การพัฒนาระบบเปลี่ยนผ่านระหว่างยานยนต์ขับขี่อัตโนมัติและระบบควบคุมระยะไกลผ่านเครือข่าย 5G

นายกฤษฎ์ ต.ศิริวัฒนา

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมเครื่องกล ภาควิชาวิศวกรรมเครื่องกล
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2564

Development of Transition System between Autonomous Driving and 5G Tele-operated Driving

Mr. Krit T.siriwattana

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 2021

หัวข้อวิทยานิพนธ์	การพัฒนาระบบเปลี่ยนผ่านระหว่างยานยนต์ขับขี่อัตโนมัติและระบบควบคุมระยะไกลผ่านเครือข่าย 5G
โดย	นายกฤษฎ์ ต.ศิริวัฒนา
สาขาวิชา	วิศวกรรมเครื่องกล
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ ดร.นักสิทธ์ นุ่มวงษ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สัณหพศ จันทรานุวัฒน์)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(ผู้ช่วยศาสตราจารย์ ดร.นักสิทธ์ นุ่มวงษ์)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.กฤษฎา พนมเชิง)

..... กรรมการภายนอกมหาวิทยาลัย
(ดร.ปาษาณ กุลวานิช)

กฤษฎ์ ต.ศิริวัฒนา : การพัฒนาระบบเปลี่ยนผ่านระหว่างยานยนต์ขับขี่อัตโนมัติและระบบควบคุมระยะไกลผ่านเครือข่าย 5G. (Development of Transition System between Autonomous Driving and 5G Tele-operated Driving) อ.ที่ปรึกษาหลัก : ผศ. ดร.นักสิทธ์ นุ่มวงษ์

ปัจจุบันยานยนต์อัตโนมัติกำลังได้รับความนิยม มีการแข่งขันสูงจากผู้ผลิตทั้งภาครัฐและเอกชนในการพัฒนาขีดความสามารถหรือระดับขั้นของความเป็นอัตโนมัติที่สูงขึ้นอย่างต่อเนื่อง โดยระดับสูงสุด (ระดับ 5) เป็นระดับที่ไม่ต้องการการควบคุมใด ๆ จากผู้ขับขี่เลย ส่วนระดับที่ต่ำลงมายังคงต้องถูกควบคุมโดยผู้ขับขี่บางสถานการณ์ที่ระบบเกิดความผิดปกติ หรือพ้นช่วงสภาพแวดล้อมที่ระบบจำกัด งานวิจัยนี้พัฒนาต้นแบบยานยนต์อัตโนมัติที่สามารถควบคุมจากระยะไกลได้ ส่วนของยานยนต์ต้นแบบ ชุดควบคุมในระดับต่ำและระดับสูง โมดูลสั่งงานจากระยะไกล และสถานีควบคุมถูกออกแบบและพัฒนาภายใต้วัตถุประสงค์ของงานวิจัยนี้ด้วย รวมถึงระบบเปลี่ยนผ่านการควบคุมซึ่งถูกออกแบบเพื่อให้ยานยนต์ต้นแบบนี้สามารถสลับระบบควบคุมได้ทุกช่วง หากเกิดความผิดปกติขึ้นระหว่างการใช้งาน เช่น มีความหน่วงเวลาสูงเกิดขึ้น หรือมีวัตถุกีดขวางเส้นทางบริเวณด้านหน้า ระบบเฝ้าระวังความผิดปกติจะตัดการทำงานเข้าสู่โหมดฉุกเฉินทันทีที่ระบบตรวจพบสถานการณ์ดังกล่าว พร้อมกับแจ้งเตือนสถานการณ์ที่เกิดขึ้นไปยังผู้ควบคุม ณ สถานี ให้ทำการสลับระบบควบคุมไปยังระบบอื่นเพื่อแก้ไขสถานการณ์ ผลการทดสอบการใช้งานระบบอัตโนมัติและระบบควบคุมระยะไกลทั้ง 4 ช่วง รอบจุฬาลงกรณ์มหาวิทยาลัย พบว่าระบบสามารถใช้งานได้ดี ส่วนเบี่ยงเบนเชิงเส้นเฉลี่ยระหว่างเส้นทางอ้างอิงและเส้นทางที่ถูกบันทึกขณะทดสอบระบบควบคุมไกลทั้งสองช่วงมีค่า 0.25 และ 0.19 เมตร ค่าสูงสุดอยู่ที่ 0.41 และ 0.58 เมตร ส่วนเบี่ยงเบนเชิงเส้นเฉลี่ยขณะใช้งานระบบอัตโนมัติทั้งสองช่วงมีค่า 0.25 และ 0.33 เมตร ค่าสูงสุดอยู่ที่ 0.55 และ 0.83 เมตร ค่าความหน่วงเวลาเฉลี่ยและสูงสุดขณะทดสอบอยู่ที่ 109.7 และ 316.6 มิลลิวินาที ตามลำดับ ผู้ควบคุมไม่รู้สึกถึงความยากในการควบคุมหรือการเปลี่ยนโหมดระบบควบคุม จากการสังเกตภาพวิดีโอบนหน้าจอมอนิเตอร์ขณะมีความหน่วงเวลาสูงสุดดังกล่าวพบว่ามีอาการกระตุกบ้างเล็กน้อย ระบบเฝ้าระวังความผิดปกติสามารถตรวจพบปัญหาขณะใช้งานทั้ง 4 จุด ที่ทำการออกแบบไว้ และสามารถหยุดรถเพื่อเข้าสู่โหมดฉุกเฉินได้ การศึกษาระยะเบรกฉับพลันจากการที่ผู้ขับขี่ทำการเบรกปกติบนรถทดสอบ และผลกระทบของความหน่วงเวลาต่อระยะเบรกหากผู้ควบคุมกดปุ่มหยุดรถฉุกเฉินจากระยะไกลขณะมีอัตราเร็ว 3 เมตรต่อวินาที (อัตราเร็วสูงสุดที่ใช้ในการทดสอบ) พบว่าการมีความหน่วงเวลาเฉลี่ยและสูงสุดดังกล่าวเกิดขึ้น ส่งผลให้ระยะเบรกเพิ่มขึ้นจากระยะเบรกปกติของรถ 1.42 เมตร เพิ่มขึ้นเป็น 1.75 และ 2.37 เมตรตามลำดับ หากมีอุปสรรคกีดขวางหน้ารถอย่างกะทันหันภายในระยะดังกล่าว รถทดสอบจะไม่สามารถเบรกได้ทัน ผู้ควบคุมมีความจำเป็นต้องเข้าควบคุมระบบบังคับเลี้ยวจากระยะไกลเพื่อหลีกเลี่ยงสิ่งกีดขวางที่เกิดขึ้นด้วย กรณีมีความหน่วงเวลาเกิดขึ้นมากกว่า 500 มิลลิวินาทีขึ้นไป ณ อัตราเร็วสูงสุดขณะทดสอบ รถทดสอบอาจชนกับอุปสรรคที่กีดขวางกะทันหันในช่วงระยะเบรกปกติของรถก่อนที่ผู้ควบคุมจะเห็นภาพอุปสรรคดังกล่าวจากภาพหน้าจอมอนิเตอร์

สาขาวิชา	วิศวกรรมเครื่องกล	ลายมือชื่อนิสิต .....
ปีการศึกษา	2564	ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

6170434021 : MAJOR MECHANICAL ENGINEERING

KEYWORD: Autonomous Vehicles, Teleoperated Vehicles, Transition system

Krit T.siriwattana : Development of Transition System between Autonomous Driving and 5G Tele-operated Driving. Advisor: Asst. Prof. NUKSIT NOOMWONGS, Ph.D.

Autonomous vehicles are gaining popularity nowadays. There is a lot of competition from both public and private manufacturers to continuously develop their capabilities or levels of driving automation. The highest level (Level 5) is the one that does not require any control from any driver. The lower levels still have to be controlled by the driver in certain situations where the system malfunctions or beyond the system's limited environment. The low-level and high-level controls of the prototype vehicle, teleoperation system and control station were designed and developed under the objectives of this research as well. Including a transition system, which is designed to allow the prototype vehicle to be able to switch control mode at any stage. If a malfunction occurs during use such as a high latency or there is an object obstructing the path. Malfunction surveillance system will switch control mode to emergency mode as soon as the system detects such a situation and alert the situation to the operator at the control station in order to change the control mode. The results of testing the use of autonomous mode and teleoperation mode in all 4 rounds around Chulalongkorn University showed that the system can be used well. The mean linear deviations between the reference paths and the recorded paths while testing the two remote control ranges were 0.25 and 0.19 m. and the maximum were 0.41 and 0.58 m. The mean linear deviations while operating the two automation ranges were 0.25 and 0.33 m. and the maximum were 0.55 and 0.83 m. The mean and maximum latency during testing were 109.7 and 316.6 ms, respectively. The operator did not feel any difficulty in controlling or changing the control mode. Observing the video image on the monitor screen at the highest latency, there was a slight lag. Malfunction surveillance system can detect problems at all 4 designed points and can stop the car to enter emergency mode. The study of stopping distance caused by the driver braking on the vehicle and the effect of the latency on stopping distance if the operator presses the emergency stop button remotely at a speed of 3 m/s (Maximum speed used in the test). It was found that the mean and maximum latency values resulting in an increase in stopping distance from 1.42 m. to 1.75 and 2.37 m. respectively. If there is a sudden obstacle in front of the car within the stopping range. The vehicle will not be able to brake in time. Operators need to remotely control the steering system to avoid obstacles. In the case of a latency more than 500 ms. at the maximum speed of the test. The vehicle may suddenly collide with an obstacle in front of the vehicle before the operator at the control station can see it from the monitor.

Field of Study:	Mechanical Engineering	Student's Signature .....
Academic Year:	2021	Advisor's Signature .....

กิตติกรรมประกาศ

งานวิจัยนี้ได้รับการสนับสนุนจากกองทุนวิจัยและพัฒนากิจการกระจายเสียง กิจการโทรทัศน์ และกิจการโทรคมนาคมเพื่อประโยชน์สาธารณะ สำนักงาน กสทช. บริษัท แอดวานซ์ อินโฟร์ เซอร์วิส จำกัด (มหาชน) บริษัท รถไฟฟ้า (ประเทศไทย) จำกัด (มหาชน) และดำเนินงานภายใต้ศูนย์วิจัยยานยนต์ และระบบขนส่งอัจฉริยะ จุฬาลงกรณ์มหาวิทยาลัย

ขอขอบคุณ ผศ.ดร.นักสิทธ์ นุ่มวงษ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้ให้โอกาสในการศึกษาวิจัย ถ่ายทอดประสบการณ์ ให้ความช่วยเหลือ ให้ความรู้ แนวคิด ชี้แนะจุดบกพร่องและวิธีแก้ไข และให้คำแนะนำในทุก ๆ ด้านที่มีคุณค่ายิ่ง ตลอดระยะเวลาที่ดำเนินการวิจัย ผู้วิจัยขอขอบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้

ขอขอบคุณ ผศ.ดร.สัณหพศ จันทรานุวัฒน์ ผศ.ดร.กฤษฎา พนมเชิง และ ดร.ปาษาณ กุลวานิช กรรมการสอบวิทยานิพนธ์ ที่ได้ให้คำแนะนำ ให้ความรู้และแนวคิดต่าง ๆ ที่เป็นประโยชน์อย่างยิ่งต่อการวิจัย ทำให้ผู้วิจัยนำมาปรับปรุงแก้ไขจนบรรลุผลสำเร็จ

ขอขอบคุณ คุณคณิน เกียรติอร่ามกุล นิสิตปริญญาโท วิศวกรรมเครื่องกล จุฬาลงกรณ์มหาวิทยาลัย และคุณชุมพล เหมพรหมราช ผู้ให้คำแนะนำ ด้านเทคนิค แนวคิดที่สำคัญ และการหาเครื่องมือที่จำเป็นในการวิจัย รวมถึงเพื่อน พี่ น้อง ทุก ๆ คน ที่สละเวลาให้ความช่วยเหลือในทุก ๆ ด้าน

ขอขอบคุณ คุณเสฏฐวุฒิ ลาภวิสุทธิสาโรจน์ คุณอานุภาพ แก้วใจ คุณวรายุทธ แก่นแก้ว และคุณณัฐวรา ชั้นสกุล นักวิจัยแห่งศูนย์วิจัยยานยนต์ และระบบขนส่งอัจฉริยะ จุฬาลงกรณ์มหาวิทยาลัย ที่ได้ให้ความช่วยเหลือ ให้ข้อมูลที่เป็นประโยชน์ในการทำวิจัยจนวิทยานิพนธ์นี้สำเร็จลุล่วงไปได้

สุดท้ายขอขอบคุณครอบครัว ต.ศิริวัฒนา และตันติเศรษฐ์ ที่ให้คำแนะนำ และคอยเป็นกำลังใจให้เสมอมา

กฤษฎ์ ต.ศิริวัฒนา

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1
บทนำ

1. ที่มาและความสำคัญของวิทยานิพนธ์

ปัจจุบันยานยนต์อัตโนมัติกำลังได้รับการพัฒนาอย่างต่อเนื่องทั้งจากผู้ผลิตยานยนต์ และนักวิจัยในสถาบันการศึกษาทั่วโลกโดยการนำเทคโนโลยีด้านต่าง ๆ มาประยุกต์ใช้เพื่อให้ได้ยานยนต์ที่มีความสะดวกสบายและมีความปลอดภัยมากขึ้น ระดับขั้นของระบบขับขี่อัตโนมัติแบ่งเป็น 6 ระดับ (ตั้งแต่ระดับ 0 ถึง 5) ซึ่งกำหนดโดยสมาคมวิศวกรรมยานยนต์นานาชาติ (Society of Automotive Engineers: SAE International) โดยระดับของความเป็นอัตโนมัติจะเพิ่มตามภาระของผู้ขับขี่ที่ลดลงไปจนถึงระดับที่ไม่ต้องการการควบควบคุมจากผู้ขับขี่เลย ในส่วนของระบบอัตโนมัติที่ต่ำกว่าระดับ 5 นั้นยังคงต้องการการควบคุมจากผู้ขับขี่ในบางสถานการณ์ที่ระบบเกิดความผิดปกติ หรือพ้นช่วงสภาพแวดล้อมที่ระบบจำกัด เช่น ตรวจพบสิ่งกีดขวางอยู่ในช่องทางเดินรถที่แคบ และช่องทางด้านข้างเป็นช่องทางเดินรถสวนซึ่งต้องใช้มนุษย์ในการตัดสินใจ โดยระบบจะแจ้งเตือนให้ผู้ขับขี่ทำการควบคุมทันทีเมื่อเจอสถานการณ์ดังกล่าว แต่ในบางกรณีที่ผู้ขับขี่ไม่สามารถทำการควบคุมได้ทันทีเมื่อระบบมีการแจ้งเตือน เช่น ในกรณีที่ผู้ขับขี่เจ็บป่วยกระทันหันและระบบอัตโนมัติดังกล่าวอาจไม่สามารถควบคุมยานพาหนะไปยังบริเวณที่ปลอดภัยได้ หรือกรณีที่ยานยนต์ขนส่งอัตโนมัติเกิดข้อผิดพลาดบางอย่าง และในขณะนั้นไม่มีผู้ควบคุมอยู่ ณ บริเวณนั้นเลย ระบบควบคุมระยะไกลจึงมีความจำเป็นอย่างมากในการเข้าควบคุมในสถานการณ์ดังกล่าว ในทางกลับกันหากมีความหน่วงเวลาของการส่งสัญญาณระยะไกลที่สูง อาจส่งผลให้เกิดความหน่วงเวลาของการทำงาน หรือถึงขั้นทำให้เกิดอุบัติเหตุได้เนื่องจากการควบคุมที่ยากกว่าปกติ ดังนั้นการมีระบบแจ้งเตือนไปยังสถานีควบคุมเพื่อให้ผู้ควบคุมทำการสลับโหมดจากการควบคุมระยะไกลเป็นการขับขี่ด้วยระบบอัตโนมัตินั้นจึงมีความจำเป็นและสำคัญยิ่ง

ในส่วนของวิทยานิพนธ์นี้ผู้วิจัยได้ทำการศึกษาการเปลี่ยนผ่านระหว่างยานยนต์อัตโนมัติและการควบคุมระยะไกล ในสถานการณ์ต่าง ๆ ที่ถูกจำลองขึ้นภายใต้สภาพแวดล้อมที่จำกัด รวมถึงการออกแบบและพัฒนายานยนต์อัตโนมัติต้นแบบที่สามารถควบคุมระยะไกลได้จากสถานีควบคุมผ่านเครือข่าย 5G

2. วัตถุประสงค์การวิจัย

1. เพื่อศึกษา และออกแบบชุดควบคุมสั่งการด้วยสัญญานไฟฟ้า (X-By-Wire) บนโครงสร้างพื้นฐานของรถกอล์ฟไฟฟ้าขนาด 6 ที่นั่ง

2. เพื่อศึกษา และพัฒนาระบบควบคุมยานยนต์ระยะไกลผ่านเครือข่าย 5G รวมถึงการวิเคราะห์ผลกระทบจากความหน่วงเวลาของสัญญาณที่ส่งผลต่อประสิทธิภาพการควบคุมระยะไกล

3. เพื่อศึกษา และออกแบบระบบเปลี่ยนผ่านระหว่างยานยนต์อัตโนมัติต้นแบบกับระบบควบคุมระยะไกล รวมถึงวิเคราะห์การใช้งานระบบเปลี่ยนผ่านในสถานการณ์ต่าง ๆ ภายใต้สภาพแวดล้อมที่จำกัด

3. ขอบเขตการวิจัย

1. การออกแบบและสร้างชุดควบคุมสั่งการด้วยสัญญาณทางไฟฟ้าในระดับต่ำ และการควบคุมในระดับสูงจะถูกพัฒนาบนพื้นฐานของรถกอล์ฟไฟฟ้าขนาด 6 ที่นั่ง เพื่อควบคุมระบบสั่งความเร็ว ระบบเบรก และระบบบังคับเลี้ยว

2. ใช้เฟรมเวิร์คร่วมกันระหว่าง Autoware และ ROS (Robot Operating System) เป็นซอฟต์แวร์ควบคุมระดับสูง (High level control) ในการพัฒนายานยนต์ต้นแบบอัตโนมัติ

3. ระบบการควบคุมยานยนต์จากระยะไกลของงานวิจัยนี้จะถูกพัฒนาบนเครือข่าย 5G แบบ Non-Standalone (NSA) เท่านั้น

4. ยานยนต์อัตโนมัติต้นแบบในงานวิจัยนี้ใช้เซ็นเซอร์ตรวจจับและวัดระยะทางด้วยแสง (Light Detection and Ranging - LiDAR) เป็นเซ็นเซอร์หลักในการสร้างแผนที่สามมิติ (3D-Mapping) การระบุตำแหน่ง (Localization) และการตรวจจับสิ่งกีดขวาง (Obstacle detection)

5. การทดสอบระบบเปลี่ยนผ่านระหว่างยานยนต์อัตโนมัติกับการควบคุมระยะไกลจะถูกทดสอบในสถานการณ์ต่าง ๆ ที่ถูกจำลองขึ้นภายใต้สภาพแวดล้อมที่จำกัด และกำหนดอัตราเร็วที่ใช้ไม่เกิน 15 กิโลเมตร / ชั่วโมง

4. ประโยชน์ที่คาดว่าจะได้รับ

องค์ความรู้ในการพัฒนายานยนต์ขับขี่อัตโนมัติร่วมกับการควบคุมระยะไกลในระดับที่สูงขึ้น

5. ระเบียบวิธีการ

1. ศึกษางานวิจัยที่เกี่ยวข้องกับงานวิจัยนี้ เช่น วิธีการแปลงยานยนต์ทั่วไป (No driving automation) เป็นยานยนต์อัตโนมัติ การควบคุมระยะไกลผ่านเครือข่าย 5G และซอฟต์แวร์ที่ใช้ในการสร้างแผนที่ การระบุตำแหน่ง การวางแผนเส้นทาง (Path planner) และการติดตามเส้นทาง (Path follower)

2. ออกแบบ และติดตั้งชุดโมดูลสั่งการด้วยสัญญานไฟฟ้าในระบบของยานยนต์ เช่น ระบบบังคับเลี้ยว (Steering system) ระบบเร่ง (Acceleration system) ระบบเบรก (Brake system) พร้อมทั้งติดตั้งระบบจ่ายไฟ (Power supply)

3. ออกแบบ และติดตั้งชุดควบคุมอุปกรณ์สั่งงานด้วยสัญญาณทางไฟฟ้าของยานยนต์อัตโนมัติ (Low-Level control) เซ็นเซอร์ที่ใช้ในการตรวจวัด สถานีควบคุมระยะไกล (Cockpit) โมดูลสื่อสารและอุปกรณ์ที่ใช้ควบคุมระยะไกลผ่านเครือข่าย 5G

4. ออกแบบระบบควบคุมยานยนต์อัตโนมัติในระดับสูง (High-Level control) และทดสอบการทำงานซอฟต์แวร์ควบคุมด้วยวิธีการจำลองสถานการณ์ (Simulation)

5. วางแผนการทดสอบการทำงานของทั้งระบบอัตโนมัติและระบบควบคุมระยะไกลในสภาพแวดล้อมจริง

6. ทดสอบการทำงานของระบบควบคุมจากระยะไกลผ่านเครือข่าย 5G และระบบควบคุมอัตโนมัติในสภาพแวดล้อมจริง

7. ทดสอบระบบเปลี่ยนผ่านระหว่างระบบควบคุมอัตโนมัติ และระบบควบคุมจากระยะไกลผ่านเครือข่าย 5G ในสถานการณ์ต่าง ๆ ที่ถูกจำลองขึ้นภายใต้สภาพแวดล้อมจริงที่จำกัด

8. วิเคราะห์และสรุปผลการทดลอง

6. แผนการดำเนินงานและผลลัพธ์แต่ละขั้นตอน

แผนการดำเนินงาน	เวลา (เดือน)																	
	3	6	9	12	15	18	21	24	27	30	33	36	26	28	30	32	34	36
1) ศึกษางานวิจัยที่เกี่ยวข้อง																		
2) ออกแบบ และติดตั้ง (Design and installation)																		
2.1) ระบบเร่งสั่งงานด้วยสัญญาณทางไฟฟ้า																		
2.2) ระบบเบรกสั่งงานด้วยสัญญาณทางไฟฟ้า																		
2.3) ระบบบังคับเลี้ยวสั่งงานด้วยสัญญาณทางไฟฟ้า																		
2.4) ชุดควบคุมอุปกรณ์สั่งงานด้วยสัญญาณทางไฟฟ้าในยานยนต์อัตโนมัติ (Low-Level controller)																		
2.5) เซนเซอร์ตรวจวัด ได้แก่ LiDAR, GPS, IMU																		
2.6) ระบบจ่ายไฟของชุดอุปกรณ์ที่ถูกติดตั้งเพิ่ม																		
2.7) โมดูลสื่อสารและอุปกรณ์ที่ใช้ควบคุมระยะไกล																		
2.8) สถานีควบคุมระยะไกล (Cockpit station)																		
2.9) ซอฟแวร์ควบคุมระดับสูง (High-Level control)																		
3) วางแผน และทดสอบ (Planning and testing)																		
3.1) ชุดอุปกรณ์สั่งงานด้วยสัญญาณทางไฟฟ้า																		
3.2) ระบบควบคุมระยะไกลผ่านเครือข่าย 5G ในสภาพแวดล้อมจริง																		
3.3) ทดสอบการทำงานของซอฟแวร์ควบคุมระดับสูงด้วยวิธีการจำลองสถานการณ์ (Simulation)																		
3.4) ทดสอบการทำงานของซอฟแวร์ควบคุมระดับสูงในสภาพแวดล้อมจริง																		
3.5) ทดสอบระบบเปลี่ยนผ่านระหว่างยานยนต์อัตโนมัติและการควบคุมระยะไกล																		
4) ปรับปรุงแก้ไขระบบควบคุมทั้งระบบ																		
5) สรุปผลการทดลอง																		

บทที่ 2
ปริทัศน์วรรณกรรม

งานวิจัยนี้แบ่งการศึกษา และทบทวนงานวิจัยออกเป็น 4 ส่วนหลัก ได้แก่ งานวิจัยที่เกี่ยวข้องกับการควบคุมอุปกรณ์สั่งงานด้วยสัญญาณทางไฟฟ้า หรือการควบคุมในระดับต่ำ (Low-Level control) งานวิจัยที่เกี่ยวข้องกับการควบคุมจากระยะไกล (Teleoperation) งานวิจัยที่เกี่ยวข้องกับการควบคุมยานยนต์อัตโนมัติในระดับสูง (High-Level control) และงานวิจัยที่เกี่ยวข้องกับการเปลี่ยนผ่านระหว่างระบบควบคุมอัตโนมัติและระบบควบคุมจากระยะไกล (Transition system)

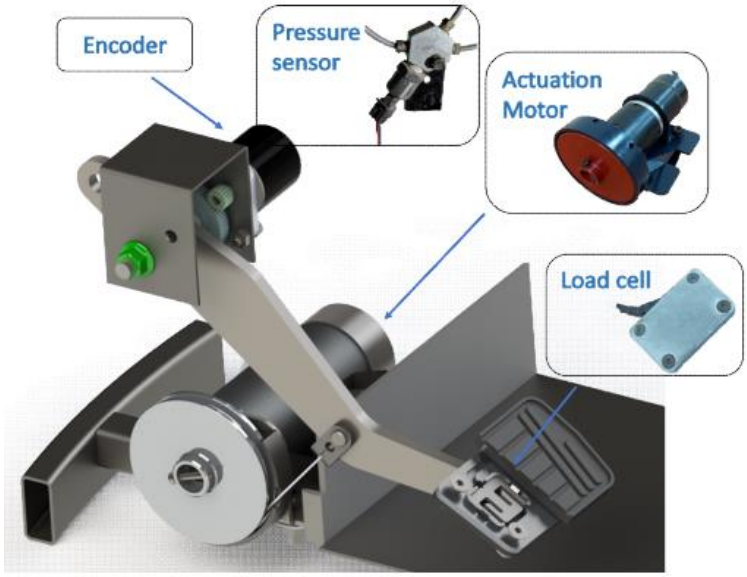
1. การควบคุมอุปกรณ์สั่งงานด้วยสัญญาณทางไฟฟ้า หรือการควบคุมในระดับต่ำ (Low-level Control)

ยานยนต์ หรือ ยานพาหนะยานยนต์โดยทั่วไปประกอบไปด้วยระบบสำคัญหลายระบบ[1] เช่น ระบบส่งกำลัง (Transmission system) ระบบกันสะเทือน (Suspension system) ระบบบังคับเลี้ยว (Steering system) ระบบเร่ง (Throttle system) ระบบเบรก (Braking system) หากกล่าวถึงการพัฒนายานยนต์ทั่วไปที่ไม่มีระบบช่วยเหลือผู้ขับขี่ใด ๆ ให้เป็นยานยนต์อัตโนมัติ การติดตั้งอุปกรณ์ที่สามารถทำงานได้อย่างอัตโนมัติกับระบบของยานยนต์เพื่อทำหน้าที่ควบคุมแทนมนุษย์[2, 3] จึงมีความสำคัญอย่างมาก ซึ่งสามารถทำได้โดยการติดตั้งระบบควบคุมสั่งงานด้วยสัญญาณทางไฟฟ้า (X-By-Wire)[4]

จากการศึกษางานวิจัย[5, 6] พบว่างานวิจัยนี้ได้มีการพัฒนารถกอล์ฟไฟฟ้าขนาด 14 ที่นั่งเป็นรถอัตโนมัติโดยเริ่มจากการใช้ชุดโมดูลสั่งงานด้วยสัญญาณทางไฟฟ้าที่ถูกติดตั้งมาอยู่แล้วกับระบบบังคับเลี้ยว ระบบเร่ง และเพิ่มเติมตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าในระบบเบรก ซึ่ง 3 ระบบดังกล่าวถือเป็นระบบในระดับต่ำของยานยนต์ (Low-Level system)[7] ในส่วนของระบบบังคับเลี้ยวงานวิจัยนี้เลือกใช้ระบบพวงมาลัยไฟฟ้า EPS (Electric Power Steering) ซึ่งสามารถเลียนแบบค่าสัญญาณแรงบิดที่เชื่อมต่อกับตัวควบคุมที่พัฒนาขึ้นได้เลยดังภาพที่ 1 การทดสอบระบุว่าวิธีนี้ให้ผลดีกว่าการติดตั้งตัวกระตุ้นจากภายนอก (Actuator) ส่วนของระบบเบรกสั่งงานด้วยสัญญาณทางไฟฟ้าได้ทำการเพิ่มตัวกระตุ้นโดยใช้มอเตอร์กระแสตรงไร้แปรงถ่าน (Brushless dc motor) เพื่อควบคุม

แป้นเบรกผ่าน สายสลิงและรอกดังภาพที่ 2 ระบบนี้ให้การตอบสนองที่รวดเร็ว และมีแรงบิดที่เพียงพอสำหรับการใช้งานรถที่ความเร็วไม่เกิน 30 กิโลเมตรต่อชั่วโมง และในส่วนของระบบเร่งสั่งงานด้วยสัญญาณทางไฟฟ้า เนื่องจากรถที่นำมาพัฒนาเป็นรถที่ใช้มอเตอร์ไฟฟ้าในการขับเคลื่อน จึงสามารถสั่งงานมอเตอร์ไฟฟ้าได้โดยการสร้างสัญญาณเลียนแบบที่มีลักษณะเดียวกับสัญญาณของคันเร่งไฟฟ้า

ภาพที่ 1 *ลักษณะระบบบังคับเลี้ยวสั่งงานด้วยสัญญาณทางไฟฟ้า[5, 6]*

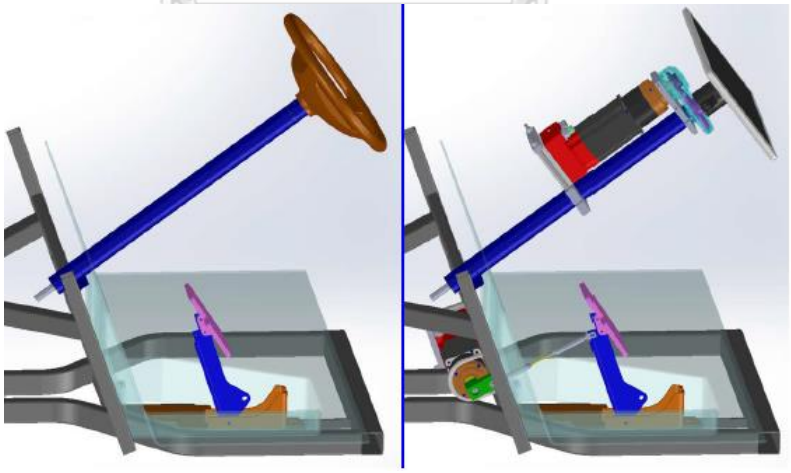


ภาพที่ 2 *ลักษณะระบบเบรกสั่งงานด้วยสัญญาณทางไฟฟ้า[5, 6]*

จากการศึกษางานวิจัยเพิ่มเติมพบว่างานวิจัย[8] ได้ทำการพัฒนารถกอล์ฟไฟฟ้า ขนาด 2 ที่นั่ง เป็นรถกอล์ฟไฟฟ้าขับขี่อัตโนมัติดังภาพที่ 3 โดยการดัดแปลงและเพิ่มเติมชุดอุปกรณ์ควบคุมสั่งงานด้วยสัญญาณทางไฟฟ้าเช่นเดียวกับงานวิจัย [5, 6] แต่ในระบบบังคับเลี้ยวสั่งงานด้วยสัญญาณ

ทางไฟฟ้าจะมีความแตกต่างกับงานวิจัยที่กล่าวมา งานวิจัยนี้เลือกใช้ตัวกระตุ้นจากภายนอกติดตั้งบริเวณคอพวงมาลัย (Steering column) ซึ่งประกอบไปด้วย เซ็นเซอร์วัดการหมุน (Encoder) เกียร์ทด และมอเตอร์ที่ให้ขนาดแรงบิดที่กระทำต่อพวงมาลัยเท่ากับขนาดแรงบิดจริงของผู้ขับขี่ที่กระทำการหมุนพวงมาลัย ลักษณะการติดตั้งตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าเป็นไปตามภาพที่ 4

ภาพที่ 3 *รถกอล์ฟไฟฟ้าที่ถูกดัดแปลงและเพิ่มเติมชุดอุปกรณ์ควบคุมให้เป็นรถอัตโนมัติในงานวิจัย[8]*

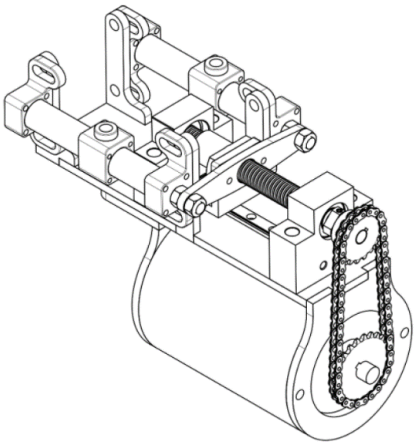


ภาพที่ 4 *ลักษณะการติดตั้งตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าในระบบบังคับเลี้ยว และระบบเบรกของงานวิจัย[8]*

ต่อมาพบว่างานวิจัย[9] ได้พัฒนารถ TOYOTA COMS ซึ่งเดิมทีเป็นรถขับเคลื่อนด้วยพลังงานไฟฟ้า 1 ที่นั่ง และถูกพัฒนาเป็นยานยนต์ขับขี่อัตโนมัติในระดับที่ 3 (Conditional

Automation) โดยใช้ดาวเทียมนำร่องในการระบุตำแหน่งสำหรับการควบคุมระดับสูง ดังภาพที่ 5 งานวิจัยนี้ได้ทำการออกแบบตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าในระบบเบรกโดยการสร้างชุดไฮดรอลิกที่ใช้มอเตอร์กระแสตรงในการดันลูกสูบเพื่อเพิ่มแรงดันน้ำมันโดยตรงกับล้อหน้า และหลัง ดังภาพที่ 6 ข้อดีของระบบนี้คือขณะสั่งเบรกด้วยสัญญาณทางไฟฟ้าจะไม่พบแป้นเบรกขยับไปมา ซึ่งแตกต่างจากระบบเบรกของงานวิจัย [5, 6] [8] ที่มีการเชื่อมก้านต่อระหว่างมอเตอร์กับแป้นเบรกเข้าด้วยกัน

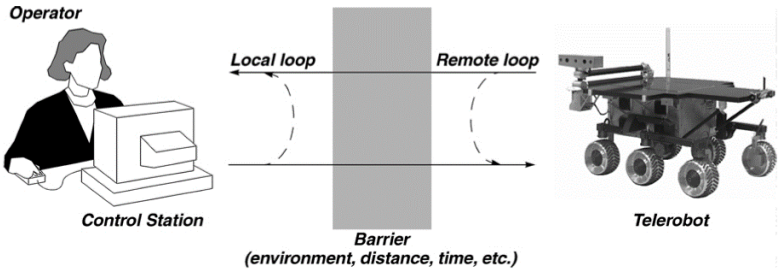
ภาพที่ 5 *รถยนต์ขับเคลื่อนด้วยพลังงานไฟฟ้า TOYOTA COMS ที่ถูกดัดแปลงเป็นรถอัตโนมัติระดับที่ 3 [9]*



ภาพที่ 6 *ตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าในระบบเบรก[9]*

2. การควบคุมจากระยะไกล (Teleoperation)

งานวิจัย[10] ได้อธิบายความหมายของการควบคุมระยะไกลด้วยวิธี Teleoperation ซึ่งหมายถึงผู้ควบคุมสามารถควบคุมอุปกรณ์ได้จากระยะไกลโดยไม่ต้องเห็นอุปกรณ์นั้น ๆ ในแนวสายตา โดยอาจใช้กล้อง เซ็นเซอร์ หรือซอฟท์แวร์เพื่อช่วยในการเพิ่มประสิทธิภาพการควบคุมให้ดียิ่งขึ้น ข้อมูลจากภาพวิดีโอ หรือเซ็นเซอร์ของอุปกรณ์จะถูกส่งมายังสถานีควบคุม (Cockpit) ที่มีผู้ควบคุมคอยสังเกตการณ์ และส่งคำสั่งควบคุมกลับไปยังอุปกรณ์ดังภาพที่ 7 วิธีนี้จะแตกต่างจากการควบคุมด้วยวิธี Remote Control ซึ่งผู้ควบคุมจำเป็นต้องควบคุมอุปกรณ์ภายในแนวสายตาเท่านั้น



ภาพที่ 7 *ลักษณะการควบคุมระยะไกลด้วยวิธี Teleoperation[10]*

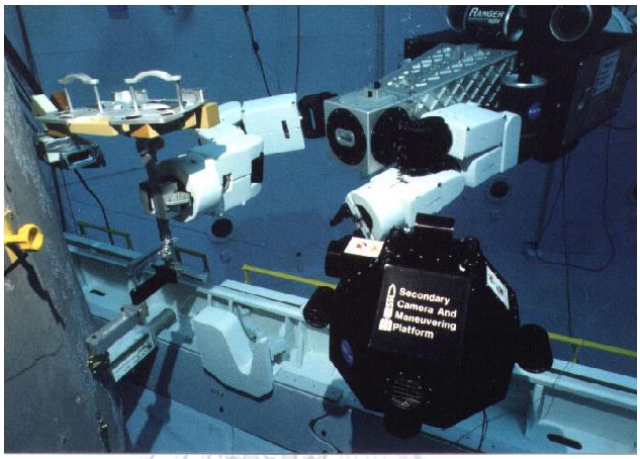
งานวิจัย[10] ได้มีการศึกษาอินเตอร์เฟสที่มีอยู่และถูกใช้งานในการควบคุมยานยนต์จากระยะไกล (Vehicle Teleoperation Interface) งานวิจัยนี้ได้จัดประเภทอินเตอร์เฟสการควบคุมระยะไกล ออกเป็น 4 ประเภท ประเภทแรกคือ Direct interface ผู้ขับขี่สามารถควบคุมยานยนต์ด้วยมือขณะดูภาพวิดีโอ หรืออาจถูกเรียกว่าการขับขี่จากภายในสู่ภายนอก (Inside-out) อินเตอร์เฟสนี้จะให้ผลดีเมื่อความหน่วงเวลามีค่าอยู่ในระดับต่ำเนื่องจากไม่มีระบบช่วยเหลืออื่นในการช่วยตัดสินใจ ซึ่งอาจส่งผลให้เกิดความล้มเหลวในการควบคุม ประเภทที่สอง คือ Multimodal/Multisensor interface นอกจากการควบคุมผ่านการมองภาพวิดีโอเหมือนประเภทแรก อินเตอร์เฟสนี้ได้เพิ่มโหมดการควบคุมที่มีลักษณะเฉพาะเพื่อให้สามารถทำงานที่ซับซ้อนได้ โดยอาจจะใช้ข้อมูลจากเซ็นเซอร์ที่ติดอยู่กับอุปกรณ์ หรืออินเตอร์เฟสอื่น ๆ ในการประมวลผลและสั่งการเพื่อเพิ่มประสิทธิภาพในการควบคุม ประเภทที่สาม คือ Supervisory control interface การควบคุมในระดับสูงถูกเปิดใช้ในอินเตอร์เฟสนี้เพื่อทำงานในบางสถานการณ์ กล่าวคืออุปกรณ์ที่ถูกควบคุมจำเป็นต้องมีระดับความเป็นอัตโนมัติระดับใดระดับหนึ่ง โดยที่ผู้ควบคุมมีหน้าที่นำทางและสร้างคำสั่งการเคลื่อนไหวเป็นหลัก และอินเตอร์เฟสประเภทสุดท้าย คือ Novel- interface

อินเตอร์เฟสประเภทนี้อาจเป็นเรื่องแปลกใหม่เหมือนกับนวนิยายตามชื่อของอินเตอร์เฟส เนื่องจากบางงานวิจัยที่ใช้งานอินเตอร์เฟสนี้ใช้วิธีการป้อนข้อมูลต่างจากอินเตอร์เฟสที่กล่าวมา ซึ่งอาจจะใช้วิธีการควบคุมระยะไกลจากการใช้คลื่นสมองและกล้ามเนื้อเพื่อกำหนดทิศทางการเคลื่อนที่ของอุปกรณ์เปรียบเสมือนกับ “อวตารในโลกแห่งความเป็นจริง” [10]

งานวิจัย[11] ได้แบ่งองค์ประกอบของยานยนต์ไร้คนขับ (Unmanned Vehicles: UAV) ที่ควบคุมด้วยวิธี Teleoperation ออกเป็น 3 องค์ประกอบ องค์ประกอบแรกคือ *The Teleoperator* งานวิจัยนี้เลือกใช้ Audi Q7 เป็นยานยนต์ทดสอบที่ติดตั้งเซ็นเซอร์และตัวกระตุ้นที่จำเป็นไว้ กล้องความละเอียด 640 x 480 จำนวน 8 กล้องถูกติดบนยานยนต์ทดสอบ โดยยึดหลักการติดตั้งตาม The European Parliament and of the Councill เพื่อให้ครอบคลุม ต้องมีมุมมองแนวนอนอย่างน้อย 180 องศาที่ด้านหน้าเพื่อบังคับรถ มุมมองกระจกมองข้างอย่างน้อยควรมีประมาณ 12 องศา มุมมองด้านหลังประมาณ 20 องศา และเพื่อให้ภาพวิดีโอราบรื่น อัตราเฟรมต่อวินาทีถูกตั้งไว้ที่ 25 เฟรมต่อวินาที (fps) เช่นเดียวกับในมาตรฐานวิดีโอ European PAL องค์ประกอบต่อมา คือ *Communication link* ความหน่วงเวลาและแบนด์วิดท์ในการสื่อสารมีความสำคัญในงานควบคุมระยะไกลอย่างมาก งานวิจัยนี้เลือกใช้การเชื่อมต่ออินเทอร์เน็ตผ่าน 3G โดยมี 4G เป็นช่องทางการสื่อสารซึ่งมีให้บริการในขณะนั้น จากการทดสอบแบนด์วิดท์จริงพบว่าขึ้นอยู่กับความแรงของสัญญาณและจำนวนผู้ใช้ในเซลล์เครือข่าย การวัดค่าความหน่วงเวลาของการส่งทั้งภาพวิดีโอและการส่งสัญญาณควบคุมในงานวิจัยนี้ใช้วิธีวัดแบบไป-กลับ (Round Trip Time: RTT) แต่ไม่รวมเวลาในการดำเนินการบน CarPC ของผู้ปฏิบัติงาน องค์ประกอบสุดท้าย คือ *Operator interface* สถานีควบคุมติดตั้งจอ 3 จอเพื่อแสดงภาพจากกล้องที่ติดตั้งบนยานยนต์ รวมถึงอุปกรณ์ควบคุมที่จำเป็นต่าง ๆ เช่น พวงมาลัย แป้นเบรค แป้นคันเร่ง ผลทดสอบการขับขี่ระยะไกลพบว่าไม่มีปัญหาจากการใช้งานที่มีความหน่วงเวลา 500 มิลลิวินาที ขณะขับที่ความเร็ว 30 กิโลเมตรต่อชั่วโมง

งานวิจัย[12] ได้ศึกษาผลกระทบจากความหน่วงเวลา (Delay) ที่เกิดขึ้นในระหว่างควบคุมแพลตฟอร์มกล้องใต้น้ำ หรือ SCAMP (Secondary Camera and Maneuvering Platform) ดังภาพที่ 8 งานวิจัยนี้ได้ให้ข้อสรุปว่าการมีทั้งความหน่วงเวลาของการส่งสัญญาณที่สูง และองศาความอิสระ (Degrees of freedom: DOF) ของอุปกรณ์ที่ถูกควบคุมในระดับที่มากขึ้น ส่งผลให้ระยะเวลาในการดำเนินงานเสร็จสิ้นทั้งหมดสูงขึ้นตามไปด้วย งานวิจัยนี้ได้มีการศึกษาและแก้ไขปัญหาการควบคุมอุปกรณ์จากระยะไกลเมื่อมีความหน่วงเวลาสูงขึ้นชั่วขณะ ด้วยวิธีการตรวจสอบ time stamp เป็นระยะ ๆ ในกรณีที่มีความหน่วงเวลาสูงขึ้น ระบบจะเก็บคำสั่งควบคุมไว้ในบัฟเฟอร์และรอให้ค่า

ความหน่วงเวลาต่ำลง จึงส่งคำสั่งดำเนินการต่อไป วิธีนี้เหมาะสมกับงานที่มีความหน่วงเวลาค่อนข้างคงที่ แต่ทว่าวิธีจัดการกับปัญหาความหน่วงเวลาที่ดีที่สุด นั่นคือการที่ผู้ควบคุมต้องทำการป้อนคำสั่งให้น้อยที่สุด “zero or minimal operator input” ซึ่งอาจจะใช้ระบบอัตโนมัติทำงานในบางสถานการณ์ เช่น การสั่งให้แขนกลของหุ่นยนต์กลับมายังจุดเริ่มต้นด้วยระบบอัตโนมัติ

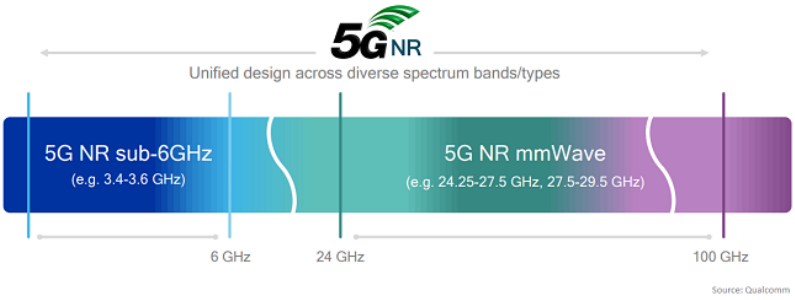


ภาพที่ 8 *Secondary Camera and Maneuvering Platform (SCAMP)[12]*

ในงานที่มีการรับ-ส่งข้อมูลภาพวิดีโอที่ค่อนข้างมาก และต้องการความเร็วในการตอบสนองแบบเรียลไทม์ เช่น งานการควบคุมยานยนต์จากระยะไกล ซึ่งต้องมีการสื่อสารกันระหว่างสถานีควบคุม และยานยานยนต์ตลอดเวลา การสื่อสารจำเป็นต้องมีประสิทธิภาพมากพอเพื่อตอบโจทย์การใช้งานระบบควบคุมดังกล่าว ณ ปัจจุบันนี้เทคโนโลยีรุ่นที่ห้าสำหรับเครือข่ายเซลลูลาร์ (5G) หรือ 5G NR ที่ถูกพัฒนาขึ้นโดยองค์กร 3rd Generation Partnership Project (3GPP) ซึ่งถูกนำไปใช้งานในหลายประเทศรวมทั้งประเทศไทย เมื่อเทียบกับเทคโนโลยี 4G เทคโนโลยีนี้มีความหน่วงเวลาที่ต่ำกว่า (Low latency) และแบนด์วิดท์ที่สูงกว่า 4G หลายเท่า[13] เกณฑ์มาตรฐาน IMT-2020 ได้แบ่งขีดความสามารถการตอบสนองต่อกลุ่มผู้ใช้งานในอีกหลายกลุ่ม เช่น การใช้งานที่รองรับการเชื่อมต่อของอุปกรณ์จำนวนมากในพื้นที่เดียวกัน (massive Machine Type Communications: mMTC) การใช้งานที่มีอัตราการส่งข้อมูลที่สูง (enhanced Mobile Broadband: eMBB) และการใช้งานที่ต้องการความเสถียรภาพสูง ความหน่วงเวลาที่ต่ำ (Ultra-reliable and Low-Latency Communications: URLLC) ซึ่งเหมาะกับกลุ่มผู้ใช้งานด้านยานยนต์อัตโนมัติ[14]

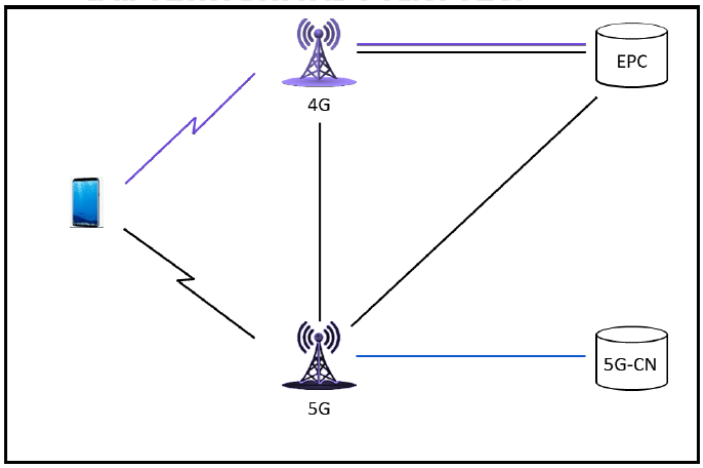
ในปัจจุบันนี้มาตรฐานการให้บริการเครือข่าย 5G ที่มีให้บริการโดยทั่วไป คือ คลื่นความถี่ Sub-6 ซึ่งเป็นคลื่นที่มีความถี่ต่ำกว่า 6 GHz ดังภาพที่ 9 ส่วนคลื่นความถี่ที่มากกว่า 6 GHz ขึ้นไป

นั้น จะอยู่ในระดับ mmWave หรืออยู่ในช่วง High-band คลื่นประเภทนี้มีการใช้กับงานบางประเภทเท่านั้น ตามทฤษฎีแล้ว คลื่นความถี่ที่สูง ความเร็วในการรับ-ส่งข้อมูลจะสูงมาก แต่ยังมีข้อเสียคือระยะการใช้งานใกล้มากเนื่องจากความยาวคลื่นที่สั้น ส่งผลให้โครงสร้างอุปกรณ์รับ-ส่งขั้นพื้นฐานอาจจำเป็นต้องติดตั้งเพิ่มขึ้นเพื่อชดเชยปัญหาดังกล่าว อีกทั้งความสามารถในการทะลุทะลวงผ่านสิ่งกีดขวางหรืออุปสรรคบางประเภทอาจเป็นไปได้ยาก ซึ่งนำไปสู่การครอบคลุมของสัญญาณที่ไม่ดีและสัญญาณขาดหายในบางช่วง[15] การให้บริการ 5G แบบ Sub-6 จึงเหมาะกับผู้ใช้งานทั่วไป



ภาพที่ 9 *ช่วงคลื่นความถี่ของเทคโนโลยีรุ่นที่ห้าสำหรับเครือข่ายเซลลูลาร์*
https://www.rfpage.com/what-are-the-challenges-in-5g-technology/

งานวิจัย[16, 17] ได้มีการศึกษาโหมดการให้บริการเครือข่าย 5G แบ่งย่อยเป็น 2 ประเภท คือ 5G Non-Standalone (5G NSA) และ 5G Standalone (5G SA) ดังภาพที่ 10



ภาพที่ 10 *ประเภทของการให้บริการเทคโนโลยีรุ่นที่ห้าสำหรับเครือข่ายเซลลูลาร์ทั้งมาตรฐาน SA และ NSA [16, 17]*

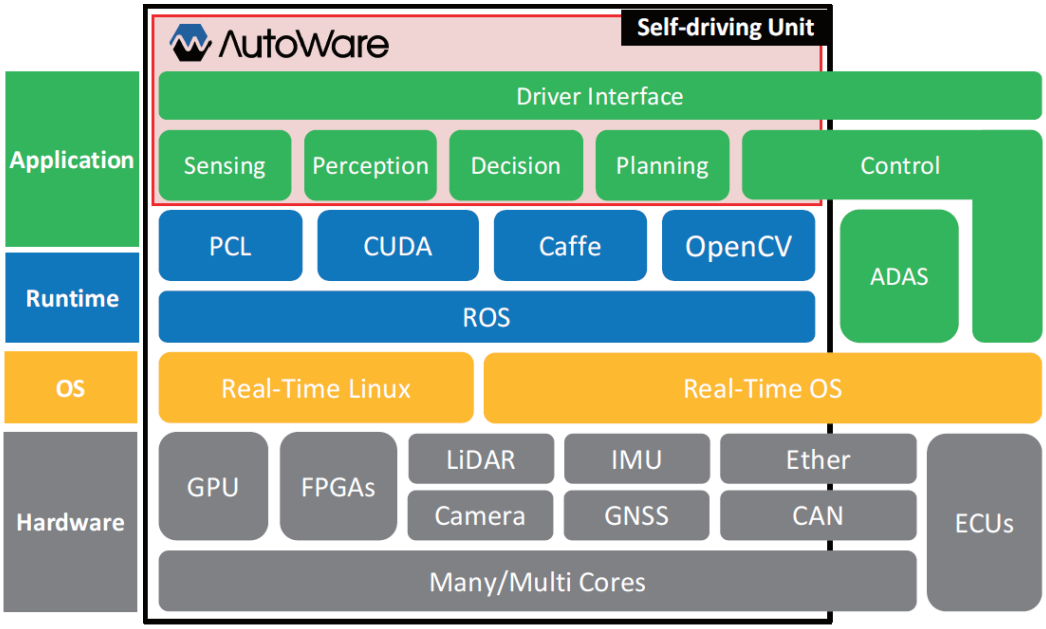
มาตรฐานการให้บริการแบบ NSA จะเป็นการใช้งาน 5G บนโครงข่ายเทคโนโลยีของ 4G เกือบทั้งหมด การใช้งานของมาตรฐานนี้จะมีการใช้งานทั้งจากโหนดหลัก (Master) และโหนดรอง (Secondary) ผลคือทำให้ความสามารถในการเชื่อมต่อกับอุปกรณ์ได้เพิ่มขึ้น และได้ความหน่วงเวลาที่ต่ำลง ข้อดีคือค่าใช้จ่ายในการลงโครงสร้างพื้นฐานที่ต่ำกว่า และสัญญาณครอบคลุมพื้นที่ได้มากกว่าเมื่อเทียบกับการให้บริการแบบ SA ส่วนมาตรฐาน 5G SA เป็นการใช้งานที่อยู่บนโครงสร้างเครือข่ายพื้นฐานของ 5G ทั้งหมด ทั้งในการเชื่อมต่อและการรับ-ส่งข้อมูล และยังรองรับการใช้งานแบบแยกส่วน (Network Slicing) จึงทำให้อัตราเร็วในการสื่อสารมีความรวดเร็ว มีเสถียรภาพมากขึ้น ความหน่วงต่ำ และปริมาณการส่งข้อมูลที่แนวโน้มสูงกว่า 5G NSA

3. การควบคุมในระดับสูง (High-level Control)

3.1 ซอฟต์แวร์การควบคุมระดับสูง

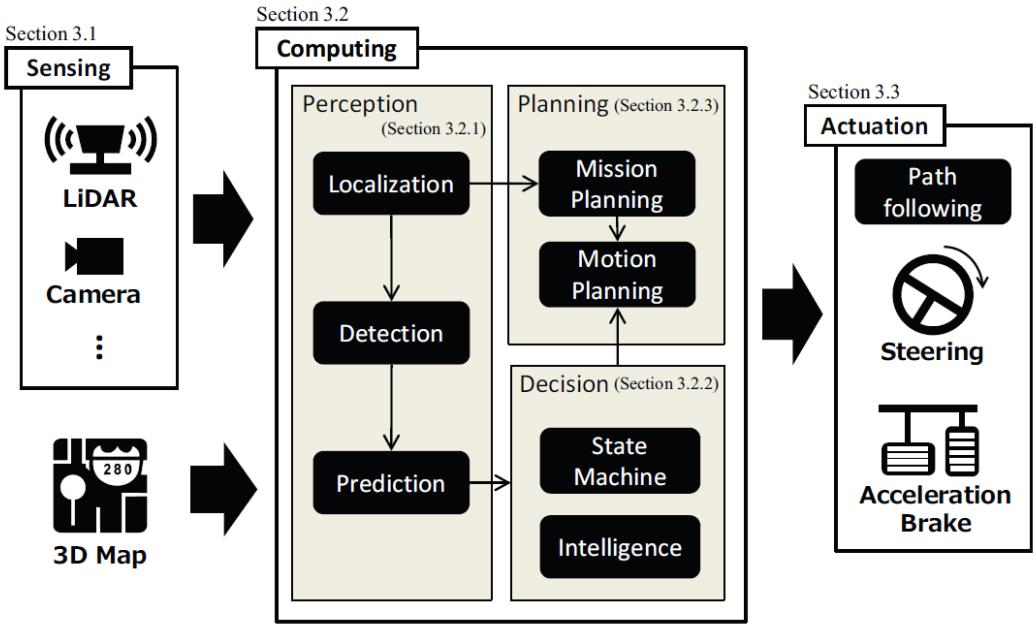
Autoware[18, 19] แอพพลิเคชั่น open source สำหรับการพัฒนายานยนต์ขับขี่อัตโนมัติโดยเฉพาะ แอพพลิเคชั่นนี้ทำงานบนเฟรมเวิร์ค Robot Operating System (ROS)[20] ที่อยู่บนระบบปฏิบัติการ Linux ดังภาพที่ 11 ROS ได้รับความนิยมอย่างมากในงานวิจัยด้านการพัฒนาหุ่นยนต์ ยานยนต์อัตโนมัติทั้งบนบก น้ำ อากาศ และอีกหลายงานวิจัยในช่วงเวลาที่ผ่านมา ROS มีการแบ่งประเภทหน้าที่การทำงานของระบบออกเป็นระบบย่อย มีเครื่องมือช่วยเหลือ ชุดคำสั่ง และอินเตอร์เฟสการสื่อสารที่จำเป็น ด้วยเหตุนี้จึงทำให้จัดการกับระบบที่ซับซ้อนได้ มีประสิทธิภาพการใช้งานที่สูง

ส่วนแอพพลิเคชั่น Autoware จะเน้นไปทางการพัฒนาเทคโนโลยีการขับขี่อัตโนมัติโดยเฉพาะ (Self-driving modules) เนื่องจากรวบรวมโมดูลที่สำคัญไว้ เช่น โมดูลที่เกี่ยวข้องกับการรับรู้ (Sensing) โมดูลที่เกี่ยวข้องกับการคำนวณ (Computing) โมดูลที่เกี่ยวข้องกับการควบคุม (Actuation) และอีกหลายโมดูลที่จำเป็น มีอินเตอร์เฟสกับผู้ใช้ที่สามารถเข้าใจได้ง่าย ทำให้นักพัฒนามีเวลามากขึ้นในการพัฒนาฟีเจอร์ใหม่ ๆ ระหว่างที่แอพพลิเคชั่น และซอฟต์แวร์เฟิร์มเวิร์คดังกล่าวอยู่เบื้องหลังของการทำงาน



ภาพที่ 11 *ภาพรวมโครงสร้างของแอพพลิเคชั่น Autoware* [18]

หลักการทำงานของแอพพลิเคชั่น Autoware ในการสั่งการและควบคุมระบบอัตโนมัติ เริ่มต้นด้วยการรับข้อมูลจากเซ็นเซอร์ตรวจวัดสภาพแวดล้อมโดยรอบของยานพาหนะ เช่น กล้อง เรดาร์ (Radio Detection And Ranging: Radar) ระบบระบุตำแหน่งบนพื้นโลก (Global Navigation Satellite System: GNSS) และเซ็นเซอร์ตรวจจับและวัดระยะด้วยแสง (Light Detection and Ranging: LiDAR) รวมถึงข้อมูลแผนที่สามมิติ (3D-mapping) เข้ามาคำนวณด้วยชุดซอฟต์แวร์ที่ทำหน้าที่ในการระบุตำแหน่ง (Localization) การตรวจจับ การวางแผนเส้นทางทั้งในระดับ Mission planning และ Motion planning รวมถึงการตามเส้นทาง (Path follower) ค่าเอาต์พุตที่ได้จากการคำนวณจะอยู่ในรูปแบบของความเร็วเชิงเส้น และความเร็วเชิงมุมของยานพาหนะที่ควรจะเป็น ทั้งสองพารามิเตอร์นี้จะถูกส่งไปยังส่วนของการควบคุมในระดับต่ำ ซึ่งมีหน้าที่ในการสั่งการและควบคุมยานพาหนะให้เคลื่อนที่ได้ตามกำหนด ดังภาพที่ 12



ภาพที่ 12 *การควบคุมขั้นพื้นฐานของยานยนต์ขับขี่อัตโนมัติ* [18]

3.2 การสร้างแผนที่สามมิติ และการระบุตำแหน่ง (3D-Mapping and Localization)

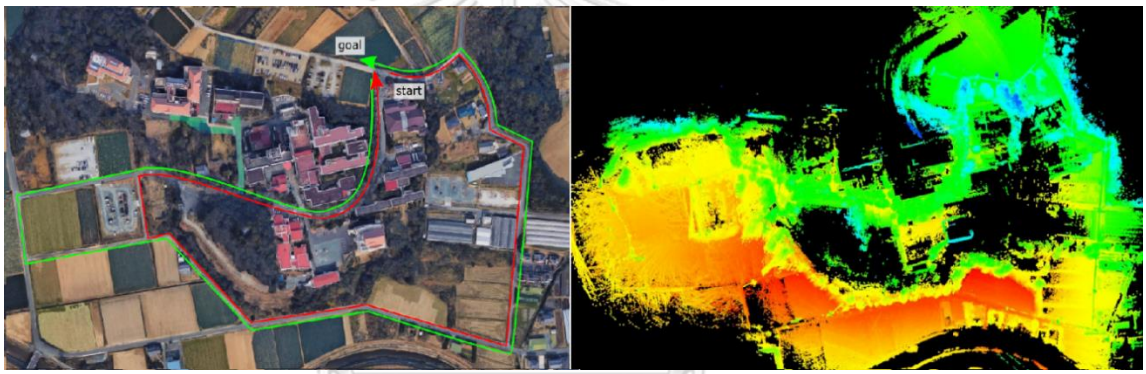
งานวิจัยนี้ ผู้เขียนได้ทำการศึกษาการสร้างแผนที่สามมิติ และการระบุตำแหน่งเพิ่มเติมจาก outsource แต่ในส่วนของโมดูลการคำนวณและโมดูลการควบคุมจะยังคงใช้โมดูลจากแอพพลิเคชั่น Autoware ที่จัดเตรียมไว้ให้

การสร้างแผนที่ และการระบุตำแหน่งมักเป็นหนึ่งในปัญหาที่เดิมทีเกิดขึ้นในการสร้างหุ่นยนต์ที่ต้องการการเคลื่อนที่และระบุตำแหน่งไปพร้อมกัน (Simultaneous Localization and Mapping – SLAM) จากการศึกษาพบว่างานวิจัย[21] ได้นำเสนอวิธีเพิ่มประสิทธิภาพในการระบุตำแหน่งพร้อมกับสร้างแผนที่ด้วยวิธีกราฟ (Graph SLAM) วิธีนี้จะชดเชยข้อผิดพลาดในการหมุนที่สะสมของการจับคู่การสแกน แต่ยังคงมีค่าความคลาดเคลื่อนสะสมในการสร้างแผนที่ซึ่งจะทำให้แผนที่และการระบุตำแหน่งที่คำนวณได้มีความผิดเพี้ยนไปบ้าง

เซ็นเซอร์ที่นิยมใช้กันทั่วไปในงานนี้ได้แก่ เซ็นเซอร์ตรวจจับและวัดระยะด้วยแสง และกล้องวิดีโอ ซึ่งเซ็นเซอร์ทั้งสองประเภทนี้มีข้อดีและข้อเสียต่างกัน งานวิจัย [5] ได้เผยให้เห็นว่า การใช้เซ็นเซอร์ตรวจจับและวัดระยะด้วยแสงในการสร้างแผนที่แบบกริดให้ผลที่ดีกว่าเมื่อเทียบกับการใช้กล้อง เนื่องจากเซ็นเซอร์แสงมีความแม่นยำมากกว่าในการวัดระยะทางทั้งในเรื่องของการวัดความลึก

หรือวัดช่องว่างระหว่างยานพาหนะ แต่ข้อเสียคือมีราคาสูง ใช้พลังงานมาก การรับค่าสีหรือการบ่งบอกลักษณะเฉพาะของวัตถุนั้นเป็นเรื่องยาก

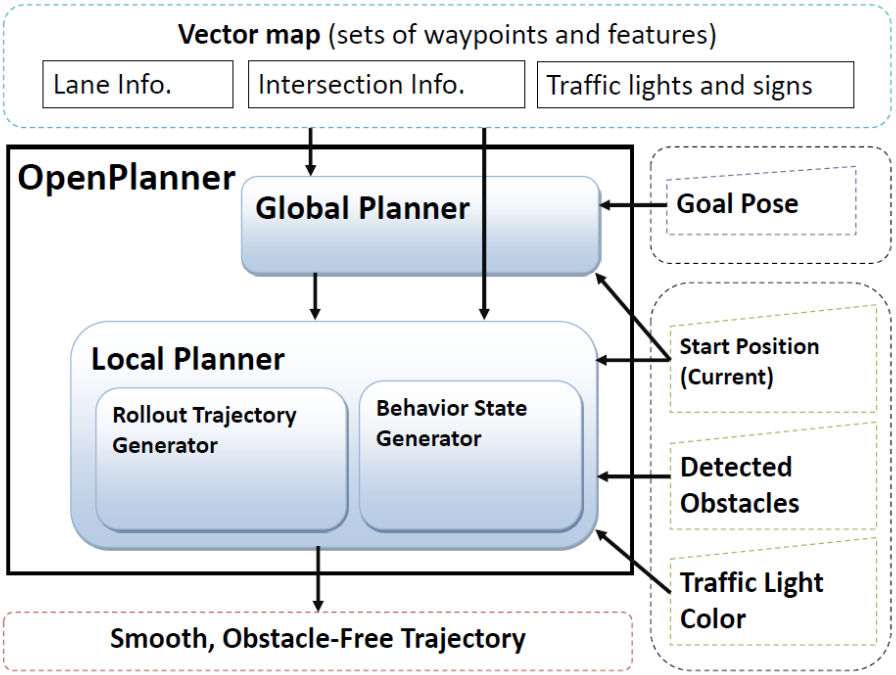
ต่อมาพบว่ามีงานวิจัยหนึ่งได้เสนอแนวทางการลดค่าความคลาดเคลื่อนสะสมของ Graph SLAM จากการเพิ่มตัวจับลูป (Loop-detection) วิธีนี้เรียกว่า HDL - Graph SLAM[22] ซึ่งจะทำงานร่วมกับอัลกอริทึม Normal Distributions Transform (NDT)[23] ซึ่งเป็นอัลกอลิทึมที่ Autoware ใช้ในการสร้างแผนที่และระบุตำแหน่ง นอกจากนี้ HDL - Graph SLAM ยังเพิ่มข้อจำกัดในการตรวจจับระนาบพื้น (Ground Plane Constraint) และการระบุตำแหน่งโดยใช้ดาวเทียมเพื่อการชดเชยข้อผิดพลาดจากการหมุนสะสมของการจับคู่สแกนทั้งในระบบ Indoor และ Outdoor ตามลำดับ ดังภาพที่ 13



ภาพที่ 13 *แผนที่สามมิติที่สร้างขึ้นด้วยวิธี HDL - Graph SLAM [21]*

3.3. ระบบวางแผนเส้นทาง (Path Planning System)

OpenPlanner[24] แพคเก็จที่รวบรวมไลบรารี่ที่สำคัญในระบบนำทางของยานยนต์อัตโนมัติ ดังภาพที่ 14 ซึ่งรองรับการใช้งานร่วมกับแอพพลิเคชั่น Autoware [18] [19] แพ็คเกจนี้ใช้ vector map เป็นข้อมูลอินพุทสำหรับการคำนวณในระดับ global planner ระบบจะเลือกเส้นทางที่สั้นที่สุดหรือเส้นทางที่ lowest cost ที่สุดของจุดเริ่มต้นและจุดสุดท้ายที่ผู้ใช้กำหนด การคำนวณที่ได้จะถูกนำไปใช้ต่อในระดับ local path planner เพื่อสร้าง roll-out trajectories ที่เหมาะสมสำหรับยานยนต์อัตโนมัติ



ภาพที่ 14 *โครงสร้างของ OpenPlanner [24]*

3.4. ระบบตามแผนเส้นทาง (Path Follower System)

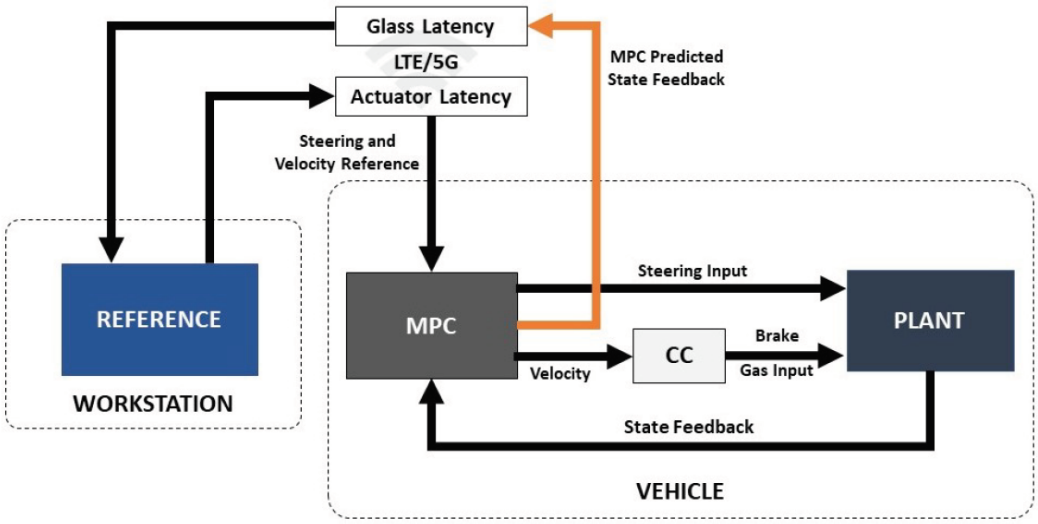
ส่วนนี้เป็นส่วนเดียวกับ local path planner ที่อยู่ในแพ็คเกจ OpenPlanner งานวิจัย[25] ได้สร้างอัลกอลิทึมที่เรียกว่า Pure Puresuit ซึ่งเป็นอัลกอลิทึมที่ใช้ในการติดตาม และค้นหาจุดอ้างอิง (Way-point) เป้าหมายใหม่ไปเรื่อย ๆ ของเส้นทางที่ถูกกำหนดโดย local path planner พร้อมทั้งคำนวณอัตราเร็วเชิงเส้นและเชิงมุมของยานยนต์ที่ควรเป็นตามจุดอ้างอิงตลอดเส้นทาง เอาต์พุตที่ได้จากอัลกอลิทึมจะถูกส่งข้อมูลไปยังส่วนของ Actuation เพื่อทำการควบคุม Low-level system ให้ได้อัตราเร็วตามที่ระบบกำหนด

4. ระบบเปลี่ยนผ่านระหว่างการควบคุมด้วยระบบอัตโนมัติและการควบคุมด้วยระบบควบคุมจากระยะไกล (Transition System)

งานวิจัย[26] นำเสนอแนวทาง ASS (Active Safety System) ซึ่งใช้แบบจำลอง MPC (Model Predictive Control) สำหรับการขับขี่ระยะไกล เพื่อช่วยเพิ่มความมั่นใจให้กับผู้ขับขี่ในสภาพแวดล้อมที่ซับซ้อน หรือความหน่วงเวลาที่สูง เนื่องจากสามารถคาดการณ์ตำแหน่งรถล่วงหน้า

และยังสามารถลดผลกระทบจากความหน่วงของเวลาได้ด้วย MPC จะติดตามอินพุทจากคำสั่งควบคุมที่ผู้ขับขี่ส่งมา ดังภาพที่ 15

ระบบนี้มีการคาดการณ์ตำแหน่งของรถที่ควรจะเป็นเพื่อชดเชยเวลาที่ล่าช้าไปในแต่ละช่วง เมื่อระบบคาดการณ์ว่ารถอาจเกิดการชนกับสิ่งกีดขวาง ระบบจะเข้าแทรกแซงเพื่อควบคุมการขับขี่ด้วยระบบอัตโนมัติเพื่อหลบหลีกสิ่งกีดขวาง ทั้งนี้ก่อนการแทรกแซงด้วยระบบกล่าว ภาพจำลองการทำงานของระบบ ASS จะถูกส่งไปยังส่วนของสถานีควบคุม เพื่อแจ้งให้ผู้ควบคุมทราบถึงเจตนาของระบบ วิธีนี้จะช่วยลดการออกคำสั่งซ้อนกันระหว่างผู้ควบคุมและระบบอัตโนมัติ และเพื่อความปลอดภัยในการใช้งาน อำนาจการตัดสินใจสูงสุดของระบบเปลี่ยนผ่านนี้จะอยู่ภายใต้การตัดสินใจของผู้ควบคุม ณ สถานี กล่าวคือ ผู้ควบคุมสามารถยกเลิกฟังก์ชันของระบบได้ทุกเมื่อ เมื่อผู้ควบคุมเห็นว่าการแทรกแซงของระบบขัดกับสัญชาตญาณของตน

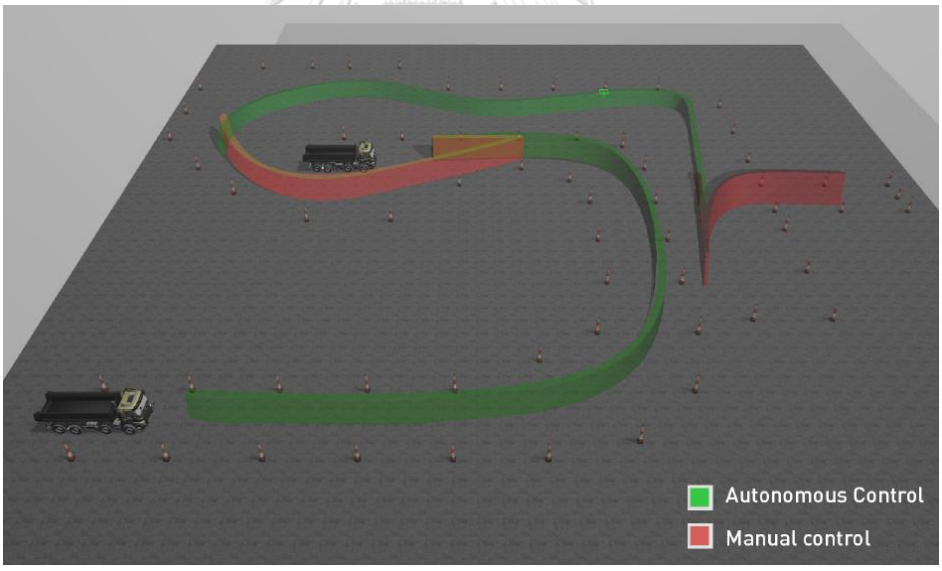


ภาพที่ 15 *การทำงานของระบบ (Active Safety System: ASS) [26]*

งานวิจัย[27] นำเสนอแนวทางการปรับเปลี่ยนโหมดระหว่างระบบอัตโนมัติและระบบควบคุมระยะไกลแบบ teleoperation โดยใช้ *Gazebo Simulation* ในการจำลองสถานการณ์ ร่วมกับโมเดลรถ Volvo FMX ซึ่งเป็นรถบรรทุกขับขี่อัตโนมัติใช้งานในเหมืองที่มีระดับความอัตโนมัติอยู่ในระหว่างระดับ 2 กับระดับ 3

ระบบเปลี่ยนผ่านถูกออกแบบมาเพื่อรองรับการใช้งานการสลับโหมดได้อย่างอิสระในทุกช่วงการใช้งาน กล่าวคือ สามารถเปลี่ยนการใช้งานระบบนำทางด้วยระบบอัตโนมัติ (autonomous navigation) ซึ่งกำลังดำเนินการทำภารกิจอยู่ในเส้นทางที่ออกแบบไว้ สลับกับการควบคุมด้วยวิธี teleoperation ได้ตลอดเวลา สิ่งนี้สามารถทำได้โดยการใช้เฟรมเวิร์ค ROS [20] ผลทดสอบการเปลี่ยนผ่านทั้ง 2 ระบบจากการจำลองโดยใช้ cone-track พบว่าระบบสามารถใช้งานได้ดี ดังภาพที่ 16

การวัดค่าความหน่วงของเวลาสามารถทำได้โดยเทียบค่า time stamp ของ massages ระหว่างโหนดบนเฟรมเวิร์ค ROS ขณะทดสอบการควบคุมด้วยการมีความหน่วงของเวลาต่ำกว่า 300 มิลลิวินาที พบว่าผู้ควบคุมสามารถชดเชยการควบคุมได้ ในกรณีที่ค่าความหน่วงของเวลามากกว่าผู้ควบคุมต้องหยุดงานเป็นช่วง ๆ ส่งผลให้ระยะเวลาการทำงานเพิ่มขึ้นตามไปด้วย การลดอัตราเฟรมของภาพต่อวินาที ส่งผลต่อประสิทธิภาพการควบคุมน้อยกว่าเมื่อเทียบกับผลกระทบของความหน่วงเวลา



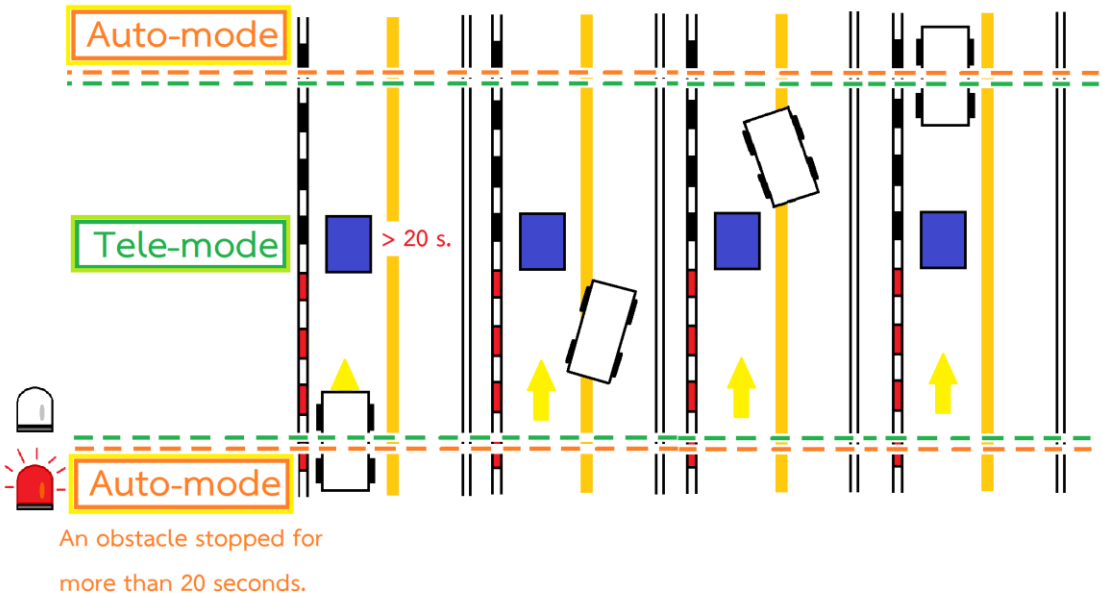
ภาพที่ 16 *ลักษณะการทดสอบระบบเปลี่ยนผ่านการควบคุมระหว่างระบบอัตโนมัติและระบบควบคุมระยะไกลด้วยการจำลอง[27]*

บทที่ 3
วิธีดำเนินการวิจัย

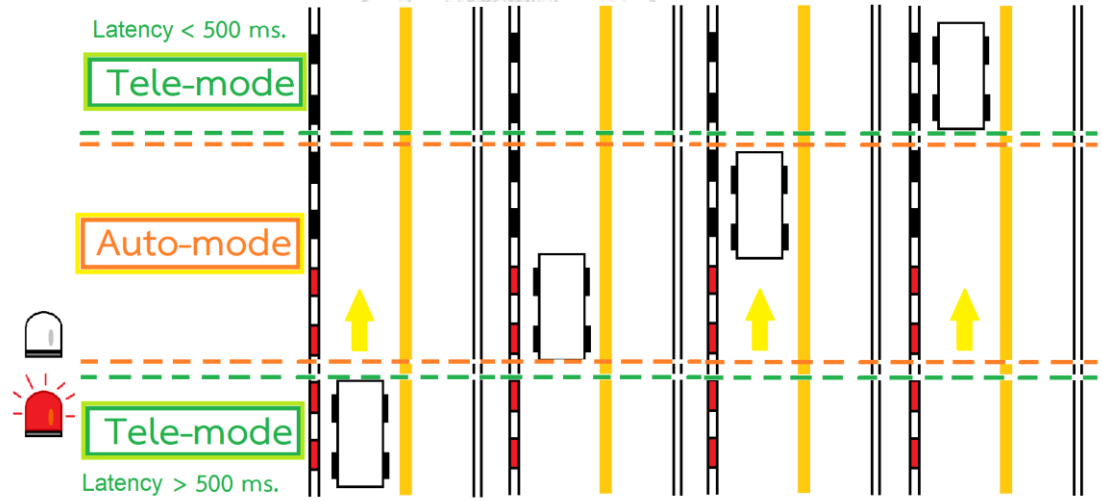
1. แนวคิดการออกแบบ

งานวิจัยนี้ศึกษาและพัฒนาระบบเปลี่ยนผ่านการควบคุมระหว่างระบบอัตโนมัติ และระบบควบคุมระยะไกลบนพื้นฐานของรถกอล์ฟไฟฟ้า ระบบเปลี่ยนผ่านนี้ถูกออกแบบให้ผู้ควบคุมสามารถเปลี่ยนโหมดควบคุมตามต้องการได้ทุกช่วงขณะระบบถูกเปิดใช้งาน หากระหว่างการดำเนินการในระบบควบคุมใดระบบหนึ่งมีเหตุผิดปกติเกิดขึ้นที่อาจส่งผลต่อประสิทธิภาพการควบคุมรถ หรือถึงขั้นที่ระบบควบคุมไม่สามารถปฏิบัติงานต่อได้เนื่องจากเกินข้อจำกัดการทำงาน เพื่อป้องกันปัญหาดังกล่าว ระบบเฝ้าระวังความผิดปกติ (Malfunction surveillance system) ถูกใช้เพื่อตรวจจับและวินิจฉัยการทำงานของทั้งระบบอัตโนมัติ และระบบควบคุมระยะไกล ออกเป็น Diagnostic types ต่าง ๆ หากเกิดความผิดปกติตามสถานการณ์ที่กำหนดไว้ ระบบนี้สามารถแจ้งเตือนสถานการณ์ฉุกเฉิน (Emergency situation) ไปยังสถานีควบคุมเพื่อแจ้งเตือนให้ผู้ควบคุมเข้าทำการเปลี่ยนโหมดเพื่อแก้ไขสถานการณ์

ระบบเฝ้าระวังความผิดปกติดังกล่าวถูกออกแบบไว้เพื่อรองรับ 2 สถานการณ์ ที่ครอบคลุมทั้งการใช้งานด้วยระบบอัตโนมัติ และระบบควบคุมระยะไกล สถานการณ์แรกรถทดสอบอยู่ในช่วงการใช้งานระบบอัตโนมัติและระบบตรวจพบว่ามีสิ่งกีดขวาง ซึ่งอาจจะเป็นรถหรือวัตถุใดๆ ในลักษณะหยุดนิ่งอยู่ด้านหน้าในช่องทางเดียวกัน รถทดสอบจะทำการเบรกจนหยุดนิ่ง หากสิ่งกีดขวางไม่มีการขยับออกให้พ้นระยะด้านหน้าและเวลากำหนด ระบบจะแจ้งเตือนไปยังสถานีควบคุมโดยการเปิดสัญญาณไฟสีแดงของ tower light ที่อยู่ ณ สถานีควบคุม เพื่อให้ผู้ควบคุมทำการเปลี่ยนโหมดมาเป็นระบบควบคุมระยะไกลเพื่อแก้ไขสถานการณ์ ดังภาพที่ 17 สถานการณ์ที่สองขณะรถทดสอบอยู่ในช่วงการใช้งานระบบควบคุมระยะไกลระบบตรวจพบว่ามีค่าความหน่วงเวลาสูงเกินกำหนด ระบบจะแจ้งเตือนไปยังสถานีควบคุมโดยการเปิดสัญญาณไฟสีแดง ณ สถานีควบคุม เพื่อให้ผู้ควบคุมทำการเปลี่ยนโหมดมาเป็นการขับขี่ด้วยระบบอัตโนมัติ ดังภาพที่ 18



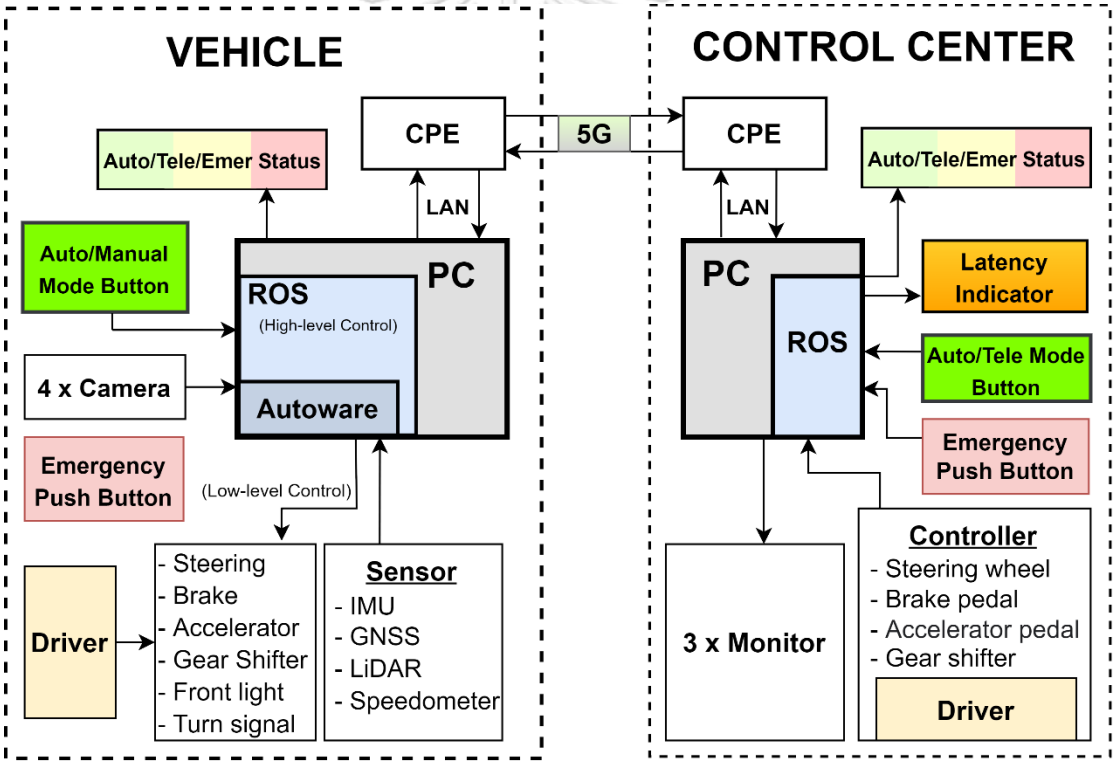
ภาพที่ 17 *ระบบแจ้งเตือนให้ผู้ควบคุมเปลี่ยนการควบคุมจากระบบขับขี่อัตโนมัติเป็นระบบควบคุมระยะไกล ในกรณีที่มีวัตถุ ขวางด้วยระยะเวลาเกินกว่าที่ระบบกำหนด*



ภาพที่ 18 *ระบบแจ้งเตือนให้ผู้ควบคุมเปลี่ยนการควบคุมจากระบบควบคุมระยะไกลเป็นระบบขับขี่อัตโนมัติในกรณีที่มีค่าความหน่วงของสัญญาณเกินกว่าค่าที่ระบบกำหนด*

ส่วนของยานยนต์อัตโนมัติต้นแบบที่สามารถควบคุมจากระยะไกลได้ และส่วนของสถานีควบคุม งานวิจัยนี้ได้มีการพัฒนาตั้งแต่เริ่มแรก ดังภาพที่ 19 กล้อง เซนเซอร์ คอมพิวเตอร์ขนาดเล็ก ชุดซอฟต์แวร์ควบคุมในระดับสูง บอร์ดไมโครคอนโทรลเลอร์ ปุ่มหยุดการทำงานฉุกเฉิน ชุดตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าถูกติดตั้งบนรถกอล์ฟเพื่อทำหน้าที่ควบคุมระบบต่าง ๆ แทนมนุษย์ ส่วนของสถานีควบคุมประกอบไปด้วย หน้าจอมอนิเตอร์ คอมพิวเตอร์ ปุ่มสลับโหมดระบบควบคุม หลอดไฟ LED แสดงสถานะความหน่วงเวลาของสัญญาณ และอุปกรณ์สั่งการเคลื่อนที่ของรถ ถูกติดตั้ง ณ สถานีควบคุมเพื่อใช้ในการควบคุมรถจากระยะไกลผ่านการมองภาพจากมอนิเตอร์

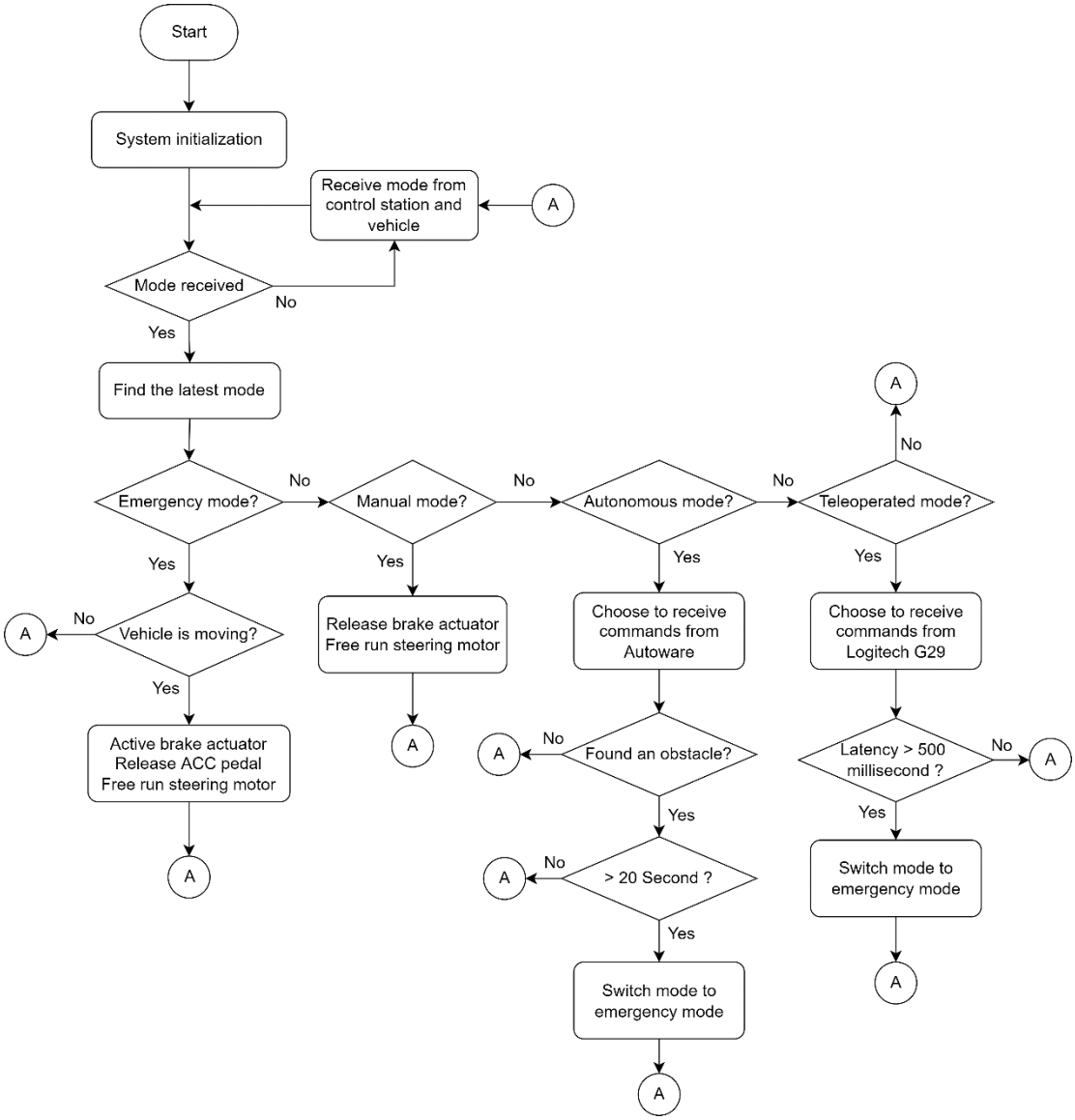
การรับ-ส่งข้อมูลระหว่างทั้งสองส่วนดังกล่าวจะใช้ชุดโมดูลควบคุมระยะไกลผ่านเครือข่าย 5G ที่ถูกติดตั้งทั้งรถต้นแบบและสถานีควบคุม



ภาพที่ 19 *แผนภาพแสดงองค์ประกอบของยานยนต์ระบบอัตโนมัติ และระบบควบคุมระยะไกลผ่านเครือข่าย 5G*

ภาพรวมของผังงาน (Flowchart) การทำงานของระบบเปลี่ยนผ่านการควบคุมทั้งสองระบบเป็นไปดังภาพที่ 20 ผู้ควบคุมสามารถเปลี่ยนโหมดควบคุมได้ทุกช่วงการใช้งานจากการกดปุ่มเปลี่ยนโหมดที่อยู่ ณ สถานีควบคุม ดังภาพที่ 19 แต่ถ้าหากระบบเฝ้าระวังความผิดปกติตรวจพบความ

ผิดปกติที่เกิดขึ้นขณะใช้งาน ซึ่งงานวิจัยนี้ใช้สถานการณ์ที่กำหนดไว้ล่วงหน้าดังภาพที่ XX และ XX ระบบจะสลับจากโหมดควบคุมล่าสุดเปลี่ยนไปเป็นโหมดฉุกเฉินทันที และสั่งให้รถทดสอบทำการเบรกฉับพลันจนกว่ารถจะหยุดนิ่ง สถานีควบคุมจะรับรู้ถึงสถานการณ์ฉุกเฉินนี้และทำการแจ้งเตือนให้ผู้ควบคุมทำการสลับโหมดควบคุมเพื่อแก้ไขสถานการณ์ดังกล่าว



ภาพที่ 20 *ผังงานการทำงานของระบบเปลี่ยนผ่านการควบคุมระหว่างระบบอัตโนมัติ และระบบควบคุมจากระยะไกล*

2. ยานยนต์ที่ใช้ในงานวิจัย

ยานยนต์ต้นแบบในงานวิจัยนี้พัฒนาจากรถกอล์ฟอเนกประสงค์พลังงานไฟฟ้า ยี่ห้อ EVT รุ่น Plus D 6s ขนาด 6 ที่นั่ง พิกัดแรงดันไฟฟ้าของระบบ 48V DC กำลังมอเตอร์ 5 กิโลวัตต์ ความเร็วสูงสุด 35 กิโลเมตร/ชั่วโมง ระยะทางวิ่งต่อการชาร์จ 70 กิโลเมตร ก่อนการดัดแปลงและเพิ่มเติมชุดอุปกรณ์ใดๆ ดังภาพที่ 21

ภาพที่ 21 *รถอเนกประสงค์พลังงานไฟฟ้า 6 ที่นั่ง รุ่น EVT Plus D 6s*

3. กลไกควบคุมระบบบังคับเลี้ยวสั่งงานด้วยสัญญาณทางไฟฟ้า (Steer-By-Wire)

เนื่องจากระบบบังคับเลี้ยวของรถกอล์ฟไฟฟ้าในงานวิจัยนี้ไม่ได้มีลักษณะเป็นพวงมาลัยไฟฟ้าเช่นเดียวกับ พวงมาลัยไฟฟ้า EPS (Electric Power Steering) ดังนั้นการสั่งการด้วยสัญญาณทางไฟฟ้าสามารถทำได้โดยการติดตั้งตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้ากับระบบบังคับเลี้ยวเดิมของรถ เพื่อให้สามารถสั่งงานระบบบังคับเลี้ยวได้แม้ไม่มีผู้ขับขี่คอยควบคุมอยู่บนรถ ชุดตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าประกอบไปด้วย มอเตอร์ โมดูลขับมอเตอร์ ไมโครคอนโทรลเลอร์ และอุปกรณ์ป้อนกลับตำแหน่งพวงมาลัย การออกแบบ-ติดตั้งชุดอุปกรณ์เหล่านี้จะกล่าวในหัวข้อถัดไป

3.1 การเลือกขนาดมอเตอร์ไฟฟ้ากระแสตรง

การเลือกมอเตอร์จะพิจารณาจากแรงบิดสูงสุด (Maximum Torque) ที่ต้องใช้ในการหมุนพวงมาลัยขณะหยุดนิ่ง (Static Load) งานวิจัยนี้ใช้วิธีการใช้ดึงพวงมาลัยด้วยตาชั่งวัดแรงจนล้อของรถเริ่มหมุนดังภาพที่ 22 ผลการทดลองหาน้ำหนักเฉลี่ยที่ใช้ในการหมุนพวงมาลัยในทิศทางทวนเข็ม 3 ครั้ง และตามเข็มนาฬิกา 3 ครั้ง ได้ผลดังตารางที่ 1 ซึ่งมีค่าเฉลี่ยอยู่ที่ 5.28 กิโลกรัม ระยะตั้งฉากจากแนวแรงถึงจุดหมุน คือ 17 เซนติเมตร ดังภาพที่ 23 แรงบิดสูงสุดที่ใช้ในการหมุนพวงมาลัยสามารถคำนวณได้ตามสมการที่ (1)

ภาพที่ 22 *ค่าที่วัดได้จากการทดลองการหมุนพวงมาลัยโดยใช้เครื่องชั่งวัดแรง*

ตารางที่ *1 ค่าเฉลี่ยน้ำหนักทั้งหมดจากการทดลองหมุนพวงมาลัยทั้งทวนเข็มและตามเข็มนาฬิกา*

ลักษณะการหมุนพวงมาลัย	น้ำหนักที่วัดได้จากการหมุนของพวงมาลัย (กิโลกรัม)			**เฉลี่ย**
	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	
หมุนทวนเข็มนาฬิกา	5.72	5.54	5.05	5.28
หมุนตามเข็มนาฬิกา	4.82	4.56	6.00	

ภาพที่ 23 *ระยะตั้งฉากจากแนวแรงถึงจุดหมุน*

ค่าแรงบิดสูงสุด

$$\tau_{\max} = FR \qquad (1)$$

โดยที่ τ คือ แรงบิดสูงสุด (Nm)

F คือ แรงดึงพวงมาลัยเฉลี่ยที่ทำให้ล้อสามารถหมุนได้ (N)

R คือ ระยะตั้งฉากจากแนวแรงถึงจุดหมุน (m)

จะได้ว่า

$$\tau = (5.28 \times 9.81) \times (17 \times 10^{-2}) = 8.8\,Nm$$

ทำการออกแบบมอเตอร์โดยใช้ค่าความปลอดภัย (Safety factor) เท่ากับ 2 ดังนั้นมอเตอร์ที่ใช้ควรมีแรงบิด 17.6 Nm จึงเลือกมอเตอร์ใช้มอเตอร์กระแสตรง (DC motor) ขนาด 24 โวลต์ กำลังมอเตอร์ 350 วัตต์ ความเร็วรอบของมอเตอร์ 450 รอบต่อนาที ดังภาพที่ 24 แต่ข้อมูลทางเทคนิคดังกล่าวไม่ระบุขนาดแรงบิด จึงต้องหาค่าแรงสูงสุดก่อน

ค่าแรงบิดสูงสุด

$$\tau_{\max} = \frac{P_r}{\omega} \qquad (2)$$

โดยที่ P_r คือ พิกัดกำลังของมอเตอร์ (kW)

ω คือ ความเร็วเชิงมุม (rad/s)

จะได้

$$\tau_{\max} = \frac{350W}{\frac{2\pi \times 450}{60}} = 7.43\ Nm$$

จากการคำนวณพบว่าแรงบิดสูงสุดของมอเตอร์มีค่าเท่ากับ 7.43 นิวตันเมตร นั้นยังไม่เพียงพอต่อแรงบิดที่ต้องการที่กระทำต่อพวงมาลัยดังสมการที่ (1) ซึ่งมีค่า 17.6 นิวตันเมตรดังนั้นจึงต้องเพิ่มอัตราทดเฟืองสายพานที่สามารถทดแรงบิดเพิ่มขึ้นได้ 3 เท่า ค่าแรงบิดที่ได้หลังจากการทดคือ 22.29 Nm ขณะที่ความเร็วรอบของการหมุนพวงมาลัยลดลงเหลือ 150 รอบต่อนาที (2.5 รอบต่อวินาที) ทั้งแรงบิดและความเร็วรอบหลังจากการทดที่เพียงพอต่อการใช้งานในระบบบังคับเลี้ยว

ภาพที่ 24 มอเตอร์กระแสตรงในระบบบังคับเลี้ยวสั่งงานด้วยสัญญาณทางไฟฟ้า

3.2 อุปกรณ์วัดการหมุนของพวงมาลัย (Feedback Device)

งานวิจัยนี้เลือกใช้ตัวต้านทานปรับค่าได้แบบแม่นยำ (Precision potentiometer) ขนาดความต้านทาน 10 กิโลโอห์ม ($k\Omega$) จำกัดรอบการหมุนอยู่ที่ 10 รอบ ดังภาพที่ 25 ซึ่งใช้เป็นอุปกรณ์ในการวัดตำแหน่งการหมุนจริงของแกนพวงมาลัยผ่านชุดฟันเฟืองที่ถูกออกแบบและปรับให้เหมาะสมกับรอบของการหมุนพวงมาลัยแบบ lock-to-lock ให้สัมพันธ์กับรอบของตัวต้านทาน เพื่อป้องกันความเสียหายของอุปกรณ์จากการหมุนเกินรอบที่จำกัด ลักษณะการติดตั้งชุดอุปกรณ์ป้อนตำแหน่งกลับนี้จะติดตั้งอยู่บนแกนพวงมาลัย และวัดรอบการหมุนของพวงมาลัยผ่านฟันเฟือง ดังภาพที่ 26

ภาพที่ 25 *ตัวต้านทานปรับค่าได้แบบแม่นยำ*

ภาพที่ 26 *ลักษณะการติดตั้งอุปกรณ์วัดตำแหน่งการหมุนของพวงมาลัย*

การติดตั้งชุดตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าบนแกนพวงมาลัยของระบบบังคับเลี้ยวของงานวิจัยนี้เป็นไปตามภาพที่ 27

ภาพที่ 27 *ลักษณะชุดตัวกระตุ้นของระบบบังคับเลี้ยวสั่งงานด้วยสัญญาณทางไฟฟ้า*

3.3 ไมโครคอนโทรลเลอร์ และโมดูลขับมอเตอร์กระแสตรง (Microcontroller and DC Motor Driver)

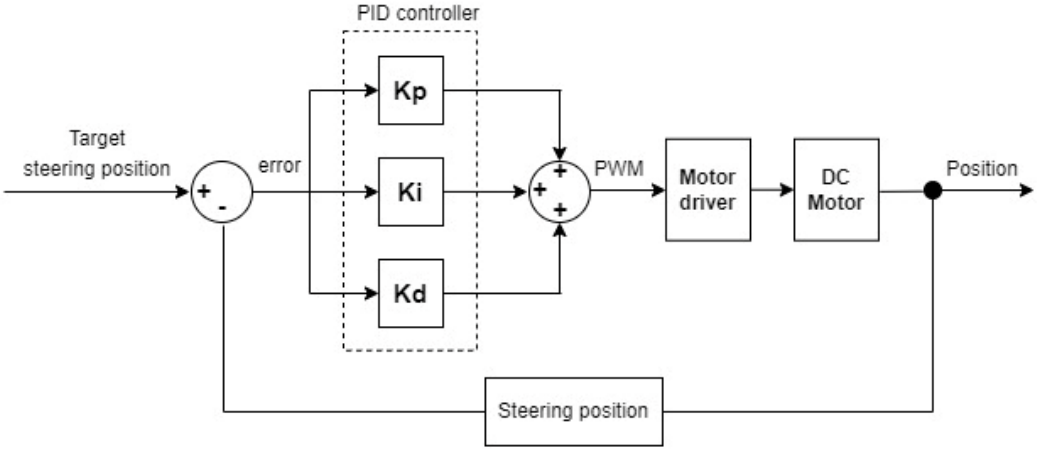
การควบคุมการหมุนของมอเตอร์ให้ได้ตำแหน่ง ความเร็ว หรือตำแหน่งการหมุน จำเป็นต้องมีอุปกรณ์ในการส่งสัญญาณควบคุมไปยังมอเตอร์ เช่น บอร์ดไมโครคอนโทรลเลอร์ Arduino ดังภาพที่ 28 ซึ่งเป็นไมโครคอนโทรลเลอร์ตระกูล AVR (Automatic Voltage Regulator) เป็นที่นิยมและใช้ทั่วไป แต่ ไมโครคอนโทรลเลอร์นี้ไม่สามารถที่จะขับมอเตอร์ที่มีแรงดันสูง กระแสสูง หรือการกลับทิศการหมุนของมอเตอร์กระแสตรงได้ จึงจำเป็นต้องมีโมดูลเสริมในการควบคุมมอเตอร์ต่อจากไมโครคอนโทรลเลอร์ งานวิจัยนี้เลือกใช้ชุดโมดูลขับมอเตอร์กระแสตรง ชนิด H-bridge ดังภาพที่ 29 โมดูลนี้สามารถเชื่อมต่อกับบอร์ดไมโครคอนโทรลเลอร์ที่มีระดับไฟ 3-5 โวลต์ กระแส 0.8 มิลลิแอมป์ และสั่งความเร็วการหมุนของมอเตอร์ด้วย PWM (Pulse Width Modulation) ได้โดยตรง เอาต์พุตที่ได้จากโมดูลเสริมนี้สามารถขับมอเตอร์ที่มีขนาดแรงดันตั้งแต่ 12-36 โวลต์ และให้กระแสสูงสุดที่ 200 แอมป์ ซึ่งเพียงพอในการขับมอเตอร์กระแสตรงที่ออกแบบไว้

ภาพที่ 28 *บอร์ดไมโครคอนโทรลเลอร์ Arduino UNO R3*

ภาพที่ 29 *ชุดโมดูลขับมอเตอร์กระแสตรง ชนิด H-bridge*

3.4 วิธีการควบคุม

การควบคุมระบบบังคับเลี้ยวให้เป็นไปตามต้องการ มีความรวดเร็ว และแม่นยำ ระบบจำเป็นต้องมีลักษณะของระบบควบคุมแบบวงปิด (Closed-loop Control หรือ Feedback Control) เพื่อตรวจสอบค่าสัญญาณเอาต์พุตที่สั่งออกไป และปรับการควบคุมให้มีประสิทธิภาพมากขึ้น งานวิจัยนี้เลือกใช้ลักษณะการควบคุมแบบ PID Controller (Proportional Integral Derivative Controller) ในการลดค่าความผิดพลาดของระบบ และปรับการควบคุมให้เหมาะสมมากขึ้น ดังภาพที่ 30



ภาพที่ 30 *ระบบควบคุมแบบป้อนกลับด้วยวิธี PID Controller*

4. กลไกควบคุมอัตราเร็วสั่งงานด้วยสัญญาณทางไฟฟ้า (Drive-By-Wire System)

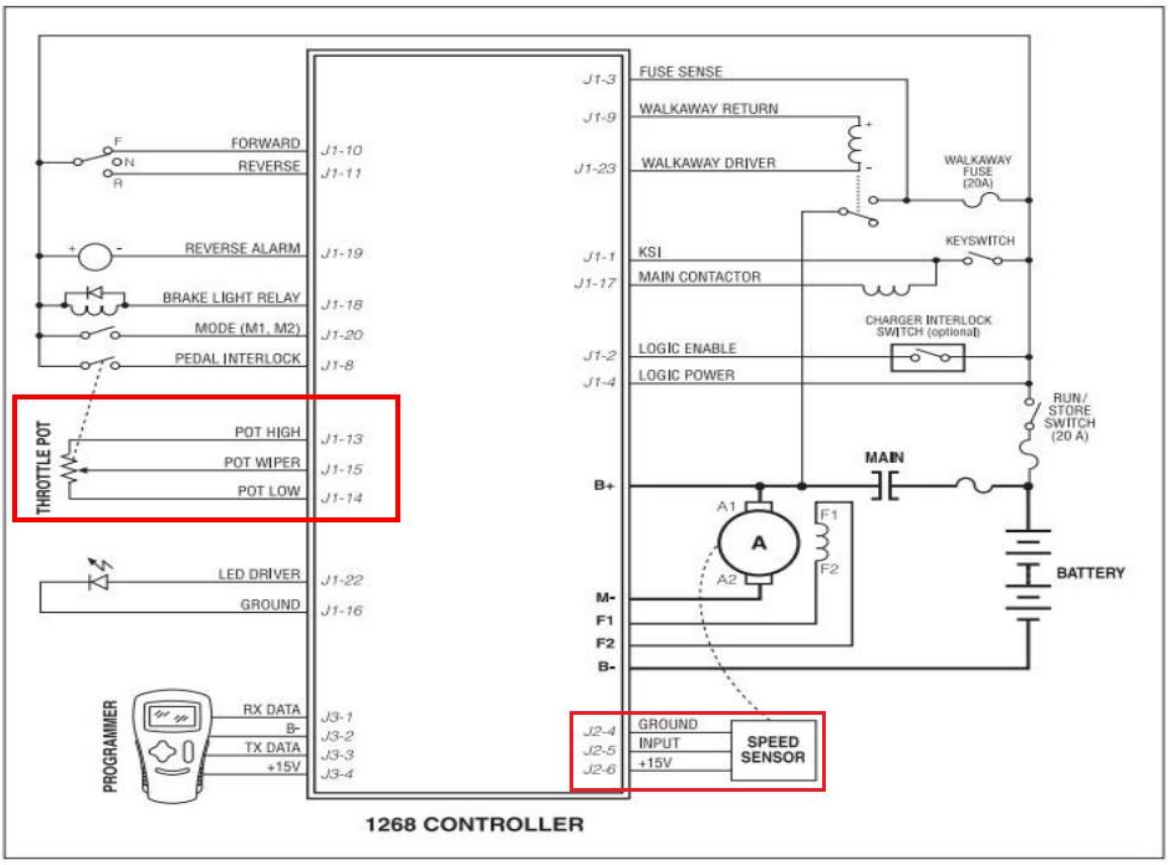
4.1 การวัดอัตราเร็ว (Speed Sensor)

เนื่องจากรถกอล์ฟไฟฟ้าไม่มีการส่งค่าอัตราเร็วกลับออกมาเหมือนกับรถยนต์อื่น ๆ งานวิจัยนี้จึงได้ศึกษาวิธีการหาอัตราเร็วของยานยนต์ไฟฟ้าหลายวิธี และได้ติดตั้งวงจรอ่านความเร็วทั้งหมด 2 แบบ บนยานยนต์ต้นแบบ ดังนี้

1. การวัดอัตราเร็วจากเซ็นเซอร์ของกล่องควบคุมอิเล็กทรอนิกส์

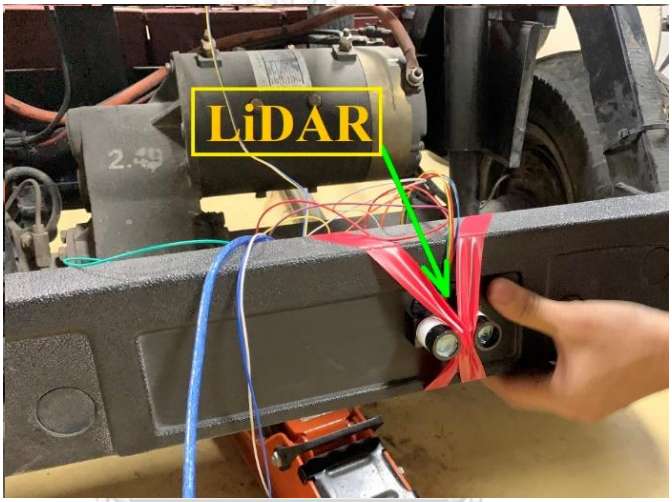
อัตราเร็วของยานยนต์ไฟฟ้าต้นแบบนี้สามารถใช้การเทียบค่าความถี่ของมอเตอร์จาก Speed sensor ภายในกล่องควบคุมอิเล็กทรอนิกส์ CURTIS 1268 Controller ดังภาพที่ 31 กล่องควบคุมนี้ใช้เพื่อควบคุมการทำงานของรถกอล์ฟไฟฟ้า เช่น ระบบเร่ง เกียร์ ไฟเลี้ยว วงจรค้างสถานะ (Interlock) รวมถึง เซ็นเซอร์อ่านค่าความถี่ดังกล่าว ซึ่งเป็นส่วนหนึ่งของวงจรที่อยู่ภายในกล่องควบคุม ดังภาพที่ 32

ภาพที่ 31 *CURTIS 1268 Controller (https://www.nocoev.com)*

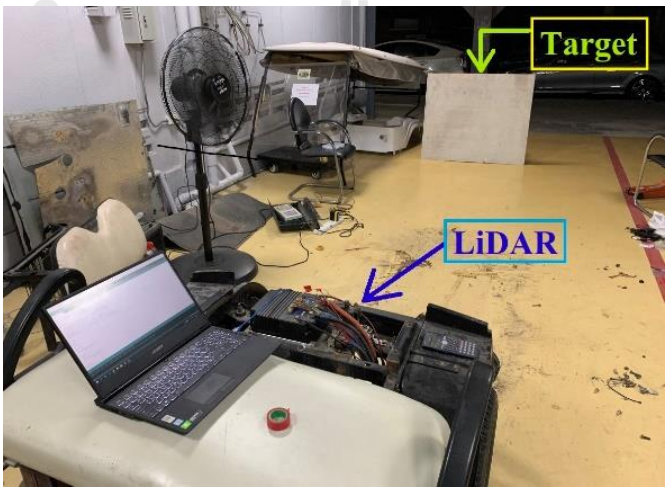


ภาพที่ 32 *แผนภาพวงจรอิเล็กทรอนิกส์ภายใน CURTIS 1268 Controller*

ค่าที่ได้จากเซ็นแซอร์นี้จะอยู่ในรูปของความถี่ ซึ่งความถี่จะแปรผันตรงกับอัตราเร็วของรถกอล์ฟไฟฟ้า กล่าวคือยิ่งความถี่สูงอัตราเร็วของรถก็จะยิ่งสูงขึ้นด้วยเช่นกัน ดังนั้นการอ่านค่าอัตราเร็วจากเซ็นเซอร์นี้ อันดับแรกจึงต้องหาระยะทางที่รถเคลื่อนที่ได้จริง ในช่วงคาบของความถี่ที่อ่านได้ขณะนั้น งานวิจัยนี้จึงทำการติดตั้งเซ็นเซอร์ตรวจจับและวัดระยะด้วยแสงประเภทสองมิติ (2D - LiDAR) ซึ่งเป็นอุปกรณ์วัดระยะทางประเภทหนึ่ง ในการสอบเทียบระหว่างระยะทางที่ยานยนต์ต้นแบบนี้เคลื่อนเข้าหาวัตถุหนึ่งเทียบกับคาบของความถี่ที่อ่านได้จาก speed sensor ดังภาพที่ 33 และ 34

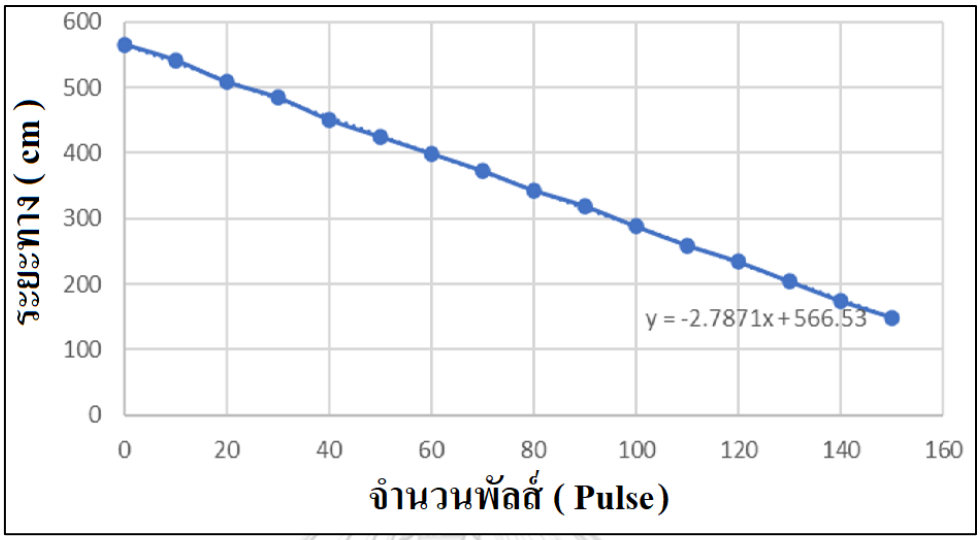


ภาพที่ 33 *เซ็นเซอร์ตรวจจับและวัดระยะทางด้วยแสงประเภทสองมิติ*



ภาพที่ 34 *การทดสอบขณะเคลื่อนที่เข้าหากำแพงเพื่ออ่านค่าระยะทางจริงเทียบกับคาบความถี่*

ผลการวัดระยะทางที่รถเคลื่อนที่เข้าหาวัตถุ เทียบกับจำนวนพัลส์ที่อ่านได้ ดังภาพที่ 35 การทดสอบนี้ทำให้ทราบระยะทางที่รถเคลื่อนที่ไปใน 1 คาบ



ภาพที่ 35 *ความสัมพันธ์ระหว่างระยะทางกับจำนวนพัลส์*

จากกราฟความสัมพันธ์พบว่าใน 1 คาบที่อ่านได้ รถเคลื่อนที่ไป 2.787 เซนติเมตร เพื่อให้ได้ค่าที่แม่นยำมากขึ้น จึงได้ทำการทดลองซ้ำทั้งหมด 4 รอบ ดังตารางที่ 2

ตารางที่ *2 ระยะทางที่รถเคลื่อนที่ได้ต่อ 1 คาบของความถี่ที่อ่านได้*

จำนวนครั้งการทดลอง	ระยะทางที่รถเคลื่อนที่ต่อคาบ (เซ็นติเมตร)
1	2.79
2	2.54
3	2.71
4	2.78

จากการทดสอบ 4 ครั้ง ระยะทางที่รถเคลื่อนที่เฉลี่ยต่อ 1 คาบของความถี่ คือ 2.705 เซนติเมตร/คาบ จึงนำไปสู่สมการอัตราเร็วของยานพาหนะต้นแบบ ดังสมการที่ 3

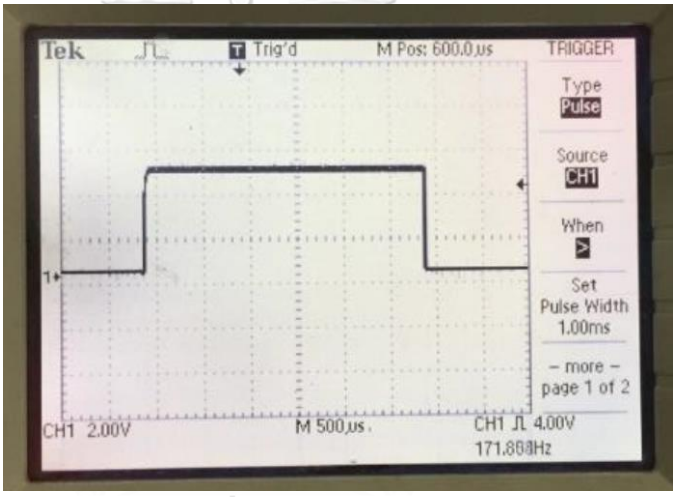
อัตราเร็วของยานพาหนะ

$$v = 0.02705\, f \tag{3}$$

โดยที่ v คือ อัตราเร็วของยานพาหนะต้นแบบ (เมตร/วินาที)

f คือ ความถี่ที่วัดได้จาก Speed sensor (เฮิรตซ์)

เมื่อทำการทดลองหาค่าความถี่สูงสุด ณ อัตราเร็วสูงสุดของยานยนต์ต้นแบบขณะล้อลอย พบว่าความถี่สูงสุดคือ 171.8 เฮิรตซ์ ดังภาพที่ 36



ภาพที่ 36 *ความถี่สูงสุดที่วัดได้จาก Speed sensor ขณะเร่งอัตราเร็วสูงสุด*

เมื่อทำการหาอัตราเร็วสูงสุดจากความถี่สูงสุดจากสมการที่ (3) พบว่าอัตราเร็วสูงสุดขณะทดสอบ อัตราเร็วจะอยู่ที่ 4.65 เมตร/วินาที หรือ 16.74 กิโลเมตร/ชั่วโมง และเมื่อเทียบกับอัตราเร็วสูงสุดจากข้อมูลจำเพาะทางเทคนิคของรถกอล์ฟ โดยระบุไว้ว่าอัตราเร็วสูงสุดอยู่ที่ 35 กิโลเมตร/ชั่วโมง ทั้งสองค่านี้ยังคงต่างกันมาก ซึ่งอาจคลาดเคลื่อนจากสัญญาณรบกวนจากสายไฟบริเวณโดยรอบ หรือความไม่เสถียรของกำลังไฟจากการเสื่อมสถาพของแบตเตอรี่

2. การวัดค่าอัตราเร็วของรถจากเซ็นเซอร์วัดอัตราการหมุนของล้อ

อีกวิธีหนึ่งในการหาอัตราเร็วของรถ คือการติดตั้งเซ็นเซอร์วัดอัตราการหมุน หรือเอ็นโค้ดเดอร์ที่ล้อรถ จึงเป็นอีกทางเลือกในการหาอัตราเร็วของรถ แต่การติดตั้งเซ็นเซอร์นี้ตรง ๆ ที่ล้อนั้นเป็น

งานที่ค่อนข้างยาก เนื่องจากเพลาของเอ็นโค้ดเดอร์ต้องยึดกับแกนของล้อรถและหมุนไปพร้อมกัน ในขณะที่ส่วน housing ของเอ็นโค้ดเดอรจะต้องอยู่นิ่งและไม่หมุนตามเพลาไปด้วย จึงต้องหาจุดยึดระหว่าง housing กับตัวรถที่เหมาะสม จากการศึกษาเซ็นเซอร์ที่เหมาะสมกับการแก้ปัญหานี้และมีจำหน่ายทั่วไป พบว่ามีราคาสูง งานวิจัยนี้จึงได้ทำการออกแบบเซ็นเซอร์วัดอัตราการหมุนของล้อที่มีความยืดหยุ่นในการติดตั้ง และสะดวกต่อการนำไปใช้งาน ดังภาพที่ 37 และ 38 โดยการประยุกต์ใช้สายอ่อนแกน 6 มิลลิเมตร เชื่อมต่อระหว่างล้อและเพลาของเอ็นโค้ดเดอร์

ภาพที่ 37 *เซ็นเซอร์วัดอัตราเร็วจากการหมุนของล้อแบบใช้สายอ่อน*

ภาพที่ 38 *ฝาครอบล้อรถที่ออกแบบใหม่โดยเฉพาะ(รูปขวา) สำหรับจับกับแกนสายอ่อน*

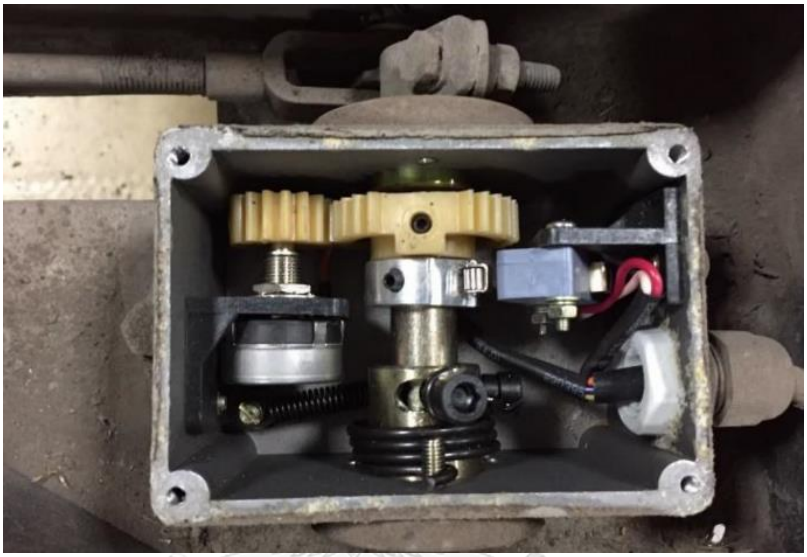
เอ็นโค้ดเดอร์ที่เลือกใช้เป็นประเภท incremental rotary encoder ความละเอียด 60 พัลส์/รอบ แต่วิธีนี้เหมาะกับการทดลองที่เป็นทางตรง และใช้ความเร็วต่ำในการทดลอง เนื่องจากความคลาดเคลื่อนจากการติดเซ็นเซอร์วัดอัตราการหมุนเพียงหนึ่งล้อ เช่น ขณะที่รถเข้าสู่ช่วงโค้งความเร็วของล้อแต่ละล้อจะมีค่าไม่เท่ากัน หรือรัศมีของล้อนั้นมีค่าไม่คงที่ในแต่ละช่วง ภาพรวมลักษณะการติดตั้งชุดเซ็นเซอร์เป็นไปดังภาพที่ 39

ภาพที่ 39 *ลักษณะการติดตั้งเซนเซอร์วัดอัตราเร็วของรถจากการวัดอัตราการหมุนของล้อ*

จากการทดสอบหาอัตราเร็วสูงสุดที่ยานยนต์สามารถทำได้ด้วยวิธีติดเซ็นเซอร์ที่ล้อพบว่าค่าอัตราเร็วสูงสุดมีค่า 16.67 กิโลเมตร/ชั่วโมง ซึ่งมีค่าใกล้เคียงกับการหาอัตราเร็วด้วยวิธีแรก

4.2. การสั่งอัตราเร็วด้วยสัญญาณทางไฟฟ้า

การสั่งอัตราเร็วในระบบเดิมของรถกอล์ฟไฟฟ้าจะใช้หลักการเปลี่ยนค่าแรงดันไฟฟ้าจากตัวต้านทานปรับค่าได้ที่อยู่บริเวณคันเร่งดังภาพที่ 40 สัญญาณทางไฟฟ้าจากจุดนี้จะส่งไปยังชุดควบคุมอิเล็กทรอนิกส์ ด้วยเหตุนี้การสั่งอัตราเร็วจึงสามารถทำได้ด้วยวิธีการเลียนแบบสัญญาณไฟฟ้าของ Throttle pot ที่อยู่ในชุดควบคุมอิเล็กทรอนิกส์ดังภาพที่ 31 และ 32



ภาพที่ 40 ชุดปรับค่าแรงดันทางไฟฟ้าบริเวณใต้คันเร่ง

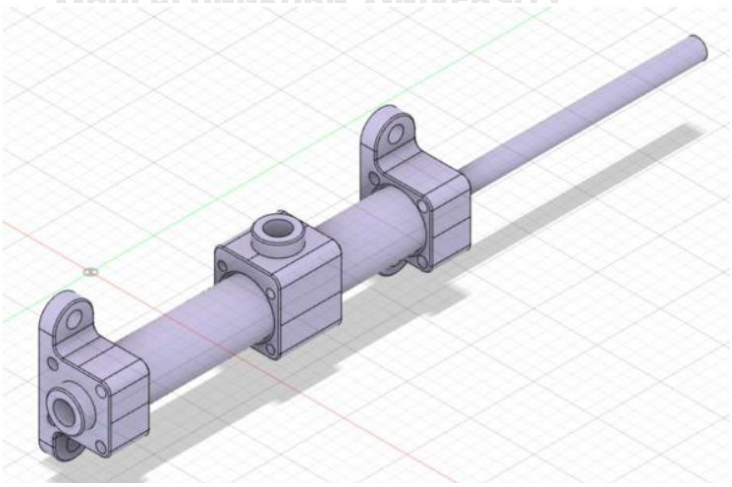
ผลการวัดค่าแรงดันทางไฟฟ้าของ Throttle pot ที่เปลี่ยนไประหว่างการสั่งอัตราเร็ว พบว่ามีค่าแรงดันอยู่ระหว่าง 0 - 4.4 โวลต์ โดยที่แรงดัน 0 โวลต์ อัตราเร็วของรถจะเป็น 0 กิโลเมตร/ชั่วโมง และอัตราเร็วจะเพิ่มขึ้นเมื่อแรงดันเพิ่มขึ้น ดังนั้นค่าของแรงดันนี้จึงสัมพันธ์กับอัตราเร็วของรถกอล์ฟ งานวิจัยนี้จึงทำการออกแบบวงจรปรับอัตราเร็วโดยใช้วงจรแปลงสัญญาณดิจิตอลเป็นอนาล็อก (Digital to Analog converter - DAC) รุ่น MCP4922 ขนาด 12 bit ซึ่งใช้อินเตอร์เฟสแบบ Serial Peripheral Interface (SPI) ที่มีความชัดเจนและเที่ยงตรงสูง ในการปรับแรงดันไฟฟ้า Throttle pot ของชุดควบคุมอิเล็กทรอนิกส์ ดังภาพที่ 41

ภาพที่ 41 *การสั่งอัตราเร็วยานยนต์ต้นแบบโดยใช้วงจรแปลงสัญญาณดิจิตอลเป็นอนาล็อก เพื่อปรับแรงดันไฟฟ้าของ Throttle pot*

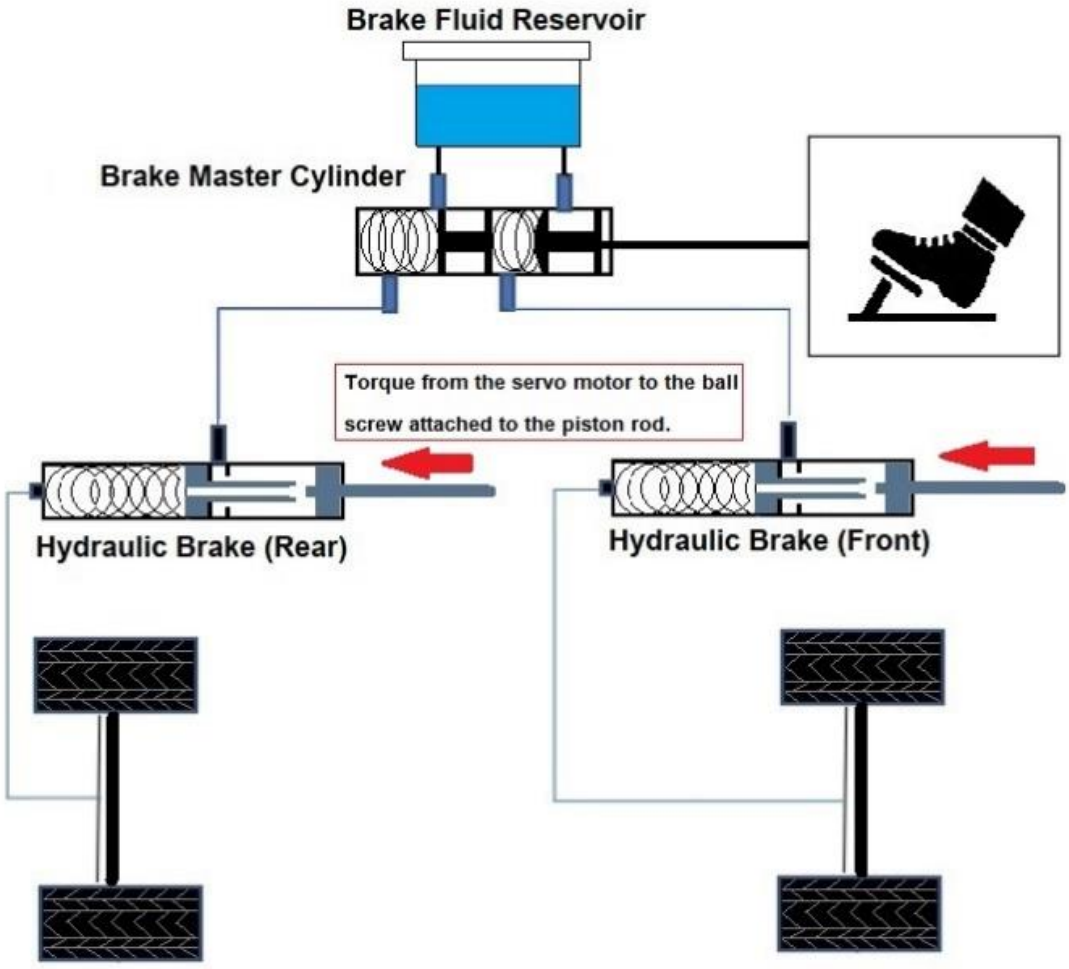
จากการทดลองสั่งความเร็วโดยใช้ DAC 12 bit ซึ่งมีค่าการใช้งาน 0 – 4095 ($2^{12}-1=4095$) พบว่าช่วงการใช้งานอยู่ที่ 0 – 3600 ถ้าสั่งเกินกว่าค่านี้รถจะไม่ทำงาน

5. กลไกควบคุมระบบเบรก (Braking System Control)

โดยปกติการทำงานของระบบเบรกในรถยนต์ทั่วไปจะใช้หลักการส่งผ่านแรงแบบไฮดรอลิกจากการเหยียบแป้นเบรกและขยายแรงเบรกด้วยแม่ปั๊มเบรก (Master cylinder) จากนั้นแรงจะกระจายไปยังชุดเบรกแต่ละล้อเพื่อทำการเบรกด้วยแรงเสียดทาน งานวิจัยนี้ได้อ้างอิงข้อมูลการออกแบบชุดเบรกสั่งงานด้วยสัญญาณทางไฟฟ้าของงานวิจัย [9] โดยงานวิจัยนี้ไม่ได้ใช้วิธีการควบคุมแรงดันน้ำมันเบรกในระบบเบรกโดยตรง แต่ใช้มอเตอร์ในการสร้างแรงบิด ซึ่งแรงบิดนี้จะส่งผ่านไปยัง linear screw เพื่อเปลี่ยนจากการเคลื่อนที่แบบหมุนเป็นการเคลื่อนที่เชิงเส้นในการดันชุดลูกปั๊มที่ออกแบบไว้ ดังภาพที่ 41 ด้วยวิธีนี้แรงบิดของมอเตอร์จะแปรผันตรงกับแรงดันของน้ำมันเบรก หลักการทำงานของชุดเบรกไฟฟ้า เป็นไปดังภาพที่ 43

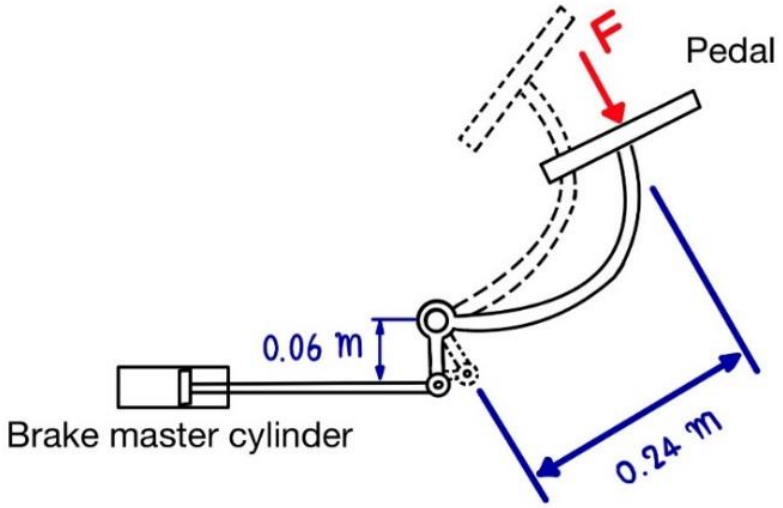


ภาพที่ 42 *กระบอกไฮดรอลิกซึ่งทำหน้าที่เป็นชุดลูกปั๊มเสริม [9]*



ภาพที่ 43 *หลักการทำงานของชุดเบรกไฟฟ้าของงานวิจัย [9]*

การออกแบบชุดเบรกไฟฟ้าของยานยนต์ต้นแบบในงานวิจัยนี้ อันดับแรกต้องทราบแรงสุทธิที่กระทำต่อแม่ปั๊มเบรกของรถ ซึ่งเป็นแรงที่ผู้ขับขี่ใช้เท้ากระทำต่อแป้นเบรกจนสุด การวัดแรงเบรกสามารถทำได้โดยการใช้เครื่องชั่งสปริงในการดึงแป้นเบรกจนสุด ค่าที่ได้จากการทดสอบ คือ 22 กิโลกรัม ส่วนระยะของแนวแรงที่กระทำที่จุดหมุนของก้านเบรกเป็นไปดังภาพที่ 44



ภาพที่ 44 *ระยะก้านเบรกของยานพาหนะที่กระทำต่อจุดหมุน*

แรงสุทธิที่กระทำต่อแม่ปั๊มเบรกหาได้จากสมการที่ (4)

$$\sum M = 0 \tag{4}$$

$$\left(F_{Brake\,master\,cylinder} \times D_{Piston\ pin\ to\ Fulcrum}\right) = \left(F_{Pedal} \times D_{Pedal\ to\ Fulcrum}\right)$$

$$\left(F_{Brake\,master\,cylinder} \times 0.06\ m\right) = \left(\left(22\,kg \times 9.81\,\frac{m}{s^2}\right) \times 0.24\ m\right)$$

$$F_{Brake\,master\,cylinder} = 823.28\ N$$

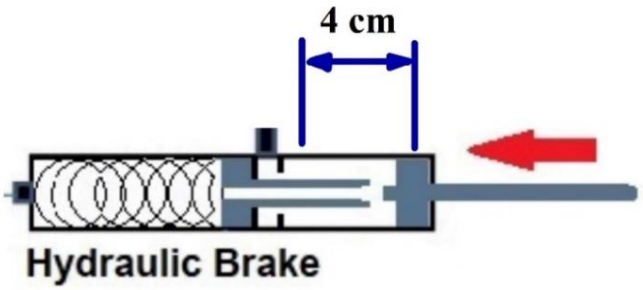
โดยที่ $F_{Brake\,master\,cylinder}$ คือ อัตราเร็วของยานพาหนะต้นแบบ (เมตร/วินาที)

F_{Pedal} คือ ความถี่ที่วัดได้จาก Speed sensor (เฮิรตซ์)

$D_{Piston\ pin\ to\ Fulcrum}$ คือ ระยะจากก้านแม่ปั๊มถึงจุดหมุน

$D_{Pedal\ to\ Fulcrum}$ คือ ระยะจากแป้นคันเร่งถึงจุดหมุน

จากสมการที่ (4) ผลจากการคำนวณพบว่าแรงสุทธิที่ก้านลูกสูบกระทำต่อแม่ปั๊มเบรกเดิมของรถมีค่า 823.28 นิวตัน ซึ่งในการออกแบบชุดเบรกไฟฟ้าจะต้องได้แรงที่ขนาดไม่ต่ำกว่าค่าที่คำนวณไว้ ดังนั้นจึงเผื่อค่าความปลอดภัยไปอีกประมาณ 2 เท่า หรือประมาณ 1800 นิวตัน ความเร็วในการทำงานของชุดเบรกไฟฟ้าถูกออกแบบให้ทำงานภายในเวลา 0.2 วินาที จากตำแหน่งเริ่มต้นของลูกสูบจนลูกสูบดันสุด (จุดที่เบรกสามารถทำงานเต็มที่) ซึ่งมีเป็นระยะ 4 เซนติเมตร ดังภาพที่ 45



ภาพที่ 45 *ระยะไฮดรอลิกเบรกเมื่อทำงานเต็มประสิทธิภาพ*

เซอร์โวมอเตอร์ที่เลือกใช้มีขนาด 400 วัตต์ ความเร็วรอบสูงสุด 3000 รอบต่อนาที ให้ค่าแรงบิดอยู่ที่ 1.4 นิวตันเมตร ดังภาพที่ 46 ซึ่งจากคุณสมบัตินี้จะพบว่าภายในระยะเวลาเบรก 0.2 วินาที เซอร์โวมอเตอร์จะสามารถหมุนได้ 10 รอบ

ภาพที่ 46 *เซอร์โวมอเตอร์ที่ใช้ในระบบเบรกสั่งงานด้วยสัญญาณทางไฟฟ้า*

เนื่องจากมอเตอร์ไม่สามารถกดเบรกไฮดรอลิกได้โดยตรง จึงจำเป็นต้องใช้อุปกรณ์ในการเปลี่ยนจากการหมุนของมอเตอร์เป็นการเคลื่อนที่เชิงเส้นเพื่อใช้ในการส่งแรงบิดจากมอเตอร์มาที่เบรกไฮดรอลิก งานวิจัยนี้เลือกใช้ Linear ball screw ขนาด lead 5 ซึ่งหมายถึงการหมุนของสกรู 1 รอบทำให้นัทเคลื่อนที่เป็นเชิงเส้นไปได้ 5 มิลลิเมตร ดังภาพที่ 47 การหาแรงบิดเอาต์พุต และกำลังที่ต้องการในการดันเบรกไฮดรอลิกไฟฟ้าจะอ้างอิงการคำนวณสมการที่ (5)

ภาพที่ 47 Linear ball screw lead 5

แรงบิดเอาต์พุตของ Linear ball screw

$$\begin{aligned} \tau_{Ball\ screw} &= 0.177\left(F_{Brake\ master\ cylinder}\right)\left(Lead\right) \\ &= 0.177\left(1800\,N\right)\left(0.005\ m\right) \\ &= 1.593\ Nm \end{aligned} \tag{5}$$

อัตราการหมุนของเฟืองเอาต์พุต จากการเลือกเฟืองที่มีจำนวนฟัน 12 และ 16 ฟัน เป็นไปดังสมการที่ 6

$$\begin{aligned} Gear\ ratio &= \frac{\omega_1}{\omega_2} = \frac{N_1}{N_2} \\ \omega_2 &= \frac{12}{16} \times 3000\ rpm \\ &= 2308\ rpm \end{aligned} \tag{6}$$

โดยที่ ω_1 คือ ความเร็วเชิงมุมของเฟืองอินพุท

ω_2 คือ ความเร็วเชิงมุมของเฟืองเอาต์พุต

N_1 คือ จำนวนฟันของเฟืองอินพุต

N_2 คือ จำนวนฟันของเฟืองเอาต์พุต

จากการคำนวณระยะทางเชิงเส้นที่ได้จากการหมุนของเฟืองเอาต์พุตภายใน 0.2 วินาที คือ 3.85 เซนติเมตร ซึ่งมีค่าใกล้เคียงกับระยะทางที่ลูกสูบดันสุด (4 เซนติเมตร) กำลังที่ต้องการในการดันเบรกไฮดรอลิกส์ไฟฟ้า ดังสมการที่ 7

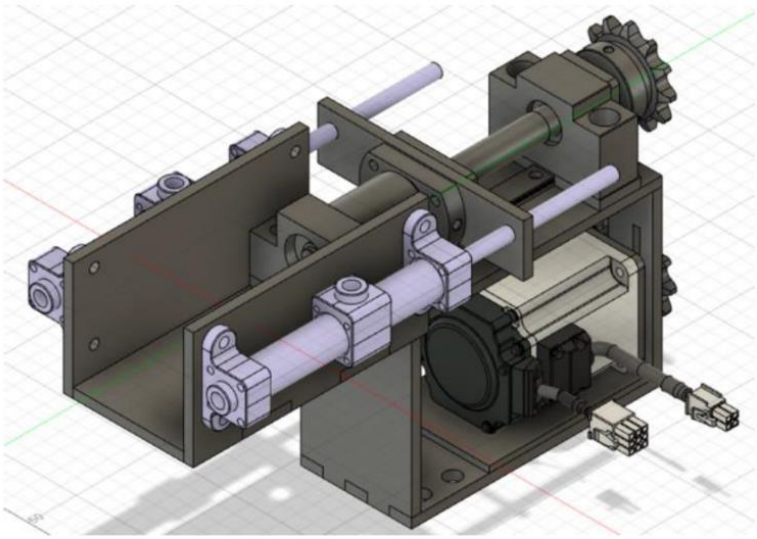
$$Power_{required} = \tau_{Ball\ screw} \times \omega_2 \times \frac{2\pi}{60} \quad (7)$$

$$= (1.593\ Nm) \times (2308\ rpm) \times \frac{2\pi}{60}$$

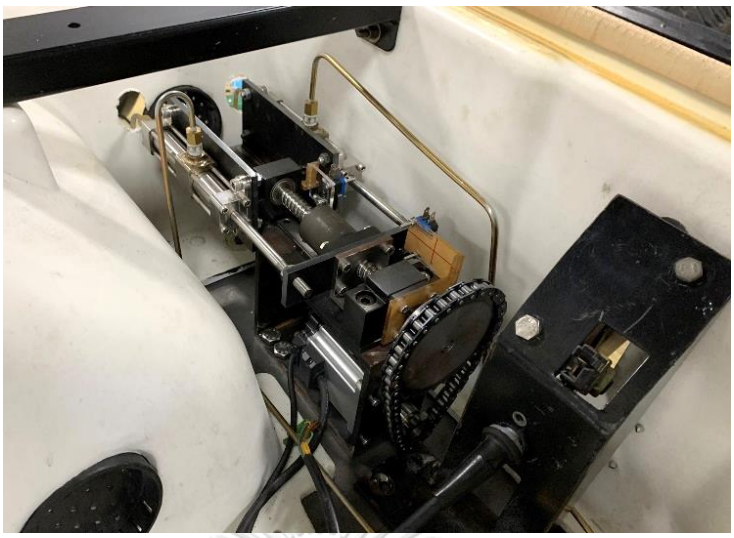
$$= 385.02\ W$$

โดยที่ $\tau_{Ball\ screw}$ คือ ความเร็วเชิงมุมของเฟืองอินพุต

กำลังที่ต้องการใช้ในการกด linear ball screw เพียงพอกับคุณสมบัติของมอเตอร์ที่เลือกไว้ คือ 400 วัตต์ ผลลัพธ์ของการออกแบบชุดเบรกไฟฟ้าเป็นไปดังภาพที่ 48 และลักษณะการติดตั้งบนยานยนต์ต้นแบบเป็นไปดังภาพที่ 49

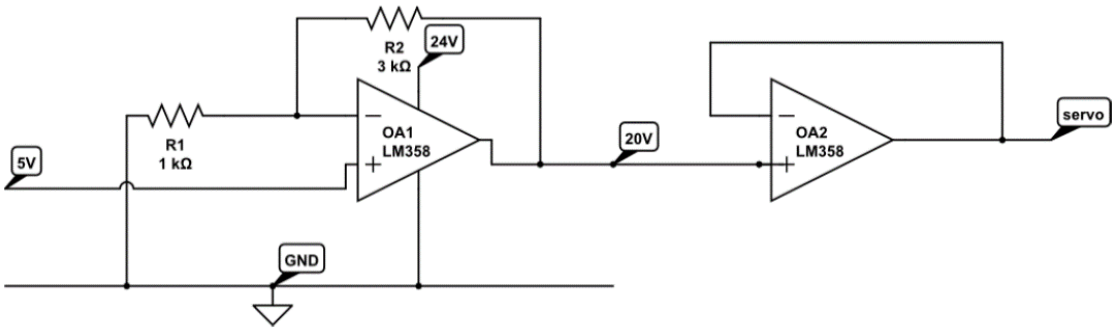


ภาพที่ 48 ลักษณะชุดเบรกสั่งงานด้วยสัญญาณทางไฟฟ้าของยานยนต์ต้นแบบ



ภาพที่ 49 ลักษณะการติดตั้งชุดเบรกไฮดรอลิกสั่งงานด้วยสัญญาณทางไฟฟ้าที่ติดตั้งแล้วบริเวณใต้เบาะกลางของยานยนต์ต้นแบบ

การสั่งงานเซอร์โวมอเตอร์ในโหมดการควบคุมแรงบิด (Torque Control Mode) นั้น ต้องการสัญญาณแรงดันไฟฟ้าที่มีค่าแรงดัน -10 โวลต์ ถึง 10 โวลต์ โดยที่ -10 ถึง 0 โวลต์ มอเตอร์จะหมุนทวนเข็มนาฬิกา 0 ถึง 10 โวลต์ มอเตอร์จะหมุนตามเข็มนาฬิกา และ 0 โวลต์ แรงบิดของมอเตอร์จะเท่ากับ 0 และเนื่องจากบอร์ดควบคุมที่เลือกใช้สามารถจ่ายไฟได้ 0 ถึง 5 โวลต์ จึงต้องทำการออกแบบวงจรที่สามารถแปลงแหล่งจ่ายจากบอร์ดควบคุมให้สามารถใช้งานได้กับ motor driver ที่ต้องการสัญญาณ -10 ถึง 10 โวลต์ โดยใช้ออปแอมป์ (Operational Amplifiers) ดังภาพที่ 50



ภาพที่ 50 *วงจรออปแอมป์ (Operational Amplifiers Circuit)*

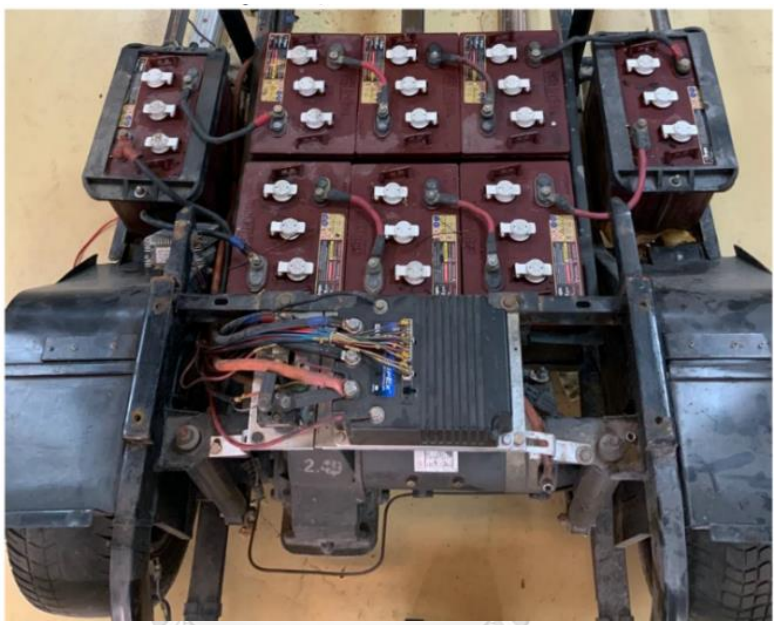
ชุดควบคุมของทั้งระบบบังคับเลี้ยว ระบบควบคุมอัตราเร็ว และระบบเบรกจากหัวข้อที่ผ่านมาจะถูกติดตั้งบริเวณใต้เบาะแถวที่ 2 ของรถกอล์ฟไฟฟ้าดังรูปที่ 51

ภาพที่ 51 *ชุดควบคุมของทั้งระบบบังคับเลี้ยว ระบบควบคุมอัตราเร็ว และระบบเบรกถูกติดตั้งภายใต้เบาะแถวที่สอง ของรถกอล์ฟไฟฟ้า*

6. ระบบไฟฟ้า (Electric System)

6.1. ระบบไฟฟ้าหลัก (Main Electric System)

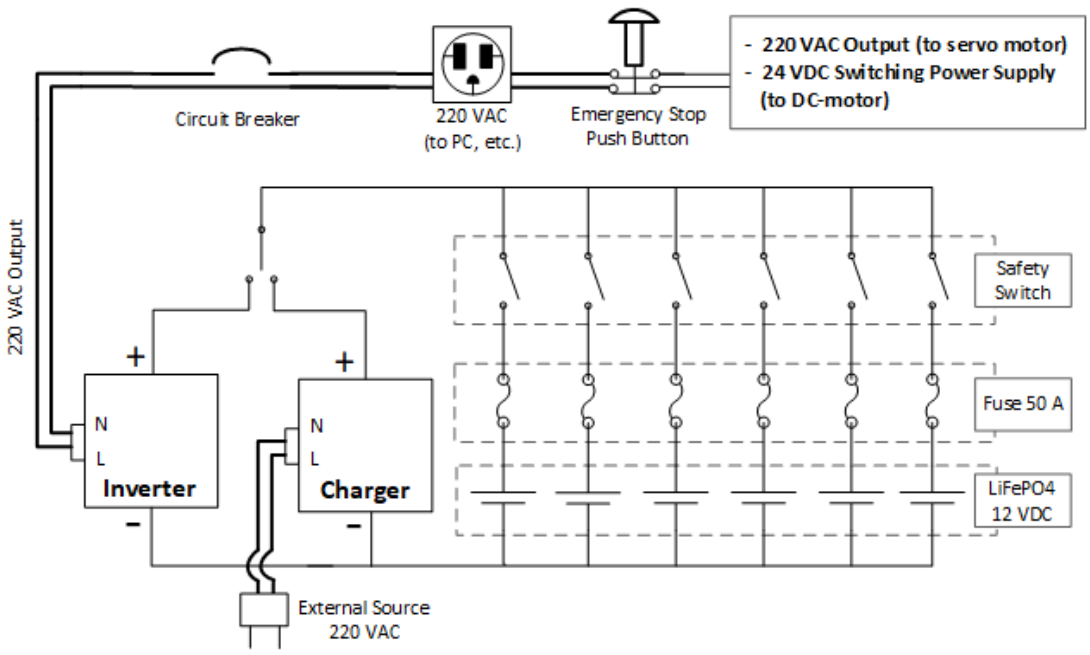
แบตเตอรี่เดิมของรถกอล์ฟขนาด 6 โวลต์ จำนวน 8 ลูก ต่ออนุกรมกันเพื่อใช้ในการจ่ายไฟให้กับกล่องควบคุมอิเล็กทรอนิกส์ในการควบคุมระบบของรถ และการขับมอเตอร์ไฟฟ้าขนาดกำลัง 5 กิโลวัตต์ พิกัดแรงดันกระแสตรง 48 โวลต์ ดังภาพที่ 52



ภาพที่ 52 *แบตเตอรี่จำนวน 8 ชุด แรงดัน 48 โวลต์ สำหรับระบบไฟฟ้าหลัก*

6.2. ระบบไฟฟ้าสำหรับอุปกรณ์เสริม (Auxiliary Electric System)

การพัฒนายานยนต์ต้นแบบในช่วงแรก งานวิจัยนี้ได้เลือกใช้มอเตอร์ไฟฟ้าชนิดเหนี่ยวนำ ซึ่งใช้กระแสสลับในการสั่งงานมอเตอร์ทั้งในระบบบังคับเลี้ยวและระบบเบรก ซึ่งทั้งสองระบบนี้เป็นอุปกรณ์ที่ใช้กำลังไฟค่อนข้างสูงเมื่อเทียบอุปกรณ์ไฟฟ้าอื่น ๆ บนรถ ดั้งนั้นการออกแบบระบบไฟฟ้าสำหรับอุปกรณ์เสริมในช่วงแรกจึงใช้ระบบไฟฟ้ากระแสสลับสำหรับระบบไฟฟ้าของอุปกรณ์เสริม ภายหลังงานวิจัยนี้ได้เปลี่ยนจากการใช้เซอร์โวมอเตอร์กระแสสลับในระบบบังคับเลี้ยว เปลี่ยนเป็นมอเตอร์กระแสตรงแต่ระบบไฟฟ้าสำหรับอุปกรณ์เสริมยังคงใช้ระบบกระแสสลับเหมือนเดิม จึงจำเป็นต้องเพิ่มเติมอุปกรณ์ที่เปลี่ยนแรงดันไฟฟ้ากระแสสลับกลับมาเป็นกระแสตรง (switching power supply) สำหรับมอเตอร์กระแสตรงของระบบบังคับเลี้ยวดังกล่าว ภาพรวมระบบไฟฟ้าสำหรับอุปกรณ์เสริมเป็นไปดังภาพที่ 53



ภาพที่ 53 *ภาพรวมระบบไฟฟ้าสำหรับอุปกรณ์เสริม*

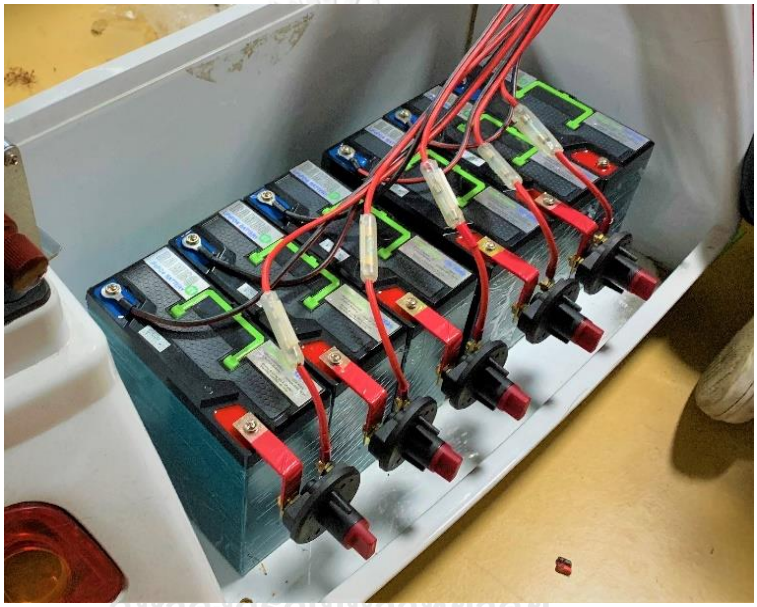
6.2.1. แบตเตอรี่สำหรับอุปกรณ์เสริม

การเลือกขนาดแบตเตอรี่สามารถคำนวณจากกำลังไฟที่ต้องใช้กับอุปกรณ์เสริม เช่น ระบบบังคับเลี้ยว ระบบควบคุมความเร็ว ที่ถูกติดตั้งบนยานพาหนะ ซึ่งแสดงในตารางที่ 3

ตารางที่ 3 *กำลังไฟรวมของอุปกรณ์เสริมที่ใช้ในรถทดสอบ*

อุปกรณ์	จำนวน	กำลังไฟที่สูงสุด (วัตต์)
Servo motor	1	440
Mini PC	1	330
LiDAR	1	8
12V Switching Power Supply	1	24
24V Switching Power Supply	1	720
LED Status Mode	1	3
รวม		1485

งานวิจัยนี้เลือกใช้แบตเตอรี่ LiFePO4 ซึ่งแบตเตอรี่ชนิดนี้มีขนาดที่เล็กและเบากว่าเมื่อเปรียบเทียบกับแบตเตอรี่ชนิดอื่น เช่น แบตเตอรี่ชนิดตะกั่ว ขนาดของแบตเตอรี่สามารถติดตั้งกับยานยนต์ต้นแบบได้ ภายในของแบตเตอรี่ถูกแพคมาพร้อมกับ BMS (Battery Management System) แบบ Active balance ในการจัดการระบบไฟของแบตเตอรี่ โดยจะตัดการทำงานเมื่อเซลล์แบตเตอรี่ภายในมีการชาร์จเต็ม หรือกระแสต่ำกว่ากำหนดระหว่างการใช้งาน เพื่อความปลอดภัย และอายุการใช้งานที่เพิ่มขึ้น เพื่อความปลอดภัยขั้นต้นสวิตช์เปิด-ปิด และฟิวส์ขนาด 50 แอมป์ ถูกติดตั้งที่แบตเตอรี่แต่ละก้อน ดังภาพที่ 54



ภาพที่ 54 *แบตเตอรี่ LiFePO4 12V 32Ah จำนวน 6 ก้อน*

กำลังไฟทั้งหมดที่ได้จากแบตเตอรี่เสริม LiFePO4 แรงดัน 12 โวลต์ กระแส 32 แอมแปร์ จำนวน 6 ก้อน ได้ผลลัพธ์ดังสมการที่ 8

$$P = VI \tag{8}$$

$$12V \times (32\,A) \times 6 = 2.304\ \ kW$$

โดยที่ P คือ กำลังไฟฟ้า (กิโลวัตต์)

V คือ แรงดันไฟฟ้า (โวลต์)

I คือ กระแสไฟฟ้า (แอมแปร์)

จากการคำนวณพบว่าแบตเตอรี่สามารถจ่ายไฟให้กับอุปกรณ์ต่าง ๆ ของยานยนต์ต้นแบบขณะใช้งานทุกอุปกรณ์พร้อมกันกรณีใช้โหลดสูงสุดได้มากกว่า 1 ชั่วโมง ซึ่งถือว่าเพียงพอในแต่ละรอบการใช้งาน

6.2.2. อุปกรณ์แปลงแรงดันไฟฟ้ากระแสตรงเป็นกระแสสลับ (Inverter)

อุปกรณ์เสริมที่ใช้บนยานพาหนะต้นแบบส่วนใหญ่ใช้ไฟฟ้ากระแสสลับ 220 โวลต์ จึงจำเป็นต้องติดตั้งเครื่องแปลงแรงดันไฟฟ้าขนาด 3000 วัตต์ ในการแปลงไฟฟ้ากระแสตรงที่ได้จากแบตเตอรี่ LiFePO4 เป็นไฟฟ้ากระแสสลับ ดังภาพที่ 55

ภาพที่ 55 *เครื่องแปลงแรงดันไฟฟ้าจากกระแสตรงเป็นกระแสสลับ*

6.2.3. เครื่องชาร์จแบตเตอรี่สำหรับอุปกรณ์เสริม

การชาร์จไฟให้กับแบตเตอรี่ LiFePO4 ทั้ง 6 ก้อน จะใช้เครื่องชาร์จขนาด 14.6V 20A ดังภาพที่ 56 และแยกการชาร์จออกจากระบบชาร์จไฟของแบตเตอรี่หลังที่ใช้ในการขับเคลื่อนยานยนต์ต้นแบบ

ภาพที่ 56 *เครื่องชาร์จแบตเตอรี่ LiFePO4*

6.2.4. อุปกรณ์เปลี่ยนแรงดันไฟฟ้ากระแสสลับเป็นกระแสตรง (Switching Power Supply)

เนื่องจากไฟฟ้าที่ได้จากอินเวอร์เตอร์เป็นไฟฟ้ากระแสสลับจึงต้องเปลี่ยนเป็นไฟฟ้ากระแสตรงก่อนเข้าอุปกรณ์ที่ใช้ไฟฟ้ากระแสตรง ดังภาพที่ 57 เช่น ตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าของระบบบังคับเลี้ยว

ภาพที่ 57 *อุปกรณ์เปลี่ยนแรงดันไฟฟ้ากระแสสลับเป็นกระแสตรง*

6.2.5. ปุ่มหยุดการทำงานฉุกเฉิน (Emergency Stop Push Button)

ในสถานการณ์ฉุกเฉิน เช่น ชุดอุปกรณ์สั่งงานด้วยสัญญาณทางไฟฟ้าทำงานพลาด หรือระหว่างที่อยู่ในระหว่างการดำเนินการของระบบอัตโนมัติมีเหตุให้ต้องหยุดการทำงานกระทันหัน ผู้ควบคุมสามารถหยุดการทำงานได้ทันที ไม่ว่าจะอยู่บนรถหรืออยู่ ณ สถานีควบคุม ดังภาพที่ 58 งานวิจัยนี้ได้แบ่งประเภทปุ่มหยุดการทำงานฉุกเฉินออกเป็น 2 โหมด ดังนี้

ภาพที่ 58 *(ก) ปุ่มหยุดการทำงานของระบบอัตโนมัติด้วยคำสั่งควบคุม(ปุ่มขวา) ที่ถูกติดตั้งบนยานยนต์ต้นแบบ และ (ข) ปุ่มหยุดการทำงานของระบบอัตโนมัติ ด้วยคำสั่งควบคุมที่ถูกติดตั้ง ณ สถานีควบคุม(ปุ่มสีแดง)*

<u>แบบที่ 1</u> หยุดการทำงานของระบบอัตโนมัติด้วยคำสั่งควบคุม

ปุ่มหยุดการทำงานฉุกเฉินในโหมดนี้จะถูกติดตั้งทั้งบนยานยนต์ต้นแบบ และสถานีควบคุม ดังภาพที่ 58 เพื่อหยุดการทำงานของระบบอัตโนมัติได้ทั้งจากบนรถหรือจากสถานีควบคุมระยะไกล ด้วยวิธีการส่งสัญญาณไปยังระบบประมวลผลบนรถให้ทำการยกเลิกการทำงานทั้งในระบบบังคับเลี้ยวและระบบเร่ง แต่ในระบบเบรกเมื่อทำการกดปุ่มในโหมดนี้ เบรกจะทำงานเต็มที่จนกระทั่งระบบตรวจพบว่าอัตราเร็วของรถเป็นศูนย์จึงทำการปล่อยเบรก

<u>แบบที่ 2</u> โหมดตัดระบบไฟฟ้าของอุปกรณ์เสริมบนรถ

ปุ่มหยุดการทำงานฉุกเฉินในโหมดนี้จะเป็นการตัดการจ่ายไฟของอุปกรณ์เสริมบนยานยนต์ต้นแบบทั้งหมด ยกเว้นส่วนของระบบคอมพิวเตอร์ และโมดูลการสื่อสาร โดยปุ่มนี้จะถูกติดตั้งอยู่เพียงบนรถเท่านั้นดังภาพที่ 58 (ก) ปุ่มซ้าย

7. เซ็นเซอร์ในงานควบคุมระดับสูง (High-level Sensors)

7.1 เซ็นเซอร์ตรวจจับและวัดระยะทางด้วยแสง (Light Detection and Ranging: LiDAR)

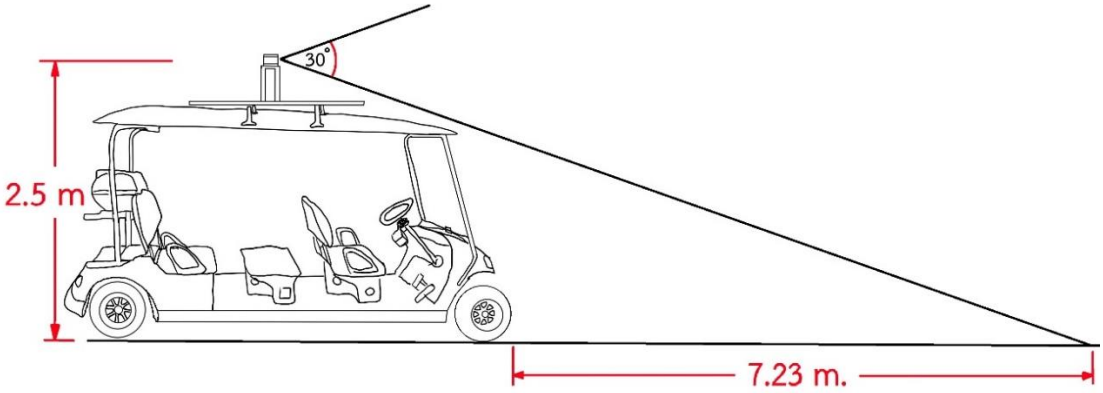
เซ็นเซอร์ตรวจจับและวัดระยะทางด้วยแสงแบบสามมิติ (3D LiDAR Scanner) ยี่ห้อ Velodyne รุ่น VLP-16 ดังภาพที่ 59 ถูกติดตั้งบนยานยนต์ต้นแบบเพื่อใช้ในการวัดระยะทางของวัตถุ โดยรอบจากการปล่อยลำแสงจำนวน 16 เส้น ภายในมุม ± 15 ของแนวตั้ง และหมุนรอบตัวเอง 360 องศา ไปยังวัตถุและสะท้อนกลับมา ด้วยวิธีการนี้จึงสามารถคำนวณระยะทางจากเวลาที่แสงเดินทางได้ (Time of Flight: TOF) และนำไปสู่การได้มาของข้อมูลสำหรับงานควบคุมระดับสูงของยานยนต์อัตโนมัติ เช่น การสร้างแผนที่สามมิติ การระบุตำแหน่งของเซ็นเซอร์เทียบกับแผนที่ และการตรวจจับสิ่งกีดขวาง

ภาพที่ 59 *อุปกรณ์ตรวจจับและวัดระยะทางด้วยแสงแบบสามมิติ*

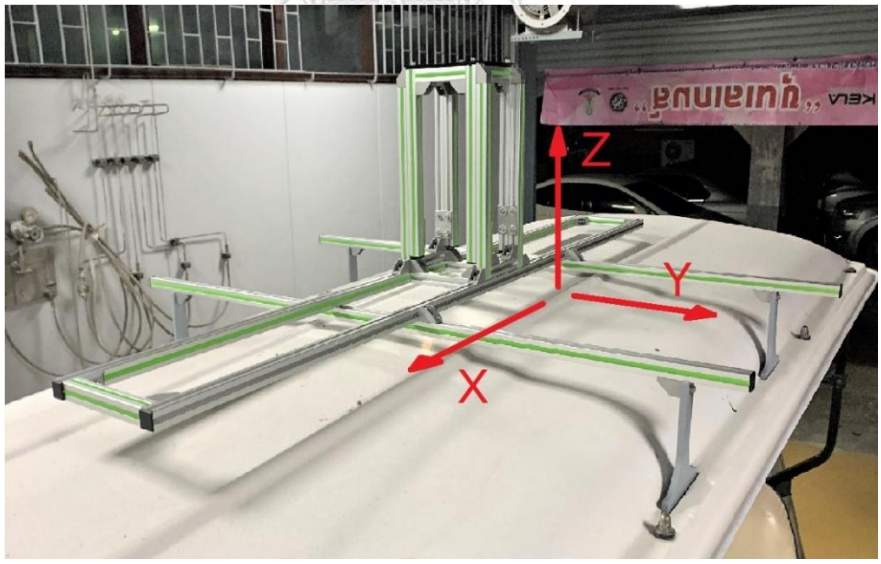
เพื่อให้ได้ข้อมูลสภาพแวดล้อมโดยรอบจากเซ็นเซอร์มากที่สุด งานวิจัยนี้จึงติดตั้งเซ็นเซอร์ตรวจจับและวัดระยะทางด้วยแสงบริเวณหลังคาของยานยนต์ต้นแบบ และออกแบบตำแหน่งการติดตั้งเซ็นเซอร์เพื่อไม่ให้เซ็นเซอร์มองเห็นหลังคารถ แต่มองเห็นสภาพแวดล้อมด้านหน้าได้ใกล้ตัวรถที่สุด ตำแหน่งการติดตั้งเซ็นเซอร์ที่เหมาะสม คือบริเวณตรงกลางของหลังคารถที่ระดับความสูงจาก

พื้นประมาณ 2.50 เมตร ซึ่งเซ็นเซอร์จะมองเห็นพื้นถนนได้ใกล้รถที่สุดและไม่เห็นหลังคาของรถในระยะห่าง 7.23 เมตร จากด้านหน้าของรถ ดังภาพที่ 60

เพื่อความยืดหยุ่นของการปรับตำแหน่งเซ็นเซอร์นี้ในแต่ละงาน โครงสร้างฐานของเซ็นเซอร์ที่ออกแบบในงานวิจัยนี้ ถูกออกแบบให้สามารถปรับตำแหน่งได้ 3 แกน ดังภาพที่ 61



ภาพที่ 60 *ระยะห่างระหว่างจุดที่เซ็นเซอร์ตรวจจับพื้นที่ใกล้ที่สุดกับตัวรถ*

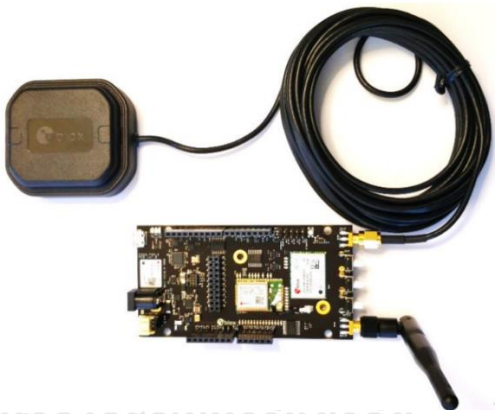


ภาพที่ 61 *โครงสร้างรองรับอุปกรณ์ตรวจจับและวัดระยะทางด้วยแสงที่สามารถปรับระยะได้*

7.2. โมดูลรับสัญญาณเพื่อประมวลผลเชิงตำแหน่งด้วยระบบดาวเทียมนำร่อง (Global Navigation Satellite System Module)

ระบบ Global Navigation Satellite System หรือ GNSS คือ ระบบนำทางด้วยดาวเทียม ซึ่งส่งสัญญาณตำแหน่งของดาวเทียม (Broadcast) พร้อมทั้งอัตราส่วนของเวลา (Time Scale) แบบรอบทิศ (Omnidirectional) กระจายลงสู่พื้นโลก และใช้อุปกรณ์อิเล็กทรอนิกส์เป็นตัวรับสัญญาณ

งานวิจัยนี้เลือกใช้โมดูล U-blox C099 F9P ดังภาพที่ 62 ในการรับสัญญาณจากโครงข่ายดาวเทียม เพื่อประมวลผลเชิงตำแหน่ง ณ จุดที่อุปกรณ์รับสัญญาณ ท้ายสุดข้อมูลที่ได้จากการประมวลผลจะอยู่ในรูปของละติจูด และลองจิจูดของโมดูล ข้อมูลเหล่านี้จะถูกนำไปใช้เป็นข้อมูลอ้างอิงในการสร้างแผนที่สามมิติให้มีความแม่นยำมากขึ้น หรือใช้เพื่อช่วยระบุตำแหน่งของยานยนต์ต้นแบบในช่วงต้นของการเปิดใช้งานระบบ



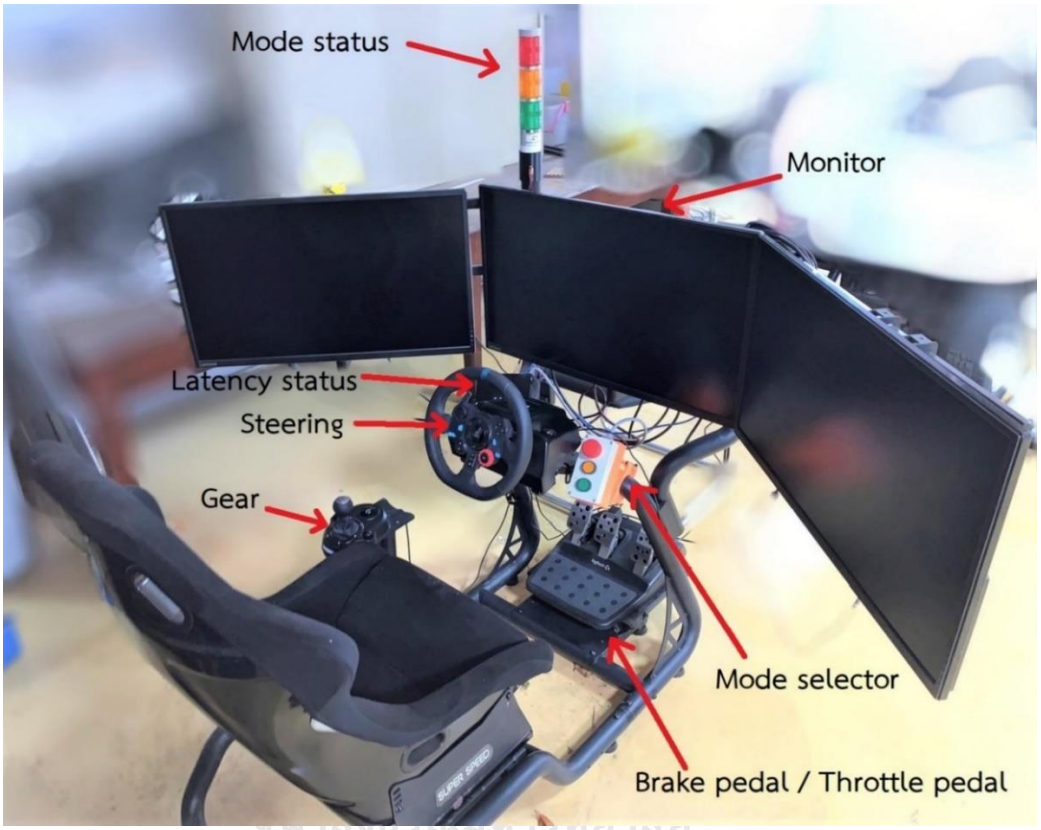
ภาพที่ 62 *อุปกรณ์รับสัญญาณจากโครงข่ายดาวเทียม U-blox c099 F9P*

8. สถานีควบคุม (Control Station or Cockpit)

ภาพจากกล้องวิดีโอที่ใช้ในการถ่ายทอดภาพที่มีความละเอียดขนาด 720p จำนวน 4 ตัว ที่ติดตั้งบริเวณต่าง ๆ บนยานยนต์ต้นแบบจะถูกส่งมายังสถานีควบคุม เพื่อให้ผู้ควบคุมสามารถตรวจสอบสภาพแวดล้อมโดยรอบของยานพาหนะได้ด้วยจอภาพ (Monitor) จำนวน 3 จอ ทำให้ผู้ขับขี่มองเห็นภาพในมุมกว้าง และเป็นการสร้างสภาพแวดล้อมที่สมจริงยิ่งขึ้นแก่ผู้ขับขี่ ดังภาพที่ 63

ชุดควบคุมของสถานีควบคุมประกอบไปด้วย คอมพิวเตอร์ Logitech gaming driving force -G29 (ประกอบไปด้วย พวงมาลัย ไฟเลี้ยว ไฟหน้า แตร คันเร่ง แป้นเบรก เกียร์) ดังภาพที่ 64 และอุปกรณ์ของระบบเปลี่ยนผ่านการควบคุม เช่น ปุ่มหยุดการทำงานฉุกเฉินของรถจากระยะไกล

ปุ่มสลับโหมดอัตโนมัติและโหมดควบคุมระยะไกล หลอดไฟบ่งบอกสถานะโหมดที่ใช้งาน (Mode status) และ หลอดไฟ LED บ่งบอกช่วงความหน่วงเวลาในการส่งสัญญาณแบบเรียลไทม์ ซึ่งถูกติดตั้งบริเวณรอบเก้าอี้ในลักษณะคล้ายกับภายในของรถยนต์ทั่วไปเพื่อให้ผู้ควบคุมเกิดความคุ้นชินเหมือนการขับขี่จริง

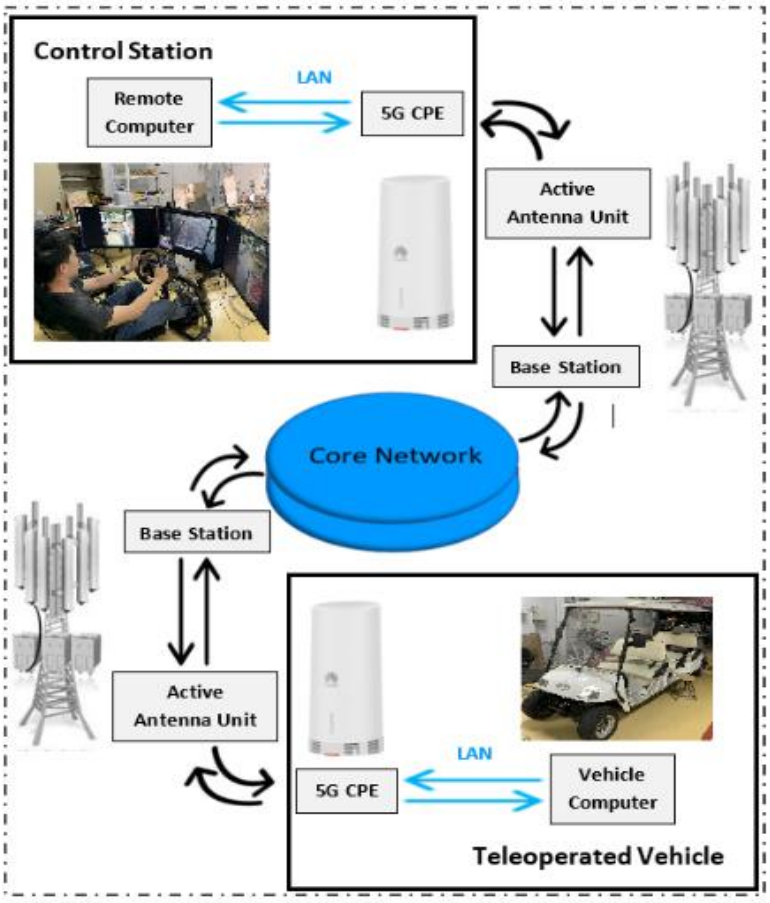


ภาพที่ 63 *ภาพรวมของสถานีควบคุมระยะไกล*

ภาพที่ 64 *Logitech gaming driving force - G29*

9. ระบบสื่อสารสำหรับการควบคุมระยะไกล

การควบคุมยานยานยนต์ต้นแบบโดยผู้ขับขี่ที่อยู่ห่างไกลออกไป จำเป็นต้องมีการสื่อสารเพื่อรับ-ส่งข้อมูลระหว่างคอมพิวเตอร์ทั้งสถานีควบคุมและยานพาหนะผ่านทางเครือข่าย 5G การสื่อสารกันระหว่างคอมพิวเตอร์ 2 เครื่อง เริ่มจากคอมพิวเตอร์เครื่องหนึ่งส่งสัญญาณผ่านสายนำสัญญาณ (Twisted pair cable) ไปยังอุปกรณ์รับกระจายสัญญาณอินเตอร์เน็ต (Customer Premises Equipment : CPE) โดยอุปกรณ์นี้จะส่งสัญญาณต่อไปยังเสากระจายสัญญาณ (Active Antenna Unit : AAU) ที่เป็นส่วนหนึ่งของสถานีฐาน (Base Station) สัญญาณจะถูกส่งผ่านทางสายเคเบิลใยแก้วนำแสงไปยังแกนเครือข่าย (Core Network) ซึ่งทำหน้าที่ส่งสัญญาณจากเครือข่ายหนึ่งข้ามไปยังอีกเครือข่ายหนึ่งจากนั้นสัญญาณจะถูกส่งไปยังคอมพิวเตอร์อีกเครื่องด้วยวิธีการเดียวกัน ดังภาพที่ 65



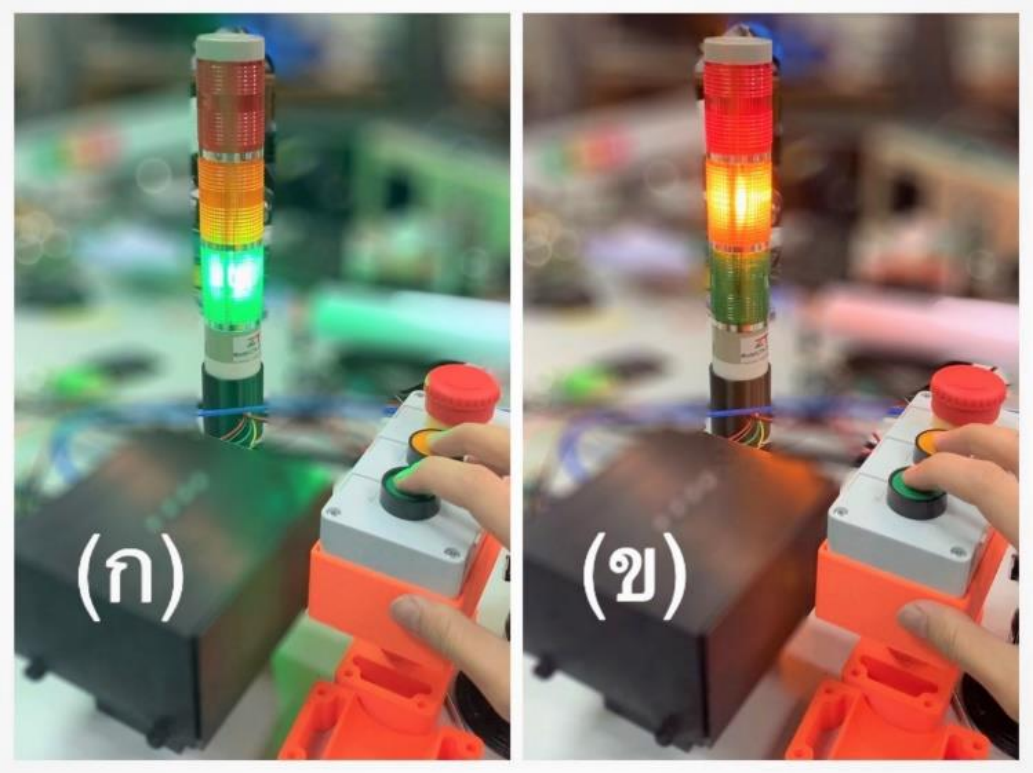
ภาพที่ 65 *ส่วนของการสื่อสารระหว่างยานพาหนะกับสถานีควบคุม*

งานวิจัยนี้ใช้เครือข่ายเซลลูลาร์ 5G Sub-6GHz ประเภท Non-Standalone ในการส่งข้อมูลทั้งจากสถานีควบคุมไปยังยานยนต์ต้นแบบและจากยานยนต์ต้นแบบกลับไปยังสถานีควบคุม ซึ่งข้อมูลที่ส่งจากสถานีควบคุมไปยังยานพาหนะจะเป็นสัญญาณที่ใช้ควบคุมการทำงานของระบบบังคับเลี้ยว ระบบเร่ง และระบบเบรก สัญญาณที่ถูกส่งไปจะต้องมีความถูกต้องคงเดิมมากที่สุด เกิดการสูญหายของข้อมูลน้อยที่สุด จึงเลือกใช้โปรโตคอล TCP (Transmission Control Protocol) ในการส่งข้อมูล เนื่องจากเป็นโปรโตคอลที่มีการตรวจสอบความถูกต้องของข้อมูลที่ถูกรับ-ส่ง ส่วนข้อมูลที่ถูกส่งจากยานพาหนะกลับมายังสถานีควบคุมเป็นสัญญาณภาพที่ถ่ายทอดสดมาจากกล้องที่ติดอยู่บนยานพาหนะ ซึ่งสัญญาณภาพที่ถูกส่งกลับไปยังสถานีนั้นมีขนาดใหญ่กว่าค่าสัญญาณที่ใช้ควบคุมยานพาหนะ จึงทำให้ใช้เวลาในการรับส่งข้อมูลมากกว่า เพื่อให้การรับส่งข้อมูลมีความรวดเร็วมากขึ้น จึงเลือกใช้โปรโตคอล UDP (User Datagram Protocol) ซึ่งเป็นโปรโตคอลที่ส่งข้อมูลได้รวดเร็วกว่าโปรโตคอล TCP เนื่องจากมีรูปแบบการสื่อสารที่เรียบง่ายกว่า แต่มีข้อเสียคืออาจมีข้อมูลสูญหายระหว่างทางได้

10. อุปกรณ์ในงานเปลี่ยนผ่านการควบคุมระหว่างระบบอัตโนมัติและระบบควบคุมระยะไกล

10.1 ไฟแจ้งสถานะ และปุ่มสลับโหมดการควบคุม

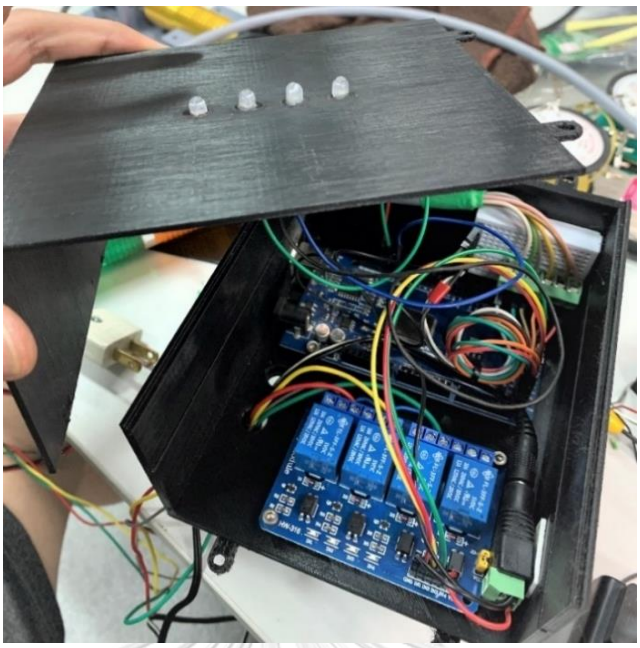
การเปลี่ยนโหมดระหว่างระบบอัตโนมัติและระบบควบคุมระยะไกล ผู้ควบคุม ณ สถานีควบคุมสามารถทำการเปลี่ยนได้ทุกช่วงการใช้งาน และเพื่อเป็นการเช็คสถานะโหมดการใช้งาน ไฟแจ้งสถานะจะเปลี่ยนไปตามโหมดที่ผู้ควบคุมเลือก ความหมายของสีหลอดไฟเป็นไปดังนี้ สีเขียวจะหมายถึงระบบกำลังทำงานอยู่ในโหมดควบคุมจากระยะไกล สีเหลืองหมายถึงระบบกำลังทำงานอยู่ในโหมดอัตโนมัติ ดังภาพที่ 66 ส่วนไฟสีแดงงานวิจัยนี้ได้ออกแบบให้ไฟสีนี้สว่างขึ้นเมื่อระบบตรวจพบสถานการณ์ฉุกเฉิน เช่น มีค่าความหน่วงของเวลาที่สูงเกินกำหนดขณะทำการควบคุมอยู่ในโหมดควบคุมระยะไกล หรือมีสิ่งกีดขวาง ขวางอยู่ด้านหน้าเป็นระยะเวลาเกินกว่าที่ระบบกำหนด ระบบจะแจ้งเตือนให้ผู้ควบคุมทำการยืนยันการเปลี่ยนโหมด และเข้าควบคุมทันหลังจากหลอดไฟสีแดงสว่างขึ้น



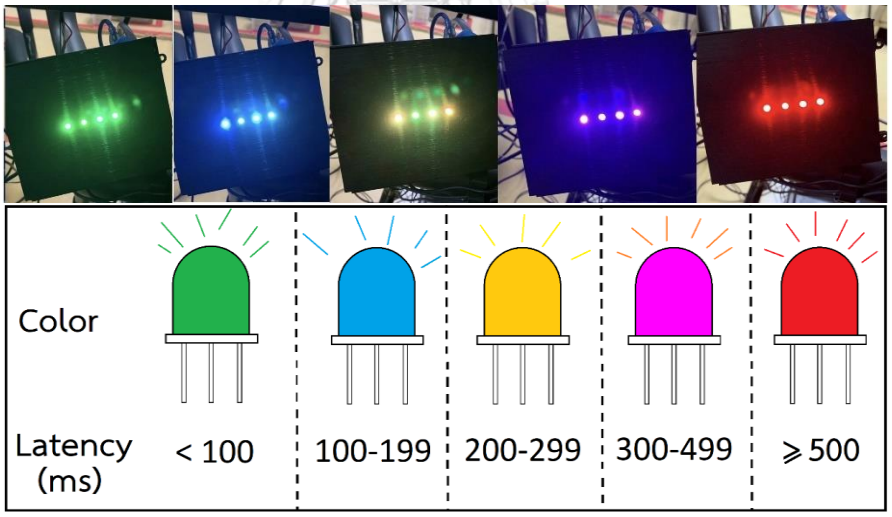
ภาพที่ 66 *(ก) ไฟสีเขียว แสดงถึงระบบกำลังดำเนินการอยู่ในโหมดควบคุมระยะไกล (ข) ไฟสีเหลือง แสดงถึงระบบกำลังดำเนินการอยู่ในโหมดอัตโนมัติ*

10.2 ไฟแจ้งสถานะความล่าช้าของสัญญาณ

การแจ้งสถานะความหน่วงเวลาของสัญญาณแบบเรียลไทม์ให้กับผู้ควบคุม ณ สถานีควบคุม รับรู้จะช่วยเพิ่มประสิทธิภาพการควบคุม ความปลอดภัย ความมั่นใจที่มากขึ้นขณะใช้งานในโหมดควบคุมระยะไกล หลอดไฟ LED ถูกติดตั้งใกล้กับพวงมาลัยเพื่อให้ผู้ควบคุม ณ สถานี สามารถสังเกตสีจากไฟที่เปลี่ยนแปลงได้ง่าย สีของ LED บ่งบอกถึงช่วงความหน่วงเวลาที่กำหนดไว้ในแต่ละช่วง อุปกรณ์ควบคุมที่ชุดกล่อง LED จะเป็นชุดควบคุมเดียวกันกับปุ่มสลับโหมด ปุ่มฉุกเฉิน และไฟสถานะของโหมดการใช้งาน ดังภาพที่ 67 สีของหลอดไฟ LED แต่ละสีบ่งบอกช่วงความหน่วงเวลาของการส่งสัญญาณ เป็นไปตามภาพที่ 68



ภาพที่ 67 *ชุดควบคุมหลอดไฟ LED แจ้งสถานะความหน่วงเวลาของสัญญาณ และชุดควบคุมของปุ่มเปลี่ยนโหมด ปุ่มหยุดการทำงานฉุกเฉิน และหลอดไฟแจ้งสถานะโหมดการใช้งาน*



ภาพที่ 68 *ช่วงความหน่วงเวลาเทียบกับสีของ LED*

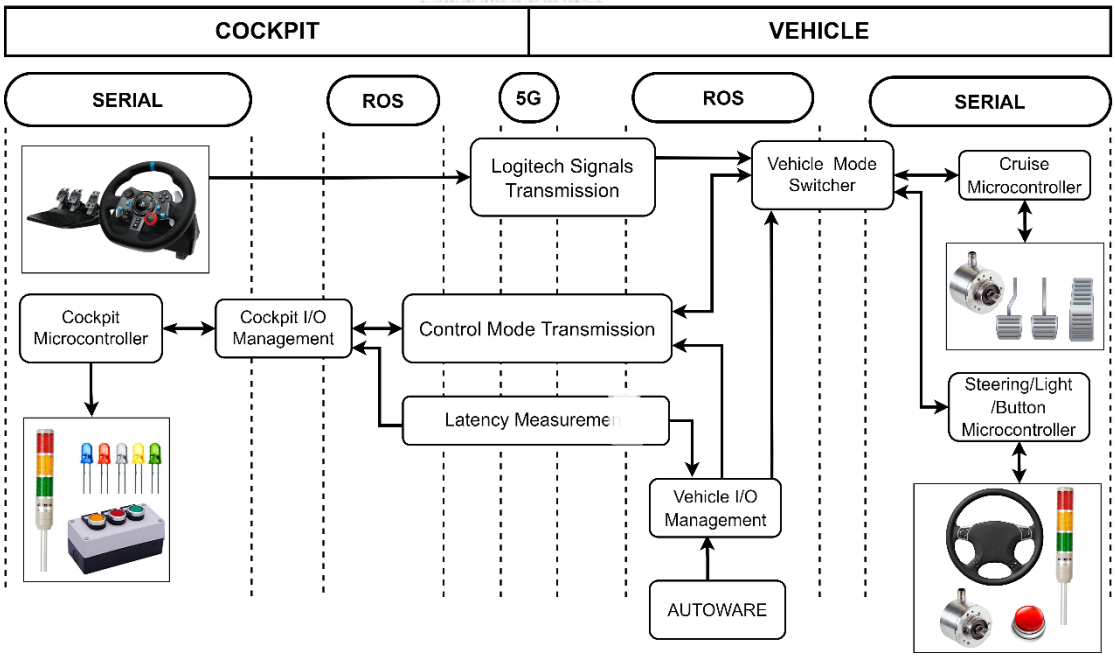
10.3 ปุ่มหยุดการทำงานฉุกเฉินจากระยะไกล

ลักษณะการทำงานของปุ่มหยุดการทำงานฉุกเฉินจากระยะไกลจะมีลักษณะการทำงานเช่นเดียวกับภาพที่ 66 (ก) ด้วยวิธีการส่งสัญญาณระยะไกลจากสถานีควบคุมไปยังระบบควบคุมบนรถเพื่อตัดการทำงานในระบบบังคับเลี้ยวและระบบเร่ง ส่วนระบบเบรกจะทำงานเต็มที่จนกว่าอัตราเร็วของรถจะเป็นศูนย์แล้วจึงคลายเบรก

11. การทำงานของระบบเปลี่ยนผ่านการควบคุมระหว่างระบบอัตโนมัติและระบบควบคุมระยะไกล

โมดูลหลักแต่ละประเภทถูกแบ่งออกตามลักษณะการทำงานที่แตกต่างกันไป ซึ่งมีหน้าที่ในการรับข้อมูลพารามิเตอร์ทั้งจากฟังก์ชันหลักอื่น ๆ เซ็นเซอร์ของรถ อุปกรณ์ที่ต่อไมโครคอนโทรลเลอร์ หรือแอพพลิเคชั่นซอฟต์แวร์ต่าง ๆ นำมาประมวลผลและส่งข้อมูลออกไปตามเส้นทางการรับ-ส่งข้อมูลระหว่างโมดูล ดังภาพภาพที่ 69 รถต้นแบบและสถานีควบคุมถูกแบ่งส่วนแยกออกจากกันเพื่อให้ความเข้าใจง่าย การสื่อสารระหว่างคอมพิวเตอร์ของทั้งสองส่วนนี้จะทำผ่านเครือข่ายไร้สาย 5G จากแผนภาพจะพบว่าโมดูล Logitech signals transmission Control mode transmission และ Latency measurements ทั้ง 3 โมดูลมีหน้าที่ในการรับข้อมูลจากโมดูลอื่นเข้ามาประมวลผลและส่งออกไปยังส่วนอื่นผ่านเครือข่ายไร้สาย

ข้อมูลที่มีการรับ-ส่งมาแล้วจะถูกจัดเก็บเข้าสู่ระบบของ Robot Operating System หรือ ROS ซึ่งทำหน้าที่บริหารจัดการทำงานของระบบทั้งหมด ทั้งข้อมูลที่รับ-ส่งมาจากเครือข่ายไร้สาย จากโมดูลอื่น ๆ ภายใน ROS หรือจากโมดูลที่รับ-ส่งคำสั่งจากอุปกรณ์ภายนอกผ่านการการสื่อสารแบบอนุกรม (Serial communication) ROS จะรับข้อมูลเหล่านี้เข้ามาและจัดประเภทออกเป็นส่วน ๆ ตามฟังก์ชันหน้าที่ที่ผู้ใช้กำหนด ซึ่งแต่ละส่วนสามารถส่งความหากันได้ผ่านทาง topic ที่สร้างไว้

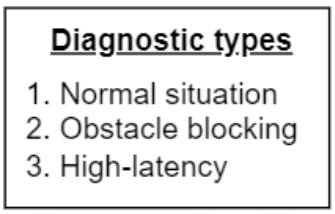


ภาพที่ 69 *ภาพรวมฟังก์ชันย่อย และเส้นทางการรับ-ส่งข้อมูลระหว่างฟังก์ชันย่อยของระบบเปลี่ยนผ่านการควบคุม*

วัตถุประสงค์หลักของการเปลี่ยนผ่านระบบควบคุม คือการเปลี่ยนลักษณะโหมดควบคุมจากโหมดหนึ่งไปยังอีกโหมดหนึ่ง นอกเหนือจากโหมดควบคุมด้วยระบบอัตโนมัติและระบบควบคุมระยะไกลแล้ว งานวิจัยนี้ได้สร้างโหมดสถานการณ์ฉุกเฉินเพื่อรองรับเหตุการณ์ที่ผิดปกติระหว่างกำลังดำเนินในโหมดควบคุมหนึ่ง ซึ่งเกิดขึ้นได้ทั้งจากการที่ผู้ควบคุมบนรถหรืออยู่ ณ สถานีควบคุมทำการกดปุ่มฉุกเฉิน หรือระบบเฝ้าระวังความผิดปกติของรถทดสอบตรวจพบสถานการณ์ที่ผิดปกติ

ระบบเฝ้าระวังความผิดปกติ (Malfunction surveillance system) ที่อยู่บนรถทดสอบ จะรับข้อมูลสภาพแวดล้อมที่เกิดขึ้นโดยรอบของรถ ประมวลผลข้อมูลที่ได้ และให้ผลการวินิจฉัยออกมา

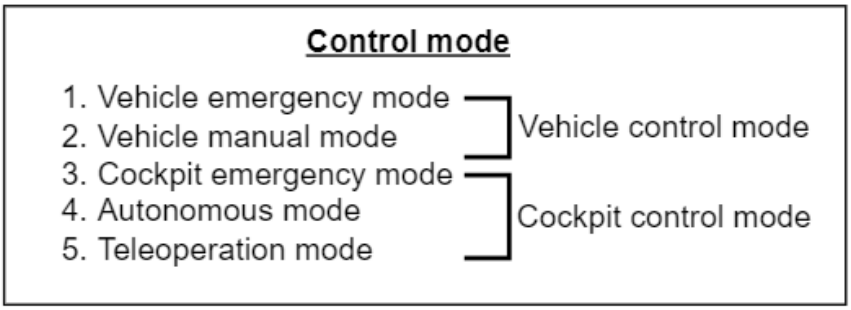
ประเภทของผลการวินิจฉัย (Diagnostic types) แบ่งออกเป็น 3 ประเภท ดังภาพที่ 70 ประเภทแรกคือ เหตุการณ์ปกติ (Normal situation type) ประเภทที่สองคือ ระบบตรวจพบสิ่งกีดขวางบริเวณด้านหน้ามากกว่าเวลาที่กำหนดไว้ (Obstacle blocking type) และประเภทที่สามคือ ระบบตรวจพบความหน่วงสูงเกิดค่าที่กำหนด (High-latency type) ประเภทของการวินิจฉัยเหล่านี้จะถูกส่งไปยังโมดูล Control mode switcher ที่ทำหน้าที่ตัดสินใจว่า รถทดสอบควรที่จะดำเนินการในโหมดอะไร ควรอยู่ในโหมดฉุกเฉินหรือไม่



ภาพที่ 70 *ชนิดของผลการวินิจฉัยความผิดปกติที่ระบบเฝ้าระวังความผิดปกติตรวจพบ*

โหมดการควบคุมของระบบถูกแบ่งออกเป็น 5 โหมดหลัก ดังภาพที่ 71 โหมดแรก คือโหมดสถานการณ์ฉุกเฉินที่แจ้งเตือนจากรถ (Vehicle emergency mode) เกิดขึ้นเมื่อระบบเฝ้าระวังความผิดปกติของรถตรวจพบความผิดปกติบางอย่าง เช่น มีความหน่วงเวลาที่สูงเกิดขึ้น มีสิ่งกีดขวางขวางทางอยู่บริเวณด้านหน้า หรือผู้โดยสารที่อยู่บนรถทำการกดปุ่มฉุกเฉิน โหมดที่สอง คือโหมดควบคุมด้วยมือ (Vehicle manual mode) เมื่อผู้โดยสารหรือผู้ควบคุมที่อยู่บนรถทดสอบทำการสวิตช์ปุ่มมาเป็นการควบคุมด้วยมือ ระบบจะตัดการทำงานของตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้าทุกระบบ และปล่อยให้รถทดสอบสามารถถูกควบคุมจากคนที่อยู่บนรถได้ โหมดต่อมา คือโหมดสถานการณ์ฉุกเฉินจากสถานีควบคุม (Cockpit emergency mode) จะเกิดขึ้นเมื่อผู้ควบคุมที่อยู่ ณ สถานีควบคุมทำการกดปุ่มฉุกเฉินเท่านั้น โหมดที่สี่และห้า คือโหมดควบคุมด้วยระบบอัตโนมัติ

(Autonomous mode) และโหมดควบคุมจากระยะไกล (Teleoperation mode) ผู้ควบคุมที่อยู่ ณ สถานีควบคุมสามาถกดปุ่มสลับโหมดควบคุมได้ทันที เว้นแต่กรณีที่ระบบเฝ้าระวังความผิดปกติที่อยู่บนรถทดสอบตรวจพบเหตุการผิดปกติที่อาจส่งผลให้เกิดอันตรายได้หากผู้ควบคุมทำการสลับไปใช้โหมดดังกล่าว ระบบจะไม่ทำการสลับโหมดจนกว่าเหตุการณ์ถูกแก้ไขแล้วเพื่อความปลอดภัย เช่น หากกำลังใช้งานอยู่ในโหมดอัตโนมัติ และขณะนั้นมีความหน่วงเวลาสูงเกินกำหนดเกิดขึ้น ผู้ควบคุม ณ สถานีควบคุมจะไม่สามารถทำการสลับโหมดไปยังโหมดควบคุมระยะไกลได้ จนกว่าความหน่วงเวลาจะต่ำกว่าที่ถูกกำหนดไว้



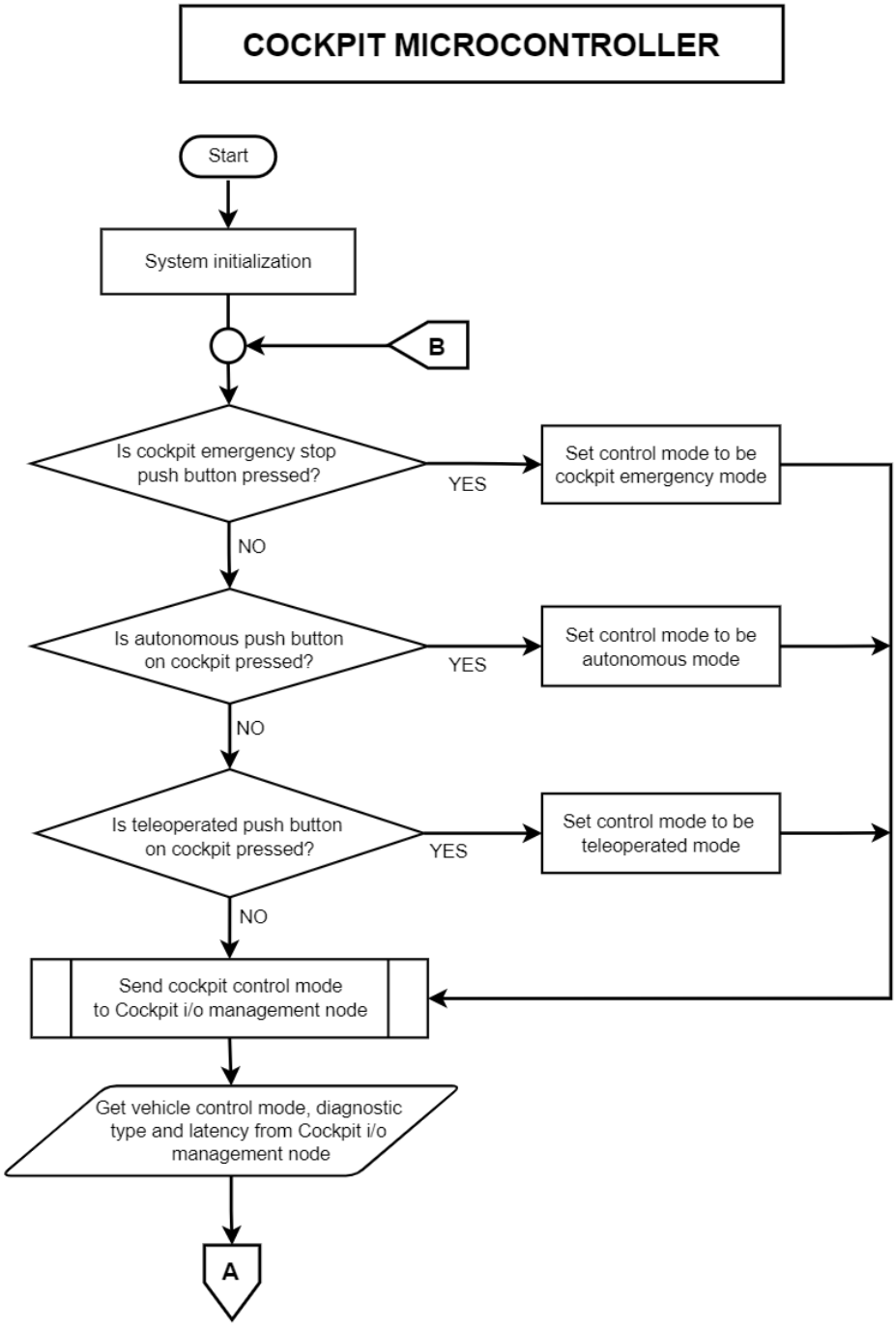
ภาพที่ 71 *โหมดระบุสถานการณ์ควบคุม*

หน้าที่ของแต่ละโมดูลบนผังการทำงานของระบบเปลี่ยนผ่านการควบคุมระหว่างระบบอัตโนมัติและระบบควบคุมระยะไกล มีดังนี้

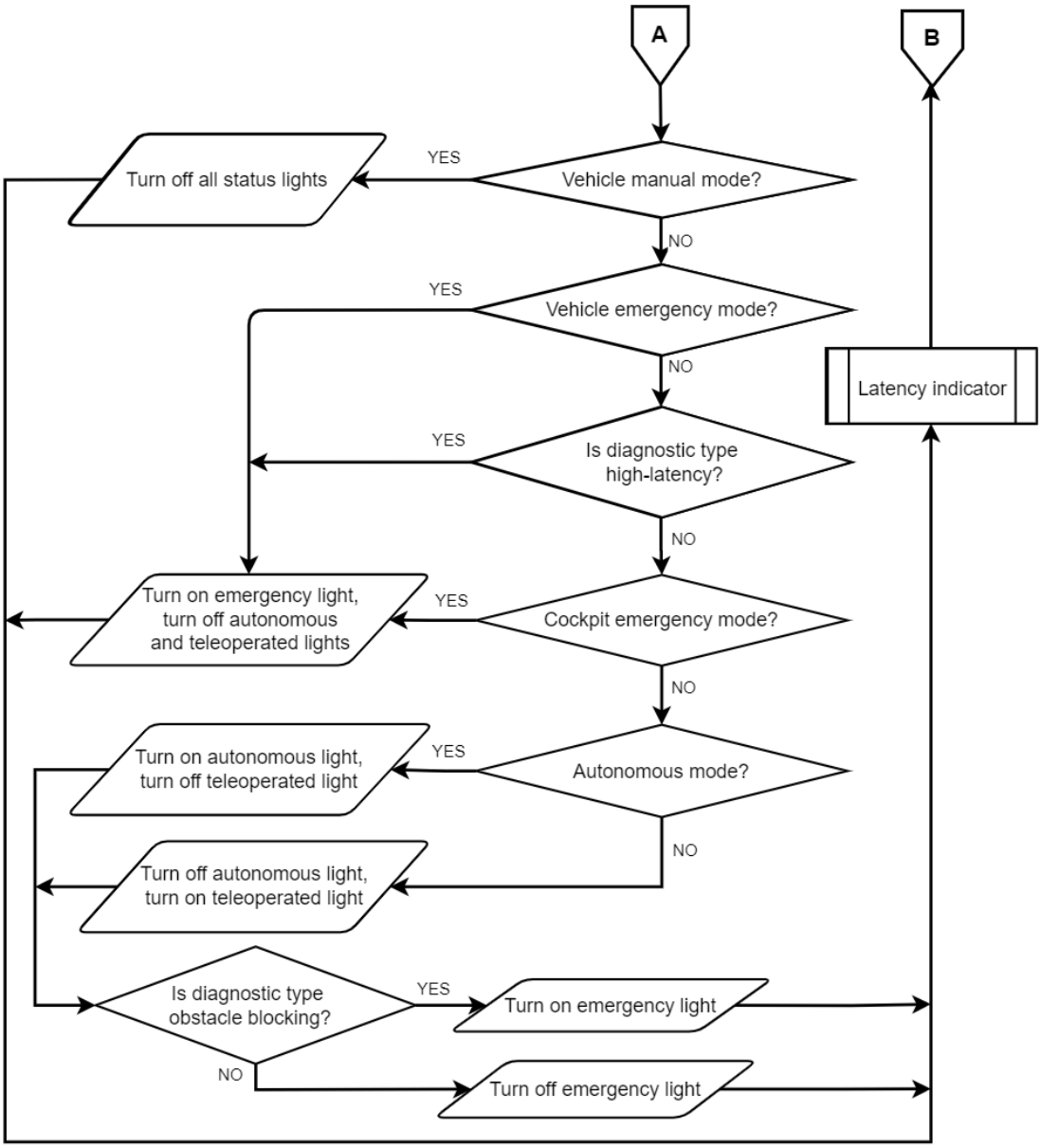
11.1 Cockpit Microcontroller

- รับสัญญาณจากปุ่มสลับโหมดควบคุม แปลงสัญญาณที่ได้เป็นค่าโหมดควบคุม และส่งไปยังโมดูล Cockpit I/O management ผ่านทางพอร์ทอนุกรม
- รับค่าโหมดควบคุม และค่าความล่าช้าของสัญญาณ จากโมดูล Cockpit I/O management ผ่านระบบ ROS
- แสดงผลไฟแจ้งสถานะโหมดควบคุม และไฟแสดงความล่าช้าของสัญญาณ

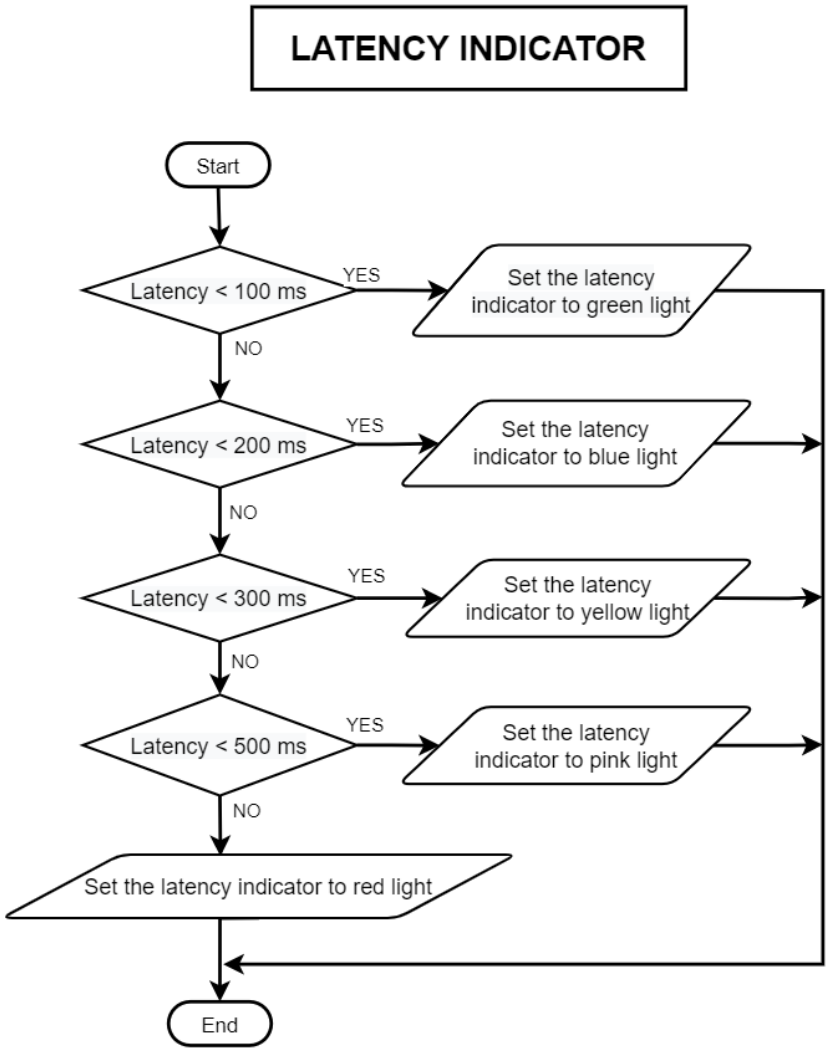
ผังการทำงานของโมดูล Cockpit microcontroller เป็นไปดังภาพที่ 72 ถึง 74



ภาพที่ 72 *ผังการทำงานของโมดูล Cockpit Microcontroller*



ภาพที่ 73 ผังการทำงานของโมดูล Cockpit Microcontroller (ต่อ)

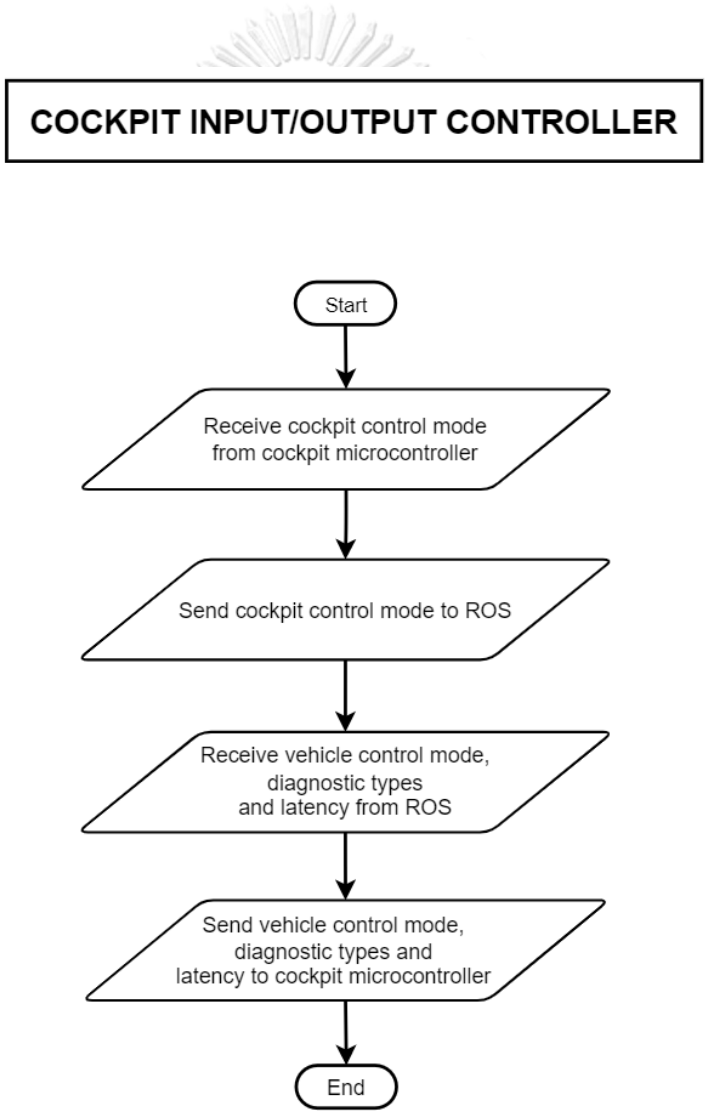


ภาพที่ 74 *ผังการทำงานของโมดูล Cockpit Microcontroller (ต่อ)*

11.2 Cockpit I/O Management

- รับค่าโหมดควบคุมจากโมดูล Cockpit microcontroller ทางการสื่อสารแบบอนุกรม เพื่อส่งไปยังโมดูล Control mode transmission ผ่านทาง ROS
- รับค่าโหมดควบคุมจากโมดูล Control mode transmission และค่าความหน่วงเวลาจากโมดูล Latency management ผ่านทาง ROS และส่งค่าทั้งหมดไปยังโมดูล Cockpit microcontroller

ผังการทำงานของโมดูล Cockpit microcontroller เป็นไปดังภาพที่ 75

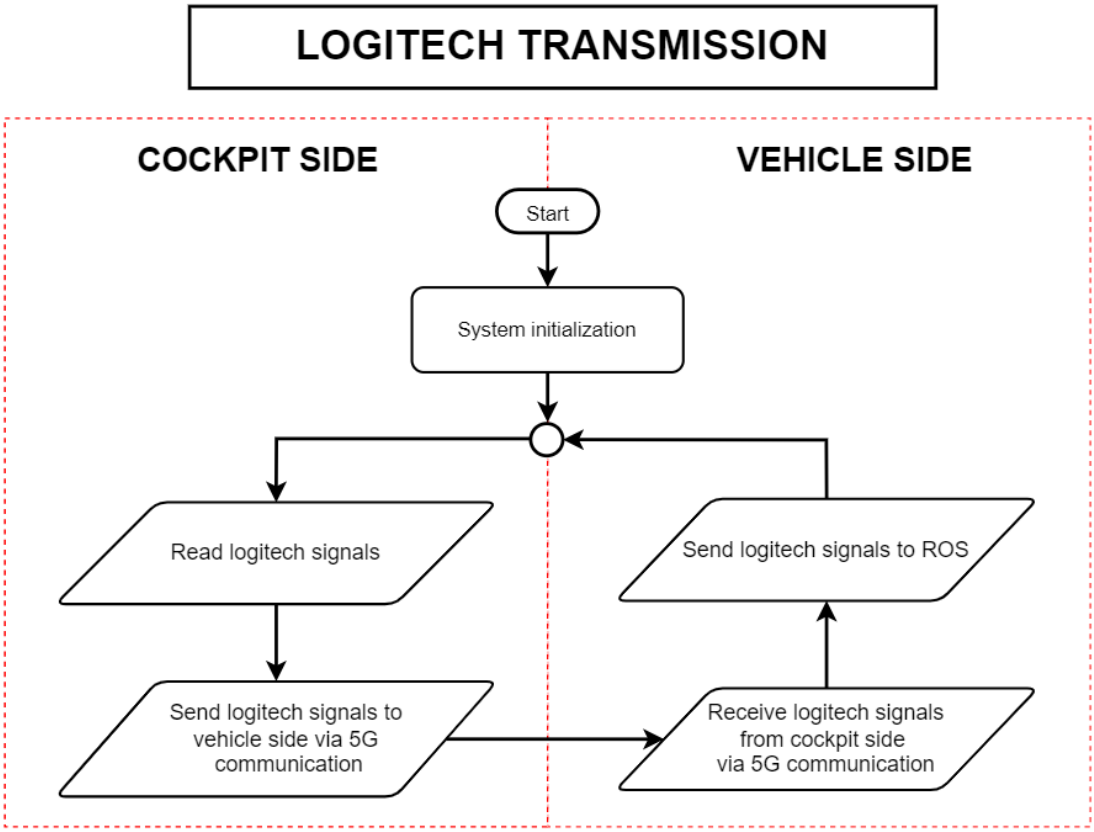


ภาพที่ 75 *ผังการทำงานของโมดูล Cockpit I/O management*

11.3 Logitech Signals Transmission

- รับค่าสัญญาณควบคุมที่ได้จากชุด Logitech gaming driving force - G29 ในฝั่งของสถานีควบคุม และส่งค่าไปยังโมดูล Control mode switcher ซึ่งอยู่ในระบบ ROS ของฝั่งรถทดสอบ ผ่านทางเครือข่าย 5G

ผังการทำงานของโมดูล Cockpit microcontroller เป็นไปดังภาพที่ 76

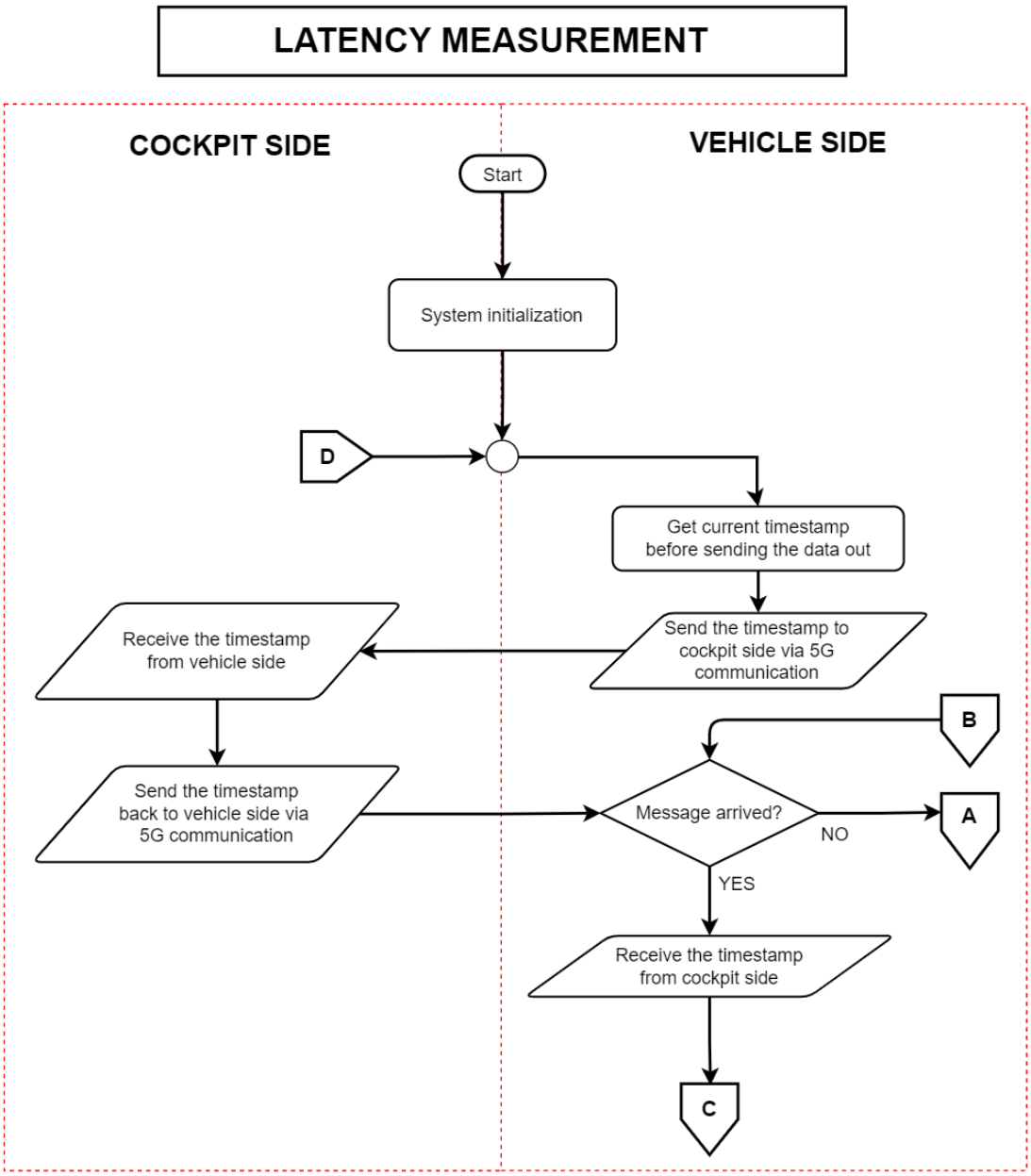


ภาพที่ 76 *ผังการทำงานของโมดูล Logitech transmission*

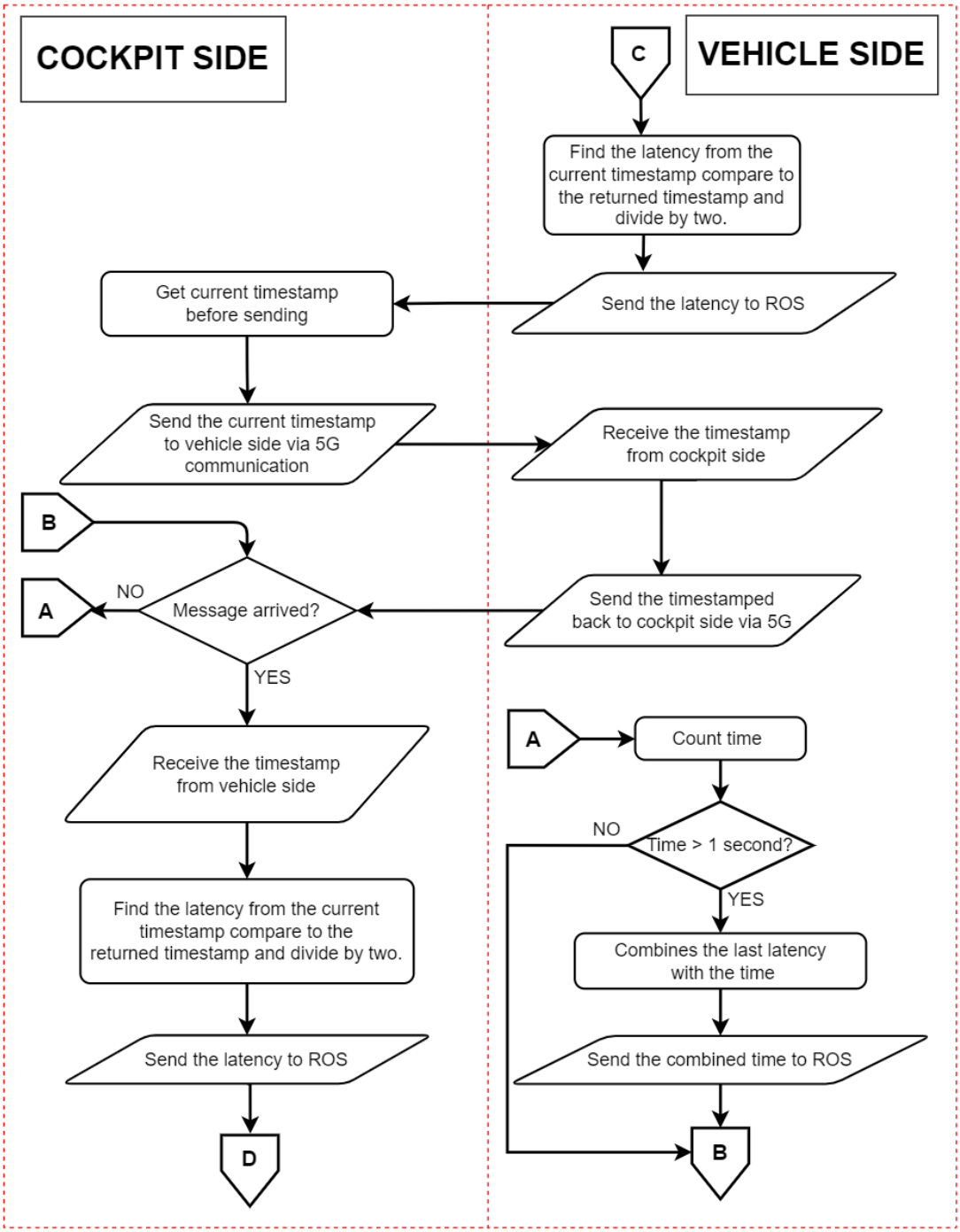
11.4 Latency Measurement

- ทำหน้าที่วัดค่าความล่าช้าของสัญญาณ และส่งผลลัพธ์ไปยังโมดูล Cockpit I/O management และ Vehicle I/O management ที่อยู่ในระบบ ROS

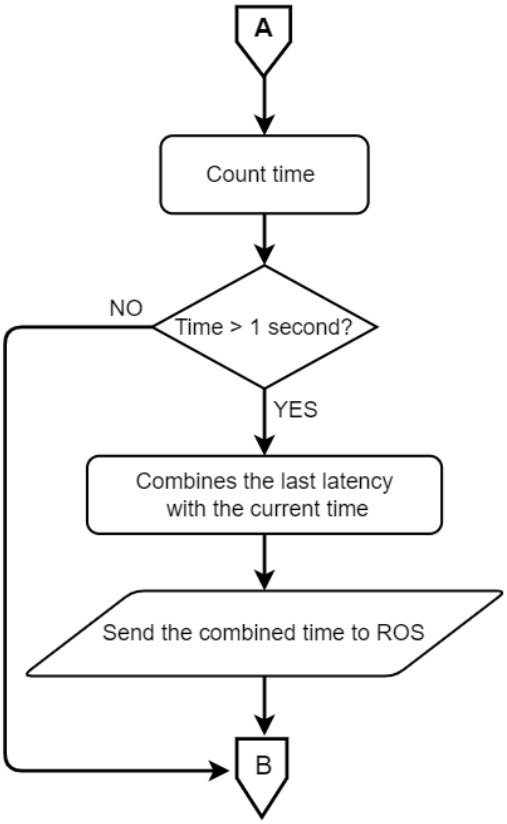
ผังการทำงานของโมดูล Cockpit microcontroller เป็นไปดังภาพที่ 77 ถึง 79



ภาพที่ 77 *ผังการทำงานของโมดูล Latency measurement*



ภาพที่ 78 *ผังการทำงานของโมดูล Latency measuremens (ต่อ)*

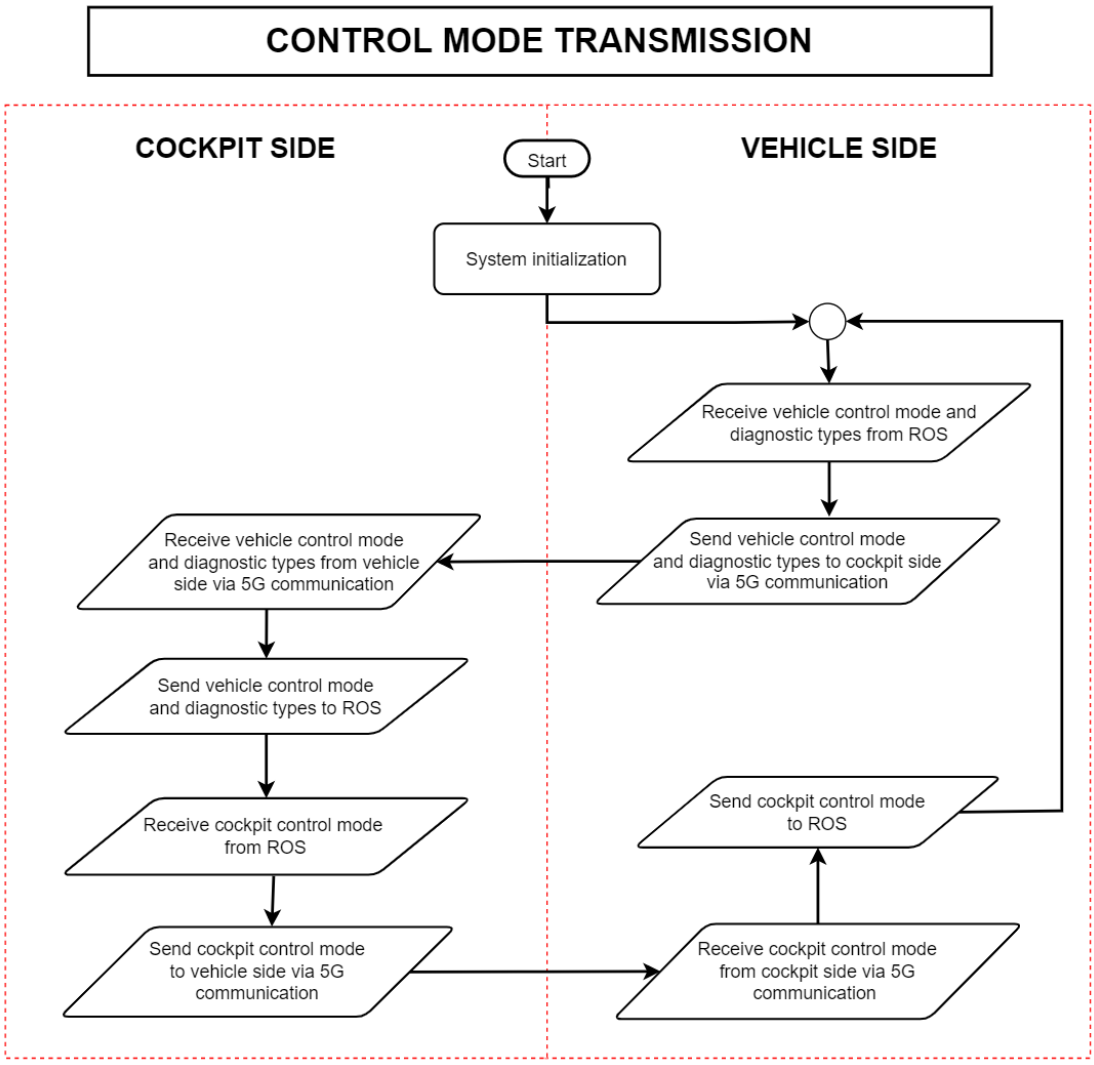


ภาพที่ 79 *ผังการทำงานของโมดูล Latency measurement (ต่อ)*

11.5 Control Mode Transmission

- ทำหน้าที่รับ-ส่งข้อมูลค่าโหมดควบคุม และผลการวินิจฉัยสถานการณ์ จากโมดูล Control mode switcher Vehicle I/O management และ Cockpit I/O management ซึ่งอยู่ในระบบ ROS ของทั้งฝั่งรถทดสอบและสถานีควบคุม ผ่านทางเครือข่าย 5G

ผังการทำงานของโมดูล Cockpit microcontroller เป็นไปดังภาพที่ 80



ภาพที่ 80 *ผังการทำงานของโมดูล Control mode transmission*

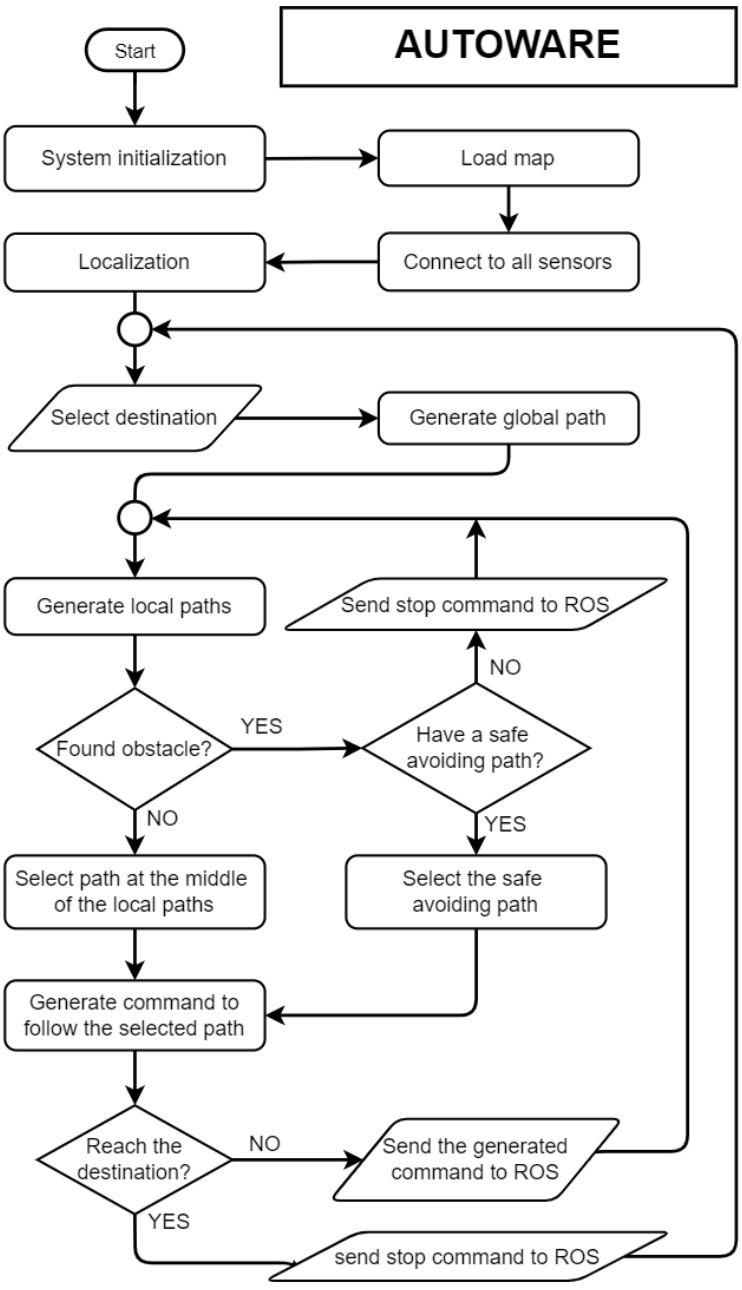
11.6 Autoware

- นำเข้าข้อมูลแผนที่สามมิติ ระบุตำแหน่งรถในแผนที่ วางแผนและสร้างเส้นทางของรถในระดับ global path และ local path ตรวจจับสิ่งกีดขวาง และส่งคำสั่งควบคุมรถออกไปยังระบบ ROS

- ประมวลผลข้อมูลที่ได้จากเซนเซอร์ต่าง ๆ และส่งข้อมูลออกไปยังโมดูล Vehicle I/O management ซึ่งข้อมูลประกอบไปด้วย current_state closest_object_distance และ twist_cmd

  - **current_state** มีหน้าที่ประมวลผลข้อมูลที่ได้จากเซนเซอร์วัดระยะทางที่ใช้ในการตรวจจับสิ่งกีดขวาง ผลลัพธ์ที่ได้มีสองสถานะคือ “Follow” และ “Stop”
    - “Follow” คือสถานะที่รถทดสอบตรวจพบสิ่งกีดขวางบริเวณด้านหน้าแต่ยังคงมีระยะห่างประมาณหนึ่ง
    - “Stop” คือสถานะที่รถทดสอบต้องทำการเบรกเนื่องจากอยู่ใกล้สิ่งกีดขวางค่อนข้างมาก
  - **closest_object_distance** คือข้อมูลระยะทางระหว่างตัวรถและสิ่งกีดขวางที่อยู่บริเวณด้านหน้าที่ใกล้ที่สุด
  - **twist_cmd** คือข้อมูลอัตราเร็วเชิงเส้น (twist.linear.x) และเชิงมุม (twist.angular) ของรถที่ควรจะเป็น ณ ขณะนั้น

ผังการทำงานของแอพพลิเคชั่น Autoware เป็นไปดังภาพที่ 81

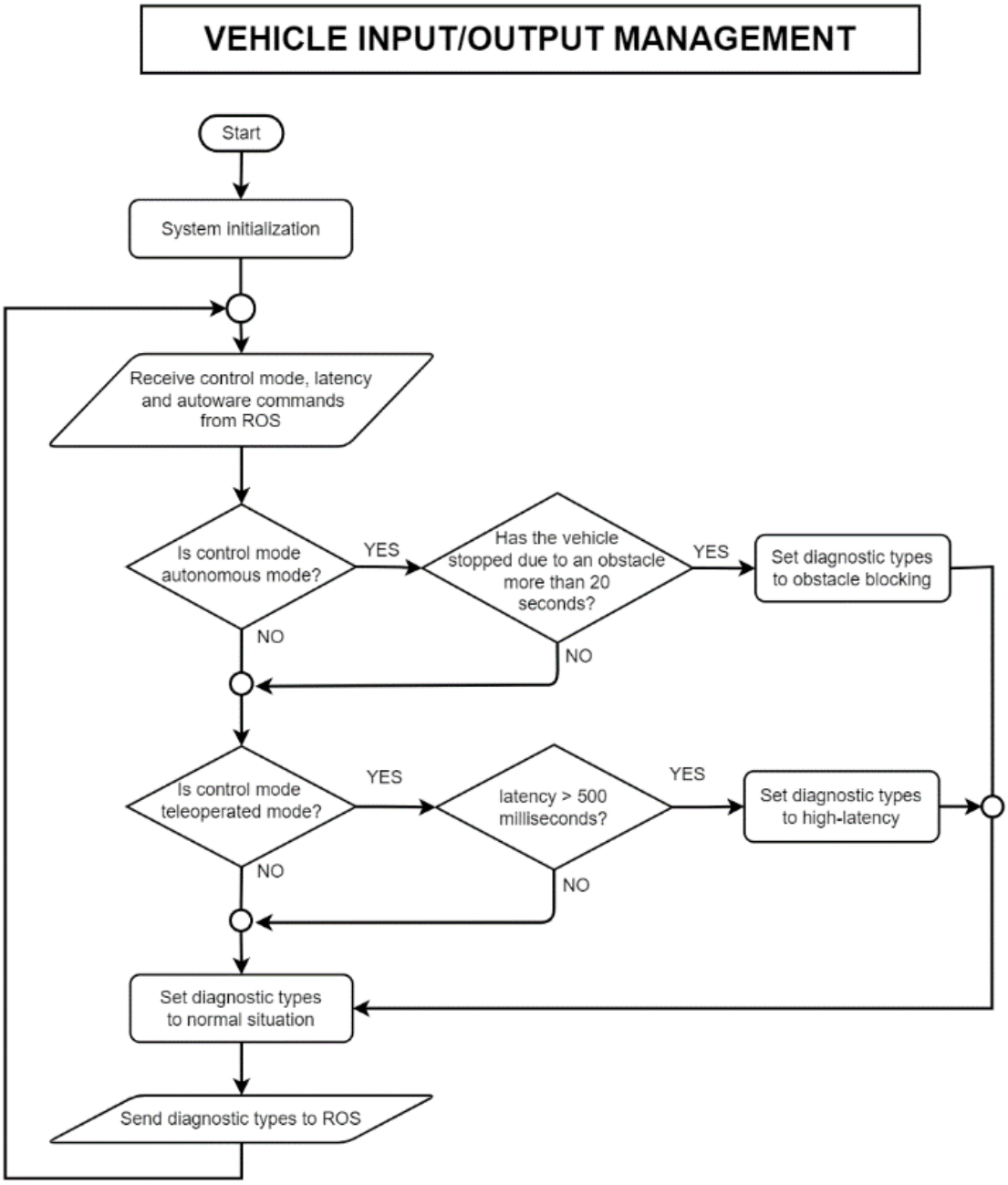


ภาพที่ 81 *ผังการทำงานของแอพพลิเคชั่น Autoware*

11.7 Vehicle I/O Management

- รับค่าความหน่วงเวลาจากโมดูล Latency measurement ผ่านทางระบบ ROS
- รับข้อมูลจาก Autoware ผ่านทางระบบ ROS แปลงค่าอัตราเราให้เหมาะสมกับ current state
- รับ-ส่งค่าโหมดควบคุมของโมดูล Control mode transmission ผ่านทางระบบ ROS
- ประมวลผลข้อมูล
  - หากพารามิเตอร์ทั้งหมดมีค่าดังนี้จะสั่งให้ diagnostic types เปลี่ยนสถานะการวินิจฉัยเป็น **obstacle blocking** ถ้าไม่เป็นไปตามนี้สถานการณ์วินิจฉัยจะเป็น normal situation
    - ค่าโหมดควบคุมอยู่ในโหมดควบคุมด้วยระบบอัตโนมัติ
    - ตรวจสอบพบว่าระบบควบคุมระยะไกลถูกเชื่อมต่อแล้ว
    - สถานะ current_state มีค่าเป็น “Follow”
    - ค่าอัตราเร็วที่ได้จาก Autoware มีค่าเป็น 0 (twist.linear.x = 0)
    - จับเวลาครบ 20 วินาที
  - หากพารามิเตอร์ทั้งหมดมีค่าดังนี้จะสั่งให้ diagnostic types เปลี่ยนสถานะการวินิจฉัยเป็น **high-latency** ถ้าไม่เป็นไปตามนี้สถานการณ์วินิจฉัยจะเป็น normal situation
    - ค่าโหมดควบคุมอยู่ในโหมดควบคุมจากระยะไกล
    - ค่าความหน่วงเวลาที่วัดได้มีค่ามากกว่า 500 มิลลิวินาที
- ส่งข้อมูลค่าโหมดควบคุม และค่าผลการวินิจฉัยทั้งหมดไปยังโมดูล Vehicle mode switcher และโมดูล Control mode transmission ผ่านทางระบบ ROS

ผังการทำงานของโมดูล Vehicle I/O management เป็นไปดังภาพที่ 82

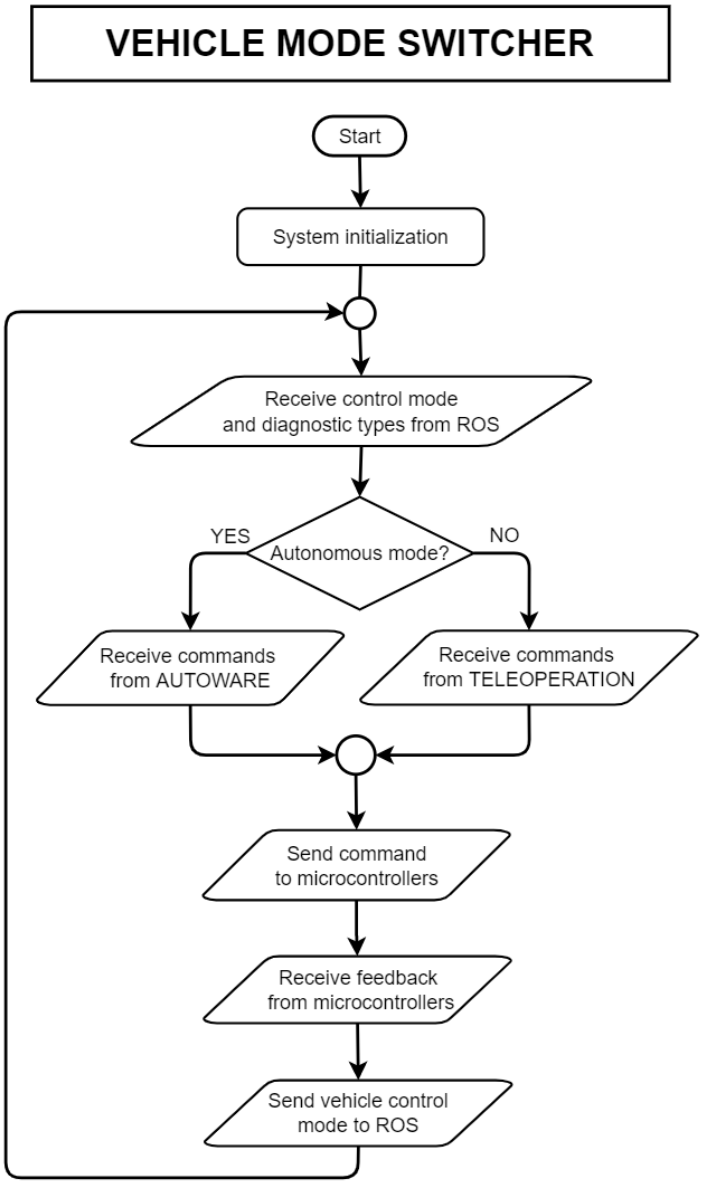


ภาพที่ 82 *ผังการทำงานของ Vehicle I/O management*

11.8 Vehicle Mode Switcher

- รับค่าโหมดควบคุม ผลการวินิจฉัยสถานการผิดปกติ ผ่านทางระบบ ROS
- ทำหน้าที่เลือกช่องสัญญาณที่มีข้อมูลอินพุตจาก Autoware และ Teleoperation จากนั้นต่อช่องสัญญาณที่มีข้อมูลนั้นเข้าเป็นสัญญาณเอาต์พุตเพียงเอาต์พุตเดียวแล้วส่งไปยังโมดูล Cruise microcontroller และ Steering/Light/Button microcontroller ผ่านทางการสื่อสารแบบอนุกรม

ผังการทำงานของ Vehicle mode switcher เป็นไปดังภาพที่ 83

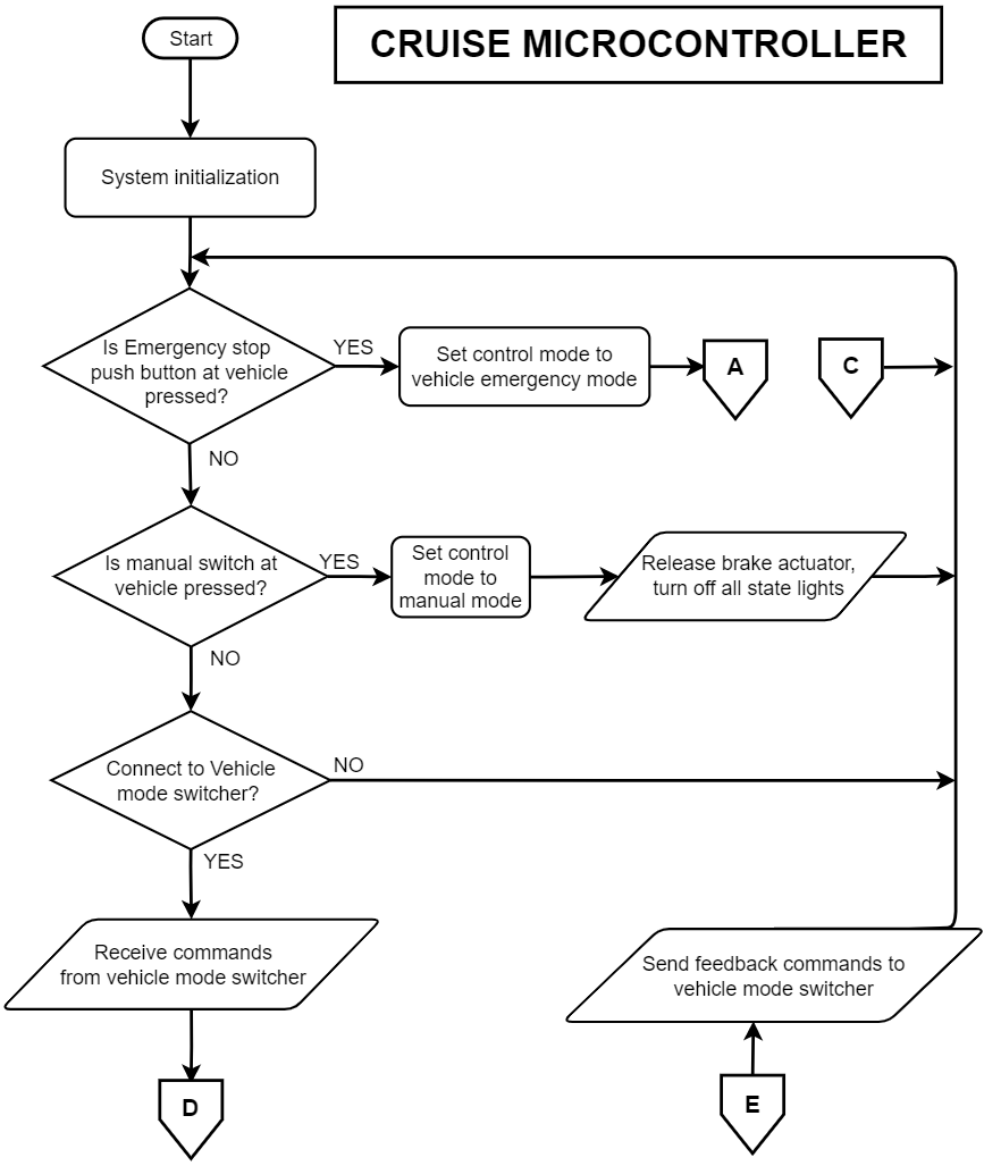


ภาพที่ 83 *ผังการทำงานของ Vehicle mode switcher*

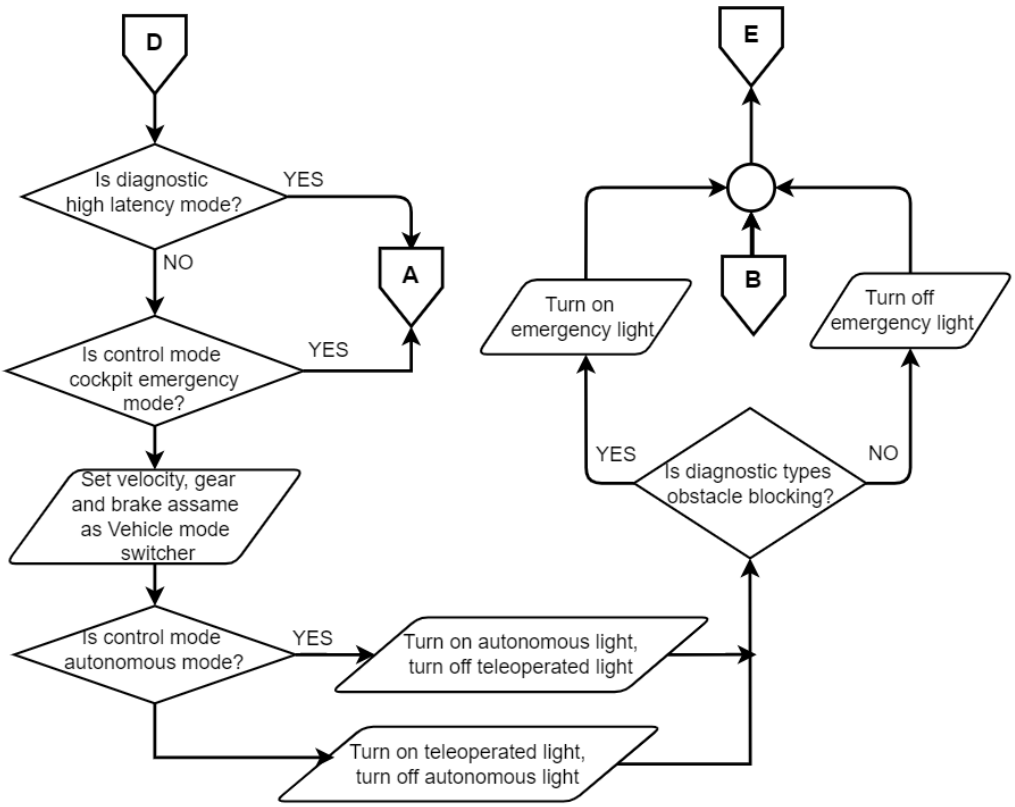
11.9 Cruise microcontroller

- รับคำสั่งควบคุมอัตราเร็ว ค่าเบรก เกียร์ จากโมดูล Vehicle mode switcher ผ่านทางการสื่อสารอนุกรมแล้วส่งไปยังตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้า
- รับค่าโหมดควบคุม ผลการวินิจฉัยสถานการผิดปกติ ผ่านทางระบบ ROS
- รับค่าสัญญาณอัตราเร็วป้อนกลับจากเอ็นโค้ดเดอร์ของรถแล้วส่งเข้าระบบ ROS

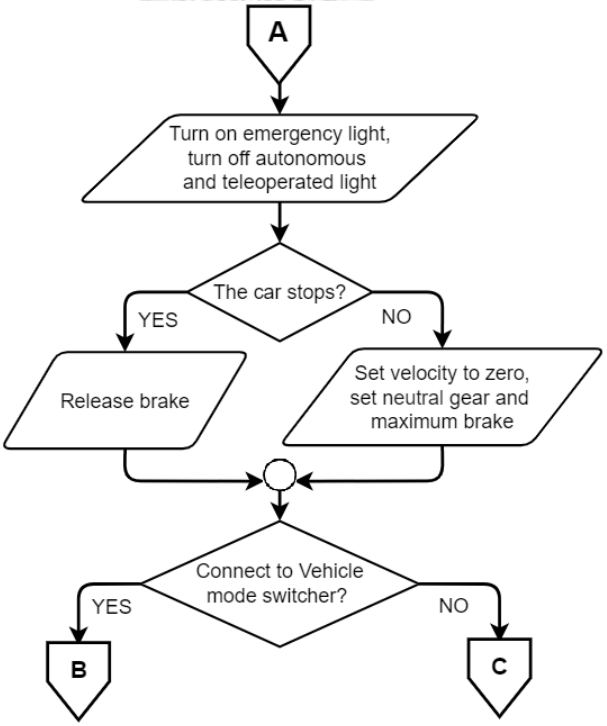
ผังการทำงานของโมดูล Cruise microcontroller เป็นไปดังภาพที่ 84 ถึง 86



ภาพที่ 84 *ผังการทำงานของโมดูล Cruise microcontroller*



ภาพที่ 85 *ผังการทำงานของโมดูล Cruise microcontroller (ต่อ)*



ภาพที่ 86 *ผังการทำงานของโมดูล Cruise microcontroller (ต่อ)*

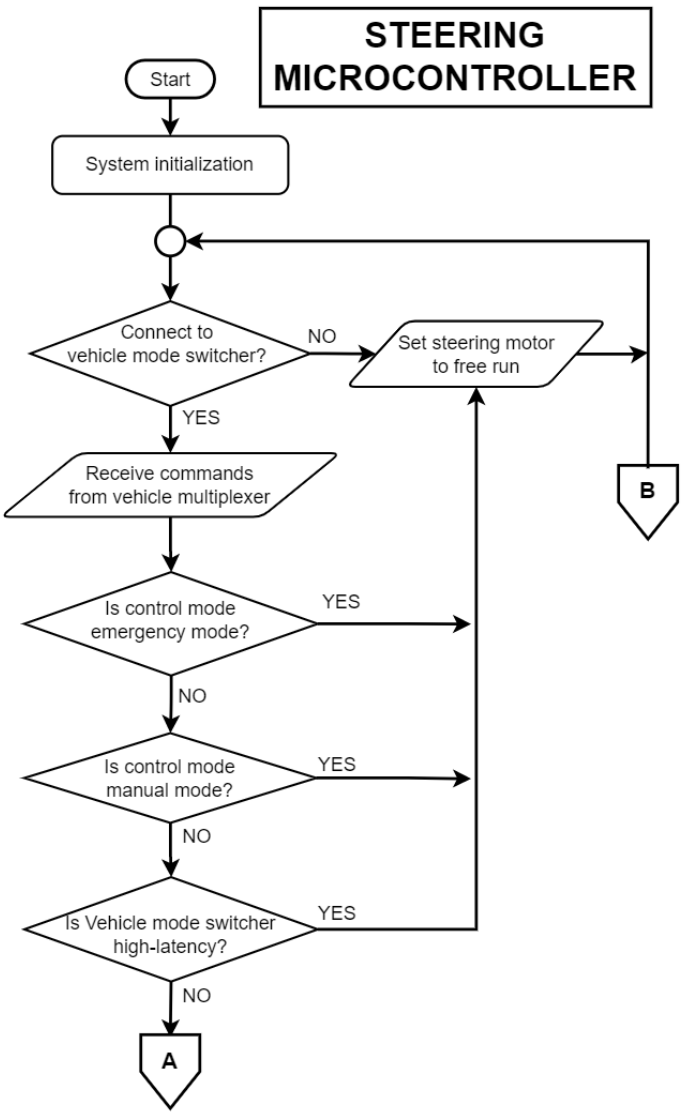
11.10 Steering microcontroller

➢ รับคำสั่งควบคุมพวงมาลัยจากโมดูล Vehicle mode switcher ผ่านทางการสื่อสารอนุกรมแล้วส่งไปยังตัวกระตุ้นสั่งงานด้วยสัญญาณทางไฟฟ้า

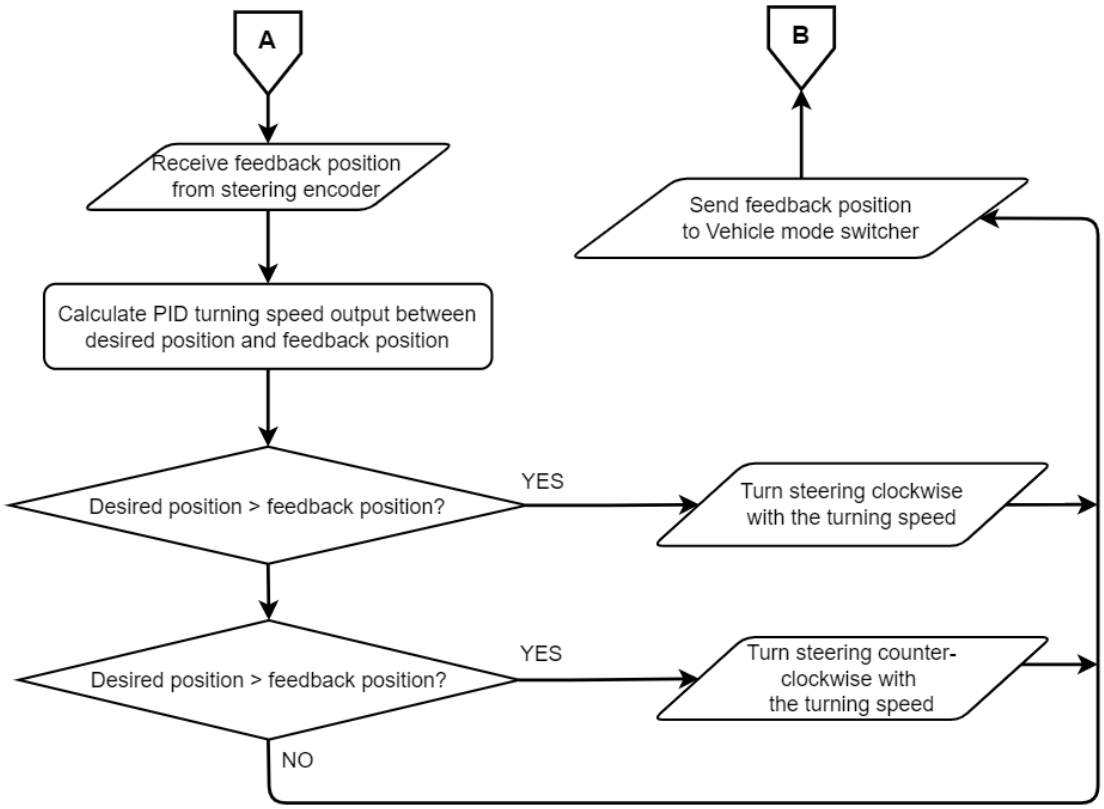
➢ รับค่าโหมดควบคุม ผลการวินิจฉัยสถานการผิดปกติจากระบบ ROS และแสดงผลค่าสีไฟสถานะควบคุมของ Tower light

➢ รับค่าสัญญาณป้อนกลับพวงมาลัยของรถ สถานะปุ่มฉุกเฉิน และปุ่มควบคุมด้วยมือแล้วจึงส่งเข้าระบบ ROS

ผังการทำงานของโมดูล Steering microcontroller เป็นไปดังภาพที่ 87 และ 88



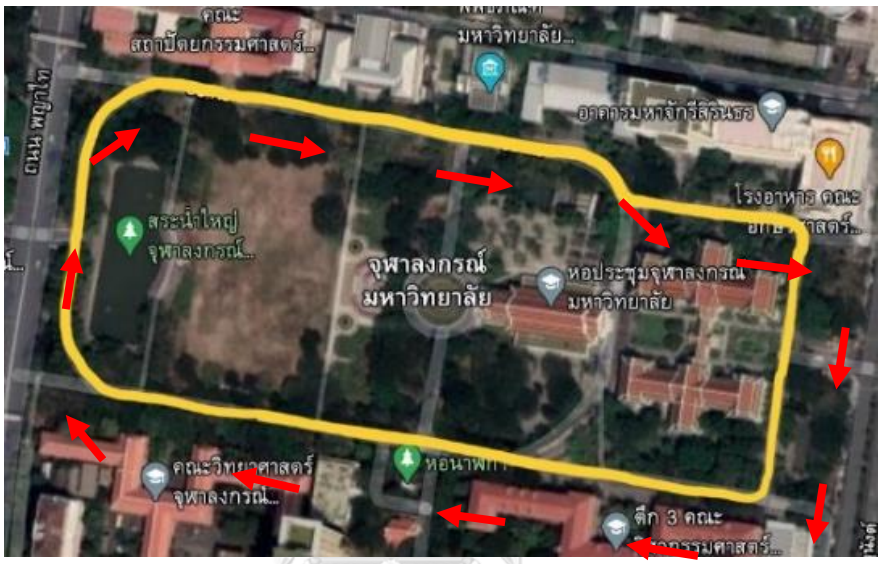
ภาพที่ 87 *ผังการทำงานของโมดูล Steering microcontroller*



ภาพที่ 88 *ผังการทำงานของโมดูล Steering microcontroller (ต่อ)*

12. การดำเนินการทดลอง

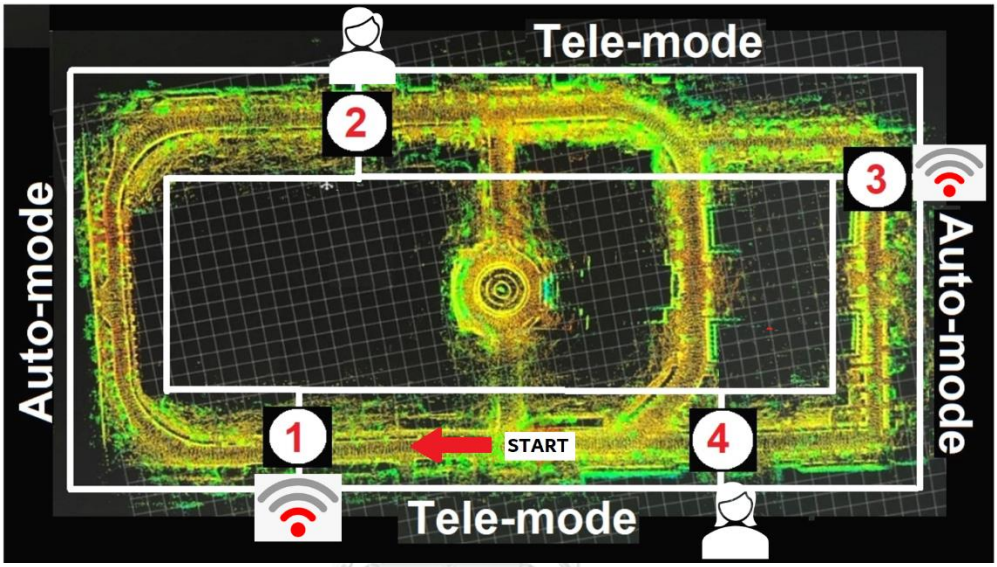
งานวิจัยนี้เมื่อได้พัฒนารถต้นแบบแล้ว การเตรียมการทดสอบการใช้งานระบบเปลี่ยนผ่านระหว่างระบบขับขี่อัตโนมัติและระบบควบคุมระยะไกลจะทดสอบบนพื้นที่บริเวณจุฬาลงกรณ์มหาวิทยาลัย เส้นทางที่ใช้ในการทดสอบจะมีลักษณะเป็นวันเวย์ ทิศทางตามเข็มนาฬิกาดังภาพที่ 89



ภาพที่ 89 *พื้นที่ทำการทดสอบหาค่าความหน่วงของสัญญาณ ณ บริเวณจุฬาลงกรณ์มหาวิทยาลัย*

อัตราเร็วที่ใช้ในการทดลองกำหนดไม่เกิน 15 กิโลเมตร/ชั่วโมง ช่วงใช้งานด้วยระบบอัตโนมัติรถจะวิ่งตามเส้นทางที่ถูกบันทึกไว้ก่อนหน้า ลักษณะการวิ่งของรถจะวิ่งชิดเส้นเลนซ้ายตลอดการทดสอบ รวมถึงช่วงการใช้งานด้วยระบบควบคุมระยะไกลเพื่อให้ผู้ควบคุมสามารถใช้เส้นเลนที่มองเห็นจากหน้าจอมอนิเตอร์เป็นเส้นอ้างอิงของรถ การทดสอบระบบเปลี่ยนผ่านการควบคุมแบ่งเป็น 2 กรณี ดังภาพที่ 90 คือ กรณีแรกขณะรถทดสอบอยู่ในช่วงการใช้งานระบบอัตโนมัติและระบบตรวจพบว่ามีสิ่งกีดขวาง ซึ่งอาจจะเป็นรถหรือวัตถุใดๆ ในลักษณะหยุดนิ่งอยู่ด้านหน้าในช่องทางเดียวกัน รถทดสอบจะทำการเบรกจนหยุดนิ่ง หากสิ่งกีดขวางไม่มีการขยับออกให้พ้นระยะด้านหน้าและเวลากำหนด ระบบจะแจ้งเตือนไปยังสถานีควบคุมโดยการเปิดสัญญาณไฟสีแดงของ tower light ที่อยู่ ณ สถานีควบคุม เพื่อให้ผู้ควบคุมทำการเปลี่ยนโหมดมาเป็นระบบควบคุมระยะไกลเพื่อแก้ไขสถานการณ์ กรณีที่สองขณะรถทดสอบอยู่ในช่วงการใช้งานระบบควบคุมระยะไกลระบบตรวจพบว่ามีค่าความหน่วงเวลาสูงเกินกำหนด ระบบจะแจ้งเตือนไปยังสถานีควบคุมโดยการเปิดสัญญาณไฟสีแดง ณ สถานีควบคุม เพื่อให้ผู้ควบคุมทำการเปลี่ยนโหมดมาเป็นการขับขี่ด้วยระบบอัตโนมัติ จุดทดสอบ

การเปลี่ยนระบบควบคุมแบ่งออกเป็น 4 จุด ในระหว่างการทดสอบ ผู้ควบคุม ณ สถานีควบคุมสามารถรับรู้ช่วงความหน่วงได้ตลอดเวลาจากการสังเกตสีของ latency indicator อีกทั้งยังสามารถปรับเปลี่ยนโหมดการควบคุมและปุ่มหยุดฉุกเฉินจากระยะไกลได้ทุกช่วงเพื่อความมั่นใจระหว่างการทดสอบ



ภาพที่ 90 *จุดทดสอบการเปลี่ยนการควบคุมระหว่างระบบอัตโนมัติและระบบควบคุมระยะไกล*

เพื่อให้ผลการทดสอบเห็นผลได้ชัดยิ่งขึ้น งานวิจัยนี้ได้ทำการเทียบผลการเบี่ยงเบนเชิงเส้นระหว่างเส้นทางที่ถูกบันทึกขณะทดสอบและเส้นทางอ้างอิง และหาผลตอบสนองสัญญาณป้อนกลับของพวงมาลัยและอัตราเร็วของรถทดสอบ เทียบกับสัญญาณควบคุมที่ส่งจากชุดควบคุม Logitech G29 และจากแอพพลิเคชั่น Autoware ณ เวลาเดียวเดียวกัน ผลการตอบสนองดังกล่าวจะถูกระบุลงในพิกัดแผนที่พร้อมกับค่าความหน่วงเวลาของการส่งสัญญาณ ณ ขณะนั้น โดยใช้เครื่องมือระบุตำแหน่งบนพื้นโลก (Global Positioning System – GPS) รุ่น U-blox FED-F9P

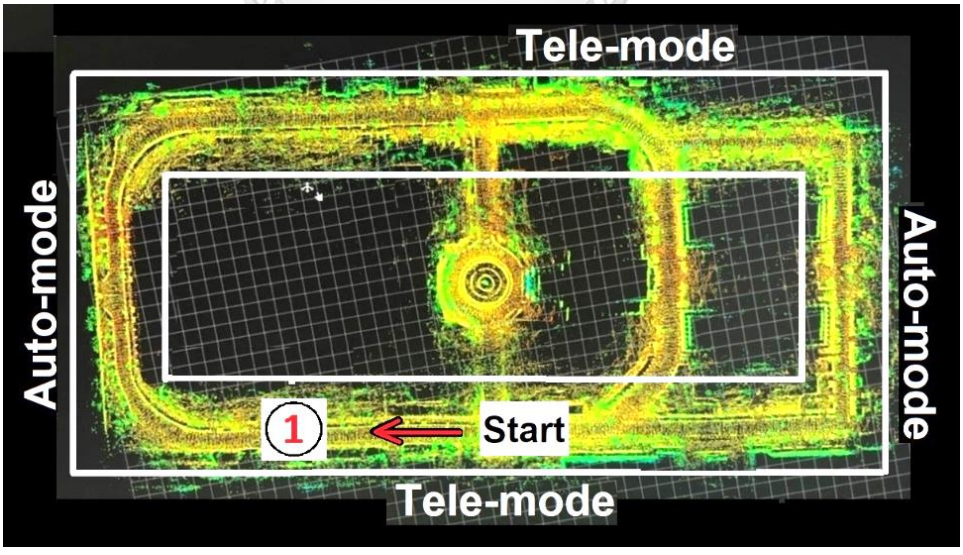
การวัดค่าความหน่วงเวลาแบบ Round Trip Time (RTT) ของการส่งข้อมูลระหว่างสถานีควบคุมกับรถทดสอบ โดยใช้โปรแกรมที่เขียนขึ้นด้วยภาษาไพทอน (Python) ในการวัดค่าความหน่วงเวลาของการส่งข้อมูลของทั้งสองคอมพิวเตอร์ ซึ่งคอมพิวเตอร์จำเป็นต้องถูกตั้งเวลาให้มีค่าใกล้เคียงกันมากที่สุด (time synchronization) ด้วยวิธี Chronyd ซึ่งเป็นการ sync เวลากับทาง NTP server ผ่านระบบ Network Time Protocol (NTP) โปรโตคอลนี้ถูกใช้ในการตั้งค่าเวลาบนคอมพิวเตอร์จากการเทียบกับเวลามาตรฐานผ่านระบบเครือข่ายซึ่งให้ความแม่นยำค่อนข้างสูง

บทที่ 4
ผลการดำเนินการวิจัย

1. ผลทดสอบระบบเปลี่ยนผ่านการควบคุม

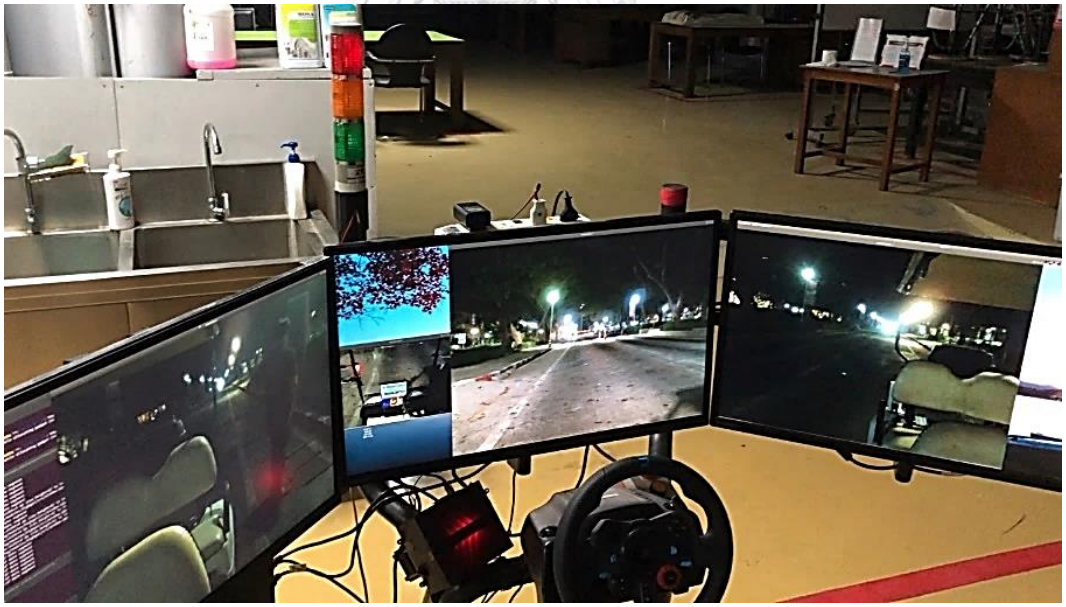
1.1 ผลการเปลี่ยนผ่านจากระบบควบคุมระยะไกลเป็นการควบคุมด้วยระบบอัตโนมัติ ณ บริเวณจุดที่ 1

การทดสอบเริ่มด้วยการควบคุมระยะไกลจากจุดเริ่มต้นไปยังจุดที่ 1 ดังภาพที่ 91 เมื่อถึงจุดที่ 1 ผู้ควบคุมทำการส่งค่าความหน่วงของสัญญาณสมมุติขนาด 500 มิลลิวินาที จากสถานีควบคุมไปยังรถทดสอบ เนื่องจากการทดสอบจริงค่าความหน่วงของสัญญาณที่วัดได้มีค่าต่ำกว่าที่ระบบกำหนดเพื่อเข้าสู่โหมดฉุกเฉิน ภาพที่ 92 และ 93 หลังจากระบบเข้าสู่โหมดฉุกเฉิน รถทดสอบสามารถเบรกจนถึงจุดหยุดนิ่งด้วยระบบอัตโนมัติ ไฟสถานะทั้งรถทดสอบและสถานีควบคุมจะติดเป็นไฟสีแดงเพื่อแจ้งเตือนผู้ควบคุมให้รับรู้สถานะของระบบและเข้าเปลี่ยนโหมดควบคุม จากการทดลองจะพบว่า ณ จุดดังกล่าวหลอดไฟ LED แสดงช่วงความหน่วงของสัญญาณ (บริเวณด้านซ้ายของพวงมาลัย) จะติดเป็นสีแดงเนื่องจากช่วงความหน่วงที่มากกว่า 500 มิลลิวินาที ด้วยเช่นกัน



ภาพที่ 91 *เส้นทางการทดสอบจากจุดเริ่มต้นไปยังจุดที่จะทำการเปลี่ยนจากโหมดควบคุมระยะไกลเป็นโหมดควบคุมด้วยระบบอัตโนมัติ ณ จุดที่ 1*

ภาพที่ 92 *ระบบเข้าสู่โหมดฉุกเฉินขณะทำการควบคุมจากระยะไกล ณ จุดที่ 1 (รถทดสอบ)*



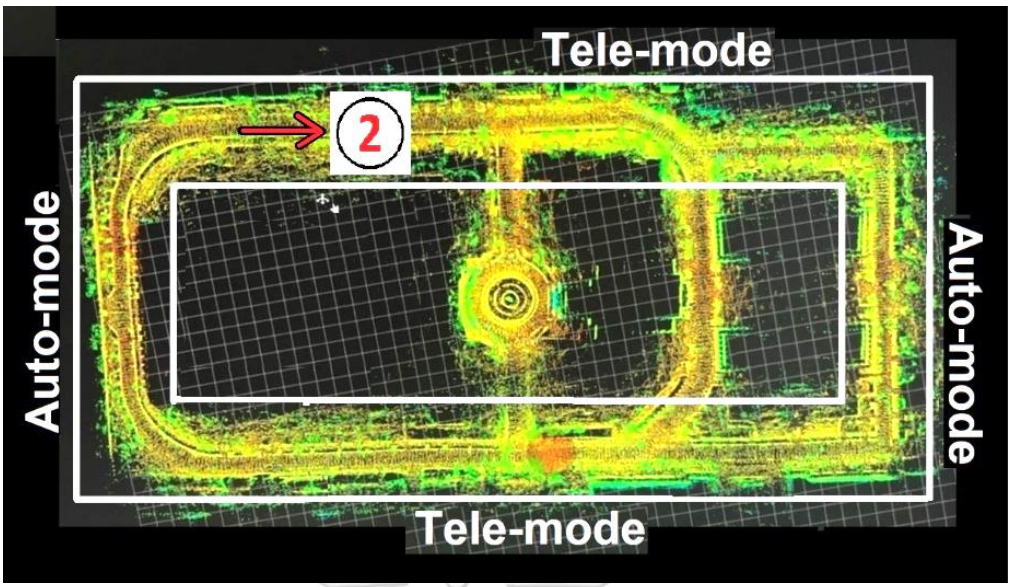
ภาพที่ 93 *ระบบเข้าสู่โหมดฉุกเฉินขณะควบคุมจากระยะไกล ณ จุดที่ 1 (สถานีควบคุม)*

หลังจากผู้ควบคุม ณ สถานีควบคุมทำการกดปุ่มเปลี่ยนโหมดจากการควบคุมระยะไกลเป็นการควบคุมด้วยระบบอัตโนมัติ ไฟแสดงสถานะจะติดเป็นสีส้มและขับขี่ต่อด้วยระบบอัตโนมัติทันที ดังภาพที่ 94

ภาพที่ 94 *ระบบเข้าสู่โหมดควบคุมด้วยระบบอัตโนมัติ ณ จุดที่ 1 (รถทดสอบ)*

1.2 ผลการเปลี่ยนผ่านจากระบบควบคุมอัตโนมัติเป็นการควบคุมจากระยะไกล ณ บริเวณจุดที่ 2

รถทดสอบอยู่ในโหมดควบคุมด้วยระบบอัตโนมัติจนถึงจุดที่ 2 ดังภาพที่ 95 ณ จุดนี้จะทดสอบโดยใช้มนุษย์แทนสิ่งกีดขวาง ซึ่งขวางอยู่บริเวณด้านหน้าของรถทดสอบ เมื่อระบบตรวจพบสิ่งกีดขวาง รถทดสอบสามารถเบรกจนถึงจุดหยุดนิ่ง และจับเวลา 20 วินาที หากสิ่งกีดขวางไม่ขยับออกจากเส้นทางทดสอบภายในเวลาดังกล่าว ระบบจะเข้าสู่โหมดฉุกเฉิน ไฟสถานะของรถทดสอบและสถานีควบคุมจะติดเป็นสีแดง ดังภาพที่ 96 และ 97



ภาพที่ 95 *จุดทำการเปลี่ยนจากโหมดควบคุมด้วยระบบอัตโนมัติเป็นโหมดการควบคุมจากระยะไกล ณ จุดที่ 2*

ภาพที่ 96 *ระบบเข้าสู่โหมดฉุกเฉินขณะพบสิ่งกีดขวางระหว่างการควบคุมด้วยระบบอัตโนมัติ ณ จุดที่ 2 (รถทดสอบ)*

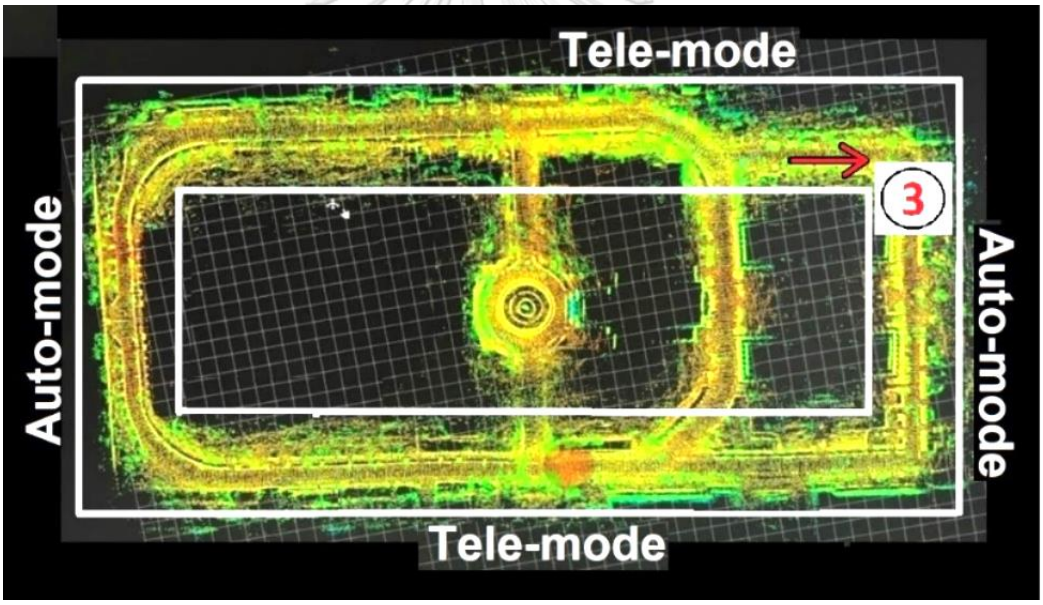
ภาพที่ 97 *ระบบเข้าสู่โหมดฉุกเฉินขณะควบคุมด้วยระบบอัตโนมัติ ณ จุดที่ 2 (สถานีควบคุม)*

หลังจากผู้ควบคุม ณ สถานีควบคุมทำการตรวจเช็คค่าความหน่วงของสัญญาณจากสีของหลอดไฟ LED เพื่อความปลอดภัยก่อนเปลี่ยนโหมด พบว่าความหน่วงของสัญญาณต่ำกว่า 100 มิลลิวินาที (สีเขียว) หลังจากนั้นผู้ควบคุมกดปุ่มเปลี่ยนโหมดจากระบบควบคุมอัตโนมัติเป็นการควบคุมจากระยะไกล ไฟแสดงสถานะของระบบจะติดเป็นสีเขียว ผู้ควบคุม ณ สถานีควบคุมเข้าควบคุมรถทดสอบและขับไปยังจุดถัดไป ดังภาพที่ 98

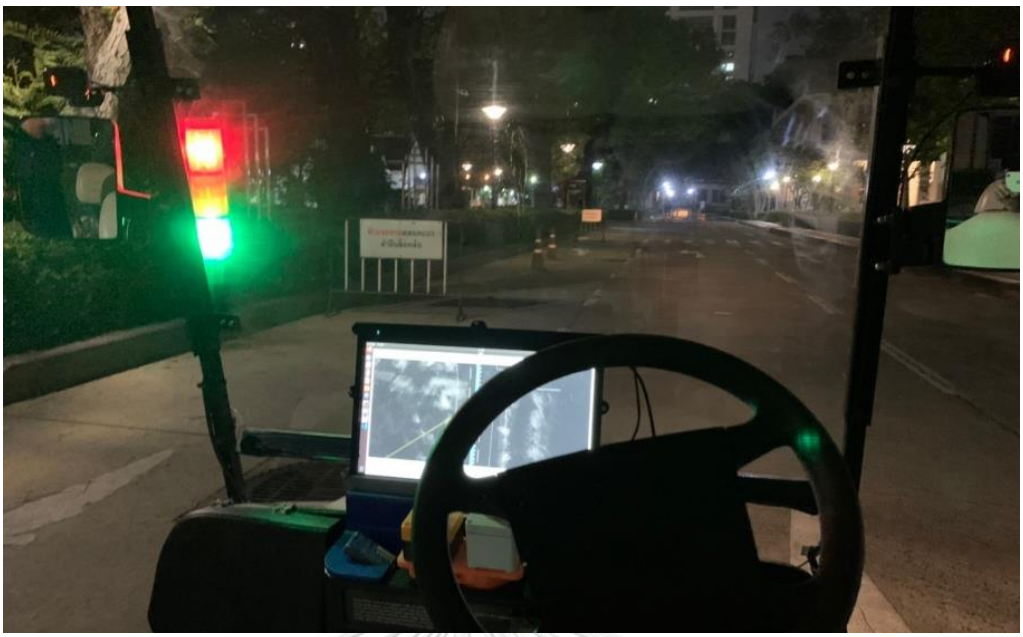
ภาพที่ 98 *ระบบเข้าสู่โหมดควบคุมจากระยะไกล ณ จุดที่ 2 (รถทดสอบ)*

1.3 ผลการเปลี่ยนผ่านจากระบบควบคุมระยะไกลเป็นการควบคุมด้วยระบบอัตโนมัติ ณ บริเวณจุดที่ 3

รถทดสอบอยู่ในโหมดควบคุมจากระยะไกลจนถึงจุดที่ 3 ดังภาพที่ 99 เมื่อถึงจุดดังกล่าว งานวิจัยนี้ทำการส่งค่าความหน่วงของสัญญาณสมมุติขนาด 500 มิลลิวินาที เช่นเดียวกับการทดลอง ณ จุดที่ 1 เมื่อระบบเข้าสู่โหมดฉุกเฉิน รถทดสอบสามารถเบรกจนถึงจุดหยุดนิ่งด้วยระบบอัตโนมัติ ไฟสถานะทั้งรถทดสอบและสถานีควบคุมจะติดเป็นไฟสีแดงเพื่อแจ้งเตือนผู้ควบคุมให้รับรู้สถานะของระบบและทำการเปลี่ยนโหมดการควบคุม ดังภาพที่ 100 และ 101 จากการทดลองจะพบว่า ณ จุดดังกล่าวไฟแสดงช่วงความหน่วงของสัญญาณ (บริเวณด้านซ้ายของพวงมาลัย) จะติดเป็นสีแดง เนื่องจากความหน่วงดังกล่าว



ภาพที่ 99 *จุดทำการเปลี่ยนโหมดจากโหมดควบคุมจากระยะไกลเป็นโหมดควบคุมด้วยระบบอัตโนมัติ ณ จุดที่ 3*



ภาพที่ 100 *ระบบเข้าสู่โหมดฉุกเฉินขณะทำการควบคุมจากระยะไกล ณ จุดที่ 3 (รถทดสอบ)*

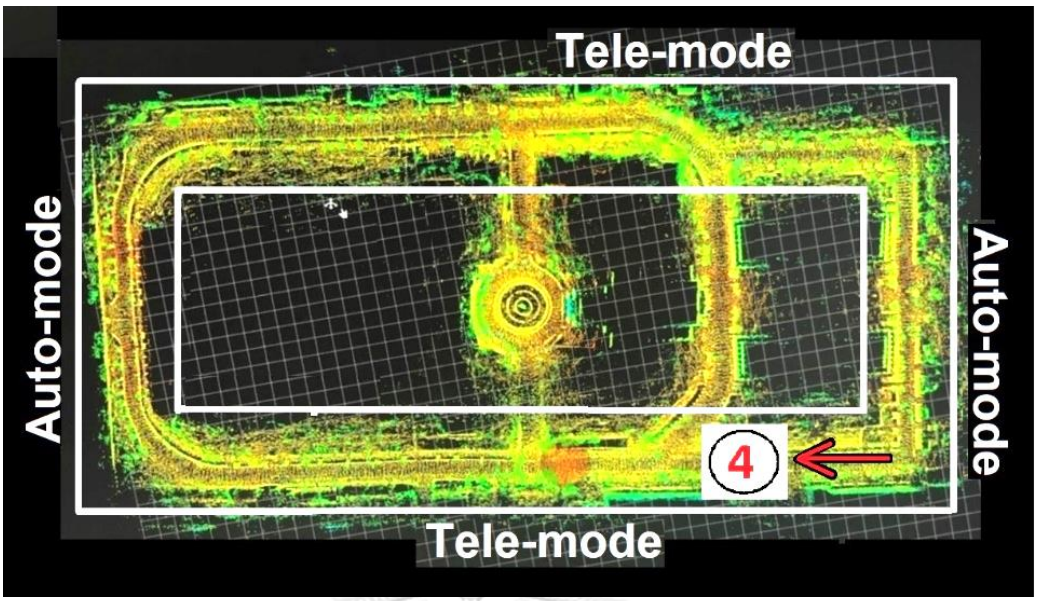
ภาพที่ 101 *ระบบเข้าสู่โหมดฉุกเฉินขณะทำการควบคุมจากระยะไกล ณ จุดที่ 3 (สถานีควบคุม)*

หลังจากผู้ควบคุม ณ สถานีควบคุมทำการกดปุ่มเปลี่ยนโหมดจากการควบคุมระยะไกลเป็นการควบคุมด้วยระบบอัตโนมัติ ไฟแสดงสถานะจะติดเป็นสีส้มและขับขี่ต่อด้วยระบบอัตโนมัติทันที ดังภาพที่ 102

ภาพที่ 102 *ระบบเข้าสู่โหมดควบคุมด้วยระบบอัตโนมัติ ณ จุดที่ 3 (รถทดสอบ)*

1.4 ผลการเปลี่ยนผ่านจากระบบควบคุมอัตโนมัติเป็นการควบคุมจากระยะไกล ณ บริเวณจุดที่ 4

รถทดสอบอยู่ในโหมดควบคุมด้วยระบบอัตโนมัติจนถึงจุดที่ 2 ดังภาพที่ 103 ณ จุดนี้จะทดสอบโดยใช้มนุษย์แทนสิ่งกีดขวาง ซึ่งขวางอยู่บริเวณด้านหน้าของรถทดสอบ เมื่อระบบตรวจพบสิ่งกีดขวาง รถทดสอบสามารถเบรกจนถึงจุดหยุดนิ่ง และจับเวลา 20 วินาที หากสิ่งกีดขวางไม่ขยับออกจากเส้นทางทดสอบภายในเวลาดังกล่าว ระบบจะเข้าสู่โหมดฉุกเฉิน ไฟสถานะของรถทดสอบและสถานีควบคุมจะติดเป็นสีแดง ดังภาพที่ 104 และ 105



ภาพที่ 103 *จุดทำการเปลี่ยนจากโหมดควบคุมด้วยระบบอัตโนมัติเป็นโหมดการควบคุมจากระยะไกล ณ จุดที่ 4*

ภาพที่ 104 *ระบบเข้าสู่โหมดฉุกเฉินขณะพบสิ่งกีดขวางระหว่างการควบคุมด้วยระบบอัตโนมัติ ณ จุดที่ 4 (รถทดสอบ)*

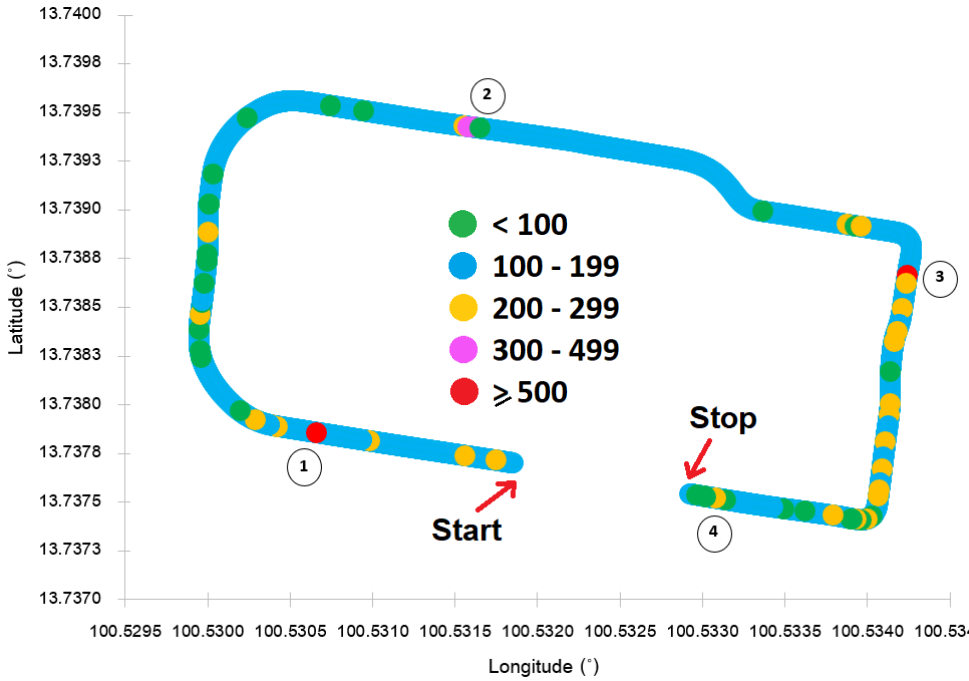
ภาพที่ 105 *ระบบเข้าสู่โหมดฉุกเฉินขณะพบสิ่งกีดขวางระหว่างการควบคุมด้วยระบบอัตโนมัติ ณ จุดที่ 4 (สถานีควบคุม)*

ผู้ควบคุม ณ สถานีควบคุมตรวจเช็คค่าความหน่วงของสัญญาณจากสีของหลอดไฟ LED ก่อนเปลี่ยนโหมดว่าอยู่ในเกณฑ์ที่ปลอดภัย จากนั้นผู้ควบคุมกดปุ่มเปลี่ยนโหมดจากระบบควบคุมอัตโนมัติเป็นการควบคุมจากระยะไกล ไฟแสดงสถานะของระบบจะติดเป็นสีเขียว ผู้ควบคุม ณ สถานีควบคุมเข้าควบคุมรถทดสอบและขับไปยังจุดถัดไป ดังภาพที่ 106

ภาพที่ 106 *ระบบเข้าสู่โหมดควบคุมจากระยะไกล ณ จุดที่ 4 (รถทดสอบ)*

2. ผลการทดสอบวัดความหน่วงเวลาเทียบกับพิกัดแผนที่ขณะทดสอบระบบเปลี่ยนผ่าน

ค่าความหน่วงเวลาขณะทดสอบเทียบกับตำแหน่งพิกัด ละติจูด และลองจิจูด ของแผนที่เป็นไปดังภาพที่ 107 ช่วงสีที่แสดงดังภาพถูกแบ่งตามสีของหลอดไฟ LED แสดงความหน่วงเวลา ณ สถานีควบคุม



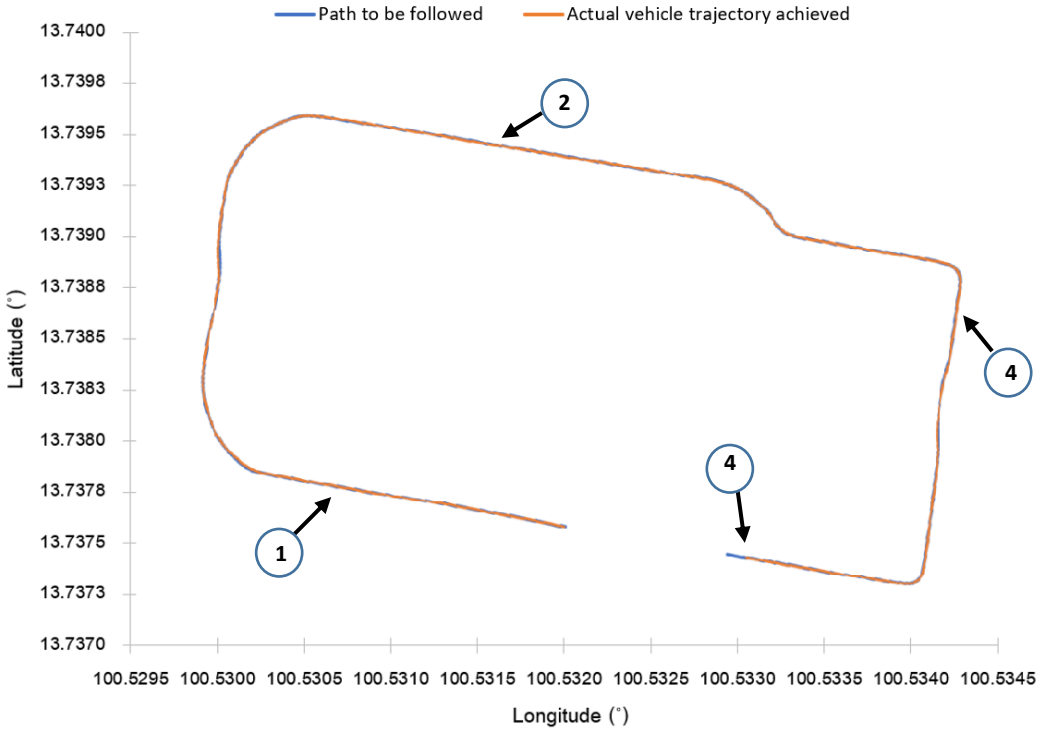
ภาพที่ 107 ความหน่วงเวลาขณะทดสอบเทียบกับพิกัดแผนที่

ผลการวัดค่าความหน่วงเวลาตลอดการทดสอบ พบว่ามีค่าเฉลี่ยและสูงสุดอยู่ที่ 109.7 และ 316.65 มิลลิวินาที ตามลำดับ ส่วนค่าความหน่วงสมมุติที่มากกว่า 500 มิลลิวินาที (สีแดง) ถูกใช้เพื่อทดสอบระบบเฝ้าระวังสถานการณ์ฉุกเฉินขณะควบคุมจากระยะไกล เพื่อให้รถทดสอบเข้าสู่โหมดฉุกเฉินและหยุดจอดในบริเวณที่เหมาะสมตามที่ตำแหน่งที่ออกแบบไว้เพื่อความปลอดภัย ณ จุดที่ 1 และ 3 ค่าความหน่วงเวลานี้จะไม่ถูกนำมาพิจารณารวมกับค่าความหน่วงเวลาที่วัดได้จริง

3. ผลทดสอบหาตำแหน่งของเส้นทางจริงที่บันทึกจากรถทดสอบเทียบกับเส้นทางอ้างอิงขณะทดสอบระบบเปลี่ยนผ่าน

รถอัตโนมัติใช้เส้นทางอ้างอิงที่มีลักษณะเป็นจุดต่อ ๆ กัน (Way point) ซึ่งถูกสร้างไว้ก่อนหน้าในการนำทางให้กับรถอัตโนมัติ งานวิจัยนี้ได้มีการสร้างเส้นทางอ้างอิงดังกล่าวจากการใช้รถ

ทดสอบเก็บตำแหน่งจริงบนถนน ภาพที่ 108 แสดงถึงผลการทดสอบเส้นทางจริงของระบบอัตโนมัติและระบบควบคุมระยะไกลที่ถูกบันทึกขณะทำการทดสอบเทียบกับเส้นทางอ้างอิง ซึ่งรถทดสอบสามารถวิ่งตามเส้นทางอ้างอิงที่ถูกกำหนดได้ตลอดการทดสอบ

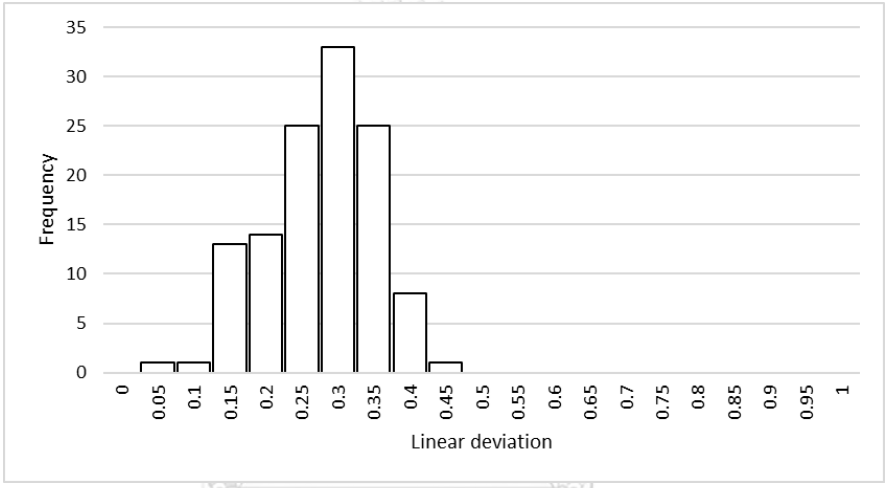


ภาพที่ 108 *เส้นทางจริงของรถทดสอบขณะทดสอบเทียบกับเส้นทางอ้างอิง*

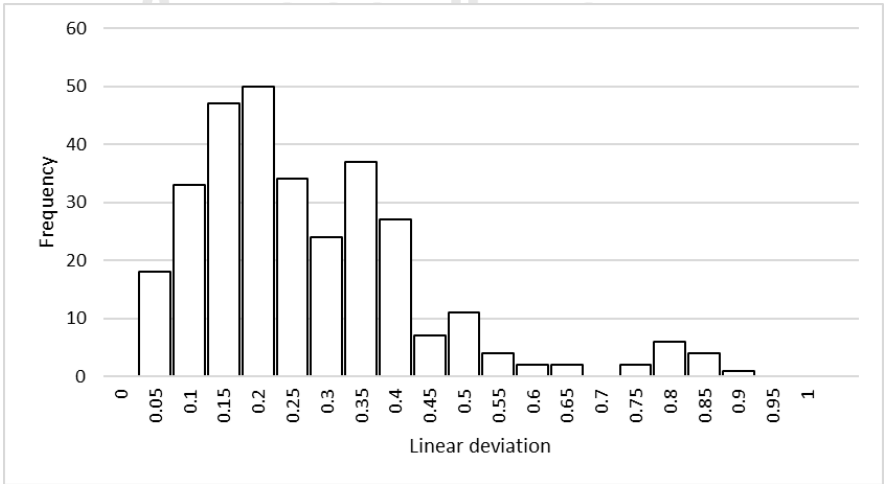
ค่าความคลาดเคลื่อนระหว่างเส้นทางอ้างอิงและเส้นทางจริงสามารถหาได้จากการเบี่ยงเบนเชิงเส้น (Linear deviation) โดยแบ่งผลการทดสอบออกเป็น 4 ช่วง ดังภาพที่ 109 ถึง 112 ช่วงใช้งานในโหมดควบคุมจากระยะไกลจากจุดเริ่มต้นไปยังจุดที่ 1 ค่าเบี่ยงเบนเชิงเส้นเฉลี่ยและสูงสุดอยู่ที่ 0.25 และ 0.41 เมตร ค่าสูงสุดที่เกิดขึ้นขณะทดสอบช่วงนี้มีค่าต่ำกว่าค่าเบี่ยงเบนเชิงเส้นสูงสุดของช่วงการทดสอบอื่น ๆ ลักษณะเส้นทางเป็นทางตรง ผู้ควบคุมสามารถควบคุมได้ง่ายกว่าทางโค้ง ถัดมาระบบเข้าสู่ช่วงการใช้งานด้วยระบบอัตโนมัติจากจุด 1 ไปยังจุดที่ 2 ค่าเบี่ยงเบนเชิงเส้นเฉลี่ยและสูงสุดอยู่ที่ 0.25 และ 0.88 เมตร ช่วงการทดสอบนี้ มีจุดเลี้ยวโค้งและทางร่วมแยกหลายช่วง รถทดสอบสั่งเปลี่ยนตำแหน่งพวงมาลัยหลายครั้ง ค่าเบี่ยงเบนสูงสุดของการทดสอบช่วงนี้มีค่าสูงที่สุดเมื่อเทียบกับค่าเบี่ยงเบนสูงสุดของช่วงการทดสอบอื่น ๆ ถัดมารถทดสอบอยู่ในช่วงการใช้งานด้วยระบบควบคุมจากระยะไกลจากจุด 2 ไปยังจุดที่ 3 ค่าเบี่ยงเบนเชิงเส้นเฉลี่ยอยู่ที่ 0.19 เมตร ค่าเฉลี่ย

ของการทดสอบช่วงนี้มีค่าต่ำที่สุดเมื่อเทียบกับค่าเฉลี่ยของการทดสอบอื่น ๆ ส่วนค่าเบี่ยงเบนสูงสุดขณะทดสอบมีค่าอยู่ที่ 0.58 เมตร ถัดมาช่วงใช้งานในโหมดควบคุมด้วยระบบอัตโนมัติจากจุด 3 ไปยังจุดที่ 4 มีค่าเบี่ยงเบนเชิงเส้นเฉลี่ยและสูงสุดอยู่ที่ 0.34 และ 0.83 เมตร

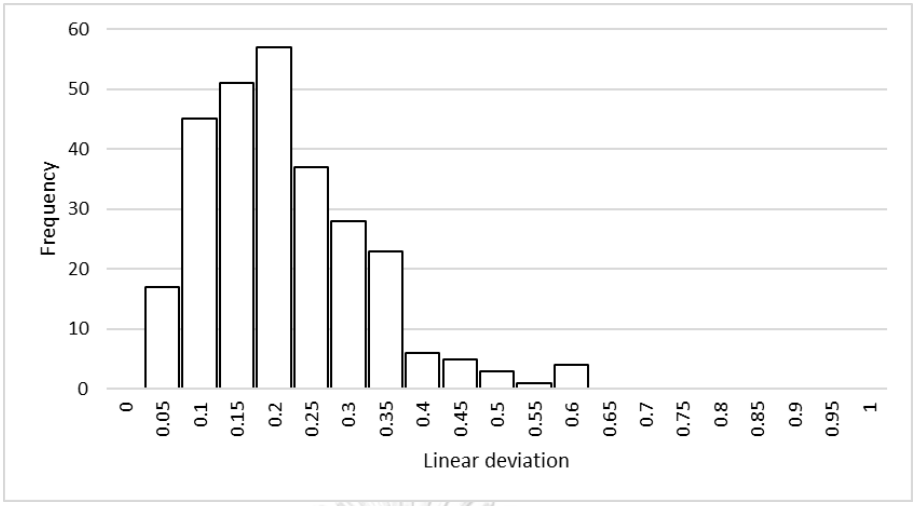
จากผลค่าคาวมคลาดเคลื่อนทั้งหมดจะพบว่าทั้งค่าเฉลี่ยของการเบี่ยงเบนเชิงเส้นขณะควบคุมจากระยะไกลมีค่าต่ำกว่าการควบคุมด้วยระบบอัตโนมัติเพียงเล็กน้อย แต่เมื่อทำการเทียบค่าเบี่ยงเบนสูงสุดจะพบว่า การควบคุมด้วยระบบอัตโนมัติจะมีค่าเบี่ยงเบนสูงสุดมากกว่าการควบคุมจากระยะไกลประมาณ 2 เท่า



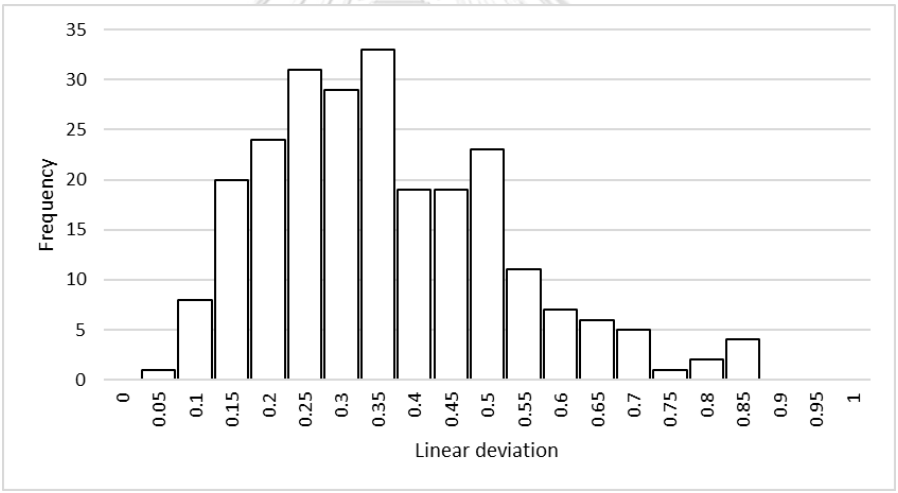
ภาพที่ 109 กราฟแสดงการแจกแจงความถี่ของการเบี่ยงเบนเชิงเส้นในช่วงใช้งานในโหมดควบคุมจากระยะไกล (จากจุดเริ่มต้นไปยังจุดที่ 1)



ภาพที่ 110 กราฟแสดงการแจกแจงความถี่ของการเบี่ยงเบนเชิงเส้นในช่วงใช้งานในโหมดควบคุมด้วยระบบอัตโนมัติ (จากจุด 1 ไปยังจุดที่ 2)



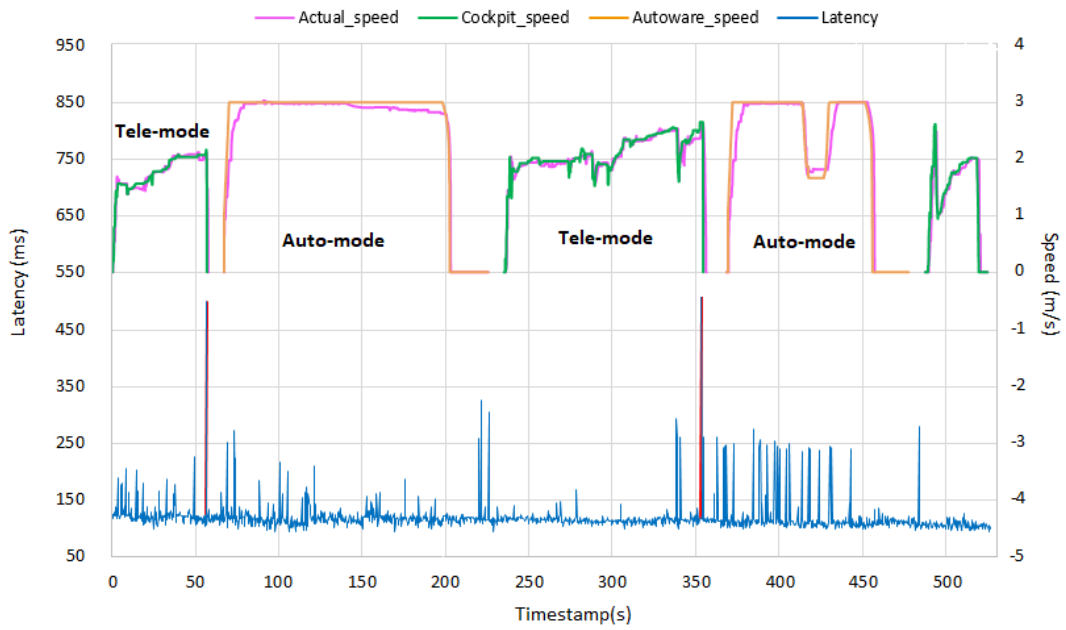
ภาพที่ 111 กราฟแสดงการแจกแจงความถี่ของการเบี่ยงเบนเชิงเส้นในช่วงใช้งานในโหมดควบคุมจากระยะไกล (จากจุด 2 ไปยังจุดที่ 3)



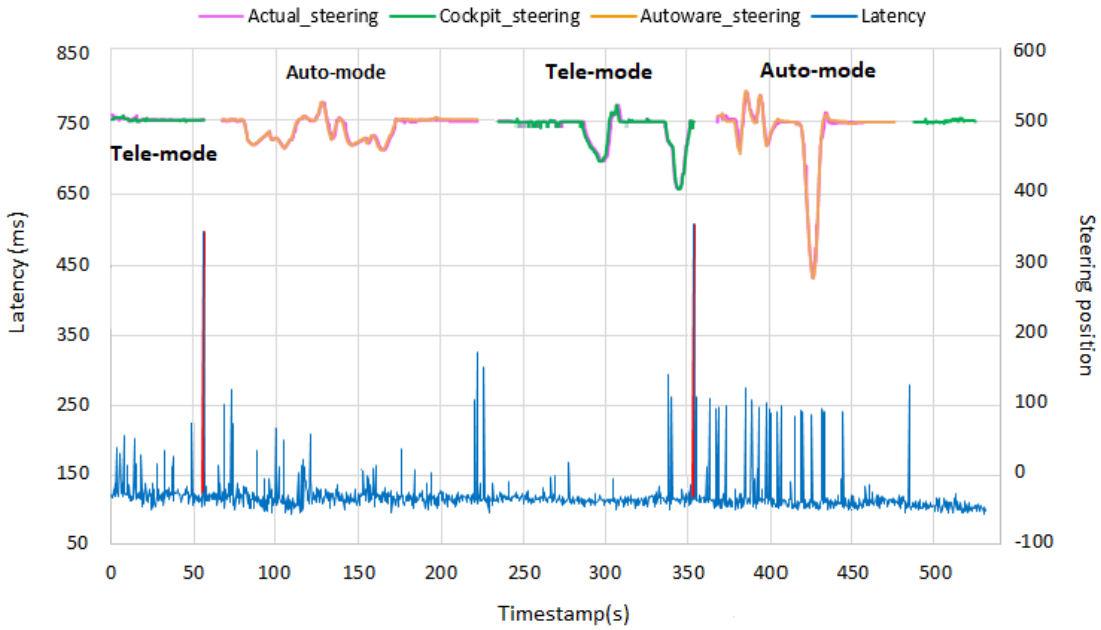
ภาพที่ 112 กราฟแสดงการแจกแจงความถี่ของการเบี่ยงเบนเชิงเส้นในช่วงใช้งานในโหมดควบคุมด้วยระบบอัตโนมัติ (จากจุด 3 ไปยังจุดที่ 4)

4. ผลตอบสนองของอัตราเร็ว และค่าพวงมาลัย เทียบกับความหน่วงเวลาขณะทดสอบระบบเปลี่ยนผ่าน

ผลการเทียบค่าอัตราเร็ว และค่าพวงมาลัยที่ได้จากเซ็นเซอร์บนรถทดสอบ กับค่าอัตราเร็วและค่าพวงมาลัยที่สั่งจาก Autoware และ Logitech G29 ถูกแสดงพร้อมกับค่าความหน่วงของสัญญาณในแต่ละช่วงของการทดสอบระบบเปลี่ยนผ่านการควบคุม เป็นไปดังภาพที่ 113 และ 114



ภาพที่ 113 *ผลตอบสนองของอัตราเร็วที่วัดจากเซ็นเซอร์วัดการหมุนของล้อ ค่าอัตราเร็วที่สั่งจาก Autoware และ Logitech G29 เทียบความหน่วงเวลาขณะทดสอบ*



ภาพที่ 114 *ผลตอบสนองของระบบบังคับเลี้ยวที่วัดจากจากเซ็นเซอร์วัดการหมุนของพวงมาลัย ค่าพวงมาลัยที่สั่งจาก Autoware และ Logitech G29 เทียบกับความหน่วงเวลาขณะทดสอบ*

ช่วงแรกระบบเริ่มทำงานในโหมดควบคุมจากระยะไกลจากจุดเริ่มต้นจนถึงจุดที่ 1 ค่าความหน่วงเวลาต่ำสุด และความหน่วงเวลาเฉลี่ยมีค่า 89.24 มิลลิวินาที และ 115.69 มิลลิวินาที ตามลำดับ จากการทดสอบพบว่าผู้ควบคุม ณ สถานีควบคุมสามารถสั่งการระบบเร่งและระบบบังคับเลี้ยวได้ โดยไม่รู้สึกถึงความยากในการควบคุม แม้มีความหน่วงเวลาสูงกว่า 200 มิลลิวินาที ในบางช่วงเป็นเวลาสั้น ๆ ค่าความหน่วงเวลาสูงสุดของการทดสอบช่วงนี้คือ 217.11 มิลลิวินาที หลังจากระบบเฝ้าระวังสถานการณ์ผิดปกติตรวจพบค่าความหน่วงเวลาสมมุติ (500 มิลลิวินาที) ที่ส่งจากสถานีควบคุมไปยังรถทดสอบ ณ จุดที่ 1 ระบบสามารถสั่งให้รถทดสอบเบรกจนรถหยุดนิ่ง และเข้าสู่โหมดฉุกเฉินได้ทันทีหลังจากตรวจพบความหน่วงเวลาที่กำหนดไว้ดังกล่าว

ถัดมาหลังจากผู้ควบคุมทำการกดปุ่มสลับโหมดเข้าสู่การควบคุมด้วยระบบอัตโนมัติ รถเริ่มเคลื่อนที่จากจุดที่ 1 ไปยังจุดที่ 2 ในช่วงนี้ระบบมีความหน่วงเวลาเฉลี่ย และความหน่วงเวลาต่ำสุดคือ 110.18 และ 83.68 มิลลิวินาที ตามลำดับ ก่อนถึงจุดปลายทางที่ 2 มีความหน่วงเวลาพุ่งสูงถึง 316.65 มิลลิวินาที แต่เนื่องจากอยู่ในโหมดการควบคุมด้วยระบบอัตโนมัติ ความหน่วงเวลาจึงไม่มีผลกับระบบควบคุมการเคลื่อนที่ของรถทดสอบ จากการสังเกตภาพวิดีโอ ณ สถานีควบคุม ขณะเกิดความหน่วงสูงสุดดังกล่าว ภาพเกิดการกระตุกเพียงแค่ชั่วครู่ เมื่อรถทดสอบเคลื่อนที่ถึงจุดที่กำหนดให้มีวัตถุกีดขวางเส้นทางบริเวณจุดที่ 2 เซนเซอร์ตรวจพบเจอสิ่งกีดขวางภายในระยะที่กำหนด Autoware จะสั่งค่าอัตราเร็วเป็น 0 เมตร/วินาที รถทดสอบจะเริ่มทำการเบรกจนถึงจุดหยุดนิ่ง ระบบเฝ้าระวังสถานการณ์ผิดปกติรับรู้ถึงสถานการณ์ที่เกิดขึ้นและจับเวลา 20 วินาที หลังจากครบเวลาดังกล่าว ระบบเข้าสู่โหมดฉุกเฉินตามที่ถูกออกแบบไว้

ต่อมาระบบเข้าสู่โหมดควบคุมจากระยะไกลอีกครั้งหลังจากผู้ควบคุมทำการกดปุ่มสลับโหมด ณ สถานีควบคุม รถทดสอบเริ่มวิ่งจากจุดที่ 2 ไปยังจุดที่ 3 ความหน่วงเวลาต่ำสุด และความหน่วงเวลาเฉลี่ยมีค่าอยู่ที่ 89.00 และ 106.49 มิลลิวินาที ตามลำดับ จากการทดสอบพบว่าผู้ควบคุม ณ สถานีควบคุมสามารถสั่งการระบบเร่งและระบบบังคับเลี้ยวได้ โดยไม่รู้สึกถึงความยากในการควบคุม แม้มีความหน่วงเวลาสูงกว่า 200 มิลลิวินาที ในบางช่วง และพุ่งขึ้นสูงสุดถึง 284.8 มิลลิวินาที หลังจากระบบเฝ้าระวังตรวจพบค่าความหน่วงเวลาสมมุติที่ส่งจากสถานีควบคุมสูงกว่า 500 มิลลิวินาที ณ จุดที่ 3 และเข้าสู่โหมดฉุกเฉินทันทีและค่าอัตราเร็วเป็น 0 เมตร/วินาที รถทดสอบทำการเบรกจนรถหยุดนิ่ง และเข้าสู่โหมดฉุกเฉินได้ทันที

ผู้ควบคุมทำการสลับโหมดเข้าสู่การควบคุมด้วยระบบอัตโนมัติอีกครั้งหลังจากอยู่ในโหมดฉุกเฉิน ณ จุดที่ 3 เพื่อทำการควบคุมรถทดสอบไปยังจุดที่ 4 ในช่วงนี้มีความหน่วงเวลาเฉลี่ย

ความหน่วงเวลาต่ำสุด และความหน่วงเวลาสูงสุดคือ 110.39 89.01 และ 266.29 มิลลิวินาที ตามลำดับ รถทดสอบเคลื่อนที่ถึงจุดที่มีวัตถุกีดขวางเส้นทางบริเวณจุดที่ 4 ระบบเริ่มลดอัตราเร็วจนถึงจุดหยุดนิ่ง ระบบเฝ้าระวังรู้สถานการณ์ที่เกิดขึ้นและจับเวลา 20 วินาที และเข้าสู่โหมดฉุกเฉิน จากสังเกตภาพบนมอนิเตอร์พบว่าภาพเกิดการกระตุกเพียงเล็กน้อย แต่เกิดขึ้นอย่างต่อเนื่องตลอดการทดสอบช่วงนี้

ช่วงสุดท้ายของการทดสอบ ผู้ควบคุมทำการสลับโหมดเป็นการควบคุมจากระยะไกล ณ จุด 4 จนถึงจุดสุดท้าย ความหน่วงเวลาต่ำสุด และความหน่วงเวลาเฉลี่ยมีค่า 89.98 และ 110.25 มิลลิวินาที ตามลำดับ ผู้ควบคุมสามารถสั่งการระบบเร่งและระบบบังคับเลี้ยวได้ โดยไม่รู้สึกถึงความยากในการควบคุม ความหน่วงสูงสุดอยู่ที่ 120.56 มิลลิวินาที ซึ่งถือว่ามีค่าน้อยกว่าการใช้งานในโหมดควบคุมระยะไกลที่ผ่านมา

5. ผลทดสอบหาระยะเบรกฉับพลันจากการกดปุ่มฉุกเฉินบนรถทดสอบ

การทดสอบหาระบบเบรกฉับพลัน ทำโดยให้ผู้ควบคุมทำการกดปุ่มฉุกเฉินบนรถทดสอบที่อัตราเร็วของรถมีค่า 3, 5, 10, 10.8(3 เมตร/วินาที), 15 กิโลเมตรต่อชั่วโมง ชุดเบรกจะทำงานเต็มที่หากปุ่มฉุกเฉินดังกล่าวถูกกด ผลการทดสอบเป็นไปดังตารางที่ 4

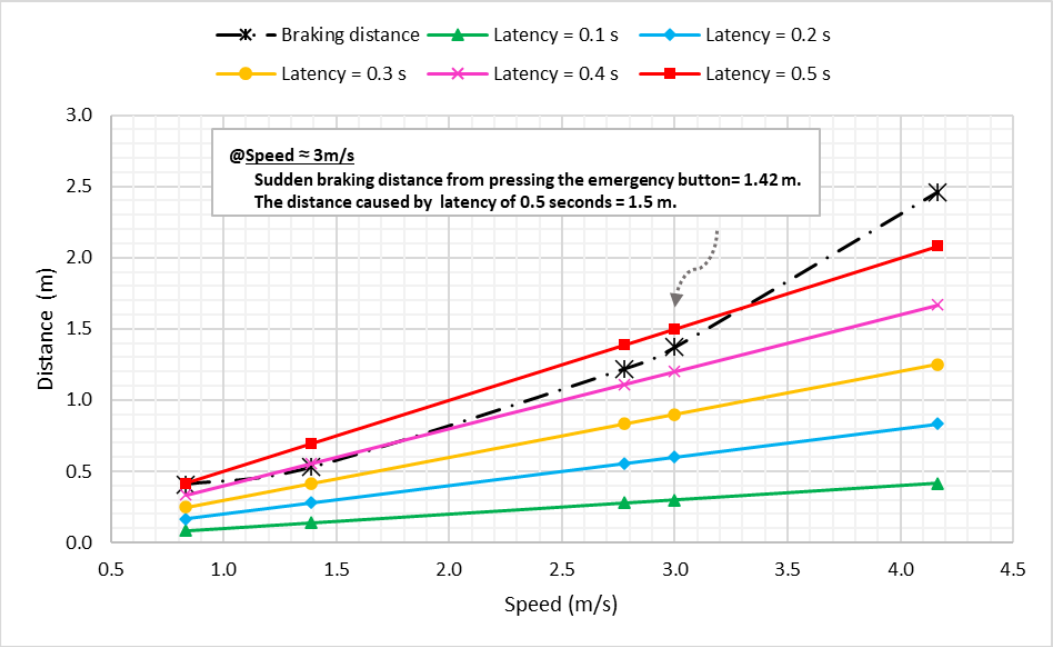
ตารางที่ 4 *ระยะเบรกที่เกิดขึ้นจากการกดปุ่มฉุกเฉินบนรถทดสอบ ณ อัตราเร็ว 0 ถึง 15 กิโลเมตรต่อชั่งโมง*

Speed (km/hr)	Braking distance from pressing the emergency button on the vehicle			
	1st Result	2nd Result	3rd Result	Average
3	0.45	0.42	0.35	0.41
5	0.62	0.55	0.42	0.53
10	1.26	1.09	1.31	1.22
10.8 (≈3m/s)	1.42	1.35	1.49	1.42
15	2.24	2.71	2.44	2.46

6. ผลกระทบของความหน่วงเวลาต่อระยะเบรกของรถทดสอบ

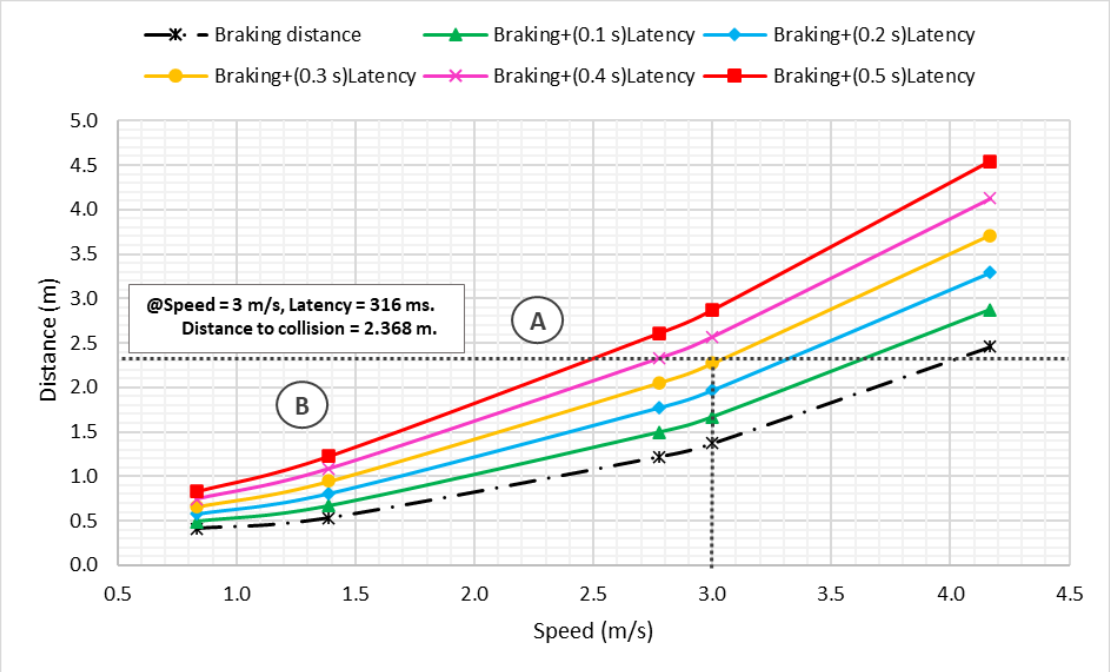
กรณีที่ผู้ควบคุม ณ สถานีควบคุมทำการสั่งเบรกจากระยะไกล ความหน่วงเวลาส่งผลให้ระยะเบรกเพิ่มขึ้น เนื่องจากรถทดสอบจะยังคงเคลื่อนที่ด้วยอัตราเร็วล่าสุดไปด้วยระยะทางหนึ่งก่อนได้รับคำสั่งเบรก ภาพที่ 115 แสดงถึงความสัมพันธ์ของระยะเบรกและระยะทางที่เกิดจากความหน่วงเวลา

เทียบกับอัตราเร็วของรถทดสอบ ที่ความหน่วงเวลา 400 ถึง 500 มิลลิวินาที และอัตราเร็วขณะทดสอบมีค่าตั้งแต่ 0 ถึง 3 เมตร/วินาที พบว่าระยะทางที่เกิดขึ้นจากความหน่วงเวลามีค่าใกล้เคียงกับระยะเบรกของรถทดสอบ การมีอุปสรรคเข้ากีดขวางอย่างกะทันหันภายในระยะเดียวกันกับระยะเบรก ณ ช่วงความหน่วงเวลา และอัตราเร็วดังกล่าว รถทดสอบอาจส่งอาจชนกับสิ่งกีดขวางก่อนที่ผู้ควบคุม ณ สถานีควบคุมจะพบเห็นอุปสรรคผ่านหน้าจอมอนิเตอร์ ส่วนความหน่วงเวลาที่ต่ำกว่า 400 มิลลิวินาทีโดยประมาณ หากมีอุปสรรคกีดขวางอย่างกะทันหัน ผู้ควบสามารถที่จะเห็นสิ่งกีดขวางผ่านหน้าจอมอนิเตอร์ได้ทัน แต่ด้วยระยะทางที่กระชั้นชิดระบบเบรกอาจไม่สามารถเบรกได้ทัน ผู้ควบคุมมีความจำเป็นต้องบังคับพวงมาลัยเพื่อหลีกเลี่ยงการชนนอกเหนือจากการเบรกได้ หรือในบางกรณีอาจทำการสลับโหมดไปยังระบบอัตโนมัติชั่วคราวหากมีความกังวลว่าอาจจะมีความหน่วงเวลาพุ่งสูงขึ้นอย่างต่อเนื่อง ระบบอัตโนมัติจะประเมินสถานการณ์แทนหากมีอุปสรรคกีดขวาง

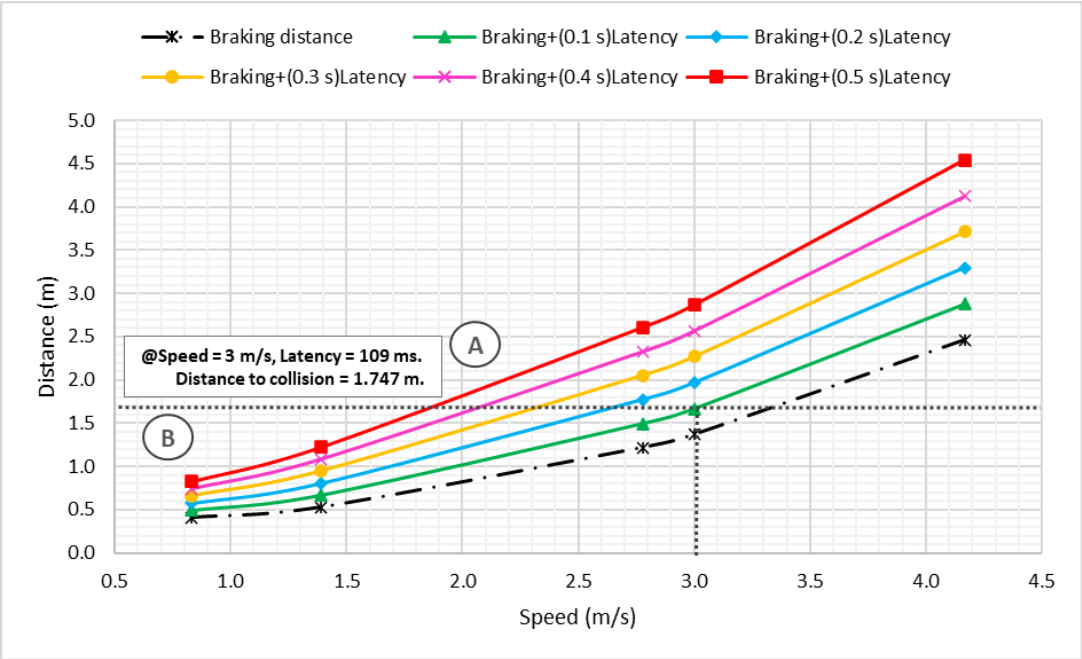


ภาพที่ 115 *กราฟความสัมพันธ์ของระยะเบรกฉับพลัน และระยะทางที่เกิดขึ้นจากความหน่วงเวลาเทียบกับอัตราเร็วของรถทดสอบ*

จากการทดสอบระบบเปลี่ยนผ่านการควบคุมที่ผ่านมา ค่าความหน่วงเวลาสูงสุดและค่าความหน่วงเวลาเฉลี่ยของการทดสอบคือ 316.65 มิลลิวินาที และ 109.70 มิลลิวินาที ตามลำดับ อัตราเร็วสูงสุดของรถขณะทดสอบคือ 3 เมตร/วินาที เมื่อหาระยะเบรกจากกดปุ่มฉุกเฉินจากระยะไกลที่อัตราเร็วสูงสุดและค่าความหน่วงดังกล่าว จากกราฟความสัมพันธ์ระหว่างระยะเบรกเทียบกับอัตราเร็วของรถทดสอบ ณ ความหน่วงเวลาต่าง ๆ ดังภาพที่ 116 และ 117 ค่าระยะเบรกขณะรถทดสอบมีอัตราเร็วสูงสุด ค่าความหน่วงเวลาสูงสุด และความหน่วงเวลาเฉลี่ย จะมีค่าอยู่ที่ 2.37 และ 1.75 เมตร ตามลำดับ หากมีวัตถุกีดขวางกะทันหันก่อนถึงระยะเบรกดังกล่าว (โซน B) รถทดสอบจะไม่สามารถเบรกได้ทันและชนกับวัตถุดังกล่าว ผู้ควบคุมมีความจำเป็นต้องใช้บังคับเลี้ยวเพื่อหลีกเลี่ยงสิ่งกีดขวางด้วย และหากมีวัตถุกีดขวางหลังระยะเบรก (โซน A) รถทดสอบจะสามารถเบรกได้ทัน



ภาพที่ 116 *กราฟความสัมพันธ์ระหว่างระยะเบรกเทียบกับอัตราเร็วของรถทดสอบ ณ ความหน่วงเวลาสูงสุดขณะทดสอบระบบเปลี่ยนผ่าน*



ภาพที่ 117 *กราฟความสัมพันธ์ระหว่างระยะเบรกเทียบกับอัตราเร็วของรถทดสอบ ณ ความหน่วงเวลาเฉลี่ยขณะทดสอบระบบเปลี่ยนผ่าน*

บทที่ 5
บทสรุปและข้อเสนอแนะ

1. สรุปผลการวิจัย

ชุดควบคุมสั่งงานด้วยสัญญาณทางไฟฟ้า ระบบควบคุมในระดับต่ำและระดับสูง ชุดโมดูลสั่งการจากระยะไกลพร้อมทั้งสถานีควบคุม และอุปกรณ์ที่จำเป็นสำหรับการพัฒนารถกอล์ฟไฟฟ้าให้เป็นรถที่สามารถควบคุมได้ทั้งระบบอัตโนมัติและระบบควบคุมจากระยะไกลถูกออกแบบและติดตั้งสำหรับงานวิจัยนี้เพื่อใช้ในการทดสอบระบบเปลี่ยนผ่านการควบคุมทั้งสองระบบดังกล่าว จากการทดสอบพบว่าระบบควบคุมในระดับต่ำสามารถทำงานได้จากการป้อนคำสั่งจากซอฟต์แวร์ควบคุมระดับสูงและจากชุดควบคุม ณ สถานีควบคุมระยะไกล

การทดสอบระบบเปลี่ยนผ่านแบ่งออกเป็น 4 ช่วง โดยมีระบบเฝ้าระวังความผิดปกติคอยตัดการทำงานเข้าสู่โหมดฉุกเฉินในแต่ละจุดที่ทำการกำหนดสถานการณ์ที่ผิดปกติไว้ จากการทดสอบพบว่าระบบสามารถตัดการทำงานเข้าสู่โหมดฉุกเฉินได้จริงเมื่อมีเหตุผิดปกติ เช่น การมีอุปสรรคกีดขวางและการมีความหน่วงเวลาที่สูงเกินกำหนด ค่าความหน่วงเวลาเฉลี่ย และต่ำสุดที่วัดได้ตลอดการทดสอบมีค่า 109.7 และ 83.7 มิลลิวินาที ตามลำดับ ผู้ควบคุมสามารถใช้งานในโหมดการควบคุมจากระยะไกล และสั่งการเปลี่ยนโหมดได้โดยไม่รู้สึกถึงความยากในการควบคุม แม้ว่ามีค่าความหน่วงเวลาสูงถึง 316.6 มิลลิวินาที ในบางช่วงขณะทดสอบ ผลของค่าความคลาดเคลื่อนระหว่างเส้นทางอ้างอิงและเส้นทางที่บันทึกขณะทดสอบระบบควบคุมระยะไกลทั้งสองช่วง มีส่วนเบี่ยงเบนเชิงเส้นเฉลี่ยอยู่ที่ 0.25 และ 0.19 เมตร และค่าสูงสุดอยู่ที่ 0.41 และ 0.58 เมตร ส่วนการควบคุมด้วยระบบอัตโนมัติทั้งสองช่วงมีค่าเบี่ยงเบนเชิงเส้นเฉลี่ยอยู่ที่ 0.25 และ 0.33 เมตร และค่าสูงสุดคือ 0.55 และ 0.83 เมตร ส่วนเบี่ยงเบนเชิงเส้นเฉลี่ยของระบบควบคุมทั้งสองระบบมีค่าใกล้เคียงกัน แต่ค่าเบี่ยงเบนสูงสุดของการควบคุมด้วยระบบอัตโนมัติยังคงสูงกว่าการควบคุมจากระยะไกล จากผลแสดงให้เห็นว่าระบบควบคุมจากระยะไกลสามารถทำงานได้ดีแม้มีความหน่วงเวลาเกิดขึ้นตลอดการทดสอบ

ผลกระทบของความหน่วงเวลาต่อระยะเบรกฉับพลันจากกดปุ่มฉุกเฉิน ณ สถานีควบคุมขณะรถมีอัตราเร็วทดสอบ 0 ถึง 3 เมตรต่อวินาที ณ กรณีที่มีความหน่วงเวลา 500 มิลลิวินาทีขึ้นไป รถทดสอบอาจชนกับอุปสรรคที่เข้ากีดขวางเส้นอย่างกะทันหันก่อนที่ผู้ควบคุมจะเห็นภาพอุปสรรคดังกล่าวจากมอนิเตอร์ และหากกดปุ่มเบรกฉุกเฉิน ณ สถานีควบคุม ที่ขณะนั้นมีค่าความหน่วงเวลาเฉลี่ยและสูงสุดเกิดขึ้นขณะที่รถมีอัตราเร็วสูงสุดที่ใช้ในการทดสอบ (3 เมตร/วินาที) การมีความหน่วง

เฉลี่ยและความหน่วงสูงสุดส่งผลให้ระยะเบรกเพิ่มขึ้นจากระยะเบรกปกติของรถ 1.42 เมตร เพิ่มขึ้นเป็น 1.75 และ 2.37 เมตรตามลำดับ ซึ่งภายในระยะดังกล่าวหากมีอุปสรรคกีดขวางกะทันหัน รถทดสอบจะไม่สามารถเบรกได้ทัน ผู้ควบคุมจำเป็นต้องบังคับพวงมาลัยเพื่อหลีกเลี่ยงสิ่งกีดขวางนอกเหนือจากการเบรก

2. ข้อเสนอแนะ

1. กลไกลควบคุมอัตราเร็วของยานยนต์ต้นแบบนี้สามารถปรับปรุงให้มีการตอบสนองที่แม่นยำได้มากขึ้น หากมีการพัฒนาต่อให้ระบบมีการควบคุมแบบวงปิด (Closed-loop control)

2. การรับ-ส่งภาพวิดีโอ และข้อมูลคำสั่งควบคุมต่าง ๆ ผ่านเครือข่ายไร้สายเป็นเรื่องท้าทายสำหรับงานควบคุมระยะไกล ซึ่งสามารถพัฒนาให้ดียิ่งขึ้นได้ เช่น การใช้เทคโนโลยีเครือข่ายไร้สายที่รองรับข้อมูลขนาดใหญ่ขึ้น การมีความหน่วงเวลาของสัญญาณที่ต่ำ หรือการเขียนโปรแกรมให้มีการประมวลผลข้อมูลเร็วที่สุดเท่าที่เป็นไปได้

3.งานวิจัยนี้มีการทดสอบระบบเฝ้าระวังเหตุฉุกเฉินขณะควบคุมทั้งหมด 2 กรณี การใช้งานจริงในระดับที่สูงขึ้น จำเป็นต้องมีวิธีการรองรับเหตุฉุกเฉินสำหรับความผิดปกติที่หลายรูปแบบมากขึ้น

ภาคผนวก ก
ข้อมูลทางเทคนิคของรถกอล์ฟไฟฟ้า

❖ EVT Plus D 6s+2

ภาพที่ 118 *รถกอล์ฟไฟฟ้า ยี่ห้อ* ***EVT*** *รุ่น Plus D 6s+2*

ตารางที่ 5 *ข้อมูลทางเทคนิคของรถกอล์ฟไฟฟ้า ยี่ห้อ* ***EVT*** *รุ่น Plus D 6s+2*

Specification	Detail
Seating	
Passenger Capacity	8
Drive System	
System Voltage	48 Volt
Motor Power	5 kW (6.7 HP)
Transmission	Automatic
Battery	6 Volt 8 Units
Charger	220 Volt
Performance	
Maximum Speed*	30 km/h
Range Per Charge*	70 km
Climbability*	10°
Steering	
Steering System	Rack and pinion
Steering Side	Right-hand Steering
Brake System	
Front	Drum brake
Rear	Drum brake

Specification	Detail
Suspension System	
Front/Rear	Leaf spring with Shock Absorber
Dimension	
Overall length	4370 mm
Overall width	1230 mm
Overall hight	1950 mm
Load capacity	550 kg
Tyre & Wheel	
Wheel	Aluminium Wheel 10 inch
Tyre size	205/50 R10
Body & Frame	
Frame	Steel Frame with Anti Rust Treatment
Roof	Plastic
Body	Fibreglass

ภาคผนวก ข
อุปกรณ์ควบคุมในระดับต่ำ

❖ กล่องควบคุมอิเล็กทรอนิกส์ รุ่น Curtis 1268

ภาพที่ 119 *Electronic Motor Controller - Curtis 1268*

FEATURES

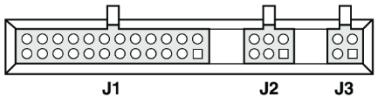
- Power MOSFET technology provides smooth, silent, efficient, and cost-effective operation.
- Adjustable parameters enable custom optimization of speed, torque, and braking control.
- Half bridge armature and full bridge field provides regenerative braking down to zero speed.
- Sealed package rated at IP64 and IP67.
- Overspeed braking (regenerative) limits speed while driving downhill.
- WalkAway™ braking feature limits any stopped or key-off rolling to very low speed.
- System uses Hall effect speed sensor on motor or drive train to control vehicle speed.

- Tow switch enables free rolling for towing of vehicle.
- Anti-rollback function provides improved control when throttle is released on hills.
- Anti-stall function helps prevent motor commutator damage.
- Controller drives warning buzzer -- steady in reverse; intermittent during WalkAway™ braking.
- Optional electromagnetic brake output.
- Optional brake light output.
- MultiMode™ inputs provide for multiple speed and power modes of operation.
- Timed shutdown of main contactor after pedal is released and vehicle has stopped.
- LED status indicator with external output.
- Fully compatible with 1311, 1314 and 1307 Programmers for parametric adjustment, tuning, test, and diagnostics.
- Extensive fault detection and diagnostic reporting using a Curtis Programmer including (partial list):
    - Main contactor weld check and driver check
    - Throttle and wiring faults
    - Open or shorted motor field winding
    - Open motor armature winding
    - Over-temperature
    - Missing or failed speed sensor
    - Armature drive failure

CONNECTIONS

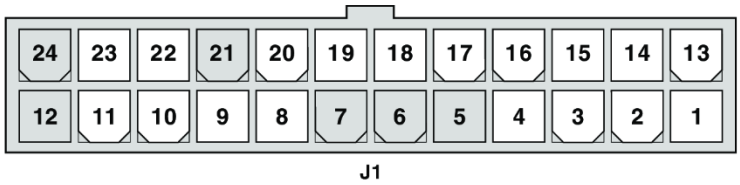
Low Current Connections

Three low current connectors are built into the 1268 controller. They are located in a row on the top of the controller:



J1 *Logic connector*
J2 *Speed Sensor connector*
J3 *Programmer connector*

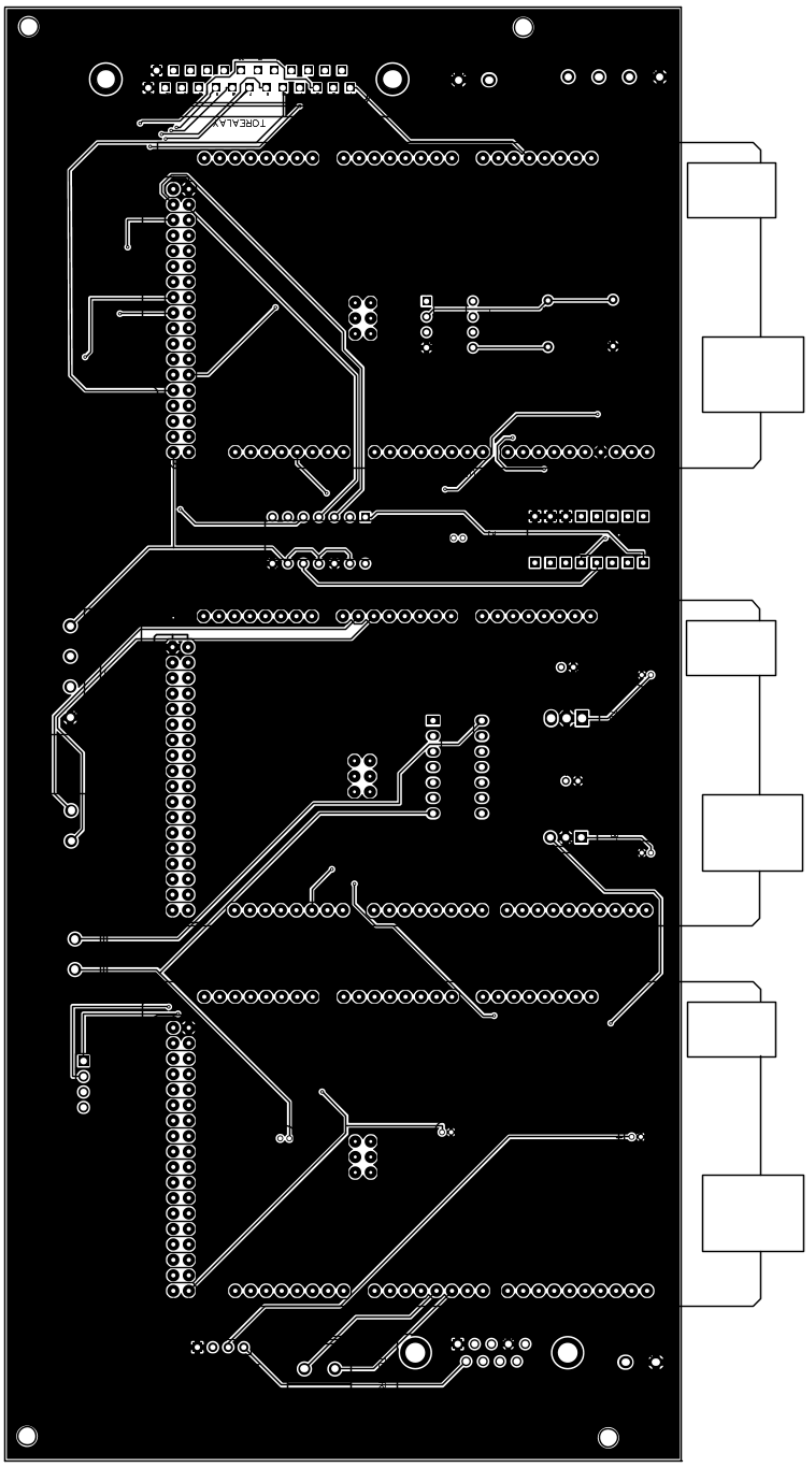
The **24-pin** connector provides the logic control connections. The mating connector is a 24-pin Molex Mini-Fit Jr. connector p/n 39-01-2245 using type 5556 terminals. The appropriate wire gauge is 18–24 AWG.



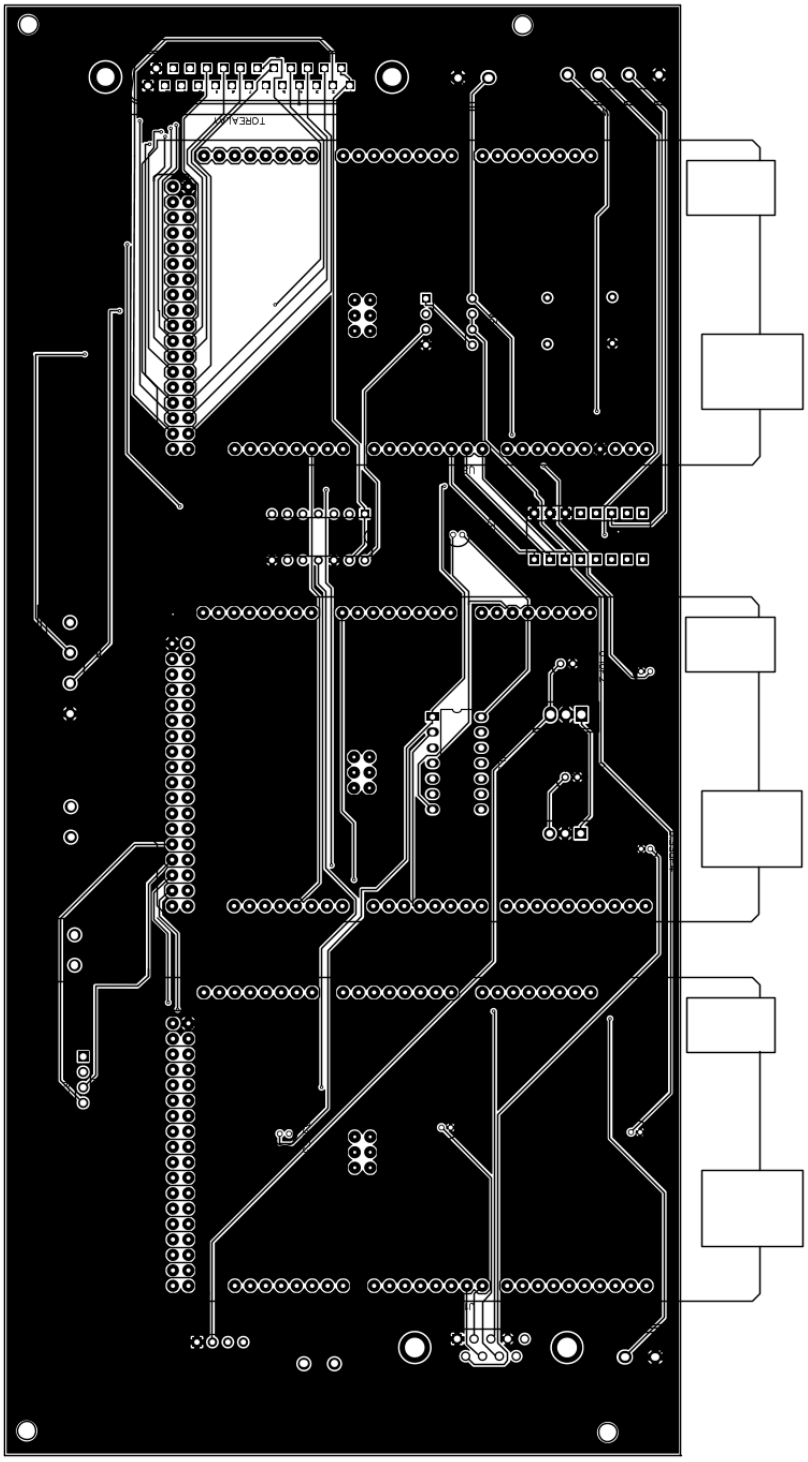
J1-1	Keyswitch Input (KSI)	*input and return for main contactor coil*
J1-2	Logic Enable	*input from run/store switch*
J1-3	Fuse Sense	*input from WalkAway™ fuse sense line*
J1-4	Logic Power	*power to controller logic*
J1-5	(not used)	—
J1-6	(not used)	—
J1-7	(not used)	—
J1-8	Pedal Interlock Switch	*input from pedal switch, wired to throttle*
J1-9	WalkAway™ Return	*coil return for WalkAway™ relay*
J1-10	Forward	*input from forward switch*
J1-11	Reverse	*input from reverse switch*
J1-12	(not used)	—
J1-13	Pot High	*+5V supply through 453Ω (or ITS input)*
J1-14	Pot Low	*453Ω to ground*
J1-15	Pot Wiper	*throttle wiper input (or ITS input)*
J1-16	LED Ground	*ground for external LED*
J1-17	Main Contactor	*contactor coil driver low-side output*
J1-18	Brake Light Driver	*relay driver low-side output*
J1-19	Reverse Alarm	*alarm low-side driver output*
J1-20	Mode Switch	*input from mode switch*
J1-21	(not used)	—
J1-22	External LED Driver	*LED driver high-side output*
J1-23	Auxiliary Driver	*WalkAway™/EMB driver low-side output*
J1-24	(not used)	—

ภาพที่ 120 *Low current connections of Curtis 1268*

❖ **แผงวงจรพิมพ์ (Printed Circuit Board)**



ภาพที่ 121 *ด้านบนของแผงวงจรพิมพ์*



ภาพที่ 122 ด้านล่างของแผงวงจรพิมพ์

ภาคผนวก ค
เซนเซอร์ตรวจจับและวัดระยะทางด้วยแสง

❖ Velodyne VLP-16

ภาพที่ 123 *อุปกรณ์ตรวจจับและวัดระยะทางด้วยแสง ยี่ห้อ Velodyne รุ่น VLP-16*

ตารางที่ 6 *ข้อมูลทางเทคนิคของอุปกรณ์ตรวจจับและวัดระยะทางด้วยแสง Velodyne VLP-16*

Specifications	Detail
Sensor	Time of flight distance measurement with calibrated reflectivities
	16 channels
	Measurement range up to 100 meters
	Accuracy: +/- 3 cm (typical)
	Dual returns
	Field of view (vertical): 30° (+15° to -15°)
	Angular resolution (vertical): 2°
	Field of view (horizontal/azimuth): 360°
	Angular resolution (horizontal/azimuth): 0.1° - 0.4°
	Rotation rate: 5 - 20 Hz
	Integrated web server for easy monitoring and configuration
Laser	Class 1 - eye safe
	905 nm wavelength
Mechanical/ Electric/ Operational	Power consumption: 8 W (typical)
	Operating voltage: 9 - 32 VDC (with interface box and regulated power supply)
	Weight: 830 grams (without cabling)
	Dimensions: 103 mm diameter x 72 mm height
	Shock: 500 m/sec² amplitude, 11 msec duration
	Vibration: 5 Hz to 2000 Hz, 3G rms
	Environmental Protection: IP67
	Operating temperature -10° to +60° C
	Storage temperature - 40° to +105° C
Output	Up to 0.3 million points/second
	100 Mbps Ethernet connection
	UDP packets containing
	$GPRMC NMEA sentence from GPS receiver (GPS not included)

ภาคผนวก ง
อุปกรณ์รับ-ส่งสัญญาณเครือข่ายไร้สาย 5G

❖ Huawei 5G Outdoor CPE N5368X

ภาพที่ 124 *Huawei 5G Outdoor CPE N5368X*

ตารางที่ 7 *ข้อมูลจำเพาะของ Huawei 5G Outdoor CPE N5368X*

Specification	Detail
Operating Frequency bands	5G NR: n78/n77/n41/n38
	LTE FDD: Band 1/Band 3/Band 7/Band 8/Band 20/Band 28
	LTE TDD: Band 38/Band 40/Band 41/Band 42/Band 43
	WLAN: 2.4 GHz/5 GHz
	Note: The supported operating frequency bands vary with areas. The n78 frequency band range is 3400–3800 MHz.
Maximum Transmit Power	5G NSA: 23 dBm@1T
	5G SA: 26 dBm@2T
	LTE: 23 dBm
	ODU WLAN 5G: 14 dBm
	ODU WLAN 2.4G: 16 dBm
	The supported WLAN maximum conducted transmit power varies
Supply	Output Voltage: 19 V ± 5%
	Output Power: 24 W
High Data Rate and Larg Bandwidth	For NSA Network : A PS “ Packet Switched rate of up to 1.5 Gbit/s in Download and 110 Mibit /s in the Uplink
	For SA Network : A PS “ Packet Switched rate of up to 1.5 Gbit/s in Download and 220 Mibit /s in the Uplink
High–Rate WLAN Access	A Maximum of 32 device can be connected at 2.4 GHZ and 5GHZ Bands
	A Maximum of 300 Mbits /s with IEEE 802 .11n at 2.4 GHZ
	A Maximum of 867 Mbit/s with IEEE 802.11ac at 5GHZ

ภาคผนวก จ
อุปกรณ์ประมวลผล

❖ คอมพิวเตอร์สถานีควบคุม

ภาพที่ 125 *คอมพิวเตอร์สถานีควบคุม*

ตารางที่ 8 *ข้อมูลจำเพาะของคอมพิวเตอร์สถานีควบคุม*

Specification	Detail
Processor	Intel Core i7-11800H (8-core 2.3 GHz, up to 4.6 GHz)
Chipset	Intel, NVIDIA
System Memory	2 x DDR4-3200 64GB
Graphics Engine	NVIDIA GeForce RTX 3080 Laptop GPU 16GB GDDR6 256-bit, up to 150W
Hard Drive	1 x 2.5-inch SATA 6.0 Gbps HDD/SSD bay
M.2	512 GB
USB Port	5 x USB 3.1 GEN 2
LAN	Dual LAN (Gigabit Ethernet, 2.5Gbps Ethernet)
Video Output	2 x HDMI 2.1* (up to 7680x4320@60Hz)
Wifi	WiFi 6 AX1650
ZBOX Dimensions	210mm x 203mm x 62.2mm (8.27in x 7.99in x 2.45in)

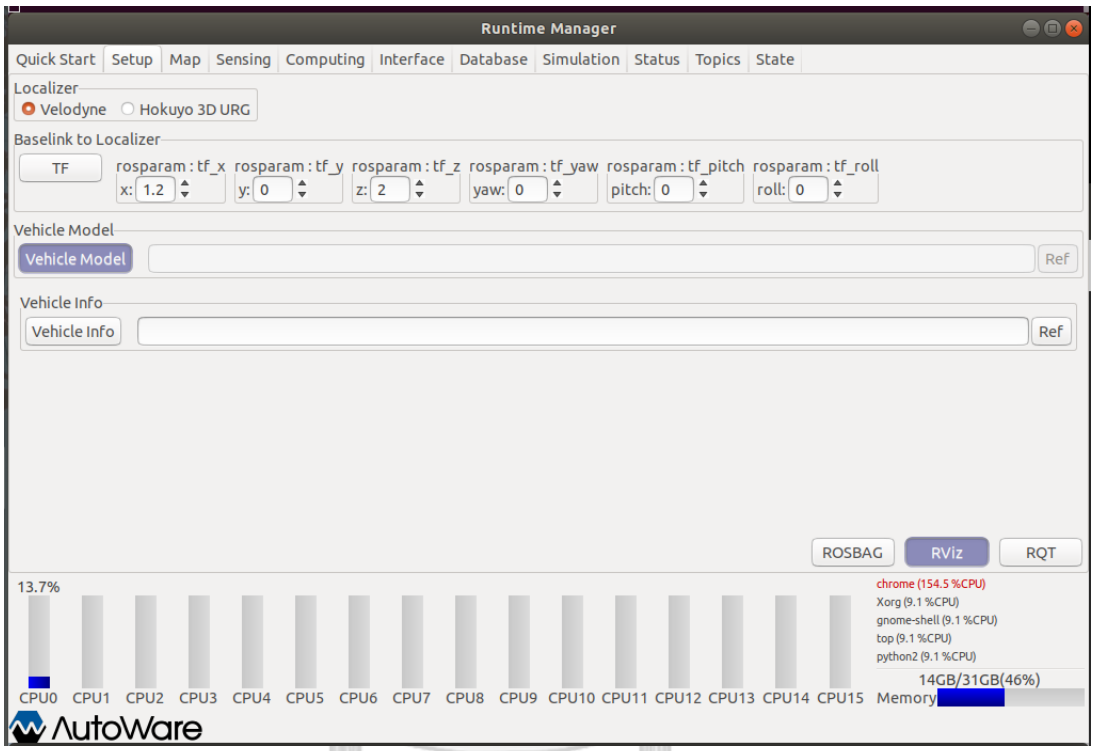
❖ **คอมพิวเตอร์ยานยนต์ต้นแบบ**

ภาพที่ 126 *คอมพิวเตอร์ยานยนต์ต้นแบบ*

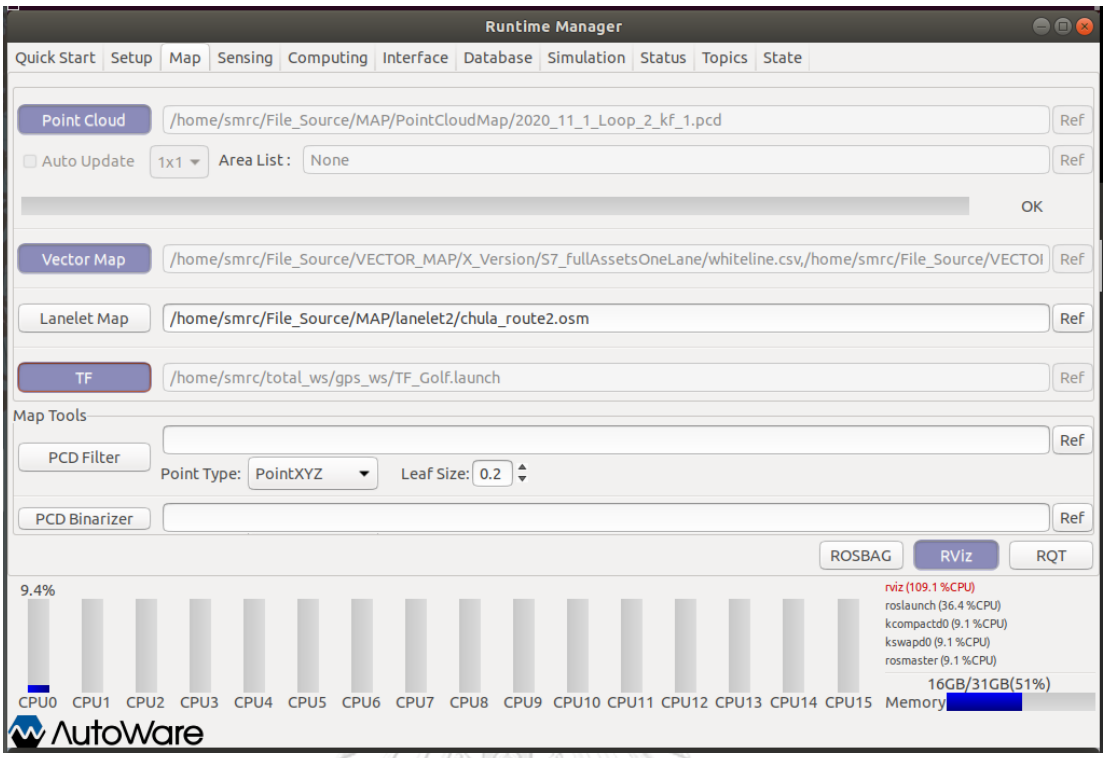
ตารางที่ 9 *ข้อมูลจำเพาะของคอมพิวเตอร์ยานยนต์ต้นแบบ*

Specification	Detail
Processor	Intel Core i7-10750H (six-core 2.6 GHz, up to 5.0 GHz)
Chipset	Intel, NVIDIA
System Memory	2 x DDR4-3200 64GB
Graphics Engine	NVIDIA GeForce RTX 2080 SUPER 8GB GDDR6 256-bit
Hard Drive	1 x 2.5-inch SATA 6.0 Gbps HDD/SSD bay
M.2	512 GB
USB Port	5 x USB 3.1 GEN 2
LAN	Dual LAN (Gigabit Ethernet, 2.5Gbps Ethernet)
Video Output	2 x HDMI 2.0, 2 x DisplayPort 1.4
Wifi	WiFi 6 AX1650
ZBOX Dimensions	210mm x 203mm x 62.2mm (8.27in x 7.99in x 2.45in)

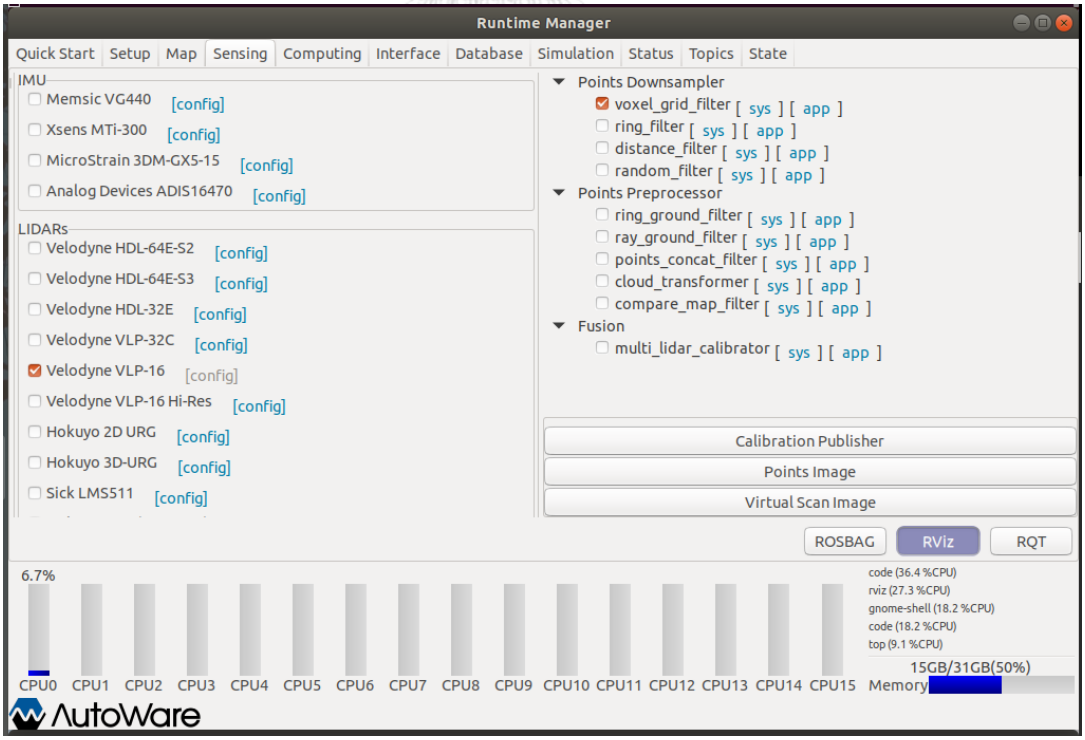
ภาคผนวก ฉ
การตั้งค่าในแอพพลิเคชั่น Autoware



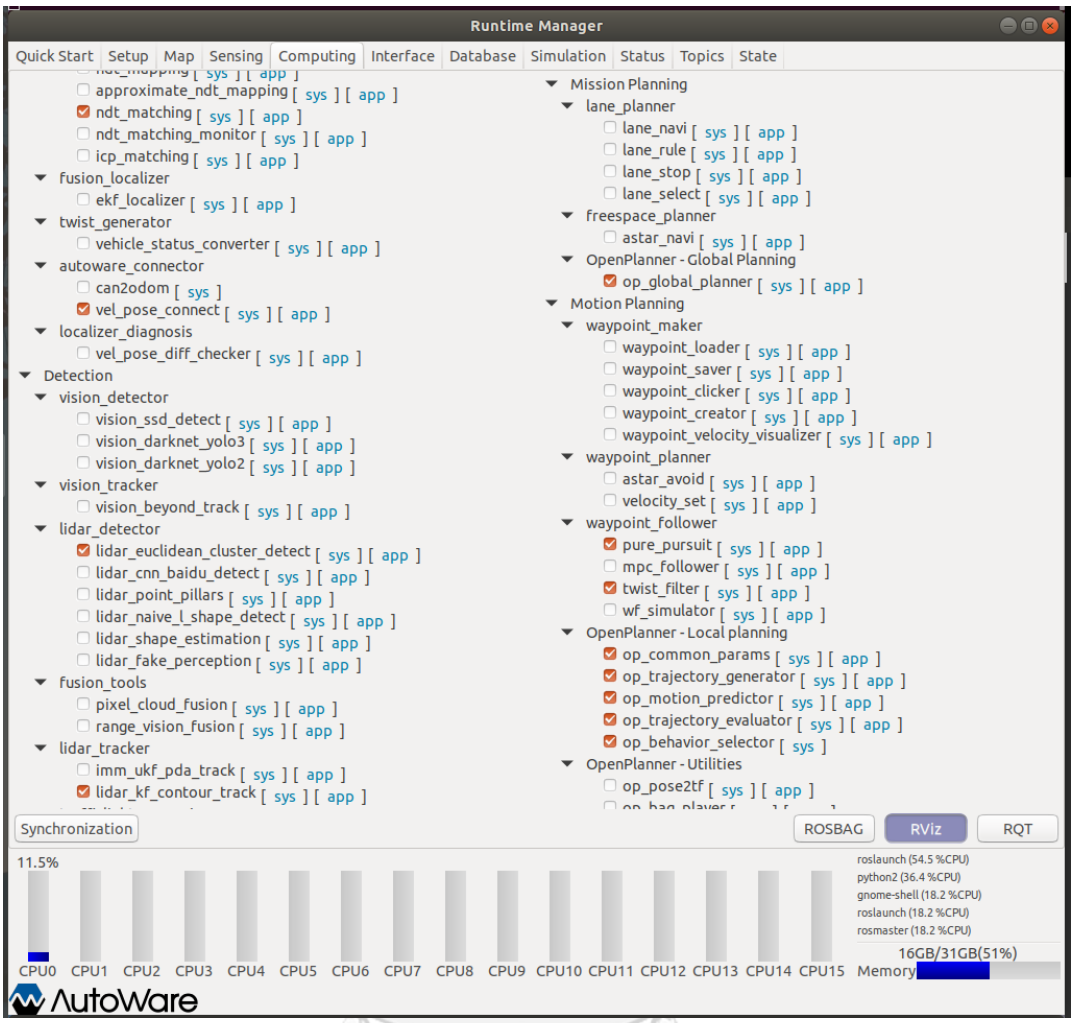
ภาพที่ 127 *การตั้งค่า Transformation Frame Lib ในหน้า Setup*



ภาพที่ 128 *การตั้งค่าในหน้า Map*



ภาพที่ 129 *การตั้งค่าในหน้า Sensing*



ภาพที่ 130 *การตั้งค่าในหน้า Computing*

ภาคผนวก ช

Source code

ตารางที่ 10 *Cockpit_camera_receiver*

```
#! /usr/bin/env python

import sys
import cv2
import rospy
import numpy as np
from teleopt_function import UDP_Communication

class CockpitCameraReceiver():
    def __init__(self, monitor_index=0):
        rospy.init_node("cockpit_camera_receiver_" + str(monitor_index))
        self.setupParam()

        rospy.on_shutdown(self.requestShutdown)

        self.monitor_index = int(monitor_index)
        self.udp = UDP_Communication(ip_address=self.cockpit_com_address,
port=self.lan_port + self.monitor_index)
        self.udp.setupServer()

        self.udp.dumpBuffer()
        self.Main()

    def __del__(self):
        rospy.loginfo("Cockpit Camera Receiver %d shutted down.", self.monitor_index)

    def Main(self):
        while(not self.shutdown):
            img_byte = self.udp.receive()
            img = cv2.imdecode(np.fromstring(img_byte, dtype=np.uint8), 1)
            cv2.namedWindow("MONITOR " + str(self.monitor_index + 1),
cv2.WINDOW_NORMAL)
            try:
                cv2.imshow("MONITOR " + str(self.monitor_index + 1), img)
            except:
                self.udp.setupServer()
```

```
35              self.udp.dumpBuffer()
36            if(cv2.waitKey(1) & 0xFF == ord('q')):
37              break
38
39            # self.udp.dumpBuffer()
40
41        cv2.destroyAllWindows()
42        self.udp.server.close()
43
44    def setupParam(self):
          self.cockpit_com_address =
45  rospy.get_param("/5G_CONNECTION/COCKPIT_COM_ADDRESS")
46        self.lan_port          = rospy.get_param("/5G_CONNECTION/UDP_PORT_1")
47        self.shutdown           = False
48
49    def requestShutdown(self):
50        self.shutdown = True
51
52  def initialized():
53    if(len(sys.argv) == 2 or len(sys.argv) == 4):
54        cam = CockpitCameraReceiver(sys.argv[1])
55    else:
          print("please fill out the arguments following the below structure.\n>>> python
56  python_file monitor_index")
57
58
59  if __name__ == "__main__":
60    initialized()
```

ตารางที่ 11 *Cockpit_latency_measurement*

Line	Code
1	#! /usr/bin/env python3
2	
3	import rospy
4	from std_msgs.msg import Int32
5	from teleopt_function import TCP_Communication
6	
7	class CockpitLatencyMeasurement(object):
8	
9	def __init__(self) -> None:
10	rospy.init_node("cockpit_latency_measurement")
11	self.setupParam()
12	
13	self.pub_cockpit_latency = rospy.Publisher("/cockpit/latency", Int32, queue_size=1) self.pub_cockpit_sub_latency = rospy.Publisher("/cockpit/sub_latency", Int32,
14	queue_size=1)
15	rospy.on_shutdown(self.requestShutdown)
16	
17	self.tcp = TCP_Communication(ip_address=self.golf_cpe_address, port=self.wan_port)
18	self.tcp.setupClient()
19	
20	self.Main()
21	
22	def __del__(self) -> None:
23	rospy.loginfo("Cockpit Latency Measurement shutted down.")
24	
25	def Main(self) -> None:
26	while(not self.shutdown):
27	received_from_tcp = self.tcp.receive(self.tcp.client, 1024)
28	self.tcp.send(self.tcp.client, received_from_tcp)
29	
30	
31	cockpit_send_time = rospy.Time.now().to_sec()
32	self.tcp.send(self.tcp.client, cockpit_send_time)
33	received_from_tcp = self.tcp.receive(self.tcp.client, 1024)
34	cockpit_receive_time = received_from_tcp
35	
36	latency = int((rospy.Time.now().to_sec() - cockpit_receive_time) * 1000 / 2)
37	self.pub_cockpit_latency.publish(latency)
38	self.pub_cockpit_sub_latency.publish(latency)
39	
40	rospy.loginfo("cockpit send:%f |cockpit recv:%f |latency:%f", cockpit_send_time, cockpit_receive_time, latency)

```
41
42      def setupParam(self) -> None:
            self.golf_cpe_address =
43  rospy.get_param("/5G_CONNECTION/GOLF_CPE_NSA_ADDRESS")
44          self.wan_port       = rospy.get_param("/5G_CONNECTION/TCP_PORT_3")
45          self.shutdown       = False
46
47      def requestShutdown(self) -> None:
48          self.shutdown = True
49
50
51  if __name__ == "__main__":
52      CockpitLatencyMeasurement()
```

ตารางที่ 12 *Cockpit_logitech_communication*

Line	Code
1	#! /usr/bin/env python3
2	
3	import rospy
4	from std_msgs.msg import UInt8, Float32, Float64, Int32MultiArray
5	from teleopt_function import TCP_Communication, LogitechG29, GEAR_DIRECTION
6	
7	class CockpitLogitechCommunication(object):
8	def __init__(self) -> None:
9	rospy.init_node("cockpit_logitech_communication")
10	self.setupParam()
11	self.pub_teleopt_steering = rospy.Publisher("/cockpit/teleopt/steering_command",
12	Float32, queue_size=1)
	self.pub_teleopt_velocity = rospy.Publisher("/cockpit/teleopt/velocity_command",
13	Float32, queue_size=1)
	self.pub_teleopt_brake = rospy.Publisher("/cockpit/teleopt/brake_command",
14	Float32, queue_size=1)
	self.pub_teleopt_gear = rospy.Publisher("/cockpit/teleopt/gear_command", UInt8,
15	queue_size=1)
	self.pub_teleopt_buttons = rospy.Publisher("/cockpit/teleopt/buttons_command",
16	Int32MultiArray, queue_size=1)
	self.pub_teleopt_hats = rospy.Publisher("/cockpit/teleopt/hats_command",
17	Int32MultiArray, queue_size=1)
	self.pub_teleopt_time = rospy.Publisher("/cockpit/teleopt/time", Float64,
18	queue_size=1)
19	rospy.on_shutdown(self.requestShutdown)
20	
21	self.logitech = LogitechG29()
22	self.tcp = TCP_Communication(ip_address=self.golf_cpe_address, port=self.wan_port)
23	self.tcp.setupClient()
24	
25	self.Main()
26	
27	def __del__(self) -> None:
28	rospy.loginfo("Cockpit Logitech Communication shutted down.")
29	
30	def Main(self) -> None:
31	while(not self.shutdown):
32	signals_array = self.logitech.readLogitech()
33	
34	measuredTime = rospy.Time.now().to_sec()
35	signals_array.append(measuredTime)
36	rospy.loginfo(signals_array)

```
37
38              self.tcp.send(self.tcp.client, signals_array)
39              self.publishAllSignals(signals_array)
40              # self.logitech.clock.tick(30)
41              self.rate.sleep()
42          # self.logitech.pygame.quit()
43
44      def setupParam(self) -> None:
            self.golf_cpe_address    =
45  rospy.get_param("/5G_CONNECTION/GOLF_CPE_NSA_ADDRESS")
46          self.wan_port          = rospy.get_param("/5G_CONNECTION/TCP_PORT_1")
47          self.shutdown          = False
48          self.steering_index     = rospy.get_param("/LogitechG29/STEERING_WHEEL")
49          self.velocity_index     = rospy.get_param("/LogitechG29/ACCEL_PEDAL")
50          self.brake_index        = rospy.get_param("/LogitechG29/BRAKE_PEDAL")
51          self.gear_forward_index  = rospy.get_param("/LogitechG29/GEAR_MID_FORWARD")
52          self.gear_backward_index = rospy.get_param("/LogitechG29/GEAR_MID_BACKWARD")
53          self.rate = rospy.Rate(30)
54
55      def requestShutdown(self) -> None:
56          self.shutdown = True
57
58      def publishAllSignals(self, signals) -> None:
59          self.pub_teleopt_steering.publish(signals[self.steering_index])
60          self.pub_teleopt_velocity.publish(signals[self.velocity_index])
61          self.pub_teleopt_brake.publish(signals[self.brake_index])
62          self.pub_teleopt_time.publish(signals[-1])
63
64          gear_msg = UInt8()
65          if signals[self.gear_forward_index] == 1:
66              gear_msg.data = GEAR_DIRECTION.FORWARD_GEAR.value
67          elif signals[self.gear_backward_index] == 1:
68              gear_msg.data = GEAR_DIRECTION.REVERSE_GEAR.value
69          else:
70              gear_msg.data = GEAR_DIRECTION.NEUTRAL_GEAR.value
71          self.pub_teleopt_gear.publish(gear_msg)
72
73          buttons_array = Int32MultiArray()
74          buttons_array.data = signals[4:self.gear_forward_index]
75          self.pub_teleopt_buttons.publish(buttons_array)
76
77          hats_array = Int32MultiArray()
78          hats_array.data = [signals[-2][0], signals[-2][1]]
79          self.pub_teleopt_hats.publish(hats_array)
80
```

```
81  if __name__ == "__main__":
82      CockpitLogitechCommunication()
```

ตารางที่ 13 *Cockpit_mode_communication*

Line	Code
1	#! /usr/bin/env python3
2	
3	import rospy
4	from std_msgs.msg import UInt8
5	from teleopt_function import CONTROL_MODE, DIAGNOSTIC_STATE, TCP_Communication
6	
7	class CockpitModeCommunication(object):
8	
9	def __init__(self) -> None:
10	rospy.init_node("cockpit_mode_communnication")
11	self.setupParam()
12	
13	self.pub_control_mode = rospy.Publisher("/cockpit/control_mode", UInt8, queue_size=1)
14	self.pub_diagnostic_mode = rospy.Publisher("/cockpit/diagnostic_mode/result", UInt8, queue_size=1)
15	self.sub_cockpit_control_mode = rospy.Subscriber("/cockpit/cockpit_control_mode", UInt8, self.callbackCockpitControlMode)
16	rospy.on_shutdown(self.requestShutdown)
17	
18	self.tcp = TCP_Communication(ip_address=self.golf_cpe_address, port=self.wan_port)
19	self.tcp.setupClient()
20	
21	self.Main()
22	
23	def __del__(self) -> None:
24	rospy.loginfo("Cockpit Mode Communication shutted down.")
25	
26	def Main(self) -> None:
27	while(not self.shutdown):
28	received_from_tcp = self.tcp.receive(self.tcp.client, 1024)
29	self.control_mode = CONTROL_MODE(int(received_from_tcp[0]))
30	self.diagnostic_mode = DIAGNOSTIC_STATE(int(received_from_tcp[1]))
31	
32	if(self.control_mode != self.prev_control_mode):
33	mode_msg = UInt8()
34	mode_msg.data = self.control_mode.value
35	self.pub_control_mode.publish(mode_msg)
36	
37	if(self.diagnostic_mode != self.prev_diagnostic_mode):
38	mode_msg = UInt8()
39	mode_msg.data = self.diagnostic_mode.value
40	self.pub_diagnostic_mode.publish(mode_msg)

```
41
42              self.tcp.send(self.tcp.client, [self.cockpit_control_mode.value])
43
                rospy.loginfo("cockpit:%d |control:%d |prev control: %d |diagnostic:%d |prev
    diagnostic:%d", self.cockpit_control_mode.value, self.control_mode.value,
    self.prev_control_mode.value, self.diagnostic_mode.value,
44  self.prev_diagnostic_mode.value)
45              self.prev_control_mode = self.control_mode
46              self.prev_diagnostic_mode = self.diagnostic_mode
47
48      def setupParam(self) -> None:
            self.golf_cpe_address =
49  rospy.get_param("/5G_CONNECTION/GOLF_CPE_NSA_ADDRESS")
50          self.wan_port        = rospy.get_param("/5G_CONNECTION/TCP_PORT_2")
51          self.shutdown        = False
52
53          self.cockpit_control_mode = CONTROL_MODE.AUTONOMOUS_MODE
54          self.control_mode        = CONTROL_MODE.AUTONOMOUS_MODE
55          self.prev_control_mode    = CONTROL_MODE.AUTONOMOUS_MODE
56          self.diagnostic_mode      = DIAGNOSTIC_STATE.NORMAL_STATE
57          self.prev_diagnostic_mode = DIAGNOSTIC_STATE.NORMAL_STATE
58
59      def requestShutdown(self) -> None:
60          self.shutdown = True
61
62      def callbackCockpitControlMode(self, msg: UInt8) -> None:
63          self.cockpit_control_mode = CONTROL_MODE(msg.data)
64
65
66  if __name__ == "__main__":
67      CockpitModeCommunication()
```

ตารางที่ 14 *Golf_camera_sender*

Line	Code
1	#! /usr/bin/env python
2	
3	import sys
4	import cv2
5	import rospy
6	from teleopt_function import UDP_Communication
7	
8	class GolfCameraSender():
9	def __init__(self, camera_index=0):
10	rospy.init_node("golf_camera_sender_" + str(camera_index))
11	self.setupParam()
12	
13	rospy.on_shutdown(self.requestShutdown)
14	
15	self.camera_index = int(camera_index)
16	self.udp = UDP_Communication(ip_address=self.cockpit_cpe_address, port=self.wan_port + self.camera_index)
17	self.udp.setupClient()
18	
19	self.Main()
20	
21	def __del__(self):
22	rospy.loginfo("Golf Camera Sender %d shutted down.", self.camera_index)
23	
24	def Main(self):
25	cap = cv2.VideoCapture(self.camera_index)
26	while(cap.isOpened() and not self.shutdown):
27	_, frame = cap.read()
28	compress_img = cv2.imencode('.jpg', frame)[1]
29	compress_img_byte = compress_img.tostring()
30	self.udp.send(compress_img_byte)
31	cap.release()
32	cv2.destroyAllWindows()
33	self.udp.client.close()
34	
35	def setupParam(self):
36	self.cockpit_cpe_address = rospy.get_param("/5G_CONNECTION/COCKPIT_CPE_NSA_ADDRESS")
37	self.wan_port = rospy.get_param("/5G_CONNECTION/UDP_PORT_1")
38	self.shutdown = False
39	
40	def requestShutdown(self):
41	self.shutdown = True

```
42
43
44  def initialized():
45      if(len(sys.argv) == 2 or len(sys.argv) == 4):
46          cam = GolfCameraSender(sys.argv[1])
47      else:
            print("please fill out the arguments following the below structure.\n>>> python
48  python_file camera_index")
49
50  if __name__ == "__main__":
51      initialized()
```

ตารางที่ 15 *Golf_latency_measurement*

Line	Code
1	#! /usr/bin/env python3
2	
3	import rospy
4	from std_msgs.msg import Int32
5	from teleopt_function import TCP_Communication
6	
7	class GolfLatencyMeasurement(object):
8	
9	def __init__(self) -> None:
10	rospy.init_node("golf_latency_measurement")
11	self.setupParam()
12	
13	self.pub_golf_latency = rospy.Publisher("/golf/latency", Int32, queue_size=1)
14	self.pub_golf_sub_latency = rospy.Publisher("/golf/sub_latency", Int32, queue_size=1)
15	rospy.on_shutdown(self.requestShutdown)
16	
17	self.tcp = TCP_Communication(ip_address=self.golf_com_address, port=self.lan_port)
18	self.tcp.setupServer()
19	
20	self.Main()
21	
22	def __del__(self) -> None:
23	rospy.loginfo("Golf Latency Measurement shutted down.")
24	
25	def Main(self) -> None:
26	while(not self.shutdown):
27	golf_send_time = rospy.Time.now().to_sec()
28	self.tcp.send(self.tcp.server, golf_send_time)
29	received_from_tcp = self.tcp.receive(self.tcp.server, 1024)
30	golf_receive_time = received_from_tcp
31	
32	latency = int((rospy.Time.now().to_sec() - golf_receive_time) * 1000 / 2)
33	self.pub_golf_latency.publish(latency)
34	self.pub_golf_sub_latency.publish(latency)
35	
36	
37	received_from_tcp = self.tcp.receive(self.tcp.server, 1024)
38	self.tcp.send(self.tcp.server, received_from_tcp)
39	
40	rospy.loginfo("golf send:%f |golf recv: %f |latency:%f", golf_send_time, golf_receive_time, latency)
41	self.main_rate.sleep()

```
42
43      def setupParam(self) -> None:
            self.golf_com_address =
44  rospy.get_param("/5G_CONNECTION/GOLF_COM_ADDRESS")
45          self.lan_port        = rospy.get_param("/5G_CONNECTION/TCP_PORT_3")
46          self.shutdown         = False
47          self.main_rate       = rospy.Rate(10)
48
49      def requestShutdown(self) -> None:
50          self.shutdown = True
51
52  if __name__ == "__main__":
53      GolfLatencyMeasurement()
```

ตารางที่ 16 *Golf_logitech_communication*

Line	Code
1	#! /usr/bin/env python3
2	
3	import rospy
4	from std_msgs.msg import UInt8, Float32, Float64, Int32MultiArray
5	from teleopt_function import GEAR_DIRECTION, TCP_Communication
6	
7	class GolfLogitechCommunication(object):
8	
9	def __init__(self) -> None:
10	rospy.init_node("golf_logitech_communication")
11	self.setupConfig()
12	
13	self.pub_teleopt_steering = rospy.Publisher("/golf/teleopt/steering_command", Float32, queue_size=1)
14	self.pub_teleopt_velocity = rospy.Publisher("/golf/teleopt/velocity_command", Float32, queue_size=1)
15	self.pub_teleopt_brake = rospy.Publisher("/golf/teleopt/brake_command", Float32, queue_size=1)
16	self.pub_teleopt_gear = rospy.Publisher("/golf/teleopt/gear_command", UInt8, queue_size=1)
17	self.pub_teleopt_buttons = rospy.Publisher("/golf/teleopt/buttons_command", Int32MultiArray, queue_size=1)
18	self.pub_teleopt_hats = rospy.Publisher("/golf/teleopt/hats_command", Int32MultiArray, queue_size=1)
19	self.pub_teleopt_time = rospy.Publisher("/golf/teleopt/time", Float64, queue_size=1)
20	rospy.on_shutdown(self.requestShutdown)
21	
22	self.tcp = TCP_Communication(ip_address=self.golf_com_address, port=self.lan_port)
23	self.tcp.setupServer()
24	
25	self.Main()
26	
27	def __del__(self) -> None:
28	rospy.loginfo("Golf Logitech Communnication shutted down.")
29	
30	def Main(self) -> None:
31	while(not self.shutdown):
32	signals = self.tcp.receive(self.tcp.server, 1024)
33	rospy.loginfo(signals)
34	
35	self.publishAllSignals(signals)
36	self.tcp.server.close()
37	

```
38      def setupConfig(self) -> None:
39          self.shutdown           = False
            self.golf_com_address    =
40  rospy.get_param("/5G_CONNECTION/GOLF_COM_ADDRESS")
41          self.lan_port           = rospy.get_param("/5G_CONNECTION/TCP_PORT_1") #28000
42          self.steering_index      = rospy.get_param("/LogitechG29/STEERING_WHEEL")
43          self.velocity_index      = rospy.get_param("/LogitechG29/ACCEL_PEDAL")
44          self.brake_index         = rospy.get_param("/LogitechG29/BRAKE_PEDAL")
45          self.gear_forward_index  = rospy.get_param("/LogitechG29/GEAR_MID_FORWARD")
            self.gear_backward_index =
46  rospy.get_param("/LogitechG29/GEAR_MID_BACKWARD")
47
48      def requestShutdown(self) -> None:
49          self.shutdown = True
50
51      def publishAllSignals(self, signals) -> None:
52          self.pub_teleopt_steering.publish(signals[self.steering_index])
53          self.pub_teleopt_velocity.publish(signals[self.velocity_index])
54          self.pub_teleopt_brake.publish(signals[self.brake_index])
55          self.pub_teleopt_time.publish(signals[-1])
56
57          gear_msg = UInt8()
58          if signals[self.gear_forward_index] == 1:
59              gear_msg.data = GEAR_DIRECTION.FORWARD_GEAR.value
60          elif signals[self.gear_backward_index] == 1:
61              gear_msg.data = GEAR_DIRECTION.REVERSE_GEAR.value
62          else:
63              gear_msg.data = GEAR_DIRECTION.NEUTRAL_GEAR.value
64          self.pub_teleopt_gear.publish(gear_msg)
65
66          buttons_array = Int32MultiArray()
67          buttons_array.data = signals[4:self.gear_forward_index]
68          self.pub_teleopt_buttons.publish(buttons_array)
69
70          hats_array = Int32MultiArray()
71          hats_array.data = [signals[-2][0], signals[-2][1]]
72          self.pub_teleopt_hats.publish(hats_array)
73
74  if __name__ == "__main__":
75      GolfLogitechCommunication()
```

ตารางที่ 17 *Golf_mode_communication*

```
Line  Code
   1  #! /usr/bin/env python3
   2
   3  import rospy
   4  from std_msgs.msg import UInt8
   5  from teleopt_function import CONTROL_MODE, DIAGNOSTIC_STATE, TCP_Communication
   6
   7  class GolfModeCommunication(object):
   8
   9      def __init__(self) -> None:
  10          rospy.init_node("golf_mode_communication")
  11          self.setupParam()
  12
  13          self.pub_cockpit_control_mode = rospy.Publisher("/golf/control_mode", UInt8, queue_size=1)
  14          self.sub_control_mode        = rospy.Subscriber("/golf/control_mode", UInt8, self.callbackControlMode)
  15          self.sub_diagnostic_mode     = rospy.Subscriber("/golf/diagnostic_mode/result", UInt8, self.callbackDiagnosticMode)
  16          rospy.on_shutdown(self.requestShutdown)
  17
  18          self.tcp = TCP_Communication(ip_address=self.golf_com_address, port=self.lan_port)
  19          self.tcp.setupServer()
  20
  21          self.Main()
  22
  23      def __del__(self) -> None:
  24          rospy.loginfo("Golf Mode Communication shutted down.")
  25
  26      def Main(self) -> None:
  27          while(not self.shutdown):
  28              self.tcp.send(self.tcp.server, [self.control_mode.value, self.diagnostic_mode.value])
  29              receive_from_tcp = self.tcp.receive(self.tcp.server, 1024)
  30
  31              self.cockpit_control_mode = CONTROL_MODE(receive_from_tcp[0])
  32              if(self.cockpit_control_mode != self.prev_cockpit_control_mode):
  33                  mode_msg = UInt8()
  34                  mode_msg.data = self.cockpit_control_mode.value
  35                  self.pub_cockpit_control_mode.publish(mode_msg)
  36
  37              rospy.loginfo("cockpit:%d |prev cockpit:%d |control: %d |diagnostic:%d", self.cockpit_control_mode.value, self.prev_cockpit_control_mode.value, self.control_mode.value, self.diagnostic_mode.value)
  38              self.prev_cockpit_control_mode = self.cockpit_control_mode
```

```
39            self.main_rate.sleep()
40
41        def setupParam(self) -> None:
42            self.golf_com_address = rospy.get_param("/5G_CONNECTION/GOLF_COM_ADDRESS")
43            self.lan_port        = rospy.get_param("/5G_CONNECTION/TCP_PORT_2")
44            self.shutdown        = False
45
46            self.cockpit_control_mode      = CONTROL_MODE.AUTONOMOUS_MODE
47            self.prev_cockpit_control_mode = CONTROL_MODE.AUTONOMOUS_MODE
48            self.control_mode              = CONTROL_MODE.AUTONOMOUS_MODE
49            self.diagnostic_mode           = DIAGNOSTIC_STATE.NORMAL_STATE
50            self.main_rate                 = rospy.Rate(30)
51
52        def requestShutdown(self) -> None:
53            self.shutdown = True
54
55        def callbackControlMode(self, msg: UInt8) -> None:
56            self.control_mode = CONTROL_MODE(msg.data)
57
58        def callbackDiagnosticMode(self, msg: UInt8) -> None:
59            self.diagnostic_mode = DIAGNOSTIC_STATE(msg.data)
60
61    if __name__ == "__main__":
62        GolfModeCommunication()
```

ตารางที่ 18 *Teleopt_function*

Line	Code
1	#! /usr/bin/env python3
2	
3	from enum import Enum
4	
5	class CONTROL_MODE(Enum):
6	AUTONOMOUS_MODE = 0
7	TELEOPERATED_MODE = 1
8	EMERGENCY_FROM_COCKPIT_MODE = 2
9	EMERGENCY_FROM_GOLF_MODE = 3
10	MANUAL_MODE = 4
11	
12	class DIAGNOSTIC_STATE(Enum):
13	NORMAL_STATE = 0
14	REQUEST_CHANGING_STATE = 1
15	HIGH_LATENCY_STATE = 2
16	
17	class GEAR_DIRECTION(Enum):
18	NEUTRAL_GEAR = 0
19	FORWARD_GEAR = 1
20	REVERSE_GEAR = 2
21	
22	
23	class LogitechG29():
24	
25	import yaml
26	import pygame
27	from time import sleep
28	
29	def __init__(self):
30	self.pygame.init()
31	self.pygame.joystick.init()
32	self.clock = self.pygame.time.Clock()
33	self.joystick_name = "Logitech G29 Driving Force Racing Wheel"
34	
35	self.isLogitechConnected()
36	
37	def isLogitechConnected(self):
38	while True:
39	joys = {}
40	for i in range(self.pygame.joystick.get_count()):
41	joy = self.pygame.joystick.Joystick(i)
42	joys[joy.get_name()] = i

```
43            if self.joystick_name in joys.keys():
44                joy_index = joys[self.joystick_name]
45                self.joy = self.pygame.joystick.Joystick(joy_index)
46                self.joy.init()
47                print("[CONNECTED] Logitech G29 is now connected.")
48                break
49            else:
50                print("[WAITING] waiting for Logitech G29.")
51                self.sleep(1)
52
53    def readLogitech(self):
54        signals = []
55        try:
56            self.pygame.event.get()
57            for i in range(self.joy.get_numaxes()):
58                signals.append(round(self.joy.get_axis(i), 5))
59            for i in range(self.joy.get_numbuttons()):
60                signals.append(self.joy.get_button(i))
61            for i in range(self.joy.get_numhats()):
62                signals.append(self.joy.get_hat(i))
63            return signals
64        except Exception as err:
65            print("[ERROR]", err)
66
67    def getIndex(self, name):
        with
    open("/home/smrc/total_ws/teleoptGolf_ws/src/teleopt_control/params/smrc_config.ya
68  ml") as f:
69            load = self.yaml.load_all(f)
70            for logitech in load:
71                return logitech["Logitech_G29"][name]
72
73  class SerialCommunication():
74
75    import serial
76    import serial.tools.list_ports as stlp
77    from time import sleep
78
79    def __init__(self, serial_number = "", baudrate = 115200):
80        self.serial_number = serial_number
81        self.baudrate = baudrate
82        self.port = self.searchSerialPortBySerialNumber()
83        self.setupSerial()
84
85    def searchSerialPortBySerialNumber(self):
86        for port in self.stlp.comports():
```

```
87              if port.serial_number == self.serial_number:
88                  print("the desired port is: ", port.device)
89                  return port.device
90          raise IOError("could not find the desired port.")
91
92      def setupSerial(self):
93          self.ser = self.serial.Serial(port=self.port, baudrate=self.baudrate)
94          print(self.ser.readline().decode("ascii").rstrip())
95
96      def send(self, floatArray):
97          string_message = "<"
98          string_message += str(floatArray[0])
99
100         for i in range(1, len(floatArray)):
101             string_message += "," + str(floatArray[i])
102         string_message += ">"
103
104         self.ser.write(string_message.encode())
105
106     def receive(self):
107         string_recv = self.ser.readline().decode("ascii").rstrip()
108         return [float(i) for i in string_recv.split(",")]
109
110 class TCP_Communication():
111
112     import socket
113     import pickle
114
115     def __init__(self,ip_address="localhost", port=3000):
116         self.ip_address = ip_address
117         self.port = port
118         self.message_received = b''
119
120     def setupServer(self):
121         sock = self.socket.socket(self.socket.AF_INET, self.socket.SOCK_STREAM)
122         sock.setsockopt(self.socket.SOL_SOCKET, self.socket.SO_REUSEADDR, 1)
123         sock.bind((self.ip_address, self.port))
124         sock.listen(1)
125         self.server, address = sock.accept()
126         print("[CONNECTED] IP: {} has made a connection.".format(address))
127
128     def setupClient(self):
129         self.client = self.socket.socket(self.socket.AF_INET, self.socket.SOCK_STREAM)
130         self.client.setsockopt(self.socket.SOL_SOCKET, self.socket.SO_REUSEADDR, 1)
131         try:
```

```
132             self.client.connect((self.ip_address, self.port))
133         except:
134             self.setupClient()
135 
136     def send(self, socket_object, msg=[]):
137         message_byte = b'Start' + self.pickle.dumps(msg) + b'ComPleteLy'
138         socket_object.sendall(message_byte)
139 
140     def receive(self, socket_object, buffer_size=256): # maximum buffer size: 1048576
141         while b'Start' not in self.message_received:
142             self.message_received += socket_object.recv(buffer_size)
143 
144         while b'ComPleteLy' not in self.message_received:
145             self.message_received += socket_object.recv(buffer_size)
146 
147         start_index = self.message_received.find(b'Start') + 5
148         end_index = self.message_received.find(b'ComPleteLy')
149         # print(len(self.message_received[start_index:end_index]))
150         used_byte_message = self.message_received[start_index:end_index]
151         self.message_received = self.message_received[end_index + 10:]
152 
153         return self.pickle.loads(used_byte_message)
154 
155 class UDP_Communication():
156 
157     import socket
158     import struct
159 
160     MAX_DGRAM = 2**16
161     MAX_MESSAGE_DGRAM = MAX_DGRAM - 64
162 
163     def __init__(self, ip_address="localhost", port=3000):
164         self.ip_address = ip_address
165         self.port = port
166 
167     def setupServer(self):
168         self.server = self.socket.socket(self.socket.AF_INET, self.socket.SOCK_DGRAM)
169         self.server.bind((self.ip_address, self.port))
170 
171     def setupClient(self):
172         self.client = self.socket.socket(self.socket.AF_INET, self.socket.SOCK_DGRAM)
173 
174     def send(self, msg_byte=b''):
175         byte_message_size = len(msg_byte)
176         cycle = int(byte_message_size/self.MAX_MESSAGE_DGRAM) + 1
```

```
177         array_pose_start = 0
178         while cycle:
              array_pose_end = min(byte_message_size, array_pose_start +
179   self.MAX_MESSAGE_DGRAM)
180           self.client.sendto(self.struct.pack("B", cycle) +
181                   msg_byte[array_pose_start:array_pose_end],
182                   (self.ip_address, self.port))
183
184           array_pose_start = array_pose_end
185           cycle -= 1
186
187       def receive(self):
188         message = b''
189         while True:
190           recv_message, address = self.server.recvfrom(self.MAX_DGRAM)
191           if self.struct.unpack("B", recv_message[0])[0] > 1:
192             message += recv_message[1:]
193           else:
194             message += recv_message[1:]
195             return message
196
197       def dumpBuffer(self):
198         while True:
199           recv_message, address = self.server.recvfrom(self.MAX_DGRAM)
200           if self.struct.unpack("B", recv_message[0])[0] == 1:
201             print("[SUCCESS] emptying buffer.")
202             break
```

ตารางที่ 19 *Cockpit_camera_receiver*

Line	Code
1	#! /usr/bin/env python
2	
3	import sys
4	import cv2
5	import rospy
6	import numpy as np
7	
8	sys.path.append("/home/smrc/total_ws/research_ws/src/golf/scripts")
9	from teleopt_function import UDP_Communication
10	
11	class CockpitCameraReceiver():
12	def __init__(self, monitor_index=0):
13	rospy.init_node("cockpit_camera_receiver_" + str(monitor_index))
14	self.setupParam()
15	
16	rospy.on_shutdown(self.requestShutdown)
17	
18	self.monitor_index = int(monitor_index) self.udp = UDP_Communication(ip_address=self.cockpit_com_address,
19	port=self.lan_port + self.monitor_index)
20	self.udp.setupServer()
21	
22	self.udp.dumpBuffer()
23	self.Main()
24	
25	def __del__(self):
26	rospy.loginfo("Cockpit Camera Receiver %d shutted down.", self.monitor_index)
27	
28	def Main(self):
29	while(not self.shutdown):
30	img_byte = self.udp.receive()
31	img = cv2.imdecode(np.fromstring(img_byte, dtype=np.uint8), 1) cv2.namedWindow("MONITOR " + str(self.monitor_index + 1),
32	cv2.WINDOW_NORMAL)
33	try:
34	cv2.imshow("MONITOR " + str(self.monitor_index + 1), img)
35	except:
36	self.udp.setupServer()
37	self.udp.dumpBuffer()
38	if(cv2.waitKey(1) & 0xFF == ord('q')):
39	break
40	
41	# self.udp.dumpBuffer()

```
42
43          cv2.destroyAllWindows()
44          self.udp.server.close()
45
46      def setupParam(self):
            self.cockpit_com_address =
47  rospy.get_param("/5G_CONNECTION/COCKPIT_COM_ADDRESS")
48          self.lan_port           = rospy.get_param("/5G_CONNECTION/UDP_PORT_1")
49          self.shutdown           = False
50
51      def requestShutdown(self):
52          self.shutdown = True
53
54  def initialized():
55      if(len(sys.argv) == 2 or len(sys.argv) == 4):
56          cam = CockpitCameraReceiver(sys.argv[1])
57      else:
            print("please fill out the arguments following the below structure.\n>>> python
58  python_file monitor_index")
59
60  if __name__ == "__main__":
61      initialized()
```

ตารางที่ 20 *Cockpit_mode_command*

Line	Code
1	#ifndef COCKPIT_MODE_COMMAND
2	#define COCKPIT_MODE_COMMAND
3	
4	enum CONTROL_MODE {AUTONOMOUS_MODE, TELEOPERATED_MODE, EMERGENCY_FROM_COCKPIT_MODE, EMERGENCY_FROM_GOLF_MODE, MANUAL_MODE};
5	enum DIAGNOSTIC_STATE {NORMAL_STATE, REQUEST_CHANGING_STATE, HIGH_LATENCY_STATE};
6	enum SWITCH_STATE {DEACTIVATED_STATE, ACTIVATED_STATE};
7	
8	#define PININ_TELEOPT 4
9	#define PININ_AUTONOMOUS 3
10	#define PININ_EMERGENCY 2
11	#define PINOUT_LIGHT_TELEOPT 7
12	#define PINOUT_LIGHT_AUTONOMOUS 6
13	#define PINOUT_LIGHT_EMERGENCY 5
14	#define PINOUT_LIGHT_LATENCY_RED 9
15	#define PINOUT_LIGHT_LATENCY_GREEN 10
16	#define PINOUT_LIGHT_LATENCY_BLUE 11
17	
18	int commands[10];
19	char inputBuffer[20];
20	bool isBufferCompleted = false;
21	unsigned int inputIndex = 0;
22	unsigned int commandIndex = 0;
23	
24	CONTROL_MODE control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
25	CONTROL_MODE cockpit_control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
26	CONTROL_MODE prev_cockpit_control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
27	DIAGNOSTIC_STATE diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;
28	unsigned int latency = 0;
29	
30	SWITCH_STATE teleoptSwitch = SWITCH_STATE::DEACTIVATED_STATE;
31	SWITCH_STATE autonomousSwitch = SWITCH_STATE::DEACTIVATED_STATE;
32	SWITCH_STATE emergencySwitch = SWITCH_STATE::DEACTIVATED_STATE;
33	
34	unsigned int rate = 20;
35	float prev_time = millis();
36	
37	void setup() {
38	Serial.begin(115200);
39	

```
40    pinMode(PININ_TELEOPT, INPUT_PULLUP);
41    pinMode(PININ_AUTONOMOUS, INPUT_PULLUP);
42    pinMode(PININ_EMERGENCY, INPUT_PULLUP);
43    pinMode(PINOUT_LIGHT_TELEOPT, OUTPUT);
44    pinMode(PINOUT_LIGHT_AUTONOMOUS, OUTPUT);
45    pinMode(PINOUT_LIGHT_EMERGENCY, OUTPUT);
46    pinMode(PINOUT_LIGHT_LATENCY_RED, OUTPUT);
47    pinMode(PINOUT_LIGHT_LATENCY_GREEN, OUTPUT);
48    pinMode(PINOUT_LIGHT_LATENCY_BLUE, OUTPUT);
49  }
50
51  void loop() {
52  //  if(millis() - prev_time > 1000/rate) {
53    if(Serial.available()) {
54      teleoptSwitch = digitalRead(PININ_TELEOPT);
55      autonomousSwitch = digitalRead(PININ_AUTONOMOUS);
56      emergencySwitch = digitalRead(PININ_EMERGENCY);
57
      if((control_mode != CONTROL_MODE::EMERGENCY_FROM_GOLF_MODE &&
    control_mode != CONTROL_MODE::MANUAL_MODE) && diagnostic_mode !=
58  DIAGNOSTIC_STATE::HIGH_LATENCY_STATE) {
59        if(autonomousSwitch == 1) {
60          cockpit_control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
61          prev_cockpit_control_mode = cockpit_control_mode;
62        }
63
64        if(teleoptSwitch == 1) {
65          cockpit_control_mode = CONTROL_MODE::TELEOPERATED_MODE;
66          prev_cockpit_control_mode = cockpit_control_mode;
67        }
68
        if(emergencySwitch == 1) cockpit_control_mode =
69  CONTROL_MODE::EMERGENCY_FROM_COCKPIT_MODE;
70        else cockpit_control_mode = prev_cockpit_control_mode;
71      }
72
      if(control_mode == CONTROL_MODE::EMERGENCY_FROM_GOLF_MODE ||
73  diagnostic_mode == DIAGNOSTIC_STATE::HIGH_LATENCY_STATE) {
74        setMultipleLightState("OFF", "OFF", "ON");
75      } else if(control_mode == CONTROL_MODE::MANUAL_MODE) {
76        setMultipleLightState("OFF", "OFF", "OFF");
77      } else {
78        if(cockpit_control_mode == CONTROL_MODE::EMERGENCY_FROM_COCKPIT_MODE) {
79          setMultipleLightState("OFF", "OFF", "ON");
80        } else {
81          if(cockpit_control_mode == CONTROL_MODE::AUTONOMOUS_MODE) {
```

```
82            setMultipleLightState("ON", "OFF", "NONE");
83          } else if(cockpit_control_mode == CONTROL_MODE::TELEOPERATED_MODE) {
84            setMultipleLightState("OFF", "ON", "NONE");
85          }
86
87          if(diagnostic_mode == DIAGNOSTIC_STATE::REQUEST_CHANGING_STATE) {
88            setMultipleLightState("NONE", "NONE", "ON");
89          } else {
90            setMultipleLightState("NONE", "NONE", "OFF");
91          }
92        }
93      }
94
95      setLatencyLight();
96      writeToSerial();
97  //    prev_time = millis();
98    }
99  }
100
101 void serialEvent() {
102   while (Serial.available()) {
103     char inChar = (char)Serial.read();
104
105     if (inChar == '\n') {
106       commands[commandIndex] = atoi(inputBuffer);
107       isBufferCompleted = true;
108
109       inputIndex = 0;
110       commandIndex = 0;
111       memset(inputBuffer, 0, sizeof(inputBuffer));
112
113     } else if (inChar == ',') {
114       commands[commandIndex] = atoi(inputBuffer);
115       commandIndex++;
116
117       inputIndex = 0;
118       memset(inputBuffer, 0, sizeof(inputBuffer));
119
120     } else {
121       inputBuffer[inputIndex] = inChar;
122       inputIndex++;
123     }
124
125   }
126
```

```
127   if(isBufferCompleted) {
128     control_mode = commands[0];
129     diagnostic_mode = commands[1];
130     latency = commands[2];
131
132     isBufferCompleted = false;
133   }
134
135 }
136
137 void writeToSerial() {
138   if(Serial.available()) {
        int message_out[8] = {control_mode, diagnostic_mode, cockpit_control_mode,
      prev_cockpit_control_mode, autonomousSwitch, teleoptSwitch, emergencySwitch,
139   latency};
140     for(unsigned int i = 0; i < 7; i++) {
141       Serial.print(message_out[i]);
142       Serial.print(",");
143     }
144     Serial.println(message_out[7]);
145   }
146 }
147
148 void setLatencyLight() {
149   if(latency < 50) {
150     analogWrite(PINOUT_LIGHT_LATENCY_RED, 0);
151     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 200);
152     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 0);
153   } else if(latency < 100) {
154     analogWrite(PINOUT_LIGHT_LATENCY_RED, 0);
155     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 255);
156     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 255);
157   } else if(latency < 200) {
158     analogWrite(PINOUT_LIGHT_LATENCY_RED, 255);
159     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 150);
160     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 0);
161   } else if(latency < 500) {
162     analogWrite(PINOUT_LIGHT_LATENCY_RED, 255);
163     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 0);
164     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 255);
165   } else {
166     analogWrite(PINOUT_LIGHT_LATENCY_RED, 255);
167     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 0);
168     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 0);
169   }
170 }
```

```
171
172  void setLightStatus(const unsigned int &pin, const char* state) {
173    if(state == "OFF") digitalWrite(pin, HIGH);
174    else if(state == "ON") digitalWrite(pin, LOW);
175  }
176
     void setMultipleLightState(const char* autonomous_state, const char* teleop_state, const
177  char* emergency_state) {
178    setLightStatus(PINOUT_LIGHT_TELEOPT, teleop_state);
179    setLightStatus(PINOUT_LIGHT_AUTONOMOUS, autonomous_state);
180    setLightStatus(PINOUT_LIGHT_EMERGENCY, emergency_state);
181  }
182
183  #endif
```

ตารางที่ 21 *Golf_cruise_command*

Line	Code
1	#ifndef COCKPIT_MODE_COMMAND
2	#define COCKPIT_MODE_COMMAND
3	
4	enum CONTROL_MODE {AUTONOMOUS_MODE, TELEOPERATED_MODE, EMERGENCY_FROM_COCKPIT_MODE, EMERGENCY_FROM_GOLF_MODE, MANUAL_MODE};
5	enum DIAGNOSTIC_STATE {NORMAL_STATE, REQUEST_CHANGING_STATE, HIGH_LATENCY_STATE};
6	enum SWITCH_STATE {DEACTIVATED_STATE, ACTIVATED_STATE};
7	
8	#define PININ_TELEOPT 4
9	#define PININ_AUTONOMOUS 3
10	#define PININ_EMERGENCY 2
11	#define PINOUT_LIGHT_TELEOPT 7
12	#define PINOUT_LIGHT_AUTONOMOUS 6
13	#define PINOUT_LIGHT_EMERGENCY 5
14	#define PINOUT_LIGHT_LATENCY_RED 9
15	#define PINOUT_LIGHT_LATENCY_GREEN 10
16	#define PINOUT_LIGHT_LATENCY_BLUE 11
17	
18	int commands[10];
19	char inputBuffer[20];
20	bool isBufferCompleted = false;
21	unsigned int inputIndex = 0;
22	unsigned int commandIndex = 0;
23	
24	CONTROL_MODE control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
25	CONTROL_MODE cockpit_control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
26	CONTROL_MODE prev_cockpit_control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
27	DIAGNOSTIC_STATE diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;
28	unsigned int latency = 0;
29	
30	SWITCH_STATE teleoptSwitch = SWITCH_STATE::DEACTIVATED_STATE;
31	SWITCH_STATE autonomousSwitch = SWITCH_STATE::DEACTIVATED_STATE;
32	SWITCH_STATE emergencySwitch = SWITCH_STATE::DEACTIVATED_STATE;
33	
34	unsigned int rate = 20;
35	float prev_time = millis();
36	
37	void setup() {
38	Serial.begin(115200);
39	

```
40    pinMode(PININ_TELEOPT, INPUT_PULLUP);
41    pinMode(PININ_AUTONOMOUS, INPUT_PULLUP);
42    pinMode(PININ_EMERGENCY, INPUT_PULLUP);
43    pinMode(PINOUT_LIGHT_TELEOPT, OUTPUT);
44    pinMode(PINOUT_LIGHT_AUTONOMOUS, OUTPUT);
45    pinMode(PINOUT_LIGHT_EMERGENCY, OUTPUT);
46    pinMode(PINOUT_LIGHT_LATENCY_RED, OUTPUT);
47    pinMode(PINOUT_LIGHT_LATENCY_GREEN, OUTPUT);
48    pinMode(PINOUT_LIGHT_LATENCY_BLUE, OUTPUT);
49  }
50
51  void loop() {
52  //  if(millis() - prev_time > 1000/rate) {
53    if(Serial.available()) {
54      teleoptSwitch = digitalRead(PININ_TELEOPT);
55      autonomousSwitch = digitalRead(PININ_AUTONOMOUS);
56      emergencySwitch = digitalRead(PININ_EMERGENCY);
57
      if((control_mode != CONTROL_MODE::EMERGENCY_FROM_GOLF_MODE &&
    control_mode != CONTROL_MODE::MANUAL_MODE) && diagnostic_mode !=
58  DIAGNOSTIC_STATE::HIGH_LATENCY_STATE) {
59        if(autonomousSwitch == 1) {
60          cockpit_control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
61          prev_cockpit_control_mode = cockpit_control_mode;
62        }
63
64        if(teleoptSwitch == 1) {
65          cockpit_control_mode = CONTROL_MODE::TELEOPERATED_MODE;
66          prev_cockpit_control_mode = cockpit_control_mode;
67        }
68
        if(emergencySwitch == 1) cockpit_control_mode =
69  CONTROL_MODE::EMERGENCY_FROM_COCKPIT_MODE;
70        else cockpit_control_mode = prev_cockpit_control_mode;
71      }
72
      if(control_mode == CONTROL_MODE::EMERGENCY_FROM_GOLF_MODE ||
73  diagnostic_mode == DIAGNOSTIC_STATE::HIGH_LATENCY_STATE) {
74        setMultipleLightState("OFF", "OFF", "ON");
75      } else if(control_mode == CONTROL_MODE::MANUAL_MODE) {
76        setMultipleLightState("OFF", "OFF", "OFF");
77      } else {
78        if(cockpit_control_mode == CONTROL_MODE::EMERGENCY_FROM_COCKPIT_MODE) {
79          setMultipleLightState("OFF", "OFF", "ON");
80        } else {
81          if(cockpit_control_mode == CONTROL_MODE::AUTONOMOUS_MODE) {
```

```
82            setMultipleLightState("ON", "OFF", "NONE");
83          } else if(cockpit_control_mode == CONTROL_MODE::TELEOPERATED_MODE) {
84            setMultipleLightState("OFF", "ON", "NONE");
85          }
86
87          if(diagnostic_mode == DIAGNOSTIC_STATE::REQUEST_CHANGING_STATE) {
88            setMultipleLightState("NONE", "NONE", "ON");
89          } else {
90            setMultipleLightState("NONE", "NONE", "OFF");
91          }
92        }
93      }
94
95      setLatencyLight();
96      writeToSerial();
97  //    prev_time = millis();
98    }
99  }
100
101 void serialEvent() {
102   while (Serial.available()) {
103     char inChar = (char)Serial.read();
104
105     if (inChar == '\n') {
106       commands[commandIndex] = atoi(inputBuffer);
107       isBufferCompleted = true;
108
109       inputIndex = 0;
110       commandIndex = 0;
111       memset(inputBuffer, 0, sizeof(inputBuffer));
112
113     } else if (inChar == ',') {
114       commands[commandIndex] = atoi(inputBuffer);
115       commandIndex++;
116
117       inputIndex = 0;
118       memset(inputBuffer, 0, sizeof(inputBuffer));
119
120     } else {
121       inputBuffer[inputIndex] = inChar;
122       inputIndex++;
123     }
124
125   }
126
```

```
127   if(isBufferCompleted) {
128     control_mode = commands[0];
129     diagnostic_mode = commands[1];
130     latency = commands[2];
131
132     isBufferCompleted = false;
133   }
134
135 }
136
137 void writeToSerial() {
138   if(Serial.available()) {
        int message_out[8] = {control_mode, diagnostic_mode, cockpit_control_mode,
      prev_cockpit_control_mode, autonomousSwitch, teleoptSwitch, emergencySwitch,
139   latency};
140     for(unsigned int i = 0; i < 7; i++) {
141       Serial.print(message_out[i]);
142       Serial.print(",");
143     }
144     Serial.println(message_out[7]);
145   }
146 }
147
148 void setLatencyLight() {
149   if(latency < 50) {
150     analogWrite(PINOUT_LIGHT_LATENCY_RED, 0);
151     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 200);
152     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 0);
153   } else if(latency < 100) {
154     analogWrite(PINOUT_LIGHT_LATENCY_RED, 0);
155     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 255);
156     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 255);
157   } else if(latency < 200) {
158     analogWrite(PINOUT_LIGHT_LATENCY_RED, 255);
159     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 150);
160     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 0);
161   } else if(latency < 500) {
162     analogWrite(PINOUT_LIGHT_LATENCY_RED, 255);
163     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 0);
164     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 255);
165   } else {
166     analogWrite(PINOUT_LIGHT_LATENCY_RED, 255);
167     analogWrite(PINOUT_LIGHT_LATENCY_GREEN, 0);
168     analogWrite(PINOUT_LIGHT_LATENCY_BLUE, 0);
169   }
170 }
```

```
171
172  void setLightStatus(const unsigned int &pin, const char* state) {
173    if(state == "OFF") digitalWrite(pin, HIGH);
174    else if(state == "ON") digitalWrite(pin, LOW);
175  }
176
     void setMultipleLightState(const char* autonomous_state, const char* teleop_state, const
177  char* emergency_state) {
178    setLightStatus(PINOUT_LIGHT_TELEOPT, teleop_state);
179    setLightStatus(PINOUT_LIGHT_AUTONOMOUS, autonomous_state);
180    setLightStatus(PINOUT_LIGHT_EMERGENCY, emergency_state);
181  }
182
183  #endif
```

ตารางที่ 22 *Golf_steering_command*

Line	Code
1	#ifndef GOLF_STEERING_COMMAND
2	#define GOLF_STEERING_COMMAND
3	
4	enum CONTROL_MODE {AUTONOMOUS_MODE, TELEOPERATED_MODE, EMERGENCY_FROM_COCKPIT_MODE, EMERGENCY_FROM_GOLF_MODE, MANUAL_MODE};
5	enum DIAGNOSTIC_STATE {NORMAL_STATE, REQUEST_CHANGING_STATE, HIGH_LATENCY_STATE};
6	
7	#define PINOUT_PWM 5
8	#define PINOUT_DIR1 31
9	#define PINOUT_DIR2 35
10	#define PININ_ENCODER A10
11	
12	#define Kp 1
13	#define Ki 0
14	#define Kd 0
15	#define LIMIT_PID 200
16	#define WINDOW_SIZE 5
17	
18	int commands[10];
19	char inputBuffer[20];
20	bool isBufferCompleted = false;
21	unsigned int inputIndex = 0;
22	unsigned int commandIndex = 0;
23	
24	CONTROL_MODE control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
25	DIAGNOSTIC_STATE diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;
26	unsigned int steering_command_pos = 500;
27	unsigned int steering_encoder_pos = 500;
28	unsigned int turning_speed = 0;
29	int window[WINDOW_SIZE];
30	
31	float prevTime = 0;
32	float prevError = 0;
33	float prevControlI = 0;
34	
35	unsigned int minimum_steering = 200;
36	unsigned int maximum_steering = 800;
37	
38	void setup() {
39	Serial.begin(115200);
40	

```
41    pinMode(PINOUT_PWM, OUTPUT);
42    pinMode(PINOUT_DIR1, OUTPUT);
43    pinMode(PINOUT_DIR2, OUTPUT);
44    pinMode(PININ_ENCODER, INPUT);
45
46    for(unsigned int i = 0; i < WINDOW_SIZE; i++) window[i] = 500;
47  }
48
49  void loop() {
50    if(Serial.available()) {
51      steering_encoder_pos = analogRead(PININ_ENCODER);
52      steering_encoder_pos = pushAndCalculateAverage(steering_encoder_pos);
53      turning_speed = computePID(steering_command_pos, steering_encoder_pos);
54
      if((control_mode == CONTROL_MODE::AUTONOMOUS_MODE || control_mode ==
    CONTROL_MODE::TELEOPERATED_MODE) && diagnostic_mode !=
55  DIAGNOSTIC_STATE::HIGH_LATENCY_STATE) {
56        if(steering_encoder_pos > maximum_steering) {
57          turning_speed = 64;
58          driveMotor(-1, turning_speed);
59        } else if(steering_encoder_pos < minimum_steering) {
60          turning_speed = 64;
61          driveMotor(1, turning_speed);
62        } else {
63          if(steering_command_pos > steering_encoder_pos) {
64            driveMotor(1, turning_speed);
65          } else if(steering_command_pos < steering_encoder_pos) {
66            driveMotor(-1, turning_speed);
67          } else {
68            driveMotor(0,0);
69          }
70        }
71      } else {
72        driveMotor(0,0);
73      }
74
75      writeToSerial();
76    }
77  }
78
79
80  void serialEvent() {
81    while (Serial.available()) {
82      char inChar = (char)Serial.read();
83
84      if (inChar == '\n') {
```

```
85        commands[commandIndex] = atoi(inputBuffer);
86        isBufferCompleted = true;
87
88        inputIndex = 0;
89        commandIndex = 0;
90        memset(inputBuffer, 0, sizeof(inputBuffer));
91
92      } else if (inChar == ',') {
93        commands[commandIndex] = atoi(inputBuffer);
94        commandIndex++;
95
96        inputIndex = 0;
97        memset(inputBuffer, 0, sizeof(inputBuffer));
98
99      } else {
100       inputBuffer[inputIndex] = inChar;
101       inputIndex++;
102     }
103
104   }
105
106   if(isBufferCompleted) {
107     control_mode = commands[0];
108     diagnostic_mode = commands[1];
109     steering_command_pos = commands[2];
110
111     isBufferCompleted = false;
112   }
113
114 }
115
116 void writeToSerial() {
117   if(Serial.available()) {
        int message_out[9] = {control_mode, diagnostic_mode, steering_command_pos,
      steering_encoder_pos, steering_command_pos - steering_encoder_pos, turning_speed,
118   Kp, Ki, Kd};
119     for(unsigned int i = 0; i < 8; i++) {
120       Serial.print(message_out[i]);
121       Serial.print(",");
122     }
123     Serial.println(message_out[8]);
124   }
125 }
126
127 unsigned int pushAndCalculateAverage(unsigned int value) {
128   for(unsigned int i = 1; i < WINDOW_SIZE; i++) {
```

```
129       window[i-1] = window[i];
130     }
131     window[WINDOW_SIZE-1] = value;
132
133     unsigned int sum = 0;
134     // for(int i:window) sum += i;
135     for(unsigned int i = 0; i < WINDOW_SIZE; i++) sum += window[i];
136     return sum/WINDOW_SIZE;
137     // return value;
138   }
139
      unsigned int computePID(const unsigned int &command_pos, const unsigned int
140   &encoder_pos) {
141     float currentTime = millis();
142     float elapsedTime = currentTime - prevTime;
143     int error = command_pos - encoder_pos;
144
145     float controlI = prevControlI + (Ki * error * elapsedTime);
146     float controlD = Kd * error / elapsedTime;
147     unsigned int output = abs((Kp * error) + controlI + controlD);
148
149     if(output > LIMIT_PID) {
150       output = LIMIT_PID;
151       controlI = prevControlI;
152     }
153
154     prevControlI = controlI;
155     prevTime = currentTime;
156     return output;
157   }
158
159   void driveMotor(int direction, unsigned int turning_speed) {
160     if(direction == 1) {
161         digitalWrite(PINOUT_DIR1, 0);
162         digitalWrite(PINOUT_DIR2, 1);
163         analogWrite(PINOUT_PWM, turning_speed);
164       } else if(direction == -1) {
165         digitalWrite(PINOUT_DIR1, 1);
166         digitalWrite(PINOUT_DIR2, 0);
167         analogWrite(PINOUT_PWM, turning_speed);
168       } else {
169         digitalWrite(PINOUT_DIR1, 0);
170         digitalWrite(PINOUT_DIR2, 0);
171         analogWrite(PINOUT_PWM, 255);
172       }
```

```
173  }
174
175  #endif
```

ตารางที่ 23 *cockpit_controller*

```
#include "golf/cockpit_controller.h"

using namespace std;
namespace Golf_NS {


CockpitController::CockpitController() {
    baudrate = 115200;
    serial_number = "8593731353035121C0C0"; // "55739323937351F07091";

    latency = 0;
    control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
    cockpit_control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
    prev_cockpit_control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
    diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;

    pub_cockpit_control_mode =
nh.advertise<std_msgs::UInt8>("/cockpit/cockpit_control_mode", 1, true);
    sub_control_mode = nh.subscribe("/cockpit/control_mode", 1,
&CockpitController::callbackControlMode, this);
    sub_diagnostic_mode = nh.subscribe("/cockpit/diagnostic_mode/result", 1,
&CockpitController::callbackDiagnosticMode, this);
    sub_latency = nh.subscribe("/cockpit/latency", 1, &CockpitController::callbackLatency,
this);
}

CockpitController::~CockpitController() {
    cout << "Cockpit Controller shutted down" << endl;
}

void CockpitController::Main() {
    string port;
    bool foundPort = SerialExtension_NS::searchSerialPortBySerialNumber(serial_number,
port);
    if(!foundPort) {
        cout << "[FAIL] port is not found." << endl;
        return;
    }

    serial::Serial serialHandler(port, baudrate, serial::Timeout::simpleTimeout(3000));
    if(!serialHandler.isOpen()) {
        cout << "[FAIL] port failed to open." << endl;
        return;
    }
```

```
40
41      ros::Rate main_rate(20);
42      this_thread::sleep_for(chrono::milliseconds(2000));
43      while(!ros::isShuttingDown()) {
44          ros::spinOnce();
45
46          serialHandler.flushOutput();
47          string message_in, message_out;
            vector<int> output =
48  {static_cast<int>(control_mode),static_cast<int>(diagnostic_mode), latency};
49          SerialExtension_NS::convertIntVectorToString(output, message_out);
50          size_t message_sent_size = serialHandler.write(message_out);
51
52          vector<int> feedback;
53          serialHandler.flushInput();
54          size_t message_size = serialHandler.readline(message_in);
            if(message_size > 0) SerialExtension_NS::convertStringToIntVector(message_in,
55  feedback, ',');
56
57          cockpit_control_mode = static_cast<CONTROL_MODE>(feedback[2]);
58          if(cockpit_control_mode != prev_cockpit_control_mode) {
59              std_msgs::UInt8 mode_msg;
60              mode_msg.data = static_cast<int>(cockpit_control_mode);
61              pub_cockpit_control_mode.publish(mode_msg);
62          }
63
64          for(int i: feedback) cout << i << "  ";
65          cout << endl;
66
67          prev_cockpit_control_mode = cockpit_control_mode;
68          main_rate.sleep();
69      }
70  }
71
72  void CockpitController::callbackControlMode(const std_msgs::UInt8ConstPtr &msg) {
73      control_mode = static_cast<CONTROL_MODE>(msg->data);
74  }
75
76  void CockpitController::callbackDiagnosticMode(const std_msgs::UInt8ConstPtr &msg) {
77      diagnostic_mode = static_cast<DIAGNOSTIC_STATE>(msg->data);
78  }
79
80  void CockpitController::callbackLatency(const std_msgs::Int32ConstPtr &msg) {
81      latency = msg->data;
82  }
83
```

```
84  }
85
86  int main(int argc, char** argv) {
87      ros::init(argc, argv, "cockpit_controller");
88      Golf_NS::CockpitController cockpit_controller;
89      cockpit_controller.Main();
90
91      return 0;
92  }
```

ตารางที่ 24 *Golf_connector*

```
#include "golf/golf_connector.h"

using namespace std;
namespace Golf_NS {

GolfConnector::GolfConnector() {
    baudrate = 115200;
    steering_serial_number = "75830333038351B08011";
    cruise_serial_number = "558333238393519082D0"; // "857333237393512112B2";

    control_mode_from_ros = CONTROL_MODE::AUTONOMOUS_MODE;
    control_mode_from_serial = CONTROL_MODE::AUTONOMOUS_MODE;
    prev_control_mode_from_serial = CONTROL_MODE::AUTONOMOUS_MODE;
    diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;

    velocity_command = 0;
    brake_command = 1500;
    steering_command = 500;
    gear_direction = GEAR_DIRECTION::NEUTRAL_GEAR;
    m_steeringEncoder = 500;
    current_velocity = 0;

    velocity_limit = 3600;
    minimum_steering = 200;
    maximum_steering = 800;
    minimum_brake = 1500;
    maximum_brake = 3500;

    private_nh.param<bool>("enable_steering", enable_steering, true);
    private_nh.param<bool>("enable_cruise", enable_cruise, true);

    pub_control_mode = nh.advertise<std_msgs::UInt8>("/golf/control_mode", 1, true);
    pub_steering_encoder =
nh.advertise<std_msgs::UInt16>("/golf/feedback/steering_encoder", 1, true);
    sub_control_mode = nh.subscribe("/golf/control_mode", 1,
&GolfConnector::callbackControlMode, this);
    sub_diagnostic_mode = nh.subscribe("/golf/diagnostic_mode/result", 1,
&GolfConnector::callbackDiagnosticMode, this);
    sub_golf_command = nh.subscribe("/golf/golf_command", 1,
&GolfConnector::callbackGolfCommand, this);
    sub_teleopt_velocity = nh.subscribe("/golf/teleopt/velocity_command", 1,
&GolfConnector::callbackTeleoptVelocity, this);
    sub_teleopt_brake = nh.subscribe("/golf/teleopt/brake_command", 1,
&GolfConnector::callbackTeleoptBrake, this);
```

```
    sub_teleopt_gear = nh.subscribe("/golf/teleopt/gear_command", 1,
39  &GolfConnector::callbackTeleoptGear, this);
    sub_teleopt_steering = nh.subscribe("/golf/teleopt/steering_command", 1,
40  &GolfConnector::callbackTeleoptSteering, this);
    sub_current_velocity = nh.subscribe("/golf/current_velocity", 1,
41  &GolfConnector::callbackCurrentVelocity, this);
42  }
43
44  GolfConnector::~GolfConnector() {
45    cout << "Golf Connector shutted down" << endl;
46  }
47
48  void GolfConnector::Main() {
49    string steering_port, cruise_port;
50    // bool ret = searchSerialPort(steering_port, cruise_port);
51    steering_port = "/dev/ttyACM0";
52    // if(!ret) return;
53
    serial::Serial steeringSerialHandler(steering_port, baudrate,
54  serial::Timeout::simpleTimeout(3000));
    // serial::Serial cruiseSerialHandler(cruise_port, baudrate,
55  serial::Timeout::simpleTimeout(3000));
56
57    // if(!steeringSerialHandler.isOpen() || !cruiseSerialHandler.isOpen()) {
58    //    cout << "[FAIL] port failed to open." << endl;
59    //    return;
60    // }
61
62    ros::Rate main_rate(20);
63    this_thread::sleep_for(chrono::milliseconds(2000));
64    while(!ros::isShuttingDown()) {
65      ros::spinOnce();
66
67      if(enable_steering) {
68        steeringSerialHandler.flushOutput();
69        string message_in, message_out;
          vector<int> steering_output =
{static_cast<int>(control_mode_from_ros),static_cast<int>(diagnostic_mode),steering_com
70  mand};
71        SerialExtension_NS::convertIntVectorToString(steering_output, message_out);
72        size_t message_sent_size = steeringSerialHandler.write(message_out);
73
74        vector<int> steering_feedback;
75        steeringSerialHandler.flushInput();
76        size_t steering_message_size = steeringSerialHandler.readline(message_in);
          if(steering_message_size > 0)
77  SerialExtension_NS::convertStringToIntVector(message_in, steering_feedback, ',');
```

```
78
79              m_steeringEncoder = steering_feedback[3];
80              cout << "[STEERING] ";
81              for(int i: steering_feedback) cout << i << "    ";
82              cout << endl;
83          }
84
85          // if(enable_cruise) {
86          //     cruiseSerialHandler.flushOutput();
87          //     string message_in, message_out;
            //     vector<int> cruise_output =
    {static_cast<int>(control_mode_from_ros),static_cast<int>(diagnostic_mode),
88  velocity_command, brake_command, static_cast<int>(gear_direction)};
89          //     SerialExtension_NS::convertIntVectorToString(cruise_output, message_out);
90          //     size_t message_sent_size = cruiseSerialHandler.write(message_out);
91
92          //     vector<int> cruise_feedback;
93          //     cruiseSerialHandler.flushInput();
94          //     size_t cruise_message_size = cruiseSerialHandler.readline(message_in);
            //     if(cruise_message_size > 0)
95  SerialExtension_NS::convertStringToIntVector(message_in, cruise_feedback, ',');
96
97          //     control_mode_from_serial = static_cast<CONTROL_MODE>(cruise_feedback[0]);
98
99          //     cout << "[CRUISE] ";
100         //     for(int i: cruise_feedback) cout << i << "    ";
101         // }
102
103         cout << endl << "-----------------------" << endl;
104
            if(control_mode_from_ros != control_mode_from_serial &&
105 prev_control_mode_from_serial != control_mode_from_serial) {
106             std_msgs::UInt8 uint_msg;
107             uint_msg.data = static_cast<int>(control_mode_from_serial);
108             pub_control_mode.publish(uint_msg);
109         }
110
111         std_msgs::UInt16 uint_msg;
112         uint_msg.data = m_steeringEncoder;
113         pub_steering_encoder.publish(uint_msg);
114
115         prev_control_mode_from_serial = control_mode_from_serial;
116         main_rate.sleep();
117     }
118 }
119
```

```
120 bool GolfConnector::searchSerialPort(string& steering_port, string& cruise_port) {
121     bool found_port;
122 
        found_port =
    SerialExtension_NS::searchSerialPortBySerialNumber(steering_serial_number,
123 steering_port);
124     if(!found_port) {
125         cout << "[FAIL] steering port is not found." << endl;
126         return false;
127     }
128 
        found_port = SerialExtension_NS::searchSerialPortBySerialNumber(cruise_serial_number,
129 cruise_port);
130     if(!found_port) {
131         cout << "[FAIL] cruise port is not found." << endl;
132         return false;
133     }
134 
135     return true;
136 }
137 
138 void GolfConnector::callbackControlMode(const std_msgs::UInt8ConstPtr &msg) {
139     control_mode_from_ros = static_cast<CONTROL_MODE>(msg->data);
140 }
141 
142 void GolfConnector::callbackDiagnosticMode(const std_msgs::UInt8ConstPtr &msg) {
143     diagnostic_mode = static_cast<DIAGNOSTIC_STATE>(msg->data);
144 }
145 
    void GolfConnector::callbackGolfCommand(const geometry_msgs::TwistStampedConstPtr
146 &msg) {
147     if(control_mode_from_ros == CONTROL_MODE::AUTONOMOUS_MODE) {
148         velocity_command = msg->twist.linear.x * 1000;
149 
150         if(velocity_command > 0) gear_direction = GEAR_DIRECTION::FORWARD_GEAR;
151         else if(velocity_command < 0) gear_direction = GEAR_DIRECTION::REVERSE_GEAR;
152         else gear_direction = GEAR_DIRECTION::NEUTRAL_GEAR;
153 
154         velocity_command = abs(velocity_command);
155         if(velocity_command > velocity_limit) velocity_command = velocity_limit;
156 
157         if(current_velocity != 0) {
                steering_command = int((0.268598355 + (msg->twist.angular.z /
158 current_velocity))/0.000548244) + 11;
159             if(steering_command > maximum_steering) steering_command = maximum_steering;
                else if(steering_command < minimum_steering) steering_command =
160 minimum_steering;
```

```
161         }
162     }
163 }
164
165 void GolfConnector::callbackTeleoptVelocity(const std_msgs::Float32ConstPtr &msg) {
166     if(control_mode_from_ros == CONTROL_MODE::TELEOPERATED_MODE) {
167         velocity_command = -1 * int(msg->data * 1800) + 1799;
168
169         if(abs(velocity_command) > velocity_limit) velocity_command = velocity_limit;
170     }
171 }
172
173 void GolfConnector::callbackTeleoptBrake(const std_msgs::Float32ConstPtr &msg) {
174     if(control_mode_from_ros == CONTROL_MODE::TELEOPERATED_MODE) {
175         brake_command = -1 * int(msg->data * 850) + 2349;
176
177         if(brake_command > maximum_brake) brake_command = maximum_brake;
178         else if(brake_command < minimum_brake) brake_command = minimum_brake;
179     }
180 }
181
182 void GolfConnector::callbackTeleoptGear(const std_msgs::UInt8ConstPtr &msg) {
183     if(control_mode_from_ros == CONTROL_MODE::TELEOPERATED_MODE) {
184         gear_direction = static_cast<GEAR_DIRECTION>(msg->data);
185     }
186 }
187
188 void GolfConnector::callbackTeleoptSteering(const std_msgs::Float32ConstPtr &msg) {
189     if(control_mode_from_ros == CONTROL_MODE::TELEOPERATED_MODE) {
190         steering_command = -1 * int(msg->data * 300) + 500;
191
192         if(steering_command > maximum_steering) steering_command = maximum_steering;
            else if(steering_command < minimum_steering) steering_command =
193 minimum_steering;
194     }
195 }
196
197 void GolfConnector::callbackCurrentVelocity(const std_msgs::Float32ConstPtr &msg) {
198     current_velocity = msg->data;
199 }
200
201 }
202
203 int main(int argc, char** argv) {
204     ros::init(argc, argv, "golf_connector");
```

```
205     Golf_NS::GolfConnector golf_connector;
206     golf_connector.Main();
207     return 0;
208 }
```

ตารางที่ 25 *Golf_controller*

```
1  #include "golf/golf_controller.h"
2
3  using namespace std;
4  namespace Golf_NS {
5
6  GolfController::GolfController() {
7     latency = 0;
8     distance_to_stop = 10;
9     closest_object_distance = 30;
10    maximum_distance_to_object_to_stop = 10;
11    current_state = "Forward";
12    cockpitConnected = false;
13    control_mode = CONTROL_MODE::AUTONOMOUS_MODE;
14    diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;
15    prev_diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;
16    timer = ros::Time::now();
17
      pub_golf_command =
18 nh.advertise<geometry_msgs::TwistStamped>("/golf/golf_command", 1, true);
      pub_diagnostic_mode = nh.advertise<std_msgs::UInt8>("/golf/diagnostic_mode/result",
19 1, true);
20    sub_twist = nh.subscribe("/twist_cmd", 1, &GolfController::callbackTwistCommand, this);
21    sub_latency = nh.subscribe("/golf/latency", 1, &GolfController::callbackLatency, this);
      sub_control_mode = nh.subscribe("/golf/control_mode", 1,
22 &GolfController::callbackControlMode, this);
      sub_behavior_state = nh.subscribe("/behavior_state", 1,
23 &GolfController::callbackBehaviorState, this);
      sub_current_behavior = nh.subscribe("/op_current_behavior", 1,
24 &GolfController::callbackCurrentBehavior, this);
      sub_teleopt_velocity = nh.subscribe("/golf/teleopt/velocity_command", 1,
25 &GolfController::callbackTeleoptVelocity, this);
26 }
27
28 GolfController::~GolfController() {
29    cout << "Golf Controller shutted down." << endl;
30 }
31
32 void GolfController::Main() {
33    ros::Rate main_rate(20);
34    while(!ros::isShuttingDown()) {
35       ros::spinOnce();
36       if(control_mode == CONTROL_MODE::AUTONOMOUS_MODE && cockpitConnected) {
37          if(current_state == "Follow" && golf_command.twist.linear.x == 0) {
38             if(ros::Time::now() - timer > ros::Duration(20)) {
```

```
39                  diagnostic_mode = DIAGNOSTIC_STATE::REQUEST_CHANGING_STATE;
40              } else diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;
41          } else {
42              diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;
43              timer = ros::Time::now();
44          }
45      } else if(control_mode == CONTROL_MODE::TELEOPERATED_MODE) {
46          if(latency > 500) diagnostic_mode = DIAGNOSTIC_STATE::HIGH_LATENCY_STATE;
47          else diagnostic_mode = DIAGNOSTIC_STATE::NORMAL_STATE;
48          timer = ros::Time::now();
49      }
50
51      if(diagnostic_mode != prev_diagnostic_mode) {
52          std_msgs::UInt8 mode_msg;
53          mode_msg.data = static_cast<int>(diagnostic_mode);
54          pub_diagnostic_mode.publish(mode_msg);
55      }
56      pub_golf_command.publish(golf_command);
57
58      prev_diagnostic_mode = diagnostic_mode;
        cout << "cockpit connect:" << cockpitConnected << "|diag:" << diagnostic_mode <<
    "|control:" << control_mode << "|twist command:" << golf_command.twist.linear.x <<
    "|state:" << current_state << "|latency:" << latency << "|cooldown:" << ros::Time::now() -
59  timer << endl;
60      main_rate.sleep();
61  }
62  }
63
64  void GolfController::callbackLatency(const std_msgs::Int32ConstPtr &msg) {
65      latency = msg->data;
66  }
67
68  void GolfController::callbackControlMode(const std_msgs::UInt8ConstPtr &msg) {
69      control_mode = static_cast<CONTROL_MODE>(msg->data);
70  }
71
    void GolfController::callbackTwistCommand(const geometry_msgs::TwistStampedConstPtr
72  &msg) {
73      float velocity;
74      if(current_state == "Follow") {
            velocity = (closest_object_distance - maximum_distance_to_object_to_stop) * (msg-
75  >twist.linear.x) / maximum_distance_to_object_to_stop;
76      } else if(current_state == "Stop") {
77          velocity = msg->twist.linear.x * distance_to_stop * 0.1;
78      } else velocity = msg->twist.linear.x;
79
```

```
80      if(velocity > msg->twist.linear.x) velocity = msg->twist.linear.x;
81      if(velocity < 0) velocity = 0;
82
83      golf_command.header = msg->header;
84      golf_command.twist.linear.x = velocity;
85      golf_command.twist.angular = msg->twist.angular;
86  }
87
    void GolfController::callbackBehaviorState(const visualization_msgs::MarkerArrayConstPtr
88  &msg) {
89      current_state.clear();
90      string message = msg->markers[0].text;
91      for(char ch: message) {
92          if(isalpha(ch)) current_state.push_back(ch);
93      }
94
95      int pos = message.find(")");
96      distance_to_stop = stof(message.substr(1,pos));
        cout << "current state:" << current_state << "|distance to stop:" << distance_to_stop <<
97  endl;
98  }
99
    void GolfController::callbackCurrentBehavior(const autoware_msgs::WaypointConstPtr
100 &msg) {
101     closest_object_distance = msg->cost;
102 }
103
104 void GolfController::callbackTeleoptVelocity(const std_msgs::Float32ConstPtr &msg) {
105     cockpitConnected = true;
106 }
107
108 }
109
110 int main(int argc, char** argv) {
111     ros::init(argc, argv, "golf_controller");
112     Golf_NS::GolfController golf_controller;
113     golf_controller.Main();
114
115     return 0;
116 }
```

บรรณานุกรม

1. Happian-Smith, J., *An Introduction to Modern Vehicle Design*. 2001: Butterworth-Heinemann.
2. Kocsis, M., et al., *A Method for Transforming Electric Vehicles to Become Autonomous Vehicles*. 2016. 752-761.
3. Bertoluzzo, M., et al., *Drive-by-wire systems for ground vehicles*. 2004. 711-716 vol. 1.
4. Najamuz, Z., *Automotive Electronics Design Fundamentals*. 1 ed. 2015: Springer, Cham.
5. Hafez, H., et al., *Platform Modifications Towards an Autonomous Multi-Passenger Golf Cart.* 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), 2020: p. 525-531.
6. Hafez, H., S.A. Maged, and M. Abdelaziz. *Modular Design of X-by-wire Systems to Facilitate Autonomous Platform Development*. in *2020 8th International Conference on Control, Mechatronics and Automation (ICCMA)*. 2020.
7. Hubbard, G.A. and K. Youcef-Toumi. *System level control of a hybrid-electric vehicle drivetrain*. in *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*. 1997.
8. Pendleton, S., et al., *Autonomous Golf Cars for Public Trial of Mobility-on-Demand Service*. 2015.
9. Kanin, K. and N. Nuksit, *Path following and obstacle avoidance for autonomous vehicle based on gnss localization*. 2019.
10. Fong, T. and C. Thorpe, *Vehicle Teleoperation Interfaces.* Autonomous Robots, 2001. **11**: p. 9-18.
11. Gnatzig, S., et al., *A System Design for Teleoperated Road Vehicles*. Vol. 2. 2013.
12. Lane, J., et al., *Effects of Time Delay on Telerobotic Control of Neutral Buoyancy Vehicles*. Vol. 3. 2002. 2874-2879.
13. Narayanan, A., et al., *A First Look at Commercial 5G Performance on Smartphones*. 2020. 894-905.

14. Commission, O.o.T.N.B.a.T., *5G: คลื่นและเทคโนโลยี.* 2018: p. 21.
15. Xu, D., et al., *Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption*. 2020. 479-494.
16. A. El, R. and M. T. *5G Architecture: Deployment scenarios and options*. in *2019 International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*. 2019.
17. Senk, S., et al. *5G NSA and SA Campus Network Testbeds for Evaluating Industrial Automation*. in *European Wireless 2021; 26th European Wireless Conference*. 2021.
18. Kato, S., et al., *Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems*. 2018. 287-296.
19. Kato, s., et al., *An Open Approach to Autonomous Vehicles.* IEEE Micro, 2015. **vol.48**.
20. Quigley, M., et al., *ROS: an open-source Robot Operating System*. Vol. 3. 2009.
21. Grisetti, G., et al., *A tutorial on graph-based SLAM.* IEEE Transactions on Intelligent Transportation Systems Magazine, 2010. **2**: p. 31-43.
22. Koide, K., J. Miura, and E. Menegatti, *A portable three-dimensional LIDAR-based system for long-term and wide-area people behavior measurement.* International Journal of Advanced Robotic Systems, 2019. **16**.
23. Magnusson, M., A. Lilienthal, and T. Duckett, *Scan Registration for Autonomous Mining Vehicles Using 3D-NDT.* Journal of Field Robotics, 2007. **24**: p. 803-827.
24. Darweesh, H., et al., *Open Source Integrated Planner for Autonomous Navigation in Highly Dynamic Environments.* Journal of Robotics and Mechatronics, 2017. **29**: p. 668-684.
25. Morales, J., et al., *Pure-Pursuit Reactive Path Tracking for Nonholonomic Mobile Robots with a 2D Laser Scanner.* EURASIP Journal on Advances in Signal Processing, 2009. **2009**: p. 1-10.
26. Saparia, S., A. Schimpe, and L. Ferranti, *Active Safety System for Semi-Autonomous Teleoperated Vehicles*. 2021.
27. BODELL, O. and E. GULLIKSSON, *Teleoperation of Autonomous Vehicle With 360° Camera Feedback*, in *Department of Signals and Systems*. 2016,

CHALMERS UNIVERSITY OF TECHNOLOGY.

ประวัติผู้เขียน

ชื่อ-สกุล	Krit T.Siriwattana
วัน เดือน ปี เกิด	3 March 1995
สถานที่เกิด	Chulalongkorn Hospital
วุฒิการศึกษา	B.Eng. Mechanical Engineering, Suranaree University of Technology
ที่อยู่ปัจจุบัน	270/3, Soi 2, 30 Kanya Road, Nai-mueang Sub-district, Mueang District, Nakhonratchasima, 30000
ผลงานตีพิมพ์	T.Siriwattana, K., Kiataramgul, K., Larbwisuthisaroj, S., Kaenkaew, W., Kaewjai, A., and Noomwongs, N. Development of 5G Teleoperated Vehicle Prototype. 13th ATRANS Annual Conference. 2020.
รางวัลที่ได้รับ	Best paper & presentation award given at 13th ATRANS Annual Conference "Digital Transfomation in Transportation and Logistics Post COVID-19 Era" on 4 December 2020, Carton Hotel, Bangkok