DESIGN AND DEVELOPMENT OF AN AUTONOMOUS OMNIDIRECTIONAL HAZARDOUS MATERIALS HANDLING ROBOT

Nicholas Charabaruk[1], Scott Nokleby[1]
[1]*Mechatronic and Robotic Systems Laboratory, University of Ontario Institute of Technology, Oshawa, ON, CA*
Email: nicholas.charabaruk@uoit.ca; scott.nokleby@uoit.ca

ABSTRACT

This paper describes the design and preliminary testing of an autonomous omnidirectional robot to be used for moving radioactive materials while minimizing human exposure. The robot, called the OmniMaxbot, uses the Robot Operating System (ROS) to allow the individual components to communicate as well as to control the movement. The hardware and software used in the OmniMaxbot are also explained. While still in development preliminary testing has shown promise. More work is required before the OmniMaxbot is considered to be fully functional however.

Keywords: omnidirectional; autonomous; mechatronics; robot operating system (ROS).

1. INTRODUCTION

One of the main motivations for robotics is removing humans from environments that may cause bodily harm. That is the motivation behind this project. In the manufacturing of uranium fuel for the nuclear industry there is a process called Fluorination. This process involves mixing fluorine and uranium in a flame reactor to produce uranium hexafluoride. Approximately ten percent of the material is output as waste. This waste, in the form of uranium ash, falls into cans at the bottom of the reactors, where it is then removed by workers. This step carries the highest risk to workers of being exposed to radiation. The OmniMaxbot was developed in order to mitigate this risk. The OmniMaxbot is a fully autonomous robot for handling hazardous materials. It is omnidirectional, allowing it to move in the tight spaces between the reactors. This paper will explain the design and programming used for the OmniMaxbot. The remainder of the paper is laid out at follows: Section 2 explains some of the related literature, Section 3 gives a brief overview of the OmniMaxbot, Section 4 provides details on the hardware used, Section 5 explains the software used, Section 6 outlines the completed testing and their results, Section 7 discusses future testing, and Section 8 lays out the conclusions.

2. RELATED WORK

Automating material handling vehicles is not a new idea. One of the most common types of automonous vehicles used in manufacturing plants is the Automated Guided Vehicle, or AGV. According to Vis [1] AGVs were first introduced in 1955. There are many different types of AGV designs and navigation systems, based primarily on the work they are doing and the environment they must work in. AGVs are primarily used for repetitive material handling jobs.

There are many navigation methods available for autonomous vehicles. Berman and Edan [2] developed a method for navigating multiple AGVs using a fuzzy behaviour based navigation scheme. The navigation method was based on two cases, whether or not the AGV in questions was carrying a load. If the AGV had a load, then it would drive directly to the drop off location. If the AGV did not have a load, then it would decide which work station to service based on which station was closest to the AGV and the due time of the product. The AGV would then determine the shortest path between its current location and the goal workstation using the configuration space and the A* search algorithm. The path was represented by a series of waypoints. The AGV would follow the waypoints until it reached its goal. Right of way rules were implemented so that multiple AGVs could be used in one area. During the path planning stage, each part of the path between waypoint was given a classification. The classification decided if obstacles in that area were assumed to be another AGV or not. If not, the AGV would bypass obstacles detected in that area. If so, the AGV would wait to see if the obstacle attempted to bypass it. If, after a short wait, the obstacle did not move then the AGV would bypass it.

Hentschel et al. [3] developed a method of autonomous navigation for a forklift. The method is based on a series of predefined waypoints. A continuous-curvature path is built using line and polar spline segments. In addition to the path, a velocity profile is constructed for the fork lift. A graph-based routing algorithm was used to combine different routes to find the most efficient path. Using this method it was found that speeds of up to 1.7 m/s could be achieved.

Simultaneous Localization and Mapping (SLAM) is used in this project to build a map of the Omni-Maxbot's environment. There is a great deal of research being done in the area of SLAM. Tuna et al. [4] compared SLAM using three different map representations, each with three different localization methods. The map representations were occupancy grid map, ladmark-based map, and topological map. The localization methods were the Extended Kalman Filter (EKF), the Compressed Extended Kalman Filter (CEKF), and the Unscented Kalman Filter (UKF). The tests explored the relationship between map accuracy and CPU processing power for each of the localization methods. It was found that the CEKF worked best when

there are lots of natural and artificial landmarks.

Durrant-Whyte and Bailey presented a two part tutorial on SLAM [5, 6]. The first article started with a background of SLAM research. It goes on to explain SLAM in Bayesian form and its further evolution. They then go on to explain SLAM using two different methods, the EKF method and the Rao-Blackwellized Particle Filter (RBPF) method. The OmniMaxbot uses the RBPF method. The second tutorial by Durrant-Whyte and Bailey looks at some of the existing literature on SLAM with a focus on computation complexity, data association, and environmental representation.

3. SYSTEM OVERVIEW

The purpose of the OmniMaxbot is to pick-up cans of radioactive ash and move them to a waste handling location. This must be performed autonomously in order to limit the possibility of worker exposure to radiation. The OmniMaxbot must be omnidirectional in order to fit into the tight spaces it will be working in. In order to achieve omnidirectional motion, mecanum wheels are used. The OmniMaxbot lifts the cans using forks. The height of the forks is determined using InfraRed (IR) range finders. The cans are found using a camera detecting a series of AR codes attached to the cans. This is used to line the robot up with the cans and to determine when the cans are close enough to be picked up.

The OmniMaxbot uses laser range finders to build a map of its environment and for navigating within the map. It also uses a Microsoft Kinect to build a point cloud of the area hidden by the forks. This point cloud is converted into laser scans, also for use with the autonomous navigation system.

The start, pick-up, and drop-off locations are all stored on the OmniMaxbot. When it is activated, the map is loaded and the robot's initial position is set. The robot makes its way to the pick-up location using the software which will be described in Section 5. Once it is in position, the stereovision camera detects the can, lines up with it, and then approaches it. When the can is close enough, the forks are raised to lift the can off of the ground. The OmniMaxbot then drives to the can drop-off location, lowers the forks, and backs up until the camera determines that it is far enough to clear the forks. The robot then returns to its start location.

4. ROBOT SETUP

Figure 1 shows the completed OmniMaxbot. The frame of the robot is built using 80/20. This is extruded aluminium, giving the OmniMaxbot good strength for its weight. The forks are made of welded steel. The system is designed to lift loads of 500 kg.

As stated in Section 3, the OmniMaxbot uses mecanum wheels. These wheels consist of a hub wheel and a series of rollers which are at a 45° to the wheel axis. As the wheels turn, friction causes the rollers to turn. This provides a force perpendicular to the axis of the hub wheel. By using four of these wheels, each with its own motor, the OmniMaxbot can travel in any direction. An image of a mecanum wheel can be seen in Figure 2.

The OmniMaxbot uses a variety of sensors. The most prominent of these are the two SICK LMS100 Laser Range Finders (LRFs). Each sensor has a 270° scanning angle, 25-50 Hz scanning frequency, and a 20 metre effective range [8]. In order to achieve a 360° scan about the robot one LRF faces forward and one faces rearward. In addition to being used for map building, the LRFs are also used for navigation and obstacle detection.

Due to the location of the LRFs the forks interfere with the scans. This is even more pronounced when there is a can in the forks. In order to see the area blocked by the forks, a Microsoft Kinect sensor is used. The Kinect is mounted above the fork frame so that it will see past cans when they are in the forks.

In order to identify the cans and determine their position relative to the OmniMaxbot, a Point Grey Research BumbleBee2 Stereovision camera is used. The BumbleBee2 has two 0.8 megapixel cameras,

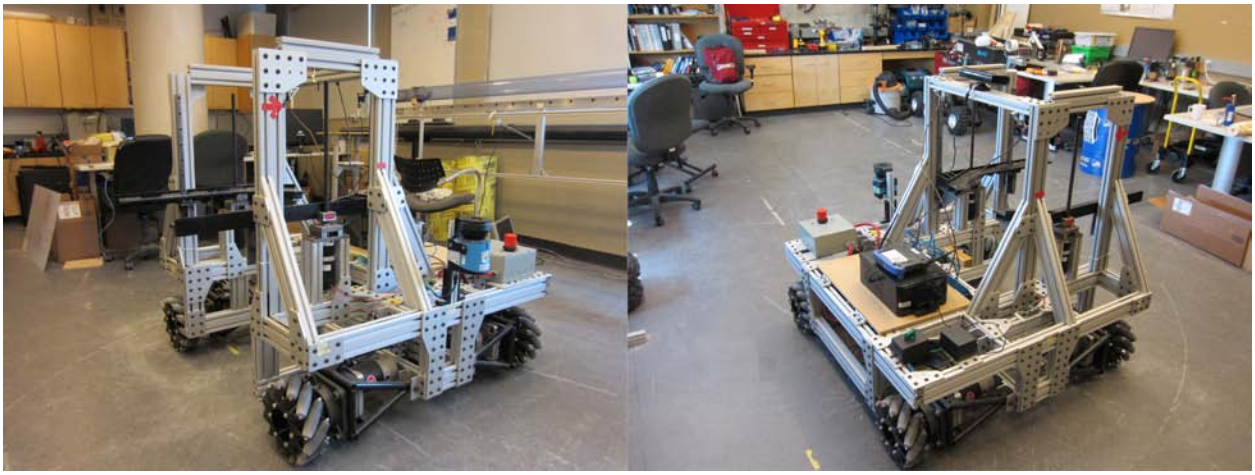

Fig. 1. The OmniMaxbot as seen from opposite corners.

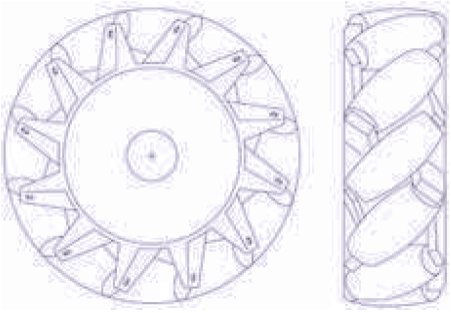

Fig. 2. A mecanum wheel [7].

3.8 mm focal length, and connects using a Firewire 1394a port [9].

To determine the height of the forks, there are two Sharp IR sensors. These are connected to an Arduino Uno board. As the board only needs to convert the raw sensor data into a distance value and send that to the laptop it was determined that nothing more powerful was required.

Several motors are required for the OmniMaxbot to work. Four NPC-T74 motors are used to turn the mecanum wheels. These motors run at 24-36 V and have a 20:1 gear ratio. Their performance parameters can be found at [10]. The motors are equiped with US Digital E5 optical encoders. These encoders provide 1,250 counts per revolution, or 5,000 pulses per revolution with quadrature. Further information on the encoders can be found at [11]. The encoders are used to calculate the odometry of the robot.

The T74s are controlled using two Roboteq AX2850 motor controllers. Information on these drivers can be found in the operations manual [12]. The AX2850s have two channels, therefore the front set of motors and rear set of motors are each run by a separate motor controller.

The forks are moved using Phidgets 2370 motors. These are controlled by Phidgets 1065 motor drivers. The forks are kept steady using linear rails as guides.

To improve the accuracy of the pose estimation, the OmniMaxbot is equipped with a CH Robotics UM7 Attitude and Heading Reference System (AHRS). The UM7 uses three axis accelerometers and rate gyros to provide data on the robot's movement [13].

Two computers are used to control the OmniMaxbot. The first is a computer mounted directly to the OmniMaxbot. This computer is used to receive sensor data, process the information, and perform the autonomous navigation. A workstation laptop is used to control the robot computer over a network from a distance. This laptop sends commands such as what programs to run, to control the deadman switch, and to teleoperate the OmniMaxbot when required.

5. PROGRAMMING

The OmniMaxbot uses the open source Robot Operating System (ROS) to allow the individual parts to communicate. According to the ROS website [14], ROS provides device drivers, packet management, message passing, and more. ROS is open source to allow robot developers to use software created by others in their own projects rather than having to reinvent the wheel. ROS Hydro was the version used in this work. Figure 3 shows the ROS nodes and topics used in the navigation system of the OmniMaxbot. Three basic parts of ROS are nodes, topics, and packages. Nodes are individual ROS executables. These are generally written in C++ or Python. Packages are collections of related nodes. Topics are where data is transferred. Nodes publish data to topics, or subscribe to topics to receive data from them.

Several ROS packages were used, either directly or in a modified form, for the OmniMaxbot. The standard ROS packages can be found on the ROS wiki [15]. The packages built specifically for the OmniMaxbot can be found on the following online public repository: https://github.com/orgs/mars-uoit/.

The `ax2550` package is used to control the AX2850 motor drivers. The package used for the OmniMaxbot is a modified version of the original version found at [16]. This package consists of four nodes. The *ax2550_front* and *ax2550_rear* nodes are used to send the speed commands to the motors and the encoder data to ROS. The *omni_cmd_vel* node is used to calculate what velocity each motor needs to turn at in order to move as needed. The inverse kinematic equations used for the OmniMaxbot were taken from Doroftei et al. [7]. These equations were modified slightly as they used a right positive y-axis where ROS uses a left positive y-axis. This node takes the desired overall linear and angular velocities as inputs and outputs the required speed each mecanum wheel needs to turn at in order to achieve those velocities. The final node in this package is the *omni_odom* node. This node takes in the encoder data from the motor drivers and publishes the robot's odometry.

One of the drawbacks of this system is that the wheel odometry is unreliable. This is because of how the

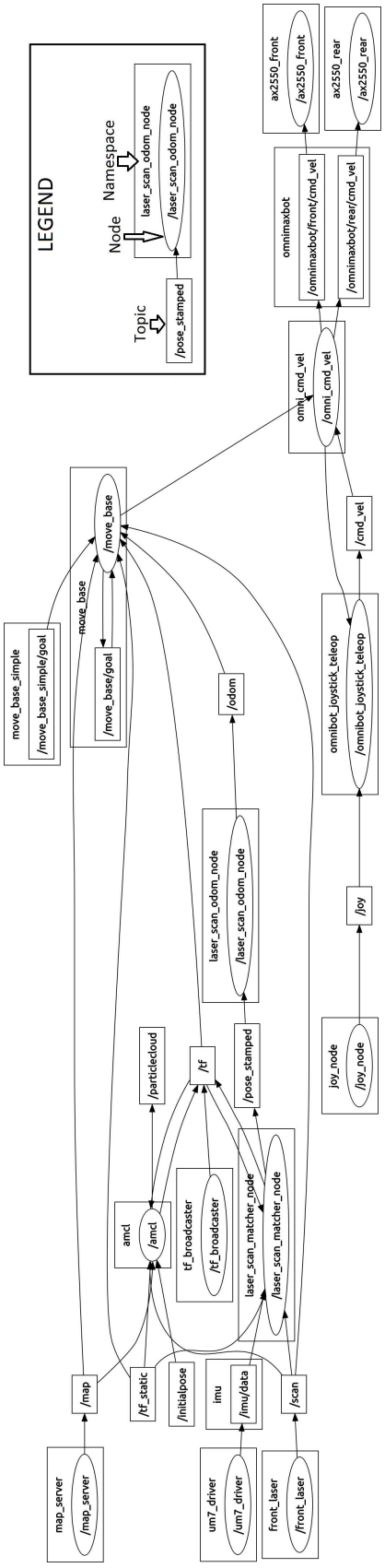

Fig. 3. The nodes, topics, and connections of the OmniMaxbot's navigation system.

meccanum wheels work. There is a great deal of slip with the wheels and the kinematics cannot accurately predict how fast the rollers rotate. Odometry is required for the autonomous navigation to work as the system needs to know what speed the OmniMaxbot is travelling at and in what directions. In order to overcome this limitation the `laser_scan_matcher` package was used. The documentation for this package can be found at [17]. Though the documentation is for ROS Fuerte the package has been updated to work with Hydro. The package works by comparing sequential laser scans. New scans are compared to the previous scan to determine how much translation and rotation there is. In order to minimize the effect of sensor noise a keyframes method is used rather than the basic scan-to-scan method. What this means is that rather than comparing each scan to the previous scan, one scan is taken, the keyframe, and subsequent scans are all compared to this. A new keyframe scan is taken every time the robot translates or rotates by a set amount. In this way small changes from sensor noise is ignored and only larger changes that can be attributed to movement are accepted. This package has two primary outputs. The first is a transform from the odom frame to the base_link frame. This is necessary for both the map making and autonomous navigation systems. It also outputs a geometry_msgs/PoseStamped message. This is a pose estimate of the robot relative to the map frame with a time stamp of when the estimate was made.

A new node was developed for the `laser_scan_matcher` package in order to estimate the velocity of the OmniMaxbot. This was required as the `move_base` package, explained later is this section, needs to know what speed the robot is travelling at in order to work. This is usually determined using odometry, but again that does not work well for this robot. In order to estimate the OmniMaxbot's speed in each direction the PoseStamped message output by the base `laser_scan_matcher` node is used. The PoseStamped message gives the OmniMaxbot's position and orientation relative to the map and a time stamp. Consecutive messages are compared to determine the changes in translation along the x and y axes, rotation about the z-axis, and time. The OmniMaxbot's velocities are determined by dividing the changes in translation and rotation by the change in time.

The LMS100s made use of the `lms1xx` package. This driver package was originally developed by the Robot Control and Program Recognition Group, now the Robot Programming and Pattern Recognition Group [18]. The package was rereleased and is now maintained by Clearpath Robotics [19]. The `lms1xx` package takes the laser scan data from the LMS100 and converts it into a sensor_msgs/LaserScan message, which is usable by other ROS packages.

The Kinect uses the `openni_launch` package in order to run. This package, found at [20], takes in raw data from the sensor and converts it into usable data. The output from the Kinect, in this case the rectified depth image and the camera information, is used by the `depthimage_to_laserscan` package [21]. This package takes in the depth image generated by the Kinect and converts it to a laserscan message. To do this the user specifies a height range, a minimum distance to consider, and a maximum distance. All of the values within the height range are looked at and the closest distance in each pixel column of the image is kept. If the distance is outside of the stated bounds, it is set to $\pm$inf, otherwise it is kept.

The `gmapping` package is used to build a map of the OmniMaxbot's environment. This map is then used as the basis for the autonomous navigation. This package was developed by Grisetti et al. [22, 23]. The `gmapping` package builds a 2 dimensional occupancy grid map using the laser scans, odometry, and AHRS data. As the name implies, the map is in the form of a grid. Each cell is given a value of either unknown, unoccupied, or occupied. An occupancy grid map of the Mechatronic and Robotic Systems (MARS) Lab at the University of Ontario Institute of Technology (UOIT) can be seen in Figure 4. This package uses a modified Rao-Blackwellized Particle Filter with adaptive resampling. More information on this package can be found on the ROS wiki [24] or on the OpenSLAM website [25].

To navigate autonomously the OmniMaxbot uses the `amcl` package [26]. This package uses an adaptive Monte Carlo particle filter with KLD-sampling. This method is described by Fox [27]. As inputs, the

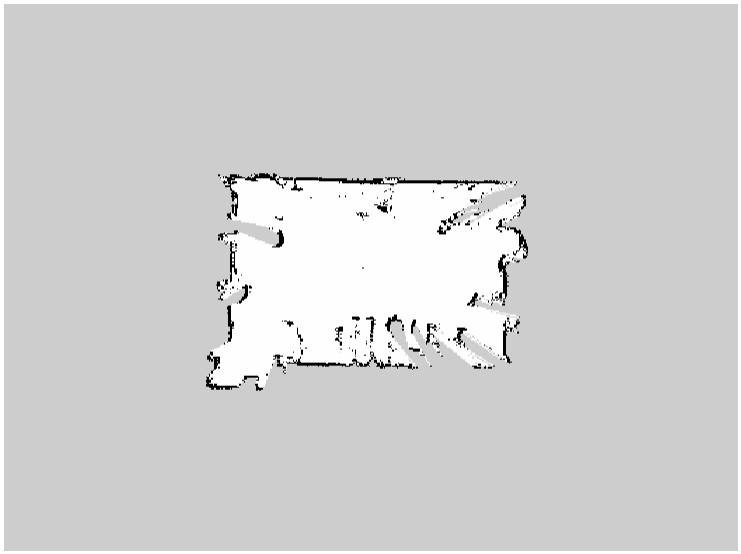

Fig. 4. An occupancy grid map of the MARS Lab built using the OmniMaxbot and the gmapping package. White cells are unoccupied, black cells are occupied, and grey cells are unknown.

package takes in the map of the robot's environment, an initial pose estimate, the transform tree, and the laser scans. It outputs a pose estimate for the robot, a transform between the map frame and the odometry frame, and the particle set from the filter.

As stated in Section 4, a Bumblebee2 stereovision camera was used to detect the cans and determine their position relative to the camera's position. The camera is controlled using the `camera1394stereo` package [28]. The `camera_calibration` package was used to calibrate the camera in order to get rectified images.

The `ar_sys` package is used to detect cans and estimate their positions. This is performed by attaching AR codes to the cans. The length of the side of the codes is used as a parameter for the package. The distance between the can and the camera is determined by comparing the known length of the side of the AR code to the length of the side as reported by the camera. The package takes the rectified image and camera information as input and outputs an image with the augmented data overlaid on it, the position and orientation of the can, and the transform between the camera and the can.

To control the Phidgets 1065 motor controllers, used to control the motors for lifting and lowering the forks, the *motor_controller_hc_1065* node of the `phidgets` package was used [29]. This node is a slight modification of the *motor_controller_hc* node. The main modification to the node was the inclusion of a breaking force for when the forks are holding a can.

The final package used was the `move_base` package. This package uses configuration data and sensor inputs to determine the path that the OmniMaxbot should follow to travel from its current location to the goal. It also determines and publishes what speed the OmniMaxbot should move at.

6. PRELIMINARY TESTING

The preliminary testing was performed in several stages. The first stage was to test the individual parts to ensure that they worked correctly. Once this was completed, subsystems were tested.

The drive system was tested first to ensure that the robot would move in a desirable fashion. First, commands were sent to the motors directly from the Ubuntu command line. These tests were performed while the OmniMaxbot was on jacks in case the robot reacted in an unpredictable manner. Once it was confirmed that the system was accepting and responding to the commands, the robot was lowered to the

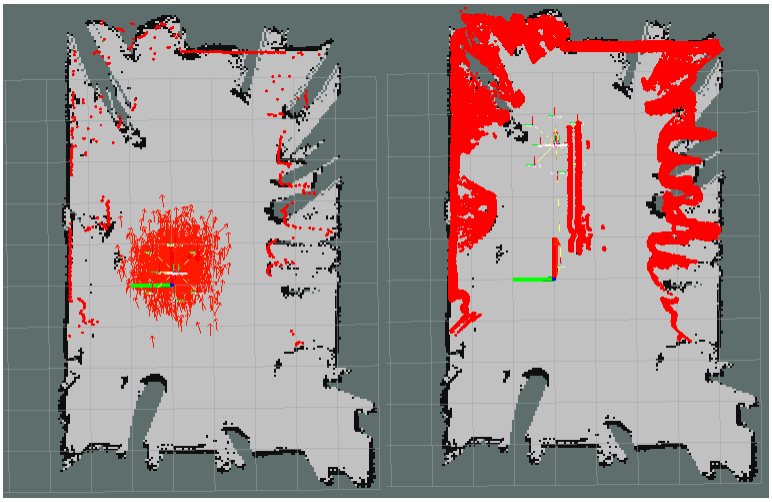

Fig. 5. The initial (left) and aggregated (right) scans from the linear odometry test.

ground. A joystick was then used to control the robot. These tests determined how well the OmniMaxbot followed changing instructions and to verify that the robot moved in the proper directions.

Once the robot was moving as expected the mapping functionality was tested. This involved driving the OmniMaxbot with the joystick while running the `gmapping` package to build the map. The result of this test can be seen in Figure 4.

The autonomous navigation was tested next. The OmniMaxbot was directed to move from its starting location to a point on the map. For this part of the testing the room was clear other than a single mock can. The results of this test were promising, but showed that more work is required. The OmniMaxbot was able to perform coarse navigation, but not the fine navigation required. This is believed to be due to error in the odometry data. Due to how the mecanum wheels work, there is a great deal of slip. This is not accounted for in the kinematic equations, making the odometry estimates inaccurate. Once the `laser_scan_matcher` package was added the odometry estimates improved greatly. To test the odometry data from the `laser_scan_matcher` package, two tests were performed. The first test was to check the translation. This involved driving the OmniMaxbot towards a flat wall. The laser scans taken while driving toward the wall were aggregated. Ideally, the aggregated scans should look like a single scan, though reasonably a few centimetres of drift is acceptable. The results of this test are shown in Figure 5. The second odometry test was to verify the rotational accuracy. This test involved taking an initial scan, rotating the OmniMaxbot by 360°, then taking a second scan. Ideally the scans would be right on top of each other, however a small about of error is acceptable. The results of this test are shown in Figure 6. Once these tests were completed the OmniMaxbot was again directed to autonomously make its way from a starting location to a goal location on the map. This time the OmniMaxbot reached its goal without issue.

Testing the `ar_sys` package with the BumbleBee2 camera and mock can has also been performed. It was found that this method works well for detecting the cans. There is approximately ± 5 cm error in the depth measurement and ± 1 cm of error in the horizontal distance measurement. The spacing of the OmniMaxbot's forks can accommodate this level of error in the measurements of the can's position.

7. FURTHER WORK

There is still some work required before the OmniMaxbot is at a usable level. More navigation testing is required to ensure that the robot reaches its goal every time and to optimize the `amcl` and `move_base`

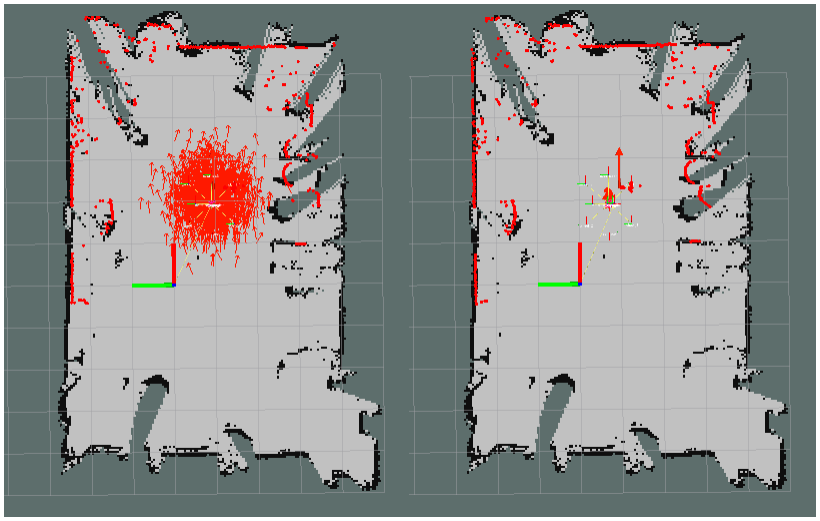

Fig. 6. Laser scans from before (left) and after (right) the 360° rotation.

configuration parameters. Once this is done, the OmniMaxbot will be tested in the same area but with obstacles added. This is to guarantee that it will be able to avoid obstacles and still make it to the goal location.

The final stage of testing will involve integrating all of the subsystems together into a final working robot. These tests will consist of the OmniMaxbot navigating its way to a pick-up location, approaching a can, lifting it up, navigating to a drop-off location, putting down the can, backing away from it without hitting it, and returning to its start location. Once the OmniMaxbot can perform this test repeatedly without error, the OmniMaxbot will be considered fully operational.

8. CONCLUSIONS

This paper outlines the hardware and software used to build the OmniMaxbot, an autonomous hazardous material handling robot. Preliminary testing is also explained. It was found that there is still much work to be done before the OmniMaxbot is able to perform its duties properly, however, the initial results are promising. Further work is required so that the OmniMaxbot will be able to achieve the goal of autonomously handling hazardous materials.

ACKNOWLEDGEMENTS

The authors would like to thank Cameco Corporation for their financial support of this research.

REFERENCES

1. Vis, I.F.A. "Survey of research in the design and control of automated guided vehicle systems." *European Journal of Operational Research*, Vol. 170, No. 3, pp. 677–709, 2006.
2. Berman, S. and Edan, Y. "Decentralized autonomous AGV system for material handling." *International Journal of Production Research*, Vol. 40, No. 15, pp. 3995–4006, October 2002.
3. Hentschel, M., Lecking, D. and Wagner, B. "Deterministic path planning and navigation for an autonomous fork lift truck." In "Proceedings of the 6th International Federation of Automatic Control Symposium on Intelligent Autonomous Vehicles," pp. 102–107. Toulouse, France, 2007.
4. Tuna, G., Gulez, K., Gungor, V.C. and Mumcu, T.V. "Evaluations of different simultaneous localization and

mapping (slam) algorithms." In "Proceedings of the 38th Annual Conference on IEEE Industrial Electronics Society," pp. 2693 – 2698. Montreal, PQ, 2012.
5. "Simultaneous localization and mapping: Part i.", June 2006.
URL http://ieeexplore.ieee.org.uproxy.library.dc-uoit.ca/stamp/stamp.jsp?tp=arnumber=1638022
6. "Simultaneous localization and mapping: Part ii.", June 2006.
URL http://ieeexplore.ieee.org.uproxy.library.dc-uoit.ca/stamp/stamp.jsp?tp=arnumber=1678144
7. Doroftei, I., Grosu, V. and Spinu, V. *Bioinspiration and Robotics Walking and Climbing Robots*, chap. 29. I-Tech Education and Publishing, 2007.
8. SICK AG. *Laser Measurement Systems of the LMS100 Product Family*, December 2009.
9. "Bumblebee2 0.8 mp color firewire 1394a 3.8mm (sony icx204).", February 2015.
URL http://www.ptgrey.com/bumblebee2-stereo-vision-08-mp-color-firewire-1394a-38mm-sony-icx204-camera
10. "Npc t74 motor.", January 2015.
URL http://www.robotmarketplace.com/products/NPC-T74.html
11. "E5 optical kit encoder.", January 2015.
URL http://www.usdigital.com/products/e5
12. Roboteq. *AX2550/AX2850 Dual Channel High Power Digital Motor Controller User's Manual*, June 2007.
13. CH Robotics. *UM7 Datasheet*, July 2014.
14. "Ros.", November 2014.
URL http://www.ros.org/
15. "Documentation.", December 2014.
URL http://wiki.ros.org/
16. Woodall, W. "ax2550.", August 2014.
URL http://wiki.ros.org/ax2550
17. Mottram, B. "laser_scan_matcher.", April 2015.
URL http://wiki.ros.org/laser_scan_matcher
18. "Home.", December 2014.
URL https://robotyka.ia.pw.edu.pl/twiki/bin/view/Main
19. Banachowicz, K. "Lms1xx.", January 2015.
URL http://wiki.ros.org/LMS1xx
20. Mihelich, P. "openni_launch.", January 2015.
URL http://wiki.ros.org/openni_launch
21. Rockey, C. "depthimage_to_laserscan.", January 2015.
URL http://wiki.ros.org/depthimage_to_laserscan
22. Grisetti, G., Stachniss, C. and Burgard, W. "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling." In "Proceedings of the IEEE International Conference on Robotics and Automation," pp. 2432–2437. Barcelona, Spain, 2005.
23. Grisetti, G., Stachniss, C. and Burgard, W. "Improved techniques for grid mapping with rao-blackwellized particle filters." *IEEE Transactions on Robotics*, Vol. 23, No. 1, pp. 34–46, February 2007.
24. Gerkey, B. "gmapping.", January 2015.
URL http://wiki.ros.org/gmapping
25. "Gmapping.", January 2015.
URL https://www.openslam.org/gmapping.html
26. Gerkey, B. "amcl.", February 2015.
URL http://wiki.ros.org/amcl
27. Fox, D. "Kld-sampling: Adaptive particle filters and mobile robot localization." Tech. Rep. UW-CSE-01-08-02, University of Washington, 2001.
28. Beltran, J.P. "camera1394stereo.", February 2015.
URL http://wiki.ros.org/camera1394stereo
29. Mottram, B. "phidgets.", January 2015.
URL http://wiki.ros.org/phidgets