Ostfalia
Hochschule für angewandte Wissenschaften

Fakultät Informatik

Entwurf und Implementierung einer selbstfahrenden Plattform mit Autokollision auf der Basis eines Lidar-Sensors

Yousef Al swidan
Matrikel-Nr. 70476722

Masterarbeit im Studiengang Infrmatik
zur Erlangung des akademischen Grades:
Computer information system

Ostfalia Hochschule für angewandte Wissenschaften

1. Prüfer: Prof. Dr.-Ing. Jürgen Kreyssig
2. Prüfer: Prof. Dr.-Ing. Ulrich Klages

Abstract

The development of self-driving platforms is progressing rapidly and continuously, with the potential to significantly impact various aspects of transportation. These developments could enhance the comfort, performance, and safety of transportation while reducing human error [15].

The objective of this project is to construct a self-driving platform using a Raspberry Pi 5 connected to a camera model Picam 2 and an LiDAR A1M8 from Slamtec [3]. The primary objectives are to understand the behavior of the LiDAR and develop a collision avoidance functionality based on the LiDAR. Additionally, the project seeks to integrate this functionality with object detection to create a robust self-driving platform.

The Raspberry Pi 5 serves as the central processing unit, coordinating inputs from the camera module and the LiDAR A1M8. The camera module provides real-time visual data for object detection, while the LiDAR A1M8 enhances the system's spatial awareness by delivering accurate distance measurements. The ultimate goal of this project is to demonstrate a functional prototype with practical applications in robotics and autonomous systems, highlighting the potential for future advancements in autonomous vehicle technology.

Contents

1 Introduction

The ethical considerations surrounding autonomous vehicles are crucial, especially given the importance of traffic rules in their operation. It's essential to establish a clear ethical framework to guide the development of these vehicles. This framework should address both the technical aspects and the ethical implications of their operation. By creating comprehensive ethical guidelines, we can ensure that autonomous vehicles are developed in a way that prioritizes safety and ethical behavior. This approach will benefit not only the vehicles but also the people and communities they interact with.

In the development process of autonomous platforms, ethics and values take precedence, followed by legislation and standardization. These elements are continually monitored and validated to ensure they align with ethical standards. This highlights the foundational role of ethics in creating responsibly developed emerging technology. By prioritizing ethics, we ensure that the technology aligns with human values and is developed responsibly [10].

The implementation of safety standards can significantly enhance safety levels. To ensure these standards function optimally, it is crucial to have a comprehensive understanding of the surrounding environment, enabling the platform to accurately analyze and make appropriate decisions. For the machine to perceive and understand its environment, it must be provided with sufficient and diverse information through various sensory inputs. Once the environment is understood, the machine can make the most suitable decision.

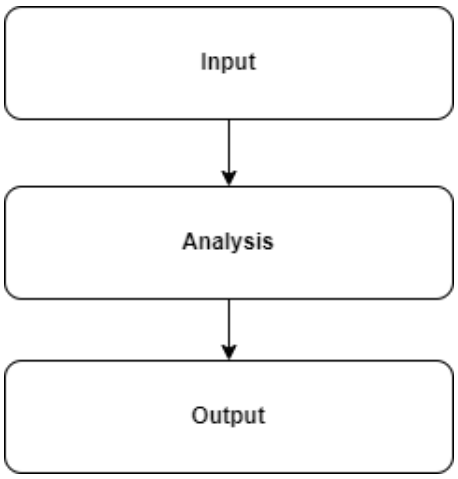


Figure 1: System implementation process

The categorization of input data in Figure [1] illustrates two primary types: LiDAR inputs and camera inputs. LiDAR inputs provide distance measurements across a 360-degree field, whereas camera inputs consist of video frames. The analysis process, relying on these inputs, encompasses three primary tasks: collision avoidance, lane keeping, and object detection in front of the platform. Based on the analysis results, decisions are made to ensure the platform operates within a safe environment. Following this, the system implementation process can be detailed as shown in Figure [2].

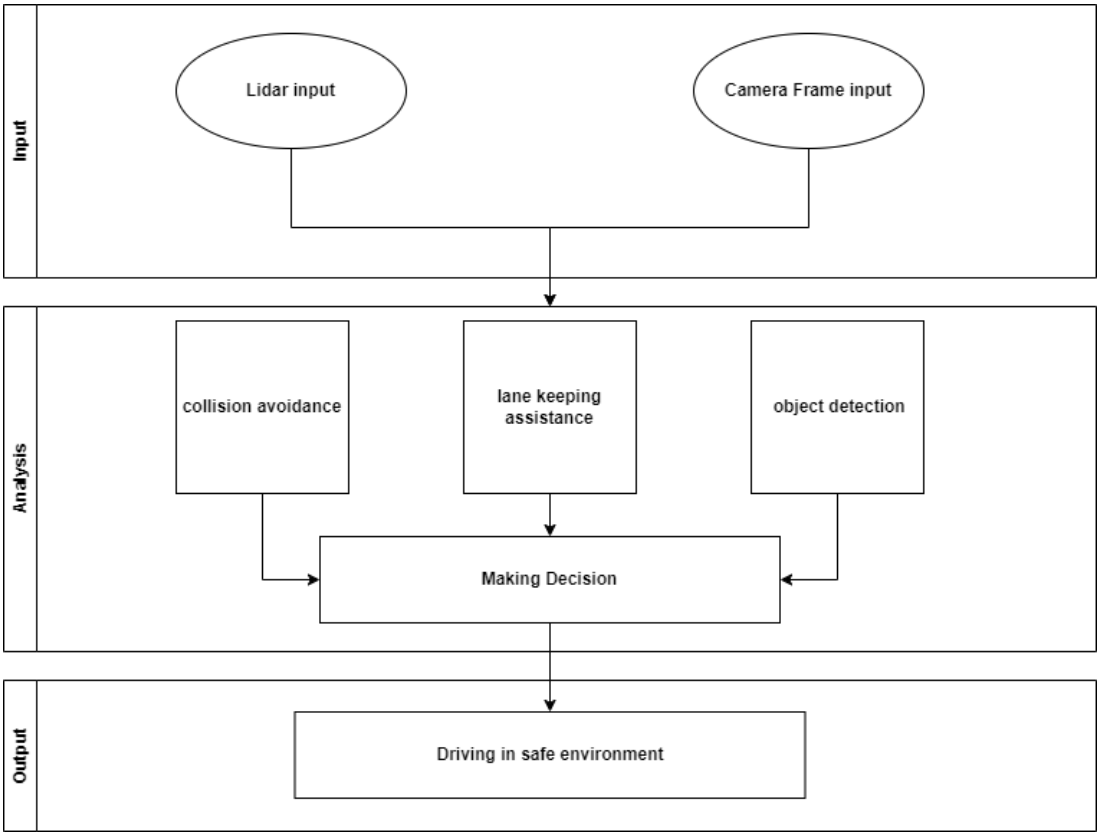


Figure 2: System implementation process with details

During the implementation phase, functionality was developed across separate modules to ensure their independent utilization. Initially, the integration of LiDAR data was prioritized to proactively avoid collisions. This was then combined with camera inputs to enable the platform to navigate paths using lane detection. In the final phase, the system was designed to enhance safe driving by incorporating features from the previous stages, including object detection.

the Scenarios can be summarized as follows:

- **Scenario 1: Complete control via another Platform**
  - In this scenario, the driving command is initiated from a server, instructing the platform to execute simple driving directions, such as moving forward, backward, left, and right. The objective is to assess the reliability and accuracy of the LiDAR data.

Figure 3: control via another Platform

- **Scenario 2 : Autonomous driving based on data from LiDAR and camera**
  - the second scenario, the primary objective is to utilize image processing capabilities to maintain the platform's position within a designated lane. To achieve this, a multi-processor operation was implemented in parallel using asynchronous I/O, thereby minimizing the CPU burden during image processing. Additionally, the LiDAR data was processed in a near-synchronous manner.

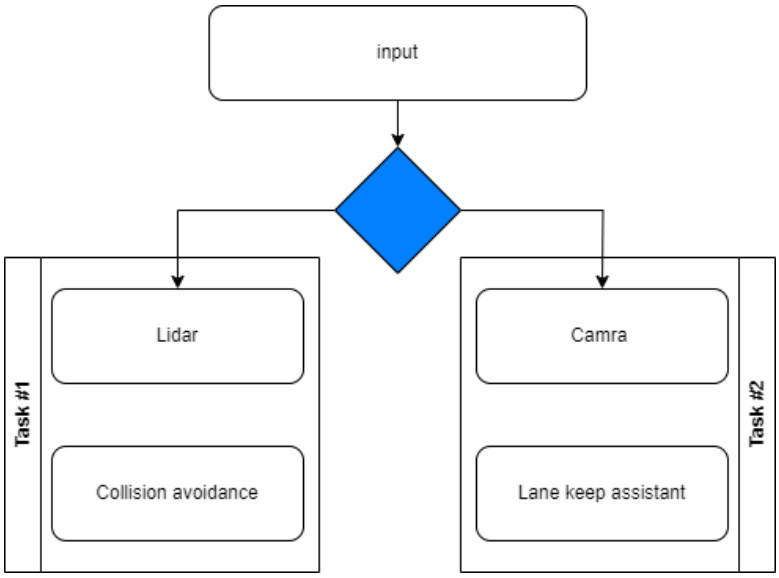


Figure 4: driving based on data from LiDAR and camera

- **Sscenario 3 : Autonomous driving based on data from LiDAR and camera including object detection**
  - This scenario presents the same characteristics as Scenario 2, including the object detection.

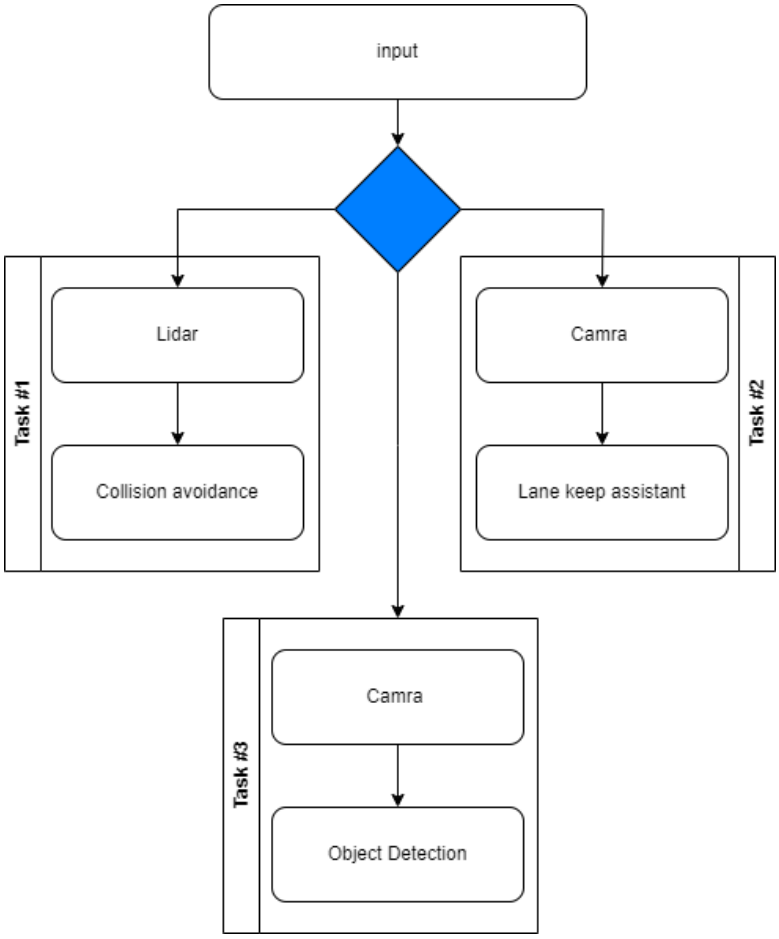


Figure 5: driving based on data from LiDAR and camera including object detection

2 Related Work

The collision prevention system comprises a suite of technologies that must work in concert to prevent collisions. After analyzing the environment, action should be taken to keep the platform passengers safe from any collision. Ultrasonic wave-based proximity sensors are often used for this purpose, but the limitations of environmental coverage, inaccurate measurements, and inflexible scanning methods are some of the reasons for abandoning them as the main element. With the advancement of technology, it has become possible to use more complex components. The camera, with its advantage of being able to capture two-dimensional objects, detect color information, and its very high resolution, could also be used frequently in driving safety systems. However, they have the disadvantage of not providing depth information about the object and are affected by strong surrounding light, such as the sun, rain, dust, etc [17].In the automotive manufacturing market, the complexity principle is always increasing, leading to the design and development of more complex integrated systems to avoid collisions. Over time, collision prevention systems have increasingly incorporated Light Detection and Ranging (LiDAR) technology, due to its high precision and rapid scanning capabilities. An important factor in LiDAR performance is the scan speed. When the scanning rate of a LiDAR system exceeds the relative motion of a moving platform, the motion-induced distortions in the scans are minimal and can often be ignored. However, if the scanning rate is slower than the platform's motion, significant distortions arise due to the slower scan speed. This issue is particularly noticeable with 2D LiDAR systems, where one axis typically operates at a slower speed than the other [6]. In the field of autonomous vehicle research focused on object and distance detection, two common types of LiDAR are utilized: 2D LiDAR (2-Dimensional Light Detection and Ranging) and 3D LiDAR (3-Dimensional Light Detection and Ranging)[16]. Combining 2D LiDAR with cameras can be advantageous because 2D LiDAR offers a simpler, lower-cost solution suitable for basic applications, while 3D LiDAR, though more advanced, is a higher-cost option designed for more complex requirements.For most automobiles, understanding their own dynamic status is essential for processing interactions and responses to their surroundings. The status can be described in various dimensions, primarily including velocity [14] and the detection of objects in the environment.To enhance obstacle detection in autonomous vehicles, integrating distance calculations from LiDAR with image data from camera sensors can be a viable approach. This combined data can potentially improve the navigation capabilities of autonomous vehicles [20]. As previously noted, LiDAR can be used for distance measurement, while cameras provide visual sensing of objects in the environment. This visual data can be leveraged through machine learning techniques. One of the most common frameworks for this purpose is TensorFlow, which can process camera data into relevant information to assist platforms in driving autonomously.TensorFlow can be utilized both for research purposes and for deploying machine learning systems into production across various fields, including computer vision, robotics, and more. It serves as an interface for implementing machine learning algorithms[5].The outcomes of the machine learning trained model can be integrated with other results to let the decision-making process, enabling the platform to enhance safety and prevent accidents.

3 Hardware

3.1 Assembly

Raspberry Pi 5 (central component):
Functions as the primary controller for the system, responsible to management of inputs and outputs from connected devices.

Component	Specification
Processor	Arm Cortex A76 64-bit Arm Cortex A76
RAM	8GB LPDDR4X SDRAM
Networking	Gigabit Ethernet, 2.4GHz/5GHz dual-frequency 802.11ac Wi-Fi Bluetooth 5.0, BLE
USB Ports	USB 3.0 x 2, USB 2.0 x 2
Display Outputs	micro HDMI x 2 (support 4Kp60)
Storage	MicroSD card slot
Expansion and I/O	40PIN GPIO header 2 × 4-lane connectors for camera or display peripherals
Power Supply	USB Type C (5V/5A), Power button
Additional Features	PoE cable ,PCIe expansion connector

Table 1: Specifications of the Raspberry Pi 5

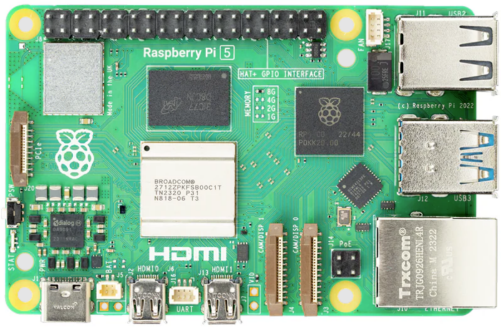


Figure 6: Raspberry pi 5
Source: `https://www.mouser.de/images/marketingid/2023/microsites/0/PI_5_TOP%20SC1111-SC1112.png`

4 x DC Motors:
The platform's mobility is enabled by four DC motors, which drive the wheels and apply the platform movement.

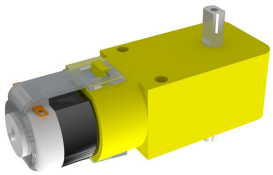


Figure 7: DC Motor
Source :https://d2t1xqejof9utc.cloudfront.net/screenshots/pics/dfb9790f1ecd25dda21693487a684cf0/original.png

2 x Motor Drivers modules:
two L298N motor driver modules , which control the power supplied to the DC motors from the Raspberry Pi .Each motor driver controls two DC motors.

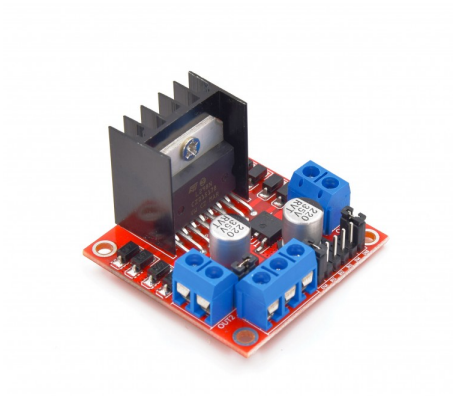


Figure 8: Dual H-Bridge Motor Driver L298N.
Source:
https://www.smart-prototyping.com/image/cache/data/9_Modules/101861%20LN298N%20dual%20H-bridge%20driver%20motor/dual-h-bridge-motor-driver-l298n-44325-750x750.jpg

LiDAR Model:
A LiDAR A1M8 from Slamtec hardware device is used for distance measurement and object detection.

Figure 9: LiDAR A1M8
Source :https://www.slamtec.ai/wp-content/uploads/2023/10/cropped-RPLIDAR-A1-2.jpg

Camera Model:
A Raspberry Pi Camera Module 2 connected to the Raspberry Pi, used for visual input, image processing and Machine learning tasks.

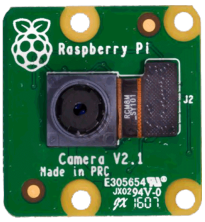


Figure 10: Raspberry Pi Camera V2 with accessories.
Source: https://market.samm.com/raspberry-pi-camera-v-2-raspberry-pi-camera-and-accessories-raspberry-pi-447-23-B.png

Power Bank:
The power source for the Raspberry Pi, providing portable power supply to the entire system.

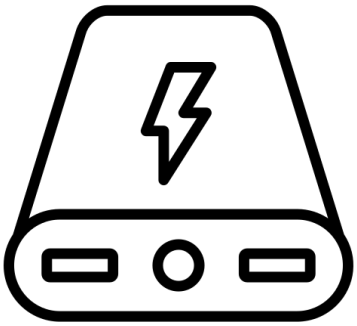


Figure 11: Power Bank
Source :https://www.flaticon.com/free-icon/power-bank_17392483?term=powerbank&page=1&position=88&origin=search&related_id=17392483

Connection:
The Raspberry Pi is connected to the motor drivers through its GPIO pins. Specific GPIO pins are used to control the inputs of the motor drivers.
The wiring from the **motor drivers** to the DC motors includes connections for power (VCC), ground (GND), and signal wires for speed control.
The LiDAR sensor is connected to the Raspberry Pi through a USB interface module.
The camera module is connected to the Raspberry Pi via the camera interface (CSI) port, which is a dedicated connector for camera modules on the Raspberry Pi. This allows the Raspberry Pi to capture images or video from the camera.
The power bank supplies the required energy to the Raspberry Pi and motor drivers module by being connected through its USB power inputs, thereby powering the entire system.

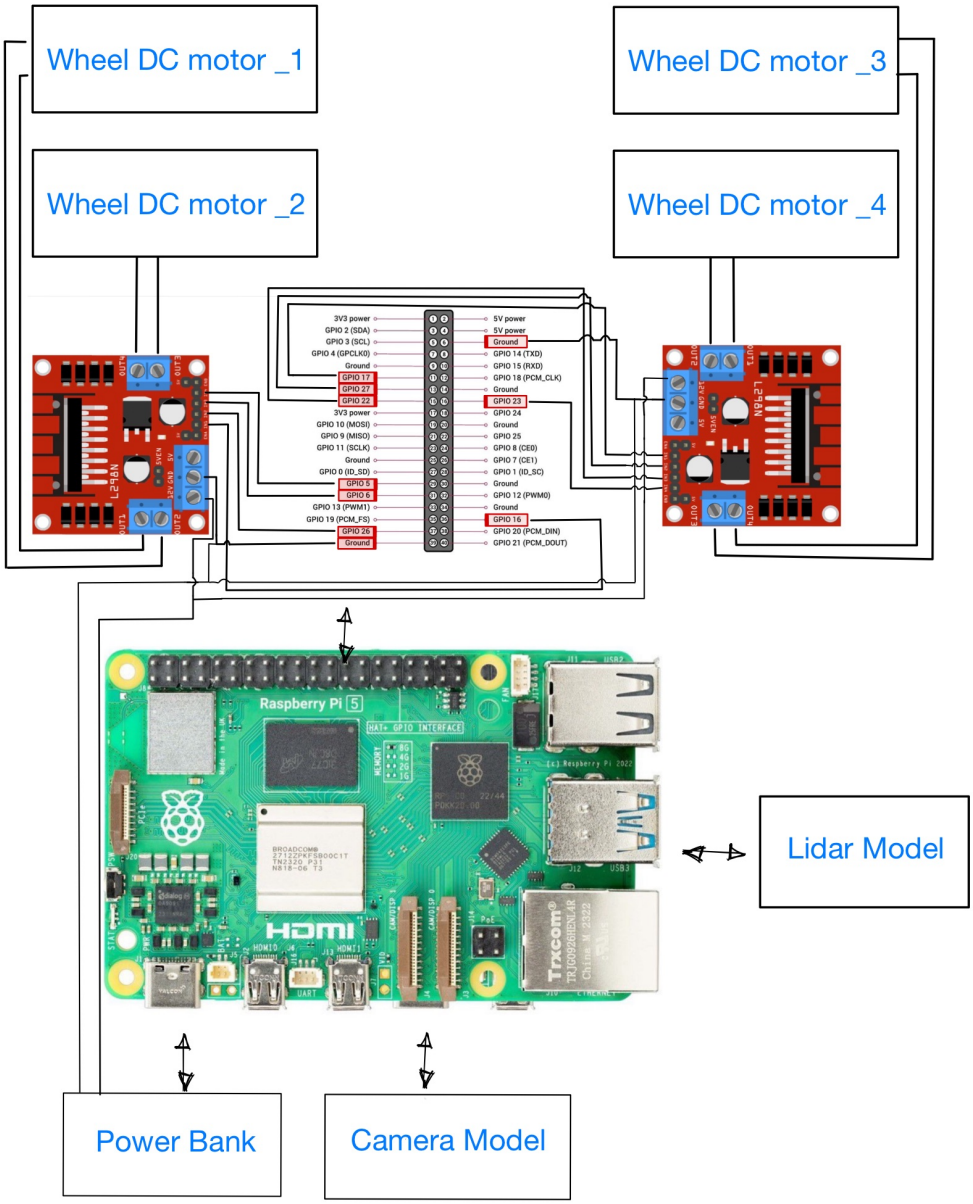


Figure 12: Hardware component connection

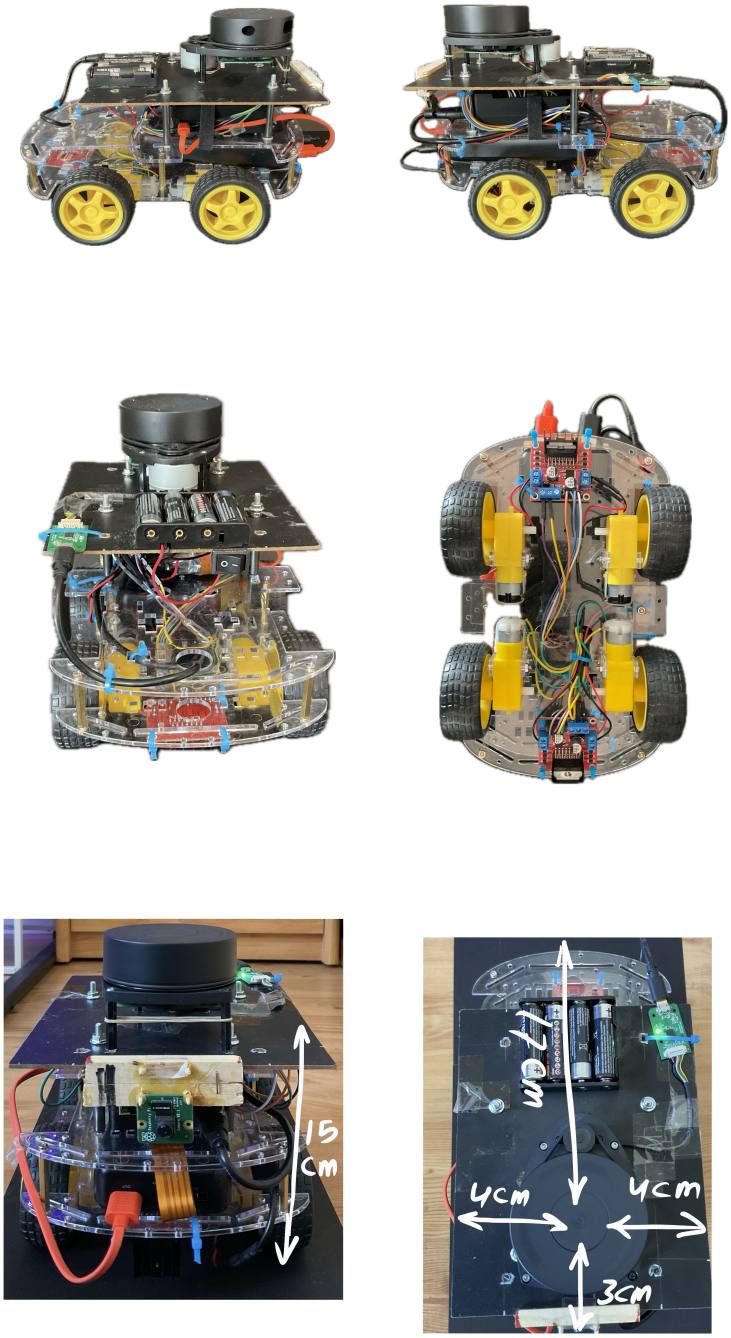


Figure 13: A comprehensive overview of the platform, highlighting the connections, wiring, and dimensions of the LiDAR components.

3.2 LiDAR A1M8

The LiDAR A1M8 represents a low-cost 360-degree 2D laser scanner LiDAR solution, designed and manufactured by SLAMTEC. The system can perform a 360-degree scan within a 6-meter range. The resulting 2D point cloud data can be utilized for mapping, localization, and object detection. The LiDAR A1 operates as a laser triangulation measurement system, demonstrating optimal performance in a range of indoor and outdoor environments without direct sunlight exposure [3]. The LiDAR A1M8 is applicable in the following areas:

- Navigation and localization
- Obstacle avoidance
- Environment scanning and 3D re-modeling
- General simultaneous localization and mapping (SLAM)

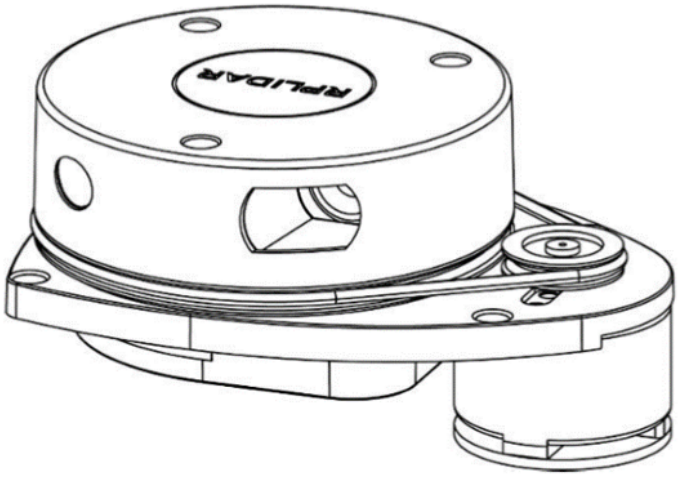


Figure 14: LiDAR A1M8 from Slamtec

3.2.1 SDK

Slamtec RoboStudio interfaces with the LiDAR through a serial connection to retrieve data. This data is then processed and presented through the user interface. The main display section features a polar plot illustrating the LiDAR scan results. This plot includes a circular grid that represents distance measurements, with the LiDAR's position at the center. Detected obstacles or surfaces are shown as red points or dots on the polar grid, relative to the LiDAR's location. The plot is divided into angular segments, indicating the direction of detected objects in degrees (0 to 360°). Additionally, the display indicates the connected device, the associated port, and the current scanning status.

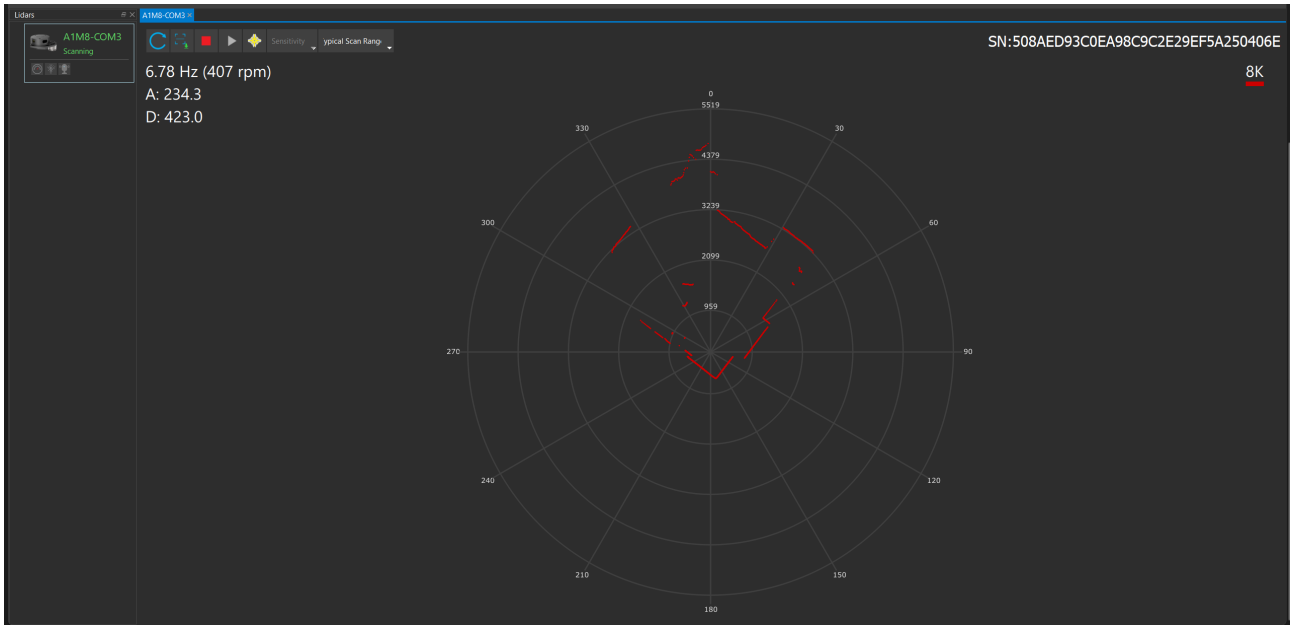


Figure 15: A snapshot of the Slamtec RoboStudio for data representation for the LiDAR.

3.2.2 System Connectivity

The RPLIDAR A1M8 is comprised of two subsystems. The first component is the motor system, which is responsible for rotating the second subsystem. The second subsystem is the scanner, which is tasked with monitoring the environment. Via the communication interface (serial port/USB), the user can receive range scan data.

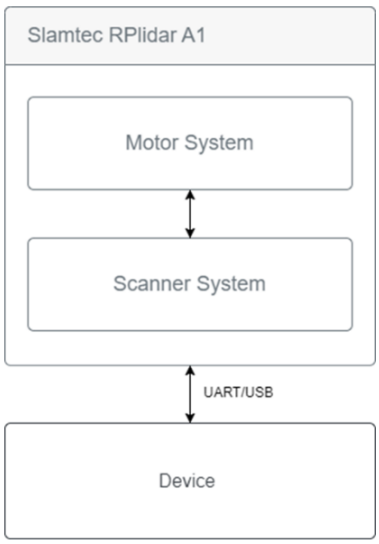


Figure 16: LiDAR A1 System Assembly

The scanner system and the motor system each have different control interfaces: the motor system has control interfaces (control motor for signaling) as well as power interfaces (VCC and ground), while the scanner system has power interfaces (VCC and ground) as well as transmitter and receiver interfaces.

Interface	Signal Name	Description
5V_MOTO	Power	Power for LiDAR A1 Motor as 5V
CTRL_MOTO	Input	Enable signal for LiDAR A1 Motor
GND	Power	GND for LiDAR A1 Motor as 0V

Table 2: Interfaces for LiDAR Motor[4]

Interface	Signal Name	Description
5V_Vcc	Power	Power for LiDAR A1 Motor as 5V
TX	Output	Serial output for Range Scanner Core
RX	Input	Serial input for Range Scanner Core
GND	Power	GND for LiDAR A1 Motor as 0V

Table 3: Interfaces for LiDAR scanner[4]

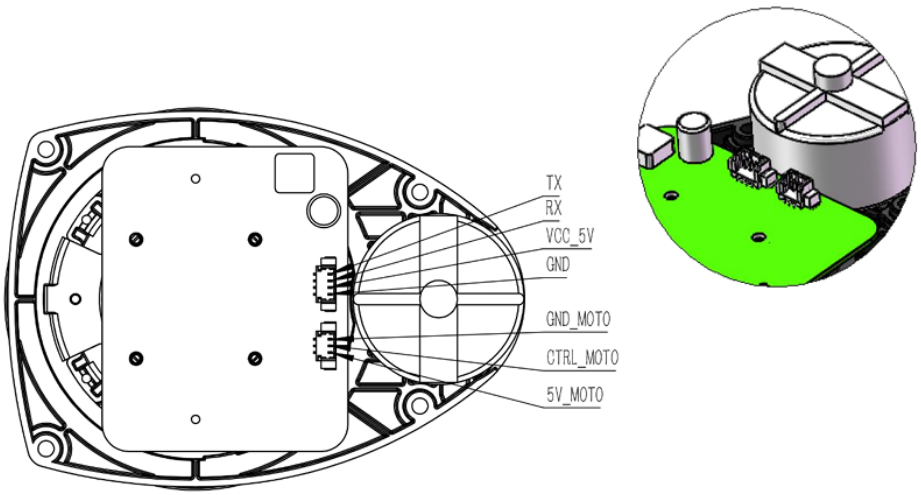


Figure 17: LiDAR communication and power Interfaces Modules [4]

3.2.3 Operational State and Mechanism

Mechanism

The LiDAR employs two fundamental principles: laser triangulation ranging and high-speed vision acquisition. The device is capable of measuring distances in the environment with a high degree of reliability, utilizing a high-resolution distance output. It achieves this by capturing more than

8000 data points per second with high resolution when the LiDAR system is configured to operate at its highest frequency of 10 Hz. The LiDAR transmits a modulated infrared signal, which is then reflected by the target. The returned signal is processed by a vision acquisition system, after which a DSP (Digital Signal Processor) embedded in the LiDAR processes the sample data and calculates the distance and angle for each point between the object and the LiDAR [3].

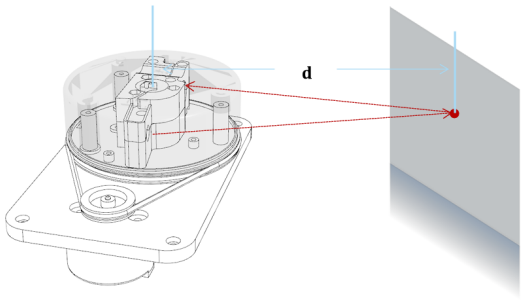


Figure 18: LiDAR A1M8-Schematic Diagram of Operation[3]

Operational States

The LiDAR operates through four primary operational states:

1. **Idle State:** The initial state in which the unit is powered on but not actively scanning.
2. **Scanning State:** This state involves actively transmitting and receiving laser pulses to perform a scanning operation.
3. **Request State:** When the device is waiting for or responding to external commands or requests from a host system.
4. **Protection Stop State:** This state occurs when the device detects a critical problem.

These states represent different operational phases that the LiDAR transitions through based on its environmental conditions and operational requirements. The LiDAR enters the **Idle State** upon power-up or reset, conserving power by disabling the laser and measurement system. Switching to the **Scanning State** activates these components for continuous distance measurement and output transmission. Upon receiving request packets from the host system, the LiDAR enters the **Request State**. During this phase, the LiDAR refrains from performing any scanning operations. After processing the request, the next state is determined by the specific request received. When the LiDAR detects hardware faults, it stops its current operation and enters a **Protection Stop State**. While in this state, the host system can still communicate with the device to query its operational status. However, scanning operations cannot be initiated unless the host system sends a reset request to reboot the LiDAR system. During the **Scanning State**, the LiDAR continuously monitors the motor rotation status. Distance measurements and the transmission of results to the host system only

begin when a stable motor speed is reached [2].

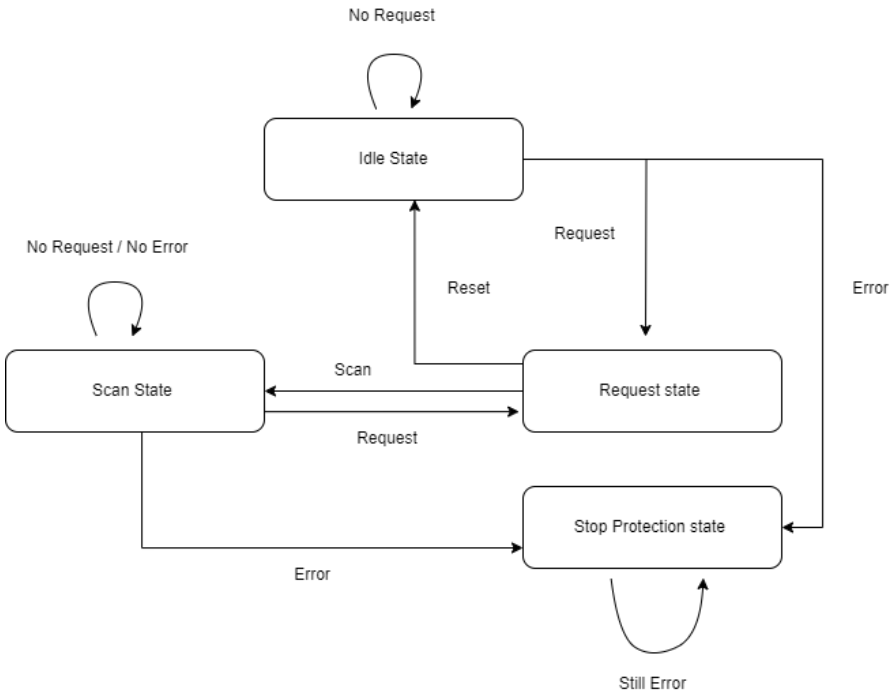


Figure 19: LiDAR A1M8 Operational Status

3.2.4 Data Output

During runtime, the LiDAR continuously outputs sample data for each scan. Each returned data sample contains a dictionary with the information shown in the table below.

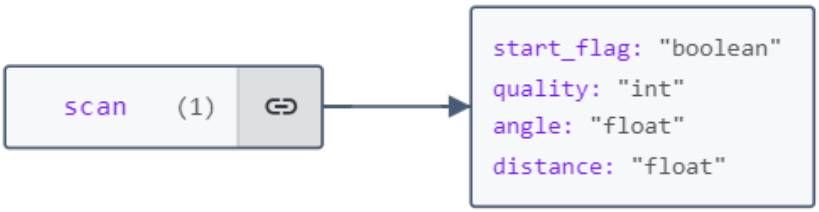


Figure 20: Sample Data Format for Each Point as Dictionary

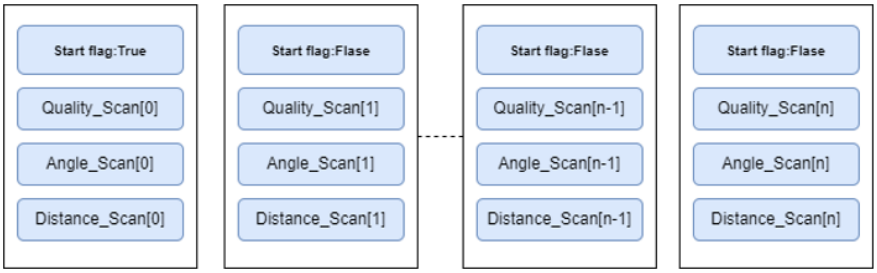


Figure 21: Sample Scan Data Points Structure

Variable	**Type**	**Unit**	**Description**
Start flag	Boolean	True or False	True if it is the first scan point out of 360 scan points, otherwise false (flag for a new rotation scan).
Quality	int	Level	Quality of measurement
Angle	Float	Degree	Current angle
Distance	Float	mm	Current measurement distance

Table 4: LiDAR A1 Sampling Point Data Information

3.2.5 Request and Response Packets Format and Type

The LiDAR utilizes Request and Response packets to facilitate communication between the LiDAR device and the Host system.

Request Packet Format The Request command from the host system follows the format shown in the following table. The **0xA5** section (start flag) serves as a consistent identifier for each request packet and marks the initiation of a new request. This is followed by a command, and subsequent bytes are optional components of the request packet [2].

Start Flag	Command	Payload Size	Payload Data	Checksum
1 byte	1 byte	1 byte	0 – 255 byte	1 byte

Table 5: Request Packet Format

Response Packet Format: This packet consists of two components:

1. Descriptor
2. Response Data

Given the diversity in response types, each request packet awaiting a response requires that the LiDAR transmit a singular descriptor for every request. Subsequently, it should dispatch the corresponding response data, which may entail either a single or multiple responses, contingent on the specifications of the request packet. Some requests may require only a single response, while others necessitate multiple responses [2].

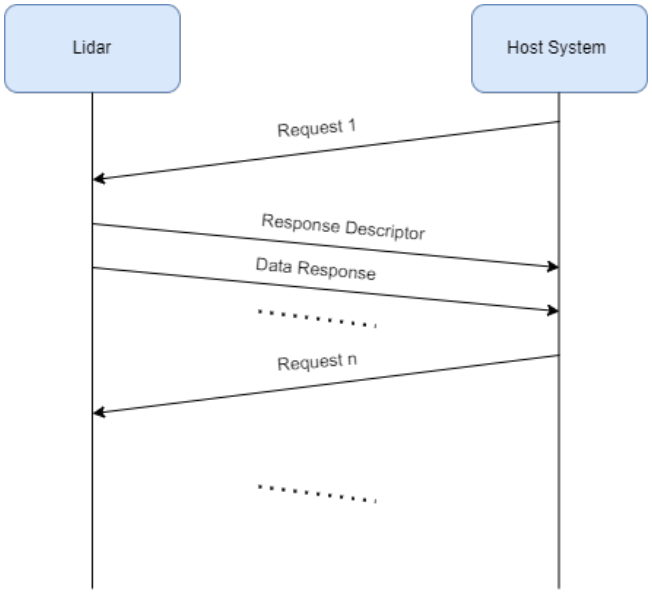


Figure 22: Response Packets

The Descriptor utilizes fixed values 0xA5 and 0x5A to acknowledge the start of the response and the descriptor. The response length is encoded in 30 bits, while the Send mode is specified in 2 bits:

- 0x0: indicates a single request and response, only one data response packet
- 0x1: indicates a single request with multiple responses, continuously sending out data response packets

The data type, represented by 1 byte, identifies the type of incoming data response packets [2].

Start Flag 1	Start Flag 2	Data Response Length	Send Mode	Data Type
1 byte	1 byte	30 bit	2 bit	1 byte

Table 6: Response Descriptors Format

Request with No Response

For commands such as STOP and RESET, **LiDAR** operates in a single request–no response mode where it does not provide a response to the host system, as there is no necessity for feedback. Consequently, host systems are advised to observe a waiting interval before issuing subsequent commands, allowing **LiDAR** sufficient time to execute the requested operations. Failure to observe this interval may result in the **LiDAR**'s protocol stack discarding the command [2].

Command	Hex Value	Description
Stop	0x25	Transition from the existing state to the idle state
Reset	0x40	Reset the **LiDAR**

Table 7: Request Commands with No Response

Stop: A command from the host system (0x25) causes the **LiDAR** to transition from an active state to an idle state. Concurrently with this transition, the **LiDAR** exits the scanning state and deactivates the measurement diodes. As **LiDAR** does not provide a response packet for this request, the host system should allow a minimum interval of 1 millisecond (ms) before issuing another request. This ensures proper communication timing and avoids potential conflicts or errors in data transmission [2].

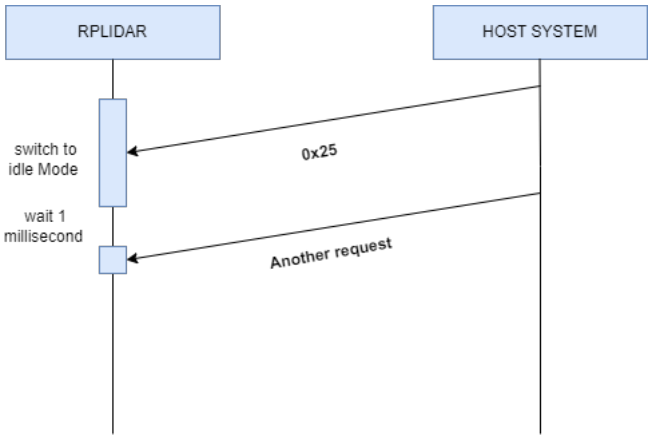


Figure 23: Stop Command as Single Request with No Response

Reset: In the event that the **LiDAR** is not in Protection Stop State but in Idle mode and receives a reset request command from the host system, it is required to transition to scan mode and resend the measurement samples. Conversely, if the **LiDAR** is already in scan mode, it will cease current measurements and recommence from the start. It should be noted that the request command is disregarded if the **LiDAR** is in protection mode. In the instance that **LiDAR** does not transmit a reset request, it is recommended that the host system waits a minimum of two milliseconds (ms) before initiating another request [2].

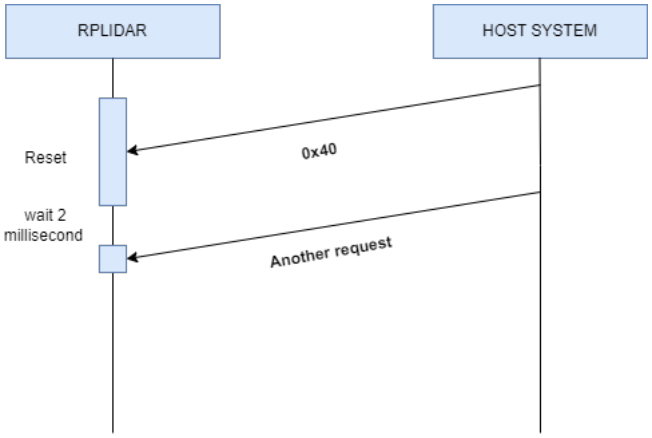


Figure 24: Reset Command as Single Request with No Response

Request with Single Response

In response to a received request, **LiDAR** initiates the requisite operations. Subsequently, a single response packet is transmitted from the **LiDAR** to the host system, containing all pertinent information [2].

Command	Hex Value	Description
Get Info	0x50	Retrieve general information about the LiDAR.
Get Health	0x52	Retrieve the health status of the LiDAR.
Get Sample Rate	0x59	Retrieve the current sampling rate of the LiDAR.

Table 8: Request Commands with Single Response

Get Info: The function returns information pertaining to the device in question, including the serial number, the model, the firmware major version, the firmware minor version, and the hardware version [2].

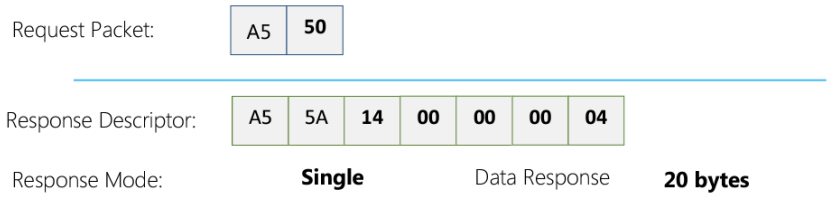


Figure 25: Get Info Command as Single Request with Single Response [2].

Response Handling Once the LiDAR has been initialized and the `get_info` function has been called, a single response should be returned to the LiDAR in the form of a hexadecimal message. This message will be of the form:

```
b'\x18\x1d\x01\x07P\x8a\xed\x93\xc0\x
ea\x98\xc9\xc2\xe2\x9e\xf5\xa2P@n'
```

This message should then be passed to the `PyRPlidarDeviceInfo` class, where the information should be extracted and converted from hex to decimal as a dictionary. The conversion process can be described as follows:

- Start with the hexadecimal message:

```
b'\x18\x1d\x01\x07P\x8a\xed\x93\xc0\x
ea\x98\xc9\xc2\xe2\x9e\xf5\xa2P@n'
```

- Extract the individual hexadecimal bytes.
- Convert bytes from hexadecimal to decimal notation, with the exception of the serial number, which should remain in hexadecimal format.
- Return the converted values in a dictionary.

```
model: 24
firmware_minor: 29
firmware_major: 1
hardware: 7
serialnumber: "508AED93C0EA98C9C2E29EF5A250406E"
```

Figure 26: Dictionary with **LiDAR** Information

```
lidar=PyRPlidar()
lidar.connect(prot="/dev/ttyUSB0,baudrate=115200")
info=lidar.get_info
```

Listing 3.1: Establishing a connection with a LiDAR device and obtaining data through the use of the `get_info` function

```
def get_info(self):
    self.send_command(RPLIDAR_CMD_GET_INFO)
    discriptor = self.receive_discriptor()
    data = self.receive_data(discriptor)
```

```
5     return PyRPlidarDeviceInfo(data)
```

Listing 3.2: Send the `get_info` request to the Lidar system and retrieve the data, representing it as hexadecimal bytes[11].

```
1 class PyRPlidarDeviceInfo:
2     def __init__(self, raw_bytes):
3         self.model = raw_bytes[0]
4         self.firmware_minor = raw_bytes[1]
5         self.firmware_major = raw_bytes[2]
6         self.hardware = raw_bytes[3]
7         self.serialnumber = codecs.encode(raw_bytes[4:], 'hex').upper()
8         self.serialnumber = codecs.decode(self.serialnumber, 'ascii')
```

Listing 3.3: Extracting the data from the received message and mapping it to a dictionary[11].

Get Device Health Status: It is possible for users to transmit the GET_HEALTH request to obtain information regarding the current status of the LiDAR. If the LiDAR transitions into the Protection Stop state because of a failure in the hardware, an error code will be generated and provided to the user in order to indicate the precise nature of the fault. In the instance that the core system identifies a potential future hardware risk, it assigns a status value of Warning (1), even though the LiDAR continues to function in the usual manner. In the event of a protection stop state, the status value is elevated to Error (2) [2].

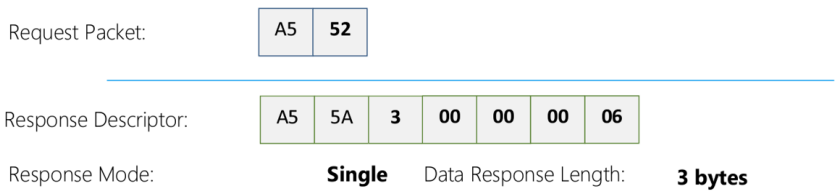


Figure 27: get health Command as Single Request with Single Response format[2]

Response Handling Once the LiDAR has been initialized and the get_health function has been called, a single response should be returned to the LiDAR in the form of a hexadecimal message. This message will be of the form:

```
b'\x00\x00\x00'
```

- Start with the hexadecimal message:

  ```
  b'\x00\x00\x00'
  ```

- After receiving data from the LiDAR, the PyRPlidarHealth class is called to extract the individual hexadecimal bytes.
- Apply the left shift operator to the terms $byte_1$ and $byte_2$, as shown:

$$\text{error\_code} = \text{byte\_1} \ll +\text{byte\_2}$$

$$\text{status} = \text{byte\_0}$$

  after that convert bytes from hexadecimal to decimal notation.
- Return the converted values in a dictionary:

```
{'status': 0, 'error_code': 0}
```

Variable	Type	Description
status	int	**0**: Good, **1**: Warning, **2**: Error
Error_Code	int	Error code for the LiDAR

Table 9: Request Get Device Health Status commands with single response result

Upon initialization of the LiDAR and invocation of the **get_health** function, a singular response in the hexadecimal message format, specifically structured as `b'\x00\x00\x00'`, is expected from the LiDAR. This message must subsequently undergo processing through the `PyRPlidarHealth` class to extract its content and convert it into a dictionary format, translating from hexadecimal to decimal representation. This conversion facilitates the structured retrieval and interpretation of health-related data from the LiDAR device.

```
lidar = PyRPlidar()
lidar.connect(port="/dev/ttyUSB0", baudrate=115200)
health = lidar.get_health()
```

Listing 3.4: Invoke the get_health function.

```
def get_health(self):
    self.send_command(RPLIDAR_CMD_GET_HEALTH)
    discriptor = self.receive_discriptor()
    data = self.receive_data(discriptor)
    return PyRPlidarHealth(data)
```

Listing 3.5: Send the get health request to the Lidar system and retrieve the data, representing it as hexadecimal bytes[11]

```
class PyRPlidarHealth:

    def __init__(self, raw_bytes):
```

```
4         self.status = raw_bytes[0]
5         self.error_code = (raw_bytes[1] << 8) + raw_bytes[2]
6
7     def __str__(self):
8         data = {
9             "status": self.status,
10            "error_code": self.error_code
11        }
12        return str(data)
```

Listing 3.6: Extracting the data from the received message and mapping it to a dictionary[11].

Get sample rate By submitting this request, the host system can obtain the duration of a single measurement in both Standard and Express Scan modes, allowing for precise calculation of the LiDAR's current speed [2].

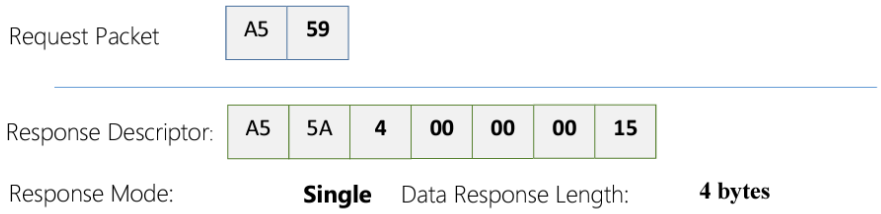


Figure 28: sample rate command as Single Request with Single Response [2]

Once the get_samplerate function has been called, a single response should be returned to the LiDAR in the form of a hexadecimal message. This message will be of the form:

`b'\xfc\x01\xfe\x00'`

- Start with the hexadecimal message:
- After receiving data from the LiDAR, the PyRPlidarSamplerate class is called to extract the individual hexadecimal bytes.
- apply the left shift operator to the terms $byte_0$ and $byte_1$, as well as $byte_2$ and $byte_3$, as shown:

$$t\_standard = byte_0 + \ll byte_1$$

$$t\_express = byte_2 + \ll byte_3$$

  after that Convert bytes from hexadecimal to decimal notation
- Return the converted values in a dictionary:

  `{'t_standard': 508, 't_express': 254}`

Request with multi Response

LiDAR, after a request is received, initiates the necessary operations and then transmits multiple response packets to the host system, usually in response to scan requests, containing all relevant information such as descriptors and data responses [2].

Command	Hex Value	Description
scan	0x20	Start scan data with normal scan rate
scan express	0x82	Start scan data with high scan rate
scan force	0x21	Immediately scan for device debugging

Table 10: Request with multi Response Command Descriptions [2]

Scan When the LiDAR is not in the stop protection state and receives a scan request (0xA5 0x20) from the host system, it shifts to the scan state at a normal scan rate, begins measurement, and sends a response packet containing the descriptor and response data back to the host system. However, if the LiDAR is in the stop state, it will ignore the request. On the other hand, if the LiDAR is in the idle state, it will transition to the scan state. Additionally, if the LiDAR is already in the scan state and receives a new scan request, it will terminate the current measurement sampling and initiate a new scan cycle [2].

Flag1	Flag2	data_lenght	send_mode	Data_type
A5	5A	05 00 00	40	81

Table 11: Descriptor message

As mentioned before, it was indicated that the send mode operates using a 2-bit system. To illustrate this, we can convert the hexadecimal value `0x40` to its binary equivalent. The conversion process yields `0100 0000`. By examining the first two bits of this binary representation, we obtain `01`. This binary sequence, `01`, corresponds to the decimal number 1, which signifies a multiple response.

The data is encapsulated in a 5-byte message, described as shown in the Data Response Format Figure 29.

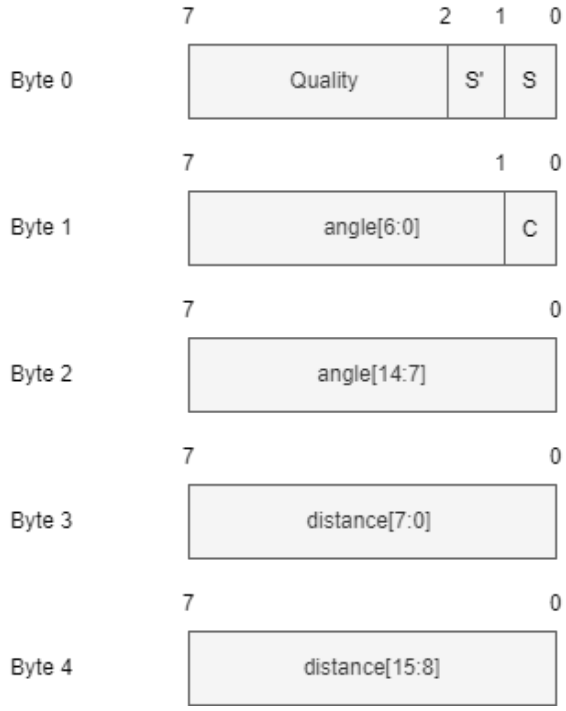


Figure 29: Data Response Fromat [2]

An example message, represented in hexadecimal format, is `b'>\d7\x93\x1F\x06'`. The first byte, `byte0`, signifies the flag and is evaluated as follows:

Quality and the Start flag bit :

1. `byte[0]` is represented by the character `'>'`.
2. Converting `'>'` to its hexadecimal form yields `0x3E`.
3. The flag `S_flag` is determined by performing a bitwise AND operation between `byte0` and `0x01`:

$$S\_flag = \text{byte}[0] \& 0x01$$

   Substituting the values, we get:

$$S\_flag = 0x3E \& 0x01$$

4. Converting these values to binary:

$$0x3E \text{ translates to } 0011\,1110$$
$$0x01 \text{ translates to } 0000\,0001$$

5. Performing the bitwise AND operation:

$$0011\,1110\&0000\,0001 = 0000\,0000$$

6. The result, `0000 0000`, translates to `S_flag = 0`.
7. Converting `S_flag` to a boolean yields `False`.
8. The inverse of `S_flag` is then computed as `not S_flag`, which evaluates to `True`.

The **quality** of the data is determined by performing a right shift operation on the first byte (`byte[0]`). The steps are as follows:

1. `byte[0]` is represented by the hexadecimal value `0x3E`.
2. Perform a right shift operation on `byte[0]` by 2 bits:

$$\text{quality} = \text{byte}[0] \gg 2$$

3. Converting `0x3E` to binary yields `0011 1110`.
4. Shifting `0011 1110` two bits to the right results in `0000 1111`.
5. The binary value `0000 1111` corresponds to the hexadecimal value `0x0F`.
6. Converting `0x0F` to decimal gives the value `15`.

Angle Calculation and Check Bit Extraction: The second `byte[1]`contains both the check bit and part of the angle representation. The angle is represented using 15 bits: bits 1 to 8 of `byte[1]` combined with all 8 bits of `byte[2]`.

1. `byte[1]` is represented by the hexadecimal value `0xD7`.
2. The check bit is determined by performing a bitwise AND operation between `byte[1]` and `0x01`:

$$\text{check\_bit} = \text{byte}[1]\&0x01$$

Substituting the values, we get:

$$\text{check\_bit} = 0xD7\&0x01$$

3. Converting these values to binary:

$$0xD7 \text{ translates to } 1101\,0111$$
$$0x01 \text{ translates to } 0000\,0001$$

4. Performing the bitwise AND operation:

$$1101\,0111 \& 0000\,0001 = 0000\,0001$$

5. The result, `0000 0001`, translates to `check_bit = 1`.

1. The angle is calculated by combining bits 1 to 8 of `byte[1]` with all bits of `byte[2]` and then dividing by 64.
2. `byte[1]` and `byte[2]` are represented by the hexadecimal values `0xD7` and `0x93`, respectively.
3. Perform a right shift on `byte[1]` by 1 bit and a left shift on `byte[2]` by 7 bits:

$$\text{angle} = (\text{byte}[1] \gg 1) + (\text{byte}[2] \ll 7)$$

   Substituting the values, we get:

$$\text{angle} = (0xD7 \gg 1) + (0x93 \ll 7)$$

4. Converting these values to binary:

$$0xD7 \text{ translates to } 1101\,0111$$
$$0x93 \text{ translates to } 1001\,0011$$

5. Shifting `1101 0111` one bit to the right results in `0110 1011`.
6. Shifting `1001 0011` seven bits to the left results in `1001 0011 0000 0000`.
7. Adding the shifted values:

$$0110\,1011 \text{ translates to } 107 \text{ in decimal}$$
$$1001\,0011\,0000\,0000 \text{ translates to } 18816 \text{ in decimal}$$

$$\text{angle} = 107 + 18816$$

8. Dividing by 64:

$$\text{angle} = \frac{107 + 18816}{64}$$
$$\text{angle} = \frac{18923}{64} \approx 295.67$$

Distance Calculation: The distance is determined using the last two bytes (byte[3] and byte[4]). The steps for calculating the distance are as follows:

1. `byte[3]` and `byte[4]` are represented by the hexadecimal values `0x1F` and `0x06`, respectively.

2. The distance is calculated by summing these two bytes and then dividing by 4:

$$\text{distance} = \text{byte}[3] + \text{byte}[4] \ll 8$$

Substituting the values, we get:

$$\text{distance} = 0x1F + 0x06 \ll 8$$

3. Performing left shift on byte[4] by 8 bits:
4. Converting these values to binary:

$$0x1F \text{ translates to } 0001\,1111$$
$$0x06 \ll 8 \text{ translates to } 0000\,0110 \ll 8 = 0000\,0110\,0000\,0000$$

5. Adding the binary values:

$$0000\,0110\,0000\,0000 + 0001\,1111 = 1536 + 31 = 1567$$

6. Dividing by 4:

$$\text{distance} = \frac{1567}{4} = 391.75$$

```
1
2         self.start_flag = bool(raw_bytes[0] & 0x1)
3         self.quality = raw_bytes[0] >> 2
4         self.angle = ((raw_bytes[1] >> 1) + (raw_bytes[2] << 7)) / 64.0
5         self.distance = (raw_bytes[3] + (raw_bytes[4] << 8)) / 4.0
```

Listing 3.7: Extraction of measurement data in the PyRPlidarMeasurement class [11].

Step	Operation	Result
Byte 0 (Hex)	`0x3E`	Binary: `0011 1110`
Byte 0 and 0x01	`0x3E and 0x01`	Hex: 0x00 ,`bool(0x00)`, **`S_flag=False`**, **`S_flag_inv=True`**
Byte 0 » 2	`0x3E » 2`	Binary: `0000 1111` is `0x0F`, **Quality** = 15
Check Bit Extraction	`check_bit = byte[1] & 0x01`	**`check_bit = 1`**
Byte 1 (Hex)	`0xD7`	Binary: `1101 0111`
Byte 2 (Hex)	`0x93`	Binary: `1001 0011`
Right Shift Byte 1	`byte[1] » 1`	`0110 1011` (Binary), `107` (Decimal)
Left Shift Byte 2	`byte[2] « 7`	`1001 0011 0000 0000` (Binary), `18816` (Decimal)
Sum	`107 + 18816`	`18923`
Division by 64	`18923 / 64`	**Angle** =`295.67`
Byte 3 (Hex)	`0x1F`	Binary: `0001 1111`
Byte 4 (Hex)	`0x06`	Binary: `0000 0110`
Left Shift Byte 4	`0x06 «8`	`0000 0110 0000 0000`(Binary),`1536`(Decimal)
Sum	`1536 + 31`	`1567`
Division by 4	`1567 / 4`	**Distance** = `391.75`

Table 12: Summary of Data Calculations: Evaluating Sample Responses via the Scanning Function.

Express Scan When the LiDAR receives an Express Scan request from the host, it switches to a high-performance measurement mode compared to the standard scanning mode. The Express Scan mode maximizes the scanning rate, achievable with the RPLiDAR A2, which supports a 4kHz sampling rate. This request allows the system to operate at that rate. However, for the model used in this experiment, the RPLiDAR A1, the sampling rate remains at 2kHz, which is equivalent to the normal scan mode [2].

Force Scan The force scan operates with the same mechanism and logic as the scan mode. However, it differs in its response format. This mode is primarily utilized for device debugging purposes [2].

4 Implementation

The project consists of specialized model classes designed in a structured way to address specific computational or distinct tasks. By establishing associations among these models, this unified approach ensures that the collective output aligns with high-level experimental goals. The objective can be applied to different scenarios within the experimental context. The key models used in the project include the camera, LiDAR, wheel, lane, and object detection systems. Overall, these models are crucial components of the system, performing tasks related to visual perception and environmental detection. They feature different methods of computation and inputs, ensuring the system can effectively interact with complex environments.

4.1 Models

4.1.1 LiDAR Model

The LiDAR class serves as an interface for managing operations associated with the LiDAR A1M8 sensor, encapsulating the functionality necessary to initialize the sensor, acquire scan data, and store this information efficiently.

Class Initialization Upon instantiation, the LiDAR class initializes several key attributes:

LiDAR: An instance object of the LiDAR library, which directly interacts with the hardware.

log: A logging object created by logFile to maintain a record of operational logs at a specified path.

scan_count: A counter for the number of scans conducted, used to manage data collection intervals.

distance_data: A dictionary prepopulated with keys representing each possible angle (0 to 359 degrees) initialized to zero, to store distance measurements.

batch_size: Set to 360, this attribute determines how many scans should accumulate before data is written to a file, aligning with the full 360-degree rotational capability of the sensor.

Method Descriptions:

create_scan_dict method: processes incoming scan data by updating the distance_data dictionary with new distance measurements at corresponding angles if they differ from previous readings. This ensures the maintenance of a current snapshot of the surroundings at 1-degree resolution.

write_to_json method: converts the distance_data dictionary into a JSON-formatted string and writes it to a file, thus enabling the data to be stored persistently.

get_scan_data: A simple getter method providing external access to the current distance data.

run method : is responsible for the continuous operation of the LiDAR sensor. It handles device connections, initiates scanning, and processes incoming data through a loop. To ensure data permanence, it calls the write_to_json function at regular intervals defined by the batch size.

```
from pyrplidar import PyRPlidar
import time
import json
import os
from Log import logFile

class Lidar:
    PORT_NAME = '/dev/ttyUSB0'
    Rate = 115200
    Timeout = 5

    def __init__(self, path):

        self.lidar = PyRPlidar()
        self.log = logFile(os.path.join(path, 'selfDriving/log/lidar_logs.log'))
        self.scan_count = 0
        self.distance_data = {angle: 0 for angle in range(360)}
        self.batch_size = 360
        self.path = path

    def create_scan_dict(self, scan_data):

        for angle in self.distance_data:
            if int(self.distance_data[angle]) !=  int(scan_data[1]) and int(scan_data
    [0]) == angle:
                if int(scan_data[1]) >= 0:
                    self.distance_data[angle] = int(scan_data[1]) / 10
```

```
27                 else:
28                     self.distance_data[angle] = scan_data[1]
29
30     def write_to_json(self, data):
31         jsonPath = os.path.join(self.path, 'selfDriving/data/lidarData.json')
32         with open(jsonPath, 'w') as f:
33             json.dump(data, f, indent=4)
34
35     def get_scan_data(self):
36
37         return self.distance_data
38
39     def run(self):
40
41         while True:
42             try:
43                 self.lidar.connect(port=self.PORT_NAME, baudrate=self.Rate, timeout=
    self.Timeout)
44                 self.lidar.get_scan_modes()
45                 info = self.lidar.get_info()
46                 self.log.log(f"info : {info}")
47                 health = self.lidar.get_health()
48                 self.log.log(f"health : {health}")
49                 self.lidar.set_motor_pwm(500)
50                 scan_generator = self.lidar.start_scan_express(0)
51                 self.log.log('Recording measurements...')
52                 scan_count = 0
53                 for _, scan in enumerate(scan_generator()):
54                     self.create_scan_dict([int(round(scan.angle)), int(round(scan.
    distance))])
55                     scan_count += 1
56                     if scan_count >= self.batch_size:
57                         self.write_to_json(self.distance_data)
58                         scan_count = 0
59                         self.log.log("Scan data are written to the JSON file.")
60             except Exception as e:
61                 self.log.error(f"An error occurred: {e}. Retrying...")
62                 self.log.log("Lidar Retry in 3 seconds to reconnect.")
63                 time.sleep(3)
64             except KeyboardInterrupt:
65                 self.lidar.stop()
66                 self.lidar.set_motor_pwm(0)
67                 self.lidar.disconnect()
68                 self.log.log("Lidar disconnected.")
69                 break
70             finally:
71                 self.lidar.stop()
72                 self.lidar.set_motor_pwm(0)
```

```
73                 self.lidar.disconnect()
74                 self.log.log("Lidar disconnected.")
75                 break
```

Python supported libraries

During the experiment, I investigated various supported libraries and found four:

1. adafruit_rplidar [13]
2. SkoltechRobotics [19]
3. Roboticia_rplidar [18]
4. HyunjePyrplidar [12]

All of these libraries support the normal scan mode, with the exception of HyunjePyrplidar, which also supports the express scan mode. A common issue among these libraries is the error rate. Some libraries initially exhibit a high error rate with more than 150 errors per 360-degree scan, which decreases over time. However, one library maintains a consistent error rate in the range of 40 to 50 errors per 360-degree scan. Due to this, it was necessary to develop a function to address and reduce the error rate. The objective is to minimize errors and produce acceptable values for subsequent processing operations.

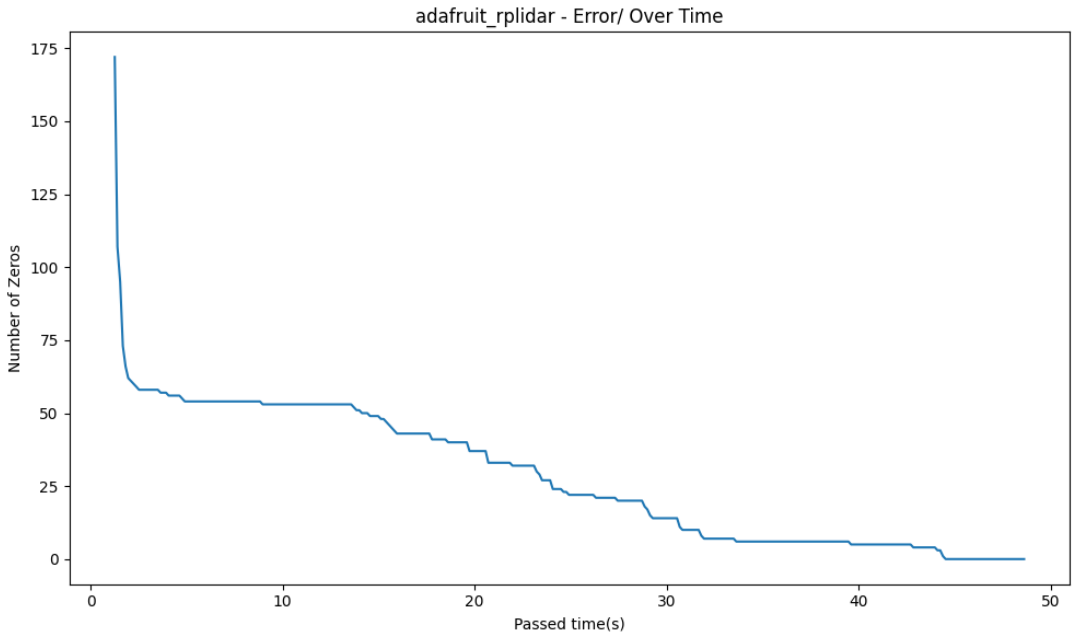


Figure 30: adafruit_rplidar error rate

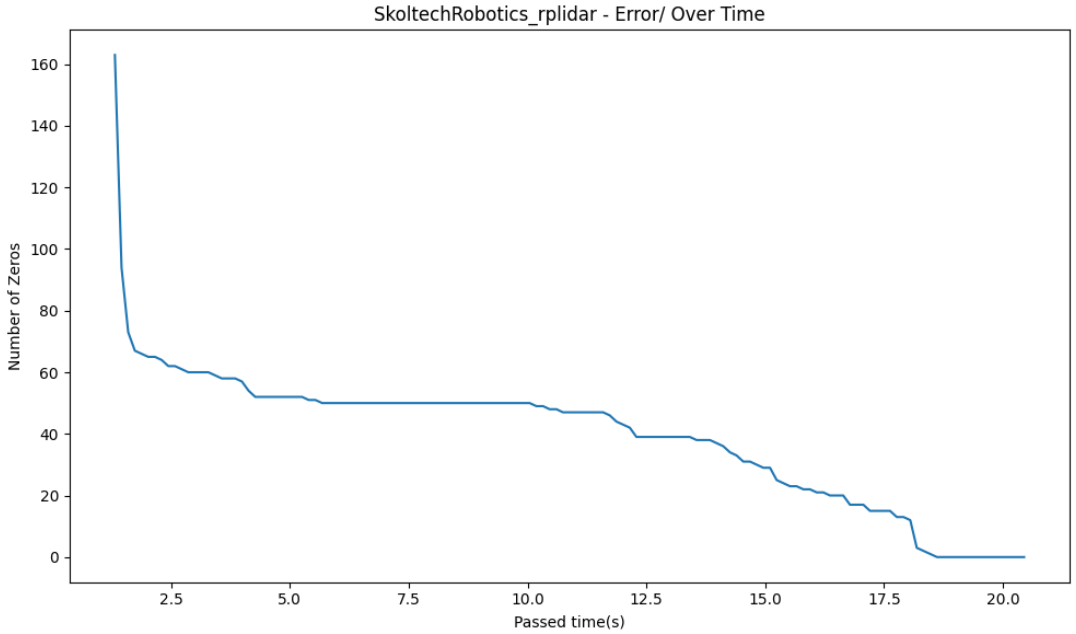


Figure 31: SkoltechRobotics error rate

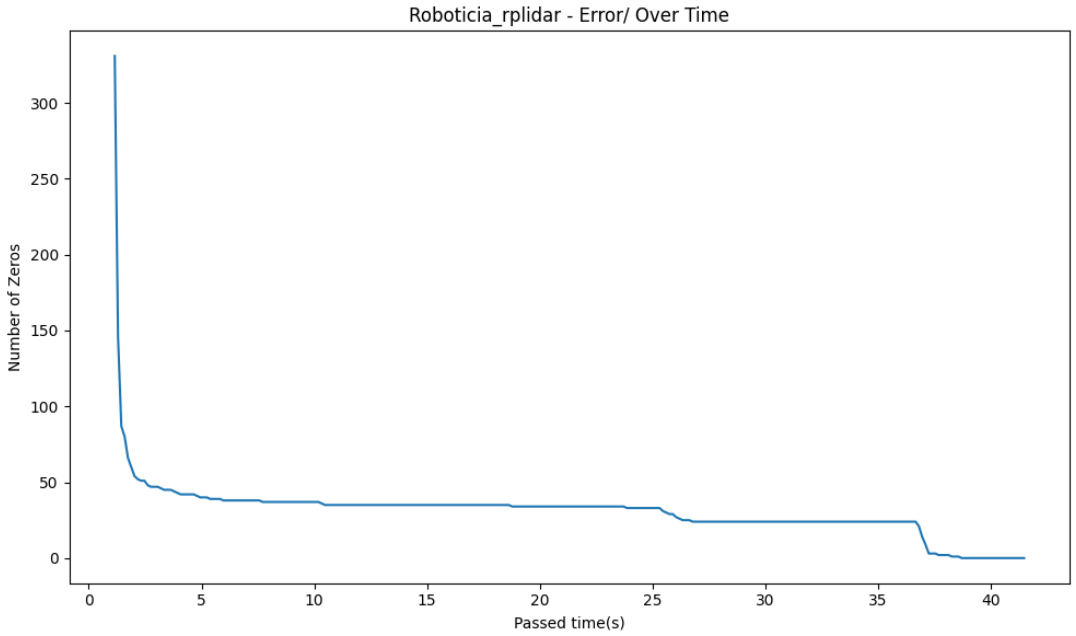


Figure 32: Roboticia_rplidar error rate

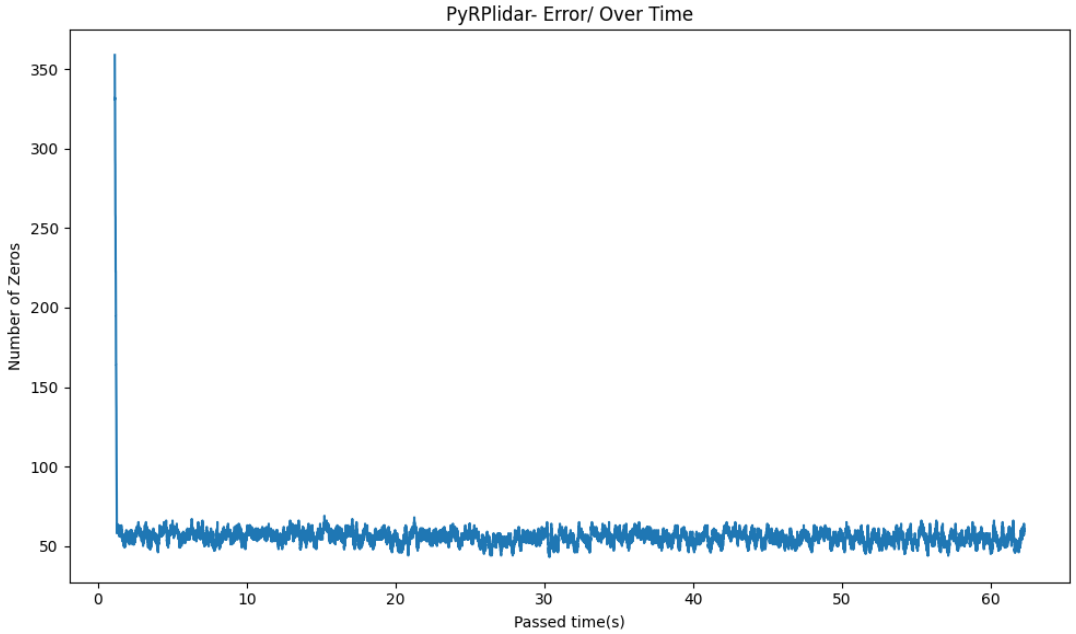


Figure 33: HyunjePyrplidar error rate

It is important to note that identifying errors in scanned data, particularly where values are zero, is relatively simple. This issue often arises due to an error in receiving some information from the LiDAR device, as zero values complicate the measurement of distance. To address this, it is useful to replace zero values with more accurate estimates. Initially, the scanned data should be partitioned into smaller clusters. Each cluster should then be examined to determine if it contains zero values. For clusters with zero values, compute the mean of the non-zero values and substitute each zero value with this calculated mean. Subsequently, update the clusters with these corrected values to enhance the accuracy of the data for use in further operations.

After completing these steps, clean data should be generated for use in subsequent stages.

1. - Begin by partitioning the scanned data into smaller clusters.
2. - Examine each cluster to detect the presence of zero values.
3. - For clusters with zero values, compute the mean of the non-zero values
4. -Replace each zero value with this calculated mean.
5. -Update and return the clusters with these corrected values

The Mean of a Cluster of scanned distance excluding zeros is given by:

$$\text{Mean} = \frac{\sum_{i=1}^{n} d_i \quad \text{where} \quad d_i \neq 0}{\text{number of } d_i \text{ where } d_i \neq 0}$$

$$\text{Mean} = \frac{\sum_{i=1}^{n} d_i(d_i \neq 0)}{\sum_{i=1}^{n} \mathbf{1}(d_i \neq 0)}$$

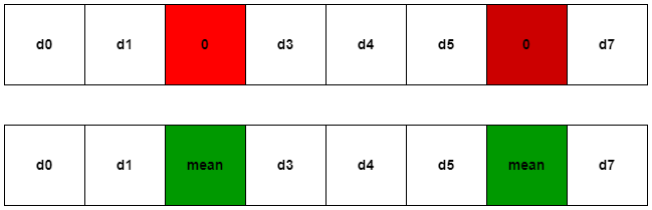


Figure 34: Methodology example for Substituting Mean Data with Zero Values

Depending on the non-zero mean in this example, the mean can be calculated accordingly.

$$\text{Mean} = \frac{d_0 + d_1 + d_3 + d_4 + d_5 + d_7}{6}$$

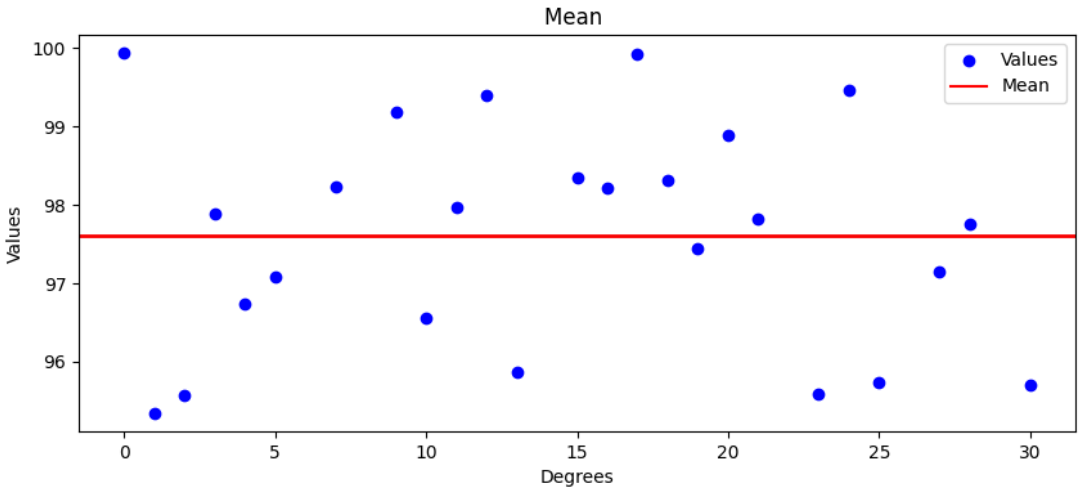


Figure 35: example illustrates the calculation of the mean for a cluster n.

Scan data clustering and object mapping and recognition

The cleaned and scanned data should consistently be segmented into clusters to accurately determine the position and distance of objects.

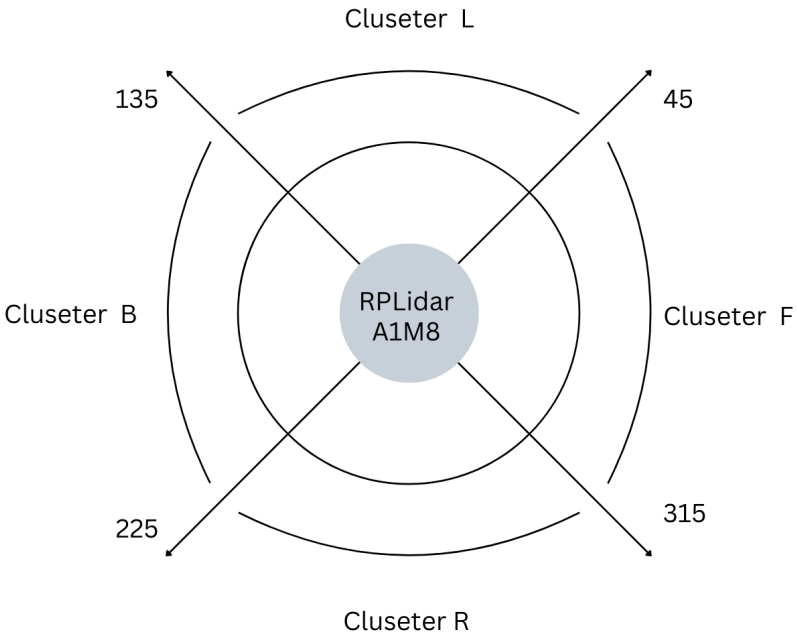


Figure 36: Data Representation of Dividing into Groups

To achieve the desired functionality, it is necessary to first process multiple data streams. Initially, the scanned data should be analyzed to identify and extract the primary four clusters, which correspond to the front, back, left, and right orientations as in Fig-36. Each cluster should be processed by dividing it into smaller clusters, as shown in Fig. 37, followed by the error-checking procedure as previously described. The mean of the non-zero values should be computed, and the approximate mean value should then be substituted for the zero values. At the end of the process, all sub-clusters should be combined into a single list that represents one of the primary clusters.

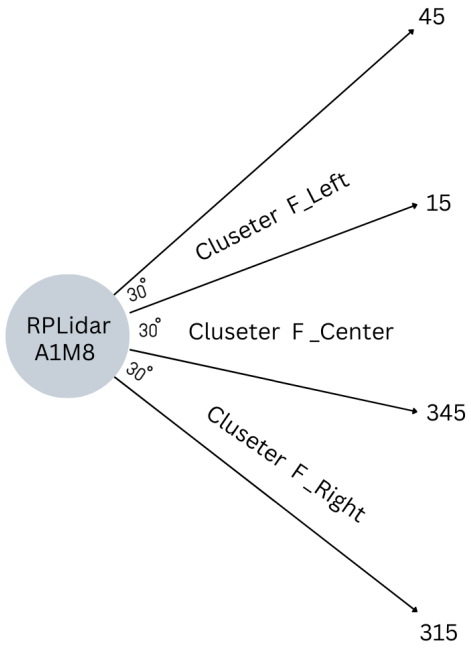


Figure 37: Example of how the primary cluster should be divided into sub-clusters

All objects can be detected from the aggregated sub-clusters. Once all objects are initialized, a determination must be made regarding which ones should be considered. This decision process is crucial in ensuring that only relevant objects are included in the subsequent analysis, thereby enhancing the overall efficiency and accuracy of the procedure. The objective of defining the object size is to determine the size of objects that could impact the driving process. This parameter is critical for assessing the minimum threshold at which objects can influence vehicle operation and safety. Each object must be evaluated against its size specifications to ensure that the object sizes are duly considered.The driving status, depending on the selected objects, should be determined, indicating whether the platform can continue driving, decrease speed, or suspend operation. Based on the data received, the platform engine must respond appropriately to ensure the platform operates within a safe environment.

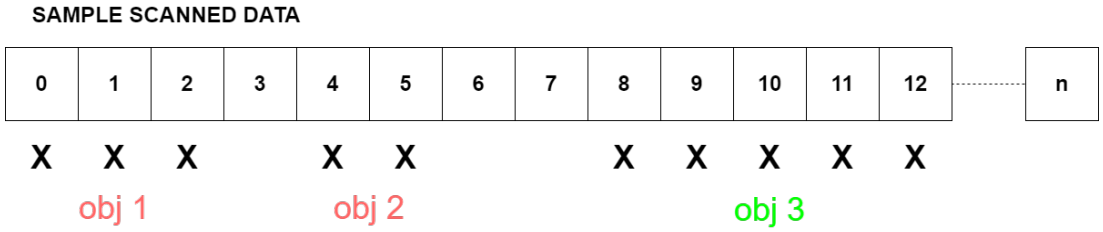


Figure 38: Example of how to decide if the object should be processed depending on the object size

Fig.38 illustrates the functionality of object size. In the example, an object size value of 5 is defined. This means any object with a value below 5 is disregarded, while those meeting or exceeding the object size value are considered for processing.

Heuristic 1: Object Comparison

```
foreach object in objects do
    if object < object size then
        continue // Pass
    else
        return object
    end
end
```

4.1.2 Camera Model

The camera class is designed to facilitate the capture of video frames through the PiCamera2 [9], a library tailored for Raspberry Pi camera modules. This class encapsulates the necessary operations to configure and retrieve video data, transforming it into a format suitable for real-time processing and display. Due to the restructuring of the Raspberry Pi 5's architecture, it is no longer possible to access the camera module directly through the OpenCV library. However, it is now possible to utilize the PiCamera2 library to access the hardware module and construct an interface, which can then be passed to the OpenCV library.

Class Initialization Upon instantiation, the Lidar class initializes several key attributes

picam2: It initializes the PiCamera2 object, enabling interaction with the camera hardware module.

config: The camera configuration is defined by two parameters: the main high-resolution and low-resolution settings. The main resolution is 2048x1536 pixels, which is optimal for high-quality image capture. In contrast, the low-resolution setting is 480x240 pixels. To achieve an appropriate configuration for accelerated processing and direct video capture or additional processing, these two parameters should be assigned to the **create_video_configuration** function from the **PiCamera2** library.

Subsequently, in order to implement the configuration and activate the camera, it is necessary to invoke the **configure()** and **start()** functions.

getVideo function has been developed with the objective of capturing a video frame from a camera and return it as object.

```
1
2 import cv2
3 from picamera2 import Picamera2
4
5 class Kamera:
6     def __init__(self):
7
8         self.picam2 = Picamera2()
9         config = self.picam2.create_video_configuration(main={"size": (2048, 1536)},
      lores={"size": (720, 480)}, encode="lores")
10         self.picam2.configure(config)
11         self.picam2.start()
12
13     def getVideo(self,display=False,size=(480,240)):
14         frame = self.picam2.capture_array()
15         frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
16         img=cv2.resize(frame,size)
17         if display:
18             cv2.imshow('Frame', img)
19
20             cv2.waitKey(1)
21         return img
```

4.1.3 Lane Detection

The lane detection functionality helps the system find the driving path by accurately identifying the lane. This ensures smooth and precise movement, improving overall control and efficiency. The lane detection process involves the following steps to find the path:

- Detect a specific color in the image.
- Identify and select the edges within the detected color region.
- Calculate the summation of the pixels in the selected area.

These steps can be illustrated using the following four functionalities:

- Image Threshold
- Image Wrapping
- Image Histogram
- Image Pixels Analysis

Image Threshold

The objective of setting the threshold is to identify and isolate specific colors. This process assists in determining the appropriate path for the platform to drive. The initial step involves converting the image to gray scale, enabling the isolation of black and white regions. Next, a Gaussian blur is applied to reduce noise in the image. A threshold function is then used to generate a bi-level image, facilitating the detection of color regions within the gray scale image. Due to challenges in the experimental environment, such as high brightness, consistently detecting the black color proved difficult. Therefore, I incorporated the ability to identify both black and white regions for more accurate detection. Moreover, the system can alternate between both colors based on the prevailing color in the region and define the path using either white or black, but not both simultaneously.

```
_, mskBlack = cv2.threshold(grayBlurred, 80, 255, cv2.THRESH_BINARY_INV)
_, mskWhite = cv2.threshold(grayBlurred, 200, 255, cv2.THRESH_BINARY)
```

THRESH_BINARY

$$dst(x,y) = \begin{cases} \text{maxValue} & \text{if } src(x,y) > T(x,y) \\ 0 & \text{otherwise} \end{cases}$$

THRESH_BINARY_INV

$$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > T(x,y) \\ \text{maxValue} & \text{otherwise} \end{cases}$$

Figure 39: Equations for thresholding from OpenCV documentation [8]

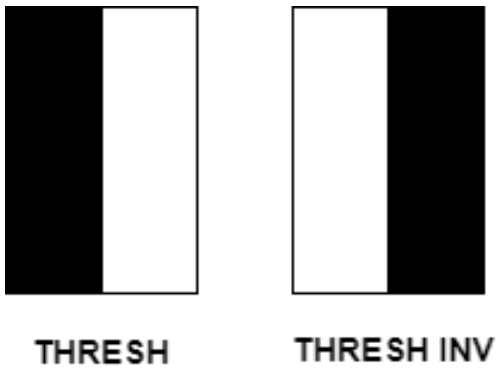


Figure 40: the result of the threshold channels Binary and Binary_INV [8]

After obtaining both the white and black masks, the next step is to determine which mask contains more pixels using the **cv2.countNonZero()** method. Based on this comparison, adaptive selection

is performed by choosing either the white or black mask, depending on which one has a higher pixel count, and using it as a reference in the subsequent steps.

```
def threshHolding(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    grayBlurred = cv2.GaussianBlur(gray, (5, 5), 0)
    _, mskBlack = cv2.threshold(grayBlurred, 60, 255, cv2.THRESH_BINARY_INV)
    _, mskWhite = cv2.threshold(grayBlurred, 180, 220, cv2.THRESH_BINARY)
    blackPixelCount = cv2.countNonZero(mskBlack)
    whitePixelCount = cv2.countNonZero(mskWhite)
    if blackPixelCount > whitePixelCount:
        return mskBlack
    else:
        return mskWhite
```

Image Wrapping

The objective of warping is to achieve a geometric transformation of the frame, making it consistent to facilitate the calculation of pixel summations. This can be implemented using the following functions. The pixel summation representation can be applied to three cases: straight, right, and left. If the number of pixels is equal on both sides, the platform should move forward. If there are more pixels on the left side compared to the right, the platform should be directed to the right. Conversely, if there are more pixels on the right side, the platform should be directed to the left

- The **'getPerspectiveTransform()'** function receive coordinates from the source image and the desired coordinates for the target image. It then returns the transformation matrix that maps the source image coordinates to the target image coordinates.
- The **'warpPerspective()'** function can be used to apply a perspective transformation to a frame, utilizing the transformation matrix obtained from the getPerspectiveTransform() function. The output is a transformed frame that reflects the applied perspective adjustment.

$$dst(x, y) = src\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right)$$

Figure 41: Equation for the `warpPerspective` function from OpenCV documentation [7].

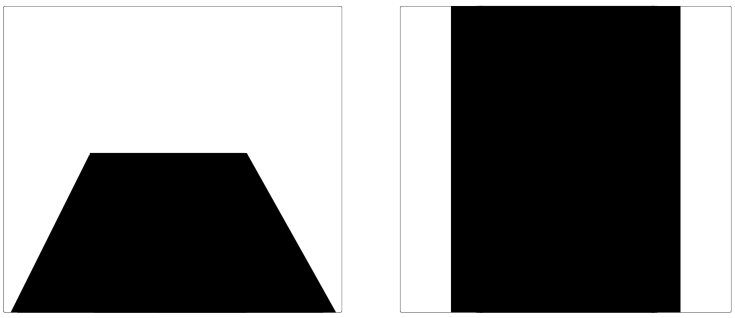


Figure 42: Applying the geometrical transformation

```
def wrapImg(img,points,w,h):
    point1 = np.float32(points)
    point2 = np.float32([[0,0],[w,0],[0,h],[w,h]])
    if inv:
        matrix = cv2.getPerspectiveTransform(point1,point2)
    imgWarp = cv2.warpPerspective(img,matrix,(w,h))
    return imgWarp
```

Image histogram and pixels summation

In the histogram function, the objective is to identify the curve along the path and determine its level. Based on this curve value, the platform direction can then be established. The underlying concept is straightforward: a center line is defined at the midpoint of the frame, and the pixels on both sides of this line are calculated and compared to guide the platform movement.

The pixel summation representation can be applied to three cases: forward, right, and left. If the number of pixels is same on both sides, the platform should move forward. If there are more pixels on the left side compared to the right side, the platform should be directed to the right. Conversely, if there are more pixels on the right side, the platform should be directed to the left.

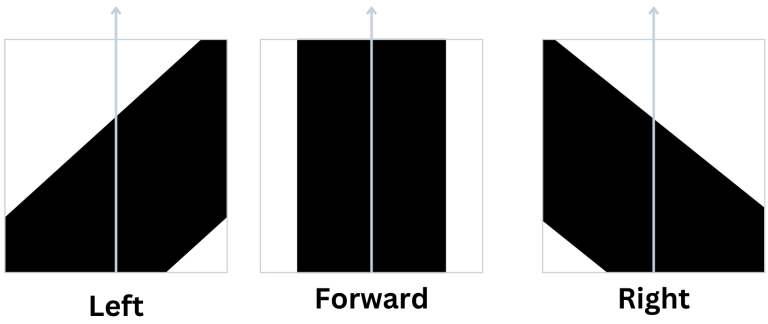


Figure 43: Cases representation

As described earlier, the function begins by adjusting with the center line to compare both sides, while the frame is passed as an input argument. All of the pixels in the image columns are summed using **np.sum(image, axis=0)**, where each cell in the resulting 1D array represents the vertical column summation.

The frame dimensions are 240 x 480, with the center point located at the midpoint along the frame's width, which is 240. Thus, the center value can be referenced using image.shape[0] as an index.

To enhance accuracy and minimize noise, the summation array should be compared to a minimum value. AAdditionally, each cell within the array needs to be evaluated against this minimum value. InInitially, define the minimum value as follows: set maxValue to np.max(np.sum(image, axis=0)), then calculate the minimum value by multiplying 'maxValue' with an accuracy parameter, which is a decimal between 0.0 and 1.0 representing a percentage. Subsequently, use 'indexArr = np.where(sumArray >= minValue)' to identify and store the indices of all elements in the summation array that exceed the minimum value in 'indexArr'.At the end, the average should be calculated for comparison with the central value. The point is determined by calculating the mean of 'indexArray' using 'np.average'.

By applying the equation "center point - average," if the result of the subtraction equals zero, the platform should proceed forward. If the result is negative, the platform should turn left; if it is positive, the platform should turn right. Let C be the center point, and A be the average.

Define the equation: Direction = C - A

$$\text{Direction} = \begin{cases} \text{Forward} & \text{if } D = 0, \\ \text{Left} & \text{if } D < 0, \\ \text{Right} & \text{if } D > 0. \end{cases}$$

To achieve smoother movement and eliminate abnormal motion behavior, values are appended to a list, and the average value is calculated at each iteration. After each calculation, the size of the list is compared against a predefined limit. If the list exceeds this limit, the oldest value is removed.

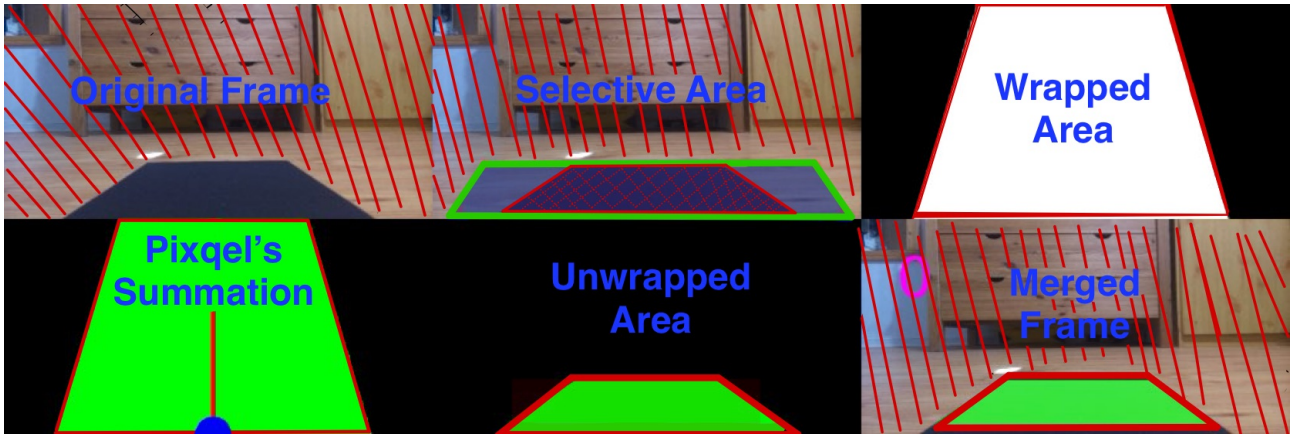


Figure 44: illustration the complete process of lane detection, including all stages

4.1.4 Object Detection

In the object detection phase, I employed a black-box machine learning model to train the input data, enabling the platform to accurately detect all previously trained objects. During the data collection phase, various cases were considered, such as differentiating object types whether it was a traffic light, a vehicle, or a pedestrian and recognizing traffic signs like stop signs and speed limits (e.g., 30 or 50 km/h zones). These were the primary objects collected to train the model.

Through data annotation, I labeled objects within photos to facilitate their recognition and utilization by machine learning models, enabling the models to generate accurate predictions.

Data collection and annotation:

- During the data collection and annotation process, I gathered my own data to serve as input for training the TensorFlow Lite model.

Figure 45: Data type that should be collected

- After data collection, the data should be cleaned by manually removing low-quality and noisy images.

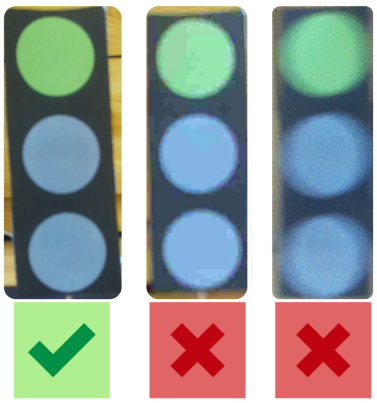


Figure 46: low-quality and noisy images

- Using the LabelImg annotation tool, all images were annotated in PASCAL VOC format to be compatible with the Model Maker format in TensorFlow.

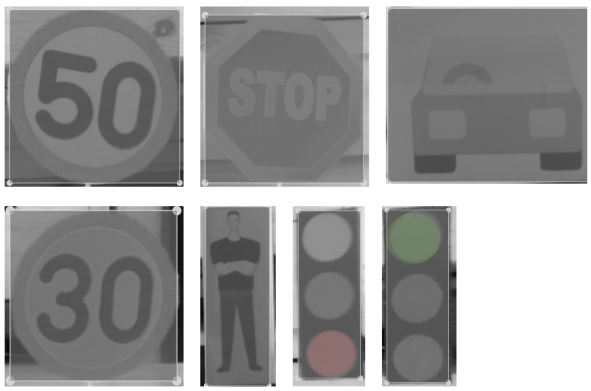


Figure 47: Enter Caption

Utilize TensorFlow's Model Maker API to train a TensorFlow Lite model:

By utilizing Google Colab's Plus plan, the training data can be processed using a TPU (Tensor Processing Unit). The advantage of employing a TPU lies in its superior speed compared to conventional GPUs.

```
import numpy as np
import os

from tflite_model_maker.config import QuantizationConfig
from tflite_model_maker.config import ExportFormat
```

```
from tflite_model_maker import model_spec
from tflite_model_maker import object_detector

import tensorflow as tf
assert tf.__version__.startswith('2')

tf.get_logger().setLevel('ERROR')
from absl import logging
logging.set_verbosity(logging.ERROR)
trainData = object_detector.DataLoader.from_pascal_voc(
    'images/train',
    'images/train',
    ['30Zone', '50Zone','red','green','car','humen','stop']
)
validationData = object_detector.DataLoader.from_pascal_voc(
    'images/validate',
    'images/validate',
    ['30Zone', '50Zone','red','green','car','humen','stop']
)
specifications = model_spec.get('efficientdet_lite0')
model = object_detector.create(trainData, model_spec=specifications, batch_size=4,
    train_whole_model=True, epochs=100, validation_data=validationData)
model.evaluate(validationData)
model.export(export_dir='.', tflite_filename='result.tflite')
```

The code outlines the entire process, from model initialization and configuration to its export, following these steps in Fig[48] .

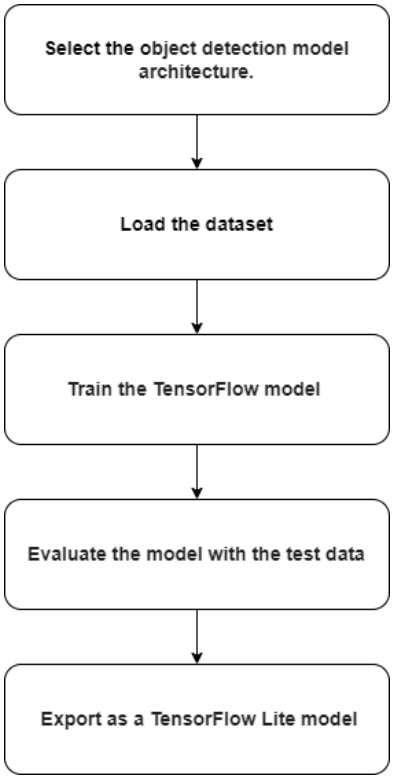


Figure 48: Training Steps for an Object Detection Model [9]

Apply the model in the project for object detection:

By using the MediaPipe object detection library, developed by Google AI, Edge is capable of identifying objects in various types of images, videos, or live captures. This functionality is achieved through a pre-trained machine learning model. As it processes continuous input frames, the output consists of a list of detected objects that the model has been trained to recognize and that appear within the input frames [1].

```
    base_options = python.BaseOptions(model_asset_path=self.model)
    options = vision.ObjectDetectorOptions(
            base_options=base_options,
            running_mode=vision.RunningMode.LIVE_STREAM,
            max_results=self.max_results,
            score_threshold=self.score_threshold,
            result_callback=self.save_result
        )
    detector = vision.ObjectDetector.create_from_options(options)
    temp={"category_name":None,"probability":0}
    while True:
        image = self.camera.getVideo()
        frame_count += 1
        if frame_count % frame_skip != 0:
```

```
15              await asyncio.sleep(0.1)
16              continue
17          image = cv2.resize(image, (self.width, self.height))
18          image = cv2.flip(image, 1)
19          rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
20          mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb_image)
21          detector.detect_async(mp_image, time.time_ns() // 1_000_000)
22          fps_text = 'FPS = {:.1f}'.format(self.FPS)
23          text_location = (self.left_margin, self.row_size)
24          current_frame = image
25          cv2.putText(current_frame, fps_text, text_location, cv2.FONT_HERSHEY_DUPLEX,
26                      self.font_size, self.text_color, self.font_thickness, cv2.LINE_AA
    )
27          if self.detection_result_list:
28              for detection in self.detection_result_list[0].detections:
29                  category = detection.categories[0]
30                  category_name = category.category_name
31                  probability = round(category.score, 2)
32                  temp["category_name"] = category_name
33                  temp["probability"] = probability
34              self.detected_object_info["category_name"] = temp["category_name"]
35              self.detected_object_info["probability"] = temp["probability"]
```

This code is from Mediapipe for object detection. it has modified to function with the Raspberry Pi 5 camera module and the AsyncIO library to be able detect an object in parallel with other processes.

4.2 Scenarios

To fulfill the objectives of the experiments, multiple scenarios are proposed:

4.2.1 Scenario 1: Complete control via Xbox controller.

The scenario presents a basic test to ensure the functionality of the LiDAR system. The scenario begins with a server attempting to establish connections with multiple clients, including an Xbox controller and a Raspberry Pi. Upon successful connection, the Xbox controller is responsible for sending control commands to the Raspberry Pi. Specifically, if the B button is pressed, the platform will stop and wait for the next command. If the A button is pressed, the platform will continue to move as long as the A button is held down, provided the platform is in a "ready to drive" state. The Raspberry Pi will then await further commands—forward, backward, left, or right. Based on these commands, the Raspberry Pi will direct the platform accordingly, while continuously considering collision avoidance using the LiDAR functionality.

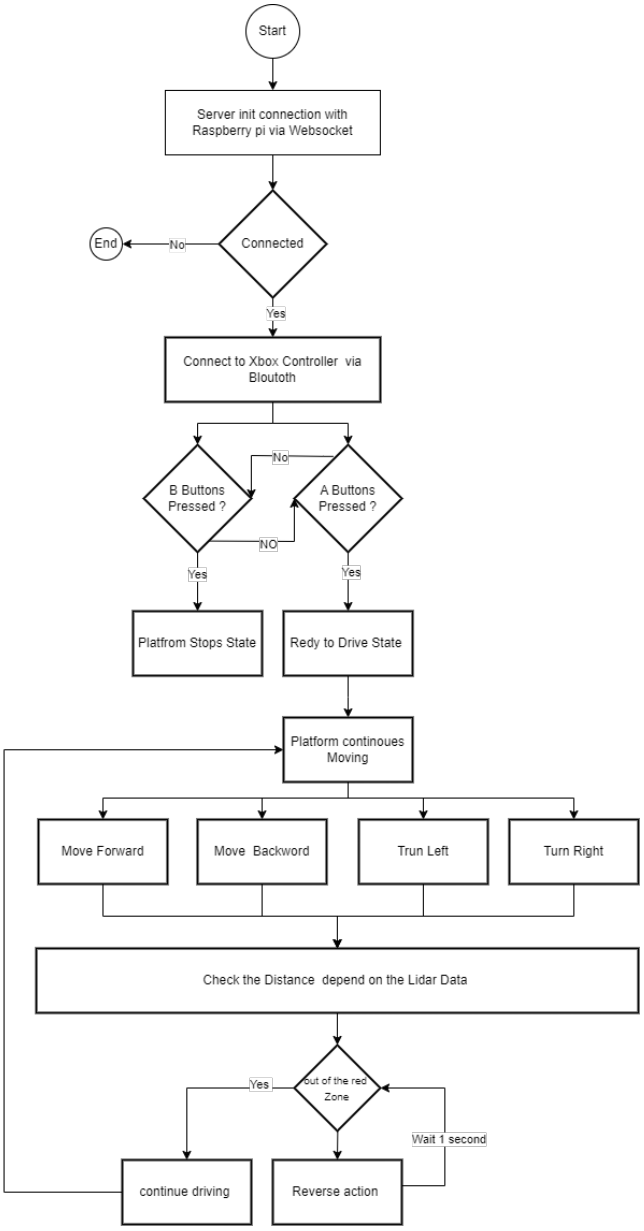


Figure 49: Scenario 1 Flowchart

After confirming that the primary LiDAR model is performing well in the first scenario, it is now appropriate to delve deeper and implement the other subsequent scenarios.

4.2.2 Scenario 2: Autonomous driving based on data received from the LiDAR and the lane detection model

The second scenario will be more complex, as it incorporates both the lane detection model and the LiDAR model for identifying surrounding objects in the environment. Additionally, the platform

is required to navigate along a specific path guided by the lane detection model. Based on these integrated models, the platform's driving status can be categorized as follows:

- **0:** An object is detected directly in front of the platform.
- **1:** Proceed without any issues.
- **2:** An object is detected very close to the platform; speed should be reduced accordingly.

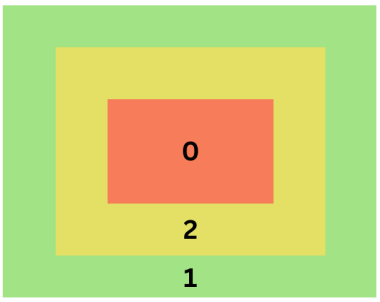


Figure 50: Representation of Driving Safety Status

The workflow for a system for the scenario of the Autonomous driving based on LiDAR ,lane detection with object detection can be represented with the following steps :
After **initializing all models** and establishing a connection between the platform and the dashboard through a WebSocket, real-time data communication was enabled.
The LiDAR object is initializedd, which is responsible for detecting and measuring distances to objects around the platform in a separate flow of execution. This means that is the LiDAR object runs in a separate task.
The system checks whether the connection is established with the LiDAR Device . If the connection is not passed, it keeps retrying until a successful connection is achieved.
Once connected, the system enters its **Main operational loop**. This loop continues to run until an interruption occurs (possibly a command from the user 'ctrl +c' or a detected fault).
Initialize the Lane Detection model: using asyncio to achieve parallelism, enabling the simultaneous execution of multiple operations. Lane Detection" instance is involved in keeping the platform within its designated path as mentioned before.

Data Request and Processing:
The system concurrently requests data from several sources:
Camera Model: responsible for image data used for tasks lane detection and the object detection .
LiDAR Model: Used to validate and enhance the environmental awareness by processing distance measurements.
Send Data to Server via WebSocket: This step streams the processed data to a server for visualizing the data .

Perform Lane Detection: using data from camera to detect the lane, guiding the platform path accordingly.

LiDAR Data Validation: The LiDAR data is validated to check whether it matches the expected ranges and distances . If the data is invalid, the system reverts to a step to make sure that is the platform cant take any action without valid data from the LiDAR model .

Monitor Ranges and Analyze Distances: Once the LiDAR data is validated, the system actively monitors ranges and analyzes distances to maintain safe driving or operational conditions.

Check Driving Status: The system evaluates the driving status and decides between the three states mentioned previously. The flowchart below illustrates all the steps involved in the second scenario of autonomous driving, which is based on data obtained from LiDAR and lane detection systems.

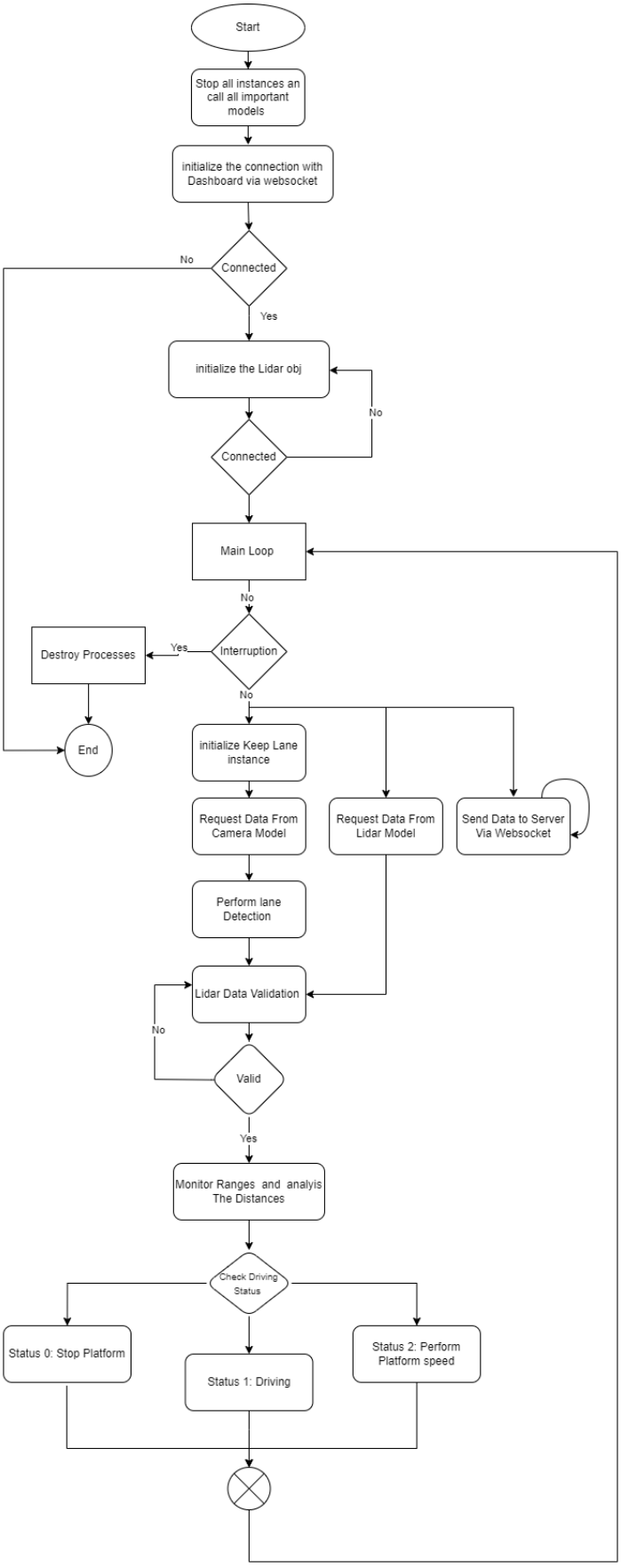


Figure 51: Scenario 2 Flowchart

4.2.3 Scenario 3: Autonomous driving based on LiDAR ,lane detection with object detection :

In the third scenario, the complexity is increased due to the use of the machine learning model responsible for object detection. Following this detection, the platform's speed must be determined based on the identified object. Once the driving status is established, the platform may need to stop in certain situations, such as at a red traffic light or when a pedestrian is present. However, in specific cases e.g. 30 km/h zone, the platform should not stop, and these cases are duly considered.

Detected Zone/Object	Drive Status 0	Drive Status 1	Drive Status 2
Pedestrian	0.0	Speed	0.0
Vehicle	0.0	Speed	0.0
30 Zone	Speed - 0.4	Speed	Speed - 0.3
50 Zone	Speed - 0.3	Speed	Speed - 0.2
Stop	0.0	Speed	0.0
Red Light	0.0	Speed	0.0
Green Light	Speed	Speed	Speed

Table 13: illustrates how the performance speed varies based on the detected object.

The platform's driving status can be classified similarly to the description in the second scenario, with the addition of initializing and invoking object detection, as well as adjusting speed based on the detected objects.
Initialize the Lane Detection instance and the object detection model using asyncio to achieve parallelism, enabling the simultaneous execution of multiple operations.The ML model is involved in object detection to enable the platform to identify objects in parallel with the Lane Detection. Data Request and Processing: In addition to calling the Camera Model, LiDAR Model and Send Data to Server via WebSocket.
Request data from **Object Detection Model**: This model processes the detected objects (e.g., cars, pedestrians) to understand their positions relative to the platform.
After validating the LiDAR data, monitor the ranges and analyze the distances then perform platform speed adjustments: Based on the analysis, the platform adjusts its speed accordingly, which could be slowing down or speeding up . The flowchart below illustrates all the steps involved in the Third scenarioincluding all processes from the second scenario , which is based on data obtained from LiDAR ,lane detection and object detection models .

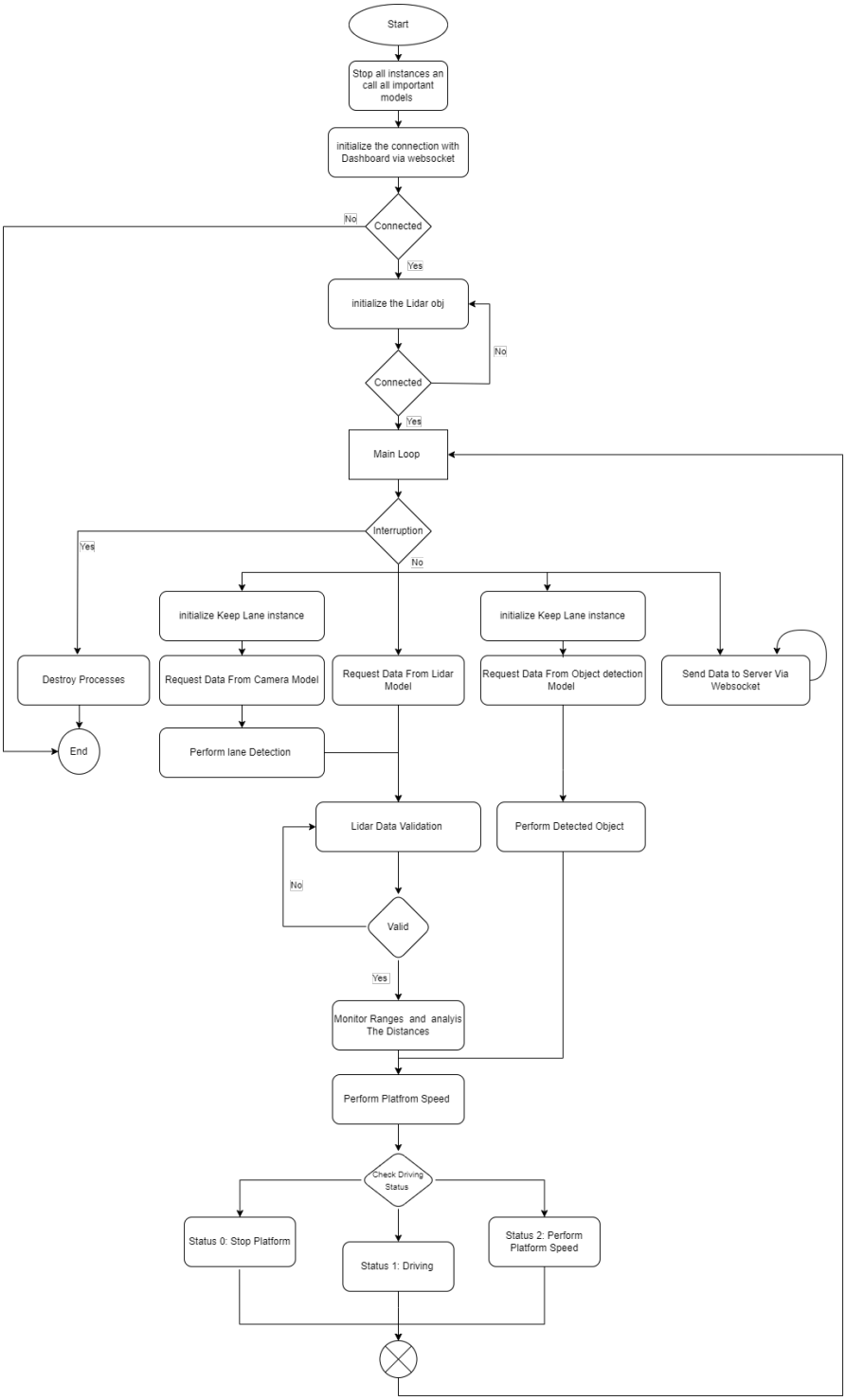


Figure 52: Scenario 3 Flowchart

4.3 Dashboard:

The analyzed data, including aspects such as speed, detected objects, platform status, and more, should be simulated (visualized) as illustrated in Figure 53.

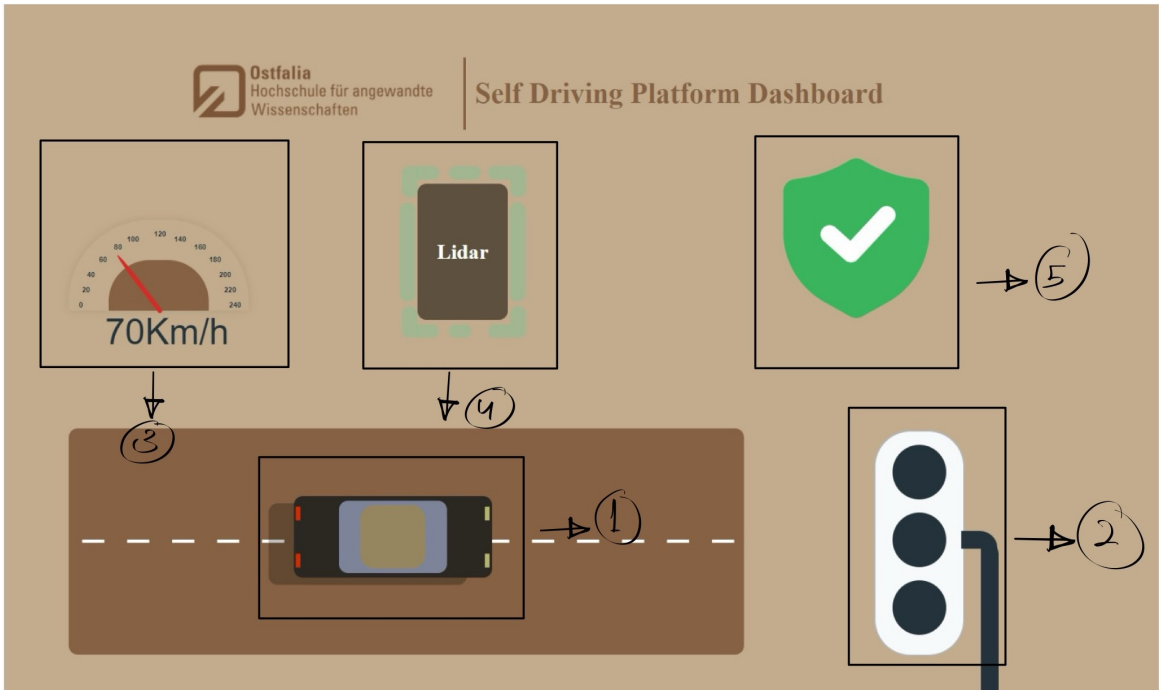


Figure 53: Data visualization

1. The platform's operational status (driving or stopped) is determined by its speed. A speed of 0.0 indicates a stopped state, while any other speed reflects driving status.
2. Traffic light representation indicates the status of the main traffic light. If a detected object is a traffic light, it is shown in either green or red.
3. The platform status is represented based on speed,represented using a speed thermometer. The maximum speed is 70 km/h(speed ratio of 1.0), with corresponding platform speeds displayed: 50 km/h (speed ratio of 0.75), 30 km/h (speed ratio of 0.6), and 0 km/h (speed ratio of 0).
4. LiDAR data visualization represents the distance of detected objects, categorized into different safety zones: green (safe), yellow (caution), and red (danger).
5. The object detection section continuously shows all detected objects, updating their status in real-time.

As previously mentioned, all data communication is conducted via WebSockets, which enables both connection and data exchange in real time. The data representation is based on the five sections outlined earlier. Technically, the dashboard is implemented using HTML5, CSS3, and JavaScript.

5 Evaluation and Conclusion

5.1 Evaluation:

With the available components, the experiment was ultimately successful; however, limited resources presented certain challenges.

Noise and Lower Data Accuracy:

The data from the LiDAR A1M8 can sometimes be abnormally noisy , This can make it harder to get clean and consistent data for accurate object detection The LiDAR's accuracy tends to be lower, or at least the interface library is compared to higher-end LiDAR systems. The accuracy decreases with continuous measurements , making it less reliable in performing the objects in the environment.One key reason of the data weakness is the frequency of operation. To function within a scanning frequency range of 5.5Hz to 10Hz, the motor requires a separate 9V power supply[4]. Currently, the operating frequency is set at 5.5Hz.

Indoor and Low-Light Conditions:

The LiDAR is specifically designed for indoor environments and is therefore not well-suited for outdoor applications where strong light sources, such as intense sunlight, are present. Exposure to high levels of sunlight can reduce measurement ranges or result in inaccurate data, making the sensor less reliable in such conditions [4].

Range Limitation:

The LiDAR A1M8 has a maximum range of approximately 12 meters, which is relatively limited compared to more advanced LiDAR models. This constraint can restrict its effectiveness in larger environments . While the current experiment may not require the maximum range, but it is important to consider this limitation when planning future upgrades or platform developments, as extended range capability would provide greater flexibility and potential for expanded use cases. A suggested that can be implemented is to replace the current LiDAR with a higher-quality scanning LiDAR that supports multiple lines instead of a single line. This would enhance the system's ability to detect objects in 3D, improve the accuracy of distance measurements, and provide coverage across multiple 2D layers, offering a more comprehensive perspective and better environmental coverage.

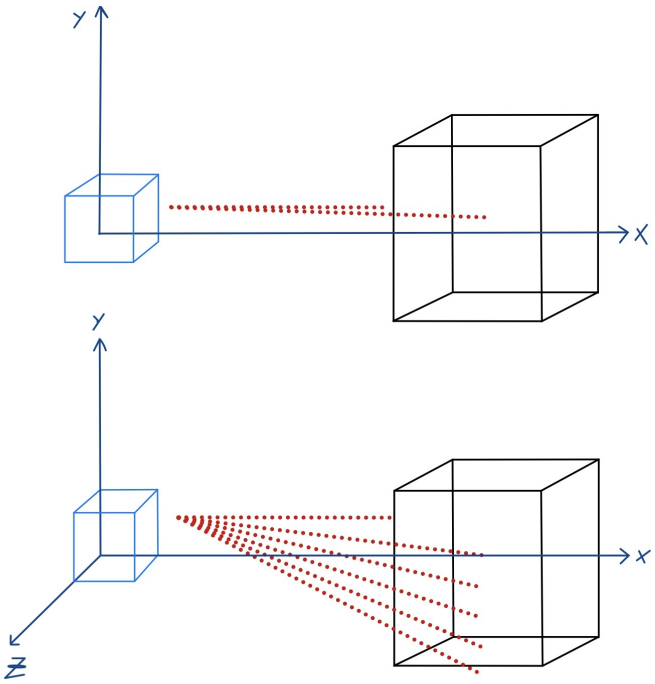


Figure 54: Conceptual Comparison Between Single-Line LiDAR 2D and Multi-Line LiDAR 3D

As an alternative solution, the platform could be equipped with integrated ultrasonic sensors sensors to detect objects that are not covered by the LiDAR. Since the LiDAR is mounted on top of the platform and performs a 2D scan, any object with a height of less than 15 cm may not be detected by the LiDAR. The ultrasonic sensor would provide coverage for these low-lying obstacles, enhancing the overall detection capability of the platform.

One significant enhancement that could be made to the platform's design involves modifying the front wheels by integrating them with a steering mechanism. This modification would improve the platform's ability to maintain its intended trajectory, addressing difficulties encountered in controlling the platform during deviations. These control challenges are particularly pronounced when the platform experiences high loads on the DC motors during left or right turns. This enhancement could lead to smoother movement during turns, while also improving the balance and accuracy of lane detection.

Measuring the distance to a specific object in the environment depends on the current advancements in monitor range for range definition and object distance measurement using LiDAR device. With these developments, every object can be detected and its direction identified, without requiring prior definition. As future work for improvement and optimization, combining object detection machine learning models with monitor range functionality should involve considering the object's position and type during data collection and training . This approach will enhance the accuracy and efficiency of the detection process and incorporate the ability to integrate object type and position, including the distance from the platform, thereby enhancing the overall accuracy and effectiveness of the detection

system.Additionally, assembling a multi-camera system could provide 360-degree vision, which would enhance the accuracy of environmental monitoring and ensure the detection of every object within the environment and around the platform .

5.2 Conclusion:

The experiment successfully achieved its objective of constructing and developing a self-driving platform utilizing the latest version of the Raspberry Pi 5, the Picam2 camera model, and the LiDAR A1M8 as the primary sensor. The integration of these components resulted in the creation of an initial functional prototype capable of collision avoidance.

During the implementation, multiple scenarios were employed to evaluate the system's performance. The platform demonstrated a robust capability to avoid obstacles and detect objects within the environment. Real-time data processing from the LiDAR and camera systems was essential in ensuring the system's operational reliability.

The limitation is particularly in handling the error rate. Future improvements should focus on enhancing error rate correction by implementing machine learning techniques to increase accuracy.As system components, such as the LiDAR sensor, are expanded and upgraded, it will become capable of performing more complex functions.

In conclusion, this thesis advances the expanding field of autonomous systems by offering practical experiments on the previously discussed concepts and demonstrating their integration to achieve self-driving platforms. The combination of LiDAR and camera technologies, coupled with processing algorithms, holds significant potential for future research and development in applications for autonomous platforms.

Bibliography

[1] , Google AI E. : *Object detection task guide | Edge.* – URL https://ai.google.dev/edge/mediapipe/solutions/vision/object_detector

[2] , Shanghai Slamtec C. : *Interface Protocol and Application Notes.* 04 2016. – URL https://bucket-download.slamtec.com/6fad02c42af6da33f89fbc043c5f165e2b222e0d/rplidar_interface_protocol_en.pdf. – Zugriffsdatum: 2024-07-10

[3] , Shanghai Slamtec C. : *RPLIDAR A1.* 10 2020. – URL https://bucket-download.slamtec.com/d1e428e7efbdcd65a8ea111061794fb8d4ccd3a0/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf

[4] , Shanghai Slamtec C. : *RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner Development Kit User Manual.* 08 2020. – URL https://www.slamtec.ai/wp-content/uploads/2023/11/LM108_SLAMTEC_rplidarkit_usermaunal_A1M8_v2.2_en.pdf. – Zugriffsdatum: 2024-07-08

[5] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Corrado, Greg S. ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Goodfellow, Ian ; Harp, Andrew ; Irving, Geoffrey ; Isard, Michael ; Jia, Yangqing ; Jozefowicz, Rafal ; Kaiser, Lukasz ; Kudlur, Manjunath ; Levenberg, Josh ; Mané, Dandelion ; Monga, Rajat ; Moore, Sherry ; Murray, Derek ; Olah, Chris ; Schuster, Mike ; Shlens, Jonathon ; Steiner, Benoit ; Sutskever, Ilya ; Talwar, Kunal ; Tucker, Paul ; Vanhoucke, Vincent ; Vasudevan, Vijay ; Viégas, Fernanda ; Vinyals, Oriol ; Warden, Pete ; Wattenberg, Martin ; Wicke, Martin ; Yu, Yuan ; Zheng, Xiaoqiang: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* 2015. – URL https://www.tensorflow.org/. – Software available from tensorflow.org

[6] Aslam, Muhammad S. ; Aziz, Muhammad I. ; Naveed, Kanwal ; Zaman, Uzair K. uz: An RPLiDAR based SLAM equipped with IMU for Autonomous Navigation of Wheeled Mobile Robot. In: *2020 IEEE 23rd International Multitopic Conference (INMIC)*, 2020, S. 1–5

[7] Documentation, OpenCV: *Image - Geometric Image Transformations - OpenCV Documentation.* https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html. 2024. – OpenCV version 4.10.0

[8] Documentation, OpenCV: *Image Thresholding - OpenCV Documentation.* https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html. 2024. – OpenCV version 4.10.0

[9] Foundation, Raspberry P.: *PiCamera2 Manual.* https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf. 2024

[10] Holstein, Tobias ; Dodig-Crnkovic, Gordana ; Pelliccione, Patrizio: Real-world Ethics for Self-Driving Cars. In: *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2020, S. 328–329

[11] Hyun-je: *pyrplidar.* https://github.com/Hyun-je/pyrplidar. 2023

[12] Hyun-je: *pyrplidar*. https://github.com/Hyun-je/pyrplidar/tree/master. 2024

[13] Industries, Adafruit: *Adafruit CircuitPython RPLIDAR*. https://github.com/adafruit/Adafruit_CircuitPython_RPLIDAR. 2024

[14] Liu, Shiying ; Nakanishi, Takafumi: Obstacles Detection and Motion Estimation by Using Multiple Lidar Sensors Data. In: *2022 13th International Congress on Advanced Applied Informatics Winter (IIAI-AAI-Winter)*, 2022, S. 196–201

[15] Muslim, H. ; Endo, S. ; Imanaga, H. ; Kitajima, S. ; Uchida, N. ; Kitahara, E. ; Ozawa, K. ; Sato, H. ; Nakamura, H.: Cut-Out Scenario Generation With Reasonability Foreseeable Parameter Range From Real Highway Dataset for Autonomous Vehicle Assessment. In: *IEEE Access* 11 (2023), S. 45349–45363

[16] Raharja, Kadek Agus W. ; Rusmin, Pranoto H.: Detection and Measurement of Object Distances Using the Combination Function of 2D LiDAR Sensor and Camera. In: *2023 International Conference on Electrical Engineering and Informatics (ICEEI)*, 2023, S. 1–5

[17] Rivai, Muhammad ; Purwanto, Djoko ; Razak, Arsya ; Hutabarat, Dony ; Aulia, Dava: Implementation of Light Detection and Ranging in Vehicle Braking System. In: *2021 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, 2021, S. 256–260

[18] Roboticia: *Roboticia RPLIDAR*. https://github.com/Roboticia/RPLidar?tab=readme-ov-file. 2024

[19] Robotics, Skoltech: *Skoltech Robotics RPLIDAR*. https://github.com/SkoltechRobotics/rplidar. 2024

[20] Yusefi, Abdullah ; Durdu, Akif ; Toy, Ibrahim: Camera/LiDAR Sensor Fusion-based Autonomous Navigation. In: *2024 23rd International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2024, S. 1–6

Zusammenfassung:

1. Einführung:

Die fortlaufende Entwicklung in der Technologie autonomes Fahrzeuge hat das Potenzial, den Transport zu verändern, indem sie die Sicherheit verbessert, die Leistung steigert und menschliche Fehler reduziert. Diese Arbeit präsentiert das Design und die Implementierung einer selbstfahrenden Plattform unter Verwendung eines Raspberry Pi 5, kombiniert mit einer Picam 2-Kamera und einem LiDAR A1M8-Sensor von Slamtec. Der Fokus liegt auf der Entwicklung eines Kollisionsvermeidungssystems unter Einsatz der LiDAR-Technologie, integriert mit Spurenerkennung und Objekterkennung, um die Fähigkeit des Fahrzeugs zur sicheren Navigation zu verbessern.

Die zunehmende Nutzung autonomer Plattformen erfordert ein Gleichgewicht zwischen technischen Fortschritten und ethischen Überlegungen. Diese ethischen Fragestellungen sind besonders wichtig, wenn es um Verkehrsregeln und die Auswirkungen autonomer Systeme auf das menschliche Leben geht. Daher muss die Entwicklung selbstfahrender Fahrzeuge die Sicherheit in den Vordergrund stellen und sicherstellen, dass diese Technologien innerhalb eines Rahmens arbeiten, der sowohl technisch fundiert als auch ethisch verantwortungsvoll ist.

Ziele:

- Das Verhalten von LiDAR-Sensoren verstehen.
- Einen Algorithmus zur Kollisionsvermeidung entwickeln.
- Die auf LiDAR basierende Kollisionsvermeidung mit kamerabasierter Objekterkennung, die auf maschinellen Lernmodellen beruht, kombinieren.
- Einen funktionierenden Prototyp mit praktischen Anwendungen demonstrieren.

2. Verwandte Arbeiten

Die Forschung im Bereich des autonomen Fahrens hat zur Entwicklung verschiedener Kollisionsvermeidungssysteme geführt. Traditionelle Methoden, wie Ultraschallsensoren, haben Einschränkungen in der Umweltabdeckung und Genauigkeit. Kameras bieten eine hohe Auflösung und Farberkennung, aber es fehlt ihnen an Tiefeninformationen, und sie sind anfällig für Lichtverhältnisse. LiDAR-Technologie, insbesondere 2D- und 3D-Systeme, bietet eine hohe Präzision und schnelles Scannen, was sie zu einer überlegenen Option für die Echtzeit-Objekterkennung und Kartierung macht.

Wichtige Punkte:

- **Kameras:** Bieten detaillierte visuelle Informationen, haben jedoch Schwierigkeiten mit der Tiefenwahrnehmung und sind lichtempfindlich.
- **LiDAR-Technologie:** Bevorzugt aufgrund ihrer Fähigkeit, Umgebungen schnell zu scannen und genaue Entfernungsangaben zu liefern, was für die Navigation und Hindernisvermeidung unerlässlich ist.

Die Arbeit untersucht die Integration von 2D-LiDAR und Kameras, um die Stärken beider Technologien zu nutzen und die Sicherheit und Effizienz autonomer Plattformen zu verbessern.

3. Hardware

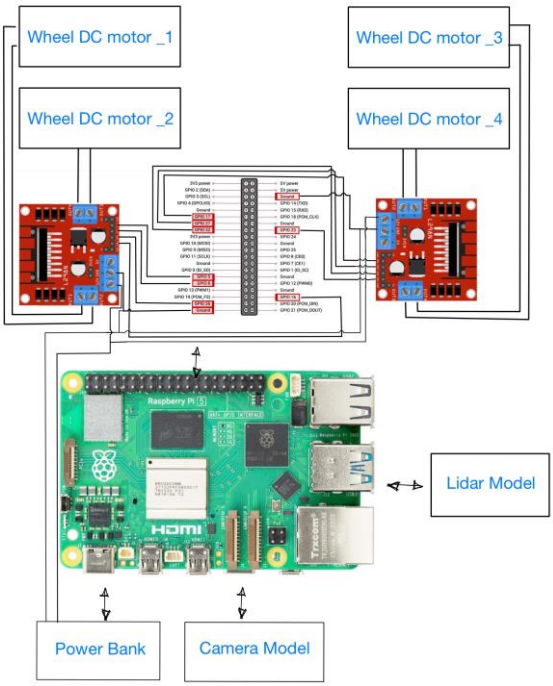
Die Hardwarekonfiguration der Plattform konzentriert sich auf einen Raspberry Pi 5, der als Hauptsteuerungsgerät tätig ist. Dieses System ist darauf ausgelegt, Eingaben von LiDAR und Kamera zu verarbeiten und die Bewegungen der Plattform über Gleichstrommotoren zu steuern.

Komponenten und Funktionen:

- **Raspberry Pi 5:** Dient als zentrale Verarbeitungseinheit, die Eingaben und Ausgaben verwaltet.
- **Gleichstrommotoren (4x):** Sorgen für die Mobilität, indem sie die Räder antreiben, gesteuert durch 2x L298N-Motortreiber-Module.
- **LiDAR A1M8:** Ein 360-Grad-2D-Laserscanner, der Echtzeit-Entfernungsmessungen und Objekterkennung innerhalb eines 6-Meter-Bereichs ermöglicht.
- **Raspberry Pi Kamera Modul 2:** Nimmt Bilder für die Spurenerkennung und Objekterkennungsaufgaben auf.
- **Stromversorgung:** Eine Powerbank gewährleistet den kontinuierlichen Betrieb, indem sie den Raspberry Pi und die Motortreiber mit tragbarer Energie versorgt.

Konnektivität:

- **GPIO-Pins:** Werden verwendet, um den Raspberry Pi mit den Motortreibern zu verbinden.
- **USB-Schnittstelle:** Verbindet das LiDAR mit dem Raspberry Pi.
- **CSI-Port:** Verbindet das Kameramodul mit dem Raspberry Pi, sodass es Bilder aufnehmen und verarbeiten kann.



4. Implementierung

Die Implementierung des Systems ist in mehrere Schlüsselmodelle unterteilt, die jeweils spezifische Aspekte des autonomen Fahrprozesses abdecken. Diese Modelle arbeiten zusammen, um eine robuste und effiziente Plattform zu schaffen.

LiDAR-Modell:

- **Initialisierung:** Das LiDAR-Modell initialisiert eine Instanz zur Interaktion mit der Hardware. Es richtet Datenstrukturen ein, um Entfernungsdaten zu speichern, und steuert den kontinuierlichen Betrieb des LiDAR-Sensors.

- **Datenverarbeitung:** Gescannte Daten werden verarbeitet, um Fehler und Nullwerte herauszufiltern. Diese werden durch berechnete Mittelwerte benachbarter Datenpunkte ersetzt, um die Genauigkeit zu gewährleisten.
- **Clustering und Objekterkennung:** Gescannte Daten werden in Gruppen unterteilt, die verschiedenen Richtungen (vorne, hinten, links, rechts) entsprechen. Objekte innerhalb dieser Gruppen werden basierend auf Größe und Nähe identifiziert, was die Navigationsentscheidungen der Plattform beeinflusst.

Kamera-Modell:

- **Videoaufzeichnung:** Nutzt die PiCamera2-Bibliothek, um Videobilder aufzunehmen, die anschließend für die Spurenerkennung und Objekterkennung verarbeitet werden.

Spurenerkennung:

- **Thresholding:** Konvertiert Bilder in Graustufen, wendet einen Gaußschen Weichzeichner an, um Rauschen zu reduzieren, und nutzt Thresholding, um wichtige Merkmale zu isolieren. Bestimmte Farben im Bild werden erkannt, um Fahrspuren zu identifizieren. Das System wählt adaptiv zwischen Schwarz-Weiß-Masken basierend auf der Pixeldichte.
- **Bildverzerrung:** Wendet geometrische Transformationen an, um die Bildperspektive zu standardisieren und eine genaue Spurenerkennung zu erleichtern.
- **Histogramm Analyse:** Fasst Pixeldaten zusammen, um die Position der Fahrspur zu bestimmen, und leitet die Plattform basierend auf der erkannten Fahrspurkrümmung dazu, vorwärts, nach links oder nach rechts zu fahren.

Objekterkennung:

- **Datensammlung und -annotation:** Das System verwendet ein Black-Box-Maschinenlernmodell, das auf einem Datensatz von Bildern trainiert wird, die mit Objekten wie Verkehrsschildern, Fahrzeugen und so weiter gekennzeichnet sind.
- **Modelltraining:** Nutzt TensorFlow Lite und die Model Maker API, um das Modell auf annotierten Daten zu trainieren und ein leichtgewichtiges Modell zu erzeugen, das für die Echtzeiterkennung auf dem Raspberry Pi optimiert ist.
- **Integration:** Das trainierte Modell wird in die Plattform integriert, sodass es in Echtzeit Objekte erkennen und darauf reagieren kann, wobei die MediaPipe-Bibliothek verwendet wird.

5. Szenarien

Die Funktionalität der Plattform wird durch drei experimentelle Szenarien getestet, die jeweils verschiedene Aspekte des Systems evaluieren sollen.

Szenario 1: Manuelle Steuerung mit Xbox-Controller

- **Ziel:** Testen der Grundfunktionalität des LiDAR-Systems durch manuelle Steuerung der Plattform über einen Xbox-Controller.

- **Prozess:** Der Xbox-Controller sendet Befehle an den Raspberry Pi, der dann die Bewegungen der Plattform steuert. Der LiDAR überwacht kontinuierlich die Umgebung auf Hindernisse, um Kollisionen zu vermeiden.

Szenario 2: Autonomes Fahren unter Verwendung von LiDAR und Spurenerkennung

- **Ziel:** Kombination von LiDAR-basierter Kollisionsvermeidung mit Spurenerkennung zur Ermöglichung autonomer Navigation.
- **Prozess:** Die Plattform nutzt den LiDAR zur Hinderniserkennung und die Kamera zur Aufrechterhaltung der Spurposition. Basierend auf dem Abstand zu erkannten Objekten und den Spurinformationen passt die Plattform ihre Geschwindigkeit und Richtung an, um eine sichere Navigation zu gewährleisten.

Szenario 3: Autonomes Fahren mit Objekterkennung

- **Ziel:** Erhöhen der Sicherheit und Entscheidungsfähigkeit der Plattform durch Integration der Objekterkennung.
- **Prozess:** Neben Spurenerkennung und Kollisionsvermeidung kann die Plattform spezifische Objekte identifizieren und entsprechend reagieren. Die Plattform stoppt beispielsweise an Stoppschildern und passt die Geschwindigkeit basierend auf Verkehrsschildern an.

6. Bewertung und Schlussfolgerung

Die Bewertung zeigt, dass die Plattform die Ziele der Kollisionsvermeidung, Spurenerkennung und Objekterkennung erfolgreich erfüllt. Die Integration von LiDAR- und Kameratechnologien führt zu einem System, das in der Lage ist, komplexe Umgebungen mit einem hohen Maß an Sicherheit und Zuverlässigkeit zu navigieren. Das Objekterkennungsmodell, das auf einem maßgeschneiderten Datensatz trainiert wurde, zeigt in Echtzeit eine gute Leistung und demonstriert das Potenzial für weitere Anwendungen in autonomen Fahrzeugen.

Schlussfolgerung:

Das Projekt demonstriert effektiv die Machbarkeit des Einsatzes von LiDAR- und kamerabasierten Systemen für autonomes Fahren. Zukünftige Arbeiten könnten sich darauf konzentrieren, die Fähigkeiten der Plattform zu erweitern, beispielsweise durch die Verbesserung der Genauigkeit der Objekterkennung, basierend auf gescannten LiDAR-Daten. Zudem könnte der Betrieb der Plattform in vielfältigeren Umgebungen ermöglicht werden. Auch die Hardware könnte verbessert werden, etwa durch die Integration eines Lenksystems, um die Balance der Plattform besser zu steuern.