การพัฒนาระบบนำทางหุ่นยนต์อัตโนมัติสำหรับใช้งานภายในอาคารจากการจำลองสภาพแวดล้อมแบบ 3 มิติ

นายกิตติยศ ยะเจริญ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมเมคคาทรอนิกส์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2566

DEVELOPMENT OF AN INDOOR AUTONOMOUS MOBILE ROBOT NAVIGATION SYSTEM FROM 3D ENVIRONMENT SIMULATION

KITTIYOS YACHARERN

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Engineering in Mechatronics Engineering
Suranaree University of Technology
Academic Year 2023

การพัฒนาระบบนำทางหุ่นยนต์อัตโนมัติสำหรับใช้งานภายในอาคาร
จากการจำลองสภาพแวดล้อมแบบ 3 มิติ

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

.....
(รศ. ดร.จิระพล ศรีเสริฐผล)
ประธานกรรมการ

.....
(ผศ. ดร.ทศพล รัตน์นิยมชัย)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

.....
(รศ. ดร.บัณฑิต กฤตาคม)
กรรมการ

.....
(รศ. ดร.ยุพาพร รักสกุลพิวัฒน์)
รองอธิการบดีฝ่ายวิชาการและประกันคุณภาพ

.....
(รศ. ดร.พรศิริ จงกล)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

กิตติยศ ยะเจริญ : การพัฒนาระบบนำทางหุ่นยนต์อัตโนมัติสำหรับใช้งานภายในอาคารจากการจำลองสภาพแวดล้อมแบบ 3 มิติ (DEVELOPMENT OF AN INDOOR AUTONOMOUS MOBILE ROBOT NAVIGATION SYSTEM FROM 3D ENVIRONMENT SIMULATION) อาจารย์ที่ปรึกษา: ผู้ช่วยศาสตราจารย์ ดร.ทศพล รัตน์นิยมชัย, 140 หน้า.

คำสำคัญ : ระบบนำทางอัตโนมัติ/ระบบจำลองหุ่นยนต์/AMR/โปรแกรม Gazebo/ROS2

ปัจจุบันธุรกิจร้านอาหาร โรงงานอุตสาหกรรมและเกษตรกรรมได้พยายามแก้ไขปัญหาด้านการขนส่งโดยมีเป้าหมาย คือ การนำหุ่นยนต์ (Autonomous Mobile Robots, AMRs) เข้ามามีบทบาทในการดำเนินกิจการ เนื่องจากความสามารถในการใช้งานในสภาพแวดล้อมที่มีความเสี่ยงและยืดหยุ่นต่อการวางแผนเส้นทาง โดยอาศัยการตรวจจับระยะสิ่งกีดขวางแบบ 360 องศา ร่วมกับระบบติดตามเส้นทาง (Path tracking) ที่สามารถปรับเปลี่ยนค่าพารามิเตอร์ให้เหมาะสมกับจลนศาสตร์ของหุ่นยนต์ซึ่งเป็นสิ่งที่จำเป็นต่อการจัดการสภาพแวดล้อมในการทดสอบการทำงาน ดังนั้น งานวิจัยนี้จึงนำเสนอการสร้างสภาพแวดล้อมจำลองหุ่นยนต์เพื่อปรับเปลี่ยนพารามิเตอร์ติดตามเส้นทาง (Regulated Pure Pursuit, RPP) ให้สามารถนำทางอัตโนมัติได้อย่างปลอดภัย ด้วยสภาพแวดล้อมจำลองในโปรแกรม Gazebo รวมถึงพัฒนาระบบที่สำคัญของหุ่นยนต์ คือ ระบบระบุตำแหน่งหุ่นยนต์ภายในอาคารโดยอาศัยวิธีการ ดังนี้ 1.) Dead Reckoning 2.) ตัวกรองคาลมานแบบขยาย และ 3.) Adaptive Monte Carlo Localization (AMCL) และนำมาทดสอบด้วยการเคลื่อนที่วงปิดสี่เหลี่ยมจัตุรัสระยะทาง 3.6 เมตร จากการศึกษาพบว่า พิกัดจาก AMCL มีค่าผิดพลาดสูงสุด 0.043 เมตร เป็นค่าที่สามารถระบุตำแหน่งของหุ่นยนต์ในการเคลื่อนที่ผ่านเส้นทางที่มีความกว้างน้อยที่สุด 52 เซนติเมตร ให้ไปถึงจุดหมาย นอกจากนี้การจำลองพารามิเตอร์ด้วยโปรแกรม Gazebo สำหรับการควบคุมแบบ RPP ที่เหมาะสม ประกอบด้วยระยะมองไปข้างหน้า (Look-ahead Distance) เท่ากับ 0.24 เมตร และระยะควบคุมรัศมีขั้นต่ำเท่ากับ 0.8 เมตร เป็นพารามิเตอร์ที่ทำให้หุ่นยนต์จริงเคลื่อนที่ออกนอกเส้นทางสูงสุดเท่ากับ 0.0163 เมตร

สาขาวิชาวิศวกรรมเมคคาทรอนิกส์ ลายมือชื่อนักศึกษา .....

ปีการศึกษา 2566 ลายมือชื่ออาจารย์ที่ปรึกษา .....

KITTIYOS YACHARERN : DEVELOPMENT OF AN INDOOR AUTONOMOUS MOBILE ROBOT NAVIGATION SYSTEM FROM 3D ENVIRONMENT SIMULATION.
THESIS ADVISOR : ASST. PROF. TOSAPHOL RATNIYOMCHAI, Ph.D., 140 PP.

Keywords: Navigation System/Robot Simulation/AMR/Gazebo/ROS2

Nowadays, industrial and agricultural factories, as well as restaurants, are attempting to use autonomous mobile robots (AMRs) to handle transportation-related issues. Since the robot has variable path planning and may work in hazardous environments by 360-degree range detection combined with a path tracking system that can adjust parameters to suit the robot's kinematics model, which is necessary to manage the test environment. This research proposes the development of a robot simulation environment to alter route tracking parameters (Regulated Pure Pursuit, RPP) to prove safe automatic navigation by Gazebo simulation program, including developing important robot systems such as a system for Indoor localization with algorithms that work together as follows; 1.) Dead Reckoning 2.) Extended Kalman Filter (EKF) and, 3.) Adaptive Monte Carlo and Localization (AMCL). In addition, all three algorithms have been tested by moving a square closed loop over 3.6 meters. The results found that the coordinates from the AMCL have a Root Mean Square Error (RMSE) of 0.043 meters, a value that can localize the robot to move through a minimum path width of 52 centimeters to reach the destination. Furthermore, the Gazebo simulation parameters for RPP control include a look-ahead distance of 0.24 meters and a minimum control radius of 0.8 meters, which are the maximum averaged tracking errors of a real robot for 0.0163 meters.

School of Mechatronics Engineering
Academic Year 2023

Student's Signature.....
Advisor's Signature.....Tosapho/ Ratniyomchai

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมเครื่องกล หลักสูตรวิศวกรรมเมคคาทรอนิกส์ ความสำเร็จของวิทยานิพนธ์ฉบับนี้เป็นไปอย่างลุล่วงได้ เนื่องจากได้รับความช่วยเหลือและความอนุเคราะห์จากบุคคลหลายท่าน

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.ทศพล รัตน์นิยมชัย และรองศาสตราจารย์ ดร. จิระพล ศรีเสริฐผล ที่ตรวจทาน ให้คำแนะนำ ปรึกษา ตลอดจนช่วยเหลือตรวจทานแก้ไขข้อบกพร่องด้วยความเอาใจส่อย่างดียิ่งจนวิทยานิพนธ์เล่มนี้เสร็จสมบูรณ์

ขอขอบพระคุณ นายศุภชัย แก้วพวง ที่ได้มอบประสบการณ์เพิ่มพูนทักษะที่สามารถนำมาประยุกต์ใช้กับงานวิจัย

ขอขอบพระคุณ นางสาวกนกวรรณ จันทา ที่ได้ช่วยตรวจทานและแก้ไขข้อบกพร่องด้วยความเอาใจส่อย่างดียิ่งจนวิทยานิพนธ์เล่มนี้เสร็จสมบูรณ์

ท้ายที่สุดนี้ ขอกราบขอบคุณ คุณบิดา มารดา ผู้ซึ่งมีพระคุณสูงสุดที่ได้ให้การอุปการะข้าพเจ้ามาโดยตลอด และผู้มีพระคุณทุกท่าน รวมถึงเพื่อนๆ ทุกคน และที่คอยช่วยเหลือให้กำลังใจในการจัดทำวิทยานิพนธ์ครั้งนี้

กิตติยศ ยะเจริญ

สารบัญ

หน้า

บทคัดย่อ (ภาษาไทย) ..... ก
บทคัดย่อ (ภาษาอังกฤษ) ..... ข
กิตติกรรมประกาศ ..... ค
สารบัญ ..... ง
สารบัญตาราง ..... ซ
สารบัญรูป ..... ฌ
บทที่
1 บทนำ 1
1.1 ที่มาและความสำคัญของปัญหา 1
1.2 วัตถุประสงค์การวิจัย 2
1.3 สมมติฐานของการวิจัย 2
1.4 ข้อตกลงเบื้องต้น 2
1.5 ขอบเขตของงานวิจัย 3
1.6 วิธีการดำเนินการของงานวิจัย 3
1.7 ประโยชน์ที่คาดว่าจะได้รับ 4
2 ปริทัศน์วรรณกรรม ทฤษฎี และงานวิจัยที่เกี่ยวข้อง 6
2.1 กล่าวนำ 6
2.2 ระบบจำลองหุ่นยนต์ 6
2.3 การจำลองโครงสร้างหุ่นยนต์ 8
2.3.1 การแสดงผลโครงสร้างหุ่นยนต์ 9
2.4 ระบบปฏิบัติการหุ่นยนต์ 10
2.5 การแสดงผลข้อมูลเซนเซอร์ด้วยกราฟฟิก 13
2.6 ระบบขับเคลื่อนหุ่นยนต์แบบองศาไม่อิสระ 13
2.6.1 แบบจำลองพลวัติของหุ่นยนต์ 15

สารบัญ (ต่อ)

หน้า

2.6.2 การคำนวณขนาดต้นกำลัง 17
2.7 การระบุตำแหน่งและสร้างแผนที่ 18
2.7.1 เซนเซอร์เข้ารหัสสัมบูรณ์ 19
2.7.2 ทฤษฎีการประมาณตำแหน่งบนระบบพิกัด 2 มิติ 21
2.7.3 ระบบนำร่องด้วยแรงเฉื่อย 22
2.7.4 ตัวกรองคาลมานแบบขยาย 23
2.7.5 เทคโนโลยี LiDAR 26
2.7.6 การสร้างแผนที่และระบุตำแหน่งไปพร้อมกัน 28
2.7.7 การแทนที่ข้อมูลของสิ่งกีดขวาง 29
2.7.8 การระบุตำแหน่งบนแผนที่ด้วย AMCL 30
2.8 ระบบควบคุมหุ่นยนต์อัตโนมัติ 32
2.8.1 ระบบควบคุมป้อนกลับ พีไอดี 32
2.8.2 การปรับตัวควบคุมแบบ พีไอดี ด้วยวิธีการวัฏจักรสุดท้าย 33
2.8.3 แบบจำลองจนศาสตร์ 34
2.9 ระบบนำทางอัตโนมัติของหุ่นยนต์ 37
2.9.1 ระบบนำทางอัตโนมัติ 38
2.9.2 การสร้าง Costmap จากสิ่งกีดขวาง 39
2.9.3 การติดตามเส้นทางแบบ Pure Pursuit 40
2.9.4 การติดตามเส้นทางแบบ Regulated Pure Pursuit 42
3 วิธีการดำเนินงานวิจัย 44
3.1 แบบแผนการดำเนินงานวิจัย 44
3.2 การออกแบบโครงสร้างและอุปกรณ์ของหุ่นยนต์ 45
3.2.1 การออกแบบขนาดต้นกำลัง. 45
3.2.2 การออกแบบโครงสร้างหุ่นยนต์ 46
3.2.3 การประกอบหุ่นยนต์จริง 48
3.3 การสอบเทียบเซนเซอร์ไจโรสโคป 51
3.4 ระบบจำลองหุ่นยนต์ร่วมกับการนำทางอัตโนมัติ 53

สารบัญ (ต่อ)

หน้า

3.4.1 การกำหนดลักษณะทางโครงสร้างหุ่นยนต์ด้วย URDF 56
3.4.2 การเผยแพร่สถานะของโครงสร้าง 58
3.4.3 พารามิเตอร์สำหรับควบคุมหุ่นยนต์ 59
3.4.4 การจำลองสภาพแวดล้อมใช้งานหุ่นยนต์ 60
3.5 ระบบสถาปัตยกรรมนำทางอัตโนมัติของหุ่นยนต์จริง 61
3.5.1 การออกแบบระบบควบคุมด้วยวิธีการของซีกเลอร์-นิโคลส์ 62
3.5.2 การบูรณาการข้อมูลเซนเซอร์ด้วยตัวกรองคาลมาน 63
3.5.3 ความผิดพลาดสัมบูรณ์เซนเซอร์ไจโรสโคป 64
3.5.4 การระบุตำแหน่งของหุ่นยนต์ 65
3.5.5 การตรวจสอบความถูกต้องระหว่างระบบจำลองกับหุ่นยนต์จริง 69
3.6 การจำลองระบบนำทางอัตโนมัติเพื่อหาพารามิเตอร์ที่เหมาะสม 70
3.6.1 การเตรียมระบบนำทางอัตโนมัติและกำหนดพารามิเตอร์เพื่อสร้างเส้นทาง 71
3.6.2 การจำลองพารามิเตอร์ติดตามเส้นทางแบบ PP 73
3.6.3 การจำลองพารามิเตอร์ติดตามเส้นทางแบบ RPP 74
3.6.4 การทดสอบหุ่นยนต์จริงด้วยพารามิเตอร์ RPP 74
3.6.5 การทดสอบการหลบหลีกสิ่งกีดขวางอัตโนมัติด้วยพารามิเตอร์ RPP 76
4 ผลการวิจัยและวิเคราะห์ผล 78
4.1 ผลการออกแบบขนาดต้นกำลัง 78
4.2 พารามิเตอร์สำหรับควบคุมหุ่นยนต์ 78
4.3 ผลการออกแบบตัวควบคุมด้วยทฤษฎีซีกเลอร์-นิโคล 79
4.4 ผลการบูรณาการข้อมูลเซนเซอร์ด้วยตัวกรองคาลมานแบบขยาย 80
4.4.1 การทดสอบพารามิเตอร์ตัวกรองคาลมาน 87
4.4.2 ผลความผิดพลาดสัมบูรณ์ของเซนเซอร์ไจโรสโคป 90
4.5 ผลการระบุตำแหน่งของหุ่นยนต์จริง 91
4.6 การตรวจสอบความถูกต้องด้วยแบบบจำลองจลนศาสตร์ 95
4.7 ผลจำลองพารามิเตอร์นำทางอัตโนมัติ 100

สารบัญ (ต่อ)

หน้า

4.7.1 ระยะมองไปข้างหน้า....100
4.7.2 พารามิเตอร์ควบคุมความเร็วด้วยรัศมีทางโค้งขั้นต่ำ....103
4.8 ผลทดสอบพารามิเตอร์ RPP กับหุ่นยนต์จริง....107
4.9 เปรียบเทียบผลการจำลองกับหุ่นยนต์จริง....110
4.10 ผลทดสอบการหลบหลีกสิ่งกีดขวางอัตโนมัติ (หุ่นยนต์จริง)....112
5 สรุปผลการวิจัยและข้อเสนอแนะ....115
5.1 สรุปผลการวิจัยและข้อเสนอแนะ....115
5.1.1 ระบบระบุตำแหน่ง....115
5.1.2 การจำลองพารามิเตอร์แบบ PP....115
5.1.3 การจำลองพารามิเตอร์แบบ RPP....116
5.1.4 การทดสอบใช้งานจริงด้วยพารามิเตอร์แบบ RPP....116
5.1.5 การหลบลีกสิ่งกีดขวางอัตโนมัติ....116
5.2 ข้อเสนอแนะ....116
รายการอ้างอิง....117
ภาคผนวก
ภาคผนวก ก. ข้อมูลจำเพาะของอุปกรณ์ที่เกี่ยวข้องและการประกอบหุ่นยนต์....121
ก.1 บอร์ด Raspberry Pi 4 Model B....122
ก.2 มอเตอร์ DYNAMIXEL รุ่น XL-430....123
ก.3 เซนเซอร์ IMU WITMOTION รุ่น WT-901C-....124
ก.4 เซนเซอร์ YDLIDAR G2....125
ภาคผนวก ข. บทความวิชาการที่ได้รับการตีพิมพ์ในระหว่างการศึกษา....129
ประวัติผู้เขียน....140

สารบัญตาราง

ตารางที่ **หน้า**

1.1 แผนงานดำเนินงานวิจัย 4
2.1 เอาต์พุตของการเข้ารหัสด้วยวงจร D Flip-flop 21
2.2 ตารางการคำนวณตัวควบคุมด้วยวิธีด้วยวิธีการวัฏจักรสุดท้าย 34
3.1 ลำดับรายการอุปกรณ์ 46
3.2 ตำแหน่งพิกัดของส่วนประกอบหุ่นยนต์ 57
3.3 ขนาดโมเมนต์ความเฉื่อยของแต่ละส่วน 58
4.1 สรุปพารามิเตอร์สำหรับจำลองหุ่นยนต์อัตโนมัติ 78
4.2 ผลการออกแบบตัวควบคุม พีไอ 80
4.3 ผลการคำนวณค่าความแปรปรวนเฉลี่ยจากการวัด 5 ครั้ง (ครั้งที่ 1) 86
4.4 ผลการคำนวณค่าความแปรปรวนเฉลี่ยจากการวัด 5 ครั้ง (ครั้งที่ 2). 87
4.5 ความผิดพลาดมุมองศาของเซนเซอร์ไจโรสโคปที่ความเร็วเชิงมุม 0.25 เรเดียนต่อวินาที 90
4.6 ความผิดพลาดมุมองศาของเซนเซอร์ไจโรสโคปที่ความเร็วเชิงมุม 0.5 เรเดียนต่อวินาที 90
4.7 ความผิดพลาดมุมองศาของเซนเซอร์ไจโรสโคปที่ความเร็วเชิงมุม 1.0 เรเดียนต่อวินาที 91
4.8 ผลการเปรียบเทียบผลจากการคำนวณเปรียบเทียบกับระบบจำลองและระบบจริง 98
4.9 ผลการลดระยะ L ต่อระยะผิดพลาดออกนอกเส้นทาง t_e 102
4.10 ความสัมพันธ์ของค่าผิดพลาดออกนอกเส้นทางกับ 104
4.11 การเปรียบเทียบค่าผิดพลาดออกนอกเส้นทางระหว่าง PP กับ RPP 106
4.12 สรุปพารามิเตอร์ที่เหมาะสมกับระบบนำทางอัตโนมัติ 107
4.13 ค่าผิดพลาดติดตามเส้นทางผ่านทางโค้งทั้งหมด 3 ช่วง จากการทดสอบ 10 ครั้ง 109
ก.1 คุณสมบัติของบอร์ด Raspberry Pi 4 Model B 122
ก.2 คุณสมบัติมอเตอร์ DYNAMIXEL รุ่น XL-430 123
ก.3 คุณสมบัติของเซนเซอร์ IMU WITMOTION รุ่น WT-901C 124
ก.4 คุณสมบัติของเซนเซอร์ YDLIDAR G2 125

สารบัญรูป

รูปที่ **หน้า**

2.1 สถาปัตยกรรมการจำลองสภาพแวดล้อมด้วย Gazebo 7
2.2 การเผยแพร่สภาวะหุ่นยนต์ 9
2.3 โครงสร้างหุ่นยนต์ที่ถูกเผยแพร่สภาวะ 9
2.4 ลักษณะการทำงานภายในระบบ ROS2 10
2.5 โครงสร้าง API บนระบบ ROS2 12
2.6 การแสดงผลข้อมูลจากเซนเซอร์แบบ 3 มิติ 13
2.7 รูปแบบการขับเคลื่อนหุ่นยนต์แบบ Differential Drive 14
2.8 หุ่นยนต์ Pioneer 2AT 14
2.9 แบบจำลองพลวัติของหุ่นยนต์ Skid-steering 15
2.10 แรงบิดที่กระทำให้เกิดแรงผลักของล้อ 17
2.11 ตำแหน่งการเจาะรูบนแผ่นดิสก์แบบ 4 บิต 20
2.12 วงจร D Flip-flop การถอดรหัสแบบ 4 บิต 20
2.13 แบบจำลองการระบุตำแหน่งของหุ่นยนต์ 21
2.14 ส่วนประกอบเซนเซอร์วัดความเร่ง 22
2.15 ความสัมพันธ์ทิศการหมุนกับเซ็นเซอร์ไจโรสโคป 23
2.16 ขั้นตอนการประมาณสถานะด้วยตัวกรองคาลมาน 24
2.17 ส่วนประกอบของ LiDAR 27
2.18 ระเบียบการประมาณตอบของ SLAM 28
2.19 การแทนที่ข้อมูลแบบ Occupancy Grid Map 30
2.20 การกระจายของอนุภาคสมมติของตัวกรองอนุภาค 32
2.21 แผนภาพบล็อกของระบบควบคุมป้อนกลับแบบ PID 33
2.22 การเกิดเสถียรภาพแบบขอบ 33
2.23 แบบจำลองทางคณิตศาสตร์การขับเคลื่อนแบบ Skid Steering 34
2.24 ความแตกต่างของจุด ICR ทั้ง ระหว่าง 2 ระบบขับเคลื่อน 36

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

2.25 ระยะมองไปข้างหน้าที่สัมพันธ์กับรัศมีโค้งของเส้นทาง 37
2.26 สถาปัตยกรรม NAV2 38
2.27 การแสดงผลระยะจากสิ่งกีดขวางด้วย Costmap 40
2.28 หลักการควบคุมอัลกอริทึมแบบ Pure-pursuit 41
2.29 ความสัมพันธ์รัศมีเส้นทางโค้ง 43
3.1 แบบแผนการดำเนินงานวิจัย 44
3.2 โครงสร้างด้านข้างของหุ่นยนต์ 47
3.3 โครงสร้างด้านหน้าของหุ่นยนต์ 48
3.4 โครงสร้างหุ่นยนต์ AMR 48
3.5 ลำดับในการติดตั้งอุปกรณ์ของ Layer 1 และ Layer 2 48
3.6 หุ่นยนต์ AMR หลังจากประกอบ Layer 1 และ Layer 2 49
3.7 หุ่นยนต์ AMR หลังจากประกอบโครงสร้าง 49
3.8 การเชื่อมต่ออุปกรณ์ภายในและระดับขั้นการประยุกต์ใช้งาน 50
3.9 ลักษณะการวางเซนเซอร์เพื่อสอบเทียบค่า ω_z ที่วัดด้วยเซนเซอร์ไจโรสโคป 51
3.10 ค่า ω_z รอบแกน Yaw ที่ปิดใช้งานระบบสอบเทียบอัตโนมัติของไจโรสโคป (สีแดง) 52
3.11 ค่า ω_z รอบแกน Yaw ที่เปิดใช้งานระบบสอบเทียบอัตโนมัติของไจโรสโคป (สีแดง) 52
3.12 ระบบจำลองหุ่นยนต์ร่วมกับ NAV2 53
3.13 ผังการทำงานเมื่อขอบเขตหุ่นยนต์เข้าไปอยู่ในส่วน Lethal cost 56
3.14 การกำหนดแกนการหมุนของล้อ 56
3.15 การ Export รูปแบบ URDF 57
3.16 การแสดงผลของสถานะหุ่นยนต์จาก URDF 58
3.17 สภาพแวดล้อมจำลองหุ่นยนต์ 60
3.18 สถาปัตยกรรม NAV2 กับหุ่นยนต์จริง 61
3.19 กระบวนการหาค่าตัวควบคุม 62
3.20 การวัดความผิดพลาดการวัดมุมด้วยไจโรสโคป 65
3.21 การแปลงเฟรมจากอัลกอริทึม AMCL 65

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

3.22 การสร้างแผนที่ในการทดสอบ AMCL 66
3.23 พิกัดบนระนาบ 2 มิติสำหรับประเมินประสิทธิภาพการระบุตำแหน่ง 66
3.24 แผนภาพการดึงข้อมูลพิกัดตำแหน่งของหุ่นยนต์ 3 ส่วน 67
3.25 ความเร็วเชิงมุมรอบแกน Yaw กับความเร็วเชิงเส้นในการประมาณตำแหน่งของหุ่นยนต์ 68
3.26 การระบุตำแหน่งด้วยความเร็วเชิงมุมรอบแกน Yaw กับความเร็วเชิงเส้นสัมพันธ์กับพิกัด 69
3.27 ความสัมพันธ์ของความเร็วเชิงมุมรอบแกน Yaw 69
3.28 คำสั่ง cmd_vel ในการควบคุม v_t 70
3.29 คำสั่ง cmd_vel ในการควบคุม ω_t 70
3.30 สถาปัตยกรรมระบบนำทางอัตโนมัติ NAV2 71
3.31 การควบคุมหุ่นยนต์ในโปรแกรม Gazebo เพื่อเก็บข้อมูลแผนที่ด้วยวิธีการ SLAM 72
3.32 ผลลัพธ์ของแผนที่จากการ SLAM 72
3.33 ขอบเขต Footprint ของหุ่นยนต์ 73
3.34 ลักษณะการวางแผนเส้นทางอ้างอิงที่อาศัยข้อมูล Costmap 73
3.36 สภาพแวดล้อมในการทดสอบหุ่นยนต์เคลื่อนที่ผ่านเส้นทางโค้ง C1 75
3.37 สภาพแวดล้อมในการทดสอบหุ่นยนต์เคลื่อนที่ผ่านเส้นทางโค้ง C2 75
3.38 สภาพแวดล้อมในการทดสอบหุ่นยนต์เคลื่อนที่ผ่านเส้นทางโค้ง C3 75
3.39 ผลลัพธ์จากการ SLAM ของหุ่นยนต์จริงเพื่อเตรียมทดสอบพารามิเตอร์ 76
3.40 สภาพแวดล้อมทดสอบการหลบหลีกสิ่งกีดขวางอัตโนมัติ 77
4.1 ผลตอบสนองความเร็ว v_x เมื่อ K_p' เท่ากับ 1190 79
4.2 ผลตอบสนองสัญญาณเมื่อ K_p' เท่ากับ 1200 79
4.3 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 1 80
4.4 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 2 81
4.5 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 3 81
4.6 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 4 81
4.7 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 5 82
4.8 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 6 82
4.9 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 7 82

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

4.10 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 8 83
4.11 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 9 83
4.12 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 10 83
4.13 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 1 84
4.14 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 2 84
4.15 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 3 84
4.16 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 4 84
4.17 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 5 85
4.18 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 6 85
4.19 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 7 85
4.20 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 8 85
4.21 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 9 86
4.22 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 10 86
4.23 สัญญาณ v_x ที่ไม่ผ่านตัวกรองคาลมาน 88
4.24 สัญญาณ v_x เมื่อเมทริกซ์ Q เท่ากับ R 88
4.25 สัญญาณ ω_z ที่ไม่ผ่านตัวกรองคาลมาน 88
4.26 สัญญาณ ω_z จากตัวกรองคาลมานเมื่อเมทริกซ์ Q เท่ากับ R 89
4.27 สัญญาณความเร็วเชิงมุมเมื่อ σ_y^2 เท่ากับ $2.81e^{-7}$ 89
4.28 ผลเปรียบเทียบพิกัดจากการระบุตำแหน่ง ครั้งที่ 1-4 92
4.29 ผลเปรียบเทียบพิกัดจากการระบุตำแหน่ง ครั้งที่ 5-8 92
4.30 ผลเปรียบเทียบพิกัดจากการระบุตำแหน่ง ครั้งที่ 9-10 93
4.31 ระยะความผิดผลาดเฉลี่ยจากการทดลอง 10 ครั้ง เทียบกับตำแหน่งจริง 93
4.32 แผนภาพลดความผิดพลาดจากการระบุตำแหน่งด้วยวิธี AMCL 94
4.33 ส่วนเบี่ยงเบนมาตรฐานการเคลื่อนที่ผ่านพิกัด 13 จุด ทั้งหมด 10 ครั้ง 95
4.34 สัญญาณป้อนกลับ v_x (จำลอง) 96
4.35 สัญญาณป้อนกลับ v_x (จริง) 96
4.36 สัญญาณป้อนกลับ ω_z (จำลอง) 96

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

4.37 สัญญาณป้อนกลับ ω_z (จริง) 96
4.38 ความเร็วล้อซ้าย ω_l และ ω_r (จำลอง) 98
4.39 ความเร็วล้อซ้าย ω_l และ ω_r (จำลอง) 98
4.40 เปรียบเทียบพิกัดการเคลื่อนที่ของหุ่นยนต์ทั้ง 2 ระบบ 99
4.41 ค่าความผิดพลาดการเคลื่อนที่ของระบบจำลองเปรียบเทียบกับหุ่นยนต์จริง 99
4.42 พิกัดตำแหน่งการเคลื่อนที่ของหุ่นยนต์ไปถึงจุดหมาย 100
4.43 พิกัดการเคลื่อนที่ของหุ่นยนต์จากการลดระยะ L 101
4.44 ความสัมพันธ์ของค่าผิดพลาดออกนอกเส้นทางกับระยะ L 102
4.45 ความเร็วเชิงเส้นจากการติดตามเส้นทางแบบ PP เมื่อ L เท่ากับ 0.24 เมตร 103
4.46 ลักษณะพิกัดการเคลื่อนที่จากการเปลี่ยนแปลง r_{min} 103
4.47 ค่าผิดพลาดออกนอกเส้นทางโดยเฉลี่ยจากการเปลี่ยนแปลง r_{min} 104
4.48 คำสั่งควบคุมความเร็วเชิงเส้น v_t' จากระบบควบคุมแบบ RPP 105
4.49 ความสัมพันธ์ของเวลาที่ใช้ในการนำทางหุ่นยนต์กับระยะ r_{min} 105
4.50 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 1 และ 2 107
4.51 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 3 และ 4 108
4.52 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 5 และ 6 108
4.53 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 7 และ 8 108
4.54 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 9 และ 10 109
4.55 ลักษณะการควบคุมด้วย v_t' และความเร็วป้อนกลับ v_x (จำลอง) 111
4.56 ลักษณะการควบคุม v_t' และความเร็วป้อนกลับ v_x (จริง) 111
4.57 ลักษณะการควบคุม ω_t และความเร็วป้อนกลับ ω_z (จำลอง) 111
4.58 ลักษณะการควบคุม ω_t และความเร็วป้อนกลับ ω_z (จริง) 111
4.59 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.7 เมตร 112
4.60 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.6 เมตร 112
4.61 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.5 เมตร 112
4.62 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.4 เมตร 113

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

4.63 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.3 เมตร 113
4.64 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.2 เมตร 113
4.65 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.3 เมตร ครั้งที่ 1 114
4.66 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.3 เมตร ครั้งที่ 2 114
4.67 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.3 เมตร ครั้งที่ 3 114
ก.1 บอร์ด Raspberry Pi 4 Model B 122
ก.2 มอเตอร์ DYNAMIXEL XL-430 W250T 123
ก.3 เซนเซอร์ WITMOTION W901C 124
ก.4 เซนเซอร์ YDLIDAR G2 125
ก.5 การทดสอบการทำงานของมอเตอร์ 126
ก.6 ส่วนประกอบของหุ่นยนต์ 126
ก.7 การทดสอบวัดค่าจากเซนเซอร์ไจโรสโคปด้วยโปรแกรม WitMotion 127
ก.8 การเปิดการทำงานระบบสอบเทียบไจโรสโคปด้วยโปรแกรม WitMotion 127
ก.9 การทดสอบเซนเซอร์ YDLIDAR G2 ด้วยโปรแกรม Lidar Viewer 128

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

ตั้งแต่ปลายทศวรรษที่ 1950 มีรถลากจูงหลายประเภทที่ใช้ในโรงงานและคลังสินค้า โดยจะใช้ Automated Guided Vehicle (AGV) นำมาดัดแปลงโดยรถแทรกเตอร์ให้วิ่งไปตามสายไฟที่จัดไว้ตามเส้นทางซึ่งช่วยเพิ่มผลผลิตแรงงานได้อย่างมากอีกทั้งยังปรับปรุงระบบอัตโนมัติในการขนถ่ายให้สะดวกขึ้น ในช่วงทศวรรษที่ 1980 มีการพัฒนา AGV โดยใช้เทคโนโลยีการนำทางแบบไร้สายซึ่งช่วยเพิ่มประสิทธิภาพของ AGV ได้อย่างมาก นอกจากนั้นยังมีส่วนสำคัญในด้านโลจิสติกส์การผลิตและกลายเป็นส่วนสำคัญของอุปกรณ์ระบบอัตโนมัติขององค์กร โดยเฉพาะในยุโรปสหรัฐอเมริกาและประเทศอื่น ๆ และใช้กันอย่างแพร่หลายมากที่สุด ในเอเชียญี่ปุ่นและเกาหลีใต้ ทำให้เห็นว่าการจัดการระบบขนส่งด้วยระบบอัตโนมัติมีการนำมาใช้งานอย่างเป็นวงกว้าง ซึ่งธุรกิจหลายแขนงได้เห็นถึงความสำคัญของการใช้ AGV เพื่อความคุ้มค่าคุ้มทุน ของกิจการ ส่งผลให้ปัจจุบัน AGV กลายเป็นสิ่งที่จำเป็นต่อโรงงาน คลังสินค้าที่มีการขนส่งวัสดุ อย่างไรก็ตามระบบนำทางแบบดังกล่าว ยังเป็นระบบที่ต้องมีการติดตั้งอยู่กับพื้นที่หรือ สภาพแวดล้อมที่ต้องการใช้งาน ทำให้ AGV มีข้อจำกัดในการเปลี่ยนแปลงเส้นทางจากเดิม รวมถึงการติดตั้งที่มีความซับซ้อนในการขยายระบบเพื่อรองรับการใช้งานในอนาคต ทำให้ค่าใช้จ่ายต่อการขยายระบบและการซ่อมบำรุงมีราคาที่ค่อนข้างสูง (Vancea and Orha., 2019) ในปี ค.ศ. 2009 Willow Garage ได้เปิดตัวระบบปฏิบัติการหุ่นยนต์โอเพ่นซอร์ส Robot Operating System (ROS) แสดงให้เห็นถึงความก้าวหน้าในเทคโนโลยีการพัฒนาโปรแกรมหุ่นยนต์เหล่านี้มีประสิทธิภาพและตอบโจทย์การทำงานได้อย่างชาญฉลาด แนวทางดังกล่าวครอบคลุมตั้งแต่การเริ่มต้นหุ่นยนต์เคลื่อนที่อัจฉริยะตัวแรกที่ชื่อ Shakey ในปี 1972 ไปจนถึงการใช้งานหุ่นยนต์เคลื่อนที่อัตโนมัติ Autonomous Mobile Robot (AMR) ในสถานพยาบาล ซึ่งหุ่นยนต์ AMR มีความท้าทายขั้นพื้นฐานในการนำทางหุ่นยนต์เคลื่อนที่ครอบคลุมสามประเด็นสำคัญประกอบด้วยการระบุตำแหน่ง การสร้างแผนที่ และวางแผนเส้นทาง (Liu et al., 2023)

การประยุกต์อุปกรณ์ฮาร์ดแวร์ที่ซับซ้อนประกอบด้วยเซนเซอร์และคอมพิวเตอร์จำนวนมากถูกควบคุมโดยซอฟต์แวร์แบบกระจายเป็นหนึ่งระบบที่จำเป็นต่อหุ่นยนต์ AMR เนื่องจากหุ่นยนต์จะต้องนำทางและปฏิบัติงานเฉพาะอย่างประสบความสำเร็จในสภาพแวดล้อมต่าง ๆ ที่มีความหลากหลาย การสร้างสนามทดสอบพฤติกรรมของหุ่นยนต์ภายใต้เงื่อนไขต่าง ๆ ทำให้มีค่าใช้จ่ายสูงและใช้เวลานาน การใช้สภาพแวดล้อมการจำลองที่ได้รับการพัฒนามาอย่างดีช่วยให้สามารถทดสอบ

ระบบหุ่นยนต์ให้ทำงานได้อย่างงปลอดภัยและคุ้มค่า การจำลองจะช่วยลดวงจรการพัฒนาและสามารถนำไปใช้ได้อย่างหลากหลายสำหรับสภาพแวดล้อมที่แตกต่างกัน (Takaya et al., 2016)

สิ่งเหล่านี้แสดงให้เห็นถึงความสำคัญในการพัฒนาหุ่นยนต์จากการจำลองสภาพแวดล้อมการทำงานให้กับหุ่นยนต์ AMR ซึ่งการทำงานร่วมกับระบบติดตามเส้นทางจากการวางแผนแบบเรียบง่ายแต่ให้ประสิทธิภาพสูงในพื้นที่ที่จำกัด คือ การควบคุมด้วย Regulated Pure Pursuit (RPP) (Macenski et al., 2023) ได้ถูกนำมาประยุกต์ใช้จากการออกแบบพารามิเตอร์ด้วยการจำลองผ่านโปรแกรม Gazebo เพื่อปรับปรุงประสิทธิภาพการติดตามในส่วนการเข้าโค้งหักศอก และหุ่นยนต์ต้องสามารถนำทางอัตโนมัติด้วยการระบุตำแหน่งแบบ AMCL ในที่แคบ โดยมีข้อมูลอินพุดของพิกัดจากการทำ Dead Reckoning ร่วมกับตัวกรองคาลมานที่ผสานเข้ากับระบบนำร่องแบบเฉื่อย (Inertial Navigation System, INS)

1.2 วัตถุประสงค์งานวิจัย

1.2.1 เพื่อออกแบบและพัฒนาหุ่นยนต์นำทางอัตโนมัติสำหรับใช้งานภายในอาคาร

1.2.2 เพื่อพัฒนาระบบประมาณตำแหน่งบนระนาบ 2 มิติ ในการใช้งานภายในอาคาร

1.2.3 ออกแบบพารามิเตอร์นำทางอัตโนมัติที่ติดตามเส้นทาง RPP ให้เหมาะสมกับสภาพแวดล้อมที่ได้ออกแบบ ด้วยระบบจำลองการทำงานที่สร้างขึ้นโดยใช้โปรแกรม Gazebo โดยเป็นค่าพารามิเตอร์ที่สามารถนำไปใช้กับหุ่นยนต์และให้ผลลัพธ์การเคลื่อนที่แบบเดียวกับระบบจำลอง

1.3 สมมติฐานของการวิจัย

ความผิดพลาดจากการระบุตำแหน่งของหุ่นยนต์ถูกลดลงด้วยอัลกอริทึมตัวกรองคาลมานและการระบุตำแหน่งด้วย AMCL โดยเป็นค่าความคลาดเคลื่อนที่สามารถนำทางในสภาพแวดล้อมที่มีความแคบ 70 เซนติเมตร ด้วยระบบติดตามเส้นทางแบบ RPP ที่ได้ออกแบบจากการจำลองสภาพแวดล้อมแบบ 3 มิติ โดยหุ่นยนต์จริงสามารถเคลื่อนที่ได้ใกล้เคียงกับการจำลองหุ่นยนต์ผ่านโปรแกรม Gazebo

1.4 ข้อตกลงเบื้องต้น

1.4.1 การประมาณองศาการหมุนใช้การวัดเฉพาะค่าความเร็วเชิงมุมด้วยเซนเซอร์ไจโรสโคปเท่านั้น

1.4.2 การทดสอบประสิทธิภาพการใช้ตัวกรองคาลมานแบบขยายได้ควบคุมหุ่นยนต์เคลื่อนที่เป็นรูปสี่เหลี่ยมวงปิดขนาด 90 x 90 ตารางเซนติเมตร ทั้งหมดจำนวน 10 รอบ เพื่อเก็บข้อมูลตำแหน่งการเคลื่อนที่ เปรียบเทียบกับตำแหน่งจริงที่ได้จากการวัดด้วยตลับเมตร

1.4.3 การจำลองพารามิเตอร์อัลกอริทึมควบคุม RPP กำหนดให้หุ่นยนต์ใน สภาพแวดล้อมจำลองนำทางอัตโนมัติในทางที่มีความกว้างขนาด 70 เซนติเมตร

1.5 ขอบเขตของงานวิจัย

1.5.1 หุ่นยนต์นำทางอัตโนมัติเป็นระบบหุ่นยนต์ AMR มีการขับเคลื่อนแบบ Skid steering

1.5.2 การออกแบบระบบควบคุมป้อนกลับใช้กฎ ซีกเลอร์-นิโคลส์

1.5.3 ใช้อัลกอริทึมตัวกรองคาลมานแบบขยายบูรณาการข้อมูลการวัดของไจโรสโคป เพื่อลดความคลาดเคลื่อนของพิกัดเป็นอินพุตให้กับการระบุตำแหน่งบนแผนที่ด้วย AMCL

1.5.4 ใช้ระบบจำลองออกแบบพารามิเตอร์ควบคุม RPP ด้วยทางโค้งหักศอก 90 องศา ทั้งหมด 3 จุด ที่มีความกว้างของเส้นทาง 70 เซนติเมตร โดยมีข้อกำหนดให้หุ่นยนต์เคลื่อนที่ออกเส้นทางได้ไม่เกิน 0.05 เมตร

1.5.5 กำหนดพารามิเตอร์นำทางที่ได้จากระบบจำลองใช้งานกับหุ่นยนต์จริงในสภาพแวดล้อมแบบเดียวกัน

1.6 วิธีการดำเนินการของงานวิจัย

1.6.1 ศึกษางานวิจัยที่เกี่ยวข้อง

1.6.2 พิจารณาการเลือกใช้อุปกรณ์ในการสร้างหุ่นยนต์

1.6.3 ออกแบบตัวขับต้นกำลัง

1.6.4 ประกอบหุ่นยนต์จริงด้วยอุปกรณ์ที่ออกแบบ

1.6.5 ศึกษาการระบุตำแหน่งด้วยตัวกรองคาลมานร่วมกับ AMCL

1.6.6 ออกแบบตัวควบคุมพีไอสำหรับมอเตอร์ขับเคลื่อนของหุ่นยนต์จริง

1.6.7 ทดสอบการประยุกต์ใช้ตัวกรองคาลมานแบบขยายร่วมกับตัวกรองอนุภาค

1.6.8 สร้างสภาพแวดล้อมจำลองเพื่อออกแบบพารามิเตอร์

1.6.9 นำพารามิเตอร์ที่ได้ออกแบบใช้งานกับหุ่นยนต์จริง

1.6.10 วิเคราะห์ผลการทดลอง

1.6.11 สรุปผลการทดลอง

1.6.12 จัดทำรูปเล่มและเนื้อหาวิทยานิพนธ์

1.7 ประโยชน์ที่คาดว่าจะได้รับ

1.7.1 สามารถพัฒนาหุ่นยนต์ AMR ที่สามารถหลบหลีกสิ่งกีดขวางแบบอัตโนมัติจากการจำลองสภาพแวดล้อมแบบ 3 มิติ

1.7.2 สามารถลดความยุ่งยากในการจัดสภาพแวดล้อมเพื่อทดสอบหุ่นยนต์ในสภาพแวดล้อมจริง

1.7.3 สามารถแก้ไขระบบจำลองเพื่อนำไปประยุกต์ใช้ในการพัฒนาระบบนำทางอัตโนมัติกับระบบขับเคลื่อนที่มีแบบจำลองจลนศาสตร์ที่แตกต่างกัน

ตารางที่ 1.1 แผนงานดำเนินงานวิจัย

กิจกรรม/ขั้นตอนการดำเนินงานวิจัย	**2565**	**2566**						**2567**
	พ.ย. ธ.ค.	**ม.ค. ก.พ.**	**มี.ค. เม.ย.**	**พ.ค. มิ.ย.**	**ก.ค. ส.ค.**	**ก.ย. ต.ค.**	**พ.ย. ธ.ค.**	**ม.ค. ก.พ.**
1. ทบทวนการศึกษารวบรวมข้อมูลและสำรวจปริทัศน์วรรณกรรม								
2. ศึกษาการสร้างระบบจำลองสภาพแวดล้อมและระบบนำทางอัตโนมัติด้วยโปรแกรม Gazebo								
3. ออกแบบโครงสร้างหุ่นยนต์และอุปกรณ์โดยใช้โปรแกรมSolidWorks								
4. ออกแบบอุปกรณ์เซนเซอร์ มอเตอร์ และประกอบกับโครงสร้างหุ่นยนต์								
5. ติดตั้งระบบ NAV2 ให้กับหุ่นยนต์ทั้ง 2 ระบบ								

ตารางที่ 1.1 แผนงานดำเนินงานวิจัย (ต่อ)

กิจกรรม/ขั้นตอนการดำเนินงานวิจัย	2565	2566						2567
	พ.ย. ธ.ค.	ม.ค. ก.พ.	มี.ค. เม.ย.	พ.ค. มิ.ย.	ก.ค. ส.ค.	ก.ย. ต.ค.	พ.ย. ธ.ค.	ม.ค. ก.พ.
6. ออกแบบพารามิเตอร์ควบคุม พีไอ								
7. ทดสอบการระบุตำแหน่งด้วยตัวกรองคาลมานร่วมกับ AMCL								
8. ใช้ระบบจำลองหาค่าพารามิเตอร์ที่เหมาะสมในการนำทางด้วย RPP								
9. วิเคราะห์ผลและสรุปผลการทดลอง								
10. จัดทำรูปเล่มและสอบวิยานิพนธ์								

บทที่ 2
ปริทัศน์วรรณกรรม ทฤษฎี และงานวิจัยที่เกี่ยวข้อง

2.1 กล่าวนำ

ระบบการรับรู้สภาพแวดล้อมแบบพลวัตและการวางแผนเส้นทางได้อย่างอิสระเป็นระบบที่ถูกพัฒนาให้สามารถนำมาใช้งานได้อย่างแพร่หลาย โดยมีจุดประสงค์ในการแก้ปัญหาที่เกิดขึ้นกับระบบนำทางอัตโนมัติ ได้แก่ ความแม่นยำ ความปลอดภัยและความน่าเชื่อถือ การเลือกใช้อัลกอริทึมที่เป็นปัจจุบันสามารถแก้ไขปัญหาที่เกิดขึ้นในอดีตและให้คุณสมบัติใหม่ ๆ เพื่อรับมือกับระบบที่มีความซับซ้อนมากยิ่งขึ้น โดยหนึ่งในระบบที่สามารถรับมือกับปัญหาดังกล่าว คือ ระบบปฏิบัติการหุ่นยนต์ ROS1 โดยปัจจุบันมีการออกเวอร์ชันล่าสุด คือ ROS2 ซึ่ง ในการพัฒนาหุ่นยนต์นำทางอัตโนมัติภายในอาคารร่วมกับระบบปฏิบัติการหุ่นยนต์ต้องอาศัยทฤษฎีที่เกี่ยวข้องหลายส่วน รวมถึงการปรับเปลี่ยนค่าพารามิเตอร์ที่มีความสัมพันธ์กับจลนศาสตร์ของหุ่นยนต์ ในบทนี้จึงนำเสนอทฤษฎีที่เกี่ยวข้องและเครื่องมือที่ใช้ในการจำลองการทำงานให้มีความทันสมัย แบ่งหัวข้อได้ดังนี้

2.2 ระบบจำลองหุ่นยนต์

Chikurtev et al., 2021 ได้ศึกษาการสร้างระบบจำลองหุ่นยนต์แบบ Differential Drive ด้วยโปรแกรม Gazebo ซึ่งเป็นโปรแกรมฟรีลิขสิทธิ์แบบโอเพ่นซอร์สที่มีความสามารถในการจำลองกฎฟิสิกส์ของนิวตันโดยใช้ Open Dynamics Engine (ODE) เป็นระบบจำลองฟิสิกส์ของสภาพแวดล้อมที่นิยมมากที่สุดในการพัฒนาหุ่นยนต์ (Erez et al., 2015) ซึ่งสามารถกำหนดโครงสร้างหุ่นยนต์ในรูปแบบ Unified Robotics Description Format (URDF) และจำลองการทำงานร่วมกับระบบนำทางอัตโนมัติที่สามารถสร้างแผนที่ วางแผนเส้นทางอ้างอิงและการระบุตำแหน่งด้วยตัวกรองอนุภาค ทำให้โปรแกรม Gazebo นั้นมีความเหมาะสมและยืดหยุ่นเพียงพอในการพัฒนาหุ่นยนต์หุ่นด้วยระบบจริง รวมถึงช่วยลดต้นทุนที่สามารถทดสอบพารามิเตอร์จากการจำลองระบบ (Tola et al., 2023)

Rivera et al., 2021 ได้กล่าวว่า การบูรณาการของเทคโนโลยีที่แตกต่างกันเป็นพื้นฐานของการปฏิวัติอุตสาหกรรมครั้งที่สี่ บริษัทได้รับการสนับสนุนให้พัฒนาเครื่องมือใหม่ ๆ ในกระบวนการผลิตเพื่อปรับปรุงสภาพการทำงาน เพิ่มผลผลิตและคุณภาพการผลิต การใช้เครื่องมือสร้างต้นแบบขั้นสูง เช่น ระบบการเขียนโปรแกรมเปิดจึงมีความจำเป็นต่อการพัฒนาโมเดลแบบมัลติบอดี้ที่มีรายละเอียดสูงผ่านการใช้ซอฟต์แวร์ CAD

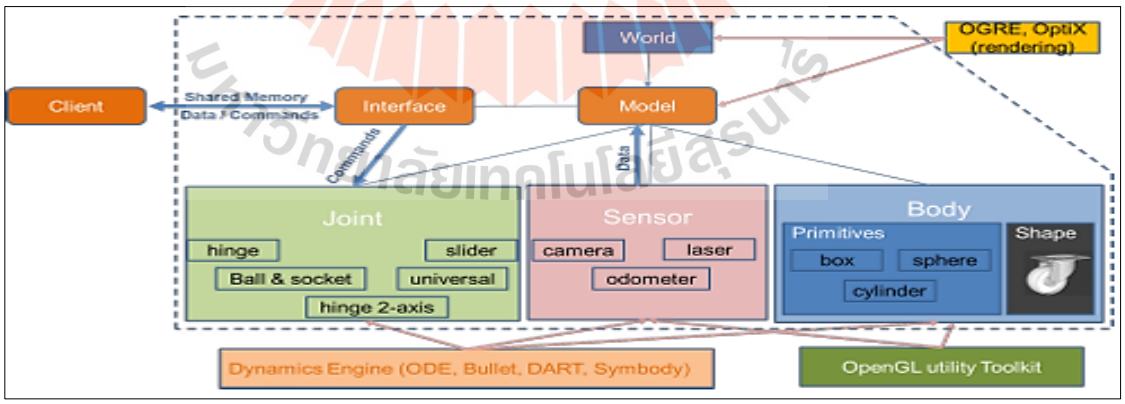
การใช้เทคนิคการเรียนรู้ด้วยตนเอง จะช่วยให้สามารถพัฒนาเครื่องจักรประเภทใหม่ได้อย่างมีประสิทธิภาพ จึงได้เสนอการพัฒนาหุ่นยนต์ขับเคลื่อนด้วยล้อ (Wheeled Mobile Robot, WMR) ร่วมกับการจำลองระบบด้วยโปรแกรม Gazebo และเสนอพารามิเตอร์ในการจำลองด้วยสัมประสิทธิ์แรงเสียดทานคูลอมบ์เท่ากับ 0.8

โปรแกรม Gazebo ถูกพัฒนาจากการเป็นส่วนหนึ่งของงานวิจัยระดับปริญญาเอกโดย Nate Koenig ซึ่งโปรแกรมสามารถคำนวณการเปลี่ยนแปลงของวัตถุแข็งเกร็งแบบจลนศาสตร์ และมีชุดเครื่องมือสร้างภาพข้อมูลอิสระของระบบเรียกว่า GLUT ที่สามารถสร้างงานแบบ 2 หรือ 3 มิติผ่านไลบรารี OpenGL ทำให้เป็นแพลตฟอร์มอิสระที่สามารถเพิ่ม Dynamic Engines เช่น Bullet, Simbody และ DART ในการสร้างหรือเพิ่มแบบจำลองที่เป็นวัตถุแบบพลวัตประกอบด้วย หุ่นยนต์ ตัวกระตุ้น (Actuators) พื้นดิน สิ่งก่อสร้าง หรือ วัตถุอื่น ๆ ด้วยสมการของ นิวตัน ออยเลอร์ เพื่อคำนวณการเคลื่อนไหวของจุดเชื่อมไร้แรงเสียดทานรวมถึงการชน สัมพันธ์กับแรงโน้มถ่วงและความเฉื่อยของวัสดุ โดยโปรแกรมประกอบด้วยการต่อประสานกับผู้ใช้งาน 2 ส่วน คือ

1) gzserver เป็นส่วนที่ใช้สำหรับการจำลองระบบฟิสิกส์ การแสดงผลและเซนเซอร์ในโลกของสภาพแวดล้อมจำลอง

2) gzclient ใช้สำหรับการแสดงผลประสานกับผู้ใช้งานผ่าน (Graphic User Interface, GUI) โดยสามารถปรับเปลี่ยนพารามิเตอร์ฟิสิกส์ให้สอดคล้องกับความต้องการของผู้ใช้

จากที่กล่าวมาระบบถูกบรรจุให้อยู่ในรูปแบบสถาปัตยกรรมที่มีการทำงานหลายส่วนเข้าด้วยกัน โดยสามารถแสดงภาพรวมของระบบการจำลองหุ่นยนต์ ดังรูปที่ 2.1



รูปที่ 2.1 สถาปัตยกรรมการจำลองสภาพแวดล้อมด้วย Gazebo

(ที่มาภาพ: Rivera et al., 2019)

2.3 การจำลองโครงสร้างหุ่นยนต์

โครงสร้างของหุ่นยนต์ทุกประเภท ประกอบด้วย มวลและโมเมนต์ของโครงสร้างในการกำหนดลักษณะการเคลื่อนที่ตามหลักแบบจำลองจลนศาสตร์และพลวัติ ซึ่งวิธีที่เป็นที่นิยมมากที่สุดในการกำหนดลักษณะและคุณสมบัติของโครงสร้างสามารถการกำหนดด้วยรูปแบบ URDF (Tola et al., 2023) เป็นรูปแบบเช่นเดียวกับภาษา (Extensible Markup Language, XML) ที่ต้องมีการระบุด้วย <tag> เปิดและ </tag> ปิด ในการอธิบายลักษณะการติดตั้งเชิงกลและกำหนดเฟรมอ้างอิงของหุ่นยนต์ให้ทราบข้อมูลของ จุดเชื่อม ลิงค์ และ เซนเซอร์ มีรายละเอียดที่ใช้กำหนดโครงสร้างการทำงาน ดังนี้

1) ลิงค์ <link> เป็นส่วนที่ใช้ในการกำหนดลักษณะโครงสร้างของส่วนต่าง ๆ มีส่วน <visual> ที่สามารถสร้างการมองเห็นเป็นรูปทรงเรขาคณิตด้วย <geometry>

2) การชน <collision> ใช้ในการกำหนดรูปร่างหรือขอบเขตของลิงค์เมื่อกระทบกับวัตถุอื่น ๆ โดยการเคลื่อนไหวจะสัมพันธ์กับโมเมนต์ความเฉื่อย

3) จุดเชื่อม <joint> ใช้กำหนดการเชื่อมต่อระหว่างลิงค์ที่สามารถกำหนดลักษณะของการเคลื่อนไหวได้ 6 ประเภท ดังนี้

3.1) Revolute Joint เป็นจุดเชื่อมที่สามารถกำหนดองศาการหมุนของวัตถุ

3.2) Continuous Joint เป็นจุดเชื่อมที่สามารถเคลื่อนที่หมุนได้อย่างต่อเนื่อง

3.3) Fixed Joint เป็นจุดเชื่อมที่ใช้ในการยึดติดแบบอยู่กับที่

3.4) Prismatic Joint เป็นจุดเชื่อมเคลื่อนที่ตามแนวแกน

3.5) floating Joint เป็นจุดเชื่อมที่สามารถเคลื่อนไหวได้แบบอิสระ (6 DOF)

3.6) Planar เป็นจุดเชื่อมที่สามารถเคลื่อนไหวในระนาบตั้งฉากกับแกน

4) วัสดุ <material> เป็นส่วนสำหรับกำหนดสีของวัสดุด้วยรหัส RGBA

5) โมเมนต์ความเฉื่อย <inertial> เป็นส่วนกำหนดคุณสมบัติความเฉื่อยของลิงค์ในรูปแบบของเมทริกซ์ขนาด [3x3] ประกอบด้วยจุดศูนย์กลางมวล <origin> และน้ำหนักของวัสดุ

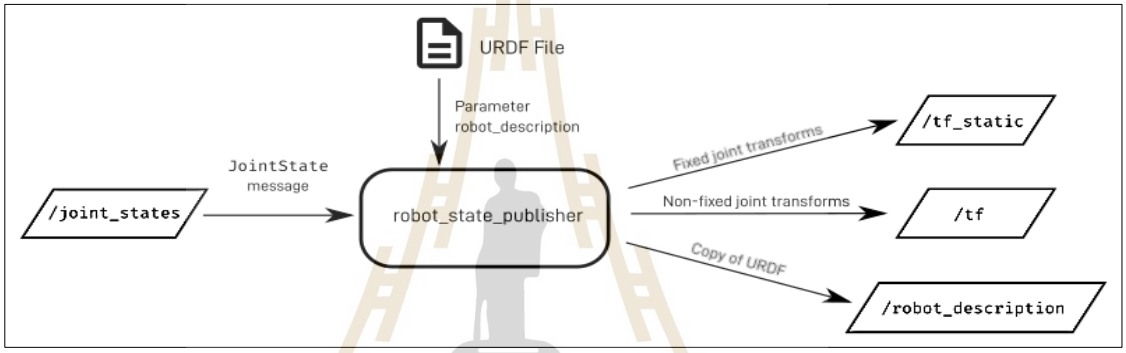
6) การแสดงผล <visual> สำหรับกำหนดลักษณะทางกราฟิกของโครงสร้าง โดยส่วนนี้สามารถนำเข้าไฟล์จากโปรแกรมเขียนแบบอย่าง SolidWorks เพื่อช่วยในการแสดงผลได้อีกด้วย

7) เซนเซอร์ < sensor> สำหรับกำหนดคุณสมบัติของเซนเซอร์ ที่ติดตั้งบนหุ่นยนต์

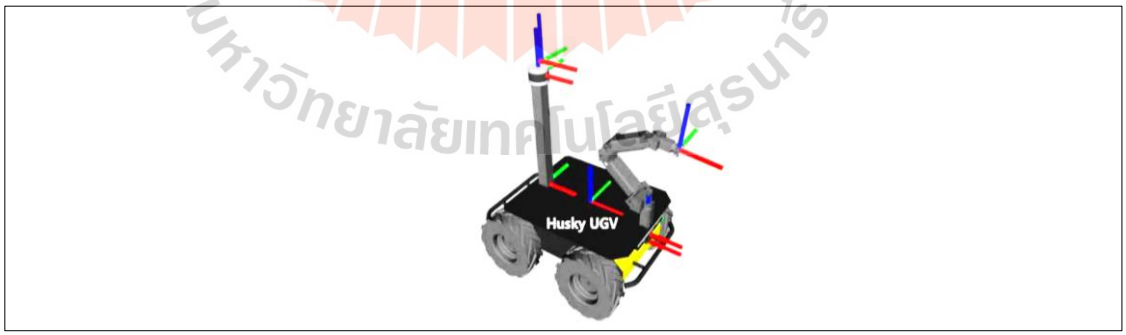
สำหรับโครงสร้างที่มีความซับซ้อนและจำเป็นต่อการเปลี่ยนแปลงคุณสมบัติ สามารถเพิ่มส่วนเสริมของ URDF เรียกว่า (XML Macros, XACRO) เพื่อให้สามารถใช้ตัวแปรในการลดการกำหนดพารามิเตอร์ที่เกี่ยวข้อง เป็นประโยชน์ต่อการปริมาณการเขียนโปรแกรมและช่วยในการจัดการโครงสร้างข้อมูลที่ได้ง่ายขึ้นอีกด้วย

2.3.1 การแสดงผลโครงสร้างหุ่นยนต์

โครงสร้างของหุ่นยนต์ที่ถูกกำหนดด้วยรูปแบบ URDF จะต้องถูกเผยแพร่ค่าสภาวะเพื่อแสดงผลการเคลื่อนไหวของโครงสร้างในรูปแบบของเฟรม (Transformation, TF) โดยให้ผลลัพธ์เป็นองศาของจุดเชื่อมบนระบบพิกัดแบบ 3 มิติ ซึ่งการแสดงผลจะส่งข้อมูลด้วย Topic ชื่อ /joint_states เป็นข้อมูลตำแหน่งของจุดเชื่อมในหน่วยที่อธิบายถึงความเร็วหรือองศาตำแหน่งปัจจุบัน ซึ่งเป็นลักษณะข้อความโดยเฉพาะให้กับไฟล์ URDF ดังรูปที่ 2.2 โดยใช้แพคเกจ robot_state_publisher ให้ที่ผลลัพธ์สุดท้ายเป็น TF ร่วมกับโครงสร้าง URDF ที่สามารถแสดงผลบนระบบพิกัดผ่านโปรแกรม RVIZ ภายใต้ Topic ชื่อว่า /robot_description ดังรูปที่ 2.3



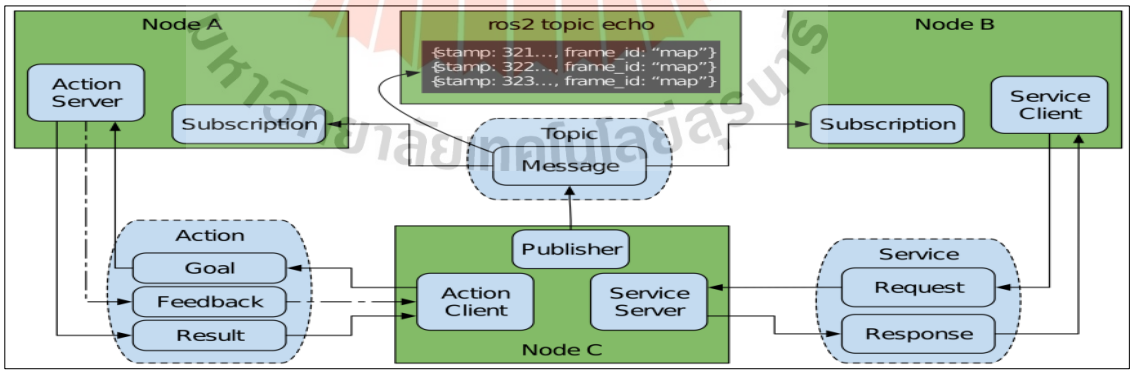
รูปที่ 2.2 การเผยแพร่สภาวะหุ่นยนต์
(ที่มาภาพ: https://articulatedrobotics.xyz/ready-for-ros-6-tf)



รูปที่ 2.3 โครงสร้างหุ่นยนต์ที่ถูกเผยแพร่สภาวะ
(ที่มาภาพ: Yoon et al., 2020)

2.4 ระบบปฏิบัติการหุ่นยนต์

ระบบพัฒนาหุ่นยนต์ได้มีการเสนอแพลตฟอร์มซอฟต์แวร์เป็นจำนวนมาก โดยเป็นระบบที่สามารถสื่อสารระหว่างกันและเชื่อมโยงแอปพลิเคชันเข้าไว้ด้วยกันเพื่อให้นักพัฒนาสามารถคิดค้นสิ่งใหม่ได้อย่างรวดเร็ว เรียกสิ่งนี้ว่า Middleware ซึ่งมีคุณสมบัติแบบแยกส่วนและปรับเปลี่ยนได้ ทำให้การสร้างระบบหุ่นยนต์ง่ายขึ้น เมื่อเวลาผ่านไปมิดเดิลแวร์บางตัวได้พัฒนาขึ้นจนกลายเป็นระบบนิเวศที่สมบูรณ์ ไม่ว่าจะเป็นการใช้ประโยชน์ อัลกอริทึมและตัวอย่างการใช้งานจริง (Macenski et al.,2022) หนึ่งในนั้น คือ ระบบปฏิบัติการ (Robot Operating System, ROS) พัฒนาโดย Willow Garage และ Open-Source Robotics Foundation (OSRF) ตั้งแต่ปี 2007 จนเกิดการแข่งขัน DARPA Urban Challenge ใน ปี ค.ศ. เดียวกันทำให้ระบบปฏิบัติการหุ่นยนต์ ROS เป็นโมดูลซอฟต์แวร์แบบเปิดที่มีการพัฒนาอย่างรวดเร็ว และได้รับการนำไปใช้งานอย่างแพร่หลาย แต่เนื่องด้วยโอกาสทางการค้าเปลี่ยนไปสู่ผลิตภัณฑ์ รากฐานของระบบ ROS เริ่มแสดงข้อจำกัดด้านความปลอดภัย ความน่าเชื่อถือและการประยุกต์ใช้งานในสภาพแวดล้อมที่หลากหลายต่อการขยายระบบ (Cairl., 2018) ให้มีขนาดใหญ่ระบบปฏิบัติการหุ่นยนต์ ROS 2 จึง ได้รับการออกแบบใหม่ตั้งแต่ต้นเพื่อจัดการกับความท้าทายเหล่านี้ด้วยระบบ (Data Distribution Service, DDS) ซึ่งเป็นมาตรฐานเปิดสำหรับการสื่อสารที่ใช้ในด้านการทหาร ยานอวกาศ และระบบการเงิน (Pardo-Castellote, 2003) เพื่อช่วยแก้ปัญหาหลายประการเกี่ยวกับการพัฒนาระบบหุ่นยนต์ที่น่าเชื่อถือ ทำให้ ROS 2 ได้รับการรักษาความปลอดภัยที่ดีที่สุด เพื่อสนับสนุนระบบฝังตัวตามเวลาจริงในสภาพแวดล้อมเครือข่ายที่ไม่เหมาะสม โดยโครงสร้างการทำงานของระบบปฏิบัติการ ROS2 อธิบาย ดังรูปที่ 2.4



รูปที่ 2.4 ลักษณะการทำงานภายในระบบ ROS2

(ที่มาภาพ: Macenski et al., 2020)

1) Node เป็นกระบวนการคำนวณของระบบแบบแยกส่วน เช่น หุ่นยนต์ต้องมีการรับข้อมูลเพื่อควบคุมให้อยู่ในเส้นทางที่กำหนดโดยมีข้อมูลจาก LiDAR ในการประมาณตำแหน่งและควบคุมหุ่นยนต์ให้สามารถเคลื่อนที่บนแผนที่โดยที่การทำงานใน 2 ส่วนนี้จะถูกแยกออกจากกัน

2) Parameter Server เป็นเซิร์ฟเวอร์ศูนย์กลางของระบบ สามารถเก็บข้อมูลพารามิเตอร์จากอัลกอริทึมที่ผู้ใช้สามารถเข้าถึงในการแก้ไขค่าเหล่านั้น

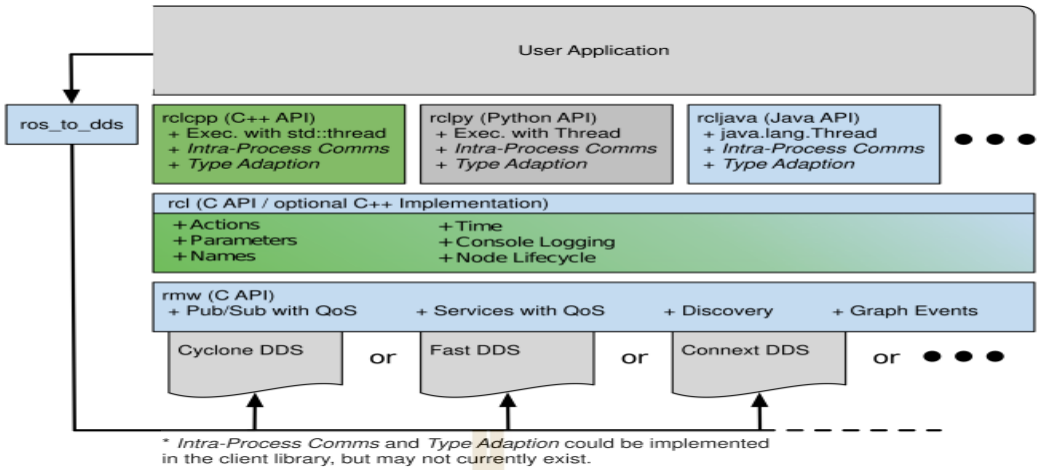
3) Topics เป็นการสื่อสารข้อมูลอย่างต่อเนื่อง สามารถกำหนดการ Publish ข้อมูลเพื่อส่งข้อมูลไปหา Node อื่น ๆ หรือ Subscribe เพื่อรับข้อมูลจากโนดอื่น ๆ โดยหนึ่ง Node สามารถสื่อสารกันได้มากกว่าหนึ่ง Topic ด้วยรูปแบบข้อมูลที่ต่างกันเรียกว่า Messages

4) Services เป็นการขอรับการบริการรวมถึงการตอบกลับจากระบบเปรียบเสมือนการส่งข้อมูลเช่นเดียวกับ Topics แต่เป็นการส่งแบบ 1 ข้อความต่อครั้ง

5) Bags ข้อมูลต่าง ๆ ในระบบที่มีการสื่อสารด้วย Topics สามารถถูกบันทึกรูปแบบไฟล์. bag หรือ .db3 โดยสามารถนำการสื่อสารของข้อมูลดังกล่าวมาเล่นซ้ำในภายหลัง

6) Messages คือ รูปแบบของข้อความที่ใช้สื่อสารกันระหว่างโนด โดยแต่ละรูปแบบจะประกอบด้วยข้อมูลที่เกี่ยวข้องกับวัตถุประสงค์การใช้งานที่แตกต่างกันออกไป เช่น การระบุตำแหน่งมีรูปแบบข้อความเป็น nav_msgs/Odometry_msgs โดยภายในจะประกอบด้วยสถานะแสดงผลการเคลื่อนที่ของหุ่นยนต์ เช่น พิกัดตำแหน่งและความเร็ว ในการเคลื่อนที่ เป็นต้น

ระบบการสื่อสารของข้อมูลภายใน ROS2 จะถูกเข้าถึงด้วย Application Programming Interfaces (API) เพื่อจัดระเบียบการสื่อสารในส่วนของ Topics, Services ,Actions, Parameters และ Timers โดยมีฟังก์ชัน Subscribe ข้อความแบบเดียวกับ ROS1 แต่มุ่งเน้นไปที่การใช้การส่งข้อความแบบไม่ใช่การโต้ตอบสื่อสารแบบทันที (Asynchronous) เพื่อจัดระเบียบการเผยแพร่ข้อมูล (Publish) เพื่อรองรับการสื่อสารในระบบที่ซับซ้อนได้ โดยสามารถแสดงการทำงานด้วยระดับขั้นในการประยุกต์ใช้ ดังรูปที่ 2.5



รูปที่ 2.5 โครงสร้าง API บนระบบ ROS2

(ที่มา: Macenski et al., 2020)

ระบบเครือข่าย (Network) เป็นส่วนสำคัญต่อการสื่อสารของระบบหุ่นยนต์ โดยมาตรฐานการสื่อสารส่วนใหญ่เป็นมาตรฐาน TCP/IP ซึ่งอาจประสบปัญหาในการส่งข้อมูลในการสื่อสารไร้สาย (Wireless) เนื่องจากการขัดจังหวะอาจทำให้เกิดการส่งข้อมูลซ้ำ และความล่าช้า ซึ่งเป็นปัญหาที่ประสบใน ROS1 โดย ROS2 ถูกพัฒนาขึ้นในการจัดการกับปัญหาดังกล่าวในการใช้ User Datagram Protocol (UDP) ในการยุติการส่งข้อมูลซ้ำแล้วให้ DDS ตัดสินใจว่าจะให้ส่งข้อมูลอีกครั้งเมื่อใดด้วย Quality of Service (QoS) ที่สามารถปรับให้เหมาะสมกับ Bandwidth ของเครือข่าย ดังนี้

1) การตั้งค่า Reliability สามารถกำหนดได้ 2 ตัวเลือก ดังนี้

(1.1) Best Effort เป็นการพยายามส่งข้อมูลให้ได้อย่างรวดเร็ว แต่ข้อมูลอาจหายไปในการณีที่สัญญาณเน็ตไม่เสถียร แต่มั่นใจได้ว่าจะไม่เกิดการล่าช้าของข้อมูลและเป็นข้อมูลล่าสุดเหมาะกับการใช้งานข้อมูลเซนเซอร์หรือข้อมูลที่ตามเวลาจริง

(1.2) Reliable เป็นการตั้งค่าที่มั่นใจได้ว่าข้อมูลที่ส่งถึงผู้รับถูกต้องครบถ้วน แต่อาจมีความล่าช้าของข้อมูลกรณีที่เครือข่ายสัญญาณไม่ดี ซึ่งการพยายามส่งให้ถึงผู้รับครบถ้วนนั้น ทำให้มีการลองใหม่หลาย ๆ ครั้ง

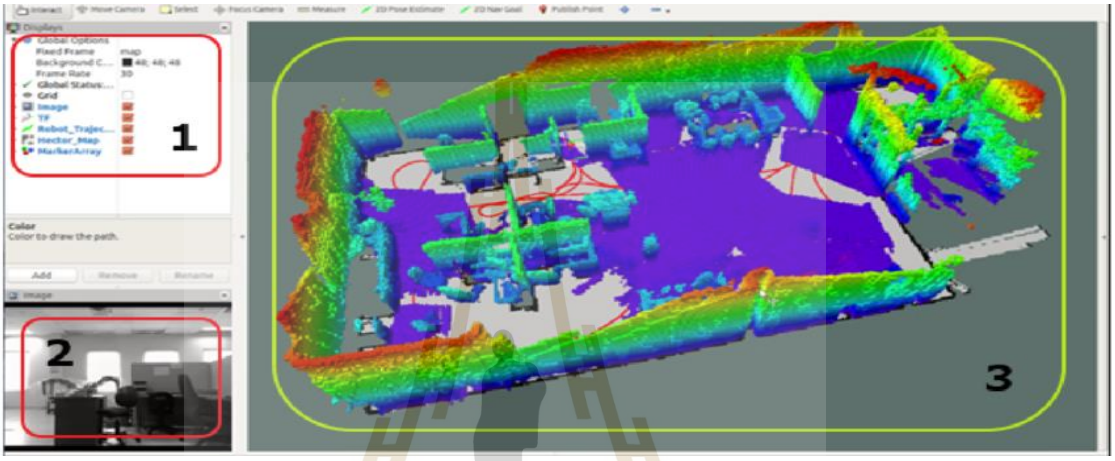
2) การตั้งค่า Durability สามารถกำหนดได้ 2 ตัวเลือก ดังนี้

(2.1) Transient Local เป็นการกำหนดข้อมูลที่ถูกเผยแพร่ไปแล้วตามประวัติคิวข้อมูล ถ้ามีการ Subscriber เกิดขึ้นมาใหม่จะสามารถได้รับข้อมูลเก่าที่อยู่ในประวัติคิวได้

(2.2) Volatile เป็นการกำหนดไม่ให้เก็บข้อมูลที่มีการเผยแพร่ออกไปแล้ว

2.5 การแสดงผลข้อมูลเซนเซอร์ด้วยกราฟฟิก

ข้อมูลที่มีการสื่อสารด้วย Topics สามารถแสดงผลลัพธ์ของข้อมูลบนระบบพิกัดแบบ 3 มิติ ด้วยโปรแกรม RVIZ ที่ขึ้นอยู่กับลักษณะของข้อความที่ใช้ในการสื่อสาร เช่น การแสดงผลของเซนเซอร์ LiDAR เป็นจุดที่มีความสัมพันธ์กับระยะห่างของตำแหน่งเซนเซอร์กับสิ่งกีดขวางในแต่ละองศาที่วัดได้บนระบบพิกัด ดังรูปที่ 2.6



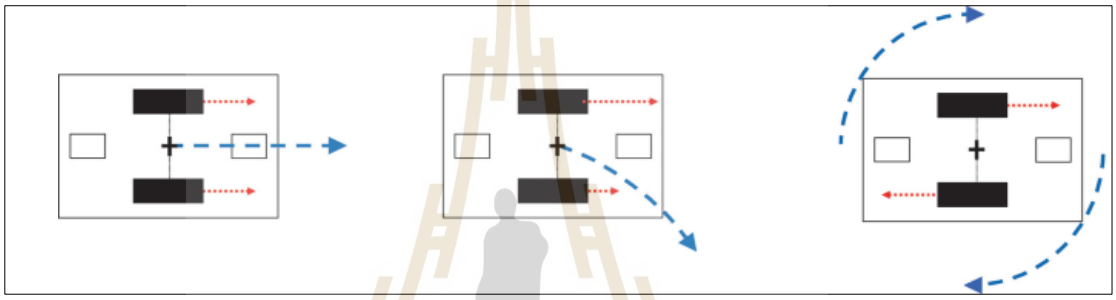
รูปที่ 2.6 การแสดงผลข้อมูลจากเซนเซอร์แบบ 3 มิติ
ที่มาภาพ: (Malavolta et al., 2021)

นอกจากนี้การพิจารณาขนาดบนระบบพิกัดของโปรแกรม RVIZ จะอ้างอิงจุดพิกัดจากจำนวนกริดที่สามารถปรับขนาดของเซลล์ให้เหมาะสม โดยปกติ 1 เซลล์มีค่าเท่ากับพื้นที่ขนาด 1 ตารางเมตร โดยสามารถเลือกข้อมูล Topic ที่ต้องการแสดงผลด้วยส่วนที่ 1 และข้อมูลจากเซนเซอร์ที่ถูกเลือกให้แสดงผลในส่วนที่ 3 นอกจากนี้ RVIZ ยังสามารถแสดงผลข้อมูลอินพุตในกรณีที่ใช้กล้องแสดงได้ในส่วนที่ 2

2.6 ระบบขับเคลื่อนหุ่นยนต์แบบองศาไม่อิสระ

(Chi et al., 2021) ได้อธิบายว่า ระบบขับเคลื่อนแบบองศาไม่อิสระ (Non-Holonomic Locomotion) หมายถึง ระบบขับเคลื่อนที่ไม่สามารถควบคุมให้เคลื่อนที่ไปยังทิศทางใด ๆ ได้โดยตรง จะต้องผ่านการเคลื่อนที่หลายขั้นตอนหรือตั้งลำตัวหุ่นยนต์ให้อยู่ในตำแหน่งที่สามารถเคลื่อนที่ไปได้ ซึ่งมีตัวแปรควบคุมน้อยกว่าตัวแปรสถานะนำไปสู่การลื่นไถล โดยมีตัวแปรสถานะเป็นสถานะการเคลื่อนที่ของหุ่นยนต์ที่สามารถอธิบายด้วยแบบจำลองทางคณิตศาสตร์ หนึ่งในระบบขับเคลื่อนองศาไม่อิสระเป็นที่นิยมมากที่สุดในการพัฒนาหุ่นยนต์ คือ ระบบขับเคลื่อนแบบ Differential Drive

(Hirpo B.D., 2017) เนื่องจากเป็นระบบที่พัฒนาง่ายที่สุดและใช้ต้นทุนต่ำที่สุด โดยสามารถเคลื่อนที่ด้วยล้อเพียง 2 ล้อ ติดตั้งที่ด้านซ้ายและขวาของโครงสร้าง รวมถึงทำงานแยกกันได้อย่างอิสระและสามารถสามารถถูกดัดแปลงจากการติดตั้งจำนวนล้อที่มากขึ้นในแต่ละด้านของโครงสร้างเพื่อเพิ่มความคล่องตัว (Caracciolo et al., 1999) แต่แลกมาซึ่งความซับซ้อนมากขึ้นของแบบจำลองจลนศาสตร์ เนื่องจากหน้าสัมผัสระหว่างพื้นมากขึ้น ส่งผลให้เกิดการลื่นไถลจากการควบคุม โดยระบบขับเคลื่อนแบบนี้ถูกเรียกว่า Skid Steering ซึ่งการควบคุมหุ่นยนต์มีรูปแบบที่เป็นไปได้ทั้งหมด 3 กรณี เหมือนกับ Differential Drive สามารถแสดงขนาดของความเร็วด้วยลูกศรเวกเตอร์ที่มีการแสดงทิศการหมุนของล้อ สัมพันธ์กับการเคลื่อนที่ของหุ่นยนต์ ดังรูปที่ 2.7



รูปที่ 2.7 รูปแบบการขับเคลื่อนหุ่นยนต์แบบ Differential Drive
ที่มาภาพ: (Bräunl., 2022)

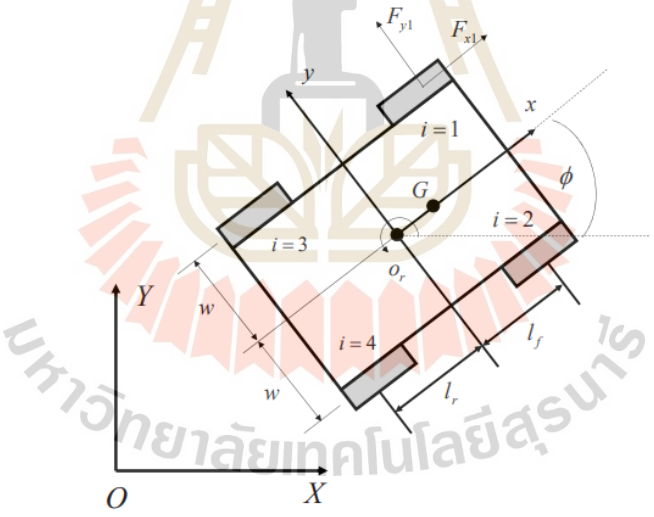
1) การบังคับทางตรง คือ กรณีที่ล้อทั้งหมดหมุนไปทิศทางเดียวกันด้วยความเร็วเท่ากัน
2) การบังคับเลี้ยว คือ กรณีที่ล้อทั้งสองด้านหมุนทิศทางเดียวกันด้วยความเร็วต่างกัน
3) การหมุน คือ กรณีที่หุ่นยนต์ควบคุมล้อหมุนทิศทางตรงข้ามกันด้วยความเร็วเท่ากัน

รูปที่ 2.8 หุ่นยนต์ Pioneer 2AT
ที่มาภาพ: (Nazari and Naraghi., 2008)

ระบบขับเคลื่อนแบบองศาไม่อิสระอีกหนึ่งชนิดที่นิยม คือ ระบบขับเคลื่อนแบบพวงมาลัย (Ackerman steering) เป็นระบบขับเคลื่อนให้กับยานยนต์ที่ใช้พวงมาลัยในการบังคับองศาเลี้ยวของล้อหน้า ทำให้ทุกล้อเคลื่อนที่ไปตามเส้นโค้งของการเคลื่อนที่โดยไม่เกิดการลื่นไถล ด้วยเหตุนี้ทำให้ระบบขับเคลื่อนแบบพวงมาลัยมีข้อจำกัดจากการต้องการองศาเลี้ยวขั้นต่ำ ส่งผลต่อความคล่องตัวที่น้อยว่าระบบขับเคลื่อนแบบ Skid steering หรือ Differential Drive ในการพิจารณาพัฒนาเป็นหุ่นยนต์อัตโนมัติ

2.6.1 แบบจำลองพลวัติของหุ่นยนต์

เมื่อยานพาหนะเกิดการเปลี่ยนแปลงพฤติกรรมจากการควบคุม การวิเคราะห์แบบจำลองพลวัติ (Dynamics Model) สามารถอธิบายพฤติกรรมการเคลื่อนที่ของหุ่นยนต์ที่มีความสัมพันธ์กับมวล (Mass) โมเมนต์ความเฉื่อย (Moment of Inertia) และแรงต่าง ๆ ที่มากระทำด้วยวิธีของลา - กรานจ์ (Lagrange's approach) หรือวิธีของนิวตัน-ออยเลอร์ (Newton-Euler's approach) (Dhaouadi and Hatab., 2013) กำหนดให้โครงสร้างมีความแข็งกระจุกตัวที่จุดศูนย์กลาง G เกิดการเคลื่อนที่บนระนาบ (x, y) ดังรูปที่ 2.9



รูปที่ 2.9 แบบจำลองพลวัติของหุ่นยนต์ Skid-steering
ที่มาภาพ: (Liao et al., 2017)

เมื่อหุ่นยนต์เกิดการเคลื่อนที่ด้วยหน้าสัมผัสระหว่างล้อกับพื้น ทำให้เกิดแรงเสียดทานที่กระทำกับล้อ สามารถอธิบายดังสมการที่ (2.1) – (2.3)

$$m(v_x - v_y\omega) = F_{x1} + F_{x2} + F_{x3} + F_{x4} + d_x \quad (2.1)$$

$$m(\dot{v}_y + v_x\omega) = F_{y1} + F_{y2} + F_{y3} + F_{y4} + d_y \quad (2.2)$$

$$J\dot{\omega} = -wF_{x1} + l_f F_{y1} + wF_{x2} + l_f F_{y2} - wF_{x3} - l_r F_{y3} + wF_{x4} - l_r F_{y4} + d_\phi \quad (2.3)$$

เมื่อ m คือ มวลของหุ่นยนต์ J คือ โมเมนต์ความเฉื่อยรอบแกน z เกิดจากแรงที่กระทำด้วย F_{xi}, F_{yi} โดยที่ $i = 1, 2, 3, 4$ เป็นเลขกำกับตำแหน่งของล้อทั้ง 4 ล้อ เมื่อ w, l_f และ l_r คือ ระยะความกว้างในหน่วยเมตรอ้างอิง ดังรูปที่ 2.9 และ d_x, d_y, d_ϕ คือ สัญญาณการรบกวนที่เข้ามาในระบบ จากการพิจารณาตัวแปรสถานะที่หุ่นยนต์สามารถเคลื่อนที่ด้วยความเร็วเชิงเส้นและความเร็วเชิงมุม $\mathrm{V} = [v_x, \omega]^T$ สามารถอธิบายดังสมการที่ (2.4) – (2.5) ดังนี้

$$\mathrm{M}\dot{\mathrm{v}} + \mathrm{A}_1 v_y = \mathrm{u} + \mathrm{d} \quad (2.4)$$

$$m(\dot{v}_y + v_x\omega) = u_y + d_y \quad (2.5)$$

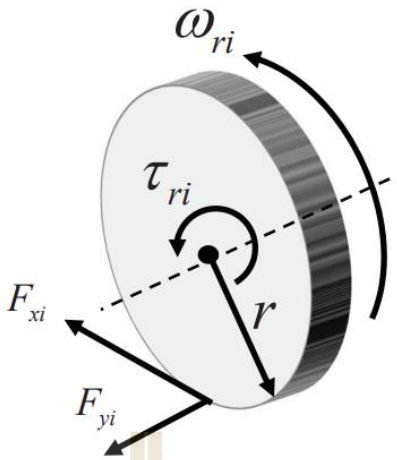
เมื่อ $\mathrm{M} = diag[m, J]$ เป็นค่าเมทริกซ์ในแนวทแยงโมเมนต์ความเฉื่อยของหุ่นยนต์ โดย $\mathrm{A}_1 = [-m\omega, 0]^T$, $\mathrm{u} = [u_x,\ u_f]^T$ คือ แรงขับที่กระทำกับหุ่นยนต์ โดยมีความไม่แน่นอนที่เกิดจากการรบกวน $\mathrm{d} = [d_x,\ d_f]^T$ ซึ่งเป็นอินพุตของระบบ ทำให้เกิดผลรวมของแรงเอาต์พุตที่ล้อตามแนวแกน x, y และรอบแกน z ดังสมการที่ (2.6) – (2.8) ตามลำดับ

$$u_x = F_{x1} + F_{x2} + F_{x3} + F_{x4} \quad (2.6)$$

$$u_y = F_{y1} + F_{y2} + F_{y3} + F_{y4} \quad (2.7)$$

$$u_\phi = -wF_{x1} + l_f F_{y1} + wF_{x2} + l_f F_{y2} - wF_{x3} - l_r F_{y3} + wF_{x4} - l_r F_{y4} \quad (2.8)$$

วิเคราะห์ผลรวมของแรงที่เกิดขึ้นเป็นแรงผลักที่กระทำต่อล้อ พบว่า ต้นกำลังเกิดจากขนาดของแรงบิดมอเตอร์ที่มีความสัมพันธ์กับแรงเสียดทานและรัศมีของล้อ ดังรูปที่ 2.10



รูปที่ 2.10 แรงบิดที่กระทำให้เกิดแรงผลักของล้อ
ที่มาภาพ: (Liao et al., 2017)

พิจารณาผลรวมของแรงที่กระทำกับล้อจนเกิดเป็นแรงผลักในการเคลื่อนที่ เป็นผลมาจากแรงบิดที่สัมพันธ์กับรัศมีของล้อหุ่นยนต์ที่วิเคราะห์แยกกันอย่างอิสระ ดังสมการที่ (2.9)

$$J_{ri}\ddot{\omega}_{ri} + c_{ri}\omega_{ri} + f_{ri} = \tau_{ri} - rF_{xi} + d_{ri} \quad (2.9)$$

เมื่อ i = 1, 2, 3, 4 แทนเลขกำกับตำแหน่งของล้อ โดยที่

J_{ri} คือ โมเมนต์ความเฉื่อยของล้อ

ω_{ri} คือ ความเร็วเชิงมุม

c_{ri} คือ ค่าคงที่ความเสียดทานของล้อ โดยมี

$f_{ri} = A_{fi}S_f(\dot{q})$ คือ แรงเสียดทานของมอเตอร์และเกียร์ทด

A_{fi} คือ สัมประสิทธิ์แรงเสียดทานคูลอมบ์และ

$S_f(\dot{q})$ คือ ฟังก์ชันการประมาณ เพื่อชดเชยแรงเสียดทาน

τ_{ri} คือ แรงบิดของมอเตอร์

r คือ รัศมีของล้อ

2.6.2 การคำนวณขนาดต้นกำลัง

เมื่อหุ่นยนต์สามารถเคลื่อนที่จากแรงผลักที่เกิดจากแรงบิดของต้นกำลัง วิธีการคำนวณหาขนาดต้นกำลังเป็นอีกวิธีหนึ่งในการตัดสินใจเลือกการเลือกอุปกรณ์ให้มีความสามารถ

ในการทำงานให้มีประสิทธิภาพและต้นทุนต่ำ เมื่อพิจารณาการเคลื่อนที่แบบ Skid steering ในกรณีที่เกิดการบังคับเลี้ยว หุ่นยนต์ต้องสามารถเคลื่อนที่ด้วยแรงบิดสูงสุดที่กระทำกับล้อด้านใดด้านหนึ่งของหุ่นยนต์ โดยที่ล้ออีกด้านมีแรงบิดเท่ากับศูนย์ สามารถอธิบายความสัมพันธ์ของแรงบิดที่กระทำดังสมการที่ (2.10)

$$\tau_R = \tau_L = 2 \cdot \mu \cdot m \cdot g \cdot r \tag{2.10}$$

เมื่อ τ_L และ τ_R คือ แรงบิดที่กระทำกับล้อด้านซ้ายและขวาตามลำดับ โดยที่แรงบิดต้องสามารถเอาชนะแรงเสียดทานที่เกิดขึ้นมีค่าเท่ากับ $\mu \cdot m \cdot g$

2.7 การระบุตำแหน่งและสร้างแผนที่

Al Khatib et al., 2020 ได้กล่าวว่า ระบบที่ต้องให้ความสำคัญและเป็นสิ่งที่ท้าทายมากที่สุดในการพัฒนาหุ่นยนต์อัตโนมัติ คือ การรับรู้ตำแหน่งของตัวเอง โดยระบบที่เป็นที่นิยมในการใช้งานภายในอาคาร คือ ระบบนำร่องด้วยแรงเฉื่อย (Inertial Navigation System, INS) เป็นการประมาณตำแหน่งโดยใช้เซนเซอร์วัดแรงเฉื่อย (Inertial Measurement Unit, IMU) ข้อดีของระบบนี้ คือ ให้อัตราการอัปเดตข้อมูลที่สูง แต่ไม่สามารถ ใช้ในการเคลื่อนที่ในระยะยาว เนื่องจากความผิดพลาดสะสมจากการอัปเดตข้อมูล ในทางกลับกัน ปัญหาดังกล่าวมักไม่พบเจอกับการระบุตำแหน่งด้วยดาวเทียม (Global Navigation Satellite System, GNSS) ซึ่งเป็นเซนเซอร์ที่สามารถวัดตำแหน่งสัมบูรณ์ที่ให้ความแม่นยำที่สูง แต่มีอัตราการอัปเดตข้อมูลที่ต่ำ ทำให้บางครั้งเกิดความไม่แน่นอนเมื่อหุ่นยนต์ขาดข้อมูลที่สำคัญ โดยการระบุตำแหน่งแบบ GNSS นั้นมักจะถูกนำไปใช้กับการนำทางภายนอกอาคาร (Huang et al., 2023) ด้วยปัญหาที่เกิดขึ้นกับระบบนำทางทั้ง 2 ระบบ ทำให้นักวิจัยมักจะคิดค้นอัลกอริทึมที่สามารถสามารถทำให้หุ่นยนต์เผชิญกับความท้าทายต่าง ๆ เพื่อรับมือกับสภาพแวดล้อมแบบพลวัติ

Moore and Stouch., 2017 กล่าวว่า ในความเป็นจริงเซนเซอร์ทุกชนิดจะถูกรบกวนสัญญาณจากการวัดและเป็นสิ่งที่ไม่สามารถเลี่ยงได้ จึงได้เสนอตัวกรองคาลมานแบบขยาย เพื่อประมาณค่าของระบบที่ไม่เป็นเชิงเส้นและกำจัดสัญญาณรบกวนจากการวัดโดยสามารถปรับพารามิเตอร์ที่อยู่ในรูปเมทริกซ์ความแปรปรวน นอกจากนี้ยังสามารถบูรณาการข้อมูลที่วัดได้อย่างหลากหลายเพื่อเป็นอินพุตในการประมาณสถานะของระบบ จากทดสอบด้วยการประยุกต์ใช้เพื่อประมาณตำแหน่งของหุ่นยนต์ภายนอกอาคารด้วยการบูรณาการเซนเซอร์ 5 ชนิด พบว่า พิกัดที่ได้จากการประมาณด้วยตัวกรองคาลมานสามารถลดความผิดพลาดสะสมได้ถึง 98.87% เปอร์เซ็นต์ และส่วนเบี่ยงเบนมาตรฐานของข้อมูล 99.1 เปอร์เซ็นต์

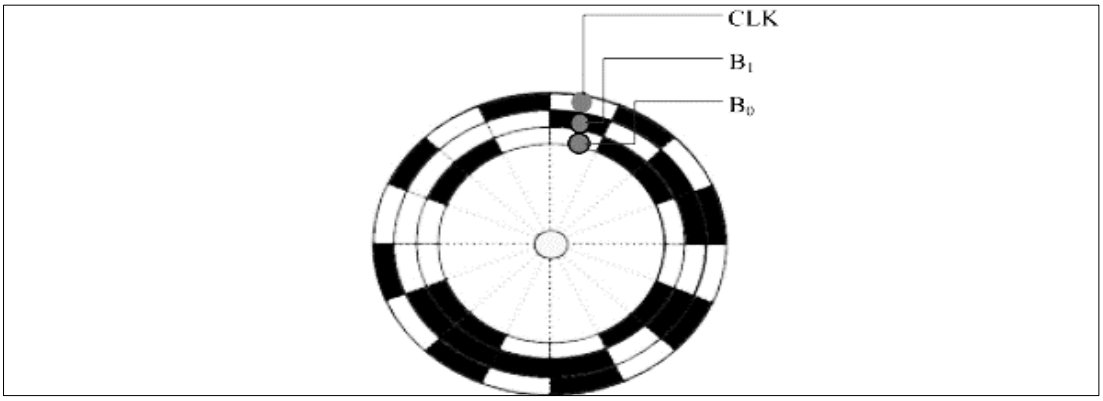
Wang et al., 2015 ได้ศึกษาแบบจำลองจลนศาสตร์ระบบขับเคลื่อนแบบ Skid Steering พบความสัมพันธ์ระหว่างสัมประสิทธิ์ (Instantaneous center of rotation, ICR) กับเส้นทางที่มีรัศมีเส้นโค้งแบบไม่คงที่ ด้วยหุ่นยนต์ P3-AT ระบุตำแหน่งด้วยการทำ Dead-Reckoning โดยใช้เซนเซอร์ LiDAR รุ่น LMS400 เพื่อวัดตำแหน่งจริงของหุ่นยนต์ (Ground truth) โดยพิจารณาให้หุ่นยนต์มีความสมมาตรด้วยการคงที่สัมประสิทธิ์ ICR เท่ากับ 1.5 เปรียบกับแบบจำลองหุ่นยนต์ด้วยระบบพลวัติร่วมกับแบบจำลองจลนศาสตร์ที่สามารถมีค่าสัมประสิทธิ์ ICR แบบปรับตัวตามรัศมีความโค้ง พบว่าแบบจำลองที่ค่าสัมประสิทธิ์ปรับตัวได้สามารถลดความผิดพลาดการระบุตำแหน่งด้วยวิธี Dead-reckoning จากลักษณะการเคลื่อนที่แบบเดียวกัน ด้วยค่า RMSE น้อยกว่า 0.03 เมตร

Long et al., 2022 ได้พบว่า หุ่นยนต์ที่ใช้ข้อมูลการวัดระยะทางแบบ Odometry ในการประมาณตำแหน่งของหุ่นยนต์ที่มีความซับซ้อนของพื้นผิวสัมผัสกับล้อเป็นผลทำให้หุ่นยนต์เกิดความผิดพลาดสะสมของพิกัดในระยะทางที่ไกลขึ้น ทำให้สูญเสียความน่าเชื่อถือต่อระบบนำทางอัตโนมัติและการเก็บข้อมูลแผนที่ จำเป็นต่อการระบุตำแหน่งด้วย (Adaptive Monte Carlo Localization, AMCL) โดยอัลกอริทึมดังกล่าวสามารถปรับตำแหน่งของหุ่นยนต์บนแผนที่ด้วยข้อมูลการวัดระยะด้วยแสงด้วยความน่าจำเป็นของอนุภาค (Particle) ที่กระจายรอบ ๆ ตำแหน่งของหุ่นยนต์ ซึ่งจากการทดสอบด้วยการเคลื่อนที่แบบเดียวกันด้วยความเร็วที่แตกต่างกัน 0.3 0.5 และ 0.7 เมตรต่อวินาที พบว่าตำแหน่งมีความผิดพลาดของพิกัดสะสมต่างกันเพียง 0.01 เมตร ที่ความเร็วแตกต่างกัน โดยมีค่า RMSE เท่ากับ 0.03 0.034 และ 0.04 เมตร ตามลำดับ

Kamarudin et al, 2015 ได้ศึกษาการสร้างแผนที่ด้วยวิธีการ SLAM โดยใช้ Gmapping เป็นการสร้างแผนที่ที่อาศัยข้อมูลการวัดระยะทางแบบ Odometry เปรียบเทียบกับ Hector Slam ที่สามารถสร้างแผนที่โดยไม่อาศัยข้อมูลการวัดระยะทาง ในการสร้างแผนที่แบบ 2 มิติ ด้วยสภาพแวดล้อมแบบเดียวกัน ซึ่งมีขนาด 4.5×3.18 ตารางเมตร พบว่าการสร้างแผนที่โดยอาศัยข้อมูลการวัดระยะทางด้วย Gmapping มีความแม่นยำมากกว่า โดยมีผลลัพธ์พื้นที่ในการสแกนเท่ากับ 4.5×3.15 ตารางเมตร เมื่อเทียบกับ Hector ให้ผลลัพธ์ขนาดแผนที่เท่ากับ 4.4×3.05 ตารางเมตร

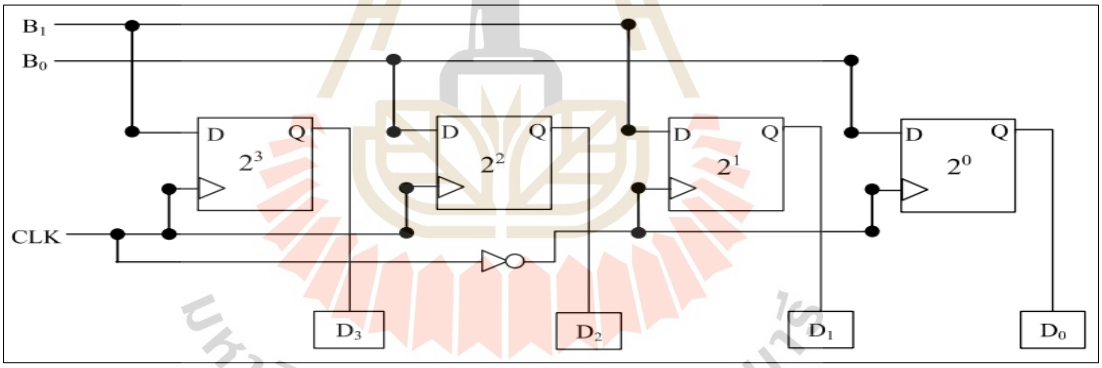
2.7.1 เซนเซอร์เข้ารหัสสัมบูรณ์

เซนเซอร์เข้ารหัสสัมบูรณ์ (Absolute Encoder) เป็นอุปกรณ์ที่ใช้ในการรับรู้ตำแหน่งหรือทิศทางของวัตถุ สามารถนำมาประยุกต์ใช้ในการประมาณตำแหน่งของหุ่นยนต์จากความสามารถในการวัดความเร็วหรือตำแหน่งที่เกิดจากการหมุน มีข้อดีมากกกว่าเซนเซอร์แบบ Incremental ด้วยการเพิ่มหัวอ่านให้เท่ากับจำนวนบิตของเอาต์พุตและเจาะรูบนแผ่นดิสก์ให้มีระยะห่างแบบทวีคูณ ทำให้เซนเซอร์สามารถจดจำตำแหน่งเริ่มต้นและมีความละเอียดสูงมากกว่าจากกลไกการถอดรหัสด้วยตรรกะเชิงดิจิทัลในรูปแบบเลขฐานสอง (Binary Code) ดังรูปที่ 2.11



รูปที่ 2.11 ตำแหน่งการเจาะรูบนแผ่นดิสก์แบบ 4 บิต
ที่มาภาพ: (Das et al, 2016)

เมื่อแผ่นดิสก์เกิดการหมุนจะถูกตรวจจับด้วยเซนเซอร์ลำแสง Optical กรณีที่สามารถตรวจจับการทะลุผ่านของแผ่นดิสก์ ทำให้เกิดการเข้ารหัสของวงจรภายใน ดังรูปที่ 2.12



รูปที่ 2.12 วงจร D Flip-flop การถอดรหัสแบบ 4 บิต
ที่มาภาพ: (Das et al, 2016)

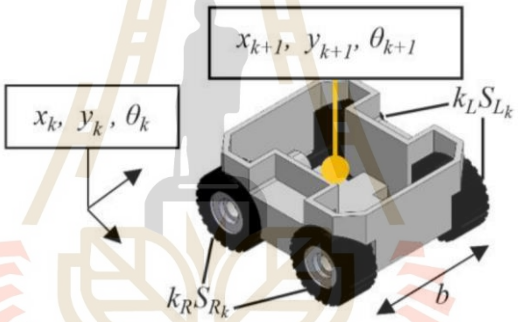
เมื่อแผ่นดิสก์เกิดการหมุนทำให้เกิดสัญญาณพัลส์ ณ CLK เป็นสัญญาณดิจิทัลมีค่า 0 กับ 1 สลับกัน จะได้ว่าสัญญาณ ณ ตำแหน่งบิตที่ 0 และ 1 ถูกเชื่อมต่อผ่านตรรกะ (Logic) แบบนิเสธ (Not Gate) เพื่อเปลี่ยนสถานะสัญญาณในการเข้ารหัสสถานะ 0 เป็น 1 และ 1 เป็น 0 ส่วนตำแหน่งบิตที่ 2 และบิตที่ 3 ถูกเชื่อมต่อผ่านจากสัญญาณ CLK โดยตรงทำให้ไม่เกิดการเปลี่ยนสถานะของสัญญาณสำหรับบิตที่ 2 และบิตที่ 3 เมื่อเซนเซอร์ส่งสถานะจากแผ่นดิสก์ ณ ตำแหน่ง B_1 และ B_0 ทำให้เกิดผลลัพธ์การเข้ารหัส ดังตารางที่ 2.1

ตารางที่ 2.1 เอาต์พุตของการเข้ารหัสด้วยวงจร D Flip-flop

CLK	D_3	D_2	D_1	D_0
HIGH	B_1	B_0	B_{1Prev}	B_{0Prev}
LOW	B_{1Prev}	B_{0Prev}	B_1	B_0

2.7.2 ทฤษฎีการประมาณตำแหน่งบนระบบพิกัด 2 มิติ

การเคลื่อนที่ออกจากจุดเริ่มต้น พิกัดตำแหน่งของหุ่นยนต์จะเปลี่ยนไปขึ้นอยู่กับลักษณะการควบคุมตามเวลาจริง สำหรับหุ่นยนต์ที่สามารถประมาณตำแหน่งโดยไม่ใช้เซนเซอร์ GPS อีกวิธีหนึ่งที่นิยม คือ การใช้การวัดความเร็วจาก Encoder แล้วนำมาคำนวณพิกัดที่เปลี่ยนไป เรียกวิธีการนี้ว่า Dead Reckoning ซึ่งให้ความแม่นยำต่ำต่อระยะทางที่มากขึ้น โดยตัวแปรสถานะที่เกี่ยวข้องจะเป็นพิกัดของหุ่นยนต์ ณ เวลาใด ๆ สัมพันธ์กับสัญญาณพัลส์ ดังรูปที่ 2.13



รูปที่ 2.13 แบบจำลองการระบุตำแหน่งของหุ่นยนต์

$$x_{k+1} = x_k + \left(\frac{k_R S_{R_k} + k_L S_{L_k}}{2}\right)\cos\left(\theta_k + \frac{k_R S_{R_k} - k_L S_{L_k}}{2b}\right) \tag{2.11}$$

$$y_{k+1} = y_k + \left(\frac{k_R S_{R_k} + k_L S_{L_k}}{2}\right)\sin\left(\theta_k + \frac{k_R S_{R_k} - k_L S_{L_k}}{2b}\right) \tag{2.12}$$

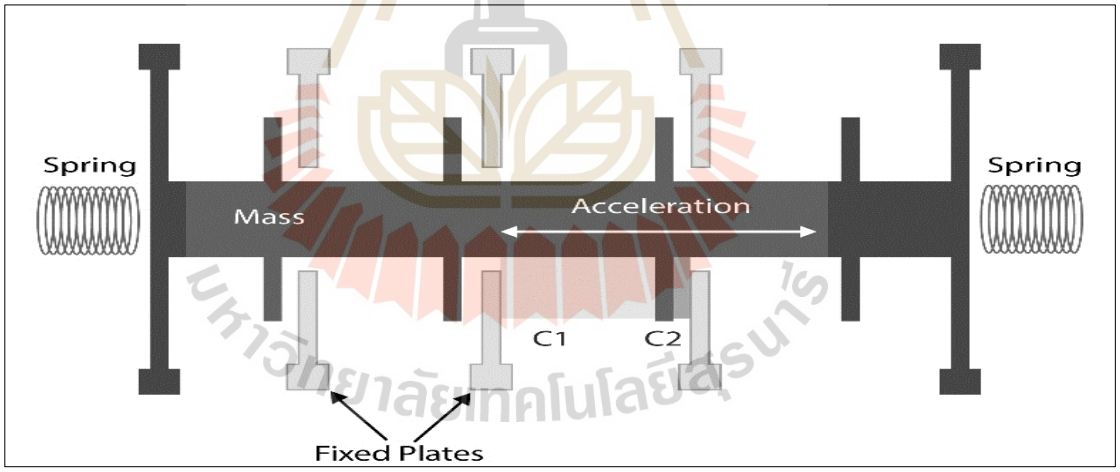
$$\theta_{k+1} = \theta_k + \left(\frac{k_R S_{R_k} - k_L S_{L_k}}{\chi b}\right) \tag{2.13}$$

โดยที่ x_k, y_k, θ_k คือ พิกัดและทิศการหมุนของหุ่นยนต์ ณ เวลาปัจจุบัน x_{k+1}, y_{k+1}, θ_{k+1} คือ พิกัดและทิศการหมุนของหุ่นยนต์ เคลื่อนที่ออกจากจุด x_k, y_k และ S_{R_k}, S_{L_k} คือ สัญญาณพัลส์ (Pulse) ของ Encoder ล้อซ้ายและขวาของหุ่นยนต์ k_R, k_L คือ สัมประสิทธิ์ปรับ

สัญญาณพัลส์ โดย b คือ ระยะระหว่างล้อซ้ายและขวาของหุ่นยนต์ และ χ คือ สัมประสิทธิ์ ICR เพื่อกำหนดจุดหมุนของหุ่นยนต์แบบ Skid Steering

2.7.3 ระบบนำร่องด้วยแรงเฉื่อย

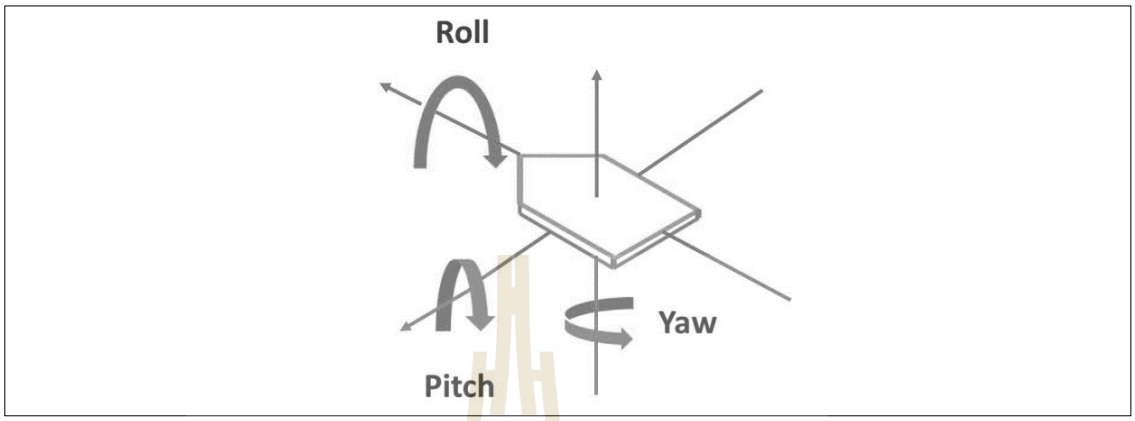
หนึ่งในเซนเซอร์เป็นที่นิยมเพื่อวัดการเคลื่อนไหวได้แบบองศาอิสระ (Degree of Freedom, DOF) คือ เซนเซอร์ (Inertial Measurement Unit, IMU) ประกอบด้วย เซนเซอร์วัดความเร่ง (Accelerometer) 3 แกน และเซนเซอร์ไจโรสโคปในการวัดความเร็วเชิงมุมรอบแกนทั้งหมด 3 แกน โดยเมื่อนานมาแล้วมีการนำ IMU ไปหาตำแหน่งของยานพาหนะ โดยการอินทิเกรทค่าความเร่ง 2 ครั้ง และองศาทิศการหมุน (Orientation Angle) โดยอินทิเกรทความเร็วเชิงมุม 1 ครั้ง ปรากฏว่าทำให้พิกัดตำแหน่งมีความผิดพลาดสะสม เนื่องจากสัญญาณรบกวน (Noise) ที่ไม่ทราบค่า (Jay, 2008) ในส่วนการวัดความเร่งตามแนวแกน (Linear Acceleration) จะมีอุปกรณ์ภายในเซนเซอร์ ที่สามารถวัดความเร่งด้วยแรงของสปริงที่ติดอยู่กับฐานของเซนเซอร์ (Fixed Plate) ดังรูปที่ 2.14 เมื่อเกิดการเคลื่อนที่ตามแนวแกนทำให้แรงสปริงมีขนาดมากกว่าแรงโน้มถ่วงของโลก สปริงจะเคลื่อนที่ออกไปตามแนวแรงที่เกิดขึ้น ทำให้สามารถวัดความเร่งเพื่อระบุวัตถุในสถานะนิ่งเฉยหรือเคลื่อนไหว



รูปที่ 2.14 ส่วนประกอบเซนเซอร์วัดความเร่ง
ที่มาภาพ: (https://shorturl.at/dwlLU)

ไจโรสโคป (Gyroscopes) เข้ามามีบทบาทในการวัดความเร็วเชิงมุมของวัตถุจากการอาศัยแรงโน้มถ่วงของโลกสัมพันธ์กับแรงดึงของโรเตอร์ ที่ถูกตรึงเอาไว้ในกรอบของเซนเซอร์ เมื่อมีแรงมากระทำมากกว่าดึงทำให้ส่วนของโรเตอร์เคลื่อนที่และเกิดการวัดค่าของเซนเซอร์ไจโรสโคป

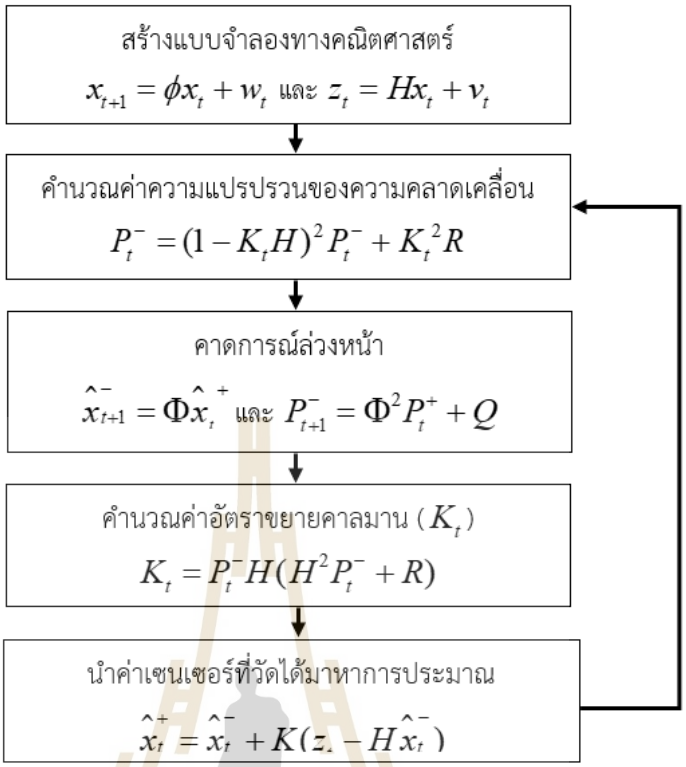
ประกอบด้วย 3 ส่วน คือ Roll, Pitch และ Yaw ดังรูที่ 2.15 ซึ่งเซนเซอร์สมัยใหม่จะอยู่ในรูปแบบ MEMS ที่มีขนาดเล็ก ทำให้การนำไปใช้งานไม่จำเป็นต้องมีพื้นที่ติดตั้งขนาดใหญ่



รูปที่ 2.15 ความสัมพันธ์ทิศการหมุนกับเซ็นเซอร์ไจโรสโคป
ที่มาภาพ: (Ahmad et al., 2017)

2.7.4 ตัวกรองคาลมานแบบขยาย

การระบุตำแหน่งที่ต้องอาศัยข้อมูลการวัด ในทางปฏิบัตินั้นสิ่งเหล่านี้เป็นปัญหาที่ท้าท้ายด้วยข้อจำกัดของเซนเซอร์ที่ไม่สามารถวัดสถานะออกมาได้อย่างถูกต้องร้อยเปอร์เซ็นต์ เนื่องมาจากความแปรปรวนของสัญญาณที่ถูกรบกวนโดยวงจรอิเล็กทรอนิกส์หรือสภาพแวดล้อม ทำให้นักพัฒนาได้คิดค้นอัลกอริทึมที่สามารถแก้ไขปัญหาดังกล่าวประกอบด้วยระบบที่เป็นเชิงเส้นและระบบที่ไม่เป็นเชิงเส้น โดยระบบเชิงเส้นนั้น หมายถึง ระบบที่มีแบบจำลองและเวกเตอร์การวัดเป็นฟังก์ชันแบบเชิงเส้นซึ่งเป็นระบบที่เหมาะสมกับการประมาณสถานะด้วยตัวกรองคาลมาน (Kalman Filter) แต่สำหรับระบบที่ไม่เป็นเชิงเส้นต้องมีการประยุกต์ใช้ตัวกรองที่ไม่เป็นเชิงเส้น เช่น ตัวกรองคาลมานแบบขยาย (Extended Kalman Filter, EKF) เป็นต้น โดยหลักการประมาณสถานะของตัวกรองคาลมานแบบ EKF จะประกอบด้วยกัน 2 ส่วนคือการคาดการณ์และการแก้ไขค่าอธิบายได้ ดังรูปที่ 2.16



รูปที่ 2.16 ขั้นตอนการประมาณสถานะด้วยตัวกรองคาลมาน

(1) การคาดการณ์

ขั้นตอนแรกของการประมาณด้วยตัวกรองคาลมาน กำหนดให้ระบบมีตัวแปรสถานะเบื้องต้น (Prior State) โดยไม่พิจารณาสัญญาณรบกวน ซึ่งในกรณีที่ระบบไม่เป็นเชิงเส้นสามารถอธิบายให้อยู่ในรูปฟังก์ชันของตัวแปรสถานะจากการคาดการณ์โดยการแก้สมการเชิงอนุพันธ์ดังสมการที่ (2.14)

$$\dot{x}^-(t) = f(x^-(t)) \tag{2.14}$$

แทนสมการนี้ด้วยอนุกรมเทเลอร์เทียบกับสถานะ x และแทนค่าตัวแปรสถานะที่ได้จากการคาดการณ์ $\dot{x}^-(t)$ โดยสมมติว่าพจน์ที่มีอันดับสูงสามารถตัดทิ้งได้ สามารถอธิบายเมทริกซ์ทางพลศาสตร์ ณ เวลา t ด้วยสมการที่ (2.15)

$$F(t_i) = \frac{\partial f(x)}{\partial x}_{x=x^-(t_i)} \tag{2.15}$$

การคาดการณ์สามารถคำนวณด้วยเมทริกซ์ที่ไม่เป็นค่าคงที่สำหรับระบบไม่เชิงเส้น โดยเป็นค่าที่ขึ้นอยู่กับเวลา ดังสมการที่ (2.16)

$$\Phi_{t_{i-1}}^{t_i} = F(t_i) \cdot \Phi_{t_{i-1}}^{t_i} - 1 \tag{2.16}$$

เมื่อ $\Phi_{t_{i-1}}^{t_i}$ คือ เมทริกซ์การถ่ายโอนตัวแปรสถานะ ซึ่งดัดแปลงมาจากตัวแปรสถานะใด ๆ สามารถนำไปคำนวณเมทริกซ์ความแปรปรวนของเวกเตอร์ตัวแปรสถานะจากการคาดการณ์ ดังสมการที่ (2.17)

$$P^{-}(t_i) = \Phi_{t_{i-1}}^{t_i} \cdot P(t_i - 1) \cdot (\Phi_{t_{i-1}}^{t_i})^T + Q \tag{2.17}$$

ในรูปทั่วไปแล้วเมทริกซ์ความแปรปรวนจะเป็นฟังก์ชันของเวลาสามารถอธิบาย ดังสมการที่ (2.18)

$$P^{-}(t_i) = \Phi_{t_{i-1}}^{t_i} \cdot P(t_i - 1) \cdot (\Phi_{t_{i-1}}^{t_i})^T + \int_{t_i}^{t_1} Q(t)dt \tag{2.18}$$

(2) การแก้ไขค่า

เช่นเดียวกับการแก้สมการเชิงอนุพันธ์ในขั้นตอนการคาดการณ์การสังเกตการณ์ที่ไม่เป็นเชิงเส้นสอดคล้องกันจะถูกทำให้เป็นเชิงเส้นด้วยอนุกรมเทเลอร์ และพจน์อันดับสูงจะถูกตัดทิ้งไป ดังนั้นเมทริกซ์การสังเกตการณ์สามารถประมาณได้ ดังสมการที่ (2.19)

$$H(t_i) = \frac{\partial h(x)}{\partial x}_{x=x^{-}(t_i)} \tag{2.19}$$

ตัวกรองคาลมานเป็นตัวกรองที่เหมาะสมที่สุดหมายความว่าความแปรปรวนของตัวแปรสถานะในเมทริกซ์ความแปรปรวนของตัวแปรสถานะ P^{+} จะมีค่าที่น้อยที่สุดขณะที่ P^{-} คำนวณมาจากขั้นตอนการคาดการณ์ได้ ΔP ที่เป็นค่าน้อยที่สุด ดังสมการที่ (2.20)

$$\Delta P(t_i) = E\left[\Delta x(t_i) \Delta x(t_i)^T\right] \tag{2.20}$$

โดยมีเงื่อนไขสอดคล้องดังสมการที่ (2.21) – (2.23)

$$\Delta x(t_i) = P^{-}H^{T}(HP - H^{T} + R(t_i))^{-1}(l(t_i) - Hx^{-}(t_i)) \quad (2.21)$$

$$\Delta x(t_i) = K(t_i)(l(t_i) - l^{-}(t_i)) \quad (2.22)$$

$$K(t_i) = P^{-}H(t_i)^{T}(H(t_i)P^{-}H(t_i)^{T}) + R(t_i))^{-1} \quad (2.23)$$

เมื่อ K เรียกกว่าเมทริกซ์อัตราขยายคาลมานและ $l(t_i) - l^{-}(t_i)$ คือ ค่าที่เหลือจากการวัด แสดงให้เห็นถึงความแตกต่างระหว่างการวัดที่คาดการณ์ $l(t_i) - Hx^{-}(t_i)$ และการวัดที่ได้จริง $l(t_i)$ สุดท้ายแล้วตัวแปรสถานะที่ถูกแก้ไขจะได้จากการคำนวณดังสมการที่ (2.24)

$$x^{+}(t_i) = x^{-}(t_i) + K(l(t_i) - l^{-}(t_i)) \quad (2.24)$$

ในสมการนี้ตัวแปรสถานะที่ถูกประมาณและค่าที่วัดได้นั้นจะถูกแบ่งความสำคัญ ตามน้ำหนักแล้วนำมารวมกันเพื่อคำนวณตัวแปรสถานะที่ถูกแก้ไข ซึ่งหมายความว่า ถ้าค่าความแปรปรวนจากการวัดนั้นมีค่าน้อยกว่าค่าความแปรปรวนของตัวแปรสถานะที่ได้จากการคาดการณ์ น้ำหนักของการวัดจะสูงกว่าและตัวแปรสถานะที่ได้จากการณ์คาดการณ์จะต่ำ ดังนั้นความไม่แน่นอนจะสามารถลดลงได้ด้วยเมทริกซ์ความแปรปรวนของตัวแปรสถานะหลังแก้ไขจะได้จากกฎของค่าแพร่กระจายของค่าความผิดพลาด ดังสมการที่ (2.25)

$$P^{+}(t_i) = (I - K(t_i)H(t_i))P^{-}(t_i) \quad (2.25)$$

2.7.5 เทคโนโลยี LiDAR

(Light Detection and Ranging, LiDAR) ถูกใช้อย่างแพร่หลายในวงการภูมิสารสนเทศและสร้างแผนที่ เป็นเซนเซอร์ที่สามารถติดตั้ง บนพื้นโลก เครื่องบิน หรือดาวเทียมสำหรับวัดความสูงของภูมิประเทศ โดยข้อมูลสำคัญที่ได้จากการวัด คือ ความแตกต่างของระยะเวลาที่ลำแสงเลเซอร์ถูกส่งไปแล้วกระทบกลับมา ทำให้สามารถคำนวณหาระยะระหว่างแหล่งกำเนิดแสงกับวัตถุ เพื่อประยุกต์ใช้ในอุตสาหกรรมหุ่นยนต์ในการตรวจจับวัตถุและสร้างแผนที่ด้วยวิธีการ SLAM แต่ถึงอย่างไรเซนเซอร์ LiDAR ใช้แสงที่มีความยาวคลื่นในช่วง 750 นาโนเมตร ถึง 1500 ไมโครเมตร เรียกว่า (Near infrared, NIR) เป็นช่วงความยาวคลื่นที่มีข้อจำกัดในการตรวจจับสีวัตถุสีดำเนื่องจาก

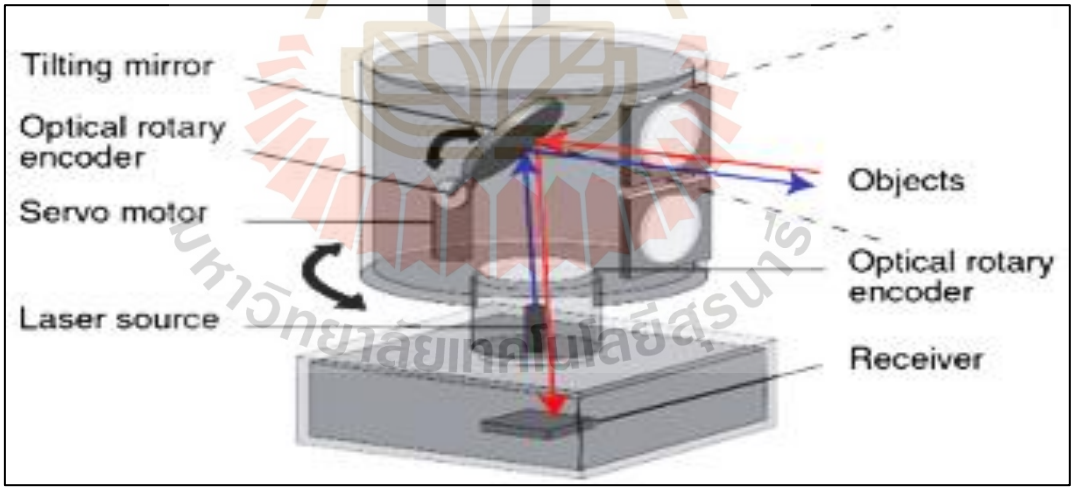
เป็นสีที่มีคุณสมบัติดูดกลืนแสง (Jekal et al., 2022) ทำให้คุณสมบัติการสะท้อนกลับไปที่ตัวเซนเซอร์ลดลง ซึ่งกระบวนการคำนวนเมื่อเลเซอร์กระทบกับวัตถุจะเกิดการสะท้อนกลับมาที่แหล่งกำเนิด สัมพันธ์กับระยะทางที่ใช้ในการเดินทางไปและกลับ ดังสมการที่ (2.26)

$$s = ut + \frac{1}{2}at^2 \tag{2.26}$$

เมื่อ $a = 0$, $u = c$ และ $s = d$ แทนลงในสมการที่ (2.26) ได้ผลลัพธ์ ดังสมการที่ (2.27) เป็นระยะที่เซนเซอร์สามารถตรวจจับได้

$$d = \frac{c \times t}{2} \tag{2.27}$$

เมื่อ d คือ ระยะห่างระหว่างแหล่งกำเนิดไปยังสิ่งกีดขวางในหน่วย เมตร c คือ ความเร็วแสงเท่ากับ 3×10^8 เมตรต่อวินาที และ t คือ เวลาที่แสงเริ่มเคลื่อนที่จากจุดกำเนิดรวมถึงสะท้อนกลับเข้ามาที่ตัวรับแสงในหน่วย วินาที โดยส่วนประกอบของเซนเซอร์สามารถอธิบายได้ดังรูปที่ 2.17



รูปที่ 2.17 ส่วนประกอบของ LiDAR
ที่มาภาพ: (https://erc-bpgc.github.io/handbook/electronics/Sensors/lidar)

(1) แหล่งกำเนิดแสง (Laser Source) เป็นแหล่งกำเนิดที่ตั้งฉากกับฐานเซนเซอร์

(2) Servo Motor ใช้ในการหมุนกระจกสะท้อนแสง (Tilting mirror) เพื่อหักเหแสงจากแหล่งกำเนิดให้แสงพุ่งออกในแนวตั้งฉากกับเซนเซอร์

(3) ตัวควบคุม (Controller) เป็นส่วนที่ใช้ควบคุมการทำงานของ Servo motor ให้มอเตอร์สามารถหมุนด้วยความเร็วคงที่ตามคุณสมบัติจากผู้ผลิต

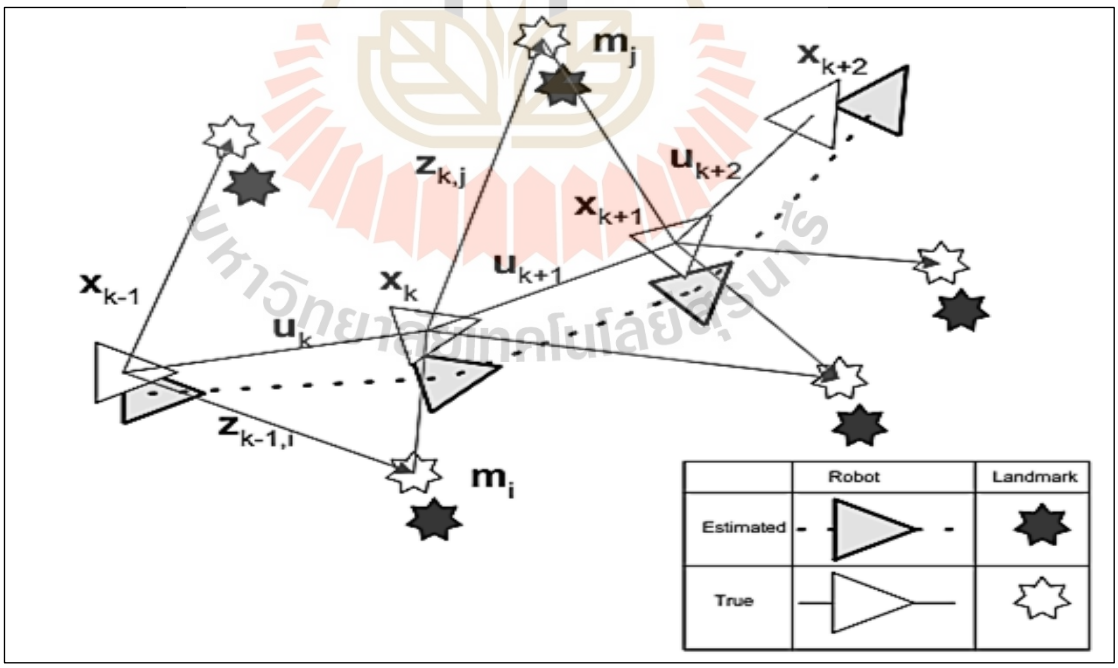
(4) ตัวอ่านข้อมูล (Scanner) มีหน้าที่อ่านข้อมูลในระบบ LiDAR เป็นส่วนที่มีหน้าที่อ่านและรับข้อมูลระยะห่างของสิ่งกีดขวางและทิศทางจากการส่องแสงเลเซอร์จากการสะท้อน

(5) เซนเซอร์เข้ารหัส (Optical rotary encoder) มีหน้าที่คำนวณองศาการหมุนของเซนเซอร์ ในการอ่านระยะระหว่างสิ่งกีดขวางกับเซนเซอร์ในแต่ละองศาที่วัดได้

(6) ตัวรับแสง (Receiver) เป็นตัวรับแสงหลังจากสะท้อนกลับจากวัตถุ

2.7.6 การสร้างแผนที่และระบุตำแหน่งไปพร้อมกัน

หนึ่งในวิธีการที่จะได้มาซึ่งข้อมูลของสภาพแวดล้อมเพื่อให้หุ่นยนต์สามารถเคลื่อนที่ได้โดยไม่ชนกับสภาพแวดล้อมโดยรอบ สามารถอธิบายด้วยจุดสังเกตของสิ่งกีดขวาง (Landmark) เป็นหนึ่งในระบบที่สำคัญต่อการพัฒนาระบบนำทางอัตโนมัติสำหรับหุ่นยนต์ร่วมกับระบบ ROS ในการให้ข้อมูลในรูปแบบการครอบครองพื้นที่ว่าง (Occupancy grid map) สำหรับการสร้างแผนที่บนระนาบ 2 มิติ ซึ่งการประมาณคำตอบของ SLAM สามารถพิจารณาการเคลื่อนที่ของหุ่นยนต์กำลังเคลื่อนที่ผ่านสภาพแวดล้อม โดยเกิดการสังเกตแบบสัมพัทธ์ของจุดสังเกตโดยใช้เซนเซอร์ ดังรูปที่ 2.18



รูปที่ 2.18 ระเบียบการประมาณตอบของ SLAM
ที่มาภาพ: (Durrant-Whyte and Bailey., 2006)

กำหนดให้รูปสามเหลี่ยมสีขาวแทนตำแหน่งจริง (Ground Truth) และ Estimated เป็นตำแหน่งจากการประมาณคำตอบจากอัลกอริทึม ณ เวลา k ใด ๆ โดย u_k เป็นเวกเตอร์ควบคุมหุ่นยนต์ ณ เวลา $k-1$ เพื่อประมาณคำตอบเวกเตอร์สถานะ x_k ณ เวลา k พร้อมกับบันทึกตำแหน่งของสิ่งกีดขวางด้วยจุดสังเกต m_i โดยไม่แปรผันกับเวลาที่ขึ้นกับค่าสถานะของการวัด ณ ตำแหน่งของหุ่นยนต์ $z_{k,i}$ โดยสามารถอธิบายในรูปการแจกแจงความน่าจะเป็น ดังสมการที่ (2.28)

$$P(x_k, m \mid z_{0:k}, U_{0:k}, x_0) \tag{2.28}$$

เมื่อ $x_{0:k} = \{x_0, x_1, ..., x_k\}$ เป็นเซตของสถานะหุ่นยนต์ประกอบด้วยตำแหน่งและมุมบนระบบพิกัด ณ เวลาที่ 0 ถึง k

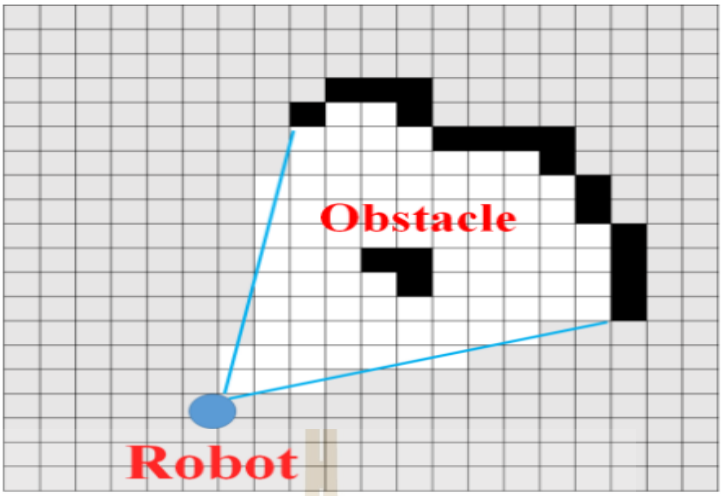
$U_{0:k} = \{u_1, u_2, ..., u_k\}$ เป็นเซตของคำสั่งควบคุมหุ่นยนต์ตั้งแต่เวลาที่ 0 เวลา k

$m_i = \{m_1, m_2, ..., m_k\}$ เป็นเซตของแผนที่โดย m_i แทนจุดสังเกตที่ใช้อธิบายแผนที่ ด้วยระยะสิ่งกีดขวาง

$z_{0:k} = \{z_1, z_2, ..., z_k\}$ เป็นเซตสถานะของค่าการวัดเพื่อแสดงตั้งแต่จุดสังเกตหลาย ๆ จุดเปรียบได้ว่าเป็นเอาต์พุตจากการประมาณ ณ เวลา 0 ถึง k

2.7.7 การแทนที่ข้อมูลของสิ่งกีดขวาง

หนึ่งในผลลัพธ์จากการสร้างแผนที่ คือ การแทนที่ข้อมูลของสภาพแวดล้อม ด้วยตารางกริดที่สามารถระบุได้ถึงสถานะความเป็นพื้นที่ว่าง (unoccupied) มีสิ่งกีดขวาง (occupied) หรือ ไม่รู้จัก (unknown) ด้วยค่าความน่าจะเป็นที่มีค่าตั้งแต่ 0 กับ 100 โดยค่า 100 หมายถึง การมีสิ่งกีดขวางอยู่ ณ กริดดังกล่าว (สีดำ) ส่วนกรณีที่กริดมีค่าเท่ากับ -1 หมายถึง พื้นที่ยังเข้าไม่ถึง (สีเทา) และ 0 คือ พื้นที่ว่างไม่มีสิ่งกีดขวาง (สีขาว) โดยข้อมูลจะถูกเก็บในรูปแบบของอภิพันธุ์ (Metadata) ดังรูปที่ 2.19



รูปที่ 2.19 การแทนที่ข้อมูลแบบ Occupancy Grid Map

2.7.8 การระบุตำแหน่งบนแผนที่ด้วย AMCL

Huang et al., 2023 ได้อธิบายว่า ปัญหาที่เกิดขึ้นกับการระบุตำแหน่งเพื่อการประมาณสถานะด้วยวิธี Dead Reckoning ไม่สามารถเลี่ยงความผิดพลาดสะสมที่เกิดขึ้นจากการวัดจำเป็นต่อการประเมินข้อมูลจากสภาพแวดล้อมในการชดเชยข้อผิดพลาดอคติโดยใช้วิธีการทางสถิติด้วยอัลกอริทึม AMCL ซึ่งเป็นวิธีระบุตำแหน่งบนแผนที่ที่ได้รับความนิยมมากที่สุดในปัจจุบัน ด้วยกลไกของตัวกรองอนุภาคและการแปลแบบ Monte Carlo เพื่อจัดการกับปัญหาได้อย่างเหมาะสม

อัลกอริทึม AMCL จะใช้ตัวกรองอนุภาค (Particle filter) เพื่อสร้างอนุภาคสมมุติจำนวนหนึ่งในการแสดงการกระจายตัวของความน่าจะเป็น ในกรณีที่ไม่มีข้อมูลตำแหน่งใด ๆ กล่าวคือหุ่นยนต์สามารถอยู่ในตำแหน่งใดก็ได้ จากนั้นเมื่อหุ่นยนต์เคลื่อนที่ ตำแหน่งความน่าจะเป็นของสถานะสามารถอัพเดตได้จากการเปรียบเทียบกับข้อมูลเซนเซอร์ร่วมกับข้อมูลแผนที่ โดยกลุ่มของอนุภาคสามารถตั้งสมมติฐานด้วยพารามิเตอร์ 2 ส่วน ดังสมการที่ (2.29)

$$\mathrm{X} = \{x^{[j]}, w^{[j]}\}_{j=1,2,\ldots,J} \tag{2.29}$$

เมื่อ $x^{[j]}$ คือ สมมติฐานของสถานะโดยแทนตำแหน่งของอันดับของอนุภาคด้วย j และ $w^{[j]}$ คือ ค่าความสำคัญของน้ำหนักในการตัดสินใจเพื่อแสดงความน่าจะเป็นเมื่อสถานะของอนุภาค j เป็นจริง หลังจากที่ทำให้น้ำหนักของอนุภาคทั้งหมดเป็นมาตรฐานแล้ว ความน่าจะเป็นของอนุภาคที่แยกจากกันสามารถประมาณได้กับฟังก์ชันต่อเนื่องโดยวิธีการประมาณค่าความหนาแน่นด้วยวิธีของเคอร์เนล ดังสมการที่ (2.30)

$$p(X) = \sum_{j=1}^{J} w^{[j]} \delta_{x^{[j]}}(X) \quad (2.30)$$

เมื่อ $\delta_{x^{[j]}}(X)$ คือ การแจกแจงปกติแบบเกาส์เซียนของค่าเฉลี่ย X สำหรับอนุภาค j ด้วยการกำหนดน้ำหนักอนุภาคเดิมใหม่และสุ่มตัวอย่างการกระจายอนุภาคใหม่ตามค่าน้ำหนักใหม่ ดังนั้นตัวกรองอนุภาคจึงเป็นฟังก์ชันเวียนบังเกิด เรียกว่า ตัวกรองแบบเบย์ (Bayesian Filter) สามารถนำไปประยุกต์ในการระบุตำแหน่งแบบ มอนติคาร์โล (Monte Carlo Localization) โดยมีการทำงานของอัลกอริทึม ดังนี้

1) ตั้งสมมติฐานของตัวแปรสถานะ เช่น พิกัดตำแหน่ง และมุมองศาปรับตัว
2) กำหนดค่าน้ำหนักหลักเกณฑ์เพื่อการตัดสินใจ (Importance weight)
3) แจกแจงอนุภาคของหุ่นยนต์ในแผนที่ และโดยได้มาจากการรับรู้หรือการแจกแจงครั้งก่อน
4) แจกแจงอนุภาคใหม่ที่ได้รับหลังจากนั้นอัพเดตค่าน้ำหนัก

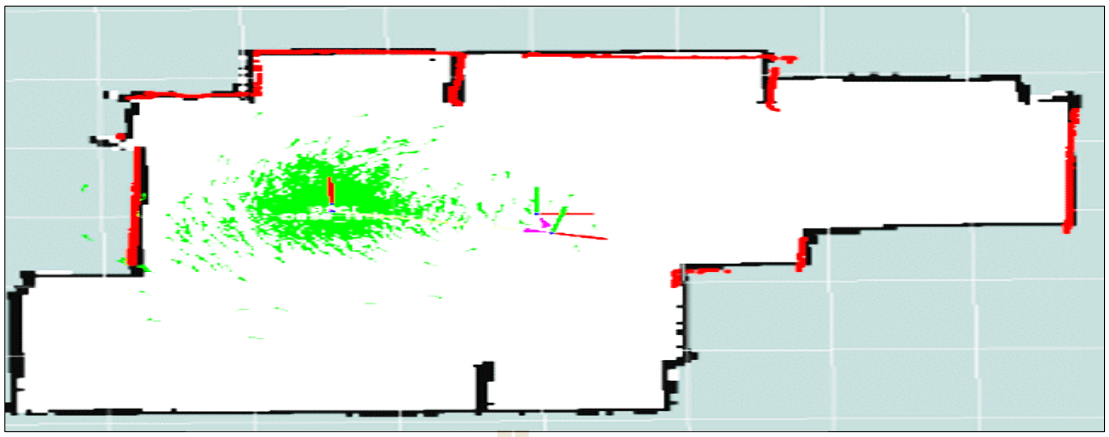
ในการระบุตำแหน่งอย่างต่อเนื่อง เช่น การกรองอนุภาคแบบวนซ้ำ การกระจายอนุภาคจะถูกนำมาใช้เป็นข้อมูลอ้างอิงสำหรับการควบคุมการเคลื่อนไหวของตำแหน่งอนุภาคสามารถกำหนดได้โดยสถานะก่อนหน้า x_{t-1} และคำสั่งการควบคุม u_t ดังสมการที่ (2.31)

$$x_t^{[j]} \sim P(x_t \mid x_{t-1,u_t}) \quad (2.31)$$

น้ำหนักที่ใช้ในการแก้ไขการกระจายตัวของอนุภาคจะใช้ข้อมูลการวัดจากเซนเซอร์ LiDAR ประกอบด้วยจุดสังเกตโดยรอบ z_t ดังสมการที่ (2.32)

$$w_t^{[j]} \propto P(z_t \mid x_{t,m}) \quad (2.32)$$

การประมาณด้วย AMCL สามารถแบ่งออกเป็นการเริ่มต้น การสังเกต การวัดการอัพเดตน้ำหนัก และการสุ่มตัวอย่างใหม่ การเริ่มต้นจะขึ้นอยู่กับวิธีการที่ระบุเพื่อสร้างการกระจายตัวของอนุภาคเริ่มต้น ส่วนขั้นตอนการสังเกต คือ การรวบรวมข้อมูลสิ่งแวดล้อมในแหล่งกำเนิดหรือหลังการเคลื่อนย้าย โดยความแตกต่างระหว่างข้อมูลสิ่งแวดล้อมและข้อมูลแผนที่จะถูกคำนวณอยู่ในขั้นตอนการวัดโดยที่น้ำหนักของอนุภาคจะได้รับการอัพเดตตามข้อมูล ซึ่งในขั้นตอนสุดท้ายอัลกอริทึมจะทำการการสุ่มตัวอย่างใหม่ทำให้การกระจายอนุภาคใหม่จะถูกสร้างขึ้นตามการกระจายน้ำหนักที่อัพเดต โดยการกระจายอนุภาพสามารถแสดงได้ในรูปที่ (2.20)



รูปที่ 2.20 การกระจายของอนุภาคสมมติของตัวกรองอนุภาค
(Boshlyakov et al., 2020)

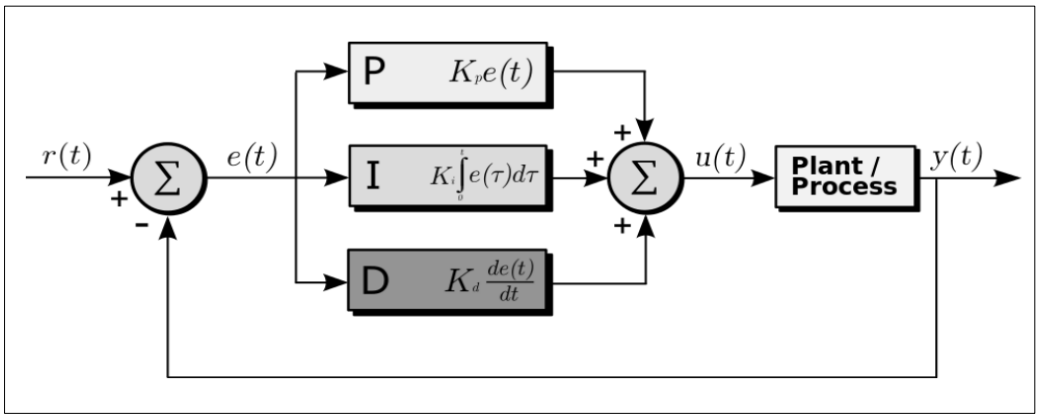
2.8 ระบบควบคุมหุ่นยนต์อัตโนมัติ

Seng Chia., 2018 ได้ศึกษาการออกแบบระบบควบคุมป้อนกลับสำหรับหุ่นยนต์ที่มีการเคลื่อนที่แบบ Skid steering โดยใช้กล้องตรวจจับเส้นสีดำ ด้วยวิธีการวัฏจักรสุดท้าย (Ultimate-cycle tuning method) ตามกฎของซีกเลอร์-นิโคลส์ (Ziegler-Nichols) ในการติดตามเส้นทางด้วยความเร็ว 25, 50 และ 75 เมตรต่อวินาที พบว่า ระบบควบคุมพี พีไอ และ พีไอดี ที่ถูกออกแบบจากการเปลี่ยนแปลงความเร็ว 3 ค่า ให้ค่าตัวควบคุมที่เหมาะสมแตกต่างกัน โดยการออกแบบที่ความเร็วสูงสุด ตัวควบคุมจะมีค่าสูงขึ้นแปรผันตรงกับความเร็วที่ใช้ในการหาค่าตัวควบคุมตามกระบวนการวัฏจักรสุดท้าย

2.8.1 ระบบควบคุมป้อนกลับ พีไอดี

ระบบควบคุม PID กล่าวได้ว่าเป็นหนึ่งในระบบควบคุมที่พบได้บ่อยในงานอุตสาหกรรมทั่วไปด้วยคุณสมบัติที่สามารถใช้งานได้ง่ายและมีหลักการที่ไม่ซับซ้อน ทำให้ผู้ใช้งานสามารถปรับเลือกค่าอัตราขยายได้ด้วยตนเองแบบลองผิดลองถูก หรือสามารถใช้กฎการปรับจูนต่าง ๆ โดยมีวัตถุประสงค์เพื่อลดค่าความผิดพลาดของสัญญาณป้อนกลับให้มีค่าน้อยที่สุดหรือเป็นศูนย์ ประกอบด้วยพารามิเตอร์ 3 ส่วนที่เกี่ยวข้อง คือ ตัวควบคุมสัดส่วน K_p ตัวควบคุมปริพันธ์ K_i และตัวควบคุมอนุพันธ์ K_d โดยเอาต์พุตของตัวควบคุมสามารถคำนวณอยู่ในรูปโดเมนเวลา ดังสมการที่ (2.33) โดยความสัมพันธ์สามารถแสดงด้วยรูปเผยภาพบล็อก (Block diagram) ดังรูปที่ 2.21

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \tag{2.33}$$



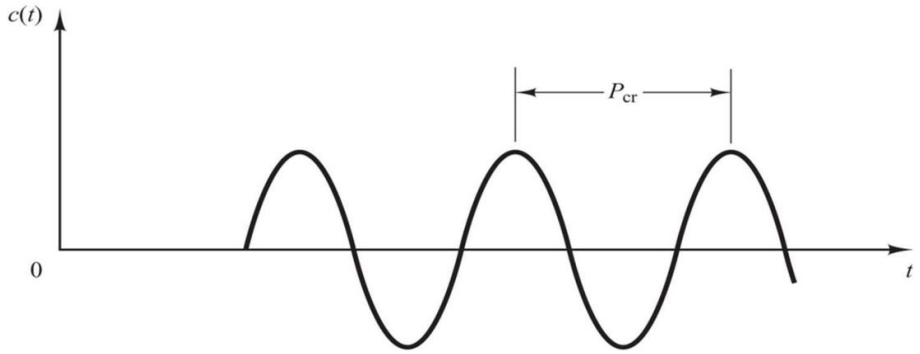
รูปที่ 2.21 แผนภาพบล็อกของระบบควบคุมป้อนกลับแบบ PID

2.8.2 การปรับตัวควบคุมแบบ พีไอดี ด้วยวิธีการวัฏจักรสุดท้าย

เนื่องจากความเป็นจริงนั้นการหาค่าที่เหมาะสมกับระบบควบคุมจำเป็นต้องทราบฟังก์ชันถ่ายโอน (Transfer Function) แต่ในทางปฏิบัติการหาแบบจำลองทางคณิตศาสตร์ของระบบนั้นไม่ใช่เรื่องง่ายรวมถึงมีค่าใช้จ่ายที่สูง ทำให้ทฤษฎีของซีเกลอร์-นิโคลส์ (Ziegler-Nichols) ถูกนำมาใช้เพื่อการหาตัวควบคุมของระบบจากการป้อนผลการตอบสนองของสัญญาณอินพุตเป็นฟังก์ชันขั้นหนึ่งหน่วย (Unit-Step Input) ร่วมกับการสังเกตผลตอบสนองที่ต้องเกิดการแกว่งลดลงด้วยอัตราหนึ่งในสี่ของโอเวอร์ชูตที่สองน้อยกว่า 25% ของค่าโอเวอร์ชูตตัวแรก สำหรับวิธีการวัฏจักรสุดท้ายสามารถดำเนินตามขั้นตอน ดังนี้

1) ปรับค่าอัตราขยายอนุพันธ์และปริพันธ์ให้มีค่าน้อยที่สุดเพื่อให้ระบบเป็นระบบควบคุมแบบสัดส่วนเพียงอย่างเดียว

2) ค่อย ๆ เพิ่มค่าอัตราขยายตัวควบคุมแบบสัดส่วนและสังเกตสัญญาณตอบสนองจากระบบจนกระทั่งระบบเกิดเสถียรภาพแบบขอบ ดังรูปที่ 2.22



รูปที่ 2.22 การเกิดเสถียรภาพแบบขอบ

ที่มาภาพ : (https://faculty.mercer.edu/jenkins_he/documents/TuningforPIDControllers.pdf)

3) บันทึกค่าอัตราขยายตัวควบคุมแบบสัดส่วน K_{cr} ที่ทำให้ระบบเกิดสเถียรภาพแบบขอบและหาค่าคาบเวลา P_{cr} เมื่อระบบเกิดเสถียรภาพแบบขอบ

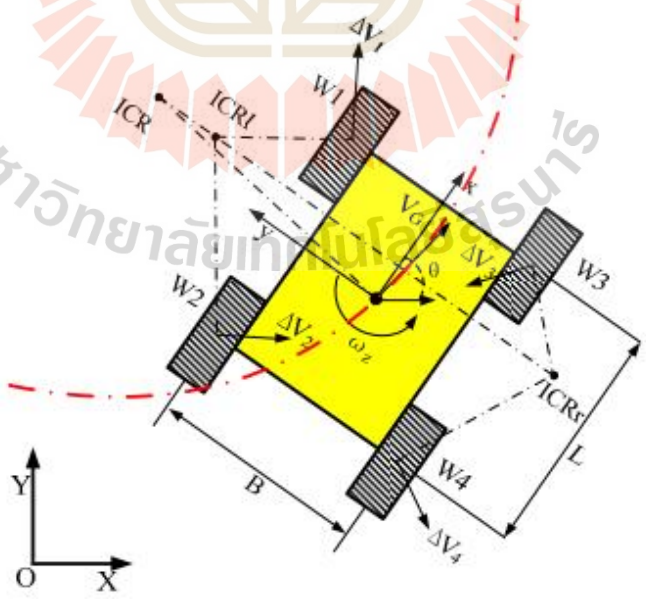
4) คำนวณพารามิเตอร์ของระบบควบคุมจาก P_{cr} และ K_{cr} ดังตารางที่ 2.2

ตารางที่ 2.2 การคำนวณตัวควบคุมด้วยวิธีด้วยวิธีการวัฏจักรสุดท้าย

ชนิดของตัวควบคุม	พารามิเตอร์ควบคุม		
	K_p	K_i	K_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1.2K_p}{P_{cr}}$	0
PID	$0.6K_{cr}$	$\frac{0.6K_p}{P_{cr}}$	$\frac{K_{cr}P_{cr}}{8}$

2.8.3 แบบจำลองจลนศาสตร์

การเคลื่อนที่แบบ Skid steering เป็นระบบขับเคลื่อนที่คล้ายกันกับ Differential Drive แตกต่างกันด้วยจำนวนล้อในแต่ละด้านของหุ่นยนต์ที่มีจำนวนมากกว่า 1 ล้อขึ้นไป ทำให้หน้าสัมผัสระหว่างล้อมากขึ้นจนเกิดแรงเสียดทานและการลื่นไถล (Slip) และจุดศูนย์กลางการหมุน (ICR) ดังรูปที่ 2.23



รูปที่ 2.23 แบบจำลองทางคณิตศาสตร์การขับเคลื่อนแบบ Skid Steering

ที่มาภาพ: (Wang et al., 2015)

พิจารณารูปที่ 2.23 เมื่อหุ่นยนต์ถูกบังคับแบบหมุนด้วยความเร็วเชิงมุมจะเกิด ICR อยู่ตรงกลางของหุ่นยนต์ด้วยพิกัด (x_G, y_G) และในกรณีที่หุ่นยนต์ถูกบังคับเลี้ยวซ้ายหรือขวาจะเกิดจุดหมุน ICR_L ที่พิกัด (x_l, y_l) หรือ จุดหมุน ICR_R ที่พิกัด (x_r, y_r) โดยความสัมพันธ์ พิกัด y ของจุดหมุนกับความเร็วเชิงเส้น v_x และความเร็วเชิงมุม ω_z สามารถอธิบาย ดังสมการที่ (2.34) – (2.36)

$$y_l = \frac{v_x - \omega_l r}{\omega_z} \tag{2.34}$$

$$y_r = \frac{v_x - \omega_r r}{\omega_z} \tag{2.35}$$

$$x_G = x_l = x_r = y_g = \frac{v_x}{\omega_z} \tag{2.36}$$

โดยที่ J_ω เป็นเมทริกซ์ที่มีความสัมพันธ์กับพิกัดของ ICR ดังสมการที่ (2.37)

$$J_\omega = \frac{1}{y_l - y_r}\begin{bmatrix} -y_r & y_l \\ x_G & -x_G \\ -1 & 1 \end{bmatrix} \tag{2.37}$$

กำหนดให้หุ่นยนต์มีความสมมาตรเพื่อให้จุดหมุนอยู่ที่จุดศูนย์กลางของหุ่นยนต์ $x_G = 0$ และ $y_0 = y_l = -y_r$ แทนค่าลงในสมการที่ (2.37) ได้ผลลัพธ์ดังสมการที่ (2.38)

$$J_\omega = \frac{1}{2y_0}\begin{bmatrix} -y_0 & y_0 \\ 0 & 0 \\ -1 & 1 \end{bmatrix} \tag{2.38}$$

เมื่อ $y_0 = y_l = -y_r$ เป็นจุดหมุน ICR หุ่นยนต์จะเคลื่อนที่สัมพันธ์กับความเร็วเชิงเส้น v_x และความเร็วเชิงมุม ω_z ดังสมการที่ (2.39) ถึง (2.40)

$$v_x = \frac{\omega_l r + \omega_r r}{2} = \frac{v_l + v_r}{2} \tag{2.39}$$

$$\omega_z = \frac{-\omega_l r + \omega_r r}{2y_0} = \frac{-v_l + v_r}{2y_0} \tag{2.40}$$

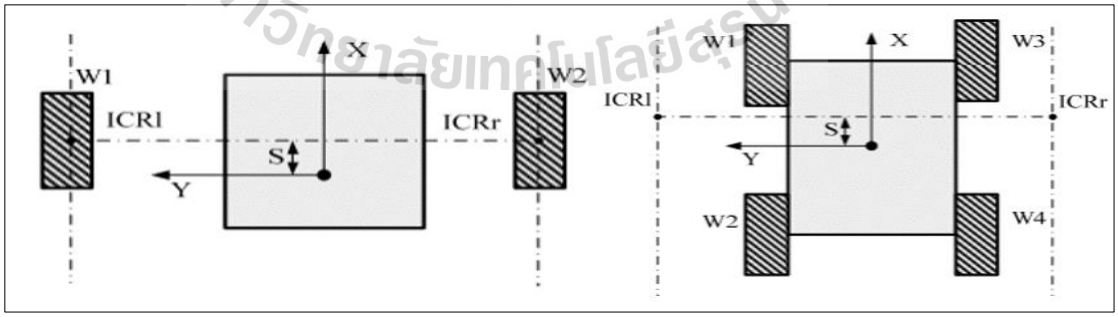
จากสมการที่ (2.36) ความสัมพันธ์ระหว่างความเร็วเชิงเส้น $v_x = v_G$ และความเร็วเชิงมุม ω_z สามารถคำนวณรัศมีความโค้งของเส้นทางเลี้ยวดังสมการที่ (2.41)

$$R = \frac{v_G}{\omega_z} = \frac{v_l + v_r}{-v_l + v_r} y_0 \tag{2.41}$$

ความสัมพันธ์ระหว่างพิกัดจุดหมุน y_0 กับระยะห่างล้อ B สามารถอธิบายด้วยค่าสัมประสิทธิ์ ICR ดังสมการที่ (2.42)

$$\chi = \frac{y_l - y_r}{B} = \frac{2y_0}{B} \tag{2.42}$$

เมื่อพิจารณาความสัมพันธ์ระหว่างค่าสัมประสิทธิ์ ICR χ โดยเมื่อ χ เท่ากับ 1 หมายความว่าหุ่นยนต์ไม่เกิดการลื่นไถลในการบังคับเลี้ยวหรือหมุนด้วยความเร็วเชิงมุม โดยมีแบบจำลองทางคณิตศาสตร์เช่นเดียวกับการเคลื่อนที่แบบ Differential Drive ด้วยพิกัดของจุดหมุน ICR ที่อยู่ ณ จุดกึ่งกลางของล้อ ดังรูปที่ 2.23 แตกต่างจากกรณีที่ χ มีค่ามากกว่า 1 หมายความว่าหุ่นยนต์เกิดการลื่นไถลในการบังคับเลี้ยวหรือหมุนด้วยความเร็วเชิงมุม ทำให้พิกัดของจุดหมุน ICR อยู่ด้านนอกของล้อสังเกตด้วยพิกัด ICR_L และ ICR_R อธิบายได้ ดังรูปที่ 2.24



รูปที่ 2.24 ความแตกต่างของจุด ICR ทั้ง ระหว่าง 2 ระบบขับเคลื่อน
ที่มาภาพ: (Wang et al., 2015)

2.9 ระบบนำทางอัตโนมัติของหุ่นยนต์

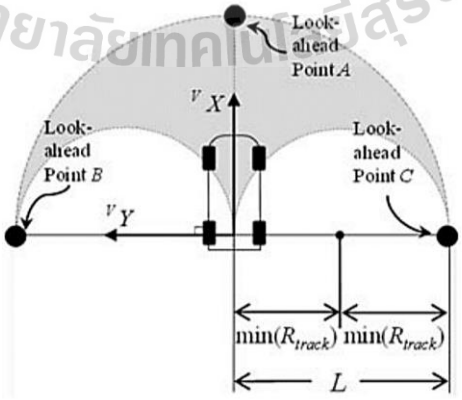
การนำทางอัตโนมัติคือการเคลื่อนที่ของหุ่นยนต์จากจุดเริ่มต้น ไปยังเป้าหมายที่กำหนด โดยไม่ให้ชนกับสิ่งกีดขวาง จากการวางแผนการเคลื่อนไหว (Motion Planning) เพื่อค้นหาเส้นทางที่เป็นไปได้ และใช้อัลกอริทึมควบคุมเพื่อติดตามเส้นทางที่วางแผนไว้ (Vishwas et al., 2021)

Kim et al., 2013 ได้กล่าวว่า ประสิทธิภาพของระบบติดตามเส้นแบบ (Pure Pursuit, PP) เป็นระบบติดตามเส้นทางแบบเรขาคณิตขึ้นอยู่กับระยะมองไปข้างหน้า (Lookahead - Distance) เปรียบเสมือนกับระยะห่างระหว่างยานพาหนะกับจุดอ้างอิงหน้ายานพาหนะที่คนขับอาจมองเพื่อติดตามเส้นทาง ดังนั้น ยานพาหนะควรมีระยะการมองไปข้างหน้าที่แปรผันตามความเร็ว ยิ่งเคลื่อนที่เร็วเท่าใด ระยะมองไปข้างหน้าก็ควรมีค่ามากขึ้น แต่สำหรับระบบติดตามเส้นทางที่แม่นยำต้องมีระยะมองไปข้างหน้าที่สั้น อย่างไรก็ตาม หากยานพาหนะมีความเร็วสูงและมองไปข้างหน้าในระยะสั้น ยานพาหนะก็จะมีความเร็วเชิงมุมสูงในขณะที่กำลังเดินไปตามเส้นทาง ซึ่งอาจทำให้ยานพาหนะเกิดการลื่นไถลออกนอกเส้นทางได้ ซึ่งความสัมพันธ์ในการกำหนดระยะมองไปข้างหน้าที่เป็นไปได้สำหรับการติดตามเส้นทางอัตโนมัติ สามารถอธิบายได้ ดังสมการที่ (2.43)

$$min(R_{track}) = \frac{v}{max(\omega)} \tag{2.43}$$

$$L = 2min(R_{track}) \tag{2.44}$$

เมื่อ $min(R_{track})$ เป็นรัศมีการเข้าโค้งขั้นต่ำของยานพาหนะที่ใช้ความเร็วเชิงเส้น v ในการเคลื่อนที่ และสามารถทำความเร็วเชิงมุมสูงสุด $max(\omega)$ โดยมีความสัมพันธ์กับรัศมีโค้งขั้นต่ำ ดังรูปที่ 2.25

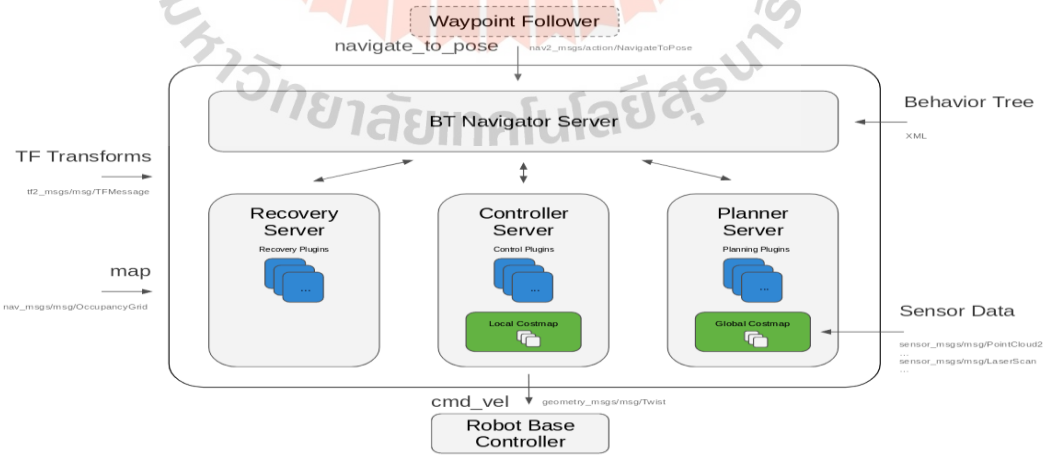


รูปที่ 2.25 ระยะมองไปข้างหน้าที่สัมพันธ์กับรัศมีโค้งของเส้นทาง
ที่มาภาพ: (Kim et al., 2013)

Macenski et al., 2021 ได้เห็นถึงปัญหาการประยุกต์ใช้งานอัลกอริทึมแบบ PP สำหรับการใช้งานในพื้นที่จำกัด เนื่องจากอัลกอริทึมแบบ PP ใช้หลักการในการตรวจจับทางโค้งด้วยการคำนวณทางเรขาคณิตอย่างง่ายโดยยานพาหนะจะเคลื่อนที่ด้วยความเร็วคงที่ตลอดการใช้งาน และไม่มีข้อกำหนดด้านเสถียรภาพใด ๆ ในการเลือกความเร็วที่เหมาะสม ซึ่งการเคลื่อนที่ด้วยความเร็วคงที่นั้นไม่สามารถเคลื่อนที่ผ่านเส้นทางเข้าโค้งที่มีรัศมีต่ำได้อย่างปลอดภัย ทำให้การควบคุมแบบ PP ไม่ได้รับการพัฒนาโดยคำนึงถึงหุ่นยนต์ในอุตสาหกรรม เนื่องจากมีข้อกำหนดด้านความปลอดภัยอื่น ๆ จากปัญหาดังกล่าวจึงได้จัดทีมในการพัฒนาอัลกอริทึมที่สามารถควบคุมความเร็วในการเคลื่อนที่ผ่านทางโค้งเรียกว่าการควบคุมแบบ Regulated Pure Pursuit (RPP) เพื่อปรับปรุงประสิทธิภาพการนำทางผ่านทางโค้งรัศมีต่ำ ด้วยฟังก์ชัน Curvature Heuristic โดยผลการทดสอบการใช้งานจริงเมื่อเปรียบเทียบการนำทางด้วยการเข้าโค้งหักศอก ระหว่างอัลกอริทึมแบบ RPP, APP และ PP พบว่าค่าผิดพลาดออกนอกเส้นทางของอัลกอริทึม RPP มีค่าเฉลี่ยน้อยที่สุดเท่ากับ 0.052 เมตร เมื่อเทียบกับ APP เท่ากับ 0.059 และ PP เท่ากับ 0.1 เมตร

2.9.1 ระบบนำทางอัตโนมัติ

ระบบ Navigation Stack เป็นชุดเครื่องมือ (Framework) ที่ช่วยให้นักพัฒนาสามารถสร้างหุ่นยนต์อัตโนมัติจากการรวมเครื่องมือและอัลกอริทึมที่แตกต่างกันเข้าด้วยกัน รวมถึงเป็นชุดเครื่องมือที่ใช้กันอย่างแพร่หลายทั้งในด้านการวิจัยและอุตสาหกรรม ของ ROS Navigation (Quigley et al., 2009) และต่อมาได้รับการเสนอให้เป็นการปรับปรุง ROS Navigation และสร้างขึ้นบนระบบ ROS2 โดยมีชื่อว่า Navigation2 (NAV2) โดยสถาปัตยกรรมของระบบสามารถอธิบายการทำงานได้ดังรูปที่ 2.26



รูปที่ 2.26 สถาปัตยกรรม NAV2

ที่มาภาพ: (Macenski et al., 2020)

จากภาพรวมของสถาปัตยกรรม ระบบนำทางอัตโนมัติสามารถสร้างเส้นทางแบบจุดอ้างอิง (Waypoint) และควบคุมหุ่นยนต์จากการทำงานร่วมกันหลายส่วน โดยรับอินพุตจากข้อมูล Odometry ในการประมาณตำแหน่งของหุ่นยนต์จากข้อมูลเซนเซอร์ที่อยู่ในรูปแบบ TF บนระบบพิกัด ซึ่งข้อมูลอินพุตเหล่านี้จะเป็นส่วนหนึ่งในการดำเนินการของเซิร์ฟเวอร์ที่สำคัญ 3 ส่วน ดังนี้

Recoveries server เป็นส่วนคำสั่งกู้คืนการทำงานของหุ่นยนต์ กล่าวคือ เมื่อระบบไม่สามารถหาเส้นทางที่เป็นไปได้สามารถกำหนดให้ระบบออกคำสั่งกระทำ (Actions) ประกอบด้วย รอ (Wait) หมุน (Spin) หรือ ถอยหลัง (Back Up) ซึ่งมีโครงสร้างแบบแผนภาพต้นไม้ (Behavior Trees)

Controller Server เป็นส่วนการจัดการคำขอของการควบคุมด้วยเส้นทางอ้างอิงในการตรวจสอบเป้าหมายและออกคำสั่งควบคุมด้วย Topics ชื่อว่า cmd_vel จากระบบควบคุม DWB, TEB Local planner, MPC หรือ Regulated Pure pursuit

Planner Server เป็นส่วนจัดการคำขอการสร้างเส้นทางโดยผู้ใช้ เป็นการหาเส้นทางที่เป็นไปได้และสั้นที่สุดด้วยข้อมูล Global Costmap ที่ถูกสร้างจากข้อมูลแผนที่

2.9.2 การสร้าง Costmap จากสิ่งกีดขวาง

หุ่นยนต์จะสามารถเคลื่อนที่ในสภาพแวดล้อมได้อย่างปลอดภัยจะขึ้นอยู่กับเส้นทางในการควบคุม โดยการวางแผนเส้นทางด้วยระยะที่ปลอดภัยจากสิ่งกีดขวางอาศัยหลักการทำงานของ Costmap 2D ในการกำหนดระยะด้วยค่า cost แสดงถึงระยะที่เป็นสิ่งกีดขวางแผ่ออกมาเป็นระยะที่ปลอดภัย มีค่าตั้งแต่ 0 ถึง 254 ตามลำดับ ซึ่งแรแผ่ของค่า cost สามารถคำนวณการแสดงผลของระยะจากสิ่งกีดขวาง ดังสมการที่ (2.45)

$$c = exp(-\alpha \cdot (d_{obs} - r_{ins})) \cdot V_{theal} \tag{2.45}$$

เมื่อ c คือ ค่า cost ของสิ่งกีดขวางในหนึ่งกริดเซลล์

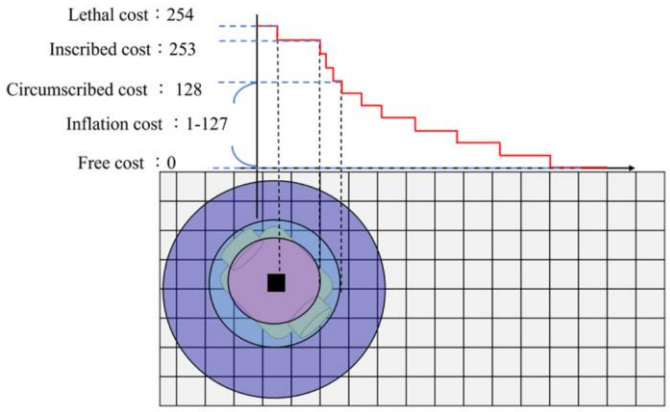
α คือ ขนาดของอัตราการสลายตัว

d_{obs} คือ ระยะจากสิ่งกีดขวาง

r_{ins} คือ รัศมีของหุ่นยนต์

V_{theal} คือ lethal cost แสดงระยะอันตรายถึงระยะที่ปลอดภัย เท่ากับ 254

จากสมการที่เกี่ยวข้องสามารถแสดงผลการคำนวณบนระบบพิกัดในรูปแบบ Occupancy Grid Map ดังรูปที่ 2.27

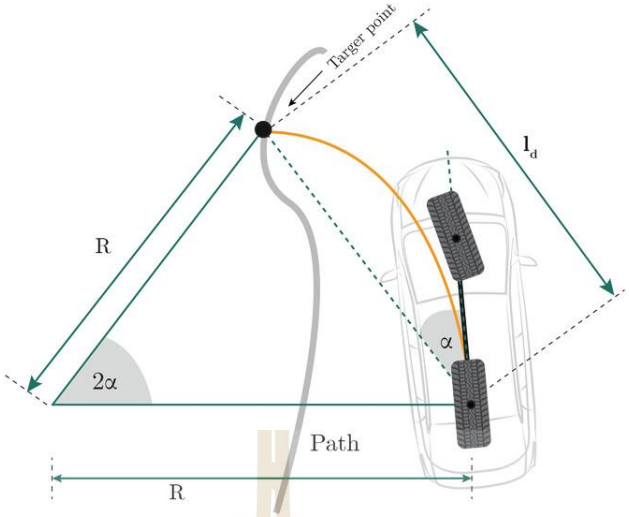


รูปที่ 2.27 การแสดงผลระยะจากสิ่งกีดขวางด้วย Costmap
ที่มาภาพ: (Chen et al., 2015)

จากรูปที่ 2.27 ค่า cost ที่เป็นสิ่งกีดขวางจะมีค่าเท่ากับ 254 เรียกว่าชั้น lethal เป็นชั้นที่หากขอบเขตหุ่นยนต์เข้าไปอยู่ในชั้นดังกล่าวหุ่นยนต์จะรับรู้ว่ากำลังชนกับสิ่งกีดขวาง ในขณะที่ Inscribed cost เป็นชั้นที่ห่างจากตำแหน่งของสิ่งกีดขวางแผ่ออกมาเท่ากับขอบเขตของหุ่นยนต์ ซึ่งในชั้นนี้หุ่นยนต์จะยังสามารถเคลื่อนที่ต่อไปได้หากขอบเขตยังไม่เข้าไปสู่ชั้นของ Lethal

2.9.3 การติดตามเส้นทางแบบ Pure Pursuit

ระบบติดตามเส้นทางแบบ Pure Pursuit กล่าวได้ว่า เป็นระบบควบคุมที่มีหลักการทำงานที่ง่ายที่สุด นิยมประยุกต์ใช้กับยานยนต์ที่มีระบบบังคับเลี้ยวแบบพวงมาลัย โดยมีพารามิเตอร์ที่สำคัญเพียงค่าเดียว คือ ระยะมองไปข้างหน้า Look-ahead Distance (l_d) เป็นระยะที่นำไปสู่การควบคุมด้วยกฎของไซน์เพื่อคำนวณความสัมพันธ์ทางเรขาคณิตโดยให้เอาต์พุตในการควบคุมของอัลกอริทึม คือ องศาบังคับเลี้ยว (Steering Angle) ดังรูปที่ 2.28



รูปที่ 2.28 หลักการควบคุมอัลกอริทึมแบบ Pure-pursuit
ที่มาภาพ: (Rokonuzzaman et al., 2021)

พิจารณายานพาหนะกำลังเคลื่อนที่ผ่านเส้นทาง (Path) ที่ถูกบังคับด้วยคำสั่งองศาเลี้ยวให้ยานพาหนะเข้าสู่จุด Target point (TP) โดยมีความเร็วเชิงเส้นคงที่สัมพันธ์กับระยะมองไปข้างหน้า l_d ดังสมการที่ (2.46)

$$\delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{l_d}\right) \tag{2.46}$$

เมื่อพิจารณาพารามิเตอร์ในการติดตามเส้นทางที่สามารถปรับเพียงพารามิเตอร์เดียวทำให้การติดตามเส้นทางด้วยวิธีนี้ไม่จำเป็นต้องกำหนดพารามิเตอร์ในการทำนายการเคลื่อนที่ของยานพาหนะเช่นเดียวกับการติดตามเส้นทางแบบ Model Predictive Control (MPC) ที่ต้องกำหนดแบบจำลองจลนศาสตร์ในการทำนายการเคลื่อนที่ หรือ Dynamic Window Approach (DWA) ที่ต้องกำหนดพารามิเตอร์ในการจำลองตัวอย่างการเคลื่อนที่ ทำให้อัลกอริทึมแบบ Pure pursuit ใช้ทรัพยากรในการประมวลผลที่ต่ำแต่มีประสิทธิภาพในการใช้งานจริงเมื่อเทียบกับการควบคุมแบบ ดังกล่าว (Rokonuzzaman et al., 2021) จากการทดสอบด้วยการเคลื่อนที่ผ่านเส้นทางโค้งเปรียบเทียบค่าความผิดพลาดในการควบคุมตามแนวด้านข้าง (Lateral) และมุมองศาปรับตัวในช่วงเวลาหนึ่ง ที่ความเร็ว 20 ถึง 60 กิโลเมตรต่อชั่วโมง พบว่า ระบบควบคุมแบบ Pure pursuit มีความเหมาะสมต่อการนำมาใช้งานจริงที่ความเร็วต่ำ

2.9.4 การติดตามเส้นทางแบบ Regulated Pure Pursuit

เนื่องจากการติดตามเส้นทางแบบ PP ที่ไม่สามารถควบคุมความเร็วเชิงเส้นในขณะเคลื่อนที่ผ่านเส้นทางโค้ง ทำให้การติดตามเส้นทางแบบ PP ถูกพัฒนาใหม่ด้วยการเพิ่มฟังก์ชัน Curvature Heuristic เป็นฟังก์ชันที่สามารถลดความเร็วเชิงเส้นของยานพาหนะด้วยเงื่อนไขจากการกำหนดพารามิเตอร์ค่าความโค้งของเส้นทาง เรียกนะบบติดตามเส้นทางนี้ว่า Regulated Pure Pursuit (RPP) โดยมีความสัมพันธ์ในการควบคุมความเร็วเชิงเส้นกับรัศมีทางโค้ง ดังสมการที่ 2.47

$$v_t^{'} = \begin{cases} v_t & \kappa > T_\kappa \\ \dfrac{v_t}{r_{min}\kappa} & \kappa \le T_\kappa \end{cases} \tag{2.47}$$

เมื่อ $v_t^{'}$ คือ ความเร็วที่ผ่านการคำนวณจากฟังก์ชัน

κ คือ ค่าความโค้งมีความสัมพันธ์กับรัศมีความโค้งของเส้นทาง

T_κ คือ ค่าความโค้งที่สามารถกำหนดได้

โดยที่ค่าความโค้ง T_κ มีความสัมพันธ์กับรัศมีความโค้งดังสมการที่ 2.48

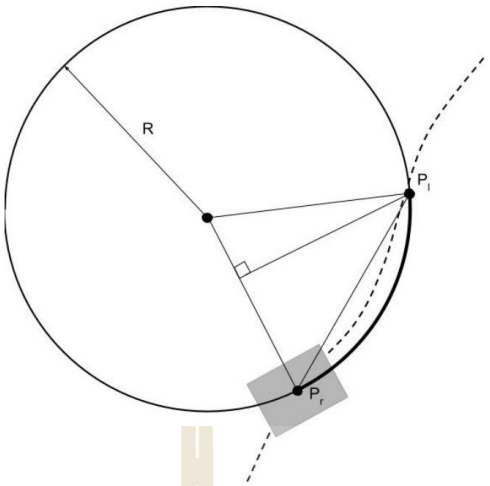
$$T_\kappa = \frac{1}{r_{min}} \tag{2.48}$$

เมื่อพิจารณาสมการที่สมการที่ 2.47 พบว่าเงื่อนไขของอัลกอริทึมสามารถแบ่งออกได้เป็น 2 กรณี เมื่อค่ารัศมีความโค้ง r_{min} เป็นอินพุตในการควบคุม ดังนี้

กรณีที่ 1 เป็นกรณีที่ยานพาหนะเคลื่อนที่ผ่านเส้นทางโค้ง κ มีค่าความโค้งมากกว่า T_κ ยานพาหนะจะถูกลดความเร็วเชิงเส้นด้วยการนำความเร็วปกติ v_t หารด้วย r_{min} คูณด้วยความโค้ง κ

กรณีที่ 2 เป็นกรณีที่ที่ยานพาหนะเคลื่อนที่ผ่านเส้นทางโค้ง κ ที่มีค่าความโค้งน้อยกว่า T_κ เป็นกรณีที่ ยานพาหนะจะถูกควบคุมด้วยความเร็วปกติจาก v_t

ระบบติดตามเส้นทาง RPP สามารถใช้จุดมองไปข้างหน้า (Look-ahead Point, P_l) ในการตรวจสอบรัศมี R ในหน่วยเมตร โดยจุด P_l ที่มีระยะห่างกับตำแหน่งของหุ่นยนต์ P_r เท่ากับระยะมองไปข้างหน้า ดังรูปที่ 2.29



รูปที่ 2.29 ความสัมพันธ์รัศมีเส้นทางโค้ง R

ระบบติดตามเส้นทางแบบ RPP นั้น การบังคับเลี้ยวจะใช้คำสั่งเป็นความเร็วเชิงมุม ω_t แทนการสั่งการด้วยองศาบังคับเลี้ยว δ โดยที่ ω_t มีความสัมพันธ์กับเส้นทางโค้ง κ และความเร็วเชิงเส้นที่ผ่านฟังก์ชัน Curvature Heuristic ดังสมการที่ 2.49

$$\omega_t = v_t' \kappa \tag{2.49}$$

เมื่อ ω_t คือ ความเร็วเชิงมุมในการควบคุม

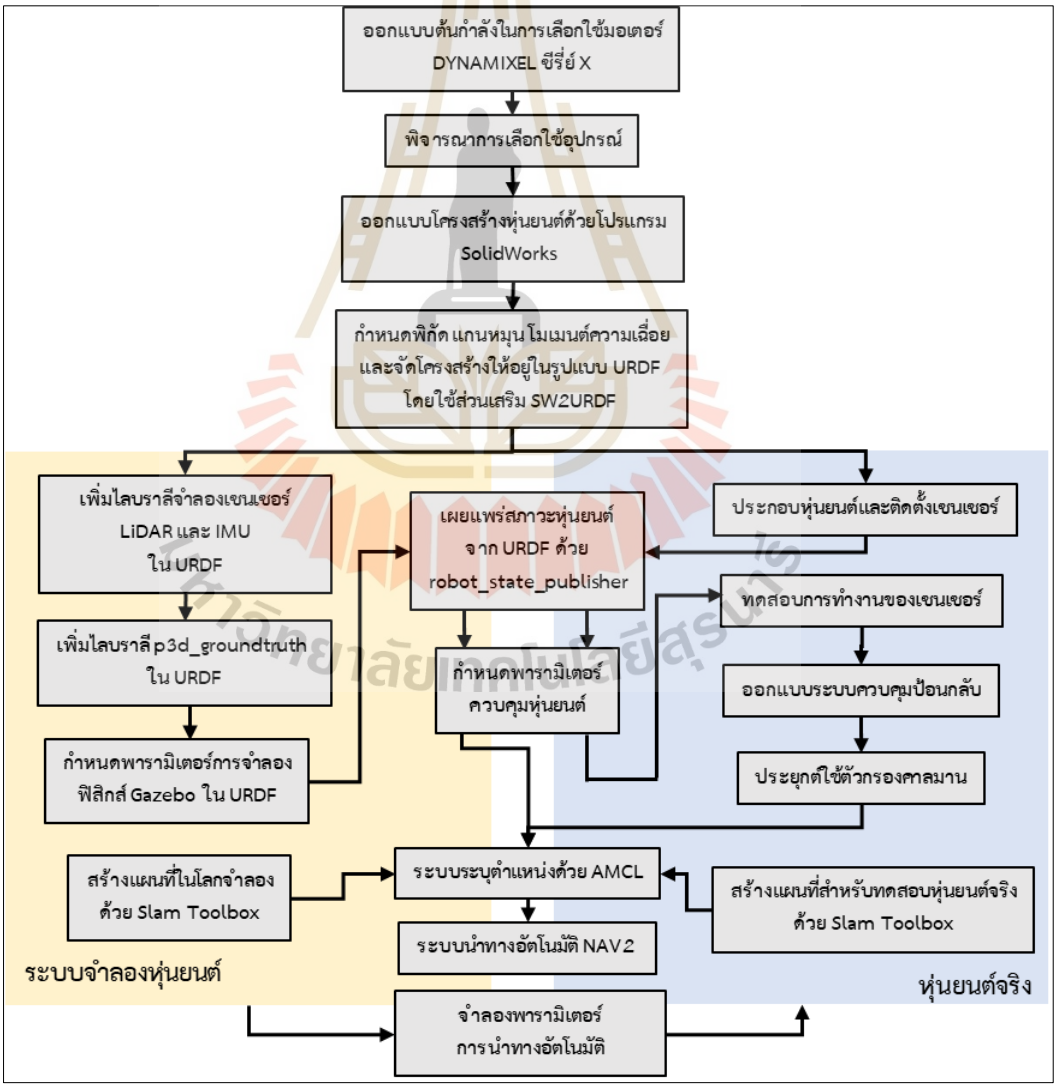
v_t' คือ ความเร็วที่ผ่านการคำนวณจากฟังก์ชัน

κ คือ ค่าความโค้งมีความสัมพันธ์กับรัศมีความโค้งของเส้นทาง

บทที่ 3
วิธีการดำเนินงานวิจัย

3.1 แบบแผนการดำเนินงานวิจัย

การดำเนินงานวิจัยนี้มุ่นเน้นในส่วนของการพัฒนาซอร์ฟแวร์ของหุ่นยนต์ให้ทำงานได้อย่างเหมาะสมต่อการใช้งานในพื้นที่แคบ โดยในบทนี้จะกล่าวถึงระเบียบวิธีและองค์ประกอบต่าง ๆ ที่จำเป็นในการดำเนินงานวิจัย สามารถอธิบาย ดังรูปที่ 3.1



รูปที่ 3.1 แบบแผนการดำเนินงานวิจัย

จากแบบแผนการดำเนินงานจะเห็นได้ว่าการสร้างหุ่นยนต์จริงจากการจำลองพฤติกรรมด้วยโปรแกรม Gazebo มีขั้นตอนบางส่วนสามารถใช้งานร่วมกันได้ทั้ง 2 ระบบ เป็นส่วนของโครงสร้าง URDF และการกำหนดพารามิเตอร์ทางจลนศาสตร์ โดยหลังจากขั้นตอนดังกล่าวงานวิจัยนี้ได้ให้ความสำคัญการทำให้หุ่นยนต์รับรู้ตำแหน่งของตัวเองได้อย่างแม่นยำซึ่งระบบจำลองไม่จำเป็นต้องให้ความสำคัญเนื่องจากปราศจากฮาร์ดแวร์ในโลกความเป็นจริงและมีไลบรารีสำเร็จรูป แต่หุ่นยนต์จริงต้องให้ความสำคัญในส่วนนี้เนื่องจากเซนเซอร์ในโลกแห่งความเป็นจริงไม่สามารถวัดค่าได้อย่างถูกต้องเนื่องจากความคลาดเคลื่อนที่เกิดจากการสอบเทียบและสัญญาณรบกวนรวมถึงการส่งข้อมูลที่ไม่ต่อเนื่องซึ่งเป็นสาเหตุของความผิดพลาดสะสม จึงจำเป็นต่อเทคนิคเพิ่มเติมในการเพิ่มประสิทธิภาพด้วยตัวกรองคาลมาน ซึ่งจะให้ผลลัพธ์สถานะหุ่นยนต์มีการรบกวนของสัญญาณน้อยที่สุด โดยให้ความผิดพลาดที่อยู่ในระดับที่ยอมรับได้และสามารถนำไประบุตำแหน่งบนแผนที่ด้วยอัลกอริทึมแบบ AMCL ในที่แคบ เมื่อทั้ง 2 ระบบทำงานได้อย่างถูกต้องตามหลักจลนศาสตร์ สามารถนำระบบจำลองมาจำลองพฤติกรรมเพื่อหาพารามิเตอร์ที่เหมาะสมในการติดตามเส้นทางด้วย Regulated Pure Pursuit (RPP) กับสภาพแวดล้อมที่ได้จัดไว้ ซึ่งการดำเนินการสามารถอธิบายรายละเอียดตั้งแต่ขั้นตอนแรกจนถึงขั้นตอนสุดท้าย ดังหัวข้อต่อไปนี้

3.2 การออกแบบโครงสร้างและอุปกรณ์ของหุ่นยนต์

โครงสร้างหุ่นยนต์ในงานวิจัยได้คำนึงถึงอุปกรณ์จากบริษัท ROBOTIS เป็นบริษัทพัฒนาหุ่นยนต์ Turtle bot ที่ประกอบฐานของโครงสร้าง TB3 Waffle Plate-IPL-01 ซึ่งเป็นแผ่นเพลตที่มีความแข็งแรงและสามารถติดตั้งมอเตอร์ DYNAMIXEL ซีรี่ย์ X ร่วมกับล้อ ISW-01 เข้าไปโดยไม่จำเป็นที่ต้องดัดแปลงเพิ่มเติม

นอกจากนี้ ยังมีส่วนประกอบอื่น ๆ ที่ได้พิจารณา ประกอบด้วย แบตเตอรี่ ระบบจ่ายพลังงาน เซนเซอร์ และระบบฝังตัว ที่มาพร้อมกับคุณสมบัติที่กำหนดมาจากผู้เผยแพร่จากแหล่งออนไลน์แบบเปิด (Open sources) ทำให้สะดวกต่อการสร้างแบบจำลองโครงสร้างของหุ่นยนต์ เป็นรูปแบบ 3D CAD โดยทำงานร่วมกับฟีเจอร์ Assembly ของโปรแกรม SolidWorks

3.2.1 การออกแบบขนาดต้นกำลัง

ในขณะที่อุปกรณ์ส่วนอื่น ๆ ได้พิจารณาเลือกเป็นที่เรียบร้อยแล้ว แต่ยังไม่ได้พิจารณามอเตอร์ขับเคลื่อน เนื่องจากมอเตอร์ DYNAMIXEL ซีรี่ย์ X นั้นมีหลายรุ่นแตกต่างกันออกไป ซึ่งในการเลือกใช้ได้พิจารณาจากทฤษฎี บทที่ 2 หัวข้อที่ 2.6.2 โดยน้ำหนักที่ใช้งานทำให้เกิดแรงเสียดทานสัมพันธ์กับความเร่งโน้มถ่วงและรัศมีล้อ ดังสมการที่ (3.1)

$$\tau_R = \tau_L = 2 \cdot \mu \cdot m \cdot g \cdot r \tag{3.1}$$

เมื่อพิจารณาจากโครงสร้างระบบขับเคลื่อนแบบ Skid Steering พบว่า หุ่นยนต์มีล้อทั้งหมดสองล้อในแต่ละด้าน ผู้วิจัยได้กำหนดพารามิเตอร์ที่เกี่ยวข้องในการออกแบบโดยให้ค่าสัมประสิทธิ์แรงเสียดทาน μ เท่ากับ 0.8 เป็นค่าสัมประสิทธิ์แรงเสียดทานของพื้นทั่วไป (เขียวนาค และพรหมพุฒ., 2565) โดยมวลของหุ่นยนต์ถูกออกแบบให้สามารถรองรับน้ำหนักสูงสุด m เท่ากับ 10 กิโลกรัม ความเร่งโน้มถ่วง g เท่ากับ 9.81 เมตรต่อวินาที^{2} และรัศมีของล้อ r เท่ากับ 0.033 เมตร เป็นขนาดรัศมีล้อ ISW-01

3.2.2 ออกแบบโครงสร้างหุ่นยนต์

เมื่อสามารถกำหนดขนาดต้นกำลังที่เหมาะสม พบว่า มอเตอร์ที่เพียงพอต่อการใช้งานในข้อกำหนดดังกล่าวคือ มอเตอร์ DYNAMIXEL รุ่น XL - 430 เป็นดิจิทัลเซอร์โวมอเตอร์ไฟฟ้ากระแสตรง 12 โวลต์ มีแรงบิดสูงสุดขณะหยุดนิ่งเท่ากับ 1.4 นิวตันเมตร โดยคุณสมบัติเซอร์โวมอเตอร์ คือ อุปกรณ์ที่สามารถควบคุมเครื่องจักรกล หรือระบบการทำงานนั้น ๆ ให้เป็นไปตามความต้องการ เช่น ควบคุมความเร็ว (Speed) ควบคุมแรงบิด (Torque) ควบคุมแรงตำแหน่ง (Position) ที่ให้ผลลัพธ์ตามความต้องการที่มีความแม่นยำสูง ซึ่งเป็นคุณสมบัติที่เหมาะสมในการติดตั้งใช้งานกับหุ่นยนต์อัตโนมัติ

เมื่อได้ทำการพิจารณาเลือกซื้ออุปกรณ์ครบแล้วได้ทำการบันทึกรายการอุปกรณ์สามารถสรุปได้ดังตารางที่ 3.1 เพื่อดำเนินการซื้ออุปกรณ์และนำไปประกอบโครงสร้างในระบบจำลองดังขั้นตอนถัดไป

ตารางที่ 3.1 ลำดับรายการอุปกรณ์

ลำดับ	รายการอุปกรณ์	จำนวน
1	บอร์ดระบบฝังตัวหุ่นยนต์ Raspberry Pi 4 Model B (8 GB)	1
2	เซนเซอร์วัดแรงเฉื่อย WITMOTION 9 Axis IMU	1
3	บอร์ดควบคุมมอเตอร์ U2D2 DYNAMIXEL + PHB Set	1
4	ดิจิทัลเซอร์โวมอเตอร์ DYNAMIXEL XL-430	4
5	แบตเตอรี่ LiFePO4 12 V (6 Ah)	1
6	โมดูลลดแรงดันไฟฟ้ากระแสตรง 12 V to 5 V Converter	1
7	อุปกรณ์เชื่อมต่อระหว่างสายไฟ Terminal Block	1
8	เซนเซอร์ตรวจจับสิ่งกีดขวาง YDLIDAR G2	1
9	TB3 Wheel Tire Set-ISW-01 (ล้อยาง)	4
10	TB3 Waffle Plate-IPL-01 (ฐานโครงสร้าง)	24
11	แท่งรองรับโครงสร้างแบบหกเหลี่ยม Female Hex M3 (45 mm)	30

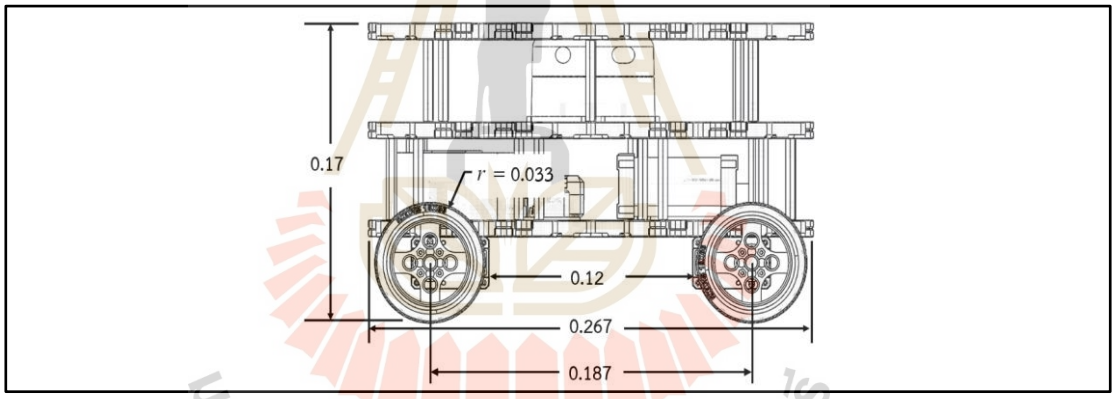
เมื่ออุปกรณ์มาครบทุกชิ้นตามรายการที่กำหนดในตารางที่ 3.1 ได้นำมาประกอบโครงสร้างจากการออกแบบด้วยโปรแกรม SolidWorks โดยการนำพาร์ทแต่ละพาร์ทที่ได้ดาวน์โหลดจากแหล่งออนไลน์แบบเปิด มาประกอบเข้าด้วยกันโดยใช้ฟีเจอร์ Assembly จะได้ว่าหุ่นยนต์ประกอบด้วยโครงสร้างทั้งหมด 3 Layer ดังนี้

Layer 1 เป็นชั้นที่มีการติดตั้ง เซ็ตบอร์ด U2D2 DYNAMIXEL PHB Set กับมอเตอร์ DYNAMIXLE XL-430 บอร์ด Raspberry Pi 4 Model B และเซนเซอร์ IMU WITMOTION WT901C ถือว่าเป็นชั้นที่มีอุปกรณ์ฮาร์ดแวร์มากที่สุดรวมถึงมีการติดตั้งระบบจ่ายพลังงาน

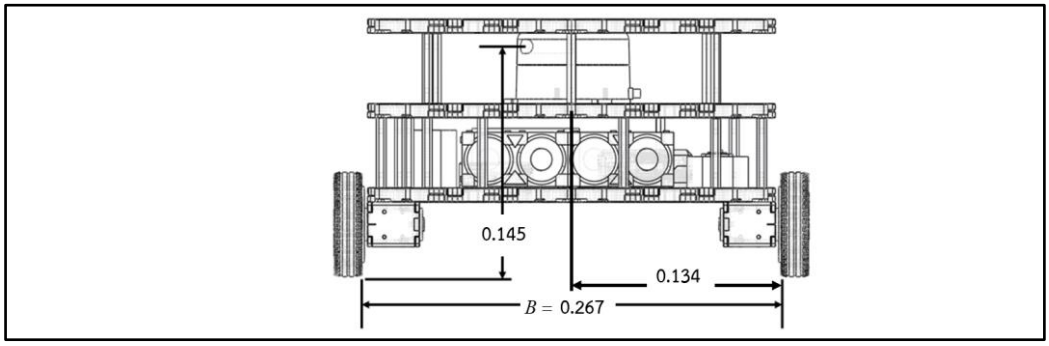
Layer 2 เป็นชั้นที่มีการติดตั้งเฉพาะเซนเซอร์ YDLIDAR G2

Layer 3 เป็นฝาปิดชั้นบนสุดของหุ่นยนต์ (Top cover)

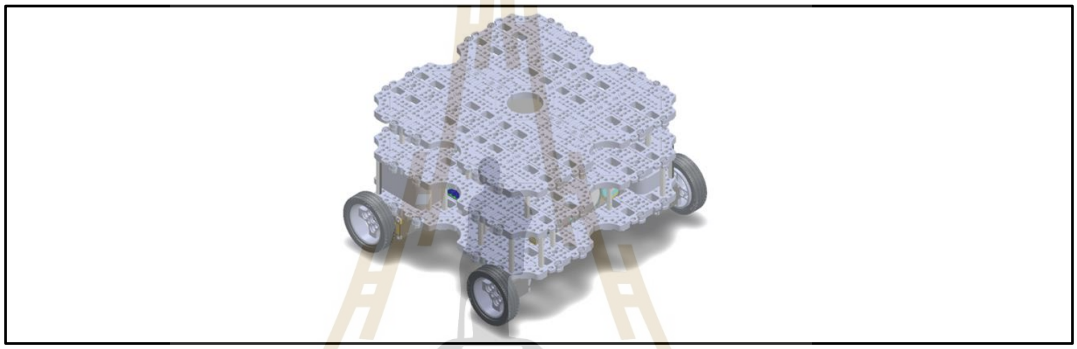
ผลลัพธ์ของโครงสร้างจากการประกอบกันทั้ง 3 Layer สามารถแสดงรายละเอียดได้ดังรูปที่ 3.2 โดยที่ขนาดของส่วนต่าง ๆ ถูกกำหนดในหน่วยเมตร เพื่อง่ายต่อการนำบางส่วนไปใช้ในการคำนวณพารามิเตอร์ที่เกี่ยวข้องในการจำลองโครงสร้างของหุ่นยนต์ ดังรูปที่ 3.2 – 3.4



รูปที่ 3.2 โครงสร้างด้านข้างของหุ่นยนต์



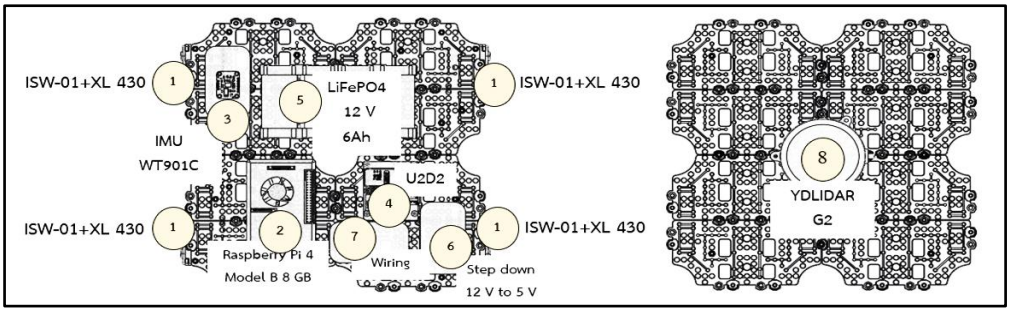
รูปที่ 3.3 โครงสร้างด้านหน้าของหุ่นยนต์



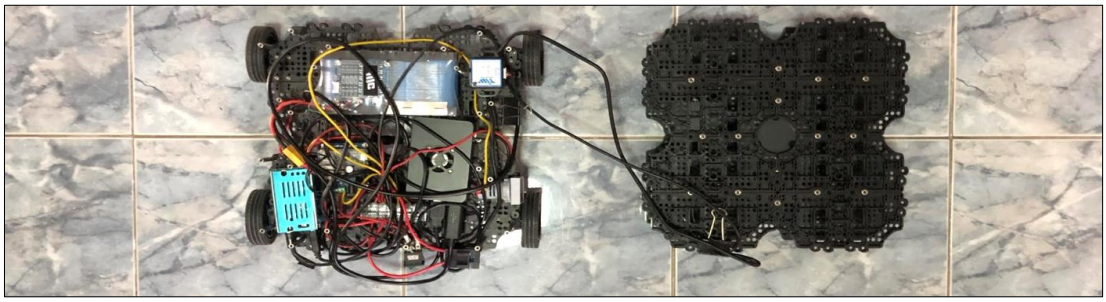
รูปที่ 3.4 โครงสร้างหุ่นยนต์ AMR

3.2.3 การประกอบหุ่นยนต์จริง

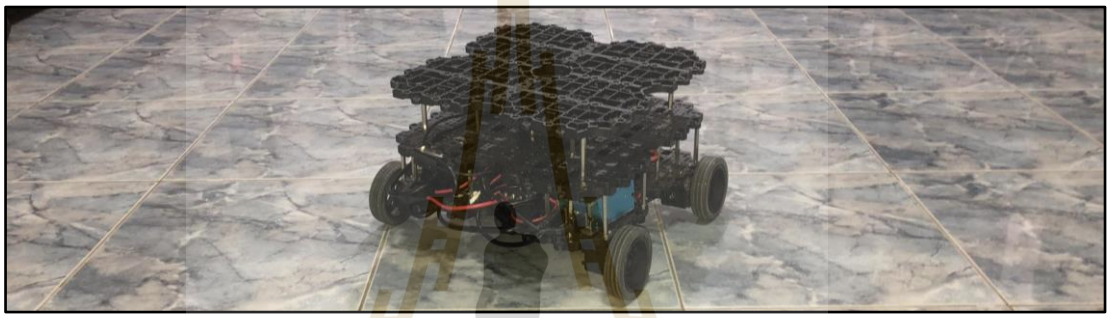
การประกอบโครงสร้างได้ทำการติดตั้งเซนเซอร์ในแต่ละ Layer ตามลำดับดังรูปที่ 3.5 เมื่อเสร็จเรียบร้อยสามารถนำแต่ละ Layer มาวางซ้อนกันโดยแต่ละชั้นถูกขั้นด้วย แท่งรองรับ 6 เหลี่ยม M3 ที่มีความสูงเท่ากับ 45 มิลลิเมตร สามารถแสดงลักษณะของโครงสร้างของหุ่นยนต์จริงหลังจากประกอบเสร็จสิ้น ดังรูปที่ 3.5 – 3.7



รูปที่ 3.5 ลำดับในการติดตั้งอุปกรณ์ของ Layer 1 และ Layer 2

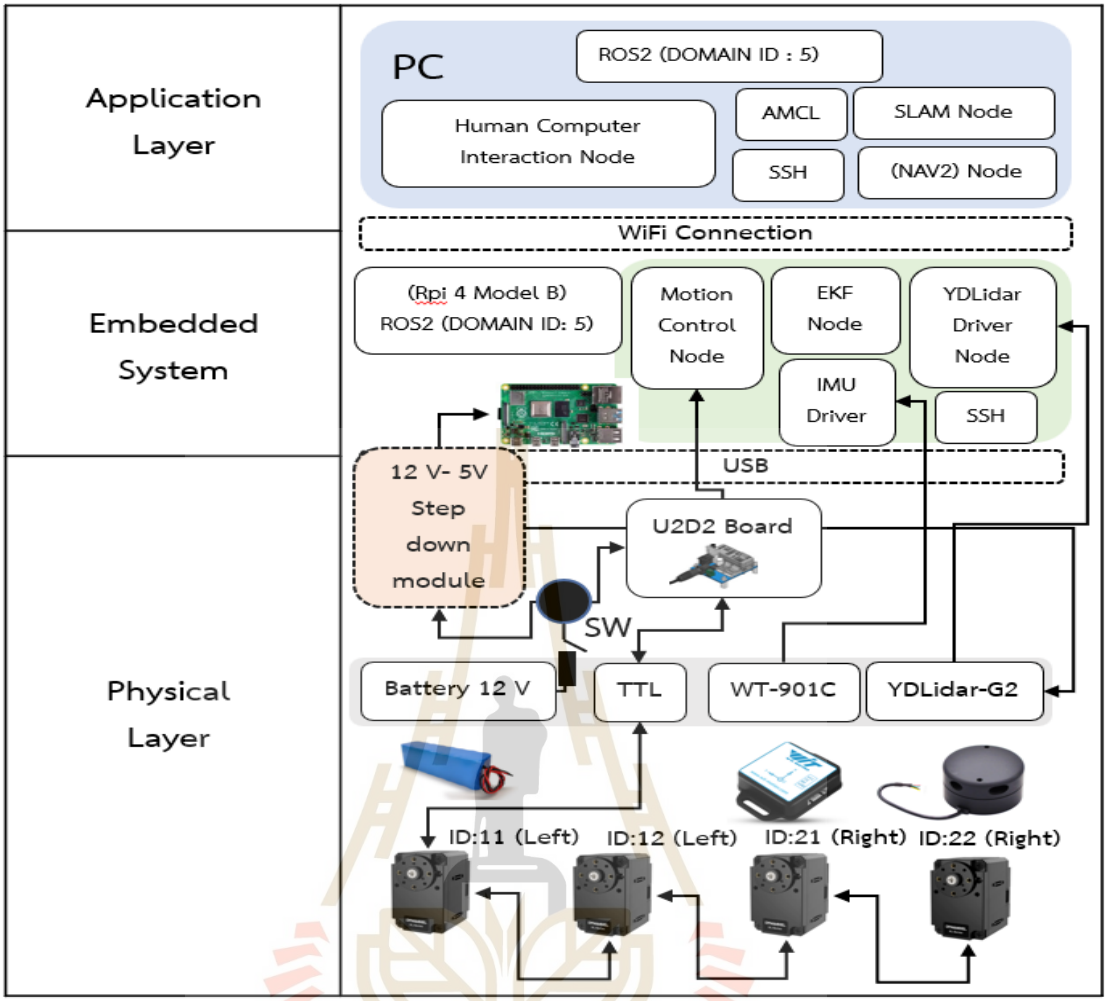


รูปที่ 3.6 หุ่นยนต์ AMR หลังจากประกอบ Layer 1 และ Layer 2



รูปที่ 3.7 หุ่นยนต์ AMR หลังจากประกอบโครงสร้าง

ส่วนประกอบภายในที่ประกอบด้วยแหล่งจ่ายพลังงานที่ให้กับระบบฝังตัว (Embedded System) เซนเซอร์ และมอเตอร์ขับเคลื่อน ถือเป็นระดับขั้นกายภาพ (Physical Layer) ที่ทำงานร่วมกับซอร์ฟแวร์ให้ระบบฝังตัวในการสื่อสารกับฮาร์ดแวร์หลายส่วน และนำข้อมูลที่ได้เข้าสู่อัลกอริทึม เป็นระดับขั้นแปลงการสื่อสารข้อมูลจากไดร์ฟเวอร์ให้อยู่ในรูปแบบข้อความของระบบ ROS กรองสัญญาณด้วยตัวกรองคาลมาน และส่งข้อมูลให้กับระดับขั้นการประยุกต์ใช้งาน (Application Layer) ดังรูปที่ 3.8



รูปที่ 3.8 การเชื่อมต่ออุปกรณ์ภายในและระดับขั้นการประยุกต์ใช้งาน

1) ระดับ Physical Layer คือ ขั้นอุปกรณ์ฮาร์ดแวร์ที่ใช้งานทั้งหมด มีการเชื่อมต่อระบบไฟ 12 V จ่ายพลังงานให้กับบอร์ดควบคุมมอเตอร์ U2D2 และใช้ Step down ลดแรงดันเหลือ 5 V ให้กับบอร์ด Raspberry Pi และเซนเซอร์ YDLidar โดยเซนเซอร์ IMU สามารถใช้ไฟจากแหล่งจ่าย USB ของบอร์ด Raspberry Pi นอกจากนี้ ขั้นนี้ได้กำหนด ID ให้กับมอเตอร์โดยใช้โปรแกรม DYNAMIXEL Wizard เพื่อให้มอเตอร์แต่ละตัวสามารถถูกควบคุมได้อย่างอิสระจากการเชื่อมต่อแบบอนุกรม โดยที่ล้อด้านซ้ายจะแทนหลักแรกด้วย 1 ส่วนด้านขวาจะแทนหลักแรกด้วย 2 อ้างอิงตามรูปที่ 3.8

2) ระดับขั้น Embedded System เป็นระดับที่มีการฝังซอร์ฟแวร์ระบบปฏิบัติการหุ่นยนต์ ROS2 ให้กับบอร์ด Raspberry Pi ประกอบด้วยแพคเกจตัวกรองคาลมาน (EKF) การรีโมทไร้สาย (SSH) และ โนดควบคุมมอเตอร์ตามแบบจำลองจลนศาสตร์ (diff_drive_controller) และ

เซนเซอร์ต่าง ๆ โดยส่วนนี้มีการกำหนดโดเมนสำหรับ ROS2 เพื่อเตรียมการสื่อสารข้อมูลร่วมกับ PC โดยที่ IP ของโดเมนต้องเป็นค่าที่ตรงกันกับบอร์ด Raspberry Pi

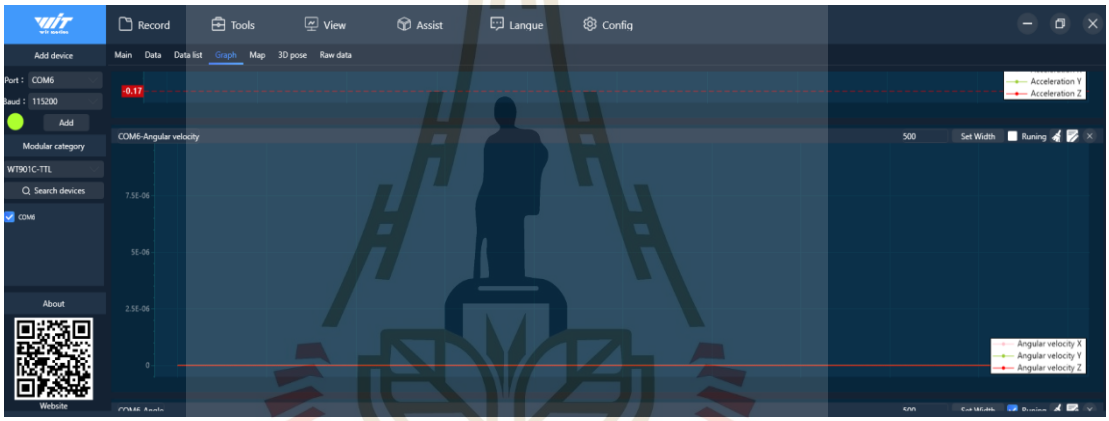
3) ระดับขั้น Application Layer เป็นระดับการนำระบบที่ได้ออกแบบขึ้นไปประยุกต์ใช้งานตามต้องการ มีการติดตั้งระบบนำทางอัตโนมัติ NAV2 ทำงานบนคอมพิวเตอร์ส่วนบุคคล (PC) ที่มีการติดตั้ง ROS2 รวมถึง SSH รองรับการสื่อสารกับระบบฝังตัวของหุ่นยนต์โดยใช้เครือข่ายไร้สายเดียวกัน เนื่องจากบอร์ด Raspberry Pi มีคุณสมบัติในการประมวลผลไม่สูงมาก การแสดงผลผ่านโปรแกรม RVIZ หรือการสื่อสารที่มีโนดเป็นจำนวนมากอาจส่งผลให้เกิดการหน่วงในการส่งข้อมูลจึงได้นำ PC เข้ามาช่วยในการประมวลผลของระบบ NAV2 และโนดอื่น ๆ

3.3 การสอบเทียบเซนเซอร์ไจโรสโคป

การทำให้หุ่นยนต์ทราบองศาปรับตัว (Orientation Angle) หรือ Yaw สามารถวัดด้วยเซนเซอร์ ไจโรสโคปที่เป็นส่วนหนึ่งของเซนเซอร์ WitMotion WT-901C จากการศึกษาคู่มือการใช้งานพบว่า เซนเซอร์ดังกล่าว มีระบบสอบเทียบอัตโนมัติ (Auto Calibration) โดยสามารถพิจารณาความแตกต่างระหว่างการเปิดและปิดใช้ระบบสอบเทียบดังกล่าวด้วยการวางหุ่นยนต์อยู่นิ่ง ๆ ที่มีลักษณะการวางเซนเซอร์อ้างอิงกับระนาบ 2 มิติ ดังรูปที่ 3.9 ขนานกับแกน X อ้างอิงที่เท่ากับ 0 และพิจารณาสัญญาณความเร็วเชิงมุมที่วัดด้วยไจโรสโคปด้วยโปรแกรม WitMotion ดังรูปที่ 3.10 – 3.11 ตามลำดับ

รูปที่ 3.9 ลักษณะการวางเซนเซอร์เพื่อสอบเทียบค่า ω_z ที่วัดด้วยเซนเซอร์ไจโรสโคป

รูปที่ 3.10 ค่า ω_z รอบแกน Yaw ที่ปิดใช้งานระบบสอบเทียบอัตโนมัติของไจโรสโคป (สีแดง)

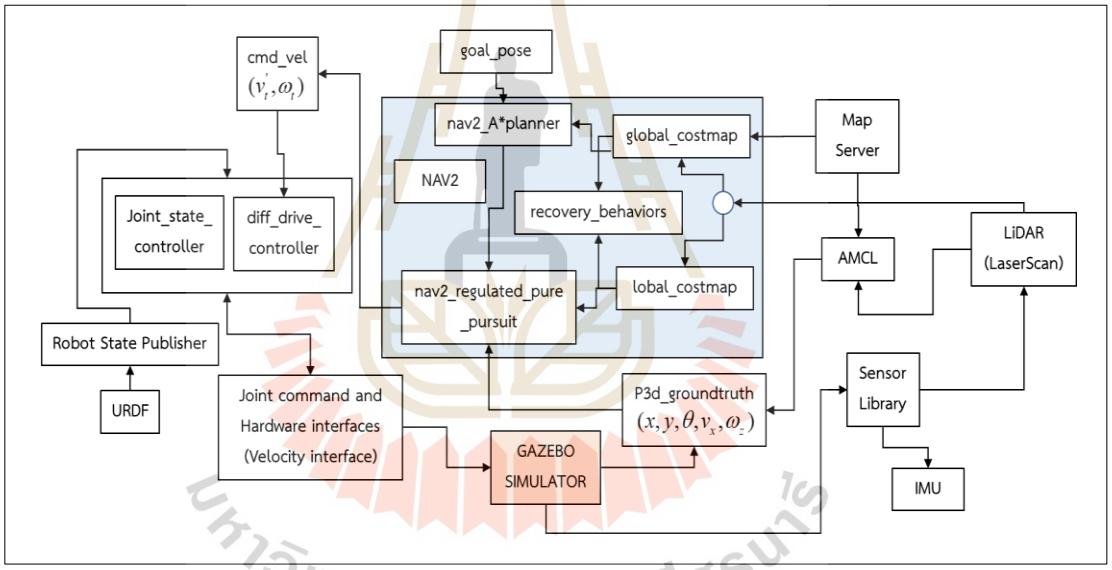


รูปที่ 3.11 ค่า ω_z รอบแกน Yaw ที่เปิดใช้งานระบบสอบเทียบอัตโนมัติของไจโรสโคป (สีแดง)

การวางหุ่นยนต์ให้อยู่กับที่โดยปราศจากการควบคุมใด ๆ ได้ตั้งสมมติฐานในการวัดความเร็วเชิงมุม ω_z คือ ต้องมีค่าเท่ากับ 0 เรเดียนต่อวินาที เมื่อพิจารณาด้วยรูปที่ 3.10 เป็นการปิดใช้งานระบบสอบเทียบได้ผลลัพธ์ความเร็วเชิงมุม ω_z มีค่าประมาณ -0.35 เรเดียนต่อวินาที ในขณะที่ความเร็วเชิงมุมส่วน Roll ω_x (สีเทา) มีค่าประมาณ -1.35 เรเดียนต่อวินาที และ ความเร็วเชิงมุมส่วน Pitch ω_y (สีเขียว) มีค่าประมาณ 0.35 เรเดียนต่อวินาที เมื่อเปรียบเทียบกับการเปิดใช้งานระบบสอบเทียบ ให้ผลลัพธ์ ดังรูปที่ 3.11 ดังนั้น พบว่า การเปิดใช้การสอบเทียบอัตโนมัติสามารถวัดค่าได้ตรงตามสมมติฐานโดยให้ผลลัพธ์ความเร็วเชิงมุมรอบแกนทั้ง 3 แกน มีค่าเท่ากับ 0 เรเดียนต่อวินาที ผู้วิจัยจึงเลือกเปิดใช้งานระบบสอบเทียบอัตโนมัติและนำไปทดสอบความผิดพลาดในการวัดมุมในขั้นตอนถัดไป

3.4 ระบบจำลองหุ่นยนต์ร่วมกับการนำทางอัตโนมัติ

กล่าวได้ว่าการสร้างระบบจำลองนั้นเป็นส่วนที่ต้องให้ความสำคัญต่อพารามิเตอร์คุณลักษณะทางโครงสร้างเพื่อคำนวณฟิสิกส์ของโครงสร้างร่วมกับสภาพแวดล้อมแบบ ODE โดยมีพารามิเตอร์ที่เกี่ยวข้อง คือ โมเมนต์ความเฉื่อย แรงเสียดทาน มวล และพิกัดในการอ้างอิงของส่วนประกอบต่าง ๆ ซึ่งการคำนวณระบบฟิสิกส์ร่วมกับการแสดงผลทางกราฟฟิกแบบ 3 มิติ ที่สามารถทำงานได้ถูกต้องตามแบบจำลองจลนศาสตร์ร่วมกับสถาปัตยกรรม Navigation Stack 2 (NAV2) แสดงได้ ดังรูปที่ 3.12 เป็นการประมวลผลที่มีความซับซ้อนจากการจำลองสภาพแวดล้อมแบ 3 มิติ ผ่านโปรแกรม Gazebo ร่วมกับสถาปัตยกรรมระบบนำทางอัตโนมัติ NAV2 ทำให้ต้องการประมวลผลที่รวดเร็ว จึงได้เลือกใช้คอมพิวเตอร์ส่วนบุคคลที่มีคุณสมบัติหน่วยประมวลผล Intel core i7-7700K ร่วมกับหน่วยประมวลผลภาพ Nvidia Geforce GTX1080



รูปที่ 3.12 ระบบจำลองหุ่นยนต์ร่วมกับ NAV2

ระบบจำลองหุ่นยนต์ดังกล่าว สามารถดำเนินการติดตั้งแพคเกจและเขียนโปรแกรม ที่เกี่ยวข้อง ได้แก่ภาษา C++ Python โดยมีบางส่วนเป็นแพคเกจสำเร็จรูปที่สามารถศึกษาการใช้งานได้จากผู้เผยแพร่และนำส่วนต่าง ๆ มาประกอบเข้าด้วยกันรวมถึงเชื่อมระบบเข้ากับ NAV2 (เป็นส่วนสีฟ้าในรูปที่ 3.12 จากการดำเนินการศึกษาและรวบรวมแพคเกจที่เกี่ยวข้องสามารถอธิบายแบ่งออกเป็น 3 ส่วน ดังนี้

1) ส่วนที่ต้องสร้างขึ้นเอง

(1.1) โครงสร้าง URDF เป็นส่วนที่สร้างขึ้นเพียงครั้งเดียวให้กับระบบจำลอง โดยที่สามารถนำมาประยุกต์ใช้ในระบบจริงร่วมกัน เนื่องจากทั้ง 2 ระบบ มีโครงสร้างแบบเดียวกัน ด้วยส่วนสำคัญ คือ การกำหนดคุณสมบัติของวัสดุจากการนำพาร์ทจากแหล่งออนไลน์แบบเปิดมาประกอบกันด้วยโปรแกรม SolidWorks และใช้ส่วนเสริม SW2URDF

2) ส่วนที่ต้องรวบรวมขึ้น

(2.1) Robot State Publisher เป็นแพคเกจสำเร็จรูปในระบบ ROS2 สามารถเผยแพร่สถานะการทำงานของโครงสร้างแสดงผลผ่าน RVIZ โดยการแปลงไฟล์ URDF ให้สามารถแสดงผลโครงสร้างของหุ่นยนต์แบบ 3 มิติ ด้วยภาษา Python ในการ Launch เพื่อเปิดการทำงานของโนดที่เกี่ยวข้อง

(2.2) diff_drive_controller เป็นปลั๊กอินแบบจำลองจลนศาสตร์ของหุ่นยนต์แบบ Differential drive แต่สามารถนำมาดัดแปลงให้สามารถใช้งานกับหุ่นยนต์แบบ Skid steering ได้ ปลั๊กอินเปรียบเสมือนเป็นตัวแปลงความเร็วที่ควบคุมด้วย (v_t, ω_t) โดยจะให้เอาต์พุตเป็นความเร็วล้อในหน่วยเรเดียนต่อวินาที รวมถึงสามารถนำมาพารามิเตอร์ที่สัมพันธ์กับจลนศาสตร์ที่ได้คำนวณไว้มาประมาณตำแหน่งการเคลื่อนที่ของหุ่นยนต์ด้วยวิธี Dead Reckoning

(2.3) p3d_groundtruth เป็นปลั๊กอินที่ระบุสถานะการเคลื่อนที่ของหุ่นยนต์ได้โดยตรง โดยมีความแม่นยำสูงทำให้ไม่จำเป็นต้องทดสอบความแม่นยำในการระบุตำแหน่ง โดยสถานะในที่นี้จะประกอบด้วย พิกัด องศาการหมุน ความเร็วเชิงเส้นและความเร็วเชิงมุม

(2.4) ไลบรารีจำลองเซนเซอร์ที่ใช้ประกอบด้วย IMU และ LiDAR โดยส่วนนี้ได้กำหนดคุณสมบัติตามอุปกรณ์จริง โดยที่ LiDAR สามารถตรวจจับได้ 360 องศา มีความละเอียด 1 องศา และมีอัตราการอัปเดตข้อมูลเท่ากับ 10 เฮิรตซ์ ส่วนเซนเซอร์ IMU ได้กำหนดเฉพาะอัตราการอัพเดตข้อมมูลให้มีค่าเท่ากับ 20 เฮิรตซ์

(2.5) Joint command and Hardware interfaces เป็นอัลกอริทึมในการรับค่าจุดเชื่อม (Joint) ของหุ่นยนต์ในโปรแกรม Gazebo และส่งคำสั่งความเร็วล้อให้กับหุ่นยนต์ในหน่วยเรเดียนต่อวินาที เป็นการสื่อสารระหว่างจุดเชื่อมของโมเดลในระบบจำลองและระบบจริงโดยความเร็วล้อที่วัดได้จะมีหน่วย เรเดียนต่อวินาที

(2.6) Map Server เป็นข้อมูลแผนที่ในรูปแบบ Occupancy Grid Map สามารถสร้างขึ้นด้วยแพคเกจ slam_toolbox โดยเป็นแพคเกจที่ใช้ร่วมกับหุ่นยนต์จริงด้วย

(2.7) AMCL อัลกอริทึมระบุตำแหน่งบนแผนที่ มีความต้องการอินพุตให้กับส่วนนี้ คือ พิกัดตำแหน่งจากไลบรารี p3d_groundtruth

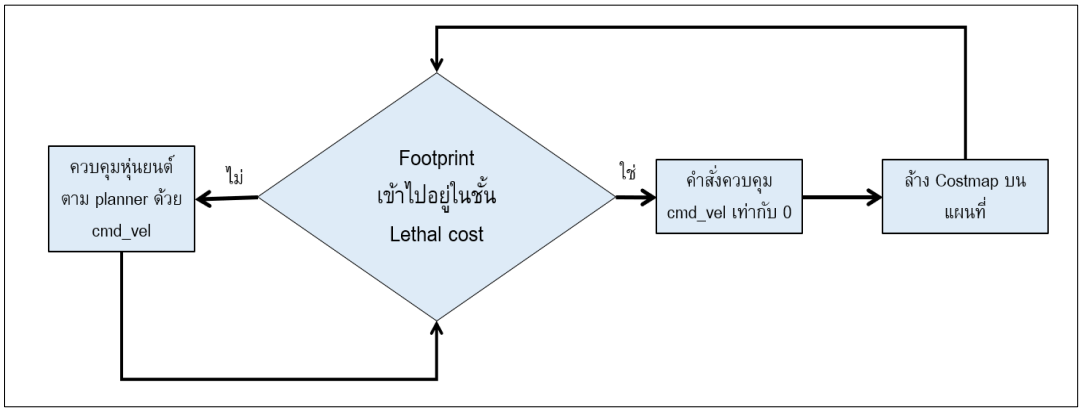
(2.8) GAZEBO SIMULATOR เป็นส่วนที่ต้องกำหนดพารามิเตอร์จำลองทางฟิสิกส์ เช่น ความเร่งโน้มถ่วง g แรงเสียดทานระหว่างล้อกับพื้น มวลและโมเมนต์ความเฉื่อยของหุ่นยนต์ ซึ่งในระบบจะประกอบด้วยโลกจำลองหุ่นยนต์ที่สามารถนำเข้าจากแหล่งโอเพนซอร์สภายนอกเป็นสภาพแวดล้อมที่สามารถปล่อยหุ่นยนต์และจำลองพฤติกรรมการเคลื่อนที่ที่เกิดจากการควบคุมด้วยคำสั่ง cmd_vel เพื่อควบคุมความเร็วเชิงเส้นและความเร็วเชิงมุม (v_t, ω_t)

3) ระบบนำทางอัตโนมัติ NAV2

ก่อนหุ่นยนต์เคลื่อนที่ออกจากจุดเริ่มต้นอย่างอัตโนมัติต้องมีการระบุพิกัดที่เป็นจุดหมายปลายทางบนแผนที่ โดย Topic ชื่อ goal_pose ซึ่งสามารถทำการคลิกบนแผนที่ผ่านโปรแกรม RVIZ ได้โดยตรงเพื่อส่งข้อมูลจุดหมายปลายทางให้หุ่นยนต์เคลื่อนที่ตาม Planning และระบบ ROS จะส่งข้อมูลพิกัดที่ประกอบด้วยพิกัด x พิกัด y และองศาปรับตัวเพื่อระบุทิศการหันของหุ่นยนต์ (Orientation angle) โดยมีเอาต์พุตเพื่อควบคุมหุ่นยนต์ด้วยคำสั่ง cmd_vel ในการติดตามเส้นทางแบบ RPP ที่ได้จากการวางแผนด้วย A-Star โดยที่ cmd_vel จะควบคุมปลั๊กอิน diff_drive_controller ที่บรรจุพารามิเตอร์ที่เกี่ยวข้องในการคำนวณแบบจำลองจลนศาสตร์ให้เอาต์พุตเป็นความเร็วล้อให้สัมพันธ์กับ (v_t', ω_z) นอกจากนี้ระบบ NAV2 จำเป็นต่อการกำหนดพารามิเตอร์อื่น ๆ ที่เกี่ยวข้องกับการวางแผนเส้นทางและความปลอดภัยให้กับหุ่นยนต์ ดังนี้

(3.1) Global_costmap และ local_costmap เป็นอัลกอริทึมกำหนดขอบเขตระยะปลอดภัยของสิ่งกีดขวางโดยที่ Global จะทำในส่วนของข้อมูล Occupancy Grid Map และ Local จะทำในส่วนข้อมูล LaserScan ที่ได้จากเซนเซอร์ LiDAR โดยพารามิเตอร์ คือ อัตราการสะลายตัว (Cost_scaling_factor, α) และขอบเขตของหุ่นยนต์ (Footprint) ซึ่งขนาดของขอบเขตควรมีขนาดเท่ากับหรือมากกว่าขนาดของหุ่นยนต์ เพื่อให้สามารถเคลื่อนที่ผ่านสิ่งกีดขวางได้อย่างปลอดภัย

(3.2) Recovery_behaviors เป็นอัลกอริทึมกำหนดพฤติกรรมเมื่อหุ่นยนต์พบว่าไม่สามารถเคลื่อนที่ต่อไปได้เนื่องจากมีการชนกับสิ่งกีดขวางที่ตรวจจับการชน (Collision detection) ของอัลกอริทึมแบบ RPP หรือขอบเขต Footprint โดยพฤติกรรมหลังจากที่ระบบอยู่สถานการณ์ดังกล่าว หุ่นยนต์จะทำการหยุดรอจนกว่าจะพบว่าเส้นทางสามารถเคลื่อนที่ต่อไปได้ สามารถอธิบายกระบวนการด้วยแผนผังดังรูปที่ 3.13



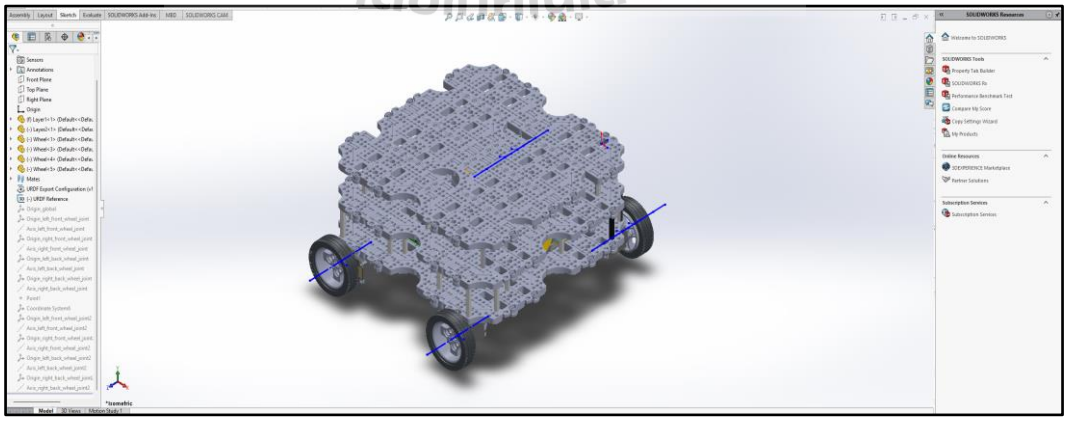
รูปที่ 3.13 ผังการทำงานเมื่อขอบเขตหุ่นยนต์เข้าไปอยู่ในส่วน Lethal cost

จากผังการทำงานจะเห็นได้ว่าหุ่นยนต์มีการตรวจสอบเงื่อนไขจากขอบเขตอยู่ตลอดเวลา เมื่อเข้าเงื่อนไขจะทำการหยุดการเคลื่อนไหวและระบบจะพยายามล้าง Costmap เพื่อกำจัดค่า Costmap ที่ไม่เป็นปัจจุบันหลังจากนั้นระบบจะตรวจสอบเงื่อนไขอีกครั้ง

(3.3) nav2_regulated_pure_pursuit คือ อัลกอริทึมติดตามเส้นทางที่จะทำการออกแบบและจำลองพารามิเตอร์ที่เหมาะสมดังวัตถุประสงค์งานวิจัย

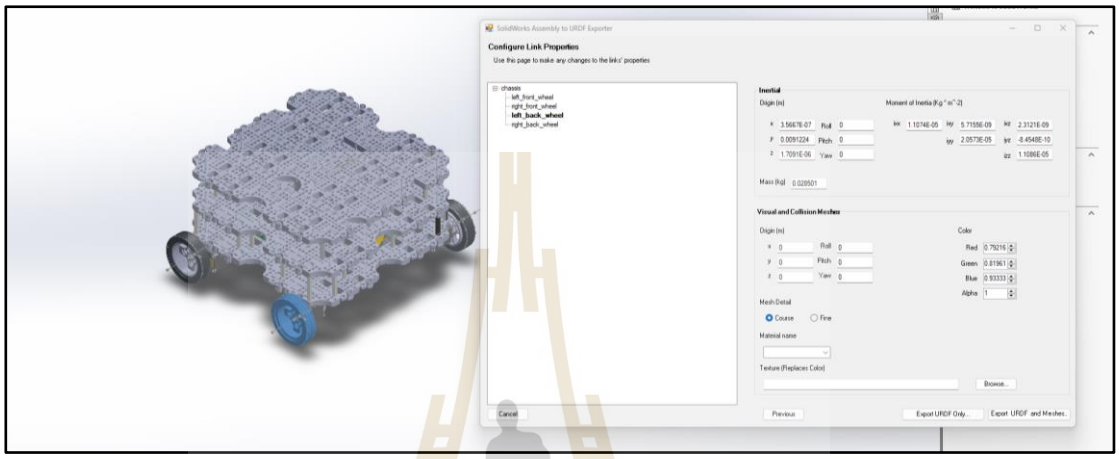
3.4.1 การกำหนดลักษณะทางโครงสร้างหุ่นยนต์ด้วย URDF

จากการนำเสนอสถาปัตยกรรม NAV2 และออกแบบโครงสร้างเรียบร้อยแล้วส่วนนี้เป็นการดำเนินการส่วนของโครงสร้าง URDF เป็นส่วนหนึ่งของสถาปัตยกรรม โดยงานวิจัยได้ใช้ฟีเจอร์การอ้างอิงเรขาคณิต (Reference Geometry) ด้วยโปรแกรม SolidWorks เพื่อกำหนด จุดพิกัด รูปและแกนหมุนอ้างอิง ดังรูปที่ 3.14 หลังจากนั้น ใช้ส่วนเสริม (Add-ins) ชื่อว่า SW2URDF ในการสร้างไฟล์ URDF ที่มีพิกัดแต่ละส่วนประกอบอ้างอิงที่กำหนดไปก่อนหน้านี้



รูปที่ 3.14 การกำหนดแกนการหมุนของล้อ

ทำการแปลงโครงสร้างหุ่นยนต์ให้อยู่ในรูปแบบ URDF ด้วยคำสั่ง Export to URDF ในการสร้างจุดเชื่อมแบบต่อเนื่อง (Continuous joint) รวมถึงสร้างจุดเชื่อมที่มีการยึดติดส่วนประกอบอื่น ๆ ของ ลิงค์ เข้าด้วยกัน โดยใช้พิกัดอ้างอิงด้วยลิงค์ chassis ซึ่งเป็นจุดกำเนิดตรงกลางโครงสร้างของหุ่นยนต์ที่ไม่รวมส่วนของล้อ ดังรูปที่ 3.15



รูปที่ 3.15 การ Export รูปแบบ URDF

หลังจากที่กดปุ่ม Export โปรแกรมจะทำการสร้างรูปแบบ URDF อัตโนมัติ โดยอ้างอิงคุณสมบัติของหุ่นยนต์จากส่วนประกอบต่าง ๆ ที่มีค่าโมเมนต์ความเฉื่อย น้ำหนัก ลักษณะทางกายภาพและเก็บข้อมูล ดังตารางที่ 3.2 – 3.3 ตามลำดับ

ตารางที่ 3.2 ตำแหน่งพิกัดของส่วนประกอบหุ่นยนต์

ชื่อลิงค์ (Link Name)	**การอ้างอิงพิกัด (เมตร)**			**การอ้างอิงรอบแกน (เรเดียน)**		
	x	**y**	**Z**	**Roll**	**Pitch**	**Yaw**
base_footprint	0.0000	0.0000	-0.0493	0.0000	0.0000	0.0000
base_link	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
chassis	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
left_front_wheel	0.0960	0.1350	-0.0163	3.1416	0.0000	-3.1416
right_front_wheel	0.0960	-0.1350	-0.0163	-3.1416	0.3931	0.0000
left_back_wheel	-0.0960	0.1350	-0.0163	3.1416	0.0000	-3.1416
right_back_wheel	-0.0960	-0.1350	-0.0163	-3.1416	0.3931	0.0000

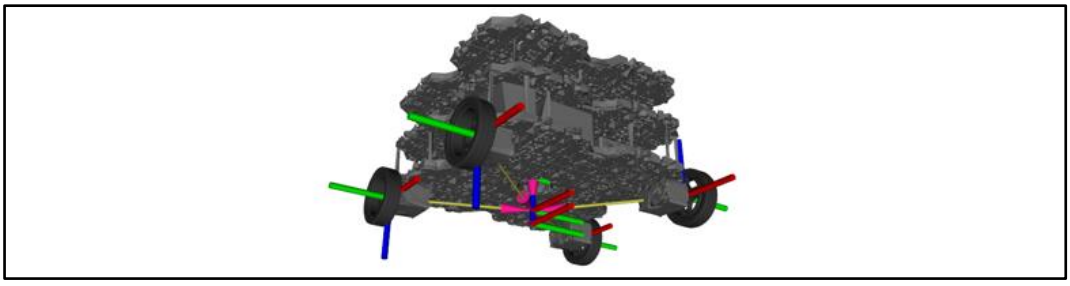
พิจารณาผลการ Export สามารถยกตัวอย่างในการอธิบายได้ว่า ลิงค์ของ chassis มีการอ้างอิงพิกัด (x, y, z) ณ จุดกำเนิดของโครงสร้าง (0, 0, 0) เป็นจุดกำเนิดที่ได้กำหนดด้วยพิกัดของฟีเจอร์ Coordinate System สำหรับอ้างอิงส่วนประกอบอื่น ๆ สามารถยกตัวอย่างในส่วนของลิงค์ left_front_wheel ที่ถูกยึดติดกับโครงสร้างโดยมีระยะห่างจากลิงค์ chassis ตามแนวแกน x เท่ากับ 0.096 เมตร ตามแนวแกน y เท่ากับ -0.135 และมีความสูงตามแนวแกน z ต่ำกว่าลิงค์ chassis เท่ากับ -0.0163 เมตร ตามลำดับ โดยที่ส่วนประกอบทุก ๆ ส่วนจะต้องมีค่าโมเมนต์ความเฉื่อยเพื่อกำหนดพฤติกรรมการเคลื่อนที่ มีข้อมูล ดังตารางที่ 3.3

ตารางที่ 3.3 ขนาดโมเมนต์ความเฉื่อยของแต่ละส่วน

ชื่อลิงค์ (Link Name)	**มวล (กิโลกรัม)**	**เมทริกซ์โมเมนต์ความเฉื่อยแนวทแยงมุม (กิโลกรัมเมตร^2)**		
		i_{xx}	i_{yy}	i_{zz}
chassis	1.82320588	0.001292054	0.001007914	0.002202039
left_front_wheel	0.028501	1.11E-05	2.06E-05	1.11E-05
right_front_wheel	0.028501	1.11E-05	2.06E-05	1.11E-05
left_back_wheel	0.028501	1.11E-05	2.06E-05	1.11E-05
right_back_wheel	0.028501	1.11E-05	2.06E-05	1.11E-05

3.4.2 การเผยแพร่สถานะของโครงสร้าง

ก่อนหน้านี้รูปแบบ URDF เป็นเพียงส่วนที่มีการโปรแกรมเพื่อกำหนดโครงสร้างของหุ่นยนต์เท่านั้น ซึ่งการแสดงผลเพื่อวิเคราะห์ความถูกต้องของโครงสร้างต้องมีแพคเกจในการแปลงข้อมูลจาก URDF เรียกว่า robot_state_publisher ในการเผยแพร่สถานะของโครงสร้างที่ได้กำหนดไว้ โดยเป็นการเขียนโปรแกรมด้วยภาษา Python เพื่อเปิดใช้งานโนด รวมถึงระบุที่อยู่ของไฟล์ URDF ที่สร้างจากส่วนเสริม SW2URDF และทำการ Launch จะได้การแสดงผลของสถานะโครงสร้างหุ่นยนต์ผ่านโปรแกรม RVIZ ดังรูปที่ 3.16



รูปที่ 3.16 การแสดงผลของสถานะหุ่นยนต์จาก URDF

พิจารณาหุ่นยนต์ที่ถูกเผยแพร่สถานะจะเห็นได้ว่าส่วนประกอบลิงค์ทั้ง 4 ล้อ ถูกยึดเข้ากับจุดศูนย์กลางลิงค์ chassis แสดงด้วยลูกศรเป็นจุดเชื่อมที่ได้กำหนดก่อนหน้านี้ โดยเชื่อมต่อกับลิงค์ base_footprint เป็นลิงค์ที่อยู่ในระนาบเดียวกับล้อเพื่ออ้างอิงข้อมูล Odometry ในการระบุตำแหน่งบนระบบพิกัด 3 มิติ

3.4.3 พารามิเตอร์สำหรับควบคุมหุ่นยนต์

การควบคุมหุ่นยนต์จำเป็นต่อการกำหนดพารามิเตอร์ให้สัมพันธ์กับแบบจำลองจลนศาสตร์ของระบบขับเคลื่อนสามารถกำหนดในปลั๊กอิน diff_drive_controller เพื่อความสามารถในการระบุตำแหน่งแบบ Dead Reckoning และคำนวณความเร็วล้อจากคำสั่งอินพุตด้วยความเร็วเชิงเส้นและความเร็วเชิงมุมได้อย่างถูกต้อง เมื่อพิจารณาระบบขับเคลื่อนแบบ Skid Steering พบว่ามีความสัมพันธ์ของความเร็วเชิงมุม ω_z กับค่าสัมประสิทธิ์ ICR ในการคำนวณความเร็วของล้อ ทำให้การคำนวณสามารถแบ่งออกเป็นทั้งหมด 2 ส่วน ดังนี้

1) พิจารณาคุณสมบัติของมอเตอร์ XL - 430 พบว่า จากการทดสอบใช้งานจริงสามารถทำความเร็วในการหมุนได้สูงสุด N เท่ากับ 60.64 รอบต่อนาที สามารถคำนวณความเร็วเชิงเส้น $max(v_x)$ สูงสุดของหุ่นยนต์ที่แปรผันตรงกับรัศมีล้อ r ด้วยสมการที่ (3.1)

$$max(v_x) = -min(v_x) = \frac{2\pi N}{60} r \tag{3.1}$$

2) พิจารณาสัมประสิทธิ์ ICR กำหนดด้วยตัวแปร χ โดยจากการพิสูจน์สมการที่ (2.38) – (2.42) พบว่า χ มีความสัมพันธ์กับความเร็วเชิงมุม ω_z เมื่อ B คือ ระยะห่างของล้อซ้ายและขวา ดังสมการที่ (3.2)

$$\chi = \frac{-\omega_l r + \omega_r r}{max(\omega_z) B} \tag{3.2}$$

ความเร็วเชิงมุมของล้อซ้าย ω_l และขวา ω_r ในหน่วยเรเดียนต่อวินาทีมีความสัมพันธ์กับความเร็วในหน่วยรอบต่อนาที ดังสมการที่ (3.3)

$$\omega_l = \omega_r = \frac{2\pi N}{60} r \tag{3.3}$$

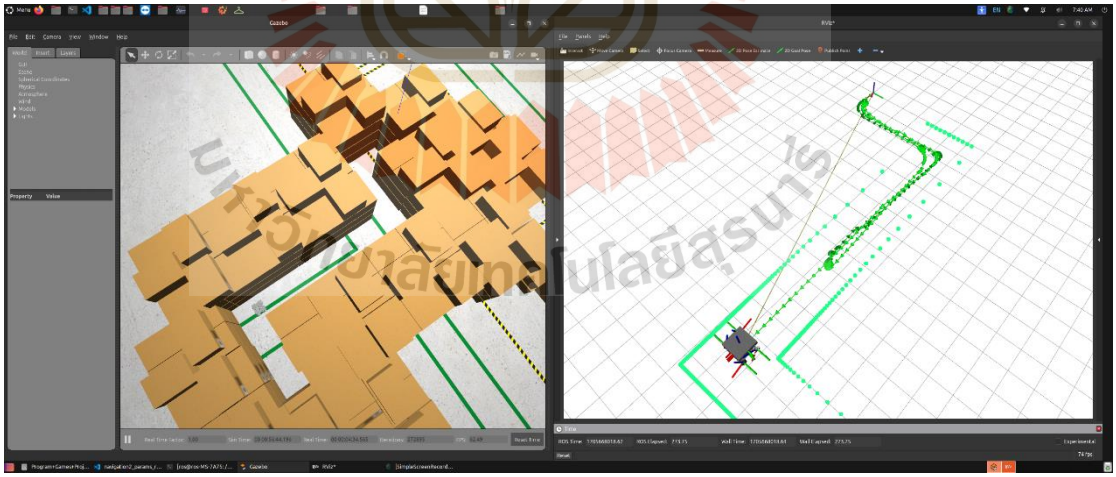
จะได้ว่าการคำนวณค่า χ สามารถออกแบบด้วยความเร็วเชิงมุมสูงสุด $max(\omega_z)$ กำหนดให้มีค่าเท่ากับ 1 เรเดียนต่อวินาที ซึ่งเป็นค่าที่ได้จากการควบคุมหุ่นยนต์ด้วยความเร็วล้อสูงสุด

และมีทิศทางของล้อแต่ละด้านสวนกัน วัดความเร็วเชิงมุมด้วยไลบราลี IMU ซึ่งหลังจากนั้นนำพารามิเตอร์ทั้งหมดที่คำนวณได้ระบุลงในปลั๊กอิน diff_drive_controller ให้กับหุ่นยนต์ทั้งสองระบบ โดยพารามิเตอร์ที่เกี่ยวข้องประกอบด้วยความเร็วเชิงเส้นสูงสุด $max(v_x)$ ความเร็วเชิงเส้นต่ำสุด $min(v_x)$ ความเร็วเชิงมุมสูงสุด $max(\omega_z)$ ความเร็วเชิงมุมต่ำสุด $min(\omega_z)$ ความเร็วล้อสูงสุด $max(\omega_l), max(\omega_r)$ ระยะห่างล้อซ้ายล้อขวา B สัมประสิทธิ์แรงเสียดทานคูลอมบ์ μ สัมประสิทธิ์ ICR χ และรัศมีล้อของหุ่นยนต์ r

3.4.4 การจำลองสภาพแวดล้อมใช้งานหุ่นยนต์

เมื่อหุ่นยนต์สามารถทำงานได้อย่างถูกต้องตามแบบจำลองจลนศาสตร์ ขั้นตอนนี้เป็นการปล่อยหุ่นยนต์ในสภาพแวดล้อมจำลอง ซึ่งต้องมีการสัมผัสกันระหว่างล้อและพื้นดิน จึงกำหนดค่าสัมประสิทธิ์แรงเสียดทานคูลอมบ์ μ เท่ากับ 0.8 ซึ่งเป็นค่าแรงเสียดทานระหว่างล้อกับพื้นในสภาพแวดล้อม Gazebo (Reveira et al., 2019) และความเร่งโน้มถ่วง g เท่ากับ 9.81 เมตรต่อวินาที^{2} โดยสภาพแวดล้อมจำลองที่ได้เลือกใช้ คือ AWS RoboMaker Warehouse

การตรวจสอบความถูกต้องของข้อมูลการวัดระยะทาง (Odometry) และ LaserScan เป็นตำแหน่งจาก p3d_groundtruth จากการควบคุมความเร็วเชิงเส้นและเชิงมุมด้วย topic ชื่อ cmd_vel ผ่านแป้นคีย์บอร์ด สามารถแสดงผลข้อมูลผ่านโปรแกรม RVIZ เทียบกับหุ่นยนต์ที่อยู่ในระบบจำลองด้วยโปรแกรม Gazebo ดังรูปที่ 3.17



รูปที่ 3.17 สภาพแวดล้อมจำลองหุ่นยนต์

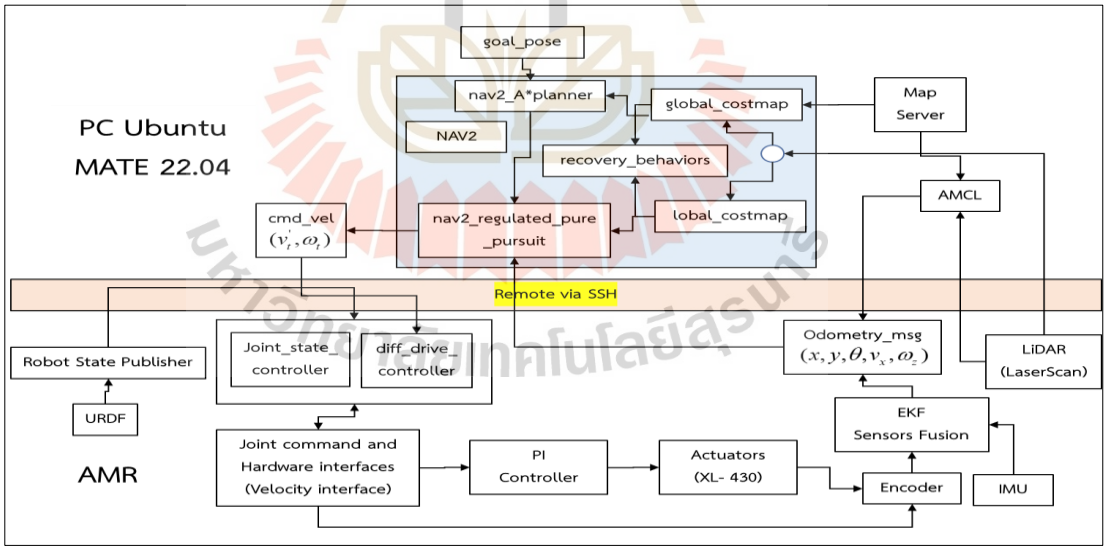
การควบคุมหุ่นยนต์ให้เคลื่อนที่ทั้ง 2 ระบบทำโดยใช้วิธีเดียวกัน คือ ควบคุมด้วย Topic ชื่อ cmd_vel โดยในการทดสอบนี้เป็นการควบคุมโดยใช้แป้นคีย์บอร์ดด้วยแพคเกจ

teleop_twist_keyboard ใช้ปุ่ม i, <, j และ l โดยที่อินพุตอยู่ในรูปความเร็วเชิงเส้นและความเร็วเชิงมุม (v_t, ω_t) ซึ่งการกดปุ่ม I และ < เป็นการควบคุม v_t ส่วนปุ่ม j, l เป็นการควบคุม ω_t

สำหรับการตรวจสอบความถูกต้องของข้อมูล Odometry สามารถสังเกตข้อมูลจาก LaserScan ว่ามีความอยู่นิ่งขณะที่หุ่นยนต์กำลังเคลื่อนไหวมากน้อยเพียงใด หากข้อมูล Odometry สัมพันธ์กับแบบจำลองจลนศาสตร์ LaserScan จะมีการขยับเพียงเล็กน้อยหรือไม่ขยับเลย ในทางกลับกันกรณีที่ LaserScan มีการขยับมาก ควรตรวจสอบความถูกต้องของพารามิเตอร์หรือตำแหน่งพิกัดติดตั้งเซนเซอร์ LiDAR เนื่องจากพฤติกรรมดังกล่าวแสดงให้เห็นว่าข้อมูลของ LaserScan ไม่สัมพันธ์กับพารามิเตอร์แบบจำลองจลนศาสตร์ที่ได้คำนวณไว้ในสมการที่ (3.1) – (3.3)

3.5 ระบบสถาปัตยกรรมนำทางอัตโนมัติของหุ่นยนต์จริง

เนื่องจากอุปกรณ์ที่ใช้งานจริงมีความไม่แน่นอนจากการวัดที่เกิดขึ้นเนื่องจากการรบกวนทางไฟฟ้าหรือสภาพแวดล้อมใช้งาน ทำให้อัลกอริทึมของระบบหุ่นยนต์จริงจำเป็นต่อความซับซ้อนและหลากหลายมากกว่า ประกอบด้วยซอร์ฟแวร์ไดร์เวอร์ของเซนเซอร์เพื่อให้สามารถสื่อสารกับระบบฝังตัวรวมถึงอัลกอริทึมที่รับมือกับปัญหาความไม่แน่นอนจากการวัดที่เกิดขึ้น โดยสถาปัตยกรรมทั้งหมดของหุ่นยนต์จริงสามารถทำงานร่วมกับ NAV2 ดังรูปที่ 3.18



รูปที่ 3.18 สถาปัตยกรรม NAV2 กับหุ่นยนต์จริง

ในขั้นตอนนี้เป็นส่วนที่รวบรวมแพคเกจที่แตกต่างกับระบบจำลองประกอบด้วย OpenSSH การกำหนดค่าควบคุมป้อนกลับ การบูรณาการข้อมูลเซนเซอร์ด้วยตัวกรองคาลมาน โดยรายละเอียดแต่ละส่วนสามารถอธิบายได้ดังต่อไปนี้

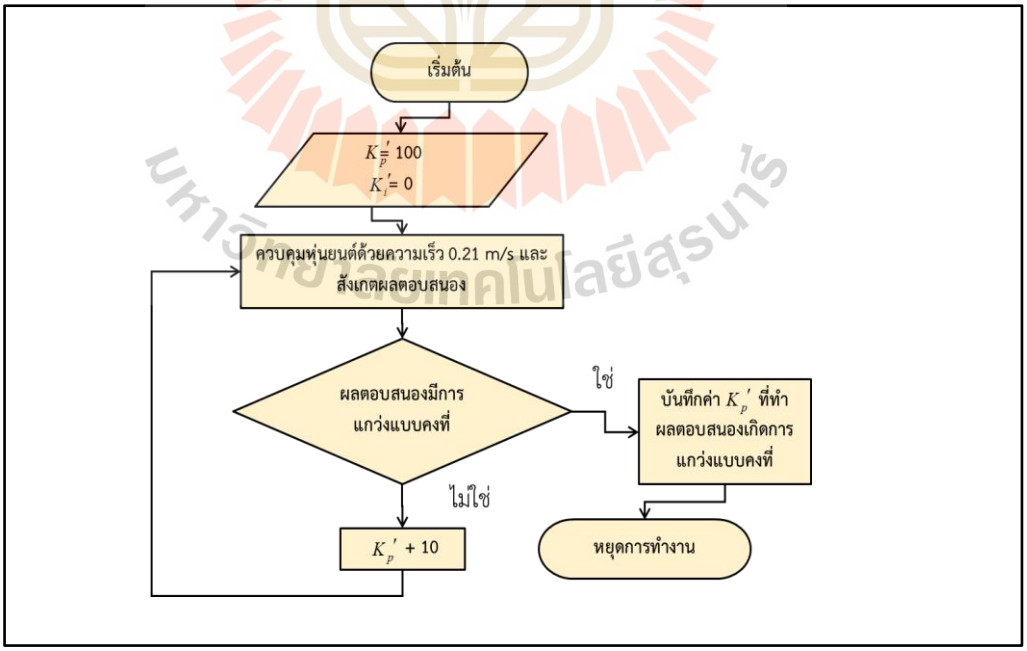
1) OpenSSH จะทำการจดจำที่อยู่ IP ของเครื่อง Raspberry Pi โดยหน้าที่ของ SSH เป็น Network Protocol ที่สามารถแลกเปลี่ยนข้อมูลที่มีความปลอดภัยระหว่างอุปกรณ์เครือข่าย 2 ตัว ใช้ Linux หรือ Unix เป็นระบบปฏิบัติการพื้นฐานในการเข้าถึงบัญชีผู้ใช้ โดยการส่งข้อมูลจะอยู่ในรูปแบบตัวอักษร (Plaintext) ที่มีการเข้ารหัสข้อมูล (Encryption) เพื่อให้ข้อมูลเป็นความลับและสามารถส่งข้อมูลผ่านเครือข่ายอินเทอร์เน็ตได้อย่างสมบูรณ์

2) ระบบควบคุมป้อนกลับแบบ พีไอ มาพร้อมกับมอเตอร์ XL-430 โดยการควบคุมความเร็วที่สามารถใช้โหมด Velocity Control โดยกำหนดค่าตัวควบคุมในรูปแบบเลขดิจิทัลขนาด 14 บิต ซึ่งมีค่าตั้งแต่ 0 - 16383

3) การบูรณาการข้อมูลเซนเซอร์ด้วยตัวกรองคาลมานแบบขยายเพื่อกรองสัญญาณรบกวนที่เกิดขึ้นจากการวัดและยังคงรูปสัญญาณเดิมอยู่ ซึ่งในส่วนนี้ต้องมีการกำหนดเมทริกซ์ความแปรปรวนให้เหมาะสมกับค่าความแปรปรวน (Variance) ของเซนเซอร์

3.5.1 การออกแบบระบบควบคุมด้วยวิธีการของซีกเลอร์-นิโคลส์

การเลือกค่าที่เหมาะสมกับระบบควบคุมหุ่นยนต์ได้เลือกใช้กฎของซีกเลอร์-นิโคลส์ด้วยวิธีการวัฏจักรสุดท้าย จากการศึกษาพบว่าควรออกแบบตัวควบคุมโดยใช้ความเร็วที่จะนำไปใช้งานจึงออกแบบโดยใช้ความเร็วอินพุตสูงสุดเท่ากับ 0.21 เมตรต่อวินาที เคลื่อนที่ในระยะทาง 7 เมตร ดำเนินการตามขั้นตอนด้วยแผนภาพรูปที่ 3.19



รูปที่ 3.19 กระบวนการหาค่าตัวควบคุม

ค่าพารามิเตอร์ระบบควบคุมความเร็วของมอเตอร์รุ่น XL-430 ตัวเลขจะอยู่ในรูปแบบของตัวเลขดิจิทัล ซึ่งการคำนวณค่าให้เป็นไปตามทฤษฎีควบคุม ทำได้โดยนำตัวควบคุม พี ในรูปแบบดิจิทัลหารด้วย 128 ส่วน ตัวควบคุม ไอ ในรูปแบบดิจิทัลหารด้วย 65536

3.5.2 การบูรณาการข้อมูลเซนเซอร์ด้วยตัวกรองคาลมาน

การใช้ตัวกรองคาลมานมีจุดประสงค์เพื่อกำจัดสัญญาณรบกวนและยังคงค่าสัญญาณที่แท้จริงอยู่สามารถเพิ่มประสิทธิภาพให้การวัดมีความแม่นยำมากยิ่งขึ้น รวมถึงการประมาณตำแหน่งด้วยข้อมูลเซนเซอร์มากกว่า 1 ข้อมูลเป็นต้นไป เพื่อความสามารถในการทดแทนข้อมูลที่ไม่สามารถวัดได้ในบางสภาพแวดล้อม ประกอบด้วย ความเร็วเชิงเส้น v_x และความเร็วเชิงมุม ω_z เป็นสัญญาณป้อนกลับจากเซนเซอร์ Encoder และ เซนเซอร์ไจโรสโคป ตามลำดับ สามารถประมาณสถานะการเคลื่อนที่เป็นผลลัพธ์ที่ได้จากตัวกรองในสมการที่ (3.4)

$$x_{t+1} = f(x_t, u_t) = Fx_t + Bu_t \tag{3.4}$$

สถาณะการประมาณ คือ พิกัดของหุ่นยนต์ (x', y') ที่เปลี่ยนไปและมุมองศาปรับตัวบนระบบพิกัด ϕ เป็นผลลัพธ์สัมพันธ์กับความเร็วเชิงเส้น v_x และความเร็วเชิงมุม ω_z แสดงในรูปเมทริกซ์ดังสมการที่ (3.5)

$$\begin{bmatrix} x' \\ y' \\ v_x' \\ \omega_z' \end{bmatrix} = f(x_t, u_t) = \begin{bmatrix} x + v_x \cos(\phi)\Delta t \\ y + v_x \sin(\phi)\Delta t \\ v_x \\ \phi + \omega_z \Delta t \end{bmatrix} \tag{3.5}$$

พิจารณาขั้นตอนคาดการณ์ล่วงหน้าของตัวกรองคาลมานแบบขยายมีความสัมพันธ์กับค่าความแปรปรวนของกระบวนการ ดังสมการที่ (3.6)

$$P^{-}(t_i) = \Phi_{t_{i-1}}^{t_i} \cdot P(t_i - 1) \cdot (\Phi_{t_{i-1}}^{t_i})^T + Q \tag{3.6}$$

จะได้ว่าเมทริกซ์ $\Phi_{t_{i-1}}^{t_i}$ คือ เมทริกซ์การถ่ายโอนตัวแปรสถานะแปลงมาจากเมทริกซ์ในรูปแบบจาโคเบียน เป็นอัตราการเปลี่ยนแปลงของสถาณะดังสมการที่ (3.7) โดยที่ Q คือ ค่าความแปรปรวนของกระบวนการ

$$\Phi_{t_{i-1}}^{t_i} = \begin{bmatrix} 1 & 0 & \cos(\phi)\Delta t & -v_x \sin(\phi)\Delta t \\ 0 & 1 & \sin(\phi)\Delta t & v_x \cos(\phi)\Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

เมื่อพิจารณาทฤษฎีบทตัวกรองคาลมานแบบขยาย ในขั้นตอนการคำนวณอัตราขยายคาลมาน K พบว่ามีความสัมพันธ์กับตัวแปรสถานะและความแปรปรวนดังสมการที่ (3.8)

$$K_k = P_k^- h_k^T(\hat{X}_k^-) P_k^{-1}(h_k(\hat{X}_k^-) P_k^- h_k^T(\hat{X}_k^-) + R) \tag{3.8}$$

เมื่อ h เป็นสัญญาณเอาต์พุตของเซนเซอร์ และ R เป็นเมทริกซ์ค่าความแปรปรวนสัญญาณรบกวนของเซนเซอร์ที่มีขนาดเท่ากับจำนวนอินพุตข้อข้อมูลที่วัดได้คือ 2x2 ดังสมการที่ (3.9)

$$R = \begin{bmatrix} \sigma_{v_x}^2 & 0 \\ 0 & \sigma_{\omega_z}^2 \end{bmatrix} \tag{3.9}$$

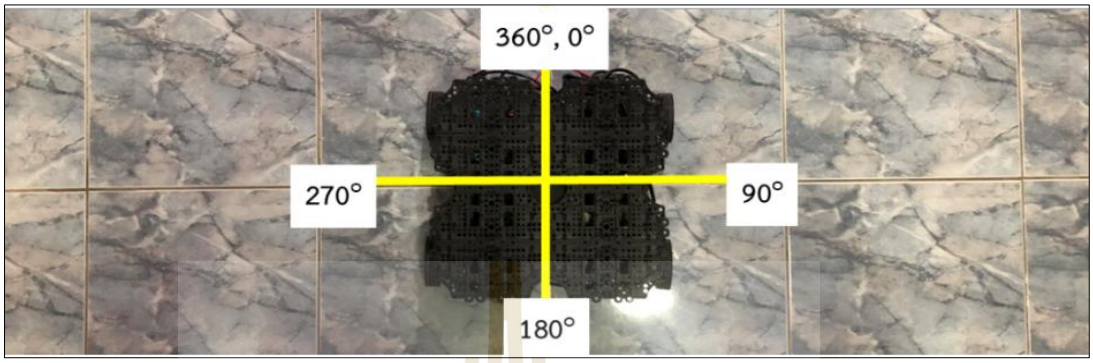
เมทริกซ์ความแปรปรวนของกระบวนการกำหนดให้แบบจำลองมีความถูกต้อง ด้วยค่าความแปรปรวนน้อยมาก ๆ เท่ากับ $1e^{-9}$ โดยปรับค่าเฉพาะ σ_x^2 และ σ_y^2 ที่เป็นค่าความแปรปรวนของสถานะ v_x และ ω_z ตามลำดับ ในการปรับปรุงประสิทธิภาพในการกรองสัญญาณซึ่งบรรจุในเมทริกซ์ Q ดังสมการที่ (3.10)

$$Q = \begin{bmatrix} 1e^{-9} & 0 & 0 & 0 \\ 0 & 1e^{-9} & 0 & 0 \\ 0 & 0 & \sigma_x^2 & 0 \\ 0 & 0 & 0 & \sigma_y^2 \end{bmatrix} \tag{3.10}$$

3.5.3 ความผิดพลาดสัมบูรณ์เซนเซอร์ไจโรสโคป

หุ่นยนต์ที่สามารถระบุตำแหน่งบนระนาบ 2 มิติ ทำให้ความแม่นยำในการวัดมุม Yaw เป็นส่วนสำคัญในการประมาณตำแหน่งของหุ่นยนต์เมื่อความเร็วเชิงมุม ω_z ที่เปลี่ยนไป หลังจากที่ระบบได้พารามิเตอร์ตัวกรองที่เหมาะสมโดยที่สามารถกรองสัญญาณรบกวนในการวัดความเร็วเชิงมุมแล้ว ขั้นตอนนี้ได้ทำการทดสอบค่าผิดพลาดในการวัดมุมเพื่อพิจารณาความแม่นยำ

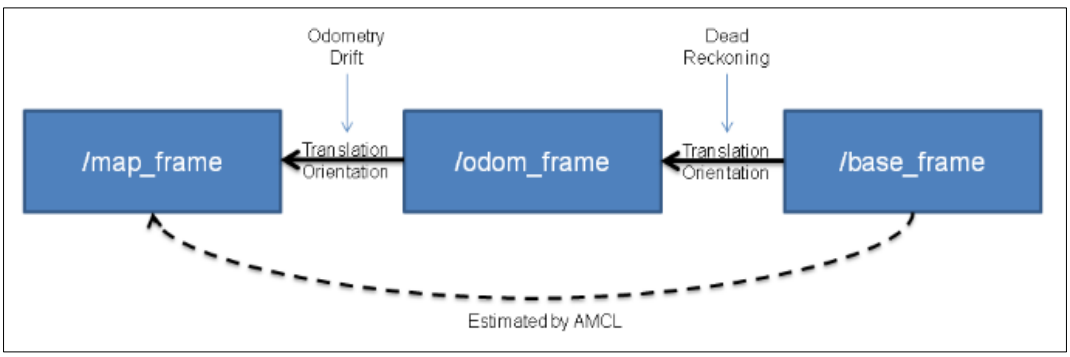
จากที่ได้เปิดระบบสอบเทียบเซนเซอร์ไจโรสโคปอัตโนมัติ ทำการทดสอบทั้งหมด 3 ครั้ง โดยแต่ละครั้งทำการควบคุมความเร็วเชิงมุมของหุ่นยนต์ที่แตกต่างกันเทียบกับมุมอ้างอิง ดังรูปที่ 3.20



รูปที่ 3.20 การวัดความผิดพลาดการวัดมุมด้วยไจโรสโคป

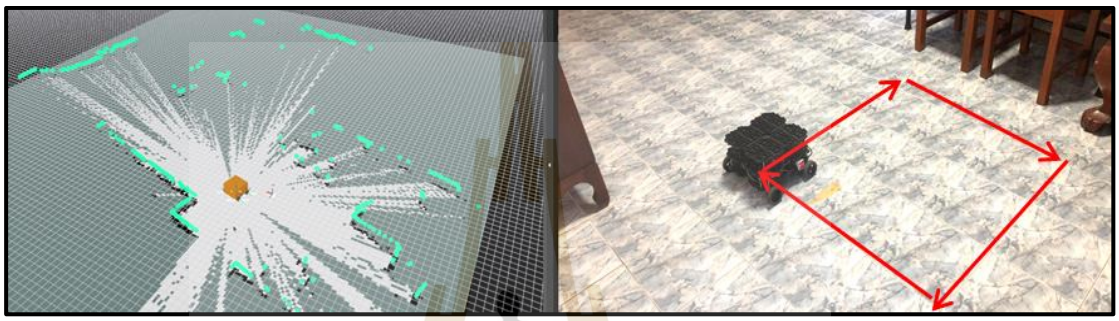
3.5.4 การระบุตำแหน่งของหุ่นยนต์

การประมาณตำแหน่งด้วยตัวกรองคาลมานสามารถตามสมมติฐานที่ได้จากการศึกษางานวิจัยที่เกี่ยวข้อง คือ สามารถเพิ่มประสิทธิภาพในการระบุตำแหน่งได้ในระดับหนึ่ง แต่ถึงอย่างไรการระบุตำแหน่งในการใช้งานจริงยังต้องพิจารณาความผิดพลาดสะสมที่ไม่สามารถเลี่ยงได้เนื่องด้วยการสื่อสารข้อมูลที่ไม่ต่อเนื่องอาจส่งผลทำให้ระบบมีความไม่น่าเชื่อถือต่อการใช้งานในระยะยาว สามารถนำการประมาณตำแหน่งที่เป็นข้อมูลแบบ Odometry จากตัวกรองคาลมานเป็นอินพุตของการระบุตำแหน่งบนแผนที่ด้วย AMCL แทนพิกัดตำแหน่งอินพุตที่เป็นข้อมูลจากวิธี Dead Reckoning ซึ่งความสามารถของ AMCL นั้นจะทำการปรับตำแหน่งของหุ่นยนต์ด้วยอนุภาคความน่าจะเป็น ณ เฟรม /base_footprint กับ /map โดยตรง โดยกระจายความน่าจะเป็นอ้างอิงด้วยข้อมูลของ LaserScan ดังรูปที่ 3.21



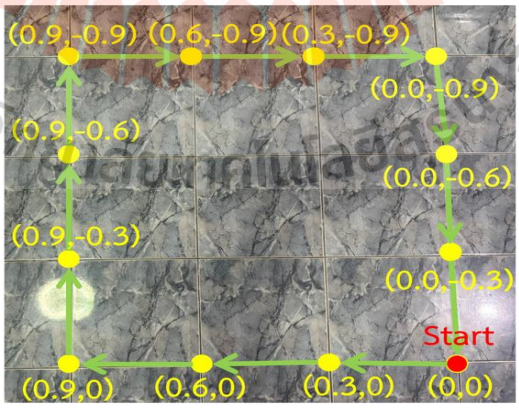
รูปที่ 3.21 การแปลงเฟรมจากอัลกอริทึม AMCL

ในส่วนของเฟรม /map เป็นส่วนที่ได้จาก Map Server เกิดจากการสร้างแผนที่ด้วยแพคเกจ slam_toolbox เพื่อเก็บแผนที่ดังรูปที่ 3.22 และได้บันทึกพิกัดอ้างอิงโดยการวัดกริดของกระเบื้องกำหนดให้เป็นตำแหน่งจริง (Ground Truth) ในขณะเดียวกัน เพื่อนำไปเปรียบเทียบกับพิกัดที่วัดได้กับ วิธี Dead Reckoning (Raw) ตัวกรองคาลมาน (EKF) และ AMCL ที่มีจุดเริ่มต้นของหุ่นยนต์ ณ ตำแหน่ง (0, 0) ทำการทดลองซ้ำทั้งหมด 10 ครั้ง โดยใช้ความเร็วอินพุตจากแป้นคีย์บอร์ด v_t เท่ากับ 0.1 และ ω_t เท่ากับ 0.25 เรเดียนต่อวินาที



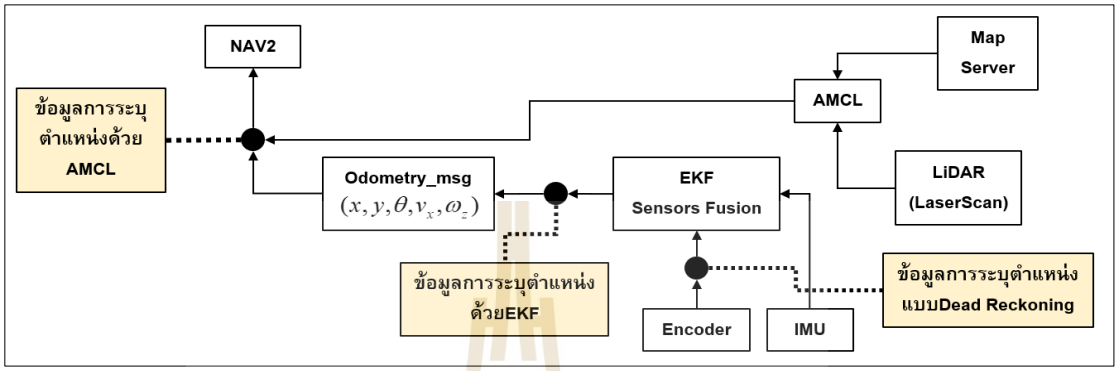
รูปที่ 3.22 การสร้างแผนที่ในการทดสอบ AMCL

สำหรับการประเมินประสิทธิภาพจากการเก็บข้อมูลตำแหน่งจริงด้วยตลับเมตร สามารถแปลงตำแหน่งที่ต้องการให้หุ่นยนต์จอดเพื่อประเมินประสิทธิภาพเป็นพิกัดทั้งหมด 12 จุด และจะได้ว่าหุ่นยนต์จะต้องเคลื่อนที่เป็นระยะทางทั้งหมด 3.6 เมตร ดังรูปที่ 3.23



รูปที่ 3.23 พิกัดบนระนาบ 2 มิติสำหรับประเมินประสิทธิภาพการระบุตำแหน่ง

จากผลลัพธ์ของอัลกอริทึม 3 ส่วนที่แตกต่างกัน จะอยู่ในการสื่อสารในระบบ ROS สามารถดึงมาวิเคราะห์ความแม่นยำในการระบุตำแหน่งของแต่ละส่วน ซึ่งมีความสัมพันธ์กับสถาปัตยกรรมของระบบ ดังแผนภาพการทำงานรูปที่ 3.24



รูปที่ 3.24 แผนภาพการดึงข้อมูลพิกัดตำแหน่งของหุ่นยนต์ 3 ส่วน

จากแผนภาพสามารถอธิบายที่มาผลลัพธ์ของพิกัดในแต่ละส่วนได้ดังต่อไปนี้

(1) Raw Odometry (Raw) เป็นข้อมูลการประมาณสถานะของหุ่นยนต์ที่คำนวณมาจากปลั๊กอิน diff_drive_controller โดยใช้การวัดความเร็วของล้อซ้ายและขวาของหุ่นยนต์ด้วยเซนเซอร์ Encoder เท่านั้น โดยมีหลักการเช่นเดียวกับทฤษฎีที่ 2.7.2

(2) Filtered Odometry (EKF) เป็นข้อมูลประมาณสถานะที่บูรณาการข้อมูล 2 ส่วนเข้าด้วยกัน ประกอบด้วย v_x ที่วัดด้วยเซนเซอร์ Encoder จากมอเตอร์ XL – 430 และ ω_z ที่วัดด้วยเซนเซอร์สโคป ของ WitMotion Wt901C โดยข้อมูล 2 ส่วนนี้ ถูกกรองสัญญาณรบกวนด้วยตัวกรองคาลมานแบบขยายให้มีสัญญาณรบกวนที่น้อยลง

(3) AMCL เป็นการระบุตำแหน่งบนแผนที่ร่วมกับข้อมูล Occupancy Grid Map เป็นผลลัพธ์มาจากการ SLAM โดยการระบุตำแหน่งส่วนนี้จะใช้อนุภาคความน่าจำเป็นในการอัพเดทสถานะซึ่งมีความแม่นยำมากที่สุดจากสมมติฐานที่ตั้งไว้เนื่องจากมีการใช้ข้อมูล LaserScan เป็นตัวอ้างอิงและปรับตำแหน่งและมุมองศาปรับตัวของหุ่นยนต์ให้เหมาะสมกับสภาพแวดล้อม

ระบบ NAV2 ต้องการรับรู้พิกัดของหุ่นยนต์ขณะเคลื่อนที่อยู่บนแผนที่ในการวางแผนเส้นทางและสร้างขอบเขตสิ่งกีดขวางด้วย Costmap จากขั้นตอนประเมินประสิทธิภาพจากอัลกอริทึมที่แตกต่างกัน จะนำผลลัพธ์มาวิเคราะห์เพื่อหาความเหมาะสมมากที่สุดในการใช้งานร่วมกับระบบนำทางอัตโนมัติ NAV2 ซึ่งต้องการอินพุต คือ ตำแหน่งของหุ่นยนต์พิกัด (x, y) ดังนั้นการประเมินประสิทธิภาพสามารถประเมินด้วยระยะผิดพลาดระหว่างพิกัดที่เป็นค่าจริงกับค่าจากการวัด $d_{e,i}$ ดังสมการที่ (3.11) – (3.13)

$$d_{e,i} = \sqrt{(x_{g,i} - x_i)^2 + (y_{g,i} - y_i)^2} \tag{3.11}$$

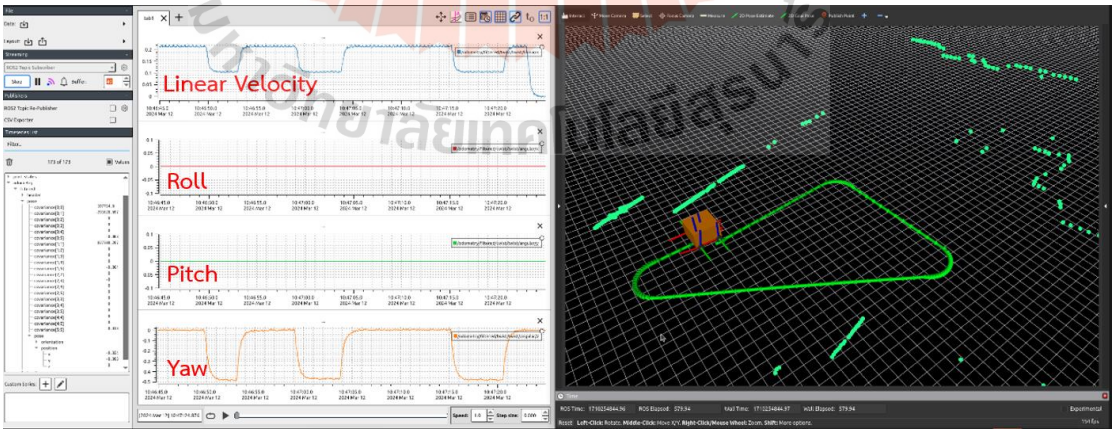
เมื่อ $(x_{g,i}, y_{g,i})$ เป็นพิกัดตำแหน่งจริง (Ground truth) และ (x_i, y_i) เป็นพิกัดที่ประมาณด้วยวิธีการ Dead Reckoning (Raw) ตัวกรองคาลมาน (EKF) และ AMCL ตามลำดับ โดยที่ $d_{e,i}$ สามารถนำมาประเมินประสิทธิภาพด้วยการคำนวณค่า RMSE ในสมการที่ (3.12)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} d_{e_i}^2} \tag{3.12}$$

เมื่อ N คือจำนวนข้อมูลพิกัด เท่ากับ 13 จุด ที่มีความสัมพันธ์กับส่วนเบี่ยงเบนมาตรฐาน σ ซึ่งเป็นค่าที่แสดงความกระจุกตัวของข้อมูลสำหรับประเมินค่าผิดพลาดที่เกิดจากการทดลองซ้ำหลาย ๆ ครั้ง เมื่อ μ_e คือค่าเฉลี่ยของ $d_{e,i}$ ดังสมการที่ (3.13)

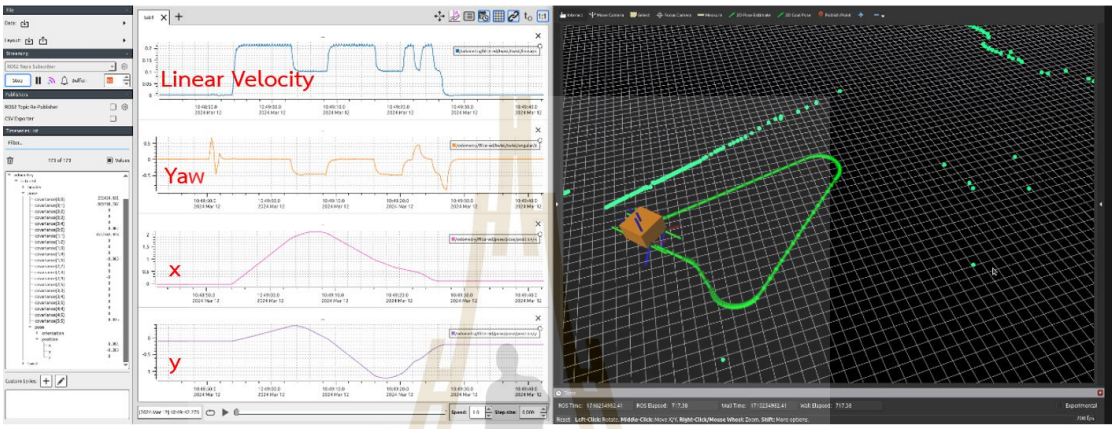
$$\sigma = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (d_{e_i} - \mu_e)^2} \tag{3.13}$$

เมื่อหุ่นยนต์สามารถรับรู้ตำแหน่งของตัวเองได้ใกล้เคียงกับความเป็นจริง โดยใช้เซนเซอร์ที่มี สามารถนำหุ่นยนต์มาทดสอบการเคลื่อนที่และสังเกตสัญญาณป้อนกลับของเซนเซอร์ที่วัดได้ ดังรูปที่ 3.25 – 3.26

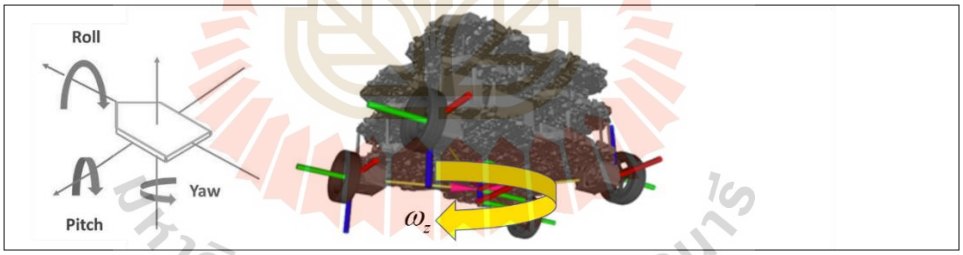


รูปที่ 3.25 ความเร็วเชิงมุมรอบแกน Yaw กับความเร็วเชิงเส้นในการประมาณตำแหน่งของหุ่นยนต์

พิจารณารูปที่ 3.25 สังเกตได้ว่าความเร็วเชิงมุมรอบแกน Roll และ Pitch มีค่าเท่ากับศูนย์เสมอเนื่องจากเป็นความเร็วเชิงมุมที่ไม่นำมาพิจารณาในงานวิจัย จึงทำการปิดการใช้งานรับข้อมูลของ 2 ส่วนนี้ ในขณะที่ข้อมูลการป้อนกลับของเซนเซอร์ ประกอบด้วยความเร็วเชิงเส้นและความเร็วเชิงมุมรอบแกน Yaw ให้ผลลัพธ์ลักษณะการเคลื่อนที่ของหุ่นยนต์ดังรูปที่ 3.26 โดยอธิบายความสัมพันธ์ของความเร็วเชิงมุมรอบแกน Yaw ดังรูปที่ 3.27



รูปที่ 3.26 การระบุตำแหน่งด้วยความเร็วเชิงมุมรอบแกน Yaw กับความเร็วเชิงเส้นสัมพันธ์กับพิกัด

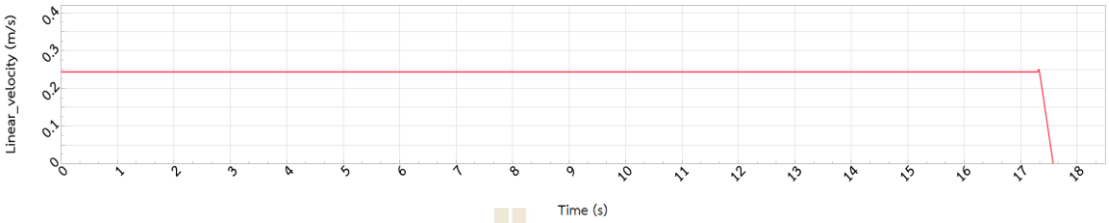


รูปที่ 3.27 ความสัมพันธ์ของความเร็วเชิงมุมรอบแกน Yaw

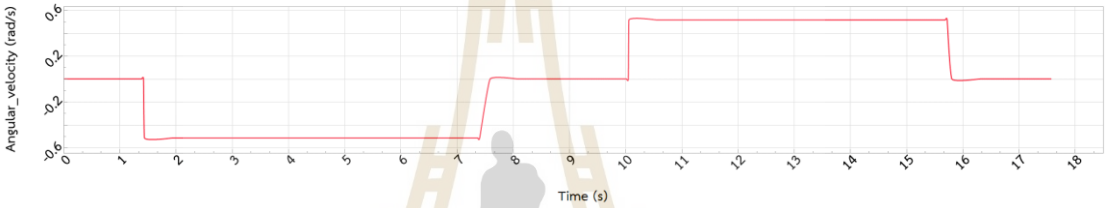
3.5.5 การตรวจสอบความถูกต้องระหว่างระบบจำลองกับหุ่นยนต์จริง

หลักจากที่หุ่นยนต์ในระบบจริงสามารถระบุตำแหน่งบนแผนที่โดยมีความผิดพลาดที่ทำให้ระบบมีความน่าเชื่อถือพอสมควรแล้ว ขั้นตอนนี้เป็นขั้นตอนที่เปรียบเทียบพฤติกรรมการเคลื่อนที่ของหุ่นยนต์ทั้ง 2 ระบบ เพื่อทำให้แน่ใจได้ว่าหุ่นยนต์ในระบบจำลองเมื่อทำงานกับระบบนำทาง NAV2 จะให้ผลลัพธ์พารามิเตอร์ที่เหมาะสมในการติดตามเส้นทางแบบ RPP ที่สามารถนำไปใช้กับหุ่นยนต์จริงแล้วมีพฤติกรรมการเคลื่อนที่แบบเดียวกัน

อาศัยคุณสมบัติของ rosbags ที่สามารถบันทึกข้อมูล Topic ที่กำลังสื่อสารภายในระบบ ROS และสามารถในมาเล่นซ้ำได้ โดยนำส่วนนี้ไปบันทึกคำสั่งควบคุมจากแป้นคีย์บอร์ด cmd_vel โดยมีลักษณะการควบคุมดังรูปที่ 3.28 – 3.29



รูปที่ 3.28 คำสั่ง cmd_vel ในการควบคุม v_t



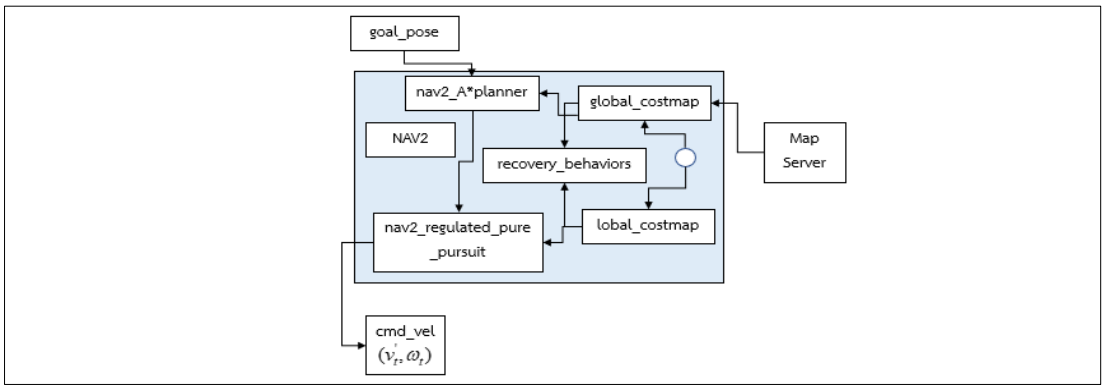
รูปที่ 3.29 คำสั่ง cmd_vel ในการควบคุม ω_t

การทดลองนี้สามารถกำหนดสมมติฐาน คือ เมื่อหุ่นยนต์ทั้ง 2 ระบบ ถูกควบคุมด้วยคำสั่ง cmd_vel แบบเดียวกัน หุ่นยนต์ในระบบจำลองต้องมีพฤติกรรมการเคลื่อนที่ที่ใกล้เคียงกับหุ่นยนต์จริงจึงจะสามารถนำระบบจำลองมาจำลองพารามิเตอร์ติดตามเส้นทางที่เหมาะสมและประยุกต์ใช้กับหุ่นยนต์จริงได้ โดยทำการวิเคราะห์จากความเร็ว v_x และ ω_z ป้อนกลับจากทั้ง 2 ระบบ รวมถึงลักษณะของพิกัดการเคลื่อนที่เมื่อหุ่นยนต์ถูกควบคุมด้วยคำสั่ง cmd_vel ดังกล่าว

3.6 การจำลองระบบนำทางอัตโนมัติเพื่อหาพารามิเตอร์ที่เหมาะสม

การจำลองพฤติกรรมหุ่นยนต์ที่ติดตามเส้นทางแบบ RPP จะสามารถเริ่มการทำงานได้ ต้องอาศัยข้อมูลการสร้างขอบเขตของสิ่งกีดขวางเพื่อทำงานร่วมกับการวางแผนเส้นทาง ในส่วนของ local_costmap และ global_costmap โดยระบบต้องกำหนดจุดหมายปลายทางด้วย goal_pose เป็นข้อมูลพิกัด (x, y, θ)

การทำงานได้อย่างสมบูรณ์ของสถาปัตยกรรมจำเป็นต้องกำหนดพารามิเตอร์ที่เกี่ยวข้องให้ครบถ้วน ประกอบด้วยพารามิเตอร์ Costmap ข้อมูลแผนที่ที่ถูกใช้งานด้วย Map Server และระบบติดตามเส้นทาง nav2_regulated_pure_pursuit โดยเมื่อระบุพารามิเตอร์ทำให้ระบบมีความสมบูรณ์แล้วจะให้เอาต์พุตเป็นการควบคุมหุ่นยนต์อัตโนมัติด้วยคำสั่ง cmd_vel ดังรูปที่ 3.30



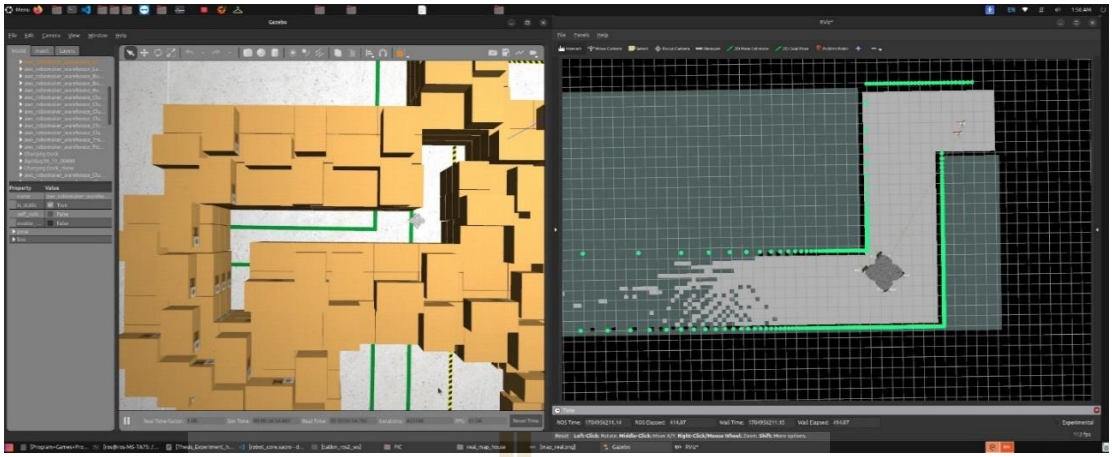
รูปที่ 3.30 สถาปัตยกรรมระบบนำทางอัตโนมัติ NAV2

ขั้นตอนต่อไปนี้เป็นการกำหนดค่าพารามิเตอร์ที่เกี่ยวข้องจากที่ได้กล่าวไปก่อนหน้านี้ โดยเป็นขั้นตอนกำหนดพารามิเตอร์ในส่วนของ Costmap ขอบเขตหุ่นยนต์ (Footprint) และ การสร้างแผนที่ ด้วยวิธีการ SLAM เพื่อเตรียมก่อนการจำลองเพื่อหาค่าพารามิเตอร์ RPP ในการติดตามเส้นทางที่เหมาะสม

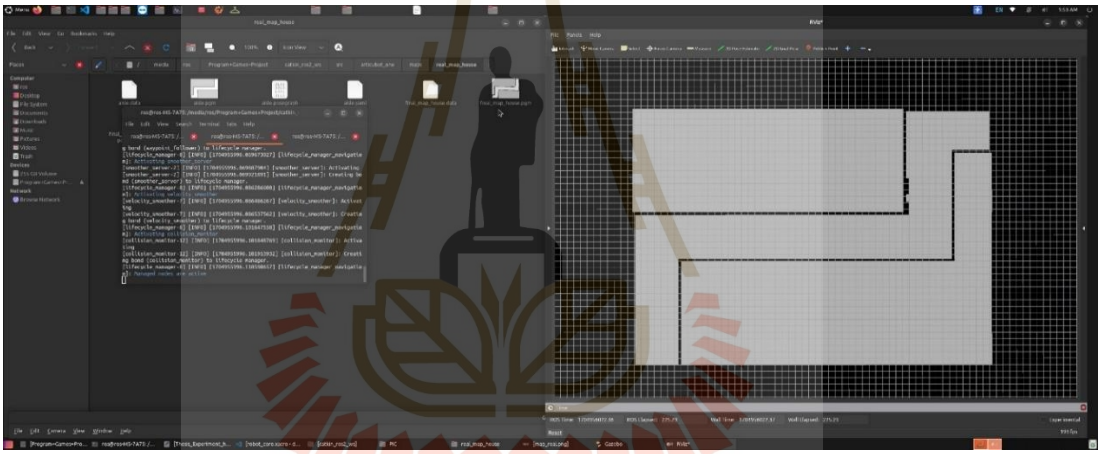
3.6.1 การเตรียมระบบนำทางอัตโนมัติและกำหนดพารามิเตอร์เพื่อสร้างเส้นทาง

จากที่ได้กล่าวไปก่อนหน้านี้ เพื่อให้ Planner Server มีการวางแผนเส้นทางอ้างอิงให้หุ่นยนต์นำทางอัตโนมัติได้อย่างปลอดภัย โดย Costmap นั้น มีข้อมูลอินพุตจาก Occupancy Grid Map ได้จากวิธีการ SLAM และนำผลลัพธ์เข้าระบบ NAV2 ด้วยแพคเกจ Map Server สามารถดำเนินการเตรียมแพคเกจที่เกี่ยวข้องได้ดังขั้นตอนต่อไปนี้

(1) เนื่องจากระบบนำทาง NAV2 ต้องอาศัยในส่วนของ Map Server ซึ่งได้จากการ SLAM แผนที่ทดสอบด้วยแพคเกจ slam_toolbox โดยมีเทคนิคเพื่อให้ได้ผลลัพธ์ที่มีความแม่นยำในการสร้างแผนที่ด้วยระบบจำลองได้ควบคุมหุ่นยนต์ด้วยความเร็วต่ำโดยที่ v_t มีค่าไม่เกิน 0.1 เมตรต่อวินาที และ ω_t ไม่เกิน 0.2 เรเดียนต่อวินาที ให้ผลลัพธ์ดังรูปที่ 3.31 – 3.32

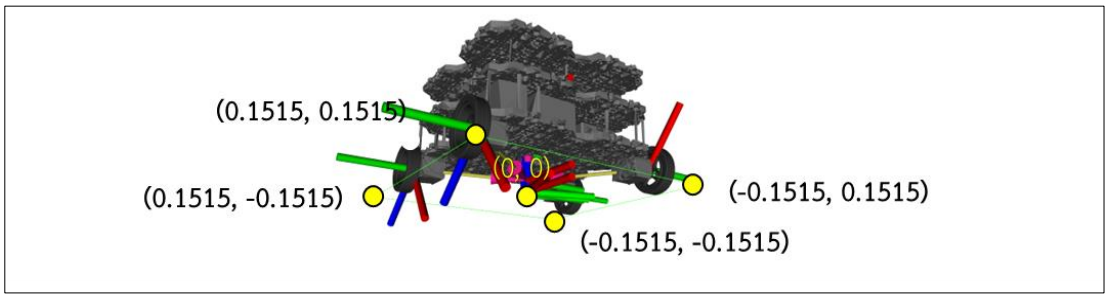


รูปที่ 3.31 การควบคุมหุ่นยนต์ในโปรแกรม Gazebo เพื่อเก็บข้อมูลแผนที่ด้วยวิธีการ SLAM



รูปที่ 3.32 ผลลัพธ์ของแผนที่จากการ SLAM

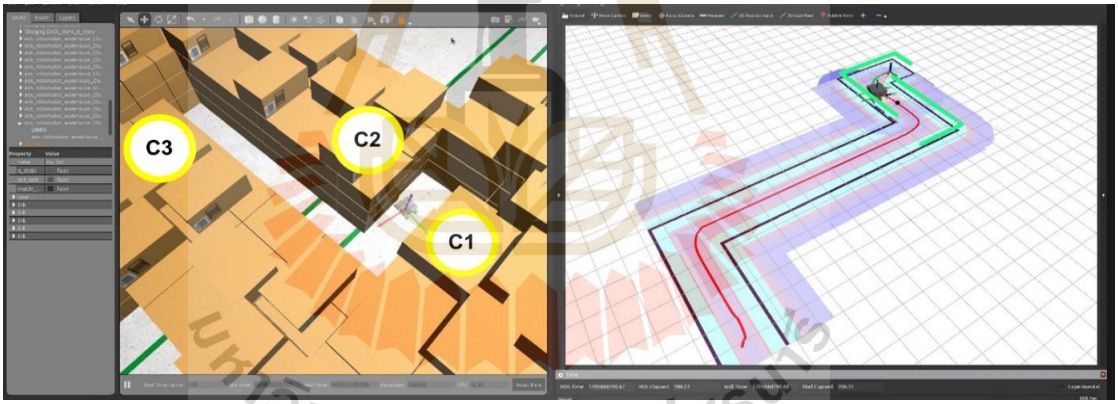
(2) การกำหนดพารามิเตอร์ขอบเขตหุ่นยนต์ (Footprint) เป็นการระบุจุดพิกัดสี่เหลี่ยมล้อมรอบหุ่นยนต์ มีผลต่อการคำนวณ Costmap ระบุความเป็นสิ่งกีดขวางในขวางในระยะ Inscribed Radius พิจารณาจากสภาพแวดล้อมที่หุ่นยนต์สามารถเคลื่อนที่ผ่านทางที่มีความแคบ จึงกำหนดให้พิกัด Footprint มีขนาดเท่ากับหุ่นยนต์แบบพอดี จะได้ขนาดพื้นที่ของขอบเขตเท่ากับ 0.303 ตารางเมตร สัมพันธ์กับพิกัดเมื่อจุดกำเนิดของหุ่นยนต์อยู่ที่จุด (0, 0) ดังรูปที่ 3.33



รูปที่ 3.33 ขอบเขต Footprint ของหุ่นยนต์

(3) พารามิเตอร์ Costmap ประกอบด้วย inflation_radius เท่ากับ 0.55 เมตร และ cost_scaling_factor เท่ากับ 10.0 เพื่อให้ planner สร้างเส้นทางอยู่ตรงกลางระหว่างสิ่งกีดขวาง เพื่อให้หุ่นยนต์อยู่ตรงกลางของเส้นทางระหว่างสิ่งกีดขวาง

(4) กำหนดพิกัดการวางแผนเส้นทางด้วย goal_pose โดยจุดหมายที่ได้กำหนด ซึ่งผลลัพธ์ทั้ง 3 ส่วนนี้ สามารถแสดงผลข้อมูลที่หุ่นยนต์รับรู้ผ่านโปรแกรม RVIZ ดังรูปที่ 3.34



รูปที่ 3.34 ลักษณะการวางแผนเส้นทางอ้างอิงที่อาศัยข้อมูล Costmap

3.6.2 การจำลองพารามิเตอร์ติดตามเส้นทางแบบ PP

ในการออกแบบพารามิเตอร์ในส่วนแรกเป็นการกำหนดเพียงระยะมองไปข้างหน้า L ทำให้ระบบติดตามเส้นทางไม่มีการควบคุมความเร็วเมื่อเคลื่อนที่ผ่านเส้นทางโค้ง จึงทำให้ระบบติดตามเส้นทางในส่วนแรกเป็นการควบคุมแบบ Pure Pursuit โดยสภาพแวดล้อมในการจำลองได้จัดสภาพแวดล้อมดังรูปที่ 3.35 ประกอบไปด้วยเส้นทางโค้ง 3 ส่วน ประกอบด้วย C1, C2 และ C3 กำหนดให้ความผิดพลาดออกนอกเส้นทางโค้ง (Tracking Error, t_e) ต้องมีค่าน้อยกว่า 0.05 เมตร

จากการลดระยะ L โดยการศึกษาพบว่าสามารถกำหนดค่าสูงสุดของระยะ L เป็นระยะที่มากที่สุดที่เป็นค่าเริ่มต้นของกระบวนการจำลอง ดังสมการที่ (3.14)

$$min(R_{track}) = \frac{max(v_x)}{max(\omega_z)} \tag{3.14}$$

จะได้รัศมีความโค้งมากที่สุดเนื่องจากเป็นการบังคับด้วยความเร็วเชิงเส้นเท่ากับ $max(v_x)$ โดยที่หุ่นยนต์พยายามหักเลี้ยวด้วยความเร็วเชิงมุมสูงสุด $max(\omega_z)$ สามารถคำนวณระยะมองไปข้างหน้าที่มากที่สุดด้วยสมการที่ (3.15)

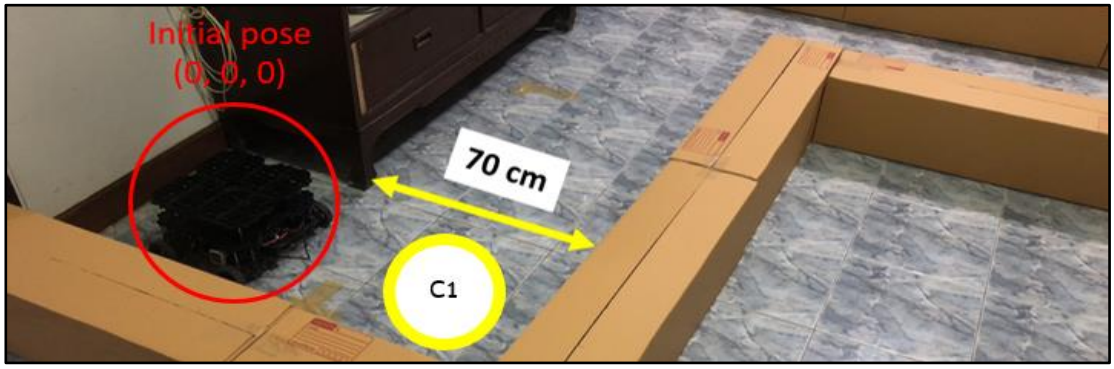
$$L = 2min(R_{track}) \tag{3.15}$$

3.6.3 การจำลองพารามิเตอร์ติดตามเส้นทางแบบ RPP

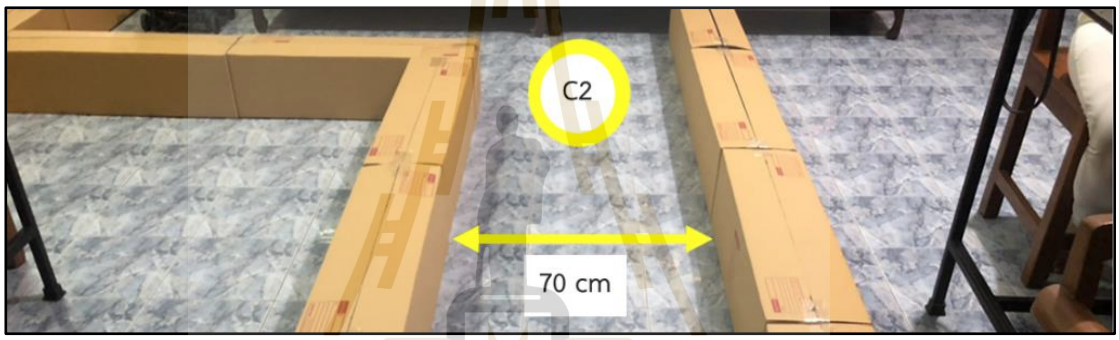
เนื่องจากการควบคุมแบบ PP ไม่สามารถควบคุมความเร็วในการเข้าโค้งทำให้หุ่นยนต์มีโอกาสเคลื่อนที่ออกนอกเส้นทางจากพฤติกรรมดังกล่าว จึงประยุกต์ใช้ระบบติดตามเส้นทางแบบ RPP ที่สามารถกำหนดค่ารัศมีความโค้งขั้นต่ำ r_{min} เป็นเงื่อนไขให้กับฟังก์ชัน Curvature Heuristic ในการควบคุมความเร็วจากการตรวจจับรัศมีเส้นทางโค้งของเส้นทาง โดยใช้สภาพแวดล้อมเดิมในการทดสอบ กำหนดให้ความเร็วเชิงเส้นต่ำสุด v_{min} เท่ากับ 0.1 เมตรต่อวินาที โดยการเปลี่ยนแปลงค่า r_{min} ได้ดำเนินการเพิ่มทีละ 0.2 เมตร ถึง 1.6 เมตร และพิจารณาด้วยค่าผิดพลาดในการติดตามเส้นทางโค้งด้วยตำแหน่ง C1, C2 และ C3 เช่นเดียวกันกับขั้นตอนก่อนหน้านี้

3.6.4 การทดสอบหุ่นยนต์จริงด้วยพารามิเตอร์ RPP

เมื่อระบบจำลองหุ่นยนต์พบพารามิเตอร์ที่สามารถเคลื่อนที่ในสภาพแวดล้อมที่กำหนดได้อย่างปลอดภัยจากการจำลองระบบติดตามเส้นทางแบบ RPP การทดสอบหุ่นยนต์ในสภาพแวดล้อมจริงนั้น ได้จัดสภาพแวดล้อมให้หุ่นยนต์สามารถสร้างแผนที่ได้เช่นเดียวกับหุ่นยนต์ในสภาพแวดล้อมจำลอง ด้วยกล่องไปรษณีย์ขนาด 20x80x20 ลูกบาศก์เซนติเมตร โดยกำหนดพารามิเตอร์แบบเดียวกันกับการจำลองระบบ ดังรูปที่ 3.36 – 3.38



รูปที่ 3.36 สภาพแวดล้อมในการทดสอบหุ่นยนต์เคลื่อนที่ผ่านเส้นทางโค้ง C1

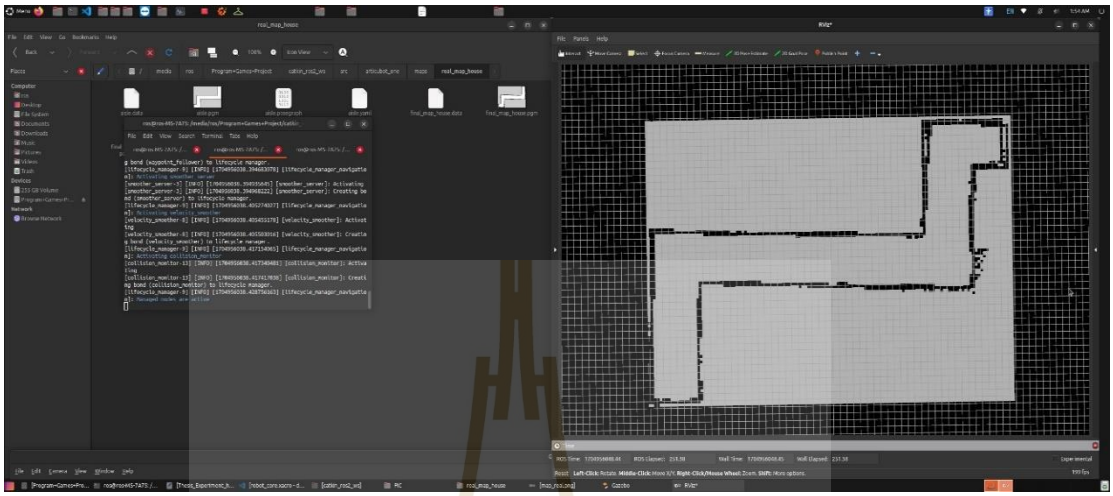


รูปที่ 3.37 สภาพแวดล้อมในการทดสอบหุ่นยนต์เคลื่อนที่ผ่านเส้นทางโค้ง C2

รูปที่ 3.38 สภาพแวดล้อมในการทดสอบหุ่นยนต์เคลื่อนที่ผ่านเส้นทางโค้ง C3

จากสภาพแวดล้อมที่กำหนด ผู้วิจัยสามารถควบคุมหุ่นยนต์ให้มีการเคลื่อนที่ตามหลักจลนศาสตร์ในการเก็บข้อมูลแผนที่ด้วยวิธีการ SLAM เช่นเดียวกับการ SLAM ในโปรแกรม

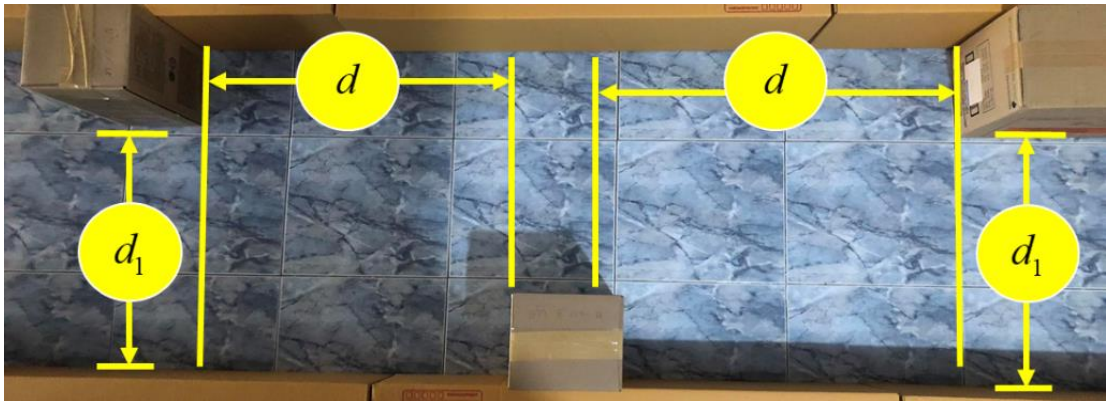
Gazebo ด้วยความเร็วต่ำเพื่อความแม่นยำของผลลัพธ์ โดยที่ v_t มีค่าไม่เกิน 0.1 เมตรต่อวินาที และ ω_t ไม่เกิน 0.2 เรเดียนต่อวินาที ให้ผลลัพธ์ของดังรูปที่ 3.39



รูปที่ 3.39 ผลลัพธ์จากการ SLAM ของหุ่นยนต์จริงเพื่อเตรียมทดสอบพารามิเตอร์

3.6.5 การทดสอบการหลบหลีกสิ่งกีดขวางอัตโนมัติด้วยพารามิเตอร์ RPP

เมื่อพารามิเตอร์ที่มีความเหมาะสมกับค่าความผิดพลาดออกนอกเส้นทางมีค่าที่เหมาะสมพอซึ่งเป็นผลมาจากการจำลองด้วยโปรแกรม Gazebo การทดสอบการหลบหลีกสิ่งกีดขวางอัตโนมัติก็เป็นอีกหนึ่งระบบที่สำคัญในการพิสูจน์ความสามารถในการวางแผนเส้นทางได้อย่างยืดหยุ่น ดำเนินการโดยการเพิ่มสิ่งกีดขวางที่ไม่เป็นส่วนหนึ่งของข้อมูล Occupancy Grid Map ที่เป็นผลลัพธ์จากการ SLAM ดังนั้น ข้อมูลแผนที่ในขั้นตอนนี้จะเป็นข้อมูลของแผนที่เช่นเดียวกับรูปที่ 3.39 โดยการลดระยะค่า d ให้มีค่าที่น้อยลงครั้งละ 0.1 เมตร เป็นการลดรัศมีการเลี้ยวโค้งของหุ่นยนต์ด้วยค่าเริ่มต้นเท่ากับ 0.7 เมตร เมื่อระยะ d_1 มีค่าคงที่เท่ากับ 0.52 เมตร โดยขั้นตอนนี้สามารถพิสูจน์ความสามารถในการทำซ้ำของระบบเมื่อพบขีดจำกัดระยะ d ที่น้อยที่สุดโดยทดลองให้หุ่นยนต์ทำงานแบบเดิมทั้งหมด 3 ครั้ง ในการหลบหลีกสิ่งกีดขวางอัตโนมัติ ดังรูปที่ 3.40



รูปที่ 3.40 สภาพแวดล้อมทดสอบการหลบหลีกสิ่งกีดขวางอัตโนมัติ

บทที่ 4
ผลการวิจัยและวิเคราะห์ผล

4.1 ผลการออกแบบขนาดต้นกำลัง

จากการคำนวณแรงบิดของมอเตอร์เพื่อเป็นตัวขับของล้อด้านซ้าย τ_R และด้านขวา τ_L พบว่ามอเตอร์ที่ใช้งานควรแรงบิดมากกว่า 2.59 นิวตันเมตร เมื่อพิจารณามอเตอร์ DYNAMIXEL ซีรี่ย์ X พบว่า ข้อมูลจำเพาะรุ่น XL-430 W250T มีคุณสมบัติที่เพียงพอต่อการใช้งานในเงื่อนไขที่กำหนด ด้วยแรงบิดสูงสุดส่งผลให้ τ_R และ τ_L เท่ากับ 2.8 นิวตันเมตร ซึ่งมีค่ามากกว่า 2.59 ถึง 8% ด้วยอัตราเกียร์ทด 258.5 ต่อ 1 จึงได้เลือกมอเตอร์รุ่นดังกล่าวเป็นตัวขับต้นกำลังในงานวิจัย

4.2 พารามิเตอร์สำหรับควบคุมหุ่นยนต์

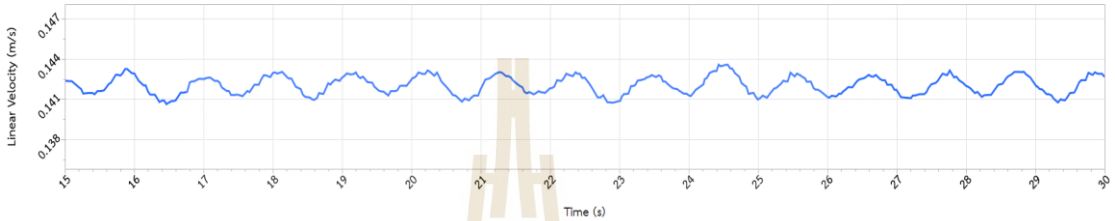
จากขั้นตอนการคำนวณพารามิเตอร์ที่เกี่ยวข้องเพื่อควบคุมหุ่นยนต์ให้สามารถเคลื่อนที่ได้ถูกต้องตามทฤษฎีจลนศาสตร์ Skid Steering ได้กำหนดความเร็วเชิงมุมสูงสุด $max(\omega_z)$ มีค่าเท่ากับ 1.0 เรเดียนต่อวินาที และคำนวณค่าพารามิเตอร์เกี่ยวข้องที่ทำให้หุ่นยนต์สามารถเคลื่อนที่สอดคล้องกับจลนศาสตร์ด้วยปลั๊กอิน diff_drive_controller และระบบฟิสิกส์ที่ใช้ประมวลผลในโปรแกรม Gazebo อ้างอิงจากการศึกษาในงานวิจัย ดังตารางที่ 4.1

ตารางที่ 4.1 สรุปพารามิเตอร์สำหรับจำลองหุ่นยนต์อัตโนมัติ

พารามิเตอร์ควบคุมหุ่นยนต์และระบบฟิสิกส์	
ความเร็วสูงสุดของล้อจากคุณสมบัติผู้ผลิต N	60.64 รอบต่อนาที
ความเร็วสูงสุดของล้อ $max(\omega_l), max(\omega_r)$	6.35 เรเดียนต่อวินาที
ระยะห่างล้อซ้ายและขวา B	0.267 เมตร
ขนาดรัศมีล้อ r	0.033 เมตร
ความเร็วเชิงเส้นสูงสุด $max(v_x)$	0.21 เมตรต่อวินาที
ความเร็วเชิงเส้นต่ำสุด $min(v_x)$	-0.21 เมตรต่อวินาที
ความเร็วเชิงมุมสูงสุด $max(\omega_z)$	1.0 เรเดียนต่อวินาที
ความเร็วเชิงมุมต่ำสุด $min(\omega_z)$	-1.0 เรเดียนต่อวินาที
สัมประสิทธิ์ระยะห่างล้อซ้ายและขวา χ	1.573
สัมประสิทธิ์แรงเสียดทานคูลอมบ์ μ	0.8
ความเร่งโน้มถ่วง g	9.81 เมตรต่อวินาที^{2}

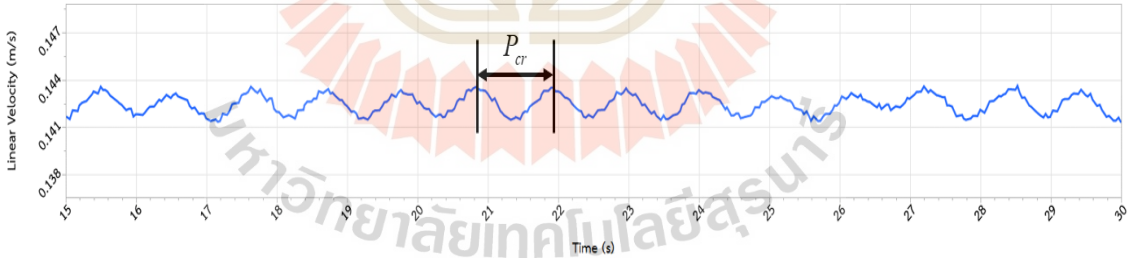
4.3 ผลการออกแบบตัวควบคุมด้วยทฤษฎีซีกเลอร์-นิโคล

ผลตอบสนองที่แม่นยำต่อสัญญาณควบคุมอินพุตที่ถูกควบคุมด้วย cmd_vel เป็นส่วนที่สำคัญในการติดตามเส้นทาง จากขั้นตอนการปรับค่าตัวควบคุมอย่างเหมาะสม เมื่อพิจารณาการเคลื่อนที่ของหุ่นยนต์และสังเกตสัญญาณความเร็วเชิงเส้น v_x ป้อนกลับ พบว่าเริ่มมีแนวโน้มการเกิดเสถียรภาพแบบขอบ เมื่อ K_p' เท่ากับ 1190 ในช่วงเวลาระหว่าง 15 ถึง 30 วินาที ดังรูปที่ 4.1



รูปที่ 4.1 ผลตอบสนองความเร็ว v_x เมื่อ K_p' เท่ากับ 1190

จากผลการทดลองพบว่าผลตอบสนองความเร็วโดยเฉลี่ยเท่ากับ 0.1425 เมตรต่อวินาที และช่วง 24 ถึง 25 วินาที มีช่วงสูงสุดสูงกว่าช่วงเวลาอื่น เมื่ออ้างอิงด้วยกริดของตารางแนวแกน y ช่วง 0.141 ถึง 0.144 เมตรต่อวินาที จึงได้เพิ่มค่า K_p' เท่ากับ 1200 ให้ผลลัพธ์ดังรูปที่ 4.2



รูปที่ 4.2 ผลตอบสนองสัญญาณเมื่อ K_p' เท่ากับ 1200

เมื่อ K_p' เท่ากับ 1200 พบว่า ผลตอบสนองมีค่าใกล้เคียงกันในทุก ช่วงเวลา และไม่มีผลตอบสนองในช่วงเวลาใดมีช่วงสูงสุดสูงกว่าช่วงเวลาอื่น อ้างอิงด้วยกริดของตารางตามแนวแกน y ช่วง 0.141 ถึง 0.144 เมตรต่อวินาที โดยคาบเวลาการเกิดเสถียรภาพแบบขอบ P_{cr} มีค่าเท่ากับ 1.06 วินาที เป็นคาบที่พิจารณาในช่วงเวลา 20.9 ถึง 21.96 วินาที เมื่อนำไปคำนวณออกแบบพารามิเตอร์ควบคุมแบบ พีไอ ได้ผลลัพธ์ ดังตารางที่ 4.2

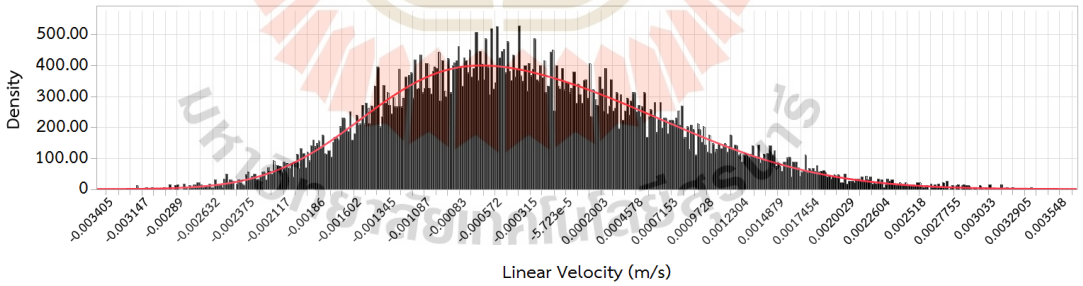
ตารางที่ 4.2 ผลการออกแบบตัวควบคุม พีไอ

ชนิดของตัวควบคุม	พารามิเตอร์ควบคุม		
	K_p	K_i	K_d
PI	$0.45K_{cr}$	$\frac{1.2K_p}{P_{cr}}$	0
การคำนวณ	0.45×1200	$\frac{1.2\times540}{(21.96-20.9)}$	0
ผลลัพธ์	540	611	0
แปลงค่าตามทฤษฎีควบคุม	4.218	0.00982	0

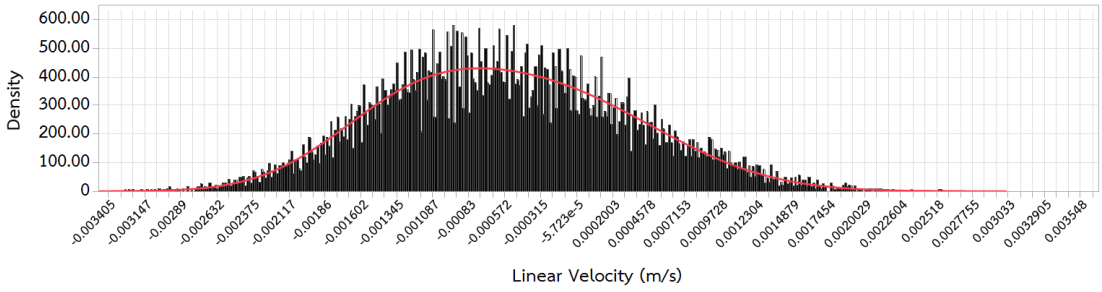
4.4 ผลการบูรณาการข้อมูลเซนเซอร์ด้วยตัวกรองคาลมานแบบขยาย

จากการพิจารณาเมทริกซ์ความแปรปรวนจากการวัด R ในทางปฏิบัติได้ทำการสมมติให้ความผิดพลาดแบบสุ่มมีลักษณะการแจกแจงแบบเกาส์เซียน หาได้จากการวางหุ่นยนต์ไว้นิ่ง ๆ ทั้งหมด 3 ครั้งและหาค่าเฉลี่ย โดยแกน x คือ ค่าความหนาแน่นหมายถึงจำนวนข้อมูลที่มีค่าเท่ากับแกน x มีผลลัพธ์ในแต่ละครั้งดังนี้

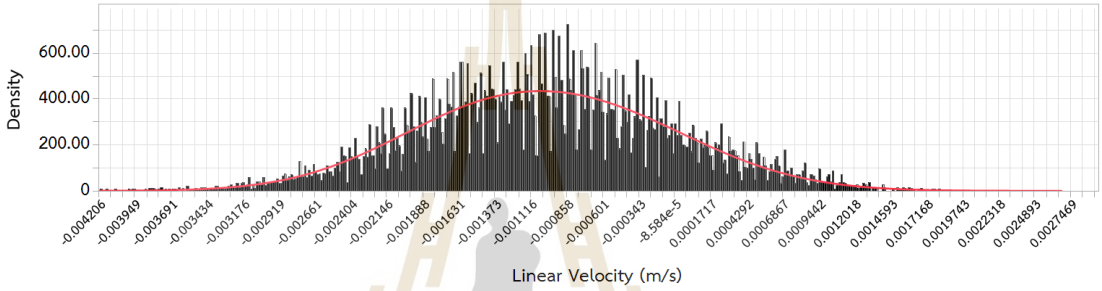
1) ผลการแจกแจงปกติของความเร็วเชิงเส้น v_x ที่วัดด้วยเซนเซอร์ Encoder ในช่วงเวลา 20 นาที ทั้งหมด 10 ครั้ง มีผลลัพธ์ดังรูปที่ 4.3 – 4.12



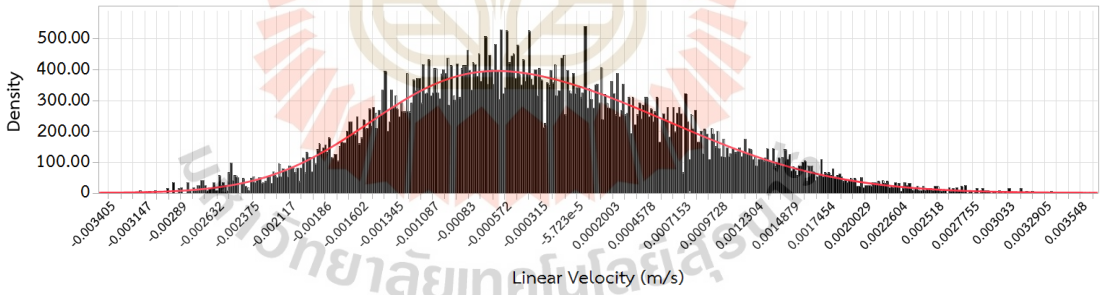
รูปที่ 4.3 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 1



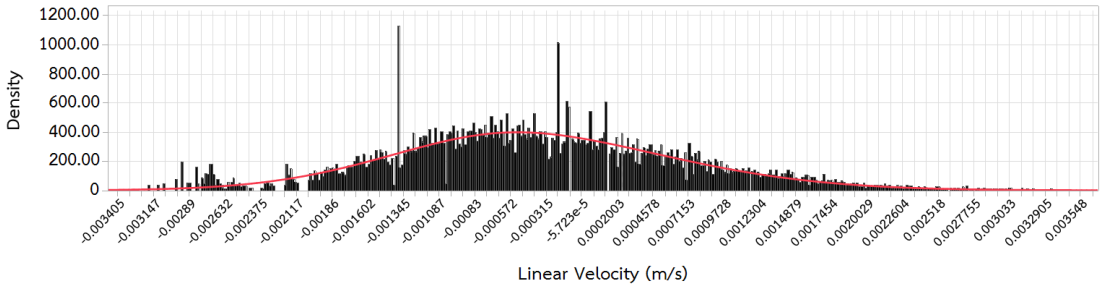
รูปที่ 4.4 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 2



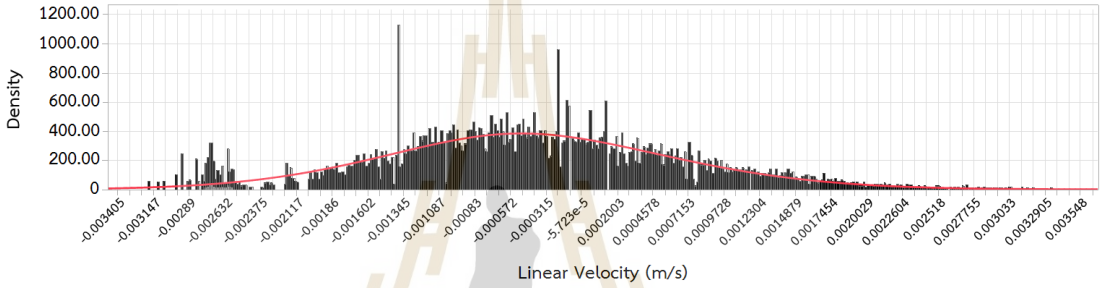
รูปที่ 4.5 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 3



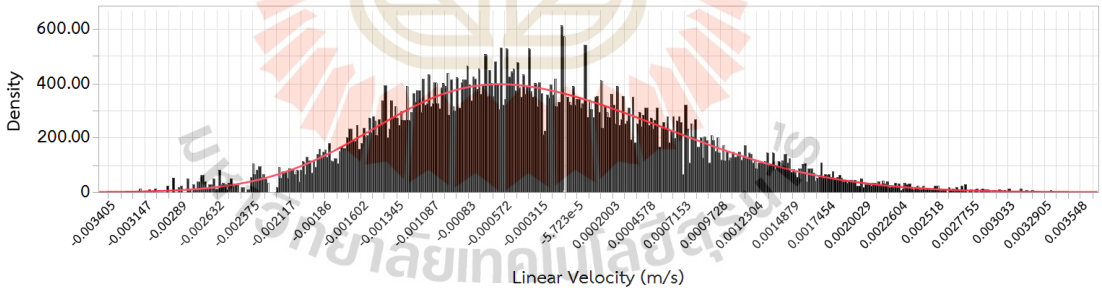
รูปที่ 4.6 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 4



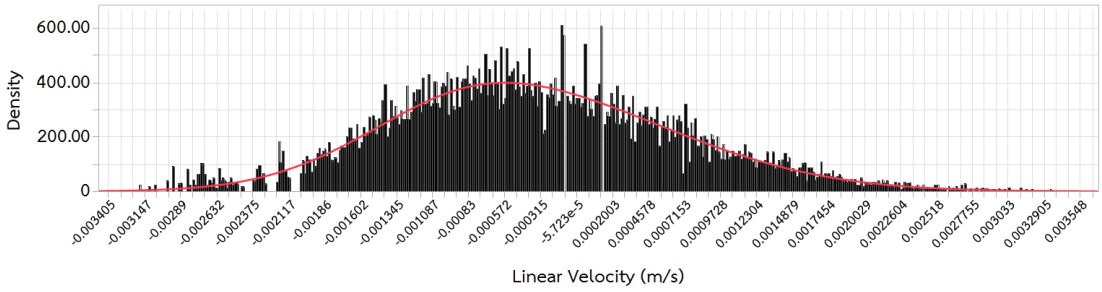
รูปที่ 4.7 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 5



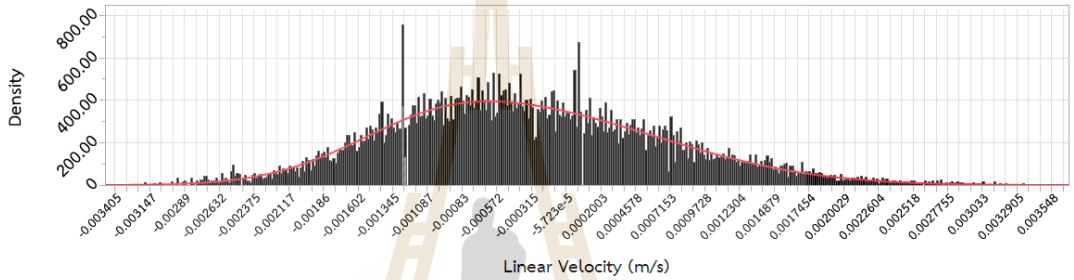
รูปที่ 4.8 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 6



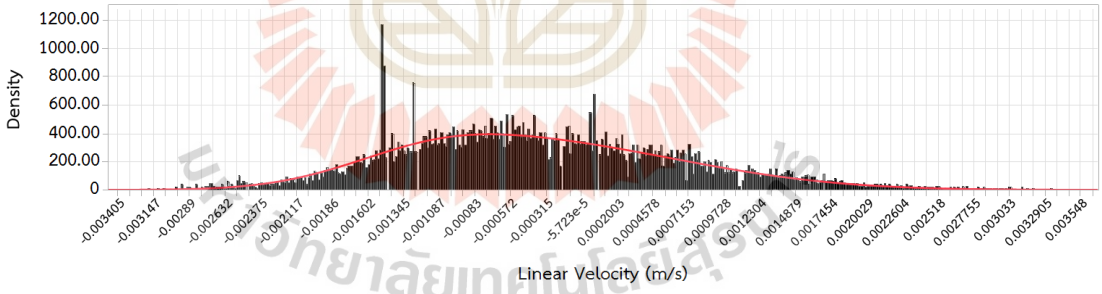
รูปที่ 4.9 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 7



รูปที่ 4.10 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 8



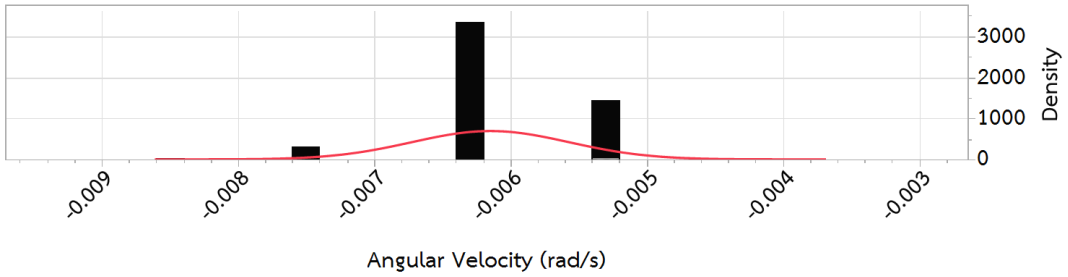
รูปที่ 4.11 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 9



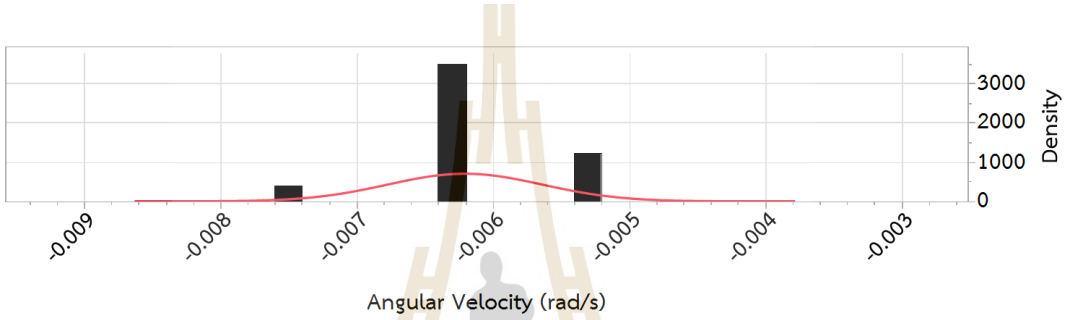
รูปที่ 4.12 การแจกแจงแบบปกติของ v_x จากการทดสอบ ครั้งที่ 10

จากการวัดค่า v_x ขณะหยุดนิ่งทั้งหมด 10 ครั้ง พบว่า เซนเซอร์มีความแปรปรวนที่ถูกรบกวนภายในอุปกรณ์ที่ใช้วัด มีการแจกแจงสูงสุดอยู่ในช่วง -0.0046 และ 0.00355 เมตรต่อวินาที สามารถนำความแปรปวนที่เกิดขึ้นหาค่าเฉลี่ยและกำหนดในเมทริกซ์ $\boldsymbol{R}$ ในขั้นตอนถัดไป

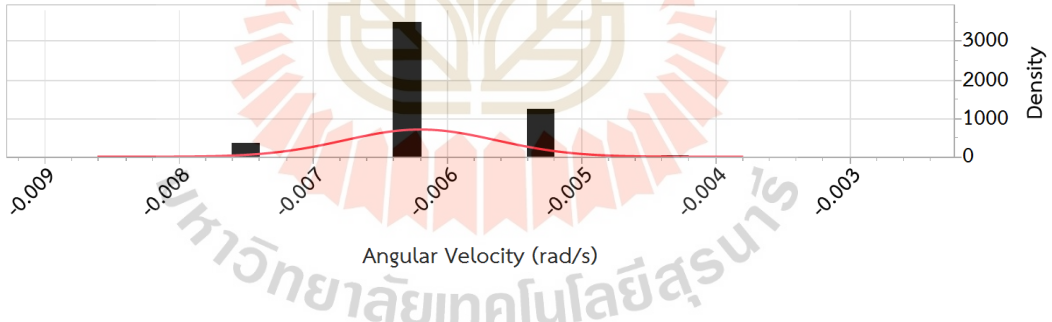
2) ผลการแจกแจงปกติของความเร็วเชิงมุม ω_z ที่วัดด้วยเซนเซอร์ ไจโรสโคป ในช่วงเวลา 20 นาที ทั้งหมด 10 ครั้ง มีผลลัพธ์ดังรูปที่ 4.13 – 4.22



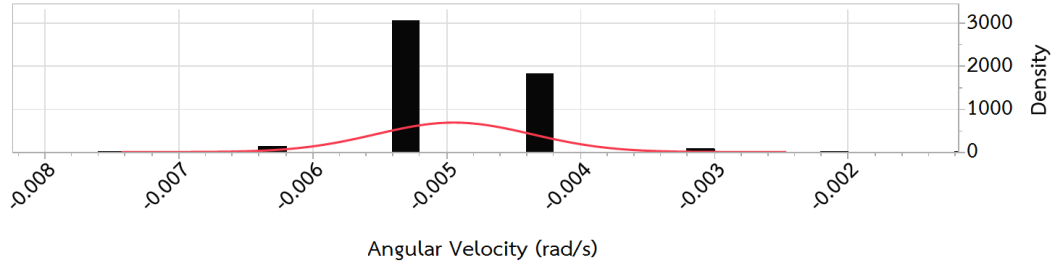
รูปที่ 4.13 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 1



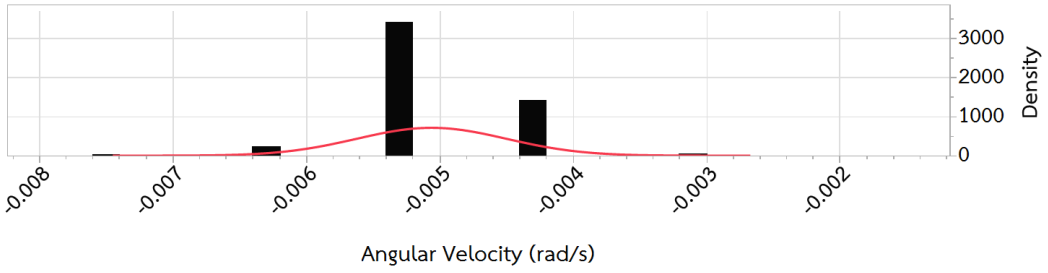
รูปที่ 4.14 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 2



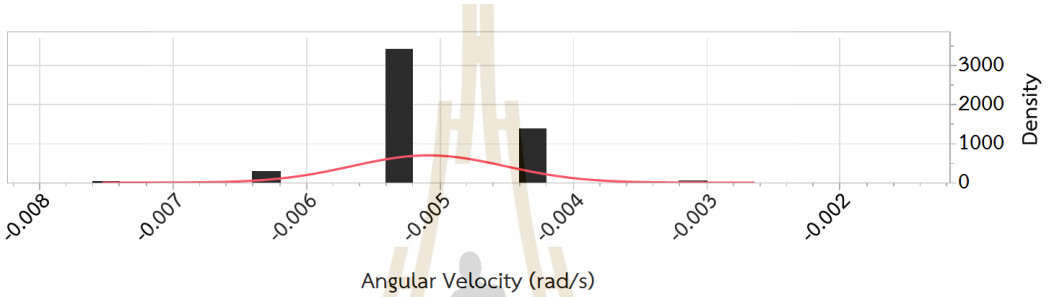
รูปที่ 4.15 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 3



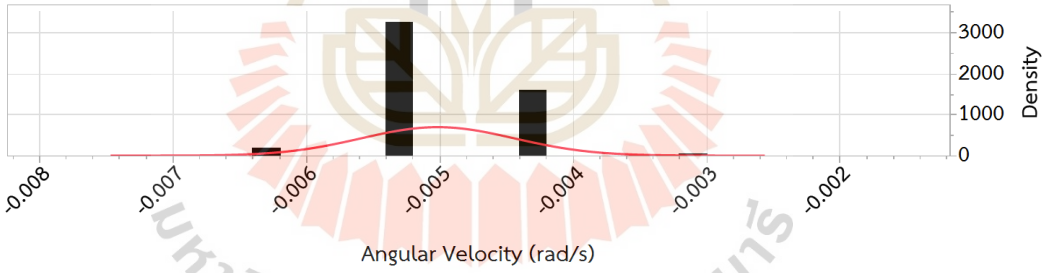
รูปที่ 4.16 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 4



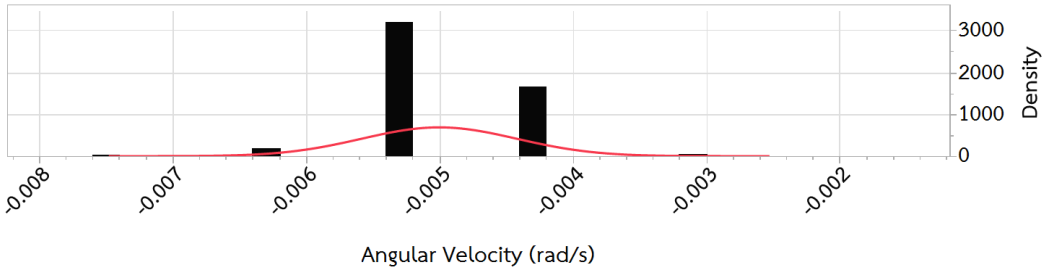
รูปที่ 4.17 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 5



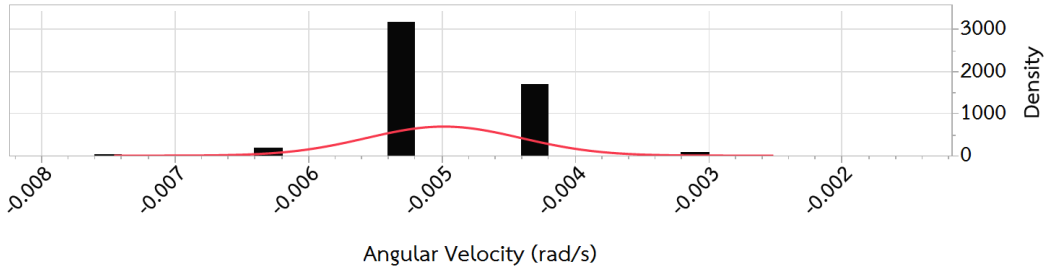
รูปที่ 4.18 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 6



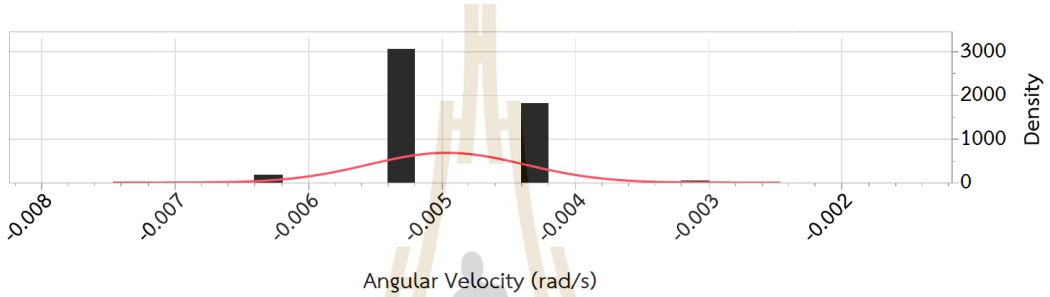
รูปที่ 4.19 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 7



รูปที่ 4.20 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 8



รูปที่ 4.21 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 9



รูปที่ 4.22 การแจกแจงแบบปกติของ ω_z จากการทดสอบ ครั้งที่ 10

จากการวัดค่า ω_z ขณะหยุดนิ่งทั้งหมด 10 ครั้ง พบว่าเซนเซอร์มีความแปรปรวนที่ถูกรบกวนภายในอุปกรณ์ โดยค่าที่มีการแจกแจงสูงสุดอยู่ในช่วง -0.004 และ -0.009 (เรเดียนต่อวินาที)^2 สามารถนำความแปรปวนที่เกิดขึ้นหาค่าเฉลี่ยและกำหนดในเมทริกซ์ R เพื่อเป็นอินพุตของอัลกอริทึม ในส่วนของการวัดความเร็วเชิงมุมด้วยเซนเซอร์ไจโรสโคป โดยสามารถสรุปความแปรปรวนทั้ง 2 ส่วน ดังตารางที่ 4.3

ตารางที่ 4.3 ผลการคำนวณค่าความแปรปรวนเฉลี่ยจากการวัด 5 ครั้ง (รอบที่ 1)

ความแปรปรวน	ค่าความแปรปรวนจากการวัด (เมตรต่อวินาที)^2				
	ครั้งที่ 1	ครั้งที่ 2	ครั้งที่ 3	ครั้งที่ 4	ครั้งที่ 5
$\sigma_{v_x}^{\ 2}$	$9.80\times e^{-7}$	$7.68\times e^{-7}$	$7.99\times e^{-7}$	$8.22\times e^{-7}$	$8.34\times e^{-7}$
$\sigma_{\omega_z}^{\ 2}$	$3.34\times e^{-7}$	$3.23\times e^{-7}$	$3.19\times e^{-7}$	$3.39\times e^{-7}$	$3.14\times e^{-7}$

ตารางที่ 4.4 ผลการคำนวณค่าความแปรปรวนเฉลี่ยจากการวัด 5 ครั้ง (รอบที่ 2)

ความแปรปรวน	ค่าความแปรปรวนจากการวัด (เมตรต่อวินาที)^2					
	ครั้งที่ 6	ครั้งที่ 7	ครั้งที่ 8	ครั้งที่ 9	ครั้งที่ 10	ค่าเฉลี่ย
$\sigma_{v_x}^{\ 2}$	$8.43\times e^{-7}$	$8.35\times e^{-7}$	$8.15\times e^{-7}$	$8.15\times e^{-7}$	$8.04\times e^{-7}$	$8.32\times e^{-7}$
$\sigma_{\omega_z}^{\ 2}$	$3.29\times e^{-7}$	$3.30\times e^{-7}$	$3.37\times e^{-7}$	$3.36\times e^{-7}$	$3.44\times e^{-7}$	$3.31\times e^{-7}$

หลังจากนั้นสามารถกำหนดพารามิเตอร์ในเมทริกซ์ R ด้วยค่าความแปรปรวนเฉลี่ย v_x และ ω_z โดยสมมติให้แบบจำลองมีความถูกต้องด้วยการกำหนดค่าเมทริกซ์ความแปรปรวนของกระบวน Q เท่ากับ 1e^{-9} ยกเว้นในส่วนของค่าประมาณ v_x และ ω_z ดังสมการที่ (4.1) – (4.2) โดยกำหนดเป็นเริ่มต้นในการปรับค่าความแปรปรวน

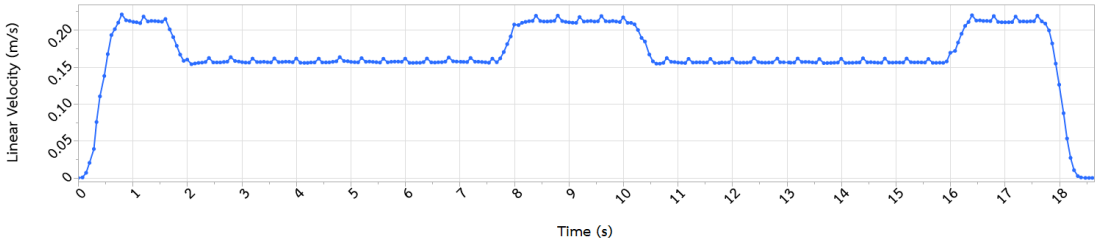
$$R=\begin{bmatrix} 8.32e^{-7} & 0 \\ 0 & 3.31e^{-7} \end{bmatrix} \tag{4.1}$$

$$Q=\begin{bmatrix} 1e^{-9} & 0 & 0 & 0 \\ 0 & 1e^{-9} & 0 & 0 \\ 0 & 0 & 8.32e^{-7} & 0 \\ 0 & 0 & 0 & 3.31e^{-7} \end{bmatrix} \tag{4.2}$$

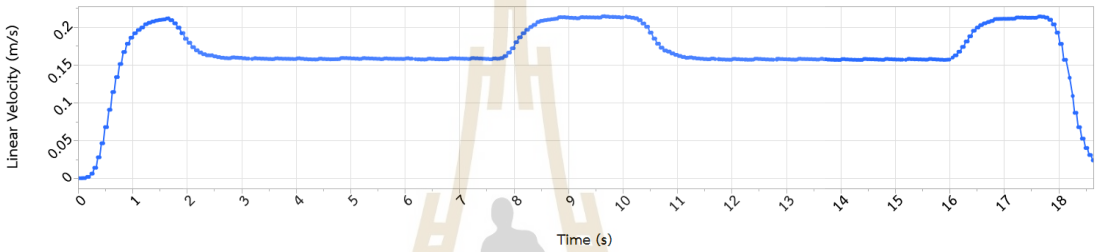
4.4.1 การทดสอบพารามิเตอร์ตัวกรองคาลมาน

จากเมทริกซ์ค่าความแปรปรวนกระบวนการสามารเพิ่มประสิทธิภาพในการกรองสัญญาณโดยการลดค่าความแปรปรวนดังกล่าวตามทฤษฎี สังเกตจากการกำหนดคุณลักษณะควบคุมหุ่นยนต์ด้วยคำสั่ง cmd_vel เมื่อหุ่นยนต์เริ่มเคลื่อนที่ออกจากจุดเริ่มต้น สามารถเปรียบเทียบสัญญาณป้อนกลับจากเซนเซอร์ได้ดังนี้

1) ผลการเปรียบเทียบ v_x ที่วัดได้ด้วยเซนเซอร์ Encoder ระหว่างสัญญาณที่ผ่านตัวกรองคาลมานกับการไม่ผ่านตัวกรองคาลมาน แสดงในรูปที่ 4.23 - 4.24 ตามลำดับ ในกรณีที่เมทริกซ์ Q เท่ากับเมทริกซ์ R เป็นค่าเมทริกซ์เช่นเดียวกับสมการที่ (4.1) และ (4.2)



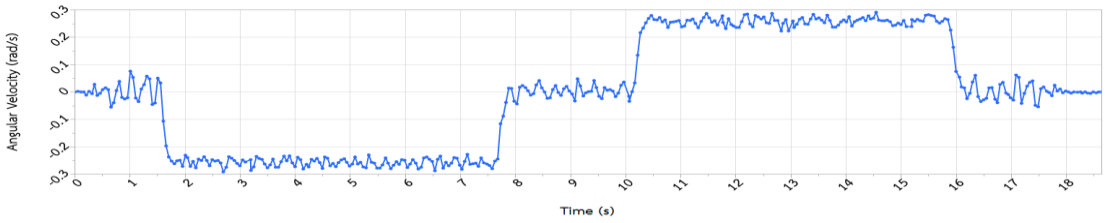
รูปที่ 4.23 สัญญาณ v_x ที่ไม่ผ่านตัวกรองคาลมาน



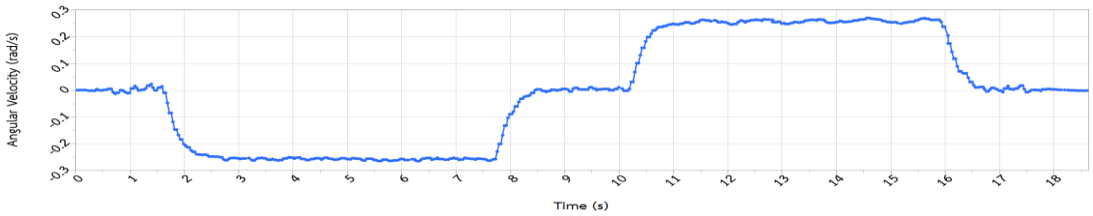
รูปที่ 4.24 สัญญาณ v_x เมื่อเมทริกซ์ Q เท่ากับ R

จากผลการทดลองพบว่า การกำหนดค่าเมทริกซ์ Q ให้มีค่าเท่ากับเมทริกซ์ R สามารถกรองสัญญาณที่มีค่าความแปรปรวนโดยเฉลี่ยเท่ากับ $8.31e^{-7}$ (เมตรต่อวินาที)^2 ได้อย่างสมบูรณ์ จึงไม่จำเป็นในการปรับลดค่าเมทริกซ์ความแปรปรวนในส่วนของ σ_x^2 เพื่อลดความไวต่อสัญญาณจากการวัดความเร็วเชิงมุม

2) ผลการเปรียบเทียบ ω_z ที่วัดได้ด้วยเซนเซอร์ไจโรสโคป ระหว่างสัญญาณที่ผ่านตัวกรองคาลมานกับสัญญาณที่ไม่ผ่านตัวกรองคาลมาน แสดงในรูปที่ 4.25 - 4.26 ตามลำดับ เมื่อเมทริกซ์ Q เท่ากับเมทริกซ์ R

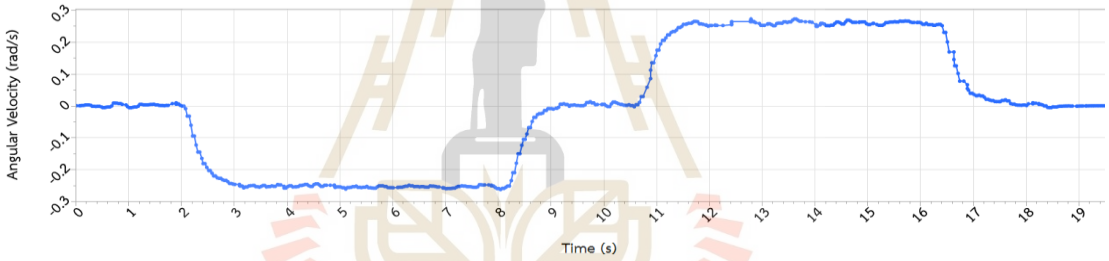


รูปที่ 4.25 สัญญาณ ω_z ที่ไม่ผ่านตัวกรองคาลมาน



รูปที่ 4.26 สัญญาณ ω_z จากตัวกรองคาลมานเมื่อเมทริกซ์ Q เท่ากับ R

พิจารณาความเร็วเชิงมุมที่ป้อนกลับ พบว่า การกำหนดค่าความแปรปรวนตัวกรองคาลมานโดยที่ Q เท่ากับ R พบว่า ยังปรากฏสัญญาณรบกวนเล็กน้อยในช่วงเวลา 0.5 - 1 วินาที และช่วงเวลา 16.5 – 17.5 วินาที จึงปรับลดค่าเมทริกซ์ Q ในส่วนของ σ_y^2 เพื่อให้การประมาณสถานะตอบสนองต่อการเปลี่ยนแปลงต่อข้อมูลการวัดได้ช้าลง ดังนั้น กำหนดให้ σ_y^2 มีค่าลดลง $0.5e^{-7}$ ทำให้ σ_y^2 มีค่าเท่ากับ $2.81e^{-7}$ หลังจากนั้น ทำการวัดอีกครั้ง ให้ผลลัพธ์แสดงในรูปที่ 4.27



รูปที่ 4.27 สัญญาณความเร็วเชิงมุมเมื่อ σ_y^2 เท่ากับ $2.81e^{-7}$

จากการปรับลดค่าเมทริกซ์ความแปรปรวนในส่วนของ σ_y^2 พบว่า ตัวกรองคาลมานสามารถรับมือกับสัญญาณรบกวนที่มีความแปรปรวน $3.31e^{-7}$ (เมตรต่อวินาที)^2 และมีประสิทธิภาพของสัญญาณที่ดียิ่งขึ้น ทำให้สามารถสรุปได้ว่าอัลกอริทึมทำงานได้ถูกต้องตามทฤษฎี จากการปรับลดค่าเมทริกซ์ Q จึงพิจารณาค่าความแปรปรวนดังกล่าวเป็นพารามิเตอร์ที่เหมาะสม โดยไม่ปรับให้น้อยกว่านี้เพราะส่งผลให้สัญญาณมีการลู่เข้าคำตอบจากการประมาณช้า ทำให้ประมาณตำแหน่งการเคลื่อนของหุ่นยนต์มีความผิดพลาดที่สูงขึ้น จึงได้สรุปพารามิเตอร์เมทริกซ์ความแปรปรวนในสมการที่ (4.3) – (4.4) เป็นเมทริกซ์ที่จะนำไปใช้ในการทดลองขั้นถัดไป

$$R = \begin{bmatrix} 8.49e^{-7} & 0 \\ 0 & 3.31e^{-7} \end{bmatrix} \tag{4.3}$$

$$Q = \begin{bmatrix} 1e^{-9} & 0 & 0 & 0 \\ 0 & 1e^{-9} & 0 & 0 \\ 0 & 0 & 8.32e^{-7} & 0 \\ 0 & 0 & 0 & 2.81e^{-7} \end{bmatrix} \quad (4.4)$$

4.4.2 ความผิดพลาดสัมบูรณ์ของเซนเซอร์ไจโรสโคป

จากการใช้ระบบสอบเทียบอัตโนมัติที่วิเคราะห์แล้วว่าให้ผลลัพธ์ที่แม่นยำในการวัดความเร็วเชิงมุมขณะไม่เคลื่อนไหว หลังจากที่ได้กรองสัญญาณด้วยตัวกรองคาลมานแล้ว จึงได้นำเซนเซอร์มาหาค่าผิดพลาดสัมบูรณ์พิจารณาความเหมาะสมต่อการใช้งานกับการระบุตำแหน่ง มีผลการทดสอบดังตารางที่ 4.5 – 4.7

ตารางที่ 4.5 ความผิดพลาดมุมองศาของเซนเซอร์ไจโรสโคปที่ความเร็วเชิงมุม 0.25 เรเดียนต่อวินาที

จำนวนครั้งทดสอบ	**องศาการหมุนของหุ่นยนต์ (องศา) ด้วยความเร็วเชิงมุม 0.25 เรเดียนต่อวินาที**				
มุมอ้างอิง	0	90	180	270	360
ครั้งที่ 1	0.000	85.420	178.090	264.140	353.170
ครั้งที่ 2	0.000	88.438	175.750	264.690	353.290
ครั้งที่ 3	0.000	88.400	176.760	265.320	352.850
ค่าเฉลี่ย	0.000	87.419	176.867	264.717	353.103
ค่าผิดพลาดสัมบูรณ์	0.000	2.581	3.133	5.283	6.897

ตารางที่ 4.6 ความผิดพลาดมุมองศาของเซนเซอร์ไจโรสโคปที่ความเร็วเชิงมุม 0.5 เรเดียนต่อวินาที

จำนวนครั้งทดสอบ	**องศาการหมุนของหุ่นยนต์ (องศา) ด้วยความเร็วเชิงมุม 0.5 เรเดียนต่อวินาที**				
มุมอ้างอิง	0	90	180	270	360
ครั้งที่ 1	0.000	88.240	177.480	267.600	356.940
ครั้งที่ 2	0.000	88.380	179.530	267.300	357.135
ครั้งที่ 3	0.000	89.470	178.700	266.600	355.912
ค่าเฉลี่ย	0.000	88.697	178.570	267.167	356.662
ค่าผิดพลาดสัมบูรณ์	0.000	1.303	1.430	2.833	3.338

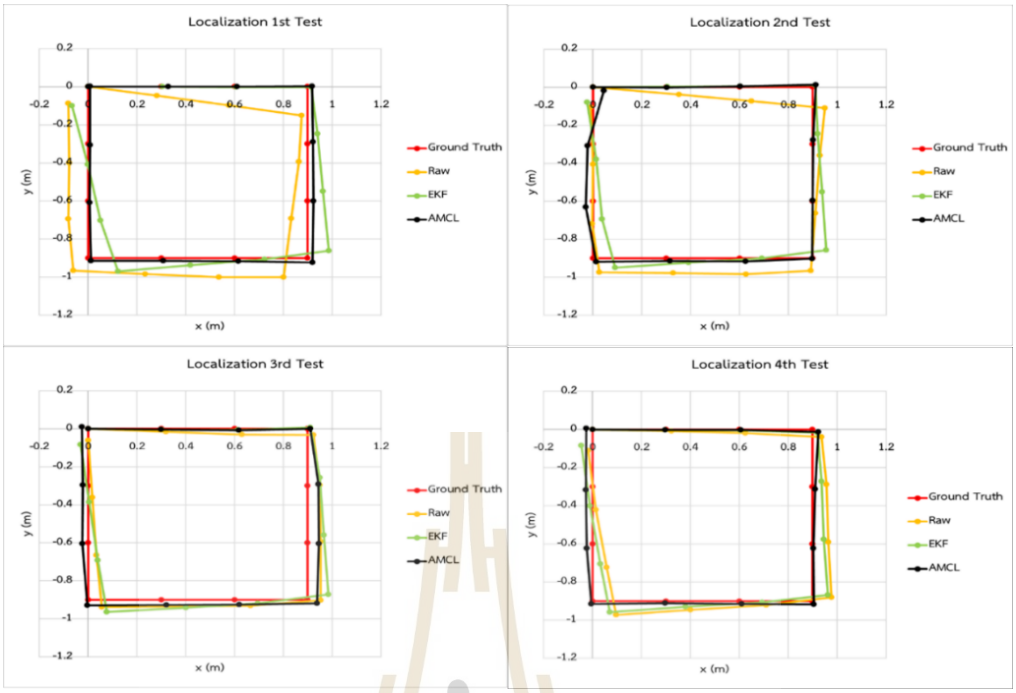
ตารางที่ 4.7 ความผิดพลาดมุมองศาของเซนเซอร์ไจโรสโคปที่ความเร็วเชิงมุม 1.0 เรเดียนต่อวินาที

จำนวนครั้งทดสอบ	**องศาการหมุนของหุ่นยนต์ (องศา) ด้วยความเร็วเชิงมุม 1.0 เรเดียนต่อวินาที**				
มุมอ้างอิง	0	90	180	270	360
ครั้งที่ 1	0.000	90.670	179.300	269.850	361.100
ครั้งที่ 2	0.000	91.185	179.860	269.530	360.749
ครั้งที่ 3	0.000	89.550	179.570	271.180	360.940
ค่าเฉลี่ย	0.000	90.468	179.577	270.187	360.930
ค่าผิดพลาดสัมบูรณ์	0.000	0.468	0.423	0.187	0.930

จากผลการทดสอบพบว่า ความผิดพลาดสัมบูรณ์องศาของ IMU มีค่าสูงสุดเมื่อควบคุม ω_z ให้หมุนด้วยความเร็ว 0.25 เรเดียนต่อวินาที เท่ากับ 6.897 องศา ซึ่งเป็นค่าที่มีความผิดพลาดมากกว่าการควบคุมด้วย ω_z เท่ากับ 1.0 เรเดียนต่อวินาที โดยสามารถนำไปพิจารณาความสามารถในการระบุตำแหน่งบนแผนที่ในการทดลองถัดไป

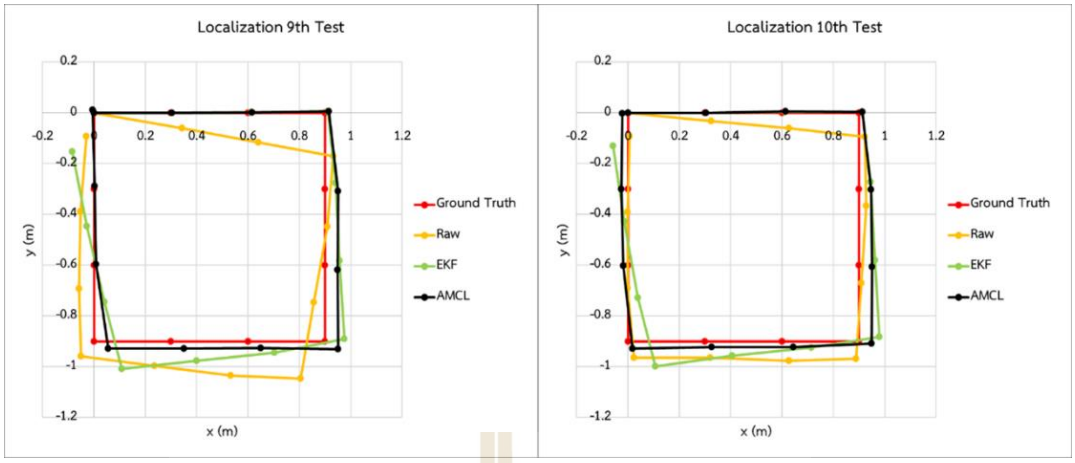
4.5 ผลการระบุตำแหน่งของหุ่นยนต์จริง

การวิเคราะห์ความแม่นยำในการระบุตำแหน่ง ได้เลือกสภาพแวดล้อมทดสอบเพื่อพิจารณาค่าผิดพลาดที่ยอมรับได้ต่อความผิดพลาดของมุม Yaw ด้วยสี่เหลี่ยมจัตุรัส โดยหุ่นยนต์ต้องหมุน 90 องศา ณ มุม ทั้ง 4 ซึ่งการทดลองนี้สามารถประเมินด้วยค่าผิดพลาดด้วยระยะห่างพิกัดจริง $d_{e,i}$ โดยผลการเคลื่อนที่ของหุ่นยนต์แสดงได้ในรูปที่ 4.28 – 4.30 ตามลำดับ



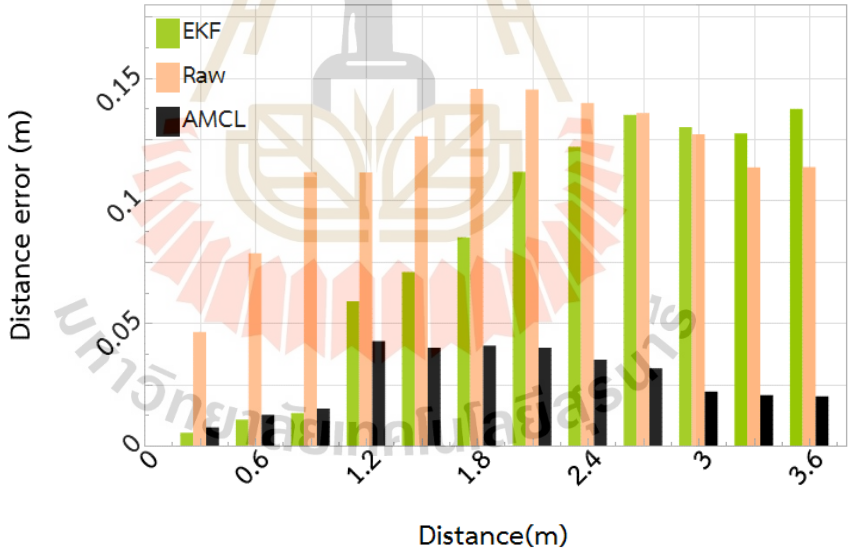
รูปที่ 4.28 ผลเปรียบเทียบพิกัดจากการระบุตำแหน่ง ครั้งที่ 1-4

รูปที่ 4.29 ผลเปรียบเทียบพิกัดการระบุตำแหน่ง ครั้งที่ 5-8



รูปที่ 4.30 ผลเปรียบเทียบพิกัดการระบุตำแหน่ง ครั้งที่ 9-10

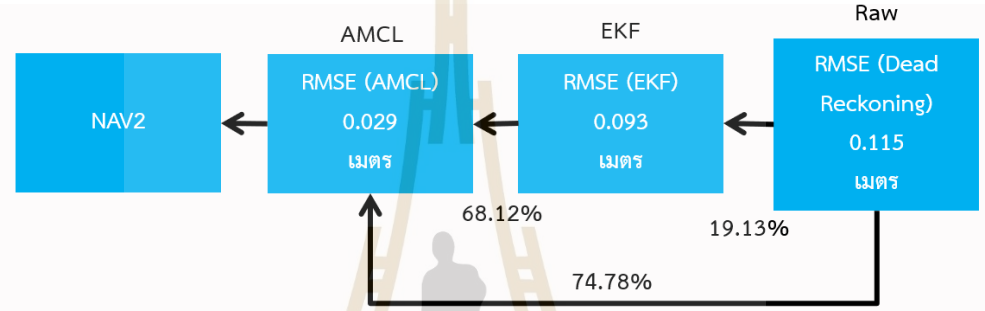
จากการทดสอบ สามารถหาค่าผิดพลาดของระยะพิกัด $d_{e,i}$ ที่เป็นค่าเฉลี่ยจากการทดสอบ 10 ครั้ง ดังรูปที่ 4.31



รูปที่ 4.31 ระยะความผิดผลาดเฉลี่ยจากการทดลอง 10 ครั้ง เทียบกับตำแหน่งจริง

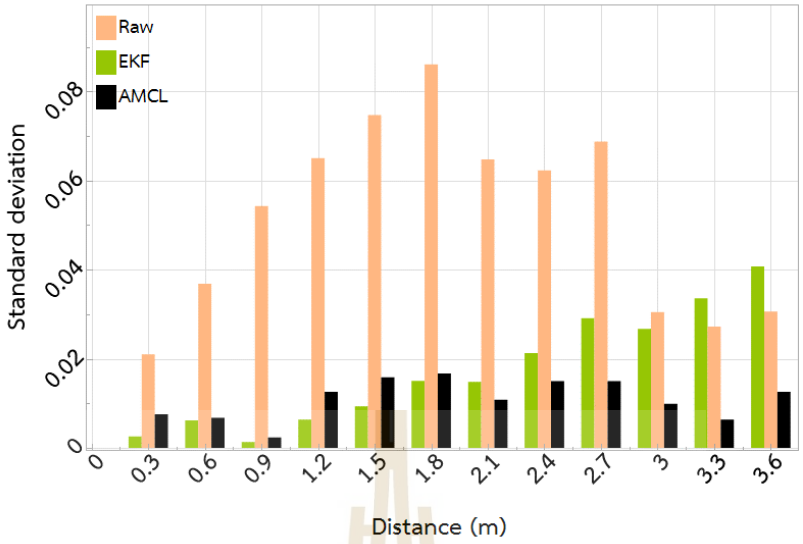
เมื่อพิจารณาการเคลื่อนที่ผ่านพิกัดทั้งหมดพบว่า AMCL สามารถรับมือต่อการใช้งานได้ดีที่สุด ด้วยค่า RMSE เท่ากับ 0.029 เมตร โดยมีความผิดพลาดสูงสุดเท่ากับ 0.043 เมตร ในขณะที่ EKF มีค่า RMSE เท่ากับ 0.093 เมตร มีค่าผิดพลาดสูงสุด 0.137 เมตร เป็นการระบุตำแหน่งที่สามารถลดความผิดพลาดของพิกัดจาก Raw ที่เป็นการระบุตำแหน่งด้วยวิธีการ Dead Reckoning

โดยมีค่า RMSE เท่ากับ 0.115 เมตร และค่าผิดพลาดสูงสุด 0.145 เมตร นั่นหมายความว่า EKF สามารถลด RMSE ของพิกัดจาก Raw ได้ 19.13% และ AMCL สามารถลด RMSE ของพิกัดจาก EKF ได้ 68.12% ทำให้เห็นว่าการระบุตำแหน่งแบบ AMCL มีความน่าเชื่อถือต่อการใช้งานในระยะยาวมากที่สุดและมีค่าผิดพลาดสูงสุดไม่เกิน 0.043 เมตร ทำให้การระบุตำแหน่งแบบ AMCL เหมาะสมต่อการนำไปใช้งานกับระบบนำทางอัตโนมัติในการหลบหลีกสิ่งกีดขวางที่มีพื้นที่แคบได้อย่างเหมาะสม ดังนั้น จึงกำหนดให้พิกัดจากการระบุตำแหน่งด้วยวิธี AMCL เป็นการระบุตำแหน่งที่นำไปใช้กับระบบนำทางอัตโนมัติ NAV2 โดยมีแผนผัง ดังรูปที่ 4.32



รูปที่ 4.32 แผนภาพลดความผิดพลาดจากการระบุตำแหน่งด้วยวิธี AMCL

การวิเคราะห์ประสิทธิภาพในการทำซ้ำของระบบก็เป็นการประเมินประสิทธิภาพได้อีกในรูปแบบหนึ่ง นั่นคือการหาส่วนเบี่ยงเบนมาตรฐาน (standard deviation, σ) เพื่อแสดงความเที่ยงจากการทดลองซ้ำหลาย ๆ ครั้ง ว่าระยะความผิดพลาด $d_{e,i}$ ณ พิกัดแต่ละจุด จากการทดลอง 10 ครั้ง มีค่าต่างจากค่าเฉลี่ยความผิดพลาดอยู่เท่าไหร่ ได้ผลลัพธ์ดังรูปที่ 4.33



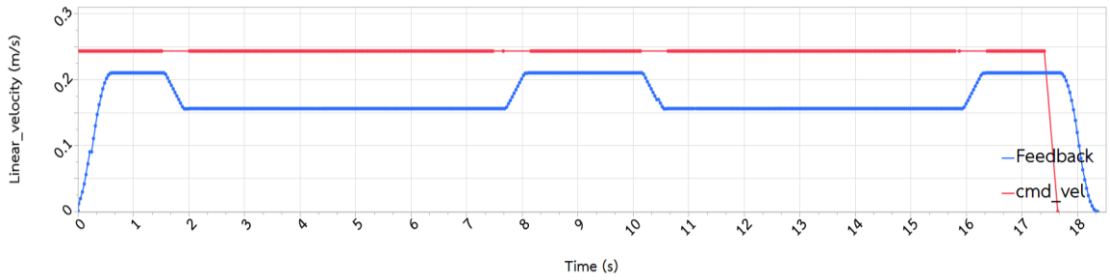
รูปที่ 4.33 ส่วนเบี่ยงเบนมาตรฐานการเคลื่อนที่ผ่านพิกัด 13 จุด ทั้งหมด 10 ครั้ง

จากผลการทดลองพบว่า AMCL นอกจากจะให้ค่า RMSE น้อยที่สุดรวมถึงยังสามารถรับมือกับค่าส่วนเบี่ยงเบนมาตรฐาน (σ) ให้ไม่เกิน 0.02 เมตร โดยมีค่าสูงสุดเท่ากับ 0.017 เมตร เมื่อเทียบกับ EKF มีค่า σ สูงสุดเท่ากับ 0.041 เมตร และ Raw เท่ากับ 0.086 เมตร ตามลำดับ

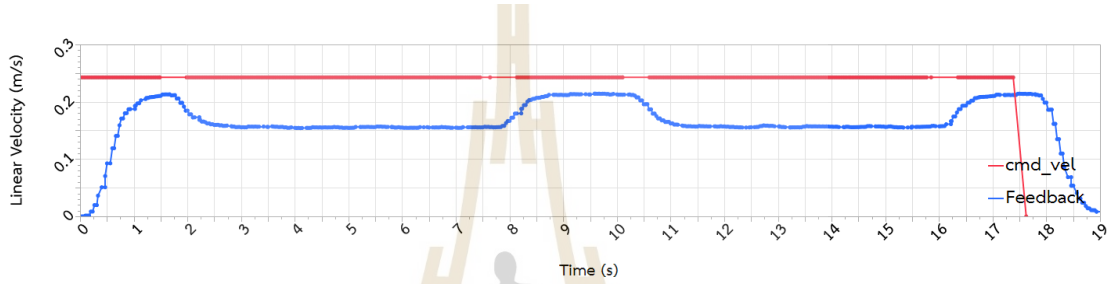
4.6 การตรวจสอบความถูกต้องด้วยแบบจำลองจลนศาสตร์

เมื่อหุ่นยนต์จริงและระบบจำลองสามารถระบุตำแหน่งได้อย่างถูกต้องจากการประมาณตำแหน่งด้วย AMCL การใช้ประโยชน์ระบบจำลองหุ่นยนต์ด้วย Gazebo นั้น จำเป็นต่อการตรวจสอบความแตกต่างของพฤติกรรมจากการควบคุม เปรียบเทียบกับหุ่นยนต์จริง เพื่อแน่ใจได้ว่าสามารถนำระบบมาจำลองการนำทางอัตโนมัติที่สามารถนำไปใช้งานในสภาพแวดล้อมจริงได้

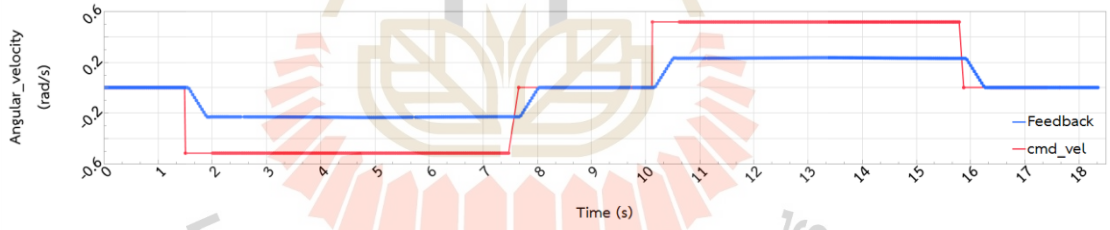
การควบคุมหุ่นยนต์ด้วย v_t และ ω_t ซึ่งเป็นคำสั่ง cmd_vel เช่นเดียวกับการทดสอบการปรับค่าเมทริกซ์ตัวกรองคาลมาน เมื่อหุ่นยนต์เริ่มเคลื่อนที่ออกจากจุดเริ่มต้นตามลักษณะควบคุม rosbags สามารถวัดสัญญาณป้อนกลับ v_x และ ω_z ให้ผลลัพธ์ดังรูปที่ 4.34 – 4.37 ตามลำดับ



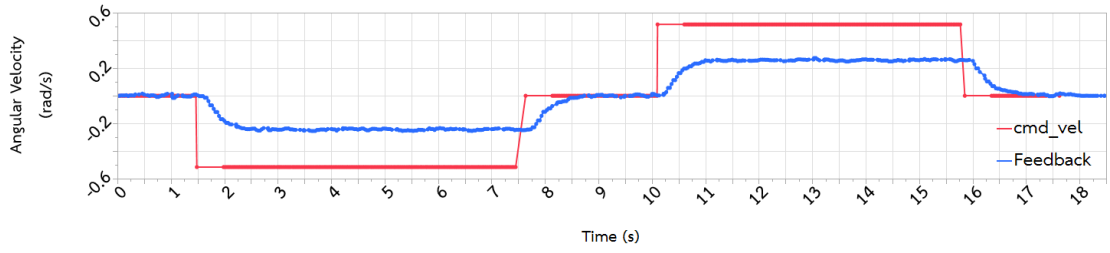
รูปที่ 4.34 สัญญาณป้อนกลับ v_x (จำลอง)



รูปที่ 4.35 สัญญาณป้อนกลับ v_x (จริง)



รูปที่ 4.36 สัญญาณป้อนกลับ ω_z (จำลอง)



รูปที่ 4.37 สัญญาณป้อนกลับ ω_z (จริง)

จากการเปรียบเทียบสัญญาณป้อนกลับ v_x และ ω_z สังเกตได้ว่าความไวต่อสัญญาณควบคุมของหุ่นยนต์จริงตอบสนองช้ากว่าระบบจำลองประมาณ 0.25 วินาที เนื่องมาจากพารามิเตอร์ตัวกรองคาลมานที่ทำให้สัญญาณป้อนกลับมีความไวต่อสัญญาณช้าลงรวมถึงการสื่อสารผ่านเครือข่ายไร้สายด้วย SSH แต่ถึงย่างไรเมื่อพิจารณาสัญญาณ cmd_vel ยังสามารถทำงานภายใต้ rosbags อย่างถูกต้อง และมีลักษณะของสัญญาณใกล้เคียงกันทั้ง 2 ระบบ จากการเปรียบเทียบกัน

พิจารณาระยะเวลาในช่วง 3 – 7 วินาที จากสมมติฐาน คือ เอาต์พุตควรมีค่าใกล้เคียงกับอินพุตภายใต้ $max(v_x)$ ที่มีค่าเท่ากับ 0.21 เมตรต่อวินาที เมื่อสังเกตผลลัพธ์พบว่า v_x มีค่าลดลงในช่วงเวลาดังกล่าว สามารถพิจารณาสาเหตุของพฤติกรรมด้วย สมการที่ (4.5)

$$v_x = \frac{\omega_l r + \omega_r r}{2} \tag{4.5}$$

โดยที่ช่วงเวลาดังกล่าวคำสั่ง cmd_vel มีการบังคับเลี้ยวด้วย ω_t ทำให้หุ่นยนต์เคลื่อนที่และเกิดความเร็วเชิงมุม ω_z เป็นสัญญาณป้อนกลับที่วัดออกมาได้ ดังสมการที่ (4.6)

$$\omega_z = \frac{-\omega_l r + \omega_r r}{\chi B} \tag{4.6}$$

เมื่อ ω_z คือ ความเร็วเชิงมุมป้อนกลับ χ เท่ากับ 1.573 และ r เท่ากับ 0.033 เมตร โดยสามารถพิจารณาการคำนวณออกเป็น 2 กรณี ดังต่อไปนี้สมมติว่าหุ่นยนต์สามารถเคลื่อนที่ได้ตามคำสั่งของ cmd_vel จึงกำหนดให้ ω_z เท่ากับ ω_t เท่ากับ -0.516 เรเดียนต่อวินาที และ v_x เท่ากับ 0.21 เมตรต่อวินาที จะได้ว่าแบบจำลองสามารถคำนวณความเร็วของล้อด้วยการแก้สมการ 2 สมการ 2 ตัวแปร ได้ผลลัพธ์ความเร็วล้อซ้าย ω_l เท่ากับ 9.65 เรเดียนต่อวินาทีส่วนล้อขวา ω_r เท่ากับ 3.08 เรเดียนต่อวินาที จะเห็นได้ว่าความเร็วที่ใช้แทนค่า v_x และ ω_z ตามสมมติฐานมีความเร็วล้อมากกว่าความเร็วสูงสุดเมื่อเทียบกับคุณสมบัติของมอเตอร์ ที่สามารถหมุนได้สูงสุดเพียง 6.35 เรเดียนต่อวินาที จึงสรุปได้ว่าหากต้องการให้ผลความเร็วป้อนกลับมีค่าใกล้เคียงกับ cmd_vel ต้องเปลี่ยนมอเตอร์ที่สามารถทำความเร็วสูงสุดเท่ากับ 9.65 เรเดียนต่อวินาที หรือ 92.15 รอบต่อนาที จึงจะได้ผลลัพธ์ตามสมมติฐาน

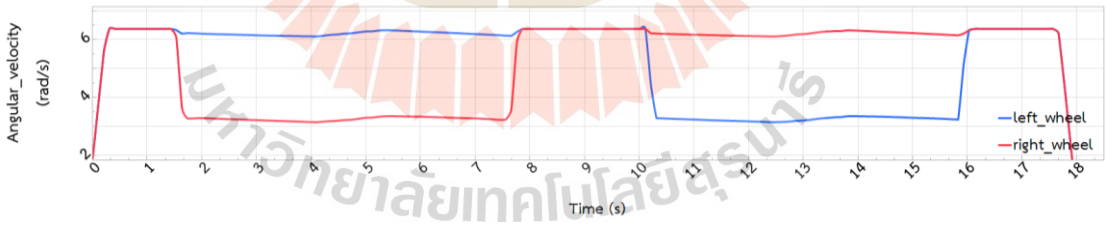
(1) กรณีที่สอง จากกรณีแรกการคำนวณตามสมมติฐานจะเห็นได้ว่าความเร็วล้อมีค่ามากกว่าความเป็นจริงดังนั้นกรณีนี้จึงกำหนดให้ ω_l เท่ากับความเร็วสูงสุด คือ 6.35 เรเดียนต่อวินาที และคำนวณย้อนกลับในสมการที่ (4.5) และ (4.6) จะได้ v_x เท่ากับ 0.156 เมตรต่อวินาที โดย ω_z

เท่ากับ -0.257 เรเดียนต่อวินาที สามารถนำค่าที่ได้จากการคำนวณเปรียบเทียบกับระบบจำลองและระบบจริง ซึ่งเป็นค่าเฉลี่ยในช่วงเวลาที่ 3 – 7 วินาที ดังตารางที่ 4.8

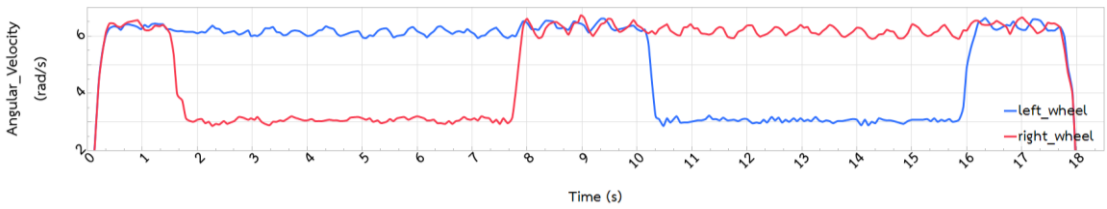
ตารางที่ 4.8 ผลการเปรียบเทียบผลจากการคำนวณเปรียบเทียบกับระบบจำลองและระบบจริง

สัญญาณความเร็วอินพุต (cmd_vel)			
ความเร็ว	**จลนศาสตร์**	**ระบบจำลอง**	**ระบบจริง**
v_t (เมตรต่อวินาที)	0.21	0.21	0.21
ω_t (เรเดียนต่อวินาที)	-0.516	-0.516	-0.516
สัญญาณความเร็วป้อนกลับ			
ความเร็ว	**จลนศาสตร์**	**ระบบจำลอง**	**ระบบจริง**
v_x (เมตรต่อวินาที)	0.156	0.156	0.156
ω_z (เรเดียนต่อวินาที)	-0.257	-0.232	-0.247

จากผลการแทนค่าย้อนกลับด้วยความเร็วล้อพบว่า เมื่อเปรียบเทียบกันทั้ง 3 ระบบ มีความแตกต่างกันเล็กน้อย และเมื่อพิจารณาด้วย ω_z พบว่า ระบบจำลอง มีความคลาดเคลื่อนจากการคำนวณด้วยจลนศาสตร์ 9.7% และ ระบบจริง 6.4% เป็นความคลาดเคลื่อนที่ทำให้ความเร็วล้อจากการเคลื่อนที่ของหุ่นยนต์ในช่วงเวลาทั้งหมดมีผลลัพธ์ ดังรูปที่ 4.38 และ 4.39

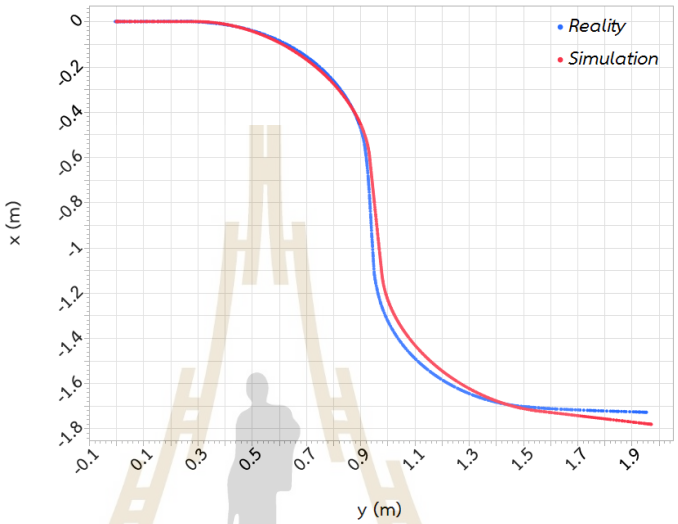


รูปที่ 4.38 ความเร็วล้อซ้าย ω_l และ ω_r (จำลอง)



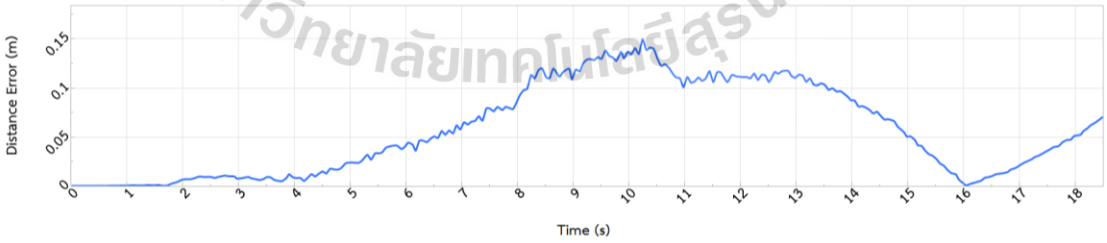
รูปที่ 4.39 ความเร็วล้อซ้าย ω_l และ ω_r (จริง)

โดยความเร็วล้อดังกล่าวมีความสัมพันธ์กับ v_x และ ω_z ในการประมาณตำแหน่ง ซึ่งตัวกรองคาลมานร่วมกับ AMCL สามารถแสดงตำแหน่งการเคลื่อนที่ของหุ่นยนต์จริงเปรียบเทียบกับลักษณะการเคลื่อนที่การจำลองหุ่นยนต์ซึ่งเป็นข้อมูลพิกัดจากไลบรารี p3d_groundtruth ดังรูปที่ 4.40



รูปที่ 4.40 เปรียบเทียบพิกัดการเคลื่อนที่ของหุ่นยนต์ทั้ง 2 ระบบ

พิกัดการเคลื่อนที่ดังกล่าวของหุ่นยนต์สามารถประเมินด้วยความผิดพลาดระยะการเคลื่อนที่เทียบกับหุ่นยนต์จริงโดยคำนวณจากสมการเดียวในการหาค่า $d_{e,i}$ ได้ดังรูปที่ 4.41



รูปที่ 4.41 ค่าความผิดพลาดการเคลื่อนที่ของระบบจำลองเปรียบเทียบกับหุ่นยนต์จริง

จากรูปที่ 4.41 สามารถสรุปได้ว่าพารามิเตอร์ของระบบจำลองที่ผู้วิจัยได้กำหนดในตารางที่ 4.1 เมื่อทำการคำนวณด้วยระบบฟิสิกส์แบบ ODE ในโปรแกรม Gazebo พบว่าพิกัดหุ่นยนต์จากการ

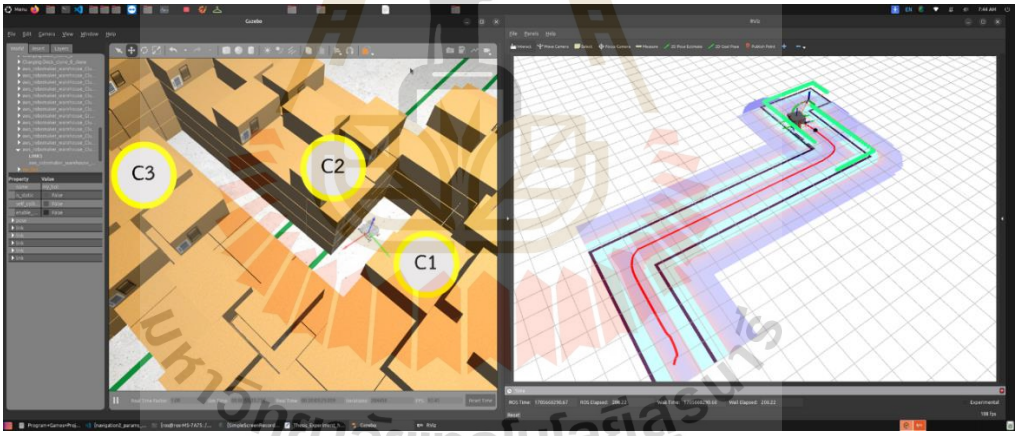
จำลองเคลื่อนที่แตกต่างกับหุ่นยนต์จริงสูงสุดเท่ากับ 0.15 เมตร โดยแนวโน้มที่ระบบให้ความแตกต่างเริ่มต้นในช่วงเวลา วินาทีที่ 2 – 16 และมีลักษณะการเคลื่อนที่แบบเดียวกัน

4.7 ผลจำลองพารามิเตอร์นำทางอัตโนมัติ

4.7.1 การจำลองพารามิเตอร์ระยะมองไปข้างหน้า

จากการคำนวณระยะมองไปข้างหน้าได้ระยะ L มากที่สุด เท่ากับ 0.42 เมตร สามารถกำหนดค่าดังกล่าวเป็นค่าเริ่มต้นในการจำลองพารามิเตอร์ลดลงครั้งละเท่ากับ 0.2 เมตร ซึ่งการควบคุมหุ่นยนต์ที่กำหนดเฉพาะระยะ L เป็นการกำหนดให้ระบบควบคุมเป็นแบบ Pure Pursuit (PP) ซึ่งควบคุมความเร็วเชิงเส้นแบบคงที่ โดยพิจารณาด้วยส่วนโค้งทั้งหมด 3 ช่วง กำหนดเป็น C1 C2 และ C3 สามารถแบ่งออกเป็นทั้งหมด 2 กรณี ดังต่อไปนี้

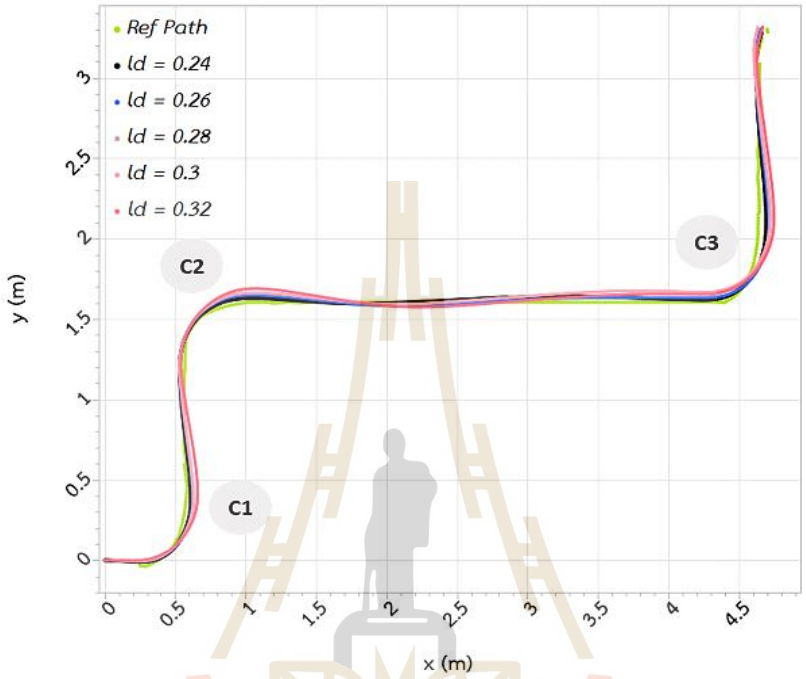
(1) เป็นกรณีที่หุ่นยนต์ไม่สามารถเคลื่อนที่ผ่านสิ่งกีดขวางเนื่องจากระบบตรวจพบการชนกับสิ่งกีดขวางจึงทำให้ระบบหยุดการทำงาน ณ ช่วงโค้ง C1 โดยระยะ L ที่ก่อให้เกิดกรณีดังกล่าวอยู่ในช่วง $0.34 \leq L \leq 0.42$ แสดงผลข้อมูลของหุ่นยนต์ผ่าน RVIZ ดังรูปที่ 4.42



รูปที่ 4.42 พิกัดตำแหน่งการเคลื่อนที่ของหุ่นยนต์ไปถึงจุดหมาย

วิเคราะห์สาเหตุการหยุดการทำงานของหุ่นยนต์สังเกตในส่วนของ planner server ที่ยังมีการวางแผนเส้นทางเพื่อให้ระบบติดตามเส้นทางยังคงทำงานอยู่ เมื่อการเคลื่อนที่ของหุ่นยนต์ถูกควบคุมไปติดที่ส่วนโค้ง C1 ทำให้ขอบเขต Footprint เข้าไปอยู่ในส่วนของค่า Lethal cost ระบบจึงสั่งให้หุ่นยนต์เข้าสู่การ Recovery behaviors เพื่อหยุดการทำงานของหุ่นยนต์ จึงพิจารณาพารามิเตอร์ที่ทำให้เกิดพฤติกรรมดังกล่าวเป็นพารามิเตอร์ที่ไม่เหมาะสมต่อ ระบบนำทางอัตโนมัติ เนื่องจากเป็นพารามิเตอร์ที่ไม่สามารถควบคุมหุ่นยนต์ให้ไปยังจุดหมายได้อย่างปลอดภัย

(2) เป็นกรณีที่หุ่นยนต์สามารถเคลื่อนที่ผ่านสิ่งกีดขวางได้ถึงจุดหมาย โดยแสดงลักษณะการเคลื่อนที่ของหุ่นยนต์ด้วยพิกัดตำแหน่งของ p3d_groundtruth ที่จำลองในโปรแกรม Gazebo ดังรูปที่ 4.43



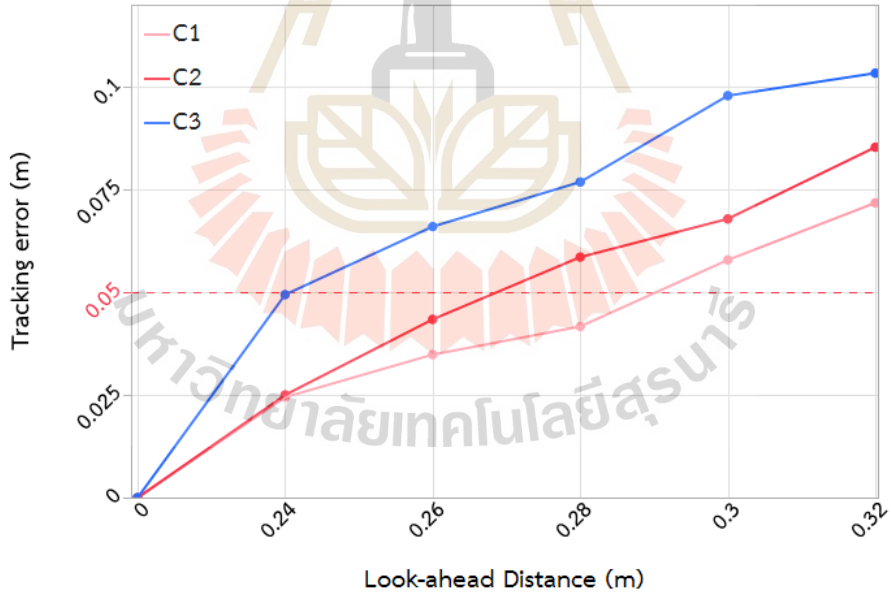
รูปที่ 4.43 พิกัดการเคลื่อนที่ของหุ่นยนต์จากการลดระยะ L

จากผลการจำลอง สังเกตได้ว่าการลดระยะ L นั้น สามารถเพิ่มประสิทธิภาพในการติดตามเส้นทางได้ดียิ่งขึ้น เมื่อพิจารณาส่วนโค้งที่เกิดจากการเลี้ยวด้วยรัศมีมากที่สุดลดลงมาสามารถหาค่าผิดพลาดออกนอกเส้นทางมากที่สุด (Tracking error, t_e) ในส่วนโค้งของเส้นทางทั้ง 3 ส่วน ทำให้เห็นได้ว่าการลดระยะ L สามารถทำให้หุ่นยนต์ติดตามเส้นทางส่วนโค้งที่มีรัศมีที่น้อยลงได้ดีมากยิ่งขึ้น โดยส่วนโค้งสามารถวัดค่าผิดพลาดออกนอกเส้นทางมากที่สุด ดังตารางที่ 4.9

ตารางที่ 4.9 ผลการลดระยะ L ต่อระยะผิดพลาดออกนอกเส้นทาง t_e

ทางโค้ง C1		ทางโค้ง C2		ทางโค้ง C3	
L (เมตร)	t_e (เมตร)	L (เมตร)	t_e (เมตร)	L (เมตร)	t_e (เมตร)
0.32	0.085	0.32	0.104	0.32	0.072
0.3	0. 063	0.3	0.068	0.3	0.077
0.28	0.059	0.28	0.077	0.28	0.042
0.26	0.043	0.26	0.066	0.26	0.035
0.24	0.025	0.24	0.049	0.24	0.024
MIN	0.025	MIN	0.049	MIN	0.024

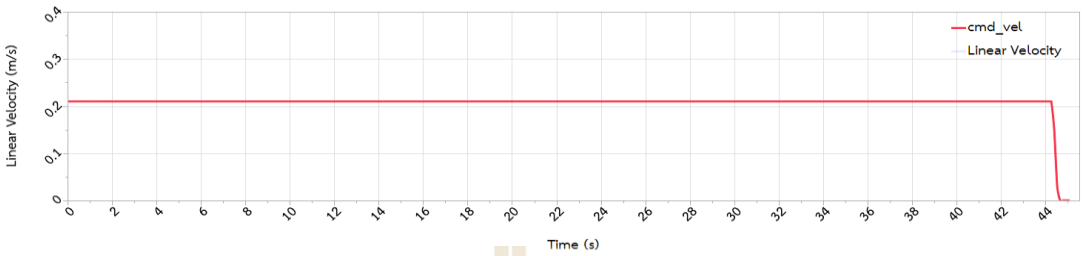
จากผลการทดลอง ดังตารางที่ 4.8 สามารถนำค่าผิดพลาดออกนอกเส้นทางมาแสดงความสัมพันธ์ระหว่างค่าผิดพลาดออกนอกเส้นทางในทางโค้ง 3 ส่วน เทียบกับการเปลี่ยนแปลงระยะ L ที่เปลี่ยนไป ดังรูปที่ 4.44



รูปที่ 4.44 ความสัมพันธ์ของค่าผิดพลาดออกนอกเส้นทางกับระยะ L

จากผลการจำลองพบว่าระยะ L ที่สามารถทำให้หุ่นยนต์เคลื่อนที่ในทางโค้งโดยมีค่าผิดพลาดเป็นไปตามเงื่อนไข เท่ากับ 0.24 เมตร โดยมีระยะความผิดพลาดออกนอกเส้นทางสูงสุด t_e ณ C1 เท่ากับ 0.025 C2 เท่ากับ 0.049 และ C3 เท่ากับ 0.0245 เมตร ซึ่งเป็นค่าให้ระยะ t_e ที่

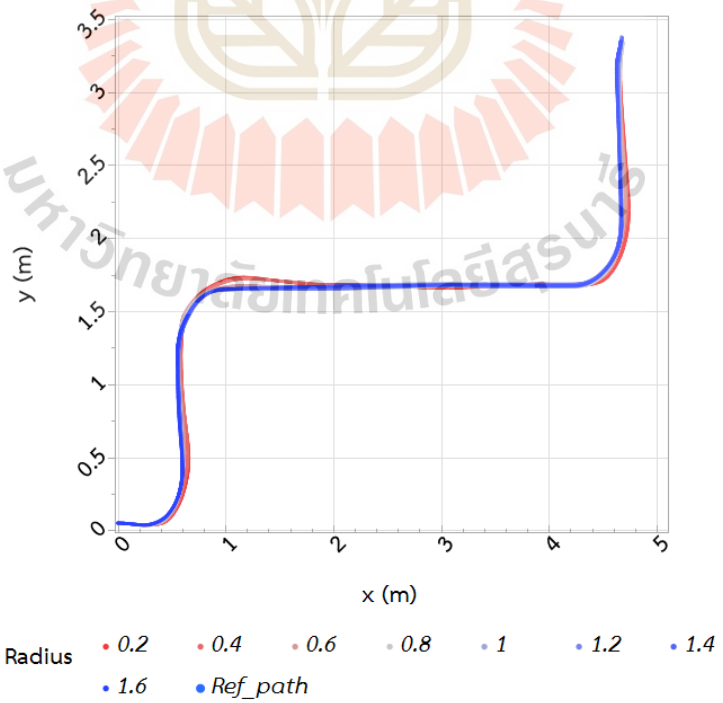
น้อยกว่า 0.05 เมตร ตามเงื่อนไขที่ได้กำหนด โดยที่ความเร็ว v_t มีค่าคงที่เป็นไปตามทฤษฎีการควบคุมแบบ PP ดังรูปที่ 4.45



รูปที่ 4.45 ความเร็วเชิงเส้นจากการติดตามเส้นทางแบบ PP เมื่อ L เท่ากับ 0.24 เมตร

4.7.2 การจำลองพารามิเตอร์ควบคุมความเร็วด้วยรัศมีทางโค้งขั้นต่ำ

เนื่องด้วยระบบควบคุมแบบ PP ไม่สามารถลดความเร็วเชิงเส้นในการควบคุมหุ่นยนต์ผ่านเส้นทางโค้งได้ สามารถจำลองฟังก์ชันควบคุมรัศมีส่วนโค้งขั้นต่ำ Curvature Heuristic ในการกำหนดรัศมีของทางโค้งต่ำสุด r_{min} ให้ระบบควบคุมเป็นแบบ RPP เมื่อ L มีค่าเท่ากับ 0.24 เมตร ให้ผลลัพธ์ ดังรูปที่ 4.46



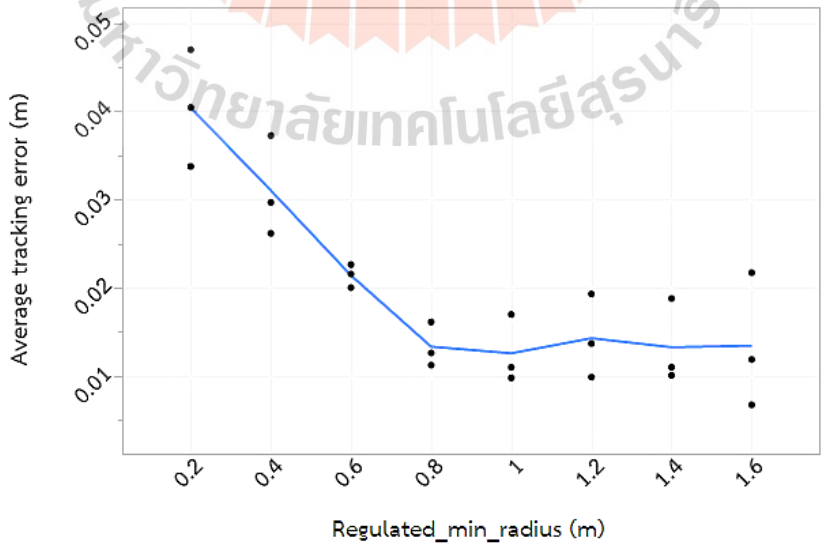
รูปที่ 4.46 ลักษณะพิกัดการเคลื่อนที่ที่สัมพันธ์กับการเปลี่ยนแปลง r_{min}

ด้วยลักษณะพิกัดการเคลื่อนที่ของหุ่นยนต์จากการเปลี่ยนแปงค่า r_{min} สามารถวัดค่าผิดพลาดออกนอกเส้นทาง t_e ณ เส้นทางโค้ง 3 ส่วน ดังตารางที่ 4.10

ตารางที่ 4.10 ความสัมพันธ์ของค่าผิดพลาดออกนอกเส้นทางกับ r_{min}

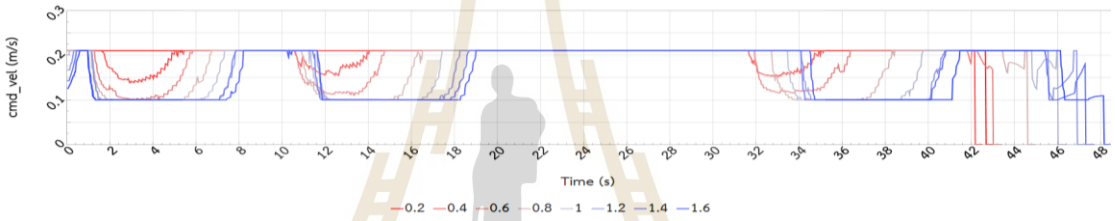
r_{min} (เมตร)	ค่าผิดพลาดทางโค้ง (เมตร)			ระยะเวลานำทาง (วินาที)
	Curvature 1	Curvature 2	Curvature 3	
0.2	0.047	0.040	0.034	43.000
0.4	0.037	0.026	0.030	42.983
0.6	0.023	0.020	0.022	43.728
0.8	0.016	0.013	0.011	45.462
1	0.017	0.011	0.010	46.807
1.2	0.019	0.010	0.014	47.747
1.4	0.019	0.010	0.011	48.125
1.6	0.022	0.007	0.012	48.786

จากตารางที่ 4.10 สามารถหาความสัมพันธ์ของค่าผิดพลาดออกนอกเส้นทาง กับการเปลี่ยนแปลงพารามิเตอร์ r_{min} ดังรูปที่ 4.47



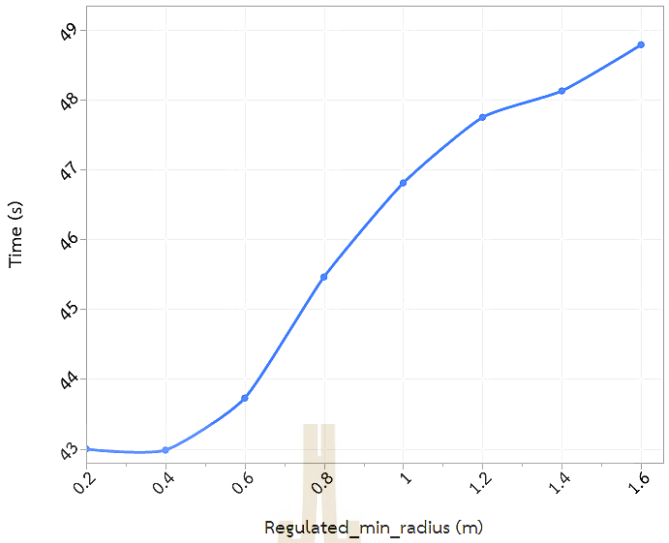
รูปที่ 4.47 ค่าผิดพลาดออกนอกเส้นทางโดยเฉลี่ยจากการเปลี่ยนแปลง r_{min}

จากผลการจำลองพบว่า การเพิ่มขึ้นของระยะ r_{min} สามารถลดค่าผิดพลาด t_e ซึ่งเป็นการควบคุมแบบ RPP โดยค่าความผิดพลาด t_e ที่น้อยที่สุด จากการจำลองเท่ากับ 0.007 เมตร ณ ช่วงโค้ง C2 เมื่อ r_{min} เท่ากับ 1.6 เมตร โดยภาพรวมจากการหาค่าเฉลี่ยในการเข้าโค้งทั้ง 3 ช่วง พบว่าค่าระยะ r_{min} ที่มีค่ามากกว่าหรือเท่ากับ 0.8 เมตร เป็นค่าที่สามารถควบคุมหุ่นยนต์เคลื่อนที่ผ่านทางโค้ง 3 ช่วงมีค่าผิดพลาด โดยเฉลี่ย t_e ไม่เกิน 0.015 เมตร ดังนั้น ค่าระยะ r_{min} ที่ถูกพิจารณาในการเลือกใช้กับหุ่นยนต์จริงมีค่าอยู่ในช่วง $0.8 \leq r_{min} \leq 1.6$ เมตร ถือเป็นช่วงอิ่มตัวของระบบควบคุมแบบ RPP และเป็นช่วงที่สามารถควบคุมความเร็วในทางโค้งให้มีความเร็วเชิงเส้นควบคุม $v_t^{'}$ ที่น้อยที่สุด ดังรูปที่ 4.48



รูปที่ 4.48 คำสั่งควบคุมความเร็วเชิงเส้น $v_t^{'}$ จากระบบควบคุมแบบ RPP

เมื่อหุ่นยนต์ถูกควบความเร็วให้ลดลงในการเข้าโค้งแปรผันตรงกับ r_{min} ที่มีค่ามากขึ้นทำให้ระยะ r_{min} มีความสัมพันธ์กับเวลาเป็นส่วนหนึ่งที่เป็นส่วนเลือกพิจารณาใช้การใช้งานจริงให้มีประสิทธิภาพที่เหมาะสมมากที่สุด โดยมีความสัมพันธ์กับระยะ r_{min} ดังรูปที่ 4.49 ตามลำดับ



รูปที่ 4.49 ความสัมพันธ์ของเวลาที่ใช้ในการนำทางหุ่นยนต์กับระยะ r_{min}

ด้วยความเร็วที่มีการชะลอในส่วนโค้งมากยิ่งขึ้น สังเกตช่วงเวลาที่หุ่นยนต์ติดตามเส้นทางผ่านทางโค้ง 3 ช่วง คือ ช่วง 1.5 - 8.2 วินาที, 10 – 19 วินาที และ 31.5 – 41.5 วินาที ซึ่งเป็นช่วงระยะที่มีความกว้างมากขึ้นแปรผันตรงกับระยะ r_{min} ซึ่งการพิจารณาในส่วนนี้เมื่อวิเคราะห์ร่วมกับเวลาที่ใช้ในการนำทาง พบว่าค่าระยะ r_{min} ในช่วงดังกล่าวมีระยะเวลานำทางอัตโนมัติอยู่ในช่วง 45.46 – 48.79 วินาที จึงสรุปได้ว่าระยะ r_{min} ที่มากขึ้นทำให้มีการควบคุมความเร็ว v_t' ช้าลงในช่วงโค้ง ส่งผลให้ใช้ระยะเวลาไปยังจุดหมายนานขึ้นเช่นเดียวกัน ดังนั้น ด้วยระยะเวลาที่เหมาะสมผนวกกับค่าผิดพลาดติดตามเส้นทางที่พิจารณาด้วยค่าเฉลี่ยพบว่าค่าความผิดพลาดเริ่มมีช่วงอิ่มตัว ณ r_{min} เท่ากับ 0.8 เมตร โดยให้เวลาการนำทางเท่ากับ 45.46 วินาที ซึ่งเป็นระยะเวลานำทางที่เร็วที่สุดในการควบคุมหุ่นยนต์มีความเร็วถึงค่าต่ำสุดโดยที่มีค่าผิดพลาด t_e ใกล้เคียงกับ r_{min} ที่มากกว่า 0.8 เมตร ซึ่งเพียงพอต่อการนำทางด้วยระบบติดตามแบบ RPP

ผลเฉลี่ยค่าผิดพลาดออกนอกเส้นทางนี้แตกต่างกับช่วงระยะ $0.2 \leq r_{min} \leq 0.6$ เมตร ซึ่งเป็นช่วงที่ไม่สามารถลดความเร็ว v_t' ในทางโค้งได้เท่ากับ 0.1 เมตรต่อวินาที ดังนั้นจากการจำลองระบบนำทางอัตโนมัติควบคุมด้วยอัลกอริทึมติดตามเส้นทางแบบ RPP สามารถสรุปได้ว่าระยะ r_{min} ที่ทำให้การควบคุมมีค่าผิดพลาดติดตามเส้นทางโดยเฉลี่ยถึงช่วงอิ่มตัวของระบบไม่เกิน 0.015 เมตร มีระยะ r_{min} เท่ากับ 0.8 เมตร โดยสามารถลดความเร็วเชิงเส้นทางโค้งได้ต่ำสุดเท่ากับ 0.1 เมตรต่อวินาที เมื่อเปรียบเทียบกับการควบคุมแบบ PP พบว่ามีประสิทธิภาพในการติดตามเส้นทางมากกว่า สรุปในตารางที่ 4.11

ตารางที่ 4.11 การเปรียบเทียบค่าผิดพลาดออกนอกเส้นทางระหว่าง PP กับ RPP

รูปแบบการควบคุม	ค่าผิดพลาดออกนอกเส้นทางมากที่สุดในส่วนโค้ง (เมตร)		
	C1	C2	C3
PP	0.025	0.049	0.024
RPP	0.016	0.013	0.011
เปอร์เซ็นต์	36.00%	73.47%	54.27%

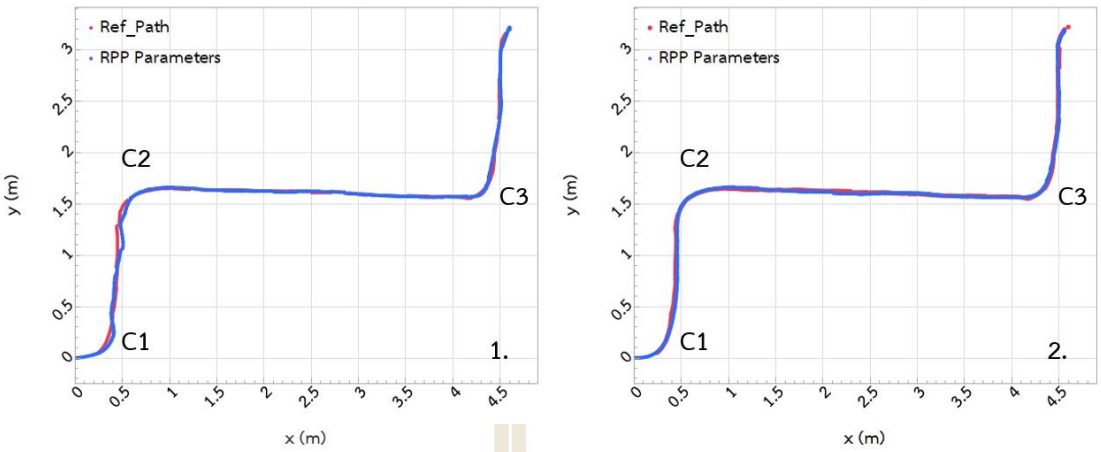
เมื่อนำพารามิเตอร์การติดตามเส้นทางแบบ RPP เปรียบเทียบกับการติดตามเส้นทางแบบ PP พบว่าสามารถลดค่าผิดพลาดออกนอกเส้นทาง t_e สูงสุด ณ C1, C2 และ C3 ได้ 36.00%, 73.47% และ 54.27% ตามลำดับ ดังนั้น สามารถสรุปพารามิเตอร์ที่เหมาะสมกับหุ่นยนต์ที่สามารถใช้งานในสภาพแวดล้อมดังกล่าว ดังตารางที่ 4.12

ตารางที่ 4.12 พารามิเตอร์ที่เหมาะสมกับระบบนำทางอัตโนมัติ

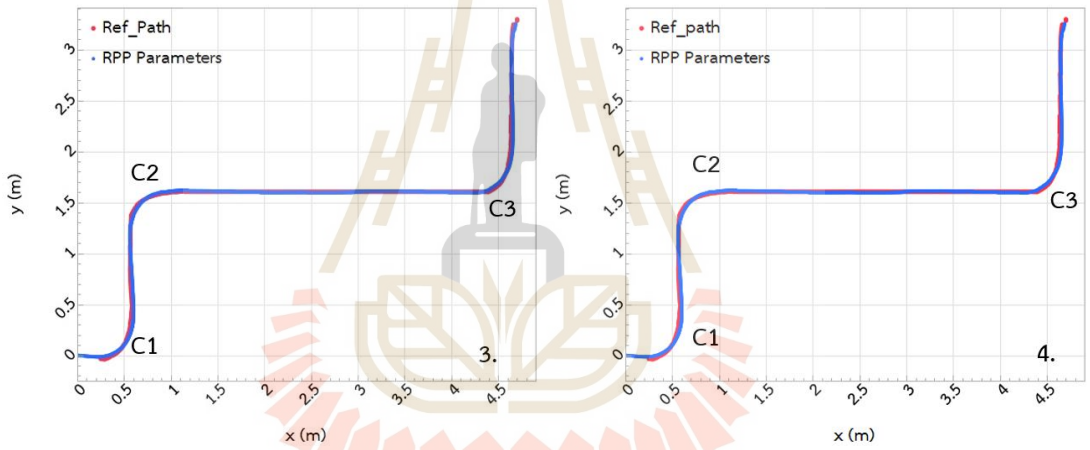
พารามิเตอร์ติดตามเส้นทางอัตโนมัติ RPP		
ระยะมองไปข้างหน้า (L) เมตร	ระยะรัศมีความโค้งขั้นต่ำ (r_{min}) เมตร	ความเร็วต่ำสุดในการเข้าโค้ง (v_{min}) เมตรต่อวินาที
0.24	0.8	0.1
พารามิเตอร์การสร้าง Costmap		
Inflation_radius	Cost_scaling_factor	
0.55	10.0	

4.8 ผลทดสอบพารามิเตอร์ RPP กับหุ่นยนต์จริง

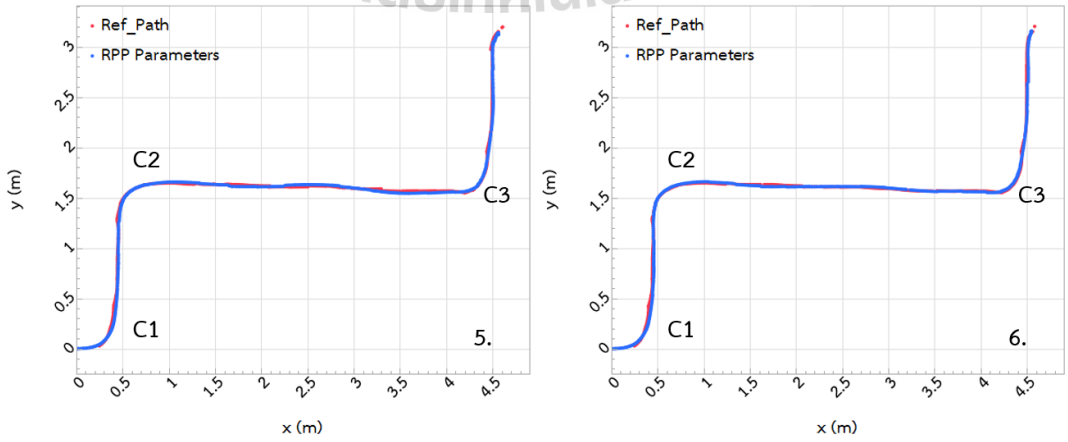
จากการจำลองระบบสู่การนำพารามิเตอร์ที่ได้จากการจำลองในตารางที่ 4.11 มาประยุกต์ใช้กับหุ่นยนต์จริงที่มีการจัดสภาพแวดล้อมแบบเดียวกัน โดยการทดสอบทั้งหมด 10 ครั้ง เมื่อหุ่นยนต์เคลื่อนที่ออกจากจุดเริ่มต้น ให้ผลลัพธ์พิกัดตำแหน่งการเคลื่อนที่ ดังรูปที่ 4.50 – 4.54 ตามลำดับ



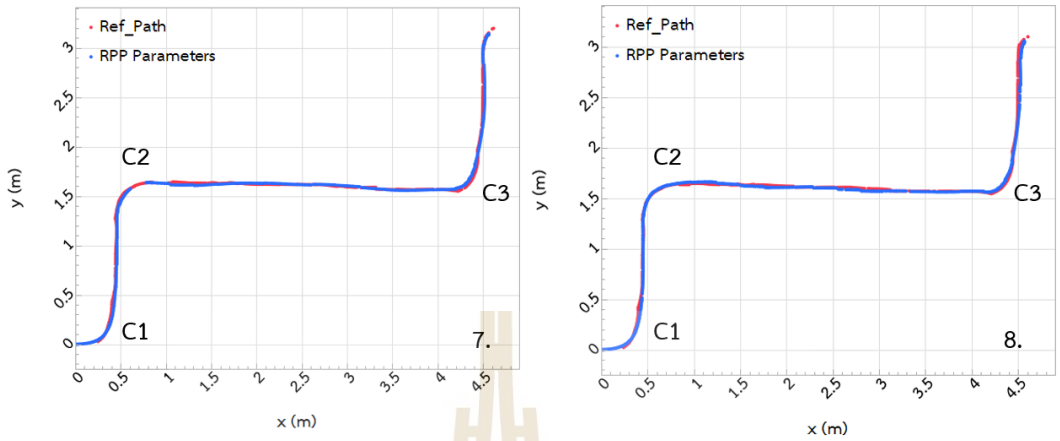
รูปที่ 4.50 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 1 และ 2



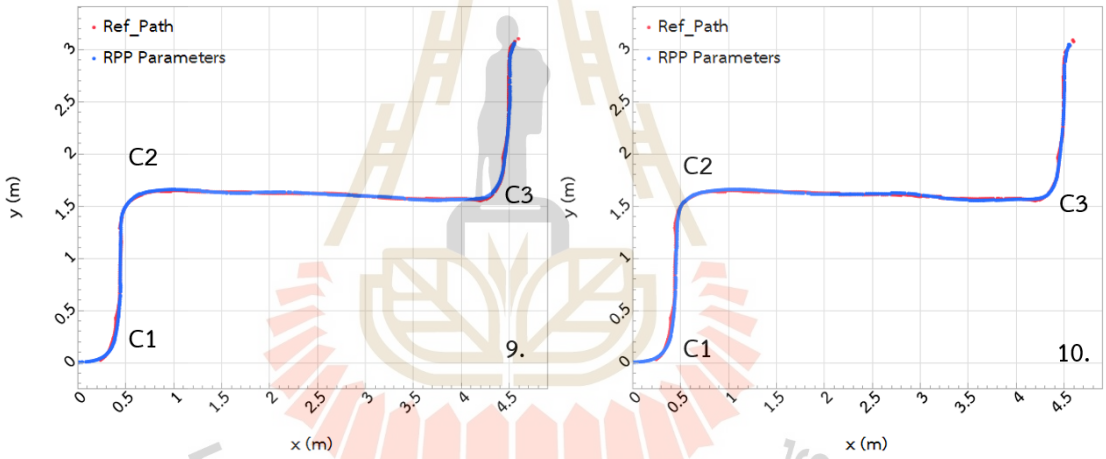
รูปที่ 4.51 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 3 และ 4



รูปที่ 4.52 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 5 และ 6



รูปที่ 4.53 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 7 และ 8



รูปที่ 4.54 ตำแหน่งพิกัดลักษณะการเคลื่อนที่ของหุ่นยนต์จริงจากการทดสอบครั้งที่ 9 และ 10

จากการเคลื่อนที่ของหุ่นยนต์ในสภาพแวดล้อมจริงสามารถวัดระยะผิดพลาดออกนอกเส้นทาง t_e ได้ทั้งหมด 3 ช่วง จากการทดลองทั้งหมด 10 ครั้ง เช่นเดียวกับระบบจำลองด้วยโปรแกรม Gazebo ให้ผลลัพธ์ดังตารางที่ 4.13

ตารางที่ 4.13 ค่าผิดพลาดติดตามเส้นทางผ่านทางโค้งทั้งหมด 3 ช่วง จากการทดสอบ 10 ครั้ง

การทดลอง	**ระยะผิดพลาดออกนอกเส้นทางมากที่สุด t_e (เมตร)**		
	C1	C2	C3
ครั้งที่ 1	0.0183	0.0116	0.0125

การทดลอง	ระยะผิดพลาดออกนอกเส้นทางมากที่สุด t_e (เมตร)		
	C1	C2	C3
ครั้งที่ 2	0.0165	0.0181	0.0167
ครั้งที่ 3	0.0137	0.0171	0.0120
ครั้งที่ 4	0.0171	0.0151	0.0040
ครั้งที่ 5	0.0158	0.0128	0.0102
ครั้งที่ 6	0.0162	0.0132	0.0103
ครั้งที่ 7	0.0163	0.0125	0.0168
ครั้งที่ 8	0.0172	0.0133	0.0174
ครั้งที่ 9	0.0164	0.0115	0.0109
ครั้งที่ 10	0.0155	0.0135	0.0102
MAX (ค่าเฉลี่ยจริง)	0.0163	0.0138	0.0121
MAX (Simulation)	0.0160	0.0130	0.0110
ค่าผิดพลาดจากการจำลอง	1.840%	6.272%	9.091%

จากตารางที่ 4.13 พบว่า การเคลื่อนที่ผ่านเส้นทางโค้งทั้ง 3 ช่วง ของหุ่นยนต์จริงจากการทดลอง 10 ครั้ง หุ่นยนต์มีระยะผิดพลาดออกนอกเส้นทางเฉลี่ยมากที่สุดเท่ากับ 0.0163 เมตร ณ ช่วงโค้ง C1 โดยที่หุ่นยนต์จริงจะมีลักษณะการเคลื่อนที่แตกต่างกันเล็กน้อย ซึ่งจากการทดลอง 10 ครั้ง พิจารณาความผิดพลาดของระบบจำลองเปรียบเทียบด้วยค่าความผิดพลาดจริง เพื่อวิเคราะห์ถึงประสิทธิภาพจากการจำลอง ดังสมการที่ 4.7

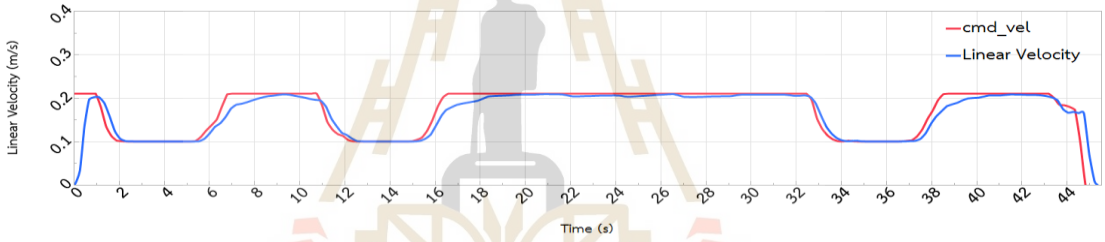
$$\left|\frac{t_{e(real)} - t_{e(sim)}}{t_{e(real)}}\right| \times 100\% \tag{4.7}$$

เมื่อ $t_{e(real)}$ คือ ค่าผิดพลาดออกนอกเส้นทางในแต่ละส่วนโค้งของหุ่นยนต์จริง $t_{e(sim)}$ คือ ค่าผิดพลาดออกนอกเส้นทางในแต่ละส่วนโค้งจากการจำลอง เมื่อพิจารณาจากสมการดังกล่าวจะเห็นได้ว่าเป็นการเปรียบเทียบเพื่อหาประสิทธิภาพในของระบบจำลองหุ่นยนต์ในการนำไปพัฒนาต้นแบบที่มีระบบแตกต่างกันออกไป ซึ่งจากการคำนวนพบว่าระบบจำลองให้ความผิดพลาดสูงสุด ณ ทางโค้ง C3 เท่ากับ 9.091% เกิดจากการหาค่าเฉลี่ยที่ค่าจริงมีแตกต่างจากการจำลองในการทดลองครั้งที่ 2 ครั้งที่ 7 และ ครั้งที่ 8 ซึ่งเป็นค่าที่เกิดขึ้นได้เนื่องจากเป็นส่วนโค้งที่ระยะทางที่ไกลมากที่สุดเมื่อเทียบกับส่วนโค้ง C1 และ C2 ทำให้ความแปรปรวนของสัญญาณในการส่งของมูลสื่อสารผ่าน SSH และความ

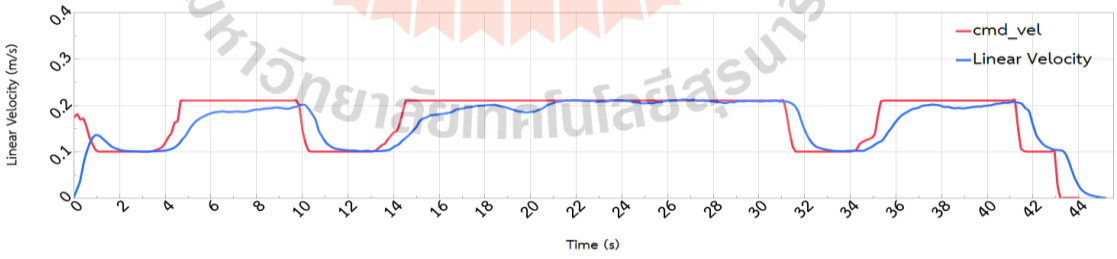
แปรปรวนของเซนเซอร์ในระยะทางที่ไกลขึ้นเป็นปัจจัยที่สำคัญที่ทำให้เกิดความผิดพลาดจากการวัด รวมถึงการหน่วงเวลาจากการส่งสัญญาณ ทำให้ค่าที่วัดได้จากการทดลองหลาย ๆ ครั้ง ในบางครั้ง มีค่าออกนอกเส้นทางมากกว่าค่าที่ควรจะเป็นเปรียบเทียบกับการทดลองครั้งอื่น ๆ

4.9 เปรียบเทียบผลการจำลองกับหุ่นยนต์จริง

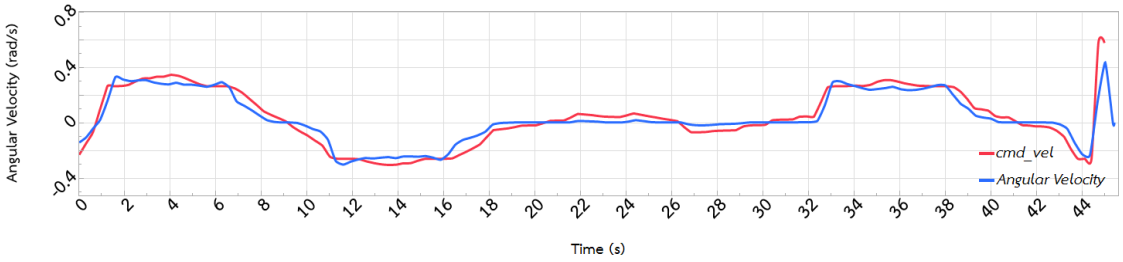
ลักษณะของสัญญาณป้อนกลับเทียบกับคำสั่งควบคุมด้วย cmd_vel ของหุ่นยนต์ที่กำลังเคลื่อนที่ในสภาพแวดล้อมแบบเดียวกันด้วยระบบติดตามเส้นทางแบบ RPP สามารถเก็บข้อมูลด้วย rosbags จากการทดสอบในแต่ละครั้งและเปรียบเทียบผลระหว่างหุ่นยนต์จริงและหุ่นยนต์ที่จำลองด้วยโปรแกรม Gazebo โดยสัญญาณป้อนกลับประกอบด้วยสัญญาณความเร็วเชิงเส้น v_x ความเร็วเชิงมุม ω_z และในส่วนของ cmd_vel ที่เป็นคำสั่งจากระบบควบคุม RPP ประกอบด้วยคำสั่งที่เป็นความเร็วเชิงเส้นอินพุต v_t' และความเร็วเชิงมุมอินพุต ω_t สามารถแสดงผลลัพธ์ได้ดังรูปที่ 4.55 – 4.58 ตามลำดับ



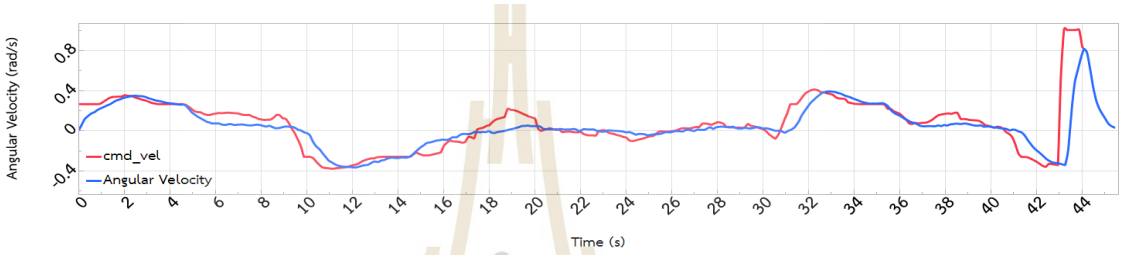
รูปที่ 4.55 ลักษณะการควบคุมด้วย v_t' และความเร็วป้อนกลับ v_x (จำลอง)



รูปที่ 4.56 ลักษณะการควบคุม v_t' และความเร็วป้อนกลับ v_x (จริง)



รูปที่ 4.57 ลักษณะการควบคุม ω_t และความเร็วป้อนกลับ ω_z (จำลอง)

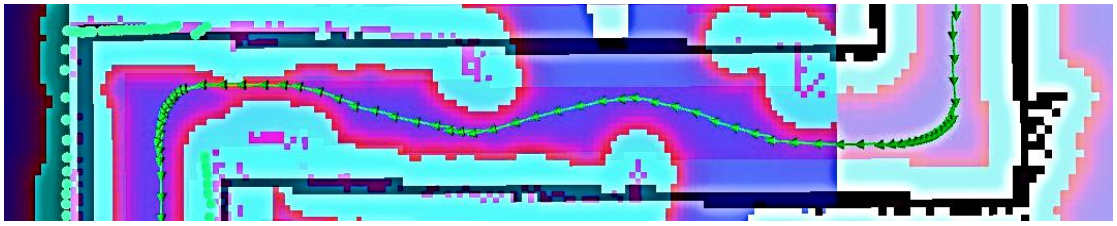


รูปที่ 4.58 ลักษณะการควบคุม ω_t และความเร็วป้อนกลับ ω_z (จริง)

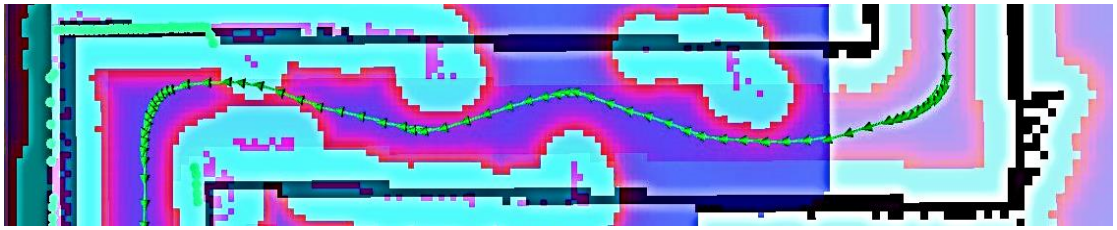
จากการเคลื่อนที่ในสภาพแวดล้อมแบบเดียวกัน หุ่นยนต์สามารถนำทางอัตโนมัติด้วยสัญญาณป้อนกลับที่มีลักษณะเดียวกัน โดยที่ระบบจริงมีความหน่วงต่อสัญญาณควบคุม cmd_vel ประมาณ 1 วินาที เนื่องมาจากประสิทธิภาพในการกรองสัญญาณจากตัวกรองคาลมานและการแลกเปลี่ยนข้อมูลผ่าน SSH

4.10 ผลทดสอบการหลบหลีกสิ่งกีดขวางอัตโนมัติ (หุ่นยนต์จริง)

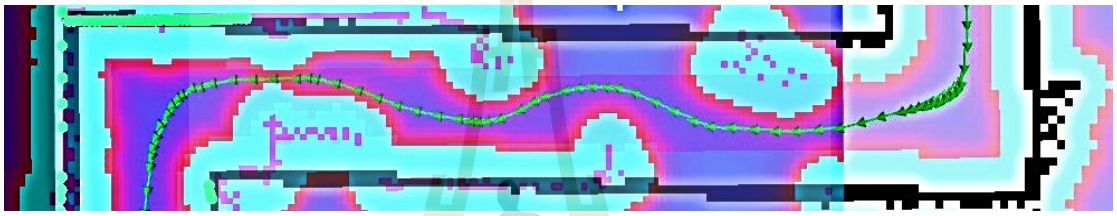
จากการนำพารามิเตอร์ที่เหมาะสมจากการจำลองด้วยโปรแกรม Gazebo ที่สรุปในตารางที่ 4.11 นำมาทดสอบความสามารถในการวางแผนและติดตามเส้นทาง ต่อการเปลี่ยนแปลงสภาพแวดล้อมด้วยการเพิ่มสิ่งกีดขวางเข้าไปในสภาพแวดล้อมที่ทดสอบ โดยที่สิ่งกีดขวางดังกล่าวไม่ได้เป็นส่วนหนึ่งของข้อมูล Occupancy Grid Map ที่ได้จากวิธีการ SLAM หุ่นยนต์สามารถหลบหลีกสิ่งกีดขวางอัตโนมัติ โดยมีพิกัดที่แสดงลักษณะการเคลื่อนที่ ดังรูปที่ 4.59 – 4.64



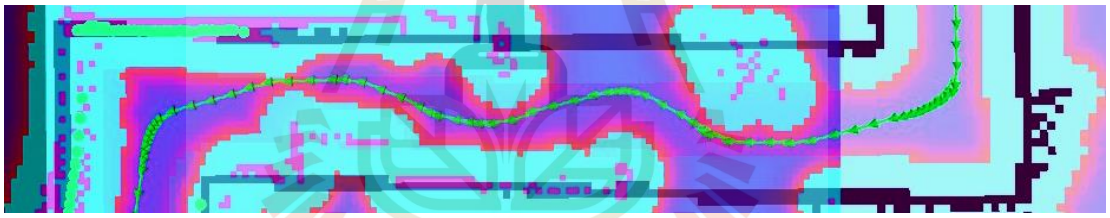
รูปที่ 4.59 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.7 เมตร



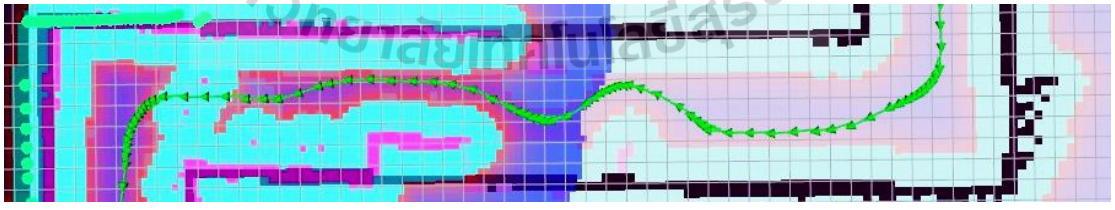
รูปที่ 4.60 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.6 เมตร



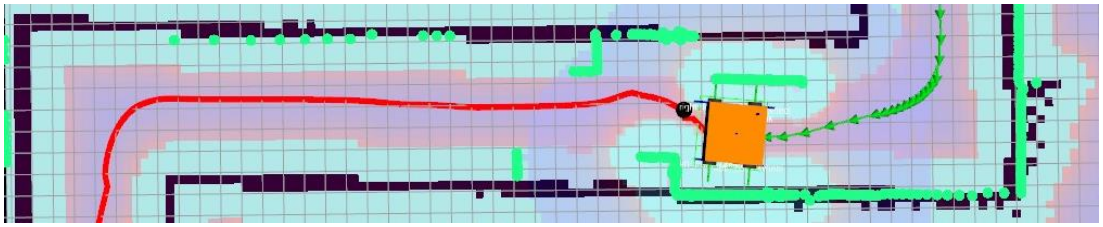
รูปที่ 4.61 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.5 เมตร



รูปที่ 4.62 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.4 เมตร



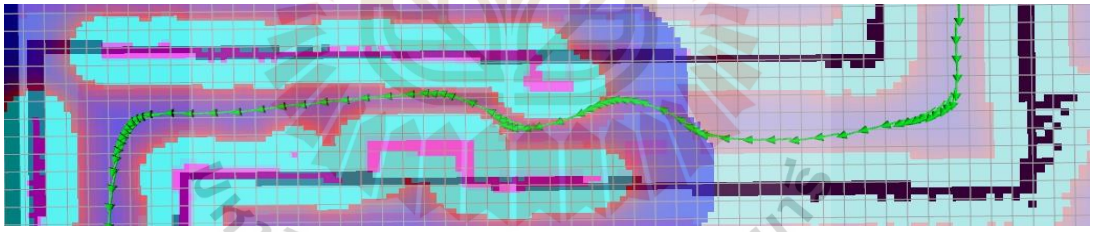
รูปที่ 4.63 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.3 เมตร



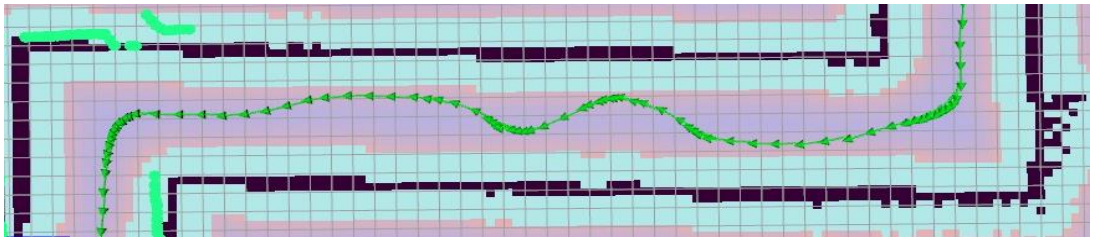
รูปที่ 4.64 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.2 เมตร

จากการทดสอบดังรูปที่ 4.64 พบว่า หุ่นยนต์ไม่สามารถเคลื่อนที่ตามเส้นทางที่วางแผนไว้สังเกตการแสดงผลผ่านโปรแกรม RVIZ โดยเป็นกรณีที่หุ่นยนต์หยุดวิ่งเนื่องจากระบบตรวจจับการชนด้วยระยะ L ที่อ้างอิงด้วยขอบเขต Footprint เพื่อตรวจจับสิ่งกีดขวางด้านหน้าหุ่นยนต์ที่มีระยะ L น้อยกว่า 0.24 เมตร เป็นระยะห่างของจุดมองไปข้างหน้ากับเฟรม base_footprint ที่อยู่ ณ จุดกึ่งกลางของหุ่นยนต์ อ้างอิงด้วยกริดของแผนที่ โดยที่ 1 กริดเท่ากับ 0.1 เมตร พบว่าหุ่นยนต์ไม่สามารถเคลื่อนที่ผ่านเส้นทางที่มีความกว้างระยะ d เท่ากับ 0.2 เมตร เมื่อ d_1 เท่ากับ 0.52 เมตร

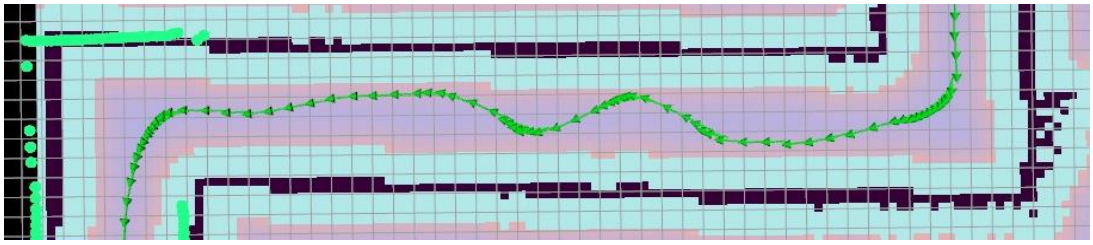
จากการทดสอบเพื่อหาขีดจำกัดการเลี้ยวหลบหลีกสิ่งกีดขวาง พบว่า สภาพแวดล้อมที่หุ่นยนต์สามารถเคลื่อนที่ผ่านได้ คือ กรณีที่ d เท่ากับ 0.3 เมตร โดยที่มีการหยุดวิ่งชั่วขณะเนื่องจากเป็นช่วงที่ตรวจพบสิ่งกีดขวางในระยะ L และยังไม่มีการวางแผนเส้นทางใหม่ โดยการทดสอบในกรณีดังกล่าว ได้ทดสอบเพิ่มอีก 3 ครั้ง ดังรูปที่ 4.65 – 4.67



รูปที่ 4.65 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.3 เมตร ครั้งที่ 1



รูปที่ 4.66 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.3 เมตร ครั้งที่ 2



รูปที่ 4.67 การหลบหลีกสิ่งกีดขวางอัตโนมัติเมื่อ d เท่ากับ 0.3 เมตร ครั้งที่ 3

จากการทดสอบพบว่า หุ่นยนต์สามารถเคลื่อนที่ซ้ำโดยมีลักษณะการเคลื่อนที่แบบเดิม พิจารณาโดยการทดสอบในครั้งที่ 2 และครั้งที่ 3 มีค่า Cost ของสิ่งกีดขวางแตกต่างจากครั้งที่ 1 เนื่องจากเมื่อหุ่นยนต์ถึงเป้าหมายจะทำการ Clear cost เพื่อกำจัดข้อมูลสิ่งกีดขวางที่เป็นข้อมูลเดิม ขณะที่ Recovery Server ทำงานเมื่อขอบเขต Footprint เข้าไปอยู่ใน Lethal Cost ทำให้ไม่มีการแสดงผลในส่วนของ Local Costmap ในกรณีดังกล่าว ดังนั้น จากการทดลองสรุปได้ว่าหุ่นยนต์สามารถสร้างเส้นทางและหลบหลีกสิ่งกีดขวางได้อย่างอัตโนมัติ โดยที่เส้นทางมีความกว้างน้อยที่สุด d_1 เท่ากับ 0.3 เมตร และ d เท่ากับ 0.52 เมตร ในขณะที่หุ่นยนต์มีความกว้าง 0.303 เมตร คิดเป็น 58.3 เปอร์เซ็นต์

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

วิทยานิพนธ์นี้มีวัตถุประสงค์ในการจำลองพฤติกรรมของระบบติดตามเส้นทางแบบ RPP ให้กับระบบนำทางอัตโนมัติด้วยโปรแกรม Gazebo เพื่อทดสอบหาพารามิเตอร์ที่เหมาะสมในการประยุกต์ใช้กับหุ่นยนต์จริงที่พัฒนาขึ้น โดยสามารถลดต้นทุนในการจัดสภาพแวดล้อมทดสอบจริง

องค์ประกอบที่สำคัญในการพัฒนาระบบนำทางอัตโนมัติได้เน้นส่วนที่สำคัญ คือ ระบบระบุตำแหน่ง จากการประยุกต์ใช้ตัวกรองคาลมานเพื่อเพิ่มประสิทธิภาพของข้อมูลที่วัดที่ได้จากเซนเซอร์ Encoder และ ไจโรสโคป และได้นำผลลัพธ์การประมาณพิกัดตำแหน่งจากตัวกรองไประบุตำแหน่งของหุ่นยนต์บนแผนที่ด้วยอัลกอริทึม AMCL

ระบบนำทางอัตโนมัติเป็นระบบ Navigation Stack ร่วมกับระบบปฏิบัติการ ROS2 ซึ่งหุ่นยนต์ที่พัฒนาเป็นหุ่นยนต์ที่สามารถวางแผนเส้นทางที่ใกล้ที่สุด รวมถึงหลบหลีกสิ่งกีดขวางได้อย่างอัตโนมัติด้วยข้อมูลแผนที่ที่สร้างด้วยวิธีการ SLAM สามารถแบ่งผลทดสอบได้ดังหัวข้อต่อไปนี้

5.1.1 การระบุตำแหน่งของหุ่นยนต์

การระบุตำแหน่งของหุ่นยนต์จริงได้เลือกใช้อัลกอริทึม AMCL เนื่องจากมีความน่าเชื่อถือต่อการใช้งานในระยะทางไกลและพื้นที่ที่จำกัด ด้วยค่าความผิดพลาด RMSE สะสมในระยะทาง 3.6 เมตร เท่ากับ 0.029 เมตร มีค่าผิดพลาดสูงสุดไม่เกิน 0.043 เมตร โดย AMCL สามารถลดค่าผิดพลาด RMSE จากการระบุตำแหน่งด้วยวิธี EKF ได้ 68.12% และวิธี Dead Reckoning ถึง 74.78% ซึ่งการระบุตำแหน่งแบบ AMCL นั้น มีอินพุตจาก EKF ซึ่งเกิดจากการบูรณาการเซนเซอร์ 2 ชนิด ทำให้สรุปได้ว่าเมทริกซ์ความแปรปรวนที่ได้ปรับใช้มีความเหมาะสมเพียงพอในการระบุตำแหน่งในพื้นที่กว้างน้อยที่สุดเท่ากับ 0.52 เมตร

5.1.2 การจำลองพารามิเตอร์แบบ PP

การใช้ระบบจำลองทดสอบค่าระยะมองไปข้างหน้าที่เหมาะสมพบว่ามีค่าเท่ากับ 0.24 เมตร เป็นค่าที่ทำให้หุ่นยนต์ออกนอกเส้นทางสูงสุดเท่ากับ 0.049 เมตร จากการเข้าโค้งทั้งหมด 3 ส่วน

5.1.3 การจำลองพารามิเตอร์แบบ RPP

ค่ารัศมีขั้นต่ำในการควบคุมความเร็วจากการตรวจจับทางโค้งที่เหมาะสมเท่ากับ 0.8 เมตรเป็นค่าที่สามารถลดความเร็วได้ตามค่าที่กำหนด คือ 0.1 เมตรต่อวินาที โดยมีค่าผิดพลาดออกนอกเส้นทางสูงสุดเท่ากับ 0.016 เมตร ซึ่งลดค่าผิดพลาดจากการควบคุมแบบ PP ได้ 67.35%

5.1.4 การทดสอบใช้งานจริงด้วยพารามิเตอร์แบบ RPP

หุ่นยนต์มีพิกัดการเคลื่อนที่เช่นเดียวกับระบบจำลองโดยที่ค่าผิดพลาดออกนอกเส้นทางสูงสุดโดยเฉลี่ยจากการทดลอง 10 ครั้ง 0.0163 เมตร เกิด ณ ทางโค้ง C1 เมื่อนำค่าผิดพลาดเปรียบเทียบกับผลการจำลองทั้ง 3 ช่วงทางโค้ง พบว่าระบบจำลองสามารถจำลองหุ่นยนต์โดยมีค่าความคลาดเคลื่อนกับระบบจริงมากที่สุด 9.091%

5.1.5 การหลบลึกสิ่งกีดขวางอัตโนมัติ

จากการจัดสภาพแวดล้อมเพื่อให้หุ่นยนต์สามารถสร้างเส้นทางใหม่ตามสภาพแวดล้อมที่เปลี่ยนไป หุ่นยนต์สามารถวางแผนเส้นทางและเคลื่อนที่ผ่านสิ่งกีดขวางที่มีความกว้างของเส้นทางแคบที่สุดเท่ากับ 0.52 เมตร ขณะที่หุ่นยนต์มีความกว้าง 0.303 เมตร คิดเป็น 58.27% โดยหลบหลีกสิ่งกีดขวางที่มีระยะ d น้อยที่สุดเท่ากับ 0.3 เมตร

5.2 ข้อเสนอแนะ

5.2.1 หุ่นยนต์ที่พัฒนาได้เลือกใช้ล้อจากบริษัท ROBOTIS ซึ่งมีขนาดรัศมีที่ต่ำ ทำให้มีความเร็วที่ค่อนข้างต่ำในการเคลื่อนที่ ซึ่งความต้องการใช้งานที่ความเร็วสูงขึ้นสามารถเปลี่ยนขนาดรัศมีล้อให้มีขนาดที่มากขึ้น

5.2.2 ระบบนำทางอัตโนมัติ NAV2 ไม่สามารถแสดงผล RVIZ กับบอร์ด Raspberry Pi 4 ได้อย่างเสถียรเนื่องจากมีคุณสมบัติไม่เพียงพอ โดยสามารถเปลี่ยนไปใช้บอร์ดที่มีคุณสมบัติที่สูงกว่า เช่น Nvidia Jetson เป็นต้น

รายการอ้างอิง

Achmad, M.S., Daud, Mohd., Razali, Saifudin. and Pebrianti, Dwi. (2016). **A ROS-based**

Al Khatib, E.I., Jaradat, M.A., and Abdel-Hafez, M.F. (2020). **Low-Cost Reduced Navigation System for Mobile Robot in Indoor/Outdoor Environments**. IEEE Access, 8, 25014-25026.

Audi, Ahmad., Deseilligny, Marc., Meynard, Christophe., and Thom, Christian. (2017). **Implementation of an IMU Aided Image Stacking Algorithm in a Digital Camera for Unmanned Aerial Vehicles**. Sensors. 17. 1646.

human-robot interaction for indoor exploration and mapping. International Journal of ADVANCED AND APPLIED SCIENCES. 3. 88-92.

Bräunl, T. (2022). **Driving Robots. In: Embedded Robotics. Springer**, Singapore.

Boshlyakov, A., and Maltsev, A. (2020). **Development of a Navigation System for an Educational Mobile Robot**. ITM Web of Conferences. 35. 04007.

Cairl, B.R. (2018). **Deterministic, asynchronous message driven task execution with ROS**. ROSCon Madrid 2018.

Caracciolo, L., Luca, A.D. and Iannitti, S. **Trajectory tracking control of a four-wheel differentially driven mobile robot**. In Proceedings of IEEE/RSJ International Conference on Robotics and Automation, Detroit, MI, USA, 10–15 May 1999; pp. 2632–2638.

Chen, C.S., Lin, C.J., Lai, C.C., and Lin, S.Y. (2022). **Velocity Estimation and Cost Map Generation for Dynamic Obstacle Avoidance of ROS Based AMR**. Machines. 2022; 10(7):501. https://doi.org/10.3390/machines10070501

Chi, Z., Yu, Z., Wei, Q., He, Q., Li, G. and Ding, S. (2023). **High-Efficiency Navigation of Nonholonomic Mobile Robots Based on Improved Hybrid A* Algorithm**. Applied Sciences. 2023, 13(10) : 6141.

Chia, Kim Seng. (2018). **Ziegler-Nichols Based Proportional-Integral-Derivative Controller for a Line Tracking Robot**. Indonesian Journal of Electrical Engineering and Computer Science. 9. 221-226. 10.11591/ijeecs.v9.i1.pp221-226.

Chikurtev, D. (2020). **Mobile Robot Simulation and Navigation in ROS and Gazebo**. International Conference Automatics and Informatics (ICAI), Varna, Bulgaria

Das, S., Sarkar, T.S., Chakraborty, B., and Dutta, H.S. (2016). **A Simple Approach to Design a Binary Coded Absolute Shaft Encoder**. IEEE Sensors Journal, 16, 2300-2305.

Dhaouadi, R. and Hatab, A.A. (2013). **Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework**. IEEE International Conference on Robotics and Automation.

Durrant-Whyte, H.F., and Bailey, T. (2006). **Simultaneous Localisation and Mapping (SLAM) : Part I The Essential Algorithms**.

Erez, J., Tassa, T. and Todorov, E. (2015) **Simulation tools for model-based robotics Comparison of Bullet ODE and PhysX**. IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA

Hirpo, B.D. and Zhongmin, Wang. (2017). **Design and Control for Differential Drive Mobile Robot.** INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 06, Issue 10 (October 2017).

Huang, J., Junginger, S., Liu, H., and Thurow, K. (2023). **Indoor Positioning Systems of Mobile Robots: A Review**. Robotics. 2023; 12(2):47.

Huang, Y-H., and Lin, C-T. (2023). **Indoor Localization Method for a Mobile Robot Using LiDAR and a Dual AprilTag**. Electronics. 2023; 12(4):1023.

Jay A. Farrell (2008). **Aided Navigation GPS with High-Rate Sensors**. :McGrawHill

Jekal, S., Kim, J., Kim, D-H., Noh, J., Kim, M-J., Kim, H-Y., Kim, M-S., Oh, W-C., and Yoon, C-M. (2022). **Synthesis of LiDAR-Detectable True Black Core/Shell Nanomaterial and Its Practical Use in LiDAR Applications. Nanomaterials**. 2022, 12(20):3689.

Kim, D., Han, C., and Lee, J.Y. (2013). **Sensor-based motion planning for path tracking and obstacle avoidance of robotic vehicles with nonholonomic constraints**. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 227, 178 - 191.

Liao, J., Chen, Z., and Yao, B. (2017). **Performance-Oriented Coordinated Adaptive Robust Control for Four-Wheel Independently Driven Skid Steer Mobile Robot**. IEEE Access, 5, 19048-19057.

Liu, L., Wang, X., Yang, X., Liu, H., Li, J., and Wang, P. (2023). **Path Planning Techniques for Mobile Robots: Review and Prospect**. Expert Syst.Appl. 2023, 227, 120254.

Long, Z., Xiang, Y., Lei, X., Li, Y., Hu, Z. and Dai, X. (2022). **Integrated Indoor Positioning System of Greenhouse Robot Based on UWB/IMU/ODOM/LIDAR.** Sensors. 2022; 22(13):4819.

Macenski, S., Foote, T., Gerkey, B.P., Lalancette, C. and Woodall, W. (2022). **Robot Operating System 2: Design, architecture, and uses in the wild**. Science Robotics, 7.

Macenski, S., Mart'in, F.J., White, R., and Clavero, J.G. (2020). **The Marathon 2: A Navigation System**. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2718-2725.

Macenski, S., Singh, S., Martín, F., and Ginés, J. (2023). **Regulated pure pursuit for robot path tracking**. Autonomous Robots, 47, 685 - 694.

Mohammad, R., Navid, M., Saeid, N. and Shady, M. (2021). **Review and Performance evaluation of path tracking controllers of autonomous vehicles.** IET Intelligent Transport Systems, vol. 15, no. 5, pp. 646-470.

Moore, T., and Daniel, W. S. (2014). **A Generalized Extended Kalman Filter Implementation for the Robot Operating System**. Annual Meeting of the IEEE Industry Applications Society (2014).

Nam TH, Shim JH, Cho YI. (2017). **A 2.5D Map-Based Mobile Robot Localization via Cooperation of Aerial and Ground Robots**. Sensors. 2017; 17(12):2730.

Nazari, V., and Naraghi, M. (2008). **Sliding mode fuzzy control of a skid steer mobile robot for path following**. 2008 10th International Conference on Control, Automation, Robotics and Vision, 549-554.

Newans, Josh. (2021). **Getting Ready for ROS Part 6: The Transform System (TF)**. [Online]. Available: https://articulatedrobotics.xyz/ready-for-ros-6-tf

Pardo-Castellote, G. (2003). **OMG Data-Distribution Service: architectural overview**. 23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings., 200-206.

Quigley, M. (2009). **ROS: an open-source Robot Operating System**. IEEE International Conference on Robotics and Automation.

Quigley, Morgan., Conley, Ken., Gerkey, Brian., Faust, Josh., Foote, Tully., Leibs, Jeremy., Wheeler, Rob,. and Ng, Andrew. (2009). **ROS: an open-source Robot Operating System**. ICRA Workshop on Open-Source Software. 3.

Rivera, Z.B., De Simone, M.C. and Guida, D. **Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments**. Machines 2019, 7, 42.

Takaya, Kenta., Toshinori, Asai., Valeri, Kroumov., and Florentin, Smarandache.(2016). **Simulation environment for mobile robots testing using ROS and Gazebo**. In 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), pp. 96-101. IEEE, 2016.

Tola, D. and Corke, P.(2023) **Understanding URDF: A Survey Based on User Experience**. IEEE 19th International Conference on Automation Science and Engineering (CASE), Auckland, New Zealand.

Vancea, A., and Orha, I. (2019). **A survey in the design and control of automated guided vehicle systems**. Carpathian Journal of Electronic and Computer Engineering, 12, 41 - 49.

Vishwas, N.S., Srikrishna, B.R., and Sudarshan, T.S.B. (2021). **ARC Nav -- A 3D Navigation Stack for Autonomous Robots**. ArXiv. /abs/2111.06923**.**

Wang, T., Wu, Y., Liang, J., Han, C., Chen, J., and Zhao, Q. (2015). **Analysis and Experimental Kinematics of a Skid-Steering Wheeled Robot Based on a Laser Scanner Sensor**. Sensors. 2015; 15(5):9681-9702.

Xu, Z., Guo, S., Song, T., and Zeng, L. (2020). **Robust Localization of the Mobile Robot Driven by Lidar Measurement and Matching for Ongoing Scene**. Applied Sciences. 2020; 10(18):6152.

ภาคผนวก ก

ข้อมูลจำเพาะของอุปกรณ์ที่เกี่ยวข้องและการประกอบหุ่นยนต์

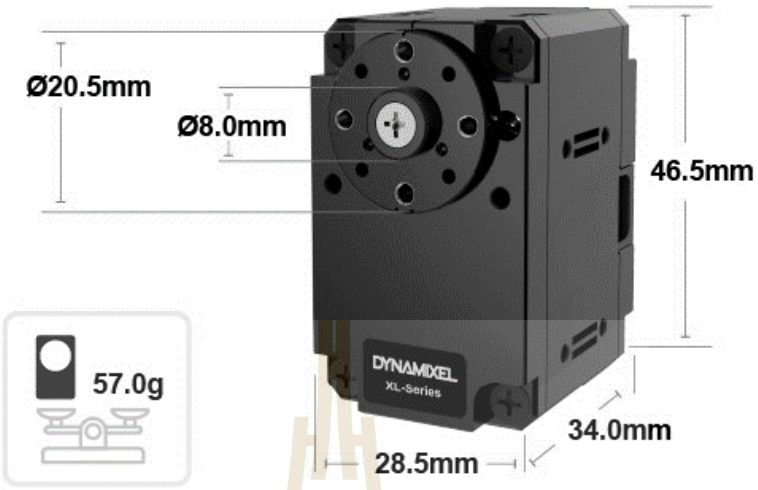
ก.1 บอร์ด Raspberry Pi 4 Model B

รูปที่ ก.1 บอร์ด Raspberry Pi 4 Model B

ตารางที่ ก.1 คุณสมบัติของบอร์ด Raspberry Pi 4 Model B

พารามิเตอร์	คุณสมบัติ
หน่วยประมวลผล	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
หน่วยความจำ (RAM)	1 GB, 2 GB ,4 GB และ 8 GB
พอร์ต USB	พอร์ตเชื่อมต่อ USB 3.0 จำนวน 2 พอร์ตและ USB 2.0 จำนวน 2 พอร์ต
พารามิเตอร์	**คุณสมบัติ**
พอร์ตเชื่อมต่อเพื่อแสดงผล	พอร์ตเชื่อมต่อเพื่อแสดงผลแบบ micro-HDMI จำนวน 2 พอร์ตรองรับความละเอียด 4K
การเชื่อมต่อแบบไร้สาย	การเชื่อมต่อแบบไร้สาย (Wireless) 2.4 GHz and 5.0 GHz มาตรฐาน IEEE 802.11ac wireless, Bluetooth 5.0 และ BLE Gigabit Ethernet
พลังงาน	ต้องการแหล่งจ่าย 5V DC ผ่ายสาย USB ชนิด C

ก.2 มอเตอร์ DYNAMIXEL รุ่น XL-430



รูปที่ ก.2 มอเตอร์ DYNAMIXEL XL-430 W250T

ตารางที่ ก.2 คุณสมบัติมอเตอร์ DYNAMIXEL XL 430 W250

พารามิเตอร์	คุณสมบัติ
หน่วยประมวลผล	ARM CORTEX-M3 (72 [MHz], 32Bit)
ระดับแรงดันไฟฟ้าใช้งาน	6.5 ถึง 12 โวลต์ (แรงดันไฟฟ้าที่แนะนำ 11.1 โวลต์)
การวัดตำแหน่งและความละเอียด	4,096 pulse/rev
อัตราทดเกียร์	258.5 : 1
แรงบิดสูงสุด	1.4 นิวตันเมตร (ที่แรงดัน 12 โวลต์ กระแส 1.4 แอมป์)
ความเร็วรอบสูงสุด	61 รอบต่อนาที (ที่แรงดัน 12 โวลต์)
การปรับตัวควบคุม PID	สามารถปรับได้ตั้งแต่ 0 ถึง 16383
รูปแบบตัวควบคุม	PI (Velocity Control), PID (Position Control)
การแปลงค่าตัวควบคุม	P/128, I/65536 และ D/16

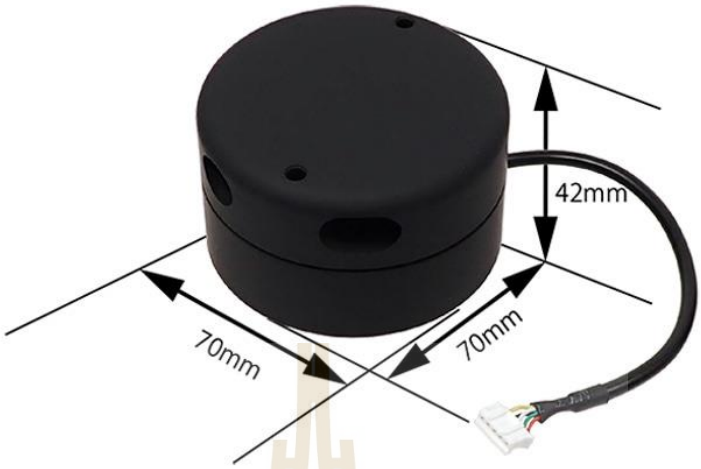
ก.3 เซนเซอร์ IMU WITMOTION รุ่น WT-901C

รูปที่ ก.3 เซนเซอร์ WITMOTION W901C

ตารางที่ ก.3 คุณสมบัติของเซนเซอร์ WITMOTION W901C

พารามิเตอร์	คุณสมบัติ
ความถี่ของข้อมูลเอาต์พุต	0.2 ถึง 200 Hz
ความเร็วในการส่งข้อมูล	สามารถปรับได้ระหว่าง 4800 ถึง 961200 bit/s
ช่วงองศาที่สามารถตรวจจับวัตถุได้	0 ถึง 360 องศา
เอาต์พุต	ความเร่งตามแนวแกนทั้งหมด 3 แกน, มุมองศา, ความเร็วเชิงมุม, สนามแม่เหล็ก, quaternion
ช่วงที่สามารถวัดได้	ความเร่ง (±6g), ความเร็วเชิงมุม (±2000°/s และ มุมองศา (X, Z-axis: ±180°, Y ±90°)
ความคลาดเคลื่อน	2.0 % ในระยะการตรวจจับ 1 ถึง 8 m
ความละเอียด	ความเร่ง (0.005g), ความเร็วเชิงมุม (0.61°/s) และ Magnet Field (16 bits)
ความแม่นยำการวัดมุม	X, Y-axis: 0.05° (Static) , X, Y-axis: 0.1° (Dynamic)

ก.4 เซนเซอร์ YDLIDAR G2

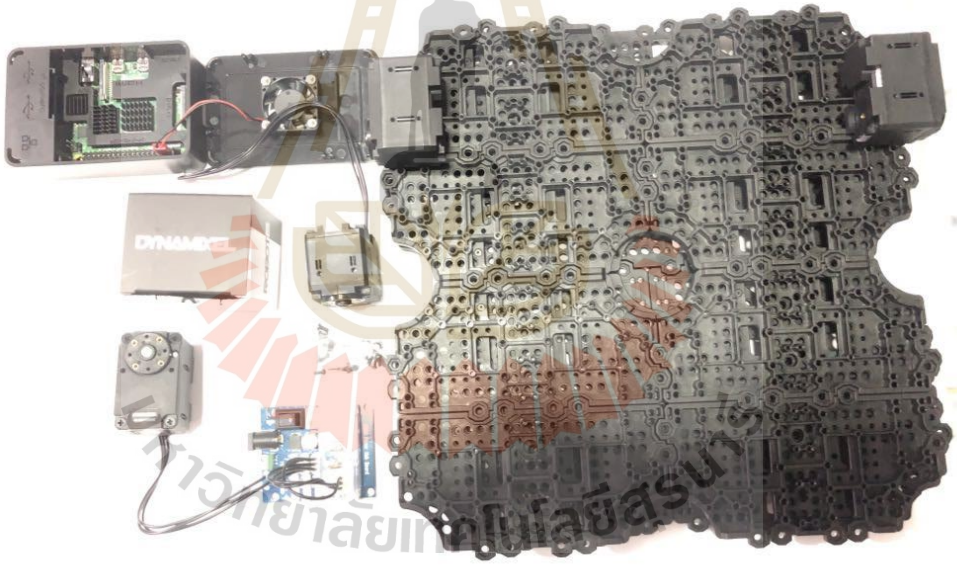


รูปที่ ก.4 เซนเซอร์ YDLIDAR G2

ตารางที่ ก.4 คุณสมบัติของเซนเซอร์ YDLIDAR G2

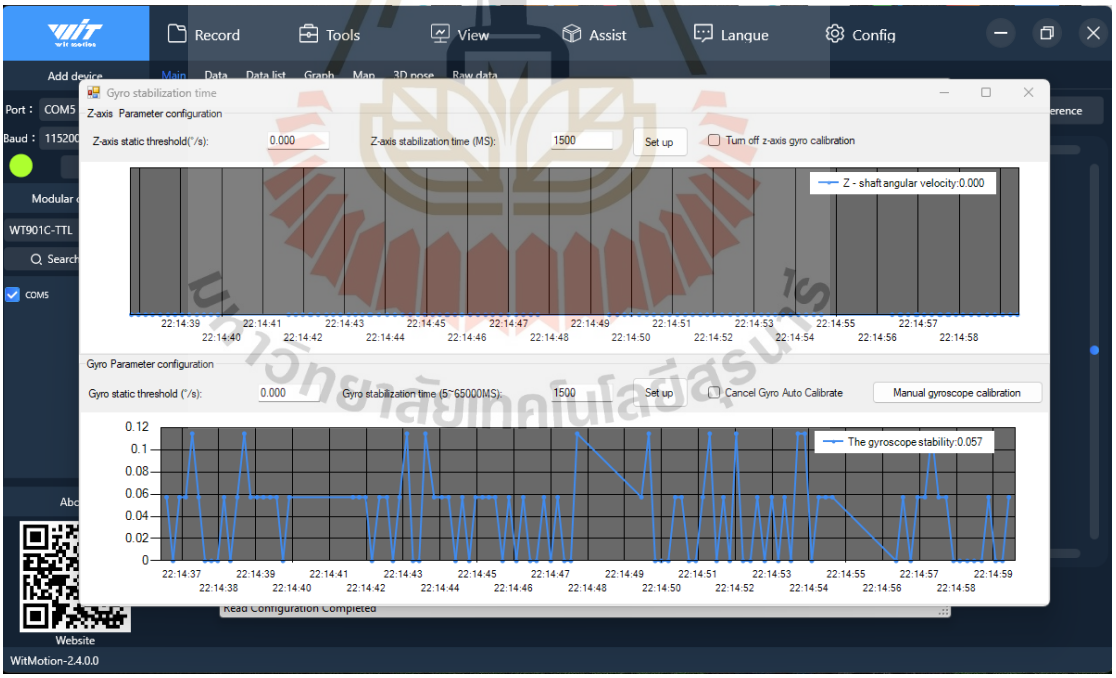
พารามิเตอร์	คุณสมบัติ
ความถี่ของมอเตอร์	สามารถปรับได้ 3 ค่าคือ 5, 7 และ 12 Hz
ระยะที่สามารถตรวจจับวัตถุได้	ในรัศมี 0.12 m ถึง 16 m
ช่วงองศาที่สามารถตรวจจับวัตถุได้	0 ถึง 360 องศา
ความละเอียดของมุม	0.36 องศา ที่ความถี่มอเตอร์เท่ากับ 5 Hz
	0.504 องศา ที่ความถี่มอเตอร์เท่ากับ 7 Hz
	0.864 องศา ที่ความถี่มอเตอร์เท่ากับ 12 Hz
ช่วงความเข้มการส่องสว่างของวัตถุ	0 ถึง 1023
ความคลาดเคลื่อน	2.0 % ในระยะการตรวจจับ 1 ถึง 8 m
แรงดันไฟฟ้า	4.8 ถึง 5.2 V
กระแสไฟฟ้าเมื่อเริ่มทำงาน	1000 mA
กระแสไฟฟ้าทำงาน	350 ถึง 500 mA

รูปที่ ก.5 การทดสอบการทำงานของมอเตอร์

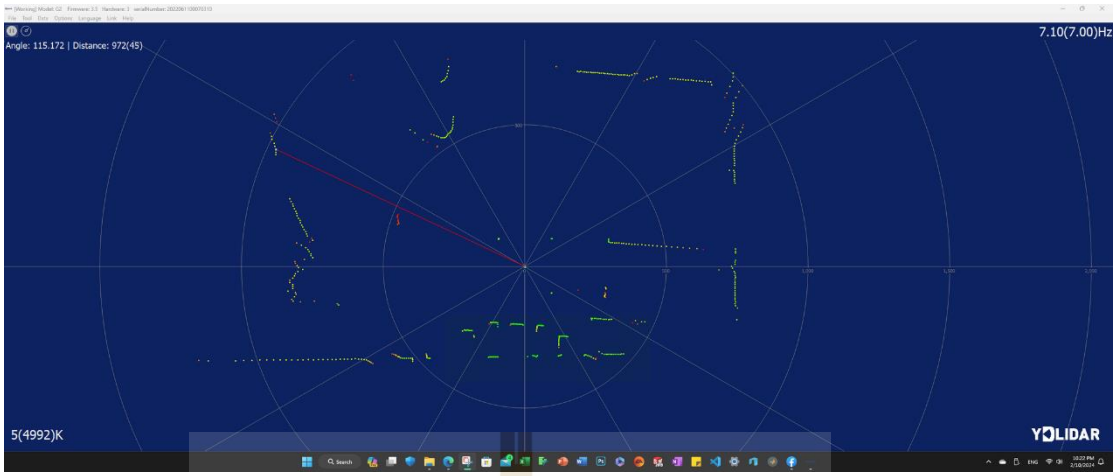


รูปที่ ก.6 ส่วนประกอบของหุ่นยนต์

รูปที่ ก.7 การทดสอบวัดค่าจากเซนเซอร์ไจโรสโคปด้วยโปรแกรม WitMotion



รูปที่ ก.8 การเปิดการทำงานระบบสอบเทียบไจโรสโคปด้วยโปรแกรม WitMotion



รูปที่ ก.9 การทดสอบเซนเซอร์ YDLIDAR G2 ด้วยโปรแกรม Lidar Viewer

ภาคผนวก ข

บทความวิชาการที่ได้รับการตีพิมพ์ในระหว่างการศึกษา

รายชื่อบทความวิชาการที่ได้รับการตีพิมพ์ในระหว่างการศึกษา

กิตติยศ ยะเจริญ, ศุภชัย แก้วพวง และ ทศพล รัตนนิยมชัย. (2566). การจำลองระบบนำทางอัตโนมัติของหุ่นยนต์ AMR โดยใช้โปรแกรม ROS และ Gazebo แบบ 3 มิติ. การประชุมวิชาการเครือข่าย วิศวกรรมไฟฟ้า ครั้งที่ 15. 2566(15), 211.

Yacharern, K., Musigapong, P., and Ratniyomchai, T. (2024). Development of Autonomic Docking Algorithm for Autonomous Mobile Robots by Integrated with Behavior Tree. 2024 International Electrical Engineering Congress (iEECON 2024) March 6-8, 2024, Pattaya Chonburi, THAILAND.

บทความวิจัย
การประชุมวิชาการเครือข่ายวิศวกรรมไฟฟ้า ครั้งที่ 15
15th Conference of Electrical Engineering Network 2023 (EENET 2023)

การจำลองระบบนำทางอัตโนมัติของหุ่นยนต์ AMR โดยใช้โปรแกรม ROS และ Gazebo แบบ 3 มิติ

Simulation of Automated Navigation Systems for AMR robot using ROS and Gazebo 3D Program

กิตติยศ ยะเจริญ[1] ศุภชัย แก้วพวง[2] และ ทศพล รัตนิยมชัย[3]
[1]สาขาวิชาวิศวกรรมแมคคาทรอนิกส์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี
111 ตำบลสุรนารี อำเภอเมืองนครราชสีมา นครราชสีมา 30000 E-mail: kittiyos2058@hotmail.com
[2]สาขาวิชาวิศวกรรมการจัดการพลังงานและโลจิสติกส์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี
111 ตำบลสุรนารี อำเภอเมืองนครราชสีมา นครราชสีมา 30000 E-mail: peper.supachai@gmail.com
[3]สาขาวิชาวิศวกรรมไฟฟ้า สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี
111 ตำบลสุรนารี อำเภอเมืองนครราชสีมา นครราชสีมา 30000 E-mail: tosaphol@sut.ac.th

บทคัดย่อ

บทความนี้นำเสนอการจำลองหุ่นยนต์ AMR เพื่อพัฒนาโปรแกรมสำหรับนำทางอัตโนมัติโดยใช้ระบบปฏิบัติการหุ่นยนต์ ROS และโปรแกรมจำลองผล Gazebo แบบ 3 มิติ เน้นวิเคราะห์การแก้ปัญหาความคลาดเคลื่อนของตำแหน่งหุ่นยนต์ของการวัดระยะทางแบบ Odometry ด้วยวิธี EKF เปรียบเทียบกับตำแหน่งจริงของหุ่นยนต์โดยจำลองการวิ่งแบบลูป และศึกษาผลกระทบการปรับค่าพารามิเตอร์การนำทาง 2 แบบ โดยวิธีที่หนึ่ง คือ Look-Ahead Distance ปรับค่าพารามิเตอร์ทดสอบที่ระยะ 0.3 m, 0.5 m, 1.5 m และ 2.5 m และแบบที่สอง คือ Inflation Radius หรือ *I* ปรับค่าพารามิเตอร์ทดสอบที่ระยะ 0.3 m, 1.0 m และ 1.3 m จากผลการจำลองพบว่าความคลาดเคลื่อนของการวัดระยะทางแบบ Odometry ด้วยวิธี EKF ลดลงร้อยละ 32 และค่าพารามิเตอร์ที่เหมาะสมในแบบแรก เท่ากับ 1.5 m และแบบที่สอง 1.3 m นอกจากนี้สามารถใช้เป็นหลักการเพื่อวิเคราะห์การปรับค่าพารามิเตอร์ด้วยโปรแกรมการจำลองหุ่นยนต์โดยใช้ ROS และ Gazebo แบบ 3 มิติ ให้เหมาะสมกับพื้นที่สภาพแวดล้อมการใช้งานจริง และประยุกต์ใช้สำหรับการพัฒนาหุ่นยนต์ต้นแบบ

คำสำคัญ: ระบบนำทางอัตโนมัติ, หุ่นยนต์ AMR, ระบบปฏิบัติการหุ่นยนต์, โปรแกรมจำลองผล Gazebo

Abstract

This paper presents a simulation of an autonomous mobile robot (AMR) robot to develop a simulated autonomous navigation program using robot operating system (ROS) and Gazebo simulator in 3D. It focuses on solving the problem of robot position discrepancy in odometry by extended kalman filter (EKF) method based on the actual position of the robot by circular motion and study the effects of adjusting 2 navigation parameters. Firstly, Look-Ahead Distance would adjust the test parameter at 0.3 m, 0.5 m, 1.5 m, and 2.5 m. Secondly, Inflation Radius would adjust the test parameter at 0.3 m, 1.0 m and 1.3 m. From the simulation results, the error of odometry by EKF method is reduced by 32 %, the optimum parameters in the first and second types were 1.5 m and 1.3 m, respectively. Furthermore, it can be used as a formality to analyze parameter adjustments with robot simulation software using ROS and Gazebo in 3D to suit the real-world environment and can be applied for the development of prototype robots.

Keywords: Autonomous Navigation System, AMR Robot, Robot Operating System (ROS), Gazebo Simulator

1. บทนำ

ในโรงงานอุตสาหกรรมส่วนใหญ่โดยเฉพาะอุตสาหกรรมผลิตชิ้นส่วนรถยนต์ ระบบขนส่งในไลน์ผลิตนิยมใช้รถ Automated Guided Vehicle (AGV) ถูกใช้กันอย่างแพร่หลายและเพิ่มเป็นจำนวนมากขึ้นในแต่ละปี ข้อดี คือ เป็นพาหนะที่ใช้ขนส่งภายในอาคารแบบไร้คนขับโดยอาศัยการนำทางที่ใช้แบบหลัก ๆ เช่น Reflective Marker, Frequency ID, Magnetic Tape, Vision และ Laser Guidance เป็นต้น ส่งผลให้เกิดปัญหาของรถ AGV ที่มีข้อจำกัดอย่างมากในการเปลี่ยนเส้นทาง เนื่องจากมีโครงสร้างของระบบนำทางที่ซับซ้อน ทำให้ระยะเวลาในการติดตั้งระบบในกรณีที่ต้องการปรับเปลี่ยนเส้นทางหรือเพิ่มเส้นทางใหม่ มีความยากลำบากและมีต้นทุนค่าใช้จ่ายที่สูงพอสมควร [1],[2]

ปัญหาดังกล่าวการออกแบบและพัฒนาหุ่นยนต์เคลื่อนที่อัตโนมัติ Autonomous Mobile Robot (AMR) ช่วยให้มีข้อได้เปรียบในเรื่องของการเปลี่ยนเส้นทางได้ง่ายกว่ารถ AGV, J. Pechiar [3] กล่าวถึงการนำทางแบบ Trackless Natural Navigation ที่เป็นระบบเซ็นเซอร์

บทความวิจัย
การประชุมวิชาการเครือข่ายวิศวกรรมไฟฟ้า ครั้งที่ 15
15th Conference of Electrical Engineering Network 2023 (EENET 2023)

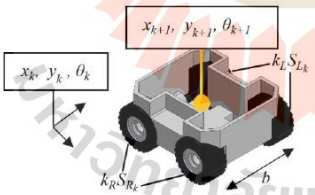
ตรวจจับสภาพแวดล้อมเพื่อใช้ระบุเส้นทางที่กำหนดในแผนที่ W. Xue และคณะ [4] ได้เสนอการปรับเปลี่ยนเส้นทางทำได้ง่ายและสะดวกโดยใช้วิธี Simultaneous Localization and Mapping (SLAM) หุ่นยนต์เคลื่อนที่อัตโนมัติ และสามารถนำทางโดยใช้ข้อมูลจากแผนที่เพื่อหลบหลีกสิ่งกีดขวางทั้งสิ่งกีดขวางแบบอยู่กับที่ (Static Obstacle) และสิ่งกีดขวางแบบเคลื่อนที่ (Dynamic Obstacle) เพื่อความปลอดภัยระหว่างหุ่นยนต์ทำงาน Shusheng Bi และคณะ [5] กล่าวถึงการปรับค่าพารามิเตอร์การวัดระยะ Odometry ของตำแหน่งหุ่นยนต์นำทางไปยังจุดหมายในแผนที่ได้อย่างมีประสิทธิภาพ นอกจากนี้ เฉลิมชนม์ สถิระพจน์ และคณะ [6] เสนอการประมวลผลข้อมูลสำหรับการหาตำแหน่งด้วย GPS แบบสถิติด้วยวิธี EKF และเปรียบเทียบประสิทธิภาพการคำนวณค่าเชิงตัวเลขที่ใช้ในการหาตำแหน่งด้วย GPS แบบจุดเดียว

ในบทความนี้นำเสนอการจำลองหุ่นยนต์นำทางอัตโนมัติ โดยใช้โปรแกรม ROS และ Gazebo แบบ 3 มิติ เพื่อแก้ปัญหาตำแหน่งหุ่นยนต์ที่คลาดเคลื่อนโดยวัดระยะทางของ Odometry ด้วยวิธี EKF ศึกษาผลกระทบการปรับค่าพารามิเตอร์แบบ Look-Ahead Distance และ Inflation Radius เพื่อนำไปประยุกต์ใช้งานกับสภาพแวดล้อมพื้นที่จริงให้เหมาะสมและมีประสิทธิภาพสูงสุด

2. แบบจำลองทางคณิตศาสตร์

2.1 การวัดระยะทางแบบ Odometry

การวัดระยะทางแบบ Odometry เป็นการสอบเทียบระยะตำแหน่งหุ่นยนต์ที่จำลองในโปรแกรมกับหุ่นยนต์จริงที่พัฒนา ดังรูปที่ 1 เป็นแบบจำลองการระบุตำแหน่งของหุ่นยนต์เพื่อใช้เป็นหลักการคำนวณพิกัดของหุ่นยนต์เคลื่อนที่ออกจากจุดเริ่มต้น โดยคำนวณหาพิกัด x_{k+1} และ y_{k+1} ของหุ่นยนต์ ในหน่วย m และ θ_{k+1} เป็นมุมทิศการหมุนในหน่วย radian คำนวณพิกัดที่เปลี่ยนไปกรณีหุ่นยนต์เคลื่อนที่ ดังสมการที่ (1) – (3) [5]



รูปที่ 1 แบบจำลองการระบุตำแหน่งของหุ่นยนต์

$$x_{k+1}=x_k+\left(\frac{k_R S_{R_k}+k_L S_{L_k}}{2}\right)\cos\left(\theta_k+\frac{k_R S_{R_k}-k_L S_{L_k}}{2b}\right) \quad (1)$$

$$y_{k+1}=y_k+\left(\frac{k_R S_{R_k}+k_L S_{L_k}}{2}\right)\sin\left(\theta_k+\frac{k_R S_{R_k}-k_L S_{L_k}}{2b}\right) \quad (2)$$

$$\theta_{k+1}=\theta_k+\left(\frac{k_R S_{R_k}-k_L S_{L_k}}{b}\right) \quad (3)$$

โดยที่ x_k, y_k, θ_k คือ พิกัดและทิศการหมุนของหุ่นยนต์ ณ เวลาปัจจุบัน x_{k+1}, y_{k+1}, θ_{k+1} คือ พิกัดและทิศการหมุนของหุ่นยนต์เคลื่อนที่ออกจากจุด x_k, y_k และ S_{R_k}, S_{L_k} คือ สัญญาณพัลส์ (Pulse) ของ Encoder ล้อซ้ายและขวาของหุ่นยนต์ k_R, k_L คือ สัมประสิทธิ์หน้าตัวแปร S_{R_k}, S_{L_k} และ b คือ ระยะระหว่างล้อซ้ายและขวาของหุ่นยนต์

2.2 Extended Kalman Filter (EKF)

EKF คือ การประมาณค่าสมการเชิงเส้นของตัวแปรในระบบพลวัต (State Vector System) ณ เวลาปัจจุบัน ด้วยข้อมูลเวลาก่อนหน้าและเวลาถัดไป ซึ่งหลักการของ EKF ใช้แก้ปัญหาสัญญาณรบกวน และใช้ปรับค่าอินพุตของเซนเซอร์ให้มีค่าใกล้เคียงกับความเป็นจริงมากที่สุด ระบบพลวัตมีสมการสัมพันธ์แปรผันกับเวลา ณ เวลา k ดังสมการที่ (4) และสอดคล้องกับเวกเตอร์ค่าสังเกต (Measurement vector, z_k) ดังสมการที่ (5) [6],[7]

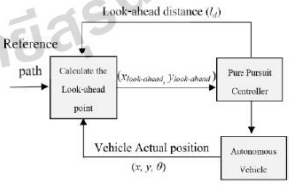
$$x_{k+1}=\phi_k x_k+w_k \quad (4)$$

$$z_k=H_k x_k+v_k \quad (5)$$

โดยที่ w_k คือ System noise, v_k คือ Measurement noise, k คือ เวลา ณ ขณะนั้น ϕ_k คือ Transition matrix และ H_k คือ Measurement connection matrix

2.3 Pure Pursuit Algorithm (PPA)

กระบวนการ Pure Pursuit Algorithm คือ ตัวป้อนคำสั่งการนำทางของหุ่นยนต์อัตโนมัติ โดยสั่งการในรูปแบบความเร็วเชิงเส้นและความเร็วเชิงมุม กระบวนการทำงานเชิงหลักการของ PPA ดังรูปที่ 2 [8]



รูปที่ 2 Pure Pursuit Algorithm Diagram

บทความวิจัย

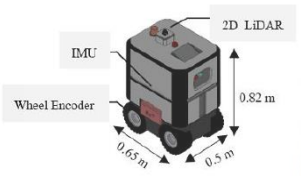
การประชุมวิชาการเครือข่ายวิศวกรรมไฟฟ้า ครั้งที่ 15

15th Conference of Electrical Engineering Network 2023 (EENET 2023)

3. ระเบียบวิธีและผลการจำลอง

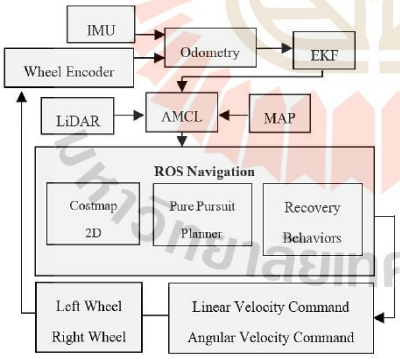
3.1 การจำลองหุ่นยนต์นำทางอัตโนมัติ

หุ่นยนต์ที่นำมาจำลองเพื่อใช้ทดสอบระบบนำทางอัตโนมัติ ออกแบบโดยใช้โปรแกรม Solidworks ประกอบด้วยล้อทั้งหมด 4 ล้อ ขับเคลื่อนแบบ Differential Drive ติดตั้งเซนเซอร์ LiDAR, Inertial Measurement Unit (IMU) และ Encoder โดยกำหนดขนาดของหุ่นยนต์และตำแหน่งของเซนเซอร์ ดังรูปที่ 3



รูปที่ 3 หุ่นยนต์นำทางอัตโนมัติออกแบบโดยใช้โปรแกรม Solidworks

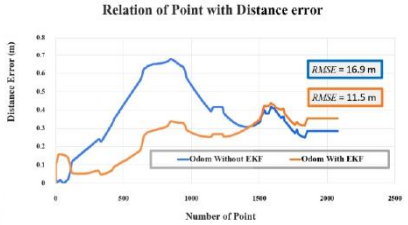
การจำลองหุ่นยนต์ที่ออกแบบใช้โปรแกรม ROS และ Gazebo ด้วยวิธีการ EKF กับการวัดระยะ Odometry ควบคุมการทำงาน เพื่อระบุตำแหน่งบนแผนที่ โดยใช้หลักการ Adaptive Monte Carlo Localization (AMCL) ร่วมกับ LiDAR แก้ปัญหาความคลาดเคลื่อนบนแผนที่ โดยใช้ Pure Pursuit Planner (PPP) นำทางหุ่นยนต์แบบอัตโนมัติ จำลองในแผนที่ด้วยวิธีการ SLAM และปรับพารามิเตอร์เพื่อตรวจจับสิ่งกีดขวางด้วย Costmap 2D ดังรูปที่ 4 สำหรับการจำลองใช้ซอฟต์แวร์ ROS 2 foxy, Gazebo, Ubuntu Mate 20.04 และคอมพิวเตอร์ CPU: i7-7700K, RAM: 16GB



รูปที่ 4 ระบบควบคุมการทำงานผ่านโปรแกรม ROS และ Gazebo

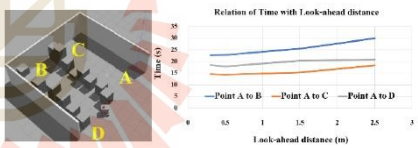
3.2 ผลการจำลอง

การเปรียบเทียบการวัดระยะ Odometry ด้วยวิธี EKF ดังรูปที่ 5 สามารถลดค่า Root Mean Square Error (*RMSE*) ได้ 5.4 m (กราฟเส้นสีส้ม มีค่า *RMSE* 11.5 m) ลดความคลาดเคลื่อนในการระบุตำแหน่งได้ร้อยละ 32 (เปรียบเทียบกับกราฟเส้นสีน้ำเงิน มีค่า *RMSE* 16.9 m) และค่าที่ได้สามารถนำไปพัฒนาใช้งานกับหุ่นยนต์จริง แต่ต้องคำนึงถึงปัจจัยอื่นที่ทำให้เกิดความผิดพลาดของฮาร์ดแวร์และการปรับพารามิเตอร์ที่เกี่ยวข้อง

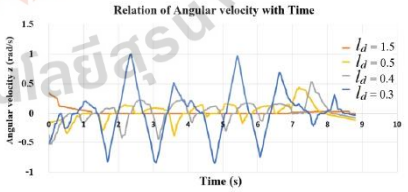


รูปที่ 5 เปรียบเทียบการวัดระยะ Odometry

การปรับพารามิเตอร์ l_d ที่ระยะ 0.3 m, 0.5 m, 1.5 m และ 2.5 m ที่ความเร็วคงที่ของหุ่นยนต์ 0.8 m/s เวลาในการนำทางจากจุด A ไป B, จุด A ไป C และจุด A ไป D โดยใช้โปรแกรม Gazebo ดังรูปที่ 6 และเปรียบเทียบผลกระทบของ l_d ต่อความเร็วเชิงมุม (ω) ดังรูปที่ 7 ของหุ่นยนต์ในกรณีที่ l_d มีค่าเท่ากับ 0.3 m, 0.4 m, 0.5 m และ 1.5 m ตามลำดับ



รูปที่ 6 ตำแหน่งที่ใช้นำทางในโปรแกรม Gazebo และผลกระทบของ l_d



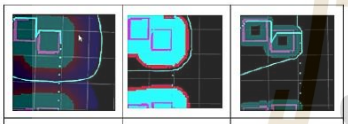
รูปที่ 7 ผลกระทบของ l_d ต่อ ω ของหุ่นยนต์

บทความวิจัย

การประชุมวิชาการเครือข่ายวิศวกรรมไฟฟ้า ครั้งที่ 15

15th Conference of Electrical Engineering Network 2023 (EENET 2023)

จากการจำลอง Inflation radius พบว่าการลดค่า l_d สามารถลดเวลาการวิ่งไปยังจุดหมายและมีความแม่นยำ ซึ่งการปรับค่า l_d น้อยเกินไป ส่งผลกระทบต่อ ω ทำให้หุ่นยนต์เกิดการแกว่ง (ทดสอบวิ่งจากจุด A ไป C) ดังรูปที่ 7 ค่า l_d ที่เหมาะสมมีค่าเท่ากับ 1.5 m ในทางกลับกันกรณีที่ปรับค่า l_d สูงเกินไปเท่ากับ 3 m ส่งผลกระทบต่อกรณีวิ่งทางโค้งจะ ทำให้หุ่นยนต์หลุดโค้งและใช้เวลานานขึ้นในการไปสู่จุดหมาย ปัจจัยที่สำคัญต่อการนำทางและไม่ให้ชนสิ่งกีดขวาง คือ การใช้ Costmap ทดสอบโดยการปรับค่า Inflation radius กำหนดเป็นตัวแปร I เท่ากับ 1.3 m, 1.0 m และ 0.3 mโดยสร้างเส้นทางให้หุ่นยนต์เคลื่อนที่ผ่าน โดยใช้โปรแกรม ROS ดังรูปที่ 8 การปรับค่าค่า I สามารถปรับได้หลากหลายขึ้นอยู่กับความกว้างของตัวหุ่นยนต์ ค่า I ที่เหมาะสมสำหรับหุ่นยนต์สามารถเคลื่อนที่ผ่านสิ่งกีดขวางมีค่าเท่ากับ 1.3 m



I = 1.3 m	I = 1.0 m	I = 0.3 m

รูปที่ 8 เปรียบเทียบการเปลี่ยนค่า Inflation radius

4. สรุป

การจำลองหุ่นยนต์นำทางอัตโนมัติ AMR ออกแบบหุ่นยนต์ต้นแบบด้วยโปรแกรม Solidworks ทดสอบการนำทางโดยใช้โปรแกรม ROS และ Gazebo แบบ 3 มิติ สามารถแก้ปัญหาความคลาดเคลื่อนของตำแหน่งหุ่นยนต์ของการวัดระยะทางแบบ Odometry ด้วยวิธี EKF ลดลง ร้อยละ 32 ผลกระทบจากการปรับค่าพารามิเตอร์การนำทางแบบ Look-Ahead Distance สามารถปรับค่าพารามิเตอร์ที่ระยะเหมาะสม 1.5 m และแบบ Inflation Radius สามารถปรับค่าพารามิเตอร์ที่ระยะเหมาะสม 1.3 m

เอกสารอ้างอิง

[1] S. Vonolfen, M. Kofler, A. Beham, M. Affenzeller and W. Achleitner, "Optimizing assembly line supply by integrating warehouse picking and forklift routing using simulation," Proceedings of the 2012 Winter Simulation Conference (WSC), 2012, pp. 1-12, doi: 10.1109/WSC.2012.6465077.

[2] S. M. M. Rahman, "Forklift Routing Optimization in a Warehouse using a Clustering-based Approach," Presented in Partial Fulfillment of the Requirements For the Degree of Master of Applied Science in Industrial Engineering, May 2019.

[3] J. Pechiar, "Architecture and design considerations for an autonomous mobile robot," 2021 IEEE URUCON, Montevideo, Uruguay, 2021, pp. 343-346.

[4] W. Xue, R. Ying, Z. Gong, R. Miao, F. Wen and P. Liu, "SLAM Based Topological Mapping and Navigation," 2020 IEEE/ION Position, Location and Navigation Symposium (PLANS), Portland, OR, USA, 2020, pp. 1336-1341, doi: 10.1109/PLANS46316.2020.9110190.

[5] Bi, S.; Yang, D.; Cai, Y. Automatic Calibration of Odometry and Robot Extrinsic Parameters Using Multi-Composite-Targets for a Differential-Drive Robot with a Camera. Sensors 2018, 18, 3097. https://doi.org/10.3390/s18093097.

[6] เฉลิมชนม์ สถิระพจน์ และ มยุรา ล้วนเส้ง. (2550). การศึกษาเปรียบเทียบวิธี Extended Kalman filter และวิธีกำลังสองน้อยที่สุดที่ใช้ ในการหาตำแหน่งด้วยจีพีเอสแบบจุดเดี่ยวที่ให้ความละเอียดสูง. วิศวกรรมสาร มก., 20 (61), 38-46.

[7] Welch, G. and Bishop G. (2007) "An Introduction to Kalman Filter [Online]."Department of Computer Science, University of North Carolina at Chapel Hill. Available from: http://www.cs.unc.edu/.

[8] R. Wang, Y. Li, J. Fan, T. Wang and X. Chen, "A Novel Pure Pursuit Algorithm for Autonomous Vehicles Based on Salp Swarm Algorithm and Velocity Controller," in IEEE Access, vol. 8, pp. 166525-166540, 2020.

ประวัติผู้เขียนบทความ

นาย กิตติยศ ยะเจริญ
นักศึกษาสาขาวิชาวิศวกรรมแมคคาทรอนิกส์
มหาวิทยาลัยเทคโนโลยีสุรนารี
งานวิจัยที่สนใจ : ระบบขับเคลื่อนอัตโนมัติ

นาย ศุภชัย แก้วพวง
นักศึกษาสาขาวิชาวิศวกรรมการจัดการพลังงานและโลจิสติกส์ มหาวิทยาลัยเทคโนโลยีสุรนารี
งานวิจัยที่สนใจ : เทคโนโลยียานยนต์ไฟฟ้า

ผศ.ดร. ทศพล รัตนนิยมชัย
อาจารย์ประจำสาขาวิศวกรรมไฟฟ้า
มหาวิทยาลัยเทคโนโลยีสุรนารี
งานวิจัยที่สนใจ : เทคโนโลยียานยนต์ไฟฟ้า

2024 International Electrical Engineering Congress (iEECON 2024)
March 6-8, 2024, Pattaya Chonburi, THAILAND

Development of Autonomic Docking Algorithm for Autonomous Mobile Robots by Integrated with Behavior Tree

Kittiyos Yacharern
School of Mechatronics Engineering
Suranaree University of Technology
Nakhon Ratchasima, Thailand
kittiyos2058@hotmail.com

Pirutchada Musigapong
School of Occupational Health&Safety
Suranaree University of Technology
Nakhon Ratchasima, Thailand
pirutchada@sut.ac.th

Tosaphol Ratniyomchai
School of Electrical Engineering
Suranaree University of Technology
Nakhon Ratchasima, Thailand
tosaphol@sut.ac.th

***Abstract*— This paper is to introduce the development of autonomic docking for autonomous mobile robots (AMR) using behavior trees (BT) and gazebo simulation software in a ROS2 environment. The virtual robot in this study is capable of moving around and locating the charging station by using the camera to identify the ARTag on the map generated through simultaneous localization and mapping (SLAM) data. BT is used to develop an autonomous docking algorithm and compared three different scenarios through experiments with the AMR. In the experiments, the parameters were adjusted when the AMR was positioned 2 m away from the charging station along the x-axis, with no discrepancy along the y-axis. Additionally, the parameters were tweaked when there was a y-axis discrepancy of ± 0.25 m in cases 2 and 3. The results indicated that modifying the heading tolerance can reduce docking errors and improve the efficiency of the algorithm. Moreover, using BT to design algorithms can enhance functionality and simplify the system's operation.**

Keywords—Behavior Tree, Gazebo, Robot Operating System 2, Autonomous Mobile Robots

I. Introduction

In the 1950s, the earliest industrial robots were operated by programmable devices without external control or communication capabilities. During that time, robots were controlled using basic non-servo controller hardware, which resulted in the arm producing a loud sound when it made contact with the mechanism. Over time, robots have evolved and are now being utilized in more complex systems [1], such as those involving mechanical arms and docking systems, survey robots, or robots that can autonomously adapt their behaviors as required and change in alternative environment. The Behavior Tree (BT) has become a preferred control structure in the robotics and video game industries [2, 3], mainly due to its ability to handle complex systems. The advantages of this control structure are evident in the design behavior diagram, where BT showcases conditionally programmable leaf nodes, prioritizing behavior over state or activities to present a logical. To execute the behavior diagram, different behaviors are switched between in response to changes. Furthermore, BTs enable the reuse of code for incremental function design and effective function testing [4, 5]. This feature is employed in creating controls for non-player characters (NPCs) in game interfaces [6]. Instead of solely relying on code, BTs are used to direct the actions of NPC characters, serving as an interface between high-level AI systems and low-level controllers for continuous actions like movement and decision-making. Finally, the safety aspect concerns with the human perception when working with robots as well as safety guidelines, such as safety clearance, and standard operation load and speed, which may be managed in an advanced algorithm, resulting in zero accident or damage.

Docking involves bringing two entities together and establishing a secure connection. This process requires precise maneuvering and synchronization of multiple subsystems [7], including perception, planning, control, and feedback. BTs offer a structured approach to modeling and executing complex behaviors by organizing them into a hierarchical framework of tasks, conditions, and actions. The hierarchical structure of BT provides flexibility, reusability, and scalability [8], making them an ideal choice for managing the complexities involved in docking procedures.

The objective of this study is to develop AMR docking system that can autonomously navigate to a charging dock using BT. The AMR utilizes its front camera and a navigation system that operates on a 2D Map generated from SLAM [9] algorithms. These algorithms, such as Adaptive Monte Carlo Localization (AMCL) [10] and Odometry Algorithms, are used to determine the robot's location. The study compares how variations in relevant parameters, such as heading error and the initial position of the AMR, affect the performance of the autonomic docking algorithm. The results are then analyzed using simulations to evaluate the advantages of combining BT with the autonomic docking algorithm.

II. The Navigation And Simulation System

A. XML Formats for Structuring AMR

Programming is required to specify the components of an AMR model for simulation. It takes the form of URDF (Unified Robot Description Format), which uses the XML (Extensible Markup Language) language to define the physical characteristics of the robot, the position of the installed sensor, and the specification of the sensor. This approach enables realistic and comprehensive testing of AMR performance, which aids in the development and assessment of advanced algorithms and control strategies. As illustrated in Fig.1, the usual form of differential drive robot is continuous, which means that all wheels may revolve 360 degrees, and fixed, which means that other sections do not move.

979-8-3503-8359-1/24/$31.00 ©2024 IEEE

2024 International Electrical Engineering Congress (iEECON 2024)
March 6-8, 2024, Pattaya Chonburi, THAILAND

B. AMR Navigation System

The AMR robot in this paper uses ROS2 as its operating system to simulate the system coupled with the Gazebo environment. The simulated AMR is divided into three parts. To make the robot aware of its surroundings, the initial component is a sensor system that contains a 2D LiDAR, Wheel Encoder, and IMU (Inertial Measurement Unit). The second component is an AMR framework made up of a chassis, four wheels, and two battery boxes constructed in XML format. The AMR movement control algorithm, which uses the Pure pursuit Planner algorithm to compute a route to sense based on data from the 2D map, is the third component. Finding obstacles and calculating robot movements are implemented to determine the AMR on the map using an AMCL (Adaptive Monte Carlo Localization) based on the odometry data for Robot localization on a 2D Map. The overall of AMR navigation system is shown in Fig. 2.

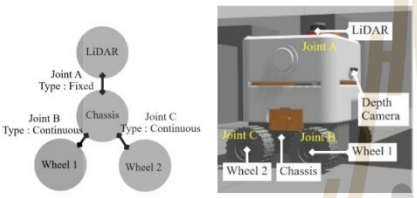


Fig. 1. A URDF example used to define a robot structure.

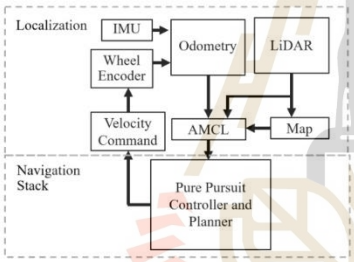


Fig. 2. Overall of AMR Navigation System.

III. RELATED WORK

A. Autonomic Docking

Goal distance error (d_{error}), Heading Error (α_{error}), and Yaw Error (θ_{error}) are all important control elements in the autonomous docking simulation. These variables were used to establish the parameters for regulating the AMR motion in terms of linear and angular velocity. The parameters of α_{error}, d_{error} and θ_{error} can be determined using the equation below.

$$\alpha_{error} = \alpha_{desired} - \theta_{yaw} \quad (1)$$

$$d = \sqrt{(x_{goal} + x)^2 - (y_{goal} - y)^2} \quad (2)$$

$$\theta_{error} = \theta_{desired} - \theta_{yaw} \quad (3)$$

The variable $\alpha_{desired}$ represents the desired heading inaccuracy (heading tolerance) in radians for checking heading angle to control the angular velocity for the initial state, while θ_{yaw} representing the angle of rotation of the AMR in the coordinate system based on the x-axis. (x_{goal}, y_{goal}) is the goal coordinate and (x, y) is the AMR position. The yaw tolerance contains the variable $\theta_{desired}$ is the desired yaw error. used to calculate with θ_{yaw} to control the AMR heading by rotating when $\theta_{desired} \neq \theta_{yaw}$ before finished docking, All the above parameters are determined by the distance of between the charging station and AMR position by $d_{desired}$, while the actual distance is calculated as d_{error} ,The calculation from the parameters used to control linear velocity v_x and angular velocity ω as illustrated in Fig. 3.

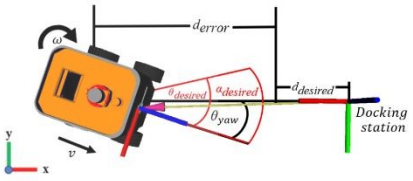


Fig. 3. Related variables in Autonomic Docking.

B. Behavior Tree

A behavior tree is represented graphically as a directed tree with nodes classified as root [12], control flow nodes, or execution nodes (tasks). For each linked pair of nodes, the outgoing node is referred to as the parent, and the incoming node is referred to as the child. The root has no parents and precisely one child, the control flow nodes have one parent and at least one child, and the execution nodes have one parent but no children. From left to right, a control flow node's child is arranged graphically below it. The execution of a BT begins with the root, which transmits ticks to its child at a predetermined node. A tick is an enabling signal that allows a child to be performed. When a node in the BT is allowed to execute, it returns to the parent a status of running if its execution has not yet completed, success if it has achieved its purpose, or failure otherwise. The operation of BT can be demonstrated using a simple Control node, as shown in Fig. 4.

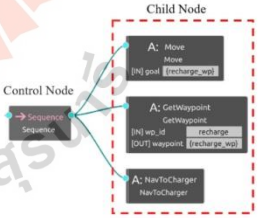


Fig. 4. Structure of Behavior Tree for Autonomic Docking.

According to the BT diagram, a child node is connected to a control node (Sequence). The operation can be explained as follows. When the sequence node is triggered, the sequence node will tick to the Move node until the Move node is in the SUCCESS state, after which

2024 International Electrical Engineering Congress (iEECON 2024)
March 6-8, 2024, Pattaya Chonburi, THAILAND

the sequence ticks GetWaypoint until GetWaypoint is in the SUCCESS state, and so is NavToCharger. There are also plenty of nodes that have different operating conditions that can be summarized in Table I.

TABLE I. LEAF NODE TYPES AND STATUS

Leaf Node Type	SUCCESS	FAILURE
Action (Execution node)	Action Completed	Action not Completed
Condition (Execution node)	If true	If false
Sequence	All child succeeds	A child fails
Fallback	A child succeeds	All child fail

The C++ library is used in this study to implement behavior trees as Behavior Tree CPP V4.1 on ROS2, which allows GROOT to visualize the tree node.

IV. RESULT AND DISCUSSION

A. *Autonomic docking process*

An environment for simulating the autonomic docking system is shown in Fig. 5. To evaluate the performance of the robot in a warehouse environment, the AMR is deployed in the area while utilizing SLAM to collect map data. The Pure Pursuit Planner is employed to control the robot's movements based on this map data. Additionally, a program is created to simulate a battery with a capacity below 25%. A Behavior Tree (BT) is used to manage the AMR's behavior, ensuring it navigates to the coordinates (-1.721, -4.131) by activating the GetWaypoint node followed by the Move node. The BT also includes the Nav2Charger node to initiate control of the AMR by Autonomic docking algorithm as outlined in the BT diagram shown in Fig. 6.

Fig. 5. Simulated Environment for Autonomic Docking System.

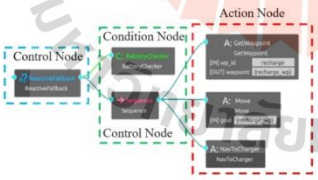


Fig. 6. Behavior Tree for autonomic docking algorithm.

From the GROOT visualization, the initial process will be executed when the batteryChecker condition node can detect the battery of an AMR less than 25% given a false condition Then the sequence node will be executed. So that is the start of the Action node group by respectively GetWaypoint and the last process is NavToCharger.

Consider the leaf node of NavToCharger. An algorithm has been written to control its behavior by using conditions from autonomic docking parameters can be as shown in the pseudo-code Table. II.

TABLE II. AUTONOMIC DOCKING PSEUDO-CODE

Input: ***heading error, distance error, yaw error,*** V_x ,ω

==========INITIAL STATE==========

```
if distance error > desired distance
    if heading error > heading tolerance
        if heading error > 0
            control angular velocity = ω radians/s
        else
            control angular velocity = - ω radians/s
    else
        control linear velocity = Vx m/s
==========FINAL STATE==========
else if yaw error > yaw tolerance
        if yaw error > 0
            do control angular velocity = ω radians/s
        else
            do control angular velocity = - ω radians/s
else
    print("Finished Docking")
    control linear velocity = 0 m/s
    control angular velocity = 0 radians/s
end
```

==========FINISH DOCKING STATE==========

In the provided pseudo-code, v_x denotes the linear velocity used to regulate an AMR travel in either a forward or reverse direction, while ω denotes the angular velocity employed for rotating the AMR clockwise when it's positive or counterclockwise when it's negative. Within the NavToCharger leaf node, three states are encompassed, including the initial state, a final state, and a finished docking state. The initial state is responsible for controlling the AMR straightforwardly by evaluating α_{error} in radians , and the final state is influenced by the θ_{error} variable, Both states are examined to ensure that the AMR stops away from the charging station based on $d_{desired}$. It's important to note that the navigation stack maintains an acceptable tolerance of 0.25 m to stop at the destination while remaining at least 2 meters away from the charging station. As a result, the experiment was divided into two parts, the first part aimed to test charging accuracy within the acceptable tolerance set by the navigation stack, and the second part was designed to observe the impact of reducing speed on accuracy.

B. *Navigation stack destination tolerance experiment*

The First Experiment will be set v_x= 0.25 m/s and ω 0.2 radians/s to navigate to the charging station. The results will be collected under three distinct conditions while maintaining a constant d_{error} = 0.5 m, Initially, the AMR will be positioned 0.5 meters away from the charging station, and its θ_{yaw}will be set to 3.14 radians ,directing it to face in that direction. Under the first case, the AMR must park in front of the charging station, maintaining a 2-meter distance from it, and aligning itself precisely with the y-axis reference of the charging station. With no discrepancies from the y-axis reference on the charging station and set to $\alpha_{desired}$ = 0.01,0.03, and 0.04 respectively. Determined the second and third case by the

AMR to be in a veer position fluctuating within y = ± 0.25 m and ± 0.1244 radians (±7.12 deg). the AMR would need to perform a half-turn from its initial position by checking with the heading error before the final state by setting the variation of the parameter $\alpha_{desired}$ to the same value as the first. This experiment can be explained as shown in Fig. 7.

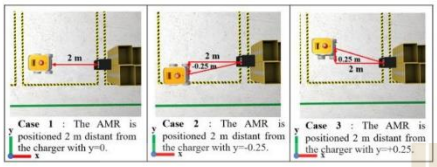


Fig. 7. Condition for Autonomic Docking Experiment.

Gathering the algorithm's output and comparing the difference relation between tuning $\alpha_{desired}$ and docking time (s) from case 1, 2 and 3 as shown in Fig. 8, 9 and 10.

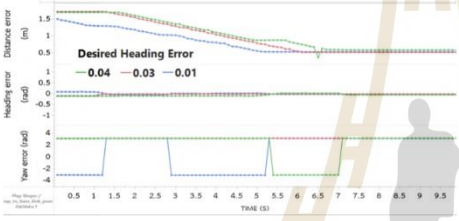


Fig. 8. Relation of d_{error} (m), θ_{error} (radians) and $\alpha_{desired}$ (radians) versus time (s) from case the 1.

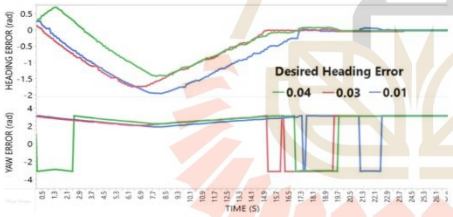


Fig. 9. Relation of $\alpha_{desired}$ (radians), θ_{yaw} (radians) versus time (s) from the case 2.

Fig. 10. Relation of $\alpha_{desired}$ (radians), θ_{yaw} (radians) versus time (s) from the case 3.

According to the simulation findings, adjusting the α_{error} parameter in the first case of the AMR positioned 2 m in front of the charging station with y = 0 m can reduce the duration of autonomic docking process. When compared to a value of $\alpha_{desired}$ = 0.04 radians, a value of $\alpha_{desired}$ = 0.01 radians can shorten the algorithm's execution time by 1.5 seconds while increasing the algorithm's accuracy before AMR enters the Finished docking state. The simulation results from adjusting the α_{error} parameter in the case 2 demonstrate that the autonomic docking time is longer than in the case 1 around 17 seconds. In cases 2 and 3, it was observed that the algorithm's performance in adjusting $\alpha_{desired}$ = 0.01 radians required longer duration compared with $\alpha_{desired}$ that was set to 0.03, 0.04 radians, respectively. However, decreasing $\alpha_{desired}$ did not have a clear effect on α_{error} in case 1. It was found that acceptable heading error and yaw error occur when adjusting $\alpha_{desired}$= 0.03. This value provides support for the case 2 and 3 scenarios with α_{error} = 0.0304 radians and θ_{error}= 0.04 radians for the v_x= 0.25 m/s, ω = 0.2 radians/s. From the experiment, It was found that acceptable heading error and yaw error occur when adjusting $\alpha_{desired}$= 0.03. This value provides support for case 2 and 3 scenarios with α_{error}= 0.0304 radians and θ_{error}= 0.04 radians.

C. *Velocity reduction experiment*

The second experiment will involve testing reductions in linear velocity and angular velocity while maintaining a constant value of $\alpha_{desired}$ = 0.01 radians and d_{error} = 0.5 m, This experiment will be conducted under case 3 conditions, The velocity variations will include (v_x= 0.25 m/s, ω = 0.2 radians/s), (v_x= 0.15 m/s, ω = 0.1 radians/s) and (v_x = 0.1 m/s, ω = 0.1 radians/s), respectively. These variations will be compared with errors in d_{error}, α_{error} and θ_{error} , and the time taken to complete the docking process will be indicated on the x-axis by red lines, as depicted in Fig. 11 and 8.

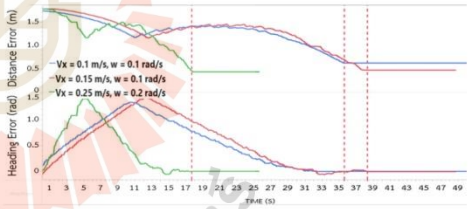


Fig. 11. Relation of d_{error} (m) and $\alpha_{desired}$ (radians) versus time (s) from the case 3

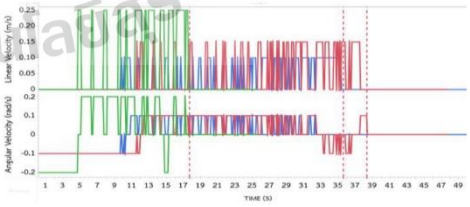


Fig. 12. Linear velocity and angular velocity in the velocity reduction experiment

TABLE III. VELOCITY REDUCTION RESULTS

v_x (m/s)	ω (rad/s)	d_{error} (m)	α_{error} (m)	t (s)
0.25	0.2	-0.12	-0.005	17.69
0.15	0.1	0.03	-0.006	35.59
0.1	0.1	0.07	-0.005	38.31

The outcomes of the experiments on autonomic docking for AMR robots reveal that the choice of v_x is crucial in determining the effectiveness of the docking process. As v_x and ω decreases from (v_x= 0.25 m/s, ω = 0.2 radians/s) to (v_x= 0.1 m/s, ω = 0.1 radians/s), the time required for docking more than doubles, highlighting its substantial impact on the process. Furthermore, variations in ω from 0.2 rad/s to 0.1 rad/s do not exhibit a consistent or linear relationship with the α_{error}, the role of angular velocity appears more complex and influenced by other factors. Additionally, the fluctuation of d_{error} in response to changes in linear velocity suggests its sensitivity to this parameter. From the results, it is evident that velocity variation has an effect on d_{error} .The optimal linear velocity v_x setting to achieve the lowest distance error from the charging station is 0.15 m/s. However, it's important to note that using v_x = 0.1 m/s may cause the AMR to park further away from the charging station than in earlier instances. Meanwhile, v_x= 0.25 m/s is not an appropriate value for this process, as it results in the AMR being offset over the charging station by 0.12 m. Nevertheless, in the autonomic docking process, the velocity variation does not significantly affect the α_{error} even when the AMR experiences a veering position.

V. CONCLUSION

Through simulations conducted using the navigation stack destination tolerance, it becomes evident that adjusting the parameter $\alpha_{desired}$ has varying effects in different cases. In case 1, modifying $\alpha_{desired}$ can leads to a reduction in algorithm execution time and an enhancement in accuracy. Conversely, in cases 2 and 3, where the AMR position in a veer position fluctuating within y = ± 0.25 m, reducing $\alpha_{desired}$ results in a longer access time to reach the finished docking state but also leads to an increased heading error in the initial state,Therefore the appropriate of the $\alpha_{desired}$ of this research is $\alpha_{desired}$ = 0.03 rad due to this value can support for the case 2 and 3 scenarios to reduce the heading error at the initial state. For the velocity reduction experiment, Lowering the linear velocity v_x and angular velocity ω significantly impacts the docking time. The effect of velocity has a direct impact on d_{error} ,with an increase in both v_x and ω, The optimal parameters identified in the velocity reduction experiment are v_x=0.15 m/s, ω =0.1 rad/s, $\alpha_{desired}$ = 0.01. These parameters have proven to be extremely effective in accurately navigating the AMR to the charging station, despite the longer access time required to reach the charging station. Nevertheless, The algorithm has consistently guided the AMR into the charging dock with precision, without any operational issues in cases 2 and 3 for the finished docking state. Additionally, when integrated with BT, the algorithm's operation can be easily understood and is highly advantageous in terms of reusability and scalability. When compared with the conventional method.

REFERENCES

[1] P. Ogren, "Increasing modularity of UAV control systems using computer game behavior trees," in Proc. AIAA Guidance, Navigation, and Control Conf., 2012, p. 4458.

[2] M. Colledanchise, R. Parasuraman, and P. Ogren, "Learning of behavior trees for autonomous agents," IEEE Trans. on Games, vol. 11, no. 2, pp. 183–189, 2018.

[3] B. Banerjee, "Autonomous Acquisition of Behavior Trees for Robot Control," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 2018, pp. 3460-3467.

[4] M. Colledanchise, D. Almeida and P. Ögren, "Towards Blended Reactive Planning and Acting using Behavior Trees," 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 8839-8845.

[5] Domínguez, D., Iannotta, M., Stork, J.A., Schaffernicht, E.J., & Stoyanov, T. (2022). A Stack-of-Tasks Approach Combined With Behavior Trees: A New Framework for Robot Control. IEEE Robotics and Automation Letters, 7, 12110-12117.

[6] X. Zhang, X. Li and X. Zhang, "Automatic Docking and Charging of Mobile Robot Based on Laser Measurement," 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing, China, 2021, pp. 2229-2234.

[7] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-robot collaborative high-level control with application to rescue robotics," in 2016 IEEE International Conference on Robotics and Automation (ICRA), May 2016, pp. 2796–2802.

[8] Y. Li, and C. Shi, "Localization and Navigation for Indoor Mobile Robot Based on ROS", 2018 Chinese Automation Congress (CAC), IEEE, pp. 1135-1139, 2018.

[9] Y. Li et al., "A SLAM with simultaneous construction of 2D and 3D maps based on Rao-Blackwellized particle filters," 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI), Xiamen, China, 2018, pp. 374-378.

[10] K. Takaya, T. Asai, V. Kroumov and F. Smarandache, "Simulation environment for mobile robots testing using ROS and Gazebo," 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2016, pp. 96-101.

ประวัติผู้เขียน

นายกิตติยศ ยะเจริญ เกิดเมื่อวันที่ 28 ตุลาคม พ.ศ.2541 จบการศึกษาในระดับชั้นมัธยมศึกษาตอนปลายจากโรงเรียนอัสสัมชัญนครราชสีมา อ.เมือง จ.นครราชสีมา และสำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิตในหลักสูตรวิศวกรรมไฟฟ้า มหาวิทยาลัยเทคโนโลยีสุรนารี จ.นครราชสีมา ในปี พ.ศ.2563 และได้เข้าศึกษาต่อในระดับปริญญาโท หลักสูตรวิศวกรรมเมคคาทรอนิกส์ มหาวิทยาลัยเทคโนโลยีสุรนารี ปี พ.ศ. 2564 ในขณะที่จบการศึกษาปริญญาตรีผู้วิจัยได้มีความสนใจและได้มีโอกาสทำงานเกี่ยวกับยานพาหะขนส่งอัตโนมัติ จึงได้นำความรู้จากการทำงานในด้านนี้พร้อมทั้งความรู้ที่ได้รับจากการศึกษาในระดับปริญญาโท มาต่อยอดและประยุกต์ใช้กับงานวิจัย