Development of a Mobile Robotics Platform for Three Dimensional Mapping

Matthew King and Oscar De Silva
Faculty of Engineering and Applied Science
Memorial University of Newfoundland
mpik32@mun.ca oscar.desilva@mun.ca

***Abstract*— This paper presents the design, implementation, and realization of a mobile robotics platform for the purpose of 3D mapping and also to enable other research work. The system described is composed of two major components. The first is the Seekur Jr unmanned ground vehicle (UGV). Second is the tilt-laser unit which is used to generate a 3D pointcloud. The system is integrated into a cohesive unit using the Robot Operating System (ROS). System validation was carried out through a series of trials on the first floor of the MUN engineering building. Results include 3D maps produced both from the Seekur Jr. and from a stationary platform. The custom control system used for the Seekur Jr. is distinct from available ROS packages because its architecture requires little to no modifications to be made on the robot's integrated computer. The package is available online through github.**

***Index Terms*— Robotics, 3D Mapping, Lidar, ROS.**

I. Introduction

The production of quality three dimensional maps of arbitrary environments is an active field a research with many potential applications such as autonomous vehicles, mining, search-and-rescue, architectural inspection, and augmented reality. Significant results have been achieved using stereo vision systems [1], structured light systems similar to the Microsoft Kinect [2], and expensive 3D Lidar systems [3]. Instead, this paper presents an approach where a much more affordable Lidar unit is mounted on a tilting servo which allows the scans to be projected into three dimensional space. This, in combination with a robust and stable localization system provided by the Seekur Jr. mobile robot, allows scans to be combined together into a map. Seekur Jr. Robot localization is still very much an active area of research and so the search for accurate localization is of equal importance to the operation of the laser unit itself.

System integration is implemented though the Robot Operating System (ROS). ROS is a middleware which handles communications, synchronization, and provides a unified development ecosystem with a wide range of open-sourced software packages available for use.

II. Robot Operating System

The robot operating system, henceforth referred to as ROS, is an extensive open-source package which enables the development of software for robotics systems [4]. It is not a true operating system but instead a 'middleware' that creates a structured communication layer which supports data exchange through asynchronous processes while providing centralized organization [4]. Each process must register with the ROS master process upon startup as a 'node'.

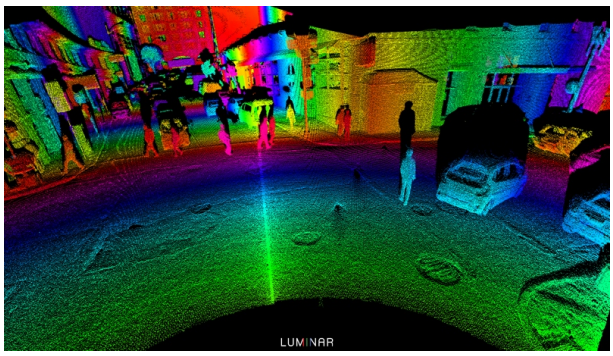

Fig. 1. Lidar is a key technology for autonomous vehicles.

Thanks to its flexible architecture and continued development, a significant open-source ROS community has formed. This leads into another major advantage of ROS which is the widespread existence of so many freely available hardware drivers, packages, and libraries which may be easily integrated into one's own robotics system with unmatched ease. ROS also boasts an extensive wiki with a wealth of documentation that adds significant value while further increasing ease of development.

A. Topics and Messages

After a node has registered with the ROS master, it may proceed to notify the master of its intent to publish or subscribe to a certain 'topic'. When two processes register themselves to the same topic - one publishing to the topic and the other subscribing to the topic - then the master will put these two nodes into direct contact and allow them to begin transferring data. Multiple nodes may publish to or subscribe from the same topic.

Each topic has a predefined message type. Such formats could be simple primitive types such as integers or complex as a detailed data structure for describing laser scan data, odometry, GPS data, and many more. Many message formats are provided by existing packages and it is easy to define custom ROS message types. All message are serialized and transferred over a TCP/IP protocol known as TCPROS.

B. tf: The Transform Library

In robotics applications it is often necessary to keep track of multiple coordinate frames. This is such a common issue that the tf library has become a core component of the ROS ecosystem [5]. Using tf, the set of all current coordinate frames are tracked in a tree structure, providing the means to calculate transforms between any two connected frames. The transform between any two connected frames may be quickly found and applied where necessary, and adding new frames is as easy as instantiating a new publisher. Almost all of the standard ROS message types must have some associated frame which represents the coordinate system from which they have been sampled. This allows data to be easily transformed between coordinate frames with relative ease.

III. Tilt-Laser Unit

At the core of this system is the scanning laser rangefinder itself. The tilt-laser unit core components consists of a Hokuyo UST-20LX scanning laser rangefinder, a Dynamixel AX-18A servo actuator, an Ordoid-XU4 single board computer, a power distribution board, and chassis. Further accessories include a SMPS2Dynamixel Adapter and a Dynamixel TTL USB adapter.

The servo is mounted to the main chassis and fitted with a custom mount upon which the UST-20LX is secured. By maintaing precise control of the servo posititon, the incoming 2D scan data is projected into 3D pointclouds relative to the axis of rotation. See Figure 2 for an image of the unit.

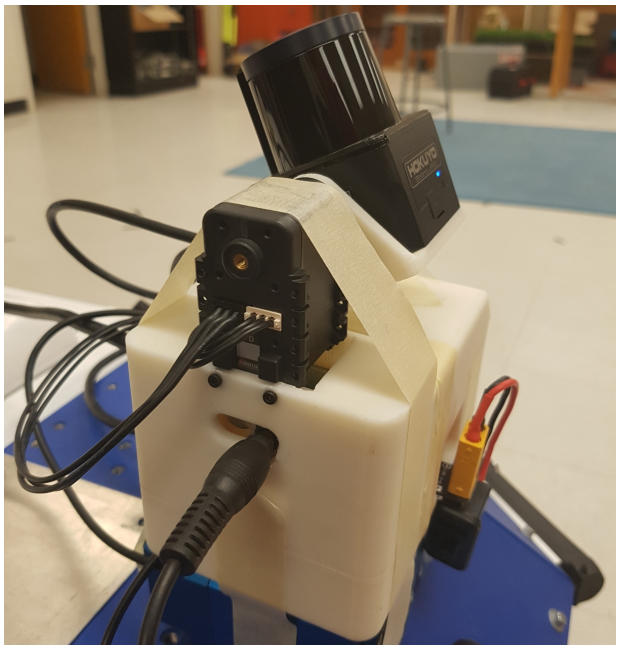

Fig. 2. Side view of the assembled unit

A. Hardware Components

1) Hokuyo UST-20LX: A scanning laser rangefinder is a common mid-range Lidar unit for academic and industrial applications. The device is classified as a Type 1 laser product and as such is safe for use without eye protection. It has up to a twenty meter range, two-hundred and seventy degree field of view (FOV) with a quarter-degree angular resolution [6]. The sensor uses time-of-flight technology to determine distance to target and also reports the intensity of return which can be useful for surface detection.

Internally, there exists a small mirror which rotates about a central axis. This mirror completes its trajectory through the sensor's FOV at a rate of 40Hz or equivalently 2400 RPM. The laser beam is bounced off this mirror and it is through said mechanism that the distance readings are obtained.

Repeated readings can differ by small amounts. Extensive calibration has shown the standard deviation of repeated measurements statically sampled from a four meter distance to be 4.7 millimeters [7]. This error may be expected to increase as a function of the true distance to the target reading; however it may decrease as a percentage of the total distance. Objects smaller than 1.5 degrees in horizontal size may not be detected unless repeated sampling is used. Additionally, the device is not sensitive to local lighting conditions and thus is suitable for both indoor and outdoor environments.

The UST-20LX communicates via an ethernet cable using the SCIP protocol. Once connected to a computer the device creates a LAN with a pre-defined IP. From the computer it is then possible to reconfigure the device if desired before beginning receiving scan data.

2) Dynamixel AX-18A: This device is a lightweight smart servo with a built-in microcontroller that allows it to communicate over a bus. The AX-18A takes advantage of its rapid digital communication to receive goal position commands while also providing feedback on variables such as current position, torque, and more. Especially, the current angular position data is of significant importance for this application. This position has a resolution of 0.29 degree angular resolution [8] and can be obtained at sufficiently high frequency so as to outpace the rate at which scans are produced by the UST-20LX.

3) Odroid-XU4: This single board computer boasts significant power in an efficient form factor. With an Octa-core CPU and 2GB RAM, the XU4 is more than fast enough to process laser data, interface with the Servo network, and produce pointcloud data without any slowdowns.

B. Software

The software being used to control this system is a mixture of open-source pre-existing nodes in addition to custom written applications. The requirements are as follows.

1) Receive Scan Data: To accomplish this task, the pre-existing URG node was deployed. After providing the UST-20LX's IP address the node automatically connects to the device and begins publishing laserscan messages on the scan topic at a rate of 40Hz.

2) Dynamixel Servo Controller: A new ROS node was developed which leveraged the Dynamixel C++ SDK to handle certain details such as the specific communication protocol. Even though controller software already exists, none published angular position data at a rate which could match the Lidar data.

Accurately knowledge of the angular position is crucial in order to generate the transform between incoming scan data and the frame where all scans are accumulated into pointclouds.

3) Computing the Transform: It is necessary to define two frames in order to describe the relation between the servo and the laser scanner. The first frame, "/laser" describes the center point of the laser scans and is coincident with the point at which the mirror internally rotates inside the UST-20LX. All incoming laser scans are said to be in the "/laser" frame. The second frame, "/laser_link" is the reference point which is concentric to the servo's axis of rotation and centered with the Lidar.

These frames are shown in Figure 3 at a relative angle of zero degrees. As the servo rotates the laser, the Z axis of the "/laser" frame remains radial relative to the Y axis of the "/laser_link" frame. Their radial offset is approximately 2.54 centimeters. The position of the origin of the "/laser" frame in reference to the "/laser_link" is computed as:

$$P(\theta) = \begin{vmatrix} x \\ y \\ z \end{vmatrix} = \begin{vmatrix} r * \cos\theta \\ 0 \\ r * \sin\theta \end{vmatrix} \quad (1)$$

4) Filtering: Several scan filters are run on the raw data before it is further processed. Filters are derived from the ROS laser_filters package. Here, a shadow removal filter is applied to remove bad scans which are the result of a scan falling directly on the edge of an object such that the scan splits and is interpreted to be halfway between the foreground edge and whatever background surfaces are behind. The readings are then passed through a second filter which replaces these bad filters with linear interpolations based on nearby 'good' readings.

5) Conversion to Pointcloud: Now given the incoming filtered scan data, as well as the transform between the laser scanner at that time and the reference frame, it is now possible to convert from laser scans, which are of the form:

$$\begin{vmatrix} r0 & r1 & r2 & ... & rn \\ \theta 0 & \theta 1 & \theta 2 & ... & \theta n \end{vmatrix} \quad (2)$$

to the more generalized pointcloud format, which is of the form:

$$\begin{vmatrix} X0 & X1 & X2 & ... & Xn \\ Y0 & Y1 & Y2 & ... & Yn \\ Z0 & Z1 & Z2 & ... & Zn \end{vmatrix} \quad (3)$$

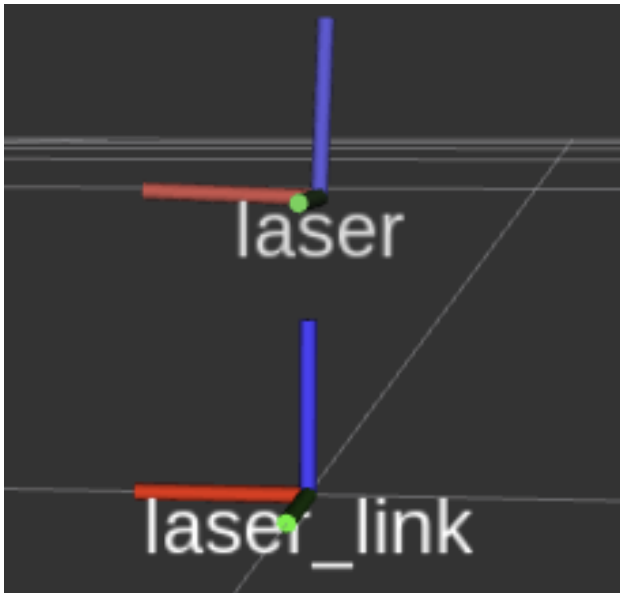

Fig. 3. Laser frames at zero angle (X axis in red. Y axis in green. Z axis in blue)

In order to enact this conversion, the laser_geometry package was used. The c++ implementation of laser_geometry provides a pair of methods for converting laser scans into pointclouds; one method is faster but less accurate and the other is slower but is more accurate. Both methods requires the scan message and additionally requires an available tf for the beginning of each scan based on the message timestamp. The second and more accurate method also requires there to exist a second transform at the end of the scan period, which it determines based on the scan message's time increment. The method linearly interpolates between the two transforms so that each point on the scan, having been captured at slightly different times, is transformed with respect that time difference [9]. The more accurate of the two methods was chosen to be the best option for mapping applications. This becomes more evident as the servo's speed is increased and the skew starts to become more noticeable within a single scan. This function is being called within a new custom node which subscribes to the scans being published by the URG node and utilizes the transforms published by the custom Dynamixel controller in order to perform this operation.

IV. Seekur Jr.

Stationary results are useful but are not sufficient for full mapping of an environment. As such, it is useful to combine the tilt-laser unit with a mobile robotics platform. For this purpose, the Seekur Jr. robot was chosen.

A. Description

The Seekur Jr. is a four wheeled skid-steer robot. It can come equipped with a wide range of sensors including stereo camera, ultrasonic range finders, IR bumpers, and even it's own foward mounted scanning laser range finder. Other accessories include Wifi router, robotic arm, and more. The robot

is 1.1 meters long, 0.8 meters wide, and weighs 77 kilograms [10].

B. Software Architecture

Seekur Jr. is part of a larger family of robots originally developed by the Mobile Robots company. These robots came shipped with an extensive software platform by the name ARIA which enabled the control of the robots and was extendable for further research and development. However, the product line is no longer supported and is not directly compatible with ROS.

1) ROS Integration: Therefore, it was deemed desirable for a new set of software to be developed which could interface the original software stack into the ROS ecosystem. Towards this purpose, the AriaClientDriver was developed.

ARIA is designed as a client-server model. The server runs on the robot itself and communicates with the client via internet connection. The client then provides commands to and requests data from the server.

In order to integrate this system into ROS, it was decided to implement the client-side functionality into a ROS node. This has the advantage that one need not install ROS onto the Seekur Jr. itself, as it can be very difficult to install on older systems and may cause hard to repair issues in case of a failed installation.

This ROS node provides wheel odometry data, laser scans, GPS data, and more. The node also provided scan filtering and allows one to steer the robot directly from the terminal window.

2) Localization: In order for the Seekur Jr. to be utlized as a mobile platform for integration with the Tilt-Laser unit it must be able to accurately localize itself within its environment. Localization allows the robot to know its approximate location relative to where it has been before. An extension of the localization problem is known as simultaneous localization and mapping (SLAM). An existing SLAM node known as GMapping was selected for use. This node merges sensory data from the wheel odometry and the (2D) laser scanner in order to produce an estimate of its location. The internal implementation involves a particle filter which takes into account both recent movements and observations [11]. Simultaneously, the program produces a 2D occupancy map based on laser data. See Figure 4 for a map of the Engineering building first floor produced using Gmapping and the Seekur Jr.

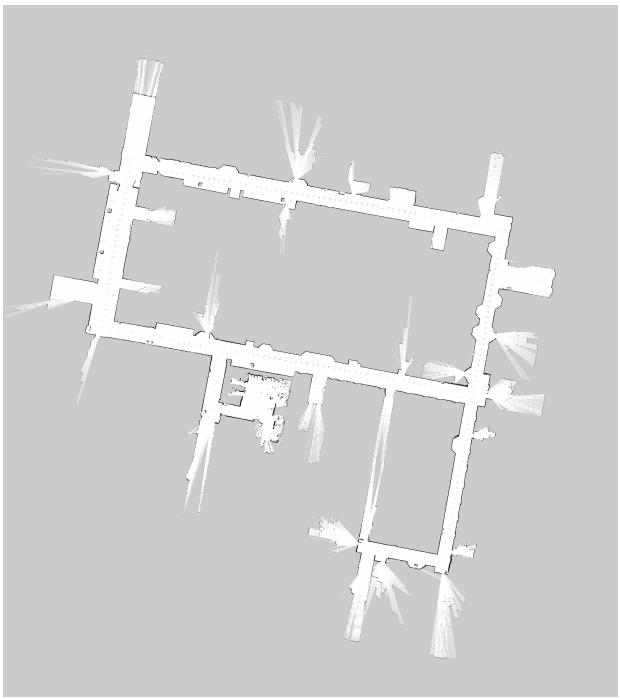

Fig. 4. Map of the MUN Engineering building first floor. Only main hallways included in addition to EN-1037.

V. Results

A. Stationary Results

The pointcloud data presented in Figure. 5 was collected with the scanner kept stationary at a single position. The scanner was allowed to complete one sweep of the environment. The intensity data, which is preserved throughout the filtering and conversion process, clearly shows the different intensity returns of a black and white checkerboard pattern. A window is visible in the far corner and the outline of a tripod is also visible by the shadow cast onto the wall. The position of the scanner is indicated by a small coordinate marker.

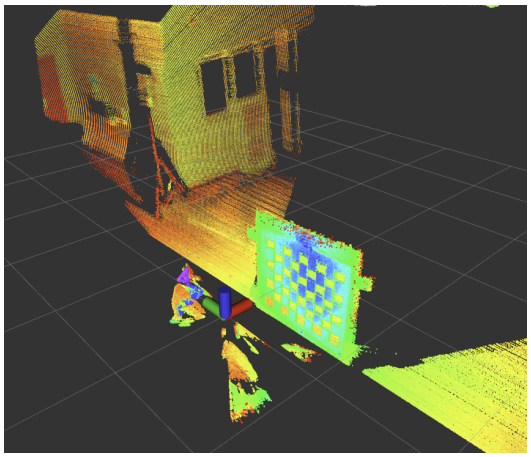

Fig. 5. Sample of results of the pointclouds collected over a single sweep. Here, the tilt-laser unit is kept stationary on a desk.

B. Mobile Results

The next sample represents the ultimate amalgamation of the work that has been described up to this point. Here, the tilt-laser unit has been mounted on the Seekur Jr. The tilt-laser unit has been configured as a ROS slave, which means that it is no longer running its own ROS core but instead is subservient to another computer. The pointclouds

are being published through the local Wifi network to a remote computer which is also in control of the Seekur and running the AriaClientDriver. The incoming pointclouds are being transformed into the global map frame using the localization being produced by Gmapping. Overall map quality is acceptable but sudden jumps in Gmapping's estimated position cause some discontinuities. Please refer to Figure 6 for more details. Additionally, please see Figure 7 for another map where some post-processing has been performed.

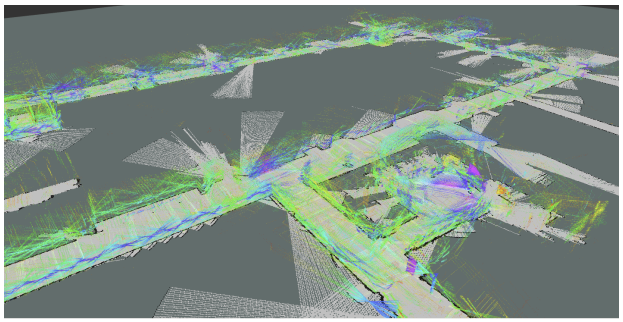

Fig. 6. MUN Engineering First floor. 3D map raw pointclouds overlaid onto 2D map.

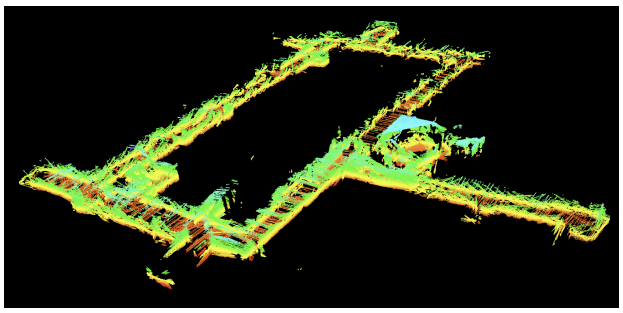

Fig. 7. MUN Engineering first floor. Pointcloud after post-processing to remove statistical outliers.

VI. Conclusions

The project was successful and accomplished the initial goal of mapping an indoor environment. Three dimensional mapping is an ongoing research topic and tilting a 2D Lidar is a valid approach to this task. The advantages of this approach include lower cost and higher customizability when compared to full 3D Lidar. Disadvantages include higher complexity and, in some cases, lower points-per-second.

Additionally, the integration of systems using the ROS ecosystem was found to be significantly advantageous especially when compared to building an entirely custom system. Using ROS allows for rapid prototyping, faster troubleshooting, and provides a wide range of open-source nodes.

Overall, the largest issue that was encountered in this project was the localization which was used to merge the pointclouds into a singular map. The localization produced by Gmapping was often inaccurate and would make sudden jumps or would not update at a sufficiently high rate. This led to discontinuities in the map and caused strange behaviour. These sudden jumps are investigated as a follow up project using Lidar odometry methods which allows to jointly solve the robot localization and point cloud registration problems [12].

Another issue which could use more work is that when a pointcloud based 3D map reaches a certain size the total number of points starts to become so large that loading and visualizing the map starts to become an issue. One way to solve this issue might be to integrate the data into a more efficient data structure for representing 3D space. This could be accomplished using an octotree structure [13] which represents space in variable-size cubes using a probabilistic model to insert laser scans into the map. This method would be significantly more memory efficient at the cost of potentially slower runtime. This might also prove useful as an aspect of the aforementioned localization process.

Acknowledgment

The authors would like to thank NSERC for providing funding for the work documented here.

References

[1] J. M. Saez and F. Escolano, "A global 3d map-building approach using stereo vision," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 2, pp. 1197–1202 Vol.2, April 2004.

[2] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, pp. 177–187, Feb 2014.

[3] M. Pierzchaa, P. Gigure, and R. Astrup, "Mapping forests using an unmanned ground vehicle with 3d lidar and graph-slam," *Computers and Electronics in Agriculture*, vol. 145, pp. 217 – 225, 2018.

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[5] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pp. 1–6, April 2013.

[6] Hokyuo, *Hokuyo UST-20LX Scanning Laser Rangefinder*, 2016. Available online at: https://www.hokuyo-aut.jp/search/single.php?serial=167.

[7] M. Cooper, J. F Raquet, and R. Patton, "Range information characterization of the hokuyo ust-20lx lidar sensor," vol. 5, 05 2018.

[8] Robotis, *Dynamixel AX-18A e-manual*, 2018. Available online at: http://emanual.robotis.com/docs/en/dxl/ax/ax-18a/.

[9] T. Foote and R. B. Rusu, "Laser geometry api documentation."

[10] "Seekur jr. overview." Available online at: http://www.mobilerobots.com/Libraries/Downloads/SeekurJr-SKRJ-RevA.sflb.ashx.

[11] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *Trans. Rob.*, vol. 23, pp. 34–46, Feb. 2007.

[12] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," in *Proceedings of Robotics: Science and Systems*, (Berkeley, USA), July 2014.

[13] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013. Software available at octomap.github.com.