LOW-COST SENSOR PACKAGE FOR SMART WHEELCHAIR OBSTACLE AVOIDANCE

CHAD A ROCKEY

Submitted in partial fulfillment of the requirements

for the degree of Master of Engineering

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

January, 2012

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

_____Chad A. Rockey_____

candidate for the _____Master of Engineering_____degree *.

(signed)_____

(chair of the committee)

_____

_____

(date) _____

*We also certify that written approval has been obtained for any proprietary material contained therein.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank Invacare for their donation of “Otto” and their generous support that enabled my research.

LOW-COST SENSOR PACKAGE FOR SMART WHEELCHAIR OBSTACLE AVOIDANCE

Abstract

CHAD A ROCKEY

This thesis shows the development of a low-cost sensor package to modify an electric wheelchair into a smart wheelchair capable of preventing its operator from directing the vehicle into obstacles. To do this, novel sensors such as the Neato XV-11 laser scanner and the Microsoft XBOX 360 Kinect are used to create efficient 3D maps of the surroundings known as Octomaps.

I. Introduction

Smart wheelchairs and self-driving cars are some of the most desired market applications of robotics technology. Whereas the autonomous vehicles descend from outdoor navigation robots that rely on GPS, smart wheelchairs are a logical extension of indoor navigation technologies that rely upon precise navigation and mapping of indoor environments. These technologies could greatly improve the life of wheelchair bound patients, but the cost and safety of smart wheelchairs needs to improve before they are commonplace.

One specific application relevant to smart wheelchairs is placement in nursing homes. In some nursing homes, electric wheelchairs are banned because of their very powerful drivetrains. If a user is not able to adequately control the vehicle, it could cause serious injury to the user or bystanders and easily cause expensive damage to equipment and the property. Those in nursing homes who need the abilities of a smart wheelchair – obstacle avoidance, safety supervision, and full autonomy – often have difficulty controlling vehicles in dynamic environments. It stands to reason that those who cannot safely operate an electrical wheelchair should not be trusted unless supervised while operating the vehicle. To open this market, ease transportation costs and improve the quality of life of these users, the wheelchair should be able to assist the driver in maintaining safety.

In the case of disabled drivers, these drivers may not have the reaction time or control ability to reliably avoid obstacles. It is very easy to miss a dynamic obstacle such as a nurse's cart coming out of a side door in a hallway.

It's also difficult to control the wheelchair with alternative control systems. Users of a sip and puff system have to move air through a tube in order to accelerate and brake or to change the rate of turning. Since their controls all rely on one dimension of control, it's difficult to quickly react and precisely control the wheelchair. Another alternative control would be a vocal joystick system. Vocal joysticks can have limited input frequency because of the typical rate of speech. It could also have difficulties with latency if a command is missed, misinterpreted, or difficult to understand. This is especially dangerous if the last command latches and is valid until a new command is issued. If the wheelchair continues moving and the stop command is missed or for some reason unable to be issued, then the wheelchair could easily continue into obstacles or hazardous areas. These situations and control schemes are all made safer through the use of smart wheelchair technology.

One technology prevalent in all of robotics and smart wheelchairs is the use of laser rangefinders. One of the most popular sensors for this application is manufactured by the German company, SICK. These lasers are large, but offer long range, high precision and high rate measurements. The lasers work by projecting beams of infrared light and measuring the time to return, thus calculating a distance. The internal laser is rotated at a fast rate, returning a reading at certain angular increments. These lasers are typically mounted either underneath the rider with the legs obscuring the measurements or between the leg rests, inconveniencing the passenger (Fernandez-Madrigal, 2004) (Galindo, 2006) (Simpson R. e., 1999). The main problem including a SICK laser scanner

is the typical cost of $6000. This cost is nearly 60% the cost of the wheelchair itself. With this expense, the completed obstacle avoidance system may nearly double the overall cost of the wheelchair.

To avoid the expense of the laser scanner, a common approach is to use alternate sensors. One of the better low cost sensor approaches is to use sonars. Sonars have a long history of applications involving smart wheelchairs. From use with electric wheelchairs (Del Castillo, 2006) (Simpson, NavChair: An Assistive Wheelchair Navigation System with Automatic Adaptation, 1999)(Kuruparan, 2006) (Simpson, The smart wheelchair component system, 2004)and even a manual wheelchair (Simpson, A prototype power assist wheelchair that provides for obstacle detection and avoidance for those with visual impairments, 2005), sonars sensors have been used for collision avoidance and control modification. The main problems with these approaches are tight areas. Sonars typically need to compromise between beam width and detection accuracy because of the physics involved with sound and echoes returning to the sensor within the detection angle.

Few smart wheelchairs have used three dimensional measurements to avoid obstacles. One example is the *Obstacle Avoidance Wheelchair System* (Hoey, 2006). This application used a 3D time of flight camera to create a depth image. This image was then projected down into a two dimensional occupancy grid map. This approach to instrumentation is most similar to the approach taken here using the new Microsoft Kinect™[1] sensor, but the approach taken in

[1] Microsoft – One Microsoft Way Redmond, WA 98052

this thesis extends mapping into three dimensions and also uses additional sensors to create a more practical system.

While there have been many smart wheelchair systems and many of those have focused on lowering cost, the performance has been limited due to the dichotomy between wheelchairs with laser scanners and wheelchairs with sonars. The recent surge in low-cost hobbyist and non-research grade sensors should enable more affordable systems that perform better than the previous low cost smart wheelchairs due to the variety of new sensors available, improving sensor diversity by allowing multiple approaches in parallel to protect the occupant. The success of this project at its extremely low cost (~$1000) is due to the novelty of the Microsoft Kinect, the Neato XV-11™[2] vacuum cleaner, and the products offered by Sparkfun™[3] Electronics.

[2] Neato Robotics – 8100 Jarvis Avenue, Suite 100 Newark, CA 94560
[3] Sparkfun Electronics – 6175 Longbow Drive, Suite 200 Boulder, CO 80301

II. Otto

For this project, Invacare Corporation provided us with a power wheelchair to serve as a research platform for Case Western Reserve University's future smart wheelchair research. In this chapter, the hardware modifications are described that transformed the Invacare TDX-SR power wheelchair into "Otto", a smart wheelchair.

Unlike the previous wheelchairs used in the lab, the TDX-SR was equipped with four-pole brushless motors. The other wheelchairs used brushed DC motors, so the motor control was done with typical robotics and small electric vehicle electronic speed controls combined with quadrature encoders on the drive motor shafts. However, the primary motivation in selecting components for Otto was that the wheelchair would be modified as little as possible and the modifications that were made could easily be performed as part of an affordable aftermarket accessory kit. This approach ensured that the cost would remain affordable to patients or their families and be general enough to be applied to any wheelchair with the appropriate control interface. Modifying the drivetrain would certainly violate these constraints and make the modifications difficult, more expensive, and possibly dangerous if performed incorrectly. For this reason, it was chosen to interface through one of the built-in joystick interfaces on the wheelchair.

Figure 2.1 - Otto, the smart wheelchair. Notable features include a Microsoft Kinect (top of mast), Neato XV-11 Laser Scanner (between front casters), and sonar rangefinders (attached to leg rests).

Joystick Control and Interface

Interfacing to the wheelchair was one of the more complicated and time-consuming aspects of this project. Unfortunately, the accuracy, reliability, and safety of control did not reach a level satisfactory for research testing or precise autonomous control without a human in the loop. The main reason for this change is that the only means of interface for a computer to the wheelchair was to emulate an accessory joystick intended for an attendant to drive the vehicle for the patient. The control loop for the patient's joystick was not exposed in a manner that would not have required disassembly and replacement of all patient interface electronics.

The remote attendant interface was that of three analog voltages read by a factory installed microcontroller and then transmitted over the wheelchair's proprietary and confidential CANbus network. These voltages represented speed command, turning command, and maximum speed. The nominal readings were 2.5V with limits of 1.0V for the lower limit and 4.0V for the upper limit, while speed could go from 0V to 5V and not cause the wheelchair to enter an error state.

To control the voltages to the remote attendant board, an Arduino™[1] microcontroller was chosen due to its ability to properly monitor and adjust the control signals. An Arduino can also be quickly and easily reconfigured for different control modes if unexpected behavior occurred, something a pure hardware interface cannot easily do. The Arduino controlled the voltages by

[1] Arduino – http://arduindo.cc/

sending a 490Hz Pulse Width Modulated (PWM) digital signal into an appropriate RC filter. This filter smoothed the cycled 0 to 5V PWM signal into a steady, but noisy, analog voltage. Conveniently, the wheelchair already expected significant noise on this control signal due to the switching of its motor controller, so the noise and ripple caused by the Arduino was tolerable. The values chosen for the RC filter were 15 kOhm and 1uF, resulting in a time constant of 0.015s which reduced the noise and ripple to adequate levels while still following the controls smoothly and without noticeable delay. To further improve performance, the Arduino also performed PID regulation on the output signal to ensure an accurate output signal.

While the control signal could come from any computer or Arduino connected device, for convenience, a wireless gaming controller connected to Otto's PC was selected. This allowed use of the controller's fast, accurate, and already digital control signal to be used by the rest of the system. It was also the control scheme used to teleop the other robots, and was thus a familiar technique.

Figure 2.2 - XBee Pro Wireless Sensor. A pair of these formed the basis of Otto's emergency stop system. Cost was $40 each and size is about inch per side. Image licensed by Sparkfun Electroncis (CC BY-NC-SA 3.0).

The attendant also had an on-off enable signal. This signal was

mapped to a pair of XBee™[2] Pro wireless network modules with one unit attached to an emergency stop button. When the button was pressed, the signal would drop low on one side of the wireless network and would quickly be mirrored to the other device which was connected directly to the attendant enable. If the two wireless devices lost contact, the signal was configured to automatically drop low. This solution worked well and only cost roughly $110 (the XBees were $40 each, with $20 of interface boards and ~$10 for the Estop button). The range on the XBees can be up to 1 mile in clear, interference free environments. However, due to their frequency of 2.4GHz in indoor environments with many wifi signals the range was reduced to around 100 feet.

This control scheme did have a complication with user expectation. Typically, power wheelchair user-interface centers on the built-in armrest joystick. However, the only method to currently drive the wheelchair under the supervisor safety programs is to use an external wireless gaming controller. It is not intuitive to the users and bystanders that the built-in joystick does not have these safety systems enabled. In fact, the built-in controls cannot employ the safety programs since these controls and wheelchair power are not accessible by the Arduino or the computer. The current policy is to not allow untrained operation of the wheelchair – not ideal for an intuitive smart wheelchair. This limitation will no longer exist once future work enables supervisory control in line with users' expectations.

[2] Digi – 11001 Bren Road East Minnetonka, MN 5534 me

Another integration issue exists with the emergency stop system. That is, the XBee emergency disable works with the assumptions of the attendant board. The attendant and built-in joystick exchange control of the wheelchair; so when the computer is under attendant (or computer) control, the built-in joystick is disabled. The issue is that when the attendant is disabled and the vehicle is in emergency stop, the wheelchair could still move under the built-in joystick without the supervisory program in the control loop. Optimally, an emergency stop would disable all vehicle motion, but the current estop sufficiently disables computer control. When the computer can interface natively with the built-in control system, this issue will also conform to expectations.

Other wheelchair-interface problems were solved with engineering approaches. One unsafe problem the wheelchair experienced was sudden backwards and turning motion if the PID Arduino lost power or was reset. This situation happens when the Arduino resets as its digital outputs all reset to ground. This causes the control voltages to drop below the nominal 2.5V and the wheelchair will rapidly accelerate backwards until the safety cutoff of 1.0V is passed. The solution for this problem was to relay the XBee Estop signal through the Arduino. In doing so, it ensured the Arduino was alive while the wheelchair is enabled. Safety was improved because the relay signal dropped to ground quickly, disabling the attendant interface before the RC filter discharged enough to cause a dangerous command.

The second problem was an unusually large deadband in the joystick commands. If the command did not surpass 12% throw in either direction then

the wheelchair did not move. This led to great difficulty driving the wheelchair through complex courses as the minimum speed could be as high as 0.5m/s. This issue was resolved by scaling the wheelchair's maximum speed to only 0.7m/s, resulting in a minimum speed of a more manageable 0.08m/s with a corresponding minimum angular velocity of approximately 0.1 rad/s. This was a manageable compromise because the much lower top speed resulted in easier control of the vehicle with a time to comfortably stop of only 0.5s.

The final problem was an interdependence of the control axes. One would initially expect that the velocity would vary linearly with input voltage and that the two axes would be independent. However, the attendant board changed the voltages with the expectation that they were being driven by the specific analog joystick intended. The attendant board also introduced coupling between the speed and spin-rate commands. This non-linearity and coupling in controls made it difficult to model the input command versus output speed. While the wheelchair did use an internal hall-effect sensor for speed control, it was also behind many proprietary interfaces and inaccessible. In order to accurately determine the vehicle's speed given a joystick command, quadrature encoders were temporarily attached to each drive wheel. The following Figures 2.3 through 2.6 show the output of this process.

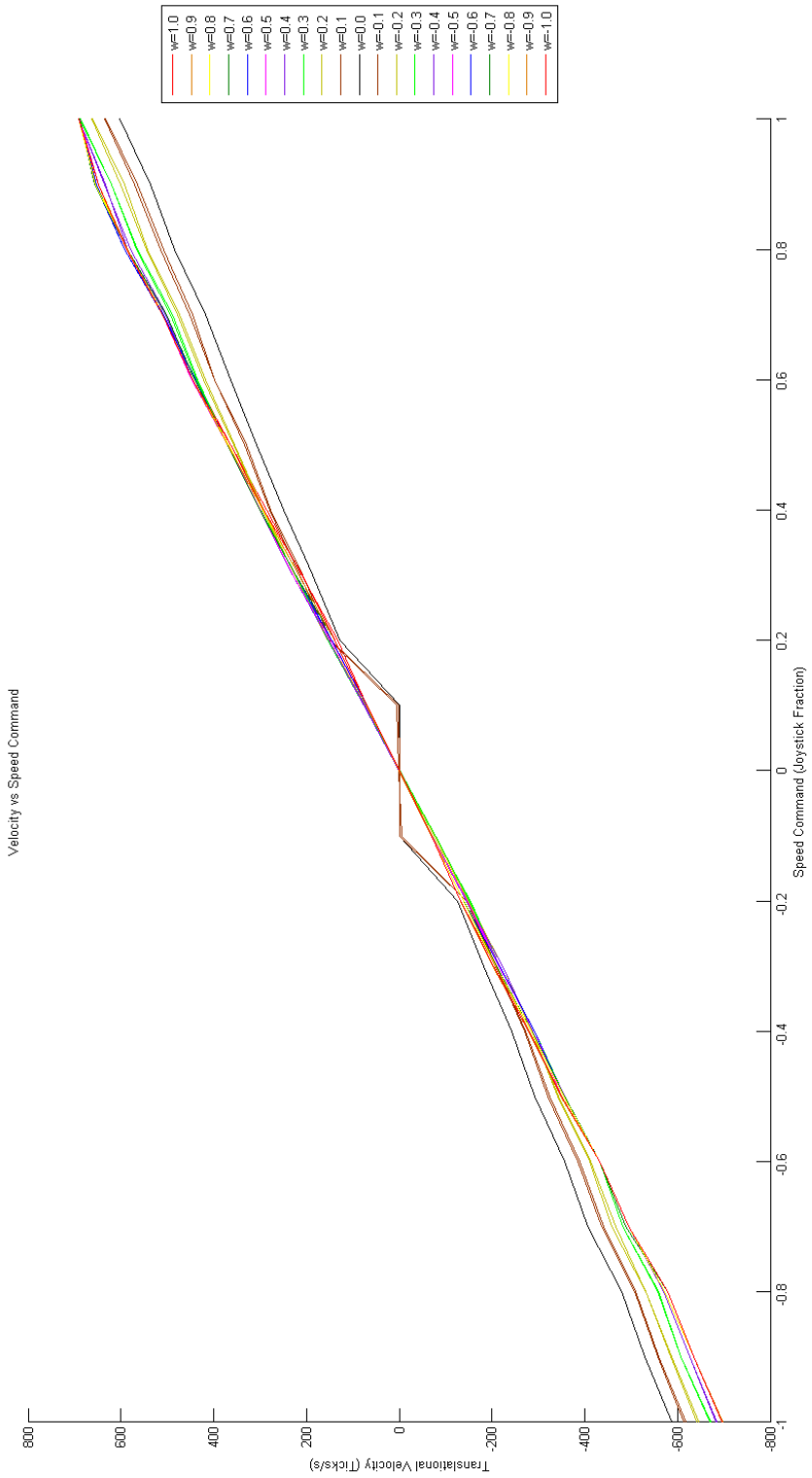

Figure 2.3 – Output Speed increases with Spin Command

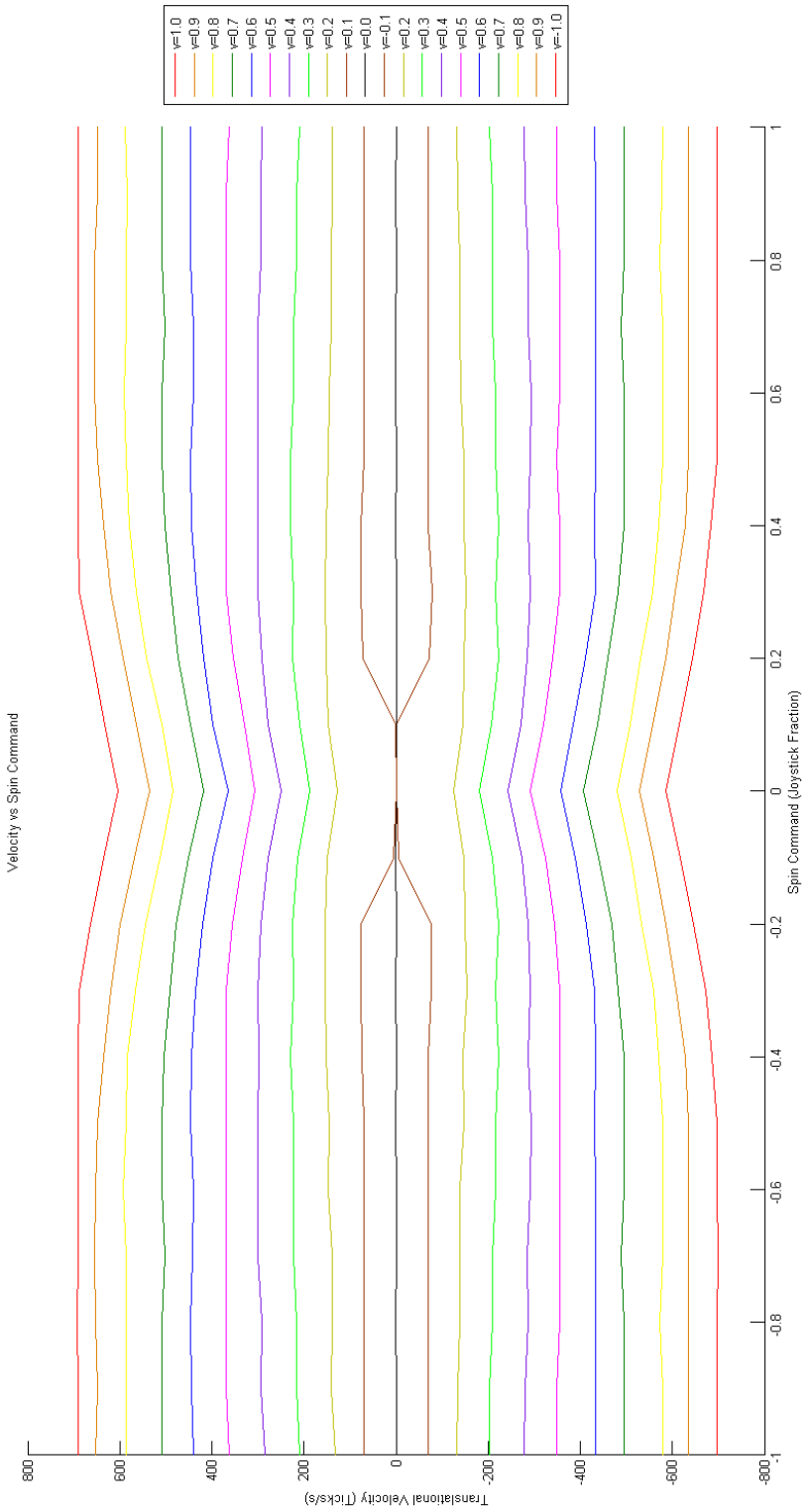

Figure 2.4 – Output Speed increases with Spin Command

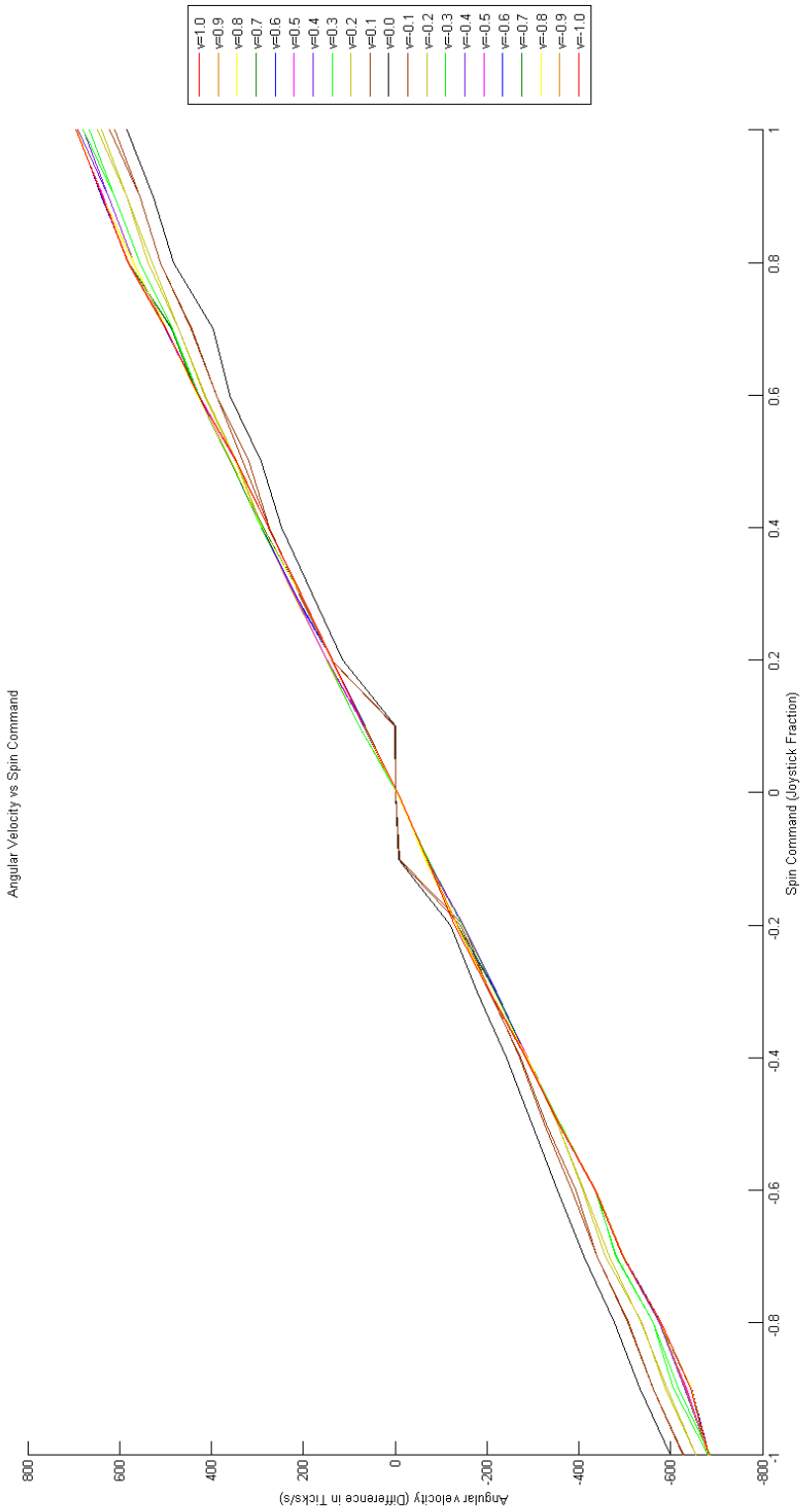

Figure 2.5 – Output Spin Increases with Speed Command

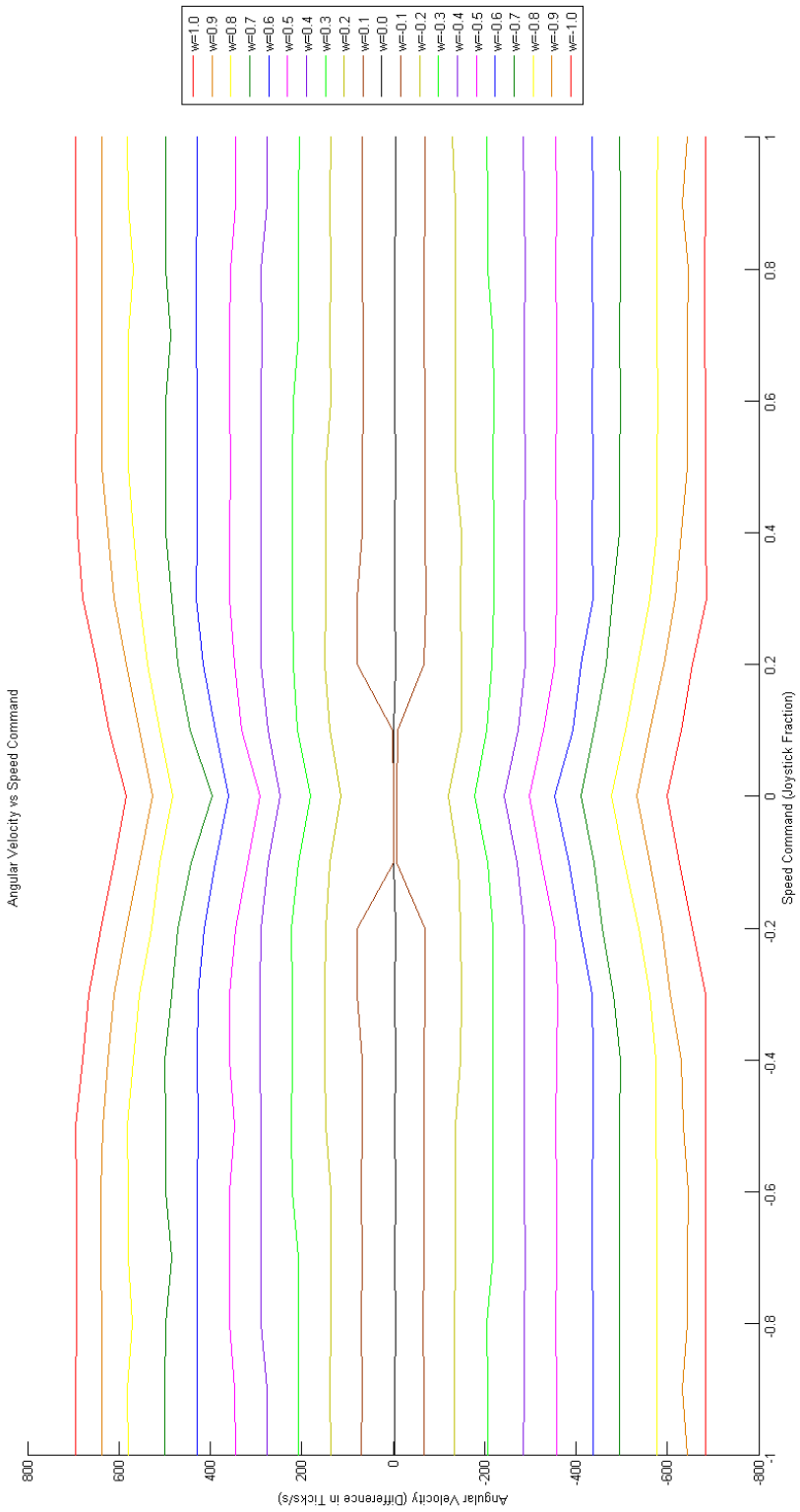

Figure 2.6 – Output Spin Increases with Speed Command

As can be seen in Figures 2.3 through 2.6, the commands were both nonlinear and coupled. To compensate for these effects, the current joystick commands were linearly interpolated into these calibrated measurements. For each axis, two calibration calculations were made for each of the 21 measurement steps: one of the slope and one of the intercept given the other axis. Since the wheelchair had precise control past the attendant interface, the commands always reliably followed this calibration. The output velocity was predicted using equations (1) and (2):

$$v_{est} = \left(v_{slope(cmd_w)} * cmd_v + v_{offset(cmd_w)}\right) * c_{encv} \qquad (1)$$

$$w_{est} = \left(w_{slope(cmd_v)} * cmd_w + w_{offset(cmd_v)}\right) * \frac{c_{encw}}{track} \qquad (2)$$

This estimate of velocity was very useful to predict vehicle speed given command. The results of this are explored more in Chapter III – Localization. However, the inverse of these functions was not feasible to calculate. In order to command the wheelchair in meters/second and radians/second, future work will be required.

While good results were able to be obtained using the methods in this section, ultimately the wheelchair will need a direct computer interface to the control system, patient controls via the onboard joystick, and output from the speed controller before future work can occur to convert Otto into a fully autonomous system.

Sonar Rangefinders

As mentioned previously, sonars have been a very popular approach for low-cost obstacle detection. Sonars are the cheapest individual sensor component for Otto, are fairly reliable, have simple operation, and detect a wide range of obstacles. For these reasons, sonars were an obvious addition to Otto.

While there are a number of sonars available in varying form factors on the market, one type very popular with hobbyists is the MaxBotix™[3] LV-MaxSonar-EZ series. These sonars are incredibly cheap and small: unit cost of only $25 with a pcb area of less than one square inch. Available in a variety of beam widths, these sonars feature analog output, serial output, pwm output, 1 inch measurement resolution, range of 6 inches to 254 inches, no sensor dead zones, 2 mA current requirement at 5V, and can be read at 20Hz.

Figure 2.7- A MaxBotix EZ1 sonar. Image licensed by Sparkfun Electroncis (CC BY-NC-SA 3.0).

For Otto, placement of sonars was very important as it affected where obstacles could be detected, how many sonars were required, and how much effort would be required to retrofit an existing wheelchair with the sonars. It was

[3] MaxBotix Inc. – 7594 South Long Lake Bay Road Brainerd, MN 56401

ultimately decided to mount the sonars on the legrests. This provided the clearest field of a view in the areas the wheelchair was likely to travel while avoiding interference from the operator. Sonars could also be easily mounted on an existing wheelchair by replacing the leg rests. Another important aspect was the proper angle to face each sonar for optimal obstacle detection. At the front of the leg rests, the decided placements were to face one sonar per side forward to directly detect obstacles, one to face inwards to detect doorways and clear errors in the forward sonars, and finally one to the side to prevent the wheelchair from rotating into obstacles.

Figure 2.8 - Location of Otto's sonars on the front leg rests.

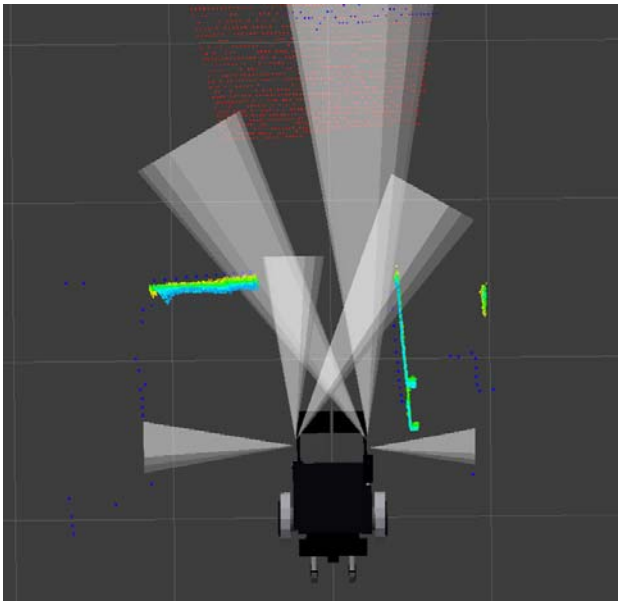

Figure 2.9 - Sonar coverage on the Otto visualization. Note the ability for the cross coverage sonars to detect between the two forward sonars and measure through an open doorway.

To interface with the

sonars, another Arduino was used. This Arduino had two modes of operation: one to trigger the sonars in serialized order and another to trigger the sonars in parallel. Due to the amount of interference between sonars while running them at the same time, it was decide to run the sonars in serialized order. Since each sonar could be triggered at a rate of 20Hz, with 6 sonars, this meant the sonar array functioned at an overall rate of ~3Hz.

Laser Scanner

Laser scanners are essential to modern robotics. Many research robots that measure their environment use either SICK™[4] or Hokuyo™[5] laser scanners, as do many non-sonar-based smart wheelchairs. For this project, we chose to use two different laser scanners: a Hokuyo URG-04LX-UG01 and an extremely low cost range scanner from a Neato XV-11 robotic vacuum cleaner.

The Hokuyo was chosen as it was the cheapest traditional laser scanner available. It operates with a range of 0.02m to 5.6m with a 240 degree scanning area and 0.36 degree angular resolution. It outputs scans at 10Hz and requires 2.5W at 5V. The major disadvantage to this Hokuyo is still its price: $1175. This is an order of magnitude more expensive than any other

Figure 2.10 - Neato XV-11 Robotic Vacuum Cleaner. Sourced from Neato Robotics Press Materials.

[4] SICK AG Erwin-Sick-Str. 1 79183 Waldkirch, Germany
[5] Hokuyo Automatic Co., Ltd. Osaka HU Building, 2-2-5 Tokiwamachi Chuo-Ku, Osaka 540-0028 Japan

single add-on to the wheelchair. The reason it was chosen for inclusion was to measure ground-truth and serve as a backup for the more experimental XV-11 laser scanner.

The XV-11 laser was significantly cheaper than the Hokuyo, but with comparable performance. The only current method to obtain a laser is to disassemble the $400 vacuum robot (Figure 2.10) and remove the laser scanner. However, it is claimed that the bill of materials for for this laser costs only around $30. The specifications are very good for that price: A minimum range of 0.08m, a maximum range of 6m, a field of view of 360 degrees, angular increment of 1 degree, measurement resolution of 0.001m, output scan rate of 5Hz, and power consumption of only 0.75W (Konolige, 2008).

Figure 2.11 - The Neato XV-11 Laser Mounted on Otto below the rider. Note the Hokuyo underneath to provide ground-truth.

The lasers were mounted as shown in Figure 2.11. This location was selected as the safest to protect the laser, as well as a location capable of proving the greatest view to the lasers. In this location, both lasers had a clear field-of-view underneath of the leg rests, so the only obstructions were the casters and drive tires of the wheelchair. The laser mount could also be inverted so that the XV-11 could see behind the wheelchair and fully utilize its 360 degree scan. However, because its rate of rotation was not monitored and controlled, it was more sensitive to disturbances while upside down.

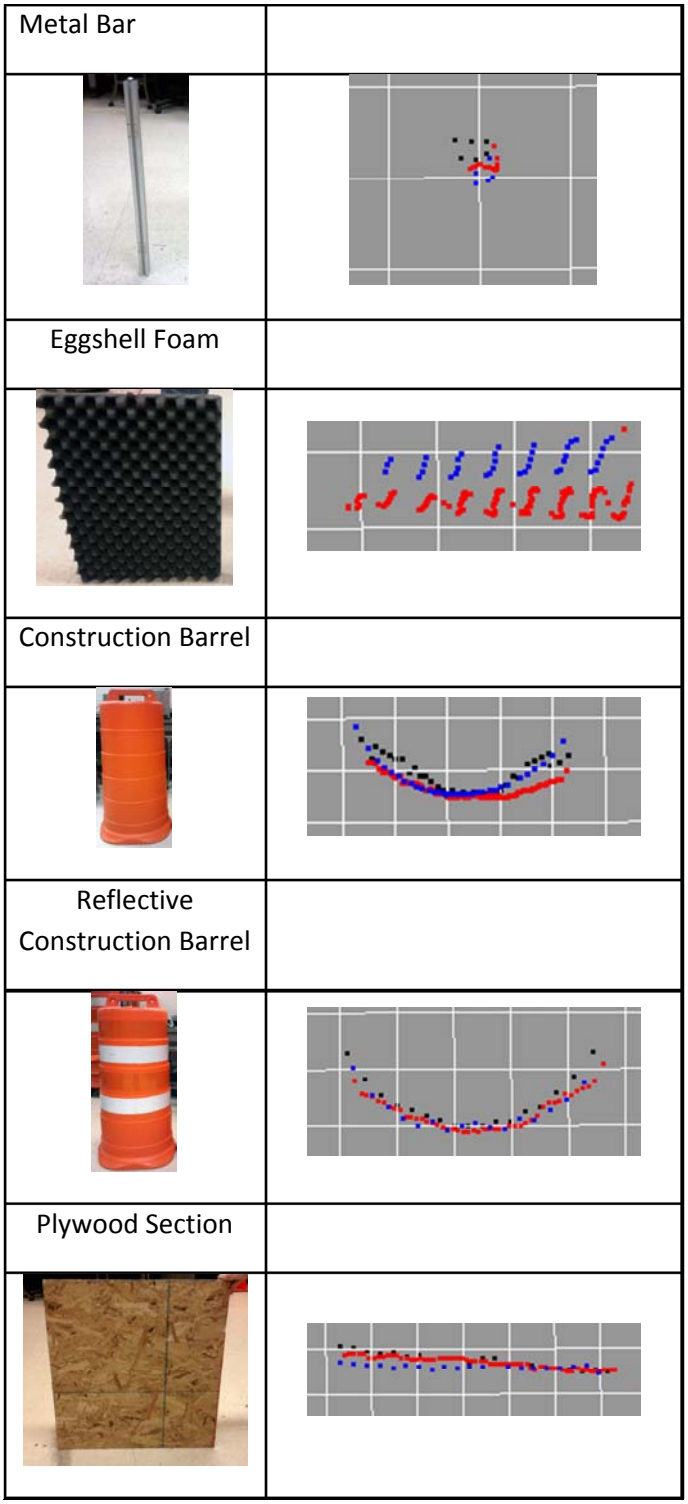

Fig. 2.12. Images of selected obstacles as well as sample laser returns for each obstacle. The grid size units are 10cm by 10cm. The black dots are returns for the **SICK**, the red are returns for the **Hokuyo**, and the blue dots are returns for the **Neato XV-11 laser**. Each measurement was taken approximately 0.5m in front of the scanner array.

As can be seen in Figure 2.12, the Neato performed very well compared to the Hokuyo and the SICK. The main limitation was its low effective max range (Figure 2.13). While the specifications stated ranges up to 6m could be detected, most obstacles were only detectable out to 2m. This was sufficient for obstacle detection, but as discussed in Chapter V, mapping with this limited range proved difficult.

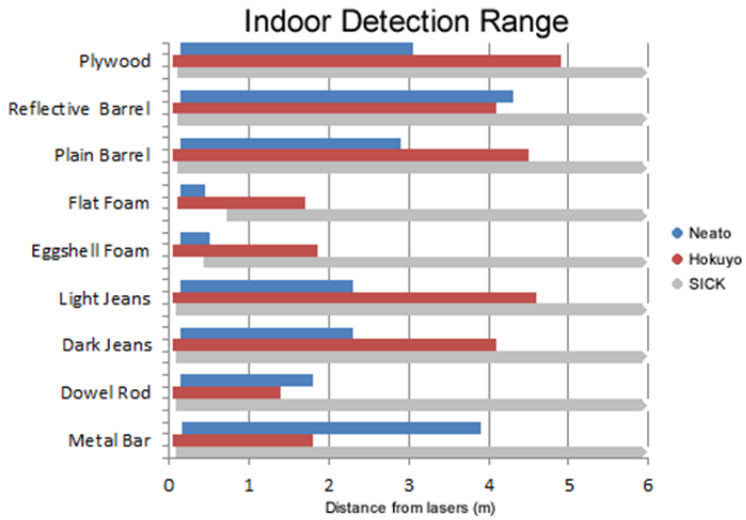

Figure 2.13 - Indoor Detection Ranges for the Neato XV-11, Hokuyo URG-04LX-UG01, and the SICK LMS291.

Microsoft XBOX 360 Kinect

The Microsoft XBOX 360 Kinect sensor was released in November of 2010. This device is intended for use with the XBOX 360 gaming console to enable gestural controls,

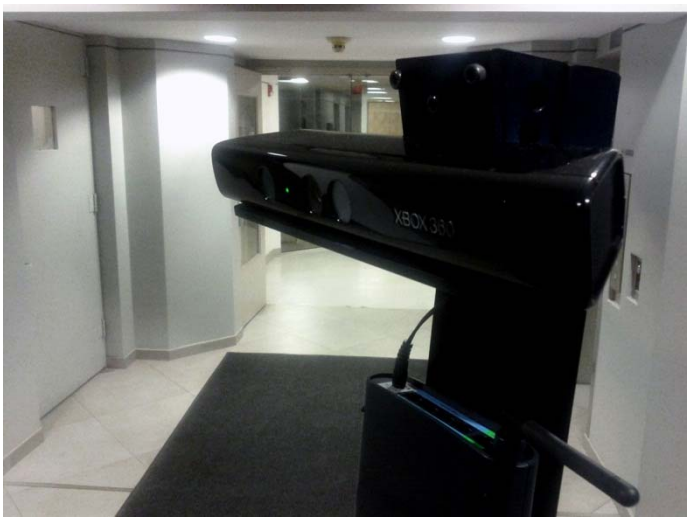

Figure 2.14 - Microsoft Kinect On Otto

dance competition games, and other hands-free approaches to controlling the games. It was released as a direct competitor to the accelerometer and gyro based WiiMote and WiiMotionPlus from Nintendo. This sensor works through the use of a projected IR pattern, an IR camera, and an RGB camera to provide two image streams: one being a depth image and another typical of a color webcam.

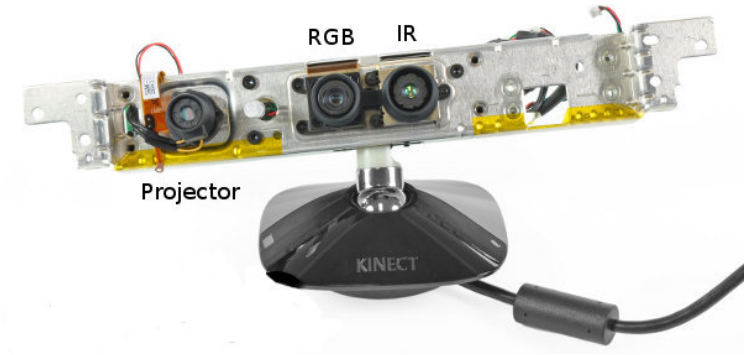

Figure 2.15 - Inside view of a Kinect. The Projector, RGB Camera, and IR Camera are labeled. Image licensed by iFixit (CC BY-NC-SA 3.0).

At release, an open-source driver bounty of $3000 was offered to those who would release the first driver capable of displaying both of the image streams. This driver eventually became the OpenKinect driver and was released into the Robot Operating System (ROS) for use with research robots. The manufacturer of the devices also released a driver (OpenNI) which provided much better calibration. This driver is the one used on Otto.

The projection patterns work through sending a pattern of near-IR light through a set of diffraction gratings. This pattern is compared to a pattern at a known distance. From the known pattern and the current pattern, a disparity (offset in pixels) can be calculated. Using this disparity value and the depth of the calibrated image plane the estimated depth of each pixel can be calculated through triangulation as in equation (3).

$$z_{depth} = baseline * focal\ length\ /\ disparity \quad (3)$$

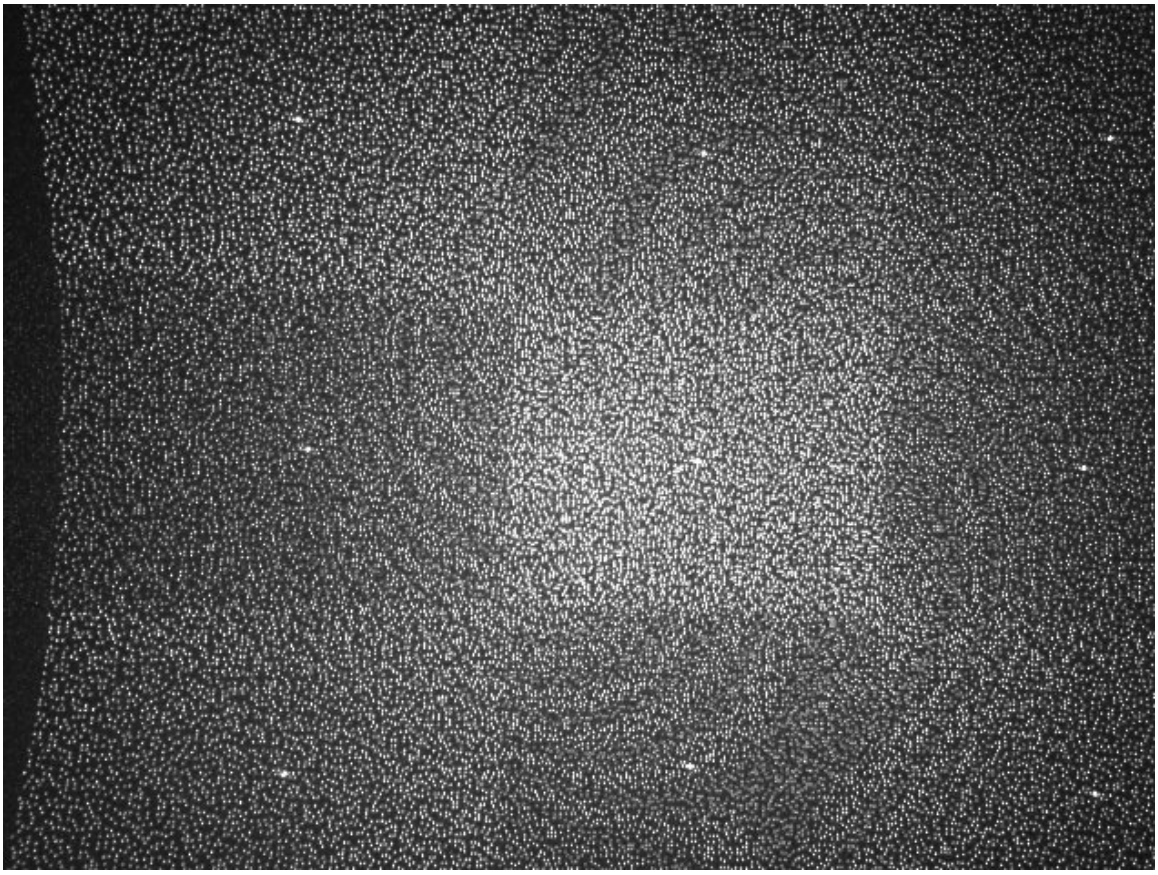

Figure 2.16 - Image of Kinect IR projection pattern used to calculate disparity and estimated distance in Equation (3). Image from ROS and Willow Garage - Kurt Konolige, Patrick Mihelich (CC BY 3.0)

The mounting location on a mast was chosen due to the unique ability of the Kinect to estimate a distance for nearly every obstacle in the view of the cameras. The effective range is roughly 0.5m to 4m but occasionally greater depending on the clarity of the disparity image. This meant that if placed above the user facing forward, the Kinect could detect overhanging obstacles, the ground in front of the wheelchair, and obstacles in wide detection range.

If the depth image is projected according to a pinhole-camera model, then a representation of data known as a pointcloud can be produced. A pointcloud is

simply a collection of sensor measurements projected into 3D space and represented with an X, Y, and Z coordinate. For the Kinect, each pixel projected produced a point in this space. The Kinect's IR camera can produce a depth image of size up to 1280x1024 pixels, resulting in 1,310,720 points at 15Hz. However, since latency is an issue when detecting obstacles, the configuration selected was 160x120 at 30Hz, resulting in 19200 points per scan, a mere fraction of the data output of which the Kinect is capable.

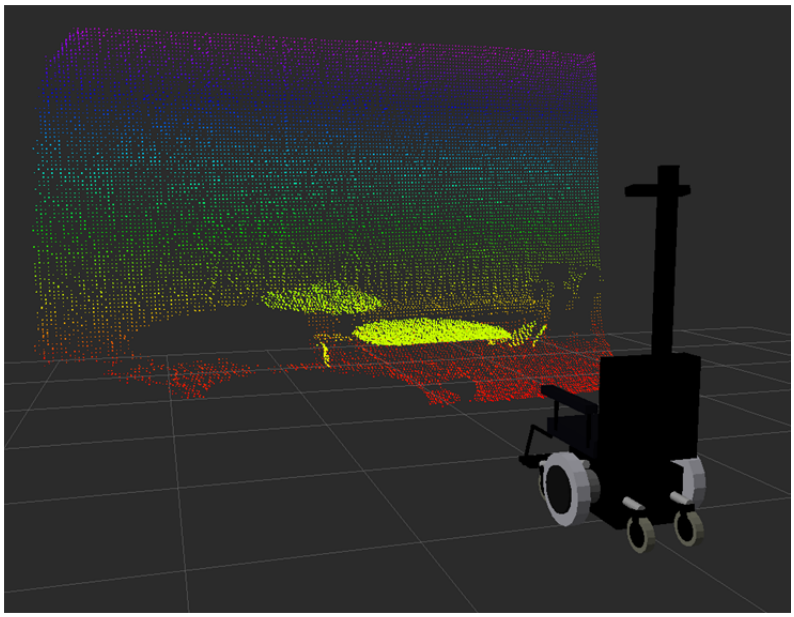

Figure 2.17 - A raw Kinect scan of 160x120 pixels on Otto. Note that the wall and floor are especially visible in front of the wheelchair.

Inertial Measurement Unit

An inertial measurement unit (IMU) was used to help measure the movement of Otto without a costly hardware or drive-train modification. Ideally, Otto's hall-effect sensors would be used for wheel odometry, but these sensors were not accessible. IMUs are typically very expensive and very inaccurate due to accumulated sensor noise integration. To solve this problem, the hobbyist community was used to find a low cost IMU – the Sparkfun 9 degrees of freedom Razor.

For only $125, the 9 degrees of freedom Razor IMU provides an LY530AL single-axis gyro, an LPR530AL dual-axis gyro, an ADXL345 triple-axis accelerometer, and an HMC5843 triple-axis magnetometer on a low power board with an Arduino compatible 16MHz ATmega328.

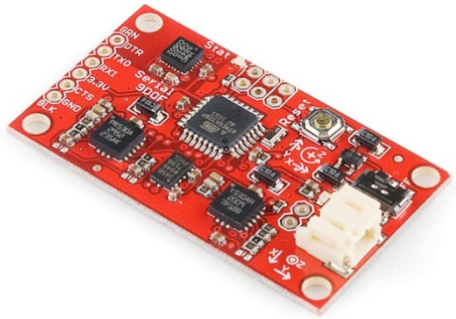

Figure 2.18 - The Sparkfun 9DOF Razor IMU. $125 provides an Arduino, 3 axes of gyro measurements, 3 axes of accelerometer measurements, and 3 axes of magnetometer measurements. Image licensed by Sparkfun CC BY-SA 3.0

Using the built in Arduino, this IMU was capable of performing absolute orientation calculations using all 9 degrees of freedom. However, due to magnetometer interference from the wheelchairs motors and from various sources inside buildings such as magnetic door locks, the magnetometers were not used.

The IMU was easily calibrated by leaving it motionless on a known-level surface and offsetting the measurements by some known a-priori amount. This allowed the values to be integrated in an Extended Kalman Filter as detailed in Chapter III – Localization. Offsetting the known sensor biases allowed the filter to integrate the gyros to produce changes in heading and double integrate the accelerometer combined with the current commands to produce surprisingly accurate estimations of forward motion.

Computing and Packaging

The last requirement for Otto was a modern computer to run the algorithms required not only for user-assistance but also capable of running the systems required for a fully-autonomous robotics system.

Figure 2.19 - The rear view of Otto. The grey box on the left holds the relevant Arduinos, power supply, and interface board. The Computer can be seen on the right.

The computer was selected for both its form factor and computational abilities. The case and power supply were purchased from an automotive computing supplier[6]. The M350 is one of the smallest desktop computer cases available, measuring only 19 x 21 x 6 cm. Inside, the case holds a mini ITX motherboard and an Intel i5 650 processor with 2 Cores, 4 Threads, a 3.2GHz clock speed, and built-in integrated graphics. The processor has a maximum power consumption of 73W and a cost of around $170 in 2011.

[6] http://www.mp3car.com/

The overall cost of the computer was roughly $600 and included 8GB of RAM, a 140W 24V DC input power supply, the mini ITX motherboard, and a 64 GB solid state drive. While the computer is the most expensive individual add-on component, the cost is justified for the ability to run the wheelchair as a fully autonomous vehicle. If productized as only an obstacle avoidance system, the computer system would not need to have nearly the capabilities of Otto.

To keep the electronic systems safe, a mounting box (Figure 2.20) was included to safely house the miscellaneous parts and package them in an easy-to-install unit. The mounting location for both the computer and the electronics was chosen to be the base of the Kinect mast. This allowed the entire system to be installed with only three components: the leg rests, the laser mount, and the Kinect

Figure 2.20 - Otto's Electronics Box, holding the IMU (top left) Sonar Arduindo (bottom left), control Arduindo (bottom right), Attendant Board (Right Middle), XBee Estop (bottom), and 24V to 5V power supply (top right).

mast. Using so few components would drastically reduce the expense of a technician's time to modify wheelchairs with the aftermarket avoidance and automation systems.

Summary

The selection of low cost, open source, and hobbyist components for Otto allowed the entire sensor cost to be less than $500. With the addition of the computer, the bill-of-materials for the final product should easily be less than $1000, an affordable value for a non-subsidized aftermarket add-on.

The sensors chosen for Otto are dramatically more capable and lower cost than what was available only a few years prior. The Neato XV-11 laser and the Kinect sensor are notable examples of new, low-cost, high-performance sensors. These sensors are revolutionizing hobby robotics and computer perception, and they offer promise for highly competent and affordable smart wheelchairs.

III. Localization and State Estimation

An accurate representation of a vehicle's current physical state is crucial to safety and a keystone to mobile robotic applications. Without an accurate idea of where the wheelchair will be in the future and where it was in the past, obstacles prove significantly more dangerous. Knowing which objects will pose collision hazards depends on knowing the expected path of the wheelchair. If this path is not known or is known poorly, then a much larger pool of potential trajectories must be considered, limiting freedom of movement and increasing computational requirements needed to ensure safety.

Almost every implementation of robotic movement control depends heavily on estimate of the control variables. Whether these algorithms control on linear velocity, angular velocity, or accelerations, inaccurate estimation of these values will produce errors in control and undesirable vehicle movement. While this work depends on good estimation of velocities to determine if a command is safe, path following algorithms depend on these estimates for reliable and repeatable execution of paths.

Within the Robot Operating System (ROS)[1], the base_local_planner depends on accurate execution and reporting of velocities (Marder-Epstein, 2011). The goal of the planner is to return twists (sets of velocities and angular velocities) that will efficiently and reliably allow the robot to follow a given global path without colliding with obstacles. Base_local_planner relies upon reported velocities to stay within allowed acceleration limits while discretely sampling

[1] http://ros.org

possible velocities to determine which is most desired at this time. By using the actual velocities, this code can guarantee better path following by staying within defined accelerations and not requesting unreasonable or dangerous changes in commands that a feed-forward only approach may if the estimated state and actual state become disjoint.

Another path following algorithm that depends greatly on state estimation is that of Eric Perko's precision navigation (Perko, 2012). This algorithm relies on velocities to rapidly correct the controls needed to follow mathematically defined path segments. This method was designed specifically to overcome errors in command execution by having reliable position and velocity feedback. This feedback allows accurate path following from a variety of initial error conditions that would not be possible if the algorithm did not have access to both short-term and long-term state estimation.

As discussed previously, due to the financial goals and desire for little modification on the actual wheelchair, few sensors were determined to be suitable for this project. While the wheelchair uses the feedback system built into the four-pole motors for its control system, this data is not accessible to higher-level systems. This meant that the controls should be very reliable, but specific feedback could not be obtained for Otto. The other affordable sensor able to be placed on the wheelchair without large modifications was the inertial measurement unit (IMU).

Using the feed-forward estimate of controls along with the actual inertial measurements from accelerometers and gyros proved to be sufficient to produce

odometry estimates comparable to our other research vehicles for a fraction of the price. The feed-forward estimate of linear velocity was combined with the measurements of angular velocity and linear acceleration in a single-hypothesis extended Kalman filter. While the IMU also contains magnetometers, these were not usable in this application due to interference from large security magnets present above most doors.

The following section was adapted from the material presented in *Probabilistic Robotics* (Thrun, Burgard, & Fox, 2005)and *An Introduction to the Kalman Filter* (Welch & Bishop, 2001).

Extended Kalman Filter Position and Velocity Estimation Algorithm

Given:

- Previous Estimated State, **x**
- Previous Covariance Estimate, **P**
- Motion model, $f(\mathbf{x})$
- Model Noise Estimate, **Q**
- Measurement Model, $h(\mathbf{x})$
- Current Measurement, **z**
- Measurement Noise Estimate, **R**

Do:

- $\mathbf{x}' = f(\mathbf{x})$
- $\mathbf{P}' = \mathbf{FPF}'+\mathbf{Q}$
- $\mathbf{y} = \mathbf{z} - h(\mathbf{x}')$
- $\mathbf{S} = \mathbf{HPH}'+\mathbf{R}$
- $\mathbf{K} = \mathbf{PH}'\mathbf{S}^{-1}$
- $\mathbf{x} = \mathbf{x}' + \mathbf{Ky}$
- $\mathbf{P} = (\mathbf{I}-\mathbf{KH})\mathbf{P}'$
- return **x**, **P**

The linear Kalman Filter is an ideal fusion technique for systems that are purely linear with normally (Gaussian) distributed noise. The Kalman Filter tracks the mean and covariance estimates for a given system over time. For applications with non-linear updates, the Kalman Filter has been adapted into an Extended Kalman Filter. The important change is that the first-order partial derivative is used to linearize the covariance prediction. Any Kalman Filter has two main steps: a

prediction step and an update step. The following is a run through and illustration for an iteration of an Extended Kalman Filter:

Prediction:

Prediction of State:

$$\boldsymbol{x}'_t = f(\boldsymbol{x}_{t-1}) \tag{1}$$

$\boldsymbol{x}_t$ is the state of the Extended Kalman Filter

$$\boldsymbol{x}_t = \begin{pmatrix} x \\ y \\ \theta \\ v \\ \omega \\ a \end{pmatrix} \tag{2}$$

The function f is the model update function. This EKF was updated at 50Hz for a dt of 0.02s.

$$f(\boldsymbol{x}_t) = \begin{pmatrix} x_{t-1} + v_{t-1} * dt * \cos(\theta_{t-1}) \\ y_{t-1} + v_{t-1} * dt * \sin(\theta_{t-1}) \\ \theta_{t-1} + \omega_{t-1} * dt \\ v_{t-1} + a_{t-1} * dt \\ \omega_{t-1} \\ a_{t-1} \end{pmatrix} \tag{3}$$

Prediction of Covariance Matrix

$$P'_t = FP_{t-1}F^T + Q \tag{4}$$

P, the estimate of the Covariance matrix, represents uncertainty along the diagonal, and tendencies to change together elsewhere.

$$P=\begin{pmatrix} \boldsymbol{\sigma_x}^2 & cov(x,y) & cov(x,\theta) & cov(x,v) & cov(x,\omega) & cov(x,a) \\ cov(y,x) & \boldsymbol{\sigma_y}^2 & cov(y,\theta) & cov(y,v) & cov(y,\omega) & cov(y,a) \\ cov(\theta,x) & cov(\theta,y) & \boldsymbol{\sigma_\theta}^2 & cov(\theta,v) & cov(\theta,\omega) & cov(\theta,a) \\ cov(v,x) & cov(v,y) & cov(v,\theta) & \boldsymbol{\sigma_v}^2 & cov(v,\omega) & cov(v,a) \\ cov(\omega,x) & cov(\omega,y) & cov(\omega x,\theta) & cov(\omega,v) & \boldsymbol{\sigma_\omega}^2 & cov(\omega,a) \\ cov(a,x) & cov(a,y) & cov(a,\theta) & cov(a,v) & cov(a,\omega) & \boldsymbol{\sigma_a}^2 \end{pmatrix} \quad (5)$$

F Is the Jacobian of $f(\boldsymbol{x}_t)$ with respect to the elements of the state vector

$$F=\begin{pmatrix} 1 & 0 & -v*dt*\sin(\theta) & dt*\cos(\theta) & 0 & 0 \\ 0 & 1 & v*dt*\cos(\theta) & dt*\sin(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (6)$$

Q is the **Process Noise**, or uncertainty in the model, expressed as a diagonal vector, the implemented process noise was Q=[1e^{-8} m^2, 1e^{-8} m^2, 1e^{-8} rad^2, 1.0 m^2/s^2, 5.0 rad^2/s^2, 1.0 m^2/s^4]

$$Q=\begin{pmatrix} \boldsymbol{\sigma_x}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \boldsymbol{\sigma_y}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boldsymbol{\sigma_\theta}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \boldsymbol{\sigma_v}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boldsymbol{\sigma_\omega}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \boldsymbol{\sigma_a}^2 \end{pmatrix} \quad (7)$$

Update:

Innovation: difference between actual, z, and predicted measurements, $h(\boldsymbol{x'}_t)$

$$y = z - h(\boldsymbol{x'}_t) \quad (8)$$

$$h(\boldsymbol{x'}_t) = \begin{pmatrix} v_t \\ w_{t-1} \\ a_{t-1} \end{pmatrix} \begin{matrix} predicted\ feed\ forward\ velocity \\ predicted\ angular\ velocity \\ predicted\ acceleration \end{matrix} \quad (9)$$

H is the Jacobian of $h(\boldsymbol{x'}_t)$ with respect to the elements of the state vector

$$H=\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (10)$$

Innovation covariance: Gaussian weights for sensor confidence

$$S = HP'H^T + R \tag{11}$$

R is the estimated covariance for the sensors. Expressed as a diagonal vector, the used estimated covariance was R=[100 m^2, 0.001 rad^2/s^2, 0.01 m^2/s^4).

$$R = \begin{pmatrix} \boldsymbol{\sigma_{v\_est}}^2 & 0 & 0 \\ 0 & \boldsymbol{\sigma_{gyro}}^2 & 0 \\ 0 & 0 & \boldsymbol{\sigma_{accelerometer}}^2 \end{pmatrix} \tag{12}$$

Kalman Gain: Used to incorporate innovation into prediction

$$K = P'H^T S^{-1} \tag{13}$$

Update state:

$$\boldsymbol{x}_t = \boldsymbol{x'}_t + Ky \tag{14}$$

Update covariance matrix:

$$P = (I - KH)P' \tag{15}$$

The noise estimates are essential to optimal performance of the filter. The values in (7) control the expected error in the model due to conditions such as integral approximation errors, linearization errors, and timing errors. If set too large, noisy sensors may dominate the filter, but if set too low, then the model's approximations will cause errors in the estimations. On the other hand, the values in (12), representing the confidence levels in the sensors, do the reverse. If these values grow too large, the sensors do not update the state Gaussians appropriately; and if they grow too small, the sensors will be too confident that their errors are representative of reality, and the filter will suffer as a result.

These values, as well as various sensor-specific conversion constants, are typically "hand-tuned" by estimation through various calibration routines, manufacturers' expected tolerances, and simply guessing appropriate values.

Although not used for this implementation, by implementing an Extended Kalman Filter and using numerical optimization techniques, it is possible to train the filter through these constants and other parameters to return better results given an accurate estimate of physical state (Abbeel, Coates, Montemerlo, Ng, & Thrun, 2005).

The odometry was tested by making long traversals of the Glennan Building second and third-floor hallways. The error estimates from the long runs in these areas provided confidence that the odometry system is accurate for short-term (less than 30 seconds of history) position estimation.

As can be seen in Figure 3.1 and Figure 3.2 for the second and third floors respectively, there are minimal errors in both position and heading. The actual accumulation in unit time as measured with a static beginning and end point result in accumulating distance error on average at a rate of 0.75% of the overall distance traveled and 0.001 rad/s (0.057 degree/s) error in heading while moving. Since the wheelchair's programmed top speed is around 0.7m/s, the greatest errors in a 30s window of time would result in a magnitude of distance error of 15.75cm and heading error of 0.03 radians (1.72 degrees) on a path of distance of 21m. These errors would be easily correctable with modern Simultaneous Localization and Mapping (SLAM) techniques and a capable laser scanner (Thrun, Burgard, & Fox, 2005). Furthermore, the furthest out the sensors can see is about 4m. With a path distance of only 4m, there can be a magnitude of position error of 3cm and heading error of only 0.0057 rads (0.327 degrees). Such errors are easily ignored or cleared when incorporated into

obstacle positions on maps, since the map resolution was chosen to be 2.5cm width of a unit cell. It should be noted that while not in motion, these errors do not accumulate. This is enforced by a modification of the algorithm that limits estimated state updates while the vehicle is known to be stationary. However, errors will rapidly accumulate if the wheelchair is moved under external forces, since external forces are not part of the system model.

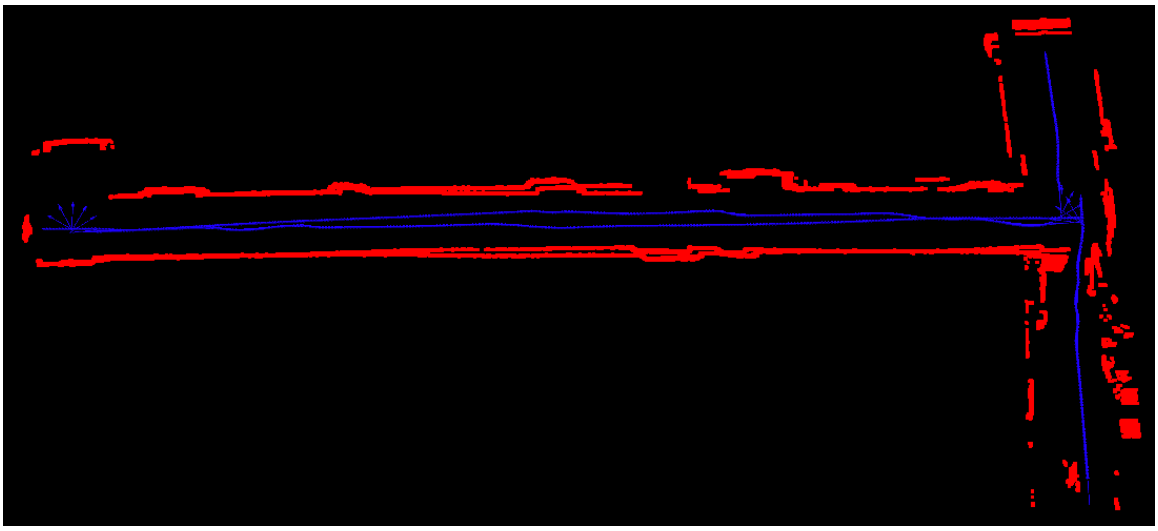

Figure 3.1 - Laser scan overlay (red) map of Glennan Building 2nd floor with estimated odometry (blue).

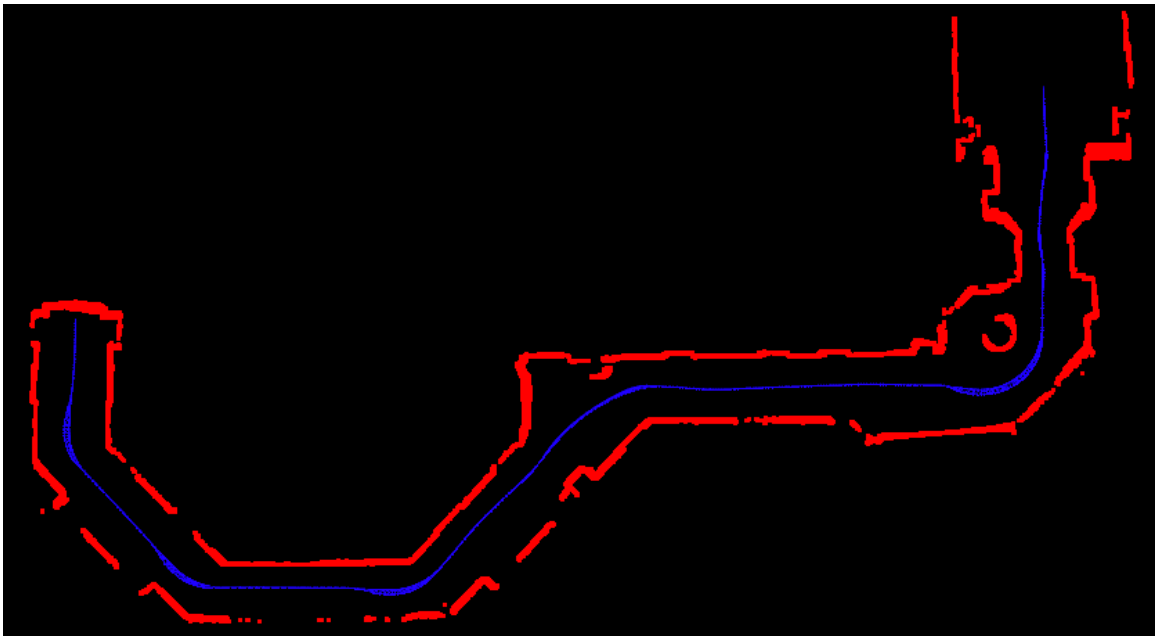

Figure 3.2 - Laser scan overlay (red) map of Glennan Building 3rd floor with estimated odometry (blue).

IV. Mapping

While approaches for obstacle avoidance using only current sensor measurements are useful for vehicles with complete observation of the surrounding environment, placing numerous sensors can be both a financial and technical challenge. For Otto, the amount of available sensor space was limited due to the constraints from placing onto an existing wheelchair and from limiting the hardware budget to be affordable for the end users. Without the ability to sense the entire environment, memory must be introduced into the system. Considerations included: efficient incorporation of new information, ray tracing, and occupancy checking. For its ability to insert dense point clouds, clear from the sensor source, and quickly check large volumes for obstacles, Octomap was chosen as the mapping technique for this project (Wurm, Hornung, Bennewitz, Stachniss, & Burgard, 2010).

One decision needed for the wheelchair was whether to use global or local mapping techniques. For local mapping, the map center moves with the robot and obstacles are 'forgotten' as they fall off the map. Global mapping provides the advantage of never losing obstacles when returning to a previously known environment. However, the localization error tolerances are much smaller for these techniques. For this reason and the unavoidable drift in the localization system, local mapping techniques were used.

The technique implemented for local mapping involved clearing the entire map while the wheelchair was not in motion and there was a large margin of safety around the vehicle. For this technique, it was decided that a one to two

meter radius was safe due to the acceleration and velocity constraints set on the wheelchair depending on if there was a reverse laser scanner. This did not result in many map resets, but due to the efficiency of Octomap, the indoor environment map size rarely exceeded a few hundred megabytes, easily within the memory constraints of today's budget computers.

Periodic map resets also help with accumulated sensor noise. Since the wheelchair was not performing a SLAM method due to limited sensor range, noise could accumulate. The two methods to combat accumulated noise were the periodic map clearing and ray-trace methods. While all sensors are subject to noise, the cheaper sensors like the Neato XV-11 laser and the Hokuyo URG-04LX-UG01 can have many spurious readings (see Figure 4.1). These readings cannot be easily distinguished from obstacles without multiple measurements. Otto relies upon the map clearing when stationary or ray tracing reducing the probability of the Octree node until there is confidence that there is not an obstacle.

Figure 4.1 - Accumulated sensor noise from a Hokuyo URG-04LX-UG01. Scans accumulated over time without measurement fusion. Position information obtained from wheel odometry and localization using a SICK LMS 291 and Slam Gmapping on a research robot.

Another possibility not used for Otto is to slowly reduce obstacle probability based on time since measurement. Since Otto can expect to move slowly and in dense environments, this method was not desired due to the chances of clearing obstacles not currently observed by sensors but still close in proximity to the wheelchair.

A simple but effective technique for mapping is to simply accumulate each sensor point measured in some data structure; typically just a list is required.

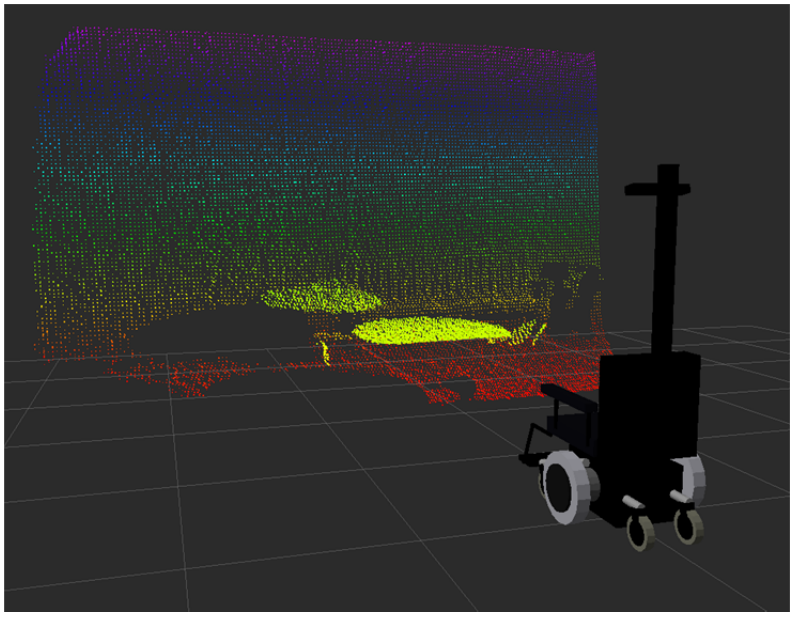

Figure 4.2 - A raw Point Cloud representation of tables, chairs, and a wall. This is a single low resolution scan from the Microsoft Kinect as measured by

This point cloud technique requires no Euclidian-space inspired memory structure and allows no loss of precision or information from sensor to map. This technique is often used for showing comparisons of raw sensor data by simply accumulating the visualization of all raw measurements onto the map (See Figures 3.1, 3.2, 4.1, 4.2). While very useful for visualization and recording sensor data, the memory requirements and ability to quickly search and parse the data are very limited. Not only is every point measured recorded to memory, but each point must be checked if a volume can be declared occupied or empty space. The increasing computational and

memory requirements become very prohibitive and render this approach not feasible for occupancy checking.

Another common technique for 3D mapping is to mark a structure known as a voxel grid as either clear or an obstacle. A voxel grid can be thought of as stacked cubic sections of physical space where the side length of a cube is the resolution of the map. The 2D equivalent would be using image pixels to hold map information.

One advantage of voxel grids over point cloud representations are memory requirements. Instead of accumulating each point into a list, voxel grids have some measure of occupancy per unit. One simple approach is that if a measurement falls within a voxel, that voxel is marked as occupied. This technique is simple, but often experiences difficulties due to sensor noise. Another approach is to keep track of the number of measurements that fall within the voxel (indicating an obstacle) and compare that to the measurement vectors that pass through the voxel (indicating free space) and use the ratio to form a probability that the voxel is actually occupied. With this method, all sensor information is used to modify the constant memory-size per unit volume voxels, so that more information only refines the already existing information and does not require additional storage.

Another advantage of voxel grids is a known a priori number of units to check per volume of physical space. In the organized voxel grid, it is easy to define a volume of interest and iterate through some known indices of voxels to produce a result using all previous sensor data in a constant number of checks.

This advantage allows voxel grids to be used efficiently for collision checking and determining free space compared to the ever increasing amount of checks necessary for point cloud representations.

Voxel grids do have limitations. Specifically, the number of voxels needed grows cubically relative to the width of the cubes used for the voxel. For Otto, this means that in order to have high precision mapping, an extraordinary amount of voxels need to be kept in memory. For 2.5cm voxels, the number of required units is 64,000 per cubic meter; at 1cm voxels that number grows to 1,000,000 voxels per cubic meter. Even with a very efficient storage system of one byte per voxel (possibly a weighted probability of occupancy), that is around 8MB per cubic meter. For a 30m hallway that is 2.5 m wide, and 2m tall, there could be well over 1.2GB of data, representing mostly empty space!

Octomap takes the advantages of voxel grids over point clouds and makes the performance even better. Octomap is based on the idea of an Octree unit. An Octree is a nested data structure of voxels. Each Octree has eight possible children, each child another Octree with a voxel-width one-half that of its

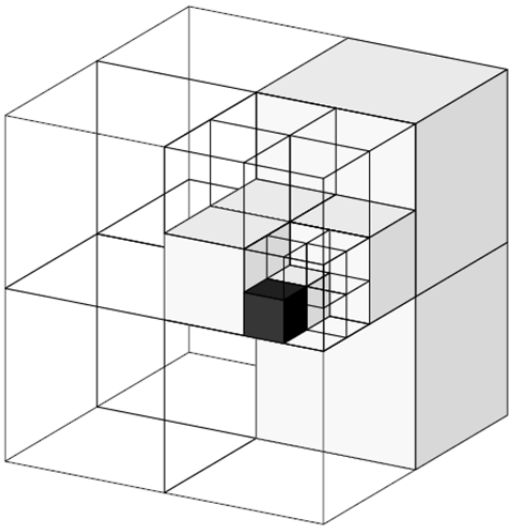

Figure 4.3 - An example Octree structure. This diagram shows four levels of Octree recursion, each level halving the voxel size. Note that larger voxels are marked completely as one class, eliminating the need to keep their children in memory. (Wurm, Hornung, Bennewitz, Stachniss, & Burgard, 2010)

parent. Using this representation, only the smallest-necessary division needed to represent the data as occupied, free, or unknown is used. Thus, if all descendent Octrees of a node are marked as one value (occupied, free, or unknown), then the parent can be marked with that class as well, and the memory previously used for the children can be freed and reallocated later if higher resolution is needed to accurately map new sensor data.

Octomap also uses a probabilistic model to determine when a node is "stable" enough to mark confidently as free-space or occupied. The Octomap implementation used in the present work is a probabilistic representation, as introduced by Moravec and Elfes (Moravec & Elfes, 1985).

$$P(leaf\ node\ n \mid measurements\ z_{1:t}) = \left[1 + \frac{1-P(n|z_t)}{P(n|z_t)} * \frac{1-P(n|z_{1:t-1})}{P(n|z_{1:t-1})} * \frac{P(n)}{1-P(n)}\right]^{-1} \quad (1)$$

By using a uniform prior of $P(n) = 0.5$, a precomputed inverse sensor model $P(n|z_t)$, and logOdds notation, Octomap simplifies equation (1) to allow faster, incremental updates:

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t) \quad (2)$$

However, this approach is poor for dynamic environments as it requires as many opposite observations to update the probability of a specific node as were previously observed. Octomap modifies equation (2) to clamp the maximum probabilities:

$$L(n|z_{1:t}) = \max(min(\, L(n|z_{1:t-1}) + L(n|z_t), l_{max}) \,, l_{min}) \quad (3)$$

Octomap checks free-space for navigation at a requested resolution. For this check, a parent node's occupancy probability is defined as:

$$L(n_{parent}) = \max_{child\ i} L(n_i) \quad (4)$$

This method allows efficient checks for navigation at a resolution different than that of the base map structure. By covering more free space in fewer operations than standard voxel maps, Octomap allows a higher dynamic update frequency. However, no flexibility is lost as the map can be polled for a finer resolution if needed.

Finally, Octomap uses the probability estimates to improve tree pruning. If the occupancy probability saturates at either l_{min} or l_{max}, then that node is considered stable. If all children of a node are stable with the same probabilistic label, then these children can be discarded as the parent is also stable with the same label as its children. If finer resolution is required to represent future measurements, the children are simply reallocated and seeded with the parent's prior plus the contradicting measurement.

Pruning also improves the efficiency of ray tracing by reducing the number of units modified for each measurement. Since rays travel through empty space preceding the obstacle return and the free space represented by Octrees can easily represent volumes orders of magnitude larger than voxels for an equivalent map, Octomap typically iterates through far fewer elements than voxels and will never iterate through more elements. This improvement in speed allows faster

map update frequencies and improved reaction times without compromising on map accuracy or the ability to handle dynamic environments.

For the Kinect and laser scanners, navigable maps were produced using the Octomap methods. See Figures 4.4-4.11 on the following pages.

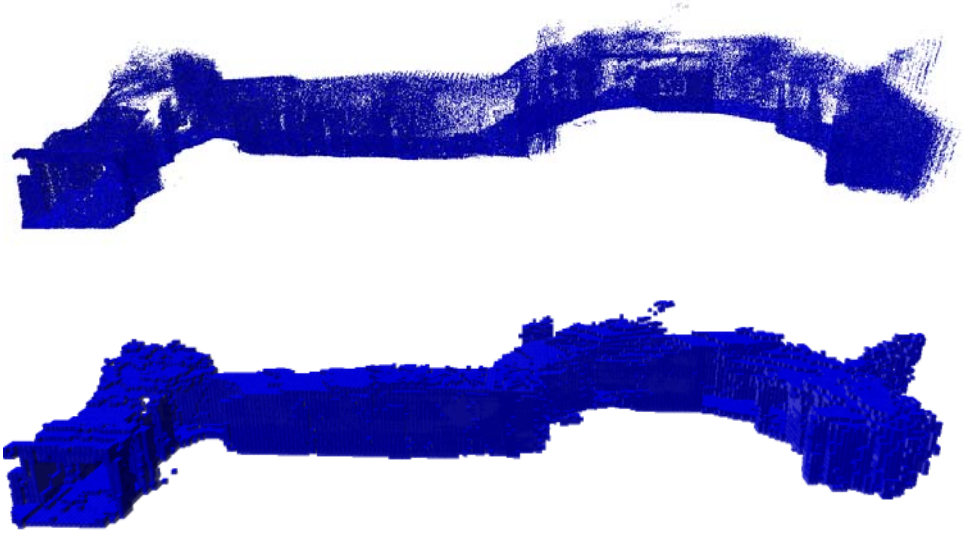

Figure 4.4 - Map of Glennan 3rd Floor produced using entire low resolution Kinect scans and Neato laser scans. Visualized at 2.5cm (top) and 20cm (bottom).

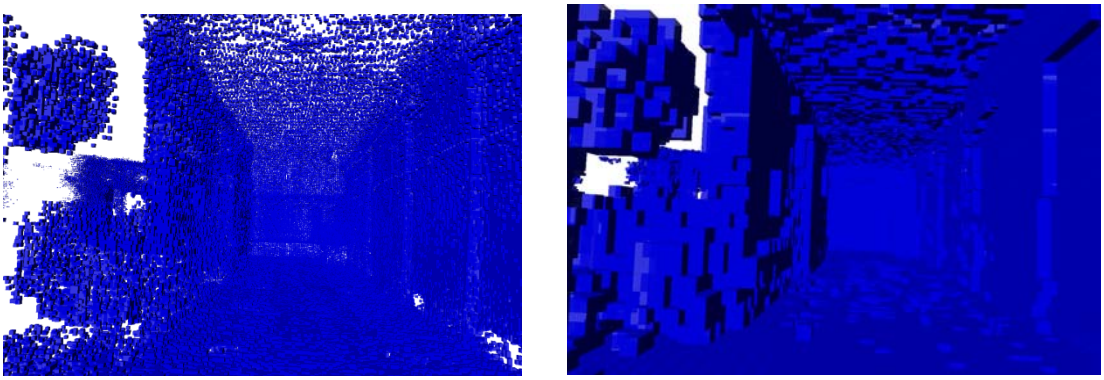

Figure 4.5 - View inside Glennan 3rd floor hallway. Visualized at 2.5cm (left) and 20cm (right).

Figure 4.6 - Map of Glennan 3rd Floor produced using 11 vertical slices of Kinect pointcloud data. Visualized at 2.5cm (top) and 20cm (bottom).

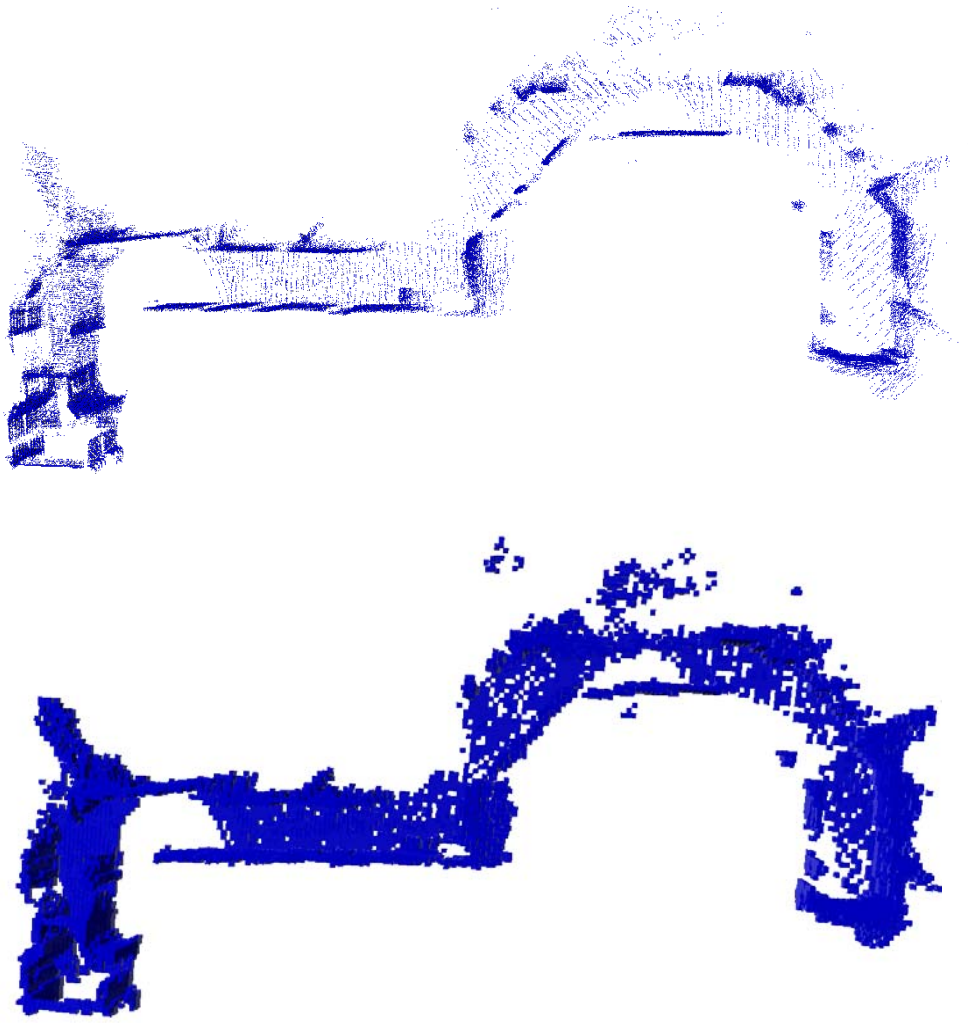

Figure 4.7 - Top view map of Glennan 3rd Floor produced using 11 vertical slices of Kinect pointcloud data. Visualized at 2.5cm (top) and 20cm (bottom).

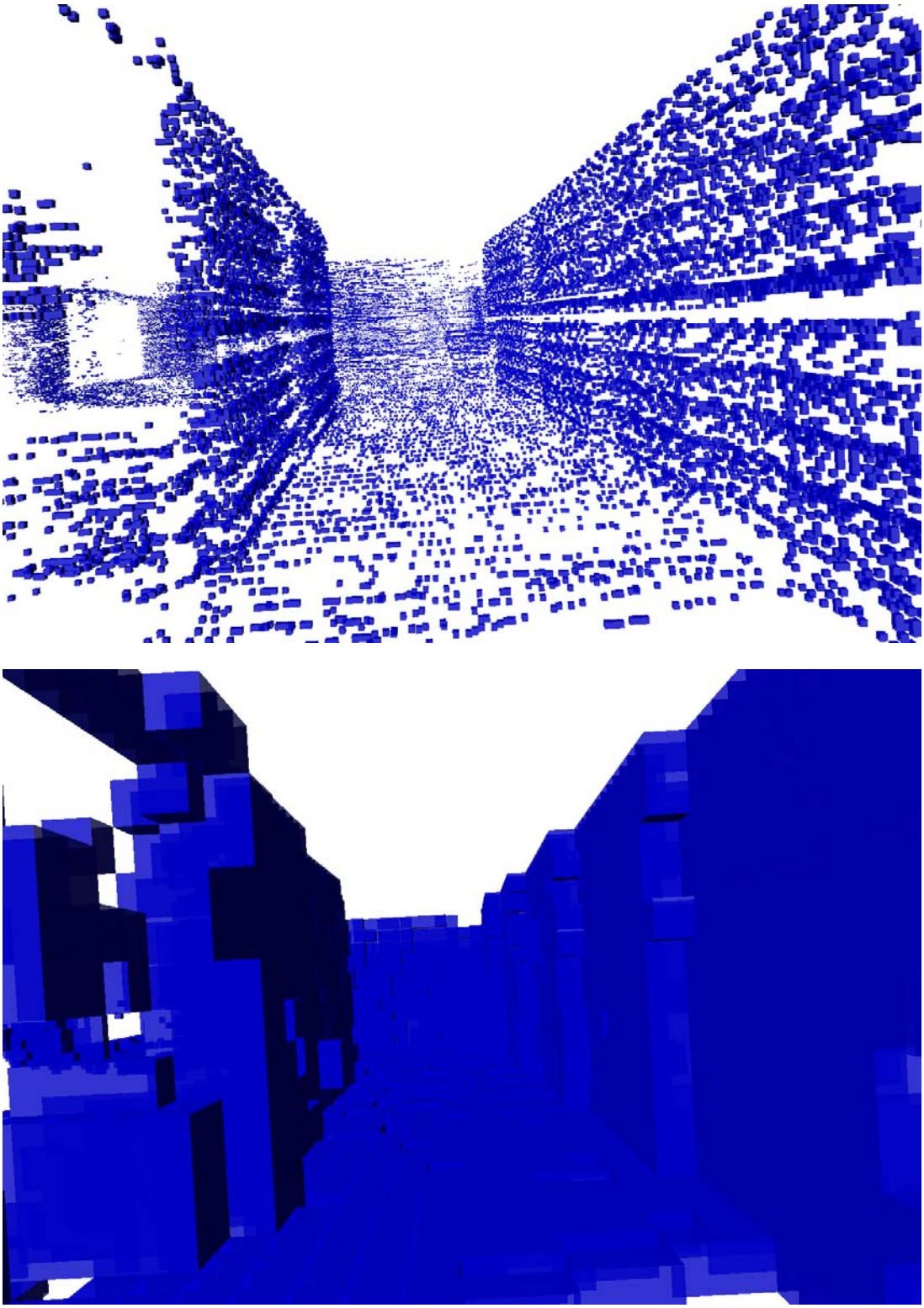

Figure 4.8 - Hallway view of map of Glennan 3rd Floor produced using 11 vertical slices of Kinect pointcloud data. Visualized at 2.5cm (top) and 20cm (bottom).

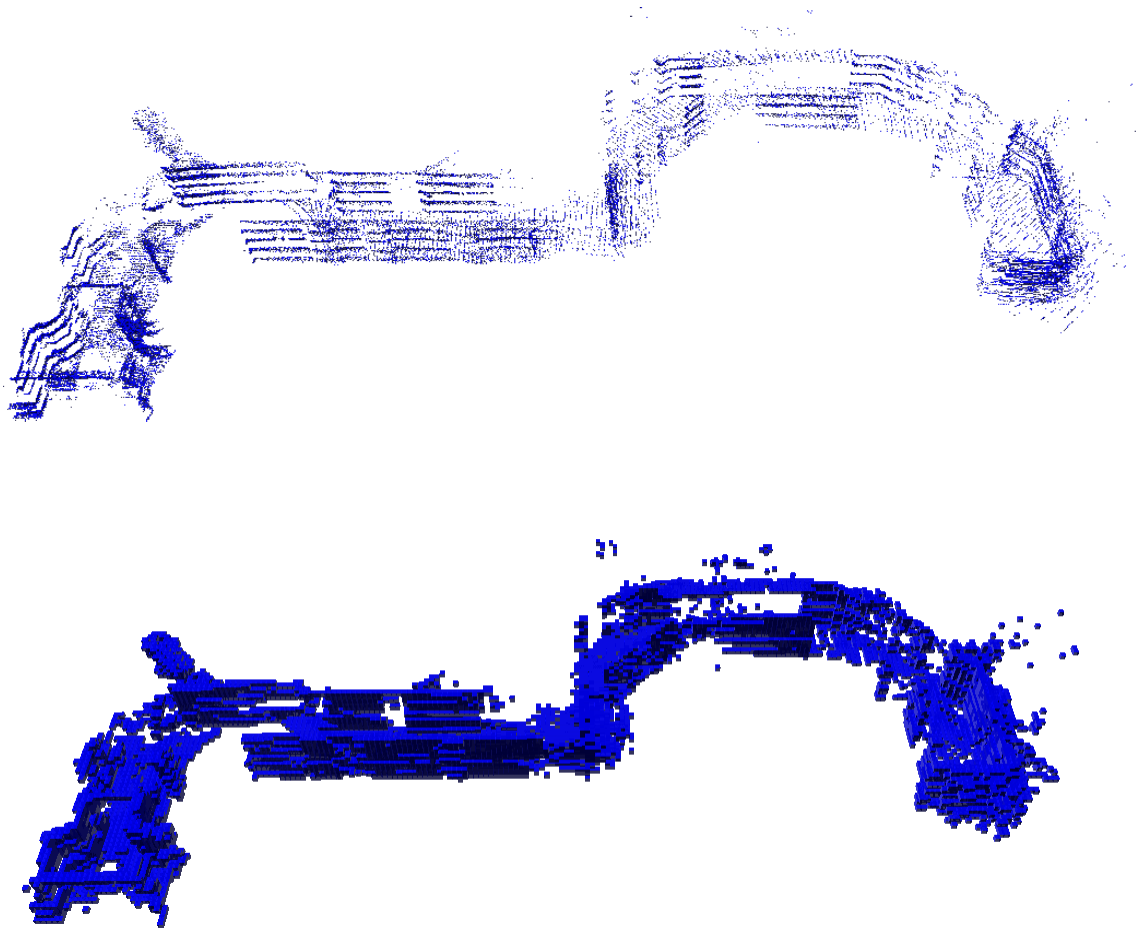

Figure 4.9 - Map of Glennan 3rd Floor produced using 5 vertical slices of Kinect pointcloud data. Visualized at 2.5cm (top) and 20cm (bottom).

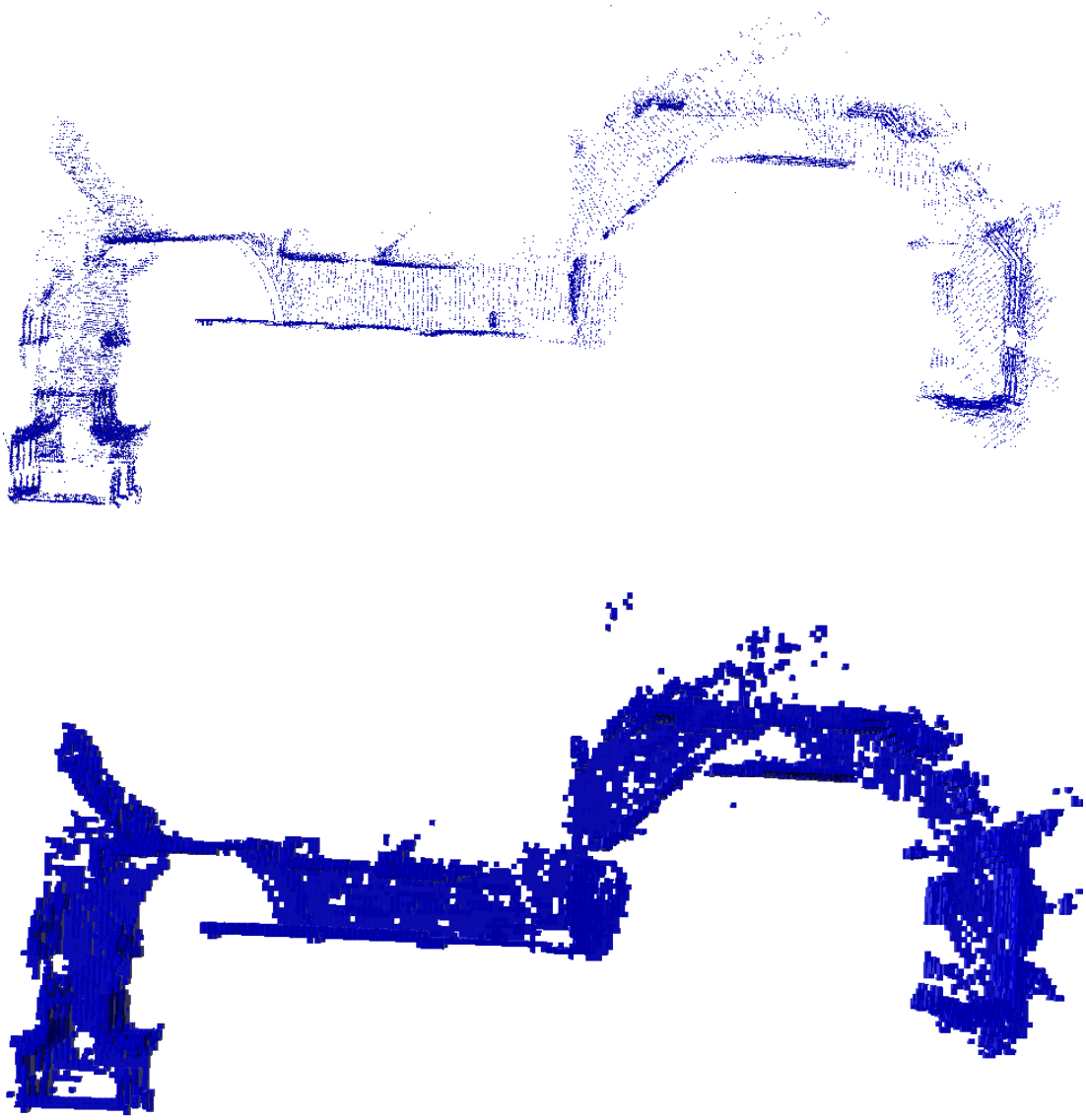

Figure 4.10 - Top view map of Glennan 3rd Floor produced using 5 vertical slices of Kinect pointcloud data. Visualized at 2.5cm (top) and 20cm (bottom).

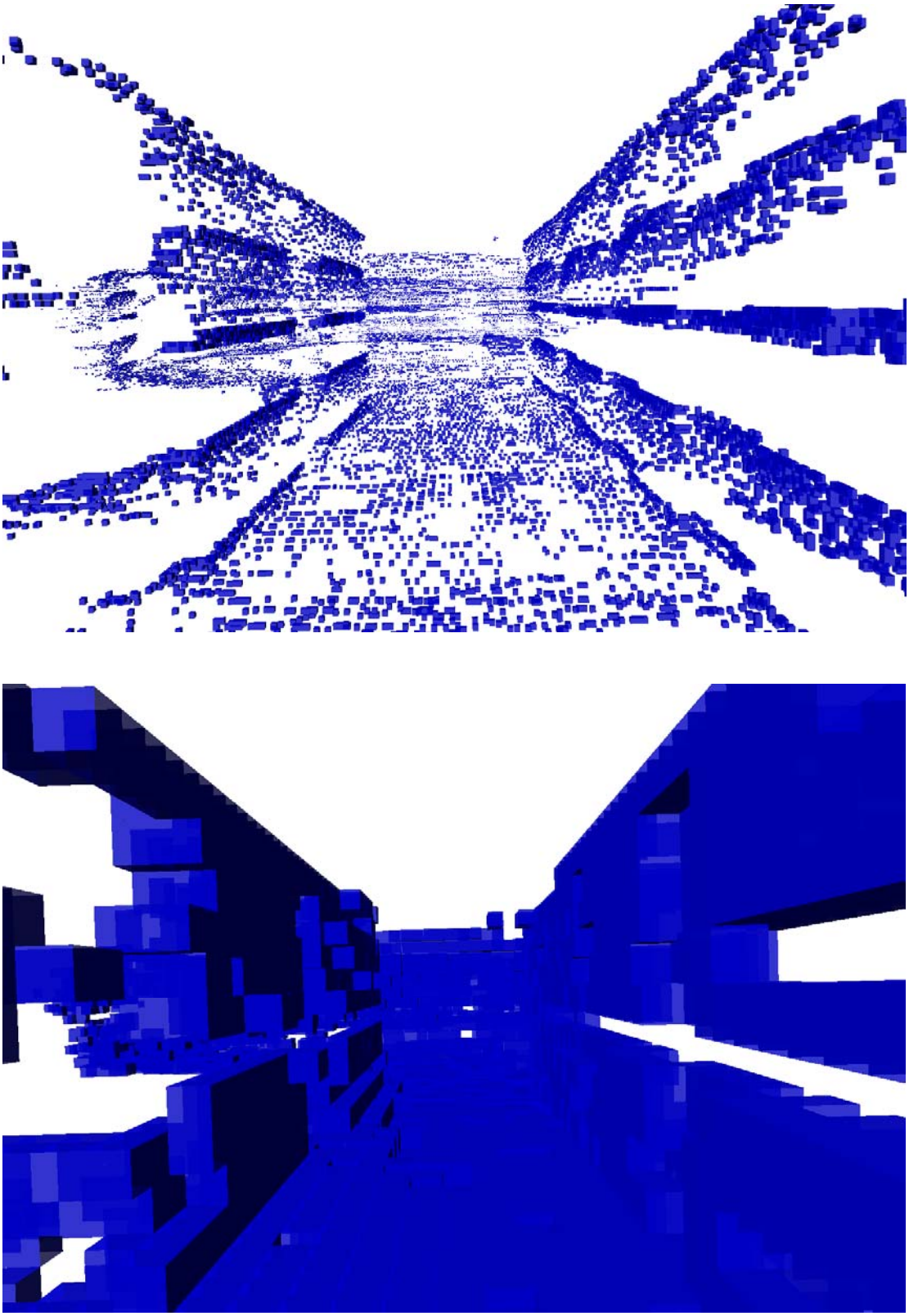

Figure 4.11 - Hallway view of map of Glennan 3rd Floor produced using 5 vertical slices of Kinect pointcloud data. Visualized at 2.5cm (top) and 20cm (bottom).

As can be seen in the preceding maps, Octomap's ability to dynamically produce useful maps at varying resolutions allows representations that use far fewer voxels than the base resolution.

It can also be seen that Octomap gracefully handles cases where fewer points are inserted per scan. In Figures 4.4 and 4.5, the full representations of the lowest-density Kinect scans were inserted into the map. This final map as shown contains 17,614,109 Octomap nodes with a combined size of 440.732MB covering an area of 25.38m x 41.45m with a height of 4.05m. Unfortunately, the large number of points per scan (~19,200) prohibited updating the map at the Kinect's realtime rate of 30Hz. It can also be seen that the Kinect mapped unnecessary surfaces such as the ceiling.

In order to not miss obstacles, each Kinect scan would have to be inserted in time for the next scan to arrive or scans would need to be dropped from the queue. The solution to this problem was to add a maximum height to the scan, above which all points were discarded, while also only taking N thin, vertical, equidistant slices of the pointcloud to insert.

The results for N=11 can be seen in Figures 4.6, 4.7, and 4.8. By reducing the number of points down to ~5000 per scan, the map was updated at the Kinect's native 30Hz rate. The completed map's size was also significantly reduced without sacrificing the data essential to navigation. The 11 slice map was only 10, 114,205 Octree nodes at a total size of 276.788MB while still covering an area of 24.80m x 39.15m at 2.45m in total height. This was a reduction of 43.6% in number of nodes and 37.2% in memory requirements. The actual area of the

map saw virtually no reduction of coverage or loss of important features, especially in the observed ground surface.

I also commonly ran with N = 5 slices. These results can be seen in Figures 4.9, 4.10, and 4.11. 5 slices resulted in only ~2000 points per scan, significantly easing processing requirements further. With this many slices, the map still required 7,828,394 nodes at 223.225MB of data covering a volume of 23.15m x 40.25m x 2.42m. The reduction in nodes was 55.6% over the full scan with a 49.4% reduction in memory requirements. While producing useful maps, some obstacles smaller than 20cm in diameter could escape detection for longer than desired until the slight downward angle of the Kinect enabled a slice to intersect with the object after some amount of vehicle travel. This effect could be easily be compensated for by modulating the slice number or adding a changing offset to the first slice location, but this was not necessary for this application.

Unlike the Kinect and lasers, sonars proved difficult to use within Octomap due to the assumption of uniform prior for the entire sonar surface. The sonars were ultimately responsible for detecting glass and other obstacles the near-IR based sensors could not. Unfortunately, the instability of detection range combined with the large detection area at far distances led to impassible maps as seen in Figures 4.12 and 4.13.

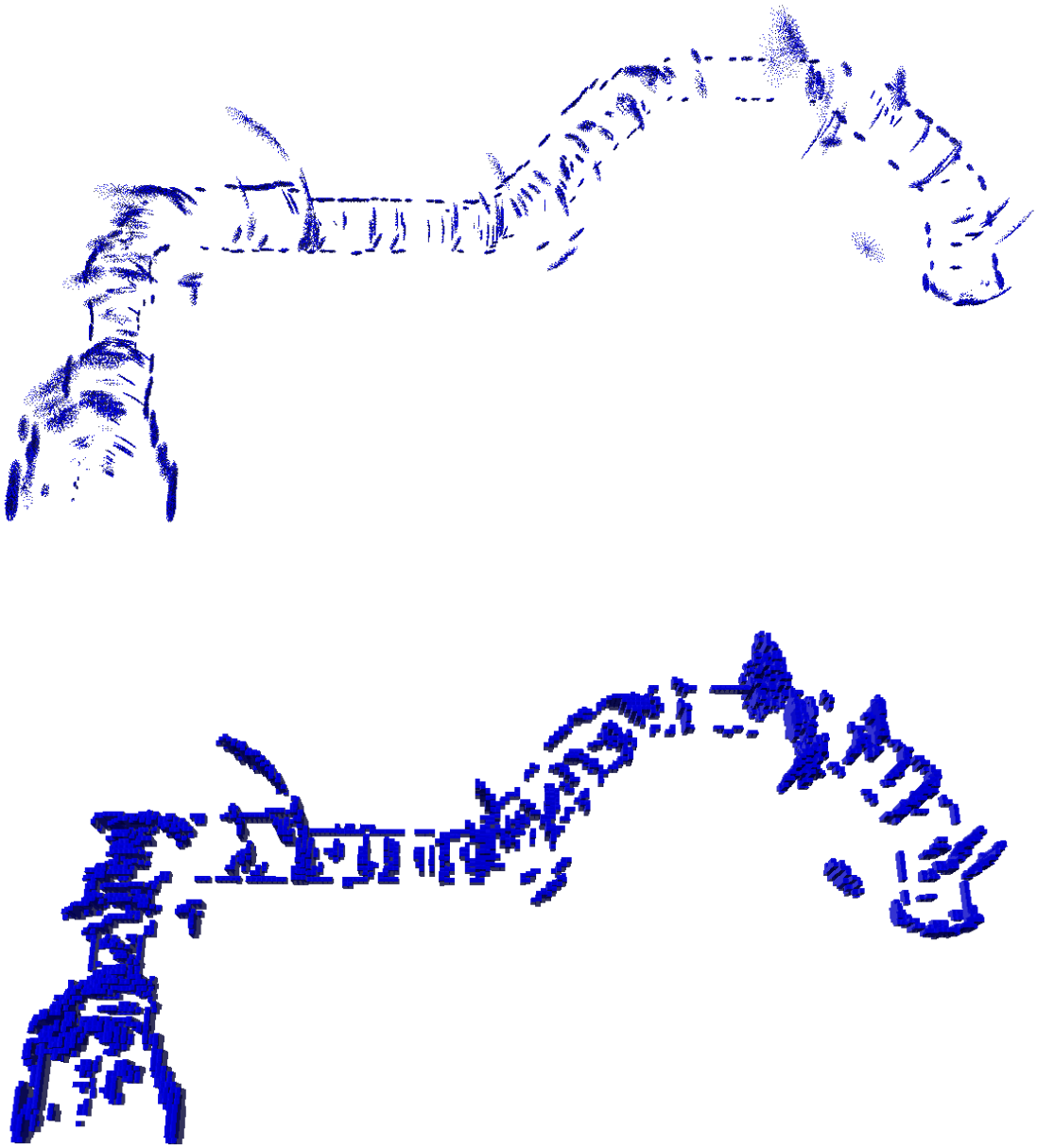

Figure 4.12 - Map of Glennan 3rd floor produced using Octomap and only Otto's sonars. Visualized using 2.5cm voxels (top) and 20cm voxels (bottom).

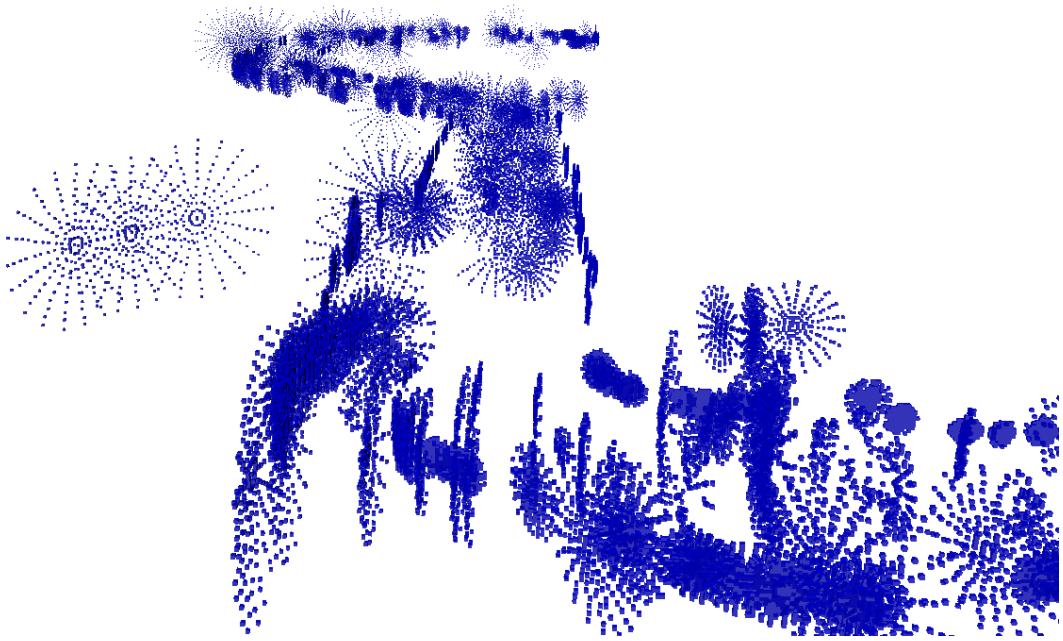

Figure 4.13 - Hallway view of Glennan 3rd floor using only Otto's sonars. Visualized using 2.5cm voxels.

While each sonar was the only sonar active during its detection period, the long, flat hallways created many acoustical reflections that made spurious measurements. Because of the time required to cycle through each sonar, the time between two consecutive readings on a single sonar was nearly one third of a second. This slow repetition frequency, combined with the extremely noisy returns, produced maps with many readings that were never cleared of noise.

Better results were obtained by limiting the maximum trusted distance for sonar measurements. The maps for this view are shown in Figures 4.14 and 4.15.

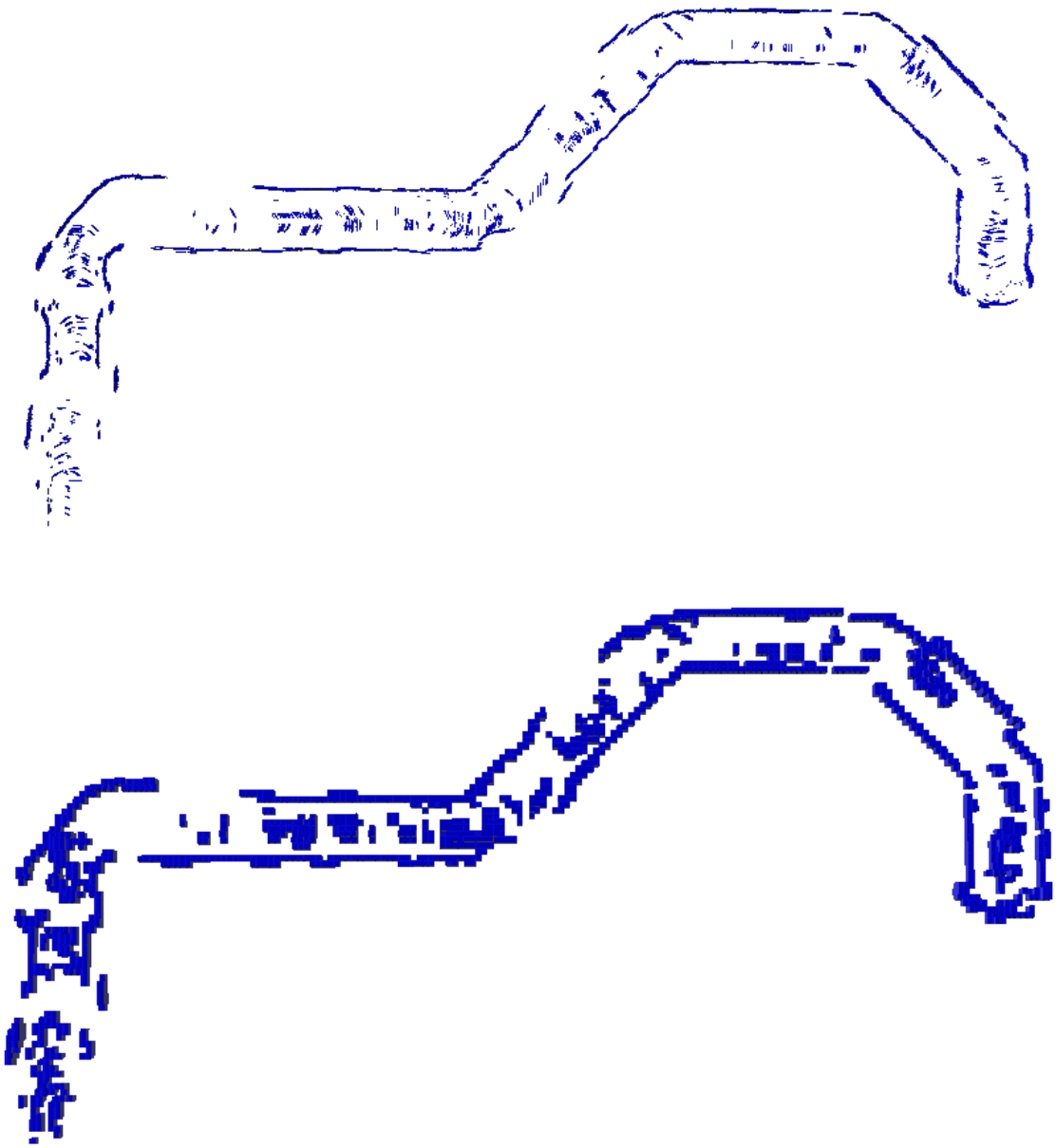

Figure 4.14 - Maps produced using only short range sonar readings. Visualized using 2.5cm voxels (top) and 20cm voxels (bottom).

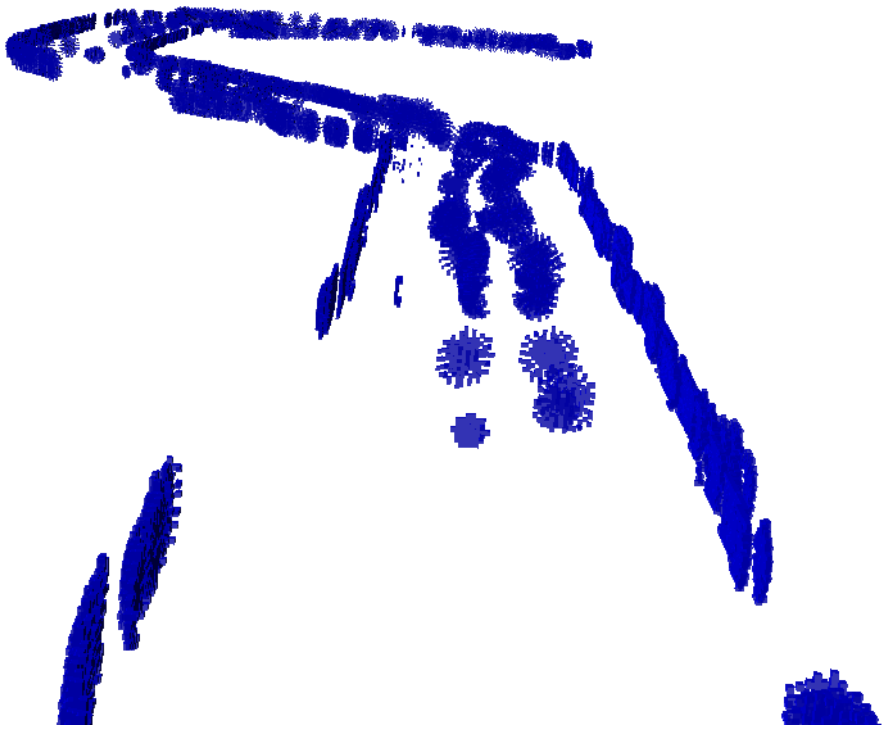

Figure 4.15 - View of Glennan Hallway produced using only short range sonar readings. Visualized using 2.5cm voxels.

These maps, while still noisy, illustrated the proof-of-concept result that the sonars did accurately and reliably detect the glass walls in the hallway (see top, right edge missing in Figure 4.9 but present in Figure 4.14). The most problematic feature of this map is caused by the two "headlight" sonars continuously producing erroneous readings. This is possibly due to the fact that these sensors faced forward into the largest areas for possible echoing returns. Unless there was an obstacle in the path, the two diagonal sonars' reflected energy did not often return to the sonar due to the angle of inflection between the sonars and walls. This prevented spurious readings, but did not often clear the readings from the front sonars. However, the two side-facing sonars typically had direct, strong reflections from walls, resulting in fewer errors.

The Octomap method provided this project the reliable, probabilistic mapping of voxels grids but with greater efficiency from the effectiveness of Octree representation. Without this approach, the large bandwidth from the Kinect sensor would quickly overwhelm the onboard computational abilities on the wheelchair, pushing the computing requirements far out of budget. The Kinect and laser showed that navigable maps could be produced within the computing and time requirements. The sonars also reliably detected their intended class of obstacles, but their high noise levels rendered maps impassible.

V. Reflexive Collision Avoidance

While approaches for obstacle avoidance can incorporate endless amounts of complexity, often the simplest approach is worth considering as a baseline measurement of what is possible. For unplanned obstacle avoidance on mobile robots, a simple approach is a threshold-based emergency stop. This approach is sometimes called reflexive collision avoidance due to the similarities with the unplanned and sudden actions of human reflexes, such as dropping a hot object or blinking one's eye. The advantages of this approach are that it is both easy to implement and conceptually simple. However, the disadvantages are numerous and include the need to constantly observe any threatening obstacle, the need for manually tuned parameters, and the dependence on sensor mounting and orientation for obstacle avoidance.

Reflexive Collision Avoidance

```
Given:
        Current Sensor Measurements, M
Do:
        for each measurement, m, in M:
                if(dangerous(m)):
                        stop vehicle in direction of m
```

Reflexive collision avoidance is based on the simple idea that if an obstacle is too close for a vehicle's sensors to determine if it is safe to proceed, then it is not safe to proceed and all motion must be discontinued. This approach works well for both accurate sensors, like laser rangefinders, and for inaccurate sensors, like sonars or radars. The computational complexity of the algorithm does not require a computer and could easily be implemented in hardware or on very simple microprocessors.

This reason makes it especially appealing for use in an inexpensive smart wheelchair, as the second most expensive add-on component for Otto was the Intel i5 computer.

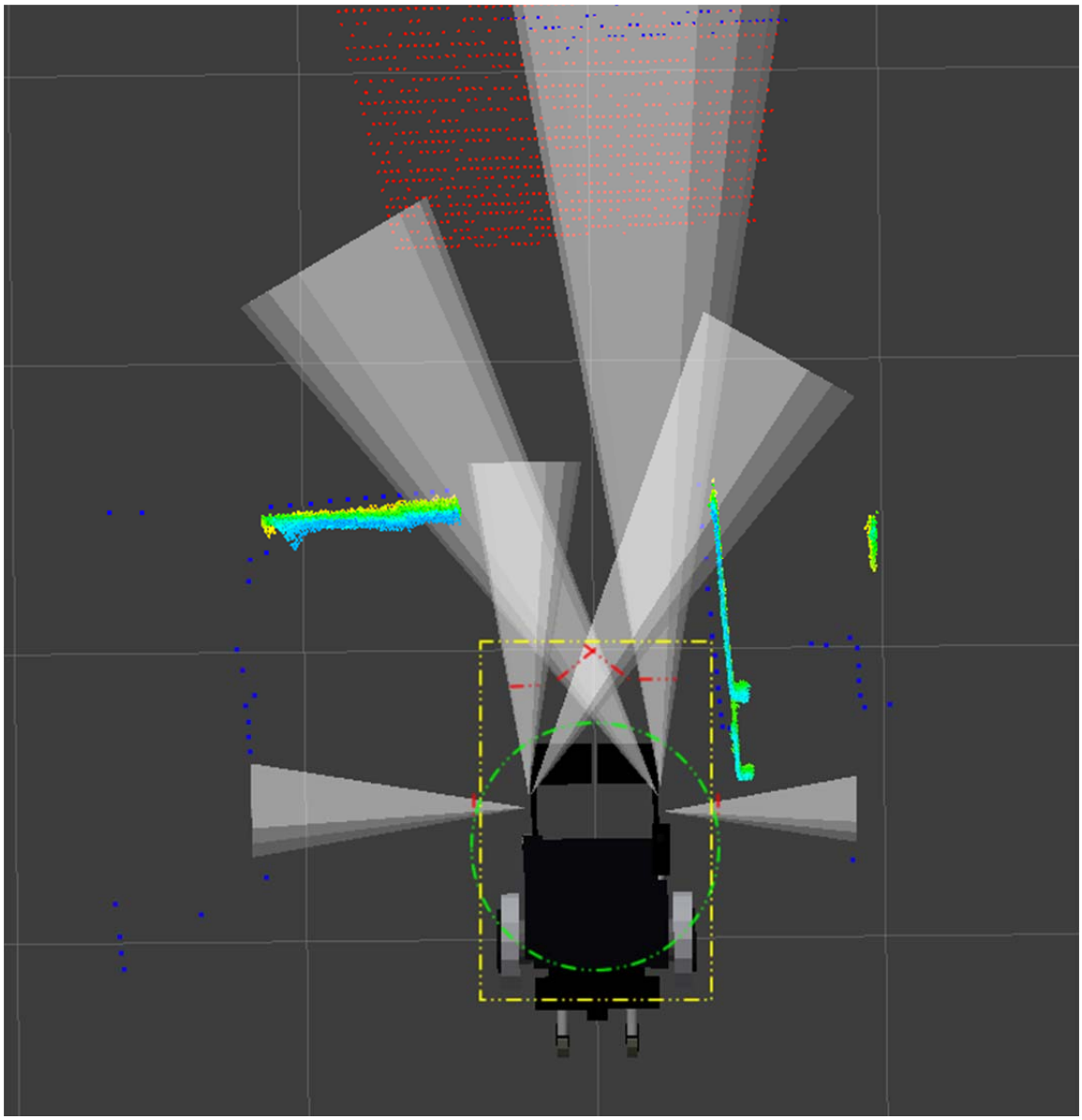

Figure 5.1 - Example Reflexive Thresholds in Yellow (Kinect PointCloud), Green (Neato Laser), and Red (Sonar Distance) dashed lines.

The implementation for Otto was fairly straightforward. Each of the sonars had a predefined threshold and a dimension of control to disable. The

forward four sonars would inhibit translational motion when a measurement reported an obstacle closer than the threshold. The two side sonars only prevented rotational motions while reporting obstacles.

The Neato laser scanner required a more sophisticated approach due to its ability to report measurements with a 360 degree field of a view. The first test was still a simple threshold, to determine whether any obstacles were close enough to actually prevent motion. Once one or more obstacles were detected as hazards, the control system needed to prevent motion towards those obstacles. This system then performed another threshold to determine the angular measurements that indicated

Sonar Safety Threshold

Given:

Current sonar measurement, **m**

Configured minimum safe distance, **d**

Configured axis of measurement, **a**

Current wheelchair command, **c**:

```
if(m < d):
    c[a] = c[a] * 0.0
return c
```

Laser Safety Threshold

Given:

Current laser measurements, **M**

Configured minimum safe radius, **r**

Angular thresholds, **a1**, **a2**

Current wheelchair command, **c**:

```
Foreach m in M
    if(m <  r):
        if(-a1 < angle(m) < a1):
            c[trans] = c[trans] * 0.0
        if(-a2 < angle(m) < -a1):
            c[right] = c[right] * 0.0
        if(a1 < angle(m) < a2):
            c[left] = c[left] * 0.0
return c
```

obstacles. If these angular measurements were directly to the front within the set tolerance, then forward motion would be restricted. Otherwise, if the measurements were outside the front angular range, but within another, larger set range, then the obstacle was off to one of the sides and the wheelchair would not be permitted to turn in that direction.

The Kinect sensor's pointclouds were the most difficult to adapt to a simple threshold algorithm, due to the number of possible thresholds available in the measurement space. To reduce computational requirements, two separate thresholds and a count were used to determine Kinect obstacles. The first threshold was to remove all measurements beyond a certain distance relative to the camera's z-axis (facing from the camera, forward into the camera's field-of-view). This distance was often not much further than two meters and so rejected most data points at this step. However, the Kinect's field-of-view was not wide enough to provide turning information from its mounting point. To remove points outside of forward collision from the data, a second threshold was implemented that removed all points outside the width of the vehicle along the translational

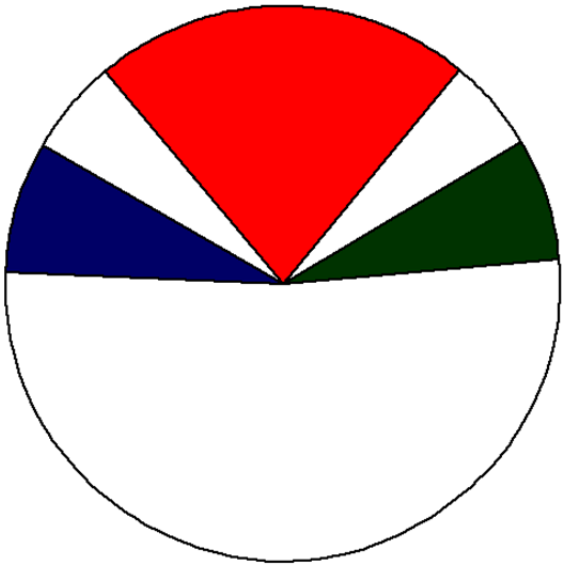

Figure 5.2 - Illustration of angle segmentation for laser scan thresholds. The red area represents radii that would provide danger to translational motion. The green and blue represent angles for left and right turns. Areas in white were not checked due to self-interference from the vehicle.

axis. The effective volume created by these two thresholds is a rectangular prism encompassing the front and center volumes of the wheelchair. If the number of points remaining in this volume is above some small noise threshold (typically 50 to 100), then the vehicle stops moving forward.

This simple approach has many advantages for efficiently processing the large number of points in each cloud measurement. The main advantage is that the first threshold removes a large number of points, easing the number of operations required to complete the check of the pointcloud. The other advantage is that each operation on physical space checks one distance axis in the Kinect's native transform frame. Transforming the large number of points before thresholding would require knowledge of the vehicle and orientation of the sensor relative to the vehicle while also greatly increasing the number of operations per point.

Kinect Safety Threshold

Given:

Current Kinect cloud, **M**

Configured safe distances, **dz**, **dy**

Number of points to stop, **N**

Current wheelchair command, **c**:

```
Foreach point p in M
    if( p.z  < 0.0 or p.z > dz):
        remove(p)

    if( p.y  < -dy or p.y > dy):
        remove(p)

if(M.size > N):
    c[trans] = c[trans] * 0.0

return c
```

The parameters chosen for each threshold were optimized to favor vehicle safety over navigational ability. If the vehicle could not closely approach

obstacles, then the risk of collision was negligible. These parameters also took into account the vehicle's top speed of ~0.7m/s while leaving a generous 0.5s for the vehicle to come to a complete stop from max speed.

The value for the forward-facing sonars was to simply multiply the 0.7m/s top speed by a stop time of 0.5s to reach a 0.35m threshold distance. The value for the side sonars was initially this same amount. However, to improve the ability for the wheelchair to escape cul-de-sac obstacle orientations, the side sonar threshold was lowered to 0.30m. These distances were mostly determined by the wheelchair's shape and dynamics. However, it often proved frustrating to move past obstacles with these values. The 0.35m threshold on the forward sonars combined with the ~30 to 40 degree field-of-view on the sonars resulted in a detection width of nearly 30cm for each sonar. Most doors and tight corridors were impassible due to the severe restriction imposed by the forward sonars, and these situations were made even worse by a near constant reporting of obstacle by the side sonars, resulting in needing to back straight out of these tight areas – not an easy task in an electric wheelchair.

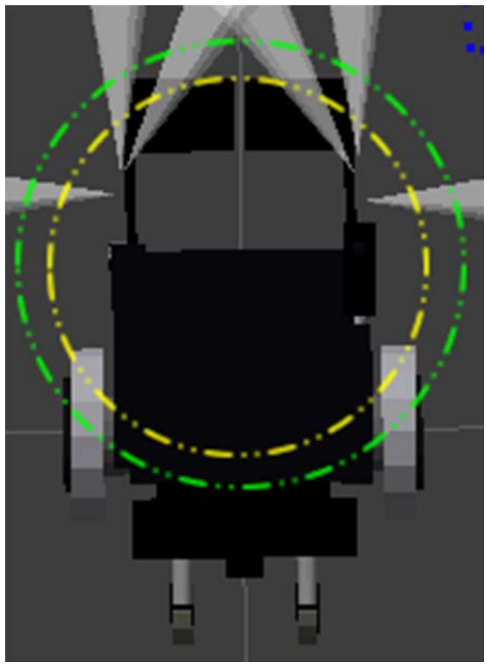

Figure 5.3 - Illustration of Laserscan detection radiuses on Otto's outline. An example front detection radius is shown in green, an example side detection radius is in yellow.

The laser thresholds required two different measurements for the front and side readings. The front collision area was

2.0 radians directly surrounding the front axis (±57 degrees from center) with a radius of 0.5m from the Neato sensor. The side detection angles extended from ±1 radian to ±2 radians (105 degrees) with a reduced radius of 0.45m. The reduced side radius improved maneuverability, since some laser obstacles could easily inhibit multiple axes of motion - something not possible with the other sensors as implemented for this technique.

The Kinect scan filters were set up to directly catch obstacles at risk of colliding with the wheelchair in areas not covered by the sonars or laser. For this reason, far more points were rejected than initially planned. The first threshold removed all points that were greater than 1.5m from the Kinect. This removed the bulk of points of each scan. The second filter removed all points further than 50cm from the side of the Kinect (conveniently, also the center of the wheelchair, making the calculations easier). At this stage, most points were removed from the cloud. Because the detection range needed to extend beyond the front of the vehicle, often the floor or ceiling would be detected. To remove these points, a final, more expensive threshold was implemented that removed points outside of the range from 10cm (Neato elevation) to 200cm (Kinect elevation). This protected the wheelchair and did not incur serious complications by needing to project the pointcloud into the vehicle's frame of reference, since typically greater than 90% of the points were eliminated before this stage. Finally, all points left were potential obstacles to be evaluated. To eliminate the potential for a spurious scan or noise suddenly halting the wheelchair, if the count of points was less than 100 at this stage, the obstacle was ignored. This method is safe because the

Kinect runs at 30Hz with maximum wheelchair travel of only 2.3cm per scan, far below the provided safety factor. Any true obstacle will easily surpass the 100 point noise threshold since the Kinect points become denser as an object comes closer, so even small obstacles will almost immediately occupy more than the 0.5% of the 19200 points per scan.

This method proved successful for not colliding with obstacles or walls in the limited areas tested. Construction barrels, A-frames, and people were all easily and reliably detected and prevented wheelchair motion.

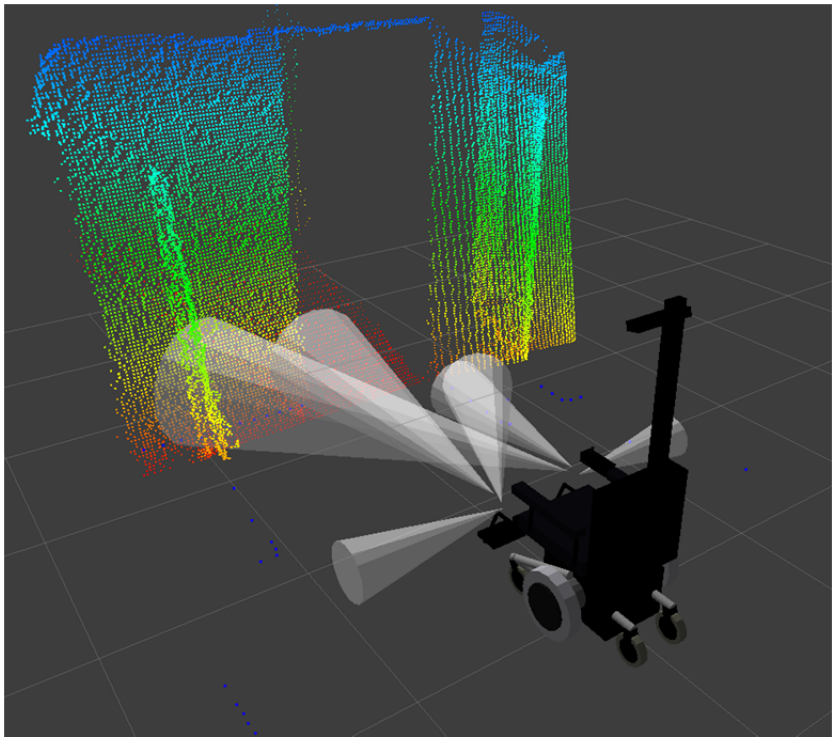

Figure 5.4 - At sufficient distance, sonars detect narrow doorways as walls. The Kinect scan can be seen projecting height (lowest in red, highest in blue) and the Neato laser measurements can be seen in blue.

However, the radii configured by calculating the a-priori required stopping distance proved too large for reliable navigation. This method could not pass through single-width doorways or through even lightly cluttered environments. Once stopped in these environments it became difficult or impossible to escape, given that the level of control was dropped immediately to zero and kept there for

as long as the obstacle was a threat. This meant that the vehicle could be entirely motionless from just a few strategically placed static obstacles.

Fortunately, reflexive collision avoidance can be easily extended to provide greater freedom to an operator while still maintaining the same level of obstacle avoidance. Reflexive collision avoidance works well in environments were the environment is known a-priori, occupied space is unexpected or rare compared to free space, or the moving platform has extremely tight controls and can react quickly to changes in environment. Unfortunately, while these conditions all apply to the typical robotic arm or manufacturing application, for a mobile wheelchair the only application for this algorithm would be in wide open areas such as auditoriums or the outdoors unless the algorithm is modified to allow greater maneuverability.

VI. Reflexive Collision Avoidance "Plus"

The main limitations of Reflexive Collision Avoidance that its influence was to enforce a full stop and that the safety limits were conservatively based on the possible need to stop from maximum speed. To improve performance of this method, it was modified to gracefully modify vehicle dynamics.

The main improvement involves limiting the influence of the least accurate sensors – the sonars. The sonars give a noisy estimate of the surrounding environment, becoming less noisy the closer an object is to the wheelchair. This is because nearby obstacles return dominant energy from true obstacles and less from multipath reflections. To improve sonar reliably and increase vehicle mobility, the sonar thresholds had to be reduced to less conservatively-safe distances. However, these sonar limits were set based on vehicle dynamics such as top speed and time required to stop. In order to change the thresholds, either top speed or time required to stop needed to be changed. Since time required to stop was programmed into the wheelchair at the factory, the speed of the wheelchair was changed as it encountered obstacles.

Reflexive Collision Avoidance "Plus"

Given:

Current Sensor Measurements, **M**

Current Vehicle Velocities, **V**

Do:

for each measurement, **m**, in **M**:

if(dangerous(**m**)):

stop vehicle in direction of **m**

else:

command new **V** = *f*(**V**,**m**)

The approach chosen was a simple and intuitive concept: as navigation complexity increases, vehicle speed should decrease.

Human drivers of automobiles naturally take this approach in complex situations. If a lane narrows due to construction or a larger passing vehicle, most drivers continue to move forward but do so at a slower pace than if the passage was not constricted. This approach works well because the driver has more time to react if something goes wrong than if the vehicle speed had not been reduced. If the lane suddenly becomes impassible, then the driver uses the extra time to stop, whereas the tolerance for driving mistakes is much lower when moving through tight spaces at the original, higher speeds. This natural compromise between speed, safety, and stopping was used to create Reflexive Collision Avoidance plus.

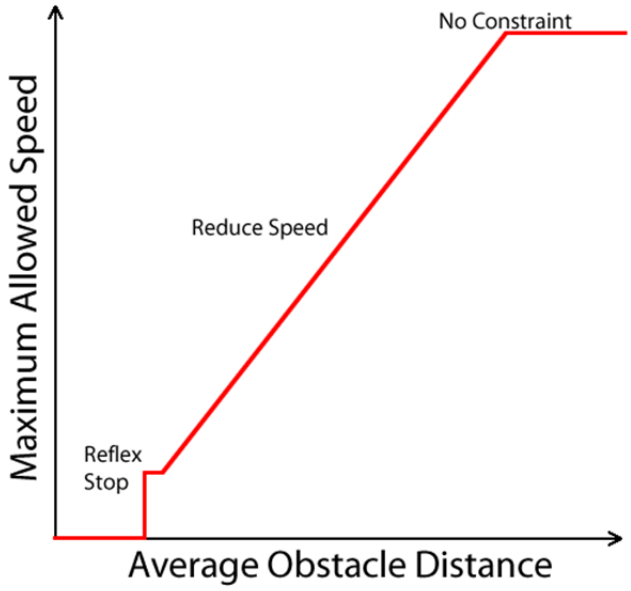

Figure 6.1 - A qualitative diagram of Reflexive Plus. The three main regions are: Reflexive Stop, Reducing Commanded Speed, and No Constraint on command.

Reflexive Collision Avoidance "Plus" (or "Reflexive Plus") introduces a new condition onto the existing algorithm. As the average sonar readings indicate the obstacle density is increasing, the maximum speed of the wheelchair is reduced. When any individual sensor measurement triggers a reflex condition, as in

Reflexive Collision Avoidance, the wheelchair will again cut off that direction of motion.

The parameters for slowing down the vehicle were the most important consideration in the implementation of this algorithm. The average distance measurements chosen for no constraint were 2.0m for both front and side sonars. If there were no obstacles for 2.0m around the wheelchair, then the wheelchair would have plenty of time to react, slow, and stop if an obstacle were to start moving toward it. The minimum average range chosen was 0.5m with a speed multiple of just 12%. This represents a translational speed reduction from 0.7m/s to just over 0.10m/s when obstacles were less than 0.5m from the wheelchair. With this approach, the reaction distance needed to stop decreased from 0.35m to only 0.05cm.

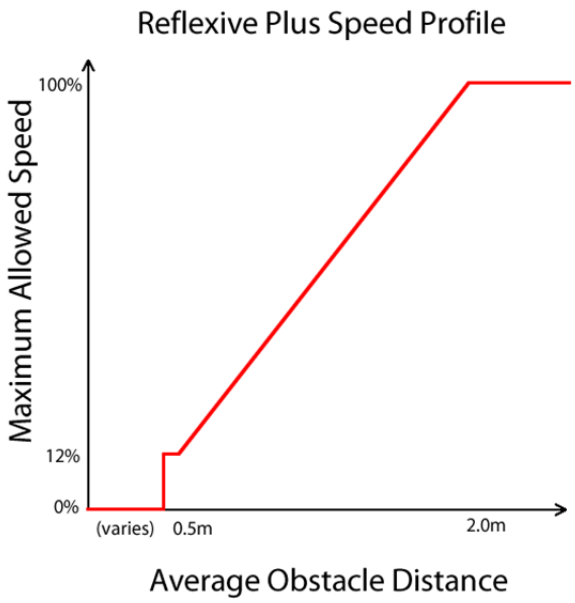

Figure 6.2 – Reflexive Collision Avoidance Plus Speed Profile Implemented on Otto. Average obstacle distance determined from averages of sonar readings.

The average obstacle distance was computed as a moving average of the some configurable amount of latest measurements. These measurements were transformed into distances from the front and from the collision boundary of the wheelchair and then averaged separately. The number of measurements in each

average was chosen to be 100 samples. Since each sonar outputs a distance at least once every 49ms, this represents a time history of at most 7.4s for the front sonars and 14.8s for the side sonars.

The moving average of closest obstacle distance combined with the gradual ramp reduction in vehicle speed resulted in smooth changes in velocity profiles. The gradual changes, as opposed to the sudden stops of pure Reflexive Collision Avoidance, helped to significantly improve rider experience. The slow changes made the wheelchair reactions less jarring to riders and also allowed some control in relatively tight spaces.

The most significant improvement to reflexive thresholds was for the sonars themselves. As mentioned previously, the only warning necessary for the vehicle's end speed was 5cm. However, the sonars had an absolute minimum reading of 15cm. The thresholds were able to be reduced to about 20cm to guarantee detection of obstacles with an appropriate tolerance for misdetection and errors caused by coming too close to an obstacle. This was an improvement of nearly 15cm over the previous method by sacrificing speed and gaining significant improvements in user experience, navigable areas, and safety.

The Neato's reflexive thresholds were not changed from the previous implementation. The Neato sensor proved most useful in mapping for determining doorway locations and as such did not play a significant role in collision avoidance. Its tolerances were not adjusted for this method as they did not falsely interfere with vehicle operation.

The Kinect point-cloud filters were able to be significantly reduced in size. Whereas the previous cloud often caused the wheelchair to get stuck in tight passages, such as doorways, this new cloud volume could tightly wrap the bounding box of the vehicle and detect three-dimensional obstacles outside of the sonar plane. The most significant gain came from the forward direction from the Kinect. This threshold was able to be reduced from 1.4m to 0.7m. This provided just tens of centimeters of redundancy at the front of the vehicle, closely mirroring the improved reflex thresholds of the front sonars. The sides of the collision boundary were also able to be brought closer together, reduced by 10cm on each side to ±40cm. Finally, the reduced distances removed the need for the point-cloud to be transformed into vehicle coordinates to remove returns from the ground or ceiling. Removing the need to check so far in front of the wheelchair dramatically improved the simplicity of the Kinect filter to something that could easily be performed without the transformation of the pointcloud into vehicle frame, enabling possibilities for future microcontroller and FPGA implementations.

While this approach greatly improved the mobility of the collision avoidance system, it did expose flaws in the sensor mounting locations that did not surface in the exploration of Reflexive Collision Avoidance. At these close ranges, not all obstacles were reliably detected. One prominent example is narrow doorways. While impassable with larger safety radii, once the radii are reduced to where the wheelchair can successfully pass through, it is possible for

the wheelchair to stop and turn in the middle of the doorway, lodging the side of the doorway between the wheel and the sonar and preventing escape.

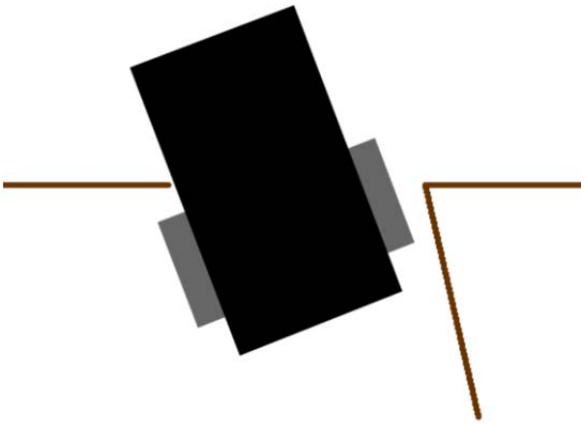

Figure 6.3 - A limitation of this approach in doorways or other areas where obstacles are not directly observable by the sensor instrumentation.

The main limitation of this approach is the lack of memory and mapping of the surrounding environment, especially using the precise Kinect and Neato sensors. However, the improvements made to comfort and maneuverability with this approach make it well suited for the environments where Reflexive Collision Avoidance was effective as well as handling areas with greater obstacle density. This algorithm also lends itself to assisting the driver in complicated or dynamic situations by automatically slowing the wheelchair, allowing the user to stay in control unless a collision is immediately imminent. Allowing the user to control the wheelchair as it slows

Figure 6.4 - Image of Otto in a situation where a doorway collision is possible using Reflexive Avoidance Plus. Remembering the environment with Octomap (Chapter VII) prevents this issue.

helps the operator move to the desired locations more gracefully with fewer sudden stops that would force the user to back up, change direction, or take an unnecessary detour.

VII. Octoflex: Octomap-based Reflexive and Predicted Collision Avoidance

The greatest benefit the Kinect provides to the wheelchair is high resolution 3D measurements at a fast frequency and extremely low cost. This information combined with the reasonably accurate odometry produces extremely accurate and high resolution maps of the environment. These maps give the best idea of the dangers for the obstacles detected by the sensors that produced the maps. These maps also allow persistence and relieve the requirement that all obstacles must be observed to stop the wheelchair. The only question is how to leverage the maps in the best way to effectively use input from all of the sensors. The eventual solution for Otto involved using aspects of Reflexive Collision Avoidance, Reflexive Collision Avoidance Plus, and Octomap and is nicknamed "Octoflex" (a portmanteau of Octomap and Reflex).

Octoflex utilizes the accuracy of the lasers and the Kinect to generate the Octomaps for this method. As seen in Chapter IV, the sonars selected for Otto do not lend themselves well to occupancy map based methods, while the laser based sensors generated precise maps that could miss certain classes of obstacles, such as those that are especially absorptive in the near-IR bands or those that are fully or partially transmissive to light (e.g. glass walls and doors). To allow for these types of obstacles and to utilize the advantages of the sonars, Octoflex utilizes Reflexive Collision Avoidance Plus for just the sonars sensors. This allows detection of those obstacles difficult to detect with near-IR light and places an extra layer of safety on top of the mapping-based techniques. The lasers still

ensure that the vehicle slows down as it approaches obstacles and will come to a full stop if an obstacle is sufficiently close. This preserves the application of the two most effective features of the sonar obstacle detection and has high reliability due to the noise characteristics of the sonars.

The generated Octomaps are used by predicting the wheelchair's future position based on its current position and velocity commands. Under the assumption that the vehicle's command will remain constant for some period of time, the future position of the wheelchair can be projected through the generated map space. If one of these future positions were to collide with an occupied Octree, then that command should not be executed, and instead the vehicle should stop and wait for a command that will not lead to a collision. This approach is similar to that used for evaluating trajectories as used in *Planning and Control in Unstructured Terrain* by Gerkey, Konolige (Gerkey & Konolige, 2008). This obstacle-avoidance approach is much simpler as only one trajectory needs to be evaluated each iteration, and the map costs can only take two forms: occupied and unoccupied. If the wheelchair did

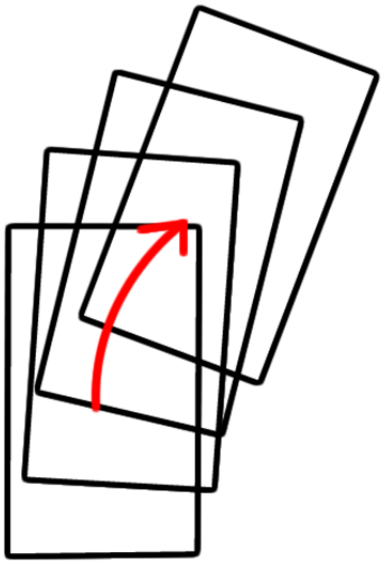

Figure 7.1 - A velocity curvature projected through time. The vehicle outlines represent future vehicle positions if the current command were to continue.

not have sonars, the gradient descent approach for reducing speed and modifying the commands around obstacles would be more essential to smooth operation.

The method for projecting vehicle position based on velocity must be a function of time projecting into the future and must assume a constant velocity throughout. An easy method for evaluating the new position given the old position, velocity, and angular velocity, v, and the angular velocity, ω, is given by equation (1) (Thrun, Burgard, & Fox, 2005):

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega} sin\theta + \frac{v}{\omega} \sin(\theta + \omega\, \Delta t) \\ \frac{v}{\omega} cos\theta - \frac{v}{\omega} \cos(\theta + \omega\, \Delta t) \\ \omega\, \Delta t \end{pmatrix} \qquad (1)$$

This equation lends itself well to being calculated with only varying Δt. The only limitation of the algorithm is that it assumes the velocities will be constant over the entire projection period. In the present case, the amount of time projected forward was typically only around half a second, just enough to give the wheelchair a chance to slow or stop. In this small window of time, the constant-speeds assumption was suitable for look-ahead on the same order of magnitude as a human's reaction time.

The method of checking for collisions is largely due to Octomap's efficient volume checking and recursive structure. The wheelchair and occupant's surrounding space is first bounded by a rectangular prism of a fixed size. These six planes form the collision volume for the wheelchair. If an obstacle is predicted to enter this collision volume, it must do so by first passing through one of the six planes. If the number of checks per second is sufficiently fast, then these planes are the only parts of the collision volume that need to be checked if an occupied unit is within the plane boundary. To further increase the simplicity of checks, the planes can be represented as voxeled units of space. These simplifications reduce the entire collision boundary to a discrete number of checks, all of which are easily mapped to Octree cells. If Octomap's Octrees all contain the same state (such as being unoccupied) then it will not need to index through nearly as many cells as a comparable non-recursive voxel map implementation, thus reducing the number of overall operations for each of the boundary points in most situations.

Octoflex

Given:

Current Sonar Measurements, **S**

Current Vehicle Position, **P**

Current Velocity Command, **V**

Vehicle Occupancy Volume, **C**

Current Octomap, **M**

Do:

Run Reflexive Collision Avoidance Plus on **S**

For each **dt** between **now** and time **t'**

Predicted Position **P' =**

P + f(V, dt)

If(collision(**P', C, M, dt)):**

Stop

Else:

Execute Command

The combination of the projection technique with collision volume surface checking is the core of Octoflex's functionality. Only through precise mapping of the environment can the predicted positions be safely evaluated. One advantage of this approach is that the extent of the memory is only tied to that of the underlying map. This means that doorways and other obstacles observed transiently are not immediately forgotten once outside of sensor detection range (or if the sensor's view is obstructed by part of the wheelchair or rider).

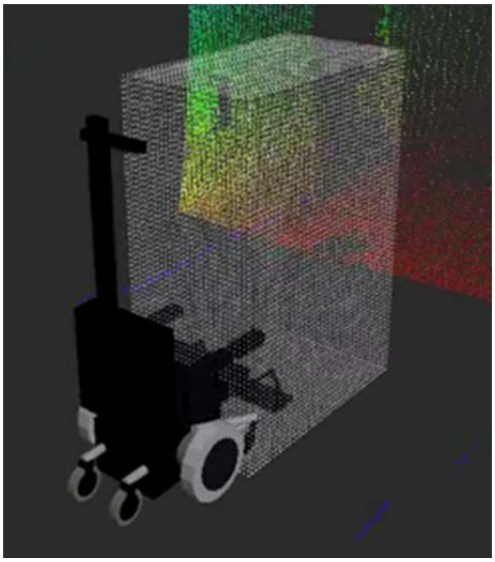

Figure 7.2 - The collision box for Otto. Note that it is currently checking the area Otto will soon enter. The raw Kinect scan and laser scan can be seen in the background.

Relative to the other methods presented, Octoflex does not require as many tunable parameters and the parameters that do need to be adjusted vary based on vehicle and not on a sensor by sensor basis.

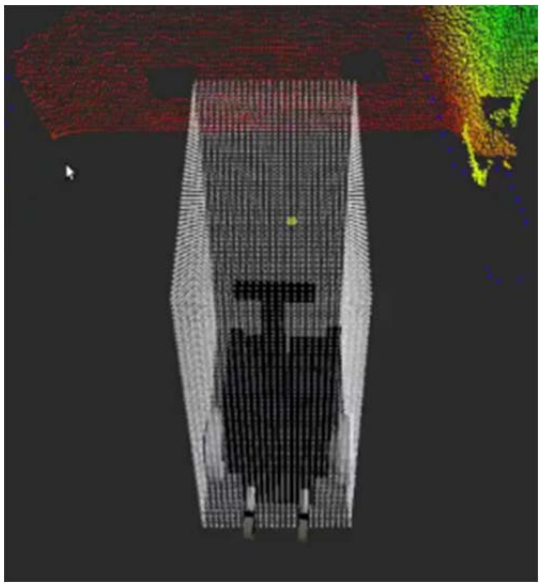

Figure 7.3 - The selected boundary box for Otto from a top-rear view.

The first set of parameters needed for this method are those used to define the collision boundary for the vehicle. Otto lended well to a

rectangular prism shape as the vehicle itself forms a large 'L' shape with the addition of the Kinect mast. The rest of the box is area that holds the occupant and should be well protected from collisions. The collision volume is tunable in the sense that it can be artifically expanded from true vehicle size to increase the safety factor associated with the method. The true bounding box for Otto is 1.14m long by 0.66m wide by 1.525m tall. The most common size of collision volume used in this work expanded each side with an extra 10cm to produce a box 1.24m x 0.76m x 1.525m. The minimum collision volume applies to slow motion while moving near obstacles. Forward projection of velocities ensures a further look-ahead distance for faster velocities. The last parameter for the bounding box is that of the cell spacing. For best results, it was found that the cells needed to be the same size or smaller than the map resolution for efficient, non-interpolating checks. For this reason, 2.5cm spacing was chosen.

The rest of the parameters concerned the actual internal operations of the algorithm. The most important of these is the amount of time to look forward for predicting collisions. A typical look-ahead of 0.5 to 0.7s was adequate since it takes Otto 0.5s to comfortable come to a full stop from maximum speed. The addition of the underlying sonar Reflexive Plus system allowed reducing this time to only 0.3s. This greatly reduced the number of forced stops if the driver temporarily commanded a fast velocity, since the sonars limit the maximum velocities thus providing a smoothing effect for Octoflex and preventing spurious emergency stops. An additional two parameters to be tuned were the number of steps to check in each arc and the rate to check for projecting if the current

command would eventually result in a collision with an obstacle. Ideally, the number of steps to check would match each step forward with that of the map resolution and each new command would be checked. However, the practical limits of the computing on Otto limited the approach to a much simpler constant number of checks at some rate. The chosen values were to check each time sample in 10 steps and to check commands at a rate of 20Hz. Combined with the 0.3s of forward checks, this meant each checked pose corresponded to 30ms increments. Since the wheelchair's maximum velocity was 0.7m/s, this resulted in a change in position of 2cm, just smaller than the typically map resolution of 2.5cm. This update rate was also synchronus with samples of the joystick input.

The main limitations of this method are the greater computational requirements and the dependency upon the underlying map for accuracy and safety.

While Octomap is efficient at mapping and checking for collisions, it is still not at the level of computation available on a cheap microcontroller. The mapping updates (30Hz Kinect, 5Hz laser) and obstacle checks (200Hz checks of ~12000 points) used roughly 3 cores on the Intel i5 processor. This amount of processingcorresponds to relatively high power consumption and expense. This means that the extra safety afforded by Octoflex is offset by shorter vehicle operation time and as of 2011, more than doubles the add-on cost for the obstacle avoidance components.

The other main limitation is that this method can only be as good as the underlying mapping, which can only be as good as the provided sensors. This

problem surfaces when running obstacle avoidance with maps created including inaccurate sonar data. These maps were impassible for the wheelchair because of the high rate of false returns and the large area false returns obstructed. Another example is the lack of many obstacles such as glass or near-IR absorbant materials in the maps. Octoflex would glady allow a user to command the vehicle into any free area in the map, even if that area is actually obstructed.

Fortunately, the problems with Octomap are solved with the underlying Reflexive Collision Avoidance Plus using only the sonar information. By not forcing the sonars to appear in the maps, Octomap produces very few false positives. By stopping the vehicle if too close to obstacles, this provides a level of redundancy if the map fails to include an obstacle (e.g. if the obstacle was of a material type that the Kinect and laser could not pick up). Finally, the slowing of the vehicle near obstacles allows two algorithms to work together to provide smooth and safe motion with greater maneuverability in more environments.

Octoflex was demonstrated to be effective using the Kinect sensor for obstacle avoidance. The combination of versatile and economical Kinect camera and appropriately tuned mapping and reflexive look-ahead algorithms offers promise in achieving the goal of affordable and effective power-wheelchair safety.

VIII. Conclusion

This thesis has presented a novel combination of low-cost hardware combined with open-source software for power-wheelchair safety. With the Arduino, the Kinect, the XV-11 Lidar, the MaxBotix Sonars, and the Sparkfun 9 degree-of-freedom Razor combined with the Robot Operating System and Octomap, this work tied together the recent improvements in hardware and software to produce a diverse and unique approach to smart wheelchairs.

At a current cost of approximately $1000, the approach merges algorithms, representations, and multiple sensors and considers multiple degrees of freedom of motion to prevent collisions.

In light of these results, the challenges to the smart-wheelchair technology are no longer technical. The technology exists to create a wheelchair that detects most practical objects, especially if placed in a semi-controlled environment such as a nursing home. There exists a trade-off in tuning the obstacle avoidance systems between safety, comfort, and precision. In order to determine the ideal amount of intervention by the safety system, patient case studies will need to be performed to determine the acceptance level allowed. It is assumed that users will accept the safety systems in order to gain access to power wheelchairs. In practice, if tuned too safe, avoidance systems can be frustrating and can impede motion through doorways; if too permissive, collisions could be possible. It is likely that a combination of some environment modification plus the smart wheelchair system will enable the optimal human computer interaction.

An essential extension of this work is detection of ledges and drop-offs. With the Kinect and sonars, detecting these dangerous conditions should be possible and useful with or without the rest of the obstacle avoidance system. Curbs, drop-offs, and stairs pose a great risk to riders of electric wheelchairs and motor scooters.

Another important extension is collision avoidance while moving in reverse. Presumably, the reverse case could be handled similar to the forward case, but typically keeping the vehicle motion safer and slower than the forward case. While moving in reverse, one is even more dependent on sensors, since the operator cannot oversee safety.

A variation that should be explored is to use the sonars to clear the Octomap, but not populate it. Adding additional sensors should also be considered, since they are relatively low cost and should be compatible with the existing structure.

Bibliography

Abbeel, P., Coates, A., Montemerlo, M., Ng, A., & Thrun, S. (2005). Discriminative Training of Kalman Filters. *Proc. of Robotics: Science and Systems*.

Del Castillo, G. e. (2006). A sonar approach to obstacle detection for a vision-based autonomous wheelchair. *Robotics and Autonomous Systems*.

Fernandez-Madrigal, J. e. (2004). Assistive navigation of a robotic wheelchair using a multiheierarchical model of the environment. *Integrated Computer-Aided Engineering*, 309.

Galindo, C. e. (2006). A Control Architecture for Human-Robot Integration. Application to a Robotic Wheelchair. *IEEE Trans. on Systems, Man, and Cybernetics*.

Gerkey, B., & Konolige, K. (2008). Planning and Control in Unstructured Terrain. *ICRA Workshop on Path Planning*.

Hoey, J. e. (2006). Obstacle Avoidance Wheelchair System. *International Conference on Robotics and Automation*.

Konolige, K. (2008). A Low-Cost Laser Distance Sensor. *IEEE International Conference on Robotics and Automation*.

Kuruparan, J. e. (2006). Semiautonomous Low Cost Wheelchair for Elderly and Disabled People. *International Conference on Information and Automation*.

Marder-Epstein, E. (2011). *base_local_planner*. Retrieved from ros.org: http://ros.org/wiki/base_local_planner

Moravec, H., & Elfes, A. (1985). High resolution maps from wide angle sonar. *IEEE International Conference on Robotics and Automation*.

Perko, E. M. (2012). Precision Navigation for Indoor Mobile Robots.

Simpson, R. e. (1999). NavChair: An Assistive Wheelchair Navigation System with Automatic Adaptation. *IEEE Transactions on Rehabilitation Engineering*.

Simpson, R. e. (2004). The smart wheelchair component system. *Journal of Rehabilitation Research & Development*.

Simpson, R. e. (2005). A prototype power assist wheelchair that provides for obstacle detection and avoidance for those with visual impairments. *Journal of Neuroengineering and Rehabilitation*.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics*. Cambridge, MA: The MIT Press.

Welch, G., & Bishop, G. (2001). An Introduction to the Kalman Filter. *SIGGRAPH*.

Wurm, K., Hornung, A., Bennewitz, M., Stachniss, C., & Burgard, W. (2010). OctoMap: A Probabalistic, Flexible, and Compact 3D Map Representation for Robotic Systems. *IEEE International Conference on Robotics and Automation*.