Autonomous Mobile Asphalt Density Profiling Robot to Reduce Worker Risk

Final Report

Nikos Papanikolopoulos

Computer Science & Engineering
University of Minnesota

CTS 23-04

CENTER FOR TRANSPORTATION STUDIES
UNIVERSITY OF MINNESOTA

Technical Report Documentation Page

1. Report No.	2.	3. Recipients Accession No.
CTS 23-04		

4. Title and Subtitle	5. Report Date
Autonomous Mobile Asphalt Density Profiling Robot to Reduce Worker Risk	June 2023
	6.

7. Author(s)	8. Performing Organization Report No.
Ted Morris, Nikolaos Papanikolopoulos	

9. Performing Organization Name and Address	10. Project/Task/Work Unit No.
Department of Computer Science and Engineering University of Minnesota 200 Union Street SE Minneapolis, MN 55455	CTS #2019059
	11. Contract (C) or Grant (G) No. (c) 1003325 (wo) 119

12. Sponsoring Organization Name and Address	13. Type of Report and Period Covered
Center for Transportation Studies University of Minnesota 440 University Office Plaza 2221 University Avenue SE Minneapolis, MN 55414	Final Report
	14. Sponsoring Agency Code

15. Supplementary Notes

https://www.cts.umn.edu/publications/researchreports/

16. Abstract (Limit: 250 words)

MnDOT pavement construction personnel have lately improved quality assurance (QA) through the use of nondestructive air coupled ground penetrating radar sensors. Although proving to be accurate, the acquisition process can be manually intensive and hazardous especially when deployed adjacent to prevailing traffic. The primary objective of this project was to deliver to MnDOT two low-cost, modular, highly transportable, mobile robot platforms designed specifically for pavement density profile testing. Several field tests were performed to assess feasibility of the platform under different operational scenarios. Modularity was ensured by integrating separate, distributed, plug-and-play modules that could be reused for other mobile platforms, should the need arise for future implementations. By implementing two robots, the transferability of the architecture was demonstrated. The mobile robotic platforms were purposely assembled from widely available, low-cost, commercial, off-the-shelf components to minimize overall cost, recognizing that the landscape for such platforms has been evolving rapidly.

17. Document Analysis/Descriptors	18. Availability Statement
Mobile robots, Pavements, Quality assurance, Road construction, Ground penetrating radar	

19. Security Class (this report)	20. Security Class (this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	110	

AUTONOMOUS MOBILE ASPHALT DENSITY PROFILING ROBOT TO REDUCE WORKER RISK

FINAL REPORT

Prepared by:

Ted Morris
Nikolaos Papanikolopoulos
Department of Computer Science & Engineering
University of Minnesota

JUNE 2023

Published by:

Center for Transportation Studies
University of Minnesota
440 University Office Plaza
2221 University Avenue SE
Minneapolis, MN 55414

ACKNOWLEDGMENTS

We would like to first thank the MnDOT Materials and Road Research section Technical Advisory Panel (TAP) Managing Lead and Research Engineer Kyle Hoegh, Research Engineer Shongtao Dia, and Research Manager Jeff Brunner for their extensive and invaluable feedback, direction, and support throughout the course of the project. In addition, we would like to thank Mercedes Maupin, an engineer at MnDOT Materials and Road Research, for her field operational deployment assistance and feedback. We would also like to acknowledge the cooperation and assistance from Hennepin County Project Engineers Zachary Rothstein and Joshua Jansen for their time and permission to coordinate access to road construction sites for robot verification and testing. Finally, we would like to thank the University of Minnesota Robotics Institute (MnRI) and the Center for Transportation Studies (CTS) for their resource and administrative support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

GPR	Ground Penetrating Radar
DGPS	Differential Global Positioning System (plus, other global positioning satellite constellations)
MnDOT	Minnesota Department of Transportation
DMI	Distance Measuring Instrument
MPH	Miles per Hour
LBCS	Local Body Coordinate System
UVC	USB Video device Class
LWIR	Longwave Infrared
NDE	Non-Destructive Evaluation
ROS	Robot Operating System ecosystem (ros.org)

EXECUTIVE SUMMARY

This study, *Autonomous Mobile Asphalt Density Profiling Robot to Reduce Worker Risk*, is undertaken in response to a solicitation by the Minnesota Department of Transportation (MnDOT) Pavement Materials and Road Research section to develop a robotic platform for nondestructive asphalt structural evaluation using ground penetrating radars (GPR). The state highway system road pavements in Minnesota are aging; more than half of the 29,362 lane miles were constructed before 1970. Consequently, according to the *2020 Minnesota Pavement Condition Inventory Report*, 325 roadway miles will reach the end of service their life annually through 2024 [1]. New roadway service life and their associated cyclical maintenance are influenced by construction methods and their adherence to standards for the end product. It is imperative, therefore, to apply quality assurance and control measures during the construction process to proactively determine problems and address them at this stage, rather than after the roadway is built.

Accordingly, a myriad of non-destructive evaluation (NDE) technologies has been devised over the years. Ground penetrating radar is one such NDE technology that has seen considerable attention for quantitatively determining the level of deviation of substructure material properties from known standards (primarily the level of compaction and air void properties). It has been well established that GPR can capture such measurements continuously across a wide area to give a complete detailed evaluation. In context to new asphalt construction, current best practices for operating the GPR technology require extensive manual labor to traverse the system across large areas, which can expose the operator to traffic hazards as well as ultimately limit scalability. A low-cost, highly transportable, open architecture, autonomous robot platform is proposed and evaluated specifically for GPR sensor payloads that will ameliorate this aspect of required human labor and address the NDE technology scalability. Furthermore, such a platform potentially will enforce consistency in the measurement process.

In this study, two transportable, battery-operated robot platforms were delivered and tested on both existing, and newly constructed pavement. The primary inputs for the robots were the roadway centerline design plans and other user input parameters to generate appropriate path planning to cover a desired pavement area. Results indicated persistent lateral accuracy well within 6 inches during the traversal process. Another QA aspect was construction adherence, which was to measure as-built joint line deviations relative to the design imposed joint lines. A robot sensing and detection architecture was tested for this purpose, with a future objective to eventually alter the path planning traversals of the robot. The feasibility was established, and the design of the module will support the future objective to alter the traversal behavior to compensate for the detected joint line location deviations.

Future testing is recommended to further refine traversal behavior and accuracy, through user tunable parameters, as well as to ascertain typical fresh asphalt joint line deviations at other construction sites. The GPR payload is a commercial end-product, which while accurate, has a closed architecture that is not favorable for seamless integration with the robot software and hardware architecture. A more open GPR system would benefit and significantly enhance the operative capabilities of the robots, as well as theoretically simplify their deployments at job sites.

CHAPTER 1: INTRODUCTION

The state highway system road pavements (29,362 lane miles, or 14,331 roadway miles) in Minnesota are aging. More than half of the pavements, or 61.4 percent, are between 40 and 80 years old, with 325 roadway miles determined to reach the end of their service life annually through 2024, according to the 2020 Minnesota Pavement Condition Inventory data [1]. To maintain the pavement infrastructure to adhere at least to US DOT FHWA standards that no more than 2 percent of all Interstate highways be categorized in poor conditions, more than 4 percent and 8 percent of non-Interstate National Highway System (NHS) and other non NHS roads, respectively, will require road repair and restoration investments totaling $12.2 billion dollars by 2032 [2, 3]. This is most acute in areas with the highest populations and high traffic loads. Many other states face similar changes, albeit with different needs and financing [4]. Critical for such investments is to ensure that pavement road life is maximized, while the ensuing maintenance is minimized.

It is of high importance then to implement a process quality control and assurance method to intervene during construction to ensure that the roadway will likely meet expected service life and lifecycle costs. Of note is that quantifying road quality to identify areas for current and future critical maintenance needs is now a regular practice conducted regularly by many transportation agencies, using commercially available sensors that measure surface quality (roughness), surface failure levels, and other pavement distress characteristics (cracks, rutting, potholes) using a laser height scanner, a 3D laser camera, an inertial profiler, and a camera sensor, coupled with GPS data to perform geospatial analysis [2, 1]. The activity and sensor system do not evaluate the conditions of the underlying pavement structures, which may be a causal factor for degradation. They are essentially after-construction measurements and thus cannot inform the construction work while in process.

Pavement service life and lifecycle maintenance are highly correlated with asphalt compaction density levels achieved during the road construction process. Studies indicate, for example, with dense graded asphalt, if the density is too high (3 percent or less, of the volume is air voids, relative to a theoretical maximum density), it results in shoving and rutting, and if the density is too low (8 percent or more) it leads to cracks, and holes (deterioration) from increased permeability and air voids and shear strength reductions [5]. Compiled data from asphalt concrete roadways indicated a 10 percent loss of expected service life for every 1 percent increase in air voids above 7 percent [6]. Measuring such structural properties has been done in practice using the destructive technique of cutting out core samples at several points spatially distributed across very large distances (on the order of hundreds to thousands of feet) [6]. The densities across different layers can be accurately obtained with coring during the road construction process but are measured after the fact, in a laboratory setting, and the coring points are distributed over very large distances. Such a destructive testing process can also damage the asphalt substructures. The remainder of this chapter will summarize non-destructive evaluation technologies (NDE) for assessing road subgrade structural properties.

As opposed to measuring asphalt pavement compaction properties, the elastic modulus of the pavement can be used to predict pavement longevity with repeated exposures to load stress from vehicle traffic. One of the earliest NDE technologies, the Benkelman beam, measures surface

deflections from fixed or very slow-moving loads [7]. The method is very time consuming and error prone (it fails to estimate moving load bearing dynamic effects). Accordingly, dynamic deflection-based methodologies have been developed to emulate similar dynamic loading conditions from a moving vehicle, the most prevalent being a falling weight deflectometer (FWD). The dynamic deflections are measured through a longitudinal spatial distribution of geophonic sensors along the roadway [8]. The deployment requires several workers (potentially exposing themselves to hazardous traffic conditions), lane closures, and limited sampling coverage. The deflection loading process itself can also damage the pavement surfaces, rendering the approach to be a destructive measurement technique. Instead, Mostafa et al. [5] devised and evaluated a pavement management system based on rolling wheel deflectometer (RWD) data, that correlated deflection measurements from a heavy moving vehicle, equipped laser distance sensors, with structural properties of the road with validation from FWD data. The results indicate variations from actual measurements of 15 percent, which are claimed to be suitable for road inventory purposes.

Electromagnetic (EM) sensors offer a key advantage by eliminating the need to deploy any load bearing apparatus. One such sensor is a nuclear gauge testing device, which measures the reflected return signal from emitted gamma rays into the pavement. Because it contains a radioactive substance, it requires specialized personnel and storage requirements to operate and store the sensor. Note that it is also a point-based sampling method and cannot differentiate layers. The electrical density gauge is a non-radioactive EM device that measures asphalt impedance, as a function of a supplied alternative current at known frequencies. The dielectric constant (electrostatic energy density) can then be deduced by the impendence [9]. Reportedly, there remains questions regarding the accuracy and practical usability of the approach [10]. Both sensing methods are point-based measurements and thus a more wholistic evaluation of the roadway cannot be completed. Such aspects are not favorable for integrating on an autonomous robotic platform.

Sensing in the very long wave EM spectrum has also been widely studied with the premise that the emitted energy can penetrate (see through) materials. In particular, research and development have focused on applying thermal imaging sensors, and the very low EM frequency spectrum of radar technologies (300 MHz ~ 3 GHz). Thermal camera sensing for the hot mix asphalt (HMA) road construction process was proposed to identify large temperature thermal segregation in the mixture and mat; the premise is that colder mix is more difficult to compact and therefore results in a sub-optimal roadway service life [11]. Researchers have proposed an unmanned micro airborne system integrated with a thermal infrared imaging IR camera with principal component thermography techniques to remove noise, detect and locate large roadbed voids that may otherwise create costly repairs, and even prevent complete roadway collapse and sinkholes [12]. Their approach focuses on locating subgrade voids associated with damaged drainage pipes and culverts and not uniform pavement density profiling. One issue with such methods is that the thermal distribution of the rolled pavement is affected by environmental conditions.

Ground penetrating radar (GPR) sensing systems have been widely applied and studied for roadway structural evaluations. The general principal is that the return emission characteristics from an EM pulse that is emitted (on the order of nanoseconds) perpendicular to the surface, reveals both dielectric

properties of the material as well as boundaries between materials with different dielectric properties. The return signal contains several temporally spaced Mexican hat peaks with varied width and amplitudes, which have been used to disambiguate subgrade structural information through applied EM wave principals and other theoretical assumptions concerning the structural medium (discussions on the signal processing, representations, and theory can be found in [13, 10]). In particular, GPR systems are either ground-coupled or air coupled. Ground coupled GPR, as the name implies, juxtaposes the radar emitting antenna next to the material surface, while air coupled GPRs locates the antenna several inches above the surface (typical for roadway structural analysis). Ground coupled GPR can penetrate deeper than air coupled GPR, but the data acquisition process is much slower. Like the RWD system, air coupled GPR systems can acquire dielectric data at driving speeds like the aforementioned wide area road inventory systems. The Georgia Department of Transportation recently deployed a 400MHz-2GHz horn antenna air coupled GPR system on a vehicle to predict construction site subgrade soil density properties for flexible pavement roadways estimated from the dielectric constant and an EM mixing dry medium model [13]. The system was then assessed on 570 miles of existing traffic lanes to estimate soil moisture level and compaction. The compaction trends were consistent with FWD pavement stiffness modulus data taken in the same locations. In [14] an array of monostatic air coupled GPRs were used to estimate pavement thickness within 1 centimeter as well as estimating the relative dielectric constant of the pavement. The array consisted of 6 emitter and 6 receivers, spread apart by an estimated effective aperture. Several existing and proposed EM mixture models were used to estimate the HMA bulk specific gravity from the GPR top surface and bottom reflection amplitude data. The vehicle mounted air coupled GPR system (2 GHz Radar) data predicted compaction levels no worse than 0.8 percent from core sample estimates. Feature extraction of proposed B-Scan GPR data, collected across multiple frequencies from 0.5 to 6 GHz, on concrete roadways was used to detect and locate sub-surface delamination voiding with centimeter level accuracies [15]. (MnDOT demonstrated the utility of wide-scale GPR measurements to adapt road construction practices, in addition to QA/QC activities, during construction. Specifically, MnDOT used the GPR measurements of the surface dielectric constant that informed a relationship between roller speed and pavement temperature, when taken shortly after the roller pass [16]).

As far back as the early 1990s, the FHWA had prioritized the study of robotics to be used in all phases of highway transportation including quality control, inspection, and monitoring. As well, emphasis was placed on acquiring as-built information as construction work occurs to reduce reinspection time or corrective operations [17]. The GPR evaluative techniques, although potentially very accurate, require on-site personnel to acquire the data. Although collecting the GPR data from vehicles is a common method, it is difficult to control for collecting data next to prescribed constrained and unconstrained asphalt joints to assess proper compaction, which is a critical criterion for quality assurance during construction. This is one of the primary reasons GPR systems have been deployed on manual pushcarts [16]. However, such a manually intensive procedure can expose operators to traffic hazards. (Over the last decade, an average of 55 workers have been killed by impeding vehicle crashes.) Second, without considering potential hazards, scaling up the latter method introduces challenges, since the detection measurement coverage is bounded by practical time and working limitations of the human operator. A

transportable robotic platform could enhance the human operative aspect of the task. The next chapter will summarize robotic and automated measurement solutions for pavement inspection related tasks.

CHAPTER 2: BACKGROUND

Robotic platforms have been designed and tested for roadway infrastructure inspection tasks. Shim et. al [18] developed a concrete distress inspection robot. A treaded skid steer robot, equipped with a robotic arm positioning a stereo camera, can autonomously traverse within tunnel environments through 3D LiDAR guided slam. A deep Generative Adversarial Network was developed to support crack localization and identification, with more precise measurements and classifications obtained through dense accurately scaled Structure from Motion point cloud features registered with the camera data. The robot architecture allows offsite inspection managers to remotely control the robot. Yuan et. al [19], also developed a vision based reinforced concrete damage assessment ground robot that is manually operated, to dynamically build a 3D structural model of the inspection environment, followed by crack identification and detection , localization, and finally volumetric quantitative assessment, through a trained (both stereo camera depth map features and RGB camera) Mask region based Convolutional Neural Network (R-CNN). Other robots have been proposed which add roadway garbage detection, in addition to crack inspection. Panthera, a geometrically reconfigurable articulated robot based on the Robot Operating System (ROS) architecture, self-reconfigures its footprint shape to reposition multiple sweeper heads as well as adapt to unique turning radii requirements. The robot, which is roughly the size of a golf cart, can also autonomously operate on human occupied sidewalks carrying out the same function [20].

Researchers have proposed motion planning strategies that adapt according to asphalt distresses that are detected in real-time [21]. They note that, unlike vehicle instrumented pavement inspection, which is best suited for large area coverage assessment, the robot was designed to conduct smaller localized assessments required for proficient QA tasks. A skid-steer robot instrumented with camera sensors, IMU and RTK differential GPS with virtual reference station (VRS) corrections was used to validate computer simulation model predictions that incorporated real-world data, and adaptive traversal behaviors to efficiently fully cover localized areas. In their case, pavement distress was detected and classified base on distribution and density of image-based corner features. Other researchers developed a two-wheel differential drive micro robot acquired image data from a mounted cell phone. The acquired images were uploaded wirelessly to a host PC which contained a CNN trained to identify cracks, with an overall accuracy of 97.5% [22].

None of the robotic proposed robot systems integrated sensors that could assess pavement structural quality. A prevalent motivation for robotic systems with such added sensor capabilities has been bridge deck inspection. La et. al [23], first developed a holonomic omnidirectional robot platform with a ground coupled GPR, seismic resonant method Impact Echo (IE) sensors, an Ultrasonic Surface Wave sensor, an Electrical Resistivity (ER) sensor, and a camera sensor for visual detection of surface cracks. The ER sensor readings are affected by levels of corrosion in steel iron rebar. A 6 degree of freedom IMU and RTK differential corrected DGPS sensor are used for robot navigation. They reported there are frequent cases where the DGPS data become unreliable when on bridges, for example the DGPS reception is degraded by frequent overhead structures on the bridges themselves. An Extended Kalman filter design was implemented to retain accurate positioning during such conditions by innovating its

state predictions using IMU sensor observations and updating predictions from the wheel encoder data. Additional studies with this robot collected B-scan ground coupled GPR data, fused with ER readings from robot traversals on bridge decks, and devised a naïve Bayes classifier using HOG features to discriminate subgrade rebar locations, followed by maxima suppression of the B-scan associated parabola location to refine the locations further [24]. The accuracy performance demonstrated results that were as good, and even better, than a leading commercial system. A commercial autonomous road pavement robot (Guimu Robotics, Shanghai, CN) integrates 2D/3D vision surface inspection, and three dimensional GPR in different frequency bands to synchronously acquire road appearance and below grade structural data. Proprietary deep learning algorithms discriminate structural deficiencies. The LxWxH = 4.7 x3.9x3.3 feet base platform uses a holonomic omnidirectional motion design. Although the payload is capable of carrying 300 Kg (661 lbs), the base weight is 490 Kg (1080 lbs) Note that the robot developed by [23] is a slightly smaller platform with a base weight of 300 Kg (661 lbs).

To conclude, sans for a few more recent works, most robotic platforms that have been proposed have primarily focused on surface level aberrations that are relevant to either flexible or rigid roadways. Such measures would be of less value for new pavement construction QA/QC tasks. The latter robots developed with NDE technologies including GPR have impressive results albeit their size and weight prohibit them from being an easily transportable platform (for example with a private vehicle).

CHAPTER 3: ROBOT DESIGN

3.1 INTRODUCTION

Two robot platforms were designed to deliver open architecture, modular designs for both hardware and software, to facilitate adaptation for future autonomous mobile pavement quality assessment (QA) platforms. Both robots accommodate a payload of 100 lbs. and can be partially disassembled for transport with a private vehicle to and from testing and job sites. A second robot design objective for the study was to utilize and test low-cost sensor and processor hardware to assess their ability to reduce barriers for adaptation of such a platform for broad use. The overall processing and communication architecture supports a multi-host ROS distributed computing framework (figure 3.1). Note that the GPS real-time kinematic (RTK) differential centimeter corrections are obtained through the Minnesota Continuous Operating Reference Station (MnCORS) Virtual Reference Station (VRS) Network, querying a NTRIPS server.

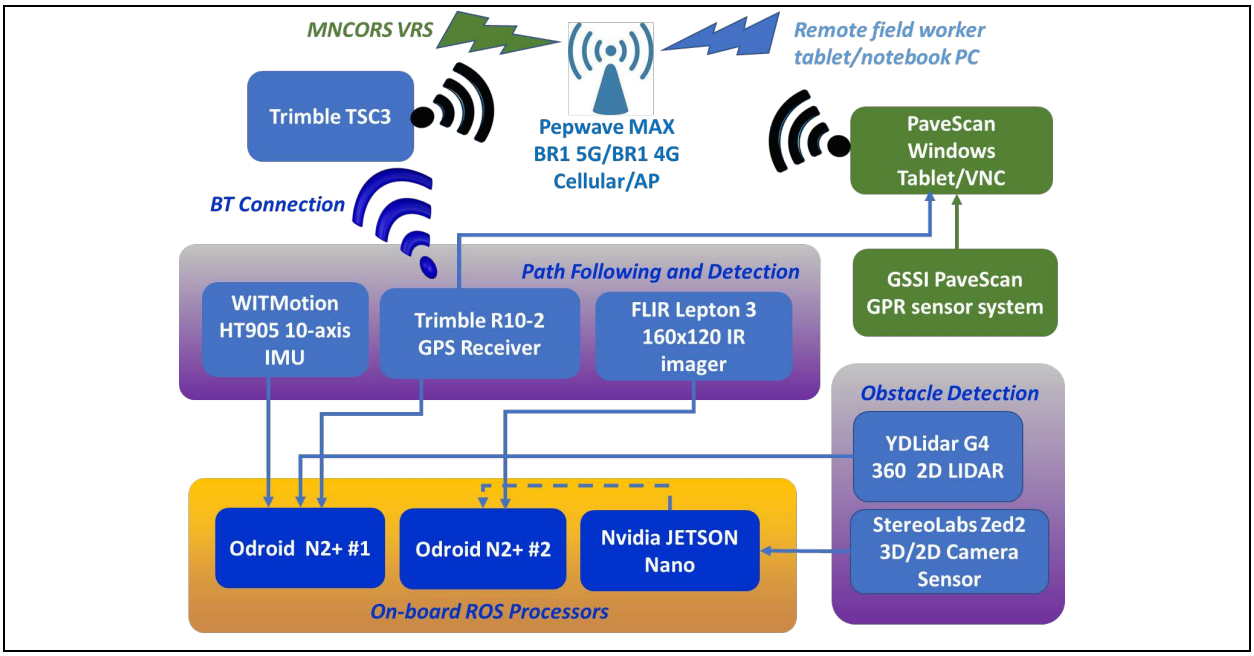


Figure 3.1: Robot hardware architecture

A description of the base robot platforms is provided next, followed by subsequent integration, and description, of the hardware and sensor components.

3.2 ROBOT BASE PLATFORMS

3.2.1 Robot base platforms

Both robots are 2-wheel differential drive custom manufactured robots (SuperDroid Robotics, Inc.). The front wheels are actuated independently using 24VDC brushed servo motors instrumented with dual quadrature encoders. A differential drive type robot was selected because they can efficiently handle high payloads and compared to skid steer robots—which are also a very common cost-effective steering and propulsion mechanism for this class of mobile robots, does not induce lateral frictional skid forces on the wheels over fresh pavement [25]. Note that a front wheel drive Ackerman robot was not considered since they are practically confined by a minimum steering radius. Each servo actuation is independently controlled through a closed loop speed controller operating in the motor controller firmware (RoboteQ MDC2460). The first robot servo motors generate 43Nm output axle torque with a full load output axle speed rating of 140 RPM (180 maximum). With the 12-inch diameter wheels, this gives a theoretical speed maximum speed range from 7.33 feet per second (5 MPH) to 9.39 feet per second (6.4 MPH). The 12x3 inch hard polyurethane wheels can withstand road surface temperatures of 180 degrees. The rear casters are 4x2 inches. Three removable 25.6V 11.1Ahr LiFePO4 batteries supply power to the motors and motor controller.

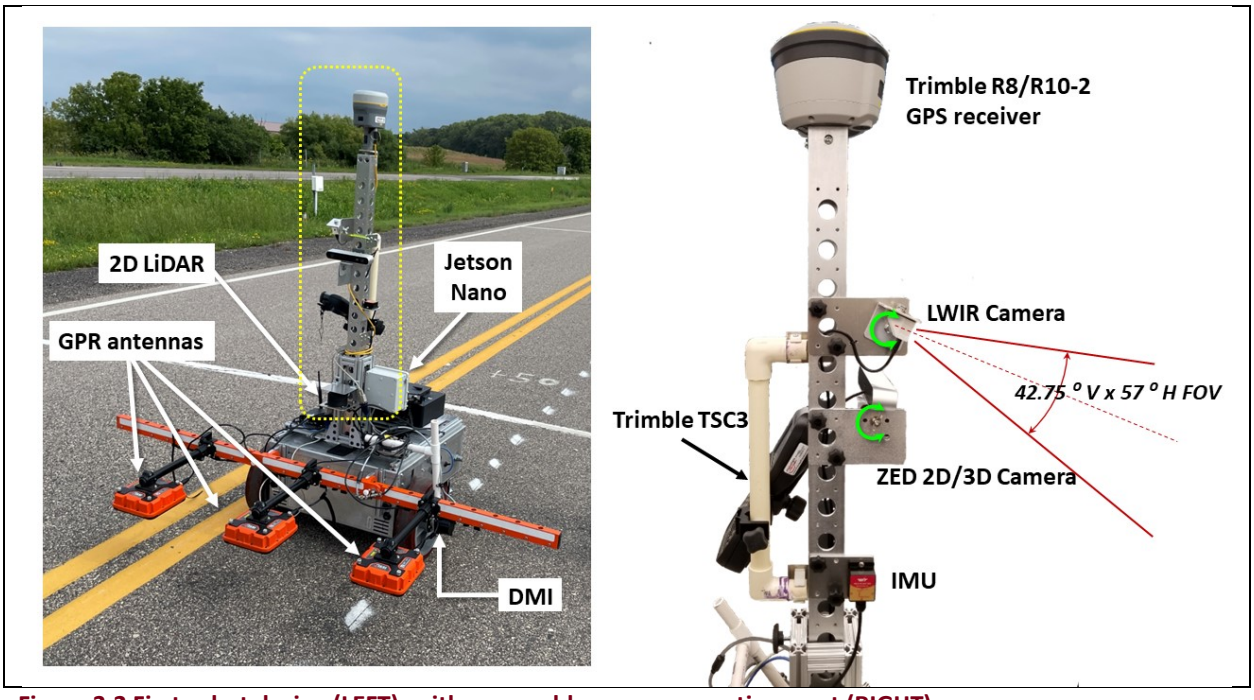


Figure 3.2 First robot design (LEFT), with removable sensor mounting mast (RIGHT).

The second robot platform, which was intentionally designed later in the project, has some improvements over the first robot design. The robot uses lighter (6.8 lbs vs. 10.5 lbs), in-line gearbox power wheelchair servomotors rated at 120 RPM at full torque load of 36 Nm. The maximum theoretical traversal speed at full torque is 6.55 feet per sec. (4.5 MPH), which is still more than sufficient for this task considering the objective is for the robots to mimic typical human walking speeds. The controller

and motors are powered by three 12A 20 VDC widely available, quick disconnect, power tool batteries—a design that is very beneficial for swapping or recharging tasks out in the field. Furthermore, the front 12.5 inch diameter front wheels qualitatively exhibit similar compliance characteristics, and tread design to the original PaveScan GPR pushcart wheels (Geophysical Survey Systems, Inc., Nashua, NH), in order to dampen some of the vibrations observed when the first robot traversed over slightly rough surfaces. A deadman's E-stop switch was also added that disables the output power to both servo motors when manually triggered.

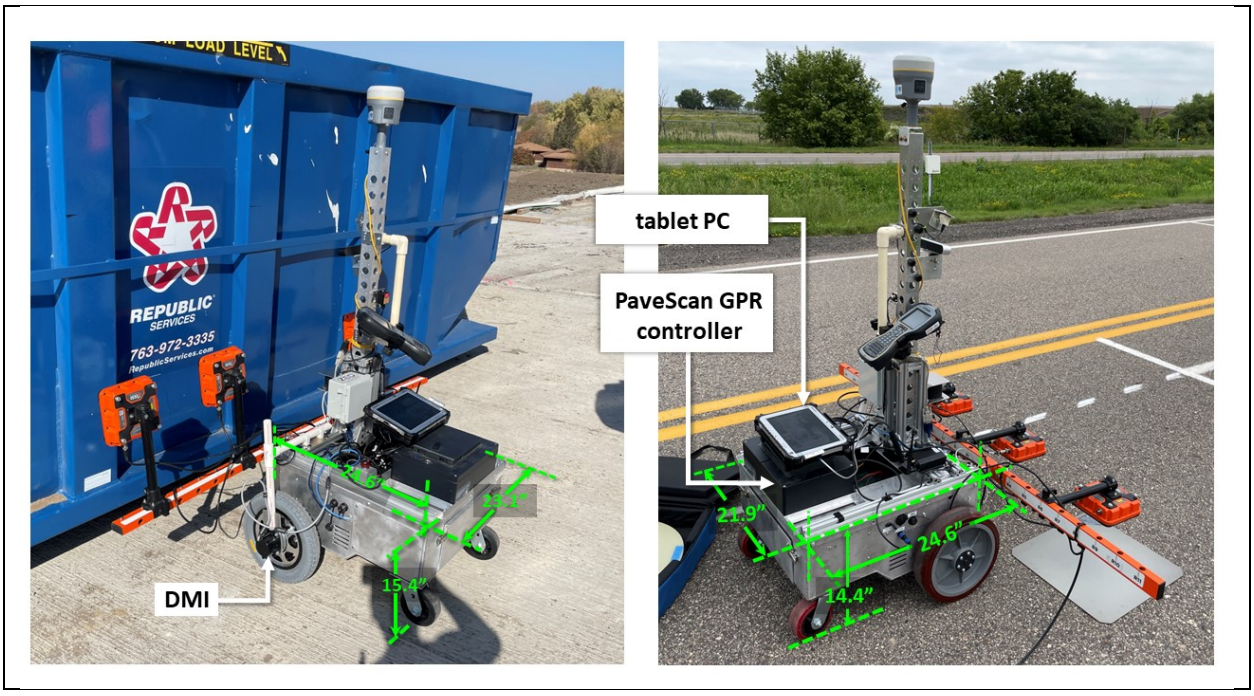


Figure 3.3 The second robot design, (LEFT) and first robot (RIGHT), during a GPR sensor calibration.

The actuation maximum run-time for both robots is rated for 3 to 4 hours under continuous full-load conditions. Due to concerns about damaging the batteries by over discharge, this design rating was not validated. The base dimensions for either robot was intended to provide stability against tipping about the longitudinal x-roll and y-pitch body axes; the chassis height was selected to elevate the GPR array bracket to achieve a GPR bottom surface-to-grade surface air gap distance of 8 inches above grade (the array bracket mounts on the robot are adjustable). Both robot platforms utilize a separate removable 12VDC battery to supply power to the robot sensor and communication hardware (the PaveScan GPR system supplies its own battery source). The first robot platform uses a 28 AH rated deep cycle AGM battery (15.5 lbs). The second robot supplies a 30 AH LiFePO4 battery (6 lbs). The battery power is conditioned and stabilized through voltage regulators. Lastly, an integrated RC 2.4 GHz 6 channel wireless PWM receiver for remote manual steering, direction, and speed control (2 channels used for this purpose).

Referring to figures 3.4 and 3.5, two ODROID N2+ processors where mounted within the enclosed chassis. The Roboteq 2460 dual axis servo motor controller is interfaced through a USB serial interface with the assigned ROS master processor, mounted internally on the side of the robot. A separate USB

RS232 serial port is also used for acquiring the Trimble DGPS receiver NMEA GGA stream. The second ODROID N2+, is used to acquire and control data from the thermal camera sensor and 2D LiDAR (YDLIDAR G4 360) sensor through 3 USB-3 ports. The Jetson Nano processor interfaces to the Zed 2D/3D camera sensor. A switchable 20W A/C power supply is currently used through an AC/DC inverter (the original 5VDC buck regulator power source did not have sufficient response time to compensate for small transient voltage drops that occurred when the GPU became heavily utilized (below 4.75 Volts).

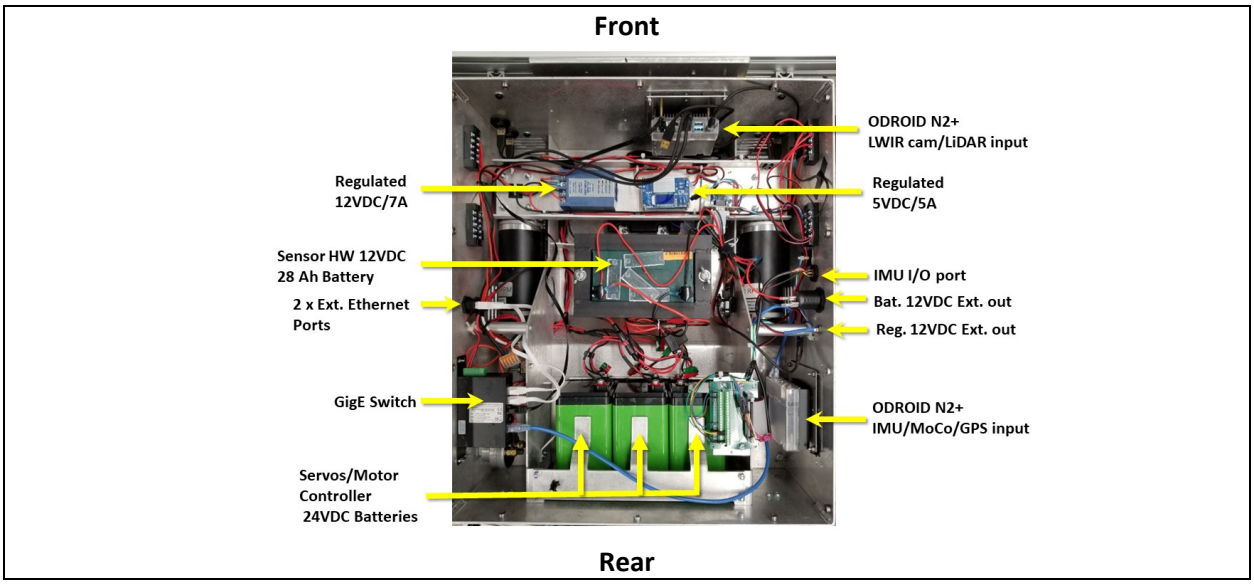


Figure 3.4 First robot in-chassis hardware and power support.

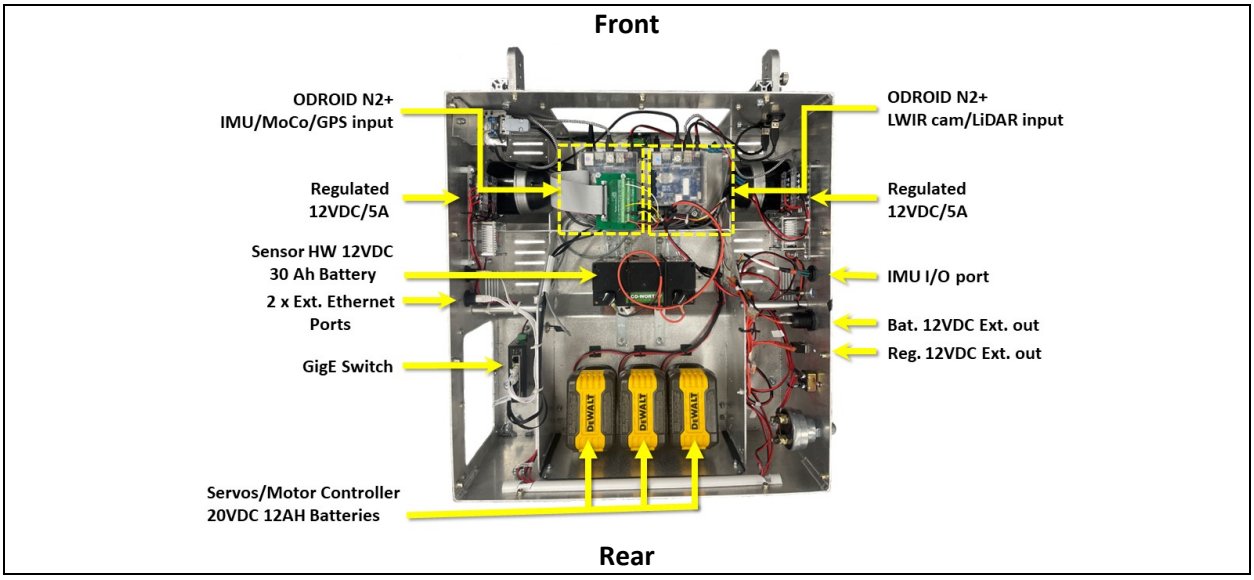


Figure 3.5 Second robot in-chassis hardware and power support.

The Odroid N2+ maximum power draw is specified at 6.5W, while the Jetson Nano maximum power draw is 10W, the switch is 5W, while the router draws approximately 10W (internet traffic dependent), and regulator circuitry generates approximately another 200mA loss. The 2D/3D Zed2 sensor consumes 380 mA x 5 VDC = 1.9 W, the LiDAR working power is 350 mA x 5 VDC =1.75W, and the Lepton LWIR

(Long wave Infrared) Camera consumes 50 mA x 5 VDC = 0.25W. Thus, the sensor and processing hardware together theoretically can draw up to 42.1W/12 VDC = 3.5A at 12 VDC. Under such conditions, the 28 AH battery could supply power for 28 x 0.75/3.5 = 6 hours, and for the 30 AH battery integrated within the second robot, could supply the same level of power for 30x0.75/3.2 ≅ 6 ½ hours. The GPS receivers utilize separate batteries.

3.2.2 Robot Sensor and communication hardware

Two low-cost sensors were integrated for evaluating object detection and collision avoidance. One sensor was a 2D LiDAR (model YDLidar G4 360, 9000 samples/sec.) with a scanning frequency range of 4 to 12Hz, and with a reported outdoor maximum range of 49 feet (15 meters). The second test sensor was a stereoscopic camera (StereoLabs ZED2). The reported empirical range for remaining within ±0.2 meters under any setting of 30 feet [26]. A resolved depth map can then be translated into a RGB 3D point cloud, registered within the coordinate system of one of the cameras. Insofar as the specifications for these sensors, they are more than suitable given the fact that the traversal robot speeds cannot exceed 5 MPH (7.3 ft. per second), and with time headways under 3 seconds, implies a maximum lookahead distance of 21.9 feet. The intent of the collision avoidance capability is to stop the robot to avoid collisions with relatively slow moving, or stationary, encroaching objects.

The single scan LiDAR sensor is located on the sensor mast holder, elevated 24.25 inches above grade (Figure 3.2). The elevation on the mast mount can be decreased to 21.5 inches, or increased up to 28.75 inches. The PaveScan Distance Measurement Instrument encoder (DMI) slides over a keyed shaft pin on the right front wheel of either robot.

Lastly, a LWIR (Longwave Infrared) camera sensor (FLIR Lepton 3.5, with a 57 degrees horizontal field of view lens). was integrated for detecting as-built asphalt joint lines. The camera radiometric wavelength range is from 8 to 14 micrometers, and with a thermal detection range from 14 to 284 degrees Fahrenheit (high gain mode). Thermal 160x120 14 bit resolution radiometric images are captured at a sampling rate of about 9 Hz, requiring 140 mW of onboard nominal power.

A 3.5-foot length removable sensor mast provides the mounting base for several sensors (Figure 3.2), affording a maximum elevation of 14.5 + 42 = 56.5 inches above grade. When installed on the robot, the mast is centered approximately over the center front axle of the robot, to avoid error affects from robot orientation estimates. The mast has a series of taped #10-32 screw holes every 2 inches to allow sensors to mounted at different locations along the robot local reference body frame Z-axis. The mast supports the Trimble GPS R8 or R10 receiver/antenna (5/8 #13 screw head), the FLIR camera with the Thermal MINI board USB interface, the Stereo Labs Zed2 stereo/2D camera 4416x1242 sensor (for front facing, object recognition and detection), and the 9 Degree Of Freedom IMU. The mast also provides a mounting bracket for the Trimble TSC control unit. Except for the Trimble TSC control unit, all the devices interface through USB serial ports which are accessible from the outside chassis of the robot. Cable management is aided by running several sensor cables through the mast center and out the bottom of the mast. The fused LiDAR/camera sensors output is illustrated in figure 3.5.

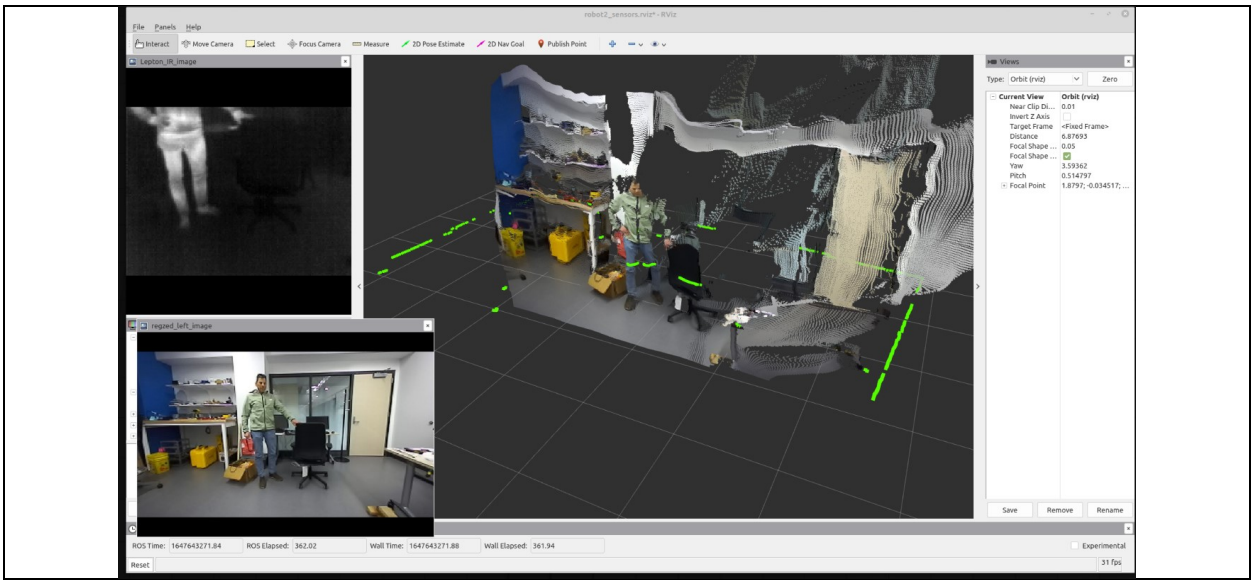


Figure 3.6 Registered 3D point cloud and 2D LiDAR measurements (green), 2D/3D sensor rectified camera image (Lower Left), and thermal camera radiometric image (Upper Right) topics.

CHAPTER 4: ROBOT SOFTWARE ARCHITECTURE

4.1 SOFTWARE ARCHITECTURE OVERVIEW

The objective of the study was to develop a general architecture through ROS, for autonomous robot navigation joint-line path following. A simulation-based platform module provided preliminary insights for validating the architecture. Specifically, this chapter summarizes the general robot software architecture; the semi-autonomous navigation requires asphalt joint line path definition data, and differential corrected GPS and IMU sensor inputs (and their fixed local body position and orientations), the steering and speed controller modules, and a status and operations control module.

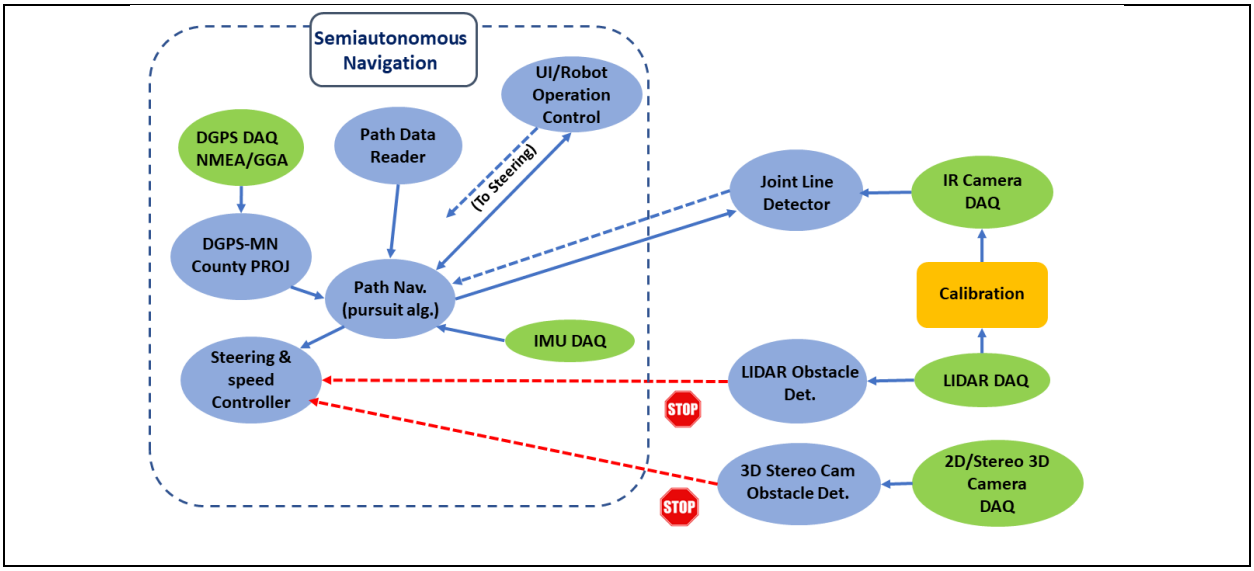


Figure 4.1: Robot software architecture

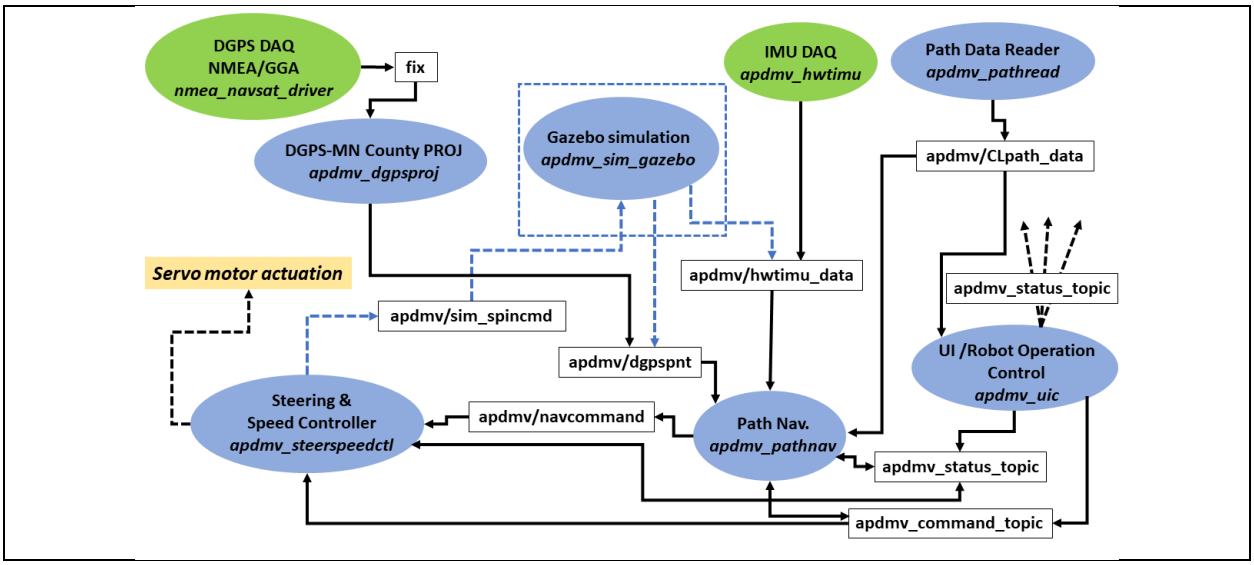


Figure 4.2: ROS semiautnomous navigation and simulation interface stack

The ROS-based architecture consists of a semiautonomous navigation stack and a perceptual stack (Figure 4.1).The semiautonomous navigation stack includes the asphalt joint line path definition data module, differential corrected RTK GPS and IMU sensor input and processing modules (including their fixed local body position and orientation information), the closed loop steering and speed control and servo actuation module, and a User Interface module to handle robot operation and status control (for human intervention and manual operation, monitoring, and pavement traversal task configuration). Figure 4.2 illustrates the navigation stack ROS messaging architecture. A physics-based simulation module integrates with the navigation and controller modules through the messaging architecture by supplying modeled sensor outputs for the DGPS and IMU sensors, as well as a dual axis differential drive torque actuation model. The simulation module was used for validating the semiautonomous control architecture. Salient results are presented in section 4.3 . Note that robot 'semiautonomous' operation refers to robot autonomous traversal without any perceptual feedback of the observed state of the environment from other sensors. The perceptual stack contains the IR camera, 2D/3D camera, and LiDAR data acquisition and control modules, object and collision detection and response action modules, and a as-built asphalt joint line detection and localization module. The remainder of this chapter will first summarize the theory of operation of semiautonomous navigation stack and then the effect of modulating top level steering control parameters on traversal behaviors. The perception stack and associated experimental results will be discussed later in chapter 6.

4.2 ROS NAVIGATION SOFTWARE MODULES

In this study design, 'modules' are generally organized as ROS packages. All software modules (e.g. labeled as 'apdmv_<*module name*>' in figure 4.2), were developed using ROS versions Noetic or Melodic. First, the path following navigation control module is described, followed by the steering/speed servo control module, and then the remaining modules associated with sensor acquisition, and the simulation platform.

4.2.1 Path Navigation Module

4.2.1.1 Path navigation algorithm

The path navigation module algorithm (*apdmv_pathnav*) uses general time-headway pursuit to derive a desired heading, $\gamma_d(k)$ and speed, $S_d(k)$ command at a time step k by deriving a 2D look-ahead goal point, $\boldsymbol{p}^g(\mathrm{k})$ located along a centerline path derived from asphalt joint centerline data (figure 4.3). The input design asphalt joint centerline is approximated by a connected set of finite length line segment vectors, $\vec{\Delta}_j$, $j = 1..N$, $\left\|\vec{\Delta}_j\right\| > 0$.

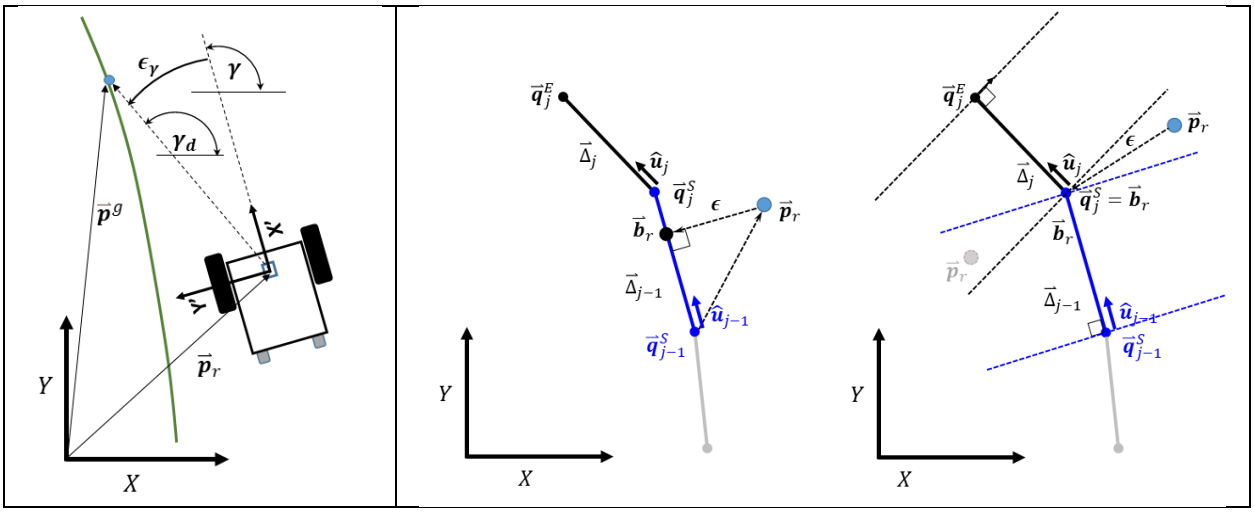


Figure 4.3: Robot pursuit-based control.
LEFT: Robot lateral stearing control; MIDDLE: Lateral error estimation using the robot location projected onto centerline path segment, and (RIGHT). between segments

$$\gamma_d(k) = \tan^{-1}\left(\frac{p_y^r(k) - p_y^g(k)}{p_x^r(k) - p_x^g(k)}\right) \qquad (4.1)$$

$$\begin{aligned} where\ \vec{\boldsymbol{p}}^{\,g}(\mathrm{k}) &= \vec{\boldsymbol{b}}^{\,r}(\mathrm{k}) + d_1 \cdot \hat{\boldsymbol{u}}_c + \sum_{j=c+1}^{j=m} \vec{\Delta}_{\boldsymbol{j}} + d_2 \cdot \hat{\boldsymbol{u}}_{m+1} \\ and\ d_1 &= \left\| \vec{\Delta}^c \right\| - \left\| \vec{\boldsymbol{q}}_{\boldsymbol{c}}^{\,S} - \vec{\boldsymbol{b}}^{\,r}(\mathrm{k}) \right\|, d_1 \geq 0 \\ d_2 &= \left\| \vec{\boldsymbol{b}}^{\,r}(\mathrm{k}) - \vec{\boldsymbol{q}}_{m+1}^{\,S} \right\| \end{aligned} \qquad (4.2)$$

$$\begin{aligned} \left\| \vec{\boldsymbol{p}}^{\,g}(\mathrm{k}) - \vec{\boldsymbol{b}}^{\,r}(\mathrm{k}) \right\| &= \tau \cdot S_d(k)\ \ iff\ \left| \vec{\boldsymbol{n}}_\gamma \cdot \vec{\boldsymbol{n}}_{\gamma_d} \right| < \upsilon \\ S_d(k) &= 0\ otherwise \end{aligned} \qquad (4.3)$$

Referring to equations (4.1) - (**4.3**), $\vec{\boldsymbol{n}}_\gamma, \vec{\boldsymbol{n}}_{\gamma_d}$ are the unit vectors for the robot heading estimate, γ, and desired robot heading, γ_d. For the current implementation, $\left\| \vec{\Delta}_{(\cdot)} \right\| \cong 1.0\ ft.$, and $\vec{\Delta}_{\boldsymbol{c}}$ refers to the segment containing the projection of the robot location, $\vec{\boldsymbol{b}}^{\,r}(\mathrm{k})$. The pursuit point, $\vec{\boldsymbol{p}}^{\,g}(\mathrm{k})$, is estimated by integrating a distance along the path from a beginning point $\vec{\boldsymbol{b}}^{\,r}(k)$ also lying on the path. Referring to figure 4.3 , the first point $\vec{\boldsymbol{q}}_j^{\,S}$, **,** of one segment $\vec{\Delta}_{\boldsymbol{j}}$, $j = s1$, is equal to the endpoint, $\vec{\boldsymbol{q}}_{j-1}^{\,E}$, of the previous segment, $j = s1 - 1$. The complete segment map, $\vec{\Delta}_{\boldsymbol{j}}$ **,** $j = 1,2 \ldots N$, is constructed from horizontal and vertical alignment data contained within LandXML files, or directly from comma-separated (csv) text files (*apdmv_pathreader*). The vector components $(\cdot)_x . (\cdot)_y$ correspond to Easting and Northing planar projection coordinates respectively (with units converted to US feet). The adjoining closest segments $\vec{\Delta}_{s1}, \vec{\Delta}_{s1-1}$**,** to the robot location $\vec{\boldsymbol{p}}^{r}$, are then are subsequently used to estimate the minimum orthogonal (e.g., lateral) distance, $\epsilon(k)$, from $\vec{\boldsymbol{p}}^{r}$ onto the closest segment, $\vec{\Delta}^c$, represented by the orthogonal projected point $\vec{\boldsymbol{b}}^{\,r}(k)$ lying on $\vec{\Delta}_{\boldsymbol{j}}$**:**

$$\epsilon(k) = sign\left(\vec{\boldsymbol{d}}(k) \times, \vec{\Delta}^c\right) \cdot \left\|\vec{\boldsymbol{d}}(k)\right\|$$
$$where$$
$$\vec{\boldsymbol{d}}(k) = \vec{\boldsymbol{p}}^{\,r}(k) - \vec{\boldsymbol{b}}^{\,r}(k)$$
$$and\ where$$
$$\vec{\Delta}^c = \begin{cases} \vec{\Delta}_{s1}, & \left\|\vec{\boldsymbol{b}}^{\,r}(k) - \vec{\boldsymbol{q}}^S_{s1}\right\| \le \left\|\vec{\Delta}_{s1}\right\| \, and \, \left\|\vec{\boldsymbol{b}}^{\,r}(k) - \vec{\boldsymbol{q}}^S_{s1}\right\| \cdot \hat{\boldsymbol{u}}_{s1} \ge 0 \\ \vec{\Delta}_{s1-1}, & \left\|\vec{\boldsymbol{b}}^{\,r}(k) - \vec{\boldsymbol{q}}^S_{j-1}\right\| \le \left\|\vec{\Delta}_{s1-1}\right\| \, and \, \left\|\vec{\boldsymbol{b}}^{\,r}(k) - \vec{\boldsymbol{q}}^S_{s1-1}\right\| \cdot \hat{\boldsymbol{u}}_{s1-1} \ge 0 \end{cases} \quad (4.4)$$

Within a curvature alignment, as shown in figure 4.3 ,a plausible condition at time step k will violate the conditions described in equation (4.4); the resolution is to approximate $\epsilon(k)$ with $\vec{\boldsymbol{b}}^{\,r}(k) = \vec{\boldsymbol{q}}^S_j$, and setting $\vec{\Delta}^c$ to either $\vec{\Delta}_j$or $\vec{\Delta}_{j-1}$. Similarly, if $\vec{\boldsymbol{p}}^r(k)$ is 'inside' the curvature alignment, the solution is to choose $\vec{\boldsymbol{b}}^{\,r}(k)$ projected onto either $\vec{\Delta}_j$or $\vec{\Delta}_{j-1}$ which results in the minimum $\epsilon(k)$ estimate. Such conditions will have very little impact on the robot performance since they will not frequently occur; the curve alignments for highway designs typically prescribe large radii to handle higher speed traffic (for example a curvature of 275 feet at 30 miles per hour up to 1065 feet at 55 miles per hour [27]) and therefore the 'shadow' regions in (bounded by the dotted lines cross in figure 4.3) will be small. In addition, given such 'small' regions, the approximations will converge close to an orthogonal projection at a segment endpoint, assuming the robot traversal positions remain within a lane width of the prescribed joint centerline. The time constant, τ = 2.5 seconds, and during joint center line traversal $S_d(k)$ is currently set to a default value of $C = 3$ mph (4.4 ft./sec). To accommodate beginning and end point ($\vec{\boldsymbol{p}}^{BE}$) path conditions, $S_d(k)$ is linearly reduced to ≈0 mph when the 2D robot location $\|\vec{\boldsymbol{p}}^r - \vec{\boldsymbol{p}}^{BE}\| \le a \cdot \tau \cdot C$. A more gradual deceleration can be achieved with $a > 1$. The value, v in (4.3) is adjusted empirically as a preventative measure against large deviations between the desired, and estimated robot heading, ϵ_γ. The angle deviation of $v = \left|\epsilon_\gamma\right| <$ 25 degrees.

4.2.1.2 Multiple pass construction and navigation

The multiple pass robot traversal maneuvers are characterized as being in one of four states. Each state transition is determined by the estimated distance of the robot from the start, or end, point location of the current traversal as described in section 4.2.1.1 . Each state defines a specific speed and heading control behavior to minimize overshooting the end points, pavement marring, and tipping at the transitions between traversal passes, at the start of the traversals at $k = 0$, and at the conclusion of the traversals. For example, the Turnaround state will not transition to the Starting Location Pursuit state until estimated moving average speed over 5 second window (experimentally determined) is under 1 MPH (1.46 feet per second). The Starting Location Pursuit state enforces a reduced speed of the robot until it is within $\|\vec{\boldsymbol{p}}^r - \vec{\boldsymbol{p}}^{BE}\| < 5\ feet$ from the starting location $\vec{\boldsymbol{p}}^{BE}$, while relaxing the aforementioned steering set point constraint, $\left|\epsilon_\gamma\right|$. The pursuit target point, $\vec{\boldsymbol{p}}^g(\mathrm{k}) = \vec{\boldsymbol{p}}^{BE}$.

As a safety mechanism, while the robot is within the Start Location Pursuit state, if the robot is positioned too far from the starting location (beyond 25 feet), autonomous traversal will be disabled (more succinctly, the speed and steering control module closed loop controller will be disabled).

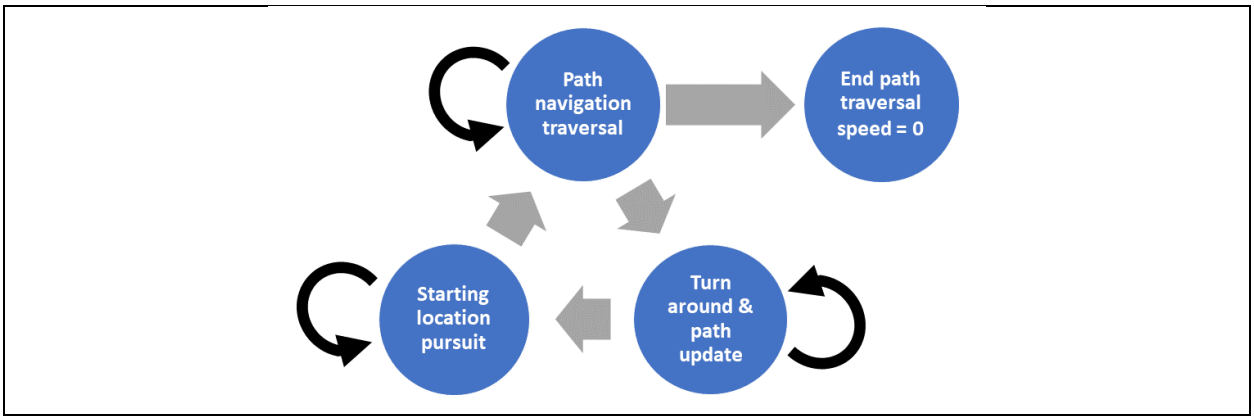


Figure 4.4: Robot state transitions used during multiple path traversals.

In order to generate trajectories for the multiple pass traversals to follow a set of specified joint and offset centerlines, a buffered copy of the discretized centerline approximation $\vec{\Delta}_j, j = 1..N$ is shifted by a specified perpendicular offset distance, before each traversal. The specified offset convention conforms to a well-accepted road design and construction convention which specifies a positive offset to the right side of a joint centerline, and a negative signed offset to the left side, when the joint centerline is proceeding from a smaller distance (station value) to a larger distance (station value). The offset positioning of the prescribed path is estimated by projecting the distance from the centerline gradient vector, that is approximated using the average gradient from sequentially adjacent segments (Figure 4.5).

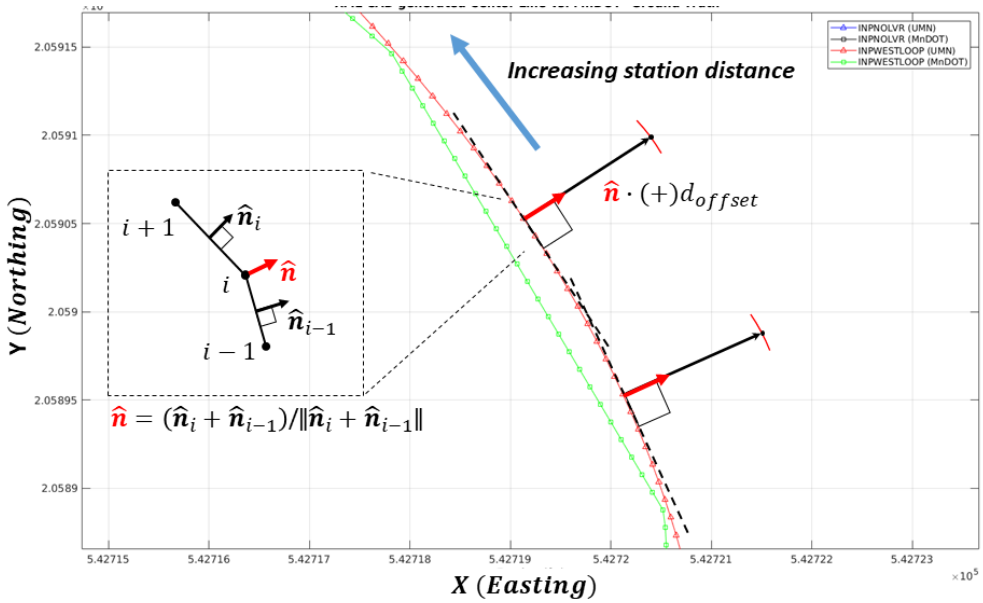


Figure 4.5: Joint centerline offset estimation. Green dotted line is an approximation of the XML curve (red) used for simulation surface road representation.

4.2.1.3 Robot heading correction

The navigation module approach initially did not consider unpredictable, and variable IMU sensor bias and variable sensor noises from the GPS and IMU. However, field-testing elucidated variable, unpredictable sensor bias with the IMU and noise in the IMU and DGPS data. Without correcting the bias errors, they contributed significantly to the lateral path errors well beyond desired specifications of under 8 inches from the prescribed joint line path (experimental traversal results will be discussed in detail in the next chapter). The quantitative effects of such a bias were ascertained from simulation experiments. Small, controlled estimation errors (e.g., $\|\Delta\gamma(k)\| \geq 3^{\circ}$) superimposed onto the robot heading estimate generated under, or over, steering behaviors which translated into the observed significant lateral deviations from the prescribed joint centerline path.

To address the problem, a Bayesian filter was designed and then implemented within the robot navigation module to continuously correct specifically the yaw sensor bias errors. The heading control set point, $\epsilon_{\gamma}(k)$ is computed using the IMU sensor yaw angle, $\vec{\boldsymbol{n}}_{\gamma(k)}$, and desired heading from the navigation controller, $\gamma_d(k)$, estimated at time instance *k* by:

$$\epsilon_{\gamma}(k) = \text{sign}\{\vec{\boldsymbol{n}}_{\gamma(k)} \times \vec{\boldsymbol{n}}_{\gamma_d(k)}\} \cdot \cos^{-1}\left(\vec{\boldsymbol{n}}_{\gamma(k)} \circ \vec{\boldsymbol{n}}_{\gamma_d(k)}\right) \tag{4.5}$$

where $\vec{\boldsymbol{n}}_{(\cdot)}$ is the representative unit normal. We then consider the heading sensor bias error correction, $\Delta\gamma(k)$, as the unknown state, s_k. If the observation estimate of $\Delta\gamma(k)$, $\theta(k)$, of the robot heading, is calculated from the DGPS Easting, Northing displacement, $(\Delta x, \Delta y)$ between time instances, k, and $k-1$.

$$\theta(k) = \gamma(k) - \text{atan2}(\Delta y, \Delta x) \tag{4.6}$$

the Bayes filter design can be defined:

$$\text{P}(s(n)|\theta(k))^{*} = \frac{\mathcal{L}(\theta(k)|s(n)) \cdot \text{P}(s(n))}{\text{P}(\theta(k))_{\forall s}} \tag{4.7}$$

where, $\text{P}(s(n)|\theta(k))^{*}$ represents the *posterior* probability of $s(k)$,with likelihood function, $\mathcal{L}(\theta(k)|s(k))$ is a Gaussian, with a pre-defined standard deviation, $\sigma = 0.5 \cdot [s_{min}, s_{max}]$, e.g.:

$$\mathcal{L}(\theta(k)|s(n)) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{\theta(k)-s(n)}{\sigma}\right)^2} \tag{4.8}$$

For this discussion the notation is abused by implying $s(n) \equiv s(n)_k$. The *a-priori* state probability $\text{P}(s)$ is initially assigned a uniform distribution across the specified state space (thus, *uninformed* the 'belief' that any state is equally probable). In our implementation our explorative state space consists of $n = 1 \ldots 200$ states, equally spaced within a range of ±20° (this is a very conservative uncertainty range for the IMU sensors used for the robots). The denominator marginal probability $\text{P}(\theta(k))_{\forall s}$ cannot be computed precisely, but is instead approximated by the sum of the posterior likelihood products over the explorative state space:

$$\mathrm{P}(\theta(k))_{\forall s} \approx \sum_{n}^{N} \mathcal{L}(\theta(k)|s(n)) \cdot \mathrm{P}(s(n)) \qquad (4.9)$$

In the actual implementation, the filter update step uses a set of observations, $\theta(j), j = (1,2, \dots M)$, from current and previous time steps, rather than a single observation at timestep k. The set selection criteria were empirically derived from analyzing previously collected data and performing simulations on straight and curved road sections (using the MnROAD Joint Line XML geometry data provided by MnDOT). A temporal window of 2 seconds is used to select the set of observations. For a given time window there will be K possible observations. An observation within the time window is added to the set if the GPS displacement from time steps $k-1$ and, is above a defined threshold, and the robot yaw rate IMU sensor estimate, $|\dot{\gamma}^r|$, is below a defined threshold. The distance threshold is currently 0.19 ft.—which for example with a 10 Hz sampling rate, corresponds to a robot traveling under 2 ft./sec., or with a sample rate of 5 Hz corresponds to a robot traveling under 1 ft./sec. The angular yaw rate threshold is 20 deg./sec. to accommodate for added 'noise' in the measurement from the pavement surfaces at the test locations. Finally, to ensure observation measure consistency within the time window, the set of $\theta(j), j = \{1,2, \dots M\}$ are used only if $\frac{M}{\mathrm{K}} \geq 0.8$. Note that all the parameters can be modified accordingly through the ROS architecture, and the detailed procedures for doing so are described in the Appendix A.

The a-posteriori update to $\mathrm{P}(s)$ is then carried out using the $\theta(j), j = \{1,2, \dots M\}$ observations, to update an estimate of the bias error correction, $\Delta\hat{\gamma}(k)$. The algorithm is summarized in table 4.1.

Table 4.1: Bayesian yaw filter module algorithm

for each $\theta(j)\ in\ \{\theta(1), \theta(1), \dots \theta(M)\}$ ***do {***

for each $s(n)\ in\ \{s(1), s(2), \dots s(N)\}$ ***do {***

$\mathrm{P}(s(n)|\theta(j)) = \mathcal{L}(\theta(j)|s(n)) \cdot \mathrm{P}(s(n))$

$\mathrm{P}_\theta(n) = \mathrm{P}(s(n)|\theta(j))$

}

$\mathrm{P}(\theta(j))_{\forall s} = \sum_{n=1}^{N} \mathrm{P}_\theta(n)$

for $n = (1,2, \dots N)$

$\mathbf{P}(\boldsymbol{s(n)}|\boldsymbol{\theta(j)})^* = \dfrac{\boldsymbol{\mathcal{L}(\theta(j)|s(n)) \cdot \mathbf{P}(s(n))}}{\mathbf{P}(\boldsymbol{\theta(j)})_{\forall \boldsymbol{s}}}$

//* posterior update of state space probabilities

$\mathrm{P}(s) = \mathrm{P}(s)^*$

}

//* Maximum a-posteriori (MAP) probability state estimate for yaw correction *//

$\Delta\hat{\gamma}(k) = s^*, \max_{s \to s^*} \mathrm{P}(s)$

Finally, a particle spreading function was used to mitigate degenerative conditions from over-convergence of $\mathrm{P}(s)^*$ [28]. In this case, a uniform weight, $w = \frac{1}{N}$ is iteratively added to all probabilities in $\mathrm{P}(s)$, and then $\mathrm{P}(s)$ is re-normalized to comply with the probability law, until the following condition holds:

$$\frac{1}{\sum_{n=1}^{N} \mathrm{P}(s(n)) \cdot \mathrm{P}(s(n))} \geq N \cdot 0.25 \qquad (4.10)$$

Figure 4.6 illustrates the Bayesian Filter off-line yaw correction estimates from a 300 ft. path following traversal task, with four passes. Large yaw error estimates from the GPS estimates occur at the turn-around locations (outside plot ranges) and when it traversed on top of pavement surface aberrations near the shoulder within the third, forward pass. Further experimental data and relevant results will be summarized in detail in chapter 5.

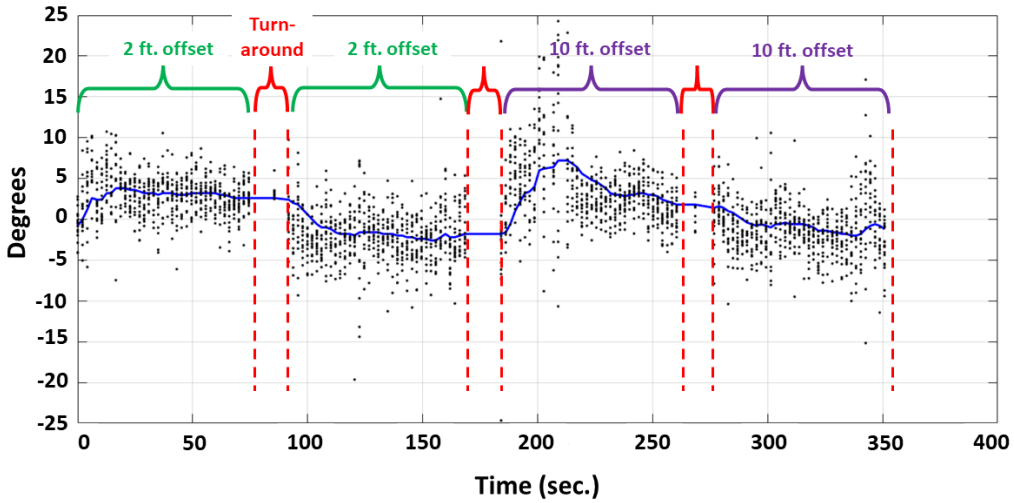


Figure 4.6: Off-line yaw error bias correction (blue) using GPS-based yaw estimates sampling (black points).

4.2.2 Steering and Speed Control Module

The steering and speed control module implements a simple unicycle control law using PID (Proportional Integral, Derivative) controller to adjust left and right rotational wheel servo speeds based on the estimated difference in desired and robot heading, $\epsilon_\gamma(k)$, (Figure 4.3 and desired speed and the measured robot speed, $\epsilon_s(k) = S_d(k) - S_r(k)$. Note that the robot longitudinal speed estimate (e.g., robot velocity estimate projected onto the local body X' axis). The steering and speed adjustments are then updated to generate equivalent servo speed control signals:

$$\kappa_\gamma = P_\gamma \cdot \epsilon_\gamma(k) + D_\gamma \cdot \left(\frac{\epsilon_\gamma(k) - \epsilon_\gamma(k-1)}{\Delta t}\right) + I_\gamma \cdot \sum_{i=k_0}^{k} \epsilon_\gamma(i) \cdot \Delta t \qquad (4.11)$$

where

$$\epsilon_\gamma = \mathrm{sign}\{\vec{\boldsymbol{n}}_\gamma \times \vec{\boldsymbol{n}}_{\gamma_d}\} \cdot \cos^{-1}\left(\vec{\boldsymbol{n}}_\gamma \circ \vec{\boldsymbol{n}}_{\gamma_d}\right) \qquad (4.12)$$

$$\kappa_s = P_s \cdot \epsilon_s(k) + \ D_s \cdot \left(\frac{\epsilon_s(k) - \epsilon_s(k-1)}{\Delta t}\right) + I_s \cdot \sum_{i=k_0}^{k} \epsilon_s(i) \cdot \Delta t \qquad (4.13)$$

$$\begin{aligned} \kappa_L &= \kappa_s + \kappa_\gamma \\ \kappa_R &= \kappa_s - \kappa_\gamma \end{aligned} \qquad (4.1)$$

and where $\pm\kappa_L$, $\pm\kappa_R$ are *scaled* values proportional to the L/R wheel rotation speeds which are tracked by the Roboteq servo controller. Note that the individual steering and speed control gains $P_{(\cdot)}$, $I_{(\cdot)}$, $D_{(\cdot)}$ are defined and set with ROS parameters. As mentioned in the previous chapter, the motor controller firmware for both robots were configured to operate in mixed mode, thus all allowing κ_γ and κ_s signals to be used directly. Again, a closed loop per axis speed control strategy was used rather than direct moment torque control since this is theoretically more stable for the specific controller hardware.

Lastly, the module also provides a mechanism to directly operate the servos to override autonomous operation. Technically this is accomplished through a ROS service mechanism that receives and executes (κ_γ, κ_s) requests. As a safety precaution, closed loop PID steering, and speed control must be enabled through a latched reception of a 'START" command published ROS topic.

4.2.3 GPS data Acquisition Module

The GPS location module processes the GPS WGS84 longitude, latitude data published by the ROS NMEA serial reader into county projection coordinates [29] by referencing the Proj4 transformation library interface [30]. The projection coordinates are expressed in US feet. The GPS receiver (Trimble TSC3) is configured to obtain CMR+ formatted pseudo-range corrections through the MNCORS VRS network.

4.2.4 IMU data Acquisition Module

The IMU data acquisition module publishes through a standard ROS topic protocol of the 3D rigid body RPY orientation α, β, γ through quaternion representation, as well as the tri-axial angular velocity, $\dot{\alpha}^r, \dot{\beta}^r, \dot{\gamma}^r$ and acceleration data, $\ddot{x}^r, \ddot{y}^r, \ddot{z}^r$.. The sensor implements an Extended Kalman Filter to update predictions at a configured rate of 20 Hz (EKF implementation is not published by the manufacturer)

4.2.5 User Interface Module

A user input module provides an interface for ROS, and non-ROS developed applications to access robot control status and the robot position/orientation, as well as supply parameters that define the traversal of the robot, through a TCP/IP socket connection to JSON data streams. The traversal Input information consists of a starting and an ending distance location, a set of offset values and traversal repetitions, the county name where the robot is located, and the desired traversal speed of the robot.

4.3 ROS ROBOT SIMULATION MODULE

An additional module, *apdmv_sim_gazebo*, based on the ROS Gazebo environment was developed to test the software architecture and to infer the robot traversal behavior under 'operating' it under

different scenarios. The module executes two nodes to interface with the L/R wheel simulated servos, and to 'acquire' robot position and orientation data. The simulated torque servo motors are 'energized' by providing angular and linear velocity data, through a built-in physics based differential torque control robot actuation driver. An estimate of this motion in the robot local frame at time *k*, given the servo speed commands, $\pm\kappa_L$, $\pm\kappa_R$ is computed by:

$$v_{\{L,R\}} = \begin{cases} \kappa_{\{L,R\}}(k) \cdot D \cdot \pi \cdot (M_{rpm}/60)/K_{max} & \kappa_{\{L,R\}} < K_{max} \\ D \cdot \pi \cdot (M_{rpm}/60) & \kappa_{\{L,R\}} \geq K_{max} \end{cases} \quad (4.15)$$

$$\acute{\gamma}_r = (v_R - v_L)/b \quad (4.16)$$

$$\dot{x}_r = R(v_R + v_L)/2, \dot{y}_r = 0, \dot{z}_r = 0, \quad (4.17)$$

where D is wheel diameter, b is the base length of the wheels, K_{max} is the maximum value of and M_{rpm} is the maximum RPM output specification of each wheel, which is determined from the product specifications of the robot servo motor/gear reduction assembly.

An SDF robot model was constructed with inertial properties of each component approximated through primitive geometric models using published, or imperially weighed, masses. The position of the sensor mast was used to approximate the GPS sensor location. A Gazebo IMU driver, associated with an attached IMU object, generated the IMU sensor data. A road surface was constructed from a series of connected polygons, with their geometries formulated from the discrete approximations of the centerline data. . All simulated roadway sections assumed zero-grade longitudinal and crown variation.

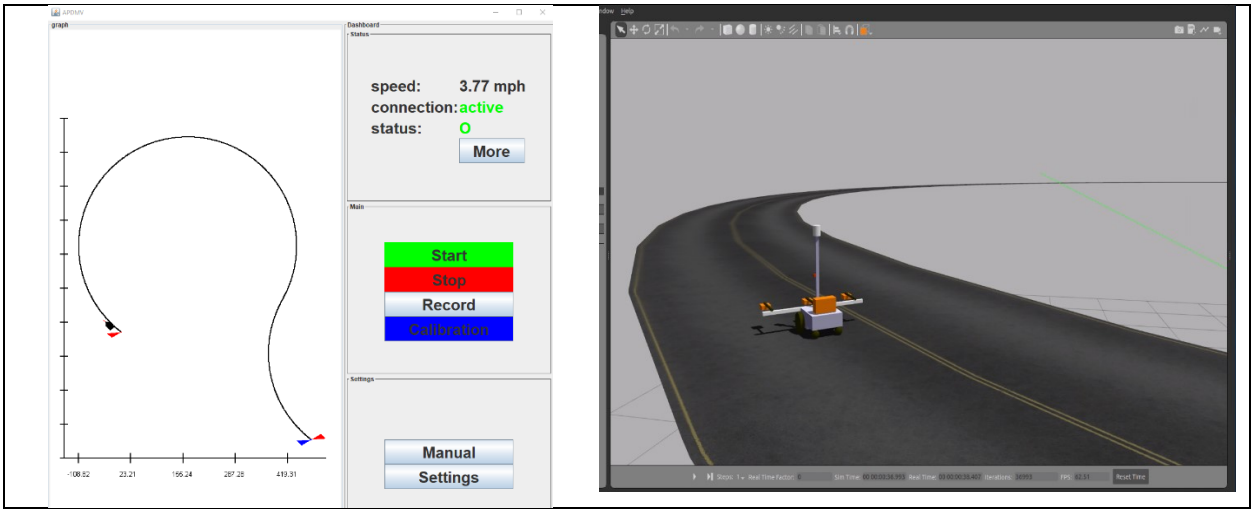


Figure 4.7: Robot simulation environment; (LEFT) User Interface application prototype plotting the MnROAD curve center joint-line alignment the robot is traversing (RIGHT).

4.3.1 Simulation experiment

Simulation experiments were conducted prior to field experiments to infer the effect of adjusting the steering and navigation parameters on robot traversal behavior. More specifically, the experiments

quantified the effects of time headway and general steering response characteristics on the overall lateral error of the robot. The experiments simulated out-and-back traversal scenarios using a +2 foot offset from the prescribed road centerline, which referenced either a MnROAD straight alignment or a MnROAD section containing circular curve alignment. Unbiased Gaussian noise was added to the simulation robot position and IMU data. The DGPS data used a standard deviation of ±0.5 inches (1.3 cm) (obtained from a static position measure estimate), and the IMU noise standard deviation was ±1 degrees (from a laboratory stationary robot measurements averaged over all three IMU axes, with the servo motors operating). The curve alignment contains a radius of 275 feet. For each experiment the robot started 10 feet behind the programmed starting point path location, rotated about 45 degrees away from the roadway centerline path. A traversal target speed of 3.1 MPH was used to reflect a relaxed walking speed. The update rate of the position data was set to 5 HZ, matching the required operating requirements for the PaveScan GPR payload. The IMU update rate, 20 Hz, also matches the configured output rate of the IMU sensor.

4.3.2 Simulation experimental results

Table 4.2 summarizes a subset of results from several repeated experiments that elucidates the general trends. Note that the lateral error data consisted of robot locations when the robot was operating in the Path Navigation Traversal state, which occur when the robot speed exceeded 1.5 feet per second. First, the speed and steering PID gain, and the pursuit point distance were tuned to produce smooth robot maneuvering and traversal behaviors with small lateral error deviations from the prescribed path. There is no claim that this result represents the 'optimal' performance from a lateral error standpoint, but nevertheless under real-world operating conditions. The smooth traversal behavior would theoretically reduce high torque transitions imposed on the servo motors that would otherwise generate high current load spikes. The same settings were then used to run the curve section scenario. Then, the steering control response was increased by increasing the proportional steering control gain with, and without, increasing the time headway to the pursuit target.

Table 4.2: Simulated robot traversal behavior performance

Steering response parameters	Traversal direction	Straight alignment lateral traversal error (μ±σ inches)	Curve alignment lateral traversal error (μ±σ inches)
2.5 sec headway, $P_\gamma = 110, D_\gamma = 8$	Forward	-0.23 ±2.21	-2.42 ±2.78
	Reverse	0.59 ±3.57	2.37 ±5.28
2.5 sec headway, $P_\gamma = 210, D_\gamma = 10$	Forward	**-0.36 ±0.92**	-0.32 ±1.02
	Reverse	**0.37 ±2.33**	0.98 ±1.95
1.5 sec headway, $P_\gamma = 110, D_\gamma = 8$	Forward	-0.13 ±2.19	-2.41 ±2.72
	Reverse	0.36 ±5.74	0.52 ±5.41
1.5 sec. headway, $P_\gamma = 210, D_\gamma = 10$	Forward	-0.86 ±1.42	**-0.93 ±0.92**
	Reverse	-0.70 ±2.15	**0.09 ±2.02**

The results indicate that increasing the controlled steering stiffness reduced under/over-steering affects, as can be observed by the reduction of the lateral error bias. Note that the predominant effect on

lateral error was influenced near the transient conditions situated at the end-point of the traversal, when the robot transitions from the Starting Location state to the Navigation state (figure 4.8). Reducing the headway was, at best, inconclusive at improving the traversal performance. The yaw error corrections are consistently between 0 and under 0.5 degrees. This is expected since the simulated sensors measurements were theoretically unbiased.

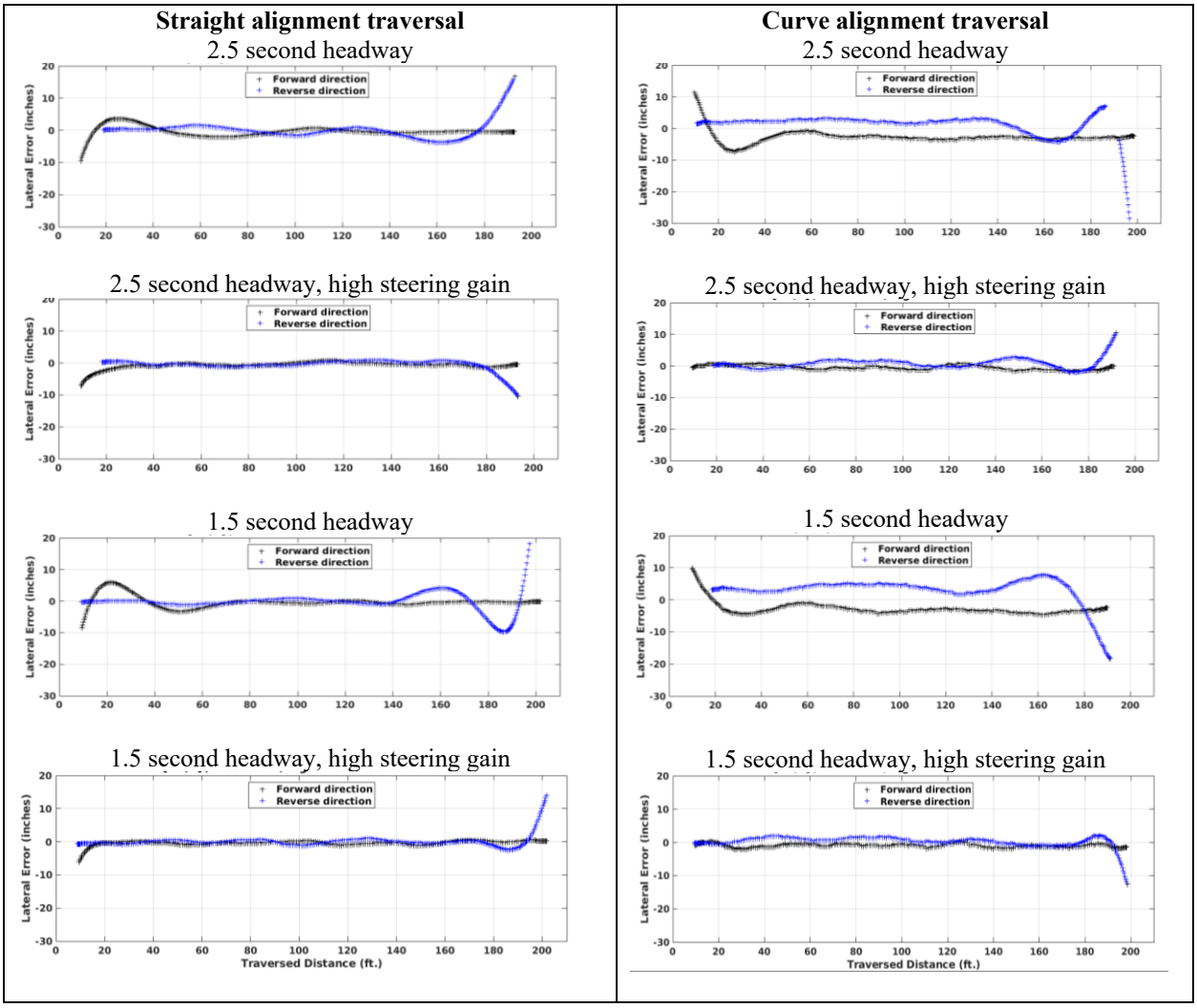


Figure 4.8: Simulation lateral error .for straight path traversal (LEFT column) and the curved path traversal (RIGHT column) experiments.

Increasing the traversal target speed was not considered, which will increase lateral errors, without increasing the closed loop steering control response. For this work, the target traversal speed equated to a casual walking speed of 3 MPH. The simulation differential actuation model did not characterize the actual current-induced torque response characteristics of the servo motors (recall that closed loop PID servo speed control was configured within the controller hardware). Multiple attachment points of the robot parts could not be modeled, and the inertial model approximations of the geometrical robot model itself do not necessarily reproduce the full dynamic response of the robot in actual field conditions. In addition, the actuation response for each wheel of the simulation robot model is

equivalent. The robot servo motors were observed to exhibit slight variations in their dynamic response to identical actuation control commands. For these reasons, the simulation tuned control parameters were considered as starting point for field experiments.

CHAPTER 5: AUTONOMOUS PATH TRAVERSAL

5.1 FIELD EXPERIMENTS

Several field experiments were conducted at the controlled MnROAD low volume loop road facility. The procedure was to specify a starting station number and the traversal distance for the robot to follow, and then the target centerline offset values to position the robot near the designed (expected) joint centerline, mid-lane, and near shoulder locations. For these experiments straight line section, with well-known asphalt pavement density characteristics was used with, and without, the PaveScan GPR system payload. The curve section roadway, which was used in the simulation experiments, constructed with concrete asphalt, was not used in the field experiments. All the MnROAD sections are representative of a 2-lane, 12 foot width roadway, with the designed asphalt joint line at the center of the roadway.

Initial tests using the first robot with a forward traversal with a +3 feet offset and a return pass (reverse direction) traversal of +9 feet offset, and with a DGPS collection frequency of 10 Hz, is illustrated for one such experiment in figure 5.1, using the first robot. The traversal distance was set to cover 450 feet. The lateral robot position errors for these experiments were -1.65±3.13 inches in the forward direction, 1.86±4.39 inches in the reverse direction. Note that the lateral errors, are calculated when the robot is within the Traversal Navigation state (which occurs when the robot speed was above 1.5 feet per second). Like the simulation experiments described in the previous chapter, most of the lateral error accrued proceeding the transient states of the robot operating between traversal passes traveling in opposite directions; the lateral robot positioning errors were less than 6 inches 94.4% of the time in the forward direction, and 85.9% in the reverse direction. During these experiments, it was determined that the PaveScan GPR was not capable of acquiring the DGPS NMEA strings at this data collection frequency. The DGPS output frequency was therefore reduced to 5 Hz. A second set of experiments were conducted but without the Bayesian bias correction filter active. Although this was an oversight, it illustrated a significant lateral error bias of 15 to 18 inches corresponding to an estimated heading bias of 5 degrees from the DGPS estimate. This corroborated with simulation experiments that artificially added biases to the IMU sensor model readings.

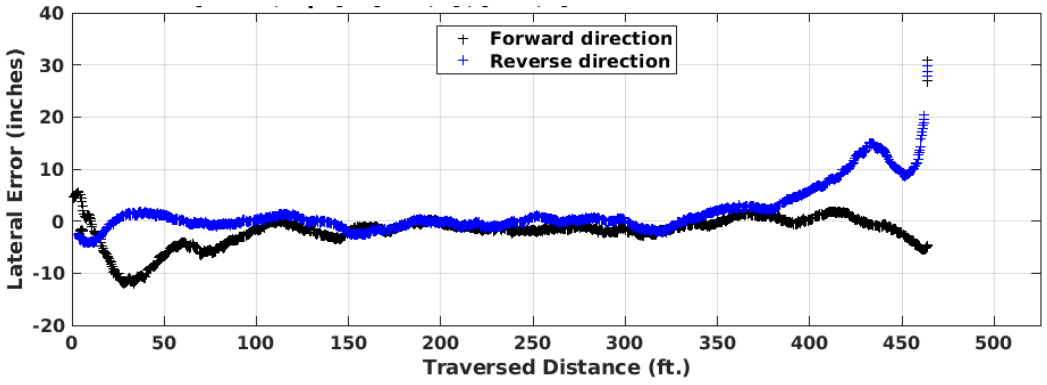


Figure 5.1: Robot semiautonomous traversal lateral error from the prescribed straight-line 3 & 9 ft. offset paths.

Additional experiments were conducted, with the navigation module IMU correction with a multiple path traversal consisting of a +2 feet, +6 feet, and +10 feet offsets. The +2 feet offset, which traversed forward in the positive station direction, positioned the most left GPR sensor approximately 6 inches from the prescribed road design center joint line. The +6 feet offset, which then returned the robot in the opposite, reverse direction to the starting distance location, positioned the center GPR sensor near the lane center. Then, the +10 feet offset pass, which proceeded again in the forward (increasing) station distance direction, positioned the right GPR sensor at the shoulder line, near unconstrainted joint line edge. A final return traversal with the +2ft. offset, which returned the robot to the initial starting distance location for the traversal experiment, positioned the right GPR sensor next to the prescribed road design center joint line. Figure 5.2 illustrates a typical result.

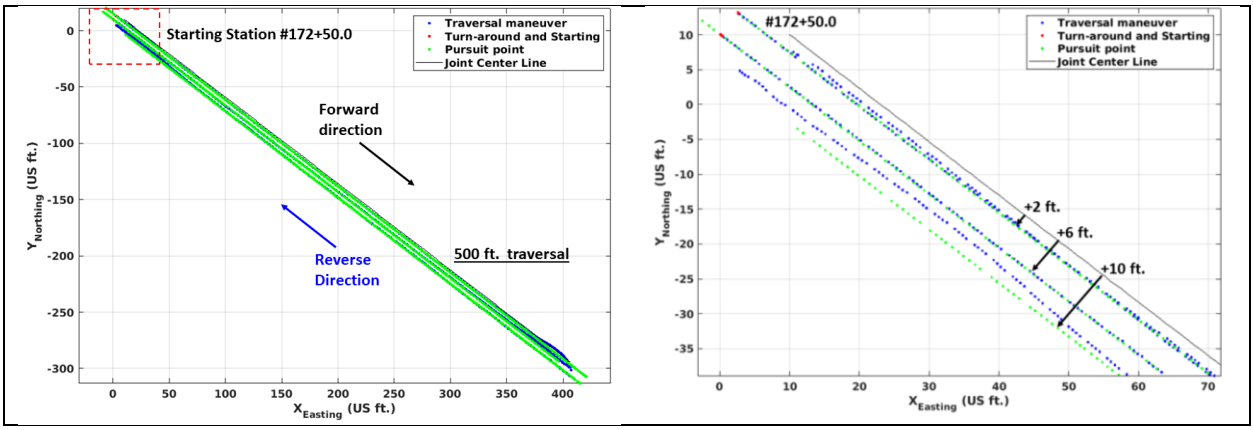


Figure 5.2: [LEFT] MnROAD Robot semiautonomous multiple path traversal data; [RIGHT] Starting end robot location tracks .

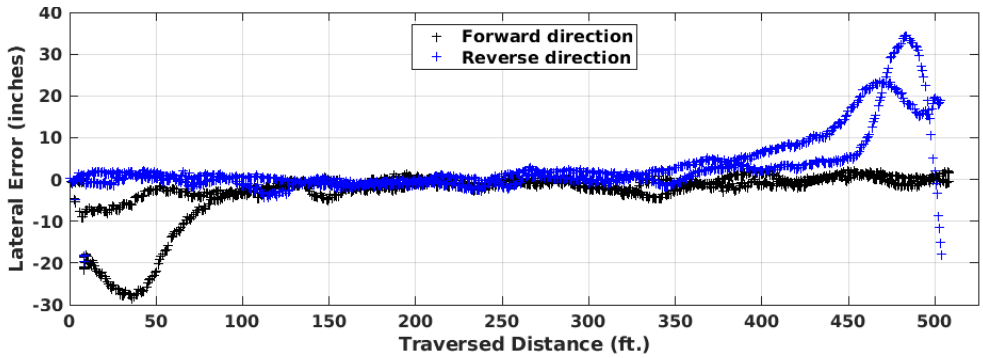


Figure 5.3: Robot semiautonomous traversal lateral error from the prescribed 2/6/10 ft. offset paths.

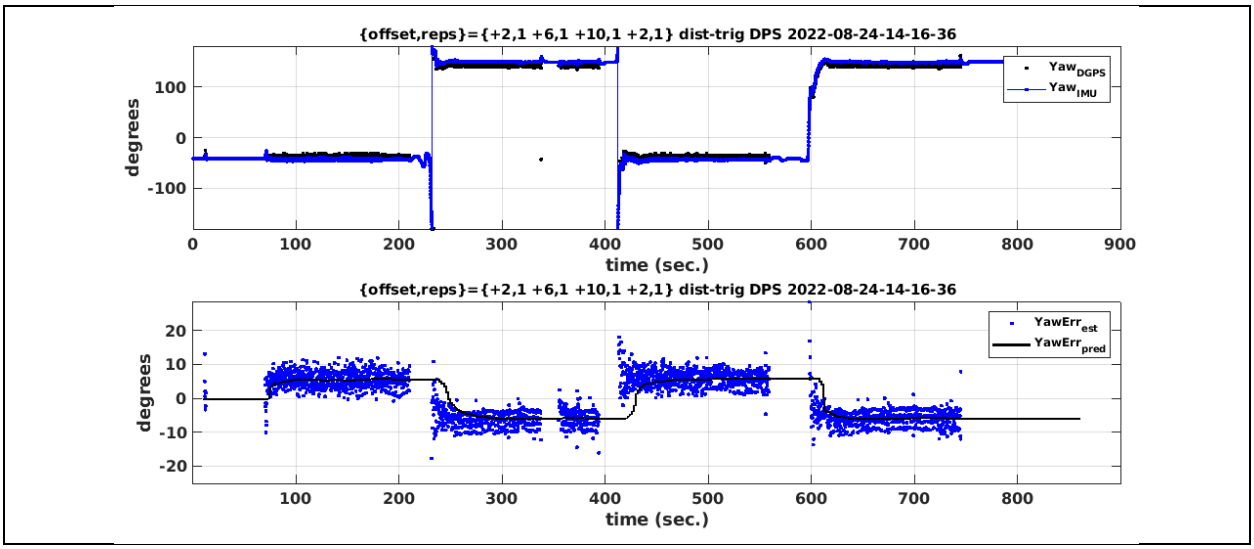
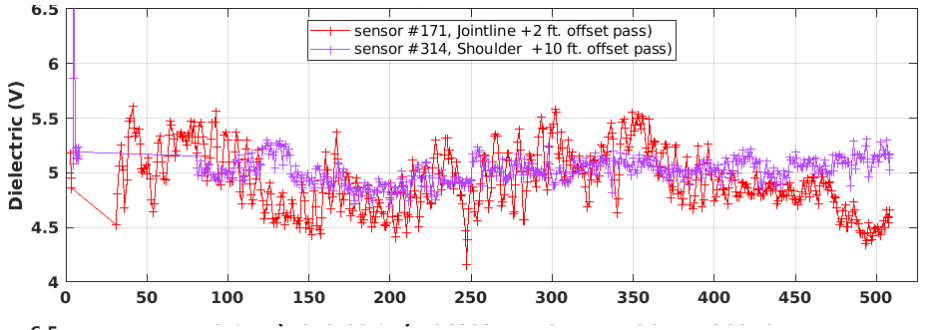


Figure 5.4: yaw error bias correction (black) using GPS-based yaw estimates (blue).

The traversal tracking behavior of the field test experiment are illustrated in figure 5.3 , with the GPR dielectric data show in figures 5.5 and 5.6 . The lateral robot position errors were -2.61±5.53 inches in each of the forward passes and 3.08±6.95 inches for the two return passes. Note that the transient conditions contributed to the bias estimates, as well as to the standard deviation errors. The lateral position errors were within 6 inches 92.8% of the time for the forward traversal passes, and 84.2% for the two reversal passes. The DMI data could not be used since there is no discrimination between path direction, or transitioning from one path to the next. Instead, the GPR dielectric sensor data was matched through associating the DGPS geodesic longitude, latitudes recorded on the robot and the PaveScan GPR system. In addition, the PaveScan GPR data were retained only for robot positions that were within 6 inches of the prescribed offset path for each pass, resulting in 95.0% and 84.5% of the robot traversal positions, 78.0% and 90.3% of the robot traversal positions, within the +2 and+10 feet forward offset passes, and+6 feet and +2 feet returning offset paths, respectively.

Forward +2 ft.,+10 ft. Traversals



Reverse +6 ft.,+2 ft. Traversals

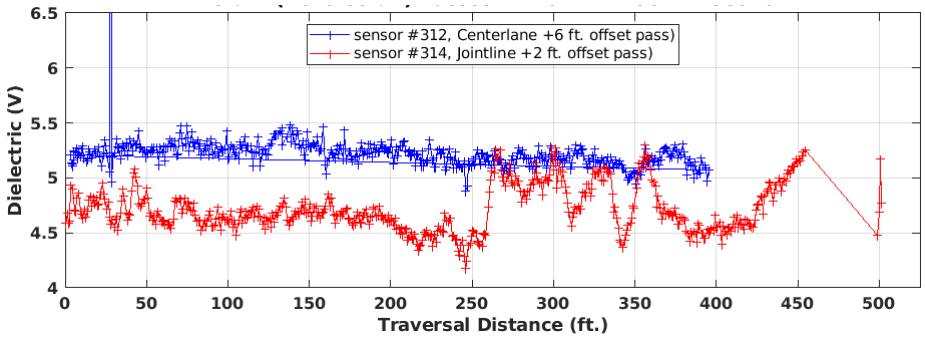


Figure 5.5: GPR recorded sensor data during the robot semiautonomous traversal of the prescribed 2/6/10 ft. offset paths.

In-situ experiments at a road construction site were also conducted on fresh pavement, using the second robot (Figure 3.3). The road section used, started at the beginning of a highway overpass bridge, and included horizontal and vertical curve alignment profiles. The experiment traversal tasks were a forward-and-return 500 feet traversal with a 2-feet offset from the designed joint centerline. Initially, the lateral error was particularly problematic during the return pass of the robot. The last half of the return pass (and accordingly the first half of the forward, outward pass), because of the highway overpass, had a considerable elevation vertical alignment component; a pitch angle between 8 and 10 degrees over the first 200 feet, and then 5 degrees, over the next 150 ft., were estimated by the robot IMU. Under this scenario, it was observed that the speed controller saturated the torque output of the motors for the desired target traversal speed, while climbing the hill, when the pitch angle exceeded 5 degrees. The robot steering controller was tuned previously in the field for level roadway conditions for the 3 MPH traversal speeds. The solution was to therefore reduce the traversal target speed to 2 mph. After this adjustment the lateral traversal error for the forward direction was 1.64±2.40 inches, and for the reverse direction was -0.59±2.50 inches, with 100% and 97.6% of the robot positions within 6 inches of the prescribed path for the forward and reversed directions respectively.

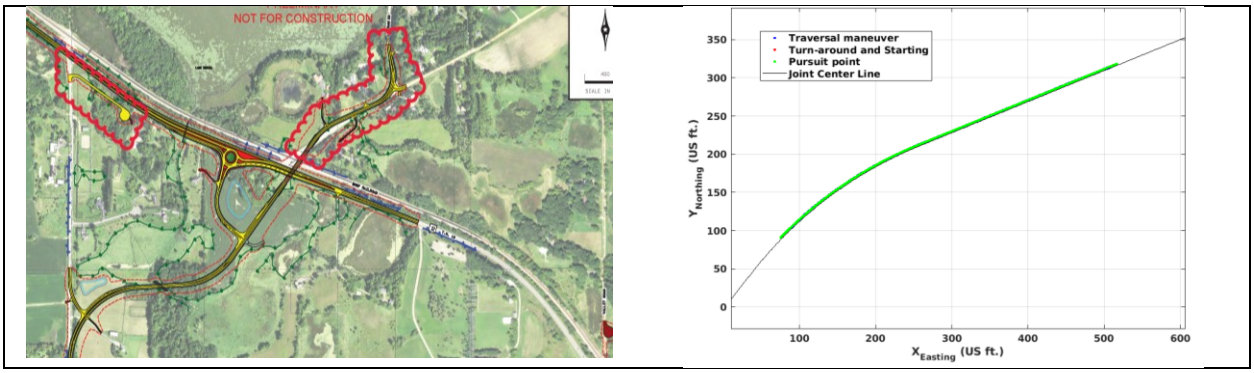


Figure 5.6: CSAH 92 site robot semiautonomous multiple path traversal data.

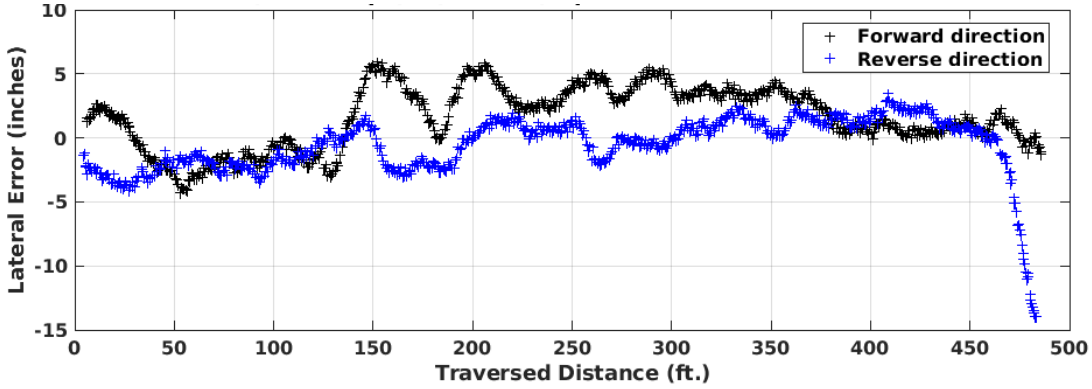


Figure 5.7: CSAH 92 semiautonomous robot traversal lateral errors.

5.2 CONCLUSIONS

Generally, the experimental data indicates the current robot and software architecture is capable of relatively accurate tracking of the design centerline. Larger lateral errors particularly at the endpoint locations, during multiple pass transitions might be further reduced by considering observed correction bias behaviors. Because the lateral GPR positioning with respect unconstrained and constrained joint lines is critical to assessing compaction levels, the recorded GPR data were filtered according to a prescribed lateral error estimated by the robot. As illustrated in figure 5.4 , generally the bias correction, at least in the same localized area will remain equal--but opposite, sign, when the robot transitions from one direction to another. Furthermore, it is then logical to assume the prior belief of the bias, would also be incorrect; or at least the prior belief should be reset to an initial uniform distribution. The software architecture did not use such assumptions and instead dynamically corrected for this within about 10 seconds. During this period, the incorrect robot heading will result in over or under steering conditions. Future implementations of the software architecture will allow these assumptions to be realized for further testing (see Appendix A). Further steering control tuning may also improve the lateral error characteristics. Nevertheless, one potential solution is to increase the target traversal starting and ending distance locations--slightly beyond the desired traversal segment, to mitigate the effects of the transient behavior.

There are some limitations to the current architecture that could be considered to improve the software architecture. One disadvantage to the approach is that there can be a tendency to over- or under-steer for small radii of curvature. However, under the assumption that the design speeds of the roads are significantly higher than the robot traversal speeds, this shortcoming does not contribute to the path following errors. Another enhancement to our architecture would accommodate GPS sensor loss when, for example, the GPS satellite views are obstructed if the robot passes under a bridge, or when timely differential correction updates from the MnCORS VRS are lost. Methods have been proposed for the latter problem using deep recurrent neural networks to forecast pseudo-distance corrections, for example [31]. However, these methods require additional processing hardware and software integration. Addressing the former issue might be achieved using the current processing architecture with a Kalman Filter, using the GPS data measurements, or with an extended KF, for the non-holonomic differential steering model using wheel odometry estimates from the servo encoder output [32, 33]. Lastly, is to evaluate the utility of the ROS Bridge protocol module to further develop the remote communication mechanism for Smart Phones and other web-based applications. The mechanism provides an accessible API for non-ROS applications to access ROS topics.

CHAPTER 6: AUTONOMOUS ROBOT PERCEPTION

6.1 INTRODUCTION

The previous chapters summarized the methods and results for robot autonomous navigation along a prescribed path and designed joint line input data. For a higher level of autonomous operation, the robot will be required to alter the traversal behavior, based on sensor information of its environment. In typical deployment scenarios the robot will be operating in coordination with road construction activities, and potential (and unknown) deviations of the paver lift trajectory from the prescribed road centerline descriptions. The focus of this chapter is to summarize the external environment sensing architecture to alter the traversal behavior if, and when, the robot encounters unexpected obstacles, as well as dynamic detection for the as-built joint lines formed during the road construction process. In particular, the object detection algorithm implemented within the ROS architecture, and its performance using collected sensor data in section is described in section 6.2 .. Then in section 6.3 a joint detection and localized path reconstruction methodology implemented through the ROS architecture is summarized as well as several experiments that evaluated the performance.

6.2 OBJECT COLLISION DETECTION

6.2.1 Object Collision Detection Algorithm

As discussed previously, an objective of the study was to test two low-cost sensors—the single row-scan LiDAR, and 2D/3D depth camera, for its efficacy in detecting potential conflicting objects and allowing the robot to act to avoid collisions with them. The objective of the collision detection algorithm is to recognize moving or fixed objects within a defined XYZ detection boundary and to then decelerate the robot up to a complete stop, until the conflicting obstacle clears, or otherwise stop the traversal operation completely and require the operator to manually intervene. The decelerative behavior is heuristically determined. The deceleration behavior is adjusted by configuring three longitudinal collision zones through ROS parameters (). Within each zone, a target speed is defined, which will affect the deceleration behavior of the robot. The default heuristic strategy is to reduce the target speed as the longitudinal distance detected object decreases relative to the robot. A negative target speed effectively increases the deceleration rate up to the robot coming to a complete stop. The default parameter values were determined experimentally.

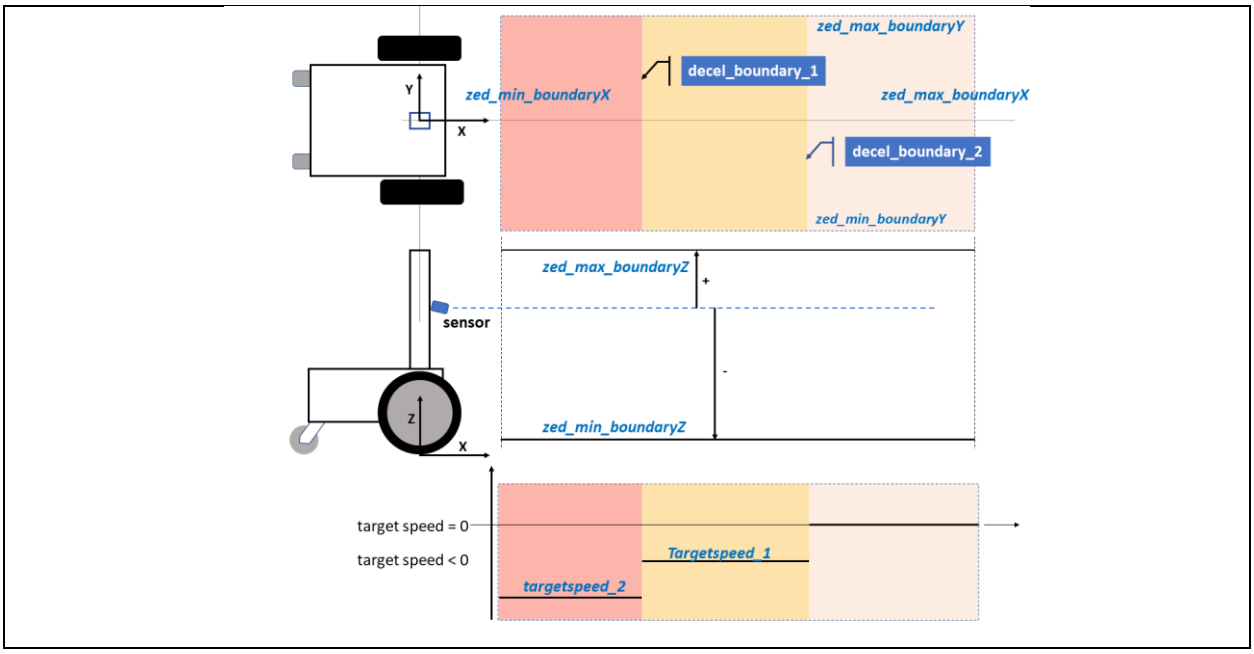


Figure 6.1: Collision detection envelope with ROS parameter definitions.

The method is applicable to both, the 2D LiDAR and the 3D camera sensors (ROS packages, *apdmv_colldetlidar* and *apdmv_colldetzed*). Both sensors generate xyz point cloud sample data; the LiDAR device driver publishes standard ROS *sensor_msgs/LaserScan* messages, while the zed 3D camera publishes standard ROS *sensor_msgs/PointCloud2* messages. The point clouds are then transformed into the robot local body coordinate system (LBCS) using parameters defined within in a published ROS *tf static transform*, followed by clipping the cloud by the boundary envelope range defined by the aforementioned ROS parameters. To reduce the point cloud size and complexity, the remaining point cloud is discretized into larger resolution (5 cm) occupancy grid XYZ voxels. The voxelated points are clustered into identifiable 'blobs' using the Point Cloud Library implementation of the K-means Euclidean cluster extraction [34, 35]. A 3D bounding box representation of the grid blobs are then stored and passed through a ROS message, apdmv/*zed3d_colldetection*. The bounding box location that is associated with the minimum longitudinal distance from the robot LBCS origin is then used to trigger robot deceleration.

6.2.2 Object Collision Detection Experimental Data

Prior to conducting field experiments to test the sensors, calibration procedures were performed to estimate the transform parameters for expressing the 2D/3D data within the robot LBCS. Regarding the LiDAR, the LiDAR sensor absolute angle is reported by the manufacturer to be within ±3.7°. Accordingly, to then estimate accurate XY point locations from the sensor's range data with respect to the LBCS, an estimate of the offset angle, β, with respect to the robot longitudinal x-axis is required (Figure6.2). First a planar object (16 inches in length) was positioned perpendicular to the robot longitudinal axis. Data are then collected with different lateral positions of the object. The offset angle, β, is then estimated from the computed minimum range distance, parallel to the longitudinal robot X-axis. Note

that in order to reduce the bias from noisy range estimates, a nearest neighbor clustering algorithm (DBSCAN, ref.) followed by a line model hypothesis estimated with RANSAC on the XY point clouds.

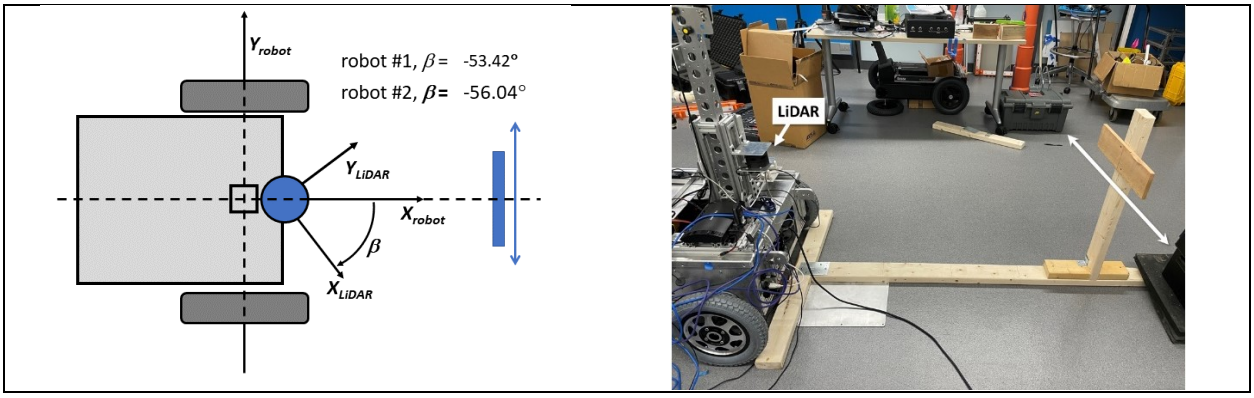


Figure 6.2: LiDAR calibration

The 3D camera sensor was calibrated through a visual alignment program. In this case, several point cloud representations of traffic cones on level pavement were aggregated, and then displayed in the visual interface with respect to a geometrical model of the robot and entering the measure XYZ location of the sensor in the LBCS. The cloud is then rotated manually about the camera sensor local axes and estimated origin location until the planar surface and cone objects are visually in front of the robot. Given the sensor mast and mounting apparatus for the camera, this a very straight forward process to perform quickly and accurately, whether in the field, or beforehand as a preparatory step before field a field deployment. Figure 6.3 illustrates a static experiment, placing the delineators at 5/10/15/20 ft., along the longitudinal robot X-axis, and ±3 ½ ft. laterally along the robot Y-axis, to test object detection performance, and to perform the 3D camera alignment procedure.

Figure 6.3: LiDAR and stereo camera static outdoor experiments using MnDOT traffic delineators.

Following the calibration, subsequent field experiments were conducted at the Woodbury site, using the test centerline provided by MnDOT. The robot traversals operated autonomously for these experiments (140 ft.), while the experimenter located himself or other objects (orange construction lane delineators) to interfere within the traversal path area of the robot. The experiments were conducted during sunny weather conditions. The detection boundaries were set to the width of the GPR sensor bar, with a minimum longitudinal boundary parameter of 2 feet (the GPR sensors mount 2 feet. The LiDAR was configured with its maximum 7 Hz scan rate at full resolution (ranging frequency = 9KHz). Despite

outdoor performance specifications advertised by the manufacturer, the LiDAR sensor could not reliably detect the orange construction delineators (a 3.5- 6mW Class 1 775-800 nm laser is used—the sensor typically draws 350mW, up to a maximum of 500mW, during operation). Anecdotally, other materials, such as wood, clothing, foliage, and vehicle bodies, were detected reasonably well within the defined target detection envelope. This was observed during collision test experiments, for example for scenarios where a person walked in front of the robot. Because of the reliability issues, no other experimentation or testing for object detection was performed with this sensor, and the stereo camera sensor data was therefore used to assess collision detection capability.

For the lowest quality/highest sampling rate setting ("vga resolution" 3D registration/30 Hz), a maximum longitudinal boundary of 12 ft. provided consistent detection. Note that that with the current time headway of 2.5 seconds for the robot, and set traversal speed of 3 mph, results in a headway distance of $2.5\ \text{sec} \times 3\ mph \times \frac{5280}{3600} = 11\ ft$. Since the GPR sensor equipment extends 22 inches from the robot LBCS center, a minimum X-longitudinal envelope parameter value of 2 ft. was assigned. The lateral envelope parameters, for the results discussed herein, were set to ±4.25 ft., which extends about 1 ft. on each side of the horizontal GPR sensor mounting bar (6.2 ft.).

Testing the algorithm was accomplished by 'playing back' the robot navigation and collision sensor data, and then evaluating expected actions. Recall the algorithm output directly intervenes with the speed and servo control of the robot, bypassing, the navigation module, and the decelerative characteristics of the robot is based on adjusting heuristic parameters empirically through experimentation (there are default values for the deceleration boundaries and their locations, which will be discussed in more detail below).

6.2.3 Object Collision Detection Results

Multiple objects – including the experimenter who also enters the detection envelope, are consistently detected. Each time a potential collision object is detected (in a continuous sequence of 2 or more, frames), a speed value is stored in the *apdmv_msgs/CollisionDetection* message and published. This message, when received by the speed and steering control package module, *apdmv_speedsteerctl,* will override any speed commands broadcast by the navigation module. Note that, after a user specified timeout period, the object collision detection module will broadcast a flag that immediately disables the closed loop steering and speed control module.

Consider scenarios with multiple overlapping and/or moving objects arising from the side or directly in front of the robot (Figure 6.4. The first case, a person is walking laterally in front of the moving robot. The second case three traffic delineators are positioned laterally from the left to right side of the robot (with the last object being moved out of the way). The next two cases, a person places the traffic delineator in front of the robot (#3); the fourth case represents detection when transitions into the turn-around state to return to the starting location of the traversal task (#4). The fifth case consists of two traffic delineators positioned laterally, with a person approaching in the middle exceeding the minimum position of the closest cone, before the robot senses the second cone; the robot is then paused (as a check of position variability). Finally, the sixth and last case, presents a person walking both laterally and

toward the robot, entering near the maximum distance of the boundary. In some cases, the person is blobbed together with a nearby traffic delineator, due to object edge reconstruction noise generating points backward into the person, or with objects behind it. However, for the purpose of collision detection, the closest longitudinal boundary position would have been similarly located even if the two objects (person and delineator) were (correctly) detected separately.

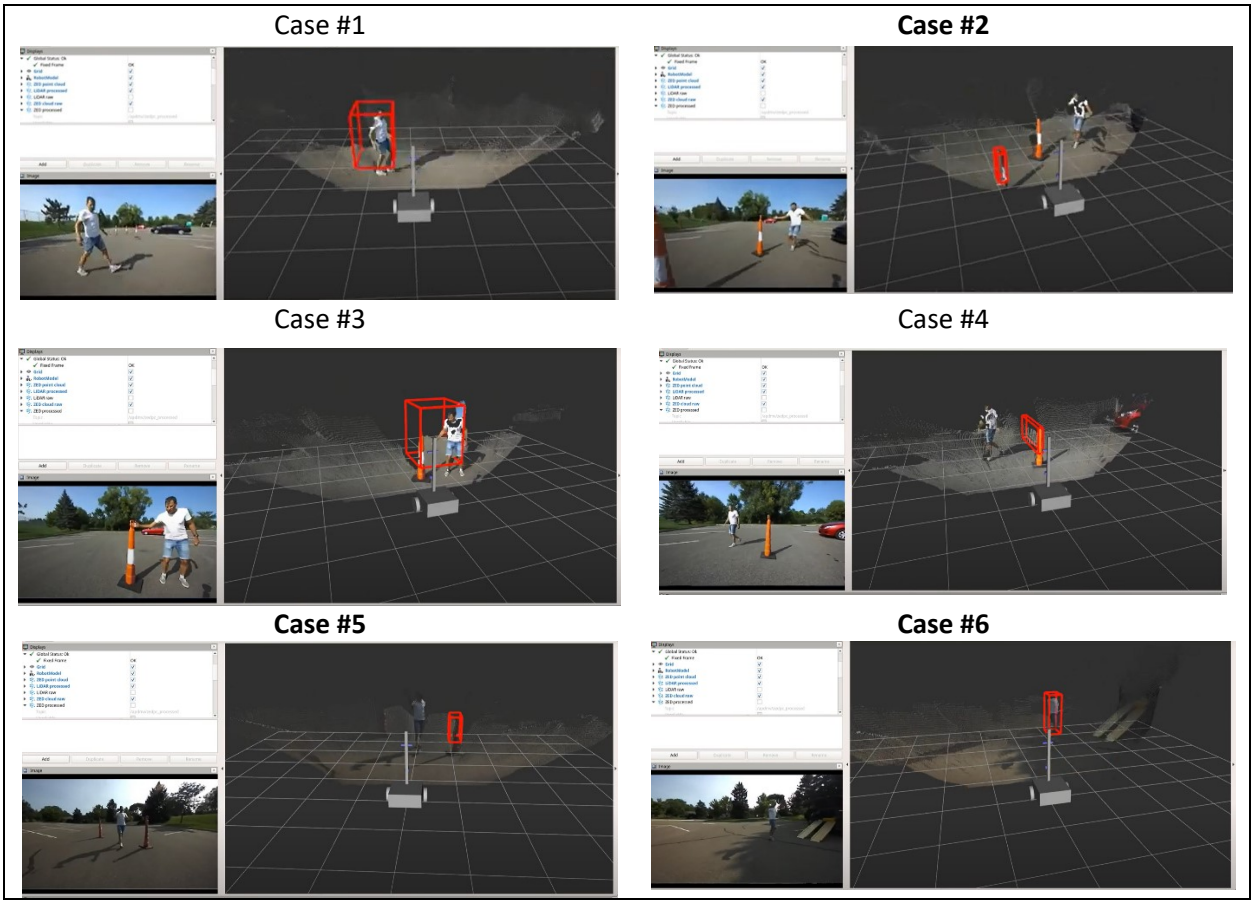


Figure 6.4: Object detection cases within the defined detection envelope (detection bounding boxes rendered in red outline).

The object detection results and actions for each of the cases is show in figures 6.5 and 6.6 . The first boundary parameter was set to 6 feet, while the second boundary parameter was set to 2 ft, with the associated target speeds were set to -2 and -4 feet per second respectively. The righthand axes in figure 6.5 indicate desired speed setpoints (green axes) triggered by the detection algorithm, that was then received by the speed/steering control module. The lefthand axes represent the longitudinal position of the closest offending object, while the corresponding lateral positions are shown in figure 6.6 . Note that these positions are estimated from the center position of the closest side of the box. As expected, when the longitudinal distance of the closest object decreases relative to the moving robot, each of the speed set points are then triggered and received by the speed and steering controller module.

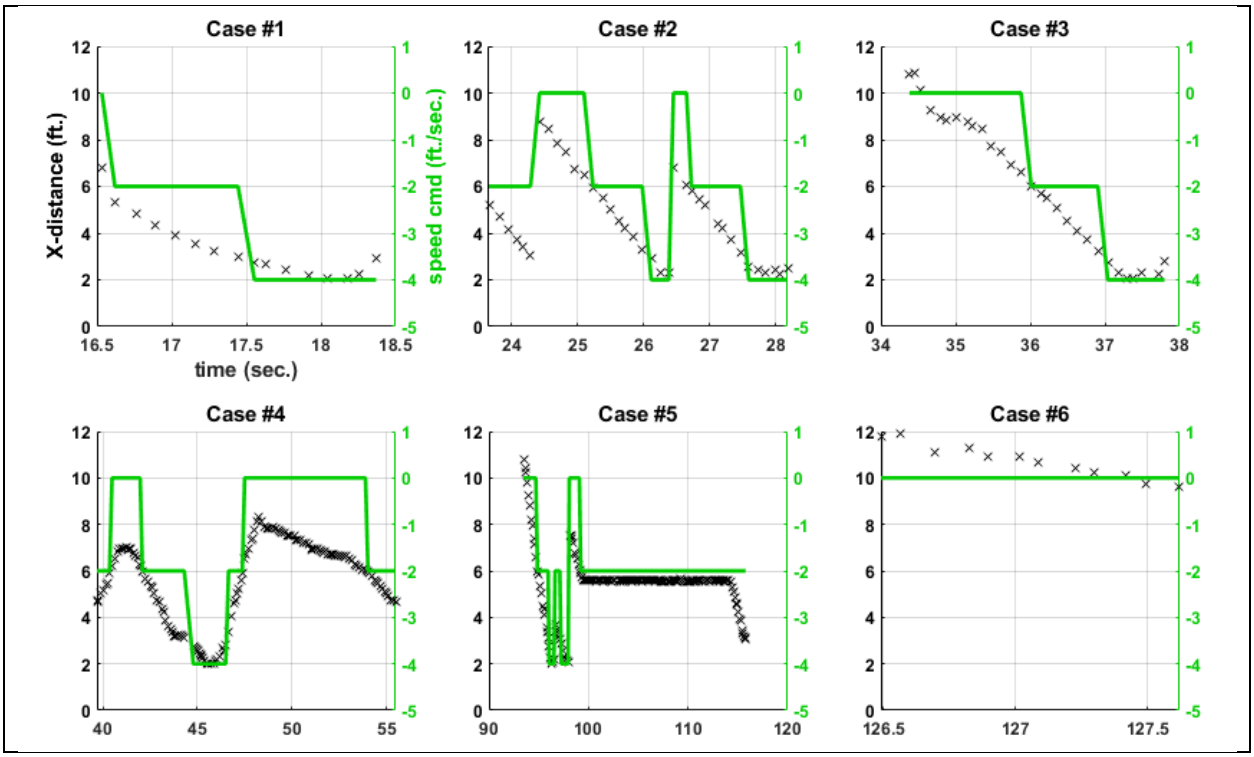


Figure 6.5: Closest-object longitudinal positions for each detection scenario in Figure 6.4 , overlaid with target speeds received by to speed control module.

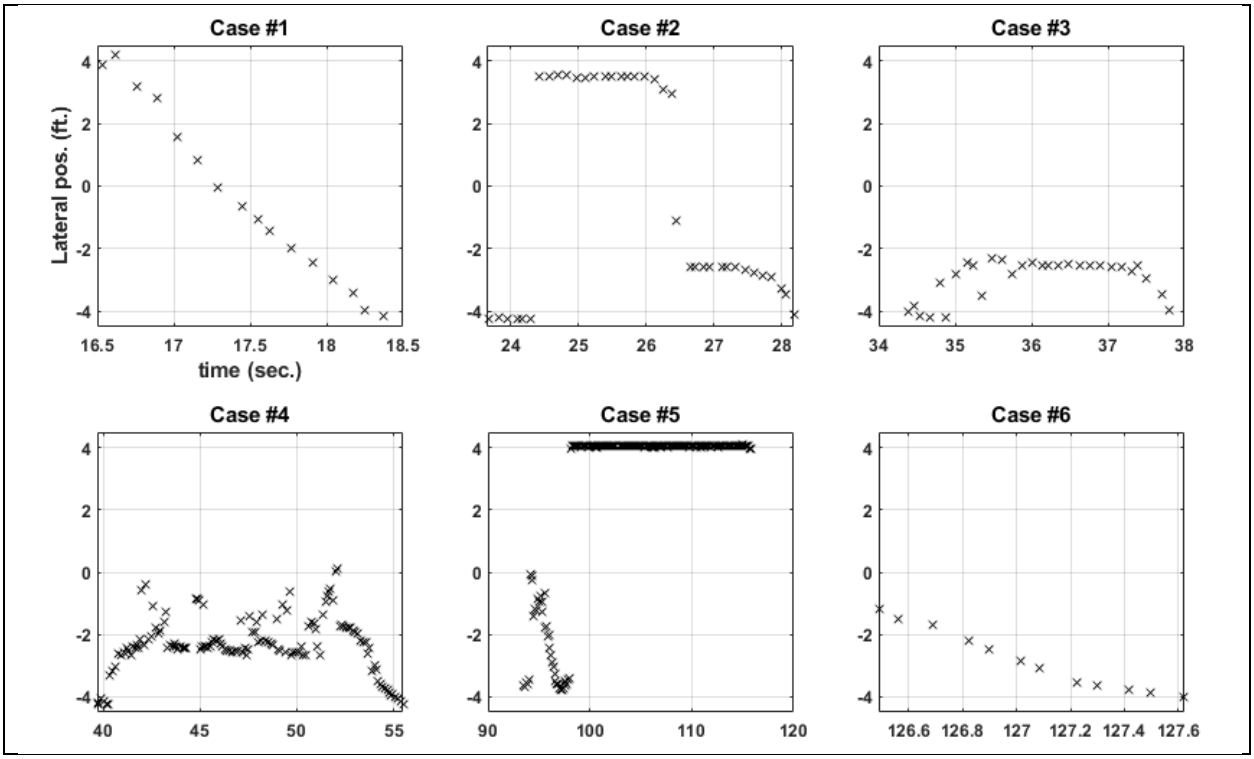


Figure 6.6: Closest-object lateral axis positions for the detection scenarios.

Effective collision avoidance by stopping the robot may not be suitable for rapidly approaching vehicles that, for example, are veering into the path envelope of the robot, due to the current Zed sensor longitudinal range limitations, as well as the fact that evasive avoidance maneuver behaviors would also likely be required. Detecting a stopped or very slow-moving vehicle (within the possible detection zone and stopping headway time of the robot) is a somewhat trivial case, given its size (the structural data of vehicles can be observed in the experiment).

6.3 AS-BUILT JOINT LINE DETECTION

6.3.1 Joint Detection Methodology

During the construction process, the as-built joint line locations can deviate from the road design joint line locations. Ideally, the robotic platform should then adjust the traversals to compensate for such deviations to adhere to operator prescribed offsets from the joint line. This is necessary for two reasons. Future road inventory and maintenance activities will benefit from having corrected as-built joint locations. Second, good construction quality assurance relies on accurately establishing correct compaction particularly near the joint lines. As mentioned previously, robot architecture approach is to utilize a thermal imaging camera to detect the joint line location during the construction process. Although the joint line—particularly constrained joint lines in fresh pavement, are visually difficult to detect precisely, there should be a significant temperature gradient across both, a constrained, and an unconstrained asphalt joint.

Figure 6.7 summarizes the image processing steps to for the as-built pavement joint lines. The thermal camera radiometric 160x120 resolution 'images' are published as a raw 16 bit representation from a UVC acquisition module through ROS (ROS *sensor_msgs/ImageConstPtr* from the ROS *apdmv_lepton* package node).

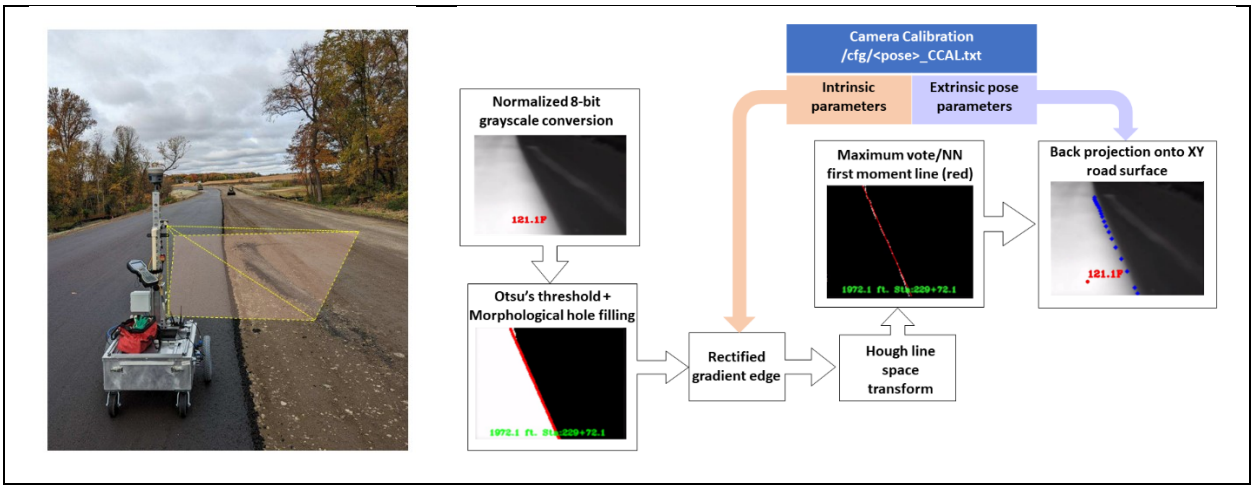


Figure 6.7: Image processing algorithm for as-built asphalt joint line location estimation.

A camera calibration procedure was designed to determine the extrinsic, and intrinsic camera parameters.

The received thermal images are first decoded to quantify temperature values, and subsequently scale normalized into to 8-bit grayscale images. Using the bottom 2/3 image region (this contains primarily foreground road surface information), a threshold to classify the 'hot' from 'cold' side pavement is derived from Otsu's method to derive a binary image [36]. Then maximal voting, with nearest neighbor first-moment Hough transform was applied to the Canny gradient edge image representation of the binary image to determine the most likely joint line within the image. The edge image is undistorted using the intrinsic radial distortion parameters estimated from a camera calibration procedure. Interestingly, 'linear crack voids' were observed within the binary image from cool water streaks running across, and longitudinally, on the pavement, which confounded the Hough line search. To remedy this issue, before the Hough voting procedure, the binary image is filtered using 5x5 morphological dilation/erosion using an isotropic ellipse mask to fill the crack (Figure 6.8). The Hough space (ρ, θ) parameters are then regularized through a linear first order Kalman filter. The *X*,*Y* coordinates are then reconstructed by back projecting the regularized, rectified image line onto the road surface, under the constant flat surface assumption (Z=0). Lastly, using the error corrected IMU robot orientation and robot global position estimates, the XY joint locations are transformed into county fixed coordinates (not shown in figure 6.7

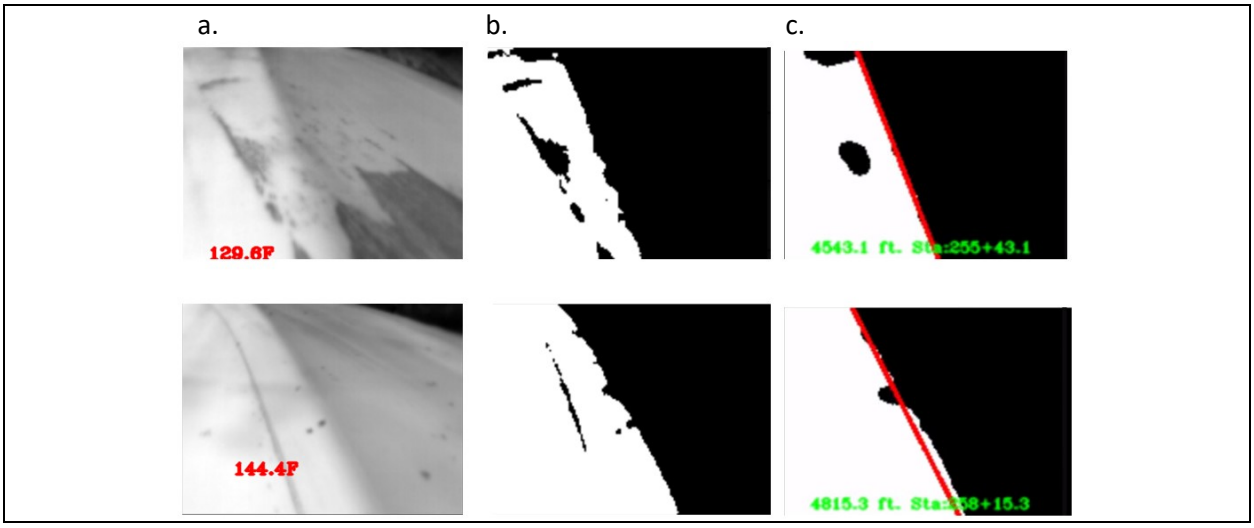


Figure 6.8: Thermal irregularities detected along constrained joint (a.) that affected line estimation, without morphological filtering (b.). The final line estimate after filtering is shown in red (c.).

The extracted coordinate list is then broadcasted through the custom ROS message, at the camera image acquisition rate of 9 Hz. The message is designed to be utilized by the robot navigation software. Under these circumstances, a specific operating mode would need to reference the dynamically constructed joint line rather than the CAD derived path centerline. Arguably, for controlling the robot, expression of the line data in the fixed county coordinates is superfluous, given that the changes in robot heading is similarly expressed in either the fixed or robot LBCS. In fact, this is why the developed

message includes the robot XY position and yaw estimates used to estimate the X, Y point estimates. However, for the forthcoming analysis, it was most convenient to leave the data in county coordinates.

An accurate estimation of the camera position, orientation, and optical characteristics are required for reconstructing the as-built joint line XY real world representation. Accordingly, a calibration procedure was also developed to estimate camera intrinsic (focal length, radial distortion model), and extrinsic parameters with respect to the robot local body coordinate system, corresponding to the adjusted LEFT or RIGHT facing poses of the LWIR camera. The procedure used the LiDAR to track real-world XYZ (Z=0) point correspondences to a thermal (*u,v*) point source identified in the radiometric thermal image. This was achieved by constructing, a calibration jig containing a 75-watt incandescent bulb mounted horizontally, 1 inch above the ground plane, and a HxW=16x4 inches wood surface, at a fixed position above the center of bulb filament, intersecting the LiDAR sweeping beam plane. Subpixel estimates were computed using the radiometric image values and blob estimate of similar temperatures [37].

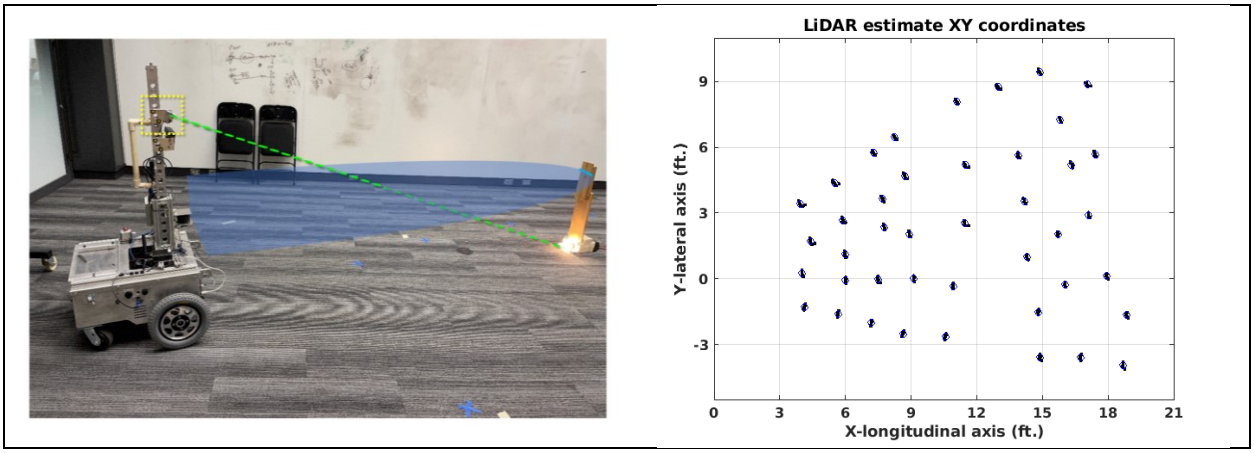


Figure 6.9: LEFT: IR camera calibration procedure; RIGHT; the generated fiducial XY target points (blue circles) and LiDAR point clouds

The procedure was carried out within the laboratory, placing the jig in different locations providing visual feedback of the radiometric binarized image output, as well as the highlighted point cloud of the tracked jig surface, to ensure that the target locations are within the LWIR camera image plane. Ideally, a large distribution of points, covering the image area is desired. Both calibrations captured point correspondences from 4 feet to between 18 and 21 feet along the longitudinal X axis of the robot and spanning from 5 to 15 feet laterally along the robot Y axis. The data collected for the RIGHT facing camera setting is shown in figure 6.9 .

The results of the calibration are summarized in Table 6.1. The resulting calibration generated reasonably accurate reconstructions. In retrospect, it may have been advantageous to tilt the camera downward to increase the near-robot coverage and accuracy, at the expense of reducing the observable range of the camera. Theoretically, the procedure could be accomplished on-site, since all the processing is done on the robot.

Table 6.1: IR camera calibration error statistics

View pose	μ±σ (mm)	Max err (mm)	*N*
Robot LEFT side	5.14 ± 2,86	11.28	47
Robot RIGHT side	5.35 ± 3.28	17.01	51

6.3.2 Joint Detection Field Experiments

Five experiments manually controlled robot traversal experiments were conducted one hour after the last roller pass on fresh pavement at the County-State Aid road construction site described previously (refer to the map shown in Figure 5.6 in chapter 05). The IMU and DGPS data were recorded along with the LWIR camera data. Referring to figure 6.1 , the first two experiments traversed the robot about ½ mile south of the HWY 12 overpass, collected data along an unconfined road centerline joint, with the robot traversing on the 'hot' pavement surface. A third experiment returned North/East bound back toward the overpass, along the unconfined shoulder edge. A fourth pass, with the robot traversing instead on the opposite, Southwest bound lane, collected the same unconstrained joint as a constrained joint with the presumably higher temperature pavement on the Southwest bound lane. Experiments were conducted North of the overpass. Specifically, the robot was manually traversed on an out (FWD) - and-back (REV) pass 1300 ft offset about 2 feet from the as-built constrained joint centerline. The robot traversed on the 'hot' final layer fresh pavement side while traversing along the forward Northbound pass (within the Southbound traffic lane), and returned in the reverse direction, Southbound, on the colder final layer side (associated with the Northbound traffic lane). The robot was operated with the GPR payload collecting profile density data.

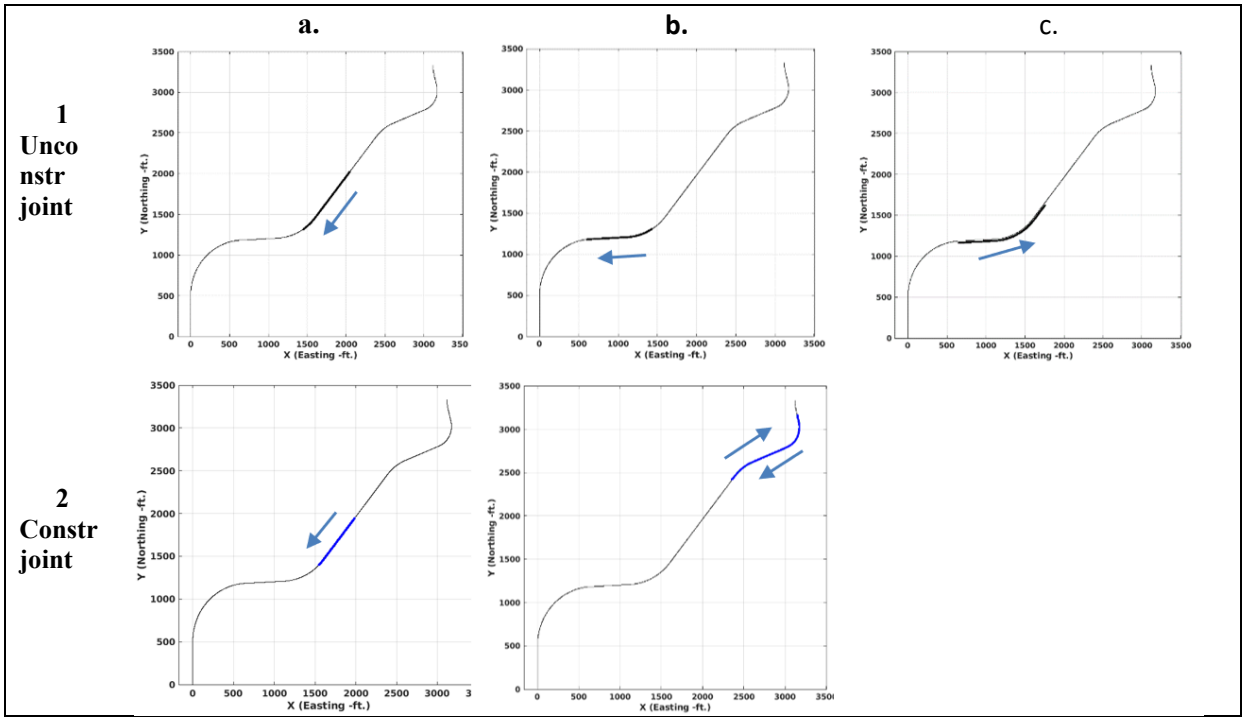


Figure 6.10: Robot as-built detections along traversals. TOP row (1); unconstrained joints, with 1.c, along shoulder, row (2); constrained joint runs.

Table 6.2: Asphalt joint detection experiments (Refer to 6.1 for visual representation)

	Joint type	Traversal task(offset rel. to road cent. Line)	Traversal distance
1.a.	Centerline-unconstrained	2 ft. offset, 'fresh' NB side	920 ft.
1.b.	Centerline-unconstrained	2 ft. offset, 'fresh' NB side	1186 ft.
1.c.	Shoulder-unconstrained	15 ft. offset, 'fresh' NB side	1830 ft.
2.a	Centerline-constrained	2 ft. offset, 'fresh' SB side	770 ft.
2.b	Centerline-constrained	Out: 2 ft. offset, 'fresh' side, Return: 2 ft offset, 'cold' side	1300 ft. 1300 ft.

The latter two experiments can infer the repeatability since there are two cases were the robot detected the same as-built joint line under two different scenarios. The first, detected the same joint—a relatively straight-line segment, both in a constrained and unconstrained representation (data from the first traversal pass experiment was also required). Furthermore, the camera positioning perspective was also adjusted in order to face either the left or right side of the robot (the robot ran parallel, with about 2 foot an offset from the as-built joint line, replicating a typical usage case when deploying the GPR

sensor to assess compaction near the as-built joint). The other scenario, encompassing the second experiment, detects the as-built constrained joint along greater horizontal and vertical alignment changes than the first experiment. The same camera position was used for the forward and reverse passes, since the asphalt joint is within the perspective view of the camera sensor traveling on opposite sides. This case eliminates any bias and error effects from two separate camera calibrations.

Table 6.3: Asphalt joint line detection repeatability and as-built offset estimation

Traversal experiment	Repeatability $\mu \pm \sigma$	As-built offset deviation from design joint line $\mu \pm \sigma$
Runs (1.a) vs. (2.a)	2.37±4.62"	8.16±4.29 (1.a) 5.88±3.60 (2.a)
Run (2.b.)	-0.57±5.69"	22.91±10.66" (outward, FWD pass) 22.45±10.57" (return, REV pass)
Run 1.b	N/A	3.14±6.67"
Run 1.c	N/A	-236.28.±6.74" (μ=-19.69 ft.) note: this is a shoulder line

The repeatability results are summarized in table 6.3 . The lateral difference error data between two detected joint line locations of the same asphalt joint are illustrated in figures 6.11 and 6.12. For the combined experiment runs 1.a, and 2.a., and the forward-return pass experiment 2.b., in-situ joint locations were detected within ±6 inches 84% of the time, and 81.8% of the time—including the biases, respectively. If one considers a moving DGPS uncertainty standard deviation to lie with ±1 inch, the remaining error arises from deviations of the joint line (recall that the near proximity estimate fits a straight line), deviations from primarily the robot yaw (heading) estimate, and camera sensor and calibration errors. Based on the lateral resolution of the camera, camera sensor errors could theoretically contribute ±1/3 to over ±1 inches. Although heading error corrections are applied, the location estimates will still be very sensitive to remaining small robot yaw errors. For example, consider a yaw error range of ±1 degree of remaining will affect the extracted measurement from the robot by: $sin(\pm 1°) \times \{distance\ from\ robot\} = \pm 0.0174$ x 5 ft. to $\pm\ 0.0174$ x 10 ft. which equates to ±1 to ±2 inches. The larger lateral error in the outward (fwd)-return(reverse) traversal experiment is partially attributed to line estimate inaccuracies caused by sporadic--or otherwise very low temperature gradients, near the as-built joint line. For example, for the first 660 ft in the forward direction, a team member was walking on top of the joint line to assist in identifying the as-built joint line, between 10 to 20 feet ahead of the robot. From observation, other portions of the traversal with significant curvature deviates from the straight-line model hypothesis that resulted in biasing the direction of the straight-line model estimate (bottom row of Figure 6.9).

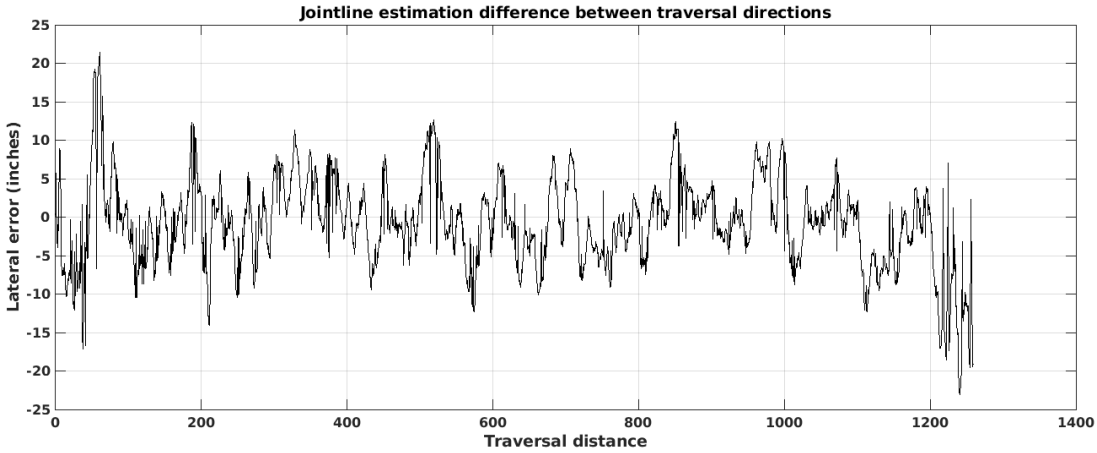


Figure 6.11: Detected joint line estimation difference between traversal directions for run 2.b.

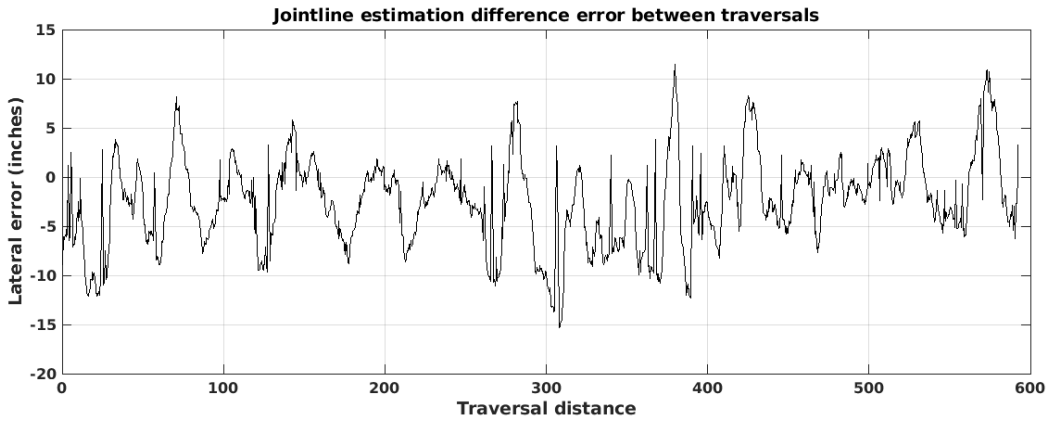


Figure 6.12: Detected joint line estimation difference between traversal runs (1.a) and (2.a)

The results of as-built joint line locations for the case detecting the same unconstrained joint proceeding further upstream, indicated a small overall bias of 3 inches and with deviations ±6 inches. The last case, following the unconstrained joint line indicated an offset with similar ±6 inches deviations, between 19 and 20 feet from the design centerline. Regardless of the repeatability error sources, the joint line detection data clearly indicate significant location offsets between the design joint reference and the as-built joint (figures 6.13and 6.14).

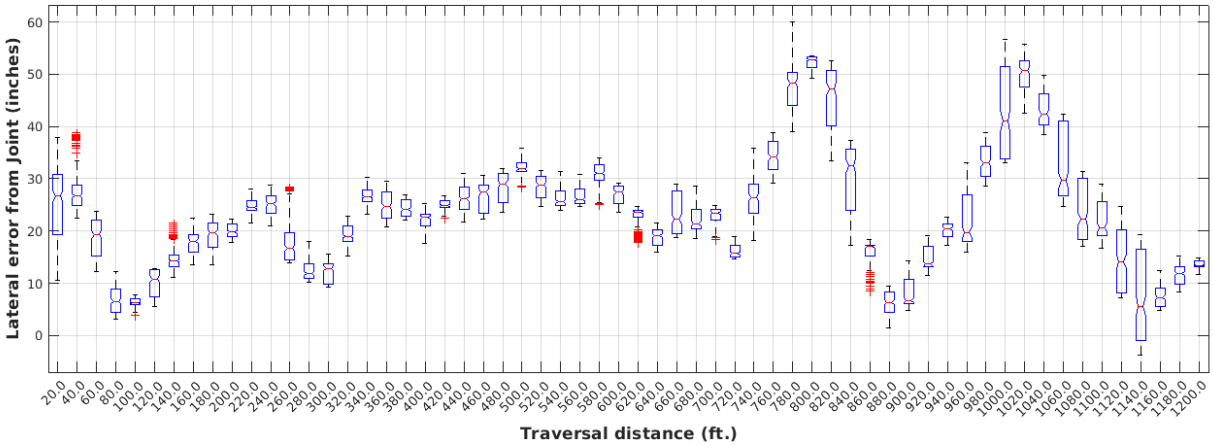


Figure 6.13: Combined path offset estimate of as-built centerline located joint from design road centerline for (1.a-2.a)

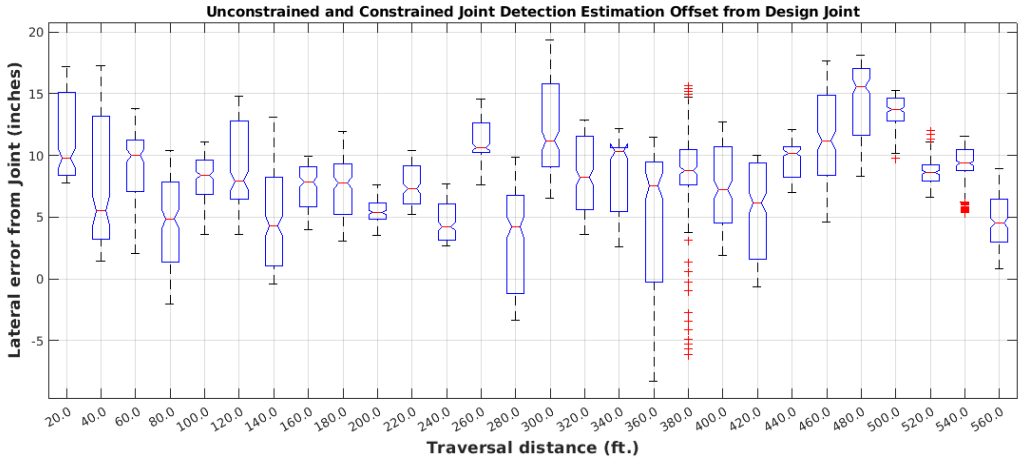


Figure 6.14: Combined path offset estimate of as-built centerline located joint from design road centerline for (2.b.)

To conclude, to conduct the 'real-time tests' for evaluating the capability of the robot to detect as-built joint lines, an 'in-situ simulation' was developed to 'playback' the recorded data through the robot navigation module and the proposed pavement joint line detection module. This was a similar approach for evaluating the in-situ collision detection performance. Note that the joint line detection experiments being completed after one hour after the last roller pass was a stipulation from the construction project manager to ensure the research teams safety and avoid any interference with the construction work. Under more common working scenarios for the GPR collection, if such operation experiments could be conducted in a shorter time period, then the temperature gradient between the 'cold' and 'hot' fresh pavement surfaces will presumably be greater —which will improve detection performance. Additional error sources could be explored and mitigated, for example using prior road curvature design specifications rather than the straight-line assumption. Accordingly, piecewise linear

model hypotheses could be devised to better accommodate curve alignments. Tilting the camera sensor further downward, would theoretically improve the camera calibration and therefore detection accuracy—assuming there is no foreground interference and occlusion from the GPR array.

CHAPTER 7: CONCLUSIONS AND RECOMMENDATIONS

A straightforward, portable, open-ended, robot architecture was proposed and tested for accomplishing autonomous asphalt pavement profile density measurements, using the sensor observations from a RTK GPS receiver and low-cost IMU sensor, and typical lateral path errors remained within 6 inches, or better. Additional tuning may reduce the lateral error, and thus further experimental testing would be beneficial. An important improvement in the robot architecture would require a significantly more complete integration of the GPR system payload.

Because the vision is to transfer and adapt autonomous robot technologies for this methodology, an important aspect for future efforts would be tightly integrating the GPR system with the robot software architecture. The tested system has a very closed architecture that cannot be easily integrated with the robot. Specifically, the robot cannot directly acquire and control the radar pulse sampling of the dielectric data, and it cannot use the servo encoder sensor outputs with the PaveScan controller device for distance triggered GPR sensor data acquisition. Although the robots directly use the PaveScan distance measurement instrument (DMI) and could be calibrated to precisely trigger collection of the GPR system, the DMI data acquired by the GPR system cannot discriminate directionality and location for multiple pass traversals. Furthermore, the operator is required to enter traversal job characteristics twice—once for the robot configuration, and the other for the GPR system configuration. Ideally, this information would be conveyed to the robot in order to use the same user interface; this would reduce the field operation complexity significantly.

Significant as-built joint line deviations from the road design centerline appeared to be evident for the field experiments. The detection architecture provided dynamic adjusted centerline data through ROS messaging, with the same format and county projection planar coordinates as the supplied LandXML path segment representation, described in Chapter 3. Future research should focus on modifying the robot path following traversal behavior in real-time and the effects of detection accuracy (e.g., dynamic path planning). That is, a pursuit point would be projected onto this line instead of the originally prescribed path, using the desired time-headway parameter and speed, assuming a decision boundary to reference it can be determined under varied location uncertainties. Similarly, under the same detection measurement uncertainties, further investigation would be needed to determine the best descriptive alignment representations for the as-built joint centerline.

The current robots require full fix correction to the DGPS data; the robot will stop traversal if this requirement is not met during any aspect of a path traversal task. However, there may be situations where this is unavoidable. For example, traversing under an overpass. Research to explore kinematic models and predictive filtering or predictive fix modeling is recommended to address this issue.

Part of the research project focused on technology transfer. An operator procedural field guide was drafted and delivered as part of the project (Appendix A). As a supplement, in-lab practice operations that included all phases of set-up and tasking the robot at the job site were performed; in this case, a DGPS emulator was used to provide the WGS84 latitude/longitude NMEA streams at the MnROAD test locations, and the robot was put up on blocks to ‘run’ the task. Although the simulation platform was

another alternative, it cannot offer the same practicum for testing and working with the actual robot hardware to enforce correct setup procedures. Two robots were built to test the transferability of the software architecture. An aspect for both robots is the scalability to potentially operate several of them simultaneously to reduce data collection time, while increasing coverage. Robot-to-robot communication should be explored to provide, for example, platooning, or optimal path planning to guarantee coverage; these capabilities would also benefit from a more open, flexible architecture of the GPR system payload.

REFERENCES

[1] MnDOT (2021). *2020 Pavement Condition Annual Report,* Office of Materials and Road Research, Minnesota Department of Transportation, Saint Paul, MN

[2] MnDOT. (2022, August). Assett Management, *MnDOT Transportation Asset Management Plan (TAMP)*, Minnesota Department of Transportation, St. Paul, MN. Retrieved from https://www.dot.state.mn.us/assetmanagement/tamp.html.].

[3] MnDOT, (2014). *Transportation Asset Management Plan* [Draft], Minnesota Department of Transportation, Saint Paul, MN. Retrieved from http://www.dot.state.mn.us/assetmanagement/pdf/tamp/2014-draft/tamp2014.pdf

[4] M. Smith-Jackson. (2021, December). *Asset Management*, Federal Highway Administration. Retrieved from https://www.fhwa.dot.gov/asset/plans.cfm. [Accessed 20 February 2023]

[5] E. Mostafa, A. M. Abdel-Khalek, & K. Dasari. (2012). *Implementation of Rolling Wheel Deflectometer (RWD) in PMS and Pavement Preservation*, Louisiana Department of Transportation and, Baton Rouge, LA

[6] R. N. Linden, J. P. Mahoney, & N. C. Jackson. (1989). Effect of Compaction on Asphalt Concrete Performance, *Transportation Research Record, 1217,* 20-28

[7] E. Zeube & R. Forsyth. (1966). Flexible Pavement Maintenance Requirements as Determined by Dflection Measurement, *Higwhay Research Record, 129*(1), 60-79.

[8] B. B. Guzina & R. H. Osburn. (2002). Effective Tool for Enhancing Elastostatic Pavement Diagnosis, *Transportation Research Record, 1806*,(1), 30-37.

[9] P. Romero & F. Kuhnow. (2002). Evaluation of New Non-Nuclear Pavement Density Gauges with Data from Field Projects,*Transportation Research Record, 1813,* 47-54.

[10] I. I. Al-Qadi, Z. Leng & A. Larkin, (2011, December). *In-place Hot Mix Asphalt Density Estimation Using Ground Penetrating Radar*, University of Illinois Advanced Transportation Research and Engineering Laboratory (ATREL), Urbana, IL.

[11] J. Song, M. Abdelrahman & E. Asa (2009, October). *Use of a Thermal Camera during Asphalt Pavement Construction*, North Dakota Department of Transportation, Bismarck, ND.

[12] A. Sabato, T. Yu, N. N. Kulkarni & S. Dabetwar. (2022). *Detecting Subsurface Voids in Roadways Using UAS with Infrared Thermal Imaging*, Massachusetts Deparment of Transportation, Boston, MA.

[13] S. S. Kim, S. A. Durhan, J. J. Yang, A. Abdelmawla & R. Romeo. (September, 2021). U*se of Ground Penetrating Radar Technology to Assess and Monitor Pavement Structural Conditions for Improved Pavement Maintenance and Rehabilitation Strategies*, Georgia Department of Transportation, Atlanta, GA..

[14] H. Liu, Z. Yan, Y. Yeu, X. Meng, C. Liu & J. Cui. (2023). Asphalt Pavement Uharacterization by GPR Using an Air-Coupled Antenna Array, *NDT and E Interational, 133*, 1-8.

[15] D. R. Huston, N. V. Peczarski, B. Esser & K. R. Maser. (2000). Damage Detection in Roadways with Ground Penetrating Radar. In *Proc. SPIE 4084, Eighth International Conference on Ground Penetrating Radar,* 4084, 91-94.

[16] K. Hoegh & S. Dai. (2017). Asphalt Pavement Compaction Assessment Using Ground Penetrating Radar-Arrays. Paper presented at the *Congress on Technical Advancement,* Duluth, MN, September, 118-126..

[17] D.-C. Woo. (1995). Robotics in Highway Construction and Maintenance, *Public Roads,* 58(3), 26-31. Retrieved from https://highways.dot.gov/public-roads/winter-1995/robotics-highway-construction-and-maintenance

[18] S. Shim, S.-W. Lee, G.-C. Cho, J. Kim & S.-M. Kang. (2023). Remote Robotic System for 3D Measurement of Concrete Damage in Tunnel with Ground Vehicle and Manipulator, *Computer Aided Civil and Infrastructure Engineering,* 00, 1-22.

[19] C. Yuan, B. Xiong, X. Li, X. Sang & Q. Kong. (2022). A Novel Intelligent Inspection Robot with Deep Stereo Vision for Three-Dimensional Concrete Damage Detection and Quantification,*Structural Health Monitoring, 21*(3), 788-802.

[20] B. Ramalingam, A. A. Hayat, R. M. Elara, B. F. Gómez, L. Yi, T. Pathmakumar, M. M. Rayguru & S. Subramanian. (2021). Deep Learning Based Pavement Inspection Using Self-Reconfigurable Robot, *Sensors, 21*(8), 1-23.

[21] Y.-H. Tseng, S.-C. Kang, J.-R. Chang & C.-H. Lee. (2011). Strategies for Autonomous Robots to Inspect Pavement Distresses, *Automation in Construction, 20,* 156-1172.

[22] A. Sheta & S. A. Mokhtar. (2022). Autonomous Robot System for Pavement Crack Inspection Based CNN Model, *Journal of Theoretical and Applied Information Technology, 100*(16), 5119-5128.

[23] H. M. La, R. S. Lim, B. B. Basily, N. Gucunski, J. Yi, A. Maher, F. A. Romero & H. Parvardeh. (2013). Mechatronic Systems Design for an Autonomous Robotic System for High-Efficiency Bridge Deck Inspection and Evaluation, *IEEE/ASME Transactions on Mechatronics, 18*(6), 1655-1664.

[24] T. Le, S. Gibb, N. Pham, H. M. La, L. Falk & T. Berendsen. (2017). Autonomous Robotic System using Non-Destructive Evaluation methods for Bridge Deck Inspection. Paper presented at the *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 29 - June 3.

[25] D. A. Krishnamurthy. (2008). Modeling and Simulation of Skid Steered Robot Pioneer 3AT (MS Thesis), Florida State University, Tallahassee, FL.

[26] L. E. Ortiz, E. V. Cabrera & L. M. Goncalves. (2018). Depth Data Error Modeling of the ZED 3D Vision Sensor from Stereolabs, *Electronic Letters on Computer Vision and Image Analysis, 17*(1), 1-15.

[27] L. E. Ortiz, E. V. Cabrera & L. M. Goncalves. (2018). Depth Data Error Modeling of the ZED 3D Vision Sensor from Stereolabs, *Electronic Letters on Computer Vision and Image Analysis, 17*(1), 1-15.

[28] J. Elfring, E. Torta and R. van de Molengraft (2021), Particle Filters: A Hands-On Tutorial, *Sensors,* vol. 21, no. 2, pp. 1-28.

[29] G. Bitner (2009, April). Map projections and parameters, Minnesota Department of Transportation,.: Retrieved from https://www.dot.state.mn.us/surveying/toolstech/mapproj.html

[30] PROJ contributors. (2003, April). PROJ Coordinate Transformation Software Library, Open Source Geospatial Foundation. Retrieved from http://proj.org/en/9

[31] J. Jackson (2018), Real-time Kinematic Position: Background, Assessment and ForeCasting,, M.S. Thesis, University of Minnesota, Minneapolis, MN..

[32] L. Angel, C. Hernandez & C. Diaz-Quintero. (2013). Modeling, Simulation and Control of a Differential Steering Type Mobile Robot. Paper presented at the 32nd Chinese Control Conference, Xi'an, China, July.

[33] P. Cork (2017). Robotics, Vision and Control, Fundamental Algorithms in MATLAB, 2nd ed., B. Siciliano and O. Khatib, Eds., Heidelberg: Springer-Verlag, p. 693.

[34] T. Serkan. (December, 2016). Euclidian Cluster Extraction, Point Cloud Library. Retrieved from https://pcl.readthedocs.io/en/latest/cluster_extraction.html#cluster-extraction

[35] R. B. Radu. (2009). Emantic 3D Object Maps for Everyday Manipulation in Human Living Environments (Ph.D. Dissertation), Technical University, Munich, DE..

[36] N. Otshu (1979). A Threshold Selection Method from Gray-Level Histograms, *IEEE Transactions on Systems, Man and Cybernetics,* vol. 9, no. 1, pp. 62-66.

[37] M. R. Shortis, T. A. Clarke & T. Short. (1994). Comparison of Some Techniques for the Subpixel Location of Discrete Target Images,. In *Proceedings Volume 2350, Videometrics III*, Boston, MA, October.

APPENDIX A:
OPERATIONAL USERS GUIDE

1.1 OPERATIONAL GUIDE SYNOPSIS

The operational guide is organized as follows. First, section A.2., summarizes the necessary components required and their interface/assembly instructions, to deploy and operate the robot at a job site and maintain them when not in use. Either a Linux or Windows 10 notebook, or tablet PC., with Wi-Fi capabilities is required to interface and communicate with the robot (Windows 11 should be compatible, but it has not been tested). An overview of recommended software tools to install for this purpose, will be described in section A.3.

Then, step-by-step instructions, using a remotely connected notebook computer will be provided to initialize and test robot sensors, configure a traversal data collection task, and finally runm monitor the task will be given in section A.4. Methods to modify robot traversal behavior, which may be useful to further optimize traversal performance, will be summarized in section A.5. Section A.6. will provide instructions how to visualize and record robot sensor data. Section A.7 provides stepwise instructions how to enable and start collision detection capabilities for the robot, and recommended settings for this capability. The last section, A.8. provides a brief troubleshooting guide.

1.2 ROBOT HARDWARE CONFIGURATION

Operating the robot requires the following components (refer to Figure A.1):

- Trimble R8/R10 DGPS receiver and controller
- Robot sensor mast, which support the 3D camera, IR camera, IMU, sensors and the DGPS receiver and controller
- Distance Measuring Instrument (DMI) and mount connector
- PaveScan GPR system (GPR(s), mounting bar, controller, and tablet PC and associated cables)
- Separate notebook/tablet PC

The robots contain two separate battery systems (Figure A.1). One battery system, which powers the robot processing and sensor hardware, uses standard 12 Volt DC batteries. The second battery system, which powers the servo motors and servo motor controller, uses multiple 20/24 Volt batteries. For the hardware power, the 28 to 30AH battery can supply adequate power from 4 to 8 hours (depending on which hardware components and features are being utilized). The motor controller batteries have a design specification for 3-4 hours. Beyond payload and continuous actuation time, the actual performance also depends on environmental conditions.

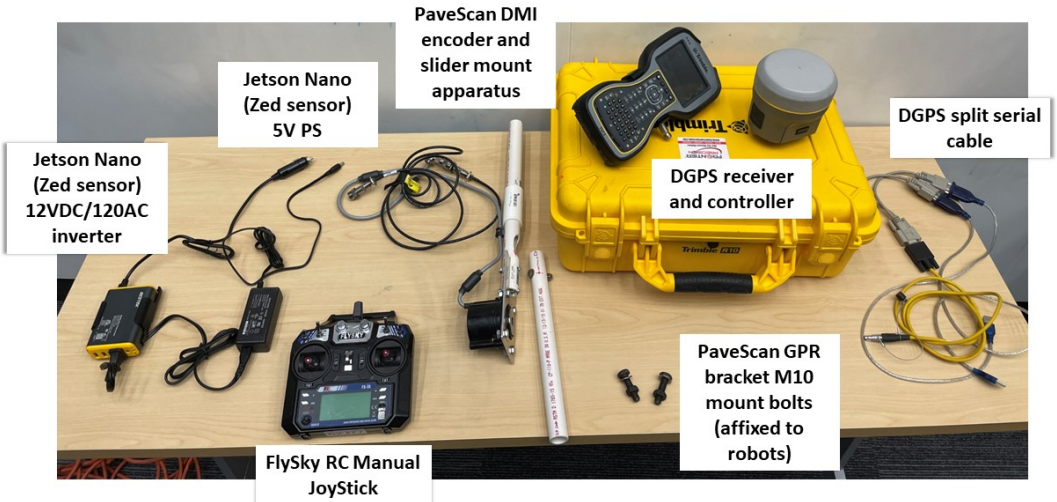


Figure A.1 Robot components (Sensor mast not shown)

1.3 TRANSPORTING THE ROBOT

Before transporting the robot, ensure the following:

- The servo motor & controller rotary switch is in the OFF position.
- The processor/sensor hardware battery red/black Anderson connector is UNPLUGGED.
- The processor/sensor hardware battery switch is in the OFF position.
- The Sensor Mast is removed.
- The cellular Wifi router power cable is UNPLUGGED from the robot external 12VDC port.
- Any Ethernet cables are UNPLUGGED from the robot external EtherNet ports.
- And, before transferring out to a site, ensure batteries are fully charged.

1.4 BATTERY CHARGING AND MAINTENANCE.

Both robots come with appropriate battery chargers for the 20/24 Volt servo motor batteries. For the 12VDC battery, a specialized LiFePO4 rapid charger is supplied for the second robot (“Isaac”). The first robot, “Azimov”, uses a high-performance AGM 12VDC battery can be charged with any marine and vehicle charger that has an AGM setting and is suitable for charging a 30AH battery (for example, Schumacher 2/15 Amp, Model #SC1359, or an IOTA 12v 15 Amp DLS-15/IQ4). Although a charger is not supplied for this robot, an Anderson plug-to-terminal clip assembly is supplied with the robot that should be adequate for interfacing with most chargers.

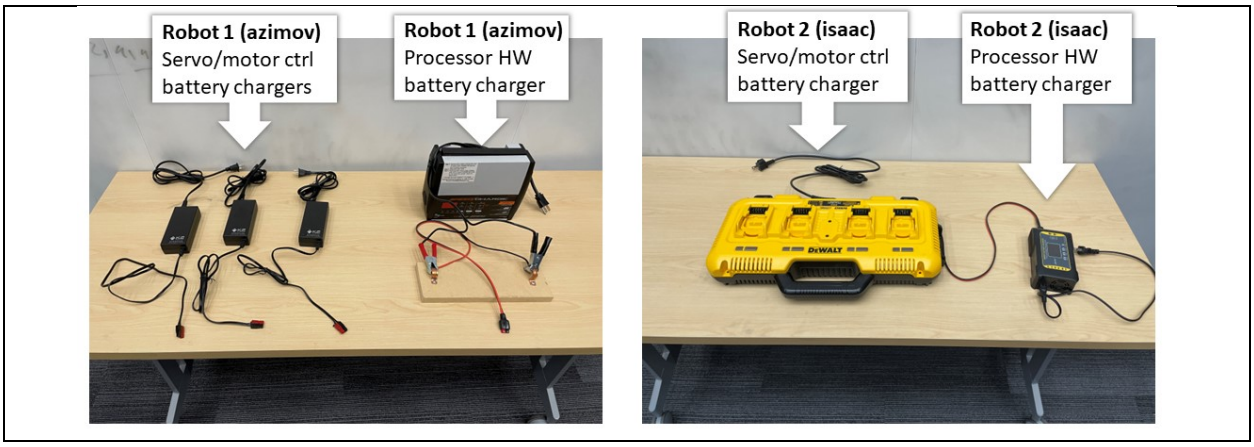


Figure A.2 Robot battery chargers

To maintain the batteries, occasionally re-charge them during robot non-use and storage. One advantage of LiFePO4 is they do not require the same level of maintenance as a lead-acid type battery. Both robots utilize such batteries for the servo/motor controller power; during non-use plugging them into their respective chargers about every four months to 'top off' (restore 100% charge) the batteries should be sufficient. The second robot, "Isaac", also utilizes a LiFePO4 battery for the 12VDC processor and sensors power supply, so the same storage maintenance also applies. The first robot, "Azimov", which instead uses an AGM 12VDC battery to power the hardware, requires more frequent topping off, perhaps every one to two months (alternatively, one could leave this battery plugged into a specialized 'smart charger' with trickle charging technology).

1.5 ASSEMBLING THE COMPONENTS ROBOT AND POWERING IT UP

The instructions for preparing assembling the robot at a job site is provided below. Note that cables can be further secured through wire-ties and Velcro straps as appropriate. It is assumed the operator has unloaded the robot in a safe area, affording enough space to assemble the robot, and manually operate it.

1.5.1 Set-up for manually controlled robot operation only.

As a minimum, to operate the robot manually while collecting GPR data, please follow the steps below. If autonomous capabilities will not be utilized, skip section A.2.3.2

1. Double check that the smaller black ON/OFF 12VDC power toggle switch is in the OFF position, and the processor/hardware battery is unplugged.

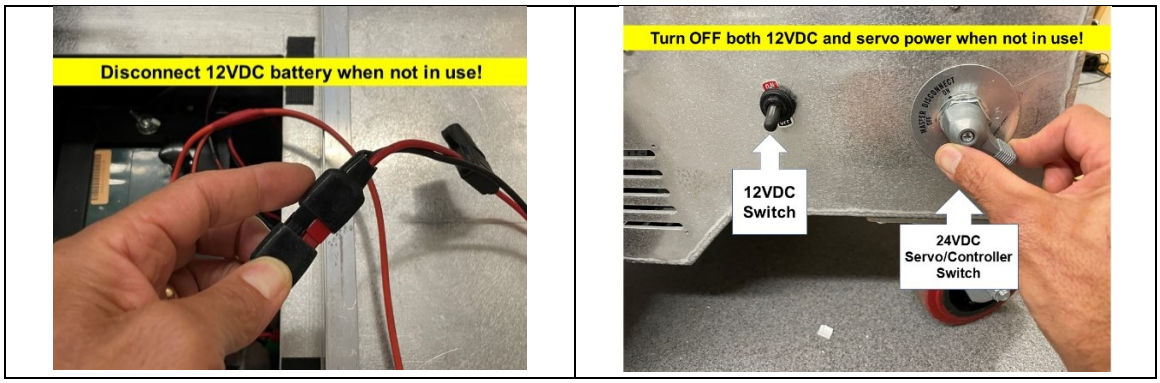


Figure A.3 LEFT: 12VDC battery Anderson connector; RIGHT: 12VDC and 24VDC Servo/controller power switches.

2. Slide in and lower mast through the square bracket frame on top of the robot, making sure to gently pull internal sensor cables out the bottom and through the two "U-slots" at the bottom of the mast, to keep them from being pinched or damaged.
3. Secure the mast by tightening the top and bottom black thumb screws. DO NOT OVER TIGHTEN!
4. Screw on the GPS receiver/antenna sensor to the top of the mast.
5. Attach the Trimble TSC-x controller to the mast PVC mount bar/carrying handle. *NOTE: the survey pole TSC mounting hardware must be provided separately*
6. Connect the Pepwave Cellular/Wifi router power cable to the 12VDC Pepwave 2-pin external port on the robot chassis.

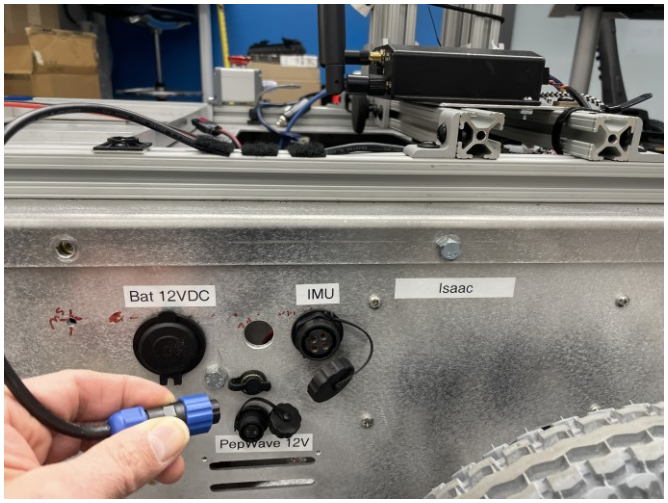


Figure A.4 Connecting the cell/wifi router power cable to the robot.

7. Screw in the Pepwave antennas. Note the narrow oblong profile antenna(s) are for the cellular reception/transmission, while the round “rubber-ducky” omni-antenna is for the Wifi AP. Refer to the labels next to the TNC sockets on the router.

1.5.2 Required set-up for robot autonomous control and on-board sensor acquisition

The following additional steps must be performed to prepare the robot for autonomous operation. Optional sensor preparations are indicated.

1. Plug in the LEMO connector serial data cable to the GPS receiver(R-10). *A DB9 splitter cable with USB serial dongles is attached to the DB-9 end of the LEMO cable. One of them will be connected to the robot in a later step in the GPS Configuration section, 1.7.1 .*
2. Connect the IMU sensor to the 4-pin external port on the robot chassis. **ENSURE THE CABLE LABEL NAMES MATCH THE CORRESPONDING ROBOT NAME!**

Figure A.5 Connecting an IMU cable to the robot.

3. Connect the Pepwave LAN Ethernet cable to one of the external Ethernet ports on the other side of the robot chassis.
4. *OPTIONAL*: for enabling collision detection:
    a. Plug in the LiDAR sensor cables to front USB ports labeled “LIDAR”
    b. Plug in the Nano processor (grey enclosure) Ethernet cable to remaining external Ethernet port.
    c. Plug in the power inverter box to the labeled external cigarette 12VDC socket located on the side of the robot chassis.

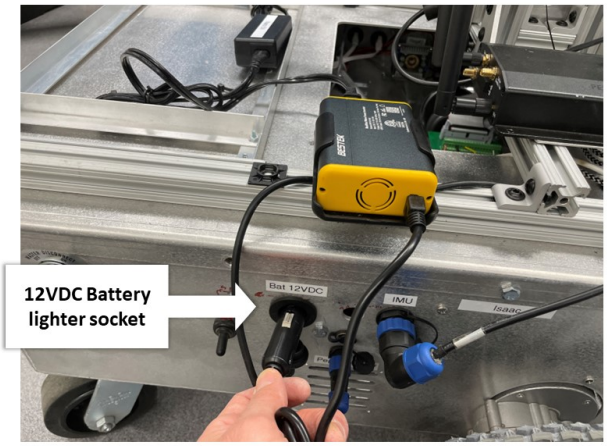


Figure A.6 Connecting Jetson Nano 12VDC/AC inverter into robot.

d. Plug in the labeled Nano power supply to the AC inverter and Nano.

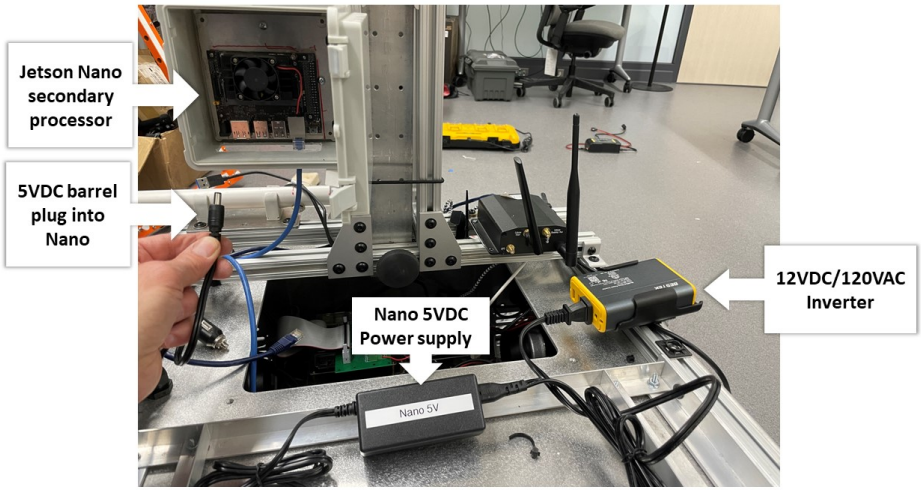


Figure A.7 connected PS components for Nano and ZED 3D/2D camera sensor.

5. *OPTIONAL*: for utilizing the IR camera sensor, plug in the IR Lepton camera sensor labeled USB cable to the external front USB port labeled IRCam (Lepton).

1.5.2.1 Prepare to power up the robot.

1. Power the servos and motor controller turning the large rotator switch to ON. The detent of this switch is heavy by design; some force will be required to turn it to “ON”, with less force to turn it to the “OFF” position.
2. Ensure the black switch is still in the off position. Re-connect the processor/sensor 12VDC Anderson red/back battery plugs.

1.5.3 Installing the PaveScan GPR payload

The following steps are for mounting the PaveScan GPR payload. These steps can be followed before, or after, powering on the servos. It may also be completed while waiting for the robot processing and sensor hardware to fully boot up, since this system is powered and configured independently from the robot.

1. Slide on the GPR sensor bar through the front mount bolts and tighten with supplied nuts and washers.
2. Install remaining GPR system; the controller lowers into brackets on top of the robot.
3. Slide the DMI encoder assembly over the left front wheel key-notch.

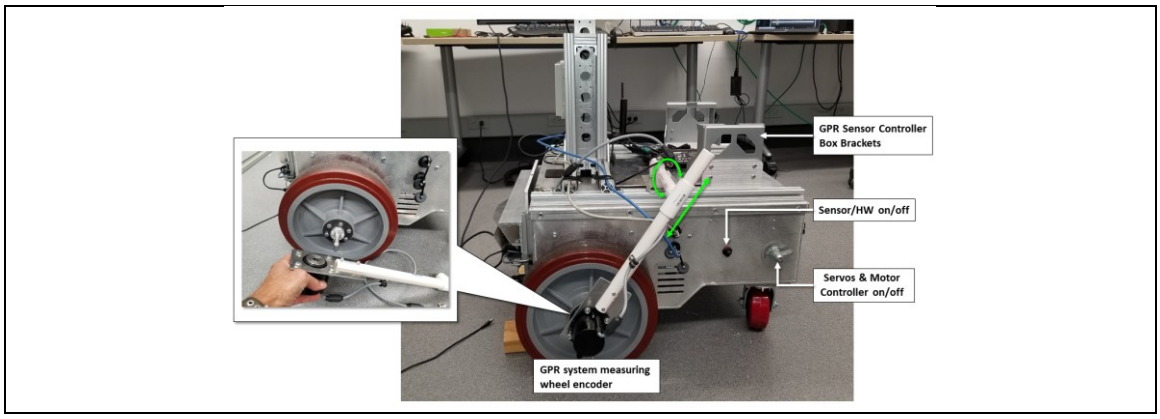


Figure A.8 PaveScan DMI encoder device robot mounting configuration

4. Slide over the T-coupling and attach it to the PVC tube, which snap-mounts into tube holders on the top front surface of the robot.
5. Plug-in the GPR cables and route appropriately (recommendation: route them underneath the horizontal, parallel sensor mast chassis mount bars to the controller.
6. The Panel currently rests ontop of cables and the mast bracket.
6. Plug in one of the USB serial dongle ends of the GPS LEMO cable to the tablet. Leave the other dongle end unplugged. *It will be connected to the robot in a later step in the GPS Configuration section, 1.7.1 .*

Figure A.9 Assembled robot; Operator performing GPR calibration.

1.6 MANUAL OPERATION OF THE ROBOT

At this point if necessary, manually steer the robot to a suitable location, with, or without the GPR payload. To manually connect and steer the robot, follow the steps below:

1. Turn on the FlySky RC controller and place all the switches in the "Up" position.
2. Simultaneously drop the Left/Right sticks to the "DOWN" (bottom) position.
3. An audible alarm presents when the RC link establishes a connection, with the LED screen indicating Rx/Tx signal characteristics. ***Immediately let go of both sticks!***
4. ***Let go of both sticks!***
5. The *right* stick controls forward/reverse (up/down), and steering (left/right).
6. Turn off the RC controller when finished moving the robot to save battery usage. After a short time of inactivity, it will output an audible warning signal sequence.

1.7 STARTING UP ROBOT PROCESSORS AND SENSORS

After all steps above, the robot sensor and hardware are ready to be energized. Turn the black switch on the side of the robot to the ON position. A Red power LED on the router should be lit, followed by other green flashing/solid ones as the router boots up. A few minutes are required for the router/cellular modem to boot up and then establish a connection to the cellular network. When all the LEDs on the front of the router are solid green, the robot is connected to both the Wifi/LAN and WAN.

During the time period the robot system hardware is initializing and booting up, this may be a good time to optionally mount the PaveScan GPR payload.

1.7.1 GPS Sensor configuration

After full cellular and LAN connectivity is apparent, turn on the DGPS receiver, then power on the TSC-x computer, and connect it to the VRS network via the router Wifi AP (password not supplied herein). For the current robot integration with the PaveSCAN, ensure the DGPS receiver is configured with the following settings:

- Serial baud rate is 115200 bps
- Update frequency is 5 Hz (0.2 sec).
- Serial data stream is set to GGA NMEA extended output.

Plug one of the DGPS serial cable USB ends into the labeled robot USB port located on the front of the robot.

If autonomous robotic operation is planned, or data collection from other on-board sensor will be conducted, a notebook computer will be required to interface with the robot. If a Windows 10 notebook has not been configured with the recommended tools, please refer to following section, 1.8 *Robot Communication and Interface Tools*. Otherwise, skip to section A.4.

1.8 ROBOT COMMUNICATION AND INTERFACE TOOLS

The robots utilize the ROS-1 (Noetic, Melodic) environment on each of the installed processors, running Linux Ubuntu 20.04/arm64 Focal Fossa. The steps discussed in this guide assume the operator will be using a Windows 10 notebook/tablet PC. For a Windows 10 device, install the following freely licensed tools (Windows 11 will probably work but it has not been tested):

- PuTTY (https://www.putty.org), PuTTY is a windows GUI for creating and opening Secure-Shell (ssh) terminal sessions to remote hosts.

- Multi PuTTY Manager (https://sourceforge.net/projects/multiputtymanager/); this is a front end GUI to manage and organize multiple PuTTY terminal session windows, and so on. Read the short manual on its usage (Figure A.10).

- WinSCP (https://winscp.net/eng/download.php). WinSCP is used for viewing, copying, updating, and synchronizing files and/or folders between remote hosts and the PC, using an intuitive Explorer-like, Drag-and-Drop UI through ssh connections.

- NoMachine (https://www.nomachine.com/download). NoMachine provides a Remote Desktop Session (RDS). The NoMachine RDS displays any graphical output provided by some of the robot software. Users may find it more intuitive to work with. The license free installation is sufficient for our purposes.

1.9 PUTTY WALKTHROUGH

Create and then start a PuTTY secure-shell (ssh) session (Figure A.11) :

1. Enter the ip address for a robot host processor (Figure A.10)
2. Under the Connection→Data panel, store the login username.

3. Rename the session if desired, in Saved Sessions.
4. Click the Save button.
5. Select the saved session name and click Load.
6. Click Open
7. Enter the user password. If it is a new host, it will prompt to accept fingerprint.
8. *All passwords are withheld from this document.*

Table A.1 Robot processor hosts

	Robot #1, Azimov **IP addr (host name)**	**Robot #2, Isaac** **IP addr (host name**	**Login username**
Primary host (ROS_MASTER)	192.168.50.11 (azimov-1)	192.168.50.110 (isaac-1)	odroid
Secondary host	192.168.50.10 (azimov-3)	192.168.50.111 (isaac-2)	odroid
Secondary host	192.168.50.12 (nano)	192.168.50.112 (nano)	astroid
Local WiFi **SSID=”PEPWAVE_D8C8”, or “PEPWAVE_5G_4069”**	*192.168.50.1 *LAN ip admin access	*192.168.50.1 *LAN ip admin access	admin

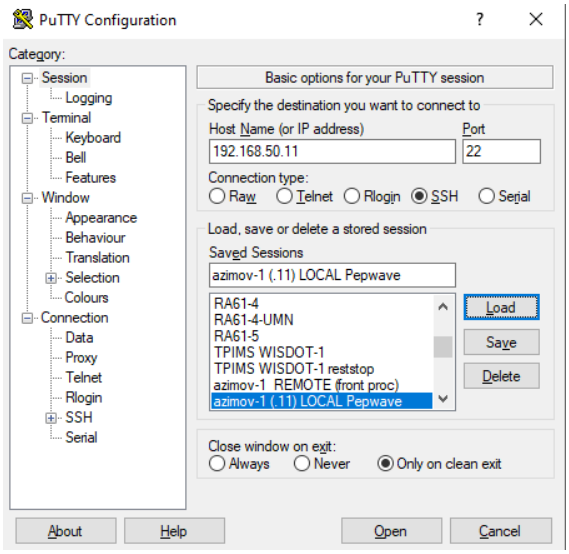


Figure A.10 Loading a saved PuTTY ssh session to a robot primary host processor.

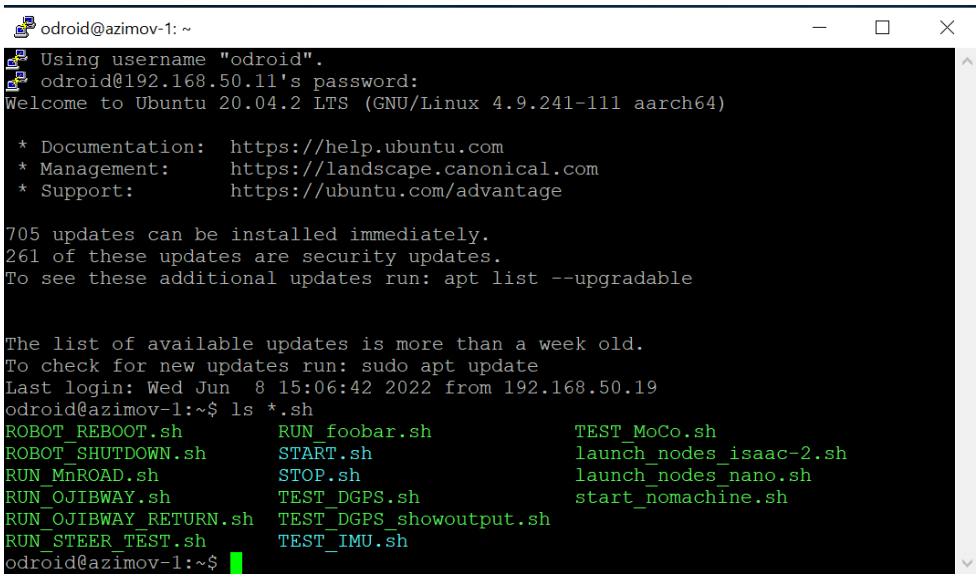


Figure A.11 PuTTY terminal window to remote ROS MASTER host processor.

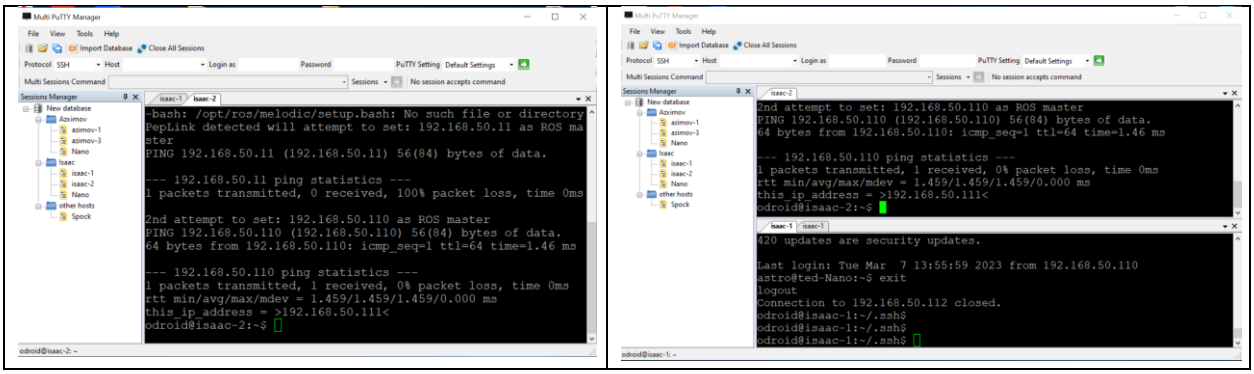


Figure A.12 Multi PuTTY Manager for referencing multiple ssh sessions.

1.10 NOMACHINE WALKTHROUGH

To configure and start and NoMachine RDS follow these steps:

1. From the main NoMachine Session window, create a session for a desired host.
2. *All passwords are withheld from this document.*
3. If not a new session, select one, and click Connect.
4. A session configuration window will be presented.
    a. Select desired layout functionality.
    b. Scaling the renders Desktop to fit completely into the NoMachine display window.

c. “Maintain original size…” will show a portion of the remote desktop and provide scrollbars, or pans the display according to the user mouse actions.

5. After selecting OK, the host login desktop is displayed.
6. Hit the carriage return, or enter user login password, if necessary.
7. Right click in the desktop to open a terminal window.
    a. “<CTL>-T” opens another terminal session tab.
    b. “<CTL>-+” (plus) increases the text and size of the terminal session.
    c. “<CTL>--” (minus) decreases the text and size of the terminal session.

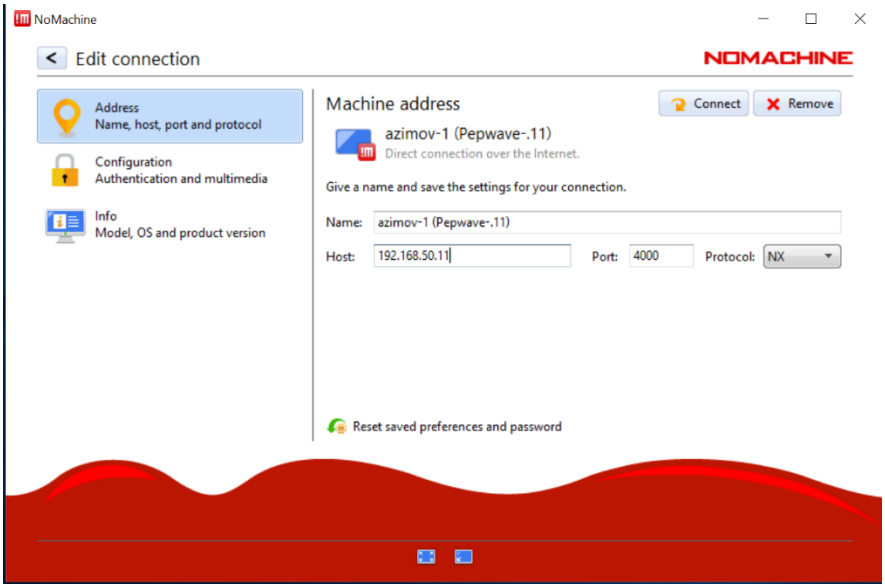


Figure A.13 PuT NoMachine initial session configuration

Figure A.14 NoMachine RDS renders login screen

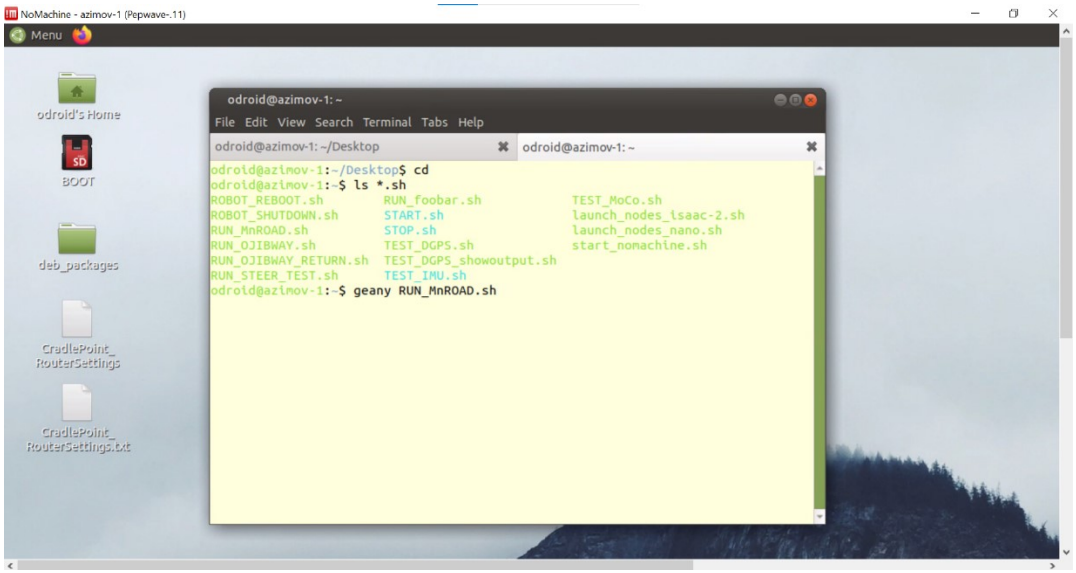


Figure A.15 The rendered host RDS Desktop. Experiment with using scaling, retaining original resolution and size—and so forth

1.11 FILE EDITING WALKTHROUGH

For editing files in a text terminal window, there are several terminal/text cursor based editors. A simple one, that is also 'language aware', is the program, **`nano`**. To edit a file simply enter the filename as the first argument following the command, for example (A.16):

```
nano ~/RUN_MnROAD.sh
```

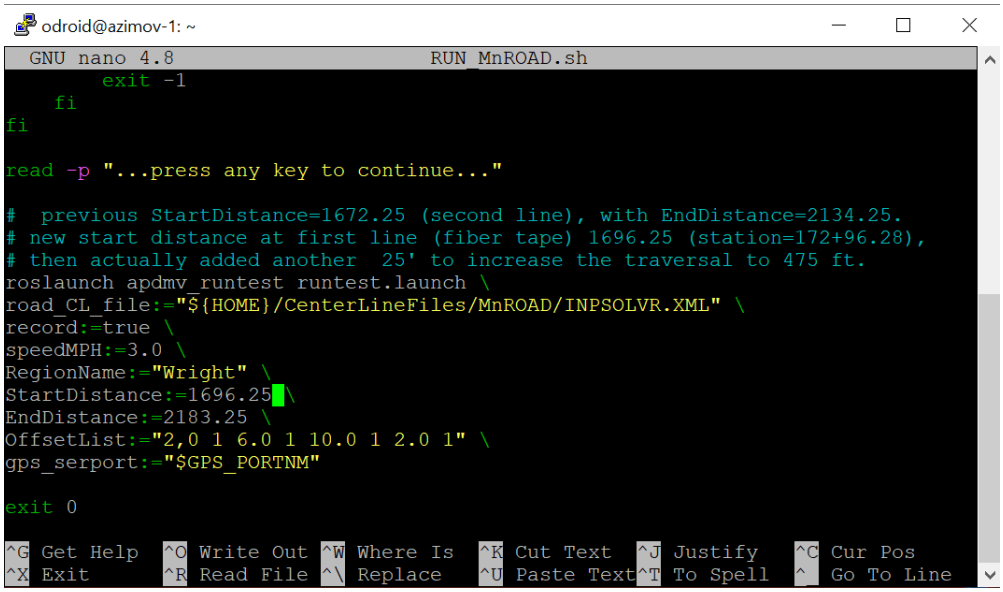


Figure A.16 Editing contents of RUN_MnROAD.sh script on terminal editor, nano

On the other hand, if instead the remote session to a robot host is through the NoMachine RDS, one can use instead more intuitive, GUI based editors. Once again, they are too copious to summarize. The editor, **geany**, is installed on the robot host processors, because it also uses most of the familiar keyboard shortcuts. It can be started via the desktop menu, or a command terminal, the command line (A.17).

```
geany ~/RUN_MnROAD.sh
```

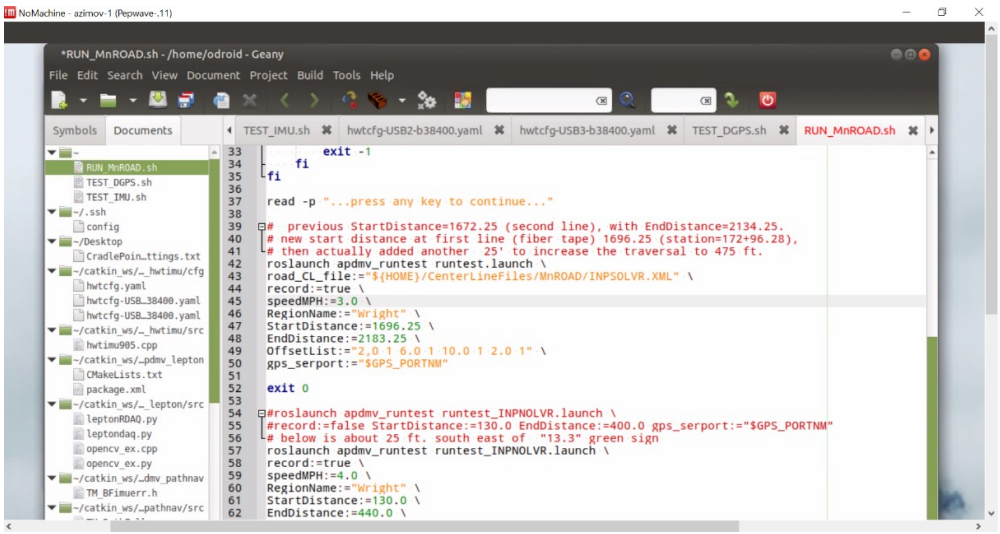


Figure A.17 Editing contents of RUN_MnROAD.sh with geany, within a NoMachine RDS.

Some familiarity with Linux is beneficial. A few very basic shell commands are summarized in Table B.1. and is by no means complete. Users are highly encouraged to practice and become very familiar with the aforementioned remote connection methods and tools. A brief description of a few relevant ROS commands is also in Appendix B.

1.12 CONFIGURATION AND RUNNING AUTONOMOUS ROBOT TRAVERSAL TASKS

1.12.1 Robot navigation sensor configuration and testing

1. Ensure that communication is working by opening either an ssh session, or NoMachine RDS to the robot primary host. If there is a failure to connect, refer to the trouble-shooting section of this appendix.

2. If using Putty, start a second remote section to the same host. Or alternatively, if using the NoMachine RDS, open a second x-terminal session (via tabbed, or second window).

3. **Test and configure the IMU sensor**. In one of the terminal windows, for example, if we wish to test the port, "/dev/ttyUSB1", enter:

   ```
   ~/TEST_IMU.sh ttyUSB1
   ```

4. Typically, this is the correct serial device name to configure for the first robot, "Azimov". For the second robot, "Isaac", the correct device name is "/dev/ttyS2". If the device name is correct, the output will print out like figure A.18.. Optionally, in a second terminal window, verify the data output from the broadcasted topic, by entering:

   ```
   rostopic echo apdmv/hwtimu_data
   ```

5. In general, if the sensor is not communicating, or functioning correctly, the data field values show "NaN" (Not a number), or all zeros.

6. To stop the test, enter <CTL>-c. This is how to stop any launched ROS nodes.

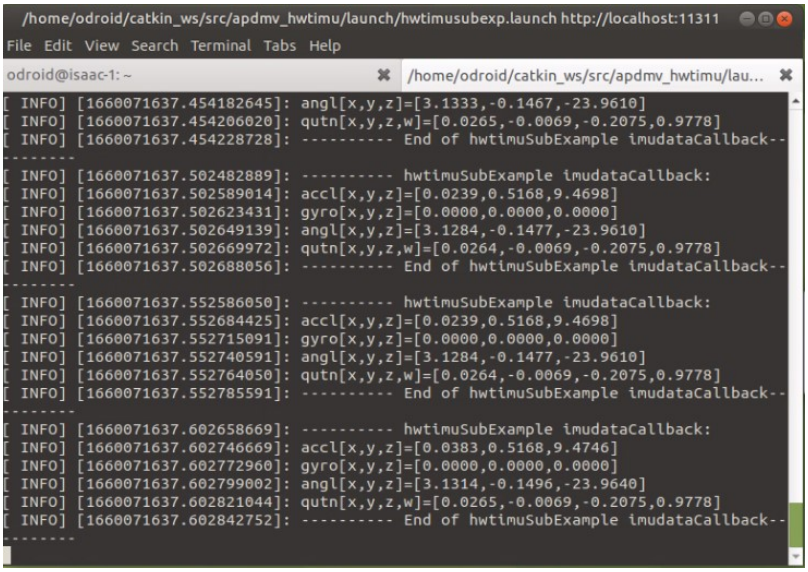


Figure A.18 IMU test data output

1. **Test the Trimble DGPS receiver processing**. Enter for the following commands, noting the specific robot:

```
odroid@azimov-1: ~/TEST_DGPS.sh ttyUSB2
odroid@isaac-1: ~/TEST_DGPS.sh ttyUSB1
```

2. This test starts many ROS packages. It may 'hang' the first time and repeatably print out the warning message, "nmea2sp: waiting for a region/county name…". If so, stop it (type <CTL>-c a couple times), and repeat the test. Otherwise, if it started successfully, the output would resemble figure A.19.

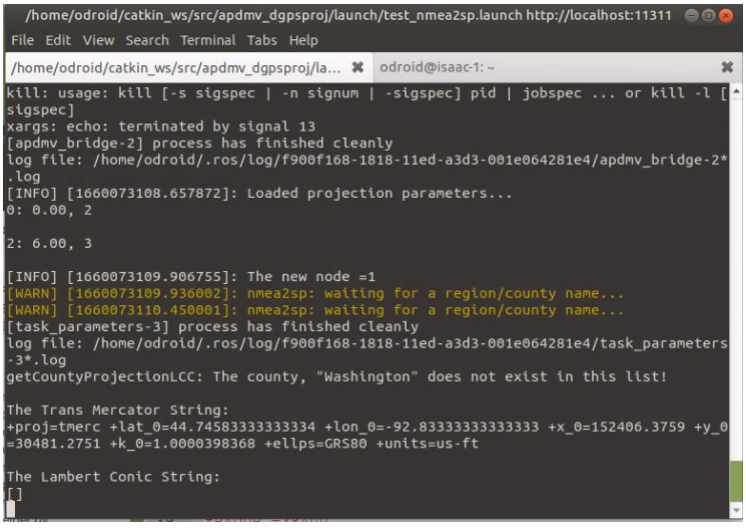


Figure A.19 Terminal screen ROS output after successful launch of the GPS Test software

3. Next from the second terminal window, verify the robot is receiving the NMEA string from the receiver. The output will resemble **Error! Reference source not found.** . Note the status value of 2, indicating a fix correction is being received from the VRS. If no data are being received, no /fix message data will be printed.

   `rostopic echo /fix`

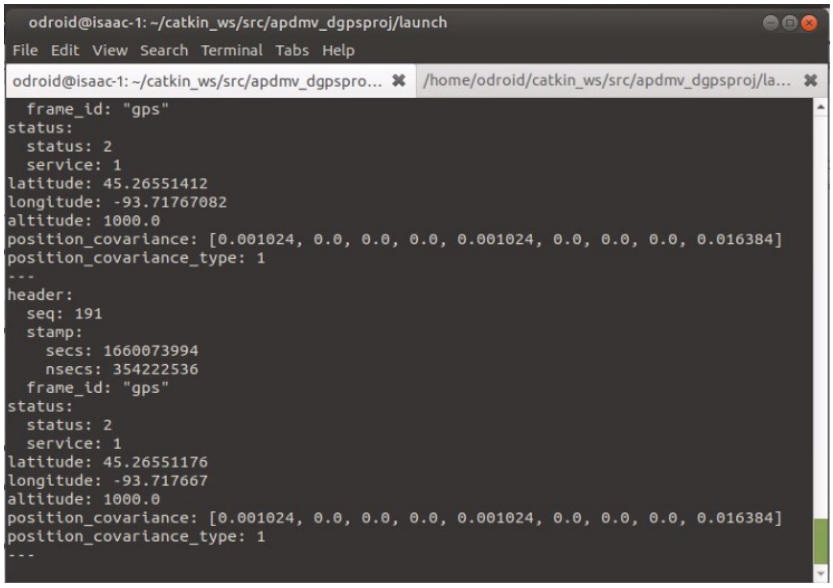


Figure A.20 DGPS RTK ROS fix topic output

4. Hit <CTL>-C in the second terminal window running the rostopic command in step 3. Next, we check to verify the projection transformation process is functioning properly. Enter the following in the second terminal window:

   `rostopic echo apdmv/dgpspnt`

5. The topic output will continuously display the converted Easting/Northing projections (Figure A.21). If the port device name is incorrect, no apdmv/dgpspnt message data will be printed to the terminal. Please note that since the test is to validate the data processing pipeline function, an arbitrary Minnesota county name is used. Thus, the Easting/Northing projection data may not represent the true county coordinates where the robot is located.

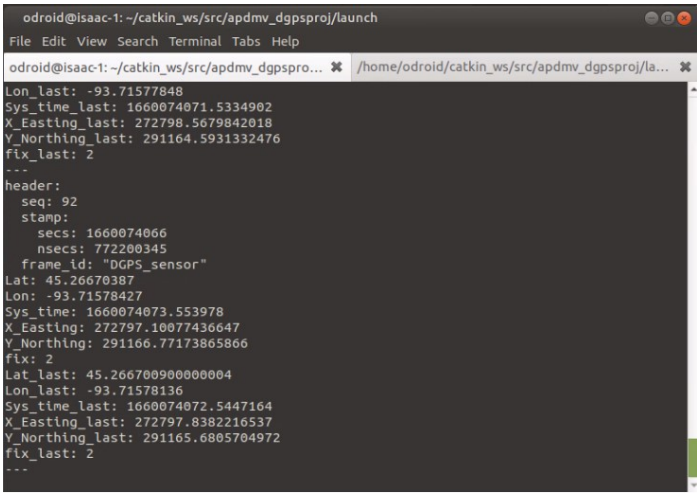


Figure A.21. MnDOT robot geospatial projected Northing/Easting topic data output

6. Enter <CTL>-C . in both windows to stop the test and any topic output.

1.12.2 Robot traversal task configuration

Once the verification of the sensors has been completed, the next step is to configure a traversal task (after a little practice, the outlined previously in 1.12.1 can be done quickly). The primary configuration objectives are to

- Upload a joint line data file, which can be formatted as:
    - a land XML file, which contains station equation information.
    - a comma-separated file (CSV) text file, containing a list of discretized XY points.
- Set up general task parameters.
- Finalize a starting location and lengthwise distance to run the traversals.

The steps to accomplish the approach are summarized.

1. Open a WinSCP session on the notebook PC, connecting to a robot primary host.
2. Navigate/select the directory location on the robot primary host processor to copy the LandXML joint line file. A new directory can also be created, if/as necessary. Please note the complete directory path name. For example, figure A.22 shows the remote full path directory name, "*/home/odroid/CenterLineFiles/MnROAD*", on the right pane.

3. Navigate to the location on the local notebook PC containing the desired LandXML file. Copy and paste it to the remote directory. For example, in figure A.22, the selected file, "INPTH94WB.XML" was cut and pasted from the local directory (left pane) into the remote directory (right pane), "*/home/odroid/CenterLineFiles/MnROAD*".

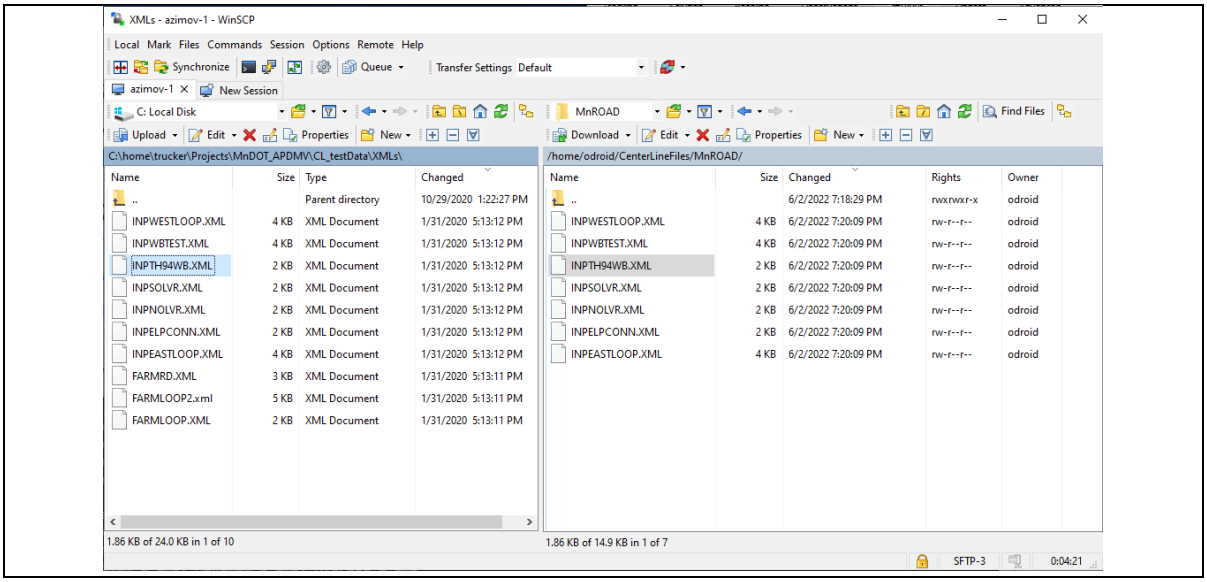


Figure A.22 Using WinSCP to copy an XML file on the PC to the robot primary host processor

4. Next, create robot task execution template files from one of the terminal windows, by entering:

   `$(rospack find apdmv_runtests)/scripts/create_run_scripts.sh`
   <optional full path filename of the copied XML file>

5. Two template scripts will be created using a prefix name entered by the user interactively, For example, suppose the entered prefix is "RUN_MnROAD", on the second robot, isaac. Then the generated script names created within the `$HOME` directory on the primary host are:

   `odroid@isaac-1:/home/odroid/RUN_MnROAD.sh`

   `odroid@isaac-1:/home/odroid/RUN_MnROAD-NOSTART.sh`

6. Edit the files (using nano, or geany). The scripts describe each of the variables that need to be defined. For example, if the full path name of the LandXML joint line file was not entered as an argument in step 4, JOINT LINE_FILE will need to be set to the correct full path.

7. Set the COUNTY name, in both files (currently limited to Minnesota counties.

8. Set SPEED_MPH. At this time, do not define a speed greater than 3 mph (Currently the firmware settings and software limit the maximum speed.).

9. Set START_DIST and END_DIST values, by running the <prefix>-NOSTART.sh. This starts the full navigation stack for the robot, but does not enable autonomous control. Take care to enter the correct host serial device name used by the DGPS receiver:

   `~/RUN_MnROAD-NOSTART.sh ttyUSB<n>` *where n={1,2}*

10. Copious output will be displayed in the terminal— very likely including warning messages indicating the robot is too far from the set starting location. Do not worry about such warning messages for this step(bolded orange/yellow).

11. Now, In the second terminal window, enter:

    `~/show_station_dist.sh`

12. The output will stream an estimated station location value, path location distance, and the estimate lateral offset from the joint line path data relative to the front center axle of the robot (Figure A.23). This axle location is 1.85 feet behind center of the mounted GPR sensors.

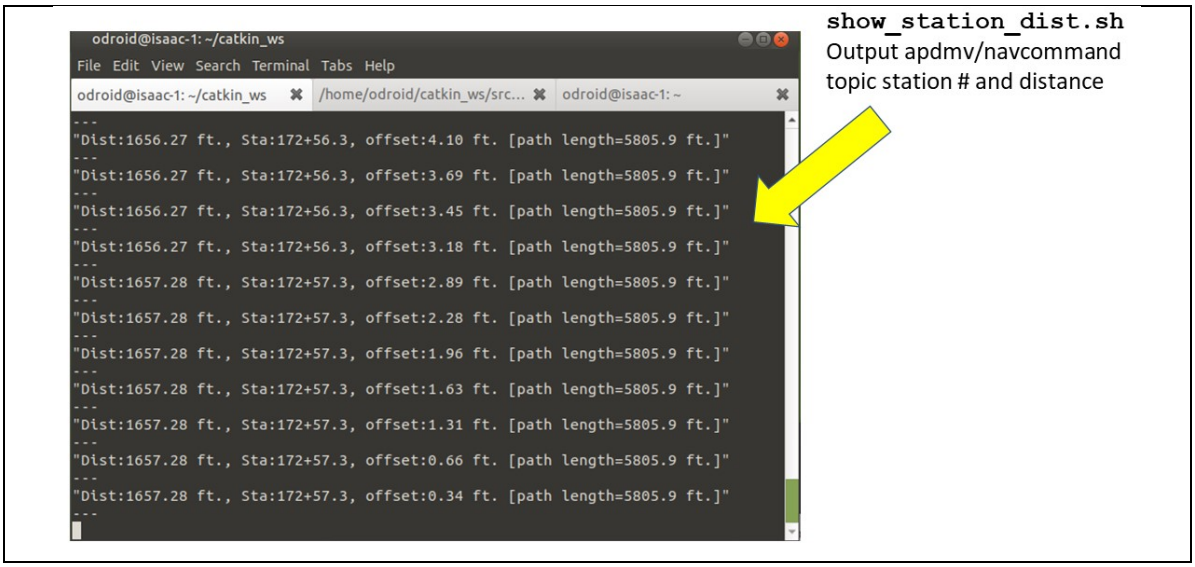


Figure A.23 Streaming navigation topic estimates of the joint line path location and the lateral offset of the of the robot front center axle.

13. Manually steer the robot to a desired location and observe the distance and offset data. Observe if the distance values are increasing or decreasing. If at the desired location, take note of the distance value estimate.

14. Stop the navigation software stack <CTL-c> in the terminal window where the script was run.

15. Modify the values of START_DIST.

16. Set the END_DIST by subtracting (decreasing distance) or adding (increasing distance) the traversal distance from the START_DIST.

17. Edit the OFFSET_LIST tuple variable, with “offset distance number-of-passes” pairs. The offset distance is in feet, with positive offset pointing to the right of the joint line path data, and a negative offset point to the left of the joint line pathdata, for increasing station locations (i.e. increasing distance values, figure A.24).

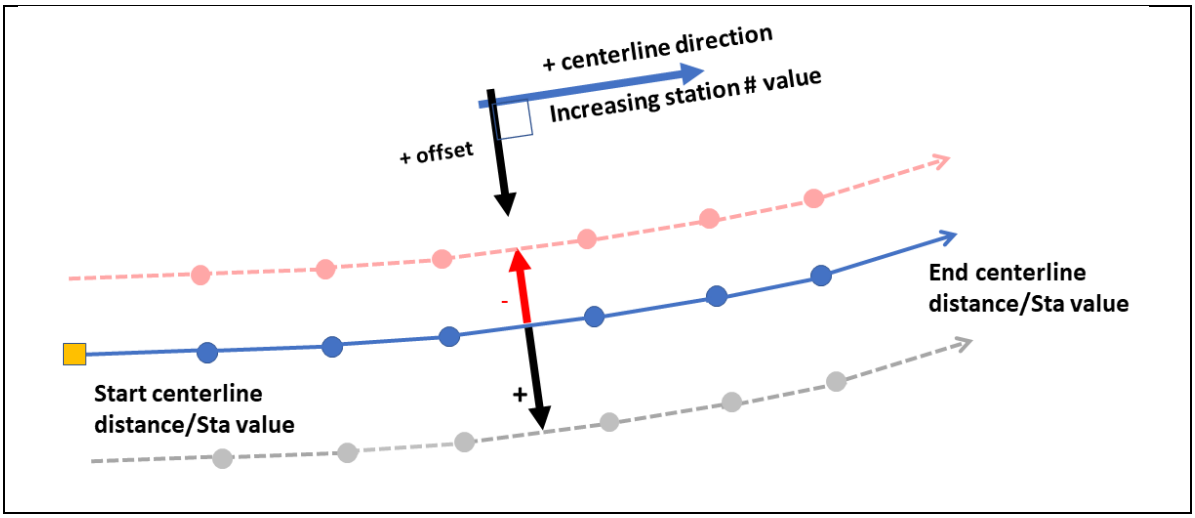


Figure A.24 Robot traversal offset ±sign convention from defined path centerline.

18. Maneuver the robot near the starting point. Then, with the example prefix name, “RUN_MnROAD”, enter the autostarting script name, with the appropriate GPS serial port name (hint: with ttyUSB1, which robot is being used?).

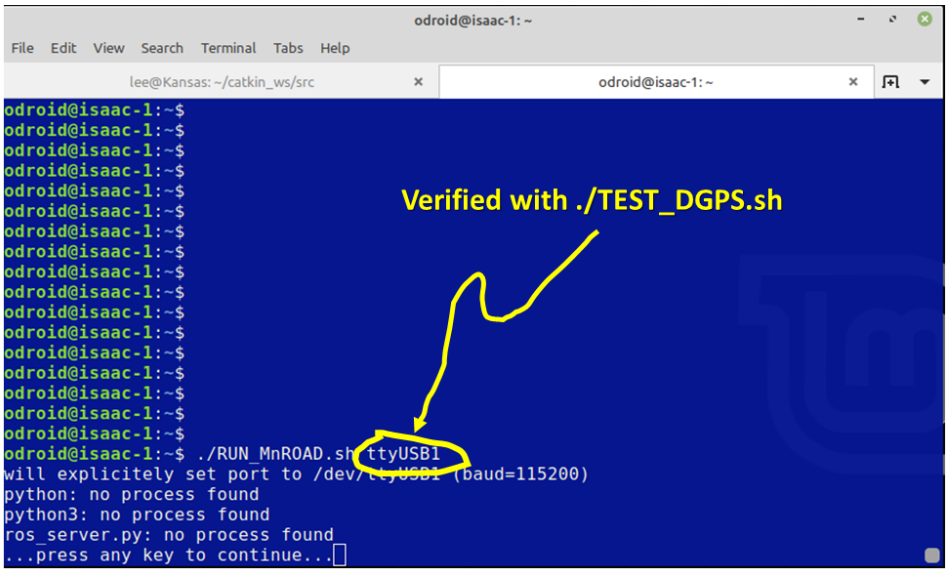


Figure A.25 Command line to start the autonomous robot GPR traversal job.

Technically speaking, if the robot is within 25 ft. of the starting distance location (with the desired offset), the robot will autonomously, and slowly steer the robot toward this location. When the robot is within 4 ft., the traversal mode will start (the robot will accelerate to the desired traversal speed, and will maintain it until the end location is reached).

1.12.3 Terminating or stopping autonomous navigation

There are a few ways to terminate the autonomous traversal:

- Simply enter <CTL>-c one or more times in the terminal window where the execution script was started in the last step above. This will completely terminate the execution.
- In a separate terminal window, enter: **`~/STOP.sh`**. This will temporarily suspend the traversal. It can be reengaged, by entering: **`~/START.sh`**
- The second robot, “Isaac”, has a mounted red panic button. Depressing it will immediately disarm the controller. The panic button will need to be manually released, and the controller hard-restarted to clear the fault condition.

1.12.4 Restarting an autonomous traversal

If, for any reason, the robot traversal terminated prematurely, the traversal task can be restarted at the last location before the traversal stopped. For example, if a battery required to be changed, and so on.

1. ***Step 0: if for any reason, the robot needs to be rebooted, ensure to unplug the GPS receiver from the USB port first! , Then repeat configuration and sensor tests!***
2. To restart the traversal task, enter the command (again using the example prefix name, "RUN_MnROAD"):

   ```
   $(rospack find apdmv_runtests)/scripts/restart_run.sh
   RUN_MnROAD.sh ttyUSB1
   ```

3. Note that if RECORD_NAV_AND_SENSORS was set to "true", a new ROS bag file will be recorded.

1.12.5 Recording and extracting navigation and traversal sensor data

Navigation ROS topic data and associated GPS and imu ROS topic data are recorded to a bag file during a traversal within the primary host processor, under /tmp/nav_pathdata-<date-timestamp>. bag. They are deleted if the processor is rebooted or powered off. To instead save the files, move them to the user home directory; enter from the terminal (assuming the directory, "bagdata" exists under the home directory:

```
mv /tmp/nav_pathdata*.bag ~/bagdata
```

They can also be backed up to the operator notebook PC, using WinSCP.

To examine, or 'play back' ROS bagfiles there are several tools under ROS. However, if one wishes to examine and analyze the data outside of ROS, we suggest using the *rosbag_to_csv* ROS package, which is installed on the host processors of either robots. To use this tool use the following steps:

1. Start a NoMachine session to a primary host processor.
2. Open two terminal windows within the remote desktop.
3. In one terminal window start roscore:

   ```
   roscore
   ```

4. In a second window enter the command:

   ```
   rosrun rosbag_to_csv rosbag_to_csv.py
   ```

5. Tick the topics checkboxes to convert into csv files, and then click the convert button. The converted *.csv files for each topic will each be saved in the same directory where the bag file is located.
6. Use WinSCP to search and download the converted *.csv files to the notebook PC.

1.13 ROBOT AUTONOMOUS TRAVERSAL BEHAVIOR MODIFICATION

The autonomous traversal behavior of the robot operating autonomous mode can be conceptualized with the causal graph show in figure A.26. Adjusting any parameters to the upper (top) levels will affect the levels below them. For example, at the topmost level, adjusting the servo max-out RPM and the servo controller closed loop speed control sensitivity will then require adjusting parameters for the software speed and steering controllers, which will then influence specific traversal control behaviors in the bottom level.

This documentation herein will describe the procedures to adjust motion behavior features of the robot ROS software framework. The ROS package names and yaml or xml launch files, associated within the middle and bottom levels indicated in figure 0.26, contain the ROS parameters can be adjusted. The adjustment of the mid-level, steering control behavior is described first, followed by the bottom level, navigation behavior adjustment The firmware adjustments were initially done according to the Roboteq 24XX motor controller manual and the RoboRun+ software guide (www.roboteq.com) and will not be addressed in this guide.

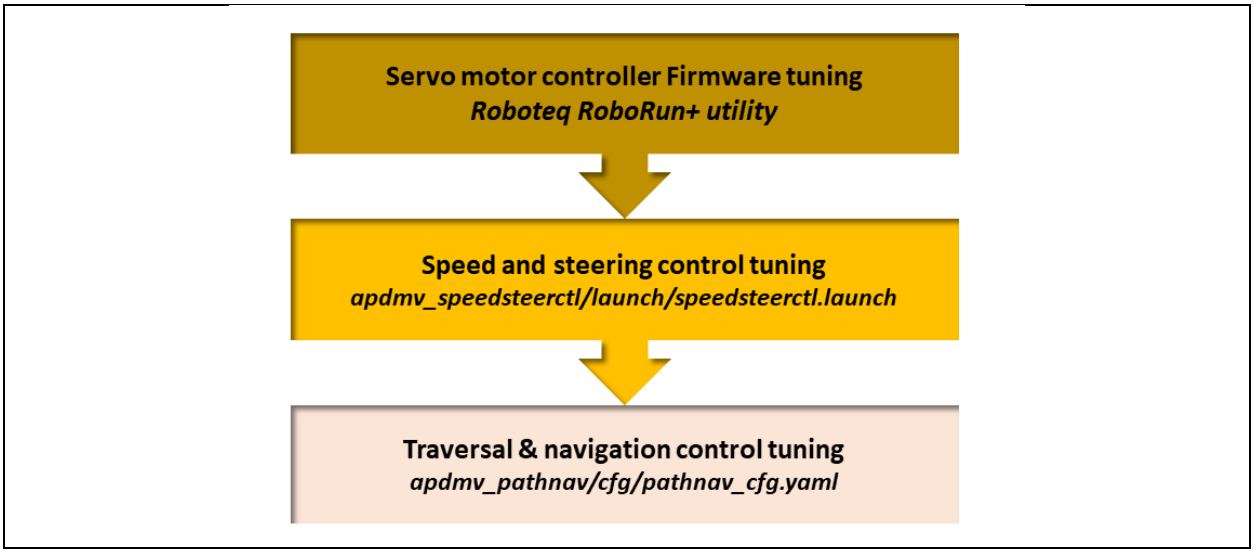


Figure A.26 Effective hierarchy for adjusting robot traversal behaviors; the lowest level is affected by adjustments made by the levels above it, and so on.

1.13.1 Speed and Steering Control

The steering and speed PID control can be adjusted by editing ROS parameters within the speed and steering control package (e.g, *apdmv_speedsteerctl*). The steps below

1. At the ssh remote terminal (or a NoMachine Desktop terminal window) prompt, first enter:

```
roscd apdmv_speedsteerctl/launch
```

2. Make a backup copy of the `speedsteerctl.launch` file! Note that the file name equivalent to a repository copy that contain the default settings for either Isaac or Azimov robots. Within the terminal window, enter the following to create a backup copy

   `cp speedsteerctl.launch speedsteerctl-bckup.launch`

3. Create a new launch file, that will contain edited changes, for example:

   `mv speedsteerctl.launch speedsteerctl-update1.launch`

4. Edit the file. Referring the example file, if using a terminal window (PuTTY, x-term, etc.):

   `nano speedsteerctl-update1.launch`

   Or, if otherwise if connected to the robot with a NoMachine RDS, go to the menu (or right click for pop-up window in desktop), open a new terminal window, and then enter the following in the new terminal window to instead use a GUI editor:

   `geany speedsteerctl-update1.launch &`

5. Save changes and exit. For **`nano`**, enter Ctl-w, Ctl-x.

6. Then create a *linked name* to reference the file by entering:

   `ln -s ./speedsteerctl-update1.launch  ./speedsteerctl.launch`

7. To ensure the link is correct, enter within the launch directory:

   `ls -l`

   The file should indicate that it is pointing correctly (typically cyan/blue font) to the reference file, speedsteerctl-update1.launch**,** for example, as shown in figure A.27**.**

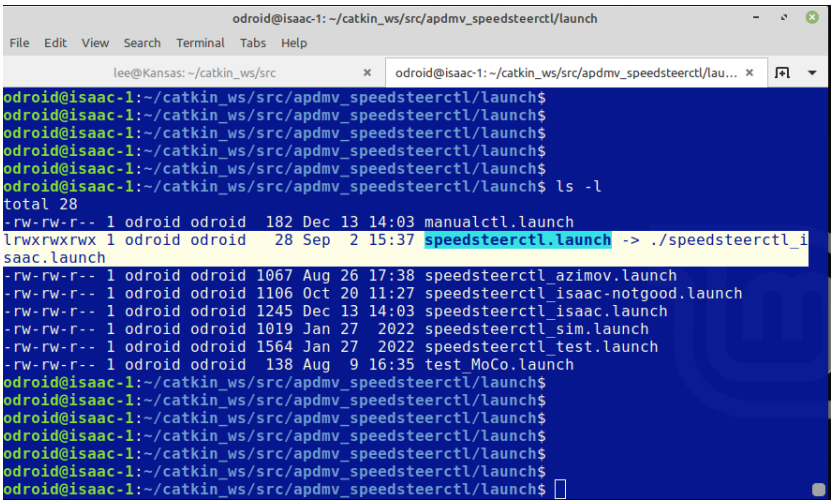


Figure A.27 Speed and steering controller launch file configuration, to refer to the required launch file name, *speedsteerctl.launch*

Table A.2 summarizes the definition of each ROS parameter within the launch file that can be edited, and, in very generalized terms, their potential effect on the steering and speed behaviors.

Table A.2 ROS params for steering and speed control

ROS Parameter name	Action-definition	General effect
Psteer_gain	Proportional gain factor of error between desired and current heading estimate.	Larger values stiffen response but can result in instability/'ringing'. May become sensitive to 'noisey' estimates.
Dsteer_gain	gain factor of the error rate between desired and current heading estimate.	Increasing values dampen response; may 'overdamp' if too large. Large lags may increase instability (response lags).
Isteer_gain	Gain factor of the cumulative error over time between desired and current heading estimate.	Removes small drift error but adds instability to control (output value can rapidly 'overtake' Psteer, Dsteer gain output values)..
Isteer_limit	Max value (absolute) to Integral gain factor value.	Can prevent Integral gain control instability.
DeadBand	± range PID steering output value will be set to the value.	Increasing value will increase sensitivity and response. Decreasing may improve response to drifting.
Pspeed_gain	Proportional gain factor of error between desired and current speed estimate.	Principle same as explained for steering.
Dspeed_gain	Gain factor of the error rate between desired and current speed estimate	Principle same as explained for steering.
Ispeed_gain	Gain factor of the cumulative error over time between desired and current estimated speed.	Principle same as explained for steering.
Channel_Mode	Mixed vs. Single affect how digital/analog commands route to motor axes.	Dependent on current firmware setting of motor controller. **Do not change.**
Dir_Polarity	Polarity of motors for 'forward' robot motion.	Dependent on current firmware setting and servo motor characteristics. **Do not change**.

1.13.2 Navigation and Traversal Control

A yaml code file, which is referenced by the navigation package launch files, is edited instead of the launch file itself. There are reasonable defaults for all the parameters, and thus none of them are required to be externally defined for autonomous traversal. Nevertheless, the steps to edit the parameters and their definitions are presented below.

1. **Navigate into the proper package directory**; at the terminal prompt enter:

   ```
   roscd apdmv_pathnav/cfg
   ```

2. **Copy of the `pathnav_cfg.yaml` file:**

   ```
   cp ./path_nav_template.yaml ./path_nav_test1.yaml
   ```

3. **Edit the copy as appropriate. Per above use the terminal editor**, **`nano`,**

   ```
   nano ./path_nav_test1.yaml
   ```

   Or, if the remote host processor connection is instead through the NoMachine RDS, edit the file with the GUI editor, **`geany`**:

   ```
   geany ./path_nav_test1.yaml &
   ```

4. **Save changes and exit.** For the **`nano`** terminal editor, enter Ctl-w, Ctl-x

5. **Edit the run task scripts generated from running the navigation package, `apdmv_pathnav`, `apdmv_pathnav/scripts/create_run_scripts.sh`** to set the variable set the NAV_PARAMS_FILE to the edited file, created in **step 2**.

Table A.3 ROS params navigation traversal behavior

summarizes the definition of each ROS parameter within the yaml file and, in general terms, their effect on the robot traversal and steering response behaviors.

Table A.3 ROS params navigation traversal behavior

ROS Parameter name	Action-definition	General effect
TURNAROUND_min_speed	Required speed (mph before the robot will turnaround, for multi-pass operation. The default is 1 mph.	Faster minimum speeds will reduce any overshoot of the end point distance but may increase risk of marring new pavement**.**
TURNAROUND_ARM_samplewin	The size of the sample window (# of samples), to estimate the turnaround speed. Default is 5 samples. (e.g. 1 sec. at 5 hz).	Larger values will increase accuracy and the elapsed time before turnaround occurs.
Heading_diff_limit	The difference between robot heading and desired heading, in degrees, where, if exceeded, the robot traversal speed is set to 0 MPH. Default is 25 deg.	This is a safety feature. Increasing the angle will likely allow larger deviations of robot from path, and the risk of permitting unstable traversal behaviors.
Path_speed	Desired traversal target speed. Default is 3 MPH.	Currently robots are limited to this maximum target speed. Future versions may relax this constraint. Slower speeds may be required if the robot is traversing large grade changes > 5%.
Headway_Time	The time the robot uses to calculate a distance headway to approximate the pursuit point. The distance is the (desired speed) * **Headway_Time**.	Default is 2.5 sec. Increasing value will reduce steering response but may create oversteering conditions. Reducing the value can lead to oversteering.
reset_BayesFilter	Resetting filter will remove prior knowledge of yaw state correction error estimate after turn-around.	Default is true.
negate_imuerror_est	If true, assumes current yaw state error correction is preserved after turn-around in opposite direction.	Default is true
SteadyState_YawRateThresh	The maximum acceptable yaw rate, in rad./sec, of the robot to consider a robot yaw estimate valid	Default is 0.0354 rad/sec. (20 deg./sec.) At the current robot speeds, most highway curvatures

		will produce yaw rates less than this value.
SteadyState_PorportionGoodSamplesThresh	The proportion of valid yaw estimate samples within a time window required to update yaw error correction estimates (0 to 1.0).	Default is 0.,8. Higher values will reduce uncertainties but may decrease correction update rate.
TimeWindow_YawDiffCapture	The time window, in seconds, applied for collecting valid GPS-based yaw estimates.	Default is 2 seconds. Larger times may improve error correction estimates while decreasing correction update rates.

1.14 VISUALIZATION AND RECORDING ROBOT VISION AND PERCEPTION SENSORS

1.14.1 Infrared Camera

To use the infrared camera sensor, first ensure that the labeled FLIR camera cable into the front USB FLIR port located on the front of the robot. It is advisable to let it warm up for 5-10 minutes. The camera is configured to automatically recalibrate the 'zero-emittance' offset level of the sensor as a function of change in temperature of the sensor, to reduce imaging noise. During the recalibration, the image will freeze for about one second.

1. Start a NoMachine RDS into either azimov-3 or isaac-2 secondary processor driver,

2. Open at least three x-term sessions. CTL>-<SHIFT>-t twice in an x-term window to open two more x-term sessions in separate tabs.

3. In one of them start a remote ssh session back to the primary (ROS_MASTER) host processor (shown below for either robot #1, **azimov**", or robot #2,:**isaac**"):

   ```
   odroid@isaac-2:ssh isaac-1
   odroid@azimov-3:ssh azimov-1
   ```

4. Then, after the connection to the primary host is established (no password or username is required) start roscore.

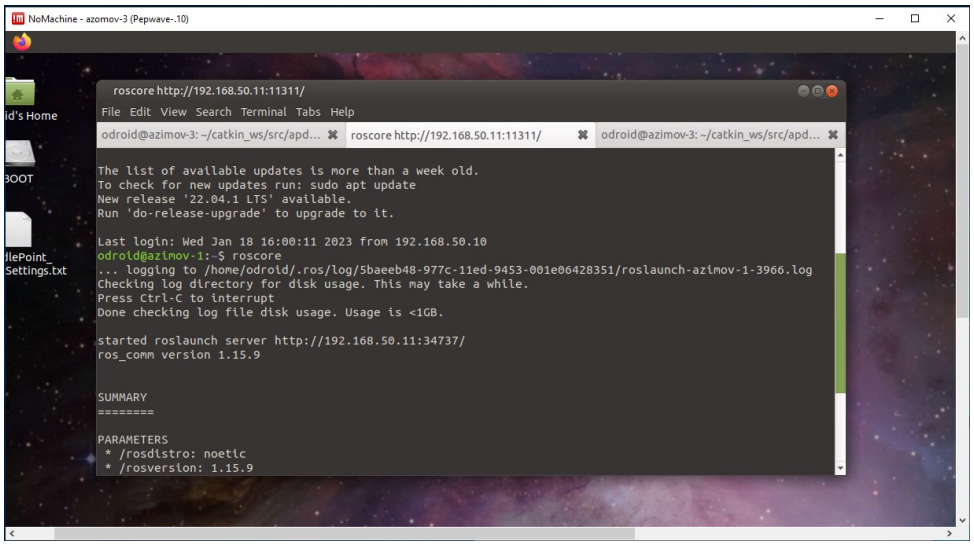


Figure A.28 NoMachine RDS into the secondary host.processor. ssh session into the ROS primary host (ROS_MASTER) to start roscore.

5. Next, from a second terminal window, start the FLIR Lepton ROS driver:

   `roslaunch apdmv_lepton leptonrad.launch`

6. Launch the image examiner tool within a third terminal window.

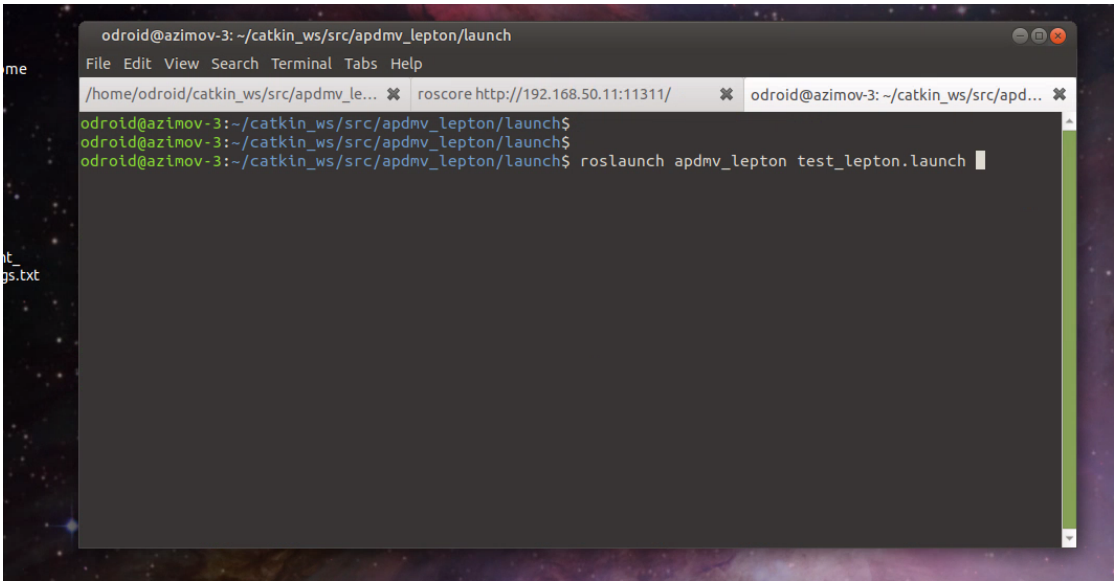


Figure A.29 Starting the FLIR lepton camera ROS driver.

Two graphic image windows will display the radiometric temperature image (and maximum temp detected), and a binary thresholded image.

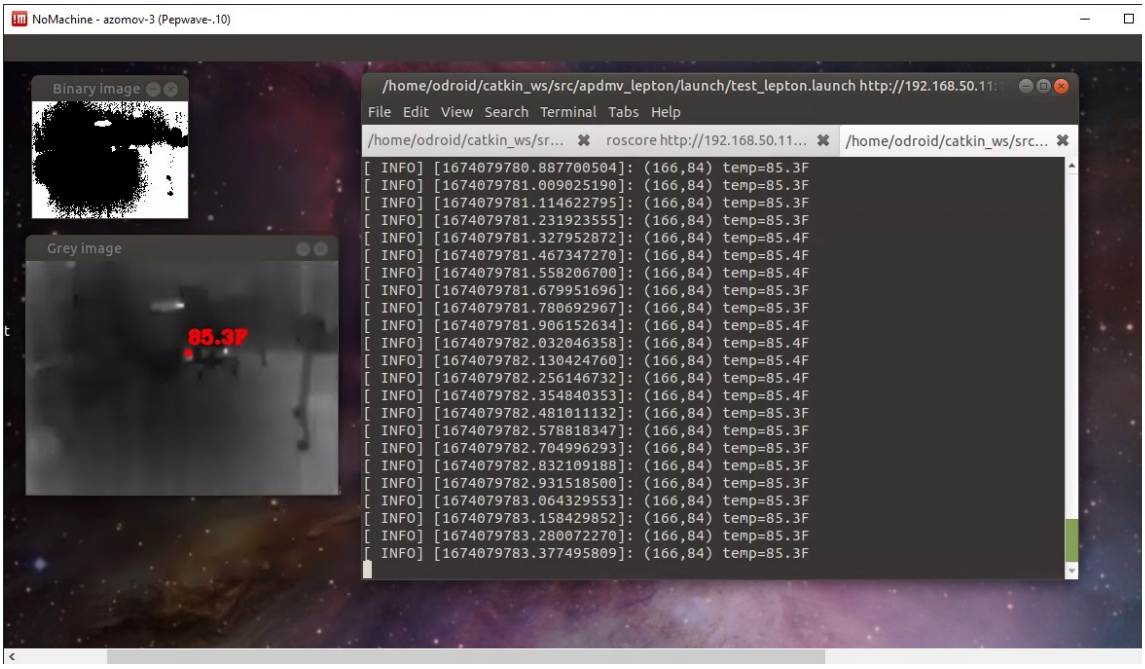


Figure A.30 FLIR Lepton camera output.

7. It is also possible to record the image data to disk (as a ROS bagfile). In yet another terminal window, enter:

   ```
   $(rospack find apdmv_recdata)/scripts/record_lepton.sh
   ```

8. To terminate the image examiner tool, enter <CTL>-c in the terminal window used to launch the node.
9. To terminate recording, similarly enter <CTL>-c in the terminal window where the ROS recording process was started.

1.14.2 ZED camera sensor

The zed camera interfaces with the Jetson Nano secondary processor. First ensure the processor is powered on. The processor uses the labeled 5V power bricked plugged into the 12VDC/120AC mini-

inverter, which plugged into the 12VDC Lighter socket on the side of the robot (Figure A.6). The Zed labeled USB cable is plugged into one of the USB ports of the processor.

1. Start a remote ssh session to a primary host processor, and in a terminal window enter:

   `roscore`.

2. Start a NoMachine RDS into the nano processor. (another secondary ROS host processor). The login name=ted. The password is provided separately to the MnDOT team.

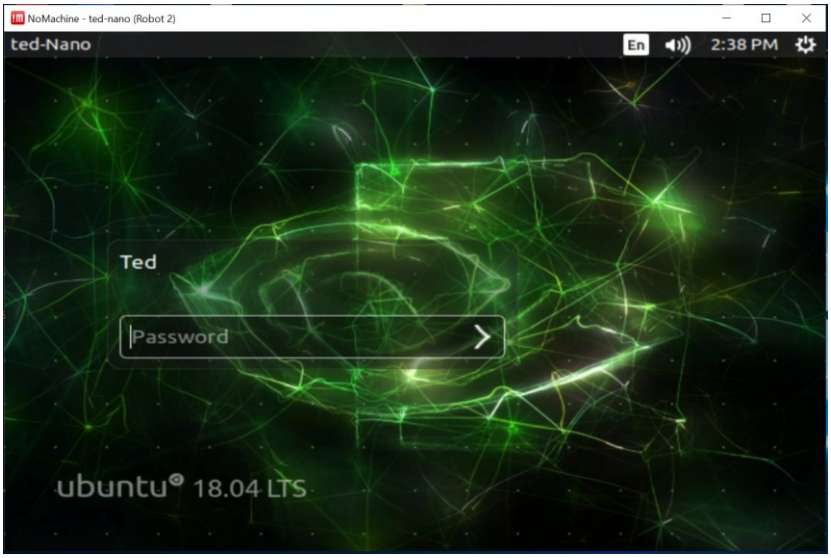


Figure A.31 NoMachine RDS into the Nano host.

3. The remote desktop is unfortunately a much smaller in resolution than the other host processor Remote Desktops. Within the desktop, right click to get the pop-up window and select an x-terminal to open, open a couple more, or as before use <CTL>-<SHIFT>-t to three x-terminal sessions.

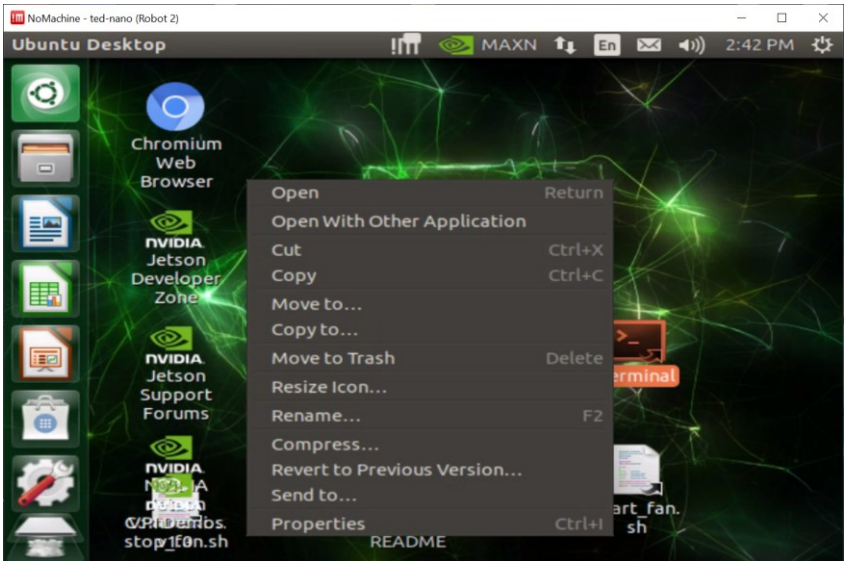


Figure A.32 RDS Opening an x-term on the nano

4. In one of the terminals, start the zed driver:

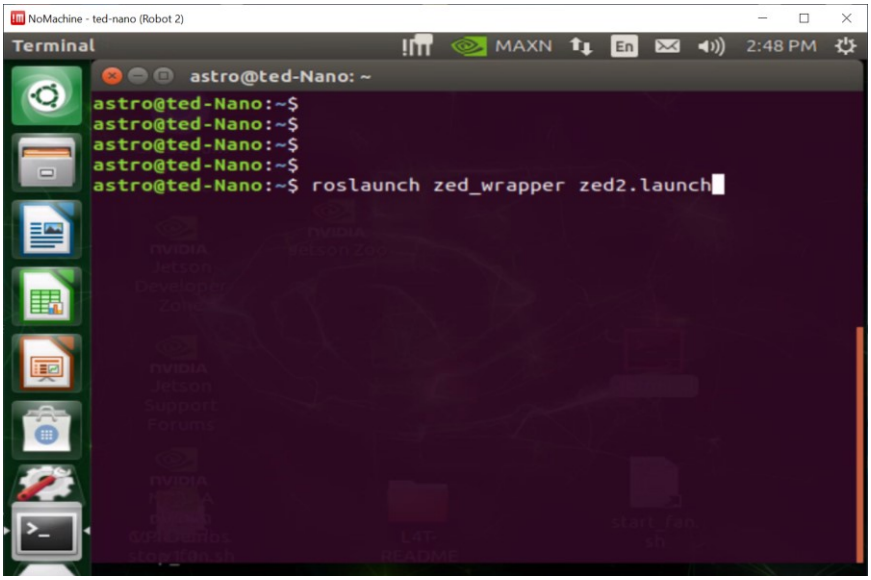


Figure A.33 Starting the ROS ZED driver. Note that the ZED2 camera is used on the robot.

5. In another terminal, run rviz with predefined settings to display Zed 3d and 2d camera data:

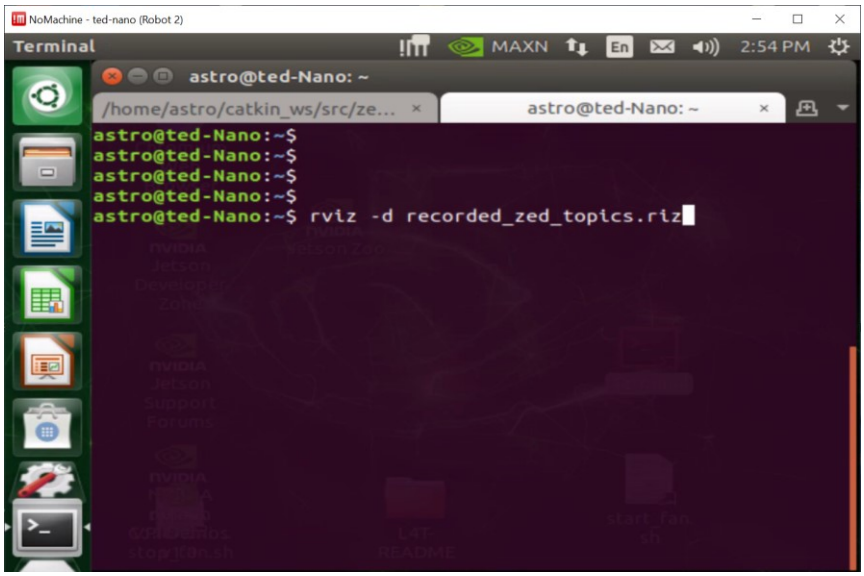


Figure A.34 Starting the ROS ZED driver. Note that the ZED2 camera is used on the robot.

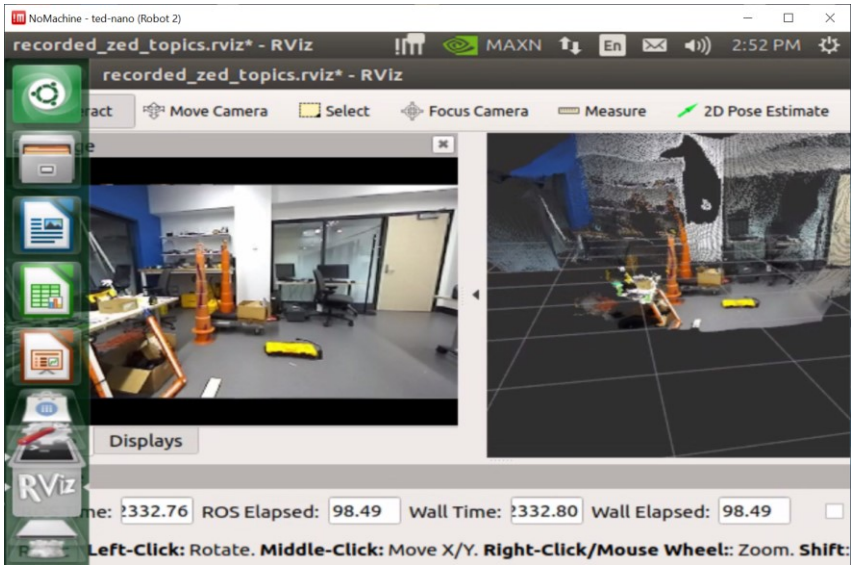


Figure A.35 Raw zed point cloud and image topic data used for collision detection.

1.14.3 LiDAR 2D Sensor

To use the LiDAR sensor, ensure that the two USB cables are plugged into the labeled LiDAR USB ports located on the front of the robot. The order of which port is used does not matter. Immediately after both cables are plugged in, the LiDAR will briefly spin and then stop.

1. Start a NoMachine RDS into secondary processor (i.e., either azimov-3 or isaac-2).

2. Open at least three x-terminal (x-term) sessions.

3. In one of them invoke a remote ssh session back to the primary (ROS_MASTER) host processor (shown below if using either robot #1 ("**azimov**") or robot #2 ("**isaac**"):

   ```
   odroid@isaac-2:ssh isaac-1
   odroid@azimov-3:ssh azimov-1
   ```

4. Then, after the connection to the primary host is established (the processor hosts are configured to require no password or username when connecting between each other) enter:

   roscore.

5. Next, from a second local x-term window (on the secondary processor), enter:

   roslaunch ydlidar_ros lidar_view.launch

   Or, alternatively, enter:

   roslaunch ydlidar_ros display.launch

6. The scan topic data should then be displayed in rviz.

7. It is also possible to record the scan data to disk In another terminal window, enter:

   roslaunch apdmv_recdata recdata.launch record_lidardata:="true"

8. To terminate any of the node processes, again enter <CTL>-c in the terminal window used to launch the node.

9. To terminate recording, enter <CTL>-c in the terminal window where recdata.launch was started.

APPENDIX B:
OBJECT COLLISION DETECTION

Either collision detection module (for LiDAR or ZED sensors) executes on a separate processor from the main navigation and control modules--which execute on the ROS master processor, isaac-1, or azimov-1. For the LiDAR sensor, the host is either isaac-2 (Isaac robot) or azimov-3 (Azimov robot). For the ZED sensor, the ROS nodes are executed on the Jetson Nano processor (nano). In either case, since the ROS detection module is running on a secondary processor, either the navigation software or *roscore* must be started prior to starting collision detection.

1.1 3D/2D CAMERA BASED OBJECT COLLISION DETECTION

The first procedure summarizes how to start collision detection without visualizing the data. As such, only remote terminal windows are needed.

1. Start a remote ssh session to a primary host processor, and in a terminal window enter:

   `roscore`.

2. Start a second ssh session to the Nano secondary processor, and start the Zed sensor ROS driver by entering:

   `roslaunch zed_wrapper zed.launch`

3. In a third PuTTY (ssh) session to the nano (Figure B.1), start the collision detection module.

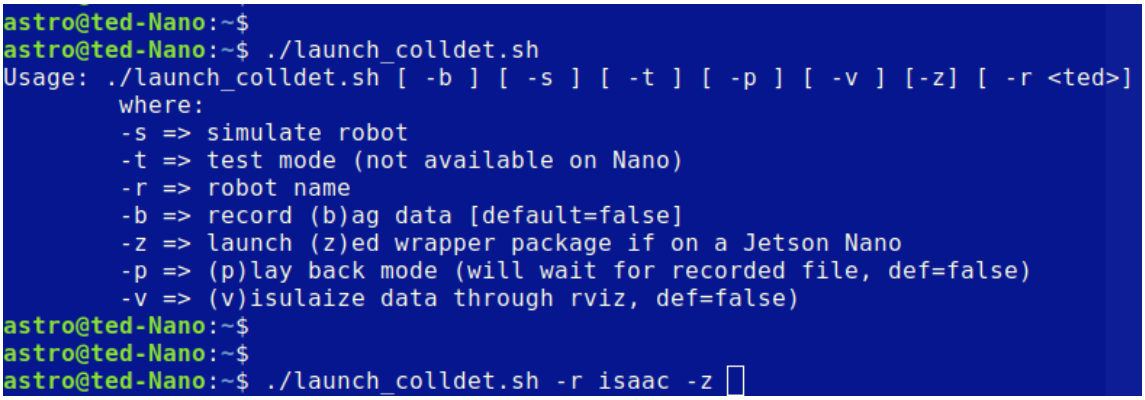


Figure B.1 Starting 3D camera based collision detection

4. After the module is started, collision data is output when objects are detected within the collision zone in front of the robot (Figure B.2).

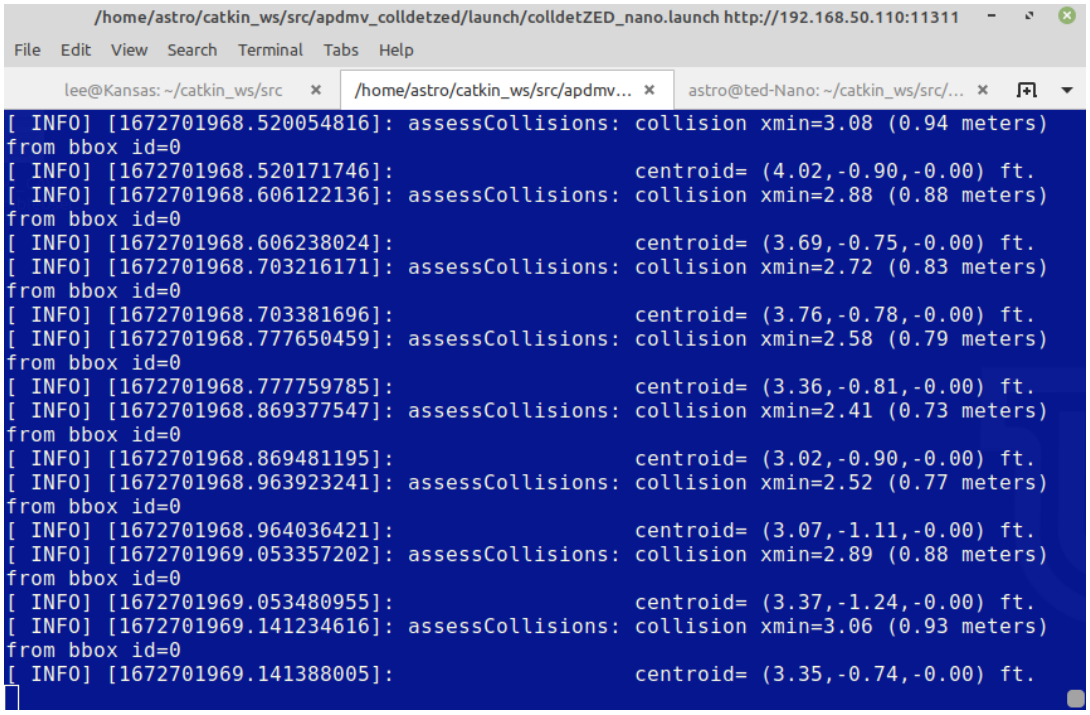


Figure B.2 Collision data indicating mid-location of detected object, and bounding box surface position that is closest to the front of the robot, in meters.

Figure B.2 Collision data indicating mid-location of detected object, and bounding box surface position that is closest to the front of the robot, in meters.). The package node output Is not very intuitive for verifying collision detections. This is addressed by visualizing the sensor data and the detected collision data in ROS rviz. The second procedure summarizes how to start collision detection and simultaneously visualize the output.

1. Start a remote ssh session to a primary host processor, and in a terminal window enter:

   `roscore`

   .

2. Start a second ssh session to the Nano secondary processor, and start the Zed sensor ROS driver by entering (on the first robot launch **`zed.launch`** and on the second robot launch **`zed2.launch`**:

   `roslaunch zed_wrapper zed.launch`

3. Open a NoMachine RDS into the nano robot secondary processor.

4. Open an x-term window, and again start the collision detection module, with the "-v" option. The ROS rviz tool will open to display both collision detection, and Zed sensor data.

5. In one of the terminals, start the zed driver:

   roslaunch zed zed.launch

6. In a second window, launch rviz

7. Launch the image examiner tool from a second terminal window:
   roslaunch apdmv_lepton test_lepton.launch

8. Two windows will display the radiometric temperature image (and maximum temp detected), and a binary threshold image.
9. It is also possible to record the image data to disk (as a ROS bagfile). In a separate terminal window, enter:

   $(rospack find apdmv_recdata)/scripts/record_lepton.sh

1.1.1 LiDAR based object/collision detection.

The first procedure summarizes how to start collision detection without visualizing the data. As such, only remote terminal windows are needed. To start the LiDAR please refer to the earlier section,

1. Start a PuTTY session to the primary host processor, and in a terminal window enter: **roscore**.

2. Start a PuTTY session to the secondary processor (azimov-3 for the first robot, or Isaac-2 for the second robot).

3. On the secondary host processor start the LiDAR ROS driver package:

   roslaunch ydlidar_ros lidar_view.launch

4. Preferably on another PuTTY session window into the same secondary processor, start the collision detection (see figure B.1 command line argument definitions)

   ~/launch_colldet.sh -r Isaac

5. Launch the image examiner tool from a second terminal window:
   roslaunch apdmv_lepton test_lepton.launch

6. Two windows will display the radiometric temperature image (and maximum temp detected), and a binary thresholded image.
7. It is also possible to record the image data to disk (as a ROS bagfile). In a separate terminal window, enter:

```
$(rospack find apdmv_recdata)/scripts/record_lepton.sh
```

APPENDIX C:
LINUX AND ROS COMMANDS

Table C.1 summarizes abridged list of relevant Linux terminal commands, for those users who are unfamiliar with Unix/Linux operating systems. It is highly advised to practice them in a terminal a few times, to internalize how and what they perform. One thing to note, like a windows command shell, you can use the TAB-key to complete filenames, directories, and even commands, which saves a lot of typing and time.

Be careful when trying out the "rm" command: if a file or directory is removed, it is deleted permanently, for all practical purposes.

Table C.1 Linux/ubuntu terminal commands

Command	**Meaning**
ls, ls -l, ls -la, or, ls /desired/directory/path, ls *foo*	List content of a directory, wildcards, * = string, ? = character.
cd '/a/given/path/to/a/directoy', cd ..	Cd to directory, parent directory
cd ~	Change directory into user home directory
~/a_script_to_run.sh	Run an executable shell script, "a_script_to_run.sh" residing in the user's home directory.
rm, rm -rf	Remove file, forcefully remove file or directory recursively (be careful!)
Cp, cp -r, cp -ru	Copy, copy recursive, copy recursive and update
mkdir, mkdir -p /path/to/new/dirname	Make directory

The ROS environment provides a multitude of tools to analyze, configure, and control the ROS processes. Table B2. summarizes a few commonly used, relevant ROS commands. It is beyond the scope of this guide to delve into these or the ROS-1 architecture. However, it is important to understand that the robot processes utilize two very basic inter-process communication ROS-1 frameworks: messages, and services. A ROS message is contained within a *message topic*, identified by a *message name*. A service message is similarly identified by a *service name*. The message and service topic naming convention for the MnDOT robots are always prepended by "apdmv" either as part of the name "apdmv_<rest of name>, or as a group namespace "apdmv/<rest of name>. Note that the MnDOT Robot customized ROS packages also use the "apdmv_<rest of name>" convention.

Table C.2 ROS commands

ROS command example	meaning
rostopic list | grep apdmv	Parse list of all custom MnDOT robot ROS topics
rostopic pub apdmv_command_topic std_msgs/String “data: STOP”	Publish topic data; ex. publishes MnDOT robot command (of type ‘std_msgs/String), ‘STOP’.
rostopic echo apdmv_status_topic	Subscribes to topic and prints out all topic data field values.
rostopic type apdmv_status_topic	Prints out the topic ROS message type associated with topic name, “apdmv_status_topic”
rosmsg show <ROS message type>	Shows all ros message field types for provided ROS message type.
roslaunch apdmv_dgpspnt nmea2sp.launch	Start a launch file, nmea2sp.launch within the MnDOT robot package, apdmv_dgpspnt.
roscore	Start ROS. (launch files starts ROS within the launch files context).

APPENDIX D:
TROUBLE SHOOTING GUIDE

Table D.1 Trouble shooting guide

Symptom	Resolution/test
Terminal window does not appear with the password prompt.	Check the Network cable from AP/router. Open a basic command window and try to PING the host IP address.
SSH client indicates a network error, and (typically) timed out.	Check the Network cable from AP/router. Open a basic command window and try to PING the host IP address.
The GPS NMEA serial ROS driver crashes or indicates and error.	For **azimov**, the serial device port *must* be **ttyUSB2**, and for **isaac, ttyUSB1.** Nevertheless, to continue testing functionality, try another port such as "ttyUSB1", or "ttyUSB0", to determine if the port was assigned incorrectly. No matter the result of the test, try to re-initialize to the correct ports by following the steps below *in order*: 1. Unplug the GPS USB serial cable from the robot USB port. 2. From the ssh BASH command terminal enter: **`~/ROBOT_REBOOT.sh`** 3. The PuTTY/ssh session window will drop, or loose the connection. Wait a minute, and then restart the session to login again. 4. Once logged in, plug the GPS USB serial cable back into the Robot. Repeat the GPS test procedure.
For the DGPS test, apdmv/dgpspnt topic does not output anything, and hangs waiting.	Is the topic there? From the remote ssh BASH command window, determine if the topic is actually being published, e.g. enter: **`rostopic list | grep apdmv`** If it does not exist in the list, determine if an arbitrary projection region was set, e.g. enter: **`rostopic echo echo apdmv_status_topic/GPSProjName`**
Symptom	**Resolution/test**

IMU data is printing out all 'NaN's instead of numbers, in either the topic data from apdmv/hwtimu_data, or the test output.	Try different ports. On Azimov, try "ttyUSB0" for example. on Isaac, it should be "ttyS2". Ensure cable/connection is secure, reboot the processors and try again (Note: if GPS cable is plugged in, please unplug before rebooting, and follow steps in this trouble-shooting table for proper reboot sequence).
After repeating steps for trying to get IMU output, it still does not present data.	At this juncture, more invasive troubleshooting may be necessary, by exposing and original 4-pin connector to a USB TTL UART level dongle and then running WitMotion configuration software to test device on Windows PC (the UMN team can supply the link to the software).
The motor controller test fails to find any valid serial port, or otherwise the robot motion is erratic or completely unresponsive.	Check RC controlled robot motion. Are the Controller/servo batteries low? Remove robot top cover and check USB port cable/connection from controller to the host microprocessor. *Please power down everything first, before proceeding*! Test motor controller function. 1. Download the free RoboRun+ utility from RoboTeq and start it. 2. Once the USB controller serial cable is exposed by removing the Robot top cover, unplug it from the host robot processor and into Windows PC. Alternatively, you can connect a separate USB-B cable directly between the PC and the controller. 3. Turn on Robot servo/controller power. The software should find, and then automatically connect to the controller.