ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

Zeynep Yasemin ERDOĞAN

DESIGNING AN OBSTACLE DETECTION AND AVOIDANCE SYSTEM USING A 2D LIDAR AND A STEREO CAMERA FOR AUTONOMOUS VEHICLES

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ADANA-2022

ABSTRACT

MSc THESIS

DESIGNING AN OBSTACLE DETECTION AND AVOIDANCE SYSTEM USING A 2D LIDAR AND A STEREO CAMERA FOR AUTONOMOUS VEHICLES

Zeynep Yasemin ERDOĞAN

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Supervisor : Prof. Dr. Ulus ÇEVİK
Year: 2022, Pages: 37
Jury : Prof. Dr. Ulus ÇEVİK
: Assoc. Prof. Dr. Murat AKSOY
: Prof. Dr. Turgay İBRİKÇİ

In this study, some software has been developed by applying processing, and deep learning methods to detect obstacles, to map the environment of the autonomous car and to plan of car's path with the Nvidia Jetson TX2, Lidar, and ZED Camera. The operating system of the tool is Linux-based operating system (Ubuntu). CUDA, VisionWorks, and OpenCV software will be installed on this operating system by installing NVIDIA's JetPack software package. Drivers will be installed for LIDAR and camera to work. Robot Operating System (ROS) for simulation and control of mobile robots and ZED SDK for ZED stereo camera software support will be uploaded.

In this study, Hector Slam Mapping and 3 different Path Planning Algorithms were used for autonomous vehicle design with Lidar and stereo camera. By applying mapping in Rviz, it was observed that map information was taken from Lidar instantly, and the next location could be determined by using route estimation. Finally, testing and simulation was performed between the start and end points with the Gazebo simulation tool, and it was observed that 3 different algorithms performed the road planning on a different route and the results were compared.

Keywords: Object Detection, Autonomous Vehicle, Path Planning, Motion Control, Deep Learning

ÖZ

YÜKSEK LİSANS TEZİ

OTONOM ARAÇLAR İÇİN 2D LIDAR VE STEREO KAMERA KULLANARAK ENGEL TESPİTİ VE KAÇINMA SİSTEMİ TASARIMI

Zeynep Yasemin ERDOĞAN

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK - ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Prof. Dr. Ulus ÇEVİK
Yıl: 2022, Sayfa: 37
Jüri : Prof. Dr. Ulus ÇEVİK
: Doç. Dr. Murat AKSOY
: Prof. Dr. Turgay İBRİKÇİ

Bu tezde, NVIDIA Jetson X2 , LİDAR, ve ZED Kamera ile engelleri tespit etmek, otonom aracın çevresini haritalamak ve yol planlaması için derin öğrenme yöntemleri uygulanarak bazı yazılımlar geliştirilmiştir. Aracın işletim sistemi Linux tabanlı işletim sistemidir (Ubuntu). NVIDIA'nın JetPack yazılım paketi yüklenerek bu işletim sistemine CUDA, VisionWorks ve OpenCV yazılımları yüklenecektir. LIDAR ve kameranın çalışması için sürücüler yüklenecektir. Mobil robotların simülasyonu ve kontrolü için Robot İşletim Sistemi (ROS) ve ZED stereo kamera yazılım desteği için ZED SDK yüklenecektir.

Bu çalışmada, Lidar ve stereo kamera ile otonom araç tasarımı için Hector Slam Haritalama ve 3 farklı Yol Planlama Algoritması kullanılmıştır. Rviz'de haritalama uygulaması yapılarak anlık olarak Lidar'dan harita bilgelrinin çekildiği gözlenlemiştir ve yol tahmini kullanılarak bir sonraki konum belirlenebilmiştir. Son olarak Gazebo simülasyon aracı ile başlangıç ve bitiş noktaları arasında test ve simülasyon yapılarak, 3 farklı algoritmanın yol planlamasını farklı bir güzergahta gerçekleştirdiği gözlemlenmiştir ve sonuçları karşılaştırılmıştır.

Anahtar Kelimeler: Nesne Algılama, Otonom Araç, Yol Planlama, Hareket Kontrolü , Derin Öğrenme

EXTENDED ABSTRACT

In general, with the development of the technology, autonomous car works have been popular and widespread in the automotive industry in recent years. With the accelerating and developing artificial intelligence studies, many algorithms and hardware parts have been developed for autonomous vehicle studies, and this development continues today. With the acceleration of these studies, autonomous car designs have continuous development. These desings aim to create a vehicle that makes the road between two points without human intervention and to make improvements to this vehicle design.

For the autonomous vehicle system to work properly, objects, namely obstacles, must be detected and subsequently identified to control the movement, and therefore, recently, object recognition studies and applications for autonomous vehicles have been increasing rapidly. Examples of this application are vehicle detection systems such as lane detection assistant, obstacle detection using stereo image, and pedestrian detection warning system using infrared image.

In autonomous vehicles, simultaneous road and obstacle situations are designed via communication systems and sensors, and the safe movement of the vehicle is ensured with various techniques. Events such as steering, braking, and acceleration are carried out as a result of the ultrasonic sensors on the wheels of the vehicle detecting the locations of the vehicles that are braking or parked, and the analysis of the data received from many such sensors via a central computer system. In addition, object detection and recognition are one of the most important points in autonomous vehicle systems. Thanks to the use of technologies and methods such as LIDAR (Light/Laser Detection and Ranging), GPS (Global Positioning System), odometry, radar (Radio Detection and Ranging), the vehicle can detect objects around it, and thus the vehicle is guided correctly.

The most important research topics of autonomous driving applications are; mapping and localization of the environment, object detection, and recognition. The reason for this is that for the control mechanisms to work in autonomous driving, the vehicle must first be positioned in the environment and mapped the environment, and also the detection of objects and then the identification those objects. is necessity. Recently, object recognition applications for real-life vehicles have grown rapidly and many mapping, path planning, and control designs have been developed. When the literature is searched, it has been seen that the operating speeds and the accuracy of the responses are what distinguish these studies from each other, and the speed and accuracy percentages are a very important issue for an autonomous system, as predicted.

In this thesis, some software has been developed by applying processing, and deep learning methods to detect obstacles and map the environment of the autonomous car with the Nvidia Jetson TX2, Lidar, and ZED Camera. Autonomous navigation of vehicles in the environment requires some subsystems that include building a map of the environment, localizing the vehicle in this map, and motion planning. All of these tasks need different algorithms and packages. Hector Slam Mapping, Path Planning, and Obstacle Detection Algorithm are algorithms that are used for autonomous vehicle design. LIDAR sensor, stereo camera, and NVIDIA TX2 card will be placed on the vehicle for this system.

The NVIDIA Jetson TX2 card is used for artificial intelligence and image processing applications with low power consumption and high computational performance, thus providing speed and performance level in work. NVIDIA Jetson provides high-speed computing and power efficiency and is a very popular and useful module used in artificial intelligence applications. It includes libraries for deep learning, computer vision, GPU computing, multimedia processing, and more, and is powered by the NVIDIA Jetpack SDK.

Thanks to the LIDAR sensor, a 2-dimensional map of the environment was created. The stereo camera, on the other hand, is a 3D camera used for depth sensing and motion tracking, and this camera predicts where pedestrians are after a few seconds based on their location and speed, preventing collisions with other vehicles. Also, some software support is required to use these hardware parts.

The RPLIDAR A2 consists of a series of scanner cores and parts that provide mechanical power that allows the core to rotate rapidly, and this rotation allows sensor data to be acquired very quickly. Also during operation the scanner will rotate and scan clockwise. For this study, Lidar read a 180-degree data.

The operating system of the tool is a Linux-based operating system (Ubuntu). NVIDIA's JetPack software package was installed and CUDA, VisionWorks, and OpenCV software were installed on this operating system. Installed ZED SDK for Robot Operating System (ROS) and ZED stereo camera software support for simulation and control of mobile robots. To obtain the results, the paths taken by the Ackermann simulation vehicle according to 3 different algorithms on the track designed in the Gazebo simulation environment were observed and compared.

By using algorithms on the data obtained from the lidar, it is ensured that the lidar detects the environment and objects in which it is located, and in this way, the shortest and safest way to reach the next location is automatically determined thanks to the data obtained from the environment, the path planning algorithm and the Rviz simulator.

It was aimed to determine the path planning of the robot using A*, BFS and Dijkstra algorithms, and to successfully go to the target points despite the obstacles with the Ackerman vehicle on Gazebo simulator. Also, the fact that these algorithms have some improvements thanks to the studies developed and accelerated in recent years is the reason for using these algorithms.

The A* algorithm aims to calculate the shortest path intuitively and with clear distance data and examines the alternative paths that can be taken from the starting node to the destination node.

The basis of Dijkstra's algorithm is to find the shortest path on a graph. This algorithm aims to calculate the shortest path from one node of a given shape to all other nodes. The idea of the algorithm is to continuously calculate the shortest distance starting from a starting point and exclude longer distances when updating.

In the BFS algorithm, a starting node is determined and after all neighbors are visited, all neighbors of the visited neighbor that can be visited at the same time are added to the queue at the same time. In addition, an element is removed from the queue at each step, and since it works with the FIFO principle, the first element comes out first.

The Gazebo simulation tool of ROS and the Ackermann vehicle were simulated according to three different algorithms on the track with the created world. Many environments (worlds), robots and objects can be accessed from Gazebo's database, and these widely available robots can work with ROS.

To summarize the general work of this study, algorithms have been developed and tested by focusing on edge detection, object detection, instant mapping-positioning, and road planning in a disabled environment, which was needed in an autonomous vehicle system through more than one algorithm, using the Linux operating system, ROS.

In the simulation results, it is clearly seen that the route of the BFS algorithm differs for the two scenarios with and without barriers. It has also been observed that the simulation result of BFS is different from the scenario where other algorithms are on the obstacle course. In addition, it has been observed that Dijkstra and BFS algorithms, which follow the same path, reach the target in a shorter time. This may indicate that Dijkstra and A* are two algorithms that are better in performance than

BFS. Although Dijkstra's algorithm goes from origin to destination in the same way as BFS, it has been shown to outperform the BFS algorithm.

There are many autonomous system developments from past to present. Despite this situation, when the literature is reviewed, it is seen that many algorithms and decision mechanism development problems and costly hardware reasons should still continue and the performance will increase only by increasing the cost. However, since this self-driving capability of vehicles and robots is thought to make human life easier and safer, studies on autonomous vehicle systems continue.

ACKNOWLEDGMENTS

I would like to thank my esteemed advisor Prof. Dr. Ulus ÇEVİK, who was with me in this thesis and my previous undergraduate education, for his support in every subject and for whom I am honored to work, for his trust, as well as the interest, tolerance, and patience he has shown me.

I sincerely thank my family for their constant support, understanding, and being there for me, and also thank my sister Hatice Kübra ERDOĞAN for her support and for not letting me get motivated.

I also want to thank my friend Elanur EKİCİ, and Birand ERDOĞAN, for moral support, for believing in me, and for always supporting me.

TABLE OF CONTENT **PAGE**

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

LIST OF SYMBOLS & ABBREVIATIONS

£ : Initial estimate of the pose
AEB : Autonomous Emergency Braking
AI : Artificial Intelligent
BFS : Breadth First Search
CPU : Central Processing Unit
f(a) : Evaluated Cell Value
g(a) : Length of the Path (from initial to goal point)
GB : Gigabyte
GPS : Global Positioning System
GPU : Graphics Processing Unit
h(a) : Heuristic Distance of the cell
LIDAR : Light Detection and Ranging
MT : Meter
ROS : Robot Operating System
Sec : Second
SLAM : Simultaneous Localization and Mapping
UI : User Interface
x : Location point in x axis
y : Location point in y axis

1. INTRODUCTION

1.1. Overview

During the literature review for this project, various articles and conference proceedings from the fields of electrical electronics, computer, and automotive engineering were examined. As a result of the studies examined, it is seen that autonomous vehicle studies and autonomous system developments have increased in the automotive industry in recent years with the developing technology. In autonomous vehicles, the existing road and traffic conditions are modeled through communication systems and sensors, and the movement of the vehicle is ensured by various methods.

Autonomous vehicles that make the journey from one point to another without the intervention of the driver (human) have many advantages such as safety, comfort, and less energy use (Bautista, et al.. 2016). The automotive engineers' community evaluates vehicle automation at 6 levels, with the first level 0 being completely driver-controlled vehicles, and level 5 being fully computer-controlled autonomous vehicles (SAE, 2016.). The operating cycle of driverless vehicles begins with the receipt of information from internal and external sensors. While internal sensors such as yaw rate and wheel speed sensor provide information about the interior perception of the vehicle, external sensors (such as radar, LIDAR, and GPS) transmit the external perception of the vehicle to the decision mechanism (Matzka, et al. 2009) While internal sensors determine the vehicle's orientation (sway, yaw, etc.), speed and acceleration, external sensors determine its position relative to the external environment. The raw data from the sensors is interpreted in the detection part and instant information about the location of the vehicle, road, and obstacles is created (Aufrère, et al. 2003) (Lozano-Perez, et al. 1984). In addition, the vehicle is in constant communication with other vehicles, existing infrastructure, and other traffic components (Nagel, et al. 2007.).

Object detection and recognition are very important for the correct functioning of the autonomous vehicle system because to control the movement, objects, namely obstacles, must be detected and then identified. For this reason, recently, object recognition studies and applications for autonomous vehicles have been increasing rapidly. Example of this application is lane detection assistant, obstacle detection using the stereo image, pedestrian detection warning system using infrared image (Coelingh, et al. 2010), and vehicle detection systems using a combination of laser radar and single-lens camera system. In addition, for autonomous vehicles, a large number of obstacle recognition technologies using LIDAR, radar, ultrasonic sensors, and cameras for Autonomous Emergency Braking (AEB) have been developed and implemented recently (Jang, 2016).

When the literature is examined, different algorithms are used according to the possible static and dynamic conditions of the environment in path planning problems in the disabled environment. The most popular is the A*, Dijkstra, Rapidly-, and exploring random tree (RRT) algorithms. When the literature is scanned, it has been observed that the A* algorithm has been used in recent years (Patle, et al. 2019) (Duchon, et al. 2014).

1.1.1. Purpose of Thesis

This study aims to obtain a safe ride system design when hardware equipment is attached to the vehicle. In this study, more than one algorithm and operation were chosen as a method. The main software of this project, ROS, was used to control the robotic component from a laptop and to run this software, a Linux operating system is required on the Ubuntu desktop. A LiDAR sensor is added to the system, which will operate in real vehicle environments, and computational operations are performed through the Robotic Operating System (ROS). The ROS system consists of several independent nodes, each of which communicates with other nodes using a broadcast communication model, and has some tools and

libraries aimed at simplifying the task of creating robust robot behavior in robotics studies.

By using the slam algorithm through the data obtained from the lidar, positioning and mapping were made in the environment where the lidar was located. Afterward, the shortest and safest path was automatically determined to reach the next position, thanks to the data obtained from the environment and the path planning algorithm, and the Rviz simulator. Finally, the first and last point of the A* algorithm, which was created with imaginary obstacles in the gazebo simulation environment, was determined automatically and the road planning algorithm was tested.

Our aim in the study is to produce algorithms that can provide obstacle detection and path planning by using efficient hardware parts commonly used in autonomous vehicle systems such as NVIDIA JETSON TX2 developer kit, RPLIDAR, and ZED camera.

1.2. Autonomous Vehicle Systems

In recent years, significant work in machine learning techniques such as deep neural networks (DNN) has led to the development of safety-critical machine learning systems such as autonomous vehicles. Many major automakers such as Tesla, Ford, and BMW produce and test these cars. Looking at recent studies and their results, autonomous vehicles have become very efficient in practice and can safely travel millions of miles without driver intervention (Abadi, et al. 2016) (Abdessalem, et al. 2016) (Goodfellow, 2016) (Anand, et al. 2013) (Anderson, 2017) (Bastani, et al. 2016) (Bengio, et al. 2017) (Bojarski, et al. 2016) (Carlini, and Wagner 2017) (Chen, et al. 1998) (Cisse, et. al. 2017.)

Autonomous vehicles need mapping and location information in an unknown environment to detect the direction and position of the vehıcle, and different slam methods have been used in many studies to determine this issue. The

first of these studies was initiated by Smith and Cheesman in 1987 and has continued to the present day (Smith, and Cheesman 1987) (Durrant-Whyte, and Bailey 2006) (Vidal-Calleja, et al.. 2004) (Wang, et al. 2007).

2. MATERIALS AND METHODS

2.1. Introduction

In this section, all stages and results of the thesis, from equipment and algorithms to methods, are examined separately.

First of all, the materials used and their properties were examined. Then the methods were examined.

In the first instance, the grayscale, blurred image of the image with edge detection algorithm and finally the edge detection image was examined. Then, simulations of mapping and path planning algorithms are studied. It has been observed that the mapping algorithm reacts instantly as a result of moving the Lidar while mapping and the simulation result obtained with 2d position estimation in the simulation interface used is also examined. Finally, the three path planning algorithms were tested in an environment where obstacles were created according to different scenarios and the results were examined and compared.

2.2. Materials

2.2.1. NVIDIA Jetson TX2

NVIDIA Jetson is a very popular and useful module used in artificial intelligence applications to provide high-speed computing and power efficiency. It is powered by the complete NVIDIA Jetpack SDK, which includes libraries for deep learning, computer vision, GPU computing, multimedia processing, and much more.

This 7.5-watt supercomputer in a single module takes real artificial intelligence computing to the extreme and is therefore effectively used in artificial intelligence applications. Due to the speed of this computer and its efficiency in autonomous systems, this card was used in this study. The design and all features of this module can be seen in Table 2.1 and Table 2.2, respectively.

This module quickly evaluates the instantaneous data from the cameras and sensors in autonomous vehicles, allowing the system to decide for the vehicle's movement planning.

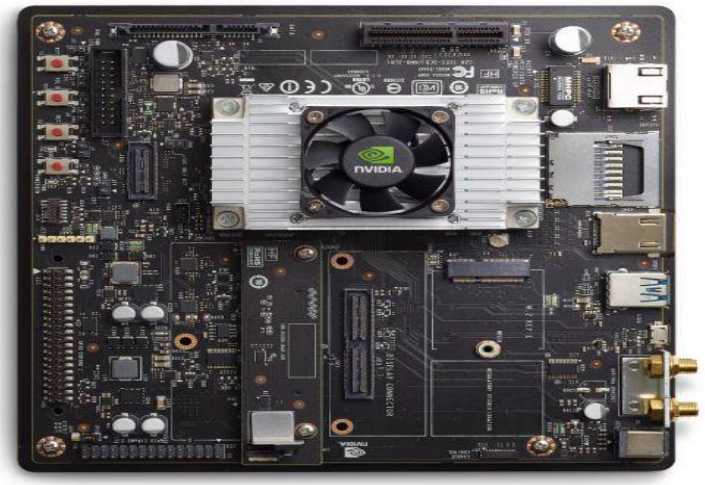

Figure 2.1. NVIDIA Jetson TX2 Developer Kit.

Table 2.1. General Features of NVIDIA Jetson TX2 Module.

GPU	Pascal
CPU	64-bit Denver 2 and A57 CPUs
Storage	32 GB
Memory	8 GB

2.2.2. LIDAR (Light Detection and Ranging)

Lidar is a popular part used in autonomous vehicle systems and can integrate with the NVIDIA Jetson TX2 module to perform many operations such as object detection, mapping, route planning, and navigation. And with this study, it has been seen that mapping path planning can be done instantly and efficiently with a lidar sensor (with NVIDIA card) without the need for any other sensor or device.

The LIDAR used has a 360-degree scanning feature and can take 8000 samples per second, which is a feature that explains why this sensor was chosen in this study. Although the normal scanning frequency is 10 Hz (600rpm), the actual scanning frequency can be adjusted in the range of 5-15 Hz according to users' requirements. All the characteristics of the sensor can be seen in Table 2.2.

The RPLIDAR A2 consists of a series of scanner cores and parts that provide mechanical strength that allows the core to spin rapidly. Thanks to this rapid rotation, sensor data is received very quickly. During operation, the scanner will rotate and scan clockwise.

Table 2.2. RPLidar A2 M8 General Features.

Laser Scanner Range	360°
Sample Frequency	2000-8000 Hz
Scan Rate	5-15 Hz
Distance Range	0.15-12 m
Angular Resolution	0.45-1.35°

2.2.3. ZED Camera

ZED Camera is a stereo camera produced for autonomous navigation and augmented reality applications, imitating human visual perception and solving problems such as spatial tracking, and obtaining 3D images of the desired quality in the open area.

The detection angle is 110 degrees thanks to its wide-angle lenses, which can capture 15 images per second while shooting a 2.2K video. In addition, the detection distance is between 0.5 meters and 20 meters. It has been a preferred stereo camera in this study because of its fast detection and distance features.

ZED Cameras are used in drone, unmanned aerial vehicles, car, or robot applications, and in these applications, it gives the devices a 3D image and depth perception.

2.3. Methods

2.3.1. Canny Edge Detection

The Canny edge detector is an edge detection operator that uses a progressive algorithm to detect a wide variety of edges in images and was developed in 1986 by John F. Canny as a result of long studies and research (Canny, 1986)

The purpose of choosing it in this study is that it is efficient and popular among edge detection algorithms, and a result is obtained with the zed camera.

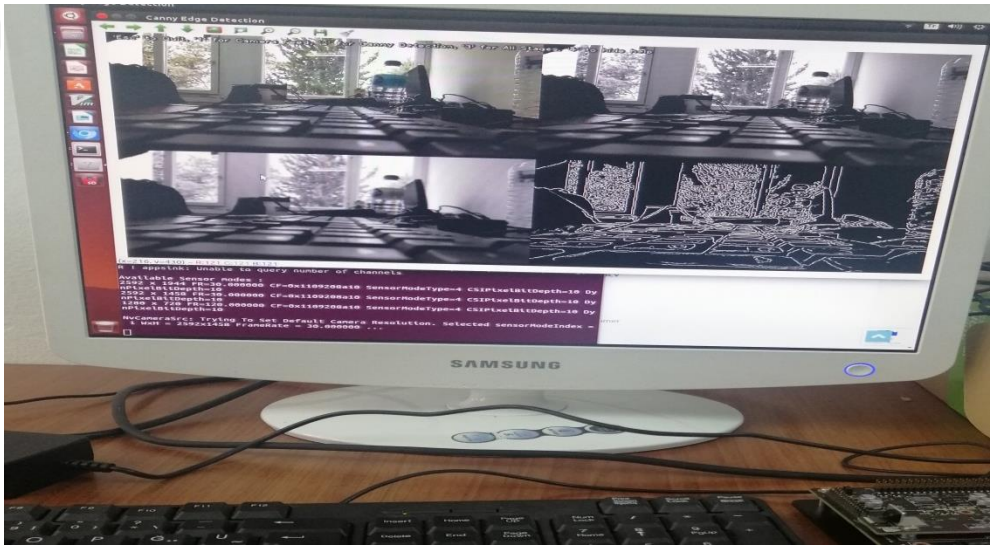

Figure 2.2. Canny Edge Detection Application.

The main part of filter processing reads data from the camera, converts it to grayscale, runs a gaussian blur on the grayscale image, and then applies the Canny Edge Detector on it. In Figure 2.2, the 1st frame is the normal image taken from the camera, the second frame is grayscale, the 3rd frame is a grayscale image with gaussian blur, and finally, the edge detection was made with a canny edge detector in the 4th frame.

Using RGB input alone when detecting lane lines is not enough to reduce the complexity of dimensions, so the camera image is converted to grayscale. To

match the detected stripes in real-time video, we have now divided the input image into two parts, one is grayscale and is called the near part, and the other is RGB, which is called the far part. To achieve maximum accuracy, the noise has to be dropped in the grayscale image by using a Gaussian filter. And in the final, the edge detection form of the original picture is obtained.

With this edge detection algorithm, lane lines can be detected in anautonomous vehicle system.

2.3.2. Hector Slam Mapping

Various studies have been actively carried out by other researchers for the application of SLAM. This study aims to present an effective and fast experiment result in which a Simultaneous Location and Mapping (SLAM) algorithm based on a LiDAR sensor is performed in terms of map creation and positioning capability. The summary flow chart of the general operation of the system is shown in Figure 2.4.

Various types of 2D SLAM algorithms are available, such as HectorSLAM, G mapping, KartoSLAM, CoreSLAM, and LagoSLAM. After various evaluations, the three most suitable SLAM algorithms were examined in detail in the study: GMapping algorithm, Hector-SLAM algorithm, and Cartographer algorithm, and it was observed that the GMapping algorithm was based on the particle filter matching algorithm, Hector-SLAM scanning matching algorithm(Zhang, et al., 2020).

In this study, the hector slam algorithm was used and it was seen that the mapping of the environment where the lidar was located was adjusted instantaneously depending on the movement. Hector SLAM was explained with some mathematical formulas by using the Taylor series (Eliwa et al., 2017).

This algorithm determines the robot's position based on scan matching and does not use the wheel odometer common in other SLAM algorithms. Moreover, using the Gaussian Newton minimization method over other algorithms has the

advantage of not needing to calculate the second derivatives. The optimal estimation is calculated with the optimal matching of the laser data and the map in the sense that the optimal £* below is solved (Zhang et al., 2020):

$$£^* = \sum_{i=1}^{N}[1 - M(Si(£))]^2.$$

Where,

$M(S_i(£))$: the value of map at $(S_i(£))$,

$S_i(£)$: world coordinate of scan endpoints $si = (s_{i,x}, s_{i,y})^T$, which fits the following function;

$$S_i(£) = \begin{bmatrix}\cos a & -\sin a \\ \sin a & \cos a\end{bmatrix} \times \begin{bmatrix}s_{i,x} \\ s_{i,y}\end{bmatrix} + \begin{bmatrix}p_x \\ p_y\end{bmatrix}.$$

Here, initial estimate of the pose £ is given an updated estimation (£+Δ£) is evaluated with aproximation $M(S_i(£+Δ£))$ by using Taylor expansion (first order) and the result is :

$$Δ£ = H^{-1}\sum_{i=1}^{N}[\nabla M(S_i\ (£))]^{\,T} \times [(\frac{\partial S_i\ (£)}{\partial\ £})]^T\,[1\text{-}M(S_i(£))]\ ,$$

when H: Hessian matrix and given with this formula;

$$H = [\nabla M(Si(£))\ (\frac{\partial S_i\ (£)}{\partial\ £})]^T\ [\nabla M\big(S_i(£)\big) \times \frac{\partial Si(£)}{\partial £}].$$

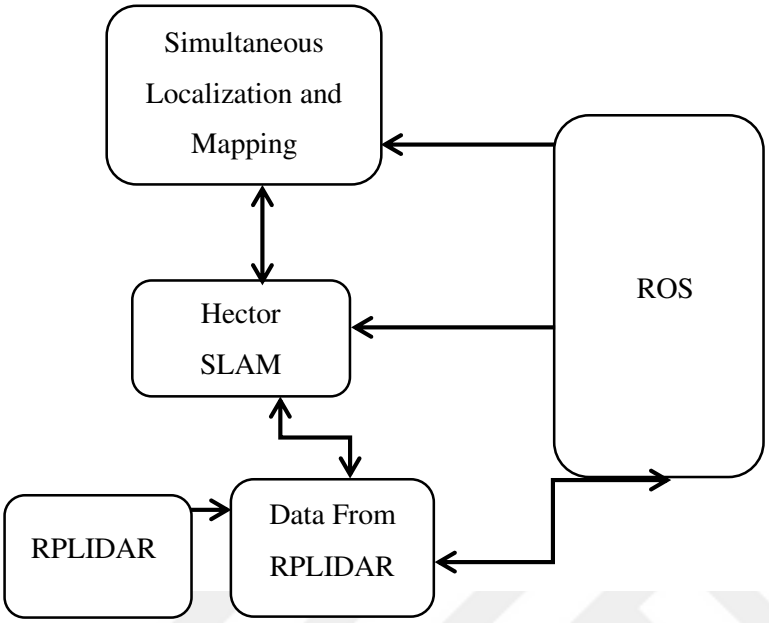

Figure 2.3. Block Diagram of Localization and Mapping of the System.

Hector algorithm uses the Gaussian Newton minimization method and second derivatives needn't be computed in this algorithm. The hector_mapping node is used to learn a map of the environment and simultaneously estimate the 2D exposure of the platform at the laser scanner frame rate.

As a result of the data coming from the lidar sensor in Rviz, the simulation results of the point and mapped environment according to the obstacles in the environment of the Lidar sensor are shown in figure 2.5 and figure 2.6, respectively. Before applying the mapping algorithm during SLAM studies, the point cloud of the environment is extracted with Lidar, and the working area is expressed in 2 dimensions with this point cloud which can be seen in Figure 2.5.

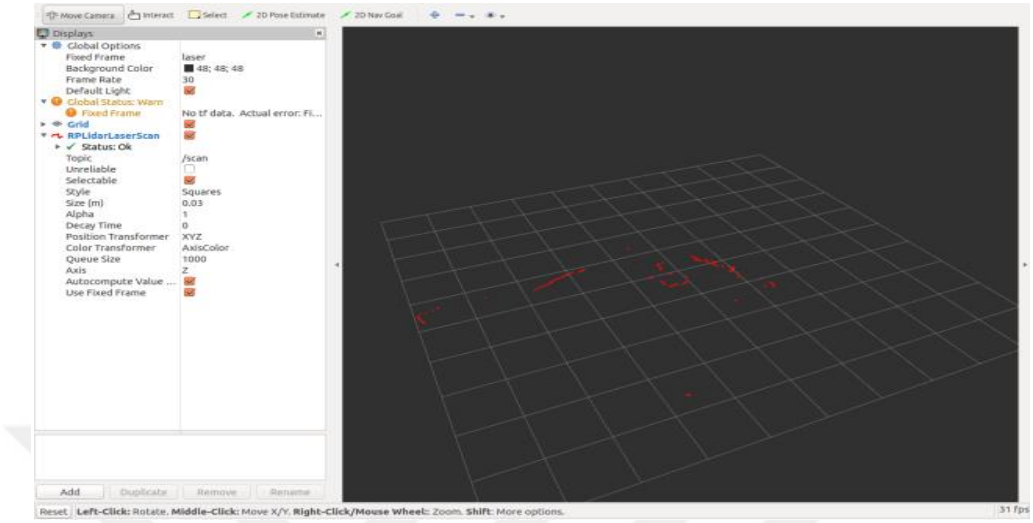

Figure 2.4. Rviz Simulation of Lidar Environment without Mapping.

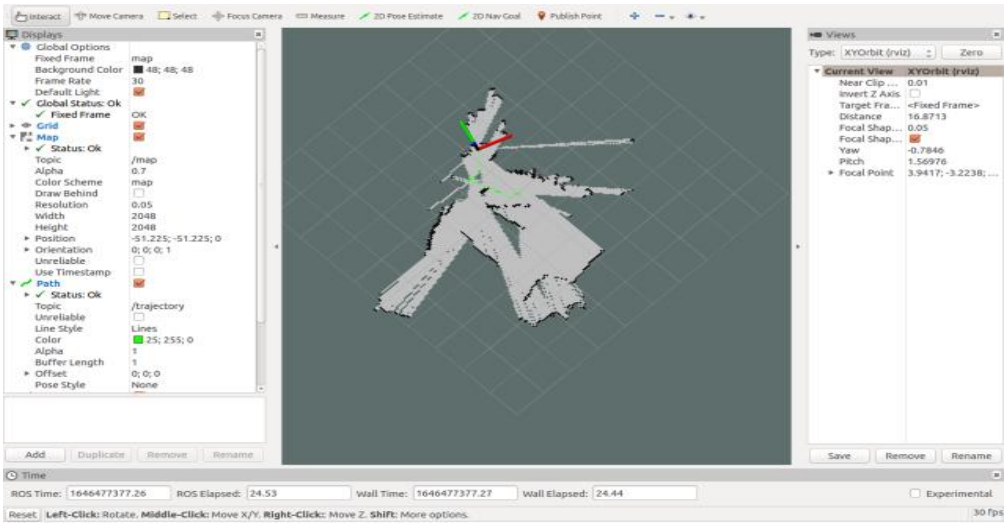

Figure 2.5. Rviz Simulation of Lidar Environment with Hector SLAM Mapping.

Furthermore, LIDAR sensor data parameters which are angle_min-max, scan_time, range_min and, max parameters of the sensor can be seen in Figure 2.7.

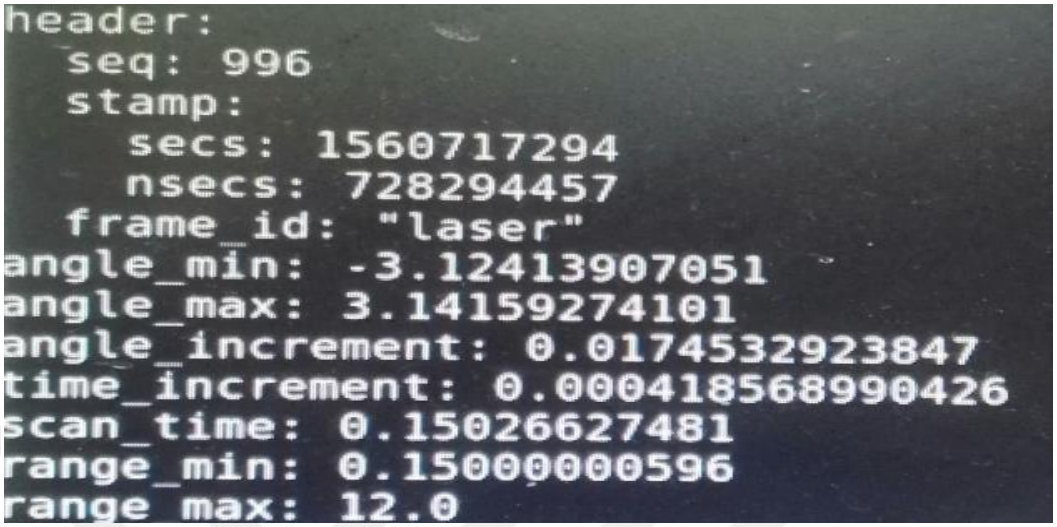

Figure 2.6. Parameters Taken From LIDAR Sensor.

Also, in Figure 2.8, the simulation result of a lidar sensor advanced with the next position estimation is shown. The shortest distance and position estimation is made with the map information of the environment and a manually selected point.

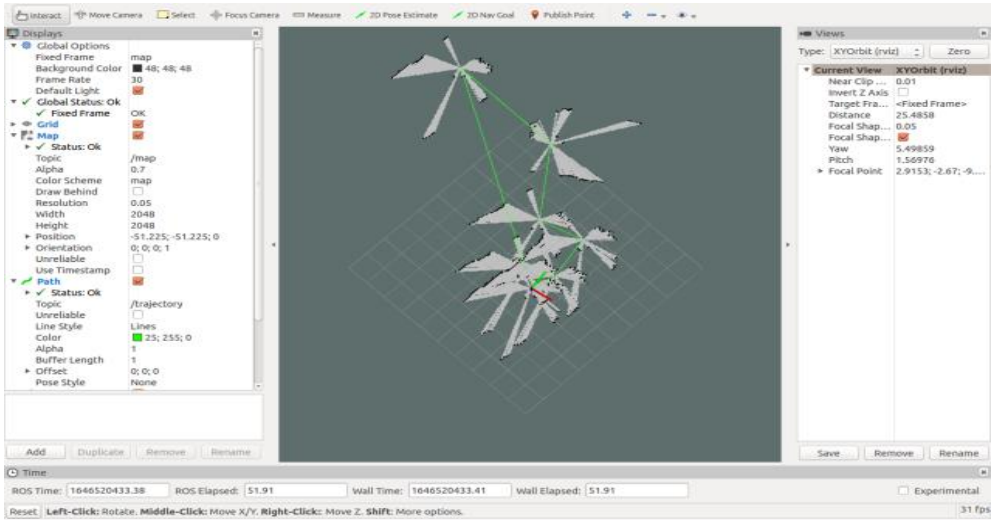
Figure 2.7. 2D Pose Estimation with Hector Slam in Rviz.

Before running these codes all algorithm files were added to the catkin workspace/src file and the roscore operation was started to communicate ROS nodes and RpLidar data.

To connect Rplidar with ROS in Rviz:

$ roslaunch rplidar_ros rplidar.launch

To start the built map in Rviz:

$ roslaunch hector_slam_launch tutorial.launch

2.3.3. Path Planning Algorithms

2.3.3.1. A* Algorithm

In this study, while applying the A* path planning algorithm, the Ackermann vehicle was used in the Gazebo simulation environment. The reason for choosing the A* algorithm is that it has an efficient and improved system when the literature is examined (Duchon, et al. 2014).

When examining the path planning algorithm, it can be seen that the x-y coordinates of the Ackermann can be adjusted automatically.

In addition, it has been seen that the path planning algorithm works with imaginary obstacles from the starting point to the goal point.

Since each cell value in the configuration space is :

$$f(a)= h(a)+g(a)$$

the A* algorithm is defined as the most efficient algorithm (Duchon, et al. 2014). Here g(a) is the cost of the path which begins with the initial point and finishes with the goal point. And h(a) is the heuristic cost of the cell (Manhattan, Euclidean, or Chebyshev) to the goal position. This array ends in the evaluated cell. Each adjacent cell of the actually reached cell is evaluated with the value f(a). The cell with the

lowest f(a) value is selected as the next cell in the array. The advantage of this algorithm is that the distances used as criteria can be adopted, modified or another distance can be added.

2.3.3.2. Dijkstra Algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph of a sample road route and was designed by computer scientist Edsger W. Dijkstra in 1956 (Frana,2010) (Dijkstra, 1959).

The basis of the operation of this algorithm is to find the shortest path (shortest path) over a graph. This algorithm calculates the shortest path from one node of a given shape to all other nodes. The idea of the algorithm is to continuously calculate the shortest distance starting from a starting point and exclude longer distances when updating.

In Figure 2.5 it can be seen from the graph which is created with an imaginary route. Let's assume that node A has been chosen as the starting node and show the operation of the algorithm starting from this node.

This algorithm assumes that there is no access to all nodes at the beginning and assigns an infinite value, so it cannot proceed to any node at the beginning. Then the starting node travels through all its neighboring nodes and thus updates the reach distance to these nodes. After this update, it updates the neighbors of the updated nodes and this process continues until all the nodes are updated and there is no new update on the figure.

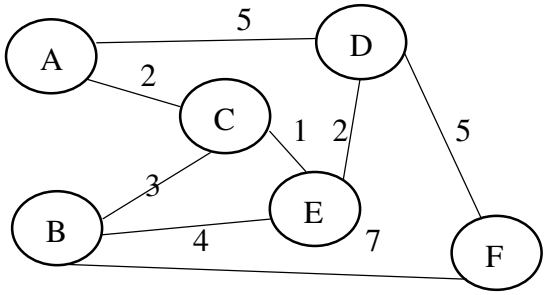

Figure 2.8. Example of Graph Structure for Dijkstra Algorithm.

According to this example graph, the nodes (C and D) that are neighbors of node A have been updated and the neighbors of these nodes will be updated in the next step.

In this graph above, the values written on the nodes give the shortest path distance to each node, starting from node A. For example, the cost of reaching node E in the figure is calculated as 3, and Dijkstra's algorithm claims that although equally different paths can be found (the shortest path will still be 3), there is no shorter path from node A to node F.

Unfortunately, this algorithm does not work well if there is a negative (-) value because the negative (-) value edge always produces a better result than the current state and the algorithm never becomes stable.

2.3.3.3. Breadth First Search (Bfs) Algorithm

It is one of the algorithms that perform searches in trees or graphs, and it performs a wide priority search in this algorithm, and the data structure that has taken its place in the literature for this algorithm is the queue data structure. In BFS, which uses a queue data structure, transactions also work on the FIFO (first in first out) principle.

Breadth first search (BFS) is an algorithm for searching the tree data structure for a node that satisfies a particular characteristic. It starts at the tree root and searches all nodes at the current depth before moving on to nodes at the next

depth level. Extra memory, usually a queue, is needed to keep track of child nodes encountered but not yet discovered.

In this algorithm, a starting node is determined and all its neighbors are visited, and all neighbors of the visited neighbor that can be visited at the same time are added to the queue at the same time. At each step, an element is removed from the queue and since it works with the FIFO principle, the first element comes out first.

Moreover, since the neighbors are visited, more than one neighbor can enter the queue in one step.

In summary, at each step, the neighbors of the frontmost node in the queue are looked at and removed and its neighbor added to the queue. If the node in the queue has no neighbors, only subtraction is made from the queue in the relevant step, that is, no additions are made since there are no neighbors.

2.4. Overview of System Software

In this study, the selection of computers that can read and process large amounts of data produced in autonomous vehicles is an important issue. This study aims to process and interpret the data received from the Lidar sensor and to make mapping and route planning safely as a result of the comments made. Thus, these received data were easily checked and algorithms were simulated on them. ROS is the main software of this study that can control robotic components from the NVIDIA Jetson card. After the literature review, it was decided to use the NVIDIA Jetson TX2 platform in this study. In addition, a computer with the Ubuntu operating system is required to install the necessary programs on this development board, and the JetPack file was installed before the operating system update. This software development kit contains the necessary applications for image processing.

In addition, algorithms that will perform autonomous driving have been developed and run in the ROS (Robotic Operating System) environment. The ZED

SDK is also installed to provide access to the ZED stereo camera and facilitate high-quality video recording and streaming.

OpenCV, which has the most advanced of the open source image processing libraries and can be used for the most different operations, has been established for the image processing part. Thanks to image processing, the data received from the cameras can be easily analyzed. Important and important parts of our software work; All versions of these software packages are compatible with each other and have been selected according to the system used.

And finally, with the Gazebo simulation tool of ROS, the Ackermann vehicle is simulated according to different three algorithms on the created track. Gazebo generally works with server-client logic. While Gzserver (server) performs concrete functions (robot navigates in the environment), Gzclient (client) ensures that the user's requests are fulfilled and the simulation is transferred to the visual. Many environments (worlds), robots, and objects can be accessed from Gazebo's online database, and these robots, which are widely available in the database, can work with ROS.

3. DISCUSSION

While testing the path planning algorithms, the attitude and path of three different algorithms were observed according to each scenario created on the track with or without obstacles. The track created with certain measures is shown in figure 3.1. and also created with obstacles as shown in figure 3.2.

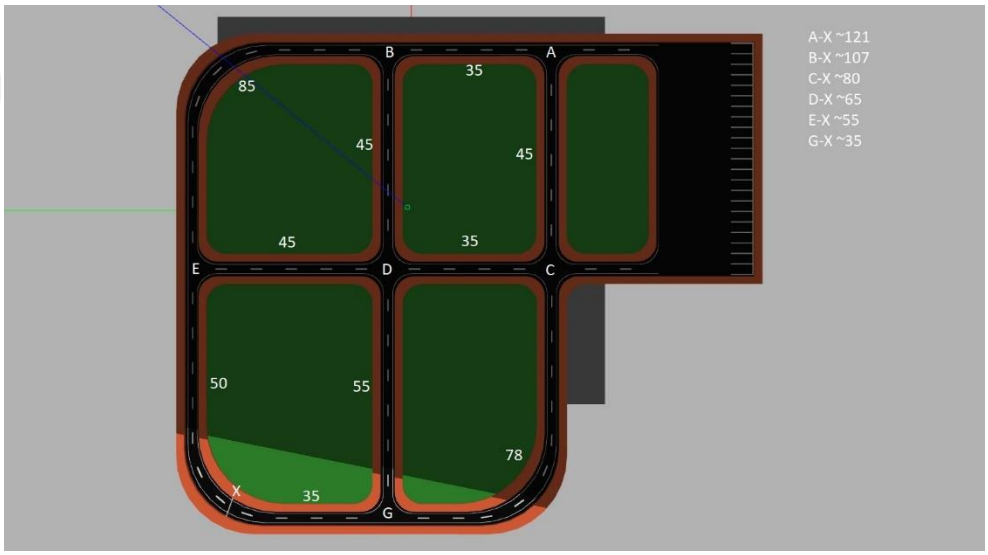

Figure 3.1. Dimensions of Created Track in Gazebo.

In this track, the starting point is A and the ending point is X, and the paths taken by each algorithm in different scenarios have been observed.

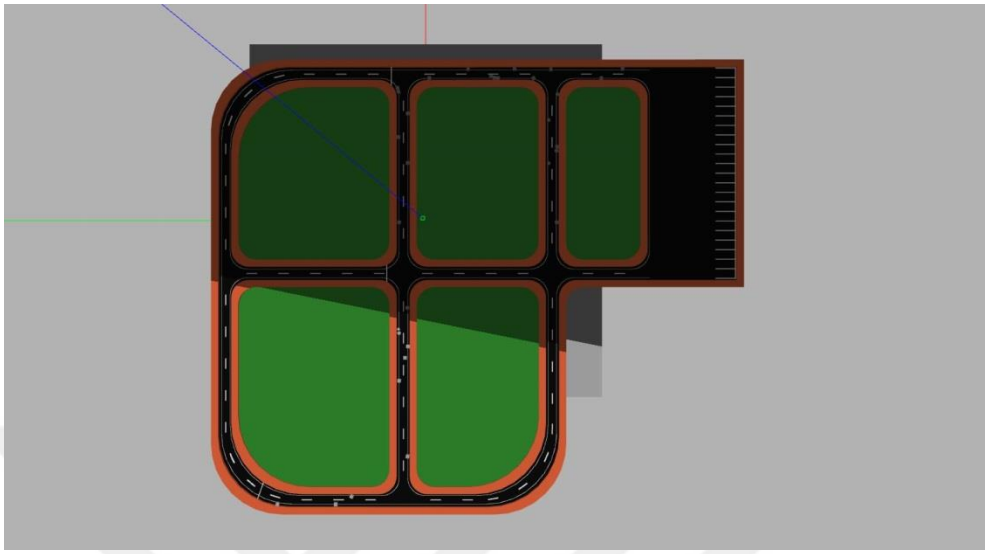

Figure 3.2. Example of a Track Created with Obstacles.

With the similarity to Djikstra's algorithm, the A* algorithm consists of a combination of the total cost, cost/distance from the current node to the starting node (path cost), and from the current node to the destination node (heuristic cost).

Ackermann is a differential drive robot and in this application, it was thought that the robot could move in one of the combinations of left and right rpm and select these parameters. The path cost of traveling to an adjacent node in the aforementioned direction is adjusted by the distance traveled. The heuristic cost is calculated by the straight line distance between the current node and the target node.

A* algorithm calculates the shortest path intuitively and using net distance data. The net distances between the alternative paths that can go from the starting node and the heuristic of these alternative paths to the destination point.

By adding the distances, the shortest path is chosen and the road continues. The same process is for each node (path) until the destination node is reached and the shortest path is calculated.

In Figure 3.3, the general structure of the A* algorithm is explained. The node to go to is selected from the found node. It is run again until the selected node

becomes the target node. Min=-1 is kept to find the minimum distance between the child nodes, that is, the path alternatives to go. MinChild is used to hold the child node, which is the shortest path. Then each child node is visited and each node (the way to go) is visited. It is visited until the index of the related parent node is found, and if the child node is controlled, the total distance value is kept in the min variable. If it is shorter than the total distance of the previously visited nodes, the min value and the node index are kept. Finally, the node with the minimum distance is returned after all children have been traversed.

```
Function minNode(ataNode)
# The node to go to is selected from the found node.
# Run again until the selected node becomes the target node

min = -1 # Keep to find the minimum distance between child nodes (route alternatives to go)
minChild = -1 # Used to hold the child node with the shortest path

# Every child node is visited
for childIndex, child in enumerate(ataNode.children):
# Every ancestor node (the way to go) is visited.
for ataIndex in range(len(child.ancestral)):
# It is browsed until the index of the related parent node is found.
if child.atalar[ataIndex] == ataNode:

# If the checked child is a node, the total distance value is kept in the min variable.
if min == -1:
min = child.estimated_distance + child.distances[ataIndex]
minChild = child
else:
# Min value and node index are kept if it is shorter than the total distance of previously visited nodes
if min > child.estimated_distance + child.distances[ataIndex]:
min = child.estimated_distance + child.distances[ataIndex]

minChild = child

end for
end for
# The node with the minimum distance is returned after all children have been traversed.
return minChild
end function
```

Figure 3.3. A* Algorithm.

The general structure of the BFS algorithm is shown in Figure 3.4. It aims to reach the target point with the least node change. It is checked whether the child nodes of each node are the target nodes, and if none of them are the target nodes, the children of these nodes and their children are continued until the target node is found.

```
Input: s as the
s = source node

function BFS(G, s)
Define the Q queue
Q.enqueue(s)

Mark s visited
while (Q is not empty)
v = Q.dequeue()

for all neighbors w of v in Graph G
if w is not visited
Q.enqueue(w)

endform()
mark w visited
end function
```

Figure 3.4. BFS Algorithm.

In Figure 3.5, the graph structure of the BFS algorithm in this track can be seen. The distances of each point between the start and end points to the neighboring nodes are summarized in this graph structure.

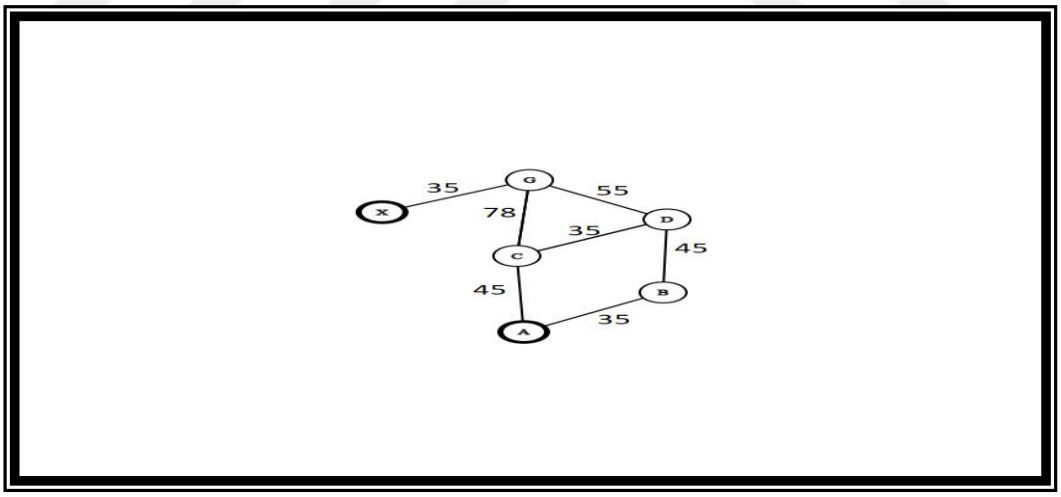

Figure 3.5. BFS Algorithm Graph Structure.

The general structure of the Dijkstra algorithm is shown in Figure 3.6. The working logic of the algorithm is to continuously calculate the shortest distance starting from a starting point and exclude longer distances when updating.

Dijkstra algorithm consists of the following steps:

- Initialization of all nodes with "infinite" distance; starting node with 0
- Marking the distance of the starting node permanent, all other distances temporary.
- Setting the starting node active.
- Calculation of the temporal distances of all neighboring nodes of the active node by adding the weights of its edges and the distance.
- If this calculated distance of a node is smaller than the current one, update the distance and set the current node as the antecedent. This step is also called updating and is Dijkstra's main idea.
- The effective setting of the node with minimum temporal distance. Mark the distance permanently.
- Repeating steps from the fourth line to the seventh line (Figure 3.6) until there are no nodes left with a permanent distance with neighbors still temporary distances.

```
function Dijkstra(Graph, source):
for each vertex v in Graph: // Initialization
dist[v] := infinity // initial distance from source to vertex v set to infinity
previous[v] := undefined //Previous node in optimal path from source
dist[source] := 0 // Distance from source to source
Q := the set of all nodes in Graph // // not all nodes in the graph are optimized - so in Q
while Q is not empty: // main loop
u := node in Q with smallest dist[ ]
remove u from Q
for each neighbor v of u:// / where v has not yet been removed from Q.
alt := dist[u] + dist_between(u, v)
if alt < dist[v]
dist[v] := sub
previous[v] := u
return previous[ ]
```

Figure 3.6. Dijkstra Algorithm.

At the end of the study, BFS, A*, and Dijkstra algorithms were observed on the obstacle and unobstructed track. In the Gazebo simulator, the routes created by the Ackermann vehicle on an obstacle track are shown in figures 3.7, 3.8, and 3.9, respectively. This obstacle track was simulated as the same scenario to compare each algorithm. First of all, when the simulation results are examined, it can be seen that the BFS and Dijkstra algorithms reach the endpoint on the same road and draw a different route from the A* algorithm.

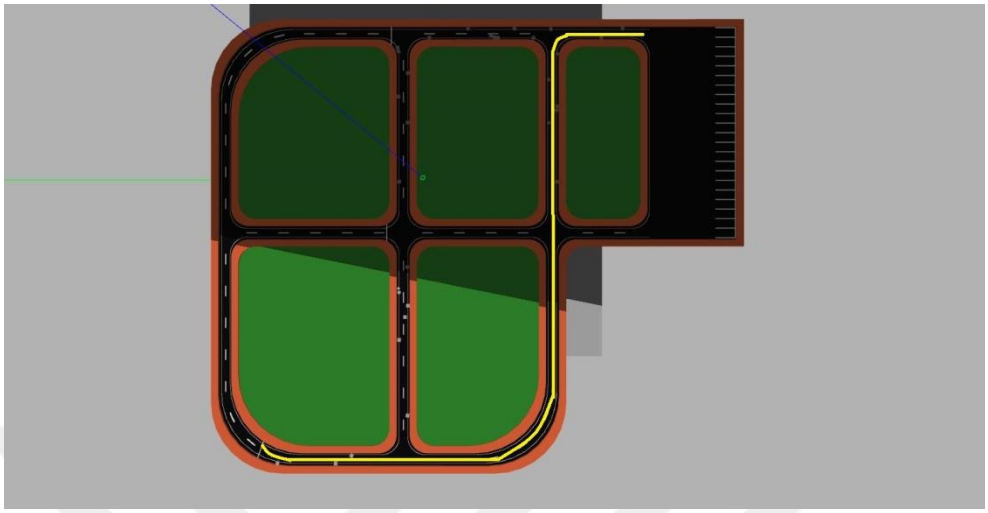

Figure 3.7. Path Taken with BFS Algorithm with Obstacles.

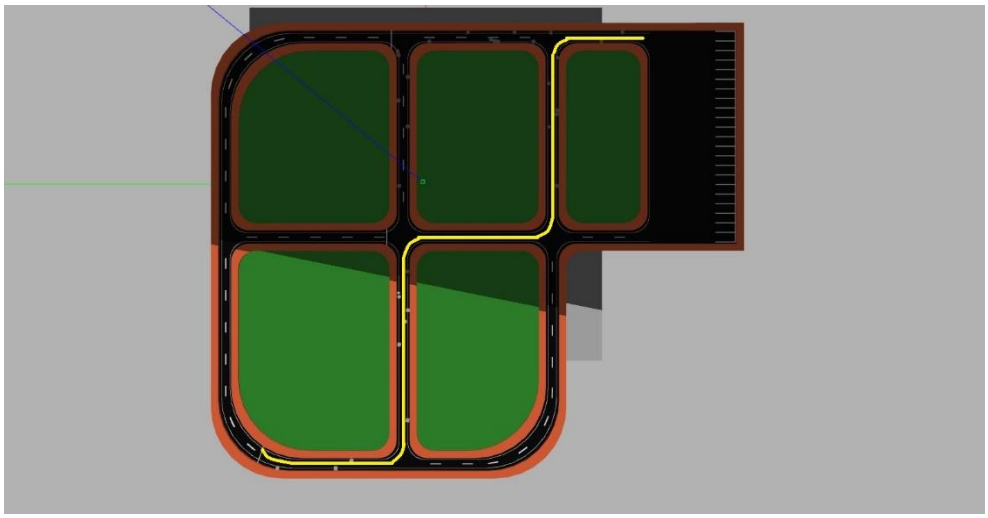

Figure 3.8. Path Taken with A*Algorithm with Obstacles.

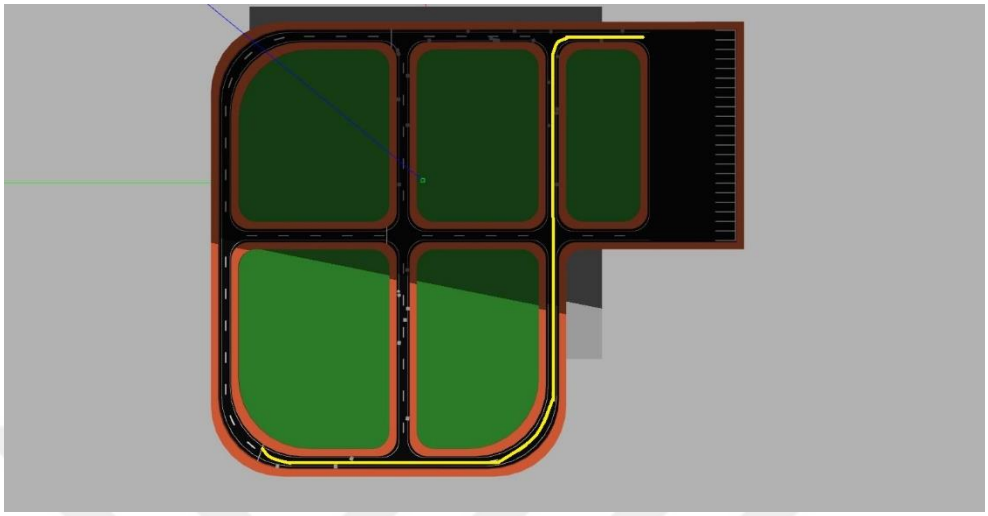

Figure 3.9. Path Taken with Dijkstra Algorithm with Obstacles.

In addition, the route of the BFS algorithm is seen on the barrier-free track. When this route and the route of the BFS algorithm for the disabled track are examined, it is seen that they differ. It is also clear that the simulation result of BFS is different from the scenario where other algorithms are on the obstacle course.

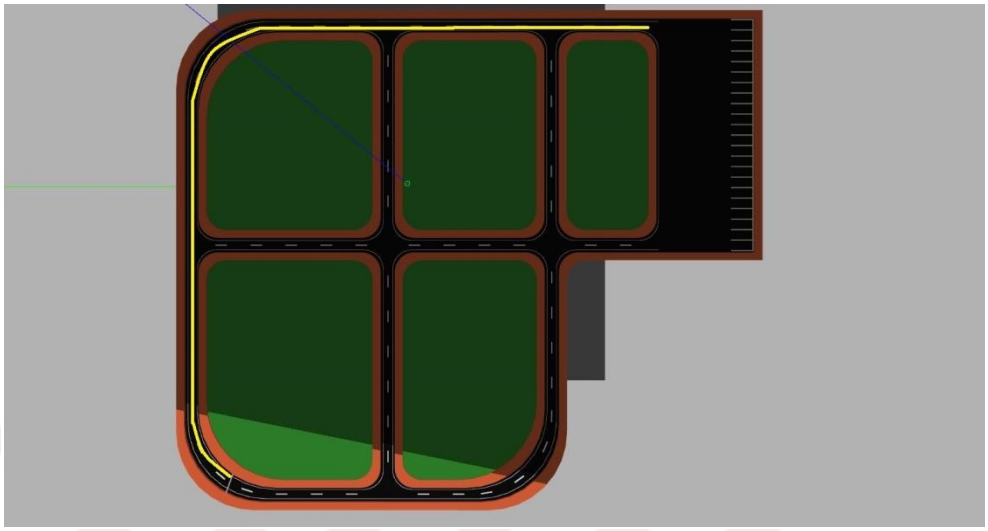

Figure 3.10. Path Taken with BFS Algorithm without Obstacles.

Table 3.1. Arrival Time of Algorithms at Destination Point.

Algorithm	Track Type	Arrival Time at Destination Point (sec)
Dijkstra	With Obstacle (Scenario 1)	117
A*	With Obstacle (Scenario 1)	117
BFS	With Obstacle (Scenario 1)	122
BFS	Without Obstacle (Scenario 2)	126
Dijkstra	Without Obstacle (Scenario 2)	117
A*	Without Obstacle (Scenario 2)	117

Table 3.1 shows the same start and end points of each algorithm, for the same scenario with obstacles in the first 3 lines, and the scenario of the BFS algorithm on the unimpeded track. When this table is examined, it can be seen that

the two algorithms that reach the target the fastest are Dijkstra and A* and that the slowest algorithm is BFS. In a simulation, it has been observed that the paths and times taken by A* and BFS algorithms for the two scenarios are the same.

Moreover, it can be seen that Dijkstra and BFS algorithms, which follow the same path, reach the target in a shorter time. This situation can show that Dijkstra and A* are two faster algorithms than the BFS. Dijkstra algorithm performs better than the BFS algorithm although it goes the same way as BFS from the initial to the target point.

In figure 3.11, it can be seen that the Ackermann vehicle cannot progress in any way due to the obstacles on the bends and has to stop to avoid hitting the obstacles.

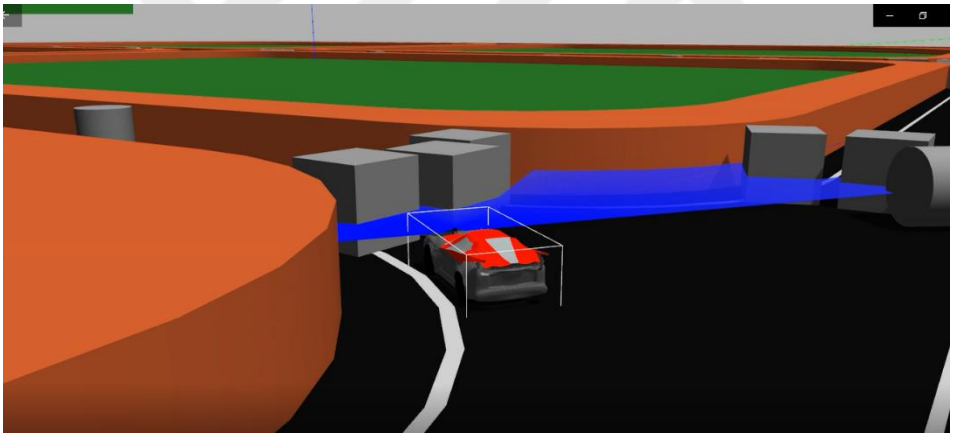

Figure 3.11. Simulation with Fully Closed Corners.

4. CONCLUSION AND FUTURE WORK

With the emergence of artificial intelligence studies, studies on autonomous systems have been focused on, and these studies have increased and accelerated with the development of hardware and software systems.

The idea of a self-driving vehicle has been an interesting and popular one for many reasons. Even though there have been many autonomous system developments from the past to the present, when we search the literature, many algorithm and decision mechanism development problems and costly hardware reasons show that these studies did not emerge in a very easy process. However, since this self-driving capability of vehicles and robots is thought to make human life easier and safer, studies on autonomous vehicle systems continue.

In this study, the path planning algorithms are simulated using the Ackermann vehicle on a track designed with disabled and unobstructed scenarios with certain dimensions.

Also, SLAM Rviz which is the newest and most popular algorithm framework for the autonomous vehicle concept is examined in the simulation environment.

In the first application, the data obtained with RP-LIDAR was visualized by mapping it with the Rviz simulator in the ROS (Robot Operating System) in the Ubuntu environment.

In the main road simulation, packages related to ROS, Gazebo simulator, and Ackermann vehicle have been loaded and a track with certain dimensions, and starting and ending points have been designed. The results for each scenario were analyzed and compared.

Using A*, BFS, and Dijkstra algorithms to map a prepared environment and determine the robot's path planning, it was observed that the Ackerman simulator successfully navigated to the target points despite obstacles.

In addition, the fact that these algorithms have some improvements thanks to the studies developed and accelerated in recent years shows that these algorithms are promising.

It has been observed that during the path planning performed, the Ackermann successfully recognize their surroundings and perform positioning, mapping, and route planning.

With this study, it has been examined that three different path planning algorithms successfully recognize Ackerman's surroundings and avoid obstacles while positioning and route planning.

As a result of the simulation made for 3 different algorithms for a specific scenario, it was observed that the two algorithms that completed the track the fastest as a result of the track times and the routes took were Dijkstra and A*, and the slowest algorithm was BFS. In addition, a simulation was performed where the obstacles blocked the first and second bends, and it was observed that the vehicle could not move forward.

For future work, this system can be tested in a real vehicle system with a more responsive hardware system (sensors and extra Lidar).

Furthermore, this system can be simulated with objects whose speed is lower than the speed of the vehicle.

In addition, in a future study, a more comprehensive and advanced sensing system can be designed and tested, and a more advanced autonomous vehicle can be realized. Minimizing the error rates in detection, mapping, and route planning, increasing the speed of the vehicle, and enabling the vehicle to operate more efficiently can be a guide for future research.

Furthermore, this work can be simulated with objects whose speed is lower than the speed of the vehicle.

REFERENCES

Abadi, Martín, et al. 2016. Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems., 2016.

Abdessalem, Raja, Ben, Nejati, Shiva, Briand, Lionel, C, and Stifter, Thomas. 2016. Testing Advanced Driver Assistance Systems Using Multi-Objective Search and Neural Networks. In Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on. IEEE. pp. 63–74.

An Goodfellow and Nicolas Papernot. 2017. The challenge of verification and testing of machine learning. http://www.cleverhans.io/security/privacy/ml/2017/06/14/verification.html/ , 25 April 2020.

Anand, Saswat, et al. 2013. An Orchestrated Survey of Methodologies for Automated Software Test Case Generation. Journal of Systems and Software , 2013. pp. 1978–2001.

Anderson, Hyrum. 2017. Evading Next-Gen AV using A.I. https://www.defcon. org/html/defcon-25/dc-25-index.html/ (24 Temmuz 2020).

Aufrère, Romuald., et al. 2003. Perception for Collision Avoidance and Autonomous Driving. Mechatronics, 2003. pp. 1149-1161.

Bastani, Osbert et al. 2016. Measuring Neural Net Robustness with Constraints. In Advances in Neural Information Processing Systems. pp. 2613–2621.

Bautista, David, Gonzalez et al.. 2016. A Review of Motion Planning Techniques for Automated Vehicles. IEEE Trans. Intelligent Transportation Systems, 2016. pp. 1135-1145.

Bengio, Yoshua et al. 2007. Greedy Layer-Wise Training of Deep Networks. In Advances in Neural Information Processing Systems. pp. 153–160.

Bojarski, Mariusz et al. 2016. End to End Learning for Self-Driving Cars. arXiv preprint arXiv:1604.07316.

Canny, John . 1986. A Computational Approach to Edge Detection. pp. 679–698.

Carlini, Nicholas and Wagner, David. 2017. Towards Evaluating the Robustness of Neural Networks. In Security and Privacy (SP), IEEE Symposium, 2017. pp. 39–57.

Chen, Tsong, Y, Cheung, Shing, C and Yiu, Shiu, Ming. 1998. Metamorphic Testing: A New Approach for Generating Next Test Cases. Technical Report. Technical Report. HKUST-CS98-01, Department of Computer

Cisse, Moustapha, et. al. 2017. Parseval networks: Improving Robustness to Adversarial Examples. In International Conference on Machine Learning. pp. 854–863.

Coelingh Erik, Eidehall Andreas, and Bengtsson Mattias. 2010. Collision Warning with Full Auto Brake and Pedestrian Detection - A Practical Example of Automatic Emergency Braking. In Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems (ITSC): 19-22 September 2010; Funchal, Portugal, 2010. pp. 155-160.

Dijkstra, E. W. 1959. A Note on Two Problems in Connexion with Graphs. pp. 269–271.

Duchon, et al. 2014. Path Planning with Modified A star Algorithm for a Mobile Robot. pp. 59-69.

Durrant-Whyte, Hugh, and Bailey, Tim. 2006. Simultaneous Localization and Mapping (SLAM): Part i The Essential Algorithms. IEEE Robot Autom. Mag.; 13(2). pp. 99-100.

Eliwa, Mustafa, Adham, Ahmed, Sami, Islam and Eldeeb, Mahmoud. 2017. A critical comparison between Fast and Hector SLAM algorithms. REST Journal on Emerging trends in Modelling and Manufacturing Vol:3(2), 2017 REST Publisher ISSN:. pp. 2455-4537

Frana, Phil. 2010. An Interview with Edsger W. Dijkstra. Communications of the ACM. 2010. pp. 41-47. Vol. 53.

Jang, Se, Jin. 2016. Trend of S/W Technology Related to Autonomous Driving Car. The Journal of the Korean Institute of Communication Sciences, 2016. pp. 27-33. Vol. 33.

Matzka, Stephan., and Altendorfer, Richard. 2009. A comparison of Track-to-Track Fusion Algorithms for Automotive Sensor Fusion Multisensor Fusion and Integration for Intelligent Systems. Seoul, Korea: Springer, 2009.

Lozano-Perez, Tomas, et al. 1984. Automatic Synthesis of Fine-Motion Strategies for Robots. The International Journal of Robotics Research, 1984. pp. 3-24.

Nagel, Robert, Eichler, Stephan, and Eberspacher, Jorg. 2007. Intelligent Wireless Communication for Future Autonomous and Cognitive Automobiles. Intelligent Vehicles Symposium, IEEE, 2007.

Patle, et al. 2019. A review: On Path Planning Strategies For Navigation of Mobile Robot. pp.582-606

SAE, T. , 2016. Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. SAE Standard J3016.

Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

Smith, Randall, and Cheesman, Peter. 1987. On the Representation of Spatial Uncertainty. Int. J. Rob. Res., 1987. pp. 56-68

Vidal-Calleja, Teresa., Andrade-Cetto, Juan and Sanfeliu, Alberto. 2004. Conditional For Suboptimal Filter Stability in SLAM, IEEE Inernational Conference on Intelligent Robots and Systems, Sendai, Japan, 28 Sept -2 Oct, 2004. pp. 27-32.

Zhang, Xuexi et al. 2020. 2D Lidar-Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots. Journal of Advanced Transportation, vol. 2020.

Wang, Chieh-Chih et al. 2007. Simultaneous Localization, Mapping and Moving Object Tracking, The International Journal of Robotics Research, vol. 26, no. 9, September 2007. pp. 889–916,

CURRICULUM VITAE

Zeynep Yasemin Erdogan. She completed her undergraduate education in 2018 and during her undergraduate education, She focused on some hardware and software studies in the electric vehicle racing project. She started her master's degree at Çukurova University in 2018. She works at her first workplace, " Merinos Carpet Industry and Trade Inc. " and still works here.