Obstacle Avoidance Using 3D LiDAR

Real-Time Point Cloud Processing

Bachelor's thesis

Electrical and Automation Engineering

Spring 2025

Tomáš Jelínek

HAMK
Häme University
of Applied Sciences

Electrical and Automation Engineering
Author Tomáš Jelínek
Subject Obstacle Avoidance Using 3D LiDAR
Supervisors Arturo Escobedo (TEC), Alfonso Gomez (TEC), Katariina Penttila (HAMK)

Abstract
Year 2025

Obstacle detection is a critical aspect of autonomous robot navigation, especially in dynamic environments. This thesis presents the development and implementation of an obstacle avoidance system for the Jackal mobile robot, utilizing 3D LiDAR technology and ROS 2 for real-time processing. The system enables the robot to navigate a dynamic environment by calculating position of obstacles and adjusting path accordingly. The main scope is to design an obstacle avoidance algorithm based on the processed data.

The system was tested in simulation as well as the the real robot platform, where the performance was analysed in multiple scenarios with different parameters. The results demonstrate that the robot can effectively avoid obstacles, although minor limitations were observed under certain conditions, where the LiDAR failed to detect all points.

Keywords LiDAR 3D, point cloud, obstacle, detection, avoidance, UGV
Pages 42 pages and appendices 1 page

Content

Figures, tables and equations

Appendices

Acronyms

LiDAR	Light Detection and Ranging
ROS	Robot Operating System
Rviz	Robot Vizualization
UGV	Unmanned Ground Vehicle
IMU	Inertial Measurement Unit
VRU	Vertical Reference Unit
AHRS	Attitude and Heading Reference Systems
RANSAC	RANdom Sample Consensus
DWA	Dynamic Window Approach
VHF	Vector Field Histogram
VLP	Velodyne Puck
RJ	Registered Jack

1 Introduction

This thesis explores the development of an obstacle avoidance system for the Jackal mobile robot using 3D LiDAR technology. Autonomous robots are increasingly deployed in dynamic environments like warehouses, where they perform tasks such as monitoring, maintenance, and transporting goods. However, navigating such spaces poses a challenge due to constant changes, such as moving equipment and workers.

This project is part of a larger initiative at Tecnológico de Monterrey, Campus Querétaro, where autonomous robotic systems are under development. Effective obstacle avoidance is a crucial aspect that ensures the robot's usability in real-world applications. Without it, the robot could get stuck or require manual assistance, reducing its operational efficiency.

Currently, the Jackal robot is limited to following pre-defined paths, which restricts its adaptability to sudden changes in its surroundings. This thesis aims to address that limitation, providing a solution that showcases the institution's expertise in robotics. The project utilizes the ClearPath Jackal mobile platform, equipped with a 3D LiDAR sensor for real-time detection and an onboard computer for processing. ROS 2 serves as the software backbone, handling sensor data, processing, and control the robot’s engines.

The primary aim of this thesis is to create a system that enables the Jackal robot to autonomously detect and avoid obstacles in real-time. The part of communication with LiDAR and Jackal robot is delegated to other students so it is not part of the scope.

To prove that the system works it will be tested primarily using a simulation environment provided by ClearPath and then it will be passed to the real Jackal robot. The main metrics used during the tests are that the obstacle detection works reliably and the robot can avoid the as well as to prove that 2D LiDAR would not be feasible in the same application.

2 Related work

Obstacle avoidance is very important part of UGV's. Many techniques to detect obstacles have been developed, such as stereo IR depth camera, as well as two or three dimensional LiDAR.

In recent years, 3D LiDAR has become popular due to its high precision and frequency of environmental scanning, Several studies focus on autonomous navigation in dynamic environments, particularly those relevant to agriculture, urban and industrial areas.

2.1 2D LiDAR obstacle detection

2D LiDAR is capable of scanning the surroundings only on a single plane. It does not require as much computational power and the output can be sent directly into ROS navigation package. However the main drawback is that it does not detect all of the obstacles if they are not regular. Resulting in that the height is crucial depending on the application.

Several studies came up with an idea of tilting the LiDAR sligthly, but it still has many disadvantages, two of which are listed below:

- A major limitation of 2D LiDAR is that it only captures data on a single plane. This means it has no vertical information and cannot detect objects above or below this plane, which could lead to missing critical obstacles, like holes, bumps, or low-lying hazards, if the LiDAR is positioned parallel to the ground.
- When the LiDAR is angled downward, the area directly in front of the vehicle is not scanned. This could lead to obstacles going undetected in the near range or create a blind spot, where objects close to the sensor might be missed. This can be mitigated by using additional sensors or other sensor fusion techniques.
  Obstacles above the sensor's scanning plane, such as overhanging branches or low-hanging wires, cannot be detected by 2D LiDAR, making it inadequate for certain environments like urban areas or forests. (Yalcin et al., 2013)

2.2 3D LiDAR obstacle detection

One of the critical steps in 3D LiDAR-based obstacle detection is filtering out ground points to focus on potential obstacles. Various methods are available for ground filtration, ranging from simple, intuitive techniques that are computationally efficient and suitable for real-time applications, to more complex algorithms that offer higher accuracy but at the cost of increased computational load. Choosing the right method is essential to balance performance and accuracy in real-time obstacle detection systems.

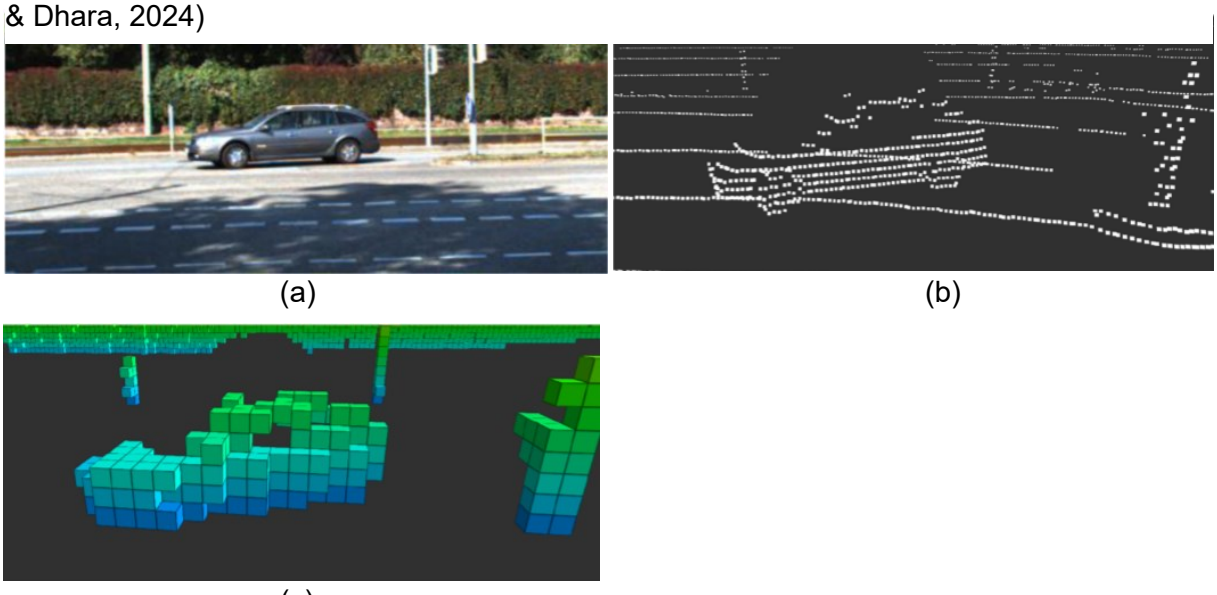
2.2.1 Non-feasible for real-time applications

These methods, although accurate, are computationally intensive and better suited for offline processing or systems with significant processing power.

Voxel Grid-based methods involve dividing the point cloud into a voxel grid, followed by nearest neighbor searches or graph-based methods (Luo et al., 2021). This technique works well for dense point clouds, such as those generated by Velodyne LiDAR, but can be slow due to the need to process millions of points (Saha & Dhara, 2024).

In the Figure 1 is an example of a voxel occupancy map calculated for a 3D LiDAR point cloud.

Figure 1 - Voxel map (**a**) Scene with object (**b**) 3D LiDAR data (**c**) Occupancy voxel map (Saha & Dhara, 2024)



(a) (b)

(c)

Supervoxel segmentation involves boundary-enhanced segmentation based on spatial and geometric properties of supervoxels. While accurate in maintaining object boundaries, the method is too slow for real-time usage due to its complex clustering mechanism.

2.2.2 Feasible for real-time applications

These methods are optimized for sparse point clouds and offer a balance between accuracy and computational speed, making them ideal for real-time applications like mobile robotics and autonomous vehicles:

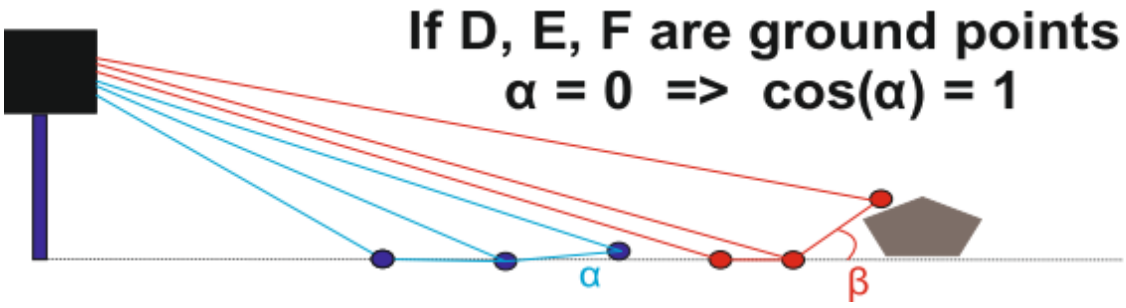
- Height-based methods leverage the fixed orientation of the sensor relative to the ground. By analyzing the Z-coordinates of points, especially when the sensor's position such as, roof-mounted LiDAR in cars, is well-defined, these methods can filter ground points efficiently. Additionally, integrating IMU, VRU, or AHRS makes the system robust to sensor movements (Lima et al., 2016).
- Spherical Coordinates and Radial Distance analyses points based on their radial distance in spherical coordinates. Points with similar radial distances are classified as ground points. This method operates quickly ~2.8 ms and does not assume the road or ground is always the lowest point, making it adaptable to various environments (Yin et al., 2016).
- Angle-based methods estimate the ground plane by analyzing the angles between vectors created from consecutive points A, B, and C in a scan. If the angle is close to zero, the points are likely on the ground. This approach, although simple, is computationally efficient and well-suited for real-time applications (Petrovskaya & Thrun, 2009).

2.2.3 Hybrid approach

A hybrid approach combines different techniques to balance accuracy and processing speed.

Vector angle and RANSAC combination uses the VLP-16 LiDAR involves projecting points onto the YZ plane, computing angles between consecutive vectors to estimate ground points shown in Figure 2, and then the RANSAC algorithm is applied for ground plane fitting. It is an iterative algorithm used to estimate the parameters of a mathematical model from a set of observed data that contains outliers, noisy data points. This two-step approach reduces the computation time, as the dot product is used to pre-filter points, followed by RANSAC for precise ground plane estimation (Miądlicki et al., 2017).

Figure 2 - Dot product method explanation (Miądlicki et al., 2017)



2.3 Obstacle avoidance

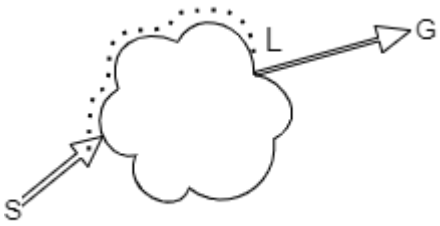
Most obstacle avoidance algorithms can be categorized into reactive and deliberative methods. Reactive approaches, such as the DWA and potential fields, enable robots to respond quickly to obstacles but can sometimes lead to local minima, where the robot becomes trapped. On the other hand, deliberative methods, such as path planning algorithms like A* or D*, provide more efficient and optimized paths but may struggle in environments with frequent, unpredictable changes.

2.3.1 Real-time obstacle avoidance in opened environment

Obstacle avoidance methods vary in complexity, sensor dependency, and their feasibility in real-time applications. These methods and their limitations are described further.

The Bug Algorithm and its variant Bug2 are simple reactive methods for obstacle avoidance (Susnea et al., 2009). The example can be seen in Figure 3.

Figure 3 - Bug algorithm



In the Bug algorithm the robot follows an obstacle boundary entirely, finds the closest point to its goal, and leaves the obstacle at that point.

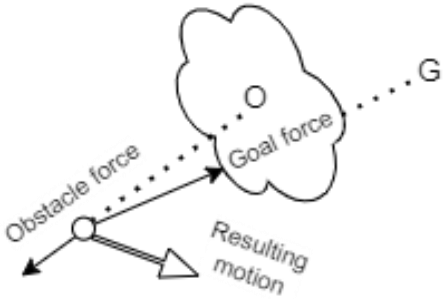
In the Bug2 algorithm the robot follows the obstacle's boundary but leaves the boundary when it intersects with the line connecting the robot's start and the goal.

Both algorithms can be inefficient, especially in complex environments where the robot may spend extra time fully circling obstacles.

These algorithms do not account for the kinematics of many real robots, wheeled robots with turning constraints, which leads to poor performance in practical scenarios..

The Potential Field Algorithm creates a virtual force field, where obstacles repel the robot, and the goal attracts it. The robot moves based on the resultant force (Susnea et al., 2009). The example can be seen in Figure 4.

Figure 4 - Potential Field Algorithm

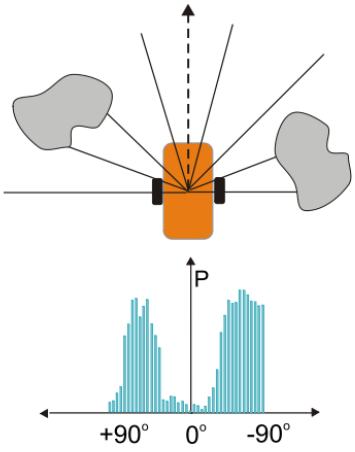


The robot can get stuck in dead zones, where the forces cancel each other out, especially near narrow passages or between obstacles.

Calculating the forces can become computationally intensive, especially when real-time decisions are required.

The VHF Algorithm addresses sensor noise by creating a polar histogram from several recent readings and then determining obstacle-free paths based on the cost function of each passage (Susnea et al., 2009). The example can be seen in Figure 5.

Figure 5 - VHF Algorithm
(Susnea et al., 2009)



VFH requires substantial computation, especially when considering a high-resolution polar grid and numerous sensor readings. This can challenge real-time applications unless powerful hardware is available.

Due to the high computational demand, this algorithm may not be suitable for robots with limited processing power, like those using embedded systems for low-cost or low-power applications.

2.4 Proposed solution

This thesis further explores the limitations of existing 2D and 3D LiDAR-based obstacle detection systems and proposes a solution that leverages 3D LiDAR to address these challenges. By using the Velodyne Puck-16 sensor, this project enhances obstacle detection accuracy through full 3D spatial data, overcoming the vertical limitations of 2D LiDAR systems. Additionally a hybrid approach is introduced, combining dot product estimation with the RANSAC algorithm to filter ground points in real-time. This method reduces computational load while maintaining high accuracy, making it feasible for real-time applications.

Given the requirements for a system that is both easy to deploy and does not rely on additional sensors, this thesis further develops and optimizes the rebound algorithm for obstacle avoidance. It is particularly well-suited for applications where simplicity and low computational overhead are essential, as it relies solely on existing sensors, such as the 3D LiDAR. The algorithm can accurately estimate nearest obstacle and dynamically adjust the robot's path to avoid collisions in real-time.without requiring supplementary systems, such as IMU.

.

3 Technologies in use

The obstacle avoidance system proposed in this thesis relies on three main components: 3D LiDAR technology, ROS 2 (Robot Operating System), and the Jackal robot platform. Each of these technologies is widely used in robotics research and has a robust body of supporting literature.

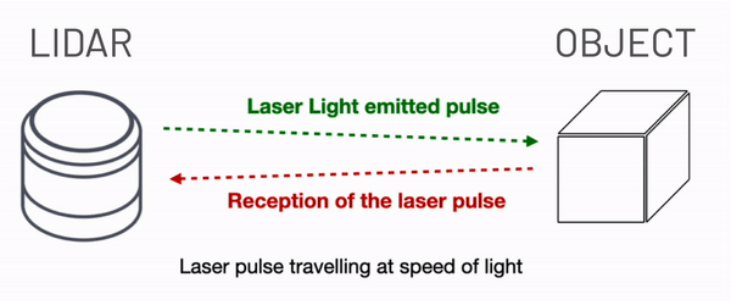
In this section will be explained technologies in use and communication protocols.

3.1 Velodyne Puck 16 3D LiDAR

3D LiDAR operates by emitting laser beams to scan the surrounding environment and measuring the time it takes for the reflected light to return as it is shown in

Figure 6 and Equation 1, where *R* is the distance of the point, *c* is a constant for the speed of light and *t* is a time the laser traveled in both directions. LiDAR systems typically consist of multiple laser layers, commonly 16, 32, or 64, that rotate around a vertical axis to create a comprehensive map of the environment. For real-time applications, a lower resolution LiDAR is often sufficient to balance data volume and processing speed (Miądlicki et al., 2017).

Figure 6 - Principle of LiDAR – (Dubois, 2022)

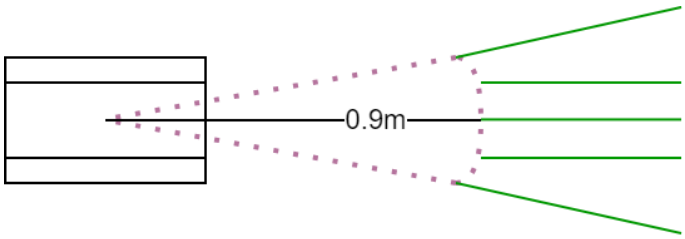


Equation 1 - Distance calculation

$$R = \frac{c * t}{2}$$

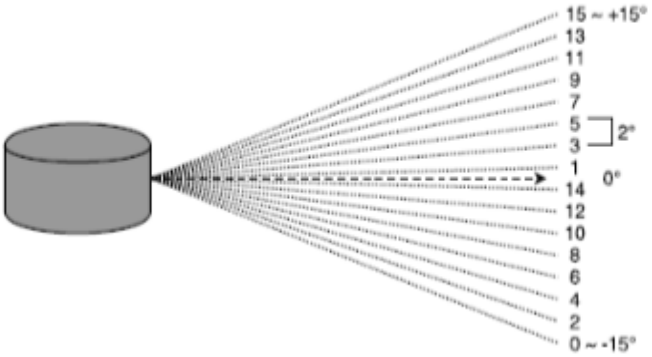
The main limititation is that only objects further than 0.9 meters can be detected. If anything get closer, the reflected laser returns too early and the sensor is not able to detect the distance, resulting in shadows that cover the objects behind and in that area are no points. This is demonstrated in Figure 7.

Figure 7 - LiDAR minimum detection distance



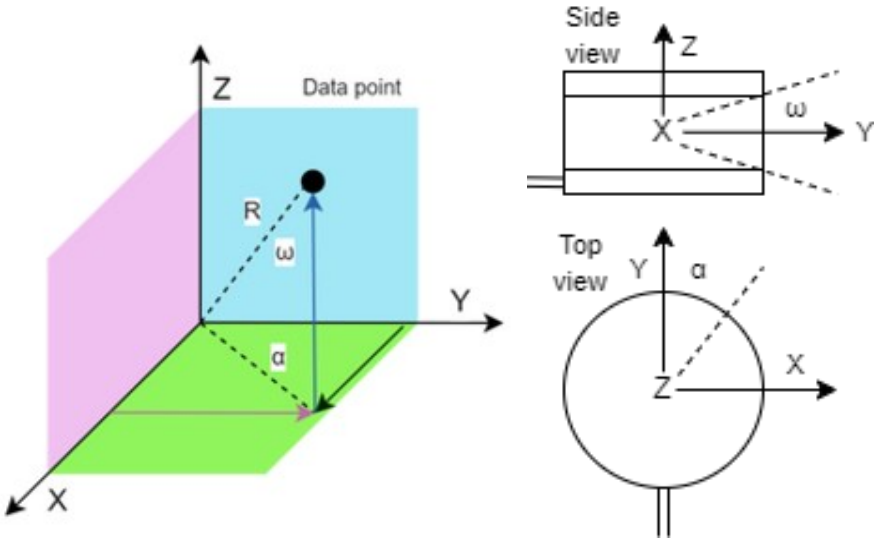
VLP 16 has 16 layer and vertical field of view is +/-15 degrees, as can be seen in Figure 8. The sensor generates approximately 15,000 points per rotation and frequency can be adjusted between 5-20 Hz, resulting in up to 300,000 points per second. (Velodyne, n.d.).

Figure 8 - LiDAR FOV (Miądlicki et al., 2017)



The raw output data from LiDAR are not possible to be used directly, the format is following, number of vertical layer and also horizontal layer, based on the steps and resolution can be calculated horizontal and vertical angle as it is illustrated in Figure 9 and Equation 2.

Figure 9 - LiDAR point calculation



Equation 2 - Point cloud calculation (Miądlicki et al., 2017)

$$X_i = R * \cos(\omega) * \sin(\alpha)$$
$$Y_i = R * \cos(\omega) * \cos(\alpha)$$
$$Z_i = R * \sin(\omega)$$

The VLP 16 LiDAR is also equipped with a small processing box, which changes the native port to the standard RJ45, so it can be connected directly to the notebook.

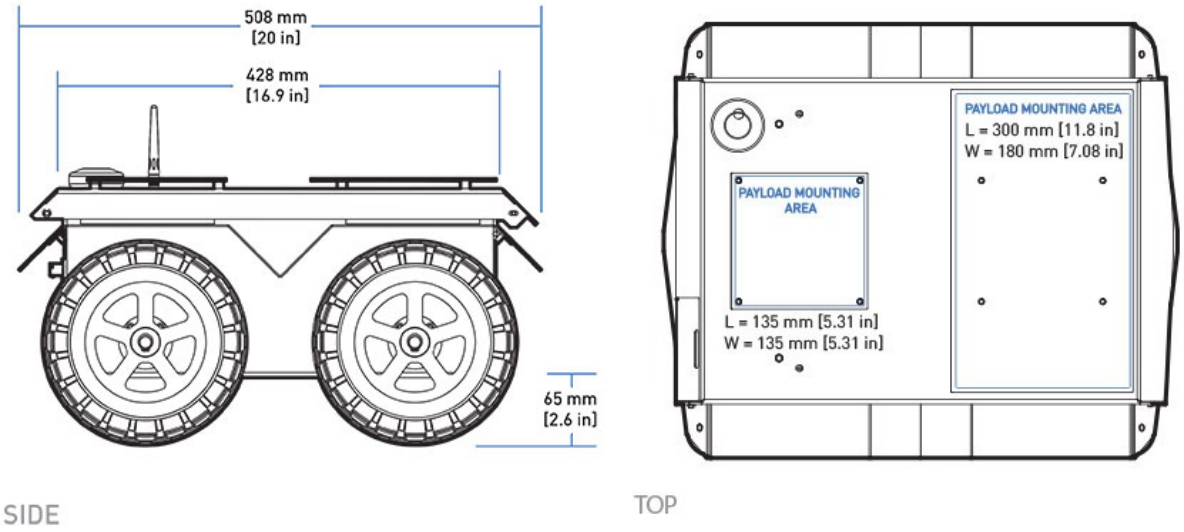
3.2 ClearPath Jackal mobile robot

The Jackal UGV by Clearpath Robotics is a compact and versatile field research platform designed for robotics applications and fully integrated with the ROS and Gazebo simulation model, which allows bypassing time-consuming setup processes and start working immediately.

Jackal has multiple power outputs, including 5V, 12V, and 24 to support additional sensors or computing hardware. Its modular design provides to mount and connect accessories, while the internal compartment offers space for further expansion, such as additional computer. The communication is handled trough both Bluetooth and Wi-Fi (ClearPath Robotics, 2020).

The robot dimensions are shown in Figure 10.

Figure 10 - Jackal Robot dimensions (ClearPath Robotics, 2020)



3.3 Notebook to provide computational performance

The point cloud data in this project is processed using a Dell Latitude 5420 notebook, equipped with an Intel Core i7-1185G7 11th generation processor and 16GB of DDR4 RAM. This powerful configuration ensures efficient data handling, while the integrated Intel Iris Xe graphics card supports visual rendering of large datasets. The notebook's 512GB solid-state drive provides ample storage for high-volume LiDAR data. With multiple connectivity options, including Thunderbolt 4, USB, and Ethernet ports, the device facilitates fast data transfers. (Dell, n.d.)

The notebook operates on Linux Ubuntu 22.04, a reliable and flexible platform well-suited for robotics development. This operating system ensures compatibility with a wide range of open-source tools and libraries, essential for this project. Key software installed includes the ROS 2 Humble, which manages sensor data acquisition, processing, and communication, and Visual Studio Code, a versatile integrated development environment for coding and debugging. Additionally, Docker is utilized for containerizing applications, while GitHub is used for version control and collaboration. The Gazebo software plays a crucial role in simulating the robot's behavior and testing algorithms in virtual environments.

3.4 Nvidia Jetson small board computer

The Jetson Xavier NX Developer Kit is a compact module equipped with a 6-core CPU, 384-core GPU, and 16 GB eMMC storage, supporting various interfaces including USB 3.1, PCIe GEN4, Gigabit Ethernet, and multiple UART, CAN, SPI, and I2C channels. It handles video encoding/decoding up to 4K resolution and offers MIPI CSI-2 lanes for high-speed sensor integration. It supports flexible power modes 10W, 15W, 20W. (nVidia, 2020)

In this system, the Jetson Xavier NX serves as a bridge between ROS 2 and the robot's motor controller. Specifically, it translates the cmd_vel ROS topic, which contains linear and angular velocity commands, into individual velocity values for the robot's right and left wheels. These calculated values are then transmitted to the motor controller.

3.5 ROS 2

ROS 2 is an open-source framework designed to facilitate the development of robot applications. It acts as a meta-operating system, providing a collection of tools, libraries, and conventions to simplify the process of creating complex, robust robot behaviors across various platforms. ROS 2 offers essential services such as hardware abstraction, device control, message passing between processes, and package management.

Its framework is built around a peer-to-peer network of processes, where communication between nodes, software modules, happens asynchronously over topics or synchronously via services, depending on the application requirements. Additionally, ROS supports multi-machine configurations. (Tellez, 2019)

The current version, ROS 2, is a complete redesign of the original ROS framework, addressing the limitations of its predecessor to meet industry standards and use cases. ROS 2 introduces support for real-time systems, embedded platforms, and non-perfect network conditions, making it ideal for production environments. (Wind River, 2020)

Although ROS 2 coexists with the first generation, it is poised to fully replace it as the go-to middleware for robot development in the near future.

3.6 Communication and data transfer

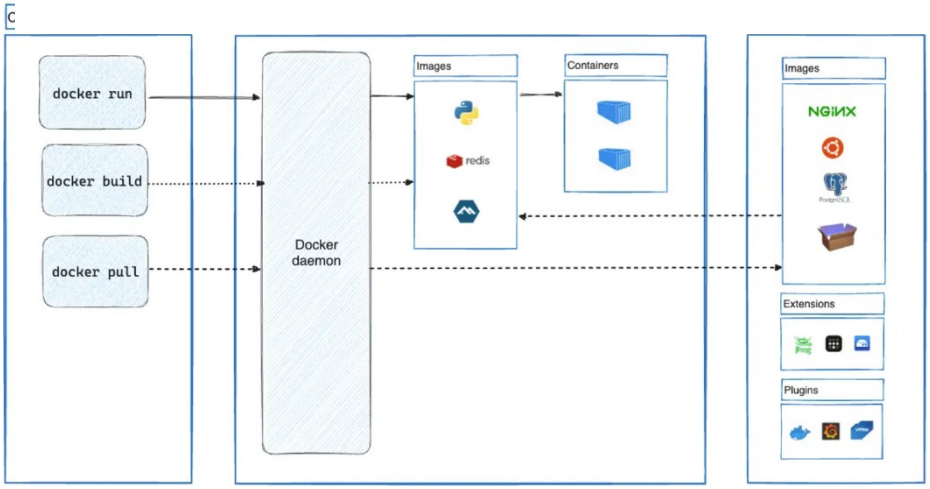
Communication with robot and LiDAR has been done by two other students as this thesis is part of a university's long-term project. It will be briefly explained and referenced as it was outsourced.

The VLP-16 LiDAR has a built-in web service client that allows for configuring various sensor settings, such as the RPM and the type of return signal from the reflected laser. It also includes a compact processing unit that converts the point cloud data from its proprietary connector into an RJ45 interface, making it compatible for direct connection to a computer. In Ubuntu, a Docker container was created to simulate the processing of the LiDAR output, converting it into a ROS 2 message that is published on the velodyne_points topic. (Mendoza & Flatscher, 2024)

Docker uses a client-server architecture. The Docker client communicates with the Docker daemon, which builds, runs, and manages containers. Docker containers are built from

images, which are immutable snapshots of a system environment, as can be seen in Figure 11. These images are defined by Dockerfiles, where instructions such as installing software, copying files, or running commands are specified. (Docker, 2013)

Figure 11 - Docker architecture (Docker, 2013)



The Jackal robot is directly connected to a Jetson small board computer, which has ROS2 installed. The original package provided by ClearPath was modified to accept velocity commands in both linear and angular formats. To enable communication with an external computer, both devices must be on the same network, allowing them to access each other's topics and publish data accordingly. (Reyes, 2024)

4 Methodology

4.1 Ground floor detection

To ensure accurate obstacle detection, it is necessary to filter out ground points from the 3D LiDAR scans. Two methods have been tested in Gazebo simulation software as well as with the Jackal robot and VLP 16 LiDAR.

4.1.1 LiDAR height based

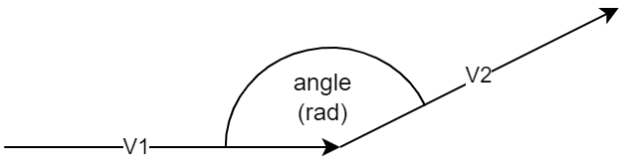
This method is very simple, the algorithm goes through every single point in the input data and compares the Z coordinate to the height of the LiDAR which is given by a parameter. The most significant drawback is that in the most basic form it can be used only on a flat surface. If the robot drives over any bumps and becomes even slightly misaligned, the points will not be filtered out correctly This can be improved by an IMU, which gives the angular orientation of the robot in the space. However, in this particular project only the 3D LIDAR was used, so this method is not feasible.

4.1.2 Vector angles and plane fitting

This process involves geometric analysis and plane-fitting techniques, where the following steps are performed for each LiDAR scan.

Three consecutive points from the point cloud are analyzed to determine whether they belong to a ground surface. This is done by computing the angle in radians between two vectors formed by these points which is represented by Figure 12 and Equation 3.

Figure 12 - angle of two vectors



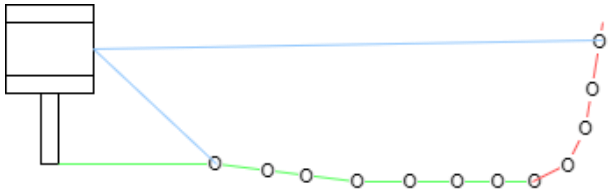
Equation 3 – angle calculation

$$\text{angle(rad)} = \text{acos}\left(\frac{v_1 * v_2}{|v_1| * |v_2|}\right)$$

Where v_1 and v_2 are vectors formed by consecutive points, and $|v_1|$ and $|v_2|$ are the magnitudes of these vectors

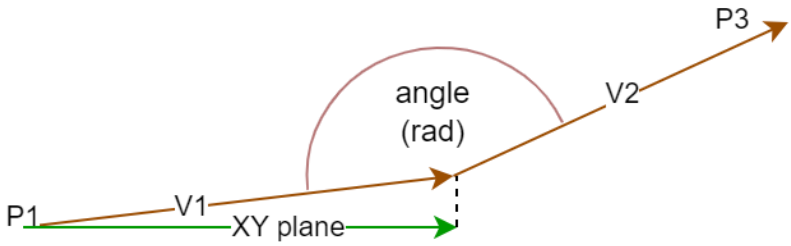
The angle helps identify points lying on a flat plane by checking that the vectors are aligned, cos of the angle is close to 1. Points that meet this condition are considered potential ground points and passed to the next step for further refinement Figure 13.

Figure 13 - Potential Ground points



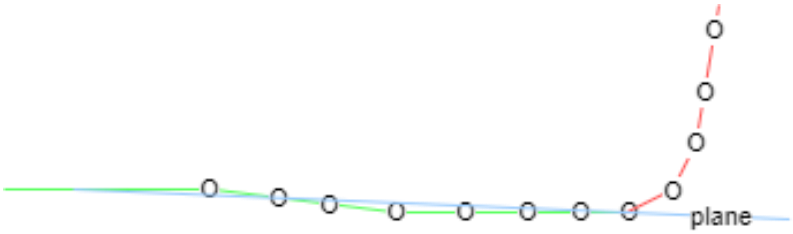
To make sure that the vectors are horizontal the same logic is also followed for v_1 and v_2 is subsidized by v_1 with Z value set to 0 representing XY plane as it is shown in Figure 14.

Figure 14 - angle calculation XY plane



After estimating the initial set of ground points, a more precise ground plane is fitted using RANSAC algorithm. A random subset of points is iteratively selected to fit a plane and checks

Figure 15 - Plane fit through marked points



how many other points agree with this plane model. The best-fitting plane is represented by the Figure 15 and Equation 4.

Equation 4 - Representation of a plane

$$ax + by + cz + d = 0$$

Here *a*, *b*, and *c* are the normal vector components of the plane, and *d* is the perpendicular distance from the origin to the plane.

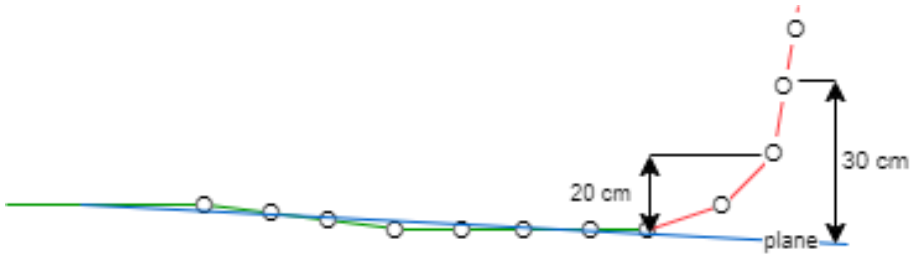
Once the ground plane is fitted, the remaining points are filtered based on their distance from this plane. The distance of each point to the plane is computed using the following Equation 5.

Equation 5 - Distance from plane calculation

$$distance = \frac{|a * x + b * y + c * z + d|}{\sqrt{a^2 + b^2 + c^2}}$$

Here *x*, *y*, and z are the coordinates of the point. Points with a distance greater than a specified threshold are retained as non-ground points, while points close to the plane are discarded as ground points. The example can be seen in Figure 16.

Figure 16 - Points distance from the plane



Filtering process significantly reduces points representing the ground, so the obstacle detection algorithm can focus on relevant points in the environment.

4.2 Obstacle detection

Once the ground floor is filtered out, the cloud point can be used to detect nearest obstacle by a function Compute Distance, which will be recalled in following section.

The logic of the computation is following, the function will get a point cloud with a filtered ground floor and then it calculates a horizontal distance for each point based on the XY coordinates. After each full rotation of the LiDAR, the distance is initialized to a large value, which is gradually decreased, until the minimum distance is found. A Figure 17 demonstrates how this is done for one layer of the cloud point:

Figure 17 - Obstacle distance calculation

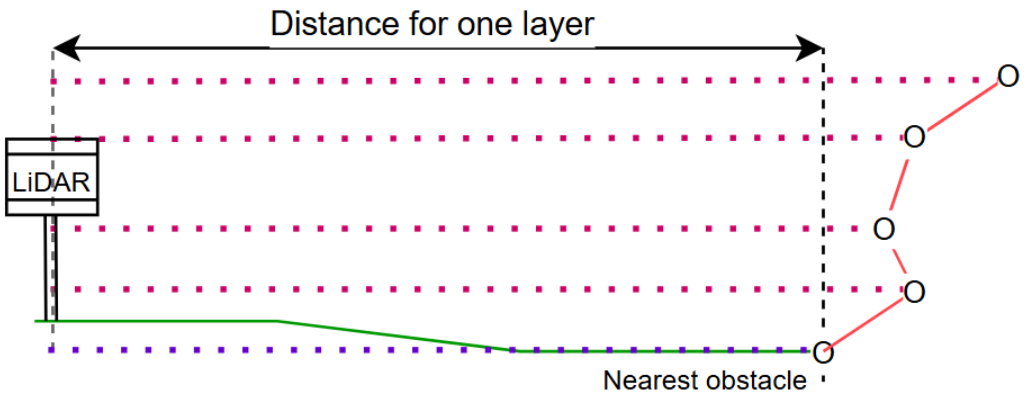
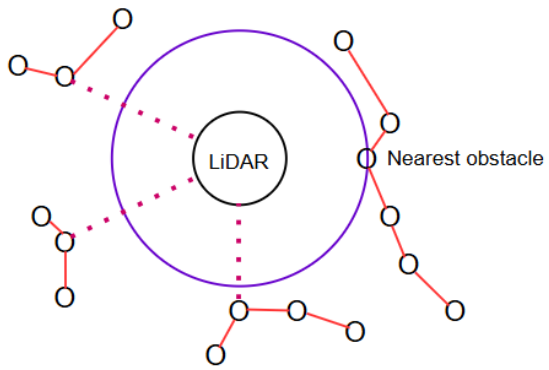


Figure 18 shows, how the system detects nearest obstacle to be handled by the obstacle avoidance algorithm. The red lines represent objects around and purple circle is a radius of the nearest detected point.

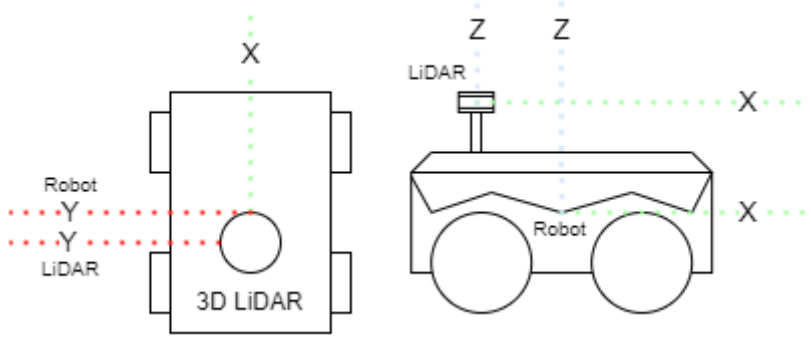
Figure 18 - Obstacle detection from the top



4.3 Obstacle Avoidance algorithm

Beforehand, the coordinate system is translated from the centre of LiDAR in respect to the robot as it can be seen in Figure 19.

Figure 19 - Coordinate system translation



Initially, the robot goes straight and constantly checks if any object is inside an envelope. At first the robot slows down, and if it detects that the distance to the object reduces, to the obstacle it stops. In both cases it gradually turns away from the obstacle based on the nearest point position, as is illustrated in Figure 20 and Figure 21.

Figure 20 - Obstacle avoidance flowchart

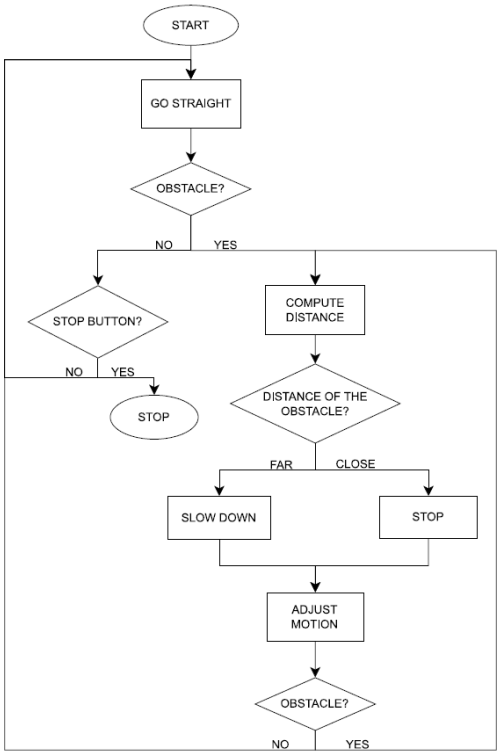
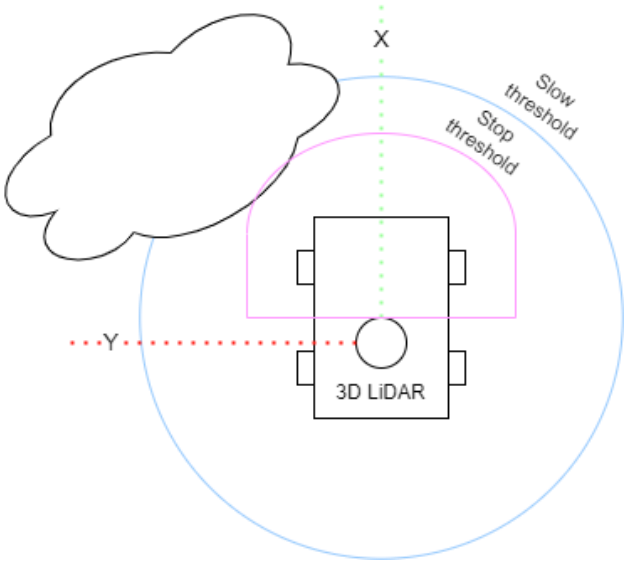


Figure 21 - Bounce algorithm



5 Implementation

5.1 System overview

The overall system for obstacle avoidance using 3D LiDAR on a Jackal robot can be represented through a block diagram that illustrates the interaction between hardware components, ROS 2 software nodes, and the algorithms responsible for ground plane detection and path planning. The system integrates sensor data from a VLP-16 3D LiDAR and wheel encoders into ROS 2 nodes, where it undergoes real-time processing for decision-making and actuation on the Jackal robot.

5.2 Hardware configuration

The robot used for this project is a ClearPath Jackal, an UGV designed for outdoor and indoor robotics applications. The Jackal robot is equipped with 3D LiDAR and its placement can be seen in Figure 22 and Figure 23.

Figure 22 - Jackal Robot with a VLP16 LiDAR

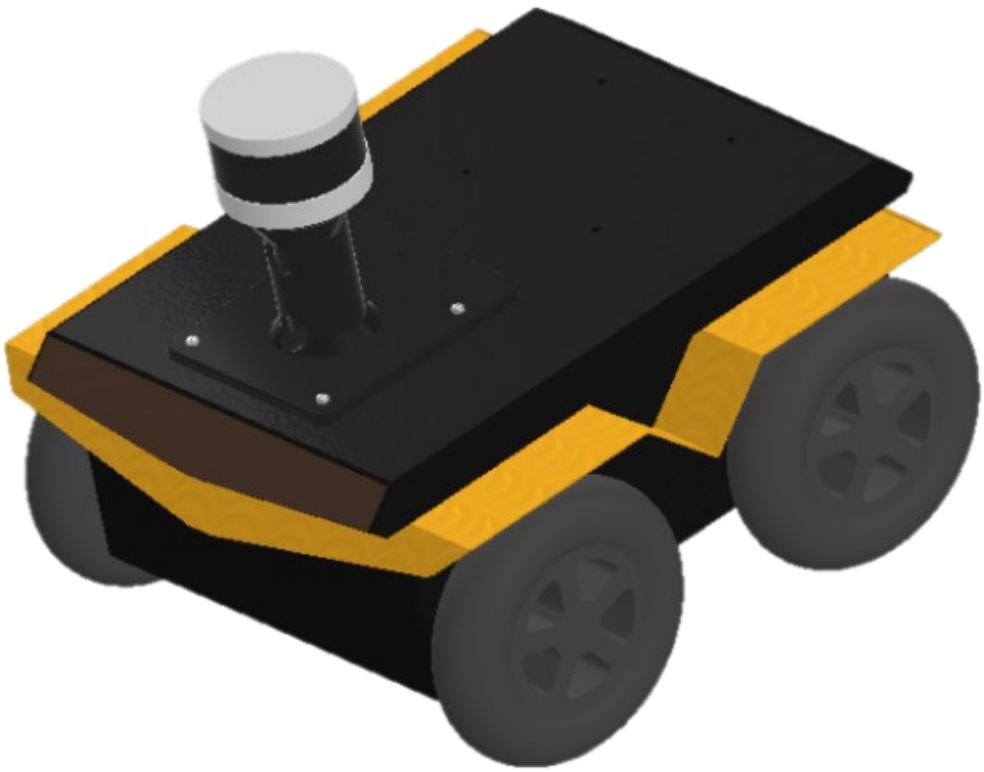
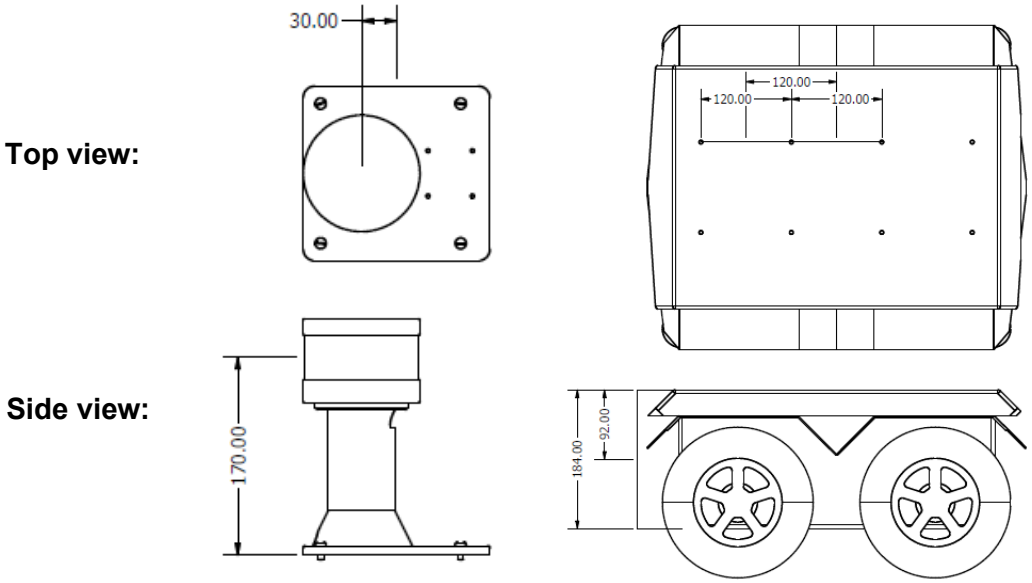


Figure 23 - Jackal robot - LiDAR placement



The following list describe key features of the Jackal and its components used in this project:

- The ClearPath Jackal robot has 2 mounting points to attach additional accessories. One of them is in the front and the second one is in the back of the robot top cover. Both of them have four threaded M5 holes in a square pattern of 120mm X 120mm.
- The VLP-16 3D LiDAR sensor, mounted on top of the robot, provides real-time point cloud data for obstacle detection. The sensor has a 360-degree field of view horizontally and a ±15-degree vertical field, with a range of up to 100 meters. This is ideal for precise mapping and navigation in environments like greenhouses, where rows of plants and obstacles need to be detected and avoided.
- The LiDAR base is printed as one piece and attached to the rear mounting point of the Jackal robot.

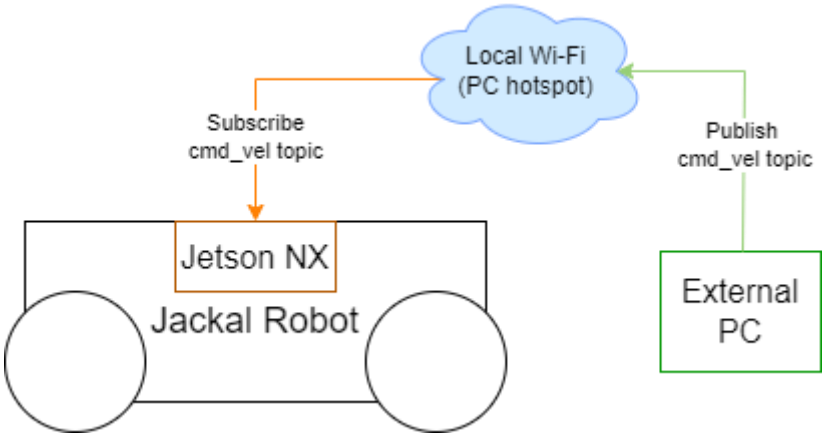
  It is composed from a flat cylinder with a height of 100mm. There is a extended circular pad with a hole where the LiDAR can be fixed using a standard camera bolt.
- Wheel encoders are integrated into the Jackal's wheels to provide odometry data, which is essential for tracking the robot's displacement over time.

5.3 System setup

The robot can be started by a power button located on the top cover on the sides. Once the system is booted, the Jetson computer automatically initiates ROS2. Jackal can be either controlled manually using a PlayStation controller, or by a ROS2 topic cmd_vel.

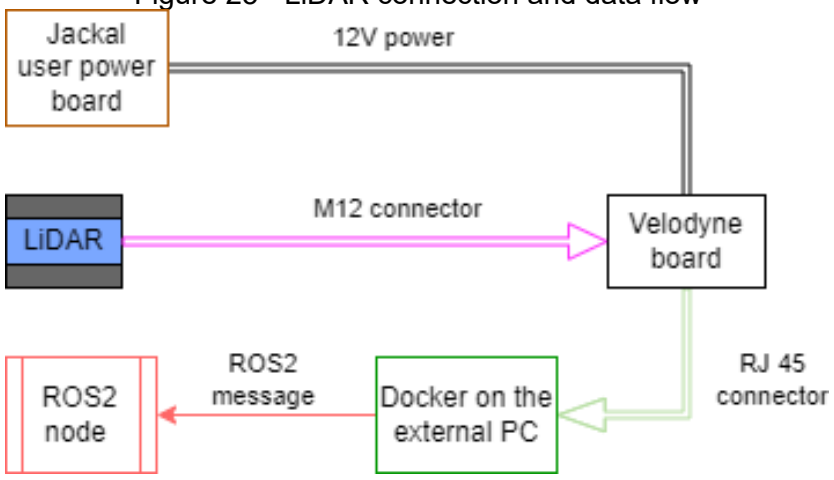
To send commands from an external computer, make sure that both devices are connected to the same Wi-Fi network, preferably by creating a hotspot on the computer. The connection is shown in Figure 24.

Figure 24 - PC/Jetson connection



The VLP 16 LiDAR is powered from the Jackal robot's internal source used for additional equipment. To receive the cloud point, the RJ45 cable must be connected between the processing box and the computer. Then it is followed by building the docker compose file from terminal. After the docker is initialized by the command *docker up*,. the VLP16 ROS2 package can be launched using a python file. The connection is shown in Figure 25.

Figure 25 - LiDAR connection and data flow



The installation and setup of the Gazebo simulation software is provided by a ClearPath in their documentation website (Clearpath Robotics, 2012). However, two ROS2 packages needs to be downloaded from the official documentation. One of them is used to start the simulation and second one is a library and configuration located in robot.yaml to adjust the robot model, attached sensors, such as the LiDAR with its position, as well as the world in which the simulation will be running.

Once everything is set up using a step guide in the attached GitHub, Appendix B:, the Gazebo software can be initiated by a specified command.

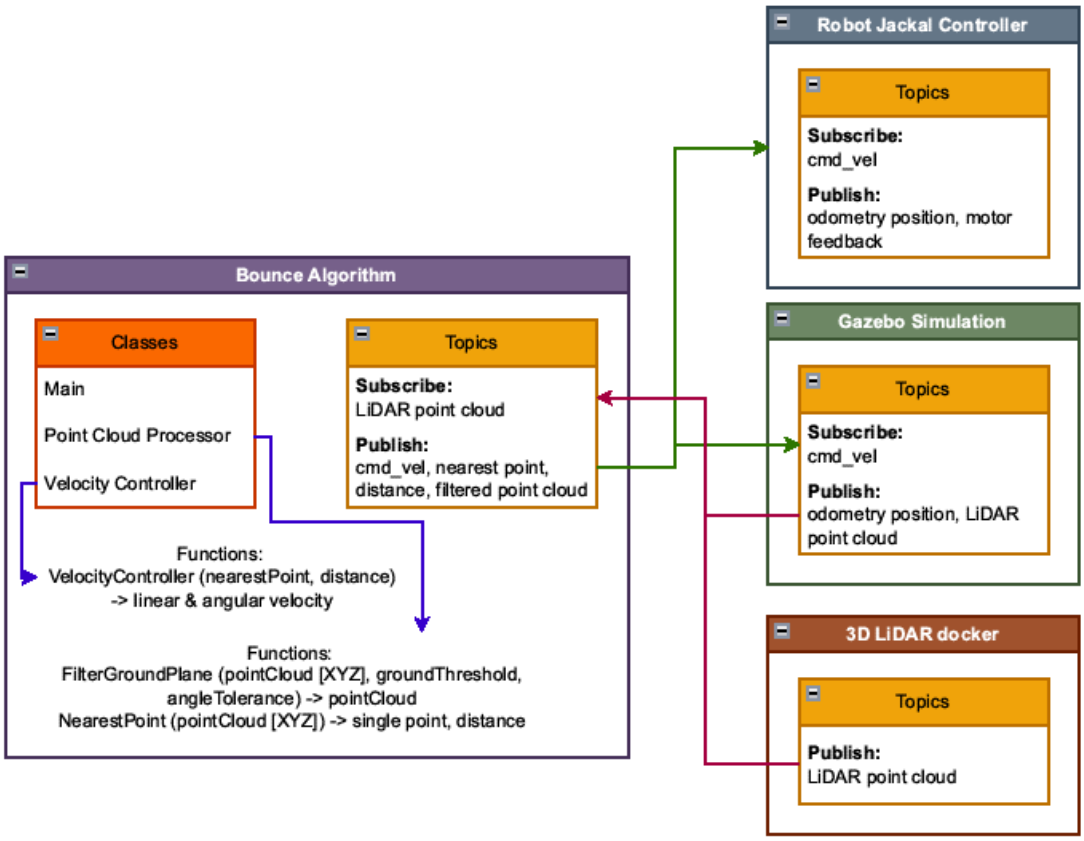
5.4 Software architecture

The software stack for this project is built on ROS 2 Humble. The robot's operation is divided into several key ROS nodes, which handle the sensor inputs, perform obstacle detection, and manage navigation. The main components include:

- A dedicated package “3D LiDAR docker” using a docker for handling data from the VLP-16 sensor, responsible for converting raw point cloud data into a ROS2 topic, so it is accessible to other nodes.
- Bounce Algorithm package processes the point cloud in dedicated classes to detect potential obstacles as well to avoid them
- Point Cloud Processor class goes through the point cloud to filter out the ground plane and find the nearest obstacle
- The Jackal’s motion is managed by a class called Velocity Controller, which takes output from the obstacle detection and adjusts the robot's trajectory.
- Robot Jackal Controller package is taken from ClearPath documentation and slightly adjusted so the robot can be controlled by linear and angular velocity under topic cmd_vel. This node is running on a Jetson computer inside the robot and is subscribed to topic which is being published from notebook via Wi-Fi network.

The whole architecture can be found in Figure 26 – ROS2 diagram.

Figure 26 – ROS2 diagram



5.5 Obstacle detection and avoidance algorithms

The core computing system is implemented as one ROS2 package and written all in C++ language. The program is well structured between main branch where all subscribers and publishers are initialized, main logic happens in separated classes. Furthermore, it has a launch python file linked to the yaml config where the set parameters are, so the thresholds for the ground filtration, robot velocity and switch for the simulation or real robot can be changed even when the code is running. This is further explained in Appendix A:.

5.5.1 Configuration files

CMake file includes libraries, source path for the executable node and package, as well as the launch file.

Parameters can be set in a config file together with their default values, which is then included in a launch file. The launch file needs to be specified while starting the package with a executable node. All used parameters and their units are listed below:

- Linear and Angular velocity are the maximum velocities the robot can reach when the program is running. [m/s] [rad/s]
- Slow and Stop threshold are used in the obstacle avoidance and are compared to the distance of the nearest point that is scanned. [m] [m]
- Obstacle zone Y is the limit which is applied when the robot reaches the obstacle pass the stop threshold. It is used to bound the sides of the robot. [m]
- Ground elevation is the angle for the vectors between first two points and XY plane [rad]
- Ground tolerance is the angle for the vectors between 3 consecutive points to be considered as a ground. [rad]
- Ground threshold filters all the ground points which are close to the plane that is fit through the calculated points based on angles. [m]
- Above filtration filters the points above the LiDAR with a given threshold. [m]
- Translation parameter translates coordinate system from LiDAR into center of robot. X,Y,Z [m]
- Simulation is a simple bool switch to easily move between simulated and real robot so the program does not have to be recompiled after each change. [true/false]

5.5.2 Main

In the constructor is initialized subscriber for the LiDAR point cloud and publishers for the filtered point cloud, nearest point which represents the obstacle, the topics are listed below:

- LiDAR subscriber is set depending on, whether it is running in a simulation or real environment. The topic is set either to the /a200_0000/points or /velodyne_points.
- Velocity publisher is set depending on, whether it is running in a simulation or real environment. The topic is set either to the /a200_0000/cmd_vel or /cmd_vel.
- Filtered point cloud and Nearest pointpublisher topics are used for the visualization and debugging of the algorithms.

. Last, but not least is the setup of the classes and declaration of the parameters.

5.5.3 Point cloud processor class

The public function Filter Ground Plane has as its own argument input point cloud and 3 parameters, ground tolerance, ground elevation, ground threshold and points above, which are passed into local variables. Then it calls 3 other functions to filter all the points.

First it estimates which points could belong to the ground. It is done by taking three points in a row from a vertical scan which has 16 laser beams, one vector is made from the first two points and second vector from the second and third point. Then the angle is calculated and if it is smaller than the parameter ground tolerance, points are considered as a part of ground. The value for the tolerance has been tested both for the simulation as well as real LiDAR and will be elaborated further on.

Second step is to fit a plane through the points with a RANSAC algorithm, which takes a random points from the input data and returns coefficients of the plane that covers most of the points.

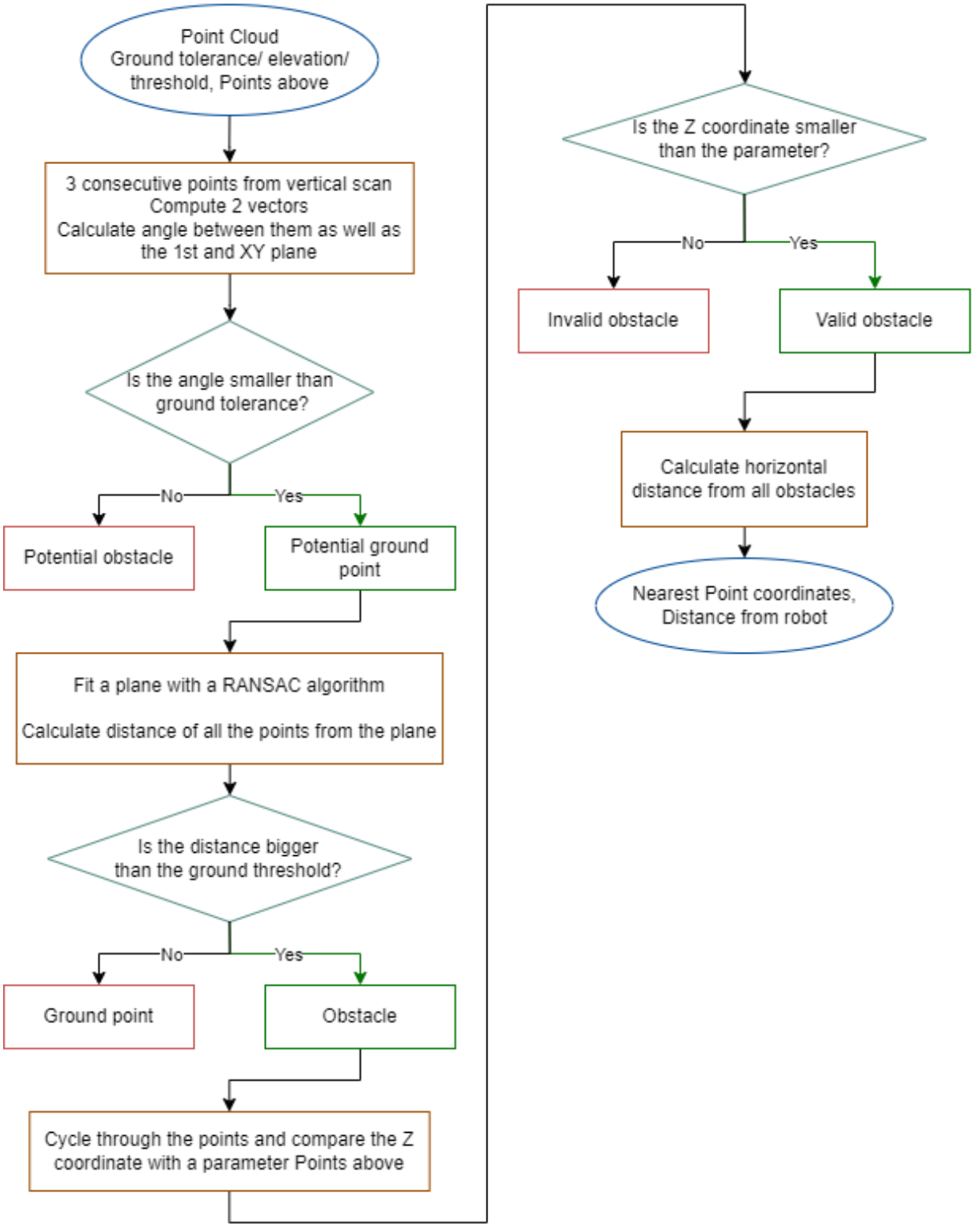
Last, but not least is to cycle through the whole point cloud and calculate a distance from the plane based on the coefficients, if it is closer than the given ground threshold it will be filtered from the points.

To make sure that the robot can pass under an obstacle, the points needs to be filtered from the upper side, as well. Based on the Z coordinate, which is also a vertical distance from the LiDAR, the points are removed based on the parameter above filtration. The value has been tested both for the simulation as well as real LiDAR and will be elaborated further on.

Get Nearest Point function is rather simple, as it cycles through the filtered point cloud and calculates horizontal distance for each point. Then it selects the closest point and returns its coordinates together with the distance.

The diagram of the whole class can be found in Figure 27.

Figure 27 - Point cloud processing diagram

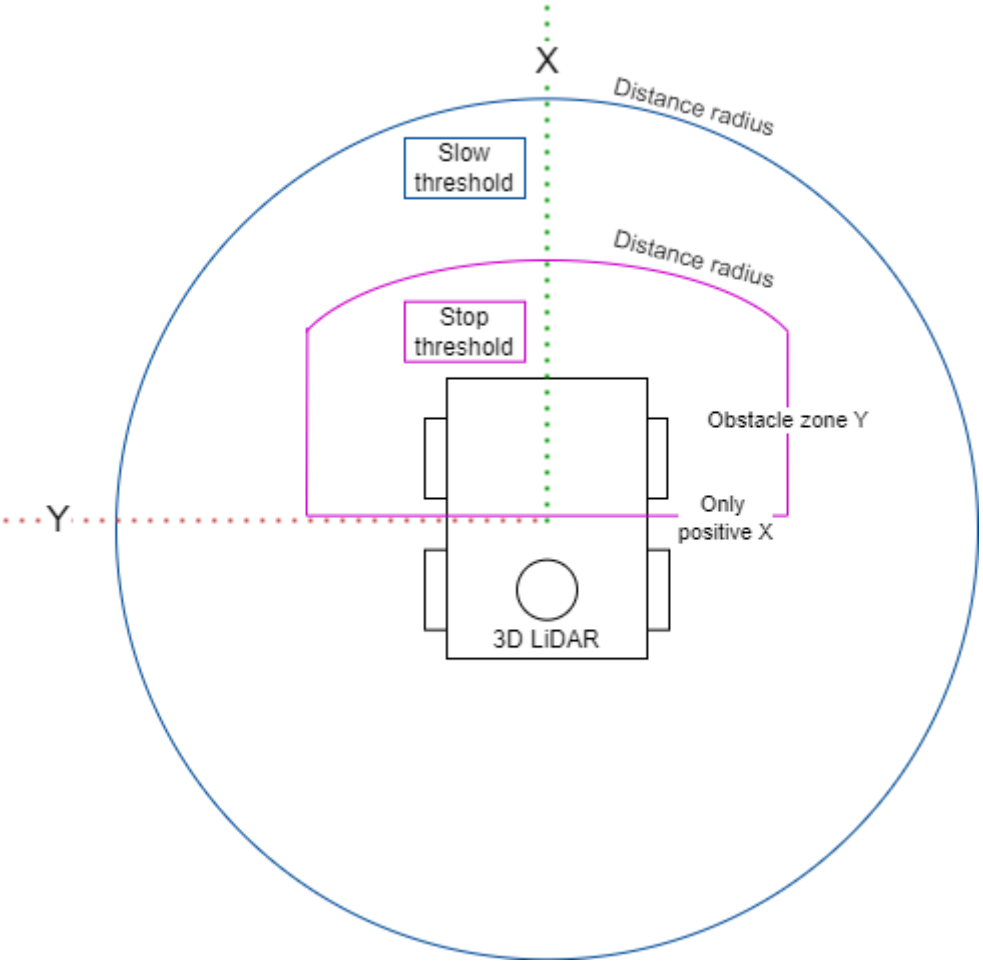


5.5.4 Velocity controller class

Based on the location and distance of the nearest point the robot will react to the obstacles.If the distance is smaller then the stop threshold, the obstacle is located in front of the robot and the Y value of the point is smaller than the obstacle zone, the robot will stop and turn to the opposite side compared to the detected obstacle. If the obstacle is sligthly further, then it will be caught in the next condition of the slow threshold. The robot will gradually slow down based on the distance and proportionally turn to the opposite side of the obstacle. And when no object is inside the envelope, the robot will go straight. The values of the thresholds have been tested both for the simulation, as well as real LiDAR, and will be elaborated further on.

The illustration of the obstacle avoidance behavior can be seen in Figure 28.

Figure 28 - Obstacle avoidance thresholds



5.6 Simulation in Gazebo

The simulation was carried out using Gazebo, a robotics simulation tool that is integrated with ROS 2. ClearPath Robotics provides a comprehensive emulation of the Jackal robot, allowing for sensor extensions, such as the 3D LiDAR, and visual data previews through RViz 2. For this project, the default warehouse environment in Gazebo was selected as the simulation setting. The robot was controlled using various methods, including WASD keys, a joystick, or by publishing velocity values to the cmd_vel topic, depending on the active configuration.

Gazebo also supports the insertion and movement of simple objects within the simulated environment, which proved useful for testing both the ground filtration and obstacle avoidance algorithms under different conditions. Various object placements allowed for an assessment of how the robot would react to real-world changes.

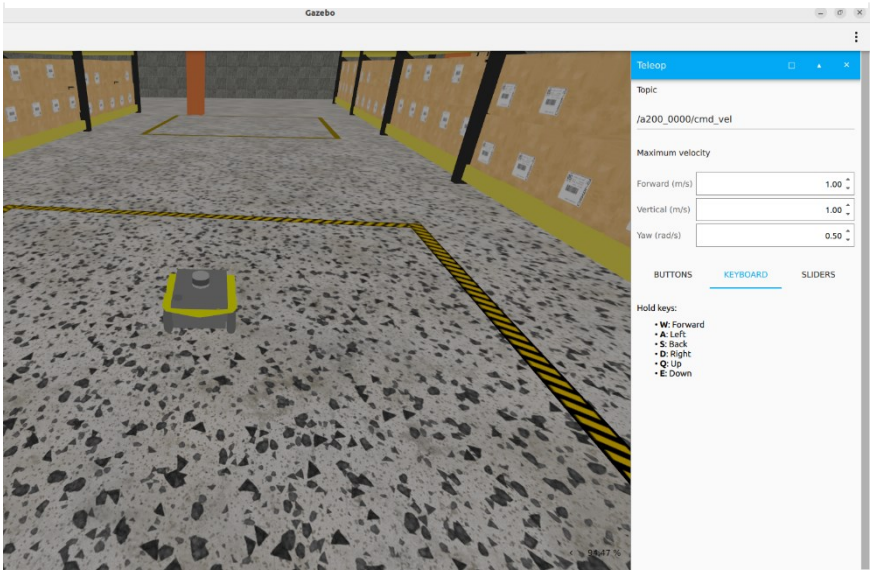
The whole simulation is futrher explained in a Appendix B: and Appendix C:.

Another important part was to test different values of the parameters. The ideal options are listed below:

- Linear and Angular velocity 1 [m/s] 0.5 [rad/s]
- Slow and Stop threshold 1.5 [m] 1.2 [m]
- Obstacle zone Y 0.9 [m]
- Ground elevation 0.1 [rad]
- Ground tolerance 0.1 [rad]
- Ground threshold 0.03 [m]
- Above filtration 0.2 [m]
- Translation X = 0,Y = 0, Z = -0 [m]

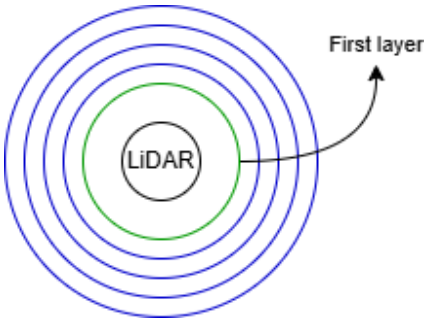
A photo from Gazebo software can be seen in the Figure 29. In the middle there is a robot with a 3D LiDAR mounted on top and sending real point cloud in the ROS2 message. On the right side of the screen, there is a side panel with a manual control of the robot. In the top, there is the name of the topic, then the preset linear and angular velocity, and below there are different options to control the movement of the robot

Figure 29 - Gazebo Simulation software



As illustrated in Figure 30, one of the minor issues found during the simulation was the structure of point cloud data from the 3D LiDAR. In a simulation the points are distributed by a rings and are saved into a single array. This makes it more difficult to cycle through by a vertical scans.

Figure 30 - Point cloud structure - simulation



To systematically access the points in the cloud, a consistent indexing is required. Given that the LiDAR sensor has R vertical rings (laser beams) and captures P points per horizontal rotation, each individual point can be accessed using the Equation 6.

Equation 6 - Point cloud index

$$index = v \times P + h$$

v represents the vertical ring index (ranging from 0 to $R - 1$), and h represents the horizontal step index (ranging from 0 to P−1). The total number of points per horizontal rotation is determined by the angular resolution of the sensor. For instance, with an angular resolution of 0.4°, the total number of horizontal steps in a full 360-degree scan is calculated in Equation 7.

Equation 7 - Points per ring

$$P = 360/0.4 = 900$$

Each point at a given horizontal step hhh and vertical ring vvv is indexed using the formula v×P+hv \times P + hv×P+h, ensuring correct alignment of points in both dimensions.

To classify ground points, a filtering approach is applied by evaluating consecutive vertical points at a fixed horizontal position. Specifically, for each horizontal step h, three vertically aligned points are extracted from the point cloud using the Equation 8

Equation 8 - Consecutive points

$$P_1 = input\_cloud[v \times P + h]$$
$$P_2 = input\_cloud[(v + 1) \times P + h]$$
$$P_3 = input\_cloud[(v + 2) \times P + h]$$

where P_1, P_2 and P_3 represent three consecutive points in the vertical scan at the same horizontal angle. To ensure the algorithm remains within valid indexing limits, the vertical loop runs only up to $R - 3$ to prevent accessing out-of-bounds data when evaluating consecutive vertical points.

In the Figure 31 and Figure 32 there is vizualization of the important data, in the middle of the pictures is a model of the robot with mounted 3D LiDAR on top as well as coordinate system for each object. The red points represent a output from the LiDAR, white points are the detected obstacles, green sphere is a nearest point from the robot and the white squares are used for a debug to make sure that the angle calculations of vectors are correct.

Figure 31 - RViz 2 preview 1

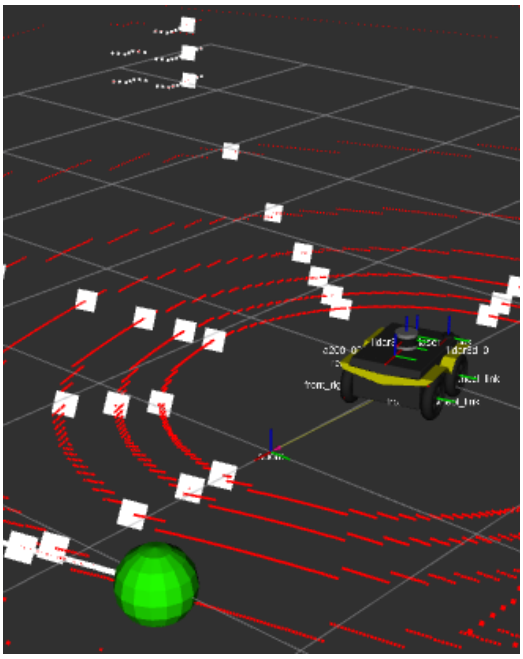
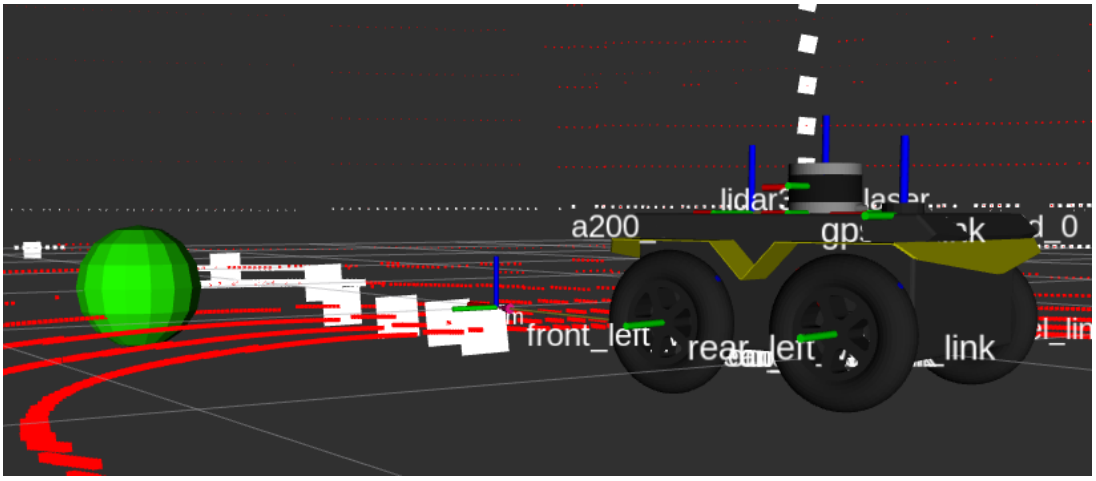


Figure 32 - RViz 2 preview 2



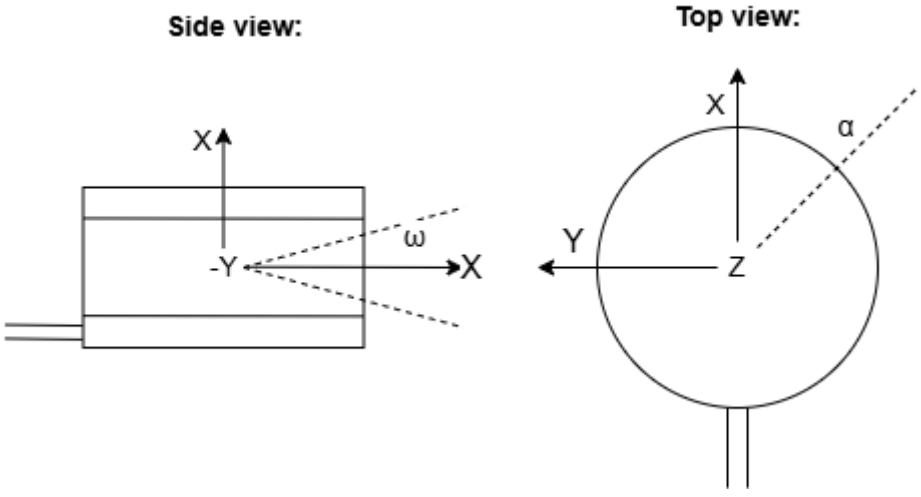
5.7 ClearPath robot test

The real-world testing was conducted using the ClearPath Jackal robot equipped with a 3D LiDAR VLP 16 and onboard computer nVidia Jetson Xavier. The primary objective was to validate the obstacle detection and ground filtration algorithms under real conditions and to compare the results with the Gazebo simulation. The robot was operated using multiple control methods, including a PlayStation controller and direct velocity commands published to the ROS 2 cmd_vel topic.

During the tests, the algorithm proved to be reliable in detecting both ground surfaces and obstacles. It was able to accurately perceive and navigate around objects, including a small elevation of minimum 5 cm. If the obstacle is smaller the robot can drive over it. However, due to the LiDAR's limited vertical field of view (±15 degrees) and a maximum detection range of 90 cm, certain obstacles, such as low steps, became undetectable once the robot approached them too closely. Another problem found during the tests is that the robot is not able to go through a narrow corridor smaller than 2 meters, because the cloud point data are processed in a real time without any further storing of the nearest obstacle

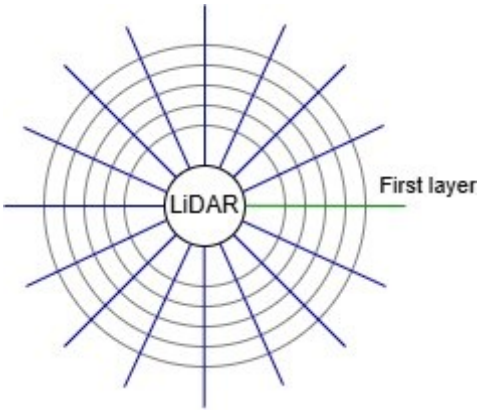
One minor issue was found regarding the orientation of the LiDAR sensor. The X-axis of the sensor was found to be pointing in the opposite direction of the cable, contrary to the specifications provided in the technical documentation. It was solved by simply turning the LiDAR by 90 degrees so the X axis is pointing forward, rather than rotating the whole matrix system. This is shown in Figure 33.

Figure 33 - LiDAR matrix orientation



Compared to the simulated LiDAR the VLP 16 has an advantage. Although the point cloud is also stored in a flat array, the points are saved by the vertical scans, so it made the point indexing a lot easier. The Figure 34 shows an example to better understand the structure.

Figure 34 – Point cloud structure – VLP 16



The data is often organized sequentially based on the sensor's internal scanning pattern, rather than a strict two-dimensional grid format. A LiDAR sensor captures a 360-degree horizontal scan, divided into multiple azimuthal steps, with each step containing multiple vertical points from different laser beams.

The total number of horizontal steps, denoted as P, depends on the sensor's horizontal resolution. The vertical beams, referred to as rings, are indexed from 0 to $R - 1$, where R represents the number of lasers in the LiDAR unit. The loop structure in the provided code iterates through the horizontal steps first and then progresses through the vertical laser rings. However, unlike the simulated case where the index is calculated as $v \times P + h$, in Equation 9 the points are accessed using a simpler linear traversal.

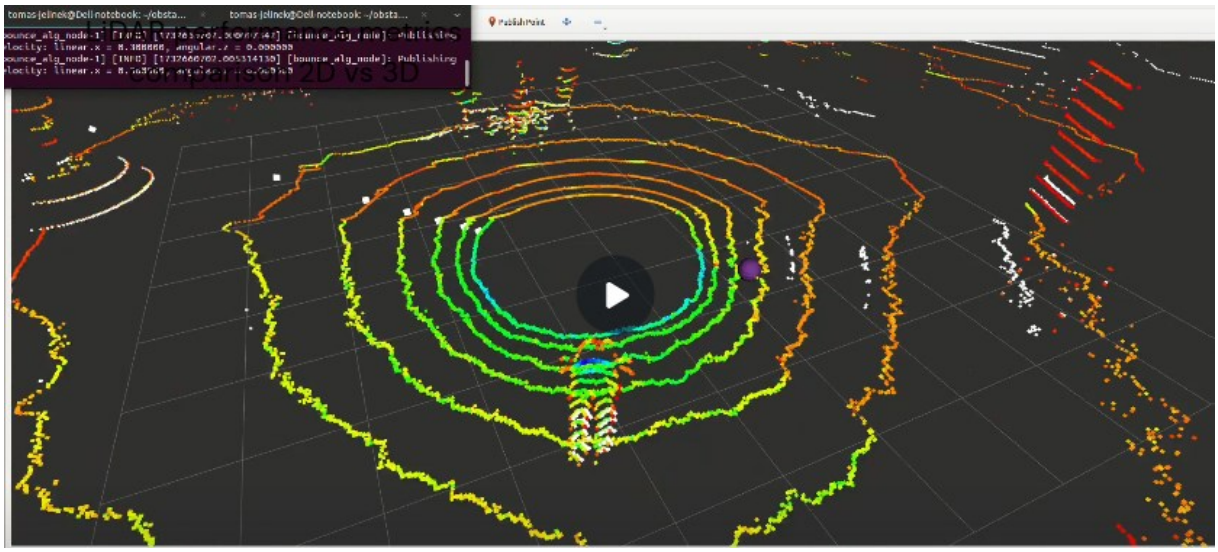
Equation 9 - Consecutive points indexing

$$P_1 = input\_cloud[h + v]$$
$$P_2 = input\_cloud[h + v + 1]$$
$$P_3 = input\_cloud[h + v + 2]$$

One key challenge in real LiDAR data is the presence of invalid or missing points, often represented as Not a Number (NaN) values. These occur due to reflections, occlusions or due to the fact, that the object is too close to the LiDAR. To account for this, the algorithm first checks if any of the selected points contain NaN values and then skips those points

The Figure 35 shows that the ground detection still has its flaws even in a fairly smooth terrain.

Figure 35 - Ground detection test in a grass



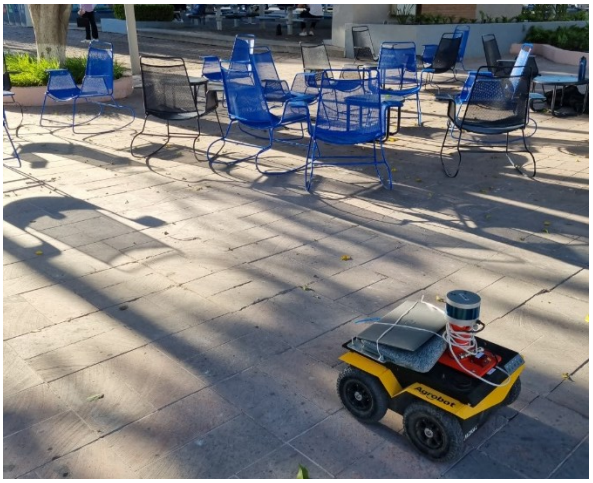
In general, it took many trial and error cycles to find the right combination of parameters, and in the end, it was only tested on a flat ground to minimize the issues it had.

Key parameter values used during the real-world tests were as follows:

- Linear and Angular velocity 1 [m/s] 0.5 [rad/s]
- Slow and Stop threshold 1.5 [m] 1.2 [m]
- Obstacle zone Y 0.9 [m]
- Ground elevation 0.15 [rad]
- Ground tolerance 0.15 [rad]
- Ground threshold 0.03 [m]
- Above filtration 0.4 [m]
- Translation X = 0.15,Y = 0, Z = -0.2 [m]

Figure 36 presents a photo from the real world testing of the Jackal robot, demonstrating the effective performance of the obstacle avoidance system.. The testing established that the robot can pass under a obstacle, but it can also see very small bumps and detect if it can ride over them or not, within 3 cm of tolerance.

Figure 36 - Photo from testing



These real-world tests confirmed that the simulation in Gazebo provided a reliable approximation of the robot's behavior, with some differences due to sensor limitations and real-world environmental factors. Future tests will focus on refining the detection algorithms to better handle low-profile obstacles and to compensate for the LiDAR orientation discrepancy.

6 Results and analysis

This section presents the evaluation of the system's performance based on key metrics, outcomes, and a discussion of findings.

6.1 Performance metrics

The project was deemed successful if the following criteria were met:.

- The filtration algorithm can detect the floor plane without any errors
- The robot is capable of avoiding obstacles reliably, and it can react to a dynamically changing environment.
- It can be demonstrated that 2D LiDAR would not be feasible in the same application.

6.2 Simulation results

As discussed, the simulation was performed in Gazebo, where the ground filtration and obstacle avoidance algorithms were thoroughly tested. In general, the filtration algorithm worked without issues, and the robot was able to avoid most obstacles. However, the robot occasionally became stuck when navigating tight corners, where the LiDAR could not detect all points, leaving gaps in the point cloud. This led to inaccurate calculation of the nearest point, affecting the obstacle avoidance behavior.

Another key finding was that the robot could pass under certain obstacles, like tables, if the vertical clearance was sufficient. This highlights that the obstacle detection works as intended.

6.3 ClearPath robot results

Testing Jackal confirmed the obstacle detection algorithm's reliability, successfully detecting obstacles and small bumps above 5 cm. However, due to the LiDAR's ±15-degree vertical field of view, low steps became undetectable at close range. Narrow corridors under 2 meters also caused navigation issues due to real-time point cloud processing. A minor LiDAR orientation error was corrected, and after tuning, the system performed well on flat terrain, though further improvements are needed for complex environments.

6.4 Discussion of findings

The results overall indicate that the minor error rate in obstacle detection could be mitigated by incorporating additional logic. For example, the robot could analyze data from the last five seconds when approaching an obstacle. By comparing the direction of nearby obstacles over time, the robot could adjust its movement, reversing slightly and turning to avoid getting stuck.

The issue with detecting ground in an uneven terrain could be solved by fitting a polynom instead of a straight plane. The polynom can have multiple curves and fit a lot better into the point candidates.

7 Conclusion

This thesis has examined the development off an obstacle avoidance system using 3D LiDAR for the Jackal robot. The system was tested through simulation in Gazebo as well as with the Jackal robot, where the robot demonstrated its ability to navigate and avoid obstacles in a dynamic environment. Although there are some minor issues with the system's performance in tight spaces, the overall results confirm the feasibility of the approach.

8 Future works

Future developments should focus on reducing the error rate in obstacle avoidance. Also, the robot currently relies on a reactive "bouncing" behavior, navigating between obstacles without a defined destination. A key improvement would be to integrate a GPS sensor, enabling the robot to determine its location relative to the environment. This would allow the robot to follow a specific path toward a goal while still reacting to changing obstacles along the way.

Anothet major part of the project is an optimilization that has not yet been conducted. Based on a observation, the bottleneck is in the obstacle detection as the distance is calculated for every single point. It would be sufficient to remove points that are outside of a defined envelope.

Last key area is filtering out ground floor points, as currently the system fits a straight plane through the potential points which has its limitation in an uneven terrain. Therefore, it would be beneficial to use polynomial instead.

9 References

Clearpath Robotics. (2012). *ClearPath Robotics documentation*. Clearpath Robotics . Retrieved 17 October 2024, from https://www.clearpathrobotics.com/assets/guides/kinetic/jackal/simulation.html

ClearPath Robotics. (2020). *Jackal UGV - Small Weatherproof Robot - Clearpath*. Retrieved 4 November 2024, from https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

Dell. (n.d.). *Dell Latitude 5420 Specifications*. Retrieved 5 November 2024, from https://www.dell.com/support/manuals/en-uk/latitude-5420-laptop/5420_sp14_setupspecs/notes-cautions-and-warnings?guid=guid-5b8de7b7-879f-45a4-88e0-732155904029&lang=en-us

Docker. (2013). *What is Docker?* Docker. Retrieved 10 November 2024, from https://docs.docker.com/get-started/docker-overview/

Dubois, A. S. (2022, December). *Basics of 3D LiDAR Technology*. Research Journal. Retrieved 5 November 2024, from https://insights.outsight.ai/understanding-the-basics-of-3d-lidar-technology/

Lima, T. A., Davi Do Nascimento Forte, M., Nogueira, F. G., Torrico, B. C., & De Paula, A. R. (2016). Trajectory tracking control of a mobile robot using lidar sensor for position and orientation estimation. *2016 12th IEEE International Conference on Industry Applications, INDUSCON 2016*. https://doi.org/10.1109/INDUSCON.2016.7874573

Luo, N., Yu, H., Huo, Z., Liu, J., Wang, Q., Xu, Y., & Gao, Y. (2021). KVGCN: A KNN Searching and VLAD Combined Graph Convolutional Network for Point Cloud Segmentation. *Remote Sensing 2021, Vol. 13, Page 1003*, *13*(5), 1003. https://doi.org/10.3390/RS13051003

Mendoza, A. G., & Flatscher, T. (2024). *GitHub - Alfredo Gomez velodyne16-ros2*. GitHub. Retrieved 10 November 2024, from https://github.com/Freddy220103/velodyne16-ros2?tab=readme-ov-file

Miądlicki, K., Pajor, M., & Mateusz, S. (2017). Ground plane estimation from sparse LIDAR data for loader crane sensor fusion system. *2017 22nd International Conference on Methods and Models in Automation and Robotics, MMAR 2017*, 717–722. https://doi.org/10.1109/MMAR.2017.8046916

nVidia. (2020). *Jetson Xavier NX Developer Kit*. NVidia Developer Data Sheet. Retrieved 7 October 2025, from https://developer.download.nvidia.com/assets/embedded/secure/tools/files/jetpack-sdks/jetpack-4.4-ga/Jetson_Xavier_NX_Developer_Kit_User_Guide.pdf?__token__=exp=1742205352~hmac=9ede48cf8c787afb9e265f746ffa483cd3afcf93f10c79cc7bd184bb6953f309&t=eyJscyI6ImdzZW8iLCJsc2QiOiJodHRwczovL3d3dy5nb29nbGUuY29tLyJ9

Petrovskaya, A., & Thrun, S. (2009). Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, *26*(2–3), 123–139. https://doi.org/10.1007/S10514-009-9115-1/METRICS

Reyes, P. (2024). *GitHub - PaoloReyes/Jackal-Jetson-Xavier-NX-SDK: Jackal Image for ROS2 Humble in Jetson Xavier NX*. GitHub. Retrieved 10 September 2024, from https://github.com/PaoloReyes/Jackal-Jetson-Xavier-NX-SDK

Saha, A., & Dhara, B. C. (2024). 3D LiDAR-based obstacle detection and tracking for autonomous navigation in dynamic environments. *International Journal of Intelligent Robotics and Applications*, *8*(1), 39–60. https://doi.org/10.1007/s41315-023-00302-1

Susnea, I., Minzu, V., & Vasiliu, G. (2009). *Simple, real-time obstacle avoidance algorithm for mobile robots*. Conference: The 8th WSEAS International Conference on Computational Intelligence Man-Machine Systems and Cybernetics (CIMMACS'09). Retrieved 9 October 2024, from https://www.researchgate.net/publication/228955195_Simple_real-time_obstacle_avoidance_algorithm_for_mobile_robots

Tellez, R. (2019). *ROS for Beginners: What is ROS? - The Construct*. Retrieved 5 November 2024, from https://www.theconstruct.ai/what-is-ros/

Velodyne. (n.d.). *Velodyne 16 3D LiDAR - Data sheet*. Retrieved 15 October 2024, from https://hexagondownloads.blob.core.windows.net/public/AutonomouStuff/wp-content/uploads/2019/05/Puck__Datasheet_whitelabel.pdf

Wind River. (2020). *What Is ROS2?* Wind River. Retrieved 5 October 2024, from https://www.windriver.com/solutions/learning/ros2

Yalcin, O., Sayar, A., Arar, O. F., Akpinar, S., & Kosunalp, S. (2013). Approaches of Road Boundary and Obstacle Detection Using LiDAR. *IFAC Proceedings Volumes, 46(25)*, *1*(n.), 211–215. https://doi.org/10.3182/20130916-2-tr-4042.00025

Yin, H., Yang, X., & He, C. (2016). Spherical coordinates based methods of ground extraction and objects segmentation using 3-D LiDAR sensor. *IEEE Intelligent Transportation Systems Magazine*, *8*(1), 61–68. https://doi.org/10.1109/MITS.2015.2494079

Appendix A: Bounce Algorithm ROS 2 package

https://github.com/Tomas-412/BounceAlgorithm

Appendix B: ClearPath Jackal simulation ROS 2 package

https://github.com/Tomas-412/clearpath_sim_ws

https://github.com/Tomas-412/clearpath

Appendix C: ClearPath driver ROS 2 package

https://github.com/PaoloReyes/Jackal-Jetson-Xavier-NX-SDK

Appendix D: 3D LiDAR docker emulator

https://github.com/Freddy220103/velodyne16-ros2