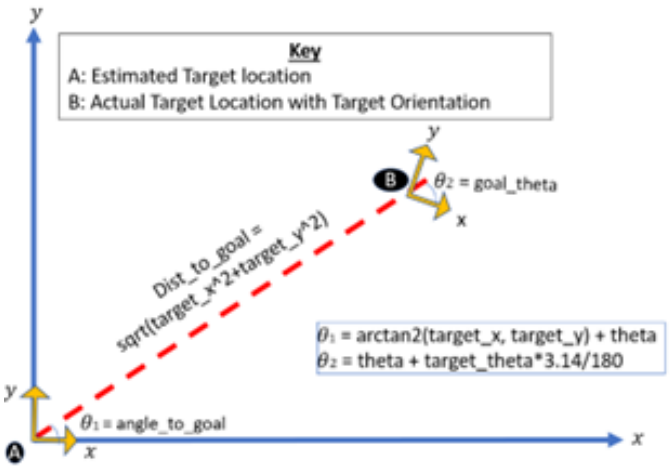


Fig. 24: Schematic Design of Automatic Position and Orientation Adjustment

REFERENCES

[1] A. Gómez-Espinosa, J. B. Rodríguez-Suárez, E. Cuan-Urquizo, J. A. E. Cabello, and R. L. Swenson, "Colored 3D Path Extraction Based on Depth-RGB Sensor for Welding Robot Trajectory Generation," *Automation*, vol. 2, no. 4, pp. 252–265, 2021.

[2] J. Grill. Welding beads: What are they? different types. [Online]. Available: https://weldguru.com/welding-beads/

[3] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," *Institute for Systems and Robotics*, vol. 43, p. 46, 2004.