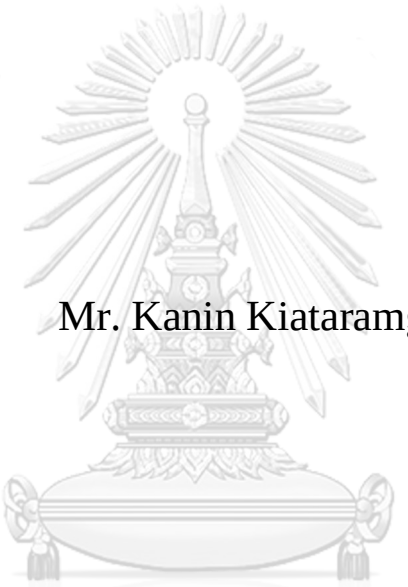
PATH FOLLOWING AND OBSTACLE AVOIDANCE FOR AUTONOMOUS VEHICLE BASED ON GNSS LOCALIZATION

Mr. Kanin Kiataramgul



A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Mechanical Engineering
Department of Mechanical Engineering
FACULTY OF ENGINEERING
Chulalongkorn University
Academic Year 2019

ระบบขับตามเส้นทางและหลบสิ่งกีดขวางสำหรับยานยนต์อัตโนมัติโดยใช้ระบบดาวเทียมนำร่องในการระบุตำแหน่ง

นายคณิน เกียรติอร่ามกุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมเครื่องกล ภาควิชาวิศวกรรมเครื่องกล
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2562

Thesis Title	PATH FOLLOWING AND OBSTACLE AVOIDANCE FOR AUTONOMOUS VEHICLE BASED ON GNSS LOCALIZATION
By	Mr. Kanin Kiataramgul
Field of Study	Mechanical Engineering
Thesis Advisor	Assistant Professor NUKSIT NOOMWONGS, Ph.D.

Accepted by the FACULTY OF ENGINEERING, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Engineering

..... Dean of the FACULTY OF ENGINEERING
(Associate Professor SUPOT TEACHAVORASINSKUN, Ph.D.)

THESIS COMMITTEE

..... Chairman
(Assistant Professor SUNHAPOS CHANTRANUWATHANA, Ph.D.)

..... Thesis Advisor
(Assistant Professor NUKSIT NOOMWONGS, Ph.D.)

..... Examiner
(Assistant Professor GRIDSADA PHANOMCHOENG, Ph.D.)

..... External Examiner
(Pasan Kulvanit, Ph.D.)

คณิน เกียรติอร่ามกุล : ระบบขับตามเส้นทางและหลบสิ่งกีดขวางสำหรับยานยนต์อัตโนมัติโดยใช้ระบบดาวเทียมนำร่องในการระบุตำแหน่ง. (PATH FOLLOWING AND OBSTACLE AVOIDANCE FOR AUTONOMOUS VEHICLE BASED ON GNSS LOCALIZATION) อ.ที่ปรึกษาหลัก : ผศ. ดร.นักสิทธ์ นุ่มวงษ์

ในปัจจุบัน ระบบต่าง ๆ ที่แต่เดิมอาศัยมนุษย์ในการดำเนินงานกำลังได้รับการพัฒนาให้สามารถทำงานได้โดยพึ่งพาการควบคุมจากมนุษย์ให้น้อยที่สุด และหนึ่งในระบบที่กำลังได้รับความสนใจมากที่สุดระบบหนึ่งนั้นได้แก่ ระบบยานยนต์อัตโนมัติ อย่างไรก็ตามระบบยานยนต์อัตโนมัติที่กำลังได้รับการพัฒนาอยู่ในปัจจุบันนั้น โดยส่วนใหญ่ได้มีการนำอุปกรณ์รับรู้ที่มีราคาสูง เช่น กล้องและเครื่องกราดตรวจเลเซอร์ มาใช้เป็นอุปกรณ์รับรู้หลักของระบบ อีกทั้งอุปกรณ์เหล่านี้ไม่สามารถทำงานได้โดยลำพัง โดยยังคงต้องอาศัยหน่วยประมวลผลที่มีประสิทธิภาพสูงมาใช้ในการทำงานร่วมกันอีกด้วย ส่งผลให้ระบบที่พัฒนาขึ้นมีราคาที่ค่อนข้างสูงจึงไม่เหมาะสมต่อการนำไปประยุกต์ใช้งานในหลายรูปแบบ ดังนั้น งานวิจัยนี้จึงมุ่งเน้นที่จะนำอุปกรณ์หรือระบบตรวจรู้ชนิดอื่นที่เมื่อนำมาใช้แล้วสามารถทำให้ระบบที่พัฒนาขึ้นมีราคาที่ค่อนข้างต่ำเมื่อเทียบกับระบบอื่นที่กำลังพัฒนาอยู่ในปัจจุบัน ดังนั้น ระบบดาวเทียมนำร่องโดยอาศัยการปรับแก้แบบจลน์ตามเวลาจริง (เรียลไทม์ไคนิเมทิก) จึงได้ถูกนำมาใช้เป็นอุปกรณ์ตรวจรู้หลักสำหรับงานวิจัยนี้ อย่างไรก็ตาม เครื่องกราดตรวจเลเซอร์จะยังคงถูกนำมาใช้ในระบบที่พัฒนาขึ้นในฐานะอุปกรณ์ตรวจรู้รองสำหรับการตรวจจับสิ่งกีดขวางซึ่งไม่สามารถรับรู้ได้โดยอาศัยระบบดาวเทียมนำร่องเพียงอย่างเดียว โดยระบบที่พัฒนาขึ้นจะถูกออกแบบให้ใช้งานได้อย่างมีประสิทธิภาพในสภาวะแวดล้อมที่เอื้ออำนวยต่อการทำงานของระบบดาวเทียมนำร่องที่ความเร็วไม่เกิน ๑๕ กิโลเมตรต่อชั่วโมง ในงานวิจัยนี้ รถยนต์พลังงานไฟฟ้าขนาดเล็กได้ถูกนำมาดัดแปลง โดยทำการติดตั้งอุปกรณ์ที่ได้ออกแบบขึ้นสำหรับควบคุมระบบบังคับเลี้ยวและระบบควบคุมความเร็วซึ่งประกอบไปด้วยระบบควบคุมอัตราเร่งและระบบห้ามล้อ โดยรับคำสั่งจากระบบควบคุมระดับสูง จากนั้นจึงทำการพัฒนาส่วนชุดคำสั่งของระบบควบคุมระดับสูง ซึ่งระบบควบคุมระดับสูงนี้ได้ถูกออกแบบให้สามารถควบคุมรถไปตามเส้นทางที่กำหนดโดยมีระบบดาวเทียมนำร่องเป็นอุปกรณ์รับรู้สำหรับกระบวนการระบุตำแหน่ง และใช้เครื่องกราดตรวจเลเซอร์ในการหลบหลีกสิ่งกีดขวางโดยอาศัยขั้นตอนวิธีการให้คะแนนเส้นทางที่คาดการณ์ ซึ่งเป็นขั้นตอนวิธีที่พัฒนาขึ้นในงานวิจัยนี้ ต่อมาจึงได้ทำการปรับแต่งตัวแปรต่าง ๆ ที่ส่งผลต่อระบบควบคุมทั้งในระดับล่างและระดับสูงจนผลการตอบสนองที่ได้เป็นที่น่าพึงพอใจ จากนั้นจึงได้ทำการทดสอบและประเมินผลระบบนำทางยานยนต์อัตโนมัติที่ได้พัฒนาขึ้นในสถานที่ทดสอบควบคุม โดยแยกทำการทดสอบระหว่างระบบขับตามเส้นทางและระบบหลบสิ่งกีดขวาง ซึ่งหลังจากได้ทำการทดสอบระบบขับตามเส้นทางที่ความเร็ว ๑๐ และ ๑๕ กิโลเมตรต่อชั่วโมง พบว่า ระบบที่ออกแบบขึ้นสามารถใช้การได้ถึงแม้ว่าบางช่วงของเส้นทางการทดสอบจะถูกปกคลุมไปด้วยต้นไม้ใหญ่หรือถูกล้อมรอบไปด้วยอาคารซึ่งเป็นข้อด้อยของระบบดาวเทียมนำร่อง โดยผลการทดสอบมีระยะเบี่ยงเบนออกจากเส้นทางเฉลี่ยประมาณ ๑๐ เซนติเมตร สำหรับระบบหลบสิ่งกีดขวาง ผลการทดสอบพบว่าระบบมีการตอบสนองต่อสิ่งกีดขวางตามขั้นตอนวิธีที่ได้กำหนดไว้ โดยสามารถหลบหลีกสิ่งกีดขวางและกลับเข้าสู่เส้นทางที่กำหนดไว้ได้อย่างปลอดภัย

สาขาวิชา	วิศวกรรมเครื่องกล	ลายมือชื่อนิสิต .....
ปีการศึกษา	2562	ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

6070403421 : MAJOR MECHANICAL ENGINEERING
KEYWORD: autonomous driving system, autonomous navigation system, path following system, obstacle avoidance system, global navigation satellite system

Kanin Kiataramgul : PATH FOLLOWING AND OBSTACLE AVOIDANCE FOR AUTONOMOUS VEHICLE BASED ON GNSS LOCALIZATION. Advisor: Asst. Prof. NUKSIT NOOMWONGS, Ph.D.

Nowadays, many systems, formerly operated by human beings, are now developed to minimize user control effort. One outstanding system that receives close attention is the autonomous driving system. However, several autonomous driving systems that are currently developed utilize expensive sensing devices, e.g., camera and laser scanner. Moreover, these devices cannot be solely employed but a high-performance processing unit is also required. These pricey components result in an expensive system that does not worth to be used in some practical applications. Therefore, this research intends to utilize other low-cost sensing devices so that the final price of the developed system can be reduced. Hence, the Global Navigation Satellite System (GNSS) with a Real-Time Kinematic (RTK) correction was applied to this research as a prior sensor. However, a laser scanner was still employed as a complementary sensor to detect obstacles which cannot be detected by the GNSS alone. This developed system was designed to effectively operate at a travel speed lower than 15 kilometers per hour in a GNSS-friendly environment. In this research, the micro electric vehicle was modified by installing the steering control system and the speed control system, which consists of the acceleration control system and the braking control system. These supplementary systems are controlled by the high-level control system. Next, the high-level control system software was developed. This software controls a vehicle to follow a predefined route by using the GNSS in a localization process and using a laser scanner in the obstacle avoidance algorithm which was developed in this research, i.e., the Scored Predicted Trajectory. Then, the parameters, which affect the high and low-level control system characteristic, was tuned until a satisfactory response was achieved. Next, the developed autonomous navigation system evaluation experiment was conducted in a controlled environment area by separately evaluated the path following system and the obstacle avoidance system. After the path following system experiment was launched using a travel speed of 10 and 15 kilometers per hour, it has been found that the developed system effectively performs even some portions of the test track are either covered by large trees or surrounded by buildings, i.e., the environment by which the performance of the GNSS is degraded. The result of this experiment shows the average deviation distance from the waypoint of about 10 centimeters. In the obstacle avoidance system experiment, the result shows that the developed system responds to the obstacle by evading it and safely converging to the predefined path according to the designed algorithm.

Field of Study:	Mechanical Engineering	Student's Signature .....
Academic Year:	2019	Advisor's Signature .....

ACKNOWLEDGEMENTS

After three years have passed, this research now reaches its destination. There are so many people contribute to this achievement. First of all, I am so grateful to Asst Prof. Nuksit Noomwongs, my supervisor in this master's degree topic and also project advisor in my bachelor's degree level, and Asst. Prof. Sunhapos Chantranuwathana for have been supporting and advising me during my years as an undergraduate student and these years of graduate studies. I thank Soravas Treenok and Kant Chaisuwan for my best time of being a member of the Smart Mobility Research Center (SMRC), Faculty of Engineering, Chulalongkorn University. I also thank Sedtawud Larbwisuthisaroj for his information that has been used in this research and thank Krit T'Siriwattana for his helping hands. For the very first year of this research, I thank Thawin Subpinyo, Nattakit LeelapornUdom, Pobtham Sookatup, Peerathad Nakdilok, Patthara Chuanakha, and Puvit Sinrat for their assistance in the early days of this research. For the second year, I am so thankful to Kuptabadee Tantipukanont, Kwannuan Klaykew, Lattapol Wongpiya, Noppakit Tang-on, Prin Phutthaburee, and Thanadol Kosolvattanasombat for their contribution to a great leap in this research. Also, I would like to thank Kanutsanant Banpakan, Junraprach Petchang, Chumpol Hamprommarat, Anan Sutapun, and Surat Kwanmuang for their tools and technical support during the time this research was in process. I also would like to express my gratitude to Akira Nagao and his Asia Technology Industry's staff for device and technical support provided to this research. Also, I am so grateful to the Toyota Motor Thailand (TMT) for the insight specification, which has been used in modifying the Toyota COMS, and for funding this research. And I would like to give my special thanks to Suphap Mou-ubom and Nutwara Chunsakul for non-technical support, however, tremendously contribute to the accomplishment of this research. Moreover, I am thankful to Asst. Prof. Gridsada Phanomchoeng for accepting to be a member of this thesis committee, and Pasan Kulvanit for accepting to be an external examiner of this thesis. At last, I would like to express my thankfulness to my family and all who somehow contributed to this accomplishment for their encouragement, either mentally or physically, throughout these years of this research.

Kanin Kiataramgul

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I
INTRODUCTION

1. Background and Significant of the Research Problem

Autonomous driving nowadays receiving great attention from the public and their authority throughout the world. Several numbers of organizations, not only the originally automotive field companies, are now focusing on the development of the autonomous vehicle. Among these corporations, there appear a common objective between them in developing of an autonomous driving system which is to reach the solutions to any complications emerge from currently traffic situation, e.g., improving traffic congestion problem, enhancing road safety by diminishing human mistake, etc. However, before moving to further topics, a common convention on terminology should be established. In the following section, a definition of levels of driving automation is presented.

1.1. Levels of Driving Automation According to SAE

According to the Society of Automotive Engineers (SAE) standard[1], levels of driving automation can be divided into 6 levels, ranging from 0th to 5th level, which a higher level represents a higher degree of driving automation, i.e. 0th and 5th level autonomous indicates no automation and full automation, respectively. The followings describe a narrative definition of each level.

1.1.1. 0th Level, No Automation

In this level, the human driver performs full control all over the driving period. Some of the warning systems may be introduced to this level, however, the driver's decision completely dominates the dynamics driving task. Example of mentioned warning systems which are now available is Lane Change Assistance (LCA), Lane Departure Warning (LDW), Forward Collision Warning (FCW), Park Distance Control (PDC), etc.

1.1.2. 1st Level, Driver Assistance

An assistance system is introduced to this level. This assistance system could be either by automated steering or acceleration control which performs on human driver request, however, other dynamic driving tasks are fully controlled by the driver. Available technologies of these assistance systems including Adaptive Cruise Control (ACC), Park Assist (PA), Lane Keeping Assist (LKA), etc.

1.1.3. 2nd Level, Partial Automation

More assistance systems are applied to this level. Identical to the 1st Level, the system is still activated only when the human driver making a request. Nevertheless, other remaining dynamic driving tasks are fully controlled by the driver. The following named assistance systems are included in this level: Park Assistance, Traffic Jam Assist, etc.

1.1.4. 3rd Level, Conditional Automation

All dynamics driving tasks perform automatedly by automated assistant systems. However, in some situations, an automated assistant system could request a human driver to take control of a dynamic driving task. Some features included in this level are Traffic Jam Chauffeur, Motorway Chauffeur (MWC), etc.

1.1.5. 4th Level, High Automation

The distinction between this 4th Level and the 3rd Level of driving automation is that in this level, whenever in some defined situations and a human driver is requested by an automated system to intervene. If the driver does not respond to this request appropriately, the automated system should have the ability to perform a proper dynamic driving task. Highway Pilot and Piloted Parking are some of the example functions in this level.

1.1.6. 5th Level, Full Automation

A fully automated system in this level performs all dynamic driving tasks for the overdriving period. There will be no request for a human driver to intervene in a

dynamic driving task in any situation. However, the human driver still possesses an ability to manage the roadways and environmental conditions.

Table 1 shows a summary of the level of driving automation according to the SAE. Note here for named technologies, features, and assistant systems mention above are not all commercially introduced[1].

Table 1 Summary of level of driving automation according to SAE[1]

Summary of Levels of Driving Automation for On-Road Vehicles

This table summarizes SAE International's levels of *driving* automation for on-road vehicles. Information Report J3016 provides full definitions for these levels and for the italicized terms used therein. The levels are descriptive rather than normative and technical rather than legal. Elements indicate minimum rather than maximum capabilities for each level. "System" refers to the driver assistance system, combination of driver assistance systems, or *automated driving system*, as appropriate.

The table also shows how SAE's levels definitively correspond to those developed by the Germany Federal Highway Research Institute (BASt) and approximately correspond to those described by the US National Highway Traffic Safety Administration (NHTSA) in its "Preliminary Statement of Policy Concerning Automated Vehicles" of May 30, 2013.

Level	Name	Narrative definition	Execution of steering and acceleration/ deceleration	Monitoring of driving environment	Fallback performance of *dynamic driving task*	System capability (*driving modes*)	BASt level	NHTSA level
Human driver **monitors the driving environment**								
0	No Automation	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a	Driver only	0
1	Driver Assistance	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes	Assisted	1
2	Partial Automation	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes	Partially automated	2
Automated driving system **("system") monitors the driving environment**								
3	Conditional Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task* with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes	Highly automated	3
4	High Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes	Fully automated	3/4
5	Full Automation	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**	-	

1.2. Level of Driving Automation According to BASt

According to the report from Transportation Research Board (TBR), the German Federal Highway Research Institute (BASt)[2]. The level of driving automation can be classified as 5 levels including 1) Driver Only 2) Driver Assistance 3) Partial Automation 4) High Automation and 5) Full Automation. For the first 3 levels, their definitions are identical to the first 3 levels described by the Society of Automotive Engineers (SAE). For the last 2 levels, their definition can also be described by the 3rd and 4th Levels of the SAE classification, respectively. Table 2 describes a narrative summary of the level of driving automation according to the BASt.

Table 2 Summary of level of driving automation according to BASt[3]

Definitions	Descriptions
Driver Only	Human driver executes manual driving task
Driver Assistance	The driver permanently controls either longitudinal or lateral control. The other task can be automated to a certain extent by the assistance system.
Partial automation	The system takes over longitudinal and lateral control, the driver shall permanently monitor the system and shall be prepared to take over control at any time.
High Automation	The system takes over longitudinal and lateral control; the driver must no longer permanently monitor the system. In case of a take-over request, the driver must take-over control with a certain time buffer
Full Automation	The system takes over longitudinal and lateral control completely and permanently. In case of a take-over request that is not carried out, the system will return to the minimal risk condition by itself.

From here on, any levels of automation mentioned will refer to the levels classified by the Society of Automotive Engineers.

Throughout this text, the main attention is given to the 2nd to 5th levels of driving automation. Since driver assistance systems are fundamentals for autonomous or self-driving vehicles, the following section gives a brief explanation for theses selected systems.

1.3. Driver Assistance Systems

Driver assistance systems are fundamentals for vehicles categorized in the 2nd to 5th driving automation level. Longitudinal and lateral directions control are mainly systems applied to an autonomous driving system. In this section, cruise control collision avoidance and lane-keeping assist systems are taken to be introduced[4].

1.3.1. Cruise Control

The cruise control system is applied to a vehicle to maintain the speed of a vehicle or space between the host and a preceding vehicle. The cruise control system can be classified as 2 different types, standard and adaptive cruise control systems. In the standard cruise control system, only speed is to be maintained by means of controlling vehicle acceleration or deceleration. For adaptive cruise control (ACC), both spacing, refer to space between vehicles, and speed are controlled. By comparing space to a

certain threshold, depending on the vehicle's speed, an adaptive cruise control system can then determine whether space or speed control is safe and appropriate to the current driving situation.

1.3.2. Collision Avoidance

Like the adaptive cruise control system, however, the collision avoidance system contains some different details that differ from the adaptive cruise control system. Collision avoidance system operates by deciding whether a current driving speed is safe or not, then, if dynamics driving take is indispensable to avoid a collision, deceleration, or even emergency braking is performed. Collision warning may also be included in this system.

Furthermore, there appear several researches suggest that collision avoidance cannot be performed by only braking or deceleration[5, 6]. But in some situations, depending mainly on the time to collision (TTC), a duration which a host vehicle will collide with a preceding one if a current velocity is maintained, evasive steering man

1.3.3. Lane Keeping Assist

Lane-keeping assist (LKA) system provides an automated lateral position control for a vehicle to keep the vehicle's lateral position in a proper region between lane marking and prevent a vehicle from an unintended lane changing. This system is extended from the lane departure warning (LDW) system by including actuators to control and perform the dynamic driving task. The crucial part of the system is to detect and estimate the lateral position of a vehicle. Several organizations have proposed different techniques in measuring a lateral position, e.g., magnetic field guided, vision base measurement, a global positioning system (GPS).

This research will utilize these driver assistance systems as fundamentals, and by further system implementation, to develop an autonomous, or a self-driving car, based on the global navigation satellite system (GNSS).

2. Objectives of Research

2.1. To develop the hardware segment of the low-level control system including the speed control system, braking control system, and steering control system.

2.2. To develop a software of the high-level control system based on the global navigation satellite system localization.

2.3. To develop the obstacle avoidance algorithm for the collision avoidance system

3. Scope of the Research

3.1. To develop hardware as implementations that can be used in an autonomous driving system.

3.2. A developed autonomous navigation system mainly relies on the GNSS positioning system, which effectively operates in an open sky area.

3.3. The expected operation speed in this research is set to be under 15 kilometers per hour.

3.4. A developed autonomous vehicle can safely interact with a detected threat in a predefined scenario.

4. Expected Benefits

Since other autonomous vehicle systems that were recently developed mostly utilize an expensive sensing device, e.g., 3D lasers scanner, and require a high-performance processing unit. Hence, this research aims to develop an alternative autonomous navigation system that employs a lower cost sensing device compare to other developed systems that are currently available. However, the developed system is not intended to be used in the same application level as the available systems. The developed system will be designed to be used in a low-speed application, i.e. below 15 kilometers per hour, and operates in a constrained environment.

CHAPTER II
LITERATURE REVIEW

1. Localization Techniques

To set a navigation course, an autonomous vehicle needs to know its position, either global or local position. Besides position in space, orientation, or pose, is also required for autonomous driving navigation. Localization is, therefore, one of the most crucial parts of mobile robotics which can also be applied to a low speed autonomous, the main consideration of this research topic. The following presents reviews of some literature on the localization method in a different technique.

1.1. Landmark-based Localization

Landmark-based localization can be classified as 2 main types, i.e. natural and artificial landmark-based localization.

1.1.1. Natural Landmark-based Localization

Natural landmark-localization is the localization method by which a position is determined using features extracted from the actual unmodified environment. Various types of sensing information can be used as input for feature extraction, e.g., radial distance from a laser rangefinder. According to R. Madhavana and H. F. Durrant-Whyte in "Natural landmark-based autonomous vehicle navigation"[7], Their research focuses on developing an algorithm to effectively extract natural dominant point landmarks obtained by using a laser range finder. In their research, features from the unmodified environment are extracted by applying a specific technique called the Curvature Scale Space (CSS) algorithm. In brief, extracted curvatures are derived from segmented range images from a laser rangefinder. These segmented range images are convoluted by Gaussian kernel with different levels of scale, depending on kernel's width to produce preferable curvature values. Dominant curvatures, extrema curvatures, are then identified by applying to a certain condition.

In the localization section, these dominant curvatures are used as natural landmarks along with the odometry method, i.e. relative positioning by dead-

reckoning estimation, by applying the Extended Kalman Filter (EKF) to determined vehicle position. After comparing with reference ground truth position, i.e. Real-Time Kinematic (RTK) Global Positioning System (GPS), they state that error results in lower than 25 centimeters for the position and 2 degrees for orientation.

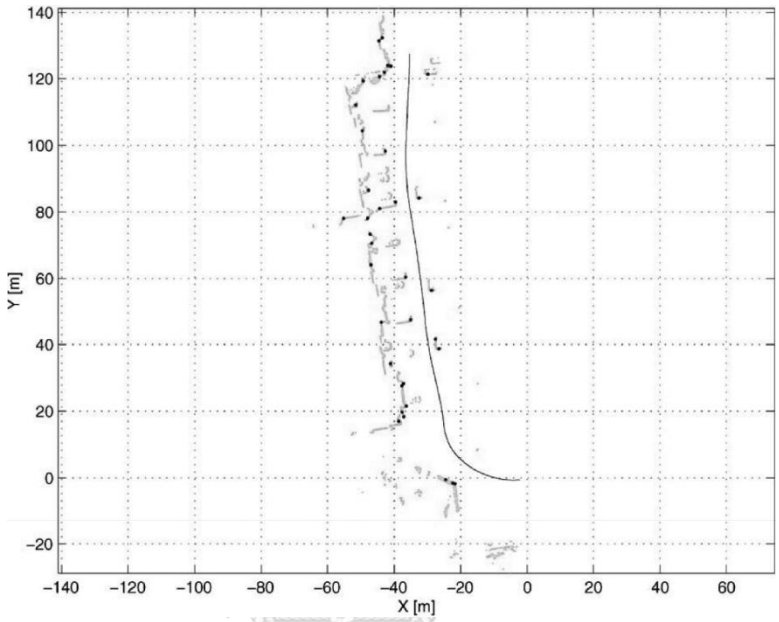


Figure 1 Extracted natural landmark (dominant curvature) and laser rangefinder image[7]

From the last article mentioned, several landmarks, more than one, are required to identify a position in the matching process. However, instead of detecting several landmarks, one can perform a modified approach by detecting only single landmarks but twice by different positions. According to Bais et al. in “Single landmark based self-localization of mobile robots”[8], a robot’s position is determined by range measurement of a single landmark in 2 different arbitrary positions. The displacement between these 2 positions is assumed to be known exactly, by adopting a dead-reckoning measurement, and be used along with 2 range measurements of this landmark from 2 different positions, determined by stereo vision approach, in geometrical approach, namely triangulation, to estimate the position relative to this landmark. Furthermore, vision-based measurement, i.e. color

transition, line detection, is used in this research to extract landmarks, e.g., color transition, corner, line intersection, and junction.

1.1.2. Artificial Landmark-based Localization

Several artifacts are utilized as artificial landmarks for landmark-based localization, most of them are vision-based landmarks, and however, different approaches are presented in some article. In vision-based landmark research, various types of machine-readable codes are mentioned. According to Kartashov et al. in "Fast artificial landmark detection for indoor mobile robots"[9], QR code is employed as an artificial landmark. Their work concentrates on the implementation of an additional color plate to a plane QR code. The additional part consists of 4 different color regions, and by selecting an appropriate color, the contour of the QR code panel is detected and QR code information is identified. Figure 2 illustrates their designed artificial landmark and Figure 3 depicts the result.

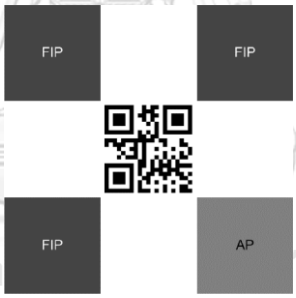
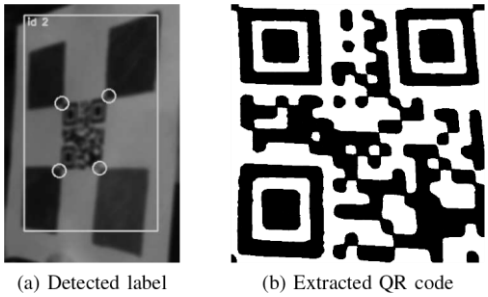


Figure 2 Designed landmark[9]



(a) Detected label (b) Extracted QR code

Figure 3 a) Detected label b) extracted QR code

Another vision-based landmark technique proposed by Salahuddin et al., "An efficient artificial landmark-based system for indoor and outdoor identification and

localization"[10] can be used for the outdoor environment. A landmark label mainly depends merely on the encoded color pattern, however, more than one pattern in each landmark label is suggested. According to this article, this approach can also be used in determining the distance between vehicle, or sometimes called headway distance, by attaching this label, containing more than a single pattern, to preceding vehicle. Figure 4 depicts the examples of the proposed landmark label for license plate and road sign.

Figure 4 Sample artifact landmarks

Apart from the vision-based artificial landmark, a technique that is frequently mentioned in several literatures is the Radio Frequency Identification (RFID)-based localization. One research on this type of landmark, "A RFID Landmark Navigation Auxiliary System"[11], illustrate the potential of RFID to be used instead of vision-based or other natural artificial landmark localization. According to this article, 7 RFID tags are organized in the hexagon array form, and by this configuration with knowledge of antenna detection radius, tag array dimension, body speed and time duration of tag detection, the position including orientation of vehicle can be identified.

According to the "Machine learning approach to self-localization of mobile robots using RFID tag"[12] by Senta et al., a different approach from the previously mentioned article for RFID landmark localization is proposed. Instead of determining position and orientation by solving the kinematic or geometric problem, this research applying the machine learning approach, namely the support vector machine (SVM), to avoid some difficulties, e.g., define every tag's position, complex kinematic problem.

1.2. Localization Base on Global Navigation Satellite System

Global Navigation Satellite System (GNSS), nowadays, receives great attention from any system developers who involve in determining a global position. In the early of global positioning by using earth-orbiting satellite, the Doppler shift technique was used in determining global position, the American Navy Navigation Satellite System (NNSS), or TRANSIT, is an example for this technology. However, TRANSIT shows the main disadvantage, i.e. lack of accuracy and its complexity[13], Global Positioning System (GPS), the American GNSS, are thereby developed to replace the TRANSIT.

Although GPS, and other equivalent systems, other GNSS, such as GLONALL, Galileo, etc., are widely acceptable, complexity and limitation still occur in determining the position. According to Zhu et al. in "GNSS Position Integrity in Urban Environments: A Review of Literature"[14], GNSS application in the urban environment may emerge from signal reception. In an urban environment, lots of obstacles, e.g., trees, buildings, etc., may cause a signal to be distorted or attenuated or sometimes these obstacles even totally block the entire signal to the receiver, even though the GNSS was designed to provide at least 4 satellite signals anytime for receiver anywhere on earth. Figure 5 depicts 2 different phenomena that cause complexity for GNSS in the urban environment, i.e. multipath interference and Non-Line of Sight (NLOS) phenomena.

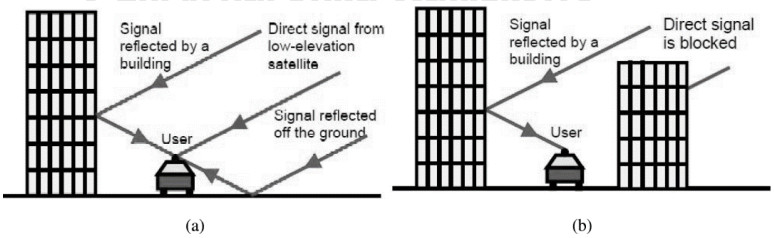


Figure 5 a) Multipath interference b) NLOS reception[14]

Apart from the terrestrial object, the ionosphere and atmosphere environment can cause the signal to undergo perturbation. From these phenomena, positioning accuracy of about 2 to 4 meters may be determined. One method, that currently is given great attention, used to reduce this deviation is the Real-Time Kinematic (RTK)

approach. Several low-cost commercial GNSS devices currently equipped with this technology. According to Jackson et al., in “A performance assessment of low-cost RTK GNSS receivers”[15], 5 low-cost (lower than 500 USD), i.e. Piksi Multi, NV08C-RTK, Reach, NEO-M8P and S2525F8-RTK, and 1 relatively expensive (more than 1,000 USD), i.e. Eclipse P307, receivers are taken to be investigated in performance. Five types of standard performance metrics are used as criteria for performance comparison, i.e. accuracy, continuity, availability, and time to first fix metric type. Metrics and their summary descriptions are given in **Table 3**.

Table 3 Summary of the performance metrics[15]

Metric Type	Metric	Description	Units
Accuracy	RTK Fixed-Integer Accuracy	Value of the difference between RTK fixed-integer horizontal position solution and the truth position.	meters
Continuity	Loss of Locks per Minute	The likelihood of losing an RTK fixed-integer position solution. Presented as loss of RTK locks per RTK minute.	number of losses per minute
	Reacquisition Time	The amount of time it took a receiver to reacquire an RTK lock after it was lost.	seconds
Availability	Fixed-Integer Availability	The total amount of time as a percentage of testing time that a receiver reported an RTK fixed-integer solution.	% of time
	RTK Availability	The total amount of time as a percentage of testing time that a receiver reported an RTK fixed-integer or floating-point solution.	% of time
Time to First Fix	Time to First RTK Fix	The amount of time that it took a receiver to report its initial RTK fixed-integer solution.	seconds

This research reveals that all the 6 receivers reached a centimeter-level accuracy up to the 95th percentile of the measurement. Another interesting result is that the performance evaluated also depends on the antenna type, i.e. rover or patch antenna.

1.3. Summary of Localization Technique

The different techniques in localizations undergo different drawbacks. In landmark-based localization, for natural landmark case, the introduced article[7] suggests that a high dynamic environment should not be used as a resource for natural landmark extraction. Since our research objectives concentrate on autonomous driving which test experiment intends to perform in a considerably dynamic environment. Then natural landmarks may not be totally suitable for this research. However, some techniques, such as maxima curvature extraction by laser rangefinder, could be applied to our system.

Another research in natural landmark-based localization presented[8] tries to utilize only a single detected landmark to determine the robot position. However, the

assumption they used, i.e. the exact relative position is known by a dead-reckoning estimation, is mentioned by several literatures[11, 16] that will gradually encounter the commutative error along the navigation path. **Figure 6** and **Figure 7** depict that commutative error growth along the distance traveled.

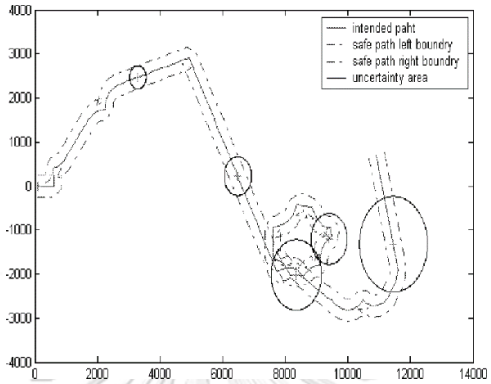


Figure 6 Uncertainty area from odometry reading only[11]

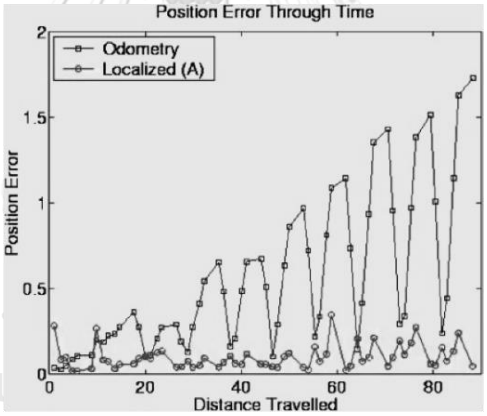


Figure 7 Experimental results from odometry and localization[16]

For vision based-landmark localization, most complexities come from environmental factors, such as ambient light. However, difficulties may emerge from the detection algorithm itself that required sophistication and robustness to effectively detect and extract information from any pattern label[9, 10].

GNSS can also provide a very precise location, however, further technique, e.g., Differential Global Positioning Systems (DGPS), Real-Time Kinematic (RTK), need to be applied to achieve a high accuracy positioning. Although high precision localization can be achieved by GNSS, as performance evaluation shown in the article

above[15], GNSS still cannot be used as a single localization approach due to lack of continuity.

2. Path Follower Autonomous

Path follower robot can be examined as basic for a more sophisticated autonomous driving system. At first glance, several path follower robots utilized numbers of the proximity sensor in determining their position. With a simple hysteresis controller, many of them perform an application acceptable result[17, 18]. Apart from a simple hysteresis controller, several more sophisticated controllers are proposed, e.g., legendary Proportional Integral Derivative (PID) controller. According to A. Al Arabi et al. in "Autonomous Rover Navigation Using GPS Based Path Planning"[19], the autonomous rover utilized the PID controller by selecting the path deviation distance derived from the GNSS positioning system as a controlled parameter, the result shows that the autonomous rover properly follows the predefined path. Furthermore, from M. Engin et al. in "Path Planning of Line Follower Robot"[20], the PID controller shows better results than the simple hysteresis controller in both maximum velocity used and tendency to astray from a predefined path.

However, a complicated autonomous driving application required more control techniques to achieve a satisfactory performance, furthermore, several real situation incidents need to be taken into consideration, e.g., the appearance of an unpredicted obstacle. Consequently, the high-level algorithm for autonomous path planning is utilized.

2.1. The Dynamic Window Approach (DWA)

Besides prescribing a fixed predefined path for an autonomous car to follow, the instantaneous path generating algorithm is popularly adopted by numbers of research, one outstanding algorithm is the Dynamic Window Approach (DWA). According to D. Fox, W. Burgard, and S. Thrun, in "The Dynamic Window Approach to Collision Avoidance"[21], DWA is a local path planner which optimized robot velocities base on its performance, i.e. maximum linear and angular acceleration or deceleration, such that results in the optimal admissible path. For this algorithm, an

admissible local path is calculated by maximizing the cost-like function, called the objective function, considering target heading, clearance to the obstacle, and robot velocity from dynamic window search space.

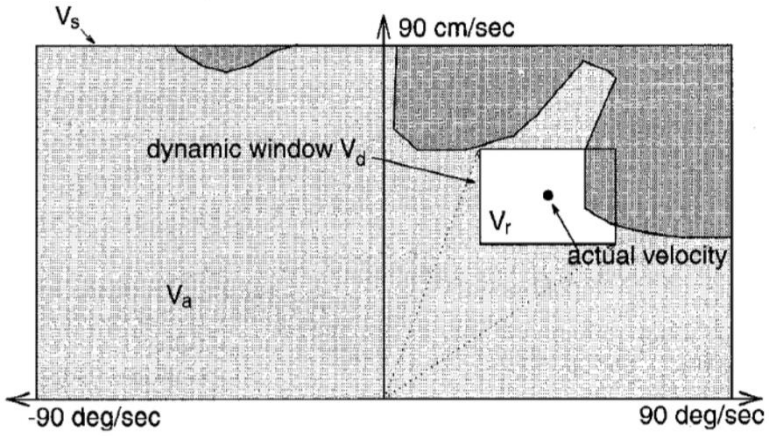


Figure 8 Dynamic window from the Dynamic Window Approach (DWA)[21]

CHAPTER III
RELATED THEORY

1. Global Navigation Satellite System (GNSS)[22]

The Global Navigation Satellite System (GNSS) is a localization system using signal broadcasted Medium Earth Orbiting (MEO) satellites whose altitude are about 20,000 km above the earth's surface. Nowadays, GNSS systems that available for civilian applications are the Global Positioning System (GPS – Unites States), GLObal NAvigation Satellite System (GLONASS – Russia), Galileo (European Union), and BeiDou (China). Among all these GNSS systems, GPS is the most outstanding system that was first developed before other systems, originally for military purposes. Therefore, a general principle of the GPS will be introduced in this section as an example that represents an overview of the GNSS.

1.1. The Global Positioning System (GPS)

By April 2020, the GPS already has 31 satellites in operation and 9 in reserve. Originally, 24 satellites are expected to be a number of the least satellites operated by the GPS. The principle underlying the GPS ability to localize a certain terrestrial object is simply measuring the distances between at least 4 satellites and a receiver attached to such object. Basic components of the GPS consist of 3 parts, i.e. a control station, satellites, and a receiver. The first component, i.e. control stations, located around the world, keeps tracking and monitoring all satellites. Each satellite's ephemeris, a satellite's predicted position and velocity, is updated by these control stations providing a precise localization to the system. Inside each satellite contain a high precision atomic clock, this atomic clock is used to generate 2 GPS carrier waves with different frequencies, i.e. 1575.42 and 1227.60 megahertz, known as L1 and L2, respectively. Each wave is modulated with a stream of bit called a Pseudo Random Noise (PRN) determined by a precise mathematic algorithm. These waves later are broadcasted to the earth and received by a terrestrial receiver. Two methods are available in distance measurement, i.e. a code ranging and a carrier-phase ranging. In a code ranging, The receiver determines a distance toward the satellite by measuring a

time lag between a received signal and a synchronous receiver-satellite generated signal, as shown by Figure 9. The equation relates a time lag and the distance between a receiver and the satellite is given by Equation (1).

$$p = c\tau \tag{1}$$

Where p denotes the distance between a receiver and the satellite, known as the pseudorange. c represents the signal traveling speed and τ is a measured time lag as shown by **Figure 9**.

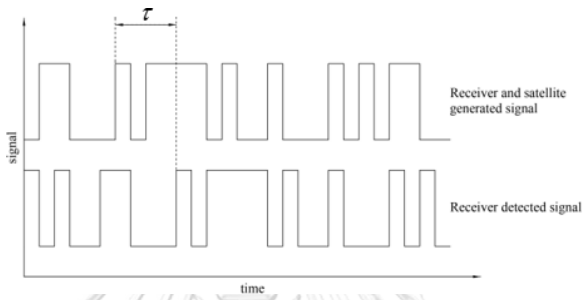


Figure 9 Time lag determined from GPS signal

The GPS signal carrier wave's phase is measure in a carrier-phase ranging method providing a higher precision than what obtained by a code ranging method. However, the carrier-phase ranging only determines the fractional phase part of the total pseudorange, i.e. $\phi\lambda$ in **Figure 10**, leading to the unknown number of complete wavelengths N, known as an integer ambiguity. To determine this integer ambiguity, the code ranging along with more receiver is utilized by a certain technique called a double differencing. Equation (2) shows the relationship of parameters in a carrier-phase ranging method.

$$p = (N + \phi)\lambda \tag{2}$$

Where p, N, ϕ, and λ denote a pseudorange, integer ambiguity, measured phase, and the carrier signal's wavelength.

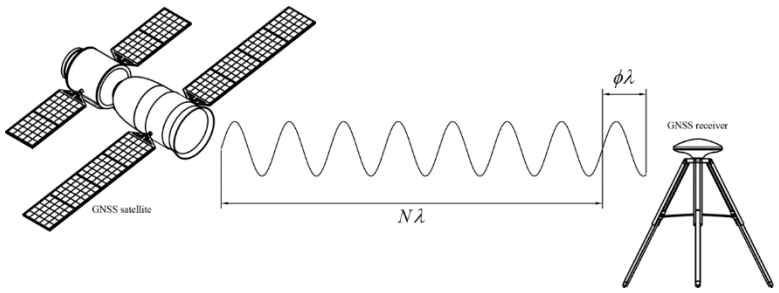


Figure 10 Phase-carrier ranging

1.2. Source of GNSS Measurement's Error

The error in measuring a pseudorange of the GNSS may emerge in different levels of the system. In the satellite level, an inaccurate ephemeris of the satellite may be broadcasted to the geodesic receiver. Therefore, the receiver will output the false position. This sort of error may be caused by insufficient monitoring by the ground station. Also, gravitational attraction by other planets, moon, or sun, and the solar radiation pressure can deviate the actual satellite position away from the prediction, i.e. an ephemeris. While traveling from the satellite to a receiver, the GNSS signal's speed is distorted along the way through the earth's atmosphere. According to Equation (1), since the signal traveling speed changes, then the pseudorange determined by using a speed of light will be invalid. This error in an atmosphere level occurs in both the ionosphere and troposphere layer. In these layers, charged and neutral particles contribute to a change in the traveling velocity of a signal. Moreover, the indirect path of the signal may occur by reflecting any terrestrial objects before reaching the receiver's antenna, this kind of error is called a multipath error.

In code ranging, the measured time lag plays a major role in determining a pseudorange. Therefore, the precise pseudorange must be determined by a precise time measurement. One major problem existed in the receiver level is a clock error. However, a clock error also happens in a satellite level but compared to the receiver level, an error in a receiver clock results in more severe to the measured pseudorange.

1.3. Pseudorange Equations

An unknown position point in 3-dimensional space can be determined by using three distances between that unknown position point and the other three

reference points whose position is exactly known. However, as previously mentioned, the measured pseudorange can be deviated from the actual distance toward a satellite by the clock error. Therefore, the relation between a pseudorange and the actual distance toward a satellite can be stated by Equation (3).

$$p = \rho + \varepsilon \tag{3}$$

Where p is the pseudorange, ρ is the actual distance toward a satellite, and ε is the distance error due to a clock error. Since the position of the satellite is assumed to be known exactly from the satellite's ephemeris. Then, Equation (3) can be rewritten to Equation (4).

$$p = \sqrt{(x - x_r)^2 + (y - y_r)^2 + (z - z_r)^2} + c\delta \tag{4}$$

Where x, y, and z denote the position of a satellite in a cartesian coordinate system. x_r, y_r, and z_r represent the position of a receiver in a cartesian coordinate system. c is the carrier wave traveling speed and δ is the clock error time. Since the clock variation among satellites is negligible compare to the time difference between a satellite and a receiver. Then the clock error time is considered to be equal for every pseudorange measured by one certain receiver. Also, by assuming that the carrier wave traveling speed is known, the pseudorange equations constructed as shown by Equation (5).

$$\begin{aligned}
p_1 &= \sqrt{(x_1 - x_r)^2 + (y_1 - y_r)^2 + (z_1 - z_r)^2} + c\delta \\
p_2 &= \sqrt{(x_2 - x_r)^2 + (y_2 - y_r)^2 + (z_2 - z_r)^2} + c\delta \\
p_3 &= \sqrt{(x_3 - x_r)^2 + (y_3 - y_r)^2 + (z_3 - z_r)^2} + c\delta \\
p_4 &= \sqrt{(x_4 - x_r)^2 + (y_4 - y_r)^2 + (z_4 - z_r)^2} + c\delta
\end{aligned} \tag{5}$$

Where x_i, y_i, and z_i denote the position of the i satellite in a cartesian coordinate system, and p_i is the pseudorange of the i satellite. Since 4 unknowns with 4 equations appear in Equation (5), then the position of the receiver can be solved using the measure pseudorange from 4 satellites. Moreover, the result has included the clock error effect. Figure 11 illustrates the parameters used in the pseudorange equation.

It can be shown that if the receiver is operated in a static positioning mode, i.e. the receiver is held in place measuring a static position, one can utilize only 2 satellites in pseudorange measurement at a time, however, the measurement process has to be repeated at least 3 times. Whereas in kinematic positioning, i.e. the receiver is moving while receiving the GNSS signal, at least 4 satellites are required for each measurement.

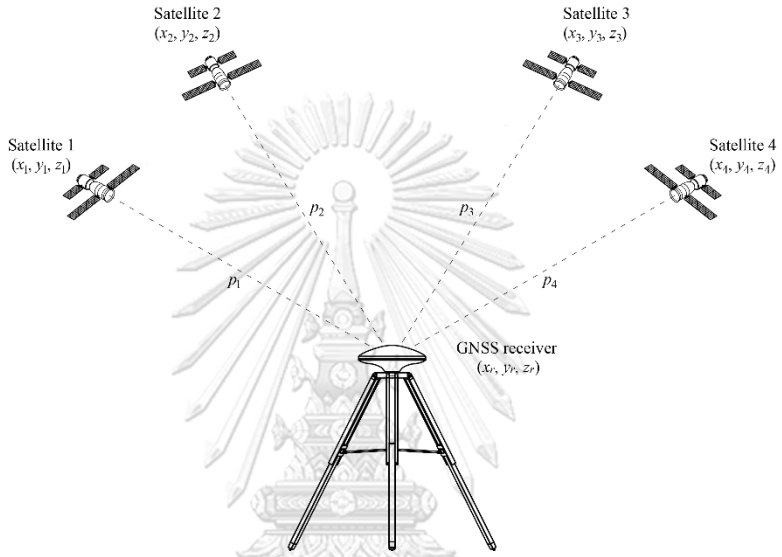


Figure 11 Parameters used in the pseudorange equation.

1.4. Differential Global Positioning System (DGPS)

The accuracy of the GNSS positioning can be improved by introducing the second receiver. This second receiver is employed as a base station, i.e. base receiver, of which the exact position is assumed to be known. The Differential Global Positioning System (DGPS) determines the error in pseudorange according to Equation (6), assuming the position of both the satellite and a base receiver is known exactly. The correction obtained from this equation is called a Pseudorange Correction (PRC).

$$\Delta p_i = \sqrt{\left(x_i - x_b\right)^2 + \left(y_i - y_b\right)^2 + \left(z_i - z_b\right)^2} - p_{i1} \tag{6}$$

Where Δp_i represents the Pseudorange Correction (PRC) for the i satellite. x_i, y_i, and z_i denote the position of the i satellite in a cartesian coordinate system. x_b, y_b,

and z_b denote an exactly known position of the base receiver and p_{i1} is the pseudorange of the i satellite measured by the base receiver.

However, if the distance between the base receiver and the rover receiver, i.e. another receiver that the position is needed to be found, is not too long, the PRC can be applied to the corresponding pseudorange measured by the rover receiver since the atmosphere for 2 areas in close proximity is considered to be the same, hence, leading to identical pseudorange corrections. Therefore, the corrected pseudorange of the rover receiver is given by Equation (7).

$$\tilde{p}_{i2} = p_{i2} + \Delta p_i \tag{7}$$

Where $\tilde{p}_{i2}$ denotes the corrected pseudorange of the i satellite measured by a rover receiver. p_{i2} is the original pseudorange of the i satellite measured by a rover receiver. **Figure 12** depicts the DGPS configuration and explains the parameters used in the above equations.

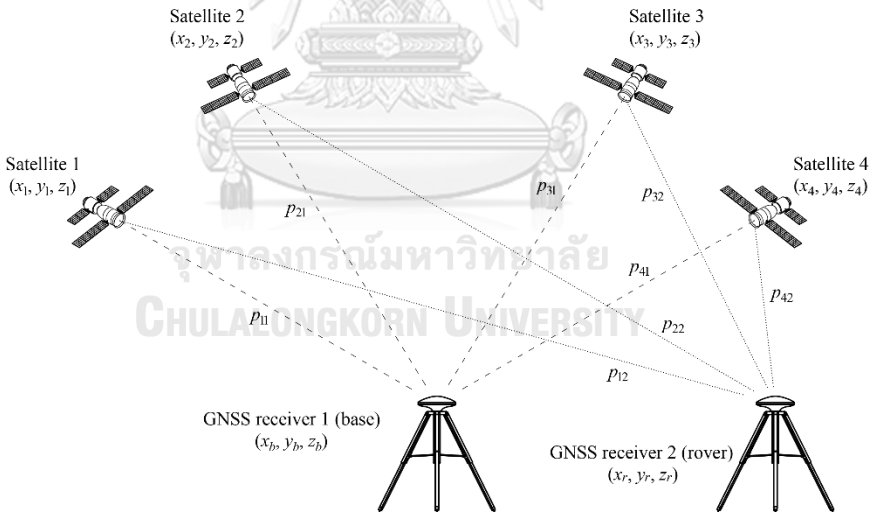


Figure 12 The Differential Global Positioning System (DGPS) configuration

The pseudorange used in determining the PRC can be determined by the code ranging or the carrier-phase ranging method. By utilizing the code-pseudorange, one can obtain the accuracy down to the decimeter level in real-time application. However, to enhance more accuracy, one famous algorithm named Real-Time

Kinematic (RTK) applies the carrier-phase pseudorange to the DGPS fundamentals, resulting in an accuracy of about 2 to 5 centimeters.

1.5. Relative Positioning

Instead of correcting the pseudorange measured by receivers that are in close proximity, the relative positioning method determines the relative position using the raw pseudorange from both the base receiver and the rover receiver. Then, similar to the DGPS, the assumed known exact position of the base receiver is added to the relative position, giving a final position. However, this method is intended to be used in post-process application in which the carrier-phase pseudorange is performed. Equation (8) describes the mathematic form of the relative positioning method.

$$\tilde{\mathbf{x}}_r = (\mathbf{x}_r - \mathbf{x}_b) + \mathbf{x}_0 \tag{8}$$

Where $\tilde{\mathbf{x}}_r$ is the corrected rover position by the relative positioning method. $\mathbf{x}_r$ and $\mathbf{x}_b$ denote the position of the rover and base receiver, respectively, determined by Equation (5) using the original pseudorange. $\mathbf{x}_0$ is the assumed known exact position of the base receiver.

2. Proportional-Integral-Derivative Controller

A classical feedback controller block diagram is shown in **Figure 13**. The dynamic error is controlled and minimized by this feedback configuration. Depending on the plant transfer function, system disturbance, and the reference signal, different types of controllers can be applied results in the different dynamic responses of the controlled parameters. Normally, the sensor transfer function is neglect and assumed to be unity, thus the plant output y then equals the sensor output y'.

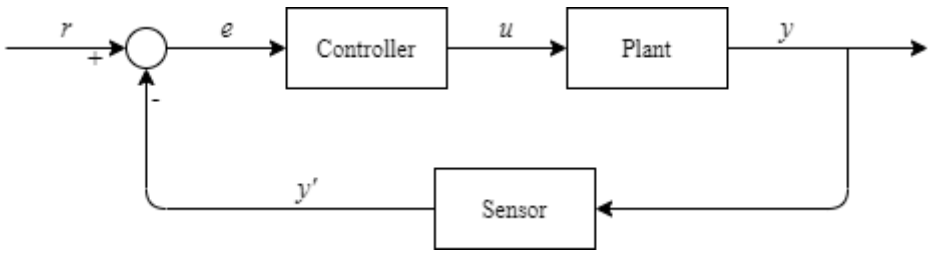


Figure 13 Classical feedback controller block diagram

The well-known feedback controller used for the low-level controller system in this research is the Proportional-Integral-Derivative controller, as known as the PID controller. The general control equation form of such controller is shown by Equation (9).

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau)\, d\tau + K_d \frac{d}{dt} e(t) \tag{9}$$

Where K_p, K_i, and K_d are the proportional, integral, and derivative coefficient, respectively, all coefficients are non-negative value. e denotes the dynamic error defined as a difference between the reference signal and the sensor output, stated mathematically by Equation (10). u represents the controller output.

$$e = r - y' = r - y \tag{10}$$

The PID controller can be separated and utilized as a combination of the individual controller. Typical combinations that always used are a proportional-integral (PI) controller and a proportional-derivative (PD) controller, depending on the characteristic of the controlled system. However, each controller has its own remarkable response to different types of dynamic errors.

2.1. Proportional Controller

The proportional controller, described by Equation (11), gives the output proportional to the dynamic error without imposing any dynamic to the output. This controller is a basic for every combination to be included. The proportional controller impacts the early portion of the response from the feedback controller by shortening a response's rise time whenever the proportional coefficient is increased and vice versa. However, increasing the coefficient also leads to an increase in response overshoot and oscillation as shown in **Figure 14**. Also, using only the proportional controller will not guarantee the zero steady-state error.

$$u(t) = K_p e(t) \tag{11}$$

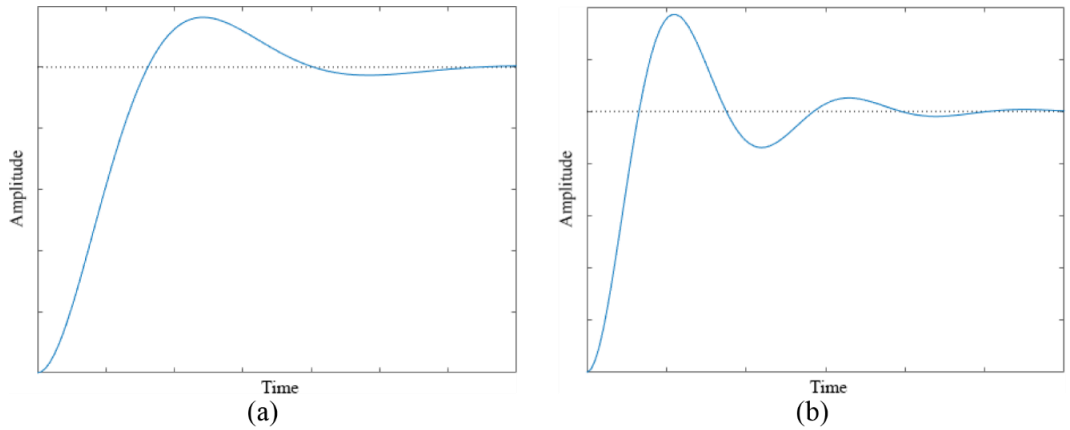


Figure 14 (a) Low proportional coefficient response (b) High proportional coefficient response

2.2. Integral Controller

As state earlier, using the proportional controller only may result in a non-zero steady-state error. This constant steady-state error may emerge from the system plant which doesn't possess any integrator of the number of integrators is not enough to cope with a certain reference signal. Also, the mechanical defect can cause the steady-state error. The integral controller, shown by Equation (12), can manage to eliminate this type of error. According to Equation (12), the integral controller can output the non-zero control signal even when the error is zero, which in case of the proportional controller will give a zero output. However, increasing the integral coefficient too high can cause a response to be unstable.

$$u(t) = K_i \int_{t_0}^{t} e(\tau)\, d\tau \tag{12}$$

2.3. Derivative Controller

The derivative controller, shown by Equation (13), resists the rapid change in a dynamic error. Considering the proportional controller with a high proportional coefficient, the response of such a controller will encounter a large overshoot and sometimes turn to constantly oscillate. The derivative controller can deal with this response behavior by decrease the other controller's output whenever the error changes too quickly.

$$u(t) = K_d \frac{d}{dt} e(t) \tag{13}$$

The result response from this derivative controller can be classified into 3 types, i.e. underdamped, critically damped, and overdamped response, illustrated by **Figure 15**. The low derivative coefficient may lead to the underdamped response which still possesses an oscillating behavior. In an overdamped case, caused by a high derivative coefficient, time takes until reaching the steady state will be long, however, without oscillating behavior. The correct derivative coefficient gives a non-oscillating response by using the least time to reach a steady state.

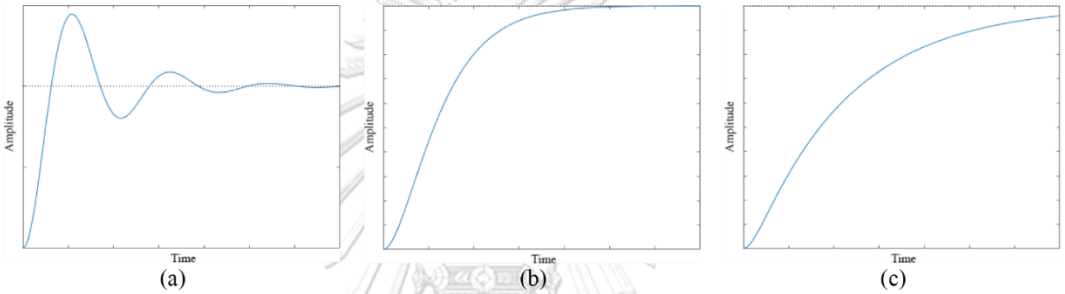


Figure 15 (a) Underdamped response (b) Critically damped response (c) Overdamped response

2.4. Implementation

Instead of using an analog device, the digital implementation of the PID controller is deployed in this research. The continuous PID controller Equation (9) is digitalized to Equation (14) using the rectangular approximation. However, the digitalized version of the PID controller may develop an overshoot behavior in the dynamic response. Thus, the sample period of the digitized PID controller used in this research will keep at a relatively low, about 5 milliseconds, such that the digitized version can imitate the continuous PID controller.

$$u(kT_s) = K_p e(kT_s) + K_i \sum_{n=0}^{k} e(nT_s) + K_d \left\{ e(kT_s) - e(kT_s - T_s) \right\} \tag{14}$$

Where $k \in I^+$ denotes the sampling number.

CHAPTER IV
DEVELOPMENT OF AUTONOMOUS DRIVING SYSTEM

1. Low-level Control System

The first step in creating the autonomous driving system is to develop a low-level system. In this research, an electric vehicle available in the market, i.e. the TOYOTA COMS, is modified by introducing the steering control and speed control system to the original car. The detail in vehicle modification is described below.

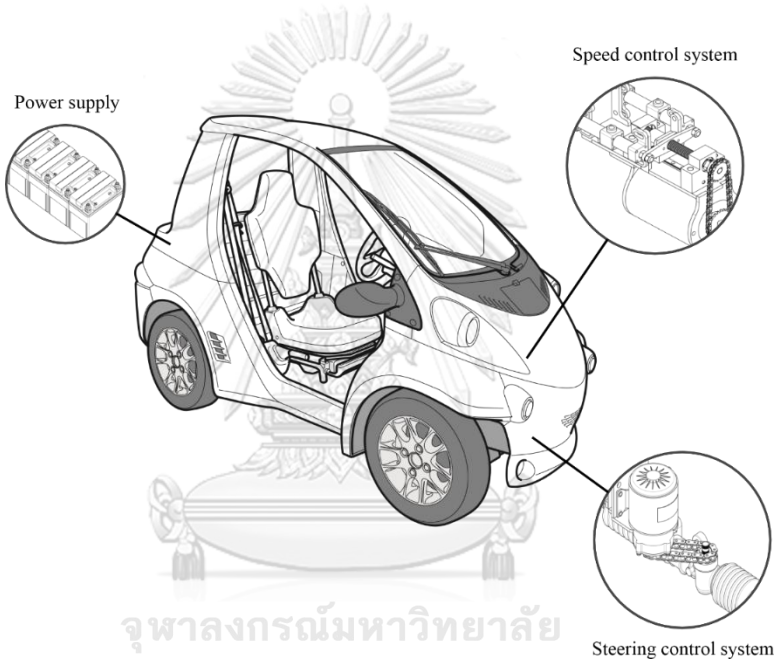


Figure 16 The TOYOTA COMS with modified low-level systems

1.1. Power Supply System Modification

Additional modified systems are supplied by a set of batteries that are separated from the car's original batteries. These batteries power the steering control system, speed control system, and sensors used in autonomous navigation, i.e. a GNSS receiver and a laser scanner. As shown by **Figure 17**, a set of 4 12V-45Ah LiFePO4 batteries is divided into 2 separate supply circuits. Two batteries in the left are connected in series producing a supply voltage of 24 volts for high current drawing devices, including a brake motor and a steering motor. Two batteries on the right are also connected in series producing a 24 volts supply for electronic devices

that consume less current, e.g. a GNSS receiver, laser scanner, control circuits, etc. when the 220V power is supplied to the charge controller, all batteries are disconnected from devices they supplied and parallelly charged by the charge controller.

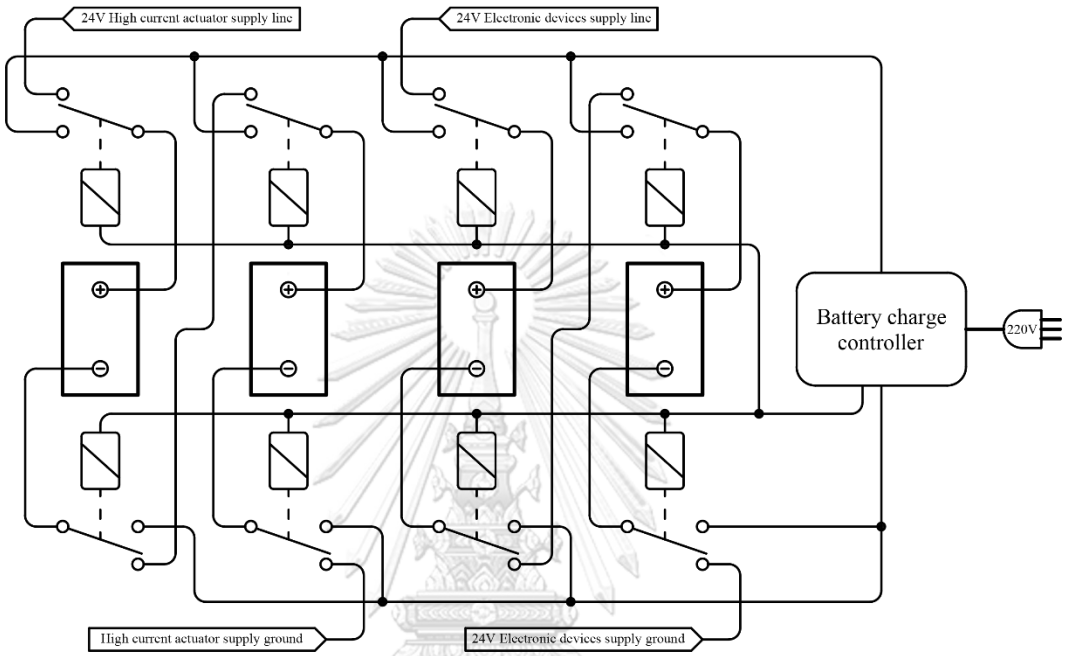


Figure 17 Power supply system's configuration

1.2. Electronic System Modification

Motor rotational direction, turn signal lights, a hazard light, a wiper system, etc. are controlled using an electronic signal. Thus, to convert a vehicle into an autonomously controlled car, a modification on the electronics system needed to be made.

1.2.1. Modified Switch Circuits

Figure 18 illustrates the component diagram of the modified electronics system. Formerly, the control signal from switches is connected to the Electronic Control Unit (ECU) directly. However, the modified system utilizes selector circuits that switch the control signal between the microcontroller and mechanical switch outputs so that the car can be controlled manually and automatically. These 3 selector circuits are employed for 3 switch circuits, i.e., a shifter switch, signal lights switch,

and wiper switch circuit. The 16-Relay module receives the control signal from selector circuits and outputs the final control signal to the ECU.

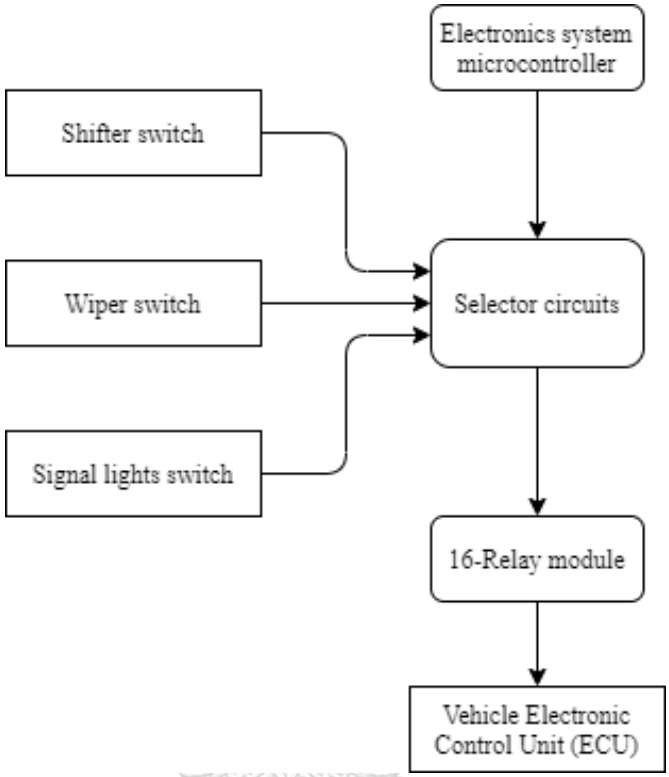


Figure 18 Electronics system components diagram

The modified circuit of a shifter, signal lights, and wiper control switch is shown by Figure 19, Figure 20, and Figure 21, respectively. These modified circuits switch the ECU input signal using signal selectors, i.e. 2-to-1 digital multiplexers. Note that all multiplexers' selector input is connected together and controlled by the electronics system microcontroller. Also, all mechanical relays appear in these figures belong to the 16-relay module.

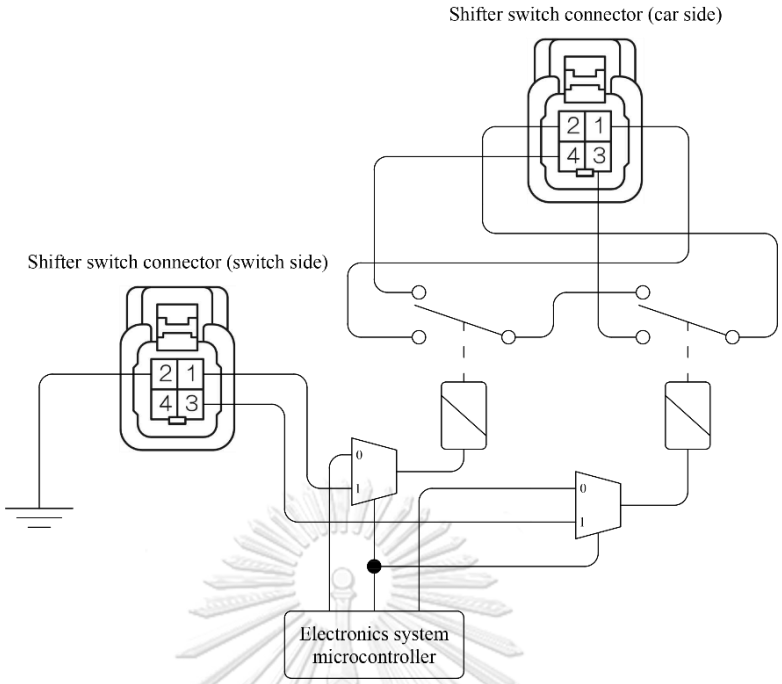


Figure 19 Modified shifter switch circuit

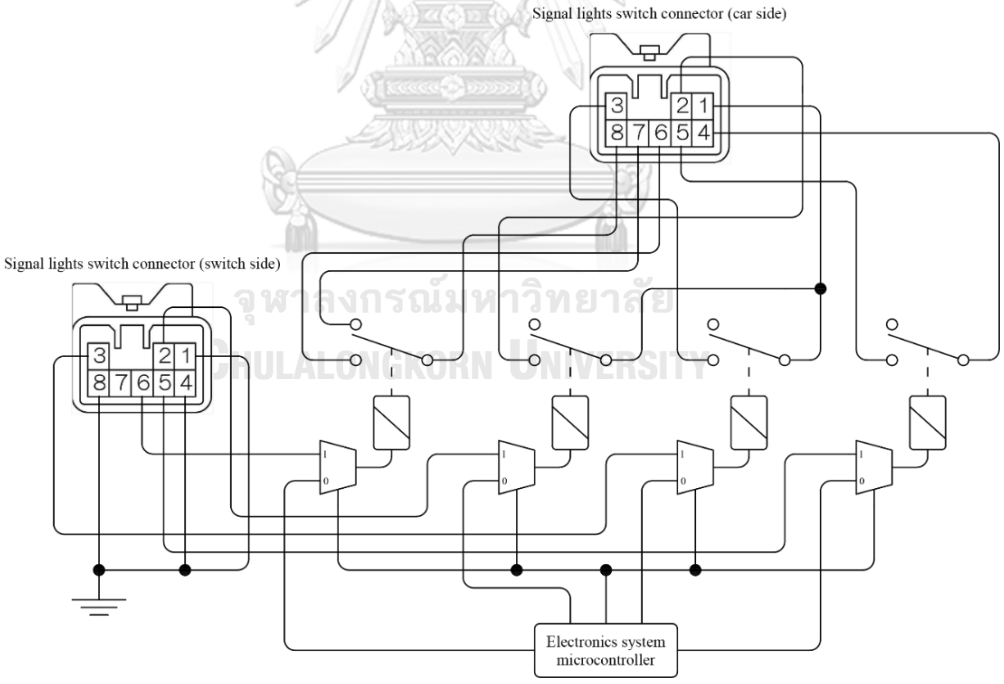


Figure 20 Modified signal lights switch circuit

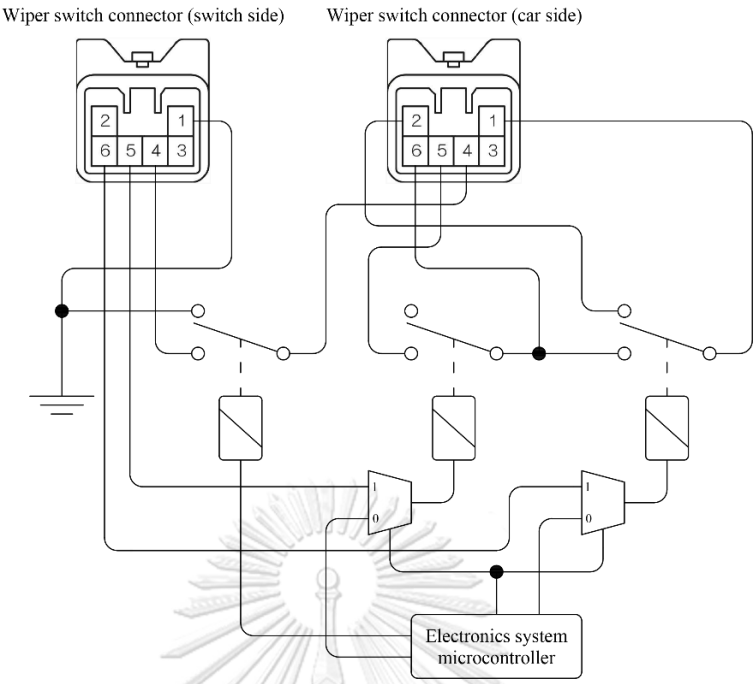


Figure 21 Modified wiper switch circuit

The modified system is installed by replacing the direct connection between car and switches with modified circuits in between. The obvious advantage of this configuration is that the car can be converted back to the original circuit system easily by removing the modified circuit connector and connect the switch connector back to the corresponding car side connector.

1.2.2. Microcontroller software

Figure 22 describes the workflow of the electronics system controller software. Firstly, a microcontroller starts an initialization routine. This includes binding the relay module to output ports and initializing the corresponding initial state to these ports. Then, a loop routine is entered starting by looking for the incoming command message from the high-level controller. If the command message is received and verified to be valid, then the controller will execute the instruction according to the received command message. Finally, the new loop routine begins repeatedly.

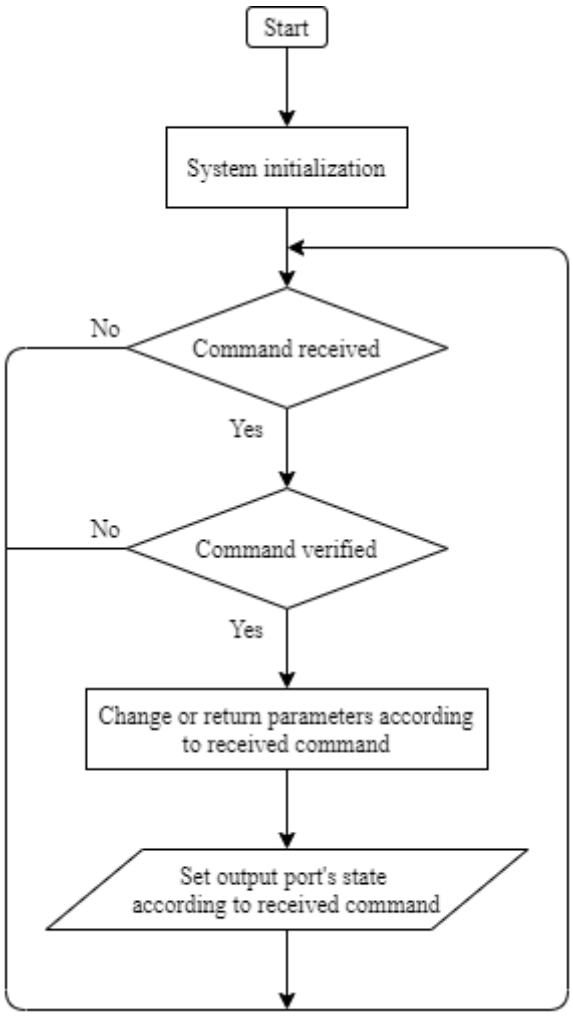


Figure 22 Electronics system controller software's flowchart

1.3. Steering Control System

In modifying the steering control of the car to be used for the autonomous navigation system, three main subsystems must be installed, i.e. mechanical actuator system, electronics system, and the low-level steering control system. These subsystems are explained below.

1.3.1. Mechanical Actuator System

The car used in this research originally came without a steering assistant system, e.g. power steering system. Thus, the steering wheel of a car cannot be controlled using the Controller Area Network (CAN) protocol communication. To control a steering wheel, hence steering angle, an electric motor is installed as shown

in **Figure 23**. The motor transmits power through a speed reduction gearbox of a 1:9.78 gear ratio, and then by a chain-sprockets system with a sprocket ratio of 1:1 to a steering rack.

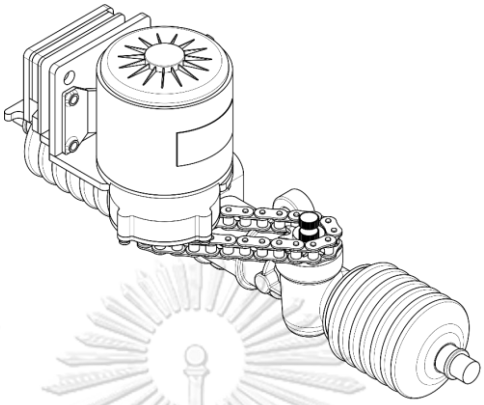


Figure 23 Steering control actuator system

To control the position of a steering wheel, the sensor for measuring the steering wheel position is required. Here, the sensor is chosen to be a 10-round audio potentiometer with a total resistance of 10 kiloohms. Since the steering wheel of the car can be turned for about 4 rounds, lock to lock, then a gear set that increases a turning round of the potentiometer should be applied so that a full measurement range of 10 rounds can be properly utilized. For this reason, a gear set of 4:9 gear ratio is installed. A 1 round excess is introduced in case the steering wheel undergoes an overturn resulted from unexpected situations. The steering wheel position sensor is attached to a steering column as shown by **Figure 24**.

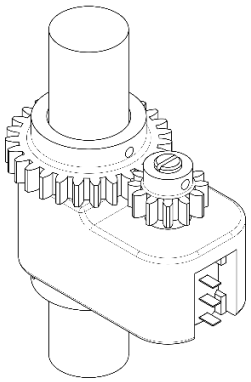


Figure 24 Steering wheel position sensor

1.3.2. Electronics System

A DC motor is driven by the H-bridge DC motor driver which is controlled by a microcontroller unit of the low-level steering controlled system. Power supplied to the motor has a voltage of 24 volts but limited by a pulse width modulation of 50 percent for safety reasons. The steering wheel position sensor, i.e. a potentiometer, is connected to an Analog to Digital Converter (ADC) which provides a digitalized steering wheel position to the microcontroller unit. An emergency stop switch and a circuit breaker are installed for the user to manually disconnects the motor and the supply battery, respectively. The electronic components diagram is shown by **Figure 25**.

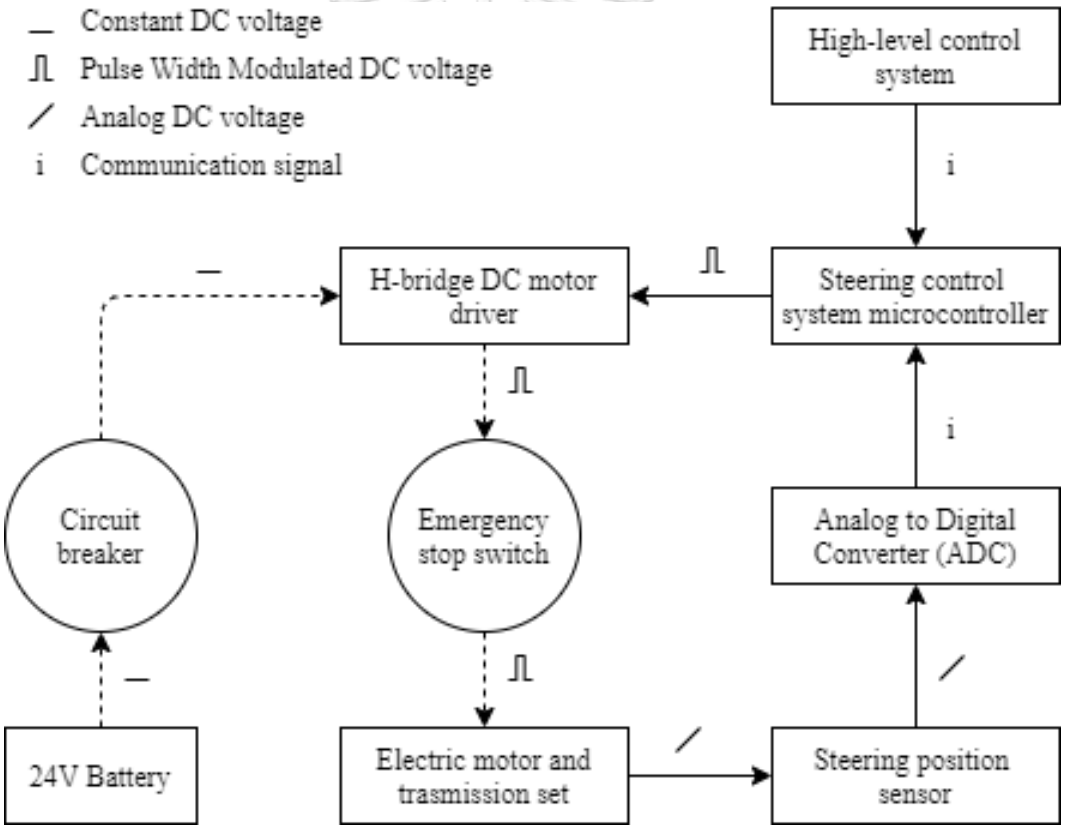


Figure 25 Steering control system's electronic components diagram

Also, note that the voltage level of links between each component is represented by a corresponding line type, i.e. a dashed line represents a 24-volt power line and a solid line represents a low voltage signal, typically a 3.3 volts COMS logic level communication line.

1.3.3. Low-level Steering Control System

A 1-dimensional Proportional Integral Derivative (PID) controller model is utilized in a low-level steering control system. By receiving a required steering wheel position, i.e. a reference signal, from the high-level control system, the controller then determines the output signal which, for this system, is defined as a duty cycle of the pulse width modulated 24 volts power supply, according to PID control law using a set of predefined tuned gain coefficient. Note that a unity gain transfer function assumption is applied to the steering wheel position sensor. A control bock diagram of the steering control system block diagram is shown by **Figure 26**.

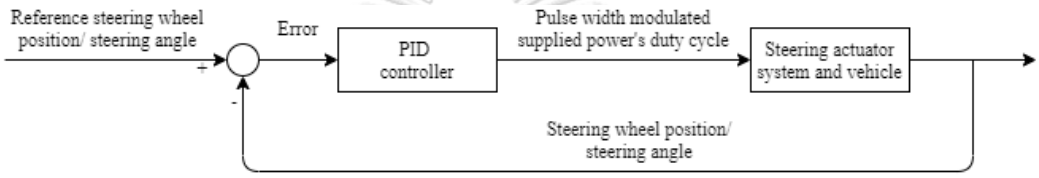


Figure 26 Steering control system block diagram

1.3.4. Microcontroller Software

In order that the low-level steering control system can be controlled or communicates with the high-level control system, a certain interface software need to be applied. This software manipulates the incoming command from a high-level controller and executes a certain routine corresponding to a received command.

The software begins with initializing parameters by reading from the microcontroller's Electrically Erasable Programmable Read-only Memory (EEPROM). The initialized parameters are listed below.

- An initial reference steering wheel position for the automatic control mode
- PID controller gain coefficients, including proportional, integral, and derivative coefficient
- Limited positions of a steering wheel, both minimum and maximum limit position
- A controlled loop time interval

After finish initialization, the current steering wheel position is measured. Subsequently, the microcontroller will check for the incoming command message

from a high-level controller. If the command message is available and verified, the microcontroller then executes a routine requested by a high-level controller. Then, if the automatic control mode is engaging, the microcontroller will calculate the output duty cycle using the PID control model and output a result to the motor driver. Note that the PID controller used in this system is discretized to be utilized in this system. However, if the controller is not in the automatic control mode, a motor will be released from a motor driver allows the driver to manually control a steering wheel. Next, the microcontroller will broadcast the system parameters if they are required by the high-level system. Eventually, the loop time interval control has proceeded before the new computational loop begins. **Figure 27** illustrates a workflow of the low-level steering control system software.

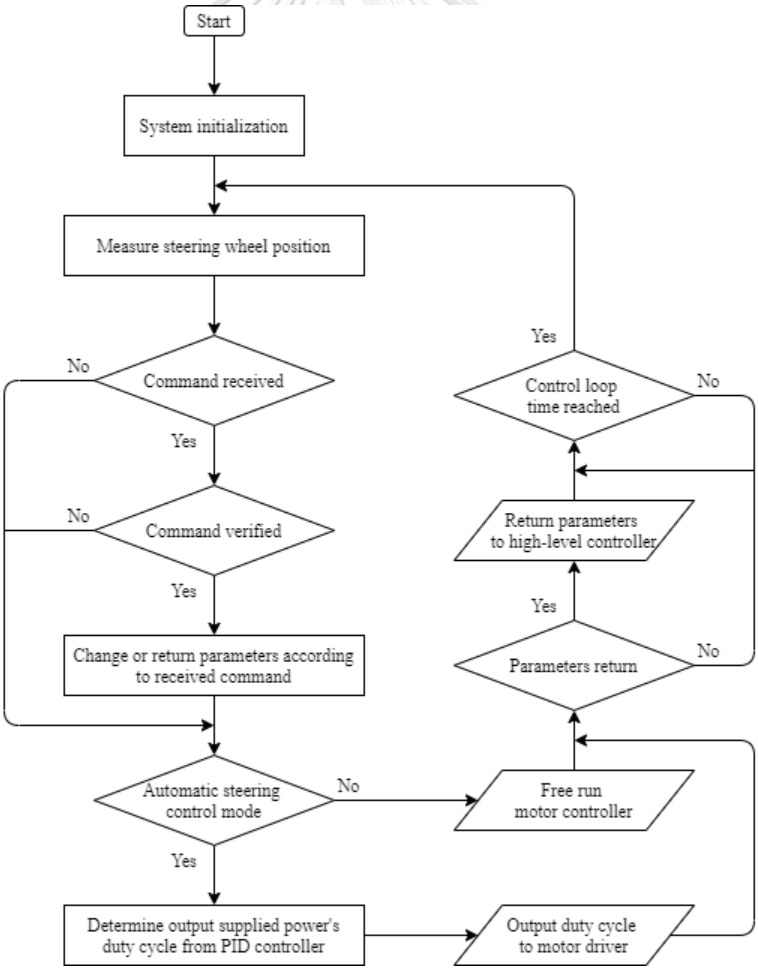


Figure 27 low-level steering control software's flowchart

This control software is embedded in the microcontroller belongs to this system. A communication protocol between this steering controller system and the high-level controller is a serial communication through a Universal Serial Bus (USB) connection.

1.3.5. Steering Units Relationship

Early in this section, the steering wheel position is described by several units. In the beginning, a steering wheel position is mentioned in a geometry degree. Then, the steering position is measured using a potentiometer through a gear set which gives a unit of voltage. Subsequently, this voltage is converted by the 16 bit-analog to digital converter with a service range of ±6.144 volts which return a measured voltage in signed-integer bits ranged from -32768 to 32767 bits.

The steering position unit relationship between an actual steering angle and a potentiometer measured voltage is determined by Equation (15) since a gear set of a 4:9 gear ratio is presented to convert a full range of 4 steering wheel revolutions to 9 turns of a 3.3-volt supplied potentiometer.

$$E = 0.04125\delta + 1.65 \tag{15}$$

Where E and δ denote a measured voltage in volts and the actual steering angle in degrees.

A second relationship is between a measured voltage and the ADC output. Since the ADC range of operation is ±6.144 volts which correspond to the output range from -32768 to 32767 bits. Also, a potentiometer is supplied by a 3.3-volt source. Thus, the relationship between these 2 units can be given by Equation (16).

$$\beta = 5333.33E \tag{16}$$

Where β denotes the ADC output in bits.

By combining Equation (15) and Equation (16) we may obtain the relationship between the actual steering angle and the ADC output as shown by Equation (17). **Figure 28** illustrates the relationship among these units.

$$\beta = 220\delta + 8800 \tag{17}$$

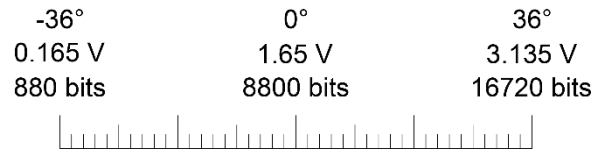


Figure 28 Relationships among different steering position units

1.3.6. PID Controller Output Description

Previously in the microcontroller section, the output of the PID control model is in a duty bit. Here, a duty cycle bit is defined as a duty cycle-like value that corresponds to a 100 percent duty cycle when such value equals 4095 bits. Equation (18) describes the definition of this value.

$$\%\ \text{duty cycle} = \frac{\text{duty cycle bit}}{4095} \times 100 \tag{18}$$

1.3.7. Parameters Tuning

Since the mathematical model of the steering system is not available. Then, the gain coefficient tuning cannot be done by using a mathematical design method. Consequently, the empirical tuning method is employed instead of determining a complex model of the whole system. Later, a satisfactory response is achieved by manually tuning gain coefficients. Equation (19) describes the values of the mentioned coefficients.

$$\begin{aligned} K_p &= 280 \\ K_i &= 150 \\ K_d &= 10 \end{aligned} \tag{19}$$

Where K_p, K_i, and K_d are the proportional, integral, and derivative coefficient, respectively. **Figure 29** depicts an example of a steering control system's step response with the initial position at 11477 bits and a reference position at 12000 bits. Note that tuning is done at a controlled loop time interval of 5 milliseconds and these controller coefficients are intended to be used in a designed microcontroller software only.

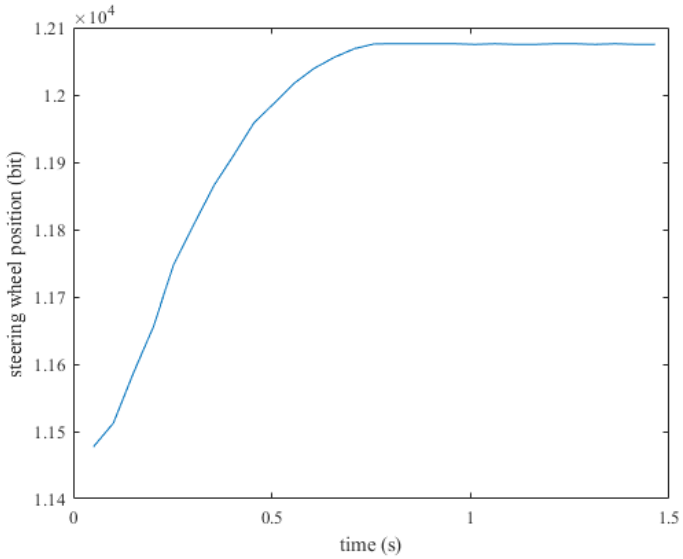


Figure 29 Step response of a steering control system

1.3.8. Response Model

In developing a high-level autonomous navigation algorithm, an important part is to determine a predicted trajectory of the vehicle. In the later section, the steering response model will be required to generate several predicted trajectories. Later, the steering response model will be referred to as a steering profile. A steering profile is a sequence of a steering position at any time instance.

Same as modeling a mathematic model of the steering system, determining an accurate response mathematic model takes many resources to accomplish. Thus, instead of investigating the response for a mathematic model, the direct recording of the steering profile is employed.

In high-level navigation software that will be introduced in the later chapter, the algorithm will determine the suitable steering position periodically. Hence, the steering command signal, i.e. reference steering position, may be considered as a step function of a certain value. Consequently, the steering profile is recorded by executing a step reference steering position of different values to several initial positions. In this research, an equal interval of 500 bits range from 2500 bits to 15500 bits is set to be both the initial position and reference signal. The record time is set to be 10 seconds for each record. Finally, 729 steering profiles of different initial and reference steering position is obtained. This set of steering profiles will be used later in the navigation

section instead of using an exact mathematic model. Another advantage of using recorded steering profile is that the computational effort will be greatly reduced, however, induce loads on the memory. **Figure 29** is also one of the steering profiles in a recorded set.

1.4. Speed Control System

The primary objective of the speed control system is to regulate the vehicle speed in response to the desire speed command received from the high-level controller. Same as the steering control system, a modification for the speed control system is divided into 3 parts, i.e. mechanical actuator system, electronics system, and the low-level speed control system. Furthermore, controlling speed in an automotive application involving 2 actuators system, including acceleration and braking system.

1.4.1. Mechanical Actuator System

Since the existed braking system of the car used in this research is a hydraulic type without a booster. Then, a desire brake pressure cannot be achieved using the electrical intervention. Consequently, the additional brake actuator is designed and installed in the original system. This additional brake actuator was designed such that the brake pressure can be applied by both user and low-level controller.

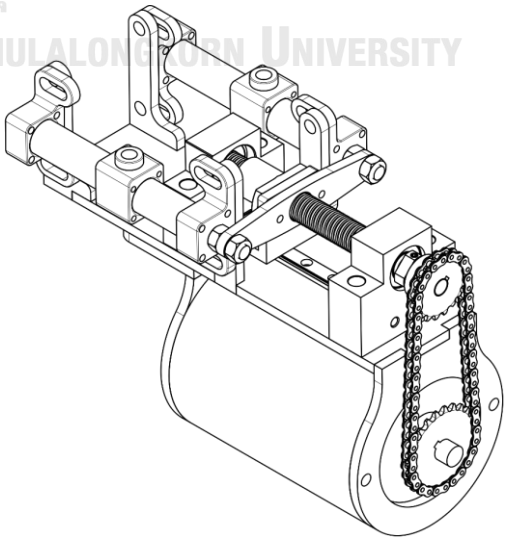


Figure 30 Designed brake actuator

A designed brake module is illustrated in **Figure 30**. This actuator consists of 2 hydraulic master cylinders for 2 available brake circuits, i.e. front and rear circuit. This actuator introduces the brake pressure to a circuit by pushing a piston at one end of the cylinder using an electric motor through a 4:3 sprocket ratio chain transmission and a 3-millimeter pitch lead screw which transforms a rotational motion to linear motion.

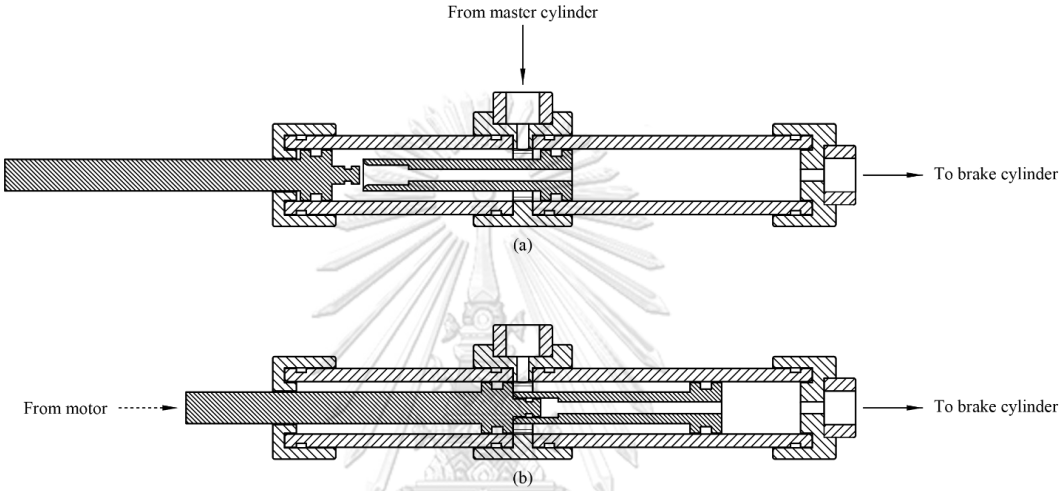


Figure 31 Section views of a designed master cylinder

Figure 31 shows section views of a designed master cylinder at 2 working positions. Inside this master cylinder, 2 pistons are working together. A piston on the left is connected to a lead screw set and a right piston is floating inside the cylinder. Not showing in the figure is 2 springs loaded inside the cylinder to maintain both pistons to be in a position shown by Figure 31 (a), whenever the driving motor is inactive. Figure 31 (a) illustrate the idle position state of the master cylinder. At this position, the user applied pressure from the original brake master cylinder enters the top port of the designed master cylinder and freely leave at the right port to a wheel cylinder. In the activated state, shown by Figure 31 (b), force is applied to a piston on the left by a motor. The left piston will also push the floating piston, isolating the brake line from the original master cylinder and generating pressure in a brake circuit, thus activates the wheel cylinder. The designed brake actuator is installed to the original braking system by replacing a portion for the original brake line as shown by **Figure 32**.

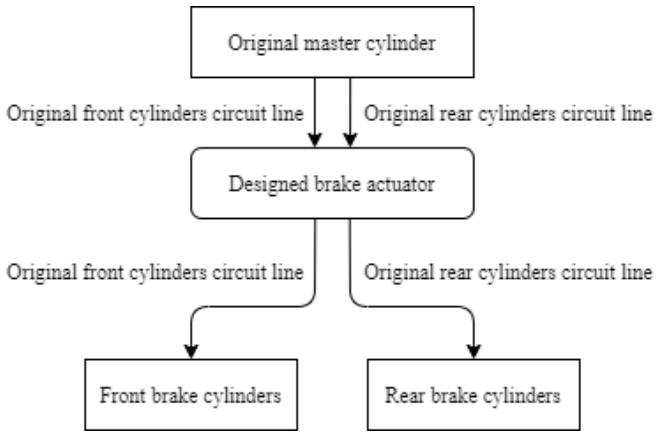


Figure 32 Installation of the designed brake actuator

1.4.2. Electronics System

The previous section presents the modification of a braking system which allows the deceleration of a car to be controlled by a high-level controller. For acceleration, the original motor controller is controlled by the microcontroller. Originally, the motor controller of the car receives the acceleration command from the accelerator pedal. Thus, to take control of a motor controller, a sensed voltage signal from the accelerator pedal is replaced by a voltage signal generated by the low-level controller.

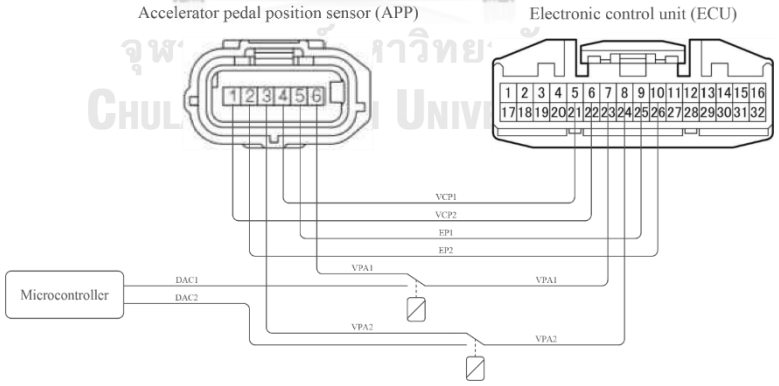


Figure 33 Modified accelerator pedal position sensor circuit wiring diagram

The Accelerator Pedal Position Sensor (APP) of the car generates 2 voltage signals and measured by the Electronic Control Unit (ECU). One of such voltage signals is offset to the other by 750 millivolts. The ECU converts these voltage signals into a driving motor torque command and then sent to the motor using a vehicle CAN.

The modified circuit of the accelerator pedal position sensor is shown in **Figure 33**. In this figure, VCP1 and VCP2 are 5-volt sensor supply lines from the ECU. EP1 and EP2 are the reference ground lines of VCP1 and VCP2, respectively. The voltage signal lines mentioned above are VPA1 and VPA2. In a modified circuit, the microcontroller of the speed control system controls a vehicle acceleration through the generated voltage signals, shown by DAC1 and DAC2 lines in **Figure 33**. These signals generated from the dual 12-bit Digital to Analog Converter (DAC) by the microcontroller and replace the signals from the APP using 2 Single-Pole-Double-Throw (SPDT) type mechanical relays. Furthermore, the actual odometry speed of the car is obtained from the vehicle CAN bus.

Similar to the steering control system, an electric motor of the brake actuator is driven by the H-bridge DC motor driver supplied by a 24-volt battery, however, limited by 75 percent of the maximum power to prevent the brake module from damage caused by a motor running at maximum power. This motor driver is directly controlled by the microcontroller. **Figure 34** illustrates the overview component diagram of the steering control electronics system.

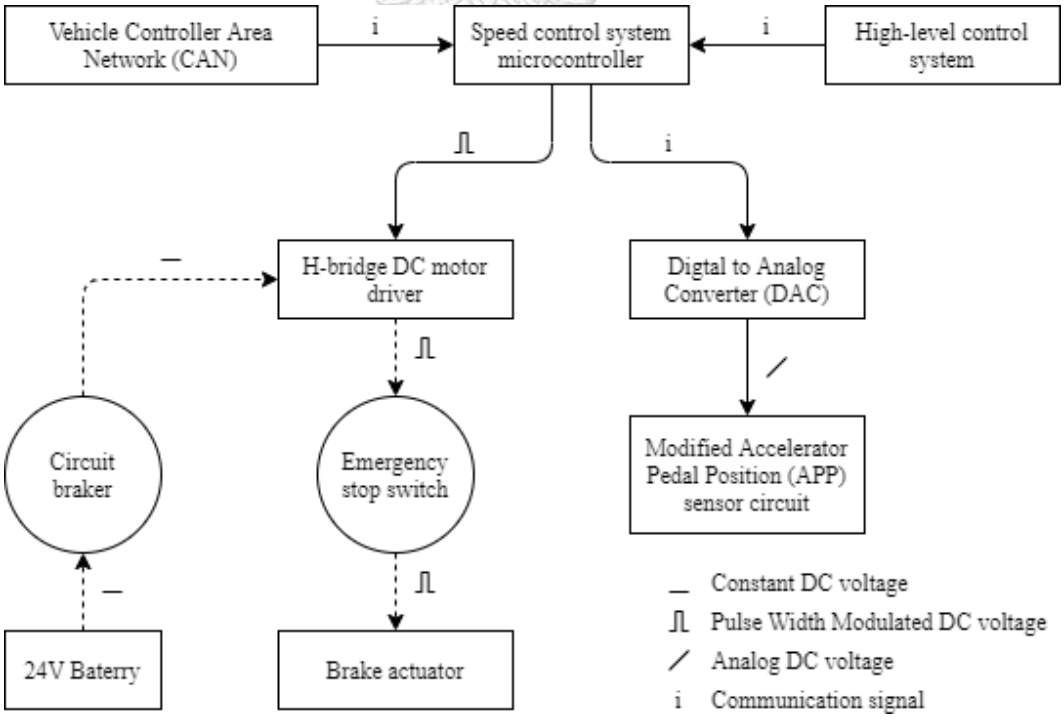


Figure 34 Speed control system's electronic components diagram

Note that the dashed line denotes the 24-volt power line and all low voltage level signal lines are represented by solid lines.

1.4.3. Low-level Speed Control System

A simple PID is implemented in the speed control system as in the steering control system. However, 2 actuators are presented in this section, i.e. the brake actuator and the modified APP circuit. Thus, a modification is applied to the PID controller output part such that both 2 actuators can be implemented in the PID controller. The modified output PID controller is shown by **Figure 35**.

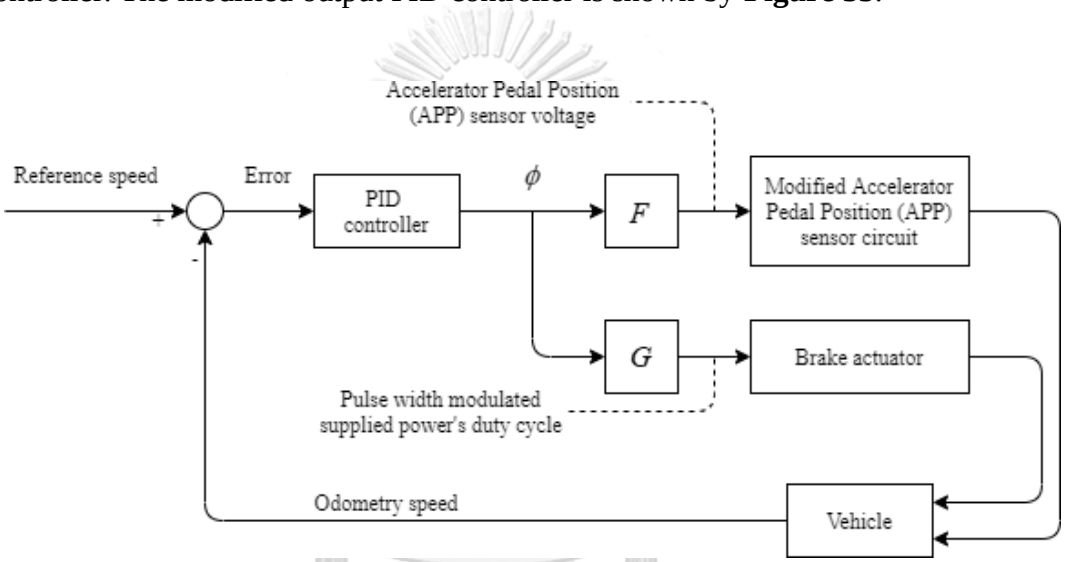


Figure 35 Speed control system block diagram

As shown in **Figure 35**, a sensed actual speed is subtracted from the reference speed received from the high-level control system result in a speed error. Then, the PID controller with a set of coefficients is fed by this speed error, given an output ϕ that later plugged into a discontinuous gain transfer function F and G. The transfer function F and G give the APP sensor voltage and the brake actuator motor driver's pulse duty cycle, respectively, and defined by the following Equation (20) and Equation (21).

$$F = H(\phi) \tag{20}$$

$$G = \left(1 - H(\phi + \delta)\right)\left(\alpha\left(1 + \frac{\delta}{\phi}\right)\right) \tag{21}$$

Where $H(x)$ represents the Heaviside step function. Since the vehicle regenerative braking is activated when is accelerator pedal is in a released position which leads to a deceleration. And similar to a human driving characteristic in which a brake force is not applied for every time the speed needed to be reduced. Then, an actuator dormant interval δ is introduced to the brake actuator gain transfer function, i.e. G. Also, because of a different actuator, a linear proportion to the PID controller output ϕ assumption is applied by introducing the brake actuator constant gain α to G. The output of the discontinuous gain G and F versus the PID controller output ϕ is shown by **Figure 36**.

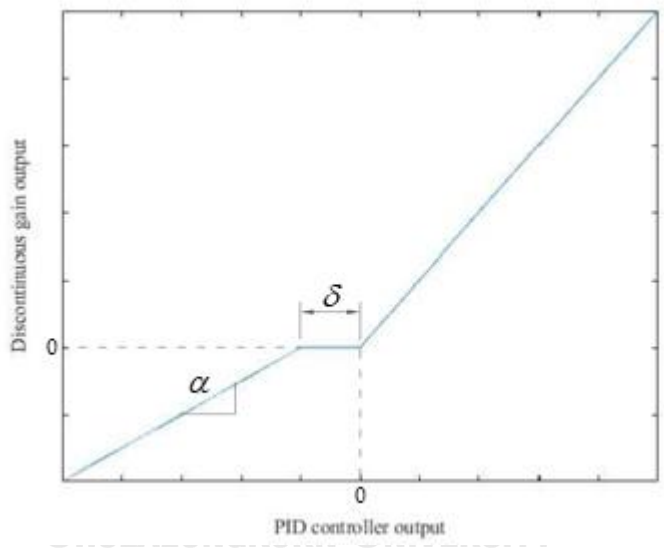


Figure 36 Output of the discontinuous gain transfer function

1.4.4. Microcontroller Software

A workflow of the low-level speed control system software is illustrated by Figure 37. Start by algorithm initialization, the microcontroller retrieves the initial parameters listed below from its EEPROM.

- The initial reference speed used in automatic control mode
- The proportional, integral and derivative coefficients of the PID controller
- An actuator dormant interval and a brake actuator constant gain
- A controlled loop time interval

Then, the car speed is read from a vehicle CAN providing an odometry speed in kilometer per hour with a resolution of 0.004 kilometers per hour. Since this odometry speed measured by using a resolver contains signal noise. Then, a simple moving average digital filter of 10 previous samples is applied to the received speed. Next, the microcontroller will check whether there is a valid command received from a high-level controller. If so, the corresponding action to the received command will be executed. Subsequently, if the emergency braking mode is engaged, the microcontroller will check for the speed and activates the brake actuator in case the car is moving. In case the automatic control mode is engaged, the controller will determine the output using the digital PID controller. The output will be classified into 3 intervals, including a deceleration, acceleration, and dormant interval. The microcontroller will release an accelerator pedal and activate the brake actuator according to the output classified into a deceleration interval. On the other hand, the brake actuator will be released and the accelerator pedal will be activated if the output is classified into an acceleration interval. If the output is in a dormant interval, then both the brake actuator and the accelerator pedal are released. However, if the controller is neither in the emergency nor the automatic control mode, a brake actuator and an accelerator pedal will be released. Finally, the microcontroller will return the required parameters to the high-level controller and idly wait for the controlled loop time interval to be reached, and then begin the next computational loop.

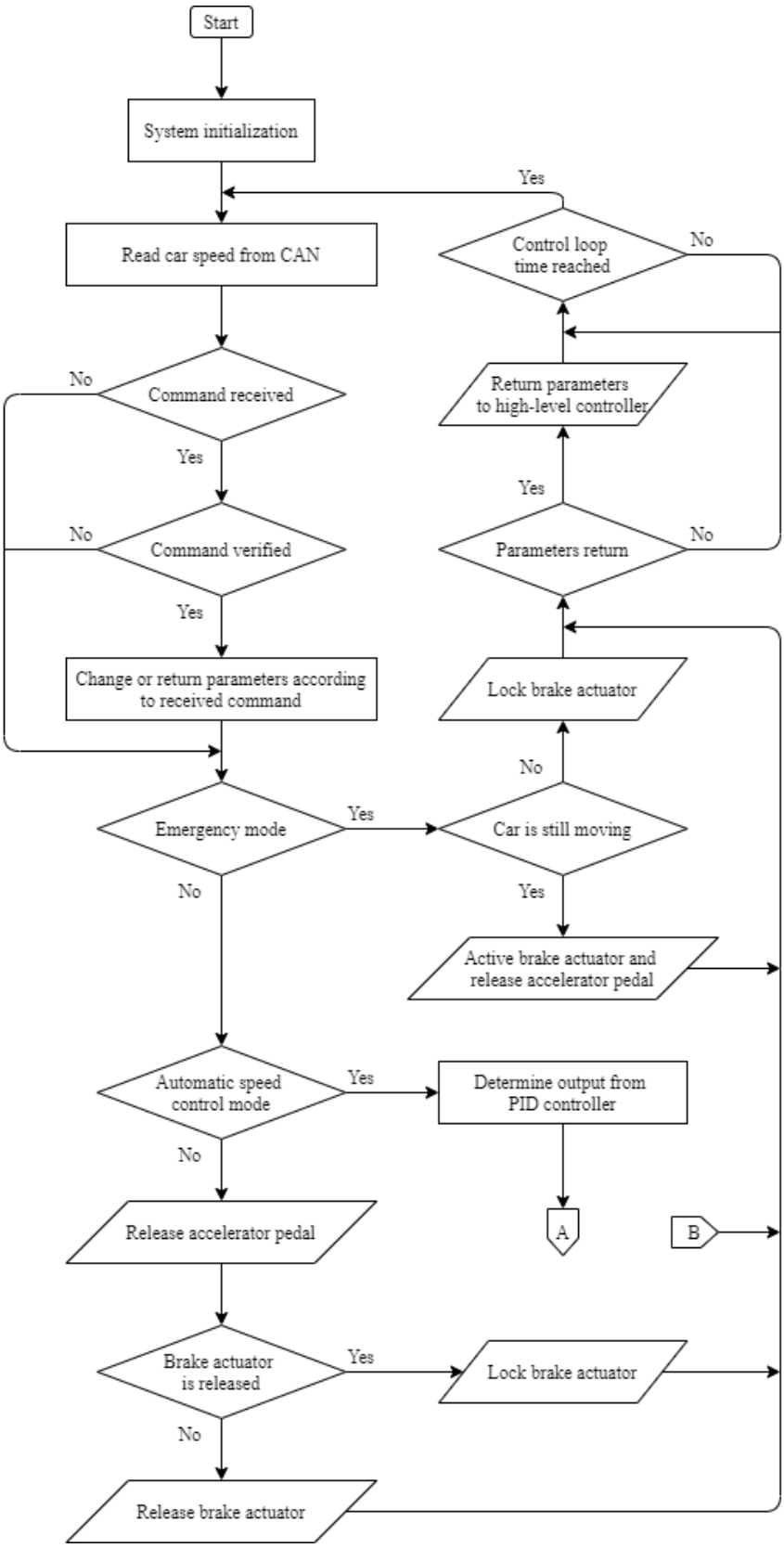


Figure 37 low-level speed control software's flowchart

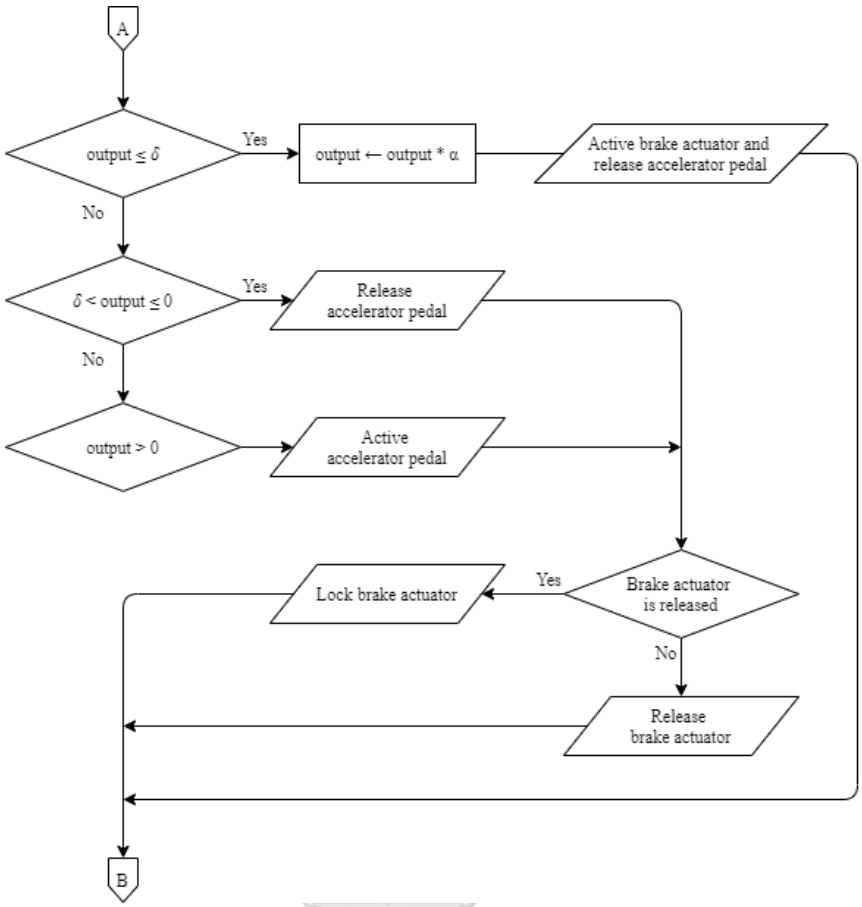


Figure 37 low-level speed control software's flowchart (continued)

1.4.5. Parameters Tuning

The proportional, integral, and derivative coefficient of the PID controller, a brake actuator constant, and the actuator dormant interval is manually tuned in this tuning process since the exact mathematical model is not available at the moment. This tuning process's controlled loop time interval is set at 5 milliseconds. By tuning until the satisfactory result is achieved, the tuned parameters are obtained as shown by Equation (22).

$$\begin{aligned} K_p &= 420 \\ K_i &= 700 \\ K_d &= 30 \\ \alpha &= 2 \\ \delta &= 1.2 \times 10^5 \end{aligned} \tag{22}$$

Where K_p, K_i, K_d, α, and δ denote the proportional, integral, and derivative coefficient of the PID controller, a brake actuator constant, and the actuator dormant interval, respectively. **Figure 38** illustrates the step response from the initial speed of 3.6 kilometers per hour to the reference speed of 30.0 kilometers per hour. Note that these tuned coefficients and constants must be used only in the designed low-level software so that the response illustrated by Figure 38 will be obtained.

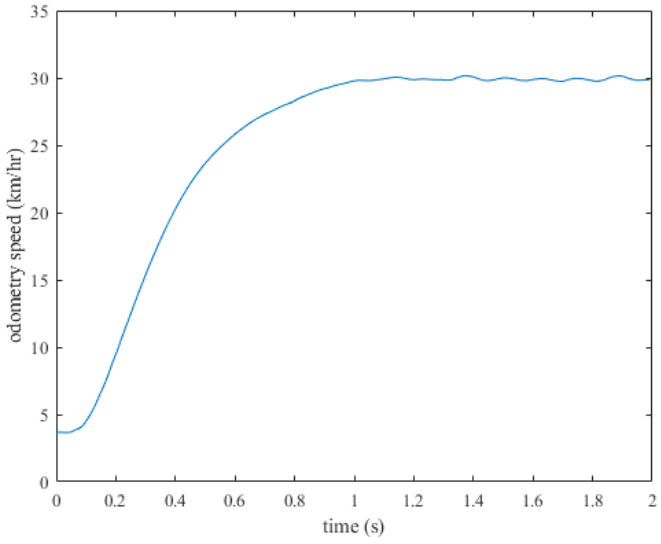


Figure 38 Step response of a speed control system

2. High-level Control System

All components that contribute to the development of the autonomous navigation software are introduced in this section. Start by considering the vehicle model, the relationship between a controlled input and the vehicle trajectory is obtained and later used in a developed path planning algorithm, i.e., a scored predicted trajectory. Eventually, this algorithm is implemented in the developed autonomous navigation high-level software.

2.1. Vehicle Model

To obtain a predicted trajectory of the vehicle to be used in the path evaluation process, a certain vehicle model needs to be obtained. There are several vehicle

models available to be used in determining the vehicle trajectory. However, they come with different levels of complexity and, of course, providing different accuracy.

Since the objective of this research is to navigate the autonomous vehicle at low speed, also 2-dimensional GNSS is equipped in a localization system. Consequently, a 2-dimensional single-track kinematic model is considered to be the most suitable model here for sake of simplicity and a relatively low computational power.

2.1.1. Single-Track Kinematic Model

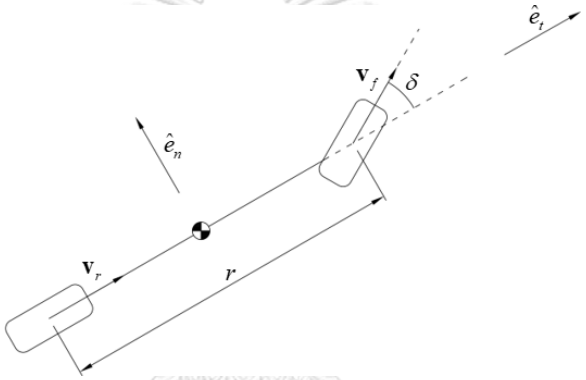


Figure 39 Single-track vehicle model

By considering Figure 39, we may obtain the kinematic relation of a single-track vehicle model as shown in Equation (23).

$$\mathbf{v}_f = \mathbf{v}_r + \boldsymbol{\omega} \times r\hat{e}_t \tag{23}$$

Where $\mathbf{v}_r = \dot{s}\hat{e}_t$ denotes the rear wheel velocity with a magnitude of $\dot{s}$ in $\hat{e}_t$ direction, $\mathbf{v}_f$ is the front wheel velocity and $\boldsymbol{\omega} = \omega\hat{k}$ is the vehicle angular velocity with a magnitude of ω in $\hat{k} = \hat{e}_t \times \hat{e}_n$ direction.

By applying a vector product between both sides of Equation (23) with $\hat{e}_t$ we may obtain Equation (24).

$$\left|\mathbf{v}_f\right| \sin\delta = \omega r \tag{24}$$

And by applying a scalar product between both sides of Equation (23) with $\hat{e}_t$, we also get the tangential component relation as shown in Equation (25).

$$\left|\mathbf{v}_f\right|\cos\delta = \dot{s} \tag{25}$$

Eventually, Equation (26) can be derived by combining Equation (24) and Equation (25).

$$\frac{\omega}{\dot{s}} = \frac{\tan\delta}{r} \tag{26}$$

From Equation (26) we can see that both sides of the equation represent a curvature of a certain vehicle trajectory. Hence, for simplicity's sake, we may use $\kappa = \frac{\omega}{\dot{s}}$ to denote a trajectory curvature. Consequently, Equation (26) can be rearranged and results in Equation (27).

$$\kappa = \frac{\tan\delta}{r} \tag{27}$$

2.1.2. Approximated Linear Relationship

A range of a steering angle in a typical commercial car is around 70 degrees, which can be divided into around 35 degrees inner steering angle and around 35 degrees outer steering angle. The car used in this research has a steering angle range of 74 degrees, say 38 degrees inner and 34 degrees outer steering angle, and a wheelbase of 1.53 meters. By applying a steering angle range and a wheelbase to Equation (27), we may obtain the plot of the curvature at a different steering angle which is depicted by Figure 40.

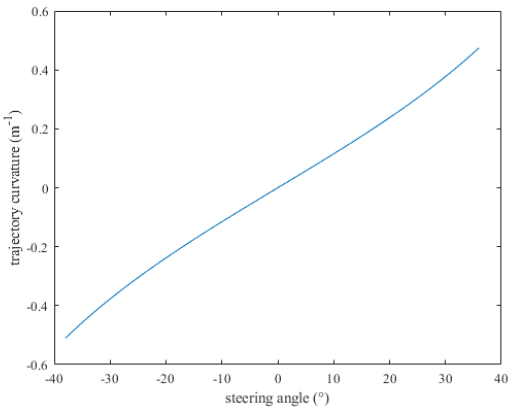


Figure 40 Trajectory curvature from the single-track kinematic model at any steering angles

From Figure 40, it can be observed that the relationship between a trajectory curvature and a steering angle is linearly proportional to each other. Consequently, by introducing Equation (27) to an infinite series expansion at $\delta = 0$ we may obtain the result as Equation (28).

$$\kappa = \frac{\delta}{r} + \frac{\delta^3}{3r} + \frac{2\delta^5}{15r} + \frac{17\delta^7}{315r} + \ldots \tag{28}$$

By considering only the first term of the right-hand side of Equation (28), an approximated linear relationship between trajectory curvature and steering angle is then obtained as Equation (29).

$$\kappa = \frac{\delta}{r} \tag{29}$$

2.1.3. Effective Inverse Wheelbase Calibration

To use the Equation (29) in autonomous navigation one crucial parameter needs to be substituted, namely a vehicle wheelbase r. Even though the vehicle wheelbase can be measured directly from the car or get from a car technical manual, that figure still cannot be used since there will be some other effects from other components of the car that will deviate the actual trajectory curvature from the result of Equation (29).

For this reason, an effective wheelbase that will be used in predicting the car trajectory must be obtained. To determine the value of this effective wheelbase, one can directly measure actual curvatures at various steering angles from a certain empirical experiment. However, directly measuring actual curvatures would be difficult without a precise localization system. Hence, instead of measuring the curvature directly, the angular velocity and the vehicle speed will be measured.

After the actual angular velocity, occasionally yaw rate, and the vehicle speed at various steering angles are collected, the regression analysis is then introduced. Since the steering position obtained from the car steering position sensor comes in scale and offset to the actual steering angle, then a more suitable Equation (30) for regression analysis is introduced to include all scale and offset parameters in one single equation.

$$\kappa = \rho\delta + \varepsilon \tag{30}$$

As mention earlier, $\kappa = \frac{\omega}{\dot{s}}$ where ω is a collected yaw rate and $\dot{s}$ is vehicle speed. ρ denotes the inverse effective wheelbase and ε demotes the steering position center.

The inverse effective wheelbase and steering position center can be determined by the following Equation (31).

$$\begin{bmatrix} \rho \\ \varepsilon \end{bmatrix} = \left(\mathbf{A}^T \mathbf{A} \right)^{-1} \mathbf{A}^T \boldsymbol{\kappa} \tag{31}$$

Where $\mathbf{A} = \begin{bmatrix} \vec{\delta} & \vec{1} \end{bmatrix}$ and $\boldsymbol{\kappa} = \vec{\kappa}$, which $\vec{\delta}$ and $\vec{\kappa}$ are $m \times 1$ vector of m collected steering positions and corresponding trajectory curvatures, respectively.

By substitute collected trajectory curvatures and steering positions into Equation (31), we may obtain the following parameters.

$$\rho = 4.980133239 \times 10^{-5} \tag{32}$$

$$\varepsilon = -0.4317661517 \tag{33}$$

Units of an inverse effective wheelbase and a steering position center are (meters · steering position sensor output unit)$^{-1}$ and (meters)$^{-1}$, respectively. Figure 41 depicts a least-square regression analysis result.

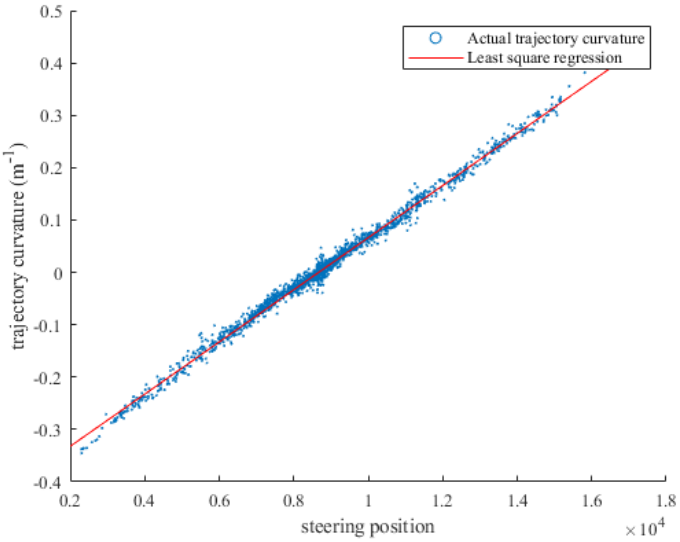


Figure 41 Actual trajectory curvature and approximated linear relationship

2.2. Scored Predicted Trajectory

Scored Predicted Trajectory is the algorithm developed in this research to be used in the autonomous navigation section. This algorithm determines the most suitable steering angle such that the car will follow a given waypoint according to the score weight that will dominate the path following behavior of a vehicle which is arbitrarily predefined by the user. However, the algorithm doesn't determine a vehicle-controlled speed as vehicle speed is one of the algorithm input parameters.

2.2.1. Trajectory Prediction

The single-track kinematic model has been introduced in the previous section. In this section, that model is applied again to determine the trajectory of a vehicle, namely predicted trajectories, for used score evaluation process.

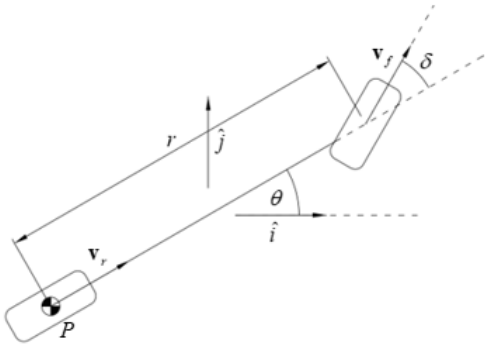


Figure 42 single-track kinematic model

Instead of locating a vehicle position reference at the center of geometry or the center of mass of a vehicle, here the center of a rear axle P is utilized. By considering Figure 42, also with the assumption that the steering system is a perfect Ackermann steering geometry so that the velocity of the car reference point P will always align with a car heading direction. Hence, we may construct the equation of the predicted trajectory of a point P as follow.

$$\boldsymbol{\rho}(t) = \int_0^t \mathbf{v}_r(\tau)\, d\tau \tag{34}$$

Where $\boldsymbol{\rho}(t)$ is a car position reference vector of a point P and $\mathbf{v}_r$ is a velocity of a point P attached to a vehicle body.

Rearranging Equation (34) gives Equation (35) that splits all parameters in Equation (34) into 2 components, namely along inertia reference axis $\hat{i}$ and $\hat{j}$.

$$\begin{bmatrix} \rho_x(t) \\ \rho_y(t) \end{bmatrix} = \int_0^t v_r(\tau) \begin{bmatrix} \cos\theta(\tau) \\ \sin\theta(\tau) \end{bmatrix} d\tau \tag{35}$$

Where $\boldsymbol{\rho}(t) = \begin{bmatrix} \rho_x(t) & \rho_y(t) \end{bmatrix}^T$, $\rho_x(t)$ and $\rho_y(t)$ are magnitudes of a car reference vector $\boldsymbol{\rho}(t)$ at any time instance t.

Since $\theta(t) = \int_0^t \omega(\tau) d\tau$ where $\omega(t)$ is the angular speed or yaw rate of a vehicle body, also of the velocity $\mathbf{v}_r$ and a vehicle heading direction. Then Equation (35) can be rearranged as follow.

$$\begin{bmatrix} \rho_x(t) \\ \rho_y(t) \end{bmatrix} = \int_0^t v_r(\tau) \begin{bmatrix} \cos\int_0^t \omega(\xi) d\xi \\ \sin\int_0^t \omega(\xi) d\xi \end{bmatrix} d\tau \tag{36}$$

Furthermore, it has been shown earlier that for a single-track kinematic model with a perfect Ackermann steering geometry, the trajectory curvature can be approximately considered as linearly proportional to a steering angle with a certain gain, independently from the vehicle speed.

$$\kappa(t) = \frac{\omega(t)}{\dot{s}(t)} = \rho\delta(t) + \varepsilon \tag{37}$$

Rearranging Equation (37) gives Equation (38).

$$\omega(t) = \dot{s}(t)\left(\rho\delta(t) + \varepsilon\right) \tag{38}$$

Since $\left|\mathbf{v}_r(t)\right| = v_r(t) = \dot{s}(t)$. Then, combining Equation (36) and Equation (38) results in the following Equation (39).

$$\begin{bmatrix} \rho_x(t) \\ \rho_y(t) \end{bmatrix} = \int_0^t \dot{s}(\tau) \begin{bmatrix} \cos \int_0^t \dot{s}(\xi)\left(\rho\delta(\xi)+\varepsilon\right)d\xi \\ \sin \int_0^t \dot{s}(\xi)\left(\rho\delta(\xi)+\varepsilon\right)d\xi \end{bmatrix} d\tau \tag{39}$$

Equation (39) above will provide several predicted trajectories for different steering angles to be evaluated by the score weight mentioned early in this section. This evaluation process will be introduced in a section below.

2.2.2. Trajectory Evaluation and Scoring

After several trajectories corresponding to several arbitrary predefined steering angles are obtained by solving the trajectory Equation (39). These trajectories will be brought into a scoring evaluation process which is mainly considered in three different topics, namely, a linear deviation, an angular deviation, and a collision distance. Then, each of the trajectories will be assigned by three scores from the three topics evaluation mention earlier and will be combined later according to weights that arbitrary given by the user.

2.2.2.1. Linear Deviation Evaluation

Linear deviation evaluation of the trajectory determines the distance between a certain trajectory and a given waypoint in cartesian space. This evaluation can be done by performing the integration of the shortest cartesian distance between along the trajectory need to be evaluated.

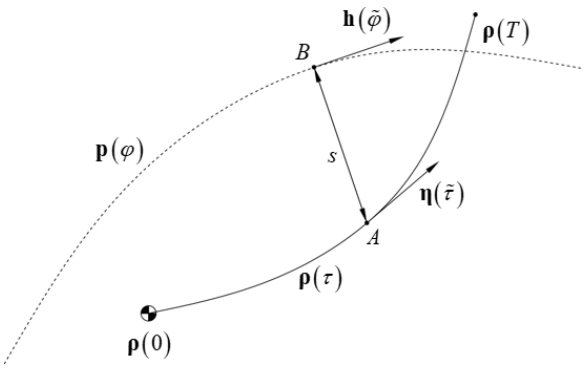


Figure 43 Linear deviation evaluation of a certain trajectory

Figure 43 depicts a linear deviation of a certain trajectory $\boldsymbol{\rho}(\tau)$, as shown by a continuous line, from a waypoint $\mathbf{p}(\varphi)$, as shown by a dashed line. Point B is the point closest to point A that lies on a waypoint $\mathbf{p}(\varphi)$. Let us rewrite the waypoint as shown by Equation (40).

$$\mathbf{p}(\varphi) = \begin{bmatrix} p_x(\varphi) & p_y(\varphi) \end{bmatrix}^T \tag{40}$$

And so do the trajectory.

$$\boldsymbol{\rho}(\tau) = \begin{bmatrix} \rho_x(\tau) & \rho_y(\tau) \end{bmatrix}^T \tag{41}$$

Let us assume that point A locates on a trajectory $\boldsymbol{\rho}$ where $\tau = \tilde{\tau}$ and point B locates on a waypoint $\mathbf{p}$ where $\varphi = \tilde{\varphi}$.

$$A = \boldsymbol{\rho}(\tilde{\tau}) = \begin{bmatrix} \rho_x(\tilde{\tau}) & \rho_y(\tilde{\tau}) \end{bmatrix}^T \tag{42}$$

$$B = \mathbf{p}(\tilde{\varphi}) = \begin{bmatrix} p_x(\tilde{\varphi}) & p_y(\tilde{\varphi}) \end{bmatrix}^T \tag{43}$$

Since B is the point closest to point A, then a formal condition for $\tilde{\varphi}$ is stated by the following Equation (44).

$$\tilde{\varphi}(\tau) = \underset{\varphi}{\operatorname{argmin}} \left(\left\| \boldsymbol{\rho}(\tau) - \mathbf{p}(\varphi) \right\| \right) \tag{44}$$

It can be shown that $\tilde{\varphi}$ happens when Equation (45) is satisfied.

$$\left(\rho_x(\tilde{\tau}) - p_x(\tilde{\varphi})\right) \frac{d}{d\varphi} p_x(\varphi) \Big|_{\varphi=\tilde{\varphi}} = \left(p_y(\tilde{\varphi}) - \rho_y(\tilde{\tau})\right) \frac{d}{d\varphi} p_y(\varphi) \Big|_{\varphi=\tilde{\varphi}} \tag{45}$$

To perform an integration over the trajectory length, an infinitesimal length along the trajectory needs to be defined. This infinitesimal length is shown by Equation (46).

$$d\rho = \sqrt{\left(\frac{d}{d\tau}\rho_x(\tau)\right)^2 + \left(\frac{d}{d\tau}\rho_y(\tau)\right)^2}\, d\tau \tag{46}$$

Here, the evaluation integral can be performed and results in the following Equation (47).

$$\sigma_l = \int_0^T \sqrt{\left(\left(\rho_x(\tau) - p_x(\tilde{\varphi})\right)^2 + \left(\rho_y(\tau) - p_y(\tilde{\varphi})\right)^2\right)\left(\left(\frac{d}{d\tau}\rho_x(\tau)\right)^2 + \left(\frac{d}{d\tau}\rho_y(\tau)\right)^2\right)} d\tau \tag{47}$$

Note here that σ_l denotes a linear deviation score of a trajectory $\boldsymbol{\rho}$ that is evaluated over a given waypoint $\mathbf{p}$.

2.2.2.2. Angular Deviation Evaluation

Angular deviation evaluation determines the difference of gradient between a trajectory and a given waypoint. This process is done in the same manner as the linear deviation evaluation. However, in this process, the parameter used in the comparison is the gradient instead of a cartesian distance.

Considering again on Figure 43, $\boldsymbol{\eta}(\tilde{\tau})$ denotes the tangential vector to a trajectory at $\tilde{\tau}$ and $\mathbf{h}(\tilde{\varphi})$ represents the tangential vector to the waypoint at $\tilde{\varphi}$. The relationship between a trajectory, waypoint, and their tangential vectors are stated in the following Equation (48) and Equation (49).

$$\boldsymbol{\eta}(\tau) = \frac{d}{d\tau}\boldsymbol{\rho}(\tau) \tag{48}$$

$$\mathbf{h}(\varphi) = \frac{d}{d\varphi}\mathbf{p}(\varphi) \tag{49}$$

Since the angular deviation in this process is defined as a difference in a gradient or heading between a trajectory and a waypoint. Hence, the following Equation (50) defines the formulation of an angular deviation ψ.

$$\psi(\tau) = \arccos\left(\frac{\boldsymbol{\eta}(\tau)\cdot\mathbf{h}(\tilde{\varphi})}{\|\boldsymbol{\eta}(\tau)\|\|\mathbf{h}(\tilde{\varphi})\|}\right) \tag{50}$$

Combining Equation (46) and Equation (50) gives the following Equation (51).

$$\sigma_a = \int_0^T \arccos\left(\frac{\boldsymbol{\eta}(\tau)\cdot\mathbf{h}(\tilde{\varphi})}{\|\boldsymbol{\eta}(\tau)\|\|\mathbf{h}(\tilde{\varphi})\|}\right)\sqrt{\left(\left(\frac{d}{d\tau}\rho_x(\tau)\right)^2+\left(\frac{d}{d\tau}\rho_y(\tau)\right)^2\right)}d\tau \tag{51}$$

Where σ_a represents the angular deviation score of a trajectory $\boldsymbol{\rho}$ that is evaluated over a waypoint $\mathbf{p}$.

2.2.2.3. Collision Distance Evaluation

Similar to the linear deviation evaluation, this collision distance evaluation utilizes a cartesian distance in a scoring process. However, instead of using a waypoint, this section uses the obstacle point scan to evaluate the score.

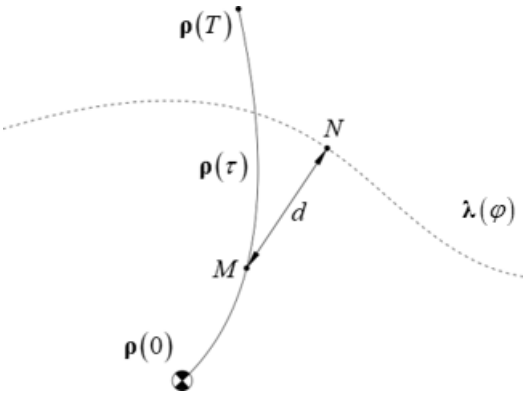


Figure 44 Collision distance evaluation of a certain trajectory

Figure 44 depicts an obstacle point represented by $\boldsymbol{\lambda}(\varphi)$ and a vehicle trajectory $\boldsymbol{\rho}(\tau)$. Also N is the obstacle point whose closest point on a trajectory is M. Since this evaluation process is done the same way as linear deviation evaluation, except using an obstacle scan instead of using a waypoint. Then, the formulation of this score can be modified from the linear deviation score Equation (47). Consequently, the collision distance score can be determined by the following Equation (52).

$$\sigma_c = \int_0^T \sqrt{\left(\left(\rho_x(\tau)-\lambda_x(\tilde{\varphi})\right)^2+\left(\rho_y(\tau)-\lambda_y(\tilde{\varphi})\right)^2\right)\left(\left(\frac{d}{d\tau}\rho_x(\tau)\right)^2+\left(\frac{d}{d\tau}\rho_y(\tau)\right)^2\right)}\,d\tau \tag{52}$$

Where σ_c is the collision distance score of a trajectory $\boldsymbol{\rho}$ that is evaluated over a given waypoint $\mathbf{p}$, $\boldsymbol{\lambda}(\varphi)=\begin{bmatrix}\lambda_x(\varphi) & \lambda_y(\varphi)\end{bmatrix}^T$, and $\tilde{\varphi}$ is determined by the following Equation (53).

$$\tilde{\varphi}(\tau) = \underset{\varphi}{\operatorname{argmin}}\left(\left\|\boldsymbol{\rho}(\tau)-\boldsymbol{\lambda}(\varphi)\right\|\right) \tag{53}$$

2.2.2.4. Overall Score Combination

As mentioned earlier in this section, to determine the total score of an individual trajectory, three scores, namely, linear deviation score, angular deviation score, and collision distance score, will be linearly combined using their corresponding arbitrary user predefined weights. Equation (54) states a mathematical formulation of the overall score combination.

$$\gamma = \begin{bmatrix} w_l & w_a & w_c \end{bmatrix} \begin{bmatrix} \sigma_l \\ \sigma_a \\ \sigma_c \end{bmatrix} \tag{54}$$

Where $w_l \in \mathbb{R}^+$, $w_a \in \mathbb{R}^+$, and $w_c \in \mathbb{R}^-$ are the linear deviation, angular deviation, and collision distance weight, respectively. These weights can be interpreted as priorities of their corresponding score in the autonomous navigation routine, i.e., a high absolute value of w indicates a high priority of a corresponding score.

2.2.3. A Modification for Algorithm Implementation

Since the Score Predicted Trajectory derived from continuous functions of a trajectory, waypoint, and obstacle scan. However, in the actual situation, those mentioned come in discrete and discontinue functions. Then, all scoring formulas need to be modified to deal with a discrete and discontinue function. Also, some parts or parameters will be neglected for the sake of computation complexity that may affect the computational time which is a critical topic in real-time implementation of the autonomous navigation algorithm.

2.2.3.1. Vehicle Trajectory Approximation

A trajectory Equation (39) indicates that the speed $\dot{s}$ varies over the time used in a trajectory integration process. Since the time used in one computational loop, i.e. a computational time interval, will and short. Also, the objective of this research is to deal with a low-speed navigation system. Then a constant speed over a finite time interval that will be used in a finite trajectory integration should be a reasonable approximation assumption. The approximated trajectory equation then states as Equation (55).

$$\mathbf{\rho}(t) = \begin{bmatrix} \rho_x(t) \\ \rho_y(t) \end{bmatrix} = \dot{s} \int_0^t \begin{bmatrix} \cos \dot{s} \int_0^t \left(\rho\delta(\xi) + \varepsilon \right) d\xi \\ \sin \dot{s} \int_0^t \left(\rho\delta(\xi) + \varepsilon \right) d\xi \end{bmatrix} d\tau \tag{55}$$

To utilize the trajectory Equation (55) in software programming, a discretized form of Equation (55) is more preferable. By replacing all true integration terms by a

trapezoidal numerical integration, Equation (55) then results in the following Equation (56).

$$\boldsymbol{\rho}_i = \begin{bmatrix} \rho_{xi} \\ \rho_{yi} \end{bmatrix} = \frac{\dot{s}}{2} \sum_{j=1}^{i} \left(\xi_{j+1} - \xi_j \right) \begin{bmatrix} \cos\theta_{j+1} + \cos\theta_j \\ \sin\theta_{j+1} + \sin\theta_j \end{bmatrix} \tag{56}$$

Where θ_j is defined by Equation (57).

$$\theta_j = \frac{\dot{s}}{2} \sum_{k=1}^{j} \left(\xi_{k+1} - \xi_k \right) \left(\rho \left(\delta \left(\xi_{k+1} \right) + \delta \left(\xi_k \right) \right) + 2\varepsilon \right) \tag{57}$$

Also, ξ_i is a discretized time of Δt interval defined by the following Equation (58).

$$\xi_i = \left(i - 1 \right) \Delta t \tag{58}$$

In which a total time interval T of the trajectory is determined by Equation (59).

$$T = \left(N - 1 \right) \Delta t \tag{59}$$

2.2.3.2. Discrete Linear Deviation Evaluation

In a previous derivation of a continuous linear deviation scoring formula, the definite integral over the trajectory length is performed. However, a linear deviation scoring intends to set a parameter that indicates a measure of the total cartesian distance between a trajectory and a given waypoint along such trajectory. Consequently, instead of including the infinitesimal distance, i.e., Equation (46), this distance will be neglected in the discrete version for linear scoring Equation (60).

$$\sigma_l = \sum_{i=1}^{N} \left\| \boldsymbol{\rho}_i - \tilde{\mathbf{p}}_i \right\| \tag{60}$$

Where $\tilde{\mathbf{p}}_i$ is the closest point to a point $\boldsymbol{\rho}_i$ lying on a waypoint $\mathbf{p}$ and can be determined by the following Equation (61).

$$\tilde{\mathbf{p}}_i = \mathbf{p}_j \tag{61}$$

where $j = \underset{\varphi}{\operatorname{argmin}} \left(\left\| \boldsymbol{\rho}_i - \mathbf{p}_\varphi \right\| \right)$.

2.2.3.3. Discrete Angular Deviation Evaluation

In software implementation of the algorithm, the waypoint used will not be in continuous function form since the waypoint will be recorded directly from a

localization system. Therefore, performing a true differentiate of a waypoint to get the tangential vector as in equation cannot be done. Also, applying a finite differentiate will result in a diffuse tangential vector. Consequently, in this research both heading angle and position will be recorded directly and will be used as a reference tangential vector direction and a waypoint, respectively.

Similar to the previous section, the infinitesimal distance determined by Equation (46) is neglected here. Thus, Equation (62) below defines an angular deviation scoring for a discrete system.

$$\sigma_a = \sum_{i=1}^{N} \left\| \eta_i - \tilde{h}_i \right\| \tag{62}$$

Where η_i represents the heading angle of trajectory at $\boldsymbol{\rho}_i$ and $\tilde{h}_i$ is a recorded heading of the point closest to a point $\boldsymbol{\rho}_i$ lying on a waypoint $\mathbf{p}$ which can be determined by the following Equation (63).

$$\tilde{h}_i = h_j \tag{63}$$

where $j = \underset{\varphi}{\operatorname{argmin}} \left(\left\| \boldsymbol{\rho}_i - \mathbf{p}_\varphi \right\| \right)$ and h is a set of recorded heading corresponding to a recorded waypoint.

2.2.3.4. Discrete Collision Distance Evaluation

The obstacle scan data returned from a laser scanner device is a set of a discrete point cloud of obstacles in range relative to the device position. To use this set of obstacle points, again discretization of the evaluate equation need to be performed first. Similar to a discrete linear deviation scoring Equation (60), the cartesian distance between trajectory and a set of obstacle points is utilized. By modifying Equation (60) for the usage in this context, we then obtained an equation for discrete collision distance scoring as follows.

$$\sigma_c = \sum_{i=1}^{N} \left\| \boldsymbol{\rho}_i - \tilde{\boldsymbol{\lambda}}_i \right\| \tag{64}$$

Where $\tilde{\boldsymbol{\lambda}}_i$ is the closest point to a point $\boldsymbol{\rho}_i$ which is a member of a set of obstacle points $\boldsymbol{\lambda}$ and can be determined by Equation (65).

$$\tilde{\boldsymbol{\lambda}}_i = \boldsymbol{\lambda}_j \tag{65}$$

As identical to previous sections.

$$j = \underset{\varphi}{\operatorname{argmin}} \left(\left\| \boldsymbol{\rho}_i - \mathbf{p}_\varphi \right\| \right) \tag{66}$$

2.2.3.5. Critical Obstacle Distance Evaluation

In all previous sections, the trajectory used in all scoring equation is derived from a particle model. However, in a practical situation, the vehicle cannot be considered as a particle and represented by merely a single point. Depending on the geometry of a vehicle used in a practical implementation, there must be a certain boundary for the distance between a trajectory and an obstacle point such that this boundary will act as a buffer area and preventing collision between vehicle and obstacle.

Similar to linear deviation and collision distance scoring, this critical obstacle distance evaluation also uses the cartesian distance in determining the score. This section utilized the same distance which was used in discrete collision distance evaluation, however, some modification to Equation (60) is applied and result in the following equation.

$$\sigma_m = \begin{cases} \infty, & \text{if } \min\limits_{i \in I^+ \cap i \le N} \left(\left\| \boldsymbol{\rho}_i - \tilde{\boldsymbol{\lambda}}_i \right\| \right) \le \phi_m \\ 0, & \text{otherwise} \end{cases} \tag{67}$$

Where σ_m is the critical obstacle distance score and ϕ_m is the critical obstacle distance which is specified according to a vehicle geometry.

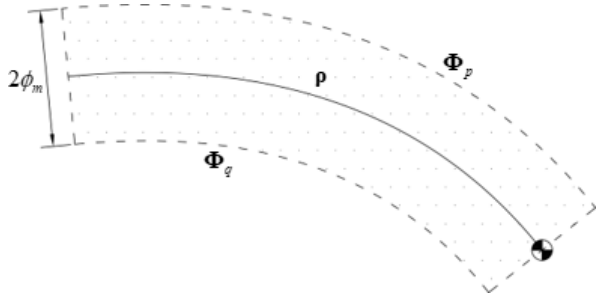


Figure 45 Critical obstacle distance and the buffer area

Figure 45 depicts the relationship between trajectory $\boldsymbol{\rho}$, critical obstacle distance ϕ_m, and a buffer area, represented by a dot space enclosed by dashed lines. The buffer area of $2\phi_m$ wide resulted from Equation (67). If there was any obstacle scan point located in this buffer area, the critical obstacle distance score belongs to the trajectory $\boldsymbol{\rho}$ will be given by ∞. This infinity score causes its corresponding trajectory to the lowest preferable track for a vehicle to follow, compare to other possible trajectories.

2.2.3.6. Overall Discrete Score Combination

All discrete scores, i.e. a discrete linear deviation, discrete angular deviation, discrete collision distance, and critical obstacle distance score, can be combined the same way as performing by Equation (54). However, since the additional critical obstacle distance score is not included by Equation (54). Then, a new linear combination equation of all discrete scores is presented by Equation (68).

$$\gamma = \begin{bmatrix} w_l & w_a & w_c & 1 \end{bmatrix} \begin{bmatrix} \sigma_l \\ \sigma_a \\ \sigma_c \\ \sigma_m \end{bmatrix} \tag{68}$$

Where $w_l \in \mathbb{R}^+$, $w_a \in \mathbb{R}^+$, and $w_c \in \mathbb{R}^-$ are the discrete linear deviation, discrete angular deviation, and discrete collision distance weight, respectively. γ is a final score of a certain trajectory that indicates the suitability of its corresponding trajectory.

2.2.4. Software Implementation

Scored Predicted Trajectory software determines several trajectories corresponding to the steering position command signals. These trajectories will later be evaluated by means of scoring processes introduced above. Figure 46 describes the algorithm details.

Inputs: Vehicle current position $\mathbf{p}$, vehicle current heading angle θ, vehicle current steering angle α, vehicle current speed $\dot{s}$, steering profile model $\boldsymbol{\psi}$, denotes the inverse effective wheelbase ρ, steering position center ε, corresponding discrete time space $\boldsymbol{\tau}$, route waypoint $\boldsymbol{\lambda}$, route heading angle $\boldsymbol{\upsilon}$, obstacle scan $\boldsymbol{\mu}$, score critical obstacle distance r_0, and weight $\mathbf{w}$

Outputs: Trajectory scores corresponding to the input steering profile model $\boldsymbol{\gamma}$, and most suitable steering command signal φ

Definitions:

1. $\boldsymbol{\eta} \triangleq$ a set of predicted trajectory heading of $\boldsymbol{\psi}$
2. $\boldsymbol{\delta}_i \triangleq$ a set of initial steering angles of $\boldsymbol{\psi}$
3. $\boldsymbol{\delta}_f \triangleq$ a set of final steering angles of $\boldsymbol{\psi}$
4. $\boldsymbol{\rho} \triangleq$ a set of predicted trajectory
5. $\boldsymbol{\sigma} \triangleq$ a score array of all predicted trajectory

Function:

1. $p \leftarrow \underset{i}{\operatorname{argmin}}\left(\left\|\delta_i - \theta\right\|\right), \delta_i \in \boldsymbol{\delta}_i$
2. $q \leftarrow \underset{i}{\operatorname{argmin}}\left(\left\|\delta_i - \theta\right\|\right), \delta_i \in \boldsymbol{\delta}_i - \left\{\delta_p\right\}$
3. $\delta_p \leftarrow \boldsymbol{\delta}_{i,p}$
4. $\delta_q \leftarrow \boldsymbol{\delta}_{i,q}$
5. $\boldsymbol{\delta}_t \leftarrow \dfrac{\left\|\delta_q - \theta\right\|\boldsymbol{\psi}_p + \left\|\delta_p - \theta\right\|\boldsymbol{\psi}_q}{\left\|\delta_p - \delta_q\right\|}$
6. **for** every final steering angle index i in $\boldsymbol{\delta}_f$ **do**
7. $\quad \boldsymbol{\eta}_{i,0} \leftarrow \theta$
8. $\quad \boldsymbol{\eta}_{i,k} \leftarrow \dfrac{\dot{s}}{2}\sum_{m=0}^{k}\left(\tau_{m+1} - \tau_m\right)\left(\rho\left(\boldsymbol{\delta}_{t,j,m+1} + \boldsymbol{\delta}_{t,j,m}\right) + 2\varepsilon\right) + \theta,$ $k \in I^+ \cap k \le N$

Figure 46 Scored Predicted Trajectory algorithm

9. $\boldsymbol{\rho}_{i,0} \leftarrow \mathbf{p}$
10. $\boldsymbol{\rho}_{i,k} \leftarrow \frac{\dot{s}}{2}\sum_{m=0}^{k}\left(\tau_{m+1}-\tau_m\right)\begin{bmatrix}\cos\boldsymbol{\eta}_{i,m+1}+\cos\boldsymbol{\eta}_{i,m}\\ \sin\boldsymbol{\eta}_{i,m+1}+\sin\boldsymbol{\eta}_{i,m}\end{bmatrix}+\mathbf{p}, k \in I^{+} \cap k \le N$
11. $\sigma_l \leftarrow 0$
12. $\sigma_a \leftarrow 0$
13. $\sigma_c \leftarrow 0$
14. $\sigma_m \leftarrow 0$
15. **for** every points index j in $\boldsymbol{\rho}_i$ **do**
16. $\phi \leftarrow \underset{k}{\operatorname{argmin}}\left(\left\|\boldsymbol{\rho}_{i,j}-\boldsymbol{\lambda}_k\right\|\right)$
17. $\sigma_l \leftarrow \sigma_l + \left\|\boldsymbol{\rho}_{i,j}-\boldsymbol{\lambda}_{\phi}\right\|$
18. $\sigma_a \leftarrow \sigma_a + \left\|\boldsymbol{\eta}_{i,j}-\boldsymbol{\upsilon}_{\phi}\right\|$
19. $\xi \leftarrow \underset{k}{\operatorname{argmin}}\left(\left\|\boldsymbol{\rho}_{i,j}-\boldsymbol{\mu}_k\right\|\right)$
20. $\sigma_c \leftarrow \sigma_c + \left\|\boldsymbol{\rho}_{i,j}-\boldsymbol{\mu}_{\xi}\right\|$
21. $\sigma_m \leftarrow \begin{cases}\infty & \text{if } \left\|\boldsymbol{\rho}_{i,j}-\boldsymbol{\mu}_{\xi}\right\| < r_0\\ \sigma_m & \text{otherwise}\end{cases}$
22. **endfor**
23. $\boldsymbol{\sigma}_i \leftarrow \begin{bmatrix}\sigma_l & \sigma_a & \sigma_c & \sigma_m\end{bmatrix}$
24. $\boldsymbol{\gamma}_i \leftarrow \boldsymbol{\sigma}_i \cdot \mathbf{w}$
25. **endfor**
26. $\zeta \leftarrow \underset{i}{\operatorname{argmin}}\left(\boldsymbol{\gamma}_i\right)$
27. $\varphi \leftarrow \boldsymbol{\delta}_{f,\zeta}$
28. **return** γ, φ

Figure 46 Scored Predicted Trajectory algorithm (continued)

2.3. High-level Autonomous Navigation Software

The speed-independent and speed-dependent autonomous navigation software are developed in this section. First, speed-independent software is introduced. This software benefits the tuning process since the traveling speed can be controlled manually. Later, speed-dependent software is presented. This software is employed as a complete path following autonomous navigation software.

2.3.1. Speed-independent Navigation System Algorithm

After the Scored Predicted Trajectory is developed, a speed-independent navigation system algorithm is now ready to be established. Figure 47 describes the workflow of the designed speed-independent autonomous navigation software.

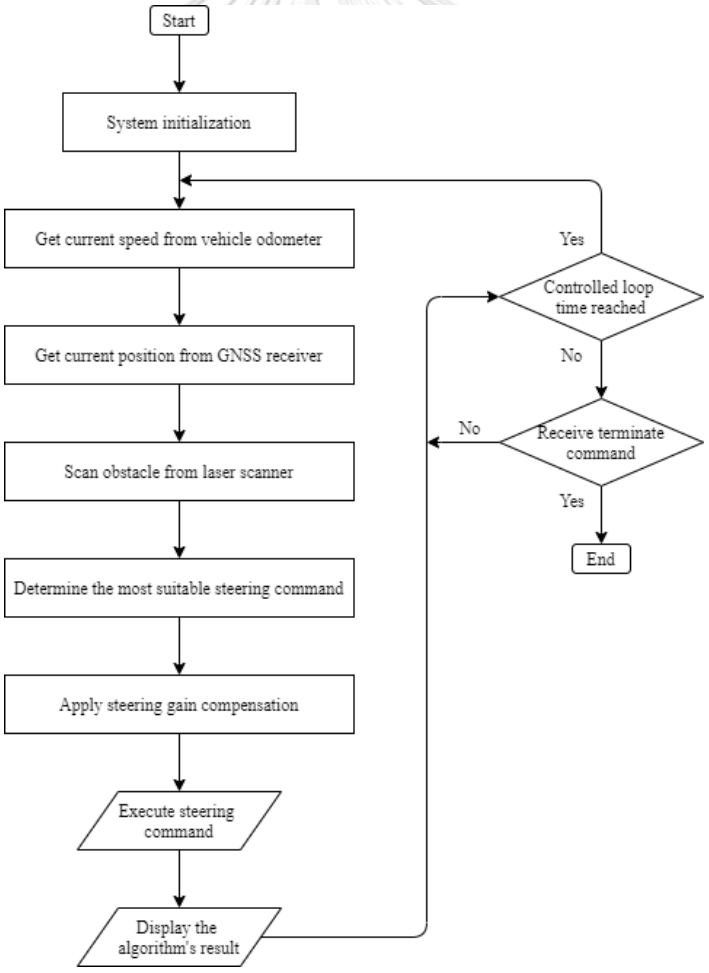


Figure 47 Speed-independent autonomous navigation software flowchart

The developed autonomous navigation software starts by initializing several parameters used in the algorithm. These parameters are listed below.

- GNSS receiver configuration
- Laser scanner configuration
- Geographical coordinate to local coordinate conversion factors
- Steering profile model
- Score predicted trajectory algorithm parameters
  - Score weights, including a linear deviation, angular deviation, and collision distance weight
  - Algorithm controlled loop time interval
  - Forward predicted trajectory distance
- Exponential gain compensation algorithm parameters
  - Initial gain increment
  - Gain increment base
  - Saturation boundary

A software enters the repeated controlled loop after initialized. This loop begins with retrieving a vehicle speed by request from a vehicle cruise control system. Then, the vehicle's geographical position is known by measuring from a GNSS receiver. Following by scanning the obstacle using a laser scanner, then all information needed is ready for a navigation algorithm.

All sensed inputs obtained by the previous step is then applied to the scored predicted trajectory algorithm. The result of this algorithm is the most suitable command steering which may be directly used as a command signal sent to a car’s steering control system. However, since the steering calibration is done under a static condition. Since the character and response of the steering control system will be different from the calibration result when used in a dynamic environment. Then, the gain compensation is required to overcome this dynamic effect. Here, the Exponential Gain Compensation algorithm is introduced to solve this problem. A detail of the algorithm will be presented later in this section.

After the gain compensation is applied to the results from the navigation algorithm, the result of gain compensation then proceeds to the steering control system. The software display then illustrates all predicted trajectories and their

corresponding scores to the user. Eventually, the software is idly waiting for the controlled loop time interval to be reached and prepare to be terminated by the user.

2.3.2. Exponential Gain Compensation

Exponential Gain Compensation, as implied by the name, is the algorithm used to determine the compensation gain for the steering command signal resulted from the main navigation algorithm. The compensation gain is defined by the following Equation (69).

$$\psi' = \alpha(\psi - \delta) + \delta \tag{69}$$

Where ψ' is the compensated steering command signal, α is the steering command signal compensation gain, ψ is the steering command signal result from a navigation algorithm, and δ is the current steering position of a vehicle.

The compensation gain is exponentially changed for every single controlled loop. The value of this gain is limited to a certain boundary by a user's predefined setting. The algorithm for Exponential Gain Compensation is described by Figure 48 below.

After the gain compensation is obtained from the algorithm presented in Figure 48, it is then substituted into equation to determine the final steering command signal, i.e. the compensated steering command signal. Finally, this command signal is then sent to the low-level steering controller. Besides, the gain transition and gain increment that also returned from the algorithm will later be used for the next calculation loop.

Input: Previous compensation gain α', previous gain increment φ', previous gain transition τ', increment base μ, current steering angle δ, expected steering angle ψ, lower saturation boundary ε_l, and upper saturation boundary ε_u

Output: Compensation gain α, gain increment φ, and gain transition τ

Figure 48 Exponential Gain Compensation algorithm

Definitions:

A previous gain transition τ' is defined as a transition direction of a previous loop's compensation gain, i.e. increase, decrease, or remain the same. τ' is positive when $\alpha' < \alpha$ in a previous calculation loop, and negative when $\alpha' > \alpha$ in a previous calculation loop, otherwise $\tau' = 0$.

Expected steering angle ψ is the steering angle forecasted by a previous control loop from a known control loop time interval

Function:

1. $\tau \leftarrow \psi - \delta$
2. **if** $\tau\tau' > 0$ **then**
3. $\lambda \leftarrow \mu\varphi'$
4. **else if** $\tau\tau' < 0$ **then**
5. $\lambda \leftarrow \frac{\varphi'}{\mu}$
6. **else**
7. $\lambda \leftarrow 0$
8. $\alpha \leftarrow \alpha' + \tau\lambda$
9. **endif**
10. **if** $\alpha > \varepsilon_u$ **then**
11. $\alpha \leftarrow \varepsilon_u$
12. $\varphi \leftarrow \varphi'$
13. **else if** $\alpha < \varepsilon_l$ **then**
14. $\alpha \leftarrow \varepsilon_l$
15. $\varphi \leftarrow \varphi'$
16. **else**
17. $\varphi \leftarrow \lambda$
18. **return** α, φ, τ

Figure 48 Exponential Gain Compensation algorithm (continued)

2.3.3. Speed-dependent Autonomous Navigation Algorithm

The previous topic deals with a speed-independent autonomous navigation system only. However, to include the speed control to the algorithm, some modifications may be applied to a speed-independent system. Also, the user override mode is included in this algorithm.

In controlling vehicle speed, the speed map is introduced. As in recording a waypoint, the speed map can be done in the same manner, i.e. by directly record a vehicle odometry speed corresponding to a certain recorded waypoint. A controlled speed is then determined by the following Equation (70) and directly sent to a vehicle cruise control system.

$$\dot{s}_c = v_j \tag{70}$$

Where $j = \underset{\varphi}{\operatorname{argmin}}\left(\left\|\boldsymbol{\rho}_i - \mathbf{p}_\varphi\right\|\right)$ which $\boldsymbol{\rho}$ is a vehicle current position and $\mathbf{p}$ is a waypoint. $\dot{s}_c$ is a controlled speed, and v is a speed map corresponding to a waypoint $\mathbf{p}$.

For the user override mode, the condition to engage this mode is determined by the scored predicted trajectory algorithm output, i.e. the user override mode engaged whenever the trajectory scores returned from the algorithm is all infinity. Equation (71) states the user override mode condition and Figure 49 presents the speed-dependent autonomous navigation software's workflow.

$$\boldsymbol{\gamma} = [\infty \;\; \infty \;\; \ldots \;\; \infty]_{1\times N} \tag{71}$$

Where $\boldsymbol{\gamma}$ is a trajectory score array of N trajectories returned from the scored predicted trajectory algorithm.

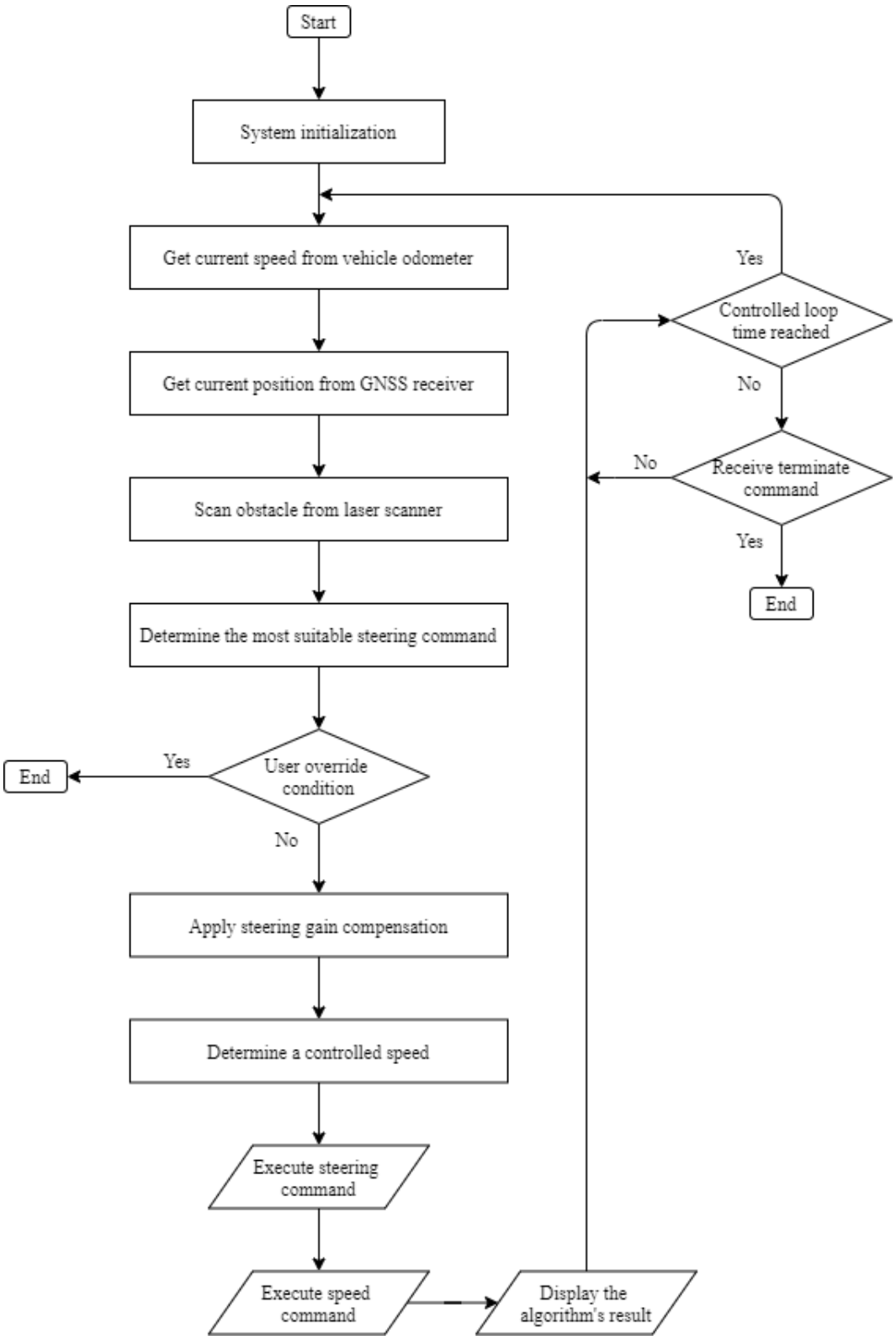


Figure 49 Speed-dependent autonomous navigation software flowchart

3. Geographic Conversion Factor Calibration

The GNSS receiver generally return the measured position in a geographical angular coordinate, however, the designed software needs a real-time position in linear cartesian coordinate. Therefore, the conversion formula must be constituted so that the position returned from the GNSS receiver can be utilized in the developed software. Even though, a distance between two points on a spherical surface can be determined by the haversine formula by given 2 spherical coordinate positions. However, to use that formula one needs to know a certain radius of the sphere. In determining the distance between two points on the earth's surface, given a longitude and latitude of those points, the haversine formula can be applied along with a known earth radius. Since the earth's radius located at the testing ground is not known exactly, then the conversion factor used in this research is determined by empirical experimentation. The procedure begins with recording a longitude and latitude position from the GNSS receiver and then brought to analyze mathematically to get a correlation between a geographical coordinate and a linear distance on the earth's surface.

In collecting data, a geographical coordinate of a certain circular path is recorded via the GNSS receiver. As shown by dots in Figure 50, a true circular path of radius 1.16 meters recorded from the testing ground appears to be an elliptic path when represented by geographical coordinate. Consequently, the ellipse Equation (72) is selected as a model for a least-square regression of the centralized circular path.

$$\alpha\lambda^2 + \beta\phi^2 = 1 \tag{72}$$

Where λ is longitude in degree, ϕ is latitude in degree, α and β are regression analysis coefficients which are determined by Equation (73).

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b} \tag{73}$$

Where $\mathbf{A} = \begin{bmatrix} \overrightarrow{\lambda^2} & \overrightarrow{\phi^2} \end{bmatrix}$ and $\mathbf{b} = \vec{1}$, which $\overrightarrow{\lambda^2}$ and $\overrightarrow{\phi^2}$ are $m \times 1$ vectors of m collected circular path's longitude and latitude, respectively.

By substitute circular path's longitude and latitude into Equation (73), the regression coefficients can be obtained as shown by Equation (74).

$$\alpha = 8782319993.5$$
$$\beta = 9228460728.4 \quad (74)$$

The earth's surface actual path should follow a 1.16 meters radius circle. Consequently, a circle equation with a 1.16 meters radius will be used as an exact path for a meter coordinate system, as shown by Equation (75), where x and y are position of points in a circular path.

$$x^2 + y^2 = 1.16^2 \quad (75)$$

Assuming that the longitude to meter and latitude to meter conversion factor is constant everywhere for a testing ground, which is considered to be an infinitesimal area compare to the earth's surface, the longitude and latitude to meter conversion factor can be defined by Equation (76) and Equation (77), where f_x and f_y are longitude to meter and latitude to meter conversion factor, respectively.

$$x = f_x \lambda \quad (76)$$

$$y = f_y \phi \quad (77)$$

By substitute Equation (76) and Equation (77) into Equation (75) and comparing the resulted equation with Equation (72), we can state the correlation between conversion factors and regression coefficients as Equation (78) and Equation (79).

$$f_x = 1.16\sqrt{\alpha} = 108657.32434 \quad (78)$$

$$f_y = 1.16\sqrt{\beta} = 111456.76004 \quad (79)$$

Note that f_x and f_y are in meters per longitudinal degrees and meters per latitudinal degrees, respectively.

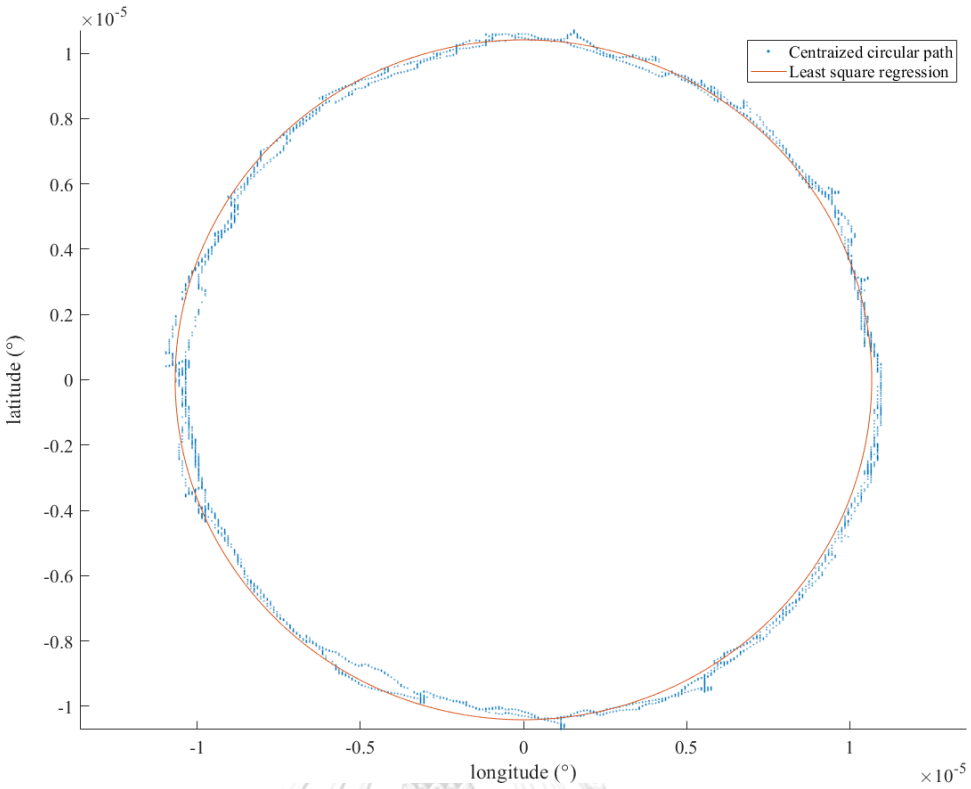


Figure 50 Recorded circular path and ellipse least square regression

CHAPTER V
SYSTEM EVALUATION EXPERIMENT

1. Experiment Setup

The experiment takes place after the prototype navigation software is completely developed. Figure 51 depicts the Graphical User Interface (GUI) of this software showing inside are the waypoint, current GNSS position, detected obstacle scan, and the predicted trajectories. Predicted trajectories are displayed in different colors corresponding to their scores determined by the scored predicted trajectory algorithm. The enlarged predicted trajectories display is shown in Figure 52.

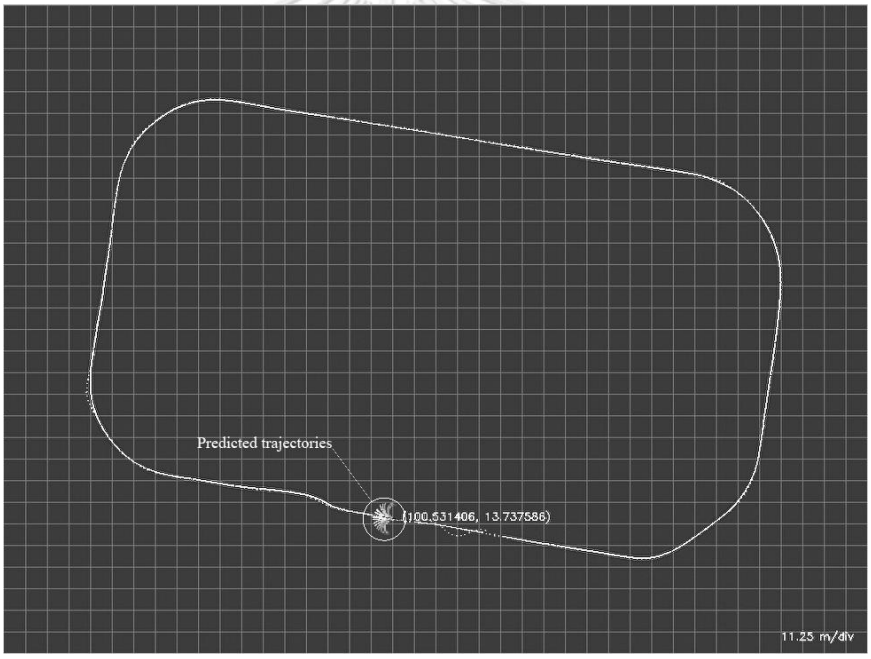


Figure 51 Prototype navigation software's graphical user interface

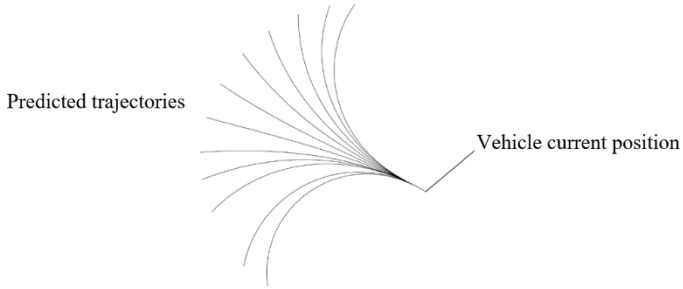


Figure 52 Enlarged predicted trajectories display

Before performing the experiment using a real vehicle, the algorithm is tested using the developed simulator software. This simulator is designed base on the assumption that the car will exactly follow the selected predicted trajectory. The result of testing the algorithm with a simulator code is the approximated controlled loop time interval, which is found playing an important role in autonomous path following characteristics, and the algorithm score weights.

The test vehicle is equipped with a 2D laser scanner and the GNSS receiver. As shown in Figure 53, 2 receiver antennas are installed along the longitudinal direction of the car. The primary antenna is installed at the same horizontal position as the rear axle, providing the position according to the assumption used in developing the scored predicted trajectory algorithm. The secondary antenna is installed 1 meter apart from the primary antenna toward the front of the car.

Figure 53 Test vehicle with the 2D laser scanner and the GNSS receiver installed

The route by which the test autonomous vehicle supposed to follow, i.e. the waypoint, is generated by manually drive and record the GNSS position along the desired path. In this experiment, the test route is set to be a close loop track located in Chulalongkorn University campus as shown in Figure 54. Furthermore, the heading angle along the desired route corresponding to the waypoint is also recorded. Note that the test experiment was performed by nighttime to avoid the undesired situation from daytime traffics inside the campus. The parameters resulted from tuning during

the experiment compared to the result from simulator tuning are shown in Table 4. Note that the dimensions of parameters in Table 4 are the same as introduced in previous sections.

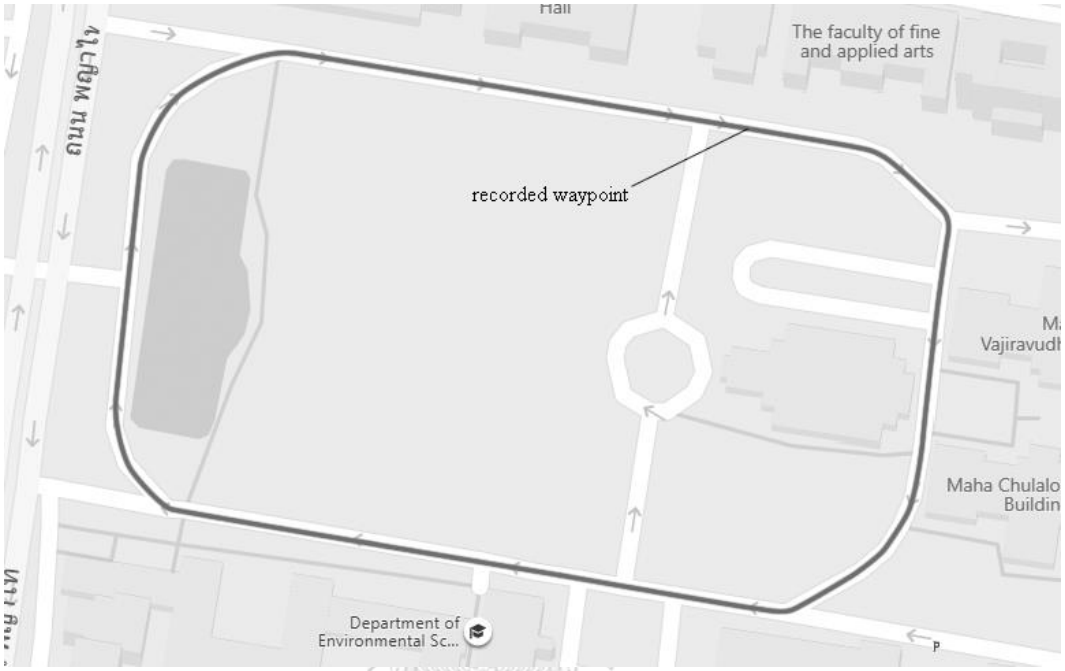


Figure 54 Test track located in Chulalongkorn University

The experiment is divided into 2 sections, the first section is the evaluation test of the autonomous path following navigation system only, and the second deals with the obstacle avoidance system only. The first section on the path following evaluation is conducted first to determine the suitable tuned score weights which are prerequisite parameters for the obstacle avoidance evaluation section.

2. Autonomous Path Following Navigation Evaluation

Since the test track is a close loop route, then the starting point can be arbitrarily chosen. From the starting point, the test vehicle will be autonomously navigated along the recorded waypoint without any intervention from the experimenter, however, still sit in a car ready to take over whenever encounter an emergency situation. The experiment is performed at 2 speeds, i.e. 10 and 15 kilometers per hour. The parameters used by the algorithm in the software are also shown in Table 4.

Table 4 Parameters in the designed autonomous software

Parameter class	Parameter name	Value	
		Experiment	**Simulator**
Device parameters	longitude conversion factor	108657.3243	-
	latitude conversion factor	111456.7600	-
	inverse effective wheelbase	4.98E-05	-
	steering position center	-0.43	-
Exponential gain compensation algorithm parameters	initial gain increment	0.005	-
	increment base	1.1	-
Scored predicted trajectory algorithm parameters	linear deviation score weight	1.5	1.5
	angular deviation score weight	0.1	0.5
	collision distance score weight	-0.1	-0.1
	controlled loop time interval	0.2	0.2
	predicted trajectory length	10	10

The actual paths recorded from the test car which autonomously navigated at the speed of 10 and 15 kilometers per hour with the corresponding waypoint are shown by Figure 55 and Figure 56, respectively. The autonomous navigation heading angle compares to the waypoint heading angle of the 10 and 15 kilometers per hour are shown in Figure 57 and Figure 58. A linear deviation is defined as a distance to the closest point on the waypoint from the test vehicle's actual position. The histogram of the linear deviation of the 10 and 15 kilometers per hour track are illustrated by Figure 59 and Figure 60. Also, an angular deviation is defined as a difference between the test car actual heading and navigation heading of the point on the waypoints closest to the car actual position. The mathematical representation of linear and angular deviations are described by Equation (80) and Equation (81), respectively.

$$\varepsilon_l = \min_{\mathbf{i}\in\mathbf{w}} \left(\left\| \boldsymbol{\rho} - \mathbf{i} \right\| \right) \tag{80}$$

$$\varepsilon_a = h - \psi \tag{81}$$

Where ε_l and ε_a denote the linear and angular deviation, respectively. $\mathbf{w}$ is the waypoint and $\boldsymbol{\rho}$ is the test vehicle's actual position. h is the actual heading and $\psi = \eta_j$ where $j = \underset{i}{\operatorname{argmin}}\left(\left\|\boldsymbol{\rho} - \mathbf{w}_i\right\|\right)$.

Considering Figure 55 to Figure 58, the developed autonomous system can perfectly navigate the car tracking almost exactly every point on the test waypoints. However, the actual heading is lightly fluctuating around the waypoint's heading angle. This small fluctuation implies that the real scenario, while the autonomous system is engaged, is somehow jerky and may be uncomfortable for the passenger in the test vehicle. According to Figure 59 and Figure 60, arithmetic means of recorded linear deviation of 10 and 15 kilometers per hour track are 0.13 and 0.20 meters. These linear deviations are considered to be relatively small compare to the operation scale, i.e. the typical lane width which is about 3.6 meters. By using this figure, then the linear deviation is merely 3.61 and 5.56 percent of a typical lane width for 10 and 15 kilometers per hour track, respectively. Also, according to Figure 61 and Figure 62, 95 percent of the angular deviation sample falls between -2.65 and 1.85 degrees in 10 kilometers per hour track and between -4.02 and 4.04 degrees in 15 kilometers per hour track, which are acceptable to be used in the objective application.

Moreover, the navigation speed also affects the tracking characteristics. In 15 kilometers per hour track, the standard deviation of the linear deviation is 0.12 meters, which is 1.68 times of one from the 10 kilometers per hour track which is 0.07 meters. Also, a linear deviation increase at higher traveling speed, from 1.13 degrees in 10 kilometers per hour track to 2.02 degrees in 15 kilometers per hour track. The original cause of this is the gradual increase of the vehicle dynamics effect. At high speed, the vehicle model developed in the previous section based on a vehicle kinematic model which is utilized in the scored predicted trajectory is not accurate and will completely fail over a certain speed.

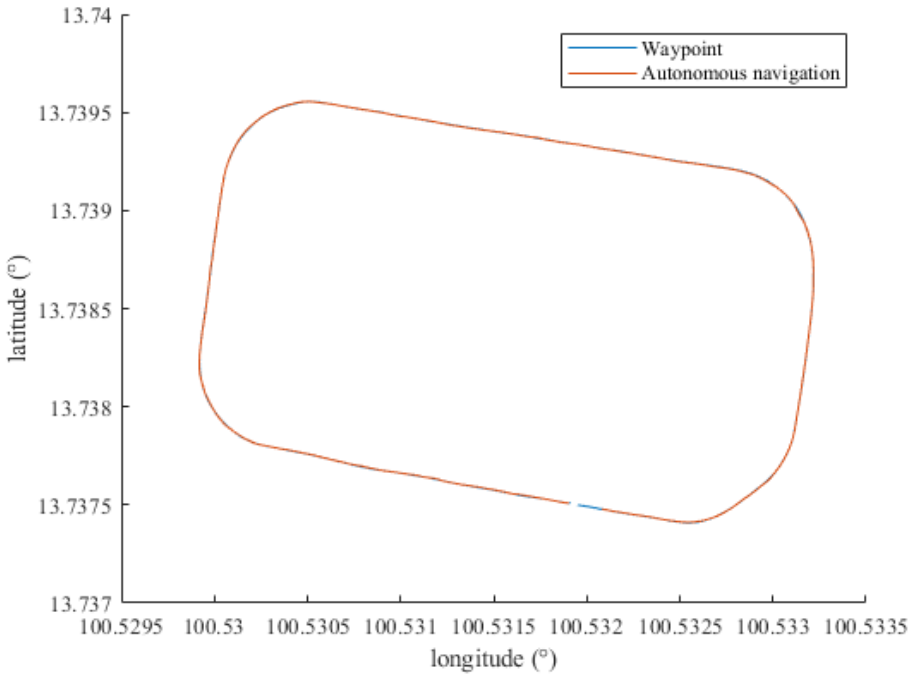


Figure 55 Autonomous path following trace at 10 kilometers per hour

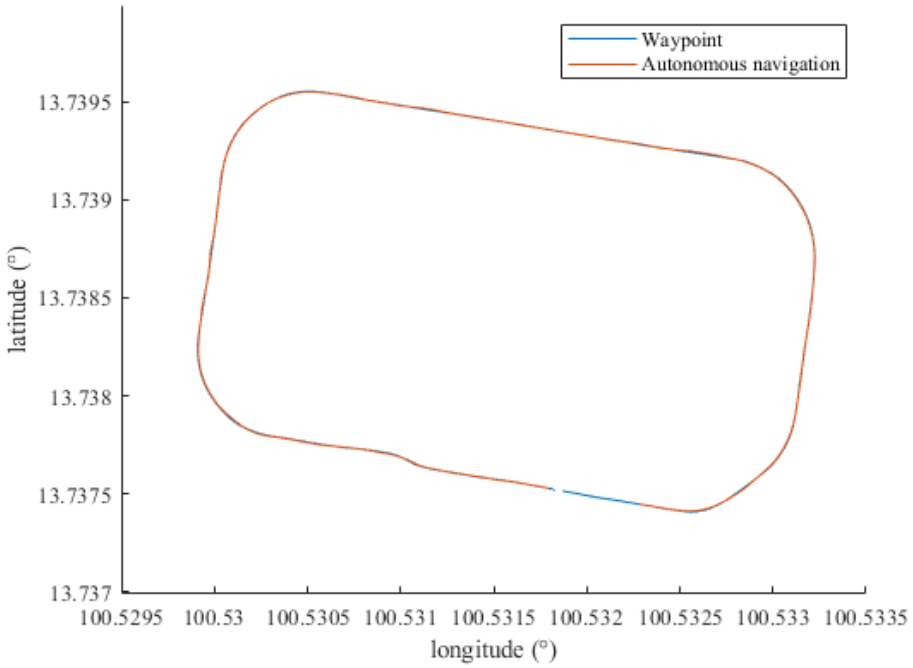


Figure 56 Autonomous path following trace at 15 kilometers per hour

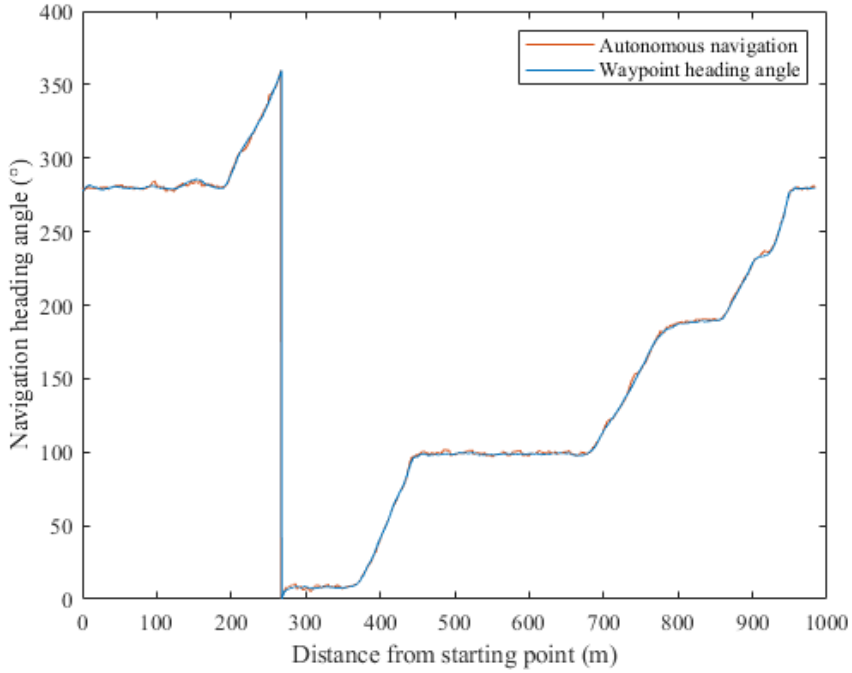


Figure 57 Autonomous path following heading angle at 10 kilometers per hour

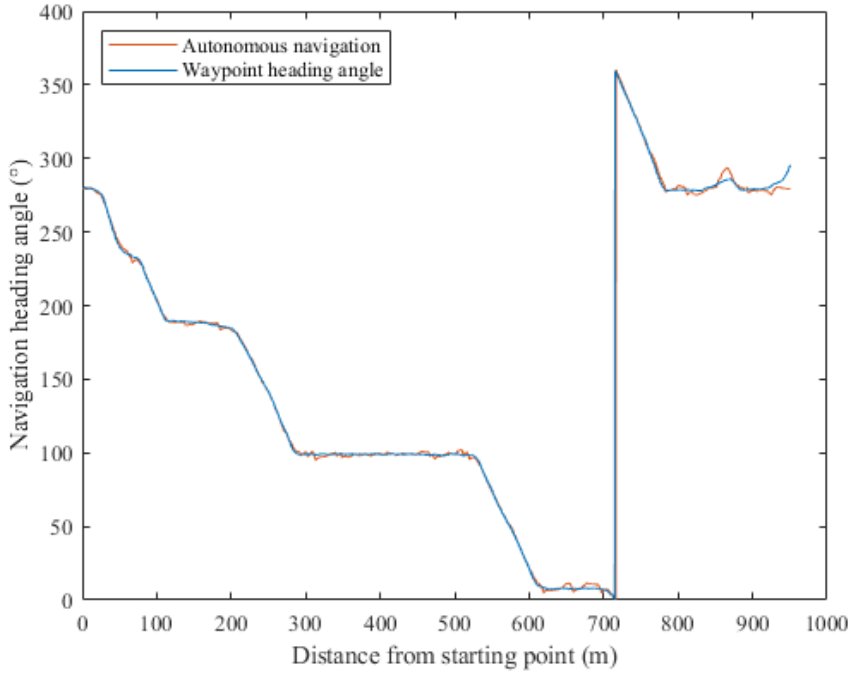


Figure 58 Autonomous path following heading angle at 15 kilometers per hour

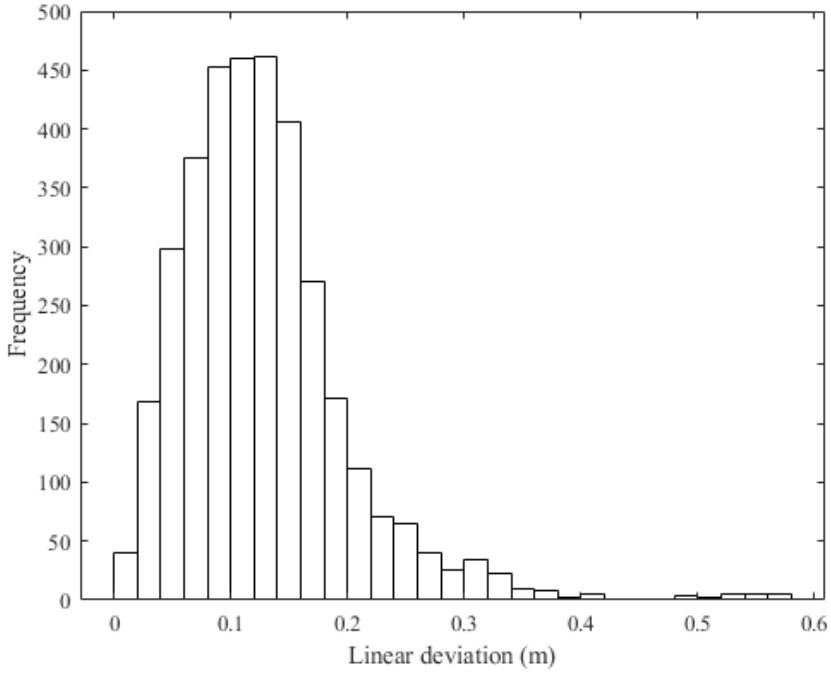


Figure 59 Linear deviation histogram of 10 kilometers per hour track

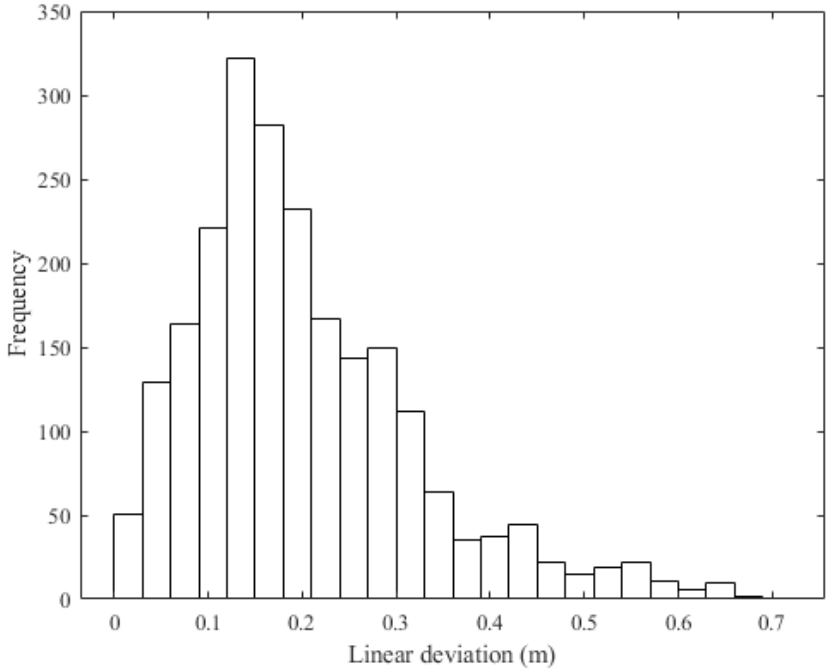


Figure 60 Linear deviation histogram of 15 kilometers per hour track

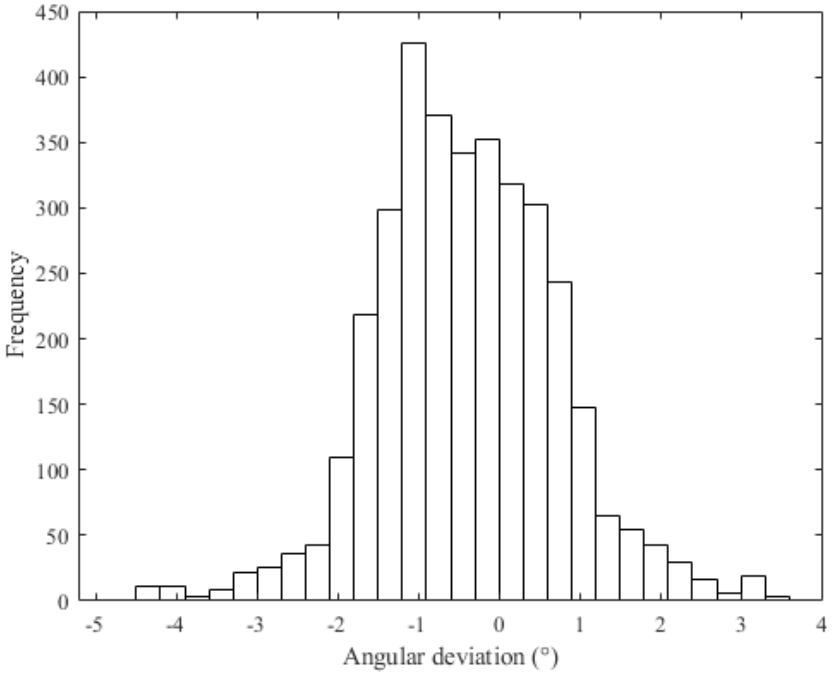


Figure 61 Angular deviation histogram of 10 kilometers per hour track

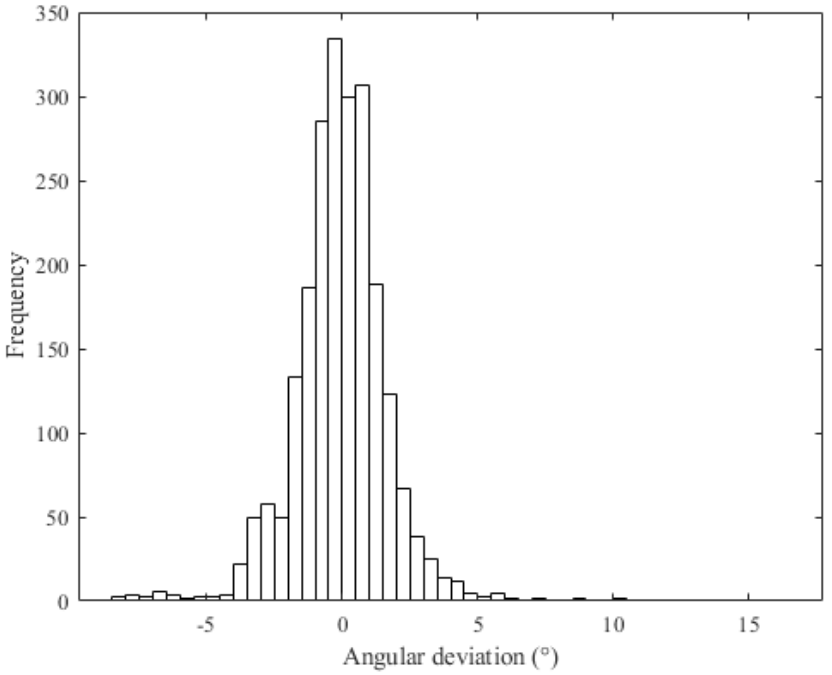


Figure 62 Angular deviation histogram of 15 kilometers per hour track

3. Obstacle Avoidance Evaluation

In this section, the path-following autonomous system is engaged by using the tuned parameters resulting from the previous experiment. Initially, the obstacle is placed in the middle of the waypoint, hinder the test vehicle from tracing such waypoint. Then, the test vehicle is launched, and autonomously follows the waypoint towards the obstacle.

Figure 63 shows the recorded position of the obstacle using a GNSS receiver. The recorded data scatter around a certain area as shown in the figure. However, such an area is relatively small compared to the application scale, i.e. about 4 centimeters in a horizontal direction and 2 centimeters in a vertical direction. The centroid of recorded data is employed to represents the position of the obstacle. Moreover, the obstacle shape is considered to be a circle with a diameter of 0.4 meters.

The experiment is conducted twice, both using the vehicle speed of 15 kilometers per hour. The trace record of 2 experiments is shown in Figure 64 and Figure 65. It can be seen from these figures that in the beginning, the test car was tracing the waypoint moving from the right side of the figure toward the left. Then, the test car detected the obstacle and avoided the impending collision by refusing to follow the waypoint and steered itself toward its left. Thereafter, when the obstacle disappears or does not obstruct the car from tracing the waypoint, the test car then autonomously converged to the waypoint again as shown in by the left portion of these figures.

Figure 66 and Figure 67 depict the linear deviation from the waypoints of both experiments. These figures affirm that the collision avoidance algorithm does work properly. Considering both figures, the test vehicle initially tracing the waypoint by keeping the linear deviation to the waypoint to be about 0.15 and 0.2 meters in the first and second experiment, respectively. Subsequently, at about 10 meters away from the starting point, the linear deviation starts increasing imply that the developed autonomous navigation algorithm realizes the impending collision and starts ignoring to follow the waypoint. Eventually, the test car merges with the waypoint again around 35 meters away from the beginning.

Figure 68 and Figure 69 show a displacement to the obstacle of the first and second experiment, respectively. The shortest displacement to the obstacle of the first

and second experiment are 1.19 and 1.22 meters. Figure 70 and Figure 71, obtained by combining Figure 66 and Figure 67 with the corresponding Figure 68 and Figure 69, shows the plot of linear deviation against displacement to the obstacle of the first and second experiment, respectively. According to these figures, the obstacle avoidance algorithm actives when the obstacle is about 10 meters away from the test vehicle, and the linear deviation increases the same time the displacement to obstacle decrease.

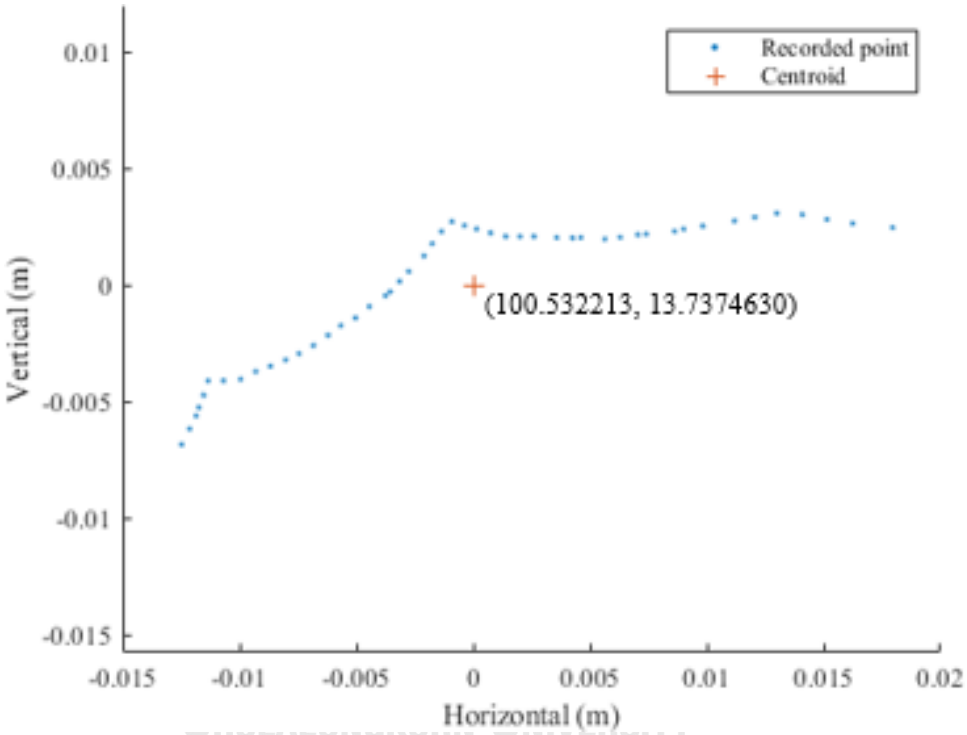


Figure 63 Recorded obstacle position

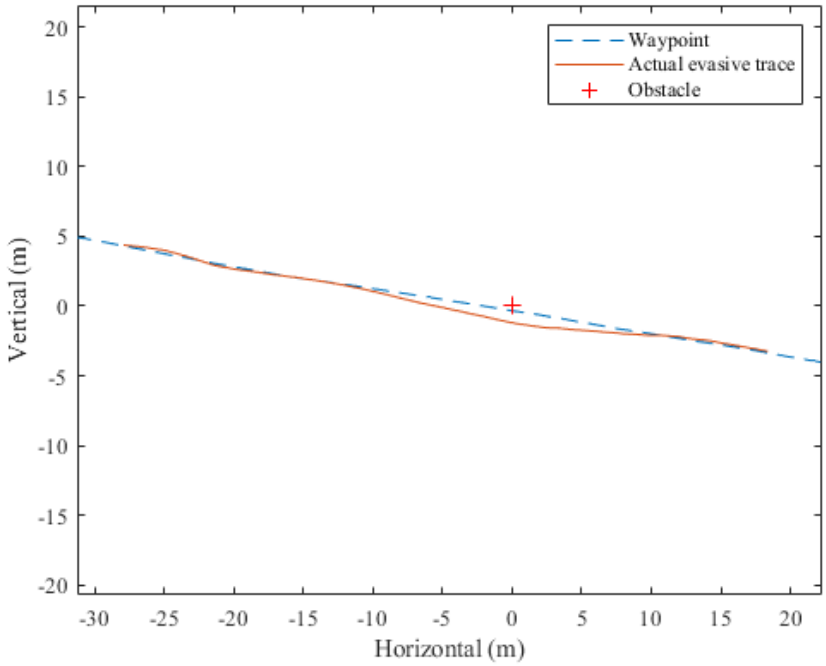


Figure 64 Autonomous path following with obstacle avoidance result (1st experiment)

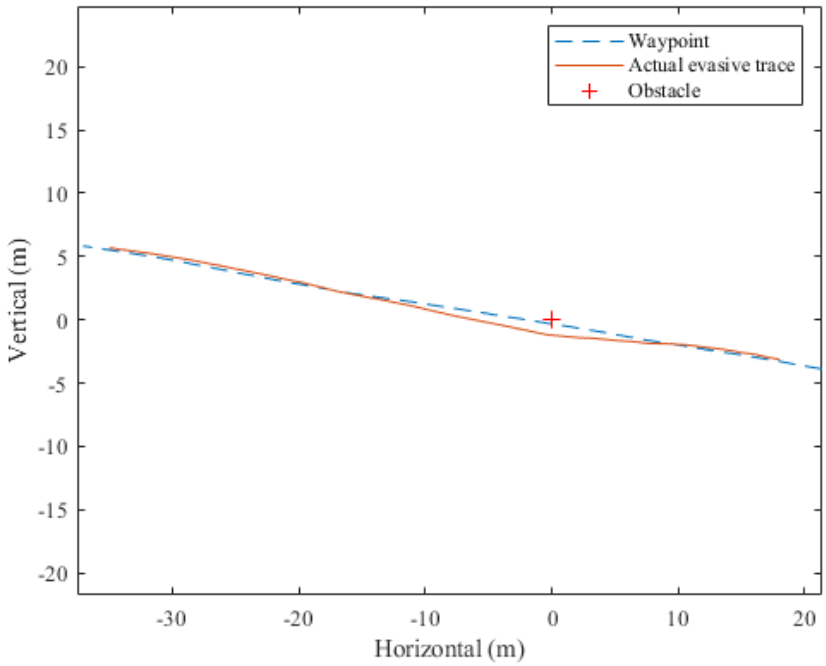


Figure 65 Autonomous path following with obstacle avoidance result (2nd experiment)

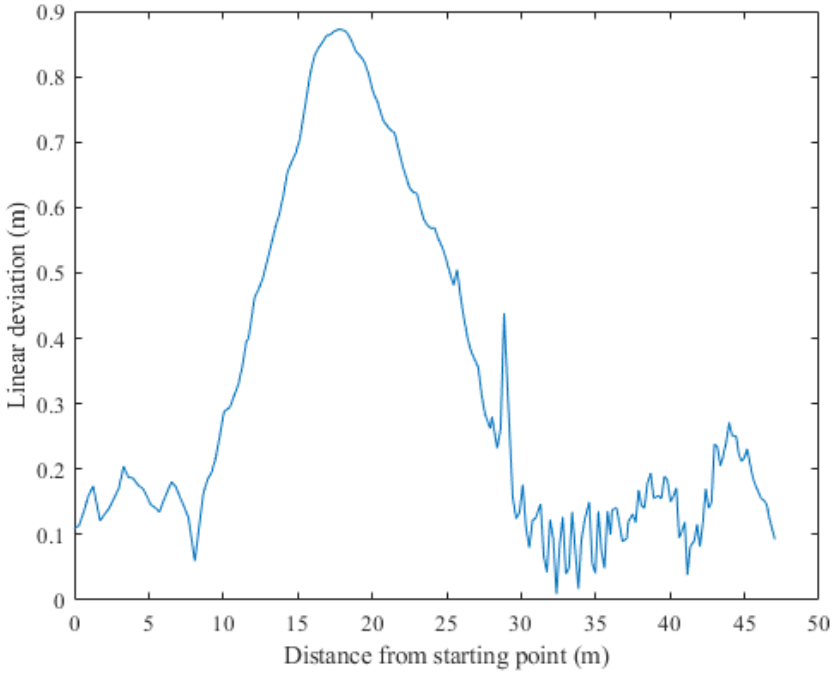


Figure 66 Linear deviation resulted from obstacle avoidance (1st experiment)

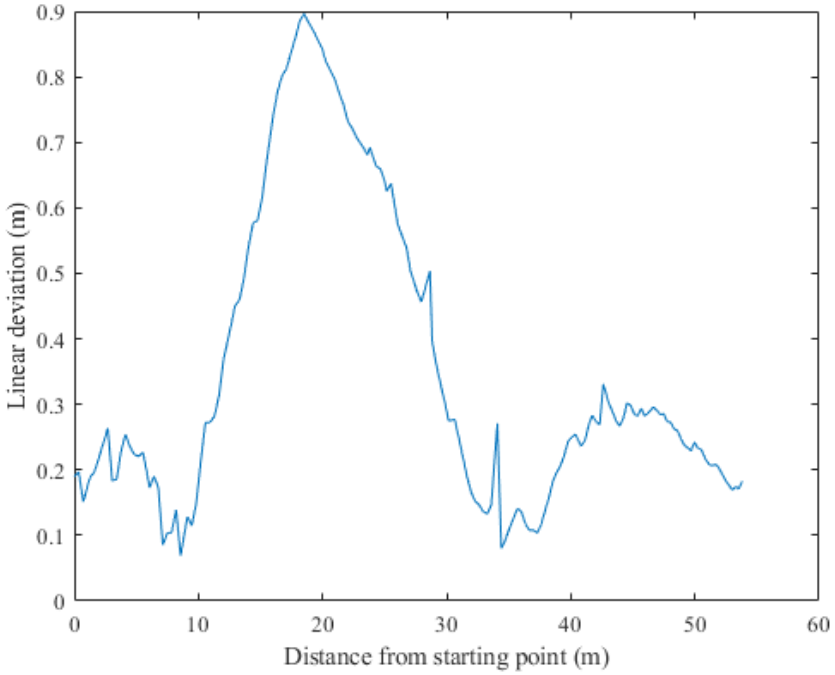


Figure 67 Linear deviation resulted from obstacle avoidance (2nd experiment)

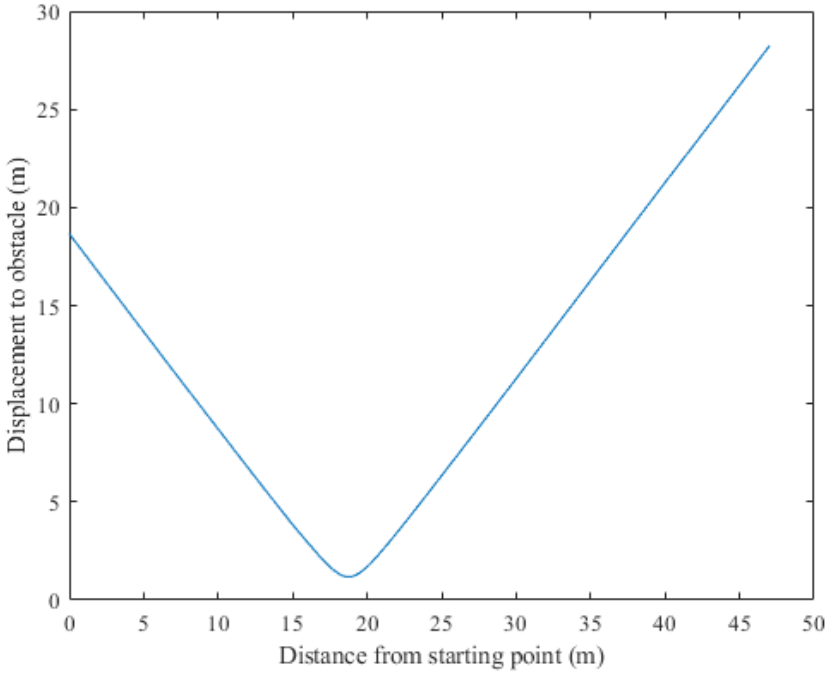


Figure 68 Displacement to obstacle at any instance (1st experiment)

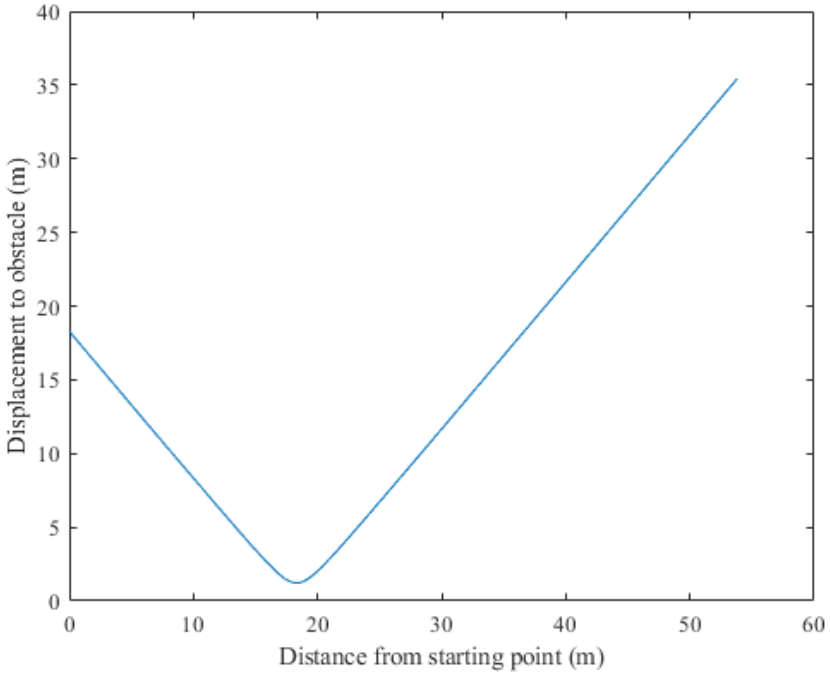


Figure 69 Displacement to obstacle at any instance (2nd experiment)

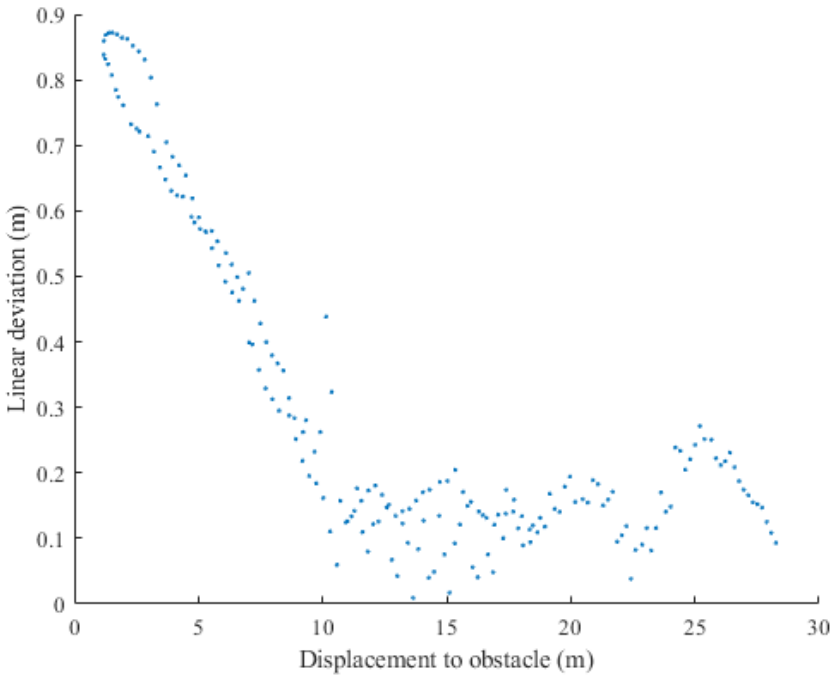


Figure 70 Linear deviation versus displacement to obstacle (1st experiment)

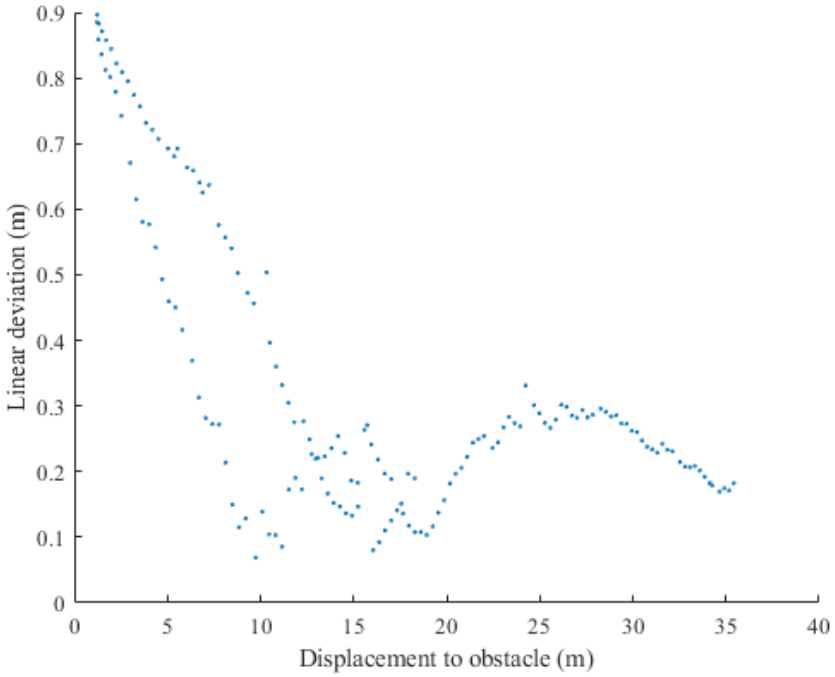


Figure 71 Linear deviation versus displacement to obstacle (2nd experiment)

4. Discussion

In low-speed application, the path following algorithm with the obstacle avoidance function, i.e. the scored predicted trajectory, gives a satisfactory result. However, unwanted jerky driving still presents when the higher speed is attempted, lead to an uncomfortable ride for the passenger. This problem can be solved by introducing the more sophisticated vehicle model, including all involved dynamics systems, to the algorithm so that the more accurate predicted trajectory can be determined. Still, only the exponential gain compensation deals with a steering system model deviation due to changes in the operating speed. Then, the applicable range of this system is limited by the deviation in a vehicle model since no model correction is applied in this part of the system. Also, in determining the possible trajectories, the speed is assumed constant at one certain speed, i.e. current measured speed. If different speeds are included in the algorithm, then more plausible trajectories can be determined, inducing more possibility to encounter the more suitable trajectory.

The angular deviation score weight plays an important role in controlling the traveling direction. By setting this weight to be zero, the traveling direction defined by the waypoint heading angle is disregarded, the car then can trace the waypoint in either same or oppose the prescribed direction. However, setting the angular deviation score weight too high cab results in a constant offset distance to the waypoint or even leaving the waypoint since the predicted trajectory with a high angular deviation score will overcome the trajectory with a high linear deviation score which leads the car back to the waypoint. A suitable angular deviation score weight will ensure the waypoint tracing smoothness, thus increase ride quality.

According to Figure 70 and Figure 71, the obstacle avoidance function seems to overshadow the waypoint tracing algorithm at about 10 meters away to the obstacle. This 10-meter distance relates to the predicted trajectory length that is set to be 10 meters in the experiment. This distance can be reduced such that a reasonable forward-collision distance can be achieved. Moreover, the least displacements to the obstacle of the first and second experiment, which are 1.19 and 1.22 meters, respectively, imply that the algorithm operate properly since the critical obstacle distance, described in the previous section, is set to be 1.0 meters and the obstacle used in this experiment has a width of 0.2 meters.

This autonomous navigation algorithm can be improved by revising some parts of the algorithm. For example, in the obstacle avoidance part, it has been pointed out that the algorithm suddenly changes its priority whenever the predicted trajectory length is reached. Here, if the continuous function is applied to the critical obstacle distance scoring procedure to obtain the suitable forward-collision distance. Furthermore, the linear deviation variant function can be introduced to the angular deviation scoring process to eliminate the constant offset from a waypoint and prevent the car from leaving the waypoint.

CHAPTER VI
CONCLUSIONS AND RECOMMENDATIONS

1. Conclusions

The three main objectives have been done in this research. The first objective is to develop a low-level control system, including the steering control system and the speed control system. Second, the high-level controller software for the autonomous path-following navigation system using the Global Navigation Satellite System (GNSS) was developed. The third objective that has been accomplished is the development of the obstacle avoidance software.

First of all, the ultra-small electric vehicle, i.e. the Toyota COMS, has been modified for a speed control system and a steering control system. In the speed control modification, the existed braking system was modified by introducing a designed brake actuator module to the original brake line in series. This modified brake system allows both driver and autonomous system to apply pressure to a vehicle brake line. Also, the accelerator pedal has been modified allowing the autonomous system to control the vehicle acceleration whenever required. The steering control system was also installed in this low-level modification. At the moment, the modified vehicle was ready to be autonomously navigated using the high-level navigation software. Then, the high-level autonomous navigation using GNSS was developed base on the kinematic model of the vehicle. The developed algorithm for path following navigation has been named the Scored Predicted Trajectory and implemented in the high-level software. The algorithm deals with both waypoint tracing and obstacle avoidance tasks at the same time, thus satisfies both the second and the third objective stated above. Furthermore, to cope with a model change impacted by vehicle speed, the Exponential Gain Compensation algorithm is developed and implemented to the high-level software. Eventually, the evaluation experiment was performed. The test site was located in Chulalongkorn University campus. The test track was set to be one the close-loop road inside the campus. The experiment on path-following performance evaluation was conducted by launching the test car, by which the developed software was deployed, to the test track. The

result shows satisfactory performance with an average linear deviation from the waypoint of 0.07 meters when the speed is 10 kilometers per hour and 0.12 meters when the speed is 15 kilometers per hour. The desirable angular deviation of about 3 degrees in 10 kilometers per hour test and about 4 degrees in 15 kilometers per hour test resulted from this experiment as well. The experiment on obstacle avoidance was performed later. In the experiment, the obstacle was placed in the middle of the waypoint. Then, the test car was launched and autonomously navigated toward the obstacle. The result shows that the test vehicle deals with the obstacle that blocks the waypoint by performing a steering evasive maneuver, as expected, keeping a minimum distance of 1.2 meters away from the obstacle which is the exact value configured in the navigation software.

From all mentioned above, the concept of using GNSS, with the Real-Time Kinematic (RTK) technique, as a sole localization system for the autonomous vehicle was proved that can work perfectly, even some portions of the route are covered by trees or surrounded by buildings. However, the obstacle sensing device, which is a 2D laser scanner in this research, still vital to the collision avoidance system.

2. Recommendations

The vehicle model used in the algorithm is the key to high-speed application performance. Thus, deploying a more sophisticated vehicle model will extend the application range to a higher speed than 15 kilometers per hour. Also, some parts of the developed high-level software can be accelerated by applying the parallel computing technique. For instance, the exponential gain compensation which can be accelerated by the parallel computing technique will give a fast response to a change in vehicle speed when a high sample rate is utilized.

Besides, this research employs an expensive 2D laser scanner. However, since the developed system is intended to be used in a low-speed application, then an expensive laser scanner can be replaced by a lower class one, therefore, increasing the potential to be commercialized for use in the controlled environment, e.g. a low-density residential area, factory, university campus, etc.

REFERENCES

1. SAE, *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*. 2014, SAE.
2. Tom M. Gasser, D.W., *BASt-study: Definitions of Automation and Legal Issues in Germany*. 2012.
3. Forum, i., *Automation in Road Transport*. 2013.
4. Rajamani, R., *Vehicle Dynamics and Control*. 2012, New York: Springer.
5. Hayashi, R., et al., *Autonomous collision avoidance system by combined control of steering and braking using geometrically optimised vehicular trajectory*. Vehicle System Dynamics, 2012. **50**(sup1): p. 151-168.
6. Larbwisuthisaroj, S., *The Vehicle Steering and Braking Guidance using Vehicle Dynamics Approach for Forward Collision Warning System.* Atrans Symposium on Transportation for a Better Life: Safe and Smart City, 2016.
7. Madhavan, R. and H.F. Durrant-Whyte, *Natural landmark-based autonomous vehicle navigation.* Robotics and Autonomous Systems, 2004. **46**(2): p. 79-95.
8. Bais, A., R. Sablatnig, and J. Gu. *Single landmark based self-localization of mobile robots*. in *The 3rd Canadian Conference on Computer and Robot Vision (CRV'06)*. 2006.
9. Kartashov, D., A. Huletski, and K. Krinkin. *Fast artificial landmark detection for indoor mobile robots*. in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*. 2015.
10. Salahuddin, M.A., et al. *An efficient artificial landmark-based system for indoor and outdoor identification and localization*. in *2011 7th International Wireless Communications and Mobile Computing Conference*. 2011.
11. Yang, G., G. Anderson, and E. Tunstel. *A RFID Landmark Navigation Auxiliary System*. in *2006 World Automation Congress*. 2006.
12. Yosuke, S., et al. *Machine learning approach to self-localization of mobile robots using RFID tag*. in *2007 IEEE/ASME international conference on advanced intelligent mechatronics*. 2007.
13. Collins, B.H.-W.L., *Global Positioning System*. 5th ed. 2001, Wien, New York: Springer-Verlag Wien GmbH.
14. Zhu, N., et al., *GNSS Position Integrity in Urban Environments: A Review of Literature.* IEEE Transactions on Intelligent Transportation Systems, 2018: p. 1-17.
15. Jackson, J., B. Davis, and D. Gebre-Egziabher. *A performance assessment of low-cost RTK GNSS receivers*. in *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. 2018.
16. Ashley W. Stroupe, K.S., and Tucker Balch. *Constraint-Based Landmark Localization*. in *RoboCup 2002: Robot Soccer World Cup VI*. 2002. Springer-Verlag Berlin Heidelberg New York.
17. Pakdaman, M., M.M. Sanaatiyan, and M.R. Ghahroudi. *A line follower robot from design to implementation: Technical issues and problems*. in *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*. 2010.
18. Bajestani, S.E.M. and A. Vosoughinia. *Technical report of building a line follower robot*. in *2010 International Conference on Electronics and Information*

Engineering. 2010.
19. Arabi, A.A., et al. *Autonomous Rover Navigation Using GPS Based Path Planning*. in *2017 Asia Modelling Symposium (AMS)*. 2017.
20. Engin, M. and D. Engin. *Path planning of line follower robot*. in *2012 5th European DSP Education and Research Conference (EDERC)*. 2012.
21. Fox, D., W. Burgard, and S. Thrun, *The dynamic window approach to collision avoidance.* IEEE Robotics & Automation Magazine, 1997. **4**(1): p. 23-33.
22. Awange, J., *Environmental Monitoring using GNSS*. 2012.
23. *POS LVX DUAL GNSS-INERTIAL SOLUTION FOR HIGH-ACCURACY POSITIONING AND ORIENTATION ON AUTONOMOUS GROUND VEHICLES*. 2018, Applanix, A Trimble Company.
24. *LMS5xx LASER MEASUREMENT SENSORS* 2015, SICK AG.
25. *LEGION Y530 (15) LAPTOP GAMING, REFINED*. 2018, Lenovo.
26. *COMS P.COM/B.COM SERVICE MANUAL*. 2017, TOYOTA AUTO BODY.
27. *Mediceo introduces ultra-compact mobility at a new logistics base*. 2013 [cited 2020; Available from: https://www.logi-today.com/69770.

APPENDICES

APPENDIX A
NAVIGATION SENSORS

POS LVX dual GNSS-inertial solution for high-accuracy positioning and orientation on autonomous ground vehicles[23]

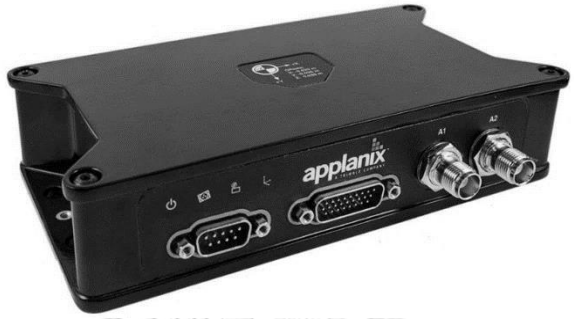


Figure 72 POS LVX dual GNSS-inertial[23]

Technical Specifications

- Advanced Applanix IN-Fusion™ GNSS-Inertial integration technology
- Solid-state MEMS inertial sensors with Applanix SmartCal™ compensation technology
- Advanced Trimble GNSS survey technology
- Position antenna based on 336 Channels Maxwell 7 chip:
  - GPS: L1 C/A, L2E, L2C, L5
  - BeiDou B1, B2, B31
  - GLONASS: L1 C/A, L2 C/A, L3 CDMA2
  - Galileo3: E1, E5A, E5B, E5AltBOC, E62
  - IRNSS L5
  - QZSS: L1 C/A, L1 SAIF, L1C, L2C, L5, LEX
  - SBAS: L1 C/A, L5
  - MSS L-Band: OmniSTAR, Trimble RTX
- Vector Antenna based on second 336 Channel Maxwell 7 chip:
  - GPS: L1 C/A, L2E, L2C, L5
  - BeiDou B1, B2, B31

- GLONASS: L1 C/A, L2 C/A, L3 CDMA2
- Galileo3: E1, E5A, E5B, E5AltBOC, E62
- IRNSS L5
- QZSS: L1 C/A, L1 SAIF, L1C, L2C, L5, LEX

- ❖ High precision multiple correlator for GNSS pseudorange measurements
- ❖ Advanced RF Spectrum Monitoring and Analysis
- ❖ Unfiltered, unsmoothed pseudorange measurements data for low noise, low multipath error, low time domain correlation and high dynamic response
- ❖ Very low noise GNSS carrier phase measurements with <1 mm precision in a 1 Hz bandwidth • Proven Trimble low elevation tracking technology
- ❖ 100 Hz real-time position and orientation output
- ❖ IMU data rate 200 Hz
- ❖ Navigation output format: ASCII (NMEA-0183), Binary (Trimble GSOF)
- ❖ Supported Reference input: – CMR, CMR+, sCMRx, RTCM 2.1, 2.2, 2.3, 3.0, 3.1, 3.2 • Support for POSPac MMS post-processing software (sold separately)
- ❖ No export permit required
- ❖ Supports Fault Detection & Exclusion (FDE), Receiver Autonomous Integrity Monitoring (RAIM)

Performance Specification

Table 5 Performance without GNSS outages

Performance	**SPS**	**DGPS**	**RTK**
Position (m)	1.5 H	0.1 H	0.02 H
	3.0 V	0.5 V	0.05 V
Roll & pitch (deg)	0.04	0.03	0.03
True heading (deg)	0.12	0.09	0.09

Table 6 Performance with 1 km or 1 minute GNSS outages

Performance	SPS	DGPS	RTK
Position (m)	2.0 H	2.0 H	1.0 H
	5.0 V	3.0 V	2.0 V
Roll & pitch (deg)	0.09	0.09	0.09
True heading (deg)	0.35	0.35	0.30

LMS511 Laser measurement sensor[24]

Figure 73 LMS511 Laser measurement sensor[24]

Table 7 LMS511 laser measurement sensor technical specifications

Specification	Detail
Field of application	Outdoor
Version	Mid-Range
Variant	Lite
Resolution power	Standard Resolution
Light source	Infrared (905 nm)
Laser class	1 (IEC 60825-1:2014) EN 60825-1:2014
Field of view	190 °
Scanning frequency	25 Hz / 35 Hz / 50 Hz / 75 Hz
Angular resolution	0.25°, 0.5°, 1°

Table 7 LMS511 laser measurement sensor technical specifications (continued)

Specification	Detail
Heating	Yes
Operating range	80 m
Max. range with 10 % reflectivity	40 m
Spot size	11.9 mrad
Amount of evaluated echoes	2

Table 8 LMS511 laser measurement sensor performance specification

Performance	Detail
Fog correction	Yes
Response time	≥ 13 ms
Detectable object shape:	Almost any
Systematic error	± 25 mm (1 m ... 10 m)
	± 35 mm (10 m ... 20 m)
	± 50 mm (20 m ... 30 m)
Statistical error	± 14 mm (20 m ... 30 m)
	± 6 mm (1 m ... 10 m)
	± 8 mm (10 m ... 20 m)
Integrated application	Field evaluation
Number of field sets	4 fields
Simultaneous processing cases	4

APPENDIX B
PROCESSING UNIT

Lenovo Legion Y530-15ICH[25]

Figure 74 Lenovo Legion Y530-15ICH[25]

Table 9 Lenovo Legion Y530-15ICH technical specification

Specification	Detail
Manufacturer	Lenovo
Model	Legion Y530-15ICH
Central Processing Unit (CPU)	Intel(R) Core(TM) i5-8300H 2.30GHz
Random-Access Memory (RAM)	20.0 GB
Operating System	Windows 10 Home 64bit

APPENDIX C
CAR SPECIFICATION

Toyota COMS ZAD-TAK30-DS[26]

Figure 75 Toyota COMS ZAD-TAK30-DS[27]

Table 10 Toyota COMS ZAD-TAK30-DS general specification

Specification	Detail
Manufacturer	Toyota
Model	ZAD-TAK30-DS
Curb weight (kg)	420
Gross vehicle weight (kg)	475
Fuel type	Electricity
Driving range (km)	50
Minimum turning radius (m)	3.2
Maximum payload (kg)	30
Maximum passengers	1
Total length (mm)	2395
Total width (mm)	1095
Total height (mm)	1495

Table 10 Toyota COMS ZAD-TAK30-DS general specification (continued)

Specification	Detail
Wheelbase (mm)	1530
Track width (mm)	930 (front), 920 (rear)
Tire	145/70R12 69Q (S)
Traction battery	6 Lead-acid batteries 12V 52Ah
Auxiliary battery	1 Lead-acid battery 12V 17Ah
Charging times (hr)	6
AC charging voltage (V)	100
AC charging current (A)	9.5

Table 11 Toyota COMS ZAD-TAK30-DS motor specification

Specification	Detail
Type	Permanent magnet synchronous motor
Typical voltage (V)	72
Typical power (kW)	0.59
Controller	Transistor inverter
Maximum power (kW)	5.0 / 1200 ~ 1400 rpm
Maximum torque (Nm)	below 40 / 1200 rpm

Table 12 Toyota COMS ZAD-TAK30-DS steering mechanism specification

Specification	Detail
Steering wheel diameter (mm)	350
Gear system	Rack and pinion
Steering angle	38° (inner)
	36° (outer)
Lock mechanism	Steering wheel lock

Table 13 Toyota COMS ZAD-TAK30-DS braking mechanism specification

Specification	Detail
type	hydraulic drum (front)
	hydraulic drum (rear)
Master cylinder inner diameter (mm)	17.4
Wheel cylinder inner diameter (mm)	17.4 (front)
	17.4 (rear)
Brake fluid grade	DOT3

APPENDIX D
DEVELOPED MASTER CYLINDER

Figure 76 Master cylinder assembly view

APPENDIX E

LOW-LEVEL CONTROLLER CIRCUIT

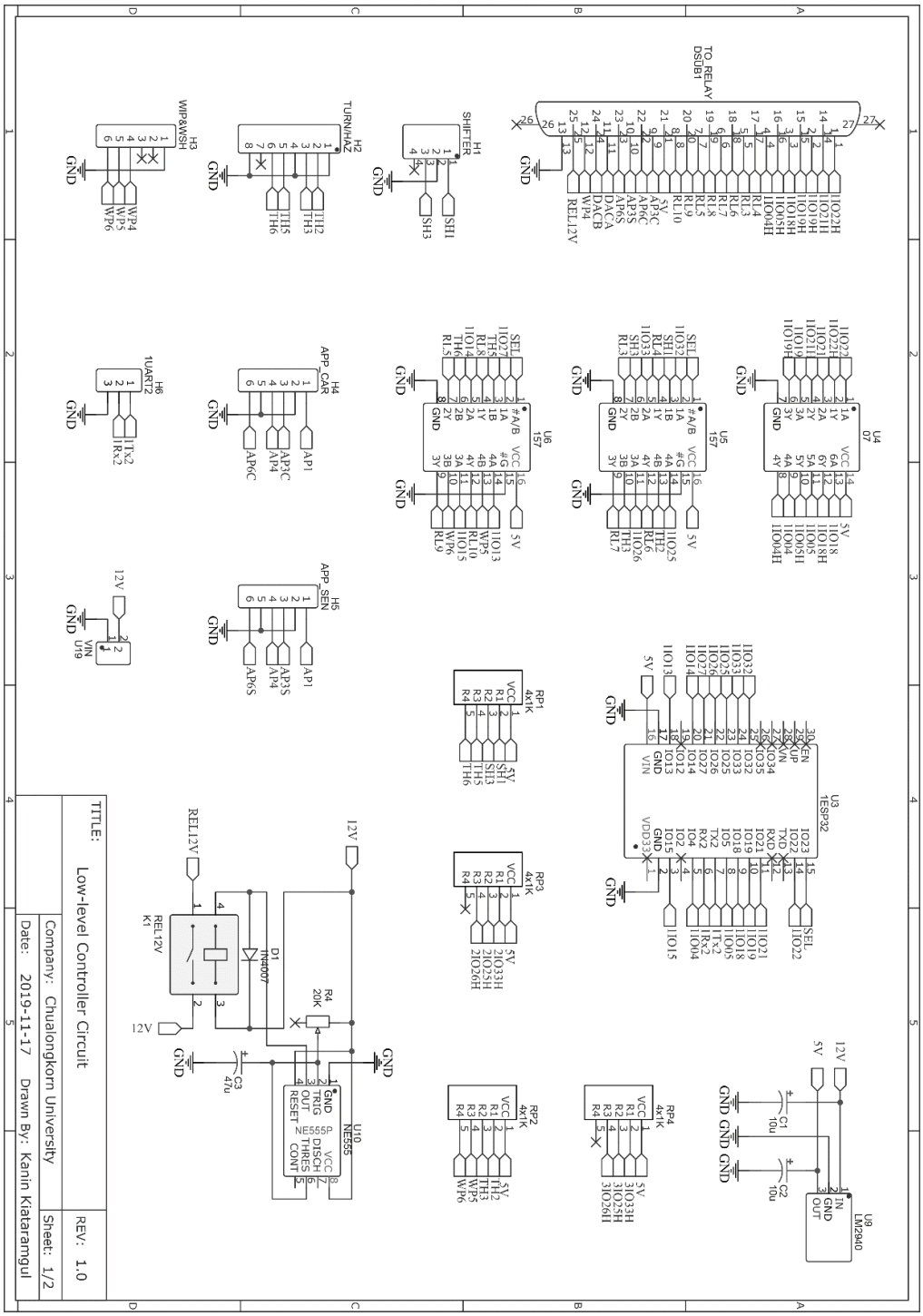


Figure 77 Low-level controller circuit diagram

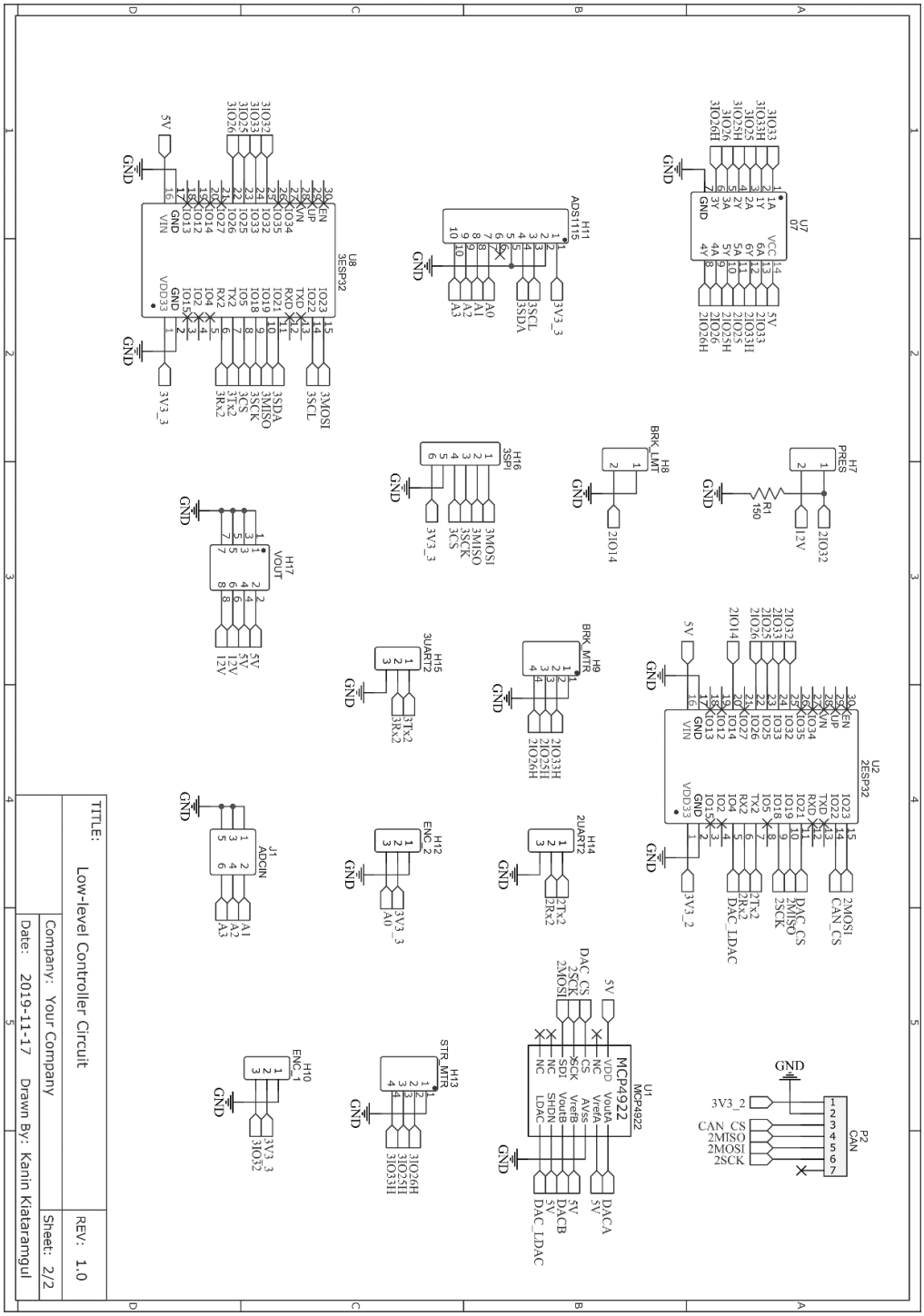


Figure 77 Low-level controller circuit diagram (continued)

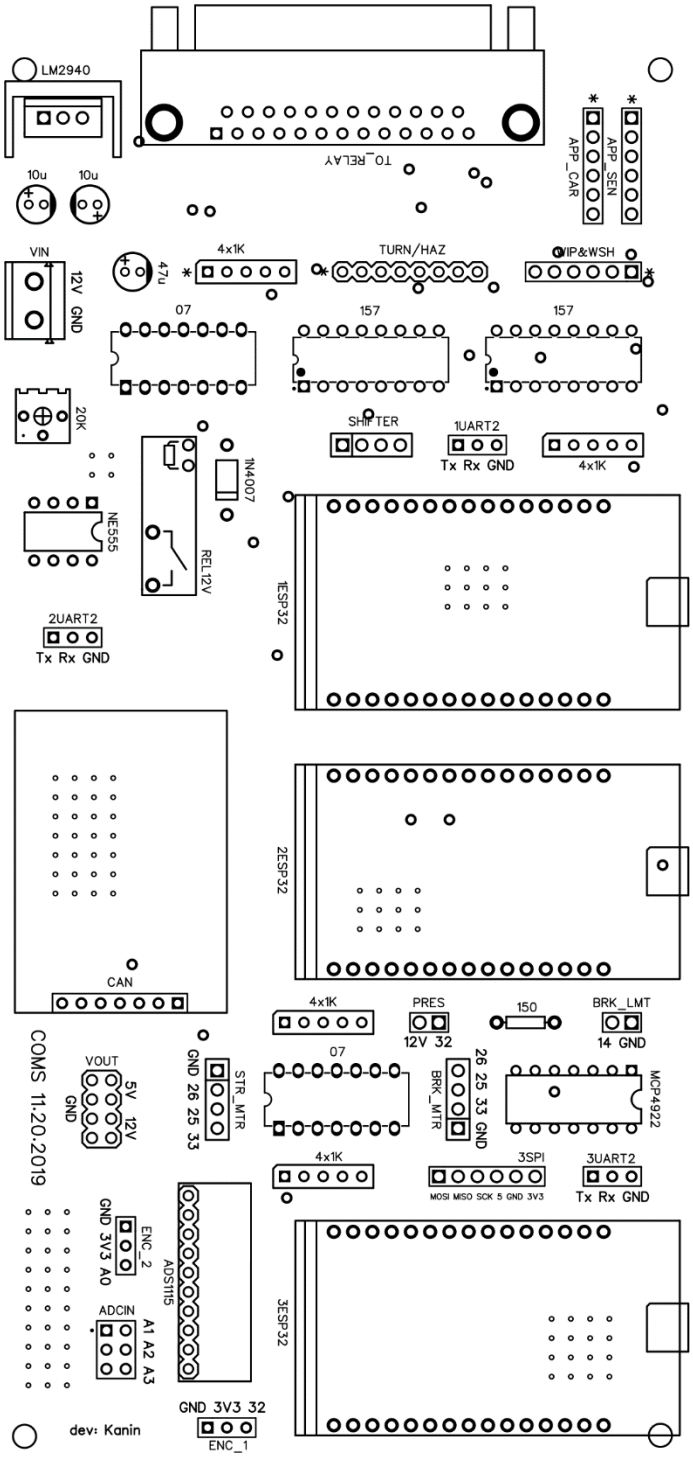


Figure 78 Low-level controller printed circuit board component outline

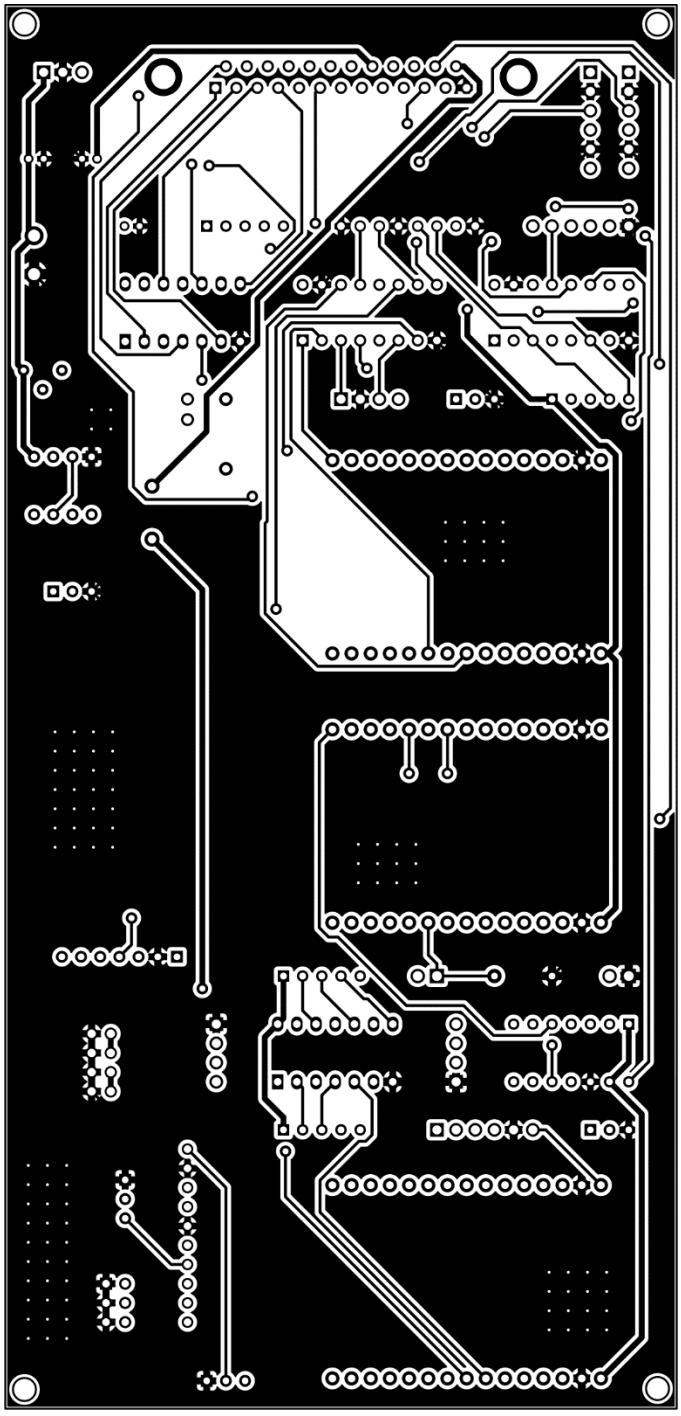


Figure 79 Low-level controller printed circuit board top layer

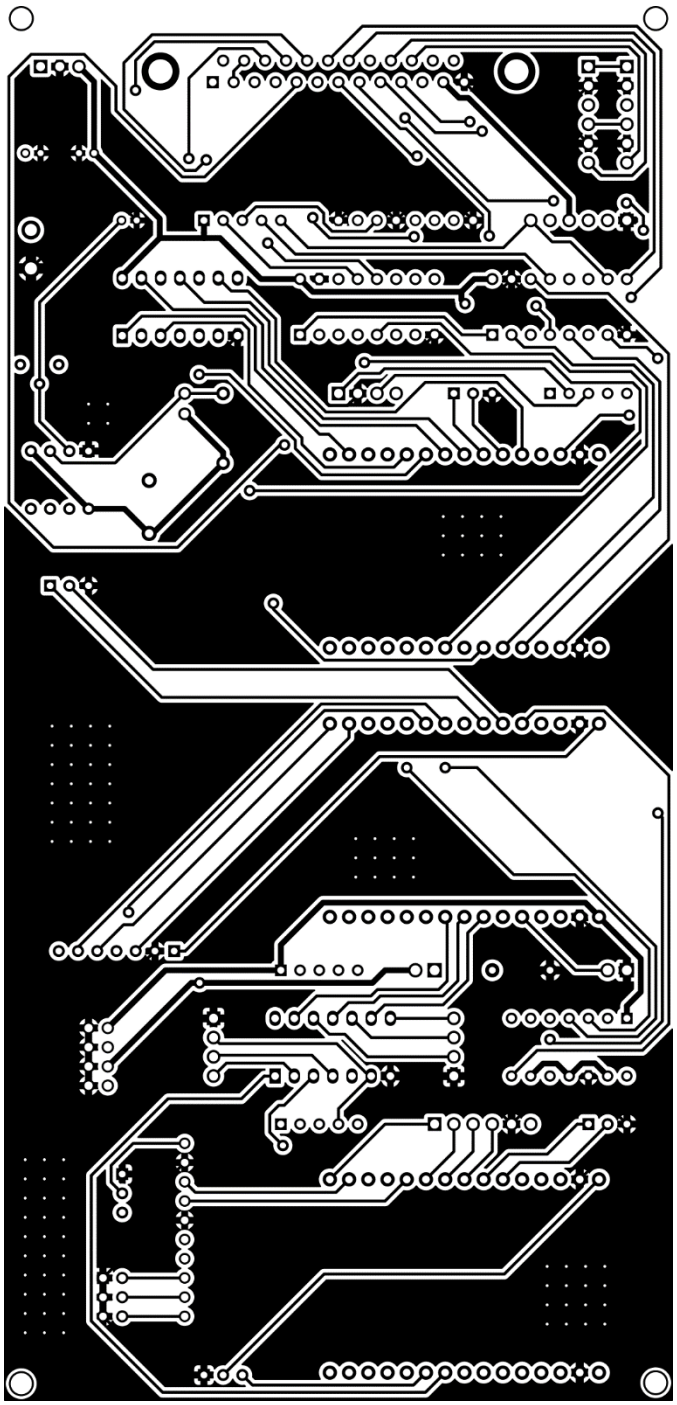


Figure 80 Low-level controller printed circuit board bottom layer

APPENDIX F
SOURCE CODE

Table 14 Electronic system controller source code (Arduino IDE)

Line	Code
1	`const byte REL_PIN[14] = {4, 33, 32, 14, 25, 26, 27, 15, 13, 5, 18, 19, 21, 22};`
2	`const byte SEL_PIN = 23;`
3	
4	`const byte DESCRIPTION_MSG[9] = {71, 82, 83, 87, 67, 84, 82, 2, 44};`
5	`const byte INVALID_MSG[9] = {71, 82, 73, 78, 86, 76, 68, 2, 22};`
6	
7	`byte SYSTEM_STATUS[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};`
8	
9	`void setup() {`
10	`pinMode(SEL_PIN, OUTPUT);`
11	`for (int i = 0; i < 14; i++)pinMode(REL_PIN[i], OUTPUT);`
12	
13	`digitalWrite(SEL_PIN, HIGH);`
14	`for (int i = 0; i < 14; i++)digitalWrite(REL_PIN[i], HIGH);`
15	
16	`Serial.begin(57600);`
17	`Serial.setTimeout(5);`
18	`}`
19	
20	`void loop() {`
21	`byte input_buffer[5];`
22	`byte output_buffer[6];`
23	`byte status_buffer[14];`
24	`short input_checksum;`
25	`short output_checksum;`
26	`short status_checksum;`
27	`short receive_checksum;`
28	
29	`Serial.readStringUntil('G');`
30	`if (Serial.available()) {`
31	`Serial.readBytes(input_buffer, 8);`
32	`input_checksum = 71 + input_buffer[0] + input_buffer[1] + input_buffer[2] + input_buffer[3] + input_buffer[4] + input_buffer[5];`
33	`receive_checksum = (input_buffer[6] << 8) | input_buffer[7];`
104	`else if (input_buffer[2] == 2) {`
105	`digitalWrite(REL_PIN[3], HIGH);`
106	`digitalWrite(REL_PIN[4], HIGH);`
107	`digitalWrite(REL_PIN[5], LOW);`
108	`}`
109	`else if (input_buffer[2] == 3) {`
110	`digitalWrite(REL_PIN[3], LOW);`
111	`digitalWrite(REL_PIN[4], LOW);`
112	`digitalWrite(REL_PIN[5], LOW);`
113	`}`
114	`}`
115	`else if (input_buffer[1] == 3) {`
116	`if (input_buffer[2] == 0) {`
117	`digitalWrite(REL_PIN[6], HIGH);`
118	`}`
119	`else if (input_buffer[2] == 1) {`
120	`digitalWrite(REL_PIN[6], LOW);`
121	`}`
122	`}`
123	`else if (input_buffer[1] == 4) {`
124	`if (input_buffer[2] == 0) {`
125	`digitalWrite(REL_PIN[7], HIGH);`
126	`digitalWrite(REL_PIN[8], HIGH);`
127	`}`
128	`else if (input_buffer[2] == 1) {`
129	`digitalWrite(REL_PIN[7], LOW);`
130	`digitalWrite(REL_PIN[8], HIGH);`
131	`}`
132	`else if (input_buffer[2] == 2) {`
133	`digitalWrite(REL_PIN[7], LOW);`
134	`digitalWrite(REL_PIN[8], LOW);`
135	`}`
136	`}`

```
34      if (input_checksum ==
        receive_checksum) {
35      if (input_buffer[0] == 73) {
36      if (input_buffer[1] == 0) {
37      Serial.write(DESCRIPTION_MSG, 9);
38      }
39      else {
40      Serial.write(INVALID_MSG, 9);
41      }
42      }
43      if (input_buffer[0] == 82) {
44      if (input_buffer[1] == 10) {
45      status_buffer[0] = 71;
46      status_buffer[1] = 83;
47      status_checksum = 154;
48      for (int i = 0; i < 10; i++) {
49      status_buffer[i + 2] =
        SYSTEM_STATUS[i];
50      status_checksum +=
        SYSTEM_STATUS[i];
51      }
52      status_buffer[12] =
        highByte(status_checksum);
53      status_buffer[13] =
        lowByte(status_checksum);
54      Serial.write(status_buffer, 14);
55      }
56      else if (input_buffer[1] < 10) {
57      output_buffer[0] = 71;
58      output_buffer[1] = 82;
59      output_buffer[2] = input_buffer[1];
60      output_buffer[3] =
        SYSTEM_STATUS[input_buffer[1]];
61      output_checksum = 153 +
        output_buffer[2] +
        output_buffer[3];
62      output_buffer[4] =
        highByte(output_checksum);
63      output_buffer[5] =
        lowByte(output_checksum);
64      Serial.write(output_buffer, 6);
65      }
66      else {
67      Serial.write(INVALID_MSG, 9);
68      }
69      }
70      else if (input_buffer[0] == 65) {
71      if (input_buffer[1] == 0) {
72      if (input_buffer[2] == 0) {
73      digitalWrite(REL_PIN[0], HIGH);
74      }
137     else if (input_buffer[1] == 5) {
138     if (input_buffer[2] == 0) {
139     digitalWrite(REL_PIN[9], HIGH);
140     }
141     else if (input_buffer[2] == 1) {
142     digitalWrite(REL_PIN[9], LOW);
143     }
144     }
145     else if (input_buffer[1] == 6) {
146     if (input_buffer[2] == 0) {
147     digitalWrite(REL_PIN[10], HIGH);
148     }
149     else if (input_buffer[2] == 1) {
150     digitalWrite(REL_PIN[10], LOW);
151     }
152     }
153     else if (input_buffer[1] == 7) {
154     if (input_buffer[2] == 0) {
155     digitalWrite(REL_PIN[11], HIGH);
156     }
157     else if (input_buffer[2] == 1) {
158     digitalWrite(REL_PIN[11], LOW);
159     }
160     }
161     else if (input_buffer[1] == 8) {
162     if (input_buffer[2] == 0) {
163     digitalWrite(REL_PIN[12], HIGH);
164     digitalWrite(REL_PIN[13], HIGH);
165     }
166     else if (input_buffer[2] == 1) {
167     digitalWrite(REL_PIN[12], LOW);
168     digitalWrite(REL_PIN[13], HIGH);
169     }
170     else if (input_buffer[2] == 2) {
171     digitalWrite(REL_PIN[12], HIGH);
172     digitalWrite(REL_PIN[13], LOW);
173     }
174     else if (input_buffer[2] == 3) {
175     digitalWrite(REL_PIN[12], LOW);
176     digitalWrite(REL_PIN[13], LOW);
177     }
```

```
75      else if (input_buffer[2] == 1) {
76      digitalWrite(REL_PIN[0], LOW);
77      }
78      }
79      else if (input_buffer[1] == 1) {
80      if (input_buffer[2] == 0) {
81      digitalWrite(REL_PIN[1], HIGH);
82      digitalWrite(REL_PIN[2], HIGH);
83      }
84      else if (input_buffer[2] == 1) {
85      digitalWrite(REL_PIN[1], LOW);
86      digitalWrite(REL_PIN[2], HIGH);
87      }
88      else if (input_buffer[2] == 2) {
89      digitalWrite(REL_PIN[1], HIGH);
90      digitalWrite(REL_PIN[2], LOW);
91      }
92      }
93      else if (input_buffer[1] == 2) {
94      if (input_buffer[2] == 0) {
95      digitalWrite(REL_PIN[3], HIGH);
96      digitalWrite(REL_PIN[4], HIGH);
97      digitalWrite(REL_PIN[5], HIGH);
98      }
99      else if (input_buffer[2] == 1) {
100     digitalWrite(REL_PIN[3], HIGH);
101     digitalWrite(REL_PIN[4], LOW);
102     digitalWrite(REL_PIN[5], HIGH);
103     }
178     }
179     else if (input_buffer[1] == 9) {
180     for (int i = 0; i < 14;
        i++)digitalWrite(REL_PIN[i], HIGH);
181     for (int i = 0; i < 10;
        i++)SYSTEM_STATUS[i] = 0;
182     if (input_buffer[2] == 0) {
183     digitalWrite(SEL_PIN, HIGH);
184     }
185     else if (input_buffer[2] == 1) {
186     digitalWrite(SEL_PIN, LOW);
187     }
188     }
189     if (input_buffer[1] < 10) {
190     SYSTEM_STATUS[input_buffer[1]] =
        input_buffer[2];
191     output_buffer[0] = 71;
192     for (int i = 0; i < 5; i++)
        output_buffer[i + 1] =
        input_buffer[i];
193     Serial.write(output_buffer, 6);
194     }
195     else {
196     Serial.write(INVALID_MSG, 9);
197     }
198     }
199     else {
200     Serial.write(INVALID_MSG, 9);
201     }
202     }
203     else {
204     Serial.write(INVALID_MSG, 9);
205     }
206     }
207     }
```

Table 15 Steering control system controller source code (Arduino IDE)

Line	Code	Line	Code
1	`#include "EEPROM.h"`	134	`#include "EEPROM.h"`
2	`#include <Wire.h>`	135	`#include <Wire.h>`
3	`#include <Adafruit_ADS1015.h>`	136	`#include <Adafruit_ADS1015.h>`
4		137	
5	`Adafruit_ADS1115 ENCODER(0x48);`	138	`Adafruit_ADS1115 ENCODER(0x48);`
6		139	
7	`const byte MTR_PWM = 26;`	140	`const byte MTR_PWM = 26;`
8	`const byte MTR_IN2 = 25;`	141	`const byte MTR_IN2 = 25;`

```
9   const byte MTR_IN1 = 33;
10  const byte MTR_PWM_CHN = 0;
11
12  const byte DESCRIPTION_MSG[9] =
    {71, 82, 83, 67, 67, 84, 82, 2,
    24};
13  const byte INVALID_MSG[9] = {71,
    82, 73, 78, 86, 76, 68, 2, 22};
14
15  //{KP, KI, KD, INITIAL_STEERING,
    MINIMUM_STEERING_OUTPUT,
    MAXIMUM_STEERING_OUTPUT,
    CONTROLLED_LOOP_INTERVAL}
16  const byte EEPROM_ADDR[7] = {0, 4,
    8, 12, 16, 20, 24};
17
18  //{REFERENCE_STEERING, MODE, KP,
    KI, KD, INITIAL_STEERING,
    MINIMUM_STEERING_OUTPUT,
    MAXIMUM_STEERING_OUTPUT,
    CONTROLLED_LOOP_INTERVAL,
    DIAGNOSTIC_STREAM_MODE,
    SENSE_STEERING, DEVIATION,
    FORMER_DEVIATION,
    COMMULATIVE_DEVIATION,
    DIFFERENT_DEVIATION, OUTPUT}
19  long PARAMETERS[16] = {0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0};
20  byte DIAG_OUTPUT_BUFFER[68];
21
22  //{DECREASE_STEERING,
    INCREASE_STEERING}
23  byte DIRECTION_CONTROL_DECREASE =
    1;
24  byte DIRECTION_CONTROL_INCREASE =
    1;
25
26  byte SENSOR_STREAM_STATUS = 0;
27  byte DEBUGGER_STATUS = 0;
28
29  unsigned long LOOP_TIMESTAMP;
30
31  void setup() {
32  ledcSetup(MTR_PWM_CHN, 5000, 12);
33  ledcAttachPin(MTR_PWM,
    MTR_PWM_CHN);
34  pinMode(MTR_IN2, OUTPUT);
35  pinMode(MTR_IN1, OUTPUT);
36  FreeSteering();
37
38  ENCODER.begin();
39
40  EEPROM.begin(32);
41  for (int i = 0; i < 7;
    i++)PARAMETERS[i + 2] =
    EEPROM.readLong(EEPROM_ADDR[i]);
```

```
142 const byte MTR_IN1 = 33;
143 const byte MTR_PWM_CHN = 0;
144
145 const byte DESCRIPTION_MSG[9] =
    {71, 82, 83, 67, 67, 84, 82, 2,
    24};
146 const byte INVALID_MSG[9] = {71,
    82, 73, 78, 86, 76, 68, 2, 22};
147
148 //{KP, KI, KD, INITIAL_STEERING,
    MINIMUM_STEERING_OUTPUT,
    MAXIMUM_STEERING_OUTPUT,
    CONTROLLED_LOOP_INTERVAL}
149 const byte EEPROM_ADDR[7] = {0, 4,
    8, 12, 16, 20, 24};
150
151 //{REFERENCE_STEERING, MODE, KP,
    KI, KD, INITIAL_STEERING,
    MINIMUM_STEERING_OUTPUT,
    MAXIMUM_STEERING_OUTPUT,
    CONTROLLED_LOOP_INTERVAL,
    DIAGNOSTIC_STREAM_MODE,
    SENSE_STEERING, DEVIATION,
    FORMER_DEVIATION,
    COMMULATIVE_DEVIATION,
    DIFFERENT_DEVIATION, OUTPUT}
152 long PARAMETERS[16] = {0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0};
153 byte DIAG_OUTPUT_BUFFER[68];
154
155 //{DECREASE_STEERING,
    INCREASE_STEERING}
156 byte DIRECTION_CONTROL_DECREASE =
    1;
157 byte DIRECTION_CONTROL_INCREASE =
    1;
158
159 byte SENSOR_STREAM_STATUS = 0;
160 byte DEBUGGER_STATUS = 0;
161
162 unsigned long LOOP_TIMESTAMP;
163
164 void setup() {
165 ledcSetup(MTR_PWM_CHN, 5000, 12);
166 ledcAttachPin(MTR_PWM,
    MTR_PWM_CHN);
167 pinMode(MTR_IN2, OUTPUT);
168 pinMode(MTR_IN1, OUTPUT);
169 FreeSteering();
170
171 ENCODER.begin();
172
173 EEPROM.begin(32);
174 for (int i = 0; i < 7;
    i++)PARAMETERS[i + 2] =
    EEPROM.readLong(EEPROM_ADDR[i]);
```

```
42
43      Serial.begin(57600);
44      Serial.setTimeout(5);
45
46      PARAMETERS[0] = PARAMETERS[5];
47      PARAMETERS[10] =
        ENCODER.readADC_SingleEnded(0);
48      PARAMETERS[12] = PARAMETERS[0] -
        PARAMETERS[10];
49
50      LOOP_TIMESTAMP = millis();
51      }
52
53      void loop() {
54      byte input_buffer[8];
55      byte output_buffer[9];
56      short input_checksum;
57      short output_checksum;
58      short receive_checksum;
59      long receive_buffer;
60
61      PARAMETERS[10] =
        ENCODER.readADC_SingleEnded(0);
        //SENSED_STEERING
62
63      Serial.readStringUntil('G');
64      if (Serial.available()) {
65      Serial.readBytes(input_buffer, 8);
66      if (input_buffer[0] == 68 &&
        input_buffer[1] == 69 &&
        input_buffer[2] == 66 &&
        input_buffer[3] == 85 &&
        input_buffer[4] == 71 &&
        input_buffer[5] == 71 &&
        input_buffer[6] == 69 &&
        input_buffer[7] == 82) {
67      DEBUGGER_STATUS = !DEBUGGER_STATUS;
68      }
69      if (input_buffer[0] == 83 &&
        input_buffer[1] == 69 &&
        input_buffer[2] == 78 &&
        input_buffer[3] == 68 &&
        input_buffer[4] == 83 &&
        input_buffer[5] == 84 &&
        input_buffer[6] == 82 &&
        input_buffer[7] == 77) {
70      SENSOR_STREAM_STATUS =
        !SENSOR_STREAM_STATUS;
71      }
72      else {
73      input_checksum = 71 +
        input_buffer[0] + input_buffer[1] +
        input_buffer[2] + input_buffer[3] +
        input_buffer[4] + input_buffer[5];
74      receive_checksum = (input_buffer[6]
        << 8) | input_buffer[7];
```

```
175
176     Serial.begin(57600);
177     Serial.setTimeout(5);
178
179     PARAMETERS[0] = PARAMETERS[5];
180     PARAMETERS[10] =
        ENCODER.readADC_SingleEnded(0);
181     PARAMETERS[12] = PARAMETERS[0] -
        PARAMETERS[10];
182
183     LOOP_TIMESTAMP = millis();
184     }
185
186     void loop() {
187     byte input_buffer[8];
188     byte output_buffer[9];
189     short input_checksum;
190     short output_checksum;
191     short receive_checksum;
192     long receive_buffer;
193
194     PARAMETERS[10] =
        ENCODER.readADC_SingleEnded(0);
        //SENSED_STEERING
195
196     Serial.readStringUntil('G');
197     if (Serial.available()) {
198     Serial.readBytes(input_buffer, 8);
199     if (input_buffer[0] == 68 &&
        input_buffer[1] == 69 &&
        input_buffer[2] == 66 &&
        input_buffer[3] == 85 &&
        input_buffer[4] == 71 &&
        input_buffer[5] == 71 &&
        input_buffer[6] == 69 &&
        input_buffer[7] == 82) {
200     DEBUGGER_STATUS = !DEBUGGER_STATUS;
201     }
202     if (input_buffer[0] == 83 &&
        input_buffer[1] == 69 &&
        input_buffer[2] == 78 &&
        input_buffer[3] == 68 &&
        input_buffer[4] == 83 &&
        input_buffer[5] == 84 &&
        input_buffer[6] == 82 &&
        input_buffer[7] == 77) {
203     SENSOR_STREAM_STATUS =
        !SENSOR_STREAM_STATUS;
204     }
205     else {
206     input_checksum = 71 +
        input_buffer[0] + input_buffer[1] +
        input_buffer[2] + input_buffer[3] +
        input_buffer[4] + input_buffer[5];
207     receive_checksum = (input_buffer[6]
        << 8) | input_buffer[7];
```

```
75      if (input_checksum ==
        receive_checksum) {
76      if (input_buffer[0] == 73) {
77      if (input_buffer[1] == 0) {
78      if (SENSOR_STREAM_STATUS == 0) {
79      Serial.write(DESCRIPTION_MSG, 9);
80      }
81      }
82      else {
83      if (SENSOR_STREAM_STATUS == 0) {
84      Serial.write(INVALID_MSG, 9);
85      }
86      }
87      }
88      else if (input_buffer[0] == 82) {
89      if (input_buffer[1] == 16) {
90      UpdateDiagOutputBuffer();
91      if (SENSOR_STREAM_STATUS == 0) {
92      Serial.write(DIAG_OUTPUT_BUFFER,
        68);
93      }
94      }
95      else if (input_buffer[1] < 16) {
96      output_buffer[0] = 71;
97      output_buffer[1] = 82;
98      output_buffer[2] = input_buffer[1];
99      output_buffer[3] =
        (PARAMETERS[input_buffer[1]] >> 24)
        & 255;
100     output_buffer[4] =
        (PARAMETERS[input_buffer[1]] >> 16)
        & 255;
101     output_buffer[5] =
        (PARAMETERS[input_buffer[1]] >> 8)
        & 255;
102     output_buffer[6] =
        PARAMETERS[input_buffer[1]]  & 255;
103     output_checksum = 153 +
        output_buffer[2] + output_buffer[3]
        + output_buffer[4] +
        output_buffer[5] +
        output_buffer[6];
104     output_buffer[7] =
        highByte(output_checksum);
105     output_buffer[8] =
        lowByte(output_checksum);
106     if (SENSOR_STREAM_STATUS == 0) {
107     Serial.write(output_buffer, 9);
108     }
109     }
110     else {
111     if (SENSOR_STREAM_STATUS == 0) {
112     Serial.write(INVALID_MSG, 9);
```

```
208     if (input_checksum ==
        receive_checksum) {
209     if (input_buffer[0] == 73) {
210     if (input_buffer[1] == 0) {
211     if (SENSOR_STREAM_STATUS == 0) {
212     Serial.write(DESCRIPTION_MSG, 9);
213     }
214     }
215     else {
216     if (SENSOR_STREAM_STATUS == 0) {
217     Serial.write(INVALID_MSG, 9);
218     }
219     }
220     }
221     else if (input_buffer[0] == 82) {
222     if (input_buffer[1] == 16) {
223     UpdateDiagOutputBuffer();
224     if (SENSOR_STREAM_STATUS == 0) {
225     Serial.write(DIAG_OUTPUT_BUFFER,
        68);
226     }
227     }
228     else if (input_buffer[1] < 16) {
229     output_buffer[0] = 71;
230     output_buffer[1] = 82;
231     output_buffer[2] = input_buffer[1];
232     output_buffer[3] =
        (PARAMETERS[input_buffer[1]] >> 24)
        & 255;
233     output_buffer[4] =
        (PARAMETERS[input_buffer[1]] >> 16)
        & 255;
234     output_buffer[5] =
        (PARAMETERS[input_buffer[1]] >> 8)
        & 255;
235     output_buffer[6] =
        PARAMETERS[input_buffer[1]]  & 255;
236     output_checksum = 153 +
        output_buffer[2] + output_buffer[3]
        + output_buffer[4] +
        output_buffer[5] +
        output_buffer[6];
237     output_buffer[7] =
        highByte(output_checksum);
238     output_buffer[8] =
        lowByte(output_checksum);
239     if (SENSOR_STREAM_STATUS == 0) {
240     Serial.write(output_buffer, 9);
241     }
242     }
243     else {
244     if (SENSOR_STREAM_STATUS == 0) {
245     Serial.write(INVALID_MSG, 9);
```

```
113     }
114     }
115     }
116     else if (input_buffer[0] == 65) {
117     PARAMETERS[input_buffer[1]] =
        (input_buffer[2] << 24) |
        (input_buffer[3] << 16) |
        (input_buffer[4] << 8) |
        input_buffer[5];
118     if (input_buffer[1] == 1) {
119     PARAMETERS[0] = PARAMETERS[5];
120     PARAMETERS[12] = PARAMETERS[0] -
        PARAMETERS[10];
121     PARAMETERS[13] = 0;
122     }
123     else if (input_buffer[1] > 1 &&
        input_buffer[1] < 9) {
124     EEPROM.writeLong(EEPROM_ADDR[input_
        buffer[1] - 2],
        PARAMETERS[input_buffer[1]]);
125     EEPROM.commit();
126     }
127     if (input_buffer[1] < 10) {
128     output_buffer[0] = 71;
129     for (int i = 0; i < 8;
        i++)output_buffer[i + 1] =
        input_buffer[i];
130     if (SENSOR_STREAM_STATUS == 0) {
131     Serial.write(output_buffer, 9);
132     }
133     }
```

```
246     }
247     }
248     }
249     else if (input_buffer[0] == 65) {
250     PARAMETERS[input_buffer[1]] =
        (input_buffer[2] << 24) |
        (input_buffer[3] << 16) |
        (input_buffer[4] << 8) |
        input_buffer[5];
251     if (input_buffer[1] == 1) {
252     PARAMETERS[0] = PARAMETERS[5];
253     PARAMETERS[12] = PARAMETERS[0] -
        PARAMETERS[10];
254     PARAMETERS[13] = 0;
255     }
256     else if (input_buffer[1] > 1 &&
        input_buffer[1] < 9) {
257     EEPROM.writeLong(EEPROM_ADDR[input_
        buffer[1] - 2],
        PARAMETERS[input_buffer[1]]);
258     EEPROM.commit();
259     }
260     if (input_buffer[1] < 10) {
261     output_buffer[0] = 71;
262     for (int i = 0; i < 8;
        i++)output_buffer[i + 1] =
        input_buffer[i];
263     if (SENSOR_STREAM_STATUS == 0) {
264     Serial.write(output_buffer, 9);
265     }
266     }
```

Table 16 Speed control system controller source code (Arduino IDE)

Line	Code	Line	Code
1	`#include "EEPROM.h"`	176	`}`
2	`#include <SPI.h>`	177	`}`
3	`#include "mcp_can.h"`	178	`else {`
4		179	`if (SENSOR_STREAM_STATUS == 0) {`
5	`const byte MTR_LMS = 14;`	180	`Serial.write(INVALID_MSG, 9);`
6	`const byte MTR_PWM = 33;`	181	`}`
7	`const byte MTR_IN2 = 25;`	182	`}`
8	`const byte MTR_IN1 = 26;`	183	`}`
9	`const byte DAC_CS = 21;`	184	`else {`
10	`const byte DAC_LDAC = 4;`	185	`if (SENSOR_STREAM_STATUS == 0) {`
11	`const byte CAN_CS = 22;`	186	`Serial.write(INVALID_MSG, 9);`
12		187	`}`
13	`MCP_CAN CAN(CAN_CS);`	188	`}`
14		189	`}`
15	`unsigned short CAN_message_ID;`	190	`}`

```
byte CAN_message_length = 0;
byte CAN_input_buffer[8];

const byte DESCRIPTION_MSG[9] = {71, 82, 67, 67, 67, 84, 82, 2, 8};
const byte INVALID_MSG[9] = {71, 82, 73, 78, 86, 76, 68, 2, 22};

//{KP, KI, KD, IDLE_DECELERATION_OFFSET, ACCELERATOR_SIGNAL_GAIN, BRAKE_ACTUATOR_GAIN, INITIAL_SPEED, CONTROLLED_LOOP_INTERVAL}
const byte EEPROM_ADDR[8] = {0, 4, 8, 12, 16, 20, 24, 28};

//{REFERENCE_SPEED, MODE, KP, KI, KD, IDLE_DECELERATION_OFFSET, ACCELERATOR_PEDAL_SIGNAL_GAIN, BRAKE_ACTUATOR_GAIN, INITIAL_SPEED, CONTROLLED_LOOP_INTERVAL, DIAGNOSTIC_STREAM_MODE, SENSED_SPEED, DEVIATION, FORMER_DEVIATION, COMMULATIVE_DEVIATION, DIFFERENT_DEVIATION, OUTPUT}
long PARAMETERS[17] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
byte DIAG_OUTPUT_BUFFER[72];

//{t, t-1, t-2, ...}
short SENSED_SPEED_WINDOW[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

byte DEBUGGER_STATUS = 0;
byte SENSOR_STREAM_STATUS = 0;

byte FIRST_LOOP_FLAG = 1;

unsigned long LOOP_TIMESTAMP;

void setup() {
pinMode(MTR_LMS, INPUT_PULLUP);
ledcSetup(0, 5000, 12);
ledcAttachPin(MTR_PWM, 0);
pinMode(MTR_IN2, OUTPUT);
pinMode(MTR_IN1, OUTPUT);

if (PARAMETERS[1] == 2) {
if (PARAMETERS[11] > 50) {
AcceleratorPedalSignalWrite(0);
ForwardBrake(2048000);
}
else {
LockBrake();
}
}
else if (PARAMETERS[1] == 1) {
PARAMETERS[12] = PARAMETERS[0] - PARAMETERS[11];    //DEVIATION
PARAMETERS[14] = PARAMETERS[14] + PARAMETERS[12];    //COMMULATIVE_DEVIATION
PARAMETERS[15] = PARAMETERS[12] - PARAMETERS[13];    //DIFFERENT_DEVIATION
PARAMETERS[13] = PARAMETERS[12];    //UPDATE
PARAMETERS[16] = long((PARAMETERS[2] * PARAMETERS[12]) + (PARAMETERS[3] * PARAMETERS[14]) / 100.0 + (PARAMETERS[4] * PARAMETERS[15]));
if (PARAMETERS[16] > 0) {
PARAMETERS[16] = long(PARAMETERS[16] * PARAMETERS[6] / 100.0);
}
else if (PARAMETERS[16] < 0) {
PARAMETERS[16] = long(PARAMETERS[16] * PARAMETERS[7] / 100.0);
}
if (PARAMETERS[16] > 4095000) {
PARAMETERS[16] = 4095000;
PARAMETERS[14] = PARAMETERS[14] - PARAMETERS[12];
}
else if (PARAMETERS[16] < -4095000) {
PARAMETERS[16] = -4095000;
PARAMETERS[14] = PARAMETERS[14] - PARAMETERS[12];
```

```
45    pinMode(DAC_CS, OUTPUT);
46    pinMode(DAC_LDAC, OUTPUT);
47    digitalWrite(DAC_CS, HIGH);
48    digitalWrite(DAC_LDAC, HIGH);
49
50    Serial.begin(57600);
51    Serial.setTimeout(5);
52    SPI.begin();
53
54    ReleaseBrake();
55    LockBrake();
56    AcceleratorPedalSignalWrite(0);
57
58    EEPROM.begin(64);
59    for (int i = 0; i < 8; i++)
      PARAMETERS[i + 2] =
      EEPROM.readLong(EEPROM_ADDR[i]);
60
61    while (CAN_OK !=
      CAN.begin(CAN_1000KBPS)) delay(10);
62
63    LOOP_TIMESTAMP = millis();
64    }
65
66    void loop() {
67    byte input_buffer[8];
68    byte output_buffer[9];
69    short input_checksum;
70    short output_checksum;
71    short receive_checksum;
72    long receive_buffer;
73    float sensed_speed_window_sum;
74    short sensed_speed;
75    byte CAN_speed_detect = 0;
76
77    while (CAN_speed_detect == 0) {
78    if (CAN_MSGAVAIL ==
      CAN.checkReceive()) {
79    CAN.readMsgBuf(&CAN_message_length,
      CAN_input_buffer);
80    CAN_message_ID = CAN.getCanId();
81    if (CAN_message_ID == 2) {
82    sensed_speed = (CAN_input_buffer[2]
      << 8) | CAN_input_buffer[3];
83    sensed_speed_window_sum = 0.0;
84    for (int i = 9; i > 0; i--) {
85    SENSED_SPEED_WINDOW[i] =
220   }
221   if (PARAMETERS[16] > 0) {
222   AcceleratorPedalSignalWrite(PARAMET
      ERS[16]);
223   if (digitalRead(MTR_LMS)) {
224   ReverseBrake(750000);
225   }
226   else {
227   LockBrake();
228   }
229   }
230   else if (PARAMETERS[16] >
      PARAMETERS[5] && PARAMETERS[16] <=
      0) {
231   AcceleratorPedalSignalWrite(0);
232   if (digitalRead(MTR_LMS)) {
233   ReverseBrake(750000);
234   }
235   else {
236   LockBrake();
237   }
238   }
239   else if (PARAMETERS[16] <=
      PARAMETERS[5]) {
240   AcceleratorPedalSignalWrite(0);
241   ForwardBrake(abs(PARAMETERS[16]));
242   }
243   }
244   else if (PARAMETERS[1] == 0) {
245   AcceleratorPedalSignalWrite(0);
246   if (digitalRead(MTR_LMS)) {
247   ReverseBrake(750000);
248   }
249   else {
250   LockBrake();
251   }
252   }
253   if (PARAMETERS[10] == 1) {
254   UpdateDiagOutputBuffer();
255   Serial.write(DIAG_OUTPUT_BUFFER,
      72);
256   }
257   if (DEBUGGER_STATUS == 1) {
258   Serial.print("DB");
259   Serial.print('\t');
260   for (int i = 0; i < 17; i++) {
```

```
        SENSED_SPEED_WINDOW[i - 1];
86      sensed_speed_window_sum +=
        SENSED_SPEED_WINDOW[i];
87      }
88      SENSED_SPEED_WINDOW[0] =
        sensed_speed;
89      sensed_speed_window_sum +=
        SENSED_SPEED_WINDOW[0];
90      PARAMETERS[11] =
        long(sensed_speed_window_sum /
        25.6);
91      CAN_speed_detect = 1;
92      }
93      }
94      }
95
96      if (FIRST_LOOP_FLAG == 1) {
97      PARAMETERS[13] = PARAMETERS[0] -
        PARAMETERS[11];
98      FIRST_LOOP_FLAG = 0;
99      }
100
101     Serial.readStringUntil('G');
102     if (Serial.available()) {
103     Serial.readBytes(input_buffer, 8);
104     if (input_buffer[0] == 68 &&
        input_buffer[1] == 69 &&
        input_buffer[2] == 66 &&
        input_buffer[3] == 85 &&
        input_buffer[4] == 71 &&
        input_buffer[5] == 71 &&
        input_buffer[6] == 69 &&
        input_buffer[7] == 82) {
105     DEBUGGER_STATUS = !DEBUGGER_STATUS;
106     }
107     if (input_buffer[0] == 83 &&
        input_buffer[1] == 69 &&
        input_buffer[2] == 78 &&
        input_buffer[3] == 68 &&
        input_buffer[4] == 83 &&
        input_buffer[5] == 84 &&
        input_buffer[6] == 82 &&
        input_buffer[7] == 77) {
108     SENSOR_STREAM_STATUS =
        !SENSOR_STREAM_STATUS;
109     }
110     else {
111     input_checksum = 71 +
        input_buffer[0] + input_buffer[1] +
        input_buffer[2] + input_buffer[3] +
        input_buffer[4] + input_buffer[5];
112     receive_checksum = (input_buffer[6]
        << 8) | input_buffer[7];
113     if (input_checksum ==
        receive_checksum) {
114     if (input_buffer[0] == 73) {
115     if (input_buffer[1] == 0) {
116     if (SENSOR_STREAM_STATUS == 0) {
261     Serial.print(PARAMETERS[i]);
262     Serial.print('\t');
263     }
264     Serial.println();
265     }
266     if (SENSOR_STREAM_STATUS == 1) {
267     Serial.print("SS");
268     Serial.print('\t');
269     Serial.println(PARAMETERS[11]);
270     }
271     while (millis() - LOOP_TIMESTAMP <
        PARAMETERS[9]) {
272     delayMicroseconds(10);
273     }
274     LOOP_TIMESTAMP = millis();
275     }
276
277     void UpdateDiagOutputBuffer() {
278     short diag_checksum;
279     byte write_buffer;
280     DIAG_OUTPUT_BUFFER[0] = 71;
281     DIAG_OUTPUT_BUFFER[1] = 68;
282     diag_checksum = 139;
283     for (int i = 0; i < 17; i++) {
284     for (int j = 0; j < 4; j++) {
285     write_buffer = (PARAMETERS[i] >> (j
        * 8)) & 255;
286     DIAG_OUTPUT_BUFFER[(4 * i) + j + 2]
        = write_buffer;
287     diag_checksum += write_buffer;
288     }
289     }
290     DIAG_OUTPUT_BUFFER[70] =
        highByte(diag_checksum);
291     DIAG_OUTPUT_BUFFER[71] =
```

```
                                                lowByte(diag_checksum);
117     Serial.write(DESCRIPTION_MSG, 9);       292     }
118     }                                       293
119     }                                       294     void
                                                AcceleratorPedalSignalWrite(long v)
                                                {
120     else {                                  295     long output_buffer;
121     if (SENSOR_STREAM_STATUS == 0) {        296     float val = (v * 0.0004776) +
                                                655.2;
122     Serial.write(INVALID_MSG, 9);           297     unsigned short low_val =
                                                short(val);             //CHANNEL_B
123     }                                       298     unsigned short high_val = short(val
                                                + 655.2);   //CHANNEL_A
124     }                                       299     output_buffer = 0b0111000000000000
                                                | low_val;
125     }                                       300     digitalWrite(DAC_CS, LOW);
126     else if (input_buffer[0] == 82) {       301     SPI.transfer((output_buffer >> 8) &
                                                255);
127     if (input_buffer[1] == 17) {            302     SPI.transfer(output_buffer & 255);
128     UpdateDiagOutputBuffer();               303     digitalWrite(DAC_CS, HIGH);
129     if (SENSOR_STREAM_STATUS == 0) {        304     digitalWrite(DAC_LDAC, LOW);
130     Serial.write(DIAG_OUTPUT_BUFFER,        305     delayMicroseconds(10);
        72);
131     }                                       306     digitalWrite(DAC_LDAC, HIGH);
132     }                                       307
133     else if (input_buffer[1] < 17) {        308     output_buffer = 0b1111000000000000
                                                | high_val;
134     output_buffer[0] = 71;                  309     digitalWrite(DAC_CS, LOW);
135     output_buffer[1] = 82;                  310     SPI.transfer((output_buffer >> 8) &
                                                255);
136     output_buffer[2] = input_buffer[1];     311     SPI.transfer(output_buffer & 255);
137     output_buffer[3] =                      312     digitalWrite(DAC_CS, HIGH);
        (PARAMETERS[input_buffer[1]] >> 24)
        & 255;
138     output_buffer[4] =                      313     digitalWrite(DAC_LDAC, LOW);
        (PARAMETERS[input_buffer[1]] >> 16)
        & 255;
139     output_buffer[5] =                      314     delayMicroseconds(10);
        (PARAMETERS[input_buffer[1]] >> 8)
        & 255;
140     output_buffer[6] =                      315     digitalWrite(DAC_LDAC, HIGH);
        PARAMETERS[input_buffer[1]]  & 255;
141     output_checksum = 153 +                 316     }
        output_buffer[2] + output_buffer[3]
        + output_buffer[4] +
        output_buffer[5] +
        output_buffer[6];
142     output_buffer[7] =                      317
        highByte(output_checksum);
143     output_buffer[8] =                      318     void ReleaseBrake() {
        lowByte(output_checksum);
144     if (SENSOR_STREAM_STATUS == 0) {        319     while (digitalRead(MTR_LMS)) {
145     Serial.write(output_buffer, 9);         320     ReverseBrake(750000);
146     }                                       321     delay(1);
147     }                                       322     }
148     else {                                  323     LockBrake();
149     if (SENSOR_STREAM_STATUS == 0) {        324     }
150     Serial.write(INVALID_MSG, 9);           325
151     }                                       326     void ForwardBrake(long p) {
```

```
152     }                                       327     unsigned short pwm = short(p /
                                                        1000.0);
153     }                                       328     ledcWrite(0, pwm);

154     else if (input_buffer[0] == 65) {       329     digitalWrite(MTR_IN1, LOW);

155     PARAMETERS[input_buffer[1]] =           330     digitalWrite(MTR_IN2, HIGH);
        (input_buffer[2] << 24) |
        (input_buffer[3] << 16) |
        (input_buffer[4] << 8) |
        input_buffer[5];
156     if (input_buffer[1] == 1) {             331     }

157     PARAMETERS[0] = PARAMETERS[8];          332

158     PARAMETERS[13] = PARAMETERS[0] -        333     void ReverseBrake(long p) {
        PARAMETERS[11];
159     PARAMETERS[14] = 0;                     334     unsigned short pwm = short(p /
                                                        1000.0);
160     }                                       335     ledcWrite(0, pwm);

161     else if (input_buffer[1] > 1 &&         336     digitalWrite(MTR_IN1, HIGH);
        input_buffer[1] < 10) {
162     EEPROM.writeLong(EEPROM_ADDR[input_     337     digitalWrite(MTR_IN2, LOW);
        buffer[1] - 2],
        PARAMETERS[input_buffer[1]]);
163     EEPROM.commit();                        338     }

164     }                                       339

165     if (input_buffer[1] < 11) {             340     void FreeBrake() {

166     output_buffer[0] = 71;                  341     ledcWrite(0, 0);

167     for (int i = 0; i < 8;                  342     digitalWrite(MTR_IN1, LOW);
        i++)output_buffer[i + 1] =
        input_buffer[i];
168     if (SENSOR_STREAM_STATUS == 0) {        343     digitalWrite(MTR_IN2, LOW);

169     Serial.write(output_buffer, 9);         344     }

170     }                                       345

171     }                                       346     void LockBrake() {

172     else {                                  347     ledcWrite(0, 4095);

173     if (SENSOR_STREAM_STATUS == 0) {        348     digitalWrite(MTR_IN1, LOW);

174     Serial.write(INVALID_MSG, 9);           349     digitalWrite(MTR_IN2, LOW);

175     }                                       350     }
```


Table 17 High-level autonomous navigation dependency source code (Python)

Line	Code
1	`class SystemController:`
2	`import serial`
3	
4	`assign = ((b'GA\x00\x00\x00\x00\x00\x00\x88', b'GA\x00\x01\x00\x00\x00\x00\x89'), (b'GA\x01\x00\x00\x00\x00\x00\x89', b'GA\x01\x01\x00\x00\x00\x00\x8a', b'GA\x01\x02\x00\x00\x00\x00\x8b'), (b'GA\x02\x00\x00\x00\x00\x00\x8a', b'GA\x02\x01\x00\x00\x00\x00\x8b', b'GA\x02\x02\x00\x00\x00\x00\x8c', b'GA\x02\x03\x00\x00\x00\x00\x8d'), (b'GA\x03\x00\x00\x00\x00\x00\x8b', b'GA\x03\x01\x00\x00\x00\x00\x8c'), (b'GA\x04\x00\x00\x00\x00\x00\x8c', b'GA\x04\x01\x00\x00\x00\x00\x8d', b'GA\x04\x02\x00\x00\x00\x00\x8e'), (b'GA\x05\x00\x00\x00\x00\x00\x8d', b'GA\x05\x01\x00\x00\x00\x00\x8e'), (b'GA\x06\x00\x00\x00\x00\x00\x8e', b'GA\x06\x01\x00\x00\x00\x00\x8f'), (b'GA\x07\x00\x00\x00\x00\x00\x8f', b'GA\x07\x01\x00\x00\x00\x00\x90'), (b'GA\x08\x00\x00\x00\x00\x00\x90', b'GA\x08\x01\x00\x00\x00\x00\x91', b'GA\x08\x02\x00\x00\x00\x00\x92', b'GA\x08\x03\x00\x00\x00\x00\x93'), (b'GA\t\x00\x00\x00\x00\x00\x91', b'GA\t\x01\x00\x00\x00\x00\x92'))`

```
5       request = (b'GR\x00\x00\x00\x00\x00\x00\x99',
        b'GR\x01\x00\x00\x00\x00\x00\x9a', b'GR\x02\x00\x00\x00\x00\x00\x9b',
        b'GR\x03\x00\x00\x00\x00\x00\x9c', b'GR\x04\x00\x00\x00\x00\x00\x9d',
        b'GR\x05\x00\x00\x00\x00\x00\x9e', b'GR\x06\x00\x00\x00\x00\x00\x9f',
        b'GR\x07\x00\x00\x00\x00\x00\xa0', b'GR\x08\x00\x00\x00\x00\x00\xa1',
        b'GR\t\x00\x00\x00\x00\x00\xa2', b'GR\n\x00\x00\x00\x00\x00\xa3',
        b'GI\x00\x00\x00\x00\x00\x00\x90')
6
7       device = None
8
9       def __init__(self, comport, baudrate):
10      self.device = self.serial.Serial(comport, baudrate)
11
12      def set(self, subsystem, state):
13      self.device.write(self.assign[subsystem][state])
14
15      def poll(self, subsystem):
16      self.device.reset_input_buffer()
17      self.device.write(subsystem)
18      if subsystem == 10:
19      recv = self.device.read(9)
20      if sum(recv[:-2]) == (recv[-2] << 8)| recv[-1]:
21      val = recv[2:-2]
22      val = (val[0] << 8) | (val[1] << 8) | (val[2] << 8) | val[3]
23      return val
24      else:
25      return -1
26      else:
27      recv = self.device.read(14)
28      if sum(recv[:-2]) == (recv[-2] << 8) | recv[-1]:
29      return list(recv[2:-2])
30      else:
31      return -1
32
33
34      class CruiseController:
35      import serial
36      import numpy
37
38      request = (b'GR\x00\x00\x00\x00\x00\x00\x99',
        b'GR\x01\x00\x00\x00\x00\x00\x9a', b'GR\x02\x00\x00\x00\x00\x00\x9b',
        b'GR\x03\x00\x00\x00\x00\x00\x9c', b'GR\x04\x00\x00\x00\x00\x00\x9d',
        b'GR\x05\x00\x00\x00\x00\x00\x9e', b'GR\x06\x00\x00\x00\x00\x00\x9f',
        b'GR\x07\x00\x00\x00\x00\x00\xa0', b'GR\x08\x00\x00\x00\x00\x00\xa1',
        b'GR\t\x00\x00\x00\x00\x00\xa2', b'GR\n\x00\x00\x00\x00\x00\xa3',
        b'GR\x0b\x00\x00\x00\x00\x00\xa4', b'GR\x0c\x00\x00\x00\x00\x00\xa5',
        b'GR\r\x00\x00\x00\x00\x00\xa6', b'GR\x0e\x00\x00\x00\x00\x00\xa7',
        b'GR\x0f\x00\x00\x00\x00\x00\xa8', b'GR\x10\x00\x00\x00\x00\x00\xa9',
        b'GR\x11\x00\x00\x00\x00\x00\xaa', b'GI\x00\x00\x00\x00\x00\x00\x90')
39
```

```
40      device = None
41
42      def __init__(self, comport, baudrate):
43      self.device = self.serial.Serial(comport, baudrate)
44
45      def set(self, subsystem, state):
46      state = self.numpy.int32(state)
47      output_buffer = b'GA' + bytes([subsystem, (state >> 24) & 255, (state >> 16) &
        255, (state >> 8) & 255, state & 255])
48      checksum = sum(output_buffer)
49      output_buffer = output_buffer + bytes([(checksum >> 8) & 255, checksum & 255])
50      self.device.write(output_buffer)
51
52      def poll(self, subsystem):
53      self.device.reset_input_buffer()
54      self.device.write(self.request[subsystem])
55      if subsystem == 17:
56      recv = self.device.read(72)
57      if sum(recv[:-2]) == (recv[-2] << 8) | recv[-1]:
58      val = self.numpy.array(recv[2:-2]).reshape((17, 4))
59      val = val[:, 0] | (val[:, 1] << 8) | (val[:, 2] << 16) | (val[:, 3] << 24)
60      val = val - (val >> 15) * (1 << 16)
61      val = val.tolist()
62      return val
63      else:
64      return -1
65      else:
66      recv = self.device.read(9)
67      if sum(recv[:-2]) == (recv[-2] << 8) | recv[-1]:
68      recv = recv[3:-2]
69      val = (recv[0] << 24) | (recv[1] << 16) | (recv[2] << 8) | recv[3]
70      val = val - (val >> 31) * (1 << 32)
71      return val
72      else:
73      return -1
74
75
76      class SteeringController:
77      import serial
78      import numpy
79
80      request = (b'GR\x00\x00\x00\x00\x00\x00\x99',
        b'GR\x01\x00\x00\x00\x00\x00\x9a', b'GR\x02\x00\x00\x00\x00\x00\x9b',
        b'GR\x03\x00\x00\x00\x00\x00\x9c', b'GR\x04\x00\x00\x00\x00\x00\x9d',
        b'GR\x05\x00\x00\x00\x00\x00\x9e', b'GR\x06\x00\x00\x00\x00\x00\x9f',
        b'GR\x07\x00\x00\x00\x00\x00\xa0', b'GR\x08\x00\x00\x00\x00\x00\xa1',
        b'GR\t\x00\x00\x00\x00\x00\xa2', b'GR\n\x00\x00\x00\x00\x00\xa3',
```

```
        b'GR\x0b\x00\x00\x00\x00\x00\xa4', b'GR\x0c\x00\x00\x00\x00\x00\xa5',
        b'GR\r\x00\x00\x00\x00\x00\xa6', b'GR\x0e\x00\x00\x00\x00\x00\xa7',
        b'GR\x0f\x00\x00\x00\x00\x00\xa8', b'GR\x10\x00\x00\x00\x00\x00\xa9',
        b'GI\x00\x00\x00\x00\x00\x00\x90')
81
82      device = None
83
84      def __init__(self, comport, baudrate):
85      self.device = self.serial.Serial(comport, baudrate)
86
87      def set(self, subsystem, state):
88      state = self.numpy.int32(state)
89      output_buffer = b'GA' + bytes([subsystem, (state >> 24) & 255, (state >> 16) &
        255, (state >> 8) & 255, state & 255])
90      checksum = sum(output_buffer)
91      output_buffer = output_buffer + bytes([(checksum >> 8) & 255, checksum & 255])
92      self.device.write(output_buffer)
93
94      def poll(self, subsystem):
95      self.device.reset_input_buffer()
96      self.device.write(self.request[subsystem])
97      if subsystem == 16:
98      recv = self.device.read(68)
99      if sum(recv[:-2]) == (recv[-2] << 8) | recv[-1]:
100     val = self.numpy.array(recv[2:-2]).reshape((16, 4))
101     val = val[:, 0] | (val[:, 1] << 8) | (val[:, 2] << 16) | (val[:, 3] << 24)
102     val = val - (val >> 31) * (1 << 32)
103     val = val.tolist()
104     return val
105     else:
106     return -1
107     else:
108     recv = self.device.read(9)
109     if sum(recv[:-2]) == (recv[-2] << 8) | recv[-1]:
110     recv = recv[3:-2]
111     val = (recv[0] << 24) | (recv[1] << 16) | (recv[2] << 8) | recv[3]
112     val = val - (val >> 31) * (1 << 32)
113     return val
114     else:
115     return -1
116
117
118     class SteeringProfile:
119     import numpy
120     from scipy import integrate
121
```

```
122     def __init__(self, filename, steering_gain, steering_offset):
123     self.filename = filename
124     self.steering_gain = steering_gain
125     self.steering_offset = steering_offset
126     with open(self.filename, 'r') as f:
127     data = [i[:-1] for i in f.readlines()]
128     data = self.numpy.array(data).astype(float)
129     self.steering_position_axis_lower_bound = data[0]
130     self.steering_position_axis_upper_bound = data[1]
131     self.steering_position_axis_resolution = int(data[2])
132     self.steering_position_axis_interval = (self.steering_position_axis_upper_bound
        - self.steering_position_axis_lower_bound) /
        (self.steering_position_axis_resolution - 1)
133     self.steering_position_axis =
        self.numpy.linspace(self.steering_position_axis_lower_bound,
        self.steering_position_axis_upper_bound,
        self.steering_position_axis_resolution)
134     self.time_axis_lower_bound = data[3]
135     self.time_axis_upper_bound = data[4]
136     self.time_axis_resolution = int(data[5])
137     self.time_axis_interval = (self.time_axis_upper_bound -
        self.time_axis_lower_bound) / (self.time_axis_resolution - 1)
138     self.time_axis = self.numpy.linspace(self.time_axis_lower_bound,
        self.time_axis_upper_bound, self.time_axis_resolution)
139
140     self.raw_profile = data[6:].reshape((self.steering_position_axis_resolution,
        self.steering_position_axis_resolution, self.time_axis_resolution))
141     self.raw_time = self.numpy.repeat(self.time_axis.reshape((1, -1)),
        self.steering_position_axis_resolution ** 2,
        axis=0).reshape((self.steering_position_axis_resolution,
        self.steering_position_axis_resolution, self.time_axis_resolution))
142     self.profile = self.integrate.cumtrapz((self.steering_gain * self.raw_profile)
        - self.steering_offset, self.raw_time, axis=2, initial=0)
143     self.time = self.raw_time[:, 0, :]
144
145     def get_profile(self, steering_position, trim=None):
146     if steering_position < self.steering_position_axis[0]:
147     adjusted_steering_profile = self.profile[0, :, :]
148     adjusted_raw_steering_profile = self.raw_profile[0, :, :]
149     adjusted_time = self.time
150     elif steering_position > self.steering_position_axis[-1]:
151     adjusted_steering_profile = self.profile[-1, :, :]
152     adjusted_raw_steering_profile = self.raw_profile[-1, :, :]
153     adjusted_time = self.time
154     else:
155     dst = self.numpy.abs(self.steering_position_axis - steering_position)
156     ind = self.numpy.argsort(dst)
157     adjusted_steering_profile = ((dst[ind[0]] * self.profile[ind[1], :, :]) +
        (dst[ind[1]] * self.profile[ind[0], :, :])) /
        self.steering_position_axis_interval
158     adjusted_raw_steering_profile = ((dst[ind[0]] * self.raw_profile[ind[1], :, :])
        + (dst[ind[1]] * self.raw_profile[ind[0], :, :])) /
        self.steering_position_axis_interval
159     adjusted_time = self.time
```

```
160    if trim is not None:
161    trim_index = self.numpy.abs(self.time_axis - trim).argmin()
162    adjusted_steering_profile = adjusted_steering_profile[:, :trim_index]
163    adjusted_raw_steering_profile = adjusted_raw_steering_profile[:, :trim_index]
164    adjusted_time = self.time[:, :trim_index]
165
166    return adjusted_steering_profile, adjusted_raw_steering_profile, adjusted_time
167
168
169    class WaypointsMap:
170    import numpy
171    from scipy.spatial import cKDTree
172
173    def __init__(self, filename, longitudinal_gain, latitudinal_gain):
174    self.filename = filename
175    self.conversion_gain = self.numpy.array([longitudinal_gain, latitudinal_gain])
176    with open(self.filename, 'r') as f:
177    data = [i[:-1].split('\t') for i in f.readlines()]
178    data = self.numpy.array(data).astype(float)
179    self.geographic_waypoints = data[:, :2]
180    self.heading = self.numpy.deg2rad(180 - ((data[:, 2] + 90) % 360))
181    self.speed = data[:, 3] / 3.6
182
183    self.geographic_origin = (self.geographic_waypoints.ptp(axis=0) / 2) +
       self.geographic_waypoints.min(axis=0)
184    self.meter_waypoints = (self.geographic_waypoints - self.geographic_origin) *
       self.conversion_gain
185
186    self.geographic_manager = self.cKDTree(self.geographic_waypoints)
187    self.meter_manager = self.cKDTree(self.meter_waypoints)
188
189    def to_meter(self, geographic_coordinate):
190    ret = (geographic_coordinate - self.geographic_origin) * self.conversion_gain
191    return ret
192
193    def to_mathematic_angle(self, navigation_angle):
194    ret = self.numpy.deg2rad(180 - ((navigation_angle + 90) % 360))
195    return ret
196
197    def to_geographic(self, meter_coordinate):
198    ret = (meter_coordinate / self.conversion_gain) + self.geographic_origin
199    return ret
200
201    def to_navigation_angle(self, mathematic_angle):
202    ret = 360 - ((self.numpy.rad2deg(mathematic_angle) - 90) % 360)
```

```
203     return ret
204
205     def get_trimmed_geographic_waypoints(self, trim_params):
206     forward_trim = int(trim_params[2] * trim_params[1])
207     reverse_trim = trim_params[1] - forward_trim
208     trim_lower_bound = trim_params[0] - reverse_trim
209     trim_upper_bound = trim_params[0] + forward_trim
210     if trim_lower_bound < 0:
211     ret = self.numpy.row_stack((self.geographic_waypoints[trim_lower_bound:, :],
        self.geographic_waypoints[: trim_upper_bound, :]))
212     elif trim_upper_bound > self.geographic_waypoints.shape[0]:
213     ret = self.numpy.row_stack((self.geographic_waypoints[trim_lower_bound:, :],
        self.geographic_waypoints[: trim_upper_bound -
        self.geographic_waypoints.shape[0], :]))
214     else:
215     ret = self.geographic_waypoints[trim_lower_bound: trim_upper_bound, :]
216     return ret
217
218     def get_trimmed_meter_waypoints(self, trim_params):
219     forward_trim = int(trim_params[2] * trim_params[1])
220     reverse_trim = trim_params[1] - forward_trim
221     trim_lower_bound = trim_params[0] - reverse_trim
222     trim_upper_bound = trim_params[0] + forward_trim
223     if trim_lower_bound < 0:
224     ret = self.numpy.row_stack((self.meter_waypoints[trim_lower_bound:, :],
        self.meter_waypoints[: trim_upper_bound, :]))
225     elif trim_upper_bound > self.meter_waypoints.shape[0]:
226     ret = self.numpy.row_stack((self.meter_waypoints[trim_lower_bound:, :],
        self.meter_waypoints[: trim_upper_bound - self.meter_waypoints.shape[0], :]))
227     else:
228     ret = self.meter_waypoints[trim_lower_bound: trim_upper_bound, :]
229     return ret
230
231     def get_trimmed_heading(self, trim_params):
232     forward_trim = int(trim_params[2] * trim_params[1])
233     reverse_trim = trim_params[1] - forward_trim
234     trim_lower_bound = trim_params[0] - reverse_trim
235     trim_upper_bound = trim_params[0] + forward_trim
236     if trim_lower_bound < 0:
237     ret = self.numpy.concatenate((self.heading[trim_lower_bound:], self.heading[:
        trim_upper_bound]))
238     elif trim_upper_bound > self.meter_waypoints.shape[0]:
239     ret = self.numpy.concatenate((self.heading[trim_lower_bound:], self.heading[:
        trim_upper_bound - self.heading.shape[0]]))
240     else:
241     ret = self.heading[trim_lower_bound: trim_upper_bound]
242     return ret
243
244     def get_trimmed_speed(self, trim_params):
```

```
245     forward_trim = int(trim_params[2] * trim_params[1])
246     reverse_trim = trim_params[1] - forward_trim
247     trim_lower_bound = trim_params[0] - reverse_trim
248     trim_upper_bound = trim_params[0] + forward_trim
249     if trim_lower_bound < 0:
250     ret = self.numpy.concatenate((self.speed[trim_lower_bound:], self.speed[:
        trim_upper_bound]))
251     elif trim_upper_bound > self.meter_waypoints.shape[0]:
252     ret = self.numpy.concatenate((self.speed[trim_lower_bound:], self.speed[:
        trim_upper_bound - self.speed.shape[0]]))
253     else:
254     ret = self.speed[trim_lower_bound: trim_upper_bound]
255     return ret
256
257     def get_trimmed(self, trim_params):
258     forward_trim = int(trim_params[2] * trim_params[1])
259     reverse_trim = trim_params[1] - forward_trim
260     trim_lower_bound = trim_params[0] - reverse_trim
261     trim_upper_bound = trim_params[0] + forward_trim
262     if trim_lower_bound < 0:
263     trimmed_geographic_waypoints =
        self.numpy.row_stack((self.geographic_waypoints[trim_lower_bound:, :],
        self.geographic_waypoints[: trim_upper_bound, :]))
264     trimmed_meter_waypoints =
        self.numpy.row_stack((self.meter_waypoints[trim_lower_bound:, :],
        self.meter_waypoints[: trim_upper_bound, :]))
265     trimmed_heading = self.numpy.concatenate((self.heading[trim_lower_bound:],
        self.heading[: trim_upper_bound]))
266     trimmed_speed = self.numpy.concatenate((self.speed[trim_lower_bound:],
        self.speed[: trim_upper_bound]))
267     elif trim_upper_bound > self.heading.shape[0]:
268     trimmed_geographic_waypoints =
        self.numpy.row_stack((self.geographic_waypoints[trim_lower_bound:, :],
        self.geographic_waypoints[: trim_upper_bound -
        self.geographic_waypoints.shape[0], :]))
269     trimmed_meter_waypoints =
        self.numpy.row_stack((self.meter_waypoints[trim_lower_bound:, :],
        self.meter_waypoints[: trim_upper_bound - self.meter_waypoints.shape[0], :]))
270     trimmed_heading = self.numpy.concatenate((self.heading[trim_lower_bound:],
        self.heading[: trim_upper_bound - self.heading.shape[0]]))
271     trimmed_speed = self.numpy.concatenate((self.speed[trim_lower_bound:],
        self.speed[: trim_upper_bound - self.speed.shape[0]]))
272     else:
273     trimmed_geographic_waypoints = self.geographic_waypoints[trim_lower_bound:
        trim_upper_bound, :]
274     trimmed_meter_waypoints = self.meter_waypoints[trim_lower_bound:
        trim_upper_bound, :]
275     trimmed_heading = self.heading[trim_lower_bound: trim_upper_bound]
276     trimmed_speed = self.speed[trim_lower_bound: trim_upper_bound]
277     return trimmed_geographic_waypoints, trimmed_meter_waypoints, trimmed_heading,
        trimmed_speed
278
279
280     class Visualizer2D:
281     import cv2
282     import numpy
```

```
283
284     def __init__(self, name, frame_width, frame_height, plotspace, division,
        waypoints):
285     self.name = name
286     self.canvas_size = self.numpy.array([frame_width, frame_height])
287     self.canvas_origin = self.canvas_size / 2
288     self.original_canvas_origin = self.canvas_origin
289     self.blank_canvas = self.numpy.zeros((frame_height, frame_width, 3),
        self.numpy.uint8)
290     self.blank_canvas[:, :] = (65, 63, 60)
291     self.division_interval = frame_width / division
292     vertical_division = frame_height // self.division_interval
293     vertical_division = self.numpy.linspace(0, frame_height,
        vertical_division+1).astype(int)
294     horizontal_division = self.numpy.linspace(0, frame_width,
        division+1).astype(int)
295     for i in horizontal_division[1:-1]:
296     self.cv2.line(self.blank_canvas, (i, 0), (i, frame_height), (93, 91, 89), 1)
297     for i in vertical_division:
298     self.cv2.line(self.blank_canvas, (0, i), (frame_width, i), (93, 91, 89), 1)
299     self.scale_text_position = (int(0.9 * frame_width), int(0.98 * frame_height))
300     self.canvas = self.blank_canvas.copy()
301     self.plotspace = plotspace
302     self.waypoints = waypoints
303     self.waypoints_color = self.numpy.repeat(self.numpy.array([[255, 255, 255]]),
        self.waypoints.shape[0], axis=0)
304
305     self.scale = self.plotspace * self.canvas_size / self.waypoints.ptp(axis=0)
306     self.scale = self.scale.min()
307     self.original_scale = self.scale
308
309     self.points = self.numpy.array([[0, 0]])
310     self.points_color = self.numpy.array([[0, 255, 0]])
311
312     self.update()
313
314     self.cv2.namedWindow(self.name)
315     self.cv2.setMouseCallback(self.name, self.mouse_callback)
316     self.mouse_drag_start = None
317     self.start_canvas_origin = self.canvas_origin
318     self.count = True
319
320     def update(self, text_info=None):
321     points = self.numpy.row_stack((self.waypoints, self.points))
322     color = self.numpy.row_stack((self.waypoints_color, self.points_color))
323     pixel = self.canvas_origin + ((points * self.scale) * self.numpy.array([1, -
        1]))
324     self.canvas = self.blank_canvas.copy()
```

```
325     for i, j in zip(pixel.astype(int), color.astype(int).tolist()):
326     self.cv2.circle(self.canvas, tuple(i), 0, j, 0)
327     scale_tex = '%.2f m/div' % (self.division_interval / self.scale)
328     self.cv2.putText(self.canvas, scale_tex, self.scale_text_position,
        self.cv2.FONT_HERSHEY_SIMPLEX, 0.4, (255, 255, 0))
329     if text_info is not None:
330     text_position = self.canvas_origin + ((text_info[1] * self.scale) *
        self.numpy.array([1, -1]))
331     self.cv2.putText(self.canvas, text_info[0], (int(text_position[0]),
        int(text_position[1])), self.cv2.FONT_HERSHEY_SIMPLEX, 0.4, (255, 255, 0))
332     self.cv2.imshow(self.name, self.canvas)
333     return self.cv2.waitKey(5)
334
335     def reset_view(self):
336     self.canvas_origin = self.original_canvas_origin
337     self.scale = self.original_scale
338
339     def mouse_callback(self, event, x, y, flags, params):
340     if flags == 7864320:
341     self.scale *= 1.1
342     self.update()
343     elif flags == -7864320:
344     self.scale /= 1.1
345     self.update()
346     if event == 1:
347     self.mouse_drag_start = self.numpy.array([x, y])
348     self.start_canvas_origin = self.canvas_origin
349     if flags == 1:
350     self.count = not self.count
351     if self.count:
352     current = self.numpy.array([x, y])
353     self.canvas_origin = self.start_canvas_origin + (current -
        self.mouse_drag_start)
354     self.update()
355
356
357     class ScoredKinematicPath2D:
358     import numpy
359     from scipy import integrate
360     from scipy.spatial import cKDTree
361
362     def __init__(self, waypoints, steering_profile, weight, collision_radius,
        predicted_distance, neglect_collision=False):
363     self.waypoints = waypoints
364     self.steering_profile = steering_profile
365     self.weight = weight
366     self.collision_radius = collision_radius
367     self.predicted_distance = predicted_distance
```

```
368     self.neglect_collision = neglect_collision
369
370     def update(self, position, heading, speed, steering_position, obstacle=None,
        trim_params=None):
371     adjusted_steering_profile, adjusted_raw_steering_profile, adjusted_time =
        self.steering_profile.get_profile(steering_position,
        trim=self.predicted_distance/speed)
372     adjusted_steering_profile_resolution, adjusted_time_axis_resolution =
        adjusted_steering_profile.shape
373     course = (speed * adjusted_steering_profile) + heading
374     x = (speed * self.integrate.cumtrapz(self.numpy.cos(course), adjusted_time,
        axis=1, initial=0)) + position[0]
375     y = (speed * self.integrate.cumtrapz(self.numpy.sin(course), adjusted_time,
        axis=1, initial=0)) + position[1]
376     xy = self.numpy.column_stack((x.reshape((-1, )), y.reshape((-1,))))
377
378     if trim_params is not None:
379     _, trimmed_waypoints, trimmed_heading, trimmed_speed =
        self.waypoints.get_trimmed(trim_params)
380     trimmed_waypoints_manager = self.cKDTree(trimmed_waypoints)
381     distance_apart, closest_point_index = trimmed_waypoints_manager.query(xy)
382     required_heading = trimmed_heading[closest_point_index]
383     required_speed = trimmed_speed[closest_point_index[0]]
384     else:
385     trimmed_waypoints = None
386     distance_apart, closest_point_index = self.waypoints.meter_manager.query(xy)
387     required_heading = self.waypoints.heading[closest_point_index]
388     required_speed = self.waypoints.speed[closest_point_index[0]]
389
390     distance_apart = distance_apart.reshape((adjusted_steering_profile_resolution,
        -1))
391     distance_score = distance_apart.sum(axis=1)
392     distance_score = distance_score / distance_score.max()
393
394     required_heading =
        required_heading.reshape((adjusted_steering_profile_resolution, -1))
395     heading_score = required_heading - course
396     heading_score = self.numpy.abs(heading_score).sum(axis=1)
397     heading_score = heading_score / heading_score.max()
398
399     steering_smoothness_score =
        self.numpy.abs(self.steering_profile.steering_position_axis -
        steering_position)
400     steering_smoothness_score = steering_smoothness_score /
        steering_smoothness_score.max()
401
402     emergency_brake_status = False
403
404     if obstacle is not None:
405     obstacle_manger = self.cKDTree(obstacle)
406     obstacle_distance, _ = obstacle_manger.query(xy)
407     obstacle_distance =
        obstacle_distance.reshape((adjusted_steering_profile_resolution, -1))
```

```
408    obstacle_score = obstacle_distance.sum(axis=1)
409    obstacle_score = obstacle_score / obstacle_score.max()
410
411    collision_score = obstacle_distance.min(axis=1)
412    if not self.neglect_collision:
413    collision_score[collision_score <= self.collision_radius] = -self.numpy.inf
414    max_collision_score = collision_score.max()
415    if max_collision_score == -self.numpy.inf:
416    emergency_brake_status = True
417    else:
418    collision_score = collision_score / max_collision_score
419    else:
420    obstacle_score = self.numpy.zeros(adjusted_steering_profile_resolution)
421    collision_score = self.numpy.zeros(adjusted_steering_profile_resolution)
422
423    raw_score = self.numpy.column_stack((distance_score, heading_score,
       steering_smoothness_score, obstacle_score, collision_score))
424    weighted_score = self.weight * raw_score
425    weighted_score = weighted_score.sum(axis=1)
426
427    if emergency_brake_status:
428    xy_color = self.numpy.repeat(self.numpy.array([[0, 0, 255]]), xy.shape[0],
       axis=0)
429    else:
430    rank = self.numpy.zeros((adjusted_steering_profile_resolution, ))
431    rank[weighted_score.argsort()] =
       self.numpy.arange(adjusted_steering_profile_resolution)
432
433    xy_color_blue = self.numpy.zeros(adjusted_steering_profile_resolution,)
434    xy_color_red = rank / adjusted_steering_profile_resolution * 255
435    xy_color_green = 255 - xy_color_red
436    xy_color = self.numpy.column_stack((xy_color_blue, xy_color_green,
       xy_color_red))
437    xy_color = self.numpy.repeat(xy_color, adjusted_time_axis_resolution, axis=0)
438
439    required_steering_index = weighted_score.argmin()
440    required_steering_position =
       self.steering_profile.steering_position_axis[required_steering_index]
441
442    return required_steering_position, required_steering_index, required_speed,
       emergency_brake_status, xy, course, xy_color, trimmed_waypoints,
       adjusted_raw_steering_profile, adjusted_steering_profile_resolution,
       adjusted_time_axis_resolution
443
444
445    class PosLVX:
446    import socket
447    import threading
448
```

```
449    TCPport = None
450    thread_alive = False
451    loop_thread = None
452    received_message = None
453    data = None
454    index = None
455
456    GPHDTheading = None
457    GPGGAlatitude = None
458    GPGGAlongitude = None
459    GPGGAquality = None
460    GPGGAsatellitesinuse = None
461    GPRMClatitude = None
462    GPRMClongitude = None
463    GPRMCspeedoverground = None
464    GPRMCmode = None
465    GPVTGtruetrack = None
466    GPVTGtrackmagnetic = None
467    GPVTGspeed = None
468    GPVTGmode = None
469
470    def __init__(self, ip, port):
471    self.TCPport = self.socket.socket(self.socket.AF_INET, self.socket.SOCK_STREAM)
472    self.TCPport.connect((ip, port))
473    self.loop_thread = self.threading.Thread(target=self.loop)
474
475    def loop(self):
476    while self.thread_alive:
477    self.received_message = self.TCPport.recv(1024)
478    self.data = self.received_message.decode().split('\r\n')[:-1]
479    self.data = [i.split(',') for i in self.data]
480    self.index = [i[0] for i in self.data]
481    try:
482    GNHDT = self.index.index('$GNHDT')
483    GNRMC = self.index.index('$GNRMC')
484    GNGGA = self.index.index('$GNGGA')
485    GNVTG = self.index.index('$GNVTG')
486
487    if self.data[GNHDT][1] == '':
488    self.GPHDTheading = None
489    else:
490    self.GPHDTheading = float(self.data[GNHDT][1])
491    if self.data[GNGGA][2] == '':
492    self.GPGGAlatitude = None
```

```
493     else:
494     self.GPGGAlatitude = float(self.data[GNGGA][2][:2]) +
        float(self.data[GNGGA][2][2:])/60
495     if self.data[GNGGA][4] == '':
496     self.GPGGAlongitude = None
497     else:
498     self.GPGGAlongitude= float(self.data[GNGGA][4][:3]) +
        float(self.data[GNGGA][4][3:])/60
499     if self.data[GNGGA][6] == '':
500     self.GPGGAquality = None
501     else:
502     self.GPGGAquality = float(self.data[GNGGA][6])
503     if self.data[GNGGA][7] == '':
504     self.GPGGAsatellitesinuse = None
505     else:
506     self.GPGGAsatellitesinuse = float(self.data[GNGGA][7])
507     if self.data[GNRMC][3] == '':
508     self.GPRMClatitude = None
509     else:
510     self.GPRMClatitude = float(self.data[GNRMC][3][:2]) +
        float(self.data[GNRMC][3][2:])/60
511     if self.data[GNRMC][5] == '':
512     self.GPRMClongitude = None
513     else:
514     self.GPRMClongitude = float(self.data[GNRMC][5][:3]) +
        float(self.data[GNRMC][5][3:])/60
515     if self.data[GNRMC][7] == '':
516     self.GPRMCspeedoverground = None
517     else:
518     self.GPRMCspeedoverground = float(self.data[GNRMC][7])
519     self.GPRMCmode = self.data[GNRMC][12][:1]
520     if self.data[GNVTG][1] == '':
521     self.GPVTGtruetrack = None
522     else:
523     self.GPVTGtruetrack = float(self.data[GNVTG][1])
524     if self.data[GNVTG][3] == '':
525     self.GPVTGtrackmagnetic = None
526     else:
527     self.GPVTGtrackmagnetic = float(self.data[GNVTG][3])
528     if self.data[GNVTG][7] == '':
529     self.GPVTGspeed = None
530     else:
531     self.GPVTGspeed = float(self.data[GNVTG][7])
532     self.GPVTGmode = self.data[GNVTG][9][:1]
533     except:
534     print('POSLVX ERROR')
535
```

```
536    def start(self):
537    self.thread_alive = True
538    self.loop_thread.start()
539
540    def kill(self):
541    self.thread_alive = False
542    del self.loop_thread
543
544
545    class LMS511:
546    import socket
547    import numpy
548
549    def __init__(self, ip, port, radius=0.0):
550    self.ip = ip
551    self.port = port
552    self.radius = radius
553    self.buffer = 2048
554    self.angle = self.numpy.deg2rad(self.numpy.linspace(-5, 185, 381))
555    self.device = None
556
557    def start(self):
558    self.device = self.socket.socket(self.socket.AF_INET, self.socket.SOCK_STREAM)
559    self.device.connect((self.ip, self.port))
560    self.device.settimeout(0.01)
561
562    def get_scan(self, heading=None, origin=None):
563    self.device.send(b'\x02sRN LMDscandata\x03')
564    raw_data = b''
565    while True:
566    try:
567    raw_data += self.device.recv(self.buffer)
568    if raw_data[-1] == 3:
569    break
570    except self.socket.timeout:
571    pass
572    raw_data = raw_data.decode().split(' ')
573    data_length = int(raw_data[raw_data.index('DIST1') + 5], 16)
574    distance = raw_data[raw_data.index('DIST1') + 6:raw_data.index('DIST1') + 6 +
       data_length]
575    distance = [int(i, 16) for i in distance]
576    distance = self.numpy.array(distance) * 0.002
577    indx = (distance >= self.radius)
578    filtered_distance = distance[indx]
```

```
579    filtered_angle = self.angle[indx]
580
581    x = filtered_distance * self.numpy.cos(filtered_angle)
582    y = filtered_distance * self.numpy.sin(filtered_angle)
583    ret = self.numpy.row_stack((x, y))
584
585    if heading is not None:
586    rotational_angle = heading - (self.numpy.pi / 2)
587    c = self.numpy.cos(rotational_angle)
588    s = self.numpy.sin(rotational_angle)
589    rotational_matrix = self.numpy.array([[c, -s], [s, c]])
590    ret = self.numpy.matmul(rotational_matrix, ret)
591    ret = ret.transpose()
592    if origin is not None:
593    ret = ret + origin
594
595    return ret
596
597
598    class ExponentialGainAdjustment:
599
600    def __init__(self, initial, increment, exponent, minimum, maximum):
601    self.gain = initial
602    self.increment = increment
603    self.exponent = exponent
604    self.previous_direction = 0
605    self.maximum = maximum
606    self.minimum = minimum
607
608    def update(self, direction):
609    if direction * self.previous_direction > 0:
610    self.increment = self.increment * self.exponent
611    self.gain = self.gain + (direction * self.increment)
612    elif direction * self.previous_direction < 0:
613    self.increment = self.increment / self.exponent
614    self.gain = self.gain + (direction * self.increment)
615    elif self.previous_direction == 0:
616    self.gain = self.gain + (direction * self.increment)
617    if self.gain < self.minimum:
618    self.increment = self.increment / self.exponent
619    self.gain = self.minimum
620    elif self.gain > self.maximum:
621    self.increment = self.increment / self.exponent
622    self.gain = self.maximum
```

```
623     self.previous_direction = direction
624     return self.gain
```

Table 18 High-level autonomous navigation software source code (Python)

Line	Code	Line	Code
1	`import smrclib`	127	`os.mkdir(log_directory)`
2	`import serial.tools.list_ports`	128	`except FileExistsError:`
3	`import time`	129	`print('[WARNING] Output directory already exist')`
4	`import numpy`	130	`log_filename = log_directory + datetime.datetime.now().strftime('\\%S%M%H%d%m%y.lg')`
5	`import datetime`	131	
6	`import os`	132	`# NAVIGATION ROUTINE`
7	`import cv2`	133	`# system_controller.set(9, 1)`
8	`from scipy.spatial import cKDTree`	134	`system_controller.set(7, 1)`
9		135	`# system_controller.set(0, 1)`
10		136	`cruise_controller.set(1, 1)`
11	`# GEOGRAPHICAL PARAMETERS`	137	`time.sleep(0.1)`
12	`longitude_gain = 108657.32434 # meter/degree_longitude`	138	`steering_controller.set(1, 1)`
13	`latitude_gain = 111456.76004 # meter/degree_latitude`	139	`time.sleep(0.1)`
14		140	`if command_speed is not None:`
15	`# SENSORS PARAMETERS`	141	`cruise_controller.set(0, int(command_speed * 100.0))`
16	`locator_distance = 0.0 # meter`	142	
17	`lidar_distance = 1.74 # meter`	143	`initial_time = time.time()`
18	`collision_radius = 0.6 # meter`	144	`timestamp = time.time()`
19	`lms511_radius = 0.001 # meter`	145	`while True:`
20	`poslvx_ip = '192.168.1.229'`	146	`longitude = poslvx.GPRMClongitude`
21	`poslvx_port = 5017`	147	`latitude = poslvx.GPRMClatitude`
22	`lms511_ip = '192.168.1.101'`	148	`navigation_heading = poslvx.GPHDTheading`
23	`lms511_port = 2111`	149	`current_steering_position = steering_controller.poll(10)`
24		150	`current_speed = cruise_controller.poll(11)`
25	`# VEHICLE PHYSICAL CALIBRATION PARAMETERS`	151	`if current_steering_position < 1500 or current_steering_position > 16000:`
26	`steering_gain = 0.0000498013323955794 # 1/meter_steering_position`	152	`current_steering_position = previous_steering_position`
27	`steering_offset = 0.431766151719804 # 1/meter`	153	`if current_speed < -6000 or current_speed > 6000:`
28		154	`current_speed = previous_speed`
29	`# DECISION WEIGHT`	155	
30	`distance_score_weight = 1.5 # dimensionless`	156	`if current_speed == 0:`
31	`heading_score_weight = 0.01 # dimensionless`	157	`current_speed = 0.01`
32	`steering_smoothness_weight = 0.0 # dimensionless`	158	

```
obstacle_score_weight = -0.1 # dimensionless
collision_score_weight = -0.25 # dimensionless

# ALGORITHM PARAMETERS
control_loop_interval = 0.20 # second
algorithm_collision_radius = 0.8 # meter
algorithm_predicted_distance = 12.0     # meter
trimmed_waypoints_length = 100 # point
trimmed_forward_ratio = 70 # percent
command_speed = 7 # km/hr , None
waypoints_filename = 'waypoints\\SKP(MAINROUTE).wp' # .wp filepath
steering_profile_filename = 'steering_profile\\(3S0).sp' # .sp filepath

# DISPLAY SETTING
display_width = 960 # pixel
display_height = 720 # pixel
display_plotspace = 0.8 # ratio
display_division = 40 # division
lidarscan_display_color = numpy.array([[255, 0, 255]]) # BGR colorspace
trimmed_waypoints_display_color =
```

```
locator_position = waypoints.to_meter(numpy.array([longitude, latitude]))
car_heading = waypoints.to_mathematic_angle(navigation_heading)
car_position = locator_position + (locator_distance * numpy.array([numpy.cos(car_heading), numpy.sin(car_heading)]))
lidar_position = locator_position + (lidar_distance * numpy.array([numpy.cos(car_heading), numpy.sin(car_heading)]))

lidarscan = lms511.get_scan(car_heading, lidar_position)
lidarscan_color = numpy.repeat(lidarscan_display_color, lidarscan.shape[0], axis=0)

_, car_position_index = waypoints.meter_manager.query(car_position)
if abs(car_position_index - previous_trim_index) > trimmed_waypoints_length and 50 < car_position_index < (waypoints.meter_waypoints.shape[0] - 50):
trim_index = previous_trim_index
else:
trim_index = car_position_index
previous_trim_index = trim_index

trim_params = (trim_index, trimmed_waypoints_length, trimmed_forward_ratio)
prior_command_steering, prior_command_index, required_speed, blocked_status, predicted_path, predicted_heading, predicted_path_color, trimmed_waypoints, raw_steering_profile, adjust_profile_resolution, time_resolution = algorithm.update(car_position, car_heading, current_speed/360.0, current_steering_position, obstacle=lidarscan, trim_params=trim_params)
trimmed_waypoints_color = numpy.repeat(trimmed_waypoints_display_color, trimmed_waypoints.shape[0], axis=0)
selected_path = predicted_path[prior_command_index * time_resolution: (prior_command_index + 1) * time_resolution, :]
selected_path_color =
```

```
        numpy.array([[255, 0, 0]])    #
        BGR colorspace
53      selected_path_display_color =
        numpy.array([[255, 0, 255]])
        # BGR colorspace
54
55      # LOG SETTING
56      log_directory = 'logdata'
57
58      # COMPENSATION ALGORITHM
        INITIALIZATION
59      gain_compensation = 0.7
        # dimensionless
60      initial_gain_compensation = 1.0
        # dimensionless
61      gain_compensation_increment =
        0.005     # dimensionless
62      gain_compensation_exponent = 1.10
        # dimensionless
63      minimum_gain = 0.3
        # dimensionless
64      maximum_gain = 1.0
        # dimensionless
65      compensation_status = True
66
67      # COMPENSATION TUNING PARAMETERS
68      tuning_gain = 1.0
69      tuning_offset = 0.0
70
71      # NAVIGATION MODE SETTING
72      cruise_control_status = False
73
74      # VEHICLE COMMUNICATION
75      available_ports =
        serial.tools.list_ports.comports()
76      ports_serial_number =
        [i.serial_number for i in
        available_ports]
77      ports_name = [i.device for i in
        available_ports]
179
180     if compensation_status:
181     gain_compensator_direction =
        (expected_steering_position -
        current_steering_position) *
        command_steering_direction
182     gain_compensator_direction =
        gain_compensator_direction/abs(gai
        n_compensator_direction) if not
        gain_compensator_direction == 0
        else 0
183     gain_compensation =
        gain_compensator.update(gain_compe
        nsator_direction)
184     command_steering =
        gain_compensation *
        (prior_command_steering -
        current_steering_position) +
        current_steering_position
185     else:
186     command_steering = tuning_gain *
        (prior_command_steering -
        current_steering_position) +
        current_steering_position +
        tuning_offset
187
188     if command_steering > 15500:
189     command_steering = 15500
190     elif command_steering < 2500:
191     command_steering = 2500
192
193     steering_controller.set(0,
        int(command_steering))
194     if cruise_control_status:
195     command_speed = required_speed *
        360.0
196     cruise_controller.set(0,
        int(command_speed))
197
198     print('[INFO]
        %d\t%d\t%d\t%s\t%.3f\t' %
        (command_steering,
        prior_command_steering,
        command_speed, blocked_status,
        gain_compensation), end='')
199     print(expected_steering_position,
        current_steering_position)
200
201     visualizer.points =
        numpy.row_stack((predicted_path,
        lidarscan, trimmed_waypoints,
        selected_path))
202     visualizer.points_color =
        numpy.row_stack((predicted_path_co
        lor, lidarscan_color,
        trimmed_waypoints_color,
        selected_path_color))
203     key_received =
        visualizer.update(text_info=('(%.6
```

```
78      ports_table =
        dict(zip(ports_serial_number,
        ports_name))
79      system_controller =
        smrclib.SystemController(ports_tab
        le['0775'], 57600)
80      cruise_controller =
        smrclib.CruiseController(ports_tab
        le['0768'], 57600)
81      steering_controller =
        smrclib.SteeringController(ports_t
        able['0918'], 57600)
82
83      # SENSOR INITIALIZATION
84      poslvx = smrclib.PosLVX(poslvx_ip,
        poslvx_port)
85      lms511 = smrclib.LMS511(lms511_ip,
        lms511_port, lms511_radius)
86      poslvx.start()
87      lms511.start()
88
89      # NAVIGATION ALGORITHM
        INITIALIZATION
90      trimmed_forward_ratio =
        trimmed_forward_ratio / 100.0
91      steering_center =
        steering_offset/steering_gain
92      score_weight =
        numpy.array([distance_score_weight
        , heading_score_weight,
        steering_smoothness_weight,
        obstacle_score_weight,
        collision_score_weight])
93
94      waypoints =
        smrclib.WaypointsMap(waypoints_fil
        ename, longitude_gain,
        latitude_gain)
95      steering_profile =
        smrclib.SteeringProfile(steering_p
        rofile_filename, steering_gain,
        steering_offset)
96      visualizer =
        smrclib.Visualizer2D('Navigator',
        display_width, display_height,
        display_plotspace,
        display_division,
        waypoints.meter_waypoints)
97      algorithm =
        smrclib.ScoredKinematicPath2D(wayp
        oints, steering_profile,
        score_weight,
        algorithm_collision_radius,
        algorithm_predicted_distance,
        neglect_collision=True)
98      gain_compensator =
        smrclib.ExponentialGainAdjustment(
        initial_gain_compensation,
        gain_compensation_increment,
        gain_compensation_exponent,
        minimum_gain, maximum_gain)
99      expected_steering_time_index =
        f, %.6f)' % (longitude, latitude),
        car_position))
204     if key_received == 114:
205     visualizer.reset_view()
206     elif key_received == 27:
207     break
208
209     previous_steering_position =
        current_steering_position
210     previous_speed = current_speed
211     expected_steering_position =
        raw_steering_profile[prior_command
        _index,
        expected_steering_time_index]
212     command_steering_direction =
        command_steering -
        current_steering_position
213
214     while time.time() - timestamp <
        control_loop_interval:
215     pass
216     timestamp = time.time()
217
218     with open(log_filename, 'a') as f:
219     f.write('%.10f\t' % (timestamp-
        initial_time))
220     f.write('%.10f\t' % longitude)
221     f.write('%.10f\t' % latitude)
222     f.write('%.10f\t' %
        navigation_heading)
223     f.write('%d\t' %
        current_steering_position)
224     f.write('%d\t' % (current_speed /
        100.0))
225     f.write('%d\t' % command_steering)
```

```
    numpy.abs(steering_profile.time_ax
    is -
    control_loop_interval).argmin()
100
101 longitude = poslvx.GPRMClongitude
102 latitude = poslvx.GPRMClatitude
103 navigation_heading =
    poslvx.GPHDTheading
104 locator_position =
    waypoints.to_meter(numpy.array([lo
    ngitude, latitude]))
105 car_heading =
    waypoints.to_mathematic_angle(navi
    gation_heading)
106 car_position = locator_position +
    (locator_distance *
    numpy.array([numpy.cos(car_heading
    ), numpy.sin(car_heading)]))
107 _, previous_trim_index =
    waypoints.meter_manager.query(car_
    position)
108
109 previous_steering_position =
    steering_controller.poll(10)
110 previous_speed =
    cruise_controller.poll(11)
111
112 while not (1500 <
    previous_steering_position <
    16500):
113 previous_steering_position =
    steering_controller.poll(10)
114 time.sleep(0.1)
115 while not (-6000 < previous_speed
    < 6000):
116 previous_speed =
    cruise_controller.poll(11)
117 time.sleep(0.1)
118
119 expected_steering_position =
    previous_steering_position
120 command_steering_direction = 0
121
122 brake_pedal_status = False
123 cruise_controller_emergency_status
    = False
124
125 # LOG INITIALIZATION
126 try:
226     f.write('%d\t' % command_speed)
227     f.write('%.10f\t' %
      longitude_gain)
228     f.write('%.10f\t' % latitude_gain)
229     f.write('%.3f\t' %
      distance_score_weight)
230     f.write('%.3f\t' %
      heading_score_weight)
231     f.write('%.3f\t' %
      obstacle_score_weight)
232     f.write('%.3f\n' %
      collision_score_weight)
233
234     cv2.destroyAllWindows()
235     cruise_controller.set(1, 2)
236     time.sleep(0.1)
237     system_controller.set(1, 0)
238     time.sleep(0.1)
239     steering_controller.set(1, 0)
240     time.sleep(0.1)
241     system_controller.set(2, 0)
242     time.sleep(0.1)
243     system_controller.set(0, 0)
244     time.sleep(0.1)
245     system_controller.set(7, 0)
246     time.sleep(0.1)
247     system_controller.set(9, 0)
248     time.sleep(0.1)
249
250     while cruise_controller.poll(11) >
      0:
251     time.sleep(0.1)
252     cruise_controller.set(1, 0)
```

VITA

NAME	Kanin Kiataramgul
DATE OF BIRTH	24 February 1995
PLACE OF BIRTH	Bangkok
INSTITUTIONS ATTENDED	B.Eng. Mechanical Engineering, Chulalongkorn University
HOME ADDRESS	97/142 Kosumruamjai39, Don Mueang, Bangkok, 10210
PUBLICATION	Kiataramgul, K. and N. Noomwongs. Development of forward collision warning system using laser scanner with camera to detect and estimate object headway distance. in The 8th TSME International Conference on Mechanical Engineering. 2017.