SELF-CALIBRATING FUSION OF MULTI-SENSOR SYSTEM FOR AUTONOMOUS VEHICLE OBSTACLE DETECTION AND TRACKING

by

KAIQIAO TIAN

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN ELECTRICAL AND COMPUTER ENGINEERING

2023

Oakland University
Rochester, Michigan

Doctoral Advisory Committee:

Ka C. Cheok, Ph.D., Chair
Khalid Mirza, Ph.D.
Micho Tomislav Radovnikovich, Ph.D.
Wing-Yue Geoffrey Louie, Ph.D.
Aycil Cesmelioglu, Ph.D.

To my mother and father, to whom I owe everything

ACKNOWLEDGMENTS

With deep gratitude, I would like to express my heartfelt appreciation to my esteemed advisory committee for their invaluable guidance during my Ph.D. journey.

Dr. Ka C Cheok enriched my understanding of engineering fundamentals and Artificial Intelligence, equipping me with a wealth of knowledge. Dr. Micho Tomislav Radovnikovich introduced me to the world of ROS and algorithm development, opening doors to simulation environments and real-world applications. Dr. Khalid Mirza generously allowed me to assist in his prestigious Industrial Robotics Lab, where I gained practical experience with indoor navigation robots, manipulators, and sensors. Dr. Wing-Yue Geoffrey Louie illuminated the fascinating field of human-robot interactions, revealing its beauty and potential for the future. Dr. Aycil Cesmelioglu refreshed my mathematical skills and their application in engineering thinking. Thanks to Dr. Kazuyuki Kobayashi from Hosei University for their invaluable support with the IGVC self-driving vehicle implementation. I also extend my gratitude to Dr. Cai Changqing from CCIT University for their support in organizing the international Autonomous Vehicle workshop. I am forever grateful to each member of my advisory committee for their invaluable contributions to my academic and research endeavors.

Lastly, I am incredibly grateful to my parents for their unwavering support and encouragement throughout my overseas study. Their belief in me has been an invaluable source of motivation and has made this journey possible.

Kaiqiao Tian

ABSTRACT

SELF-CALIBRATING INFORMATION FUSION OF MULTI-SENSOR SYSTEM FOR AUTONOMOUS VEHICLE OBSTACLE DETECTION AND TRACKING

by

Kaiqiao Tian

Adviser: Ka C. Cheok, Ph.D.

Mobile robots have gained significant attention due to their ability to undertake complex tasks in various applications, ranging from autonomous vehicles to robotics and augmented reality. To achieve safe and efficient navigation, these robots rely on sensor data from RADAR, LiDAR, and Cameras to understand their surroundings. However, the integration of data from these sensors presents challenges, including data inconsistencies and sensor limitations. This thesis proposes a novel LiDAR and Camera sensor fusion algorithm that addresses these challenges, enabling more accurate and reliable perception for mobile robots and autonomous vehicles.

The proposed algorithm leverages the unique strengths of both LiDAR and Camera sensors to create a holistic representation of the environment. It adopts a multi-sensor data fusion (MSDF) approach, combining the complementary characteristics of LiDAR's precise 3D Point Cloud Data and the rich visual information provided by Cameras. The fusion process involves sensor data registration, calibration, and synchronization, ensuring accurate alignment and temporal coherence.

The algorithm introduces a robust data association technique that matches LiDAR points with visual features extracted from Camera images. By fusing these data, the algorithm enhances object detection and recognition capabilities, enabling the robot to perceive the environment with higher accuracy and efficiency. Additionally, the fusion technique compensates for sensor-specific limitations, such as LiDAR's susceptibility to adverse weather conditions and the Camera's vulnerability to lighting changes, resulting in a more reliable perception system.

The thesis contributes to advancing mobile robot perception by providing a comprehensive and practical LiDAR and Camera sensor fusion algorithm. This novel approach has significant implications for autonomous vehicles, robotics, and augmented reality applications, where accurate and reliable perception is vital for successful navigation and task execution. By addressing the limitations of individual sensors and offering a more unified and coherent perception system, the proposed algorithm paves the way for safer, more efficient, and intelligent mobile robot solutions in various real-world settings.

TABLE OF CONTENTS

TABLE OF CONTENTS—Continued

TABLE OF CONTENTS—Continued

TABLE OF CONTENTS—Continued

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

LiDAR	Light Detection and Ranging
RADAR	Radio Detection and Ranging
MSDF	Multi-Sensor Data Fusion
MTT	Multiple Target Tracking
AMRs	Autonomous Mobile Robots
SLAM	Simultaneous Localization and Mapping
IGVC	Intelligent Ground Vehicle Competition
PCD	Point Cloud Data
EOH	Electric Over Hydraulic
PID	Proportional–Integral–Derivative
CNNs	Convolutional neural networks
DBSCAN	Density-based spatial clustering of applications with noise
RANSAC	Random sample consensus
EKF	Extended Kalman Filter
UKF	Unscented Kalman Filter
PF	Particle Filter
SVD	Singular Value Decomposition
PAC	Principal component analysis

CHAPTER ONE

INTRODUCTION

1.1 Mobile Robot Technology

Mobile robots are intelligent machines controlled by software and equipped with sensors and other advanced technologies to perceive and navigate their surroundings. They are gaining popularity in both business and academic research areas due to their ability to perform tasks that are challenging, dangerous, or impossible for humans. With their advanced sensors, mobile robots can assist with a wide range of work processes, helping to increase efficiency and safety while reducing costs.

1.2 Autonomous Vehicles Technology

Autonomous Vehicles (AVs) have been in development for almost eighty years, with General Motors displaying a vision-based self-guided car and an automated highway system as early as 1939. The development of AV technology continued in 1958 with General Motors introducing coils detect sensors to control vehicle wheels and follow wires on the ground. In 1977, a Camera image processing system was developed by a Japanese scholar to control cars moving below 20 mph, while a German scholar developed the first object detection system using multiple cameras and micro-processing modules in 1987. In 2004, the US Department of Defense's research led to the start of the DARPA Challenges, where the Stanford University Stanley team won the first competition by completing 150 miles of desert roadway in autonomous mode driving within 7 hours. The second challenge in 2007 upgraded the course to a 60-mile-long urban environment, and four cars finished the route within the allotted six-hour time

limit. In 2015, Tesla Model S released the "Autopilot" package enabling hands-free control for highways and freeway driving, while the University of Michigan established the MCity program, a world-class test facility for autonomous vehicle technology where Ford became the first automaker to test autonomous vehicles.

Many self-driving companies are also embarking on their journey to explore autonomous vehicle technology. Argo AI, founded in 2016 by Bryan Salesky and Peter Rander, partnered with Ford and Volkswagen. Waymo, founded in 2009, partnered with Stellantis and Volvo. The Cruise, founded in 2013 by Kyle Vogt, Ian Rust, and Rita Ciaravino in California, and AutoX, a China-based self-driving company focused on robotaxi, founded in 2016, are also among the leading players in the field.

1.3 Advanced Sensors and Multiple Sensor Fusion System

The development of ranging sensors such as three-dimensional Light Detection and Ranging (LiDAR), Radio Detection and Ranging (RADAR), and Depth Cameras has revolutionized the way mobile robots and autonomous vehicles understand their surrounding environment.

LiDAR is a sensing technology that uses a laser to send out pulses of light that bounce off objects in the surrounding environment. By measuring the time, it takes for these pulses to bounce back to the sensor, a 3D map of the environment can be created. LiDAR is particularly useful in environments with low light or where color information is not necessary, such as in self-driving cars or drones. There are several manufacturers that produce LiDAR sensors, including Velodyne, Quanergy, Innoviz, and Luminar. These companies offer a range of LiDAR sensors with varying capabilities, from small, low-cost sensors for drones to high-resolution sensors for autonomous vehicles.

Additionally, LiDAR technology is continually improving, with new sensors being developed that offer longer range, higher resolution, and better accuracy.

RADAR is a sensing technology that uses radio waves to detect objects in the surrounding environment. A RADAR sensor emits radio waves, which then bounce off objects and are received back by the sensor. By analyzing the frequency shift of the reflected waves, the sensor can determine the distance, speed, and direction of moving objects. RADAR is particularly useful in environments with poor visibility, such as in fog or heavy rain. It is commonly used in aviation, shipping, and military applications, as well as in self-driving cars and drones. Major producers of RADAR sensors include Bosch, Continental AG, Hella KGaA Hueck & Co., and Valeo SA.

Depth Cameras, also known as depth sensors or 3D cameras, are sensors that measure the distance to a surface by emitting a light or sound signal and measuring the time it takes for the signal to bounce back to the sensor. They are commonly used in robotics and autonomous vehicles for perception tasks, such as obstacle detection, mapping, and localization. One type of depth camera is the Infrared (IR) based Kinect, which uses infrared light to measure the distance of objects in its field of view. By using a pattern of infrared dots and a camera to detect their distortion, the Kinect is able to create a 3D point cloud of the surrounding environment. This technology also allows the camera to see in the dark, making it useful in low-light conditions. Another type of depth camera is the vision-based camera, which uses two or more cameras to calculate the distance between objects based on their position in each camera's field of view. This method is also known as stereo vision, and it requires advanced algorithms to calculate the depth information accurately.

Multi-sensor data fusion (MSDF) is the process of integrating information from multiple sensors to create a more complete and accurate representation of the environment. This technique can improve the reliability and accuracy of the sensor data, which is crucial for autonomous vehicles and robots to make safe and effective decisions. In addition to MSDF, multiple target tracking (MTT) is an essential problem in autonomous vehicle technology. MTT involves tracking the movement and behavior of multiple objects in the environment, such as other vehicles, pedestrians, and cyclists. This requires sophisticated algorithms that can handle the high-dimensional data generated by multiple sensors and accurately identify and track each target over time.

LiDAR and RADAR sensor fusion is a popular approach for obstacle detection and object tracking. By combining the strengths of both sensors, such as the high accuracy and resolution of LiDAR and the ability of RADAR to detect objects through obstacles and in adverse weather conditions, the resulting data can provide a more comprehensive view of the environment. Similarly, LiDAR and camera sensor fusion and RADAR and camera sensor fusion can also be used to enhance the accuracy and reliability of the sensor data. Overall, the development of multi-sensor data fusion and multiple target tracking techniques is critical for advancing the capabilities of autonomous vehicles and robots and ensuring their safe and reliable operation in real-world environments.

1.4 Contributions

With the development of advanced sensors such as LiDAR, RADAR, and Depth Cameras, mobile robots and self-driving vehicles are able to perceive their environment faster and with greater accuracy. As a result, sensor fusion has become an active topic in

research and will continue to be for many years. The objective of this dissertation is to explore and develop new approaches to enhance the ability of mobile robots and AVs to understand their environment deeply. The work presented in this dissertation is intended to establish the foundation for lifelong academic research in this field. The dissertation makes several novel contributions, including:

1. Developing a Kinect Point Cloud data-based algorithm for locating potholes for ground mobile robots (Chapter 2.2.2).
2. Developing the distance clustering algorithm for 3D Point Cloud Data (Chapter 3.4).
3. Developing EKF, UKF, and PF Tracking algorithms for autonomous vehicle highway driving (Chapter 4.2).
4. Developing a LiDAR and Camera sensor fusion and self-calibrating algorithm (Chapter 5.4).

CHAPTER TWO

BACKGROUND – SENSING, CONTROL, AND APPLICATIONS

2.1 Literature

Robot and self-driving vehicle navigation and localization have become a significant research topic in recent years. The development of advanced ranging sensors, such as 3D LiDAR, RADAR, and Depth Cameras, has improved the understanding of the environment by autonomous systems. However, relying on a single sensor can provide limited information, making multi-sensor fusion an attractive solution for navigation and localization. By combining data from multiple sensors, sensor fusion technology can address two critical challenges in robot navigation. Autonomous mobile robots (AMRs) have revolutionized industrial automation by providing reliable and efficient material handling and transport. But the potential of mobile robots extends beyond manufacturing industries, with the development of new technologies and applications in fields such as entertainment, medicine, mining, education, and military. Their versatility and adaptability make them an increasingly attractive solution for various tasks, from delivering medication in hospitals to performing search and rescue operations in disaster-stricken areas [1].

Advanced sensors play a crucial role in improving the autonomous operation of robots by enabling them to understand their environment, navigate, localize, detect obstacles, and control motion [2-3]. Three-dimensional object detection has become the primary method for object-level perception for robots, replacing 2D detection [4]. Moreover, developers and researchers are increasingly using different types of sensors

and employing the sensor fusion method to enhance the autonomous capability of robots [5]. Sensor fusion using multiple sensors provides content-rich data, which can significantly increase the accuracy and robustness of mobile robots for various applications. For indoor navigation of autonomous robots, LiDAR and Camera fusion improve the position estimation accuracy [6]. Path planning, which requires determining the robot's current position and the position of a goal location within the same frame of reference or coordinates [7], effectively extends localization. Odometry and GPS fusion increase the robustness of outdoor robot navigation, especially when receiving a weak GPS signal. With self-contained odometry methods, an outdoor autonomous robot system can maintain good position estimation even when GPS is disabled [8]. Implementing good path-planning technology for mobile robots can save time and investment by reducing wear and tear [9]. Once a map is generated using Simultaneous Localization and Mapping (SLAM), the application of better path planning to avoid obstacles becomes an essential problem to be solved.

Recent technological advancements in computers, sensors, and AI have led to increased interest in unmanned mobile robots and self-driving vehicles [10-11]. The implementation and application of such systems requires the integration of numerous technological components and systems, which presents a challenge [14]. To facilitate hands-on learning, the IGVC offers challenges to university students worldwide, encouraging them to build their unmanned mobile robots and self-driving vehicles using the latest technologies and integration methods [15].

Self-driving vehicle technology has become a prominent research topic, and terms like "information fusion" and "multi-sensor data fusion" are widely used in various

applications that utilize data from multiple sources [16]. The architecture of a self-driving car typically comprises perception and decision-making systems [17]. Perception systems include computer vision techniques [18], 3-D point cloud data detection and classification [19], and localization [20]. Decision-making systems encompass obstacle detection [21] and path planning [22]. Cameras are cost-effective and small sensors that can provide appearance information about the environment [23-25]. RADAR sensors have been utilized in the automotive industry for a few decades and can detect obstacle speed in foggy, dusty, and snowy weather. Modern RADARs are cost-effective and can sense the environment in greater detail [26-28]. LiDAR is another popular and extensively used sensor in self-driving vehicle applications, owing to its reliable perception capability [29-33].

2.2 TurtleBot: Indoor Navigation Robot

TurtleBot, a ground robot educational platform, offers an affordable personal robot kit with open-source software. It was developed by Melonee Wise and Tully Foote in November 2010 at Willow Garage [34]. Despite being an entry-level robotics platform, TurtleBot has similar capabilities to larger platforms such as PR2 and Care-O-Bot. TurtleBot users can navigate their surroundings, create 3D maps, and develop customized actions. The second and third-generation models of TurtleBot are the most popular. TurtleBot 2 is a differential-drive robot that stands at 40 cm and can carry a personal laptop and sensors on its KOBUKI base. Meanwhile, TurtleBot 3 is a smaller navigation robot, standing at only 20 cm, equipped with a Raspberry Pi, an Arduino control board, and a 360-degree LIDAR sensor.

2.2.1 Differential Drive Robot Modeling

Robot The TurtleBot utilizes a drive mechanism called the differential drive, as depicted in Figure 2.1. This system is comprised of two drive wheels mounted on a shared axis, and each can be individually controlled to move forward or backward. When the two wheels are turned in opposite directions at the same speed, the robot will rotate about the axis's central point. On the other hand, if both wheels are turned in the same direction at the same speed, the TurtleBot will move in a straight line. This design allows the robot to navigate through different environments with agility and flexibility.

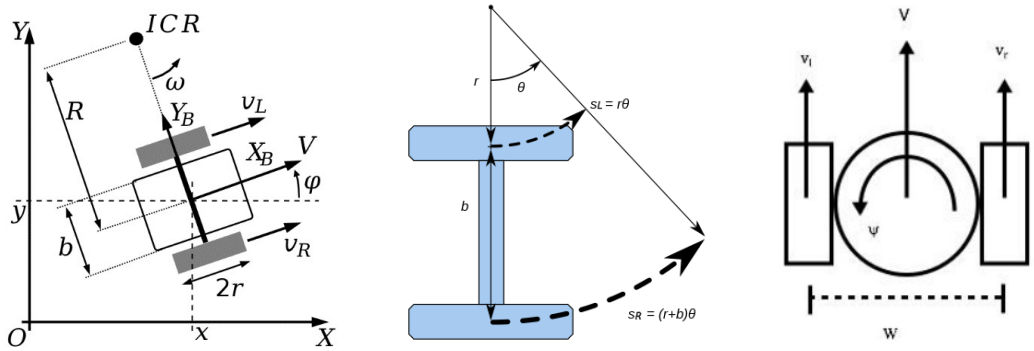

Figure 2.1 Kinematics of a Differential-Drive robot

Robot individual wheel speed v_R, v_L can be calculated from wheel distance b, vehicle angular speed ω, and rotation radius R.

$$\omega\left(R+\frac{b}{2}\right)=v_R \tag{2.1}$$

$$\omega\left(R-\frac{b}{2}\right)=v_L \tag{2.2}$$

The robot kinematics in local body coordinates can be written as

$$one\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\varphi}\end{bmatrix}=\begin{bmatrix}cos\varphi & 0\\ sin\varphi & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}V\\ \omega\end{bmatrix} \tag{2.3}$$

From TurtleBot control respective, we always give robot velocity V and the angular velocity ω as input. Then, we need the formula to calculate robot two-wheel rotation.

$$\omega_R = \frac{V + \frac{\omega b}{2}}{r} \tag{2.4}$$

$$\omega_L = \frac{V - \frac{\omega b}{2}}{r} \tag{2.5}$$

2.2.2 TurtleBot 2: 2D LiDAR and Stereo Camera Kinect Sensor Fusion

The TurtleBot utilizes a drive mechanism called the differential drive. ROS RViz is a powerful tool for robot simulation and visualization. It allows users to create and interact with 3D models of robots and their environments. With RViz, users can simulate robot motion, sensor data, and even test their algorithms before deploying them on actual robots. The Hokuyo 2D LiDAR is a popular sensor for robot navigation and mapping, while the Kinect camera is often used for human-robot interaction and object recognition tasks. By combining these sensors, TurtleBot 2 can accurately perceive its environment and navigate through it autonomously, as shown in Figure 2.2.

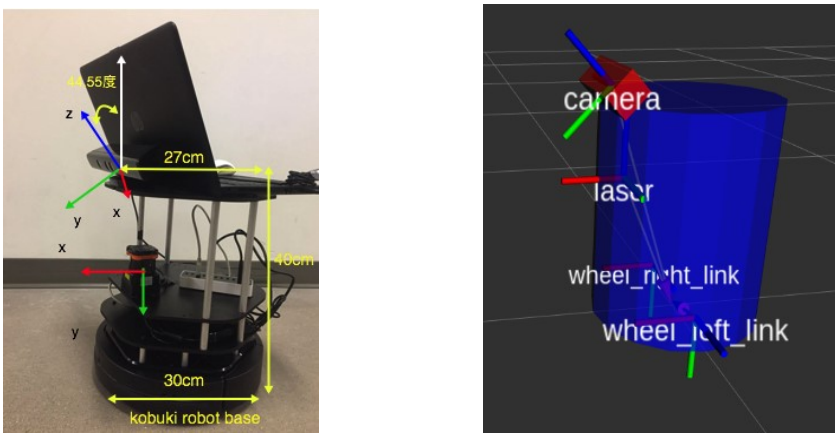

Figure 2.2 TurtleBot 2 and ROS simulation model

A stereo or depth Camera can provide a 3D structure and get the object's distance information. Microsoft Kinect sensor measures distance using the standard time-of-flight (ToF) technique [35]. The sensor has one Camera that provides an RGB image and an IR projector and receiver for depth measurements. Figure 2.3 shows the Kinect and Kinect model for the IR sensor that measures the distance of a projected point in space to the Camera. Kinect's IR Projector projects a known pattern of dots which is compared with the image seen by the IR Receiver to calculate the distance of each point from the sensor. This point cloud data now has (x, y, z) value of each point to locate it in the space.

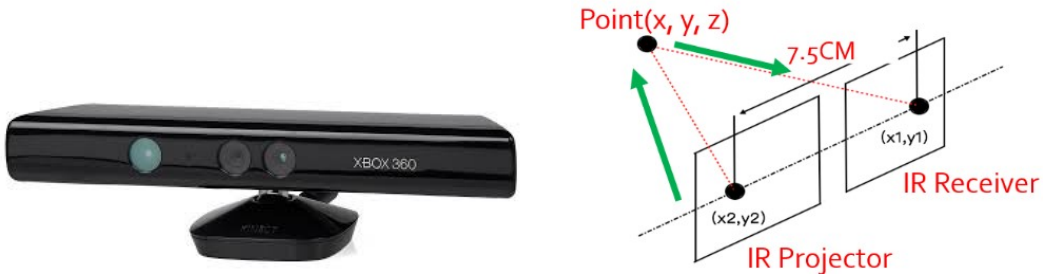

Figure 2.3 Kinect depth Camera IR structure

Point Cloud Data (PCD) [36] from Kinect sensor needs a processing step before it can be used for object detection. The first step for PCD processing is transforming the sensor frame into the robot frame by applying a rotation matrix. The first picture in Figure 2.4 shows the raw point cloud data in the Kinect sensor frame. A pass-through filter is then used to cut unnecessary point cloud data, followed by a down-sample algorithm to decrease point cloud size and reduce processing time. The second figure in Figure 2.4 shows the raw PCD data generated from Kinect, the point clouds are not in correct rotation and come with lots of noisy data.

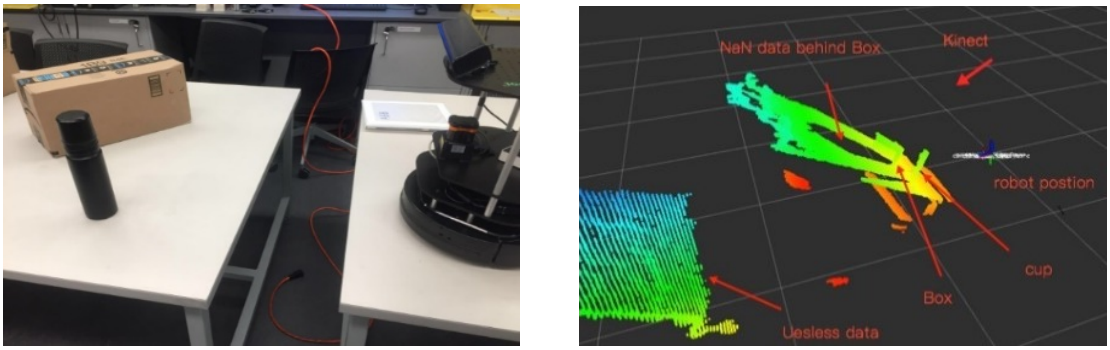

Figure 2.4 TurtleBot 2 and Kinect raw PCD

Figure 2.5 shows the justified PCD processing result and frame transfer structure. All PCD data are in the Camera Frame, a rotation matrix R can transfer all data to the World Frame. a is the Kinect angle, z is the height of the Camera.

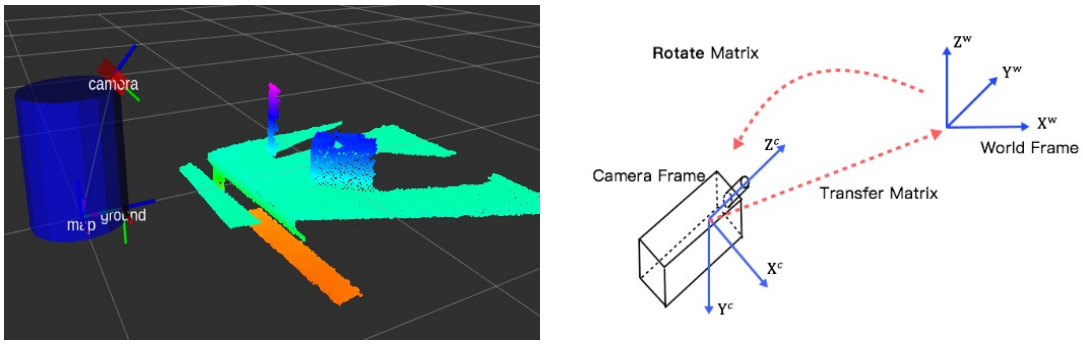

Figure 2.5 Justified PCD processing result and frame transfer structure

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\ a & -\sin a & z(-0.418) \\ 0 & sin\ a & \cos a & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.6}$$

Point cloud data helps the robot understand its surrounding objects. The above-ground obstacles can be detected using the point cloud data generated by the Kinect sensor. Figure 2.6 shows color-coded point cloud data representing the distances from ground information in RVIZ. Point cloud data on the ground is shown in one color. When the robot faces the wall, the distance from the bottom to points on the wall will change

and is indicated by a change in colors. Based on those values, the robot can understand where the obstacle is on the ground.

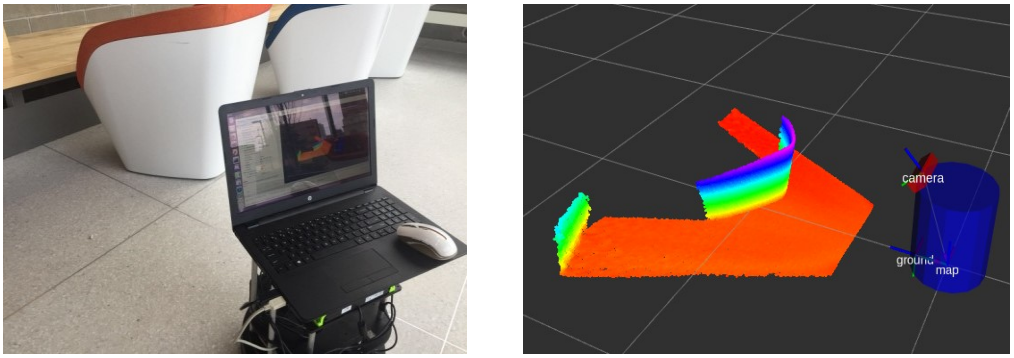

Figure 2.6 TurtleBot navigation and PCD visualization

Mobile robots only detect above-ground objects for obstacle avoidance. But in the real world, a pothole or gap along the robot's path can cause damage to the robot and needs to be avoided.

All autonomous mobile robots use various methods to detect above-ground obstacles for safe navigation through their environment. However, not all obstacles a mobile robot may encounter are above the ground. Below-ground obstacles like holes or gaps present a hazard for mobile robots and are constantly challenging for autonomous navigation. TurtleBot uses the Kinect depth sensor to detect below-ground obstacles and above-ground ones. Based on distance information from the Kinect sensor, an algorithm is designed to detect and locate potentially hazardous holes in the ground. The algorithm first generates a 2D image with distance information, as shown in Figure 2.7.

The (x, y) location for each point on the contour of the hole is subsequently calculated. The last step uses a rotation matrix to transform the contour position from the sensor to the robot base frame. A depth map generated by up-sampling the Kinect point cloud data shows the average distance for the plane.

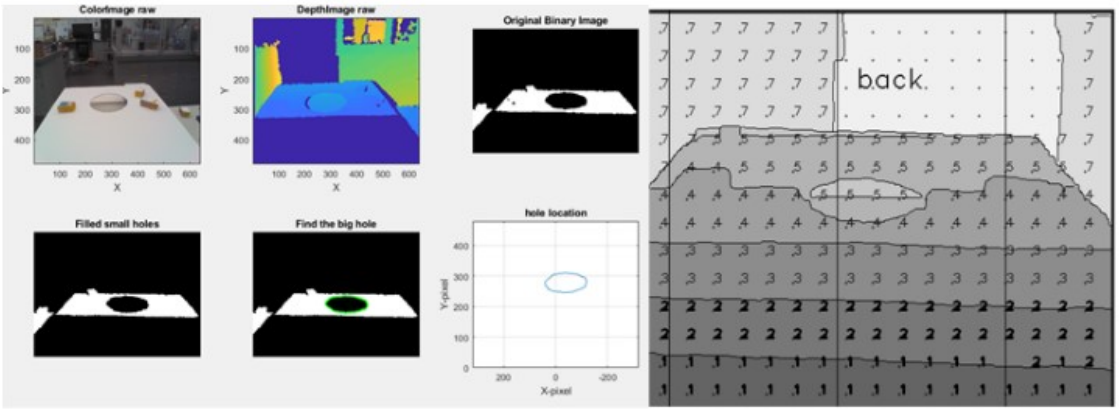

Figure 2.7 Hole-finder algorithm result

Figure 2.8 shows the ROS RViz visualization, colorful Point Cloud data from Kinect, and white Point Cloud data from the LiDAR.

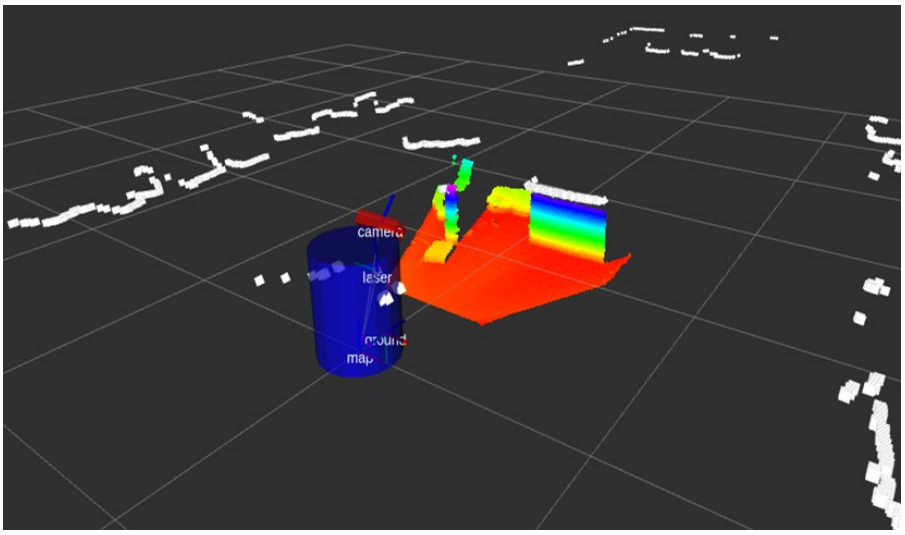

Figure 2.8 Robot data visualization in ROS RViz

2.2.3 TurtleBot 3: SLAM and Path Planning

TurtleBot 3, standing at just 20 cm tall, is equipped with a 360-degree LiDAR for SLAM and Navigation. Unlike other robots, TurtleBot 3 can seamlessly communicate with a laptop on the same network, allowing for easy remote control from a remote

station. The Raspberry Pi 3 onboard computer is a powerful yet compact device that can handle many of the computational tasks required for autonomous navigation. The OpenCL board, based on the Arduino platform, controls the two DYNAMIXEL wheels and allows for precise motor control. The Wi-Fi module allows for wireless communication with a remote station, making it easy to control and monitor the robot from a distance.

We won the second prize in the 2019 China National Senior Helper Indoor Navigation Robot Competition using TurtleBot 3. As shown in Figure 2.9, the competition required the robot to navigate to each room and communicate with various sensors installed in each of the three rooms, including GAS detection sensors, temperature sensors, and humidity sensors. The robot was programmed to move to the next room only if all sensor readings were within regular limits. In the event of any readings exceeding safety thresholds, the robot would immediately alert the emergency contact if not reset by anyone at home.

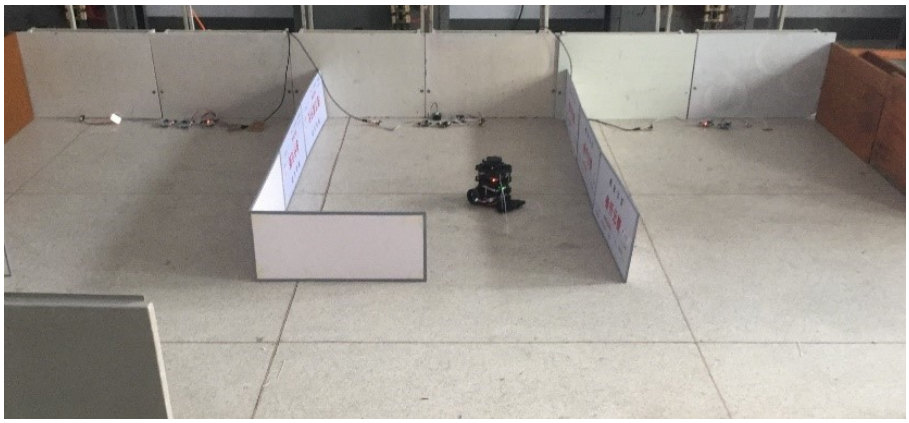

Figure 2.9 TurtleBot 2 indoor navigating

Using the LiDAR and Odom data, we were able to generate a 2D map of the environment using the G-mapping SLAM algorithm. Figure 2.10 illustrates the robot's global and local path-planning strategies during indoor navigation.

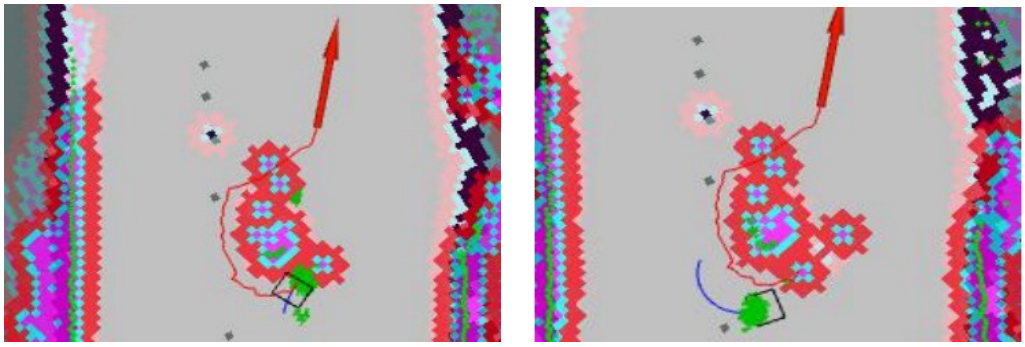

Figure 2.10 TurtleBot 2 indoor navigating data visualization

Simultaneous Localization and Mapping (SLAM) is a technique commonly used in autonomous vehicles that enables simultaneous mapping and localization of the vehicle in an unknown environment. By using SLAM algorithms, the vehicle can construct a map of its surroundings and determine its location within that map in real time. This map information is then utilized by engineers to perform tasks such as path planning and obstacle avoidance. Figure 2.11 depicts a map generated by TurtleBot 3 through the use of the SLAM algorithm.

Figure 2.11 Oakland University 3rd floor 2D map

2.3 Octagon: Outdoor Navigation Robot

Typically, robot navigation relies on a single approach for solving the localization problem, such as using an indoor map or GPS. However, Octagon is an autonomous mobile robot designed for industry-level applications that utilizes sensor fusion to achieve indoor and outdoor navigation tasks. For safe navigation, a below-ground obstacle detection algorithm based on point cloud data from the Kinect depth camera is used in conjunction with above-ground obstacle data provided by the 2D LiDAR SLAM algorithm. Additionally, a navigation source switching algorithm is implemented to switch between indoor maps and outdoor GPS navigation systems. Octagon incorporates various sensor technologies to create a versatile robot platform capable of autonomously navigating and transporting payloads in various environments. The fusion of the 2D LiDAR and Kinect sensors in the SLAM algorithm enables obstacle avoidance and the detection and location of below-ground obstacles. Path planning and position estimation techniques are also employed to switch between navigation data sources as needed.

2.3.1 Octagon: Outdoor Navigation Robot

Octagon is an autonomous mobile robot that has been designed to operate in both indoor and outdoor environments. The robot's platform has been equipped with all the necessary components and sensors to achieve this objective. The robot's base has been built to be solid and stable, enabling it to carry heavy loads, while its differential drive allows for zero-point turns, facilitating quick reactions to obstacles. To enable the robot to perform heavy-duty tasks, it has been equipped with high-torque industrial gearboxes manufactured by Nabtesco, featuring a 103:1 gear ratio. Figure 2.12 depicts the Octagon

robot's base, differential wheel, and modular motor-gear design, showcasing the robot's mechanical design.

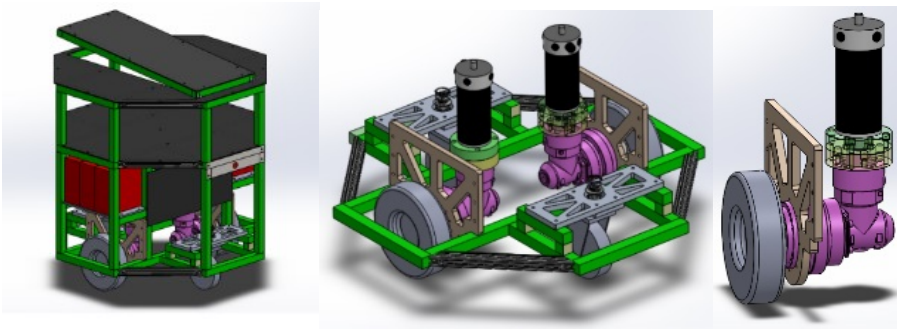

Figure 2.12 Electrical and mechanical design of Octagon (by the team)

The octagon's body is constructed using 80/20 T-slot extruded aluminum structural framing, which is a lightweight yet robust material. This material is readily available and makes it easy to modify or rebuild the robot in the future. To increase the robot's stability, the electrical equipment and batteries are placed low to the ground in the center of the robot. The brushless DC motors are mounted securely to Nabtesco gearboxes using custom-made adaptor discs. The gearbox is then mounted to a plate, making the entire unit modular and easily bolted to the robot frame. This design ensures that the Octagon remains balanced and stable even while driving on uneven surfaces. Table 2.1 provides details on some of the basic mechanical parameters of the robot.

Table 2.1 Octagon robot parameter

Item	**Description**
Dimensions	37 x 30 x 66 inches
Payload	300 lbs.
Runtime	3 hours
Weight	200 lbs.
Top Speed	5 mph

The power system of the Octagon robot operates on 24 V DC, which is provided by six 12-volt batteries. The power system module then converts this to lower voltages and provides 5V, 12V, and 24V for various electrical components, sensors, and motors. An overview of the electrical system is shown in Figure 2.13. The robot's two Arduino microcontrollers control each motor separately, which helps to reduce the processing time for the PID loop. Another Arduino microcontroller is used to receive information from both sides of the robot and to coordinate control between the radio controller and the NUC, which is a powerful minicomputer used for camera image processing on the robot. Additionally, the robot has a safety system that includes two emergency stops on the platform and an additional remote stop that can disable the main power to stop the robot immediately in an emergency.

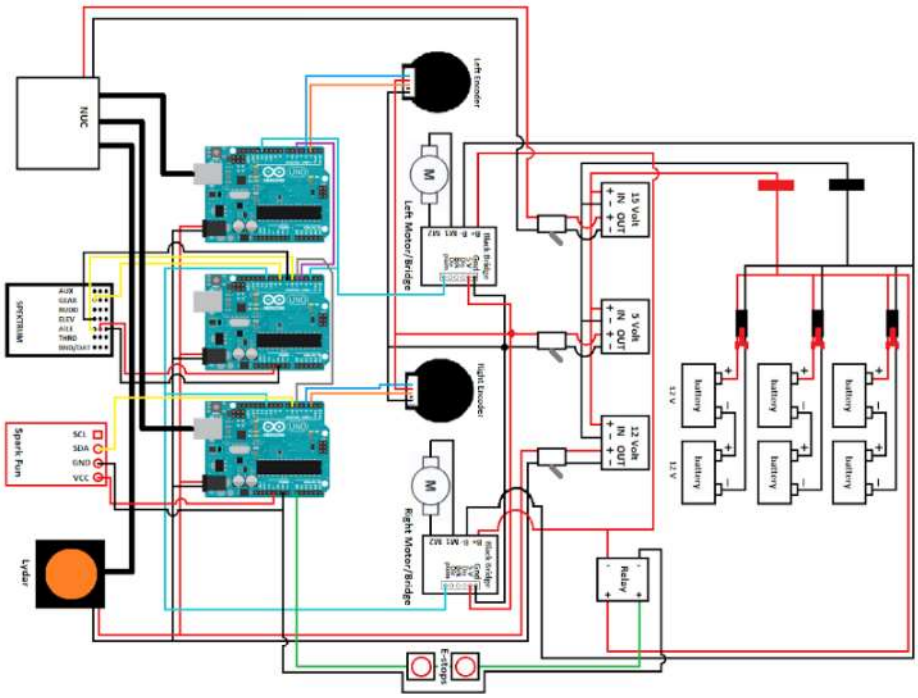

Figure 2.13 Controller drive design of Octagon (by the team)

2.3.2 Octagon Navigation and Sensor Fusion

The transform nodes in ROS handle the coordinate frames for the various sensors and the robot itself. The path planning and obstacle avoidance are done using the global and local planner nodes. The local planner node generates a safe and feasible path for the robot to follow, while the global planner node computes a long-term path using the provided map information. The Octagon robot can switch between indoor and outdoor navigation modes seamlessly, thanks to the navigation source switching algorithm. The ROS architecture allows for easy integration of additional sensors or algorithms in the future to enhance the robot's capabilities. Figure 2.14 shows the ROS node relationship in rqt_graph. The move_base node takes care of the robot motor and movement.

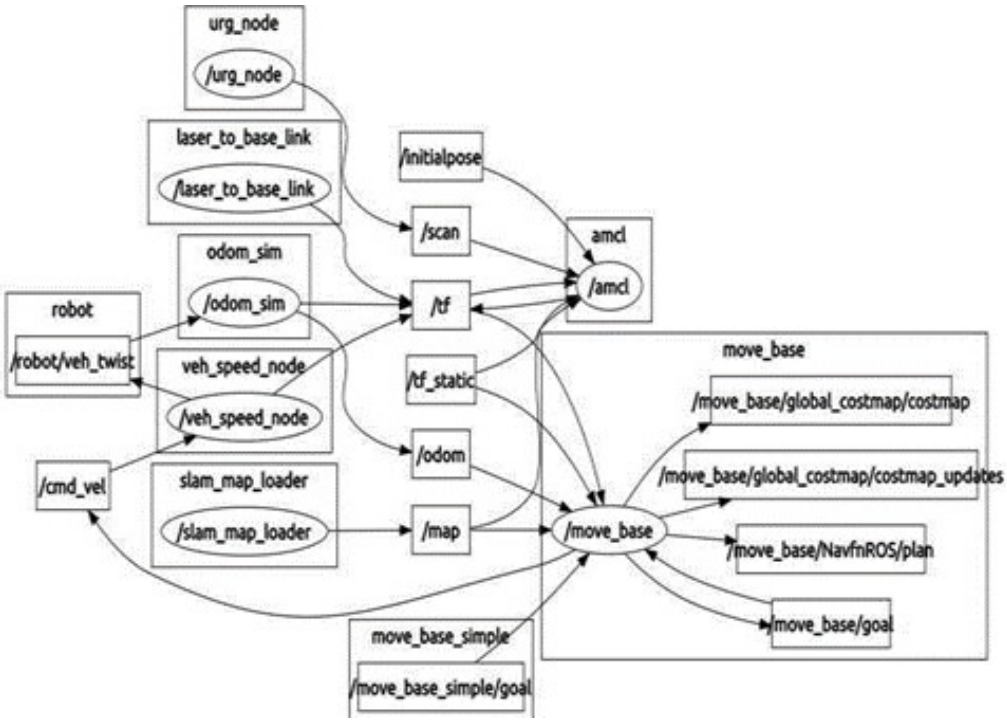

Figure 2.14 Octagon control system ROS node in rqt_graph (contribution)

Figure 2.15 shows the real robot and its ROS simulation model, and the simulation model has the same size and same sensor TF tree as the real robot.

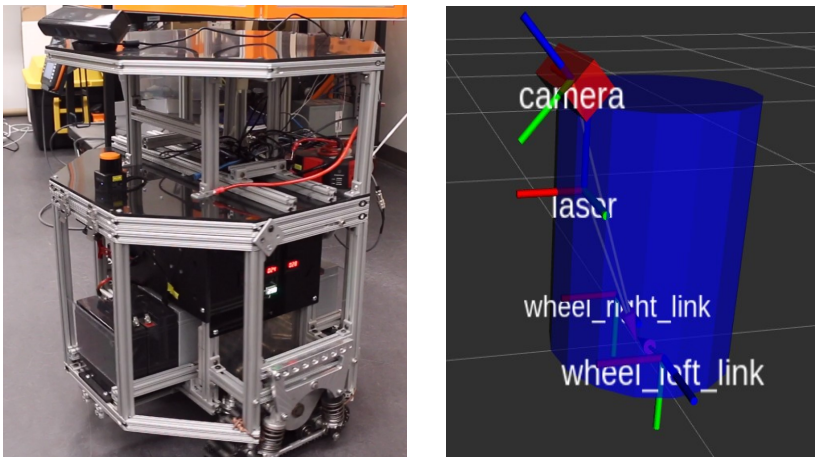

Figure 2.15 Octagon and ROS simulation

2.4 IGVC Autonomous Vehicle

In the Intelligent Ground Vehicle Competition (IGVC) Self-Drive Challenges, we converted a Polaris GEM2 electric vehicle into a basic self-driving by-wire electric vehicle. Using ROS for the IGVC self-driving challenge enabled the team to create a flexible and modular system that allowed for quick integration of hardware components and software programs. The ROS environment allowed for the creation of multiple microcontroller-based sensor and actuator modules, which could be easily networked together using ROS serial protocol. The use of ROS also allowed for frequent reassessments and debugging of the system, which was critical for ensuring that the self-driving vehicle could perform the required tasks. With the help of ROS, the team was able to rapidly prototype and debug their system, enabling them to achieve their goal of converting the Polaris GEM2 electric vehicle into a basic self-driving by-wire electric vehicle.

2.4.1 Polaris Gem 2: Autonomous Vehicle

Ackermann's steering geometry is named after Rudolph Ackermann, who patented it in 1818. It is a type of steering system that uses a specific arrangement of linkages to ensure that all four wheels of a vehicle follow circular paths with the same center point during turns. This reduces tire wear and provides more precise handling. The geometry of the steering linkage determines the degree to which the wheels can turn and how much they must turn in response to the steering wheel's input.

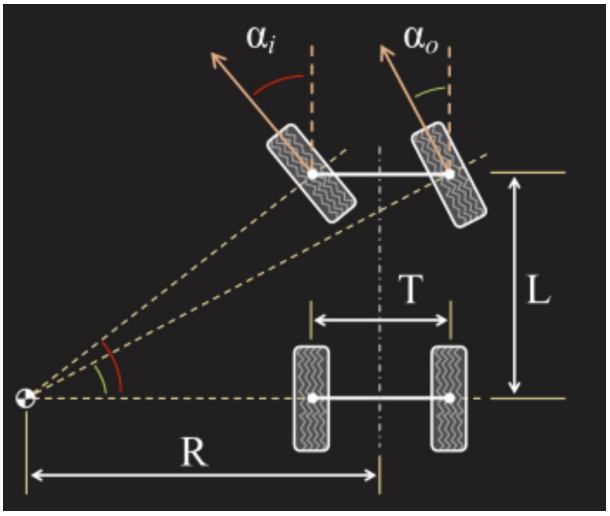

Figure 2.16 Ackermann steering Structure

When a vehicle follows the Ackermann steering geometry while turning, the inside wheel rotates at a greater angle than the outside wheel. Assuming constant speed and no external forces, body roll, or suspension effects and considering only the front wheels' steering, we can determine the optimal angles for the wheels using basic trigonometry. In the diagram, the wheelbase (distance between the two axles) is denoted by L, the track (distance between the center lines of the two wheels) by T, and the radius of the turn as experienced by the centerline of the vehicle by R. The angle of the inside wheel from

straight ahead is represented by α_i, while the angle of the outside wheel from straight forward is denoted by α_j.

$$\alpha_i = tan^{-1}\left(\frac{L}{R - \frac{T}{2}}\right) \tag{2.7}$$

$$\alpha_j = tan^{-1}\left(\frac{L}{R + \frac{T}{2}}\right) \tag{2.8}$$

2.4.2 A Drive-by-wire System

Figure 2.17 depicts the by-wire vehicle system developed on a Gem2 electric vehicle. The system features a 2D LiDAR (UTM-30LX) installed in the front for obstacle detection, while a Logitech C920HD Pro camera is mounted on the right side of the roof to facilitate lane detection for the lane-following function in self-driving tasks. The by-wire control box, which houses all the subsystem components, is installed in the vehicle's trunk.

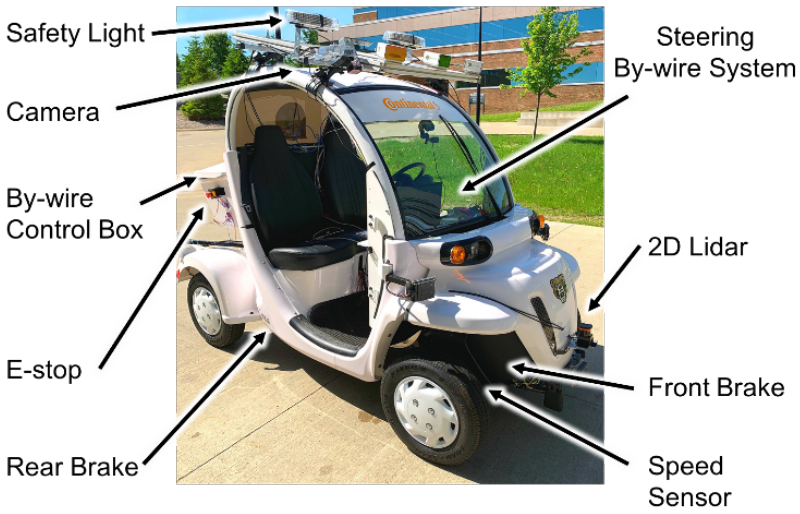

Figure 2.17 The drive-by-wire vehicle sensor system demonstration

Figure 2.18 shows an overview of the integrated by-wire system, where we employed the Robot Operating System (ROS) to interact with five subsystem modules for rapid prototyping, debugging, and testing. For hardware and software design, speculation and collaboration were carried out simultaneously to save the release development circle. Detailed functions of five subsystems are described in the following subsections.

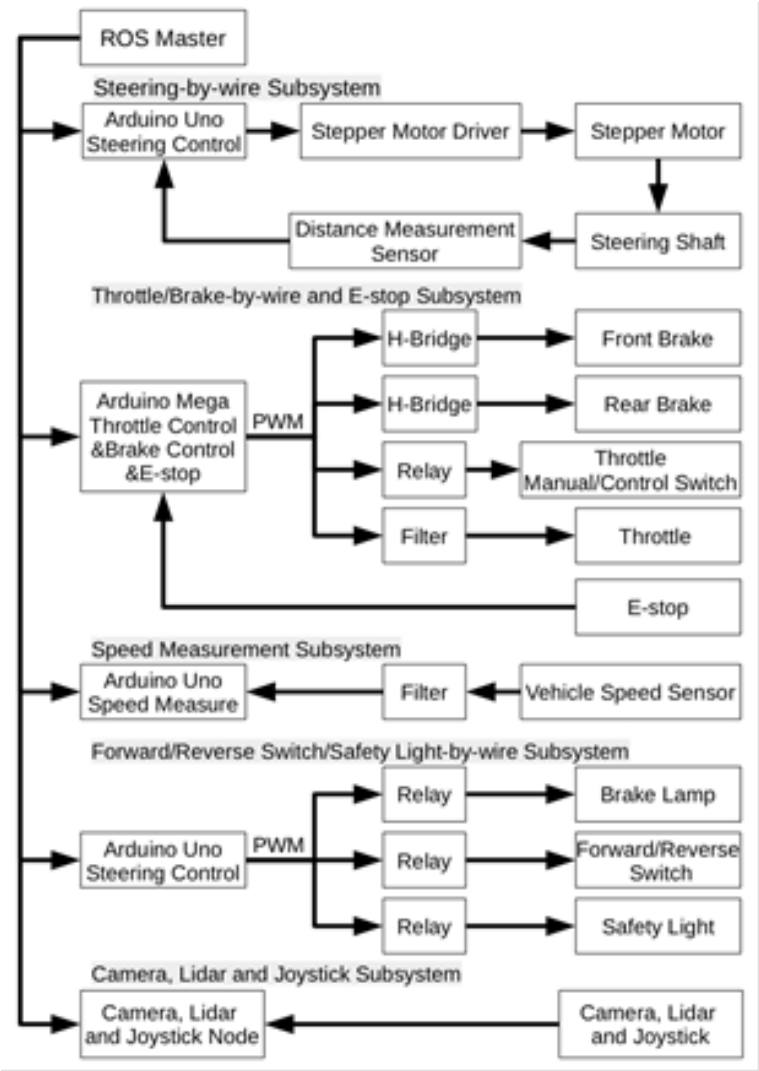

Figure 2.18 Vehicle ROS node structure demonstration

A. Steering-by-wire subsystem

The original electric vehicle had no power steering subsystem. Therefore, an additional by-wire steering control subsystem was developed and implemented. Figure 2.19 shows the components and signal flow for the developed steer-by-wire subsystem. A distance/displacement sensor over the shaft pulley measured the steering angle rotation. A proportional and integral feedback control was used to actuate the stepper motor for steering angle.

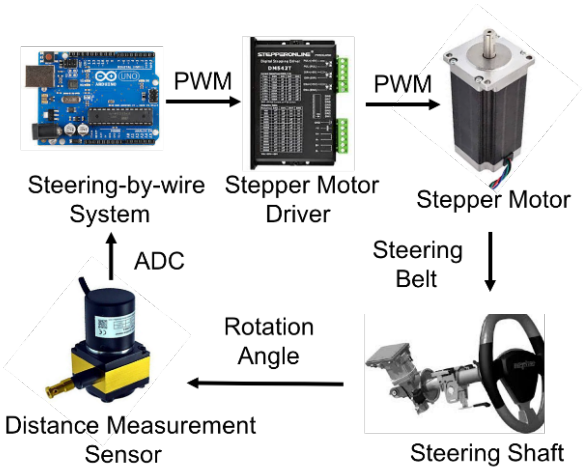

Figure 2.19 Steering-by-wire subsystem demonstration

Figure 2.20 shows the physical implementation of the steer-by-wire subsystem. Due to the limitation of the space around the steering shaft, the stepper motor drives the steering shaft by timing pulleys and timing belts. The pulley belts were fixed to the stepper motor shaft, and the other pulley was appointed to the steering shaft.

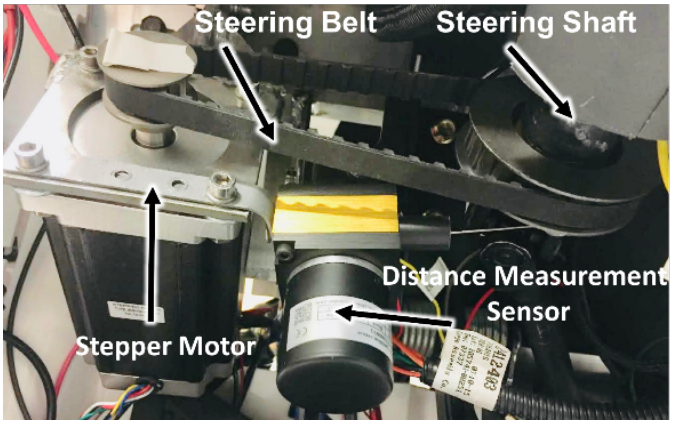

Figure 2.20 Steering-by-wire subsystem physical display

B. Throttle/brake-by-wire and E-stop subsystem

Throttle and brake control are essential functions for by-wire vehicles. As shown in Figure 2.21, two braking systems were implemented for the brake-by-wire subsystems. The front brake is actuated by an Electric Over Hydraulic (EOH) actuator (DX1600 Dexter). A linear hydraulic actuator actuates the rear brake. Both the hydraulic actuators are activated through Arduino PWM and H-bridge power controllers. An electrical relay was used to switch the throttle control mode into manual and automated mode. The relay was controlled using an Arduino digital signal. The rules require a hardware E-stop and a wireless E-stop for emergency purposes. We installed two manual E-stop buttons: one in the driver and passenger cabin and the other at the rear exterior of the vehicle; and a wireless remote-control E-stop. A hardware E-Stop function disconnects power from the vehicle motor and applies to the front brake.

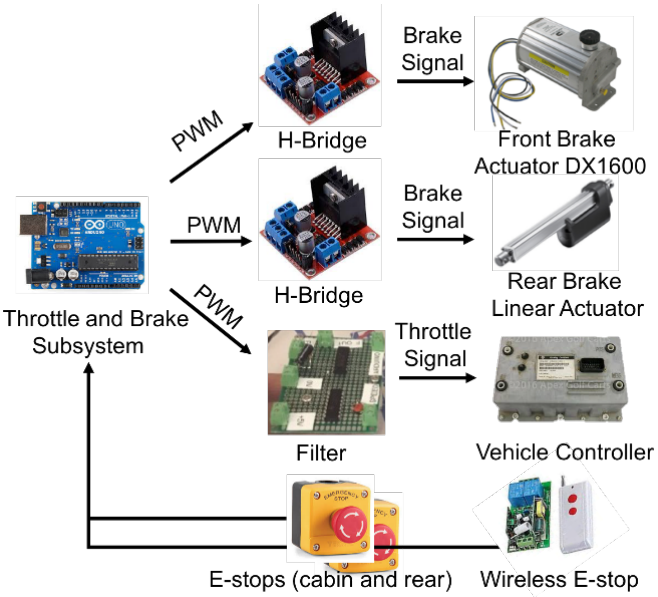

Figure 2.21 Throttle/brake-by-wire subsystem demonstration

C. Forward/reverse switch, safety light-by-wire subsystem

Three modes are available in the GEM2 electric vehicle: The reverse mode, low-speed forward mode, and high-speed forward mode. According to challenging rules, we only implement low-speed forward and reverse modes using an Arduino relay switch shown in Figure 2.22.

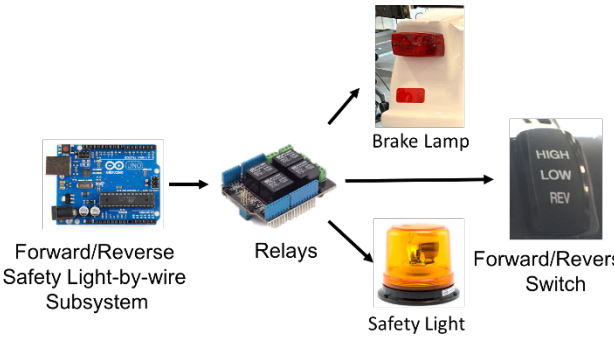

Figure 2.22 Safety light-by-wire subsystem demonstration

Also installed with the safety light for indicating the mode of self-driving. A flashing red light indicates that the vehicle is operating in self-driving mode, and a solid yellow light indicates standby mode.

D. Speed measurement subsystem

Figure 2.23 shows the interface for measuring the vehicle's speed using the existing hall effect sensor with the Gem2. The estimated filtered speed was fed to a proportional and derivative controller which actuated the throttle to regulate the vehicle speed.

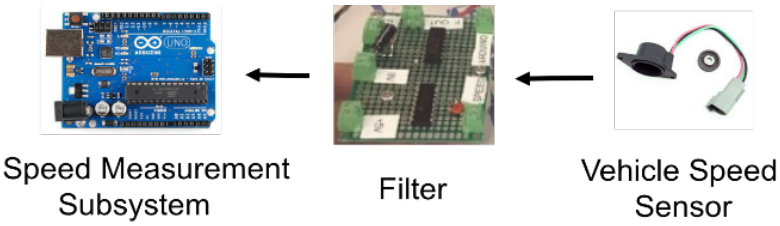

Figure 2.23 Speed measurement subsystem demonstration

E. Camera, LiDAR, and Joystick subsystem

The Camera and LiDAR are the two primary sensors needed to overcome the basic autonomous sub-functions of the IGVC self-drive challenge: namely, lane-following, collision avoidance, left and right turns, road sign detection, stop lane detection, right angle, and parallel parking. The LiDAR provides a planar range map with a point cloud showing where objects are located. It was used to prevent frontal collisions as well as to veer around obstacles in the vehicle path.

Processed images from the Camera show the locations of lanes, road edges, road signs, and objects of interest. We used line image processing techniques to detect lanes, borders, and stop lines. Also, we trained a convolution neural network (YOLO) using the

deep learning method to recognize road signs (stop, left and right turn arrows, etc.) and objects appearing along the road.

The fusion of this processed information leads to a robust and reliable inference of the cues from the roads. We are still investigating further fusion techniques for the project and will report the findings and results in a future paper. Aside from having stable, correct sensing inference, agile testing with a short turn-around time is another key for reducing the troubleshooting and debugging time of the path planning navigation and guidance scheme. For this purpose, we implemented the open-source ROS nodes for Joystick, Camera, and LiDAR, shown in Figure 2.24. The Joystick was essential for manual operation.

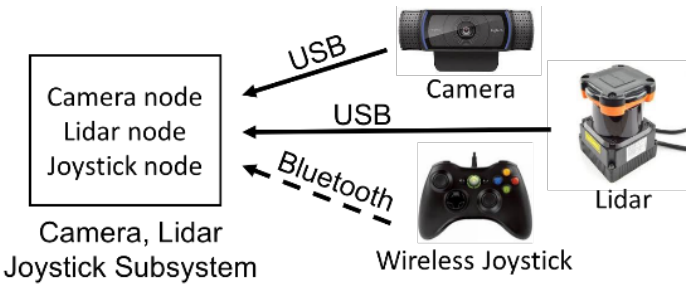

Figure 2.24 Camera, LiDAR, and Joystick subsystem demonstration

2.4.3 PID Vehicle Steering and Speed Controller

In order to achieve smooth and accurate control of a vehicle's steering and speed, we designed and implemented a proportional–integral–derivative (PID) controller. The PID controller is a popular control loop mechanism that uses feedback to adjust the system's output based on the difference between the desired setpoint and the measured output. By adjusting the gains of the proportional, integral, and derivative terms, we can

fine-tune the controller's behavior to match the specific requirements of our application, allowing us to achieve precise and stable control of the vehicle's steering and speed.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.7}$$

Where K_p, K_p, and K_p all non-negative, denote the coefficients for the proportional, integral, and derivative terms, respectively.

2.4.4 PID Vehicle Steering and Speed Controller

Due to the potential version conflicts with the standard ROS navigation and localization stacks, we opted to develop a custom navigation stack from scratch using the ROS serial bridge to facilitate inter-topic communication. This approach is ideal for targeted tasks and enables rapid adjustments in response to challenges. It is important to verify the behavior of each individual module by testing and debugging subsystems. By modifying the ROS launch file, we can manipulate the modules and observe the performance of the by-wire electric vehicle. To illustrate our ROS implementations, we present two representative examples below.

Figure 2.20 showcases the integration of the joystick with the by-wire actuation subsystems diagram. The shaded oval boxes denote the developed hardware and software subsystems, acting as input and output nodes. The non-shaded oval boxes represent software nodes where data is stored. The rectangular boxes represent topics that consist of programs subscribing to, reading from, publishing to, and manipulating data, as well as executing algorithms.

The joystick plays a crucial role in providing manual remote control to the by-wire vehicle. To access commands from the joystick, the /joy topic subscribes to the

Joystick node and publishes to the Joystick topic converter node. Subsequently, other topics, such as /steering, /throttle, etc., that subscribe to this converter node can access the joystick commands and act accordingly.

By applying joystick commands and analyzing the behavior, each subsystem can be debugged effectively. Figure 2.25 demonstrates the versatility of the joystick in performing experiments and troubleshooting.

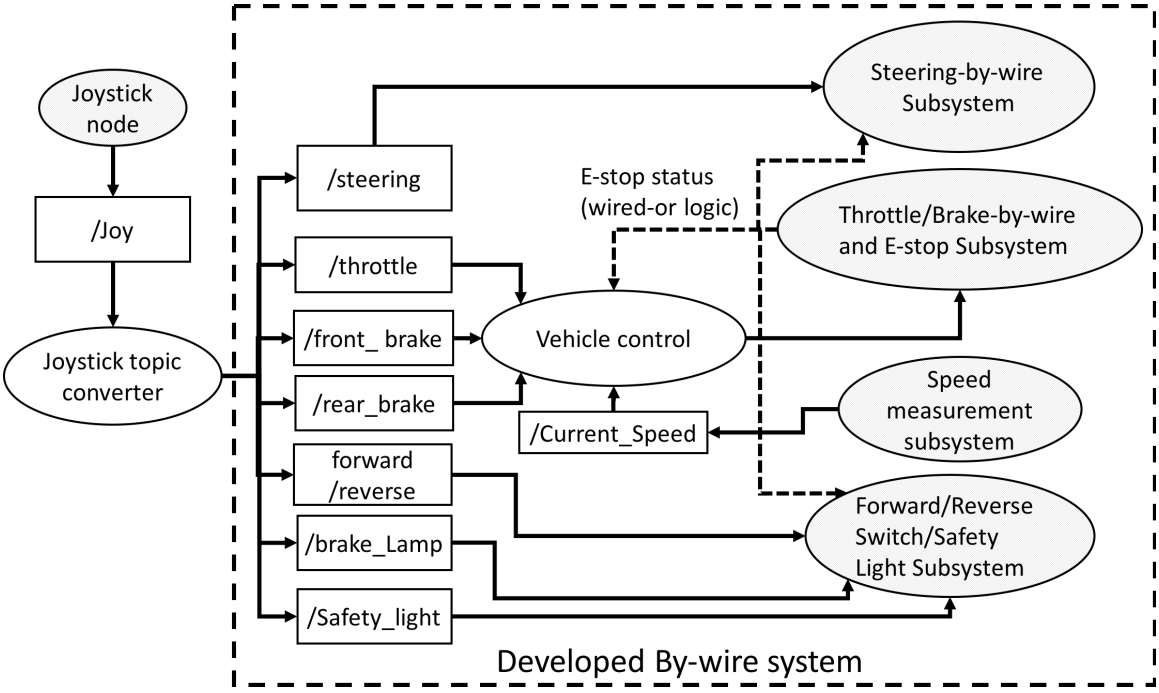

Figure 2.25 The developed by-wire system demonstration

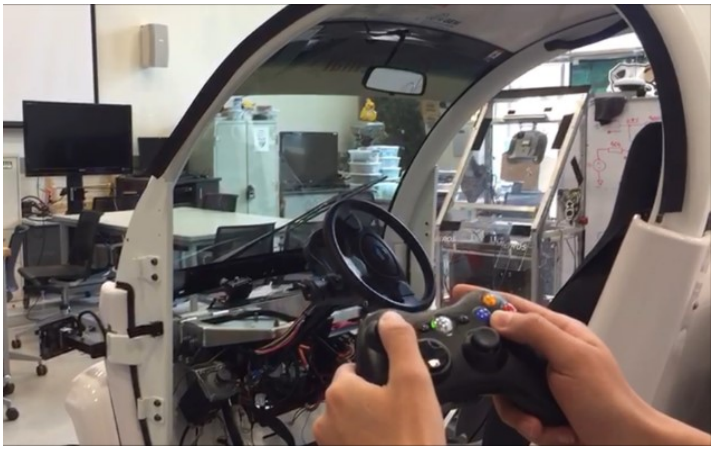

Figure 2.26 Self-driving vehicle system demonstration

The implementation of lane-following differs from the by-wire joystick control in that it utilizes the LiDAR and Camera nodes instead of the joystick node, as depicted in Figure 2.27. The /scan and /image_raw topics are subscribed to in order to read data from the sensors and feed it to the obstacle detection and lane detection nodes. The collected information is processed and integrated to control the acceleration and steering of the vehicle. Obstacle detection is performed using the LiDAR, while lane and stop line detection are performed using the Camera. Once again, the modular nature of the topics enables the algorithm to be quickly altered, allowing for the testing of various functions such as lane following and collision prevention. This agile development process greatly reduced the testing and evaluation time.

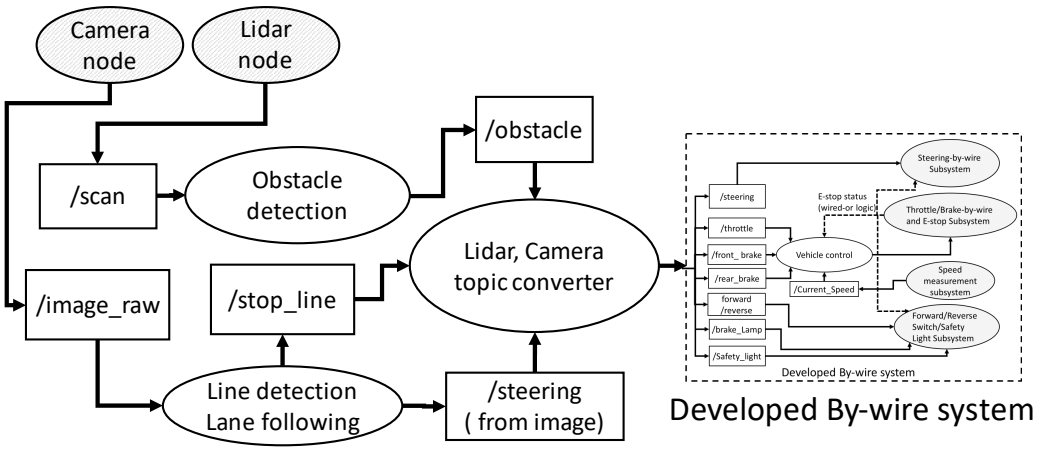

Figure 2.27 Vehicle sensor system demonstration

Figure 2.28 depicts the by-wire vehicle running in auto-driving mode during the competition, where it is required to move straight between the two-lane lines and come to a stop before reaching the obstacle. To ensure that the vehicle stays on the correct path, cameras track the lanes while LiDAR detects obstacles in front of the vehicle.

Figure 2.28 Vehicle running in self-driving mode (IGVC)

2.5 Convolutional Neural Network 2D Image Classification

Convolutional neural networks (CNNs) are a specific type of feed-forward neural network that have artificial neurons capable of responding to a specific area of the surrounding cells within a defined range. They are highly effective in large-scale image-processing tasks. For autonomous vehicles, 2D image detection and classification using cameras is critical. By detecting and classifying road lane lines, traffic signs, traffic lights, pedestrians, and other objects, the vehicle can make informed decisions about how to navigate its environment. In our self-driving vehicle, we utilize a single camera to track road lane lines and ensure the vehicle stays on course. We have also implemented a traffic sign classifier based on the LeNet Model to detect stop signs. Additionally, we use YOLO neural networks for object classification to further improve the vehicle's ability to identify and respond to its surroundings.

2.5.1 LeNet Traffic Sign Classifier

LeNet is a convolutional neural network architecture introduced by LeCun et al. in 1998 [37]. LeNet-5 is the most commonly referred version of LeNet, and it is a relatively straightforward convolutional neural network that consists of two convolutional layers followed by two fully connected layers, as depicted in Figure 2.29.

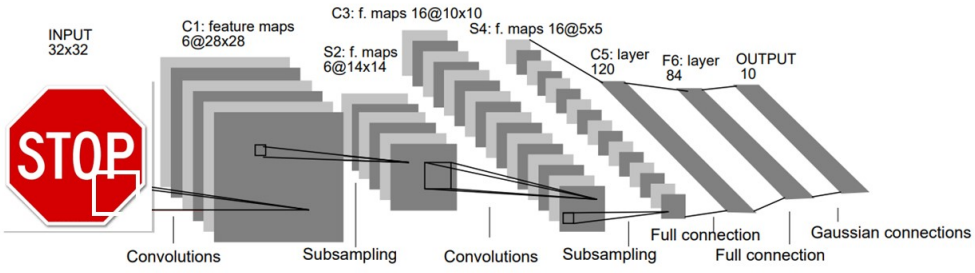

Figure 2.29 The LeNet traffic sign classifier demonstration

We trained LeNet using the German Traffic Sign Dataset (GTSRB), which contains 50,000 training samples across 43 classes of traffic signs. Figure 2.30 depicts the 43 different training classes utilized in the training process.

Figure 2.30 GTSRB traffic classes

The high accuracy of 99.5% obtained by LeNet for traffic sign detection, as depicted in Figure 2.31, is a promising achievement for the autonomous vehicle's ability to recognize and respond to traffic signs.

Figure 2.31 Traffic sign classification result

2.5.2 Convolution Neural Network: YOLO

The YOLO algorithm divides an image into a grid and predicts the class probabilities and bounding boxes for each grid cell. The algorithm then selects the most probable bounding boxes that have an intersection over union (IoU) greater than a certain threshold. This approach allows the YOLO algorithm to achieve high accuracy and real-time object detection.

In our self-driving vehicle, we use the YOLOv3 model to detect and classify objects in real time. The model is trained on the COCO dataset, which contains 80 object classes. The YOLOv3 model is capable of detecting multiple objects in an image and providing accurate classification information, which is crucial for autonomous driving.

2.6 Unsupervised Machine Learning Clustering Algorithms

Clustering methods are an essential tool in unsupervised machine learning, as they classify data points into specific groups based on their similar properties or features, providing valuable insights from the dataset. The most commonly used clustering algorithms include hierarchical, k-means, and density-based methods, each with its own strengths and limitations. In this section, we compare these three popular clustering techniques and propose a 3-dimensional dynamic clustering algorithm that can be used for real-time object clustering and tracking during vehicle highway driving.

2.6.1 Hierarchical Clustering Algorithms

Hierarchical clustering is an agglomerative method that follows a bottom-up approach. In this algorithm, each data point is treated as an individual cluster at the

beginning. Then, as the algorithm proceeds, similar clusters are merged together until only one or K clusters remain, as shown in Table 2.2.

Table 2.2 Hierarchical clustering algorithm steps

Steps	Algorithm
1	Compute the proximity matrix
2	Set each data point to a unique cluster
3	Repeat: Merge the two closest clusters and update the proximity matrix
4	Until only a single cluster remains or meets the clustering number K

As depicted in Figure 2.32, the hierarchical clustering algorithm operates by merging similar clusters at each iteration. While this approach is intuitive and offers flexibility in adjusting the number of groups through distance merge limits, it can be computationally expensive for medium-sized datasets.

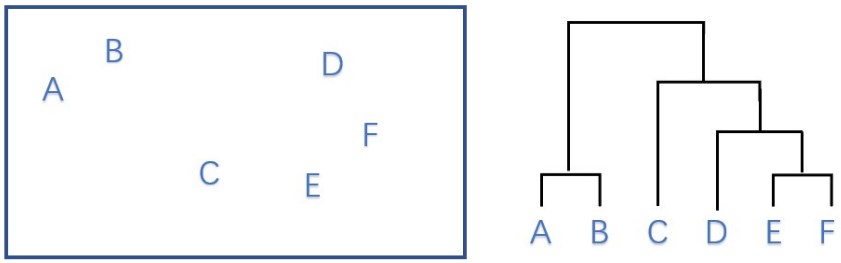

Figure 2.32 Hierarchical clustering algorithm

2.6.2 K-means Clustering Algorithms

The k-means clustering algorithm is one of the simplest unsupervised learning algorithms, and it belongs to the heuristic strategy. The main idea of K-means clustering is seeking the centroids of each clustering k and assigning different points $\Phi(x_i)$ in the dataset, the steps of K-means clustering as shown in Table 2.3:

Table 2.3 K-mean clustering algorithm steps

Steps	Algorithm
1	Randomly select cluster centroid μ_k
2	Calculate the distance $d_i(1,2..k)$ between $\Phi(x_i)$ and μ_k
3	Select the minimum distance d_i for $\Phi(x_i)$ and assign it to related μ_k
4	Calculate the new centered μ_k
5	Keep looping until no change to the centroids μ_k

Dataset $\Phi(x_i)$ points in the dataset, each cluster $k = 1,2,..K$ is represented by a centroid $\mu_k \in R$. The objective function is known as the squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\left| \left| x_i - v_j \right| \right| \right)^2 \quad (2.1)$$

Where, $x_i - v_j$ is the Euclidean distance between x_i and v_j, c_i is the number of data points in its cluster c is the number of cluster centers.

K-means Clustering Algorithm is fast, robust, and has a good result when data is well separated and has no overlap. However, we should know the exact number of clusters in a non-linear dataset.

2.6.3 Density Clustering Algorithms

Density-based spatial clustering of applications with noise (DBSCAN) is one of the most popular algorithms that can cluster non-linear data and arbitrary shapes data. The dataset presented as $x = \{x_1, x_2, x_3 \ldots, x_n\}$ DBSCAN requires two parameters: ε (eps) and the minimum number of points needed to form a cluster (minimal points), step as shown in Table 2.4：

Table 2.4 Density clustering algorithm steps

Steps	Algorithm
1	Select a random point that has not been visited.
2	Extract the neighborhood points that are within ε distance from the select point.
3	Clustering processes begin if points are more significant than minimal points; mark this point as visited.
4	If a point belongs to part of the cluster, its ε neighborhood is also part of the cluster, and repeat the above procedure from step 2 for all ε neighborhood points. Until all points in the cluster are determined.
5	A new unvisited point is retrieved and processed, leading to further cluster or noise discovery.
6	This process continues until all points are marked as visited.

Density-based clustering can handle non-linear data with noise and find arbitrarily sized and shaped clusters. But DBSCAN will fail in the varying and high dimensional density clusters situation and neck type of dataset.

2.6.4 K-Dimensional Tree Algorithms

The K-dimensional tree algorithm is a binary search tree that is commonly used for efficient nearest neighbor searches and range searches in a K-dimensional space. The basic idea behind the KD tree algorithm is to partition the data recursively points into a hierarchical structure of nested rectangles, dividing the data points into two halves at each level based on their position along one of the k dimensions.

As shown in Figure 2.33, to perform a nearest neighbor search using a KD tree, we start at the root node of the tree and recursively traverse the tree, visiting only those nodes that could potentially contain a closer point than the best one found so far. At each node, we check whether the current node is closer than the best point found so far, and if so, update the best point.

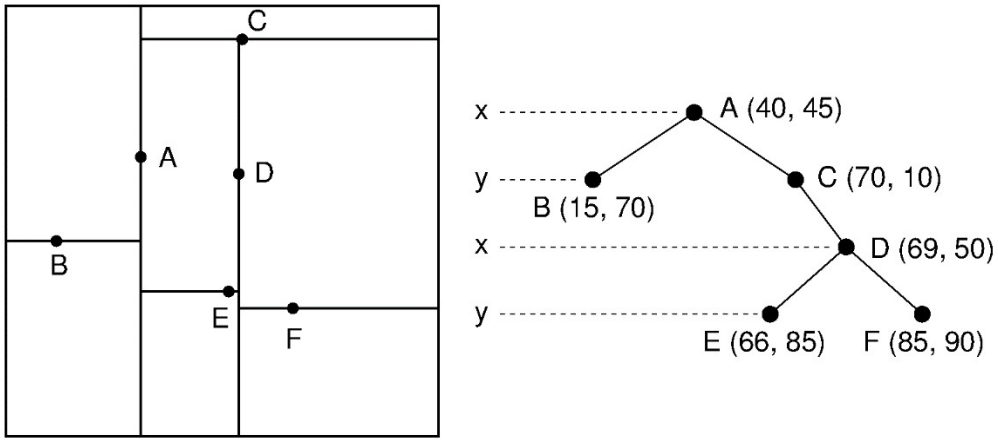

Figure 2.33 Example of how KD Trees split space.

CHAPTER THREE

MULTIPLE SENSORS POINT CLOUD CLUSTERING ALGORITHM

3.1 Introduction

LiDAR, RADAR, and Cameras provide various data detected by observation and have pros and cons. The core of multi-sensor fusion is how to fuse all information and generate a reliable and stable result. Different types of sensors are applied to self-driving vehicles to mitigate the weaknesses of individual sensors and increase safety levels. This chapter explains the designed clustering algorithm for point cloud data generated from an autonomous vehicle with 3 LiDAR and 1 RADAR.

3.2 Autonomous Vehicle Platform and Sensors Setup

Equipped with four distance sensors, the vehicle platform features a Ford Fusion mount with a middle-range Cepton LiDAR and a long-range RADAR mounted on the front bumper to detect obstacles ahead. Additionally, two Ouster LiDARs are mounted on the vehicle's front and rear roof to capture a 360-degree view of the ego vehicle's environment. For more information on the sensors, refer to Table 3.1.

Table 3.1 Autonomous vehicle platform sensors summary

Type	Model	Position	Detect Angle	Detect Range
LiDAR	Cepton	Front bumper	60°	200 m
LiDAR	Ouster OS1	Front Roof	360°	120 m
LiDAR	Ouster OS1	Rear Roof	360°	120 m
RADAR	Continental	Front bumper	60°	300 m

To implement and test the tracking algorithms, we recorded precise sensor data while driving the vehicle on the highway and played it back in ROS RViz simulation. The vehicle platform, designed specifically for testing purposes, is shown in Figure 3.1. Additionally, Figure 3.2 provides a closer look at the Cepton LiDAR, Ouster LiDAR, and Continental RADAR installed on the vehicle.

Figure 3.1 Autonomous vehicle platform and ROS simulation (by Dataspeed and Professor Micho)

Figure 3.2 Cepton LiDAR, Ouster LiDAR, Continental RADAR

In Figure 3.3, a snapshot of a vehicle driving on the highway is shown, along with the corresponding vehicle model and raw point cloud data (PCD). The PCD is color-coded based on the LiDAR that generated it, with three distinct colors representing the different LiDAR sensors used. Two driving vehicles are detected in the scene, one in front of the ego vehicle and one behind it. The front Ouster LiDAR and Cepton LiDAR

detect the vehicle ahead and represent it as yellow and orange PCD, while the rear vehicle is captured by the two Ouster LiDAR and is shown as yellow and blue PCD.

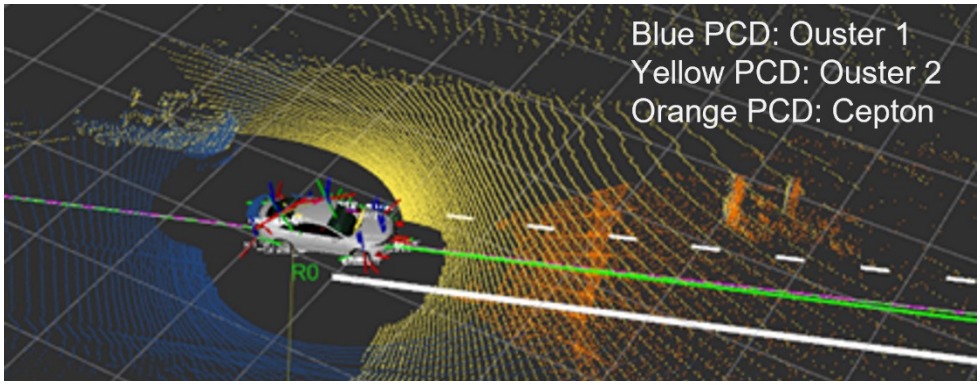

Figure 3.3 Vehicle highway driving in ROS Rviz

Each LiDAR operates in its own frame of reference and can generate millions of spatial points per second. The subsequent step involves processing these vast amounts of point cloud data to determine the location and speed of the detected objects.

3.3 Single-Sensor PCD Segmentation and Normalization

LiDAR and RADAR provide Point Cloud Data (PCD) that describes detected objects and environments using plenty of space points. The first step of multi-sensor fusion and tracking is single-sensor data segmentation [38]. In the PCD segmentation process, we use a random sample consensus (RANSAC) algorithm [39] to distinguish PCD between obstacle and ground. RANSAC uses a voting scheme to estimate and find the optimal fitting data for inliers and outliers. First, randomly select a sample subset containing minimal data to determine a model from the input dataset. The remaining data

is considered as an inlier if it fits the model within some error threshold that defines the maximum deviation attributable to the effect of noise. Second, the algorithm iteratively repeats the first step until the obtained consensus set in a particular iteration has enough inliers. In our case, randomly select three points to determine a plane in each iteration and calculate the average distance between the remaining points to this plane. Figure 3.4 shows the LiDAR data divided into two parts: the green ground and the red obstacles above the floor.

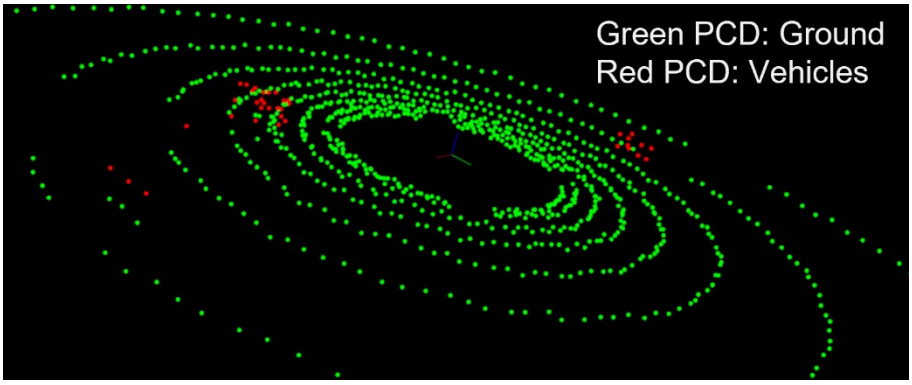

Figure 3.4 Single LiDAR PCD segmentation result

The next step is to group the above-ground red spatial points into two clusters, as they belong to two different vehicles. To achieve this, we employ the Euclidean Clustering method using the K-Dimensional (K-D) tree algorithm [40-41], which is a space partitioning data structure that is well-suited for clustering points in a k-Dimensional space. The K-D tree algorithm uses the Nearest Neighbor (NN) search algorithm to find the nearest point in the tree. By implementing the K-D tree algorithm on the PCD, we obtain two groups of spatial points that correspond to the two vehicles.

Representing an object using numerous space points can be inefficient and may make tracking more difficult. To address this issue, we have designed a bounding box

data type that facilitates object normalization. This data type consists of object label (x), position (x, y, z), scale (x, y, z), and speed (V_x, V_y, V_z), which allows us to represent an object in a more concise and manageable manner.

In Figure 3.5, a single bounding box is used to represent a highway vehicle. The label number displayed on the bounding box indicates the number of objects detected at that moment. The scale of the bounding box covers all the point cloud data, and the speed of the vehicle is calculated based on the position change and frame update time.

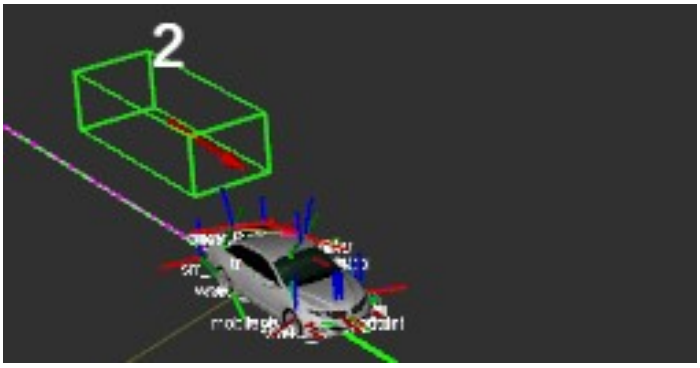

Figure 3.5 A bounding box presents the object vehicle.

To group the bounding boxes representing the same vehicle, a multi-sensor data fusion technique is applied using the Kalman Filter algorithm. The Kalman Filter algorithm estimates the state of the object, including position, velocity, and acceleration, based on noise measurements from multiple sensors. By fusing the data from multiple sensors, the Kalman Filter algorithm provides more accurate and reliable estimates of the object state than using a single sensor alone. The output of the Kalman Filter algorithm is a set of fused tracks that represent the objects' trajectories. These tracks are represented by a series of bounding boxes that correspond to the object's estimated position and size at each time step, as shown in Figure 3.7.

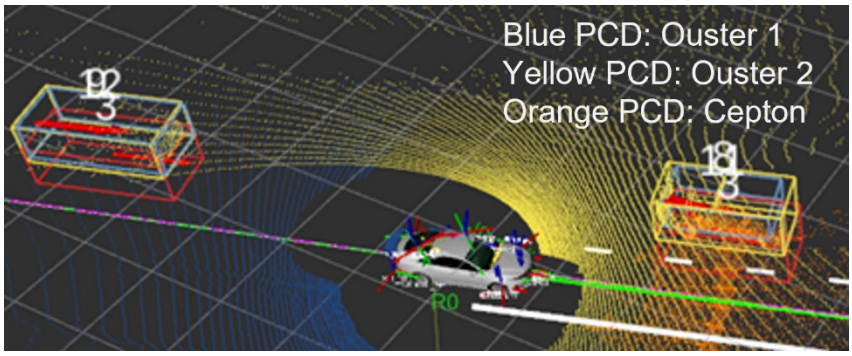

Figure 3.6 Multiple sensors result.

3.4 Multiple Sensors PCD Clustering Algorithm

Although traditional clustering methods such as hierarchical clustering [42], k-means clustering [43], and density-based clustering [44] have their advantages and can be applied to many applications, they have limitations when it comes to the multiple sensor fusion scenarios of self-driving vehicles. One of the main challenges is that it is difficult to determine the number of objects or cluster centers in highway multi-target tracking. Moreover, in real-time situations, each sensor updates at a different frequency, which can make it challenging to synchronize data. Finally, multi-target tracking is a non-linear 3-dimensional clustering problem. To address these challenges, we designed a real-time 3-dimensional clustering algorithm for multiple sensor applications, which is explained in Table 3.2:

Table 3.2 3-Dimensional clustering algorithm steps

Steps	Algorithm
1	Reset all received bounding box labels as 0, as shown in Figure 3.7
2	When the first bounding box is received, create an empty map with label 1 and push the received data to the vector. Map can show as: $[1, (b_1, b_2, b_3, \dots b_i)]$, where b_n is the associated bounding box.
3	Calculate the average center point $p(x, y, z)$ for all bounding box in each map vector. $p_x = \frac{1}{n}(x_{b1} + x_{b2} + \cdots + x_{bn})$, $p_y = \frac{1}{n}(y_{b1} + y_{b2} + \cdots + y_{bn})$, $p_z = \frac{1}{n}(z_{b1} + z_{b2} + \cdots + z_{bn})$
4	To determine whether the newer bounding box data is associated with an existing map, The newer data center point $h(x, y, z)$ must calculate space distance d with all current map average center points. $d = \sqrt{(h_x - p_x)^2 + (h_y - p_y)^2 + (h_z - p_z)^2}$
5	If newer data distance is smaller than a threshold, $d < D_t$, we consider this as associate data and push it to that vector. If not, create a new map and store this data.
6	If a map didn't associate with new data for a specific time, delete this map.
7	Delete the old data in the dynamic vector map elements by existing time.
8	Publish the average position for each map and assign a unique label. Then we can visualize in ROS RViz, as shown in Figure 3.8

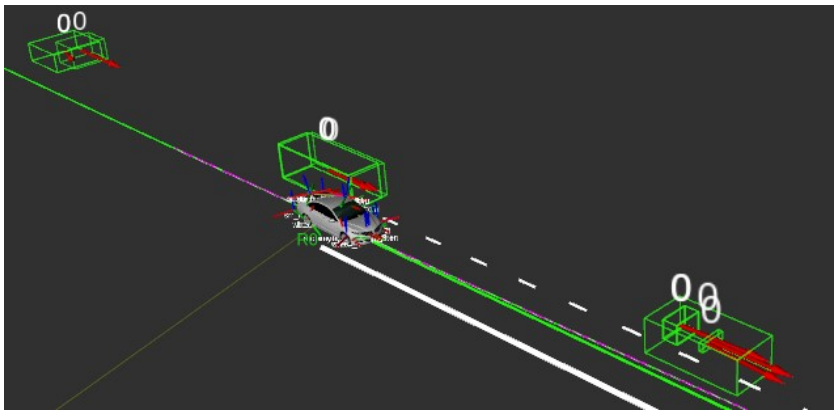

Figure 3.7 Bounding box initial label result

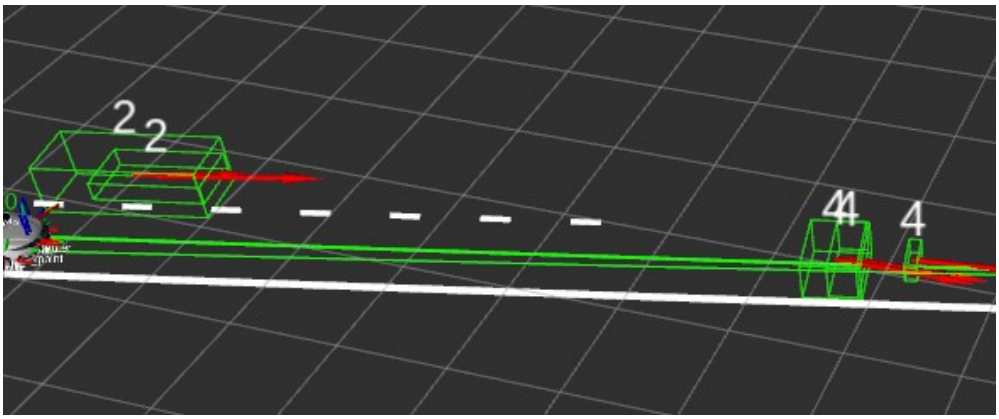

Figure 3.8 Bounding box regroup and relabel result.

3.5 Results

Figure 3.8 displays the outcome of the clustering algorithm for bounding box regrouping and relabeling. At this moment, two object vehicles were detected: one close to the ego vehicle and the other in front of it. The number of bounding boxes indicates the number of sensors that captured them, and the label indicates that they represent the same object. However, the bounding boxes are not identical in size or exact location, as each sensor has its own perspective, highlighting the beauty of multiple-sensor fusion. By combining the different sensor data and verifying each other, the accuracy of the results is increased. Nevertheless, this results only groups and relabels bounding boxes at time t. In the next chapter, we will explain how to track object vehicles using these bounding boxes.

CHAPTER FOUR

AUTONOMOUS VEHICLE OBJECT TRACKING ALGORITHM

4.1 Introduction

Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), and Particle Filter (PF) are three popular algorithms to address obstacle position estimate and tracking problems [45-55]. However, as technology develops, autonomous vehicles pursue a better understanding of the environment and higher safety driving. Different modern sensors are mounted on the car, such as three-dimensional Light Detection and Ranging (LiDAR) and Radio Detection and Ranging (RADAR). Sensor fusion from various data types can improve the position estimate accuracy and challenge the traditional tracking algorithm. In order to explore which tracking algorithm has better performance in multi-sensor data fusion (MSDF) and multi-target tracking (MTT) problems, we implement and analyze EKF, UKF, and PF algorithm for an autonomous vehicle with three LiDAR and two RADAR in a highway scenario. Our first contribution is processing the point cloud data for each sensor and using a bounding box data type to normalize individual obstacles. Then, we designed a tracking system that can switch EKF, UKF, and PF tracking algorithms. Third, we use different state vector update matrices for LiDAR and RADAR for position updates and speed updates. Actual highway driving data is recorded, and a Robotic Operating System (ROS) model is built for algorithm development and result analysis.

4.2 Object Tracking Algorithm Implementation

The tracking system algorithm needs to be capable of simultaneously tracking multiple objects and handling diverse LiDAR and RADAR measurements. The structure of the proposed tracking algorithm is depicted in Figure 4.1.

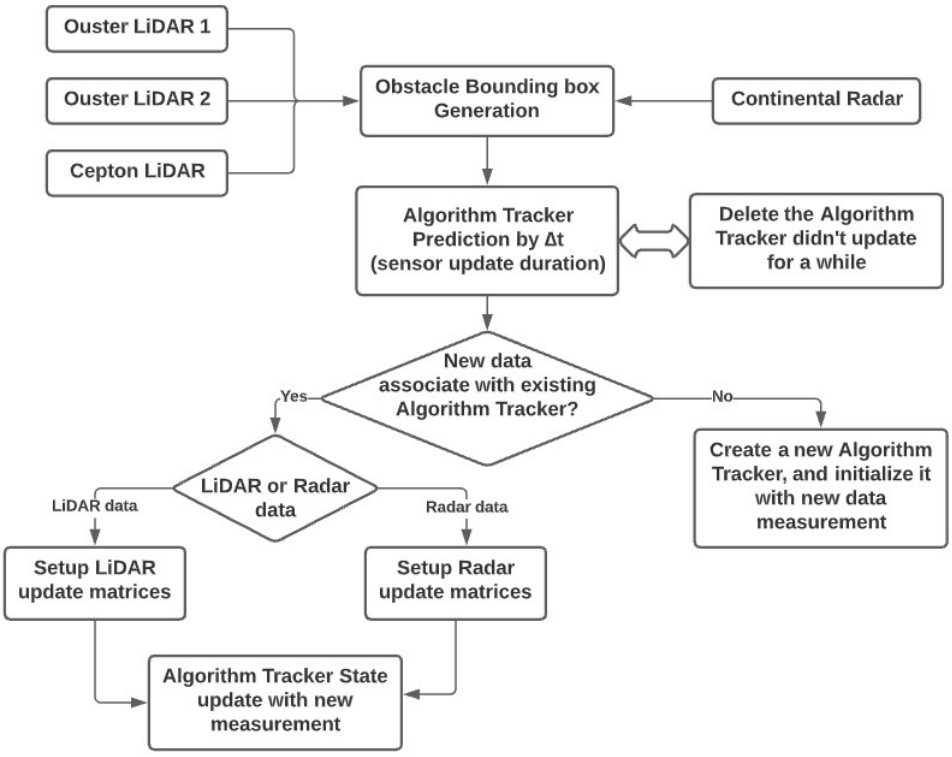

Figure 4.1 Vehicle tracking algorithm flowchart

The algorithm tracker also monitors the state of every detected obstacle, including its position, speed, size, and ID. When a new detection is received from LiDAR or RADAR, all existing algorithm trackers predict the obstacle's state by updating its state from the last timestamp with the duration of the sensor update Δt. Additionally, the algorithm tracker automatically deletes any old trackers that haven't received an update

for a certain duration based on Euclidean distance. This ensures that the algorithm is always tracking fresh obstacles and discarding obstacles that are out of detection range.

The designed tracking algorithm structure enables the tracking of multiple objects simultaneously and the handling of different LiDAR and RADAR measurements. After updating the algorithm tracking state, new data is associated with existing trackers based on Euclidean distance. Depending on whether the data is from LiDAR or RADAR, the tracker updates its state function by extracting position and velocity from LiDAR or velocity information from RADAR. If the data is not associated with any existing trackers, a new tracker is created and initialized with location and speed. The EKF, UKF, and PF algorithms are processed independently and simultaneously. The final step involves the correction of the algorithms by comparing the difference between prediction values and measurement data. It is possible to implement different tracking algorithms to the initialization, prediction, and correction steps of the designed tracking algorithm structure.

Define x_k as algorithm tracker movement state vector, shown in equation (4.1). Where P_x and P_y is the target's position, V_x and V_y is the target's velocity:

$$x_k = \left[P_x, V_x, P_y, V_y\right]^T \tag{4.1}$$

4.3 Extended Kalman Filter Tracking Model

EKF using a Jacobian matrix to construct a linear approximation of nonlinear, shown in equation (4.2), Δt is the observed variable duration between the existing tracking timestamp and new data timestamp. Equations (4.3-4.4) show the system measurement matrices; the matrix C_l is used for extracting position and velocity data from LiDAR, and the matrix C_r is used for extracting speed from RADAR measurement.

$$A(\Delta t) = \begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

$$C_l = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

$$C_r = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

Equation (4.5) shows measurement noise v_k and system process noise w_k are added to the system. They both obey Gaussian distribution.

$$\begin{cases} x_{k+1} = A(\Delta t)x_k + w_k \\ y_k = C_k x_k + v_k \end{cases} \tag{4.5}$$

Specify the process noise covariance matrix Q and the measurement noise covariance matrix R in equation (4.6). They both have zero means and a standard deviation of 0.5.

$$\begin{cases} Q = E[(w_k - \bar{w})(w_k - \bar{w})^T] \\ R = E[(v_k - \bar{v})(v_k - \bar{v})^T] \end{cases} \tag{4.6}$$

Define $x_{k|k-1}$ is system prediction vector of $x_{k-1|k-1}$ at time $k-1$ can obtain in equation (4.7). The system prediction covariance matrix $P_{k|k-1}$ can compute from equation (4.8).

$$x_{k|k-1} = f\left(x_{k-1|k-1}\right) = \begin{bmatrix} P_x + \Delta t \cdot V_x \\ V_x \\ P_y + \Delta t \cdot V_y \\ V_y \end{bmatrix} \tag{4.7}$$

$$P_{k|k-1} = A_{k-1} P_{k-1|k-1} A_{k-1}^T + Q \tag{4.8}$$

Equations (4.9) and (4.10) show the optimal state estimation $x_{k|k}$ and optimal covariance estimation $P_{k|k}$ update by the prediction at the previous timestamp $k-1$, and balance with the error of prediction value and measurement value.

$$x_{k|k} = x_{k|k-1} + K\left(y_k - f\left(x_{k|k-1}\right)\right) \tag{4.9}$$

$$P_{k|k} = (I - KC_{k-1}) P_{k|k-1} \tag{4.10}$$

Kalman gain K is computed from equation (4.11), depending on whether the current update is from LiDAR or RADAR data, replaced with measurement matrix C_l or C_r, R is also changed accordingly to match the covariance of the noise vectors corresponding to C_{k-1}.

$$K = P_{k|k-1} C_{k-1}^T \left(C_{k-1} P_{k|k-1} C_{k-1}^T + R\right)^{-1} \tag{4.11}$$

4.4 Unscented Kalman Filter Tracking Model

Figure 3.8 displays the outcome of the clustering algorithm for the bounding box. The UKF algorithm selects several sigma points around the mean state and approximates the nonlinear transformation of these points to obtain a better estimate of the state distribution. This is done by taking into account the particular relationship between the mean and standard deviation signals of each state dimension. The number of sigma points selected is dependent on the size of the state function and can be calculated using equation (4.12). In the case of the present vehicle nonlinear system with a state vector dimension of four, nine sigma points will be selected.

$$n_\sigma = 2n_x + 1 = 9 \tag{4.12}$$

The system state vector becomes a four-by-nine matrix based on the equation (4.13-4.15).

$$X_{k|k}^{[0]} = x_{k|k}, i = 0 \tag{4.13}$$

$$X_{k|k}^{[i]} = x_{k|k} + \sqrt{(n_x + \lambda)P_{k|k}^{[i]}}, i = 1, 2, \dots, n_x \tag{4.14}$$

$$X_{k|k}^{[i]} = x_{k|k} - \sqrt{(n_x + \lambda)P_{k|k}^{[i]}}, i = n_x + 1, \dots, n_x + n_x \tag{4.15}$$

Apply weight for nine state vectors from equation (4.16, 4.17) and the λ value calculated from equation (4.18).

$$W^{[0]} = \frac{\lambda}{n_\sigma + \lambda}, \qquad for\ i = 0 \tag{4.16}$$

$$W^{[i]} = \frac{1}{2(n_\sigma + \lambda)}, \qquad for\ i = 1, 2, \dots, n_\sigma \tag{4.17}$$

$$\lambda = 3 - n_x = -1 \tag{4.18}$$

The original state matrix X_{k+1} is the prediction from the state vector x_k, based on the sensor update rate Δt, shown in equation (4.19).

$$X_{k+1} = f(x_k, v_k) = \begin{bmatrix} P_x + \Delta t \cdot V_x \\ V_x \\ P_y + \Delta t \cdot V_y \\ V_y \end{bmatrix} \tag{4.19}$$

Calculate predicted mean and predicted covariance for original state and sigma vector in equation (4.20, 4.21)

$$x_{k+1|k} = \sum_{i=0}^{n_\sigma} W^{[i]} X_{k+1|k}^{[i]} \tag{4.20}$$

$$P_{k+1|k}^{[i]} = \sum_{i=0}^{2n_\sigma} W^{[i]} (X_{k+1|k}^{[i]} - x_{k+1|k})(X_{k+1|k}^{[i]} - x_{k+1|k})^T \tag{4.21}$$

Transfer predicted state to measurement space and calculate expected measurement means and covariance, shown in equation (4.14-4.13). For LiDAR measurement prediction, it is a four-by-nine matrix to extract position and velocity and z_{k+1} will be a two-by-nine matrix to extract velocity from RADAR measurement prediction. Equation (4.22-4.25) shows the defined measurement noise covariance matrix with zero means and a standard deviation of 0.5.

$$Z_{k+1}^{[i]} = h(x_{k+1}) \tag{4.22}$$

$$z_{k+1|k} = \sum_{i=0}^{n_\sigma} W^{[i]} Z_{k+1|k}^{[i]} \tag{4.23}$$

$$S_{k+1|k}^{[i]} = \sum_{i=0}^{2n_\sigma} W^{[i]} (Z_{k+1|k}^{[i]} - z_{k+1|k})(Z_{k+1|k}^{[i]} - z_{k+1|k})^T + R \tag{4.24}$$

$$R = E[w_k \cdot w_k{}^T] \tag{4.25}$$

Kalman gain is calculated from equation (3.26), and the state vector update and UKF cross correction between sigma points in state space and measurement space come from equation (4.27-4.29).

$$K_{k+1|k} = T_{k+1|k} S_{k+1|k}^{-1} \tag{4.26}$$

$$x_{k+1|k+1} = x_{k+1|k} + K_{k+1|k}\left(z_{k+1} - z_{k+1|k}\right) \tag{4.27}$$

$$P_{k+1|k+1} = P_{k+1|k} - K_{k+1|k} S_{k+1|k} K_{k+1|k}^{T} \tag{4.28}$$

$$T_{k+1|k} = \sum_{i=0}^{2n_\sigma} W^{[i]} \left(X_{k+1|k}^{[i]} - x_{k+1|k}\right)\left(Z_{k+1|k}^{[i]} - z_{k+1|k}\right)^{T} \tag{4.29}$$

4.5 Particle Filter Tracking Model

Particle Filter (PF) is a popular algorithm for tracking in nonlinear and non-Gaussian systems. It uses Monte Carlo methods to represent the posterior probability distribution by extracting random state particles. The PF algorithm starts by defining the number of particles (i), and each particle has the same properties as the algorithm tracker, including position, speed, size, and id. Then, the algorithm initializes all the particles with measurements from LiDAR or RADAR.

In the prediction step, each particle's state is updated by predicting its position and speed based on the previous state and the system's dynamic model. The measurement update step involves computing the likelihood of each particle given the current measurement. The prediction of the Particle state is based on the current value and sensor update sequence, as shown in equation (4.30). A random computer engine generates the added measurement.

$$Particle[i]_{k+1} = f(x_k, v_k) = \begin{bmatrix} P_x + \Delta t \cdot V_x \\ V_x \\ P_y + \Delta t \cdot V_y \\ V_y \end{bmatrix} + noise \tag{4.30}$$

Each particle has its weight, which presents the likelihood between prediction and new measurement. This paper uses multivariate Gaussian distribution to sign the weight for each particle, shown as equation (4.31).

$$Weight = P(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{(x-\mu_x)^2}{2\pi\sigma_x^2} + \frac{(y-\mu_y)^2}{2\pi\sigma_y^2}\right)} \tag{4.31}$$

Where, $P(x, y)$ is current particle position, σ_x, σ_y is the standard deviation, and μ_x, μ_y is the new measurement from sensors.

After all particles are weighted, a resampling process will apply to reselect i particles depend on the value of weight. High-weight particle has more probability of being reselected, and the selection continues over i times. The resampled particles are a combination of the estimated position for the obstacle, and this is the final prediction of the particle filter in the current iteration.

4.6 Results

Figure 3.8 displays the outcome of the clustering algorithm for the bounding box. To compare the tracking results of EKF, UKF, and PF, this study employs parallel computing to run all three algorithms on the same dataset. Figure 4.2 illustrates the bounding box arrays generated by the three algorithm trackers: UKFObjectArray, EKFObjectArray, and PFObjectArray, shown in purple, red, and blue colors, respectively, in RVIZ. The tracking IDs are also displayed in RViz, and each algorithm has a unique counting system. Additionally, two green-colored bounding boxes are generated by either LiDAR or RADAR.

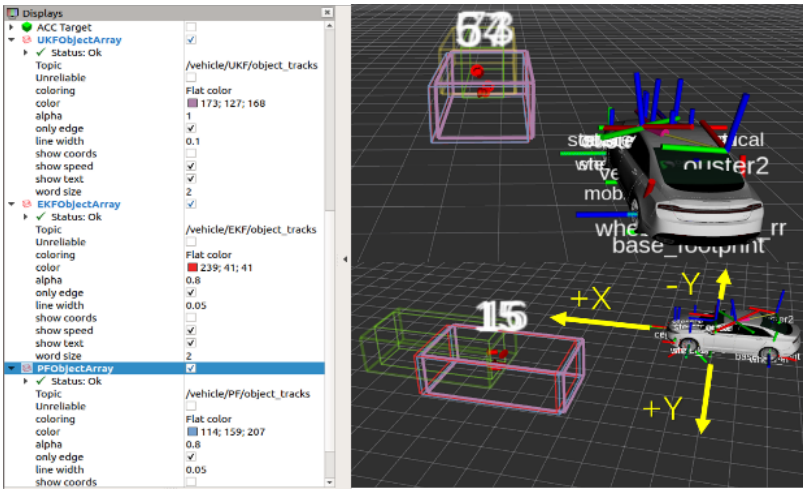

Figure 4.2 Object tracker (pink bounding box) shows in ROS Rviz

There are two situations when ego vehicle tracking surrounding obstacles: the obstacle detected by a single sensor or multiple sensors. Figure 4.3 shows the single sensor tracking performance for an object position P_x, $+X$is the vehicle heading direction. The purple curve presents the LiDAR measurement for obstacle P_x value. This obstacle is on the ego vehicle's left side, and the nonlinear system makes the sensor's measurement

not smooth. EKF and PF trackers react fast, follow the detection data tightly, and UKF performance as a smooth tracking path.

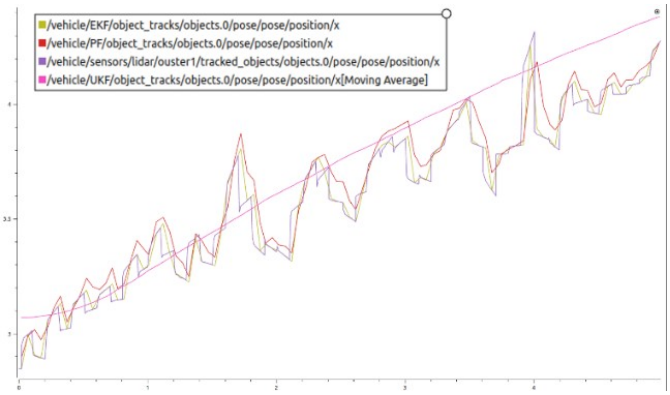

Figure 4.3 Object real-time position x vs. tracker position x

Figure 4.4 shows the tracking object's position P_y, Y direction is the vehicle's horizontal value, changing in small numbers. EKF and PF algorithm tracking in a fast reaction, UKF tracking still in a smooth way.

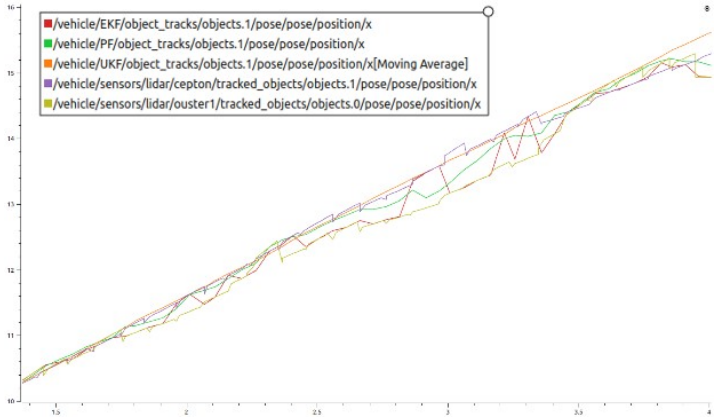

Figure 4.4 Object real-time position y vs. tracker position y

Figure 4.5 shows the object velocity V_x tracking results. EKF reaction is very fast and almost matches each detection value, PF prediction moves between the two measurements, and UKF presents a soft tracking path.

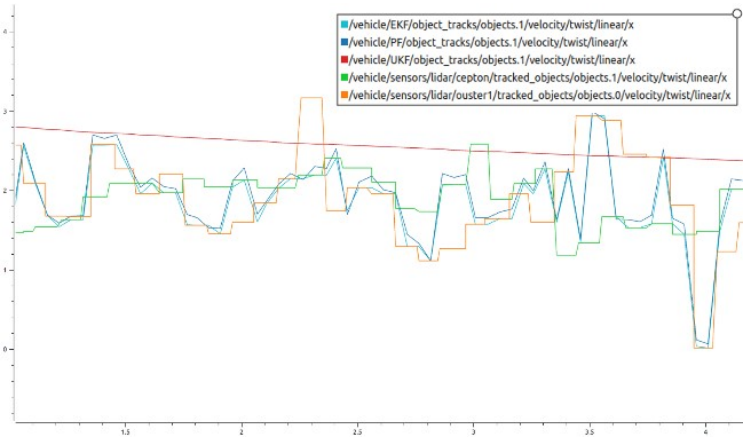

Figure 4.5 Object real-time velocity x vs. tracker position x

For obstacle V_y speed tracking performance is shown in Figure 4.6, vehicle lateral speed V_y moving in a small range. EKF hits two measurement values, PF moving in between, and the soft middle curve is generated from the UKF algorithm.

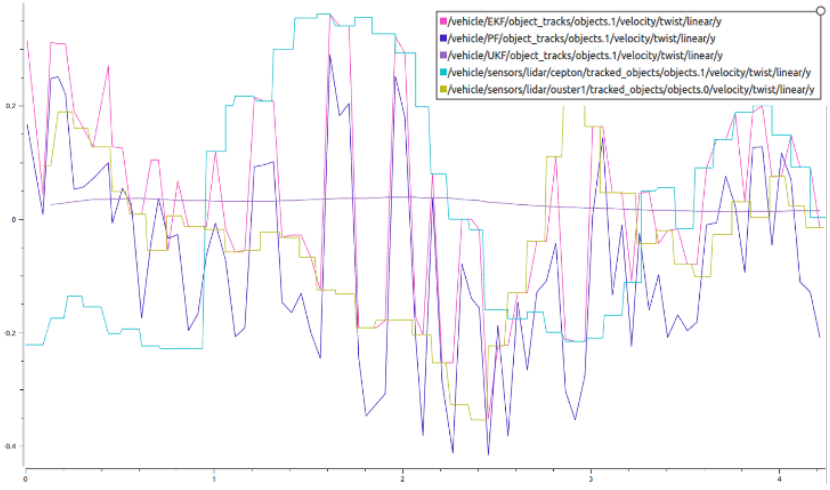

Figure 4.6 Object real-time velocity y vs. tracker position y

This chapter investigated and compared the performance of three popular tracking algorithms, namely the Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), and Particle Filter (PF) for an autonomous vehicle equipped with three different sensors in a highway tracking scenario. The EKF algorithm employs the Jacobian linearization function and predicts obstacle states with Gaussian distribution noise. The UKF applies the Taylor expansion equation and generates $n_\sigma = 2n_x + 1$ sigma points to improve predictions. The PF uses Monte Carlo methods to generate particles and selects high-probability particles using a weight method to predict obstacle states. The EKF tracker demonstrated the fastest reaction speed and robustly handles different measurements. The PF tracker exhibited moderate reaction speed and performed well in integrating data from multiple sensors. The UKF tracker provided the smoothest prediction curve and presented a visually appealing output. The tracking algorithm, while effective for its intended application in highway driving scenarios, does have certain limitations. Its design primarily revolves around optimizing the identification of highway vehicles through the bounding box. However, this specialization poses challenges when attempting to distinguish other objects, such as pedestrians and bicycles. Therefore, further refinements and adaptations are necessary to extend its functionality to a broader range of objects and scenarios.

The next chapter will explore combining these algorithms with deep learning and reinforcement learning to develop a more advanced tracking algorithm for autonomous vehicles with multi-sensor fusion and tracking capabilities.

CHAPTER FIVE

LiDAR CAMERA SENSOR FUSION AND CALIBRATION ALGORITHM

5.1 Introduce LiDAR and Camera Sensor Fusion

Sensor fusion is a powerful technique that involves combining data from multiple sensors to obtain a more complete and accurate understanding of the environment. The integration of different types of sensors can provide complementary information, improving the overall quality and reliability of the data. One example of sensor fusion is the combination of data from LiDAR and camera sensors. LiDAR sensors use laser beams to detect the distance and position of objects in the environment, while camera sensors capture visual information that provides valuable context about the scene [56-65]. By merging LiDAR and camera data, it becomes possible to obtain a more comprehensive and precise understanding of the environment, which in turn enables better object detection and recognition. This technique also enhances system reliability by compensating for errors or failures in one sensor with the other. LiDAR and camera sensor fusion is a powerful tool with numerous applications, including autonomous vehicles, robotics, and augmented reality.

The main challenge in fusing data from camera and LiDAR sensors is the discrepancy in their data representations. Cameras capture 2D images that require feature extraction algorithms for object recognition and tracking, whereas LiDAR sensors capture 3D point clouds that require processing techniques for feature extraction and object segmentation. These differences in data representations and processing

requirements can make it difficult to effectively fuse the data from these two sensors. Additionally, errors in sensor calibration and alignment can further complicate the fusion process, leading to inaccurate or unreliable results.

5.2 Vehicle Platform Setup and Sensor Calibration

Particle Filter (PF) is a popular algorithm for tracking in nonlinear. The calibration of LiDAR and camera sensors is a crucial process that requires aligning their coordinate systems to facilitate precise data fusion. Due to their distinct measurement principles and units, it is necessary to transform their data into a common coordinate system before fusing it. The calibration process usually involves collecting and preprocessing data, intrinsic and extrinsic calibration, as well as validation.

Calibrating LiDAR and Camera sensors can pose challenges due to their complexity and variability, and accurately modeling their internal parameters can be difficult. Nonetheless, precise calibration is crucial for achieving accurate fusion of LiDAR and Camera data, which is vital for autonomous driving and other applications. Despite its importance, the process of calibrating these sensors can be time-consuming and resource-intensive. To tackle these challenges, this chapter introduces a self-calibrating algorithm for LiDAR and Camera sensors. Automating the calibration process with this algorithm can significantly reduce the time and effort required for calibration while also enhancing accuracy and consistency.

The platform for developing the algorithm features a Chevrolet Cruse equipped with an Ouster 64 LiDAR mounted on the roof and a Logitech Camera attached to the

windshield, as shown in picture 5.1. To create a vehicle simulation model in ROS with identical dimensions and sensor placement, the team designed it to match the length, height, and sensor position of the actual vehicle.

Figure 5.1 Vehicle platform and ROS simulation

The TF tree for the vehicle model illustrates the spatial relationship between the vehicle body and its sensors, in terms of Roll, Pitch, Yaw, x, y, and z values. The base_footprint frame represents the vehicle's lowest point on the ground, positioned at the midpoint between the rear wheels. The vehicle_cam frame displays the position of the camera in space and is also considered a child frame under the LiDAR frame.

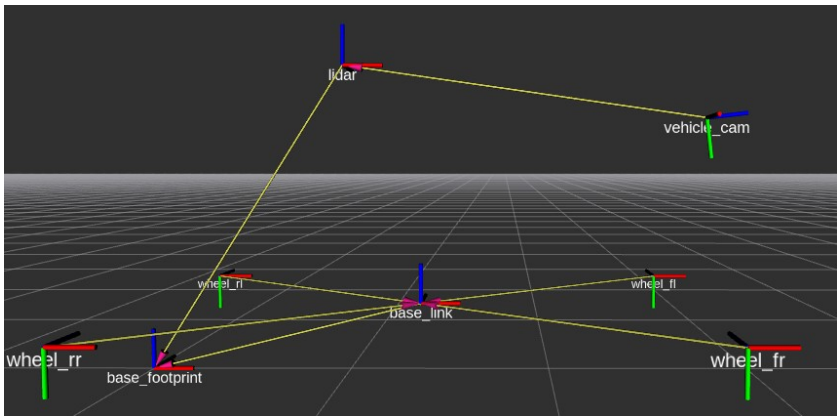

Figure 5.2 Vehicle sensor system TF (transfer frame) tree

The accuracy of the sensor frames' transformation can significantly impact the results of sensor fusion. As depicted in Picture 5.3, the fusion of LiDAR and Camera data was tested using a checkerboard with two holes. In ROS RViz, the LiDAR point cloud data and vehicle TF tree are displayed, and all LiDAR points are aligned with the 2D image. The point cloud data presents the ground, tall lights, and the calibration board.

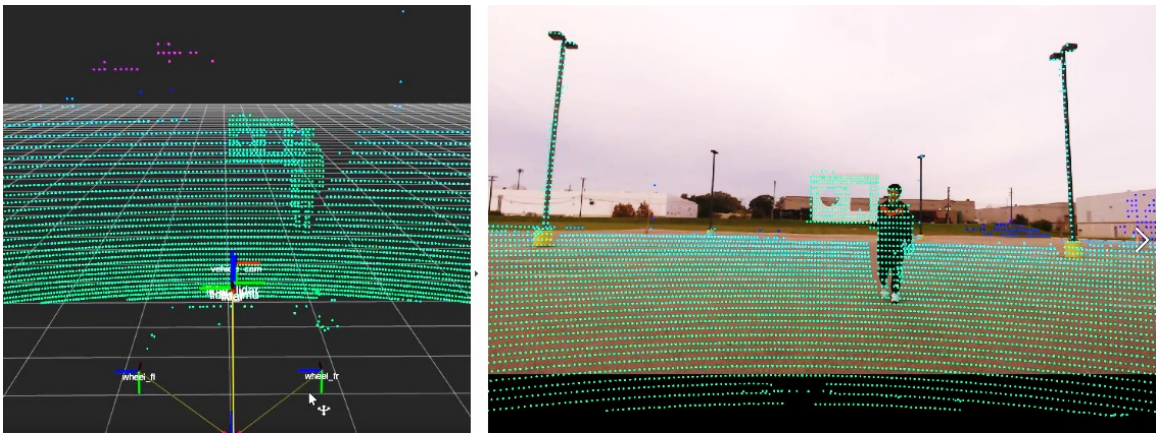

Figure 5.3 LiDAR and Camera manual calibration

5.3 LiDAR and Camera Projection Algorithm

To begin the LiDAR-to-2D image project, the first step involves determining the intrinsic and extrinsic parameters of both the LiDAR and Camera sensors. The intrinsic parameters of the Camera include the focal length, principal point, and distortion coefficients, whereas the extrinsic parameters describe the relative orientation and position of the LiDAR and Camera coordinate systems. With these parameters known, the 3D point cloud obtained from the LiDAR sensor can be transformed into the Camera coordinate system using the extrinsic parameters.

Transforming the 3D point cloud into the Camera coordinate system involves a rigid transformation that takes into account the position and orientation of the Camera relative to the LiDAR. This transformation enables the LiDAR data to be projected onto the 2D image plane of the Camera, which facilitates the creation of a 2D image that corresponds to the 3D point cloud. The accuracy of this transformation is crucial for ensuring that the resulting 2D image accurately represents the features and geometry of the 3D point cloud. PCD contains space point (x, y, z), 2D image present by 2D pixels(x, y). The project 2D LiDAR can get from the Camera intrinsic matrix K multiple extrinsic matrix $[R|t]$.

$$P = K \times [R|t] = \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & \frac{s}{f_x} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times (I|t) \times \begin{pmatrix} R & 0 \\ 0 & 1 \end{pmatrix} \quad (4.1)$$

The 2D rotation matrix for R_x, R_y, R_z show as below:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi_x & -sin\phi_x \\ 0 & sin\phi_x & cos\phi_x \end{bmatrix}$$

$$R_y = \begin{bmatrix} cos\Theta_y & 0 & sin\Theta_y \\ 0 & 1 & 0 \\ -sin\Theta_y & 0 & cos\Theta_y \end{bmatrix} \quad (4.2)$$

$$R_z = \begin{bmatrix} cos\Theta_z & -sin\Theta_z & 0 \\ sin\Theta_z & cos\Theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$Joint\ Rotation: R = R_z \cdot R_y \cdot R_x$

Figure 5.4 displays the outcome of the 2D projection of the PCD and the CNN classification. The YOLO neural network generated a green bounding box with a "person" label for the detected individual. A green dot on the image represents the 3D point cloud projected onto the 2D plane.

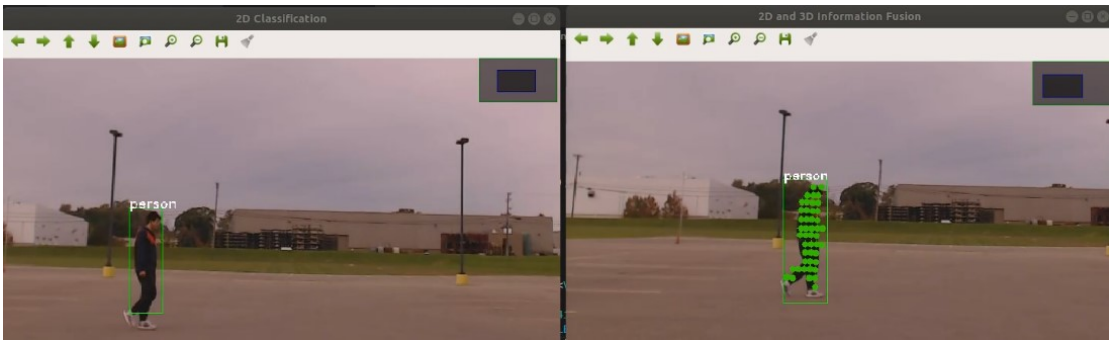

Figure 5.4 Camera CNN detection and LiDAR PCD 2D Projection

However, the main challenge in calibrating the Camera and LiDAR sensors for fusion is accurately determining the relative position and orientation of the two sensors. This requires determining both the intrinsic parameters of each sensor (such as focal length, principal point, and distortion coefficients for cameras) and the extrinsic parameters that describe their relative positions and orientations. Errors in calibration can lead to misalignment between the two-sensor data, resulting in inaccurate fusion and potentially compromising the quality of the output. Additionally, differences in sensor hardware and mounting configurations can introduce additional challenges, requiring careful consideration during the calibration process. Figure 5.5 shows the LiDAR and Camera mis-calibration example in real data.

Figure 5.5 Calibration error results in LiDAR-Camera sensor fusion

Another challenge in calibration is the time and effort required to collect and label data for calibration. This is particularly challenging for dynamic environments, where the scene is constantly changing, and calibration data may need to be updated frequently to maintain accuracy.

Therefore, self-calibration algorithms are very important for LiDAR and camera sensor fusion because they help to overcome the challenges and limitations of manual calibration methods.

5.4 LiDAR and Camera Calibration Algorithm

Particle Filter (PF) is a popular algorithm for tracking in nonlinear. Manual calibration of LiDAR and camera sensors can be time-consuming and can lead to significant discrepancies in the output due to small errors in calibration. To address these

challenges, a self-tuning algorithm can be utilized to refine the calibration and automatically correct any errors in the transfer matrix. The initial step in this process involves identifying the 2D shapes of objects in both the camera and LiDAR data. To accomplish this, a contour hull is used to create an outline of the contour by tracing lines around the entire image, while a convex hull is used to create a convex curve around the object. The main difference between a contour and a convex hull is that a contour represents the boundary of an object, while a convex hull represents the smallest convex shape that can contain a set of points. Contours can be irregular and have concave sections, while a convex hull is always a convex shape with straight edges connecting the points.

As shown in Figure 5.6, the blue line in the picture represents the 2D image contour, while the green line represents the convex hull.

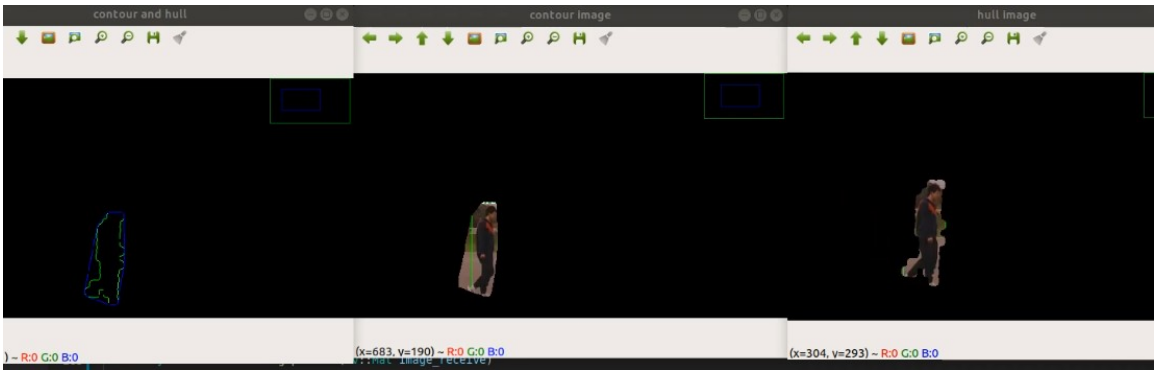

Figure 5.6 2D image pedestrian tracking using contours and convex hull algorithms.

In LiDAR and camera fusion, 2D images are utilized to track the location of individuals and create contours as well as convex hulls. In a similar vein, a 2D point

cloud is utilized to generate an alternative representation of individuals. An algorithm is then employed to compare these two shapes in terms of scale and rotation, resulting in an accurate transfer matrix. Figure 5.7 showcases all the preparatory data. The top left image depicts a 2D classification with people labeled in green. The top right image illustrates object tracking with a binary image decomposed into 2D pixel points. The middle-left picture shows LiDAR points projected onto the RGB image with green dots. The middle right picture shows the hull and contour algorithm that describes the tracked object. Finally, the bottom images exhibit the RGB image extracted by the contour and hull algorithms.

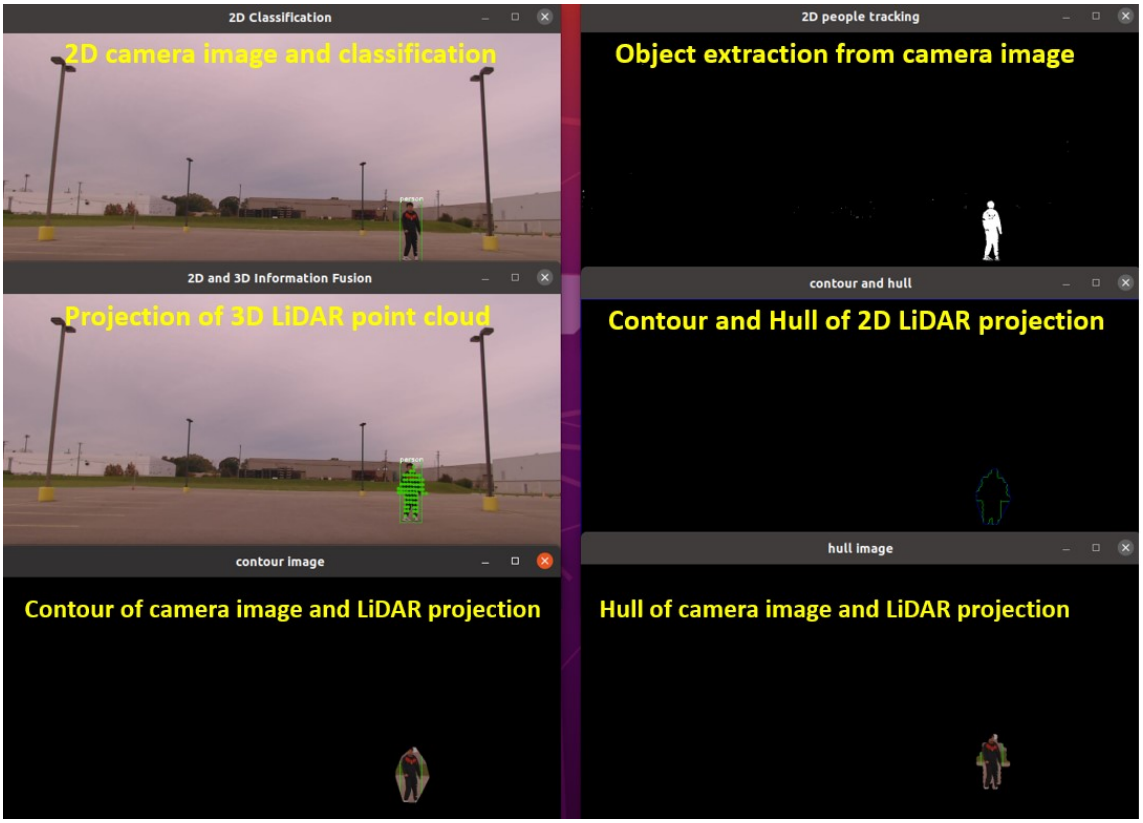

Figure 5.7 Combining 2D image and 3D point cloud data.

In the context of LiDAR and Camera calibration algorithm, Singular Value Decomposition (SVD) is a powerful mathematical technique that can be used to identify the relationship between LiDAR coordinates and camera coordinates by decomposing a matrix into three separate matrices. This helps to uncover any patterns or features in the data and can be a valuable tool for understanding how the LiDAR and camera data relate to each other and for optimizing the fusion algorithm. Additionally, the Gradient Descent (GD) algorithm can be used to optimize the SVD algorithm by minimizing the cost function of the machine learning model. By breaking the sensor fusion process into smaller, smoother steps, GD can help to identify the distance vector and rotation matrix between LiDAR coordinates and Camera coordinates in a more efficient way, ultimately improving the accuracy and speed of the sensor fusion process. The LiDAR and camera calibration algorithm is a mathematical process that helps to align and synchronize the data from both sensors. This algorithm is developed and implemented using MATLAB software.

In the beginning, the algorithm takes the raw data from both LiDAR and camera sensors at time t_0, which is shown in Figure 5.8. The red outline data represents the people detected by the camera in 2D, while the pink outline data represents the people detected by LiDAR. However, due to some random errors or misalignments, there might be a discrepancy between the two data sets, leading to a mis-calibration situation. This discrepancy can be in the form of incorrect scaling or rotation values.

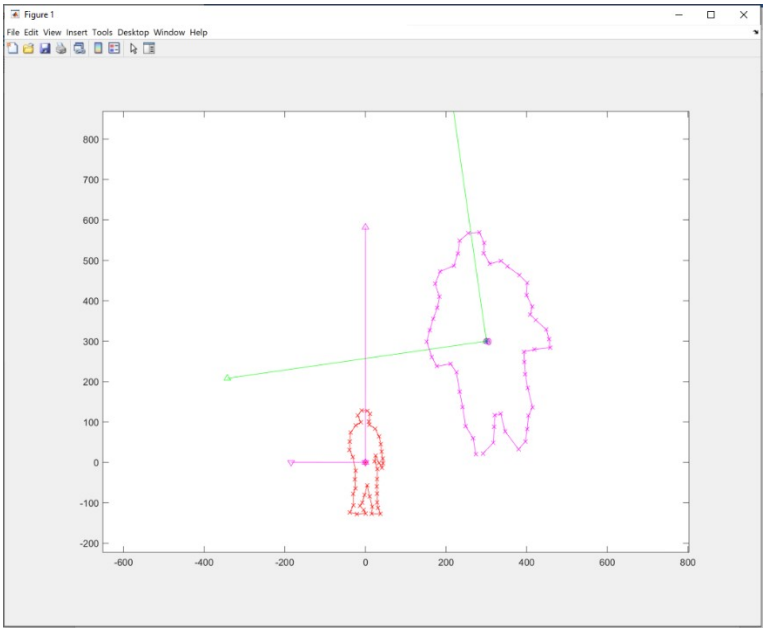

Figure 5.8 Camera and LiDAR contours calibration process at time t_0

The LiDAR and camera calibration algorithm is designed to identify and correct these random values, which will make the two systems well-matched and calibrated. By applying this algorithm, the discrepancies between the two data sets can be minimized, leading to more accurate and precise sensor fusion results.

Let's denote and respective LiDAR data as matrix p_i and Camera data as matrix q_i:

$$p_i = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}_i = \begin{bmatrix} p_{x_i} \\ p_{y_i} \\ p_{z_i} \end{bmatrix} \qquad q_i = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}_i = \begin{bmatrix} q_{x_i} \\ q_{y_i} \\ q_{z_i} \end{bmatrix} \tag{5.1}$$

With reference to the p-frame, the distance vector and rotation matrix from the p-frame to the q-frame is:

$$d_q^p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_q^p$$

$$R_q^p = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix}$$
$$= \begin{bmatrix} c\psi c\theta & -s\psi c\theta + c\psi s\theta s\phi & s\psi s\phi + c\psi s\theta c\phi \\ s\psi c\theta & c\psi c\phi + s\psi s\theta s\phi & -c\psi s\phi + s\psi s\theta c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \quad (5.2)$$

The forward kinematics relationship between the position vector p and q is given by:

$$p_i = d_q^p + R_q^p q_i \quad (5.3)$$

If a collection of m sets of data at each frame can be stored as metrics

$$P = [p_1\ p_2 \dots\ p_m] \qquad Q = [q_1\ q_2 \dots\ q_m] \quad (5.4)$$

Then, the following matrix relationship follow:

$$P = D_q^p + R_q^p Q \qquad D_q^p = d_q^p 1_{1*m} \quad (5.5)$$

The relationship between the means of the data is similarly related as

$$\bar{p} = d_q^p + R_q^p \bar{p} \qquad where \qquad \bar{p} = \frac{1}{m}\sum_{i=1}^{m} p_i\ , \bar{q} = \frac{1}{m}\sum_{i=1}^{m} q_i \quad (5.6)$$

Let H be the product $H = [P - \bar{P}][Q - \bar{Q}\ ]'$, where $[P - \bar{P}]$ be the variations of positions P from the mean position $\bar{p}$, where $\bar{P} = \bar{p}1_{1*m}$, and $[Q - \bar{Q}\ ]'$ where $\bar{Q} = \bar{q}1_{1*m}$. The goal is decomposed $H = USV'$, where U and V are unitary matrices in the minimal singular value decomposition, and S is a matrix of singular values of H.

Use the forward kinematics to express H as:

$$H = \left[D_q^p + R_q^p Q - D_q^p + R_q^p \bar{Q}\right][Q - \bar{Q}\ ]' \quad (5.7)$$
$$= R_q^p [Q - \bar{Q}\ ][Q - \bar{Q}\ ]'$$

Use similarity transformation for a symmetric matrix to decompose relationship into:

$$H = R_q^p T_{QQ} \Lambda_{QQ} T'_{QQ} \tag{5.8}$$
$$\Lambda_{QQ} = \left[diag\{\lambda_j([Q-\bar{Q}][Q-\bar{Q}]'), j = 1,2,3; \lambda_1 \geq \lambda_2 \geq \lambda_3\}\right]$$
$$\lambda_j = eigenvalue\ of\ [Q-\bar{Q}][Q-\bar{Q}]'$$
$$T_{QQ} = modal\ matrix\ of\ [Q-\bar{Q}][Q-\bar{Q}]',\ T'_{QQ} = T_{QQ}\ \ or\ \ T'_{QQ} = I$$

Decomposing H using SVD:

$$H = USV' \tag{5.9}$$

Where,

$$S = \left[diag\left\{\sqrt{\lambda_j(H'H)}, j = 1,2,3; \lambda_1 \geq \lambda_2 \geq \lambda_3\right\}\right]$$
$$UU' = U'U = I \tag{5.10}$$
$$V'V = VV' = I$$

It is seen that,

$$H'H = [Q-\bar{Q}][Q-\bar{Q}]' R_q^{p'} R_q^p [Q-\bar{Q}][Q-\bar{Q}]' \tag{5.11}$$
$$= \left[[Q-\bar{Q}][Q-\bar{Q}]'\right]^2$$

So, it turns out that,

$$S = \left[diag\left\{\sqrt{\lambda_j(H'H)}, j = 1,2,3\right\}\right]$$
$$= \left[diag\left\{\sqrt{\lambda_j\left(\left[[Q-\bar{Q}][Q-\bar{Q}]'\right]^2\right)}, j = 1,2,3\right\}\right]$$
$$= \left[diag\{\lambda_j([Q-\bar{Q}][Q-\bar{Q}]'), j = 1,2,3\}\right] = \Lambda_{QQ} \tag{5.12}$$

From the relationship $H = R_q^p T_{QQ} \Lambda_{QQ} T'_{QQ} = USV'$, one finds that.

$$U = R_q^p T_{QQ} \tag{5.13}$$
$$V' = T'_{QQ}$$

Therefore, the rotation matrix between LiDAR and Camera can be present as:

$$R_q^p = UV' \tag{5.14}$$

Once we have obtained the rotation matrix that describes the orientation between the LiDAR and Camera, we can input this result into the Gradient Descent function. This function allows us to iteratively optimize the values of rotation and the distance scale vector by minimizing the cost function. This helps us to obtain more accurate results. Once we have calculated the optimal values for these parameters, we can apply the new frame transfer tree to get a more precise estimate of the orientation and distance between the LiDAR and the Camera.

The designed Gradient Descent state is presented as:

$$\theta = [p_{x_i}, p_{y_i}, radMaja, radMajb] \tag{5.15}$$

Where, p_{x_i} , p_{y_i} are the LiDAR data (able switch to Camera data); s is the scale change between LiDAR and Camera data, initial as 1; radMajb is the major rotation calculated from the SVD algorithm, $\text{radMajb} = atan2(V'(2,1)V'(1,1))$.

Gradient Decent integration process:

$$\theta\_new = \theta\_old - \gamma \nabla J(\theta) \tag{5.16}$$

The cost function is presented as:

$$J(\theta) = q_1\left(p_{x_i} - \bar{p}_{x_i}\right)^2 + q_2\left(p_{y_i} - \bar{p}_{y_i}\right)^2 + q_3(radMaja - s * radMajb)^2 - q_4(angMaja - angMajb)^2 \tag{5.17}$$

The partial derivative cost function is presented as:

$$\Delta J(\theta) = [2q_1\left(p_{x_i} - \bar{p}_{x_i}\right), 2q_2\left(p_{y_i} - \bar{p}_{y_i}\right), -2q_3(radMaja - s * radMajb) * s, -2q_4(angMaja - angMajb)] \tag{5.18}$$

Where $\text{q} = [q_1, q_1, q_1, q_1]$ is an adjust matrix.

After deriving the necessary equation for the calibration process, MATLAB can be used to implement and execute the required computations. Figure 5.9 provides a visual representation of the LiDAR data changing as a result of rotation, scale, and position adjustments during the gradient descent iteration from t_0 to t_n. As the calibration process iteratively optimizes these parameters, the LiDAR data eventually matches the target data with minimal error in rotation, scale, and position. This figure allows us to see the calibration process in action and visualize the changes to the LiDAR data as the calibration progresses.

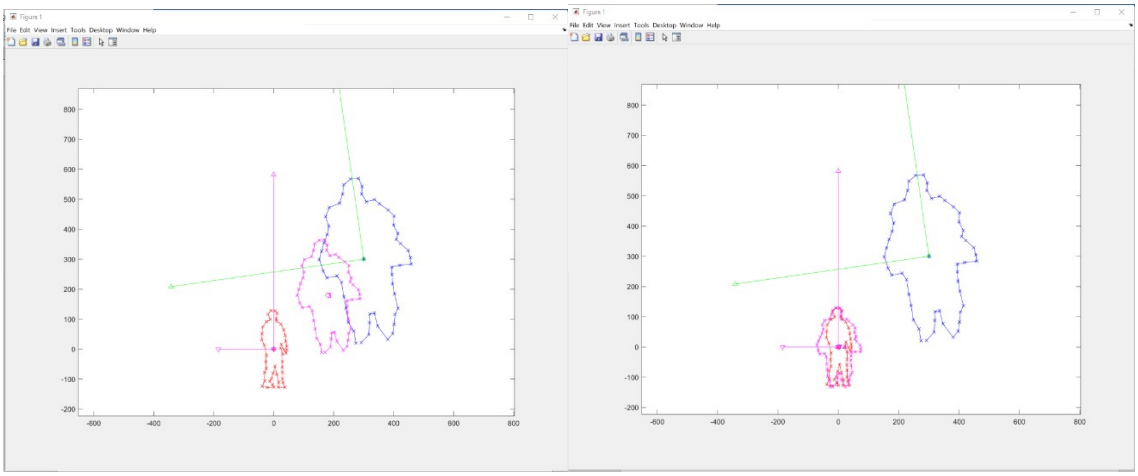

Figure 5.9 Camera and LiDAR calibration process at time $t_{n/2}$ and t_n

The combination of the Singular Value Decomposition (SVD) and Gradient Descent (GD) algorithms provides a highly effective approach for completing the LiDAR and Camera calibration task. By tuning the parameters of both algorithms, we can achieve fast and accurate results. Additionally, the designed algorithm is not limited to a specific type of sensor or calibration system, making it versatile and applicable in a variety of contexts.

5.5 Results

Particle Filter (PF) is a popular algorithm for tracking in nonlinear. Figure 5.10 and 5.11 illustrates the mis-calibration of the LiDAR-Camera sensor fusion, and correction from t_0 to t_n.The LiDAR data is represented by blue points, the Camera data by red points, and the processing state by pink points.

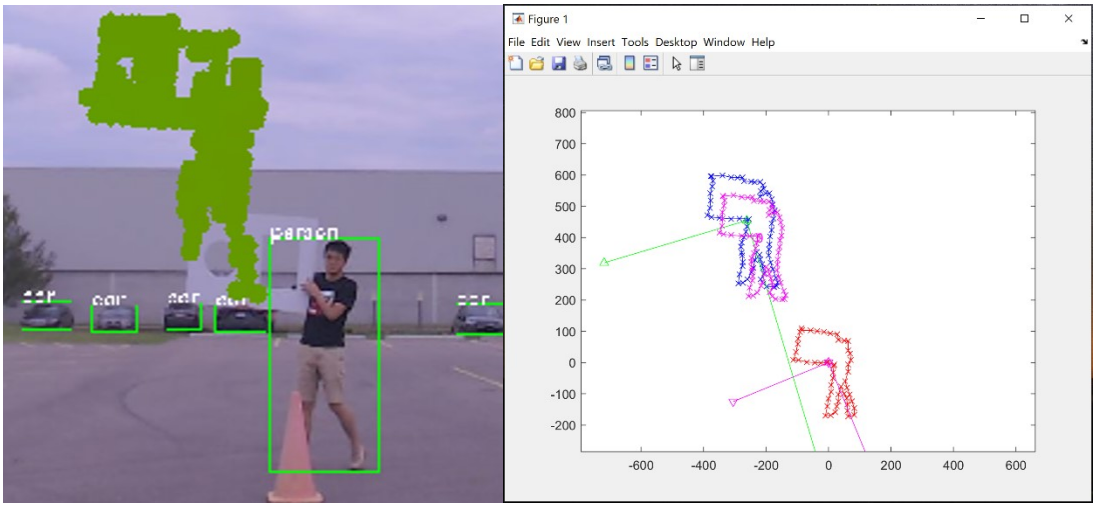

Figure 5.10 Self-calibration example 1 at time t_0

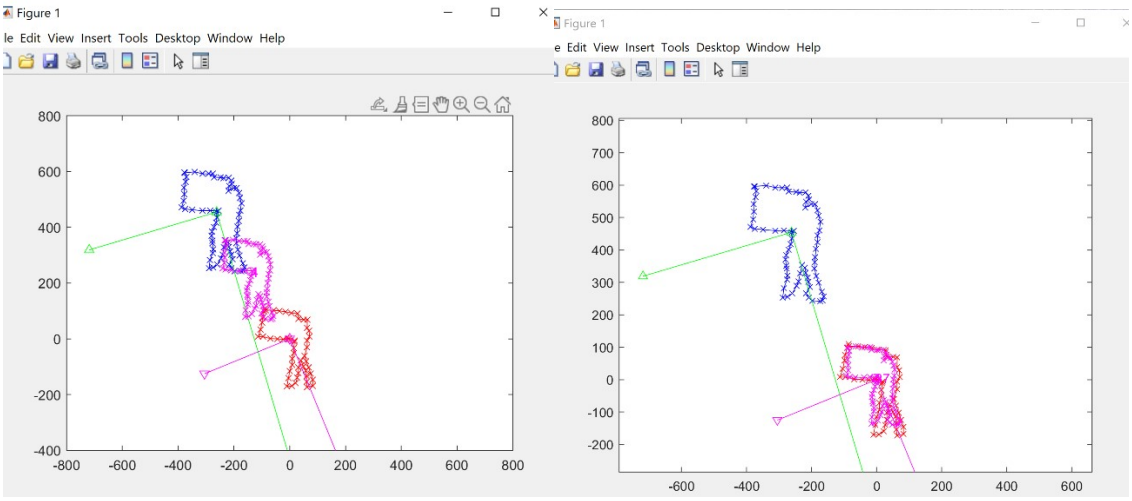

Figure 5.11 Self-calibration example 1 at time $t_{n/2}$ and t_n

Figure 5.12 displays an example of mis-calibration where the LiDAR data exhibits scale change and offset in relation to the Camera data. This may be caused by errors in the Camera's TF tree or distortion calibration. On the other hand, Figure 5.13 illustrates how the self-calibration algorithm can effectively solve this issue.

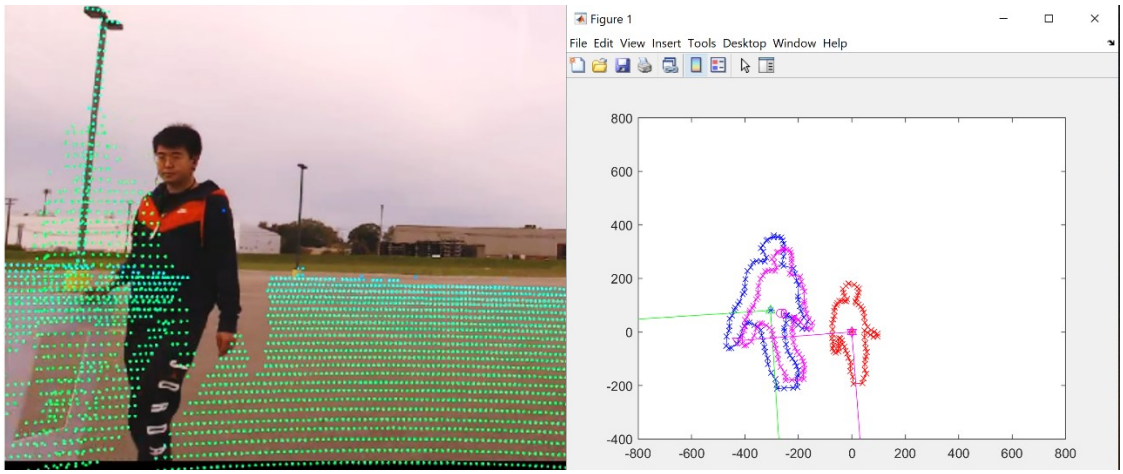

Figure 5.12 Self-calibration example 2 at time t_0

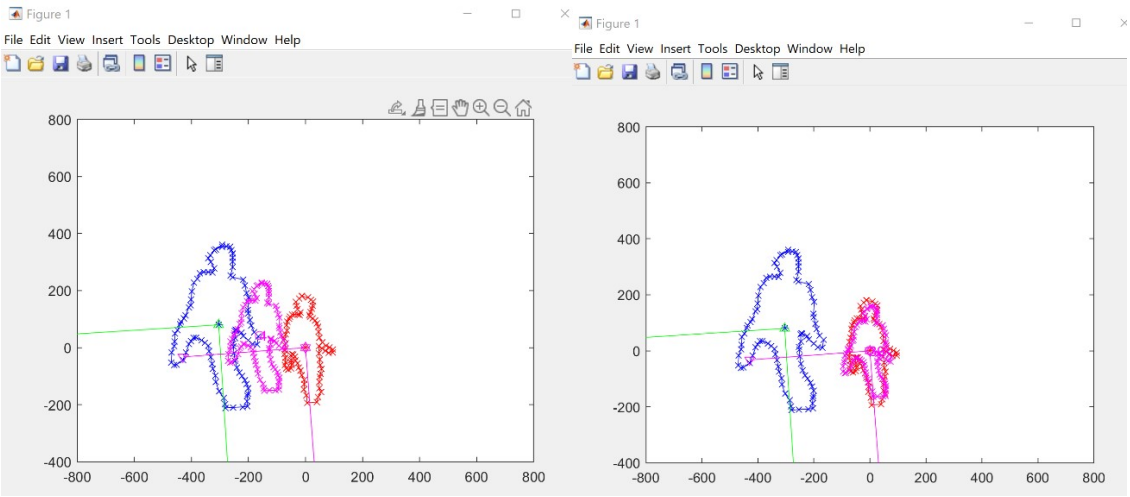

Figure 5.13 Self-calibration example 2 at time $t_{n/2}$ and t_n

When self-driving vehicles operate on the highway, the position of their sensors may be affected by strong winds or vibrations. As illustrated in Figure 5.14, the LiDAR

data appears to be positioned above the Camera detection. Figure 5.15 demonstrates the algorithmic processing involved in addressing this issue.

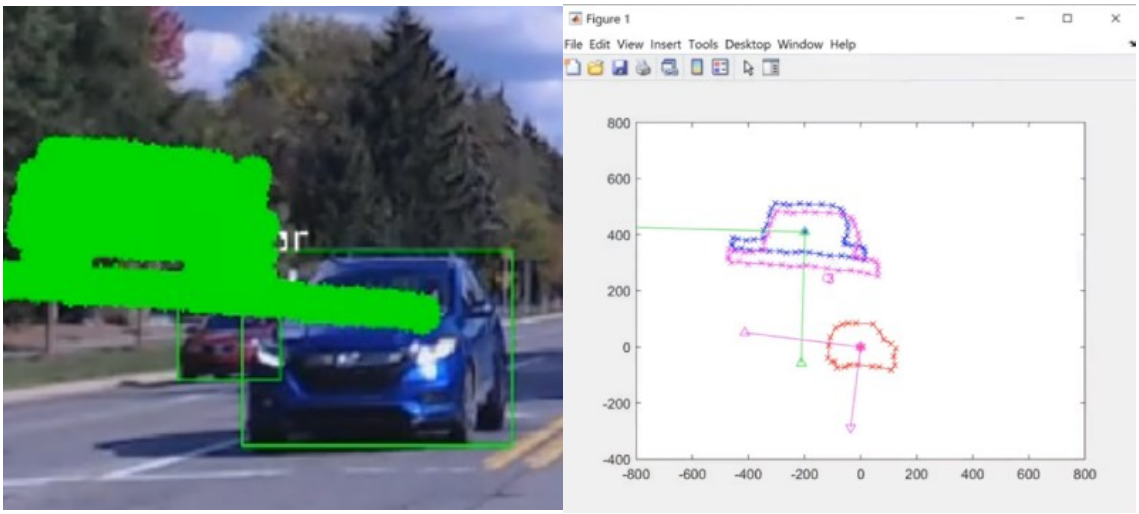

Figure 5.14 Self-calibration example 3 at time t_0

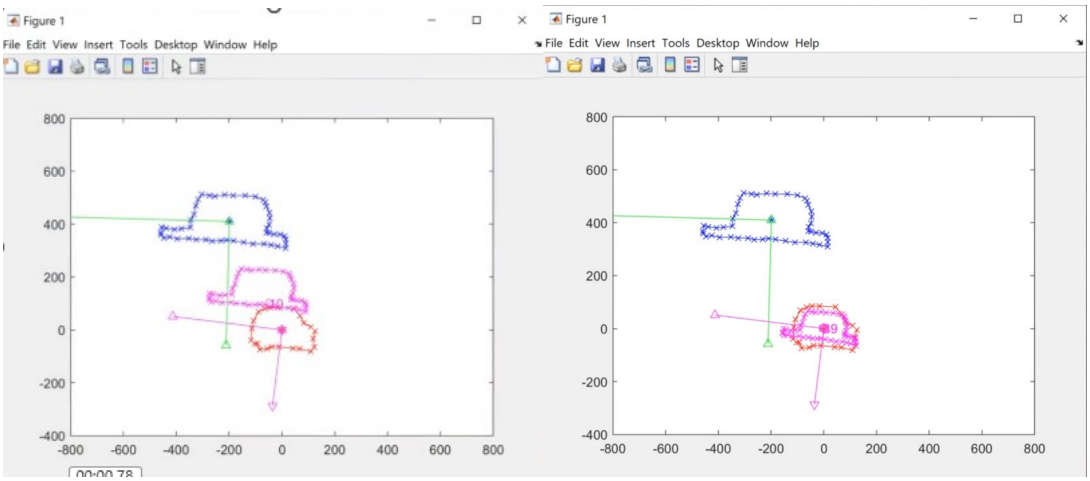

Figure 5.15 Self-calibration example 3 at time $t_{n/2}$ and t_n

Another example of mis-calibration is depicted in Figure 5.16, where the LiDAR point appears to be above and offset to the right of the Camera detection. Such misalignment can negatively impact the vehicle warning system's ability to accurately

sense the object's position. The processing steps involved in solving this issue are shown in Figure 5.17.

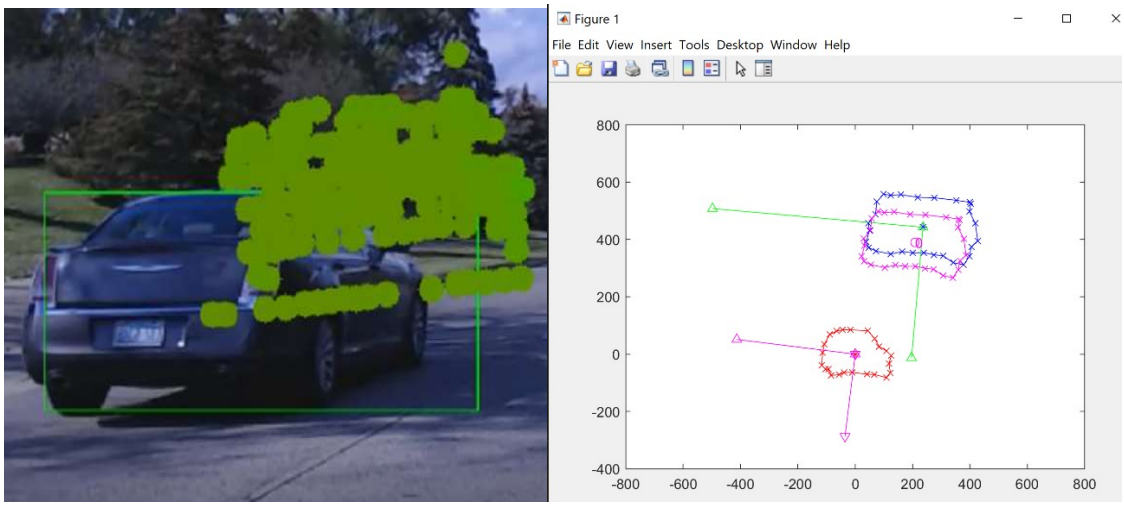

Figure 5.16 Self-calibration example 4 at time t_0

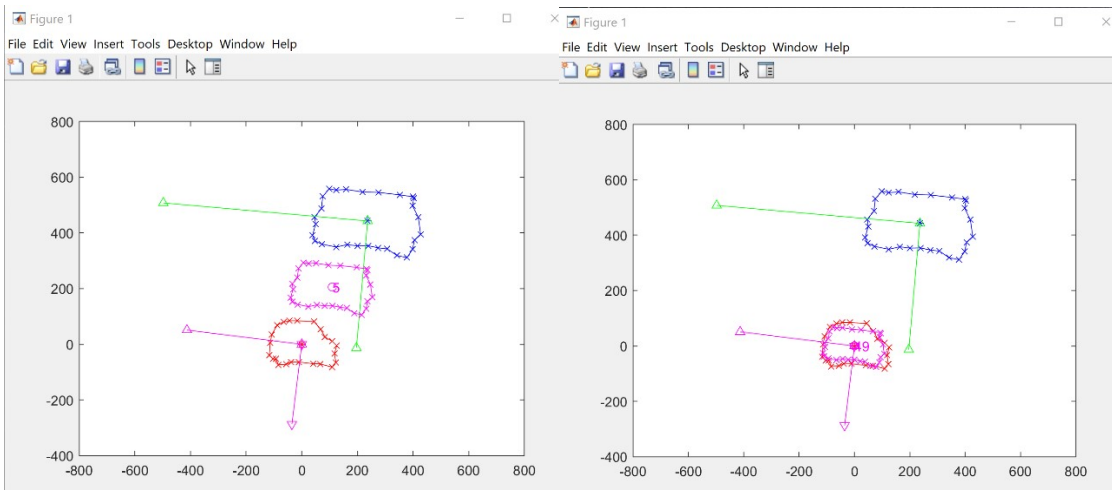

Figure 5.17 Self-calibration example 4 at time $t_{n/2}$ and t_n

To test the limitations of the designed LiDAR and Camera self-calibration algorithm, we deliberately introduced manual offsets and rotations to the LiDAR data and subsequently applied the algorithm. The results are visually represented in Figures 5.18

and 5.19. Upon examining the outcomes, it is evident that the algorithm is capable of accurately identifying the correct transformation matrix from LiDAR to Camera, even in the presence of these intentional distortions.

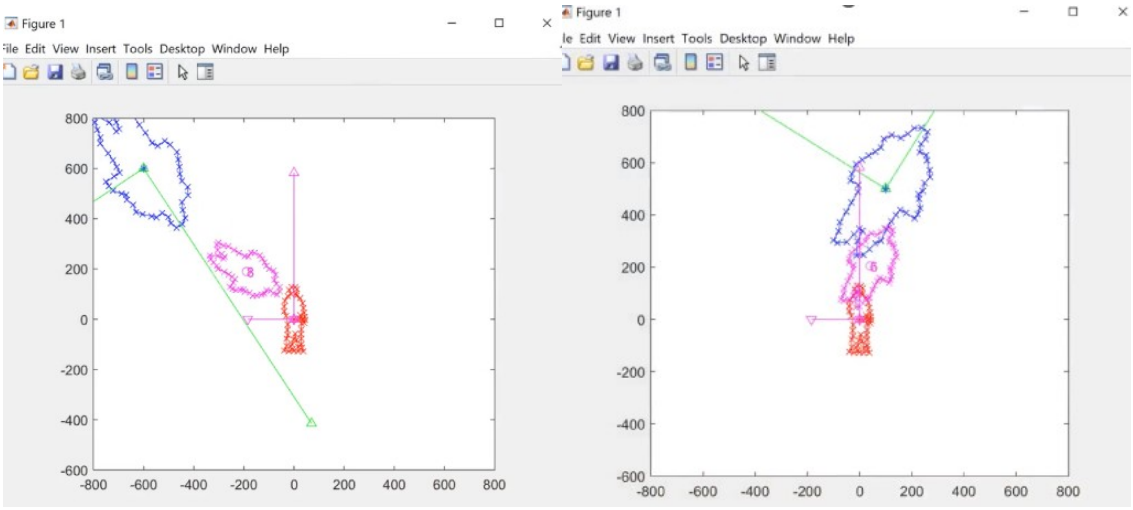

Figure 5.18 Self-calibration example 5 and 6 at time $t_{n/2}$

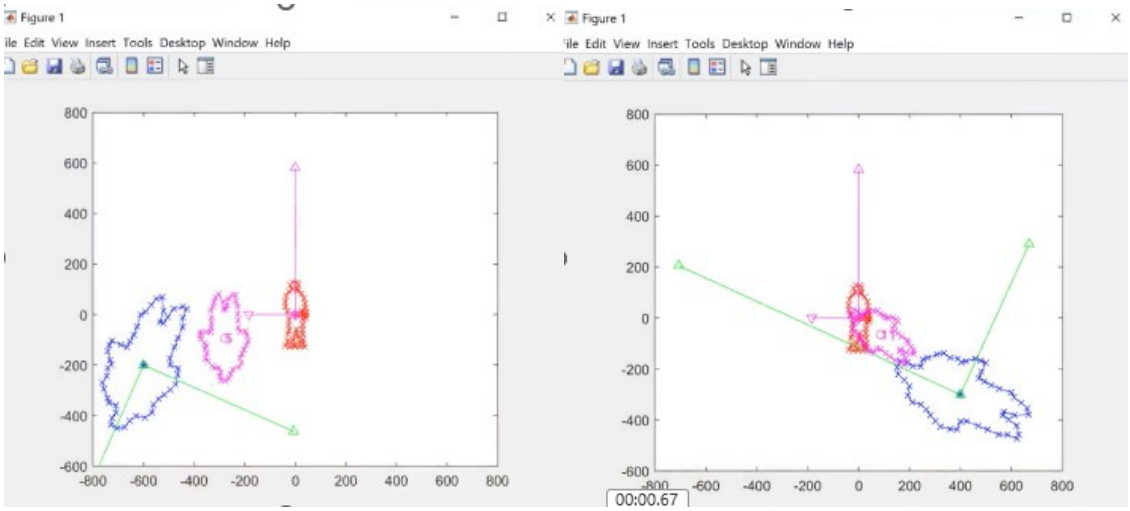

Figure 5.19 Self-calibration example 7 and 8 at time $t_{n/2}$

CHAPTER SIX

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This dissertation delves into the exploration and development of various cutting-edge technologies and methods for mobile robots and autonomous vehicles, encompassing robot kinematics, indoor and outdoor navigation, advanced sensors for 3D perception systems, multiple sensor fusion systems, 3D tracking algorithms, and self-calibration algorithms for LiDAR-Camera sensor fusion.

Chapter two of the dissertation provides a comprehensive overview of ground robot kinematics and the use of advanced machine learning techniques for image classification and point cloud processing. The chapter also proposes a drive-by-wire method for converting an electric vehicle into a self-driving vehicle suitable for any Ackermann steering vehicle like the Polaris Gem 2, involving retrofitting with an actuator control system. This cost-effective method enables a wide range of applications, such as transportation, logistics, and delivery services.

Chapter three of the dissertation introduces an autonomous vehicle platform and perception system that integrates LiDAR, RADAR, and Cameras. The chapter describes the use of a bounding box data type to represent object position, velocity, and size. Additionally, the chapter outlines the development of a single-sensor point cloud processing method and a multiple-sensor point cloud clustering algorithm that can adapt

to different types and quantities of sensors used in autonomous vehicle perception systems.

Chapter four of the dissertation presents a 3D object tracking algorithm for multiple sensors point cloud data. It includes extended Kalman filter model, unscented Kalman filter model, and particle filter tracking model, which are evaluated and compared for accuracy and reactive time based on highway driving scenario data recorded from an autonomous vehicle platform. The proposed tracking algorithm is designed to accommodate various distance sensors that can provide point cloud data, making it suitable for different types of autonomous vehicles.

Chapter five of the dissertation addresses the implementation of a LiDAR-Camera perception system for standard home-use vehicles. It discusses the challenges of calibrating LiDAR and Camera sensors due to potential mis-calibration leading to dangerous inaccuracies in object detection for autonomous vehicles. The chapter introduces a self-calibration algorithm for the LiDAR-Camera system, which utilizes 2D image CNN classification results and point cloud 2D projection data. The algorithm combines singular value decomposition and gradient descent algorithms to determine the correct transformation matrix for the sensors. The self-calibration algorithm is applicable for use with multiple LiDAR and Camera perception systems.

6.2 Future Work

Several future works could be explored related to LiDAR and camera sensor fusion in autonomous vehicles. One possible area of future work could be improving the accuracy of the LiDAR and camera projection algorithm. This could involve exploring more advanced techniques for aligning LiDAR and camera data, such as using point cloud registration methods or feature-based alignment approaches. One of the next steps is to apply the correct LiDAR and Camera matrix to the real-time system and assess the results through testing. Additionally, the calibration algorithm could be further refined to improve the accuracy of the extrinsic calibration between the LiDAR and camera sensors.

Another potential area for future work is developing more advanced object detection and tracking algorithms that make use of the fused LiDAR and camera data. For example, deep learning-based methods could be explored for detecting and tracking objects in 3D space using both LiDAR and camera data.

Finally, the LiDAR and camera fusion system could be integrated with other sensor modalities, such as radar or ultrasonic sensors, to improve the overall perception capabilities of the autonomous vehicle. This would require developing more advanced sensor fusion algorithms that can effectively integrate data from multiple sensor modalities to provide a more complete picture of the vehicle's surroundings.

.

REFERENCE

[1] Patle, B. K., Pandey, A., Parhi, D. R. K., & Jagadeesh, A. J. D. T. (2019). A review: On path planning strategies for navigation of mobile robot. Defence Technology, 15(4), 582-606.

[2] Pandey, A., Pandey, S., & Parhi, D. R. (2017). Mobile robot navigation and obstacle avoidance techniques: A review. Int Rob Auto J, 2(3), 00022.

[3] McGuire, K. N., de Croon, G. C., & Tuyls, K. (2019). A comparative study of bug algorithms for robot navigation. Robotics and Autonomous Systems, 121, 103261.

[4] Wang, L., Li, R., Sun, J., Liu, X., Zhao, L., Seah, H. S., ... & Tandianus, B. (2019). Multi-view fusion-based 3D object detection for robot indoor scene perception. Sensors, 19(19), 4092.

[5] Alatise, M. B., & Hancke, G. P. (2020). A review on challenges of autonomous mobile robot and sensor fusion methods. IEEE Access, 8, 39830-39846.

[6] Patel, N., Krishnamurthy, P., Fang, Y., & Khorrami, F. (2017, March). Reducing operator workload for indoor navigation of autonomous robots via multimodal sensor fusion. In Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction (pp. 253-254).

[7] Gibb, S., La, H. M., Le, T., Nguyen, L., Schmid, R., & Pham, H. (2018). Nondestructive evaluation sensor fusion with autonomous robotic system for civil infrastructure inspection. Journal of Field Robotics, 35(6), 988-1004.

[8] Mohamed, S. A., Haghbayan, M. H., Westerlund, T., Heikkonen, J., Tenhunen, H., & Plosila, J. (2019). A survey on odometry for autonomous navigation systems. IEEE access, 7, 97466-97486.

[9] El-kenawy, E.S.M., Khan, Z.S., Ibrahim, A., Aloyaydi, B.A., Ali, H.A. and Takieldeen, A.E., 2022. Metaheuristic Optimization for Mobile Robot Navigation Based on Path Planning. CMC-COMPUTERS MATERIALS & CONTINUA, 73(2), pp.2241-2255.

[10] Ranyal, E., Sadhu, A. and Jain, K., 2022. Road condition monitoring using smart sensing and artificial intelligence: A review. Sensors, 22(8), p.3044.

[11] Ma, Y., Wang, Z., Yang, H. and Yang, L., 2020. Artificial intelligence applications in the development of autonomous vehicles: A survey. IEEE/CAA Journal of Automatica Sinica, 7(2), pp.315-329.

[12] Bathla, G., Bhadane, K., Singh, R.K., Kumar, R., Aluvalu, R., Krishnamurthi, R., Kumar, A., Thakur, R.N. and Basheer, S., 2022. Autonomous vehicles and intelligent automation: Applications, challenges, and opportunities. Mobile Information Systems, 2022.

[13] Janeera, D.A., Gnanamalar, S.S.R., Ramya, K.C. and Kumar, A.A., 2021. Internet of things and artificial intelligence-enabled secure autonomous vehicles for smart cities. In Automotive Embedded Systems: Key Technologies, Innovations, and Applications (pp. 201-218). Cham: Springer International Publishing.

[14] Ni, J., Chen, Y., Chen, Y., Zhu, J., Ali, D. and Cao, W., 2020. A survey on theories and applications for self-driving cars based on deep learning methods. Applied Sciences, 10(8), p.2749.

[15] Tian, K., Kobayashi, K. and Cheok, K.C., 2020, January. Agile Development Process of a By-Wire Electric Vehicle for Intelligent Ground Vehicle Competition Self-Driving Challenges. In 2020 IEEE/SICE International Symposium on System Integration (SII) (pp. 726-730). IEEE.

[16] Gravina, R., Alinia, P., Ghasemzadeh, H. and Fortino, G., 2017. Multi-sensor fusion in body sensor networks: State-of-the-art and research challenges. Information Fusion, 35, pp.68-80.

[17] Badue, C., Guidolini, R., Carneiro, R.V., Azevedo, P., Cardoso, V.B., Forechi, A., Jesus, L., Berriel, R., Paixao, T.M., Mutz, F. and de Paula Veronese, L., 2021. Self-driving cars: A survey. Expert Systems with Applications, 165, p.113816.

[18] Cyganek, B. and Siebert, J.P., 2011. An introduction to 3D computer vision techniques and algorithms. John Wiley & Sons.

[19] Wang, Z., Zhang, L., Zhang, L., Li, R., Zheng, Y. and Zhu, Z., 2018. A deep neural network with spatial pooling (DNNSP) for 3-D point cloud classification. IEEE Transactions on Geoscience and Remote Sensing, 56(8), pp.4594-4604.

[20] Mattern, N., Schubert, R. and Wanielik, G., 2010, June. High-accurate vehicle localization using digital maps and coherency images. In 2010 IEEE intelligent vehicles symposium (pp. 462-469). IEEE.

[21] Li, L., Ota, K. and Dong, M., 2018. Humanlike driving: Empirical decision-making system for autonomous vehicles. IEEE Transactions on Vehicular Technology, 67(8), pp.6814-6823.

[22] Hang, P., Lv, C., Huang, C., Cai, J., Hu, Z. and Xing, Y., 2020. An integrated framework of decision making and motion planning for autonomous vehicles considering social behaviors. IEEE transactions on vehicular technology, 69(12), pp.14458-14469.

[23] Häne, C., Heng, L., Lee, G.H., Fraundorfer, F., Furgale, P., Sattler, T. and Pollefeys, M., 2017. 3D visual perception for self-driving cars using a multi-Camera system: Calibration, mapping, localization, and obstacle detection. Image and Vision Computing, 68, pp.14-27.

[24] Herman, S. and Ismail, K., 2017. Single Camera Object Detection for Self-Driving Vehicle: A Review. Journal of the Society of Automotive Engineers Malaysia, 1(3), pp.198-207.

[25] Chen, Z. and Huang, X., 2017, June. End-to-end learning for lane keeping of self-driving cars. In 2017 IEEE intelligent vehicles symposium (IV) (pp. 1856-1860). IEEE.

[26] Singh, R., Saluja, D. and Kumar, S., 2019, April. Power controlled adaptive range RADAR for self driving vehicles. In 2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring) (pp. 1-4). IEEE.

[27] Patole, S.M., Torlak, M., Wang, D. and Ali, M., 2017. Automotive RADARs: A review of signal processing techniques. IEEE Signal Processing Magazine, 34(2), pp.22-35.

[28] Singh, R., Saluja, D. and Kumar, S., 2020. Blind cancellation in RADAR-based self driving vehicles. IEEE transactions on vehicular technology, 69(7), pp.6977-6986.

[29] Zhao, J., Xu, H., Liu, H., Wu, J., Zheng, Y. and Wu, D., 2019. Detection and tracking of pedestrians and vehicles using roadside LiDAR sensors. Transportation research part C: emerging technologies, 100, pp.68-87.

[30] Tu, C., Takeuchi, E., Carballo, A. and Takeda, K., 2019, May. Point cloud compression for 3d LiDAR sensor using recurrent neural network with residual blocks. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 3274-3280). IEEE.

[31] Tu, C., Takeuchi, E., Carballo, A. and Takeda, K., 2019, May. Point cloud compression for 3d LiDAR sensor using recurrent neural network with residual blocks. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 3274-3280). IEEE.

[32] Matti, D., Ekenel, H.K. and Thiran, J.P., 2017, August. Combining LiDAR space clustering and convolutional neural networks for pedestrian detection. In 2017 14th IEEE international conference on advanced video and signal based surveillance (AVSS) (pp. 1-6). IEEE.

[33] Bates, J.S., Montzka, C., Schmidt, M. and Jonard, F., 2021. Estimating canopy density parameters time-series for winter wheat using UAS Mounted LiDAR. Remote Sensing, 13(4), p.710.

[34] https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[35] Sarbolandi, H., Lefloch, D. and Kolb, A., 2015. Kinect range sensing: Structured-light versus Time-of-Flight Kinect. Computer vision and image understanding, 139, pp.1-20.

[36] https://point.cloud/

[37] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324.

[38] Bai, L., Li, Y., Cen, M. and Hu, F., 2021. 3D instance segmentation and object detection framework based on the fusion of LIDAR remote sensing and optical image sensing. Remote Sensing, 13(16), p.3288.

[39] Derpanis, K.G., 2010. Overview of the RANSAC Algorithm. Image Rochester NY, 4(1), pp.2-3.

[40] Hou, W., Li, D., Xu, C., Zhang, H. and Li, T., 2018, December. An advanced k nearest neighbor classification algorithm based on KD-tree. In 2018 IEEE International Conference of Safety Produce Informatization (IICSPI) (pp. 902-905). IEEE.

[41] Murtagh, F. and Contreras, P., 2012. Algorithms for hierarchical clustering: an overview. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2(1), pp.86-97.

[42] Gamal, A., Wibisono, A., Wicaksono, S.B., Abyan, M.A., Hamid, N., Wisesa, H.A., Jatmiko, W. and Ardhianto, R., 2020. Automatic LIDAR building segmentation based on DGCNN and euclidean clustering. Journal of Big Data, 7, pp.1-18.

[43] Hartigan, J.A. and Wong, M.A., 1979. Algorithm AS 136: A k-means clustering algorithm. Journal of the royal statistical society. series c (applied statistics), 28(1), pp.100-108.

[44] Kriegel, H.P., Kröger, P., Sander, J. and Zimek, A., 2011. Density-based clustering. Wiley interdisciplinary reviews: data mining and knowledge discovery, 1(3), pp.231-240.

[45] Lerro, D. and Bar-Shalom, Y., 1993. Tracking with debiased consistent converted measurements versus EKF. IEEE transactions on aerospace and electronic systems, 29(3), pp.1015-1022.

[46] Strom, J., Jebara, T., Basu, S. and Pentland, A., 1999, September. Real time tracking and modeling of faces: An ekf-based analysis by synthesis approach. In Proceedings IEEE International Workshop on Modelling People. MPeople'99 (pp. 55-61). IEEE.

[47] Hu, C., Wang, Z., Taghavifar, H., Na, J., Qin, Y., Guo, J. and Wei, C., 2019. MME-EKF-based path-tracking control of autonomous vehicles considering input saturation. IEEE Transactions on Vehicular Technology, 68(6), pp.5246-5259.

[48] Farag, W., 2021. Real-time autonomous vehicle localization based on particle and unscented kalman filters. Journal of Control, Automation and Electrical Systems, 32(2), pp.309-325.

[49] Farag, W., 2021. Kalman-filter-based sensor fusion applied to road-objects detection and tracking for autonomous vehicles. Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering, 235(7), pp.1125-1138.

[50] Farag, W., 2022. Multiple road-objects detection and tracking for autonomous driving. Journal of Engineering Research, 10(1A), pp.237-262.

[51] Wen, T., Zhang, Y. and Freris, N.M., 2022, May. PF-MOT: Probability fusion based 3D multi-object tracking for autonomous vehicles. In 2022 International Conference on Robotics and Automation (ICRA) (pp. 700-706). IEEE.

[52] Vaskov, S., Quirynen, R., Menner, M. and Berntorp, K., 2022, June. Friction-adaptive stochastic predictive control for trajectory tracking of autonomous vehicles. In 2022 American Control Conference (ACC) (pp. 1970-1975). IEEE.

[53] Chen, G., Zhao, X., Gao, Z. and Hua, M., 2023. Dynamic drifting control for general path tracking of autonomous vehicles. IEEE Transactions on Intelligent Vehicles.

[54] Awad, N., Lasheen, A., Elnaggar, M. and Kamel, A., 2022. Model predictive control with fuzzy logic switching for path tracking of autonomous vehicles. ISA transactions, 129, pp.193-205.

[55] Ding, C., Ding, S., Wei, X. and Mei, K., 2022. Output feedback sliding mode control for path-tracking of autonomous agricultural vehicles. Nonlinear Dynamics, 110(3), pp.2429-2445.

[56] Wen, L.H. and Jo, K.H., 2022. Deep learning-based perception systems for autonomous driving: A comprehensive survey. Neurocomputing, 489, pp.255-270.

[57] Ghasemieh, A. and Kashef, R., 2022. 3D object detection for autonomous driving: Methods, models, sensors, data, and challenges. Transportation Engineering, 8, p.100115.

[58] Zhao, L., Wang, M. and Yue, Y., 2022. Sem-aug: Improving camera-lidar feature fusion with semantic augmentation for 3d vehicle detection. IEEE Robotics and Automation Letters, 7(4), pp.9358-9365.

[59] Alfred Daniel, J., Chandru Vignesh, C., Muthu, B.A., Senthil Kumar, R., Sivaparthipan, C.B. and Marin, C.E.M., 2023. Fully convolutional neural networks

for LIDAR–camera fusion for pedestrian detection in autonomous vehicle. Multimedia Tools and Applications, pp.1-24.

[60] Carranza-García, M., Galán-Sales, F.J., Luna-Romera, J.M. and Riquelme, J.C., 2022. Object detection using depth completion and camera-LiDAR fusion for autonomous driving. Integrated Computer-Aided Engineering, 29(3), pp.241-258.

[61] Anisha, A.M., Abdel-Aty, M., Abdelraouf, A., Islam, Z. and Zheng, O., 2023. Automated vehicle to vehicle conflict analysis at signalized intersections by camera and LiDAR sensor fusion. Transportation research record, 2677(5), pp.117-132.

[62] Zhang, K., Liu, Y., Mei, F., Jin, J. and Wang, Y., 2023. Boost Correlation Features with 3D-MiIoU-Based Camera-LiDAR Fusion for MODT in Autonomous Driving. Remote Sensing, 15(4), p.874.

[63] Singh, A., 2023. Transformer-based sensor fusion for autonomous driving: A survey. arXiv preprint arXiv:2302.11481.

[64] Xu, X., Dong, S., Xu, T., Ding, L., Wang, J., Jiang, P., Song, L. and Li, J., 2023. Fusionrcnn: Lidar-camera fusion for two-stage 3d object detection. Remote Sensing, 15(7), p.1839.

[65] Hasanujjaman, M., Chowdhury, M.Z. and Jang, Y.M., 2023. Sensor fusion in autonomous vehicle with traffic surveillance camera system: detection, localization, and AI networking. Sensors, 23(6), p.3335.