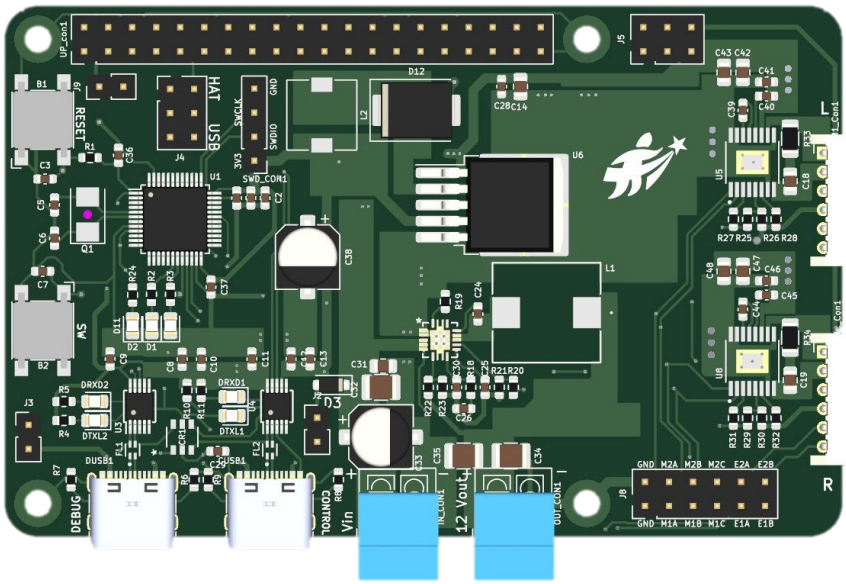
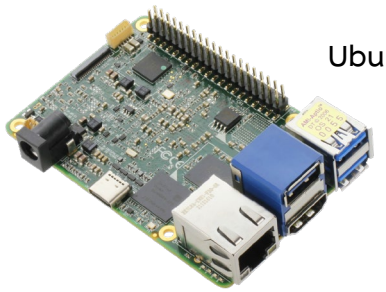
Bogie Board manual V1.1



1 ระบบควบคุมการเคลื่อนที่ของหุ่นยนต์

หุ่นยนต์ที่เป็นลักษณะ Mobile Robot ที่ถูกผลิตออกมาในปัจจุบันมีการออกแบบมีส่วนสำคัญอยู่ 3 ส่วนได้แก่ ระบบการเคลื่อนที่ (Mechanism) ระบบการรับรู้ (Sensor) และ ระบบการสั่งงานวางแผนและตัดสินใจ (Application) การออกแบบหุ่นยนต์จึงขึ้นอยู่กับการนำไปใช้งาน

ในส่วนนี้จะยกตัวอย่างสำหรับระบบการเคลื่อนที่ของหุ่นยนต์แบบ Differential drive โดยใช้บอร์ดควบคุมสำเร็จรูป “Bogie Board” อย่างไรก็ตามผู้อ่านสามารถนำหลักการไปประยุกต์ใช้กับอุปกรณ์อื่นๆได้ต่อไป

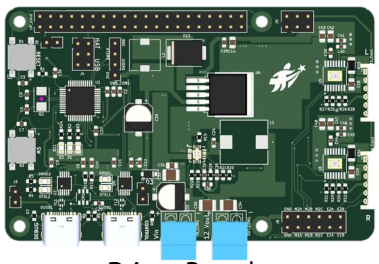


Ubuntu & Ros

มอเตอร์ล้อซ้าย

Communication

UART or USB

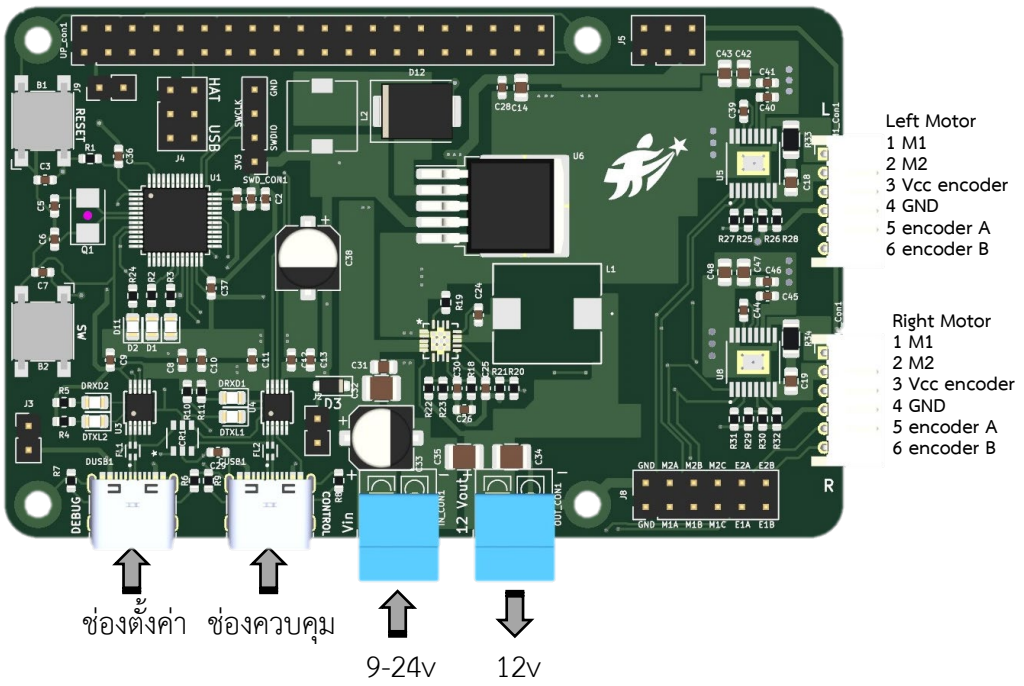


Driver Board

มอเตอร์ล้อขวา

ในส่วนนี้จะอธิบายถือการสื่อสารกันระหว่าง ROS และบอร์ดควบคุม จากการเชื่อมต่อด้านบน ROS จะสร้าง Node ชื่อ base_controller ที่รอรับ Topic ชื่อ cmd_vel จากนั้นจะทำการส่งความเร็วของล้อซ้ายและขวามายัง บอร์ดควบคุม ผ่านช่องทาง UART หรือสาย USB

บอร์ดควบคุมจะรอรับความเร็วของล้อซ้ายและขวาและควบคุมความเร็วของล้อซ้ายและขวาให้เป็นไปตามที่ได้รับข้อมูลมาในขณะเดียวกันบอร์ดควบคุมจะส่งข้อมูลความเร็วล้อซ้ายและล้อขวาผ่านช่องทาง UART หรือสาย USB กลับไปยัง Node เดิมที่ชื่อ base_controller เพื่อใช้ในการระบุตำแหน่งหุ่นยนต์ต่อไป



การเชื่อมต่อต่างๆของบอร์ดควบคุม

ในการสร้างโครงหุ่นยนต์สำหรับระบบการเคลื่อนที่ของหุ่นยนต์แบบ Differential drive สิ่งที่จำเป็นต้องมีคือมอเตอร์และชุดล้อ เพื่อให้ได้แรงบิดที่เยอะมอเตอร์ควรเป็นแบบทดเกียร์ อัตราส่วนทดจะแปรผันตรงกับแรงบิดของเพลามอเตอร์ ยิ่งอัตราส่วนการทดมากเท่าไร แรงบิดของเพลาก็จะยิ่งมากขึ้นเท่านั้น แต่ความเร็วรอบของเพลาก็จะยิ่งช้าลงเท่านั้นด้วยเช่นกัน

การเลือกขนาดเส้นผ่านศูนย์กลางล้อก็เช่นเดียวกันที่ต้องพิจารณายิ่งเส้นผ่านศูนย์กลางมากเท่าไร ความสามารถของรถหุ่นยนต์ในการข้ามสิ่งกีดขวางก็จะดีขึ้นเท่านั้น แต่แรงบิดที่ส่งถึงพื้นจากมอเตอร์จะลดลง

เมื่ออ่านมาถึงตรงนี้ผู้อ่านอาจจะอยากทำให้หุ่นยนต์ของเราขยับได้แล้ว ในตอนนี้ผู้เขียนขออธิบายในฐานที่ผู้อ่านทุกท่าน มีมอเตอร์ บอร์ดควบคุม และชุดประมวลผลที่ติดตั้ง ROS2 ไว้แล้ว หากท่านใดยังไม่มี ผู้เขียนจะอธิบายหลักการหลังจากนี้เพื่อให้สามารถนำไปประยุกต์ใช้ในรูปแบบตามที่ต้องการได้

ผู้อ่านสามารถดาวน์โหลดได้จาก https://github.com/midnightpen/bogie2 จากนั้นหาไฟล์ชื่อ base_controller.py ที่อยู่ใน Package ที่ชื่อว่า bogie_bringup เนื่องจากการเชื่อมของผู้ใช้งานอาจมีความแตกต่างกัน ดังนั้นจึงต้องเข้าไปแก้ช่องทางการติดต่อที่อยู่ในไฟล์ base_controller.py บรรทัดที่ 26 โดยค่าตั้งต้นจะตั้งไว้ที่ "/dev/ttyS5" ซึ่งเป็นการสื่อสารแบบ UART ของบอร์ด UP4000 ผู้ใช้สามารถแก้ไขได้ตามช่องทางการติดต่อของผู้ใช้งานเอง ดังตัวอย่างด้านล่าง

```
class Uart():
    def __init__(self,mp):
        self.mana = mp.Manager()
        self.Buff = self.mana.list()

        self.flag_send =  mp.Value('i',0)


        self.Buff_SSend = self.mana.list()

        self.port = "/dev/ttyS5"
        self.baudrate = 38400
```

จากนั้นทำการ Build Package และสั่งเริ่มการทำงานของโปรแกรมดังตัวอย่างด้านล่าง

```
ros2@book: ~/ros2_ws$colcon build --packages-select bogie_bringup

Starting >>> bogie_bringup
Finished <<< bogie_bringup [3.81s]

Summary: 1 package finished [6.71s]

ros2@book: ~$ source ~/.bashrc
ros2@book: ~$ ros2 run bogie_bringup base_controller

[base_mix.py-8] [INFO] [1705987231.782257560] [base_controller]: left
target = 0 left feedback = 0 right target = 0 right feedback = 0
[base_mix.py-8] [INFO] [1705987231.796239077] [base_controller]: left
target = 0 left feedback = 0 right target = 0 right feedback = 0
[base_mix.py-8] [INFO] [1705987231.896365478] [base_controller]: left
target = 0 left feedback = 0 right target = 0 right feedback = 0
```

จากคำสั่งข้างต้น ROS จะสร้าง Node ชื่อ base_controller ที่รอรับ Topic ชื่อ cmd_vel ในตอนนี้สามารถส่ง Topic cmd_vel ได้ด้วยคีย์บอร์ดในเบื้องต้น ซึ่งในนี้นี้จะขอใช้การสั่งการความเร็วด้วย Python Node ที่ผู้เขียนพัฒนาขึ้นมาเองโดยสามารถสั่งเริ่มการทำงานได้โดยใช้คำสั่งด้านล่าง

```
ros2 run bogie_bringup my_teleop
```

ในตอนนี้จะมี Node my_teleop ที่ทำการส่ง Topic ชื่อ cmd_vel จากนั้นผู้รับข้อมูล นั่นก็คือ base_controller จะทำการรับ cmd_vel เพื่อทำการคำนวณเป็นความเร็วล้อซ้ายและล้อขวาเพื่อส่งให้บอร์ดควบคุมทำการควบคุมความเร็วล้อให้เป็นไปตามการสั่งการต่อไป

จากการสั่งการ base_controller จะมีข้อความแสดงออกมายังหน้าต่างแสดงผล ซึ่งประกอบไปด้วยค่า 4 ค่าดังนี้

left target	ความเร็วเป้าหมายล้อซ้าย	ได้มาจากการคำนวณย้อนกลับมาจาก cmd_vel
left feedback	ความเร็วจริงล้อซ้าย	ได้มาจากการคำนวณย้อนกลับมาจาก Encoder
right target	ความเร็วเป้าหมายล้อขวา	ได้มาจากการคำนวณย้อนกลับมาจาก cmd_vel
right feedback	ความเร็วจริงล้อขวา	ได้มาจากการคำนวณย้อนกลับมาจาก Encoder

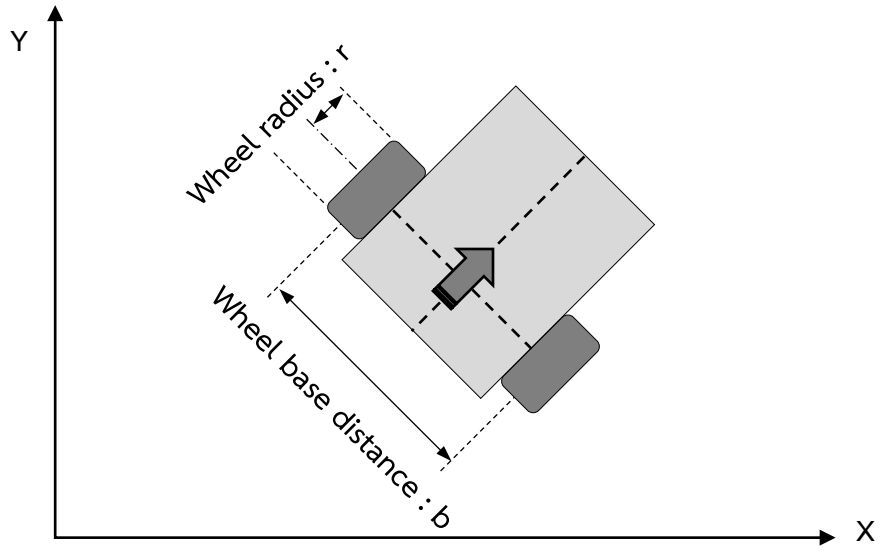
ก่อนจะอธิบายต่อไป ผู้เขียนขอกล่าวถึงตัวแปรสองตัวที่มีความสำคัญในตัวอย่างนี้ ซึ่งก็คือ speed_ratio และ wheel_distance ซึ่งเป็นตัวแปรที่จะใช้ในการคำนวณความเร็วล้อซ้ายและขวาจากความเร็วที่ได้มาจาก cmd_vel ซึ่งสามารถดูได้ที่ไฟล์ base_controller.py บรรทัดที่ 30 และ 31

```
self.speed_ratio = mp.Value('d',0.0000415)    # unit: m/encode
self.wheel_distance = mp.Value('d',0.29)    # unit: m
```

speed_ratio คืออัตราส่วนระหว่าง ระยะทางที่หุ่นยนต์เดินทางต่อหนึ่งค่าสัญญาณของ encoder เดิมแล้วตัวแปรที่มีผลต่อการคำนวณคือ รัศมีวงล้อและระยะห่างระหว่างล้อซ้ายและล้อขวา ในการเดินทางเป็นเส้นตรงรัศมีวงล้อจะส่งผลโดยตรงกับความเร็ว กล่าวคือในการหมุนของล้อที่เท่ากัน รัศมีวงล้อที่ใหญ่กว่าจะได้ระยะทางที่ไกลกว่า ส่งผลให้ได้รับความเร็วที่สูงกว่า และต่อจากนั้นคือมอเตอร์และ encoder หากใช้มอเตอร์ที่มีจำนวนค่าสัญญาณต่อการหมุนหนึ่งรอบของล้อไม่เท่ากันก็จะส่งผลต่อข้อมูลที่เข้ามาใช้ในการคำนวณเช่นกัน ดังนั้นเพื่อความง่ายในการคำนวณผู้เขียนจึงรวมตัวแปรทั้งหมดและแทนค่าภาพรวมของระบบด้วย ตราส่วนระหว่างระยะทางที่หุ่นยนต์เดินทางต่อหนึ่งค่าสัญญาณของ encoder ซึ่งหากผู้ใช้งานนำไปใช้แล้วไม่ได้ระยะทางการเดินแบบเชิงเส้นของหุ่นที่ไม่ตรงกับค่าที่แสดงใน ROS ผู้ใช้สามารถปรับค่า speed_ratio ให้ตรงกับการทำงานจริงได้

wheel_distance คือระยะห่างระหว่างล้อซ้ายและล้อขวา ในการสั่งการหุ่นยนต์ให้หมุนรอบตัวเอง ระยะห่างระหว่างล้อซ้ายและล้อขวาจะส่งผลโดยตรงกับความเร็วเชิงมุม กล่าวคือในการหมุนของล้อที่เท่ากัน ระยะห่างระหว่างล้อซ้ายและล้อขวาที่กว้างกว่าจะได้ความเร็วของหุ่นยนต์ที่น้อยกว่า ซึ่งหากผู้ใช้งานนำไปใช้แล้วไม่ได้ระยะทางเชิงมุมของหุ่นที่ไม่ตรงกับค่าที่แสดงใน ROS ผู้ใช้สามารถปรับค่า wheel_distance ให้ตรงกับการทำงานจริงได้

จากที่กล่าวมาข้างต้น การแปลงจากค่าความเร็วเชิงเส้น และความเร็วเชิงมุมของหุ่นยนต์ ไปเป็นความเร็วล้อซ้าย และขวานั้น ตัวแปรที่มีความสำคัญได้แก่ รัศมีล้อ (**Wheel radius : r**) และความกว้างของฐานล้อ **Wheel base distance : b** นั่นเองดังแสดงในภาพ



รัศมีล้อ และความกว้างฐานล้อ

สมการการแปลงค่าความเร็วเชิงเส้น และความเร็วเชิงมุมของหุ่นยนต์ ไปเป็นความเร็วล้อซ้ายและขวานั้นดังแสดงในสมการ

$$\omega_L = \frac{V}{r} - \frac{\omega \cdot (b/2)}{r}$$

$$\omega_R = \frac{V}{r} + \frac{\omega \cdot (b/2)}{r}$$

เมื่อ ω_L คือ ความเร็วเชิงมุมของล้อซ้าย มีหน่วยเป็น rad/s

ω_R คือ ความเร็วเชิงมุมของล้อขวา มีหน่วยเป็น rad/s

V คือ ความเร็วเชิงเส้นของหุ่นยนต์ของล้อขวา มีหน่วยเป็น m/s

ω คือ ความเร็วเชิงมุมของหุ่นยนต์ มีหน่วยเป็น rad/s

r คือ รัศมีล้อ มีหน่วยเป็น m

b คือ ระยะความกว้างของฐานล้อ มีหน่วยเป็น m

เมื่อสั่งการทำงานของโปรแกรม Node ชื่อ base_controller จะรอรับ Topic ชื่อ cmd_vel เมื่อมีข้อความ cmd_vel เข้ามา โปรแกรมจะคำนวณหาความเร็วเป้าหมายล้อซ้ายและขวาจากความเร็วเชิงเส้นและความเร็วเชิงมุมที่ได้รับเพื่อคำนวณเป็น ความเร็วเชิงมุมของล้อซ้าย และความเร็วเชิงมุมของล้อขวา

ในทางกลับกัน base_controller จะได้รับความเร็วจริงล้อซ้าย ความเร็วจริงล้อขวาจากบอร์ดควบคุมส่งกลับมาด้วยเช่นกัน โดยผู้ใช้สามารถใช้ข้อมูลมาเปรียบเทียบกันได้ ว่าบอร์ดควบคุมสามารถควบคุมมอเตอร์ได้เป็นไปตามความเร็วที่สั่งการหรือไม่

จากที่กล่าวไปความเร็วเป้าหมายความเร็วจริงของล้อมีหน่วยเป็นองศาต่อเวลา เพื่อลดปัญหาหน่วยความจำไม่เพียงพอหรือ Stack overflow ข้อมูลที่ส่งจริงจะส่งไปและกลับมาในรูป จำนวนค่าสัญญาณของ encoder ต่อเวลา 0.04 วินาที หรือจำนวนค่าสัญญาณของ encoder ในทุกความถี่ที่ 25 Hz นั่นเอง

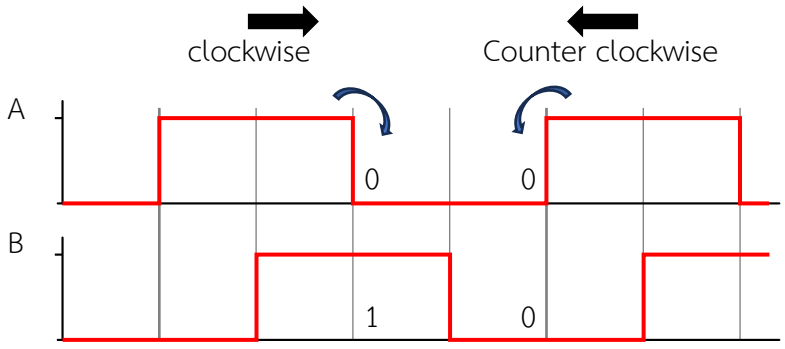
2 เอ็นโค้ดเดอร์ (Encoder)

หุ่นยนต์ที่เป็นลักษณะ Mobile Robot ที่ถูกผลิตออกมาในปัจจุบันมีการออกแบบมีส่วนสำคัญอยู่ 3 ส่วนได้แก่ ระบบการเคลื่อนที่ (Mechanism) ระบบการรับรู้ (Sensor) และ ระบบการสั่งงานวางแผนและตัดสินใจ (Application) การออกแบบหุ่นยนต์จึงขึ้นอยู่กับการนำไปใช้งาน

ในส่วนนี้จะยกตัวอย่างสำหรับระบบการเคลื่อนที่ของหุ่นยนต์แบบ Differential drive โดยใช้บอร์ดควบคุมสำเร็จรูป “Bogie Board” อย่างไรก็ตามผู้อ่านสามารถนำหลักการไปประยุกต์ใช้กับ

เอ็นโค้ดเดอร์เป็นเซนเซอร์ที่ใช้สำหรับการวัดระยะทาง, ความเร็วรอบ , ทิศทางการหมุนของมอเตอร์ และองศาการเคลื่อนที่ โดยอาศัยหลักการทำงานโดยการเข้ารหัสจากระยะทางจากการหมุนของแกนเพลา แล้วทำการแปลงออกมาในรูปแบบของสัญญาณไฟฟ้า เพื่อนำมาประมวลผลต่อ โดยสามารถแบ่งออกได้เป็น 2 ประเภทใหญ่ๆ ได้ดังนี้

Incremental Rotary Encoder เป็นเอ็นโค้ดเดอร์ที่รูปแบบสัญญาณขาออกเป็นลักษณะของสัญญาณลูกคลื่นเป็นคลื่นรูปสี่เหลี่ยม โดยจำนวนลูกคลื่นที่ออกมานั้น จะมีความสัมพันธ์กับระยะการเคลื่อนที่ ตำแหน่ง ระยะห่าง ความเร็ว และ ความเร่ง นอกจากนี้ยังสามารถระบุได้ถึงทิศทางการหมุนของตัวเอ็นโค้ดเดอร์ได้ว่าหมุนตามเข็มนาฬิกา หรือ ทวนเข็มนาฬิกาโดยอาศัยการตรวจจับทิศทางการหมุนจากมุมเฟสของสัญญาณขอออก A เทียบกับ B ว่าสัญญาณใดเกิดก่อนกัน ซึ่งจะมีมุมเฟสที่ต่างกันอยู่ 90 องศา

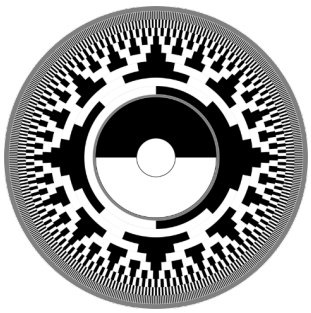


องศาเฟสที่ต่างกันของสัญญาณจาก Incremental Rotary Encoder

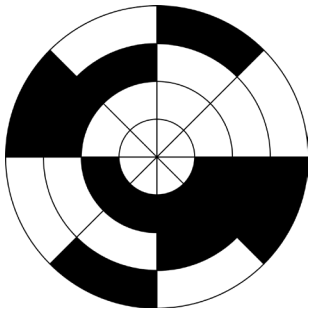
หากช่องสัญญาณ A มีค่าลงมาเป็นศูนย์ ในขณะนั้นจะทำการตรวจสอบช่องสัญญาณ B หากขณะนั้นช่องสัญญาณ B มีค่าเป็น 1 แสดงว่ามอเตอร์กำลังหมุนตามเข็มนาฬิกา แต่หากขณะนั้นช่องสัญญาณ B มีค่าเป็น 0 แสดงว่ามอเตอร์กำลังหมุนทวนเข็มนาฬิกา

เอ็นโค้ดเดอร์ลักษณะนี้ไม่สามารถจดจำตำแหน่งแกนหมุนของตัวเองได้ว่าอยู่ที่จุดใด ด้วยเหตุนี้การหมุนกลับไปยังตำแหน่งเริ่มต้น นั้นจะทำได้ยาก จึงต้องอาศัยการเก็บข้อมูลลูกคลื่นตั้งแต่เริ่มต้น เพื่อหาค่าตำแหน่ง

Absolute Rotary Encoder เป็นเอ็นโค้ดเดอร์ที่ออกแบบมาให้มีรูปแบบสัญญาณขอออกที่เป็นลักษณะของการเข้ารหัส เพื่อต้องการแก้ปัญหาของ Incremental Encoder เนื่องจากสัญญาณขาออกของ Incremental Encoder ไม่สามารถระบุตำแหน่งองศาของแกนเอ็นโค้ดเดอร์ได้ จึงแก้ปัญหาเหล่านี้ได้โดยการใช้ Absolute Rotary Encoder ซึ่งจะใช้รหัสแทนการนับสัญญาณลูกคลื่น โดยรหัสเหล่านี้ มีหลากหลายรูปแบบ เช่น BCD, Binary, Gray Code ซึ่งจะสามารถแทนค่าตำแหน่งองศาที่แกนของเอ็นโค้ดเดอร์หยุดอยู่ได้ แม้จะหยุดจ่ายไฟ และจ่ายไฟเข้าไปใหม่ก็ยังสามารถบ่งบอกได้ว่าตำแหน่งองศาที่อยู่นั้นคือเท่าใด



3-bit binary rotary

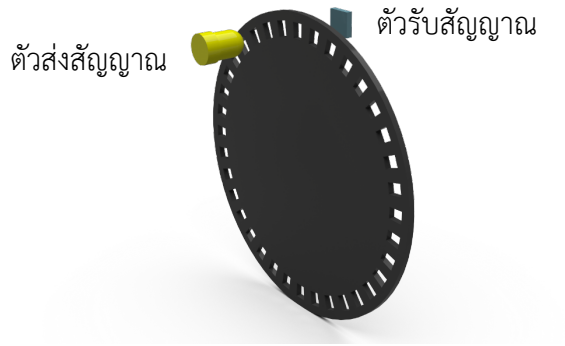


9-bit binary rotary

จานรหัสของ Absolute Rotary Encoder

นอกจากนี้ยังมีการแบ่งตามโครงสร้างของเอ็นโค้ดเดอร์ โครงสร้างก็เป็นอีกส่วนหนึ่งที่สามารถใช้บ่งบอกได้ถึงประเภทของเอ็นโค้ดเดอร์ได้เช่นกัน โดยจะแบ่งออกได้เป็น 2 ลักษณะได้ดังนี้

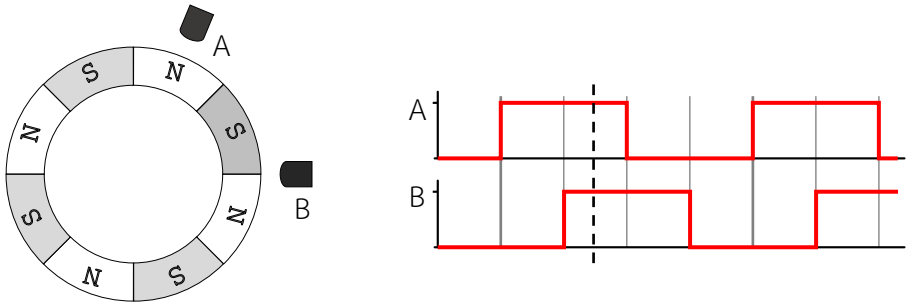
Rotary Optical Encoder เป็นเซนเซอร์แบบพื้นฐานที่ถูกออกแบบมาใช้งานตั้งแต่เริ่มต้น โดยใช้หลักของ Optical Sensing เป็นตัวส่งสัญญาณผ่านจานหมุนหรือ Code Disk ที่ติดอยู่กับแกนมอเตอร์ และใช้ตัว Scanning reticle ซึ่งจะเป็นตัวช่วยบังคับแสงให้ไปตรงที่ตัวรับแสง หรือ Phototransistor ได้อย่างแม่นยำ



การทำงานของ Rotary Optical Encoder

Rotary Magnetic Encoder จากการทำงานของตัว Rotary Optical Encoder ซึ่งโครงสร้างภายในนั้นสร้างมาจาก Code Disk ซึ่งเป็นวัสดุที่มีความเปราะบาง ไม่ทนต่อการใช้งานที่มีแรงสั่นสะเทือนและการกระแทกได้มาก ดังนั้นจึงได้มีการพัฒนาตัว Rotary Magnetic Encoder โดยจะมีความสามารถในการใช้งานที่สูงกว่า และทนต่อแรงสั่นสะเทือนได้ดีกว่าแบบ Code Disk จึงเป็นการยืดอายุการใช้งานของ Encoder ได้ยาวนานมากยิ่งขึ้น

ซึ่งหลักการทำงานของตัว Magnetic Encoder นั้นจะใช้ Magnetic Code Ring เป็นตัวหมุน และใช้เซนเซอร์ในการตรวจจับสัญญาณจากสนามแม่เหล็ก โดยเซนเซอร์ที่ว่านี้มีอยู่ 2 แบบ คือ Magneto-Resistance และ IC Sensor โดยเซนเซอร์ทั้ง 2 รูปแบบนี้ เมื่อมีสนามแม่เหล็กตัดผ่านจะทำให้เกิดปรากฏการ Hall Effect จากปรากฏการณ์นี้จะทำให้ได้แรงดันไฟฟ้าที่เป็นแบบ Sine Wave ออกมา ต่อจากนั้นจะมีการใช้กระบวนการแปลงสัญญาณจากสัญญาณ Analog to Digital A/D เพื่อนำเอาสัญญาณดิจิทัลเหล่านี้ไปใช้งานต่อไป ดังแสดงโครงสร้างการทำงานในภาพที่ 3-13

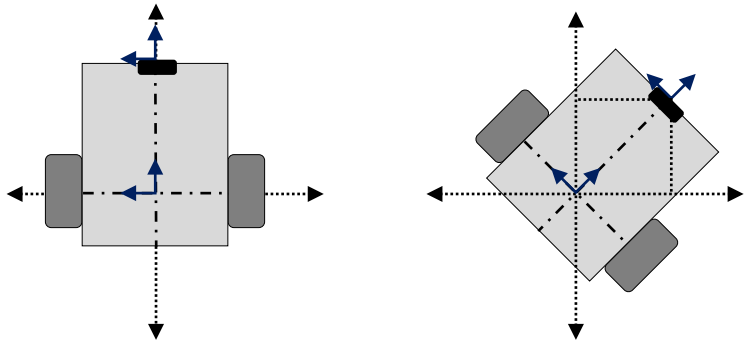


โครงสร้างการทำงานของ Rotary Magnetic Encoder

3 การระบุตำแหน่งของหุ่นยนต์

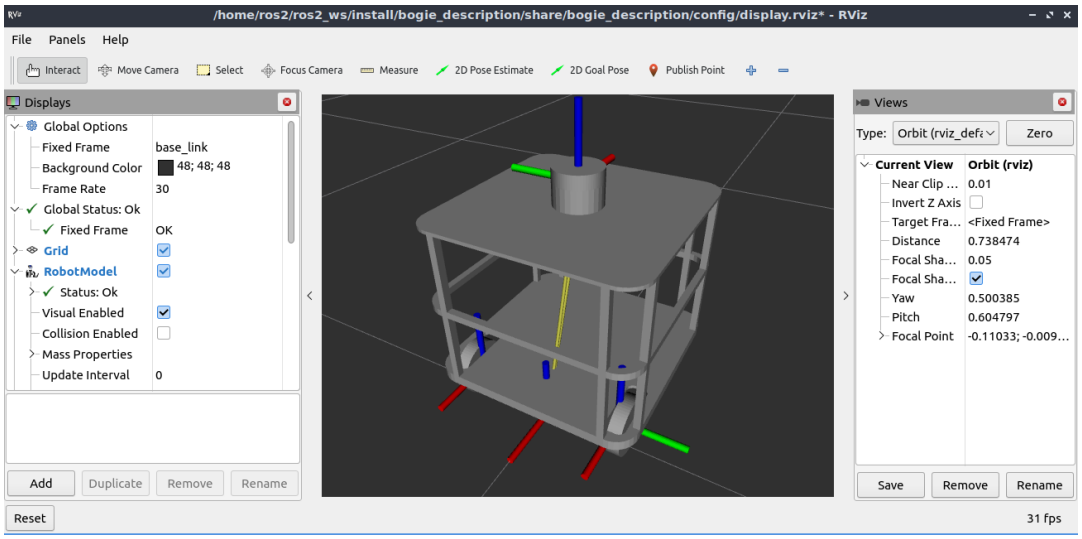
การระบุตำแหน่งของหุ่นยนต์ (Localization) เป็นกระบวนการที่หุ่นยนต์สามารถบอกตำแหน่งตัวเองได้ว่าอยู่ที่จุดใดในแผนที่ โดยแผนที่นั้นอาจเป็นแผนที่ที่กำหนดไว้ล่วงหน้าแล้ว หรือเป็นแผนที่ที่สร้างไปพร้อมกับการระบุตำแหน่งก็ได้ การระบุตำแหน่งของหุ่นยนต์ ทำได้โดยวิธีการที่เรียกว่า Odometry คือ การระบุว่าหุ่นยนต์เคลื่อนที่ไปได้ในระยะทางเท่าไร ตำแหน่งและทิศทางต่างๆของหุ่นยนต์เมื่อเทียบกับจุดเริ่มต้น การทำงานพื้นฐานของวิธีการ Odometry จะนับจำนวนรอบการหมุนที่ล้อของหุ่นยนต์โดยใช้เอ็นโค้ดเดอร์ จากนั้นนำข้อมูลที่ได้มาแปลงกลับเป็นค่าระยะทาง ตำแหน่ง และทิศทางการเคลื่อนที่ การระบุตำแหน่งของหุ่นยนต์ด้วยวิธี Odometry นั้นจะมีความถูกต้อง แม่นยำค่อนข้างต่ำเกิดจากปัญหาทางกายภาพของตัวเซนเซอร์ ปัญหาการลื่นไถล ของหุ่นยนต์ รวมถึงสัญญาณรบกวนต่างๆ ทำให้ข้อมูลที่ได้มาไม่มีความสมบูรณ์ อีกทั้งในแต่ละช่วงเวลาที่เคลื่อนที่ไป ระบบจะสะสมความผิดพลาดเพิ่มขึ้น ซึ่งทำให้หุ่นยนต์มีความผิดพลาดได้ ดังนั้นจำเป็นต้องมีการประยุกต์ใช้ เทคนิคหรือวิธีการอื่นๆ ควบคู่ไปด้วยเพื่อให้หุ่นยนต์มีความถูกต้องและแม่นยำมากขึ้น

ในการเคลื่อนที่ของหุ่นยนต์ที่มีโครงสร้างแบบ Differential Drive หากล้ออยู่ตรงกลาง และติดเซนเซอร์อะไรซักอย่างอยู่ด้านหน้า หากขณะนั้นหุ่นยนต์กำลังหมุนรอบตัวเอง ดังนั้นโครงสร้างของหุ่นจะอยู่จุดเดิมเพียงแต่มีมุมที่เปลี่ยนไป หากมองในจุดของเซนเซอร์ที่ติดตั้งไปนั้นเซนเซอร์จะไม่ได้อยู่จุดเดิมและไม่ได้อยู่ในมุมเดิม ดังแสดงในภาพ ดังนั้นการหาตำแหน่งต่างๆที่เปลี่ยนแปลงไปจะถูกเรียกว่า tf หรือ transform ซึ่งก็คือการเลื่อนเฟรมหนึ่งไปหาอีกเฟรมหนึ่งหรือเคลื่อนที่เฟรมไปยังตำแหน่งต่างๆ ซึ่งในการเคลื่อนที่ของเฟรมนั้นโดยปกติแล้วจะต้องมี เฟรมอ้างอิงเสมอ



การหมุนของหุ่นยนต์ที่มีผลต่อพิกัดที่เปลี่ยนแปลง

อาจเข้าใจได้ยากสำหรับผู้เริ่มเรียนรูป เพื่อง่ายต่อการเข้าใจ ในส่วนนี้จะกล่าวถึงตัวอย่าง URDF ที่อยู่ใน Package ที่มากับหนังสือเล่มนี้ด้วย URDF คือ ไฟล์ที่เอาไว้ใช้อธิบายลักษณะทางกายภาพของหุ่นยนต์ การเขียน URDF สามารถทำได้หลายวิธี วิธีที่ง่ายและเห็นภาพที่สุดคือการสร้าง URDF จากไฟล์แบบสามมิติผ่านโปรแกรมสำเร็จรูป โดยโปรแกรมที่กล่าวถึงเป็นตัวติดตั้งเสริมสำหรับโปรแกรม Fusion360 ผู้ใช้งานสามารถเขียนแบบสามมิติเป็นโครงสร้างหุ่นยนต์ จากนั้นเข้าไปคัดลอกจาก https://github.com/dheena2k2/fusion2urdf-ros2.git มาไว้ในเครื่องเพื่อแปลงไฟล์แบบสามมิติไปเป็นไฟล์ URDF ได้ทันที ดังตัวอย่างที่แสดงในภาพที่ 3-15



ภาพที่ 3-1 ตัวอย่างการทำงานของ URDF

4 การหาแผนที่

ในตอนนี้นอกจากหุ่นยนต์ของเราสามารถเคลื่อนไหว รับและส่งข้อมูลระหว่างบอร์ดควบคุมและระบบปฏิบัติการได้แล้ว ยังสามารถรับข้อมูลขาเข้าและแสดงผลได้อีก แต่หากเราสามารถทำได้เพียงบังคับด้วยตัวเองเพื่อให้ไปถึงจุดหมาย คงไม่ครบวัตถุประสงค์ของการใช้งาน ROS2 ในส่วนนี้จะพูดถึงการหาแผนที่เพื่อใช้ในการนำทางอัตโนมัติต่อไป โดยในเบื่องต้นจะกล่าวถึง Package สำเร็จรูปที่ผู้เขียนได้พัฒนาขึ้นเพื่ออำนวยความสะดวกในการเรียนรู้ในส่วนนี้

1.1 ชุดโปรแกรมสำเร็จรูปหุ่นยนต์ Bogie

เพื่อความสะดวกในการศึกษา ผู้เขียนได้พัฒนาชุดโปรแกรมสำเร็จรูปที่รวมชุดคำสั่งในการทำงานเพื่อเคลื่อนตัวหุ่นยนต์ การหาแผนที่ การนำทาง และการแสดงผลในชุดโปรแกรมดังกล่าว ในที่นี้ผู้เขียนขอตั้งชื่อหุ่นยนต์ตัวนี้ว่า “Bogie” หากหลังจากนี้ได้มีการกล่าวอ้างถึงคำนี้ให้เป็นที่ทราบกันว่าผู้เขียนได้กล่าวถึงหุ่นยนต์ตัวอย่างดังกล่าว

ในชุดโปรแกรมประกอบด้วย Package ที่มีหน้าที่แตกต่างกัน ได้แก่ bogie_bringup bogie_description bogie_navigation imu_module และ rplidar_ros ดังแสดงในแผนภาพโครงสร้างด้านล่าง

```
src
├── bogie_bringup
├── bogie_description
├── bogie_navigation
├── imu_module
└── rplidar_ros
```

1.1.1 ชุดโปรแกรมสำหรับเซนเซอร์วัดระยะทางรอบทิศทาง

ในชุดโปรแกรมสำเร็จรูปนี้ใช้อุปกรณ์ RPLIDAR รุ่น A1 สำหรับเซนเซอร์วัดระยะทางรอบทิศทาง โดยผู้ใช้งานสามารถสามารถใช้ชุดโปรแกรมที่ทางผู้ผลิตให้มาโดยตรง หรือใช้ชุดโปรแกรมที่แนบมากับหนังสือเล่มนี้ได้เช่นเดียวกัน โดยสามารถดูได้ใน Package ที่ชื่อ rplidar_ros โดยชุดโปรแกรมประกอบด้วยโครงสร้างดังแสดงด้านล่าง

```
rplidar_ros
├── CHANGELOG.rst
├── CMakeLists.txt
```

```
├── debian
├── include
├── launch
├── LICENSE
├── package.xml
├── rviz
├── scripts
├── sdk
└── src
```

สำหรับการสั่งการเริ่มต้นการทำงานของ RPLIDAR รุ่น A1 สามารถสั่งการได้โดยเรียกการทำงานของไฟล์ rplidar_a1_launch.py ที่อยู่ในโฟลเดอร์ launch โดยผู้ใช้สามารถเข้าไปเปลี่ยนค่าช่องทางเชื่อมต่อหรือพารามิเตอร์ต่างๆได้ในไฟล์ดังกล่าวเช่นกัน

ชุดโปรแกรมดังกล่าวรองรับเซนเซอร์วัดระยะทางรอบทิศทางสำหรับ RPLIDAR หลายรุ่นด้วยกัน ดังแสดงในรายละเอียดด้านล่าง

```
ros2@book: ~/ros2_ws/src/rplidar_ros/launch$ ls
rplidar_a1_launch.py      rplidar_s2e_launch.py         view_rplidar_a3_launch.py
rplidar_a2m12_launch.py   rplidar_s2_launch.py          view_rplidar_c1_launch.py
rplidar_a2m7_launch.py    rplidar_s3_launch.py          view_rplidar_s1_launch.py
rplidar_a2m8_launch.py    rplidar_t1_launch.py          view_rplidar_s1_tcp_launch.py
rplidar_a3_launch.py      view_rplidar_a1_launch.py     view_rplidar_s2e_launch.py
rplidar_c1_launch.py      view_rplidar_a2m12_launch.py  view_rplidar_s2_launch.py
rplidar_s1_launch.py      view_rplidar_a2m7_launch.py   view_rplidar_s3_launch.py
rplidar_s1_tcp_launch.py  view_rplidar_a2m8_launch.py   view_rplidar_t1_launch.py
```

1.1.2 ชุดโปรแกรมสำหรับเซนเซอร์วัดแนวทิศทางการวางตัว

ในชุดโปรแกรมสำเร็จรูปนี้ใช้อุปกรณ์ HIPNUC รุ่น HI229 สำหรับเซนเซอร์วัดแนวทิศทางการวางตัว โดยผู้ใช้งานสามารถสามารถใช้ชุดโปรแกรมที่ทางผู้ผลิตให้มาโดยตรง หรือใช้ชุดโปรแกรมที่แนบมากับหนังสือเล่มนี้ได้เช่นเดียวกัน โดยสามารถดูได้ใน Package ที่ชื่อ imu_module โดยชุดโปรแกรมประกอบด้วยโครงสร้างดังแสดงด้านล่าง

```
imu_module
├── CMakeLists.txt
├── include
├── launch
├── package.xml
└── src
```

สำหรับการสั่งการเริ่มต้นการทำงานของ HIPNUC รุ่น HI229 สามารถสั่งการได้โดยเรียกการทำงานของไฟล์ imu.launch.py ที่อยู่ในโฟลเดอร์ launch โดยผู้ใช้สามารถเข้าไปเปลี่ยนค่าช่องทางเชื่อมต่อหรือพารามิเตอร์ต่างๆได้ในไฟล์ serial_port.cpp ที่อยู่ในโฟลเดอร์ src ดังรายละเอียดที่แสดงด้านล่าง

```
...
...
...
#ifdef __cplusplus
extern "C"{
#endif

#include "ch_serial.h"

#define IMU_SERIAL   ("/dev/ttyUSB0")
#define BAUD         (B115200)
#define GRA_ACC      (9.8)
#define DEG_TO_RAD   (0.01745329)
#define BUF_SIZE     (1024)
...
...
...
```

1.1.3 ชุดโปรแกรมสำหรับแสดงผลแบบหุ่นยนต์

ในการพัฒนาหุ่นยนต์โดย ros2 ที่มีการติดตั้งเซนเซอร์ จะสามารถนำค่าเซนเซอร์เหล่านั้นมาทำการแสดงผลในรูปแบบสามมิติได้ผ่านเครื่องมือที่ช่วยในการแสดงผล คือ rviz2 ที่สามารถเชื่อมต่อได้กับทั้งหุ่นยนต์ในโลกจริงและโลกเสมือน กล่าวคือ rviz2 มีหน้าที่ในการแสดงผลภาพสามมิติที่มีข้อมูลตำแหน่ง รวมถึงเฟรมอ้างอิงที่ถูกกำหนดให้อยู่ในรูปแบบของเวกเตอร์สามมิติบนหุ่นยนต์ รวมถึงข้อมูลเซนเซอร์ต่างๆด้วยเช่นเดียวกัน

Robot Description หรือ แบบหุ่นยนต์ นั้นสามารถพัฒนาขึ้นเป็นรูปแบบไฟล์ URDF (Unified Robot Description Format) กล่าวคือ URDF มีหน้าที่อธิบายลักษณะการติดตั้งเชิงกลหรือโครงสร้างหุ่นยนต์ซึ่งภายใน URDF นั้นจะมีการกำหนดเฟรมอ้างอิงของหุ่นยนต์ ทำให้ทราบข้อมูลของ ข้อต่อ (Joint), ความสัมพันธ์ (Link) และ เซนเซอร์ ที่ถูกเคลื่อนย้ายในเวลาต่างๆ โดยสถานะของการเปลี่ยนแปลงเหล่านี้จะปรากฏที่ State Publisher โดยต้องลงโปรแกรมเพิ่มเติมดังแสดงด้านล่าง

```
sudo apt install ros-iron-robot-state-publisher
sudo apt install ros-iron-xacro
sudo apt install ros-iron-joint-state-publisher-gui
```

การเขียน URDF นั้นจะสามารถเขียนได้สองรูปแบบใหญ่ๆคือการเขียนโดยกำหนดค่าคุณสมบัติต่างๆเข้าไปที่ URDF โดยตรงและทางที่สองคือ การเขียนไฟล์สกุล Xacro เพื่อให้สามารถใส่ Constant , Prefix , Suffix เข้าไปยัง URDF ได้กล่าวคือเป็นการสร้างตัวแปรเก็บค่าที่ให้ URDF สามารถดึงไปใช้ได้ ซึ่งทั้งสองสกุลไฟล์ดังกล่าวนั้นจะพัฒนาผ่านภาษา “XML” โดยตัวอย่างนี้สามารถดูได้ใน Package ที่ชื่อ bogie_description โดยชุดโปรแกรมประกอบด้วยโครงสร้างดังแสดงด้านล่าง

```
bogie_description
├── bogie_description
├── config
├── launch
├── meshes
├── package.xml
├── resource
├── setup.cfg
├── setup.py
└── urdf
```

1.1.4 ชุดโปรแกรมสำหรับหาแผนที่และนำทาง

จุดประสงค์หลักของ ROS navigation ก็คือการเคลื่อนที่ของหุ่นยนต์จากจุดเริ่มต้นไปยังเป้าหมายที่กำหนด โดยไม่ให้ชนกับอุปสรรค หรือของรอบๆตัวหุ่นยนต์นั้นเอง ซึ่ง ROS navigation มีระเบียบวิธีการต่างๆที่จะมาช่วยทำให้เราพัฒนาหุ่นยนต์เคลื่อนที่ได้ง่ายขึ้น

การนำทางจะไม่สามารถทำงานได้หากไม่มีแผนที่เข้ามาเกี่ยวข้อง ซึ่งแผนที่สามารถสร้างได้จากการใช้เซนเซอร์วัดระยะทางรอบทิศทางหรืออุปกรณ์อื่นๆ เช่นกล้องวัดความลึกเป็นต้น ในที่นี้จะนำเสนอการหาแผนที่โดยใช้ Slam Toolbox และ Cartographer ในส่วนของการนำทางในที่นี้จะนำเสนอการนำทางโดยใช้ DWB Controller, Regulated Pure Pursuit Controller และ Rotation Shim Controller ซึ่งมีจำกัดในการใช้งานต่างกัน โดยสามารถดูรายละเอียดได้ใน Package ที่ชื่อ bogie_navigation โดยชุดโปรแกรมประกอบด้วยโครงสร้างดังแสดงด้านล่าง

```
bogie_navigation
├── bogie_navigation
├── config
├── launch
├── map
├── package.xml
├── resource
├── setup.cfg
└── setup.py
```

นอกจากนี้ชุดโปรแกรมดังกล่าวยังประกอบไปด้วยโปรแกรมย่อยในการทำงานต่างๆ เช่นการสั่งการ LED dot matrix การส่งพิกัดนำทาง การติดต่อสื่อสารกับกล้อง การอ่าน QR code และอื่นๆ โดยสามารถดูได้จากไฟล์ที่อยู่ในโฟลเดอร์ย่อย bogie_navigation ดังแสดงรายละเอียดด้านล่าง

```
ros2@book:~/ros2_ws/src/bogie_navigation/bogie_navigation$ ls
__init__.py             pub_str_point.py       sub_str_point.py
led.py                  qr_code.py             teleop.py
nav_command_test.py     robot_navigator.py     webcam.py
navigation_command.py   sub_str_point_copy.py
```

1.1.5 ชุดโปรแกรมเริ่มการทำงานหลักของหุ่นยนต์

ชุดโปรแกรมเริ่มการทำงานหลักเป็นชุดโปรแกรมที่ประกอบด้วยโปรแกรมย่อยที่ใช้สื่อสารกันระหว่าง ROS และบอร์ดควบคุม จากการเชื่อมต่อด้านบน ROS จะสร้าง Node ชื่อ base_controller ที่รอรับ Topic ชื่อ cmd_vel จากนั้นจะทำการส่งความเร็วของล้อซ้ายและขวามายัง บอร์ดควบคุม ผ่านช่องทาง UART หรือสาย USB โดยโปรแกรมดังกล่าวจะอยู่ในโฟลเดอร์ย่อย bogie_bringup

```
bogie_bringup
├── bogie_bringup
│   ├── base_controller.py
│   └── __init__.py
├── launch
│   └── bringup.launch.py
├── package.xml
├── resource
│   └── bogie_bringup
├── setup.cfg
└── setup.py
```

นอกจากนี้ยังมีไฟล์ Launch ที่รวมการเปิดการทำงานของ Node ย่อยต่างๆใน Package ที่แตกต่างกันดังตัวอย่างที่แสดงด้านล่าง โดยจะเห็นว่าในไฟล์ดังกล่าวนอกจากจะทำการเริ่มการทำงานของ base_controller แล้วยังมีการเรียกการทำงานของไฟล์ Launch อื่นๆที่อยู่ใน Package ที่ต่างกันอีกด้วย

```
[FindPackageShare('rplidar_ros'),'launch','rplidar_a1_launch.py']
[FindPackageShare('bogie_navigation'),'launch','qr_camera.launch.py']
[FindPackageShare('bogie_description'),'launch','display.launch.py']
[FindPackageShare('serial_imu'),'launch','imu.launch.py']
[FindPackageShare('bogie_navigation'),'config','ekf.yaml']
```

ในการทำหุ่นยนต์ที่สามารถหาแผนที่และนำทางได้นั้น สิ่งสำคัญคือการคำนวณ Odometry ซึ่งคือค่าที่บอกว่าหุ่นยนต์นั้นอยู่ที่ตำแหน่งไหนเทียบกับจุดที่เริ่มวางหุ่นยนต์ จากนั้นจึงสามารถนำไปหาแผนที่ หรือนำทางได้ การได้มาซึ่ง Odometry นั้นสามารถหาได้หลายวิธีโดยในเบื่องต้นคือข้อมูลที่ได้มาจากเอ็นโค้ดเดอร์หรือที่เรียกว่า Wheel Odometry เป็นต้น อย่างไรก็

ตามในขณะที่หุ่นยนต์เคลื่อนที่สิ่งที่หลีกเลี่ยงไม่ได้คือการไถลตัวของหุ่นยนต์ ซึ่งส่งผลโดยตรงให้การคำนวณนั้นเกิดความผิดพลาดสะสม เพื่อลดความผิดพลาดดังกล่าวจึงจำเป็นต้องนำข้อมูลจากเซนเซอร์อื่นมาช่วยหรือชดเชยความผิดพลาดให้ลดลง เช่นข้อมูลจาก lidar หรือ imu ในที่นี้จะใช้งาน Package ที่ชื่อว่า robot localization ซึ่งมี filter ต่างๆ เช่น EKF (Extended Kalman Filters) ไว้สำเร็จรูปให้ใช้งาน โดยวิธีการติดตั้งสามารถทำการติดตั้งทำได้ดังแสดงด้านล่าง

```
sudo apt-get install ros-iron-robot-localization
```

จากไฟล์ Launch ใน ชุดโปรแกรมเริ่มการทำงานหลักของหุ่นยนต์จะพบว่ามีการสั่งการทำงานของ robot localization ดังแสดงด้านล่าง

```
      Node(
           package='robot_localization',
           executable='ekf_node',
           name='ekf_filter_node',
           output='screen',
           remappings=[("/odometry/filtered","/odom")],
           parameters=[Ekf_config_path],
           ),
```

โดยการตั้งค่าตัวแปรของ robot localization จะถูกชี้ไปที่ ชุดโปรแกรมสำหรับหาแผนที่และนำทางโฟลเดอร์ config ที่ชื่อไฟล์ ekf.yaml โดยในไฟล์ดังกล่าวจะตั้งค่าเบื่องต้นให้รวมค่า Wheel Odometry เข้ากับ imu

1.2 การหาแผนที่โดยใช้ Cartographer

SLAM ย่อมาจาก Simultaneous Localization and Mapping ซึ่งหมายถึงการสร้างแผนที่และการค้นหาตำแหน่งของตนเอง ซึ่งส่วนใหญ่แล้วหุ่นยนต์สร้างแผนที่จะสร้างแผนที่ด้วยเซนเซอร์ต่าง ๆ เช่น เซนเซอร์วัดระยะรอบทิศทาง หรือกล้องวัดความลึก โดย ROS2 มีโปรแกรมสำเร็จรูปในการหาแผนที่หลายชุด โปรแกรมสำเร็จรูปแรกที่จะกล่าวถึงได้แก่ Cartographer ซึ่งเป็นระบบการหาแผนที่แบบ real-time ทั้งในรูปแบบสองมิติ และสามมิติ

การทำงานของ Cartographer ประกอบด้วยสองส่วน ได้แก่ ระบบ local SLAM และ global SLAM ในส่วนแรกจะทำการเก็บแผนที่ย่อยๆและเก็บในรูปแบบ submap จากนั้นข้อมูลดังกล่าวจะนำมาประมวลผลใน global SLAM โดยการเชื่อมโยงข้อมูล หารูปแบบต่างๆ และพยายามจับคู่แผนที่ย่อยปัจจุบันกับรูปแบบของแผนที่ก่อนหน้าเพื่อให้ได้แผนที่ที่ตรงกับความจริงมากที่สุด การติดตั้งทำได้ดังแสดงด้านล่าง

```
sudo apt-get install ros-iron-nav2-map-server
sudo apt install ros-iron-cartographer-ros
cd ros2_ws
rosdep install --from-paths src --ignore-src -r -y
colcon build
```

หากการติดตั้งสมบูรณ์ สามารถตรวจสอบการติดตั้งได้ตามคำสั่งด้านล่าง

```
ros2@book:~$ ros2 pkg list |grep cartographer
cartographer_ros
cartographer_ros_msgs
```

จากนั้นให้ลองพิจารณาที่ชุดโปรแกรมสำหรับหาแผนที่และนำทาง ในโฟลเดอร์ launch จะพบไฟล์ชื่อ cartographer.launch.py ไฟล์ดังกล่าวคือไฟล์สั่งการทำงานของ cartographer ซึ่งจะสั่งการทำงานของชุดโปรแกรม cartographer โดยใช้การตั้งค่าที่อยู่ในโฟลเดอร์ config ในไฟล์ชื่อ cartographer.lua โดยไฟล์ Launch มีรายละเอียดดังแสดงด้านล่าง

```
from launch.actions import SetEnvironmentVariable
from ament_index_python.packages import get_package_share_directory
from launch_ros.actions import Node
from launch import LaunchDescription
import os
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():
    share_dir = get_package_share_directory('bogie_navigation')
    rviz_config_file = os.path.join(share_dir, 'config', 'rviz.rviz')
    rviz_node = Node(
        package='rviz2',
        executable='rviz2',
        name='rviz2',
        arguments=['-d', rviz_config_file],
        output='screen'
    )
    return LaunchDescription([
        SetEnvironmentVariable('RCUTILS_LOGGING_BUFFERED_STREAM', '1'),
        Node(
            package='cartographer_ros', executable='cartographer_node',
            output='screen',
            arguments=[
                '-configuration_directory',
              get_package_share_directory('bogie_navigation')+'/config',
                            '-configuration_basename', 'cartographer.lua'
            ],
        ),
        Node(
            package='cartographer_ros',
            executable='cartographer_occupancy_grid_node',
            output='screen',
            arguments=['-resolution','0.02','-publish_period_sec','1.0']
        ),
        rviz_node,
    ])
```

จากไฟล์ launch ข้างต้นเห็นได้ว่ามีการชี้ไปที่โฟลเดอร์ config ไฟล์ชื่อ cartographer.lua โดยไฟล์ดังกล่าจะรวมค่าตัวแปรต่างๆที่ใช้ในการหาแผนที่โดยมีรายละเอียดดังแสดงด้านล่าง

```
include "map_builder.lua"
include "trajectory_builder.lua"
options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_link",
  published_frame = "odom",
  odom_frame = "odom",
  provide_odom_frame = false,
  publish_frame_projected_to_2d = true,
  use_odometry = true,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.5,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}
MAP_BUILDER.use_trajectory_builder_2d = true
TRAJECTORY_BUILDER_2D.min_range = 0.5
TRAJECTORY_BUILDER_2D.max_range = 7.0
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 7.1
TRAJECTORY_BUILDER_2D.use_imu_data = false
TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = false
TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians = math.rad(0.1)
POSE_GRAPH.constraint_builder.min_score = 0.65
POSE_GRAPH.constraint_builder.global_localization_min_score = 0.7
-- POSE_GRAPH.optimize_every_n_nodes = 0
return options
```

จาก cartographer.launch.py และ cartographer.lua ที่แสดงก่อนหน้า ใบเบื้องต้นการตั้งค่าที่ให้มาเพียงพอต่อการหาแผนที่โดยใช้ cartographer แล้วแต่หากผู้ใช้งานต้องการปรับความละเอียดของแผนที่ สามารถเข้าไปปรับได้ที่ไฟล์ cartographer.launch.py ตัวแปร resolution ซึ่งเดิมจะถูกตั้งค่าไว้ที่ 0.02 มีหน่วยเป็นเมตร หมายความว่าในทุกความระยะหนึ่งพิกเซลจะเปรียบเทียบได้กับระยะ 0.02 เมตร หรือ 2 เซนติเมตรนั้นเอง การตั้งค่านี้ขึ้นอยู่กับความสามารถของหน่วยประมวลผลหลักด้วย หากค่าความละเอียดถูกตั้งค่าไว้สูงเกินกว่าที่หน่วยประมวลผลที่ผู้ใช้งานนำมาใช้งานจะรับได้ อาจทำให้การประมวลผลเกิดการล่าช้าเกินไป ดังนั้นควรปรับตั้งให้เหมาะสม และเพียงพอต่อการใช้งานของแต่ละหน่วยประมวลผล

จากนั้นทำการ Build Package และสั่งเริ่มการทำงานของโปรแกรมดังตัวอย่างด้านล่าง

```
ros2@book: ~$ ros2 launch bogie_bringup bringup.launch.py

[base_controller.py-8] [INFO] [1705987231.782257560]
[base_controller]: left target = 0 left feedback = 0 right target
= 0 right feedback = 0
...
...
...
```

จากคำสั่งข้างต้น ROS จะสร้าง Node ชื่อ base_controller ที่รอรับ Topic ชื่อ cmd_vel ในตอนนี้สามารถส่ง Topic cmd_vel ได้ด้วยคีย์บอร์ด โดยเปิด Terminal ใหม่ขึ้นมาจากนั้นสามารถสั่งเริ่มการทำงานได้โดยใช้คำสั่งด้านล่างเพื่อทำการบังคับหุ่นยนต์ขณะทำการหาแผนที่

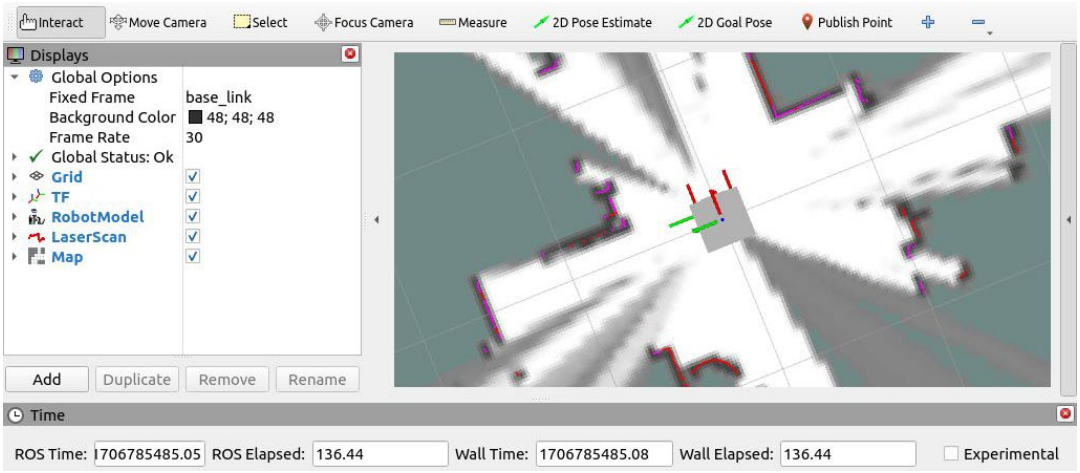
```
ros2@book:~$ ros2 run bogie_navigation teleop
```

ในตอนนี้จะมี Node my_teleop ที่ทำการส่ง Topic ชื่อ cmd_vel จากนั้นผู้รับข้อมูล นั่นก็คือ base_controller จะทำการรับ cmd_vel เพื่อทำการคำนวณเป็นความเร็วล้อซ้ายและล้อขวาเพื่อส่งให้บอร์ดควบคุมทำการควบคุมความเร็วล้อให้เป็นไปตามการสั่งการต่อไป

จากนั้นทำการเริ่มการทำงานของระบบการหาแผนที่โดยใช้ Cartographer โดยใช้คำสั่งด้านล่าง

```
ros2@book:~$ ros2 launch bogie_navigation cartographer.launch.py
```

จากคำสั่งดังกล่าวจะเป็นการเริ่มต้นหาแผนที่โดยใช้ Cartographer โดยสามารถใช้การส่ง cmd_vel จาก Node my_teleop เพื่อดำเนินการหาแผนที่ได้ กล่าวคือใช้คีย์บอร์ดเพื่อควบคุมหุ่นยนต์ไปยังตำแหน่งต่างๆที่ต้องการเก็บแผนที่นั่นเอง โดยภาพตัวอย่างการ แผนที่โดยใช้ Cartographer ดังแสดงในภาพ



ภาพตัวอย่างการ แผนที่โดยใช้ Cartographer

จากนั้นทำการบันทึกแผนที่ที่เราพอใจแล้ว โดยการเปิดหน้าต่าง Terminal ใหม่ และใช้คำสั่งด้านล่าง โดยข้อความที่ตามหลังฟังชั่นก์ -f คือตำแหน่งและชื่อไฟล์ที่ต้องการบันทึก

```
ros2 run nav2_map_server map_saver_cli [arguments] [--ros-args ROS remapping args]
```

Map server เป็นเครื่องมือที่จัดเตรียมการจัดการทางด้านแผนที่เพื่อนำไปใช้กับระบบนำทางใน ros2 เช่น การสั่งการ map_server map_saver และ ไลบรารี map_io โดย map_server มีหน้าที่เรียกใช้แผนที่จากไฟล์มาใช้งาน map_saver มีหน้าที่บันทึกแผนที่ลงในไฟล์ และ map_io เป็นไลบรารีทั้งขาเข้าและขาออกของแผนที่ โดยไลบรารีได้รับการออกแบบมาให้ไม่ขึ้นกับวัตถุเพื่อให้สามารถบันทึกเรียกใช้แผนที่จากคำสั่งภายนอกได้

โดยผลลัพธ์ของการเปิดการทำงานเพื่อบันทึกแผนที่ดังแสดงด้านล่าง

```
ros2@book:~$ ros2 run nav2_map_server map_saver_cli -f
~/ros2_ws/src/bogie_navigation/maps/maptest

[INFO] [1706785575.062942777] [map_saver]:
        map_saver lifecycle node launched.
        Waiting on external lifecycle transitions to activate
        See https://design.ros2.org/articles/node_lifecycle.html
for more information.

[INFO] [1706785575.063618749] [map_saver]: Creating
...
...
...
unspecified. Setting it to default value: 0.250000
[WARN] [1706785575.080769729] [map_saver]: Occupied threshold
unspecified. Setting it to default value: 0.650000
[WARN] [map_io]: Image format unspecified. Setting it to: pgm
[INFO] [map_io]: Received a 310 X 408 map @ 0.02 m/pix
...
...
...
[INFO] [1706785575.679565468] [map_saver]: Map saved successfully
```

จากผลการเริ่มการทำงานดังกล่าวจะสร้างไฟล์ขึ้นมาสองไฟล์ตามตำแหน่งการบันทึกไฟล์ที่ได้ทำการระบุไว้ได้แก่ไฟล์นามสกุล yaml และ pgm ในไฟล์ yaml จะเป็นกำรอธิบายรายละเอียดแผนที่ดังแสดงตามตัวอย่างด้านล่าง ส่วนในส่วนไฟล์ pgm จะเป็นไฟล์รูปภาพแผนที่ที่ได้ทำการหา ซึ่งไฟล์ทั้งสองจะถูกนำไปใช้งานในการนำทางในบทต่อไป

```yaml
image: maptest.pgm
mode: trinary
resolution: 0.02
origin: [-2.45, -3.46, 0]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

5 การนำทางด้วย DWB Controller

DWB เป็นวิธีการนำทางที่พัฒนาต่อยอดมาจาก DWA (Dynamic Window Approach) ซึ่งเป็นวิธีที่ได้รับความนิยมสำหรับการวางแผนในการหลบหลีกสิ่งกีดขวางและหลีกเลี่ยงจากการชน ซึ่งเป็นวิธีการเลือกความเร็วที่สามารถ เข้าถึงจุดเป้าหมายได้อย่างรวดเร็วโดยไม่ต้องเจอกับสิ่งกีดขวางที่อาจจะทำให้ชนกับหุ่นยนต์ โดยใช้หลักการ Dynamic Window โดยมีฟังก์ชัน $G(V, \omega)$ ใช้ในการคำนวณหาความเร็วในการเคลื่อนที่เชิงเส้น และความเร็วในการเคลื่อนที่เชิงมุม การใช้ DWA เหมาะกับสถานที่ที่มีฉากชัดเจน และมีสิ่งกีดขวางที่ไม่เปลี่ยนแปลงมากนัก การติดตั้งทำได้ดังแสดงด้านล่าง

```
sudo apt-get install ros-iron-nav2-costmap-2d
sudo apt-get install ros-iron-nav2-core
sudo apt-get install ros-iron-nav2-behaviors
sudo apt install ros-iron-nav2-bringup
```

หากการติดตั้งสมบูรณ์ สามารถตรวจสอบการติดตั้งได้ตามคำสั่งด้านล่าง

```
ros2@book:~$ ros2 pkg list |grep iron-nav2
...
nav2_behavior_tree
nav2_bringup
nav2_core
nav2_costmap_2d
...
...
```

จากนั้นให้ลองพิจารณาที่ชุดโปรแกรมสำหรับหาแผนที่และนำทาง ในโฟลเดอร์ launch จะพบไฟล์ชื่อ nav2_DWB.launch.py ไฟล์ดังกล่าวคือไฟล์สั่งการทำงานของ DWB Controller ซึ่งจะสั่งการทำงานของชุดโปรแกรม DWB โดยใช้การตั้งค่าที่อยู่ในโฟลเดอร์ config ในไฟล์ชื่อ nav2_DWB_params.yaml โดยไฟล์ Launch มีรายละเอียดดังแสดงด้านล่าง โดยตัวแปร map_dir จะเก็บตำแหน่งและชื่อของแผนที่ที่ต้องการใช้ หากต้องการใช้แผนที่ที่ชื่อต่างออกไปต้องแก้ไขตำแหน่งนี้ด้วยเช่นกัน

```
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription, DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from launch.launch_description_sources import PythonLaunchDescriptionSource
from ament_index_python.packages import get_package_share_directory
import os
def generate_launch_description():
    share_dir = get_package_share_directory('bogie_navigation')
    nav2_share_dir = get_package_share_directory('nav2_bringup')
    use_sim_time = LaunchConfiguration('use_sim_time', default='false')
    map_dir = LaunchConfiguration(
        'map',default = os.path.join(share dir,'maps', 'maptest.yaml'))
    param_dir = LaunchConfiguration(
        'params file',
        default = os.path.join(share_dir,'config', 'nav2_DWB_params.yaml'))
    nav2 launch file dir = os.path.join(nav2 share dir,'launch')
    rviz_config_dir = os.path.join(nav2_share_dir,'rviz',
                                   'nav2 default view.rviz')
    return LaunchDescription([
        DeclareLaunchArgument(
            'map',default_value = map_dir,description = 'Path of map'),
        DeclareLaunchArgument(
            'params_file',default_value = param_dir,),
        DeclareLaunchArgument(
            'use_sim_time',default_value = 'false',),
        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([nav2_launch_file_dir,
                                           '/bringup_launch.py']),
            launch_arguments = {'map':map_dir,'use_sim_time': use_sim_time,
                                'params_file': param_dir}.items(),
        ),
        Node(
            package= 'rviz2', executable= 'rviz2',
            name= 'rviz2', arguments= ['-d', rviz_config_dir],
            parameters = [{'use_sim_time': use_sim_time}],
            output = 'screen',
            ),
    ])
```

จากไฟล์ข้างต้นเห็นได้ว่ามีการชี้ไปที่โฟลเดอร์ config ไฟล์ชื่อ nav2_DWB_params.yaml โดยไฟล์ดังกล่าวจะรวมค่าตัวแปรต่างๆที่ใช้ในการนำทางโดยใช้ DWA ซึ่งสามารถดูได้จากไฟล์ที่แนบมาด้วยกับหนังสือเล่มนี้ ในเบื้องต้นตัวแปรต่างๆได้ดูตั้งค่าไว้เพียงพอต่อการศึกษาเบื้องต้นแล้ว อย่างไรก็ตามเพื่อประสิทธิภาพที่ดีที่สุด ผู้ศึกษาควรปรับแต่งค่าตัวแปรต่างๆให้เหมาะสมกับสภาวะการใช้งานของตัวเอง จากนั้นทำการ Build Package และสั่งเริ่มการทำงานของโปรแกรมดังตัวอย่างด้านล่าง

```
ros2@book: ~$ ros2 launch bogie_bringup bringup.launch.py

[base_controller.py-8] [INFO] [1705987231.782257560]
[base_controller]: left target = 0 left feedback = 0 right target
= 0 right feedback = 0
...
...
...
```

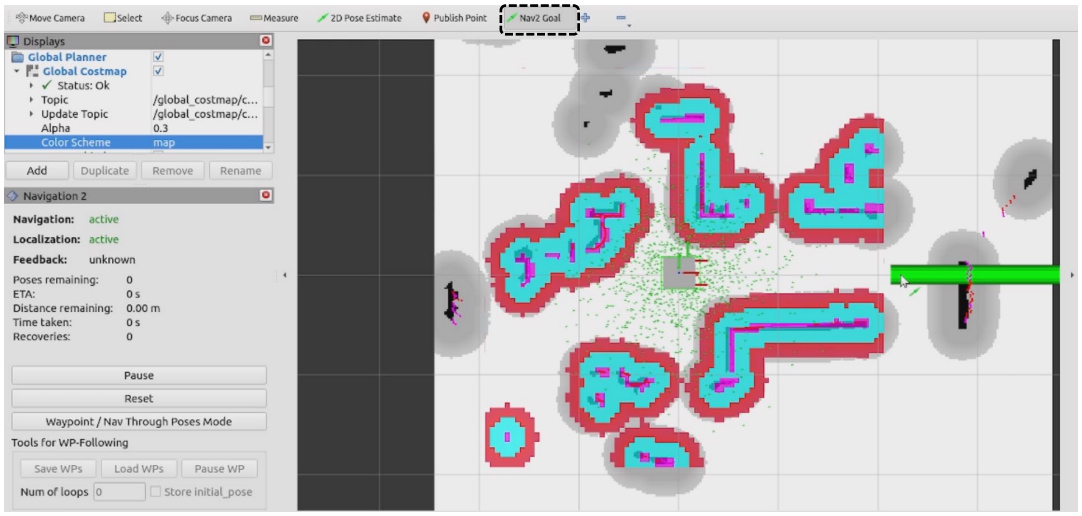
จากคำสั่งข้างต้น ROS จะสร้าง Node ชื่อ base_controller ที่รอรับ Topic ชื่อ cmd_vel ซึ่งข้อความดังกล่าวจะถูกส่งไปจากระบบนำทาง จากนั้นเปิด Terminal ใหม่ขึ้นมาและสั่งเริ่มการทำงาน DWB ได้โดยใช้คำสั่งด้านล่าง

```
ros2@book:~$ ros2 launch bogie_navigation navigation_DWB.launch
```

เมื่อถึงตรงนี้จะมีหน้าต่าง rviz ขึ้นมาโดยการสั่งการให้สั่งให้หุ่นยนต์เคลื่อนที่ไปยังเป้าหมายในระบบสามารถทำได้สองวิธีได้แก่การสั่งการทำงานในโปรแกรม Rviz หรือเขียนโปรแกรมเข้าไปเพื่อส่งข้อความเข้าไปสั่งการโดยตรง ในที่นี้จะทดสอบเบื้องต้นโดยการสั่งการผ่าน rviz ในตอนนี้ให้กดที่ปุ่ม 2d pose estimate จากนั้นทำการคลิกที่ตำแหน่งในแผนที่ในตำแหน่งที่หุ่นวางอยู่ เพื่อให้ระบบประมาณตำแหน่งของตัวเองเบื้องต้น

จากนั้นส่งคำสั่งตำแหน่งและทิศทางที่จะให้หุ่นยนต์เดินทางไปโดยกดที่ปุ่ม Nav2 Goal จากนั้นทำการคลิกที่ตำแหน่งในแผนที่ในตำแหน่งที่ต้องการให้หุ่นเคลื่อนที่ไปดังแสดงในภาพ จะพบว่าหุ่นยนต์จะเคลื่อนตัวไปยังเป้าหมายดังแสดงในภาพ

ภาพตัวอย่างการประมาณตำแหน่งโดย 2d pose estimate



ภาพตัวอย่างการระบุตำแหน่งเป้าหมายโดย Nav2 Goal

ภาพตัวอย่างการนำทางโดย DWB