การพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6 องศาอิสระ

โดย

นายสฤษฎ์พงศ์ มีบุญ รหัสนักศึกษา 643040306-2

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ประมินทร์ อาจฤทธิ์

สาขาวิชาวิศวกรรมไฟฟ้า

หลักสูตร วิศวกรรมระบบอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น

ปีการศึกษา 2567

ใบประเมินผลงาน

ชื่อเรื่องภาษาไทย การพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6 องศาอิสระ

ชื่อเรื่องภาษาอังกฤษ Development of control software and simulation model a 6 DOF robotic arm

ผู้จัดทำ

นายสฤษฎ์พงศ์ มีบุญ รหัสนักศึกษา 643040306-2

อาจารย์ที่ปรึกษา

_____

(รองศาสตราจารย์ประมินทร์ อาจฤทธิ์)

อาจารย์ผู้ร่วมประเมิน

_____

(.....)

_____

(.....)

ประเมินผล ณ วันที่ 27 มีนาคม 2568

กิตติกรรมประกาศ

โครงงานเรื่อง การพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6 องศาอิสระ (Development of Control Software and Simulation Model for a 6DOF Robotic Arm) จัดทำขึ้นในรายวิชา ELECTRONIC SYSTEMS ENGINEERING PROJECT โดยโครงงานนี้สำเร็จลุล่วงไปได้ด้วยดี อันเป็นผลมาจากการสนับสนุนด้านวัสดุอุปกรณ์ คำแนะนำ และความช่วยเหลือจากหลายฝ่าย ขอขอบพระคุณ รองศาสตราจารย์ประมินทร์ อาจฤทธิ์ เป็นอย่างสูง ที่ให้ความรู้และคำแนะนำตลอดการดำเนินโครงงาน ทำให้โครงงานนี้สามารถสำเร็จลุล่วงไปได้ด้วยดี นอกจากนี้ ขอขอบคุณ นายเจษฎา ประณามตรังค์ และ นายบุญเต็ม จิกจักร์ คณะผู้จัดทำ ที่ร่วมแรงร่วมใจดำเนินโครงงานจนบรรลุเป้าหมายตามที่ตั้งไว้ สุดท้ายนี้ คณะผู้จัดทำหวังเป็นอย่างยิ่งว่าโครงงานนี้จะเป็นประโยชน์แก่ผู้ที่สนใจศึกษาและพัฒนาต่อยอดในอนาคต

นายสฤษฎ์พงศ์ มีบุญ

บทคัดย่อ

การพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6 องศาอิสระ (Development of Control Software and Simulation Model for a 6DOF Robotic Arm) มีความสำคัญในการช่วยทดสอบและฝึกการใช้งานหุ่นยนต์ผ่านระบบจำลอง (*simulation*) ซึ่งช่วยลดต้นทุนและความเสี่ยงในการทดสอบจริง โครงงานนี้มุ่งเน้นการพัฒนาและทดสอบซอฟต์แวร์ควบคุมสำหรับหุ่นยนต์แขนกล 6 DOF โดยใช้ *Robot Operating System* (ROS) และ *Arduino Mega 2560* ร่วมกับมอเตอร์สเต็ปเปอร์ เพื่อให้สามารถกำหนดค่ามุมข้อต่อและทิศทางการเคลื่อนที่ได้อย่างแม่นยำ

ในส่วนของการจำลอง ได้ใช้ Gazebo ซึ่งเป็นซอฟต์แวร์จำลองทางฟิสิกส์ที่สามารถเลียนแบบพฤติกรรมของหุ่นยนต์ในสภาพแวดล้อมเสมือนจริง โดยรวมถึงแรงโน้มถ่วง, แรงเสียดทาน และแรงเฉื่อย นอกจากนี้ยังใช้ Rviz ซึ่งเป็นเครื่องมือแสดงผลข้อมูลสามมิติของหุ่นยนต์ในระบบ ROS เพื่อให้ผู้ใช้สามารถกำหนดค่ามุมข้อต่อและสังเกตการเคลื่อนที่ของหุ่นยนต์

โครงงานนี้แบ่งการทดลองออกเป็นสองส่วน ได้แก่ การควบคุมการเคลื่อนที่ของแบบจำลองใน Gazebo และการควบคุมหุ่นยนต์จริงโดยใช้หลักการเดียวกัน ผลการทดลองแสดงให้เห็นว่าระบบจำลองสามารถช่วยประเมินประสิทธิภาพของหุ่นยนต์และลดความเสี่ยงที่อาจเกิดขึ้นในการทดสอบจริง นอกจากนี้ยังได้ทำการเปรียบเทียบความแตกต่างระหว่างการเคลื่อนที่ใน Gazebo และหุ่นยนต์จริง โดยพิจารณาปัจจัยต่างๆ เช่น ความแม่นยำของตำแหน่ง, เวลาตอบสนอง และเสถียรภาพของการเคลื่อนที่

ผลการทดลองแสดงให้เห็นถึงศักยภาพของระบบในการนำไปประยุกต์ใช้ทั้งในงานอุตสาหกรรมและการศึกษา โดยสามารถใช้เป็นแนวทางในการพัฒนาระบบควบคุมหุ่นยนต์แขนกลให้มีประสิทธิภาพมากยิ่งขึ้น

สารบัญ

หน้า

สารบัญภาพ

สารบัญตาราง

บทที่ 1 บทนำ

1.1 หลักการและเหตุผล

ในยุคปัจจุบัน เทคโนโลยีหุ่นยนต์มีบทบาทสำคัญในอุตสาหกรรมหลายประเภท โดยเฉพาะ หุ่นยนต์แขนกลแบบ Articulated Robot ซึ่งเป็นหุ่นยนต์ที่สามารถเคลื่อนที่ได้หลายทิศทางด้วยข้อต่อที่มีความซับซ้อน โดยทั่วไป หุ่นยนต์ประเภทนี้จะมีข้อต่อจำนวน 6 จุด ทำให้สามารถเคลื่อนที่ได้อย่างแม่นยำและรองรับงานที่มีความซับซ้อนสูง เช่น การเชื่อม การประกอบชิ้นส่วน หรือการบรรจุสินค้า เป็นต้น

โครงงานนี้มุ่งเน้นการพัฒนาชุดจำลองหุ่นยนต์แขนกล ผ่านระบบจำลองการทำงาน (Simulation) ซึ่งมีความสำคัญอย่างมากในการออกแบบและทดสอบการควบคุมหุ่นยนต์ เนื่องจากการทดลองกับฮาร์ดแวร์จริงมักมีข้อจำกัดด้านต้นทุนและความปลอดภัย การใช้ระบบจำลองการทำงาน ช่วยให้สามารถจำลองการเคลื่อนที่ของหุ่นยนต์ได้อย่างแม่นยำก่อนนำไปใช้งานจริง โดยสามารถวิเคราะห์และตรวจสอบพฤติกรรมการเคลื่อนที่ของแขนกลในสภาพแวดล้อมจำลอง เพื่อดูว่าสามารถทำงานได้ถูกต้องหรือไม่ นอกจากนี้ยังสามารถใช้การตรวจสอบผลลัพธ์ระหว่างการจำลองกับการทำงานจริง เพื่อเปรียบเทียบความแม่นยำและหาค่าความคลาดเคลื่อน

ระบบที่พัฒนาขึ้นจะใช้ ROS (Robot Operating System) ร่วมกับ Gazebo และ RViz เพื่อจำลองสภาพแวดล้อมและการทำงานของหุ่นยนต์ โดยหุ่นยนต์จะได้รับข้อมูลการควบคุมจากซอฟต์แวร์ที่พัฒนาขึ้น และทำการเคลื่อนที่ในระบบจำลองการทำงาน ซึ่งจะช่วยให้สามารถตรวจสอบข้อผิดพลาด ปรับปรุงพารามิเตอร์ของระบบ และเพิ่มประสิทธิภาพในการควบคุมก่อนที่จะนำไปใช้งานกับฮาร์ดแวร์จริง โดยการใช้แบบจำลองในโครงงานนี้มีเป้าหมายเพื่อ ลดข้อผิดพลาดในการพัฒนาและควบคุมหุ่นยนต์ วิเคราะห์พฤติกรรมของหุ่นยนต์ก่อนนำไปใช้งานจริง เปรียบเทียบผลการทำงานระหว่างระบบจำลองและฮาร์ดแวร์ และเพิ่มความแม่นยำในการควบคุมหุ่นยนต์และลดต้นทุนการทดลอง

ดังนั้น การนำแบบจำลองมาใช้เป็นเครื่องมือหลักในการพัฒนาซอฟต์แวร์ควบคุมหุ่นยนต์จึงมีความจำเป็นอย่างยิ่ง เพราะช่วยให้สามารถตรวจสอบและปรับปรุงระบบได้อย่างมีประสิทธิภาพ ก่อนที่จะนำไปทดสอบและใช้งานจริงกับหุ่นยนต์แขนกล 6 องศาอิสระในอนาคต

1.2 วัตถุประสงค์

โครงงาน การพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6 องศาอิสระ (Development of control software and simulation model a 6DOF robotic arm) มีวัตถุประสงค์ของโครงงานดังต่อไปนี้

1. เพื่อปรับปรุงและพัฒนาโปรแกรมควบคุมหุ่นยนต์และสร้างแบบจำลองการทำงานของหุ่นยนต์แขนกล 6 องศา อิระ (6DOF)
2. เพื่อทดสอบและวิเคราะห์การทำงานของแขนหุ่นยนต์ในสภาพแวดล้อมจำลองก่อนนำไปใช้งานจริง

1.3 ขอบเขตของงาน

โครงงาน การพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6องศาอิสระ (Development of control software and simulation model a 6DOF robotic arm) มีขอบเขตของโครงงานดังต่อไปนี้

1. จำลองหุ่นยนต์อุตสาหกรรมประเภท Articulated Robot โดยมีข้อต่อทั้งหมด 6 จุด
2. ปรับปรุงและพัฒนาโปรแกรมควบคุมหุ่นยนต์จากโปรแกรมที่มีอยู่เดิม โดยทำงานบน ROS
3. สร้างแบบจำลองการทำงานของหุ่นยนต์แขนกลในสภาพแวดล้อมเสมือนจริง

1.4 แผนการดำเนินงาน

	รายการ	พ.ศ. 2567						พ.ศ. 2568		
		ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.
1	เสนอหัวข้อและชื่ออาจารย์ที่ปรึกษาโครงงาน									
2	เขียนแผนการดำเนินงานและข้อเสนอโครงงาน									
3	ศึกษาและค้นคว้าทฤษฎีที่เกี่ยวข้อง									
4	ปรับปรุงและพัฒนาโปรแกรมควบคุมหุ่นยนต์									
5	ทดสอบและบันทึกผล									
6	วิเคราะห์ผลทดสอบและปรับปรุงแก้ไข									
7	สรุปผลการดำเนินงานและเตรียมนำเสนอโครงงาน									

ตารางที่ 1.1 แผนการดำเนินงาน

1.5 แนวทางการดำเนินงาน

โครงงานการพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6 องศาอิสระ เป็นการนำโครงงานเรื่อง การออกแบบชุดจำลองแขนหุ่นยนต์อุตสาหกรรม (Design of an Industrial Robot Arm) [4] ที่เป็นชุดจำลองแขนหุ่นยนต์อุตสาหกรรมประเภท Articulated Robot โดยมีข้อต่อทั้งหมด 6 จุด มาพัฒนาต่อ

ยอดในด้านโปรแกรมให้มีความง่ายต่อการใช้งานและมีประสิทธิภาพมากยิ่งขึ้น โดยได้ดำเนินการตามแผนที่กำหนดไว้ โดยเริ่มจากการศึกษาข้อมูลพื้นฐานเกี่ยวกับหุ่นยนต์แขนกลและระบบควบคุม จากนั้นทำการออกแบบและพัฒนาซอฟต์แวร์ควบคุมให้สามารถทำงานร่วมกับ ROS Noetic, Arduino และ RViz/Gazebo ได้อย่างมีประสิทธิภาพ

หลังจากการพัฒนาได้มีการทดสอบระบบเพื่อประเมินประสิทธิภาพและความถูกต้องของการควบคุม โดยทดสอบการรับส่งข้อมูลระหว่าง ROS และ Arduino รวมถึงตรวจสอบการเคลื่อนที่ของมอเตอร์สเต็ปเปอร์ทั้งในซอฟต์แวร์จำลองและฮาร์ดแวร์จริง เพื่อให้ได้ผลลัพธ์ที่แม่นยำและมีเสถียรภาพมากขึ้น สุดท้ายได้มีการวิเคราะห์ผลการทดสอบ พร้อมทั้งปรับปรุงซอฟต์แวร์ให้เหมาะสม ก่อนทำการสรุปผลและจัดทำรายงานเพื่อนำเสนอต่ออาจารย์ที่ปรึกษา ซึ่งโครงงานนี้สามารถเป็นแนวทางสำหรับการพัฒนาระบบควบคุมหุ่นยนต์แขนกลให้มีประสิทธิภาพยิ่งขึ้นในอนาคต

1.6 ผลที่คาดว่าจะได้รับ

การพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6องศาอิสระ (Development of control software and simulation model a 6DOF robotic arm) มีผลที่คาดว่าจะได้รับดังต่อไปนี้

1. ได้ความรู้ความเข้าใจในส่วนประกอบ กลไก ทฤษฎีที่เกี่ยวข้อง และหลักการทำงานของหุ่นยนต์อุตสาหกรรม

2. ได้โปรแกรมในการทดสอบการควบคุมหุ่นยนต์ผ่านหุ่นยนต์จำลอง เพื่อนำมาวิเคราะห์การทำงานของหุ่นยนต์จากแบบจำลองสภาพแวดล้อมเสมือนจริง

บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

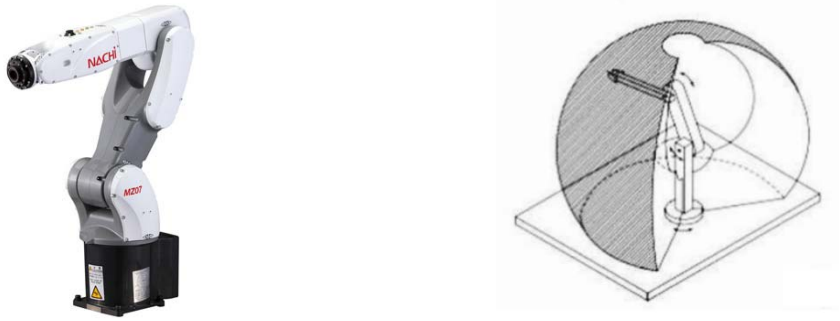
ในการพัฒนาหุ่นยนต์แขนกล 6 องศาอิสระ (6DOF) การสร้างแบบจำลองและการจำลองการทำงานของหุ่นยนต์ (Robot Simulation) เป็นขั้นตอนสำคัญที่ช่วยให้สามารถทดสอบและวิเคราะห์ระบบได้ก่อนการใช้งานจริง การจำลอง (Simulation) ช่วยให้สามารถประเมินพฤติกรรมของหุ่นยนต์ในสภาพแวดล้อมเสมือนจริง ตรวจสอบความถูกต้องของซอฟต์แวร์ควบคุม และลดความเสี่ยงจากข้อผิดพลาดที่อาจเกิดขึ้นเมื่อนำไปใช้งานจริง

โครงงานนี้มุ่งเน้นการพัฒนาชุดจำลองแขนหุ่นยนต์อุตสาหกรรมประเภท Articulated Robot โดยเฉพาะการปรับปรุงด้านโปรแกรมควบคุม เพื่อให้สามารถใช้งานได้ง่ายขึ้น โดยพัฒนาอยู่บนพื้นฐานระบบปฏิบัติการ Linux และใช้ ROS (Robot Operating System) เป็นแพลตฟอร์มหลักในการพัฒนา นอกจากนี้ การจำลองการทำงานของหุ่นยนต์ยังต้องอาศัยองค์ความรู้ที่เกี่ยวข้องหลายด้านดังนี้

2.1 หุ่นยนต์แขนกล 6 องศาอิสระ (6 DOF Robotic Arm)

หุ่นยนต์แขนกล 6 องศาอิสระ (6 Degrees of Freedom, 6 DOF) [1] เป็นหุ่นยนต์ที่สามารถเคลื่อนที่ได้ใน 6 ทิศทาง ได้แก่ 3 ทิศทางการเคลื่อนที่เชิงเส้น (Translational Movement) บนแกน X, Y, และ Z และ 3 ทิศทางการหมุนรอบแกน (Rotational Movement) การเคลื่อนที่ในหลายทิศทางนี้ทำให้หุ่นยนต์มีความยืดหยุ่นในการหยิบจับหรือเคลื่อนย้ายวัตถุในตำแหน่งและทิศทางต่าง ๆ

แขนกล 6 DOF มักประกอบด้วยข้อต่อ 6 ข้อ (Joint) ที่แต่ละข้อสามารถหมุนหรือเลื่อนเพื่อควบคุมการเคลื่อนที่ของแขนกล จุดเด่นของแขนกลแบบนี้คือความสามารถในการเคลื่อนที่และกำหนดตำแหน่งของปลายแขน (End-Effector) ได้อย่างอิสระ ซึ่งทำให้มันสามารถเข้าถึงจุดต่าง ๆ ในพื้นที่ทำงานได้ดี และสามารถใช้งานได้ในหลากหลายงานอุตสาหกรรม เช่น การหยิบจับชิ้นงาน การเชื่อม การประกอบชิ้นส่วน หรือการเคลื่อนย้ายวัตถุที่ต้องการความแม่นยำสูง



ก) *ข)*

รูปที่ 2.1 ก) 6 DOF Robot ข) พื้นที่การทำงานของ 6 DOF Robot

ความท้าทายในการควบคุมแขนกล 6DOF คือความซับซ้อนในการคำนวณทิศทางและตำแหน่งของข้อต่อและปลายแขน ซึ่งจำเป็นต้องมีการใช้วิธีการทางคณิตศาสตร์ในการคำนวณ เช่น kinematics ที่จะกล่าวถึงในหัวข้อต่อไป

2.3 จลนศาสตร์ของหุ่นยนต์ (Robot Kinematics)

Kinematics เป็นศาสตร์ที่ศึกษาเกี่ยวกับการเคลื่อนที่ของวัตถุโดยไม่คำนึงถึงแรงที่ทำให้เกิดการเคลื่อนที่ ในการควบคุมหุ่นยนต์ Kinematics ถูกใช้ในการคำนวณทิศทางและตำแหน่งของหุ่นยนต์ เพื่อให้หุ่นยนต์สามารถเคลื่อนที่ได้ตามคำสั่งที่กำหนด ซึ่ง Kinematics แบ่งออกเป็นสองประเภทหลัก ได้แก่:

1. จลนศาสตร์ไปข้างหน้า (FK): Forward Kinematics เป็นการคำนวณตำแหน่งและทิศทางของปลายแขน (End-Effector) เมื่อทราบมุมของข้อต่อแต่ละข้อในแขนกล การคำนวณ FK จะใช้สมการทางคณิตศาสตร์ในการกำหนดตำแหน่งของ End-Effector ในพิกัดของพื้นที่ทำงาน ตัวอย่างเช่น หากทราบมุมของข้อต่อที่ 1 ถึงข้อต่อที่ 6 ของหุ่นยนต์แขนกล 6DOF เราสามารถใช้ FK เพื่อคำนวณตำแหน่งของปลายแขนในพิกัด X, Y, Z การคำนวณนี้เป็นพื้นฐานในการตั้งโปรแกรมควบคุมการเคลื่อนที่ของหุ่นยนต์

2. จลนศาสตร์ผกผัน (IK): Inverse Kinematics ตรงข้ามกับ Forward Kinematics โดย IK เป็นการคำนวณมุมของข้อต่อแต่ละข้อจากตำแหน่งและทิศทางที่ต้องการของ End-Effector ตัวอย่างเช่น หากเราต้องการให้ปลายแขนกลไปยังจุดใดจุดหนึ่งในพื้นที่ทำงาน เราจำเป็นต้องคำนวณมุมของข้อต่อแต่ละข้อให้สอดคล้องกับตำแหน่งนั้น การคำนวณ IK มีความซับซ้อนกว่า FK เพราะอาจมีหลายคำตอบในการแก้ปัญหาหรืออาจไม่มีคำตอบเลย ในกรณีที่ตำแหน่งที่ต้องการอยู่นอกขอบเขตการเคลื่อนที่ของหุ่นยนต์

การคำนวณ FK และ IK จึงเป็นองค์ประกอบที่สำคัญในการควบคุมหุ่นยนต์แขนกล และเป็นส่วนสำคัญที่ต้องพิจารณาในการออกแบบซอฟต์แวร์ควบคุม

2.3 ระบบควบคุมหุ่นยนต์ (Robotic Control System)

การควบคุมหุ่นยนต์ (Robotic Control) มีวัตถุประสงค์เพื่อให้หุ่นยนต์ทำงานได้อย่างแม่นยำตามคำสั่งที่กำหนด โดยระบบควบคุมแบ่งออกเป็นสองประเภทหลัก คือ

1. Open-loop Control เป็นระบบการควบคุมที่ไม่ต้องการการป้อนกลับ (Feedback) การควบคุมประเภทนี้มักจะส่งคำสั่งตรงไปยังมอเตอร์หรือข้อต่อโดยไม่ตรวจสอบว่าผลลัพธ์เป็นอย่างไร ตัวอย่างของการควบคุมประเภทนี้คือการส่งสัญญาณให้มอเตอร์หมุนไปที่มุมที่กำหนดโดยไม่ตรวจสอบตำแหน่งของมอเตอร์ ระบบ Open-loop Control ง่ายต่อการออกแบบและใช้งาน แต่มีข้อเสียคือขาดความแม่นยำเมื่อต้องทำงานในสภาพแวดล้อมที่มีการเปลี่ยนแปลงหรือมีสิ่งรบกวน

2. Closed-loop Control (Feedback Control) หรือระบบที่มีการป้อนกลับเป็นการควบคุมที่สามารถตรวจสอบผลลัพธ์ของการเคลื่อนที่และทำการปรับปรุงได้ตลอดเวลา ข้อมูลจากเซนเซอร์ เช่น เซนเซอร์ตรวจจับตำแหน่ง (Position sensor) หรือเซนเซอร์ความเร็ว (Velocity sensor) จะถูกใช้เพื่อตรวจสอบสถานะของหุ่นยนต์ และหากพบว่าผลลัพธ์ไม่เป็นไปตามคำสั่ง ระบบจะทำการปรับปรุงเพื่อให้หุ่นยนต์ทำงานได้แม่นยำยิ่งขึ้น ระบบ Closed-loop Control จึงเหมาะกับการควบคุมหุ่นยนต์ที่ต้องการความแม่นยำสูง เช่น แขนกลในสายการผลิตที่ต้องการการเคลื่อนที่ซ้ำๆ อย่างแม่นยำ

การใช้ Closed-loop Control ในหุ่นยนต์แขนกล 6 DOF จะช่วยให้มั่นใจได้ว่าหุ่นยนต์สามารถเคลื่อนที่ไปยังตำแหน่งที่ต้องการได้อย่างถูกต้อง แม้จะมีการเปลี่ยนแปลงสภาพแวดล้อมหรือมีความคลาดเคลื่อนเกิดขึ้น

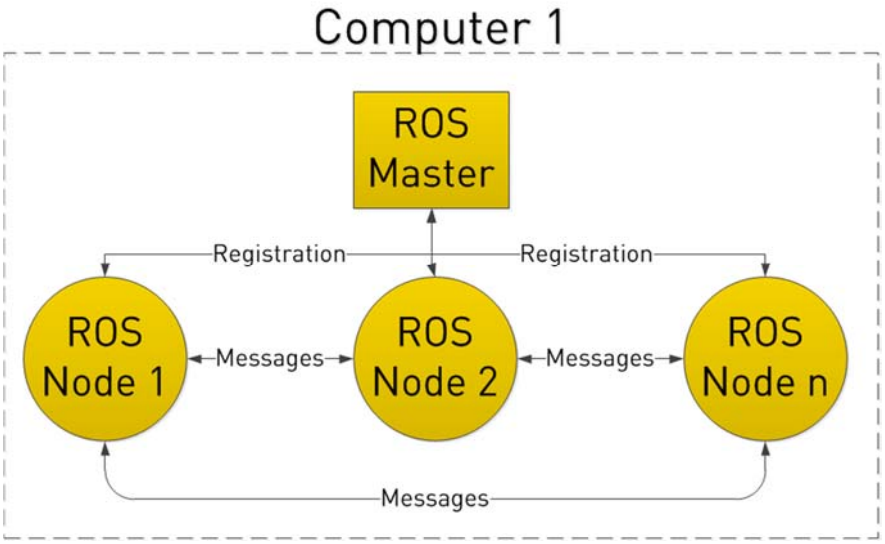
2.4 Robot Operating System (ROS)

Robot Operating System (ROS) เป็นระบบปฏิบัติการที่ออกแบบมาเพื่อใช้ในการพัฒนาหุ่นยนต์ โดย ROS ไม่ใช่ระบบปฏิบัติการในความหมายของ OS ทั่วไป เช่น Windows หรือ Linux แต่เป็นชุดของเครื่องมือ ไลบรารี และเฟรมเวิร์กที่ช่วยให้การพัฒนาหุ่นยนต์เป็นไปอย่างมีประสิทธิภาพ

การทำงานของ ROS นั้นใช้โครงสร้างแบบ Node-based Communication ซึ่งหมายความว่าฟังก์ชันแต่ละส่วนของหุ่นยนต์จะถูกแบ่งเป็นโหนด (Node) และแต่ละโหนดสามารถสื่อสารกันผ่าน Topic หรือ Service ตามลักษณะการรับส่งข้อมูลดังนี้

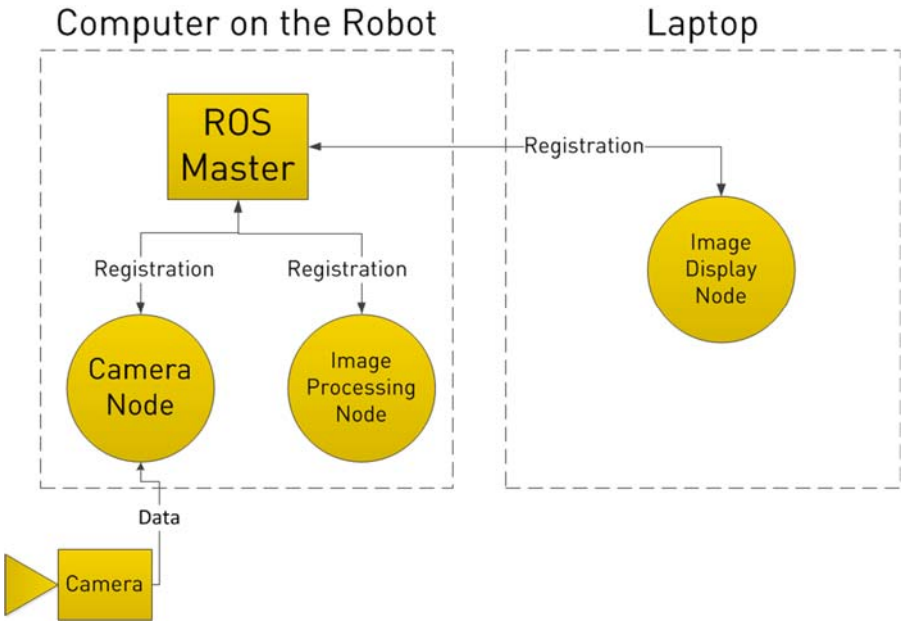
1. Topic-based Communication: ใช้รูปแบบ Publish-Subscribe โดยมีโหนดที่ทำหน้าที่ Publish ข้อมูลออกไป และมีโหนดที่ทำหน้าที่ Subscribe เพื่อรับข้อมูล

2. Service-based Communication: เป็นการสื่อสารแบบ Request-Response ใช้สำหรับการส่งคำสั่งที่ต้องการผลลัพธ์ในทันที



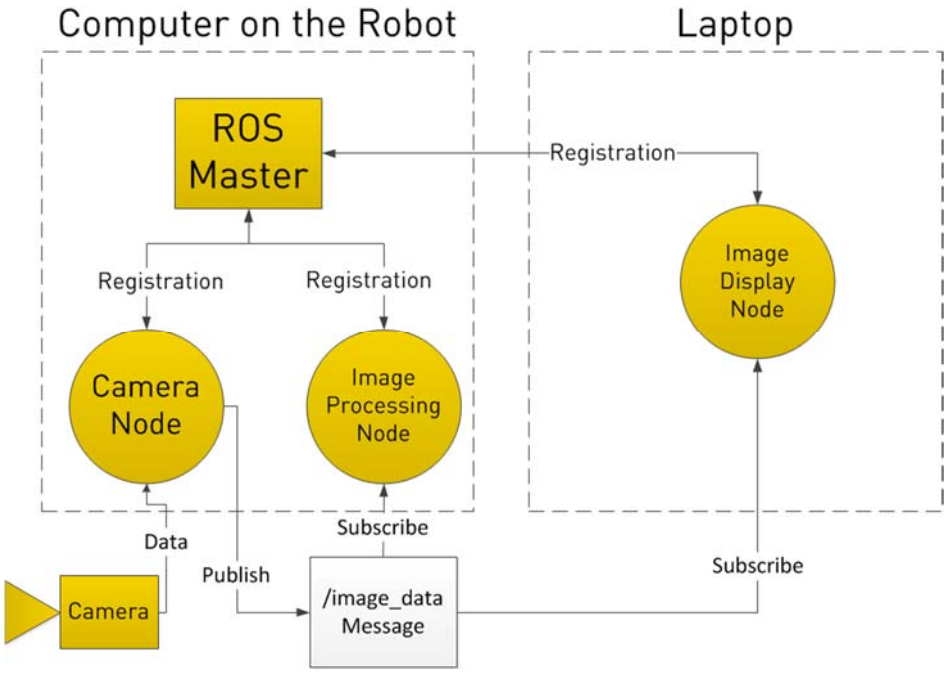
รูปที่ 2.2 ROS system

จากรูปที่ 2.2 อธิบายการทำงานของ ROS เริ่มต้นด้วย ROS Master โดยที่ Master อนุญาตให้ซอฟต์แวร์ ROS อื่นๆ ทั้งหมด (โหนด) ค้นหาและสื่อสารกันได้ ด้วยวิธีนี้ทำให้ ROS ไม่จำเป็นต้องระบุอย่างชัดเจน เช่น ส่งข้อมูลเซนเซอร์นี้ไปยังคอมพิวเตอร์เครื่องที่ IP: 127.0.0.1 โดย ROS สามารถบอกโหนด 1 ให้ส่งข้อความไปยังโหนด 2 ได้อย่างง่ายดายโดยการเผยแพร่และสมัครรับหัวข้อ สมมติว่ามีกล้องบนหุ่นยนต์ต้องการดูภาพจากกล้องได้ทั้งบนหุ่นยนต์เองและบนแล็ปท็อปเครื่องอื่น



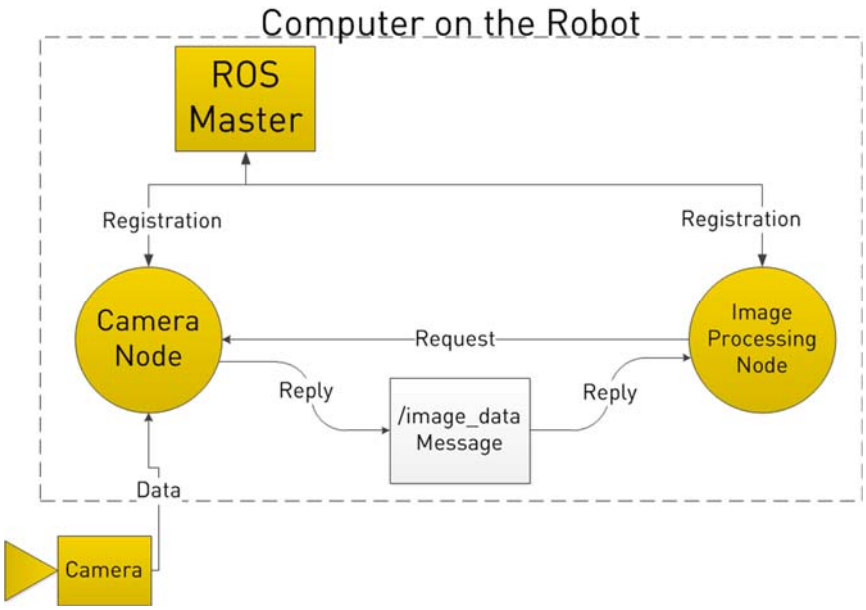
รูปที่ 2.3 ตัวอย่างแสดงการทำงานของ ROS

ในตัวอย่างจากรูปที่ 2.3 แสดงการทำงานของ ROS โดยมีโหนดกล้องที่ดูแลการสื่อสารกับกล้อง โหนดประมวลผลภาพบนหุ่นยนต์ที่ประมวลผลข้อมูลภาพ และโหนดแสดงภาพที่แสดงภาพบนหน้าจอ ในการเริ่มต้น โหนดทั้งหมดได้ลงทะเบียนกับมาสเตอร์แล้ว ให้นึกถึงมาสเตอร์เป็นตารางค้นหาที่โหนดทั้งหมดจะไปค้นหาว่าจะส่งข้อความไปที่ไหนโดยเฉพาะ



รูปที่ 2.4 แสดงการรับ/ส่งข้อมูล

จากรูปที่ 2.4 ในการลงทะเบียนกับ ROS Master โหนดกล้องจะระบุว่าจะเผยแพร่หัวข้อที่ชื่อว่า /image_data โหนดอื่นทั้งสองจะลงทะเบียนว่าตนเองสมัครรับหัวข้อ /image_data แล้ว ดังนั้น เมื่อโหนดกล้องได้รับข้อมูลบางอย่างจากกล้องแล้ว โหนดจะส่งข้อความ /image_data ไปยังโหนดอื่นทั้งสองโดยตรง



รูปที่ 2.5 โหนดประมวลผลภาพร้องขอข้อมูลจากโหนดกล้อง

จากรูปที่ 2.5 จากตัวอย่างถ้าหากต้องการให้โหนดประมวลผลภาพร้องขอข้อมูลจากโหนดกล้องในเวลาที่กำหนด ในกรณีนี้ จะใช้ ROS Services ใช้สำหรับการสื่อสารแบบ Request-Response ซึ่งช่วยให้โหนดสามารถร้องขอข้อมูลจากโหนดอื่นเมื่อจำเป็น โดยไม่ต้องรับส่งข้อมูลตลอดเวลา เหมาะสำหรับการดึงข้อมูลเฉพาะช่วงเวลาที่ต้องการ เช่น การร้องขอภาพจากกล้องเพื่อนำไปประมวลผล

ในโครงงานนี้มีการใช้แพ็กเกจจาก ROS ที่ช่วยในการพัฒนาและจำลองหุ่นยนต์ เช่น URDF/Xacro MoveIt, ROS Controller, Rviz และ Gazebo โดยมีหน้าที่ดังนี้

1. ไฟล์ URDF (Unified Robot Description Format) เป็นไฟล์ที่ใช้ในการกำหนดโครงสร้างของหุ่นยนต์ในรูปแบบ XML โดยประกอบด้วย โหนดสำหรับกำหนดลิงก์ (Link) และข้อต่อ (Joint) ข้อมูลของวัสดุ รูปร่าง พิกัด และข้อมูลเกี่ยวกับไดนามิกส์ของหุ่นยนต์

2. Xacro (XML Macros) เป็นส่วนขยายของ URDF ที่ช่วยลดความซ้ำซ้อนของโค้ด โดยใช้แมโคร (Macros) ในการกำหนดค่า ทำให้สามารถกำหนดโครงสร้างหุ่นยนต์ได้อย่างมีประสิทธิภาพ

3. MoveIt เป็นแพลตฟอร์มที่ใช้สำหรับวางแผนและควบคุมการเคลื่อนที่ของหุ่นยนต์ โดยทำงานร่วมกับ ROS และ Rviz โดย MoveIt มีความสำคัญอย่างมากในโครงงานนี้ เนื่องจากช่วยในการคำนวณ Inverse Kinematics (IK) เพื่อหามุมของข้อต่อให้แขนหุ่นยนต์เคลื่อนที่ไปยังตำแหน่งที่ต้องการ ใช้วางแผนเส้นทางการเคลื่อนที่ (Path Planning) โดยใช้ Trajectory Generation และควบคุมการเคลื่อนที่ของหุ่นยนต์ให้เป็นไปตามที่กำหนด

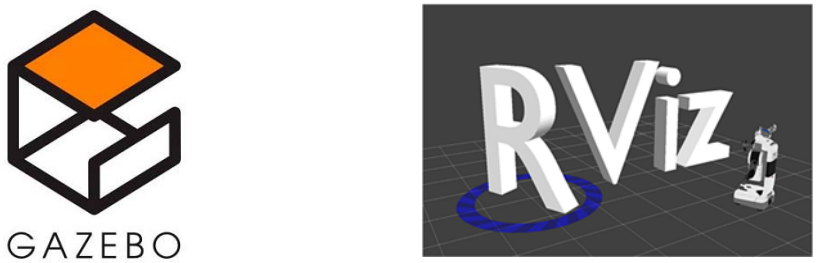
4. ROS Controller เป็นส่วนหนึ่งของ ROS Control ซึ่งเป็นแพ็กเกจใน ROS ที่ช่วยจัดการและควบคุมการเคลื่อนที่ของหุ่นยนต์ผ่าน controller plugins ต่างๆ โดย ROS Controller ทำหน้าที่เป็นตัวกลางระหว่าง planner (เช่น MoveIt หรือ RViz) และ ฮาร์ดแวร์ของหุ่นยนต์ เช่น มอเตอร์

5. Rviz ใช้แสดงผลข้อมูลหุ่นยนต์ เช่น ข้อมูลเซ็นเซอร์และการเคลื่อนที่

6. Gazebo ใช้จำลองหุ่นยนต์ในสภาพแวดล้อมเสมือนจริง

2.5 การจำลองหุ่นยนต์ (Robotic Simulation)

การจำลองหุ่นยนต์ (Robotic Simulation) เป็นขั้นตอนสำคัญในการทดสอบและพัฒนาระบบควบคุมหุ่นยนต์ โดยไม่จำเป็นต้องใช้ฮาร์ดแวร์จริง ซึ่งช่วยลดความเสี่ยงในการเกิดความเสียหายต่ออุปกรณ์และประหยัดค่าใช้จ่าย ซอฟต์แวร์จำลองหุ่นยนต์ที่ใช้กันอย่างแพร่หลาย ได้แก่ Gazebo และ Rviz ซึ่งทำงานบน Robot Operating System (ROS) [2]



ก) *ข)*

รูปที่ 2.6 ก) Gazebo ข) Ros Rviz

การจำลองหุ่นยนต์ในคอมพิวเตอร์ช่วยให้ผู้พัฒนาสามารถทดสอบความสามารถในการเคลื่อนที่ของหุ่นยนต์ในสภาพแวดล้อมเสมือนจริง เช่น การหลบหลีกสิ่งกีดขวาง การเคลื่อนที่ในพื้นที่แคบ หรือการทำงานร่วมกับหุ่นยนต์ตัวอื่นๆ นอกจากนี้ยังสามารถทดสอบการควบคุมแบบเรียลไทม์และวิเคราะห์ประสิทธิภาพของระบบก่อนนำไปใช้งานจริง

ในการจำลองหุ่นยนต์จะต้องทำการสร้างแบบจำลองในรูปแบบของกราฟฟิคและให้ข้อมูลต่างๆ เพื่อให้เกิดการจำลองเสมือนจริงได้โดยโครงงานนี้ใช้ ROS เป็นแพลตฟอร์มหลักในการจำลองหุ่นยนต์ เพื่อให้สามารถทดสอบและแก้ไขข้อผิดพลาดของซอฟต์แวร์ควบคุมได้อย่างมีประสิทธิภาพก่อนนำไปใช้กับหุ่นยนต์จริง ทำให้มั่นใจได้ว่าหุ่นยนต์สามารถทำงานได้ตามที่ออกแบบไว้ โครงงานนี้ใช้ Rviz และ Gazebo ในการจำลองหุ่นยนต์ ซึ่งแต่ละตัวมีหน้าที่ต่างกันคือ

1. Rviz ใช้แสดงผลข้อมูลและกำหนดเส้นทางการเคลื่อนที่

2. Gazebo ใช้จำลองฟิสิกส์และแสดงผลการเคลื่อนที่ของหุ่นยนต์ตามคำสั่งจาก Rviz

2.6 Rviz (ROS Visualization)

Rviz (ROS Visualization) เป็นเครื่องมือสำหรับแสดงผลข้อมูลเชิงภาพ ที่เกี่ยวข้องกับหุ่นยนต์ภายในระบบ ROS โดยมีหน้าที่หลักคือ แสดงโครงสร้างของหุ่นยนต์ จากไฟล์ URDF/Xacro แสดงผลข้อมูลเซนเซอร์ เช่น LiDAR, กล้อง, IMU และข้อมูลการตรวจจับวัตถุ แสดงเส้นทางการเคลื่อนที่ (Trajectory Visualization) ที่คำนวณโดย MoveIt หรือระบบควบคุม แสดงข้อมูล TF (Transform Frames) สำหรับติดตามการเคลื่อนที่ของข้อต่อและการเชื่อมต่อของเฟรมพิกัด แสดงข้อมูล Topic และ Message ต่างๆ ที่รับ-ส่งภายในระบบ ROS และหน้าที่ของ Rviz เหมือนอินเทอร์เฟซที่ให้ผู้ใช้สามารถสั่งงานและตรวจสอบสถานะของหุ่นยนต์แบบเรียลไทม์ แต่ไม่รองรับการจำลองฟิสิกส์ เช่น แรงโน้มถ่วงหรือการชนของวัตถุ

ข้อจำกัดของ Rviz คือไม่สามารถจำลองฟิสิกส์ ได้ เช่น แรงโน้มถ่วง แรงเสียดทาน หรือการชนของวัตถุ ใช้สำหรับการแสดงผลเท่านั้น ไม่มีการจำลองหรือคำนวณไดนามิกส์ของหุ่นยนต์

2.7 Gazebo

Gazebo (Physics Simulation) เป็นเครื่องมือสำหรับ การจำลองสภาพแวดล้อมทางฟิสิกส์แบบ 3 มิติ (Physics-Based Simulation) ที่ใช้ร่วมกับ ROS โดยมีหน้าที่หลักคือ จำลองไดนามิกส์ของหุ่นยนต์ เช่น แรงโน้มถ่วง, โมเมนตัม, แรงกระทำจากภายนอก จำลองสภาพแวดล้อมที่ซับซ้อน เช่น พื้นผิวที่ไม่เรียบ, สิ่งกีดขวาง, และวัตถุที่มีปฏิสัมพันธ์กับหุ่นยนต์ ทดสอบเซนเซอร์เสมือน (Simulated Sensors) เช่น กล้อง RGB-D, LiDAR, IMU และ GPS จำลองการควบคุมของหุ่นยนต์ โดยใช้ ROS สำหรับส่งคำสั่งไปยังตัวจำลอง Gazebo ทำหน้าที่เหมือนเครื่องจำลองฟิสิกส์ที่คำนวณและแสดงผลลัพธ์จากคำสั่งที่ส่งมาจาก Rviz ซึ่งช่วยให้สามารถทดสอบการเคลื่อนที่ของหุ่นยนต์ในโลกเสมือนจริงได้อย่างแม่นยำ

ข้อจำกัดของ Gazebo คือใช้ทรัพยากรคอมพิวเตอร์สูง เนื่องจากต้องคำนวณฟิสิกส์แบบเรียลไทม์ อาจมีความแตกต่างจากการทำงานจริง เนื่องจากโมเดลฟิสิกส์อาจไม่สามารถเลียนแบบโลกจริงได้อย่างสมบูรณ์

2.8 เปรียบเทียบความแตกต่างระหว่าง Rviz และ Gazebo

จากการที่ Rviz ทำหน้าที่เหมือนอินเทอร์เฟซที่ให้ผู้ใช้สามารถสั่งงานและตรวจสอบสถานะของหุ่นยนต์แบบเรียลไทม์ แต่ไม่รองรับการจำลองฟิสิกส์ เช่น แรงโน้มถ่วงหรือการชนของวัตถุ และ Gazebo ทำหน้าที่เหมือนเครื่องจำลองฟิสิกส์ที่คำนวณและแสดงผลลัพธ์จากคำสั่งที่ส่งมาจาก Rviz ซึ่งช่วยให้สามารถทดสอบการเคลื่อนที่ของหุ่นยนต์ในโลกเสมือนจริงได้อย่างแม่นยำ ทำให้สามารถสร้างตารางเปรียบเทียบได้ดังนี้

คุณสมบัติ	**Rviz (Graphic Input)**	**Gazebo (Graphic Output)**
หน้าที่หลัก	แสดงข้อมูลของหุ่นยนต์ และใช้กำหนดเส้นทาง	จำลองฟิสิกส์และแสดงผลการเคลื่อนที่จริง
การประมวลผลฟิสิกส์	ไม่รองรับ	รองรับ
การควบคุมการเคลื่อนที่	ใช้กำหนดเส้นทาง (Trajectory Planning)	ใช้ทดสอบและแสดงผลลัพธ์ของการเคลื่อนที่

ตารางที่ 2.1 ตารางเปรียบเทียบ Rviz กับ Gazebo

ภาพรวมการทำงานของ Rviz และ Gazebo ดังนี้

1. ผู้ใช้กำหนดเส้นทางการเคลื่อนที่ของหุ่นยนต์ใน Rviz
2. Rviz ส่งข้อมูลไปยัง ROS เพื่อควบคุมหุ่นยนต์เสมือน
3. Gazebo จำลองฟิสิกส์และแสดงผลลัพธ์ของการเคลื่อนที่จริงตามคำสั่งจาก Rviz
4. ผลลัพธ์ที่ได้สามารถนำไปวิเคราะห์และปรับปรุงการควบคุมได้ก่อนนำไปใช้งานกับหุ่นยนต์จริง

ดังนั้น ในโครงงานนี้ Rviz ทำหน้าที่เป็นเครื่องมือแสดงผลและกำหนดเส้นทางการเคลื่อนที่ของหุ่นยนต์ ในขณะที่ Gazebo เป็นเครื่องมือจำลองสภาพแวดล้อมและพฤติกรรมของหุ่นยนต์ตามกฎฟิสิกส์ ด้วยการใช้ Rviz และ Gazebo ควบคู่กัน นักพัฒนาสามารถตรวจสอบการทำงานของหุ่นยนต์ได้ทั้งในแง่ของเส้นทางและการตอบสนองตามฟิสิกส์จริง ทำให้สามารถปรับปรุงประสิทธิภาพของระบบควบคุมได้อย่างมีประสิทธิภาพ

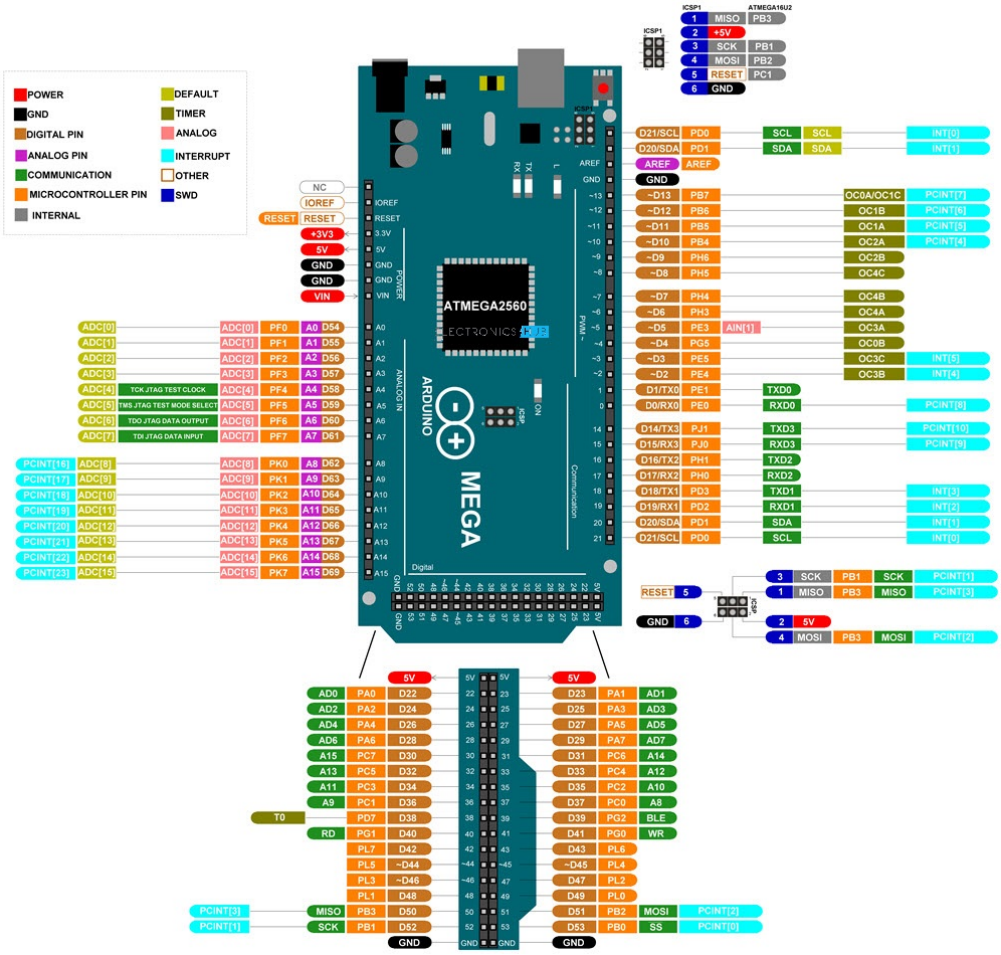
2.9 Arduino MEGA 2560

Arduino MEGA 2560 เป็นบอร์ดไมโครคอนโทรลเลอร์ ATmega328P ชนิดหนึ่ง โดยใช้การควบคุมหุ่นยนต์ 6DOF ที่สร้างขึ้นจริง ได้ใช้บอร์ด Arduino Mega 2560 ในการควบคุมร่วมกับวงจรขับเคลื่อนต่างๆ ดูเพิ่มเติมจากโครงงาน การออกแบบชุดจำลองแขนหุ่นยนต์อุตสาหกรรม (Design of an Industrial Robot Arm) [4] ซึ่งรวมถึงสิ่งที่จำเป็นทั้งหมดบนบอร์ด โดยไมโครคอนโทรลเลอร์เชื่อมต่อกับพีซีด้วยสาย USB บอร์ด Arduino MEGA 2560 มีข้อมูลจำเพาะดังตารางต่อไปนี้

ตารางที่ 2.2 ข้อมูลจำเพาะของบอร์ดบอร์ดอาร์ดูโน เมก้า 2560 (Arduino MEGA 2560)

ชิปไอซีไมโครคอนโทรเลอร์	ATmega2560
ใช้แรงดันไฟฟ้า	5V
รองรับการจ่ายแรงดันไฟฟ้า (ที่แนะนำ)	7 – 12V
รองรับการจ่ายแรงดันไฟฟ้า (ที่จำกัด)	6 – 20V
พอร์ต Digital I/O	54 พอร์ต (มี 15 พอร์ต PWM output)
พอร์ต Analog Input	16 พอร์ต
กระแสไฟที่จ่ายได้ในแต่ละพอร์ต	40mA
กระแสไฟที่จ่ายได้ในพอร์ต 3.3V	50mA
พื้นที่โปรแกรมภายใน	256KB พื้นที่โปรแกรม, 8KB ใช้โดย Bootloader
พื้นที่แรม	2KB
พื้นที่หน่วยความจำถาวร (EEPROM)	4KB
ความถี่คริสตัล	16MHz
น้ำหนัก	25 กรัม

โดยโครงงานนี้การเชื่อมต่อ Robot Operating System (ROS) กับ Arduino Mega 2560 เป็นกระบวนการสำคัญที่ช่วยให้สามารถควบคุมฮาร์ดแวร์ของหุ่นยนต์ผ่าน ROS ได้ โดยใช้ rosserial ซึ่งเป็นแพ็กเกจที่ออกแบบมาเพื่อเชื่อมต่อ ROS กับไมโครคอนโทรลเลอร์ เช่น Arduino วิธีการติดตั้งและเชื่อมต่อสามารถดูเพิ่มเติมได้ที่ภาคผนวก ก.3



รูปที่ 2.7 พอร์ตของ Arduino MEGA 2560 [3]

บทที่ 3 ขั้นตอนการดำเนินงาน

ในขั้นตอนการพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6 องศาอิสระ (Development of control software and simulation model a 6DOF robotic arm) ได้แบ่งขั้นตอนการดำเนินงานเป็น 2 ส่วนดังต่อไปนี้

1. แบบจำลองการทำงานของ หุ่นยนต์แขนกล 6 องศาอิสระ
2. โปรแกรมควบคุมตัวหุ่นยนต์แขนกล 6 องศาอิสระ

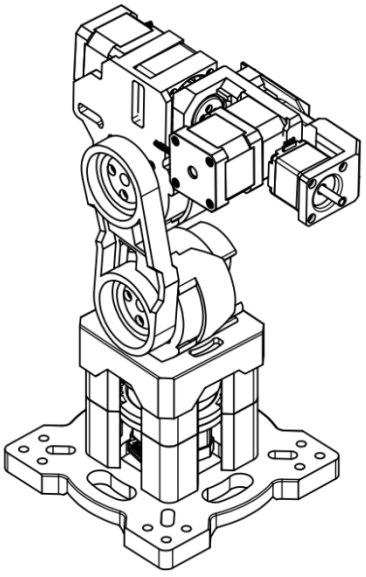
โดยซอฟต์แวร์ที่ใช้ในการพัฒนาโครงงาน การพัฒนาโปรแกรมควบคุมและจำลองหุ่นยนต์แขนกล 6 องศาอิสระ มีดังนี้

1. Arduino IDE โดยใช้เขียนโปรแกรมให้ Arduino Mega2560 รับค่าจาก ROS Topics และควบคุมการทำงานของ Stepper motor
2. ROS จะทำการแสดงผลข้อมูลหุ่นยนต์ และ ตรวจสอบการทำงานของ ROS Topics
3. Gazebo ใช้สำหรับการจำลองฟิสิกส์ของหุ่นยนต์ และทดสอบการทำงานในสภาพแวดล้อมเสมือนจริง
4. SolidWorks ใช้ในการสร้างกราฟฟิกโมเดลหุ่นยนต์แขนกล 6 องศาอิสระ

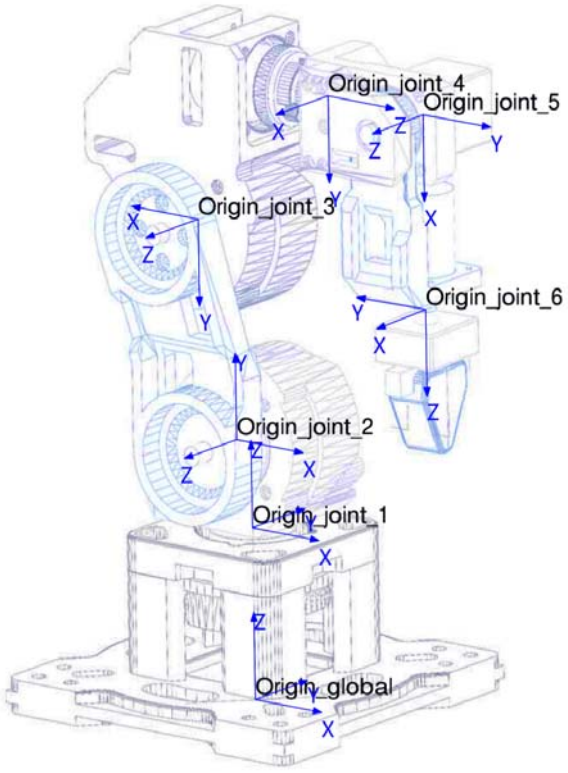
โดยแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ สร้างโมเดลหุ่นยนต์บน SolidWorks แล้วใช้ ROS ในการสร้างโปรแกรมต่างๆ และใช้ Gazebo เป็นตัวแสดงผลการจำลองแขนกล 6 องศาอิสระ

3.1 แบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ

แบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ จากรูปภาพที่ 3.1 เป็นโมเดลหุ่นยนต์แขนกลจากโครงงาน การออกแบบชุดจำลองแขนหุ่นยนต์อุตสาหกรรม [4] และรูปที่ 3.2 เป็นภาพแสดงชื่อและตำแหน่งจุดหมุน (Joint) ขอโมเดลหุ่นยนต์ที่สร้างขึ้น โดยใช้โปรแกรม SolidWorks

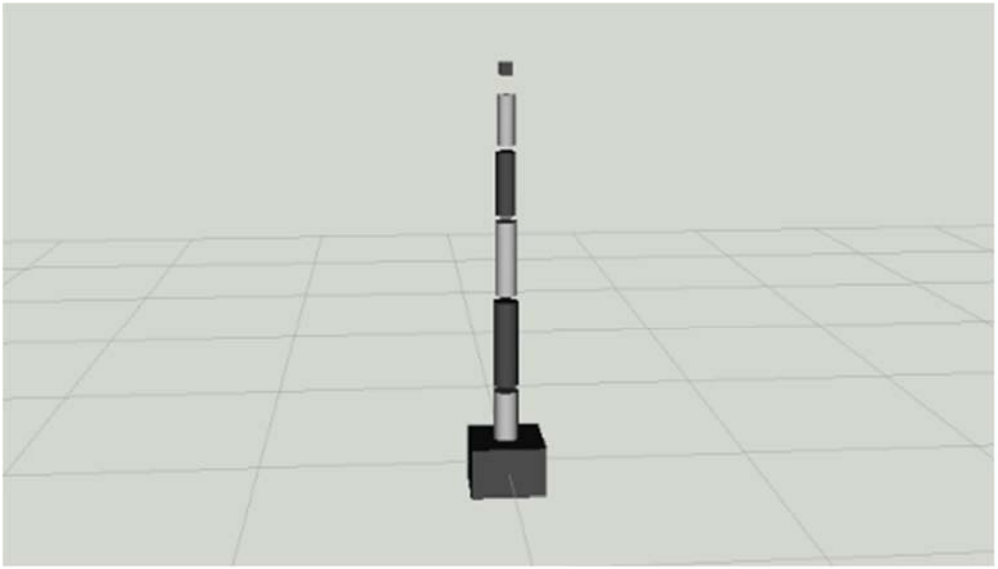


รูปที่ 3.1 โมเดลหุ่นยนต์แขนกลจากโครงงานการออกแบบชุดจำลองแขนหุ่นยนต์ อุตสาหกรรมในปี 2565 [4]



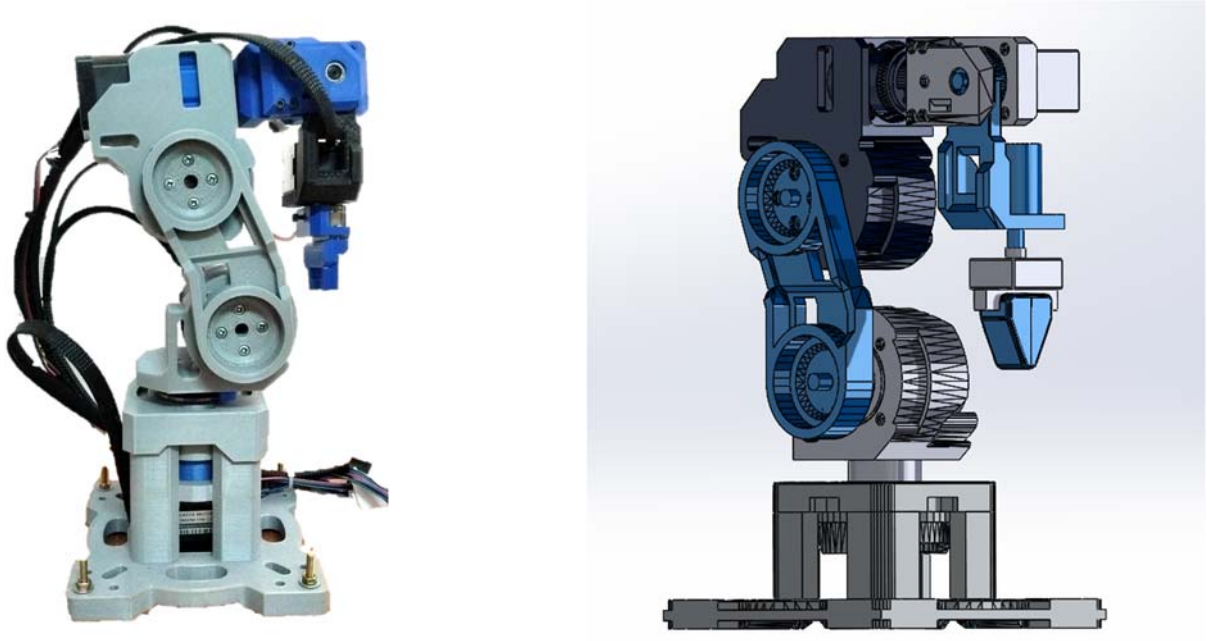
รูปที่ 3.2 ภาพแสดงชื่อและตำแหน่งจุดหมุน (Joint) ของแบบจำลองในโครงงานนี้

จากการวิเคราะห์โครงสร้างของแขนหุ่นยนต์แขนกล 6 องศาอิสระและนำมาทดลองทำโมเดลต้นแบบโดยใช้ Rviz จากรูปที่ 3.3 ซึ่งเป็นผลงานในรายวิชาเตรียมโครงงานในภาคการศึกษาต้น จะเห็นได้ว่าโมเดลต้นแบบหุ่นยนต์แขนกล 6 องศาอิสระ มีรูปทรงเป็นแท่งทรงกระบอกต่อกันเท่านั้น และในส่วนของข้อต่อต่างๆมีพื้นที่ว่างระหว่างกัน



รูปที่ 3.3 โมเดลต้นแบบหุ่นยนต์แขนกล 6 องศาอิสระ

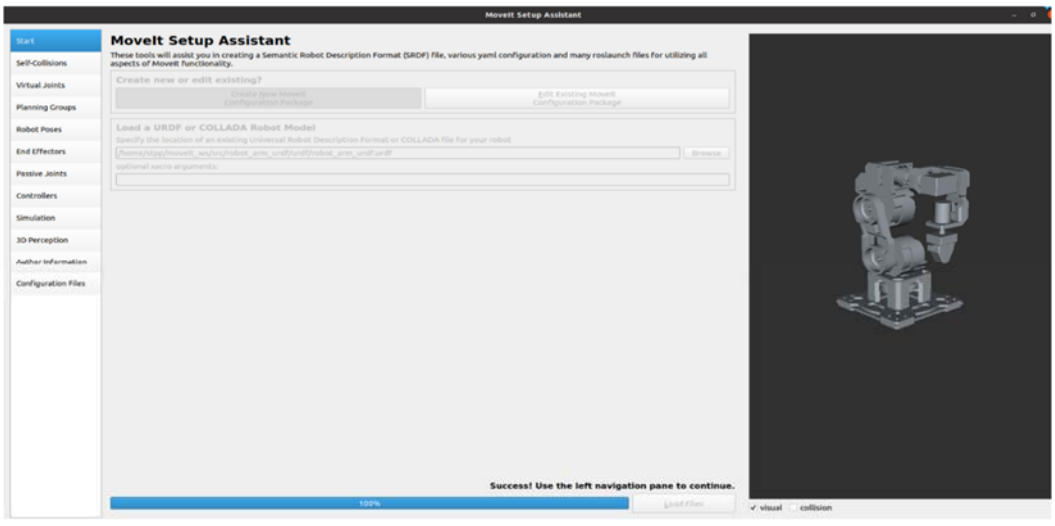
ในโครงงานนี้ได้ปรับปรุงจากโมเดลตัวต้นแบบโดยมีการพัฒนาโมเดลแบบใหม่ให้มีความใกล้เคียงกับของจริง โดยใช้ SolidWorks ในการสร้างโมเดลหุ่นยนต์แขนกล 6 องศาอิสระ เพื่อนำไปใช้ทำแบบจำลองดังรูปที่ 3.4 ข)



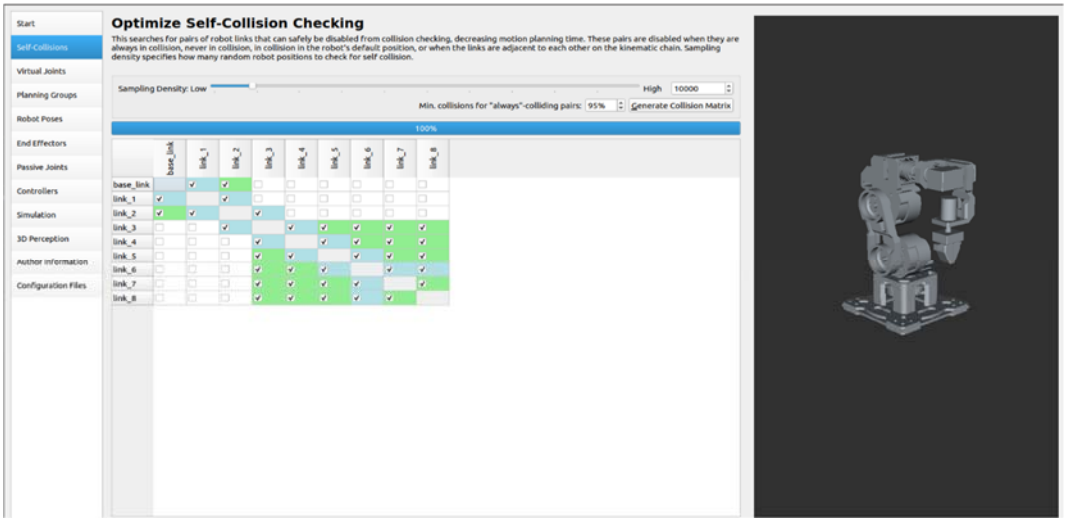
ก) *ข)*

รูปที่ 3.4 ก) หุ่นยนต์แขนกล 6 องศาอิสระ ข) โมเดลหุ่นยนต์แขนกล 6 องศาอิสระ

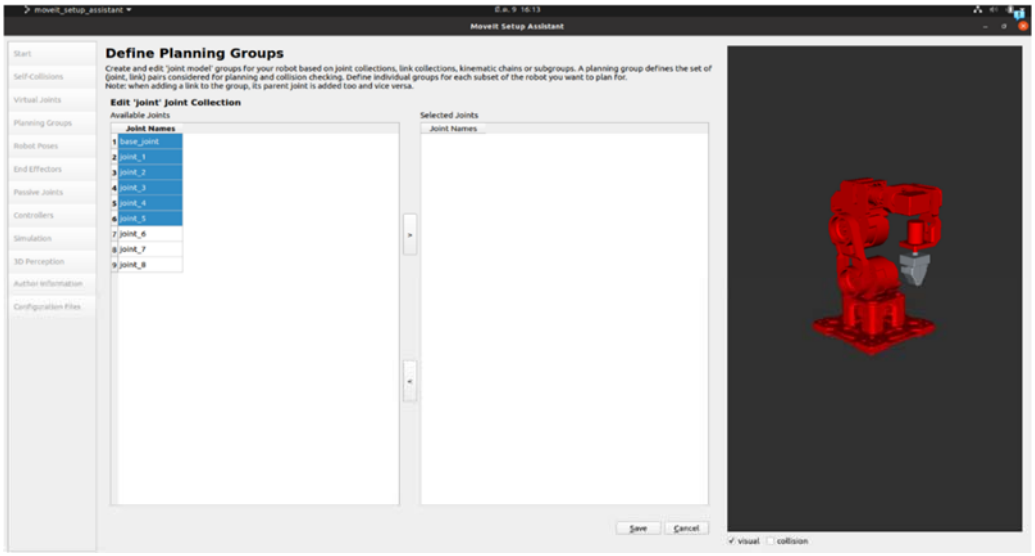
จากรูปที่ 3.4 ข) โมเดลมีรายละเอียดเพิ่มขึ้นมา เมื่อได้โมเดลมาแล้วทำการกำหนดตัวแปรต่างๆ ให้กับแบบจำลอง โดยค่าที่กำหนดคือ แรงบิดแต่ละข้อต่อ น้ำหนักของแต่ละชิ้นส่วน ความเร็วความเร่งของการหมุนข้อต่อ องศาที่เคลื่อนที่ได้ของแต่ละข้อต่อ เมื่อกำหนดค่าแล้วนั้น ทำการใช้ Moveit โดยทำงานร่วมกับ ROS และ Rviz ซึ่งเป็นฟังก์ชันเสริมใน ROS โดยฟังก์ชัน Moveit มีหน้าที่ในการวางแผนเส้นทางการเคลื่อนที่ (Path Planning) โดยใช้ Trajectory Generation และควบคุมการเคลื่อนที่ของหุ่นยนต์ให้เป็นไปตามที่กำหนดกำหนดตัวแปรต่างๆของโมเดลเพื่อให้สามารถเคลื่อนที่ได้ จากรูปที่ 3.5 ถึง 3.9 เป็นการกำหนดค่าต่างๆ



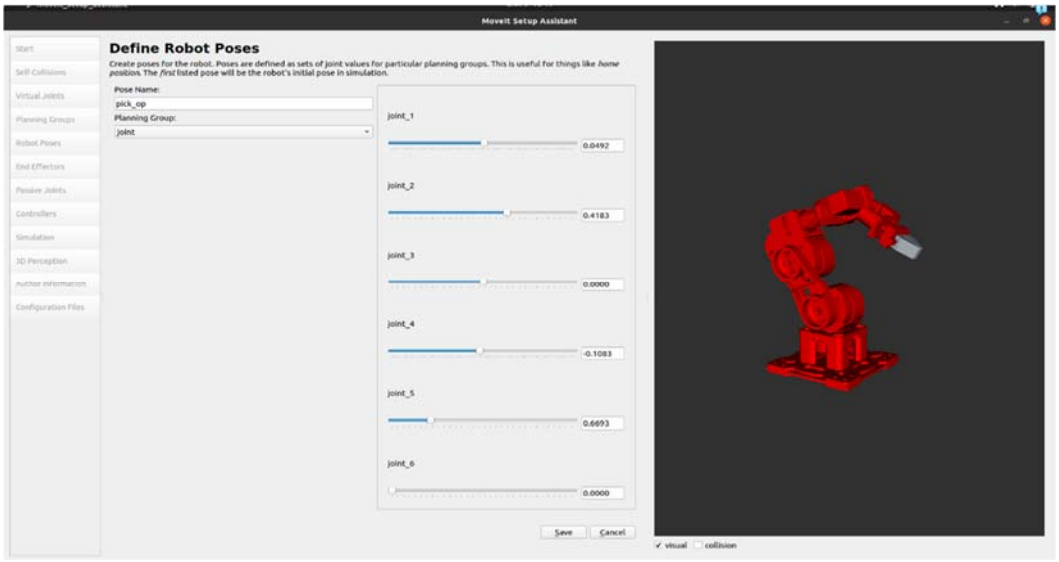
รูปที่ 3.5 หน้าต่างโปรแกรม Moveit โดยมีโมเดลที่สร้างเพื่อตั้งค่าใช้กับ ROS



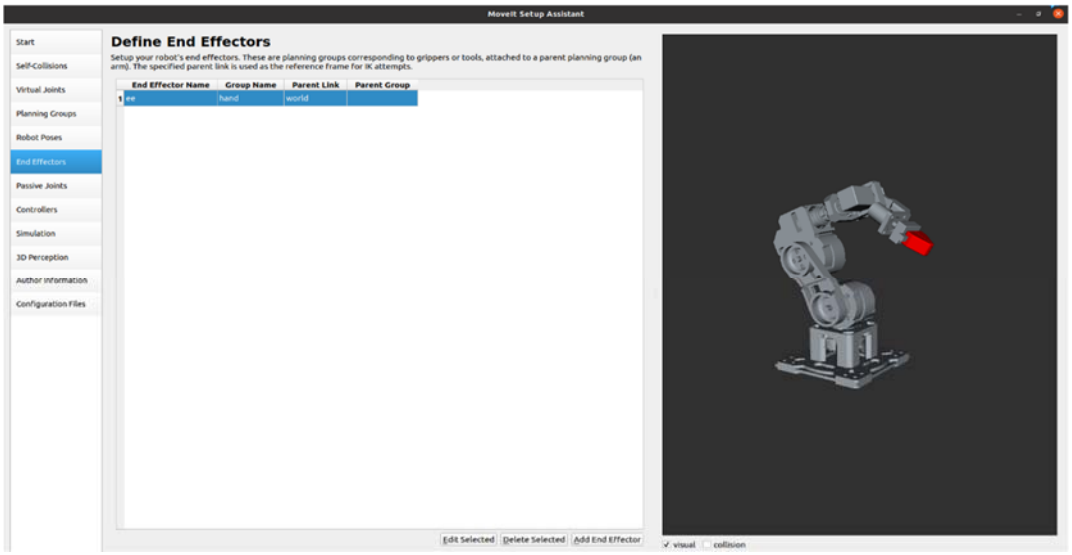
รูปที่ 3.6 กำหนดการตรวจสอบการชนกันระหว่างส่วนต่าง ๆ ของแขนกล



รูปที่ 3.7 กำหนดกลุ่มของข้อต่อสำหรับโมเดลหุ่นยนต์

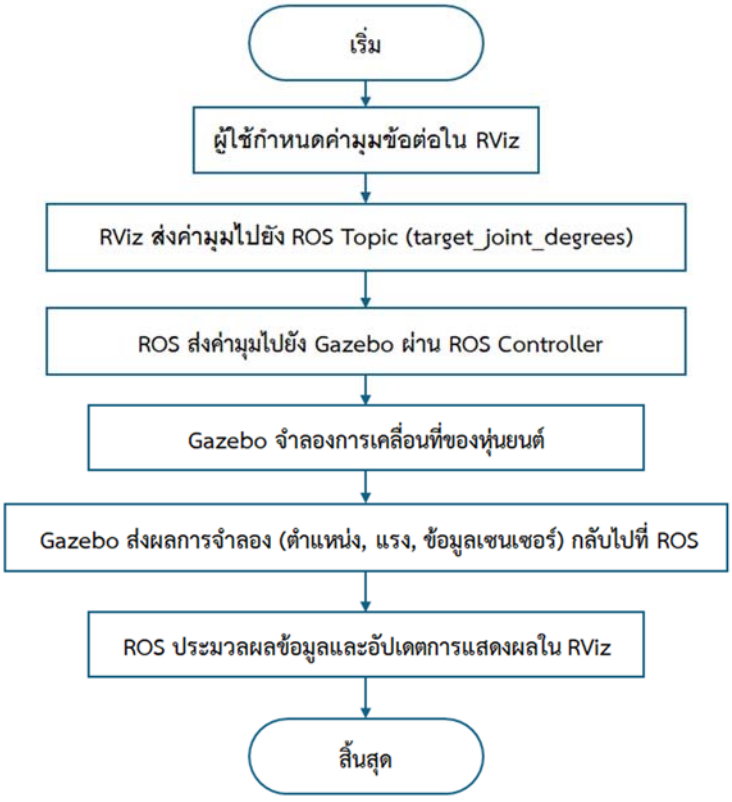


รูปที่ 3.8 กำหนดตำแหน่งโมเดลหุ่นยนต์



รูปที่ 3.9 กำหนดส่วนปลายของแขนหุ่นยนต์

เมื่อกำหนดค่าทุกอย่างแล้ว จะได้แบบจำลองที่สมบูรณ์สามารถนำไปแสดงผลใน Gazebo ได้แล้ว และวิธีการสร้างแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ สามารถดูเพิ่มเติมได้ที่ภาคผนวก ข และ [6] โดย RViz ใช้กำหนดค่า input (มุมข้อต่อ) และแสดงผลข้อมูล แล้ว ROS เป็นตัวกลางที่ส่งข้อมูลระหว่าง RViz และ Gazebo จากนั้น Gazebo จำลองการเคลื่อนที่ตามฟิสิกส์จริงและส่งผลกลับไปยัง ROS ดังรูปที่ 3.10 Flowchart แสดงการทำงานระหว่าง ROS และ Gazebo



รูปที่ 3.10 Flowchart แสดงการทำงานระหว่าง ROS และ Gazebo

1. เริ่มต้นระบบ (Start System)

- เปิดใช้งาน ROS Master ซึ่งเป็นศูนย์กลางที่ช่วยจัดการการสื่อสารระหว่าง Node ต่าง ๆ
- เปิดใช้งาน Gazebo เพื่อจำลองสภาพแวดล้อมของหุ่นยนต์

2. โหลดไฟล์ URDF/Xacro (Load URDF/Xacro)

- นำเข้าไฟล์ URDF/Xacro ซึ่งเป็นโครงสร้างของหุ่นยนต์ 6DOF
- ROS ใช้ robot_state_publisher เพื่อเผยแพร่โครงสร้างข้อต่อและลิงก์ของหุ่นยนต์

3. สร้างการจำลองใน Gazebo (Spawn Robot in Gazebo)

- ROS ใช้ Gazebo ROS Plugin เพื่อสร้างหุ่นยนต์จำลอง
- หุ่นยนต์จะถูกโหลดเข้าสู่สภาพแวดล้อม 3D ของ Gazebo

4. รับคำสั่งควบคุมจาก ROS (Receive Commands from ROS)

- ROS Node ที่เป็น Controller จะส่งคำสั่งไปยัง Gazebo ผ่าน ROS Topic หรือ ROS Service
- ใช้ Joint Position Controller หรือ Effort Controller เพื่อควบคุมการเคลื่อนที่ของข้อต่อ

5. คำนวณฟิสิกส์และแสดงผล (Physics Simulation & Visualization)

- Gazebo ใช้ Physics Engine (ODE, Bullet, DART, Simbody)
- จำลองแรงโน้มถ่วง, โมเมนตัม, แรงเสียดทาน และการชนกันของวัตถุ
- แสดงผลการเคลื่อนที่ของหุ่นยนต์ตามคำสั่งจาก ROS

6. ส่งข้อมูลสถานะกลับไปที่ ROS (Publish Sensor Data to ROS)

- Gazebo ส่งข้อมูลตำแหน่งข้อต่อ (Joint States)
- ข้อมูลถูกส่งผ่าน ROS Topic เพื่อให้ ROS Node อื่นนำไปประมวลผล

7. แสดงผลใน Rviz (Visualize in Rviz)

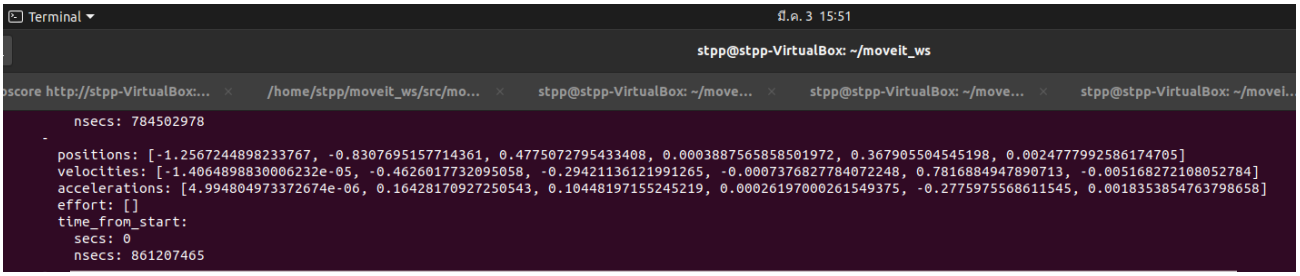
- ข้อมูลจาก Gazebo ถูกส่งไปแสดงใน Rviz เพื่อช่วยให้ผู้พัฒนาตรวจสอบสถานะของหุ่นยนต์
- สามารถดูโครงสร้างข้อต่อ, เส้นทางการเคลื่อนที่

9. สิ้นสุดการจำลอง (End Simulation)

- เมื่อการทดสอบเสร็จสิ้น สามารถบันทึกข้อมูลการจำลองเพื่อใช้วิเคราะห์ต่อไป

3.2 โปรแกรมควบคุมหุ่นยนต์แขนกล 6 องศาอิสระ

วิธีควบคุมหุ่นยนต์แขนกล 6 องศาอิสระ ให้ทำการเคลื่อนที่โดยการดึงค่าที่ได้จาก Rviz ที่มีค่าเป็น position:[θ_1(เรเดียน), θ_2(เรเดียน), θ_3(เรเดียน), θ_4(เรเดียน), θ_5(เรเดียน), θ_6(เรเดียน)] โดยมุมแต่ละมุมหมายถึงมุมใน joint 1 joint 2 joint 3 joint 4 joint 5 และ joint 6 ตามลำดับ มุมใน Rviz จะเป็นมุมในหน่วยเรเดียน จากนั้นจะทำการแปลงเป็นชุดข้อมูล 6 ค่า คือ Published target joint angle (int degrees) :[θ_1(องศา), θ_2(องศา), θ_3(องศา), θ_4(องศา), θ_5(องศา), θ_6(องศา)] โดยมุมแต่ละมุมหมายถึงมุมในหน่วยองศาของ joint 1 joint 2 joint 3 joint 4 joint 5 และ joint 6 ตามลำดับ โดยใช้ Python ในการดึงค่าและแปลงข้อมูลที่ได้ไป publish topic ชื่อ target_joint_degrees แล้วนำ Arduino Mega 2560 ไป subscribe topic ชื่อ target_joint_degrees แล้วนำข้อมูลไปขับเคลื่อนมอเตอร์ดังรูปที่ 3.11 ถึง รูปที่ 3.13



รูปที่ 3.11 หน้าต่าง Terminal แสดงข้อมูลของแบบจำลอง

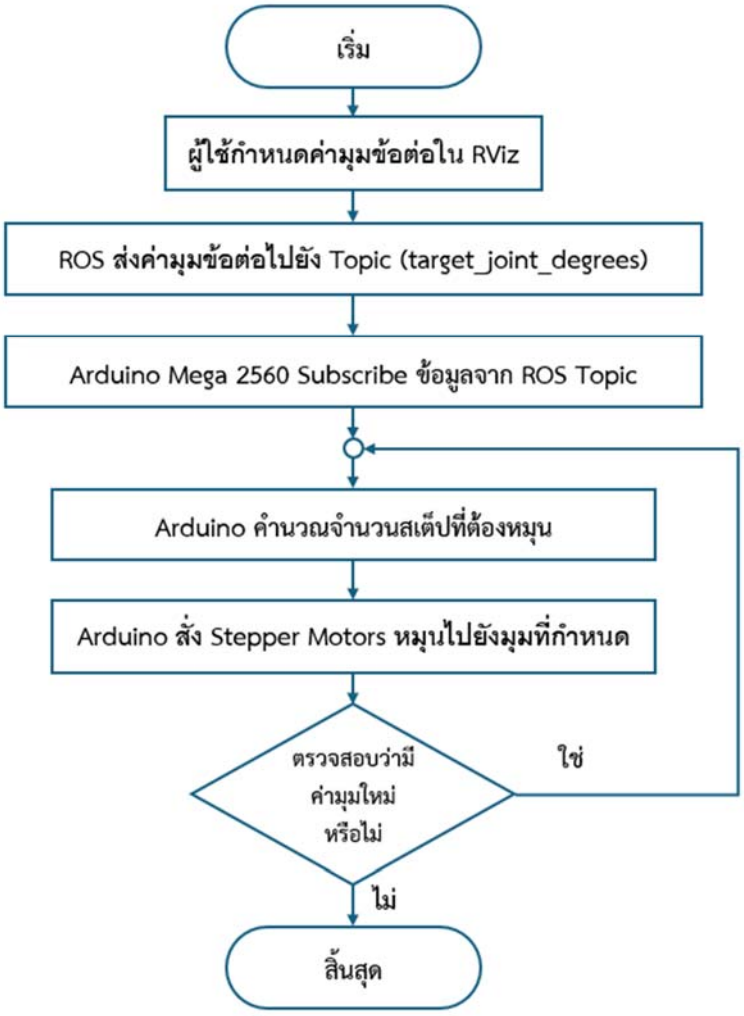
```
stpp@stpp-VirtualBox:~/moveit_ws$ rosrun robot_arm_urdf moveit_joint_publisher.py
[INFO] [1740991570.797046]: Published target joint angles (int degrees): [0, 0, 0, 0, 0, 0]
[INFO] [1740991574.727013]: Published target joint angles (int degrees): [0, 0, 90, 0, 90, 0]
[INFO] [1740991602.126034]: Published target joint angles (int degrees): [72, -60, 19, 0, 42, 0]
[INFO] [1740991649.383635]: Published target joint angles (int degrees): [72, -35, 35, 0, 0, 0]
[INFO] [1740991672.129313]: Published target joint angles (int degrees): [-72, -35, 35, 0, 0, 0]
[INFO] [1740991799.431509]: Published target joint angles (int degrees): [-72, -60, 19, 0, 42, 0]
[INFO] [1740991975.590068]: Published target joint angles (int degrees): [0, 0, 0, 0, 0, 0]
```

รูปที่ 3.12 หน้าต่าง Terminal แสดงข้อมูลองศาเป็น Array

```
stpp@stpp-VirtualBox:~/moveit_ws$ rostopic echo /target_joint_degrees
WARNING: no messages received and simulated time is active.
Is /clock being published?
layout:
  dim: []
  data_offset: 0
data: [72, -35, 35, 0, 0, 0]
---
layout:
  dim: []
  data_offset: 0
data: [-72, -35, 35, 0, 0, 0]
---
layout:
  dim: []
  data_offset: 0
data: [-72, -60, 19, 0, 42, 0]
---
```

รูปที่ 3.13 หน้าต่าง Terminal แสดงข้อมูลจาก Topic target_joint_degrees

จากรูปที่ 3.11 แสดงข้อมูลที่ Rviz ส่งออกมาในรูปแบบ position: [-1.25672, -0.83076, 0.47750, 0.00038, 0.37690, 0.00247] ซึ่งเป็นค่ามุมของแต่ละ joint ในหน่วยเรเดียน ตามลำดับ เมื่อได้รับค่ามุมในหน่วยเรเดียนแล้ว จะทำการแปลงค่าเป็นหน่วยองศาโดยใช้ Python และจัดเก็บในรูปแบบ Array ที่มี 6 ค่า ตามรูปที่ 3.12 โดยค่าที่ได้คือ:Published target joint angle (int degrees): [-72, -60, 19, 0, 42, 0] แต่ละค่าจะสอดคล้องกับมุมของ joint ตามลำดับ โดยค่าบวกหมายถึงการหมุนตามเข็มนาฬิกา หลังจากนั้น ข้อมูลจะถูกส่งไปยัง topic ชื่อ target_joint_degrees ดังที่แสดงในรูปที่ 3.13 ซึ่งประกอบด้วยชุดข้อมูล 6 ค่า ได้แก่ [-72, -60, 19, 0, 42, 0] สุดท้าย Arduino Mega 2560 จะทำการ subscribe topic target_joint_degrees เพื่อนำข้อมูลที่ได้รับไปใช้ควบคุมมอเตอร์ให้ทำงานตามค่ามุมที่กำหนดโดยมีหลักการทำงานของโปรแกรมการทำงานของระบบควบคุมแขนกลผ่าน ROS และ Arduino ดังรูปที่ 3.14 Flowchart การทำงานระหว่าง ROS และ Arduino Mega 2560



รูปที่ 3.14 Flowchart การทำงานระหว่าง ROS และ Arduino Mega 2560

1. เริ่มต้นระบบ (Start)

    - ระบบเริ่มทำงาน โดยเปิดใช้งาน ROS (Robot Operating System) และ Arduino Mega 2560

2. ตรวจสอบการเชื่อมต่อของอุปกรณ์ (I/O Check)

    - ตรวจสอบการเชื่อมต่อของ Arduino กับ ROS ผ่าน Serial Communication (/dev/ttyUSB0)

    - ตรวจสอบการเชื่อมต่อของ มอเตอร์สเต็ป (Stepper Motor)

3. รับค่ามุมข้อต่อจาก ROS (Receive Joint Angles from ROS)

    - ROS Node ที่ทำหน้าที่เป็น Publisher ส่งค่ามุมข้อต่อผ่าน Topic: /target_joint_degrees

    - Arduino ทำหน้าที่เป็น Subscriber รับค่ามุมที่ต้องการให้แต่ละข้อต่อหมุน

4. ตรวจสอบค่ามุมที่ได้รับ (Validate Data)

    - ตรวจสอบว่าค่ามุมที่ได้รับมี ขนาด 6 ค่า (สำหรับข้อต่อของแขนกล) หากค่าไม่ถูกต้อง แสดง Error และรอข้อมูลใหม่

5. คำนวณจำนวนสเต็ปที่ต้องหมุน (Compute Steps)

    - คำนวณจำนวนสเต็ปที่ต้องหมุนโดยใช้สูตร:

$$\text{step} = (\text{target angle} - \text{current angle}) \times \text{Step per degree}$$

    - กำหนด ทิศทางของการหมุน โดยตั้งค่าขา DIR_PIN

6. ควบคุมมอเตอร์ให้เคลื่อนที่ (Move Motors)

    - ส่งสัญญาณไปยังขา STEP_PIN ของแต่ละมอเตอร์ ใช้ฟังก์ชัน delayMicroseconds() เพื่อควบคุมความเร็วของมอเตอร์ และอัปเดตค่ามุมปัจจุบันของแต่ละข้อต่อ

7. อัปเดตค่ามุมปัจจุบันและแสดงผล (Update Current Angles & Display)

    - แสดงค่ามุมปัจจุบันบน Serial Monitor แล้วทำให้ LED บน Arduino ติด เพื่อแสดงว่ามีการรับค่ามุมใหม่

8. ตรวจสอบว่ามีค่ามุมใหม่หรือไม่ (Check for New Data)

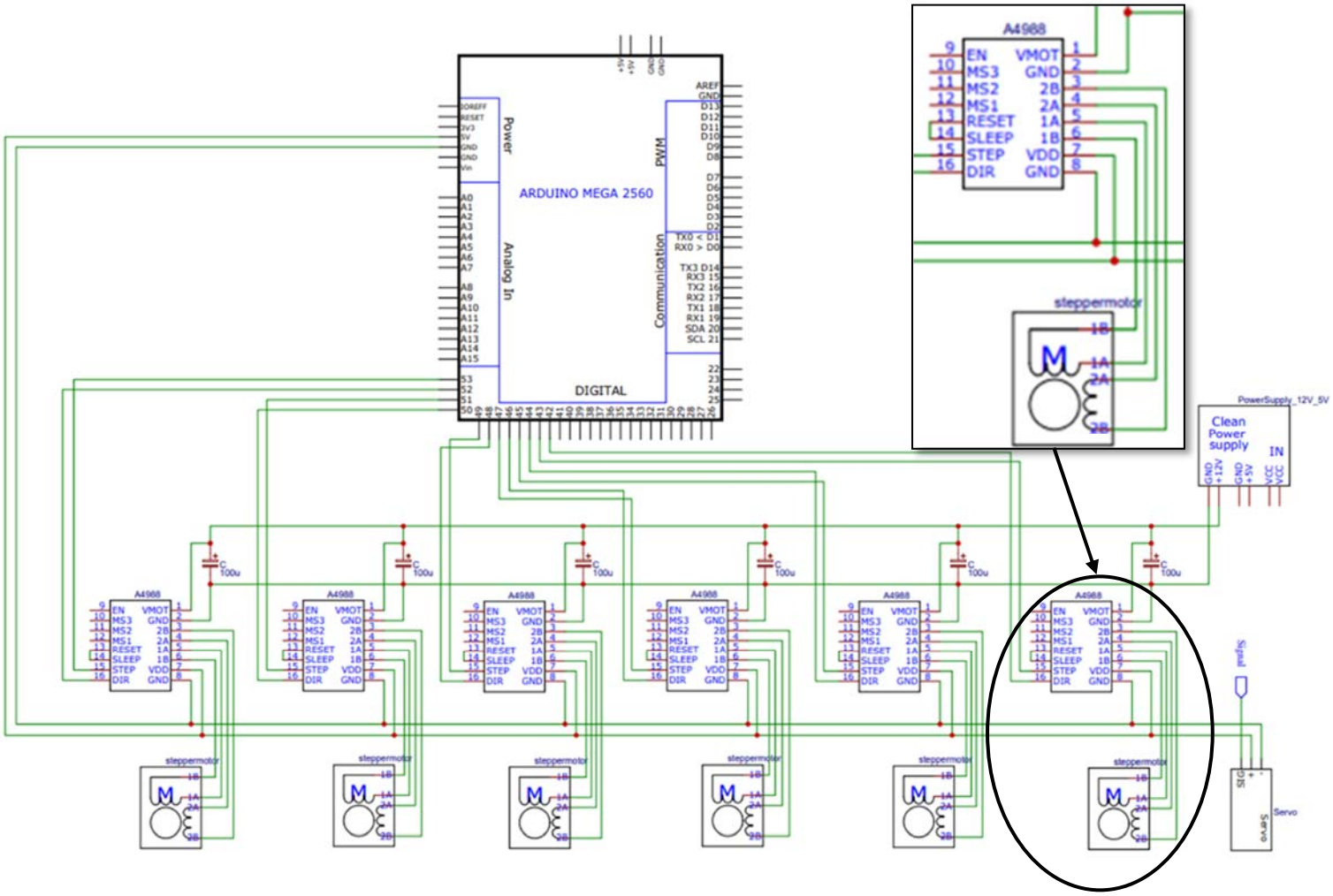
    - หากมีค่ามุมใหม่ ให้กลับไปทำขั้นตอนที่ 3 หากไม่มีข้อมูลใหม่ ให้รอค่ามุมใหม่จาก ROS

9. สิ้นสุดกระบวนการ (End)

    - ระบบจะทำงานต่อเนื่องจนกว่าจะได้รับคำสั่งหยุด หากมีข้อผิดพลาด เช่น การเชื่อมต่อขาดหาย หรือค่ามุมไม่ถูกต้อง ระบบจะแสดง Error

หมายเหตุ: Code ของโปรแกรมอยู่ในส่วนภาคผนวก ก

เนื่องจากโครงงานนี้เป็นการพัฒนาและปรับปรุงจากโครงงาน การออกแบบชุดจำลองแขนหุ่นยนต์อุตสาหกรรม [4] จึงมีการนำวงจรควบคุมมาใช้ในการควบคุมมอเตอร์โดยมีแผงวงจรดังรูปที่ 3.15



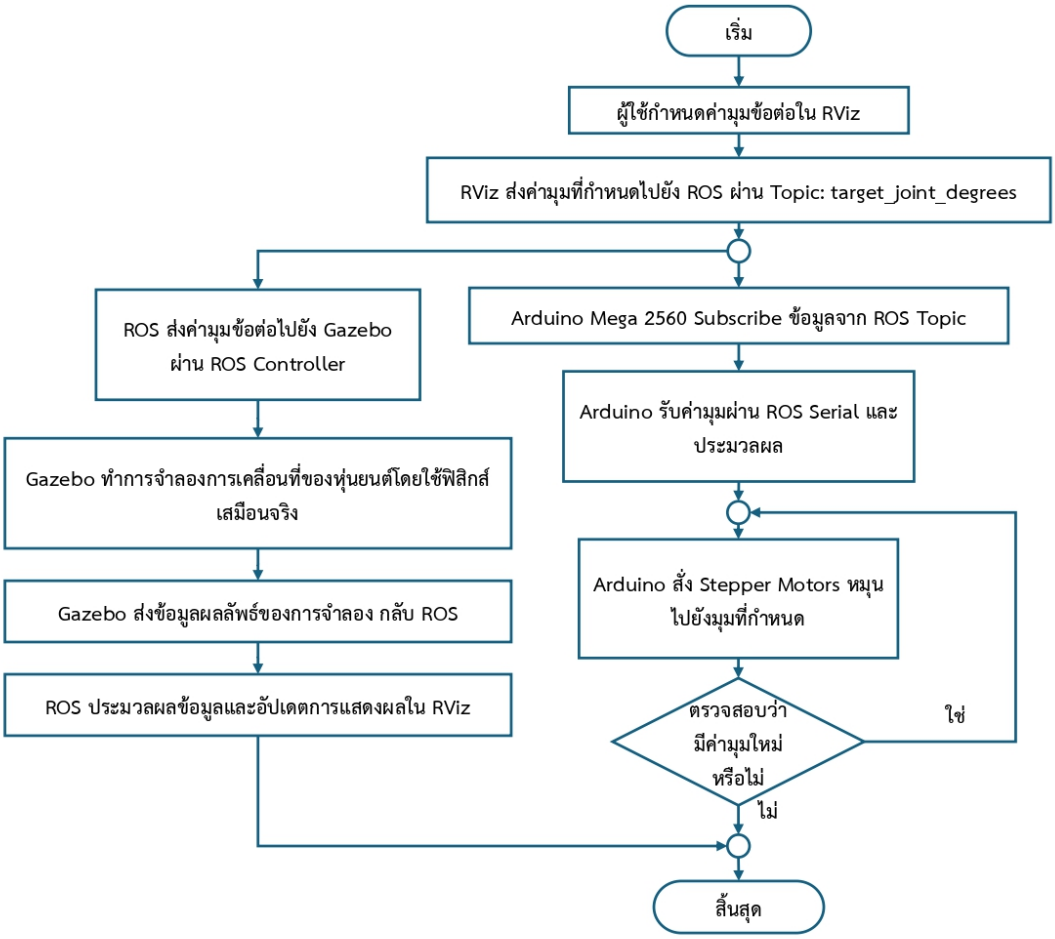
รูปที่ 3.15 ชุดวงจรควบคุมแขนหุ่นยนต์แขนกล 6 องศาอิสระ

โดยในโครงงานนี้มีการเชื่อมต่ออุปกรณ์ขา DIR และ STEP ดังนี้

ชื่อ	DIR_PIN	STEP_PIN
Stepper motor ตัวที่ 1	52	53
Stepper motor ตัวที่ 2	50	51
Stepper motor ตัวที่ 3	48	49
Stepper motor ตัวที่ 4	46	47
Stepper motor ตัวที่ 5	44	45
Stepper motor ตัวที่ 6	42	43

ตารางที่ 3.1 แสดงการเชื่อมต่อขา DIR และ STEP ของ Stepper motor

3.3 Flowchart การทำงานของโปรแกรม



รูปที่ 3.16 Flowchart การทำงานร่วมกันระหว่าง ROS, Gazebo และ Arduino Mega 2560

เมื่อสามารถสร้างแบบจำลองหุ่นยนต์แขนกลและโปรแกรมควบคุมหุ่นยนต์แขนกลสำเร็จแล้วนั้น จากรูปที่ 3.16 แสดง Flowchart การทำงานร่วมกันระหว่าง ROS, Gazebo และ Arduino Mega 2560 ดังนี้

1. กระบวนการทำงานของ ROS และ Gazebo

1.1 เริ่มต้นระบบ

- ผู้ใช้กำหนดค่ามุมข้อต่อของหุ่นยนต์ใน RViz
- RViz ส่งค่ามุมที่กำหนดไปยัง ROS ผ่าน Topic: target_joint_degrees

1.2 ส่งค่ามุมไปยัง Gazebo

- ROS ส่งค่ามุมข้อต่อไปยัง Gazebo ผ่าน ROS Controller
- Gazebo ทำการจำลองการเคลื่อนที่ของหุ่นยนต์โดยใช้ฟิสิกส์เสมือนจริง

1.3 รับผลจากการจำลอง

- Gazebo ส่งข้อมูลผลลัพธ์ของการจำลอง เช่น ตำแหน่ง, แรง, ข้อมูลจากเซนเซอร์ กลับไปยัง ROS

- ROS ประมวลผลข้อมูลและอัปเดตการแสดงผลใน RViz เพื่อให้ผู้ใช้เห็นการเคลื่อนไหวของหุ่นยนต์เสมือนจริง

2. กระบวนการทำงานของ ROS และ Arduino Mega 2560

2.1 รับค่ามุมข้อต่อจาก ROS

- ROS ส่งค่ามุมข้อต่อไปยัง Topic: target_joint_degrees
- Arduino Mega 2560 ทำหน้าที่เป็น Subscriber รับค่ามุมจาก ROS

2.2 คำนวณจำนวนสเต็ปที่ต้องหมุน

- Arduino คำนวณจำนวนสเต็ปที่ต้องหมุนของ Stepper Motor โดยใช้สูตร:

$$step = (target\ angle - current\ angle) \times Step\ per\ degree$$

- กำหนดทิศทางของมอเตอร์โดยตั้งค่าขา DIR_PIN ตามตารางที่ 3.1

2.3 ควบคุม Stepper Motor ให้เคลื่อนที่

- Arduino ส่งคำสั่งไปยัง Stepper Motors เพื่อให้หมุนไปยังมุมที่กำหนด
- ใช้ฟังก์ชัน delayMicroseconds() เพื่อควบคุมความเร็ว

2.4 อัปเดตสถานการณ์เคลื่อนที่

- Arduino อัปเดตค่ามุมปัจจุบัน
- แสดงข้อมูลมุมข้อต่อบน Serial Monitor

2.5 ตรวจสอบว่ามีค่ามุมใหม่หรือไม่

- หากมีค่ามุมใหม่ ระบบจะกลับไปเริ่มกระบวนการคำนวณใหม่
- หากไม่มีข้อมูลใหม่ ระบบจะรอค่าจาก ROS

สรุปได้ว่า Flowchart ที่รวมทั้ง ROS, Gazebo และ Arduino Mega 2560 แสดงให้เห็นว่ากระบวนการทำงานของระบบนี้เป็นแบบ Loop โดยเริ่มจากผู้ใช้กำหนดค่ามุมใน RViz ซึ่งข้อมูลจะถูกส่งไปที่ Gazebo เพื่อจำลองการเคลื่อนที่ และหากต้องการควบคุมหุ่นยนต์จริง ข้อมูลจะถูกส่งต่อไปยัง Arduino Mega 2560 เพื่อคำนวณและสั่งการ Stepper Motors ให้เคลื่อนที่ตามค่ามุมที่กำหนด

ระบบนี้ช่วยให้สามารถทดสอบการเคลื่อนที่ของหุ่นยนต์ได้ทั้งใน การจำลอง (Simulation) และฮาร์ดแวร์จริง (Hardware Control) ซึ่งเป็นแนวทางสำคัญในการพัฒนาหุ่นยนต์เพื่อให้มั่นใจว่าทำงานได้ถูกต้องตามที่ออกแบบไว้

บทที่ 4 การทดลองและผลการทดลอง

การทดลองนี้มีวัตถุประสงค์เพื่อทดสอบประสิทธิภาพการแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ โดยในบทนี้จะอธิบายถึงผลที่ได้จากการทดลอง ด้วยการป้อนองศาการเคลื่อนที่ของ จุดหมุนบนแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ บนโปรแกรม Rviz แล้วให้แสดงผลการควบคุมผ่านโปรแกรม Gazebo ซึ่งแบบเป็นการทดลอง 2 ชุด ดังนี้

4.1 การทดลองควบคุมการเคลื่อนที่แบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระผ่านโปรแกรม Rviz

4.2 การทดลองควบคุมการเคลื่อนที่หุ่นยนต์จริงแขนกล 6 องศาอิสระผ่านโปรแกรม Rviz

4.1 การทดลองควบคุมการเคลื่อนที่แบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระผ่านโปรแกรม Rviz

เป็นการทดลองควบคุมการเคลื่อนที่แบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระผ่าน Rviz โดยกำหนดองศาในแต่ละจุดหมุนโดยใช้ Moveit สร้างเป็นตำแหน่งที่ตั้งค่าไว้แล้วจากนั้นส่งการแสดงผลไปยัง Gazebo เพื่อแสดงผลการเคลื่อนที่ของหุ่นยนต์แขนกล 6 องศาอิสระ มีวิธีการทดลองคือ

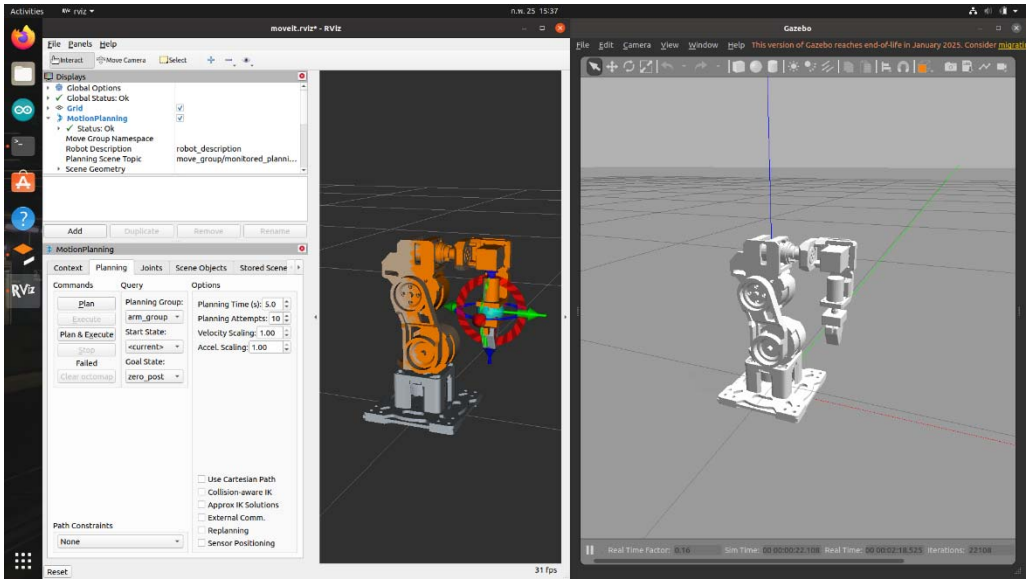
1. ทำการกำหนดตำแหน่งให้กับแบบจำลองผ่าน Moveit Setup Assistant
2. ทำการรันไฟล์ Rviz ของหุ่นยนต์
3. สร้างโลกเปล่าใน Gazebo และนำแบบจำลองของหุ่นยนต์เข้าไป
4. กำหนดตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายใน Rviz โดยแต่ละตำแหน่งเป็นค่าที่ตั้งไว้
5. สั่งให้แบบจำลองหุ่นยนต์เคลื่อนที่ใน Rviz
6. ดูการเปลี่ยนตำแหน่งของแบบจำลองหุ่นยนต์ใน Gazebo
7. ทำซ้ำข้อ 4 ถึงข้อ 6 แล้วบันทึกผล

ตารางที่ 4.1 ตารางแสดงตำแหน่งที่ตั้งไว้ใน Moveit Setup Assistant

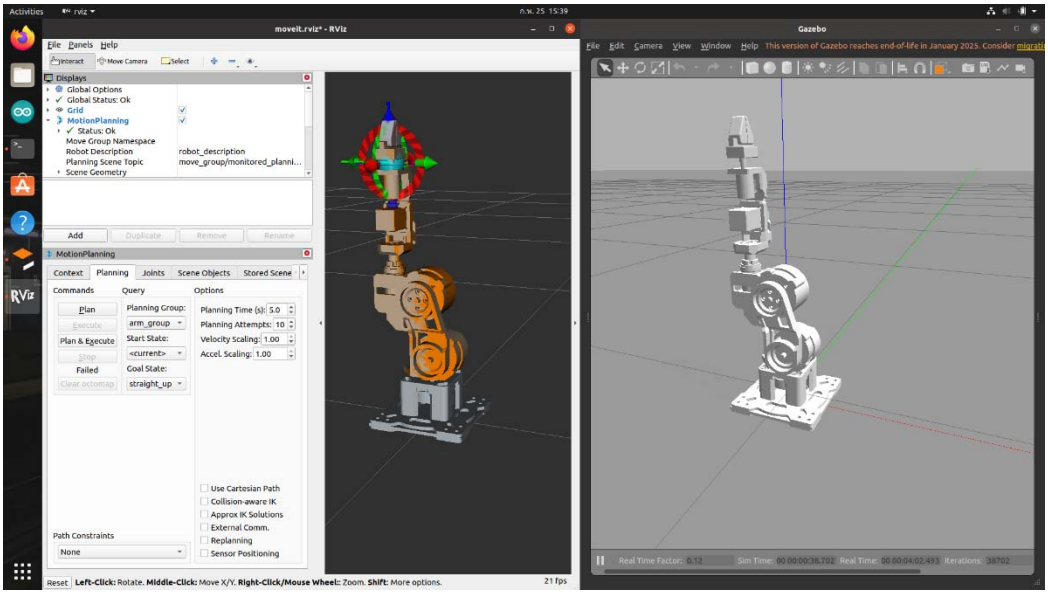
ชื่อตำแหน่ง	มุมที่กำหนด $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$	รูปที่
Home position	0,0,0,0,0,0	4.1
Straight up	0,0, -90, -90,0,0	4.2
Pick object	151,73, -39, -36,0,0	4.3
Lift object	151,51, -50, -2,0,0	4.4
Opposite position	0,51, -50, -2,0,0	4.5
Drop object	0,78, -52, -25,0,0	4.6

หมายเหตุ: มุม θ เป็นบวกคือหมุนตามเข็มนาฬิกา

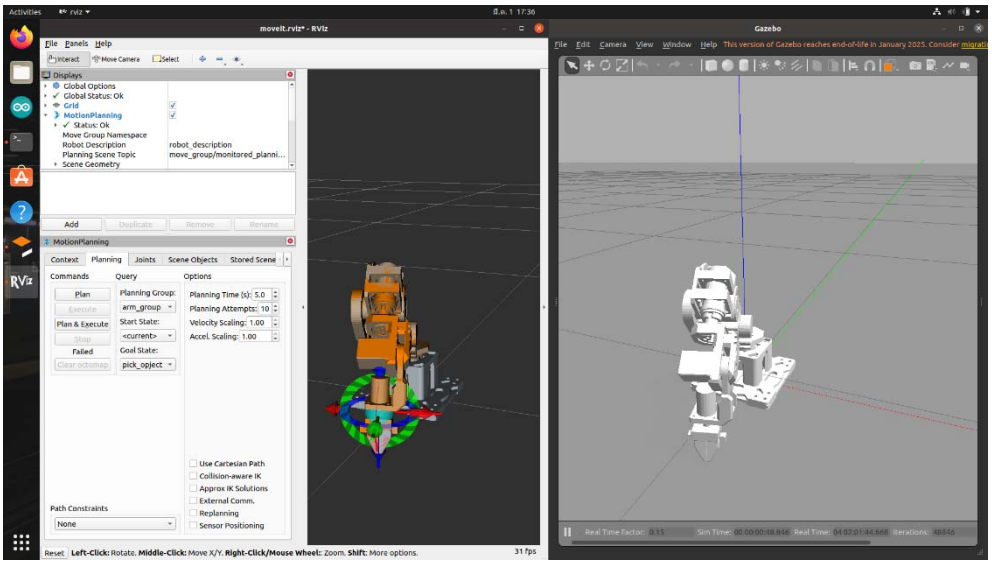
จากการผลการทดลองสรุปได้ว่าสามารถแสดงผลแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ ผ่านโปรแกรม Gazebo โดยมีโปรแกรมควบคุมคำสั่งคือโปรแกรม Rviz สามารถดูผลการทดลองได้จากรูปที่ 4.1 ถึงรูปที่ 4.6



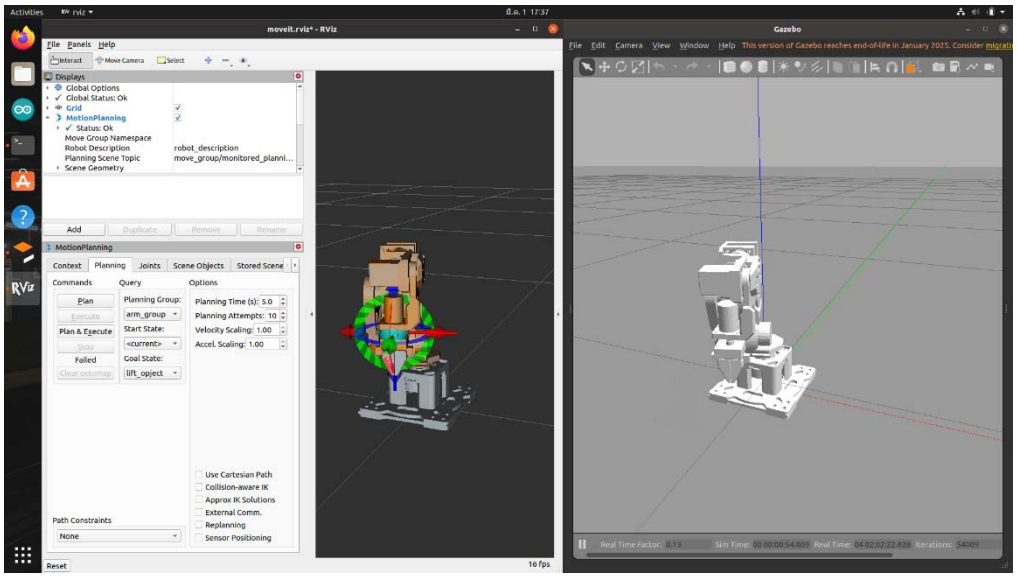
รูปที่ 4.1 แบบจำลองหุ่นยนต์ตำแหน่ง Home position



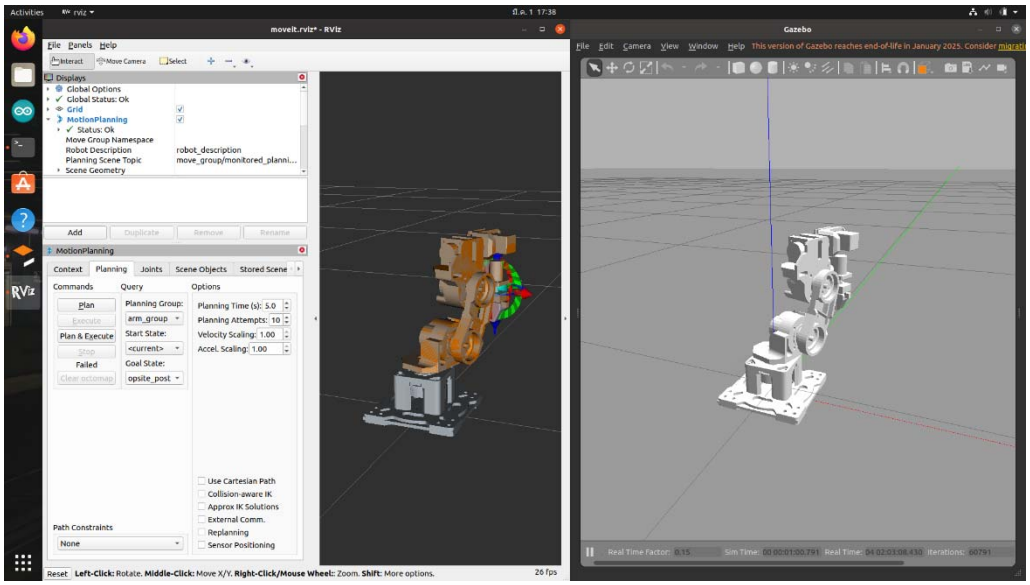
รูปที่ 4.2 แบบจำลองหุ่นยนต์ตำแหน่ง Straight up



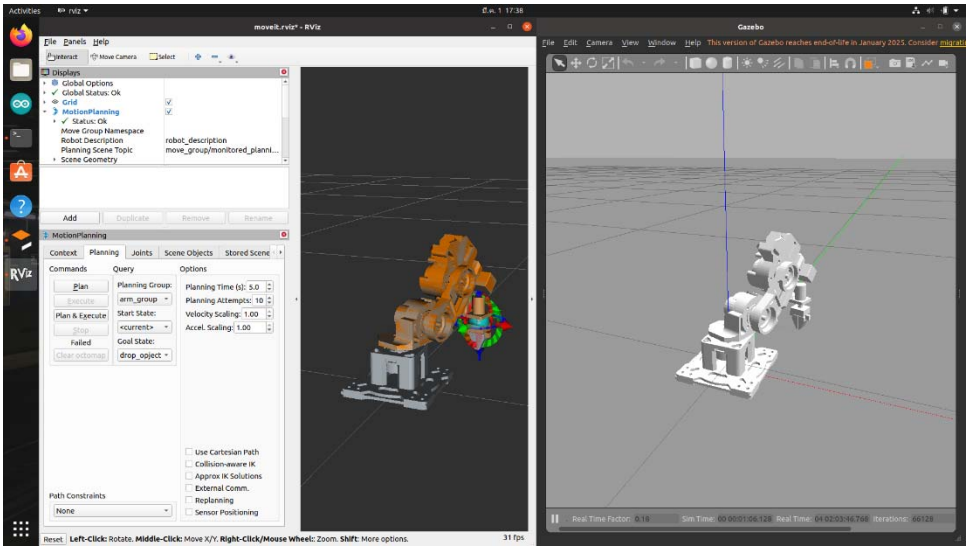
รูปที่ 4.3 แบบจำลองหุ่นยนต์ตำแหน่ง Pick object



รูปที่ 4.4 แบบจำลองหุ่นยนต์ตำแหน่ง Lift object



รูปที่ 4.5 แบบจำลองหุ่นยนต์ตำแหน่ง Opposite position



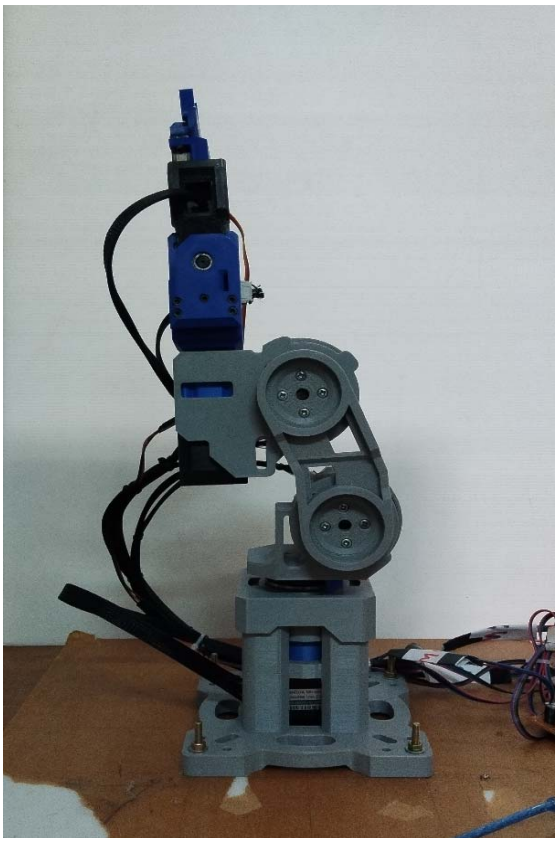
รูปที่ 4.6 แบบจำลองหุ่นยนต์ตำแหน่ง Drop object

4.2 การทดลองควบคุมการเคลื่อนที่หุ่นยนต์จริงแขนกล 6 องศาอิสระผ่านโปรแกรม Rviz

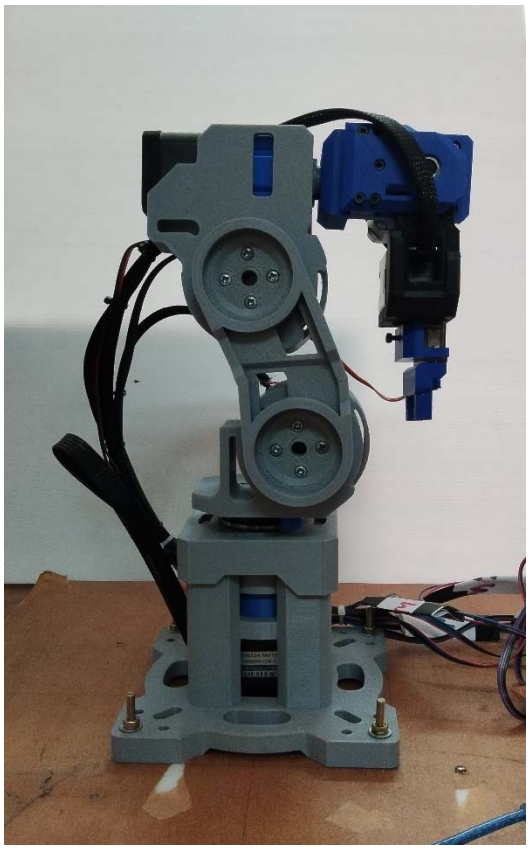
เป็นการทดลองควบคุมการเคลื่อนที่หุ่นยนต์แขนกล 6 องศาอิสระผ่าน Rviz โดยกำหนดองศาในแต่ละจุดหมุนและใช้ Moveit สร้างเป็นตำแหน่งที่ตั้งค่าไว้แล้วจากนั้นส่งค่าองศาไปยัง topic ชื่อ target_joint_degrees และจากนั้นนำข้อมูลที่ได้ให้ Arduino Mega 2560 ไป subscribe topic ชื่อ target_joint_degrees แล้วนำข้อมูลไปขับเครื่องมอเตอร์ มีวิธีการดังนี้

1. ทำการกำหนดตำแหน่งให้กับแบบจำลองผ่าน Moveit Setup Assistant
2. ทำการรันไฟล์ Rviz ของหุ่นยนต์
3. ทำการรันไฟล์ Python ที่รับค่าจาก Rviz แล้วทำการแปลงข้อมูลจากนั้นส่งข้อมูลไปที่ topic ชื่อ target_joint_degrees
4. ทำการให้ Arduino Mega 2560 รับค่าที่ topic ชื่อ target_joint_degrees
5. กำหนดตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายใน Rviz โดยแต่ละตำแหน่งเป็นค่าที่ตั้งไว้
6. สั่งให้แบบจำลองหุ่นยนต์เคลื่อนที่ใน Rviz
7. ดูการเปลี่ยนตำแหน่งของหุ่นยนต์แขนกล 6 องศาอิสระ
8. ทำซ้ำข้อ 5 ถึงข้อ 7 แล้วบันทึกผล

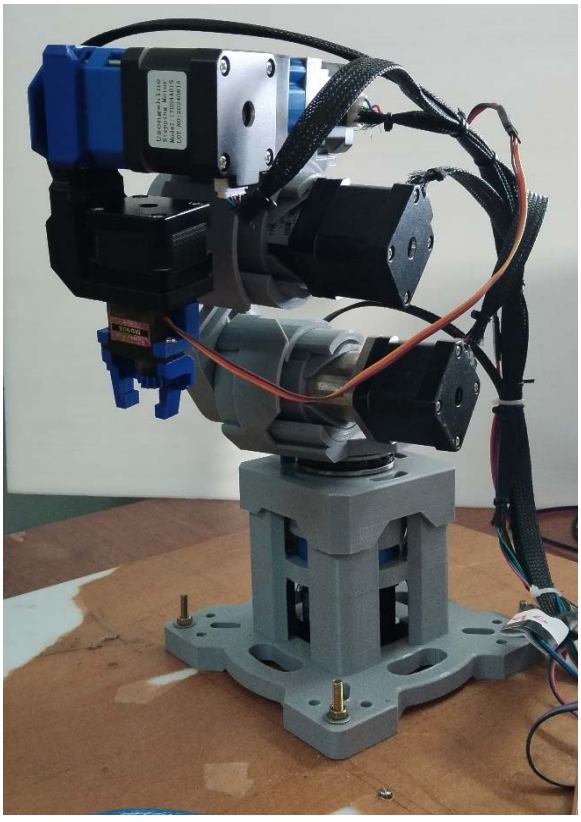
จากการผลการทดลองสรุปได้ว่าสามารถแสดงผลการเคลื่อนที่ของหุ่นยนต์แขนกล 6 องศาอิสระ โดยมีโปรแกรมควบคุมคำสั่งการเคลื่อนที่คือโปรแกรม Rviz สามารถดูผลการทดลองได้จากรูปที่ 4.7 ถึงรูปที่ 4.12



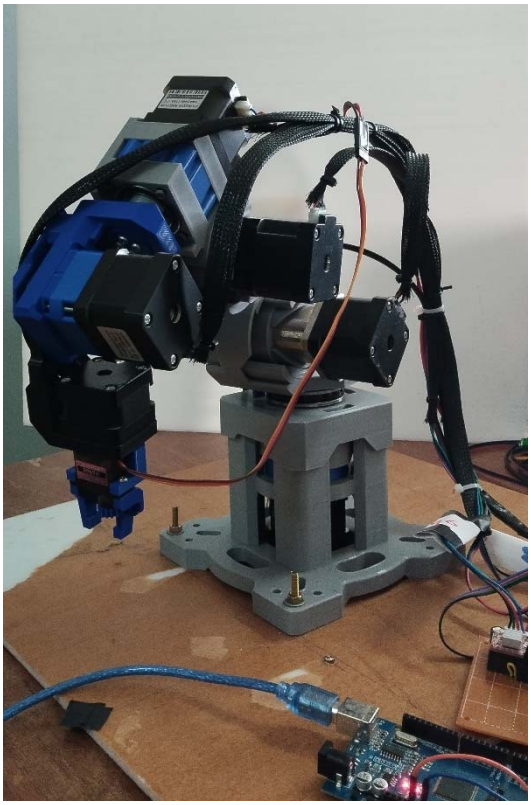
รูปที่ 4.8 หุ่นยนต์ตำแหน่ง Straight up



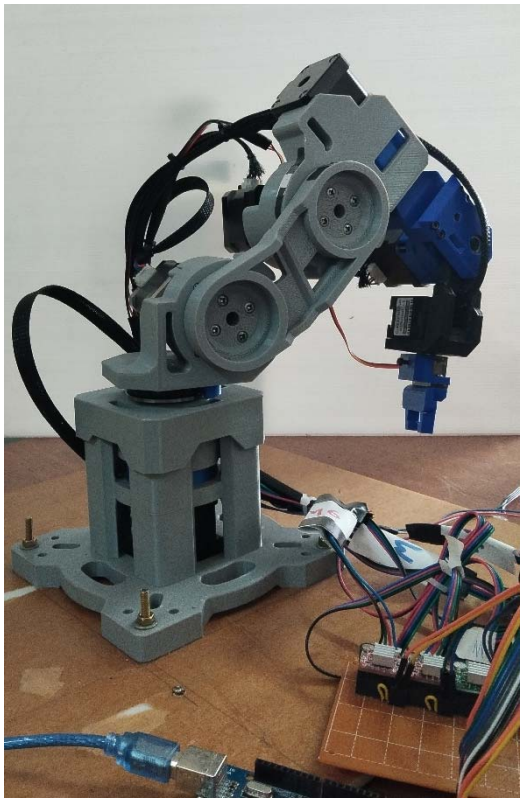
รูปที่ 4.7 หุ่นยนต์ตำแหน่ง Home position



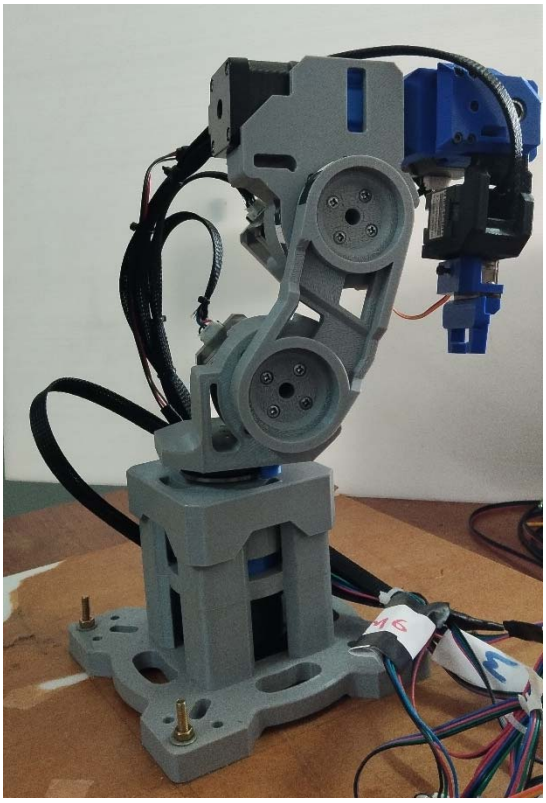
รูปที่ 4.9 หุ่นยนต์ตำแหน่ง Lift object



รูปที่ 4.10 หุ่นยนต์ตำแหน่ง Pick object



รูปที่ 4.12 หุ่นยนต์ตำแหน่ง Drop object



รูปที่ 4.11 หุ่นยนต์ตำแหน่ง Opposite position

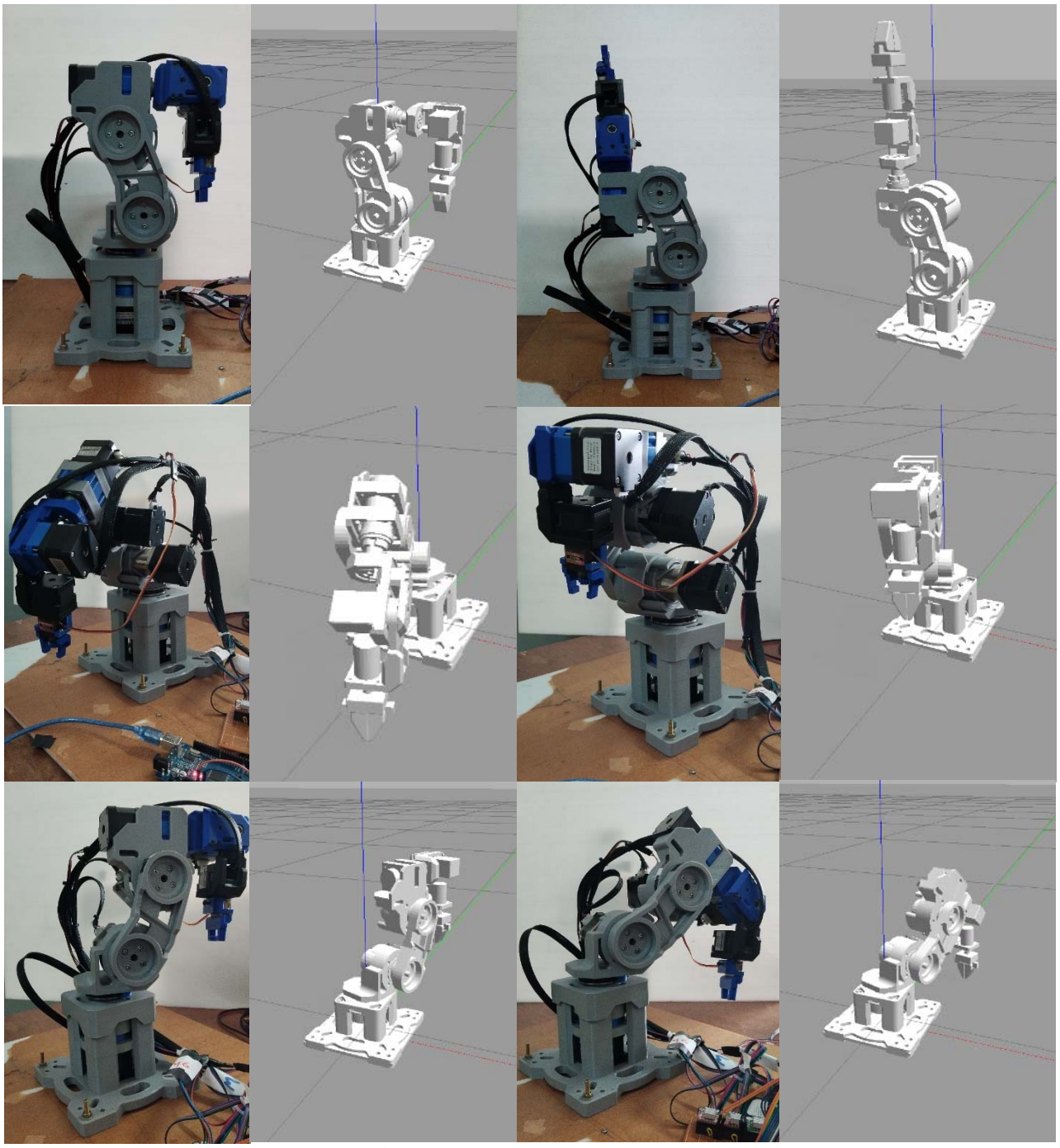
4.3 เปรียบเทียบผลการทดลอง

จากการทดลองในหัวข้อ 4.1 การควบคุมการเคลื่อนที่แบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระผ่านโปรแกรม Rviz และ 4.2 การควบคุมการเคลื่อนที่ของหุ่นยนต์แขนกลจริงผ่านโปรแกรม Rviz พบว่าทั้งสองระบบมีหลักการทำงานที่คล้ายคลึงกัน โดยผู้ใช้สามารถกำหนดค่ามุมข้อต่อผ่าน Rviz และส่งคำสั่งควบคุมไปยัง ROS เพื่อสั่งให้แขนกลเคลื่อนที่

อย่างไรก็ตาม ผลลัพธ์จากการทดลองในสภาพแวดล้อมจำลองและสภาพแวดล้อมจริงอาจมีความแตกต่างกัน เนื่องจากปัจจัยหลายประการ เช่น ข้อจำกัดของมอเตอร์, ความแม่นยำของการส่งข้อมูล, ผลกระทบจากแรงเสียดทานและแรงโน้มถ่วง, ตลอดจนปัจจัยภายนอกที่อาจส่งผลต่อการเคลื่อนที่ของแขนกลจริง

ดังนั้น ในหัวข้อนี้จะทำการ เปรียบเทียบผลการทดลองระหว่างระบบจำลอง (Gazebo) และหุ่นยนต์จริง โดยใช้เกณฑ์ต่างๆ เช่น ความถูกต้องของตำแหน่งและมุมข้อต่อ, เวลาตอบสนองของระบบ, เสถียรภาพของการเคลื่อนที่ และความแม่นยำของการควบคุม เพื่อวิเคราะห์ข้อแตกต่างที่เกิดขึ้น และหาแนวทางปรับปรุงให้การควบคุมหุ่นยนต์ในสภาพแวดล้อมจริงมีประสิทธิภาพมากที่สุด

4.3.1 รูปเปรียบเทียบผลการทดลอง



รูปที่ 4.13 รูปเปรียบเทียบผลการทดลอง

4.3.2 ผลการเปรียบเทียบระหว่างแบบจำลองใน Gazebo และหุ่นยนต์จริง

1. ความแม่นยำของการเคลื่อนที่

แบบจำลองใน Gazebo: แขนกลสามารถเคลื่อนที่ได้อย่างแม่นยำตามค่ามุมข้อต่อที่กำหนด เนื่องจากไม่มีข้อจำกัดจากแรงทางกายภาพ เช่น แรงเสียดทานหรือแรงโน้มถ่วง
หุ่นยนต์จริง: มีความคลาดเคลื่อนบางส่วน เนื่องจากแรงเสียดทานของข้อต่อ, แรงโน้มถ่วง และความแม่นยำของมอเตอร์ โดยเฉพาะอย่างยิ่งเมื่อต้องเคลื่อนที่ในท่าทางที่ซับซ้อน

2. เวลาตอบสนองของระบบ

แบบจำลองใน Gazebo: การส่งค่ามุมข้อต่อและการเคลื่อนที่ของแขนกลเกิดขึ้นแบบเรียลไทม์ เนื่องจากเป็นการคำนวณในซอฟต์แวร์ที่ไม่มีข้อจำกัดทางกายภาพ
หุ่นยนต์จริง: มีความหน่วงบางประการ เนื่องจากต้องใช้เวลาส่งข้อมูลจาก ROS ไปยัง Arduino และแปลงคำสั่งให้มอเตอร์เคลื่อนที่ นอกจากนี้ ความล่าช้าจากการตอบสนองของมอเตอร์ยังเป็นปัจจัยที่ต้องพิจารณา

3. เสถียรภาพของการเคลื่อนที่
แบบจำลองใน Gazebo: การเคลื่อนที่มีความราบรื่นและต่อเนื่อง เนื่องจากใช้การจำลองทางคณิตศาสตร์ที่ไม่มีแรงกระทำภายนอก
หุ่นยนต์จริง: บางจุดของการเคลื่อนที่อาจมีการสั่นหรือกระตุก ซึ่งอาจเกิดจากการเร่งและลดความเร็วของมอเตอร์, ความแข็งแรงของโครงสร้างแขนกล, หรือข้อจำกัดทางกลไกของข้อต่อ

4. การควบคุมและการสื่อสารข้อมูล

แบบจำลองใน Gazebo: การส่งค่ามุมข้อต่อผ่าน ROS Controller สามารถทำได้อย่างรวดเร็วและแม่นยำ
หุ่นยนต์จริง: การสื่อสารระหว่าง ROS และ Arduino มีความล่าช้าเล็กน้อย เนื่องจากต้องทำการประมวลผลค่ามุมข้อต่อและแปลงเป็นคำสั่งสำหรับ Stepper Motor นอกจากนี้ ยังมีโอกาสเกิดข้อผิดพลาดจากสัญญาณรบกวนหรือการสูญเสียข้อมูล

สรุปผลการเปรียบเทียบ:

จากการเปรียบเทียบพบว่า การจำลองผ่าน Gazebo ช่วยให้สามารถทดสอบและปรับแต่งการเคลื่อนที่ของแขนกลได้อย่างแม่นยำก่อนนำไปใช้กับหุ่นยนต์จริง โดยไม่มีข้อจำกัดจากแรงภายนอก อย่างไรก็ตาม เมื่อทดสอบกับ หุ่นยนต์จริง พบว่ามีปัจจัยทางกายภาพ เช่น แรงเสียดทาน แรงโน้มถ่วง และข้อจำกัดของมอเตอร์ที่ส่งผลต่อความแม่นยำของการเคลื่อนที่

ดังนั้น การนำค่าที่ได้จากการจำลองไปใช้กับหุ่นยนต์จริงควรมีการปรับแก้ค่ามุมข้อต่อหรือใช้เทคนิคการควบคุมเพิ่มเติม เช่น การชดเชยแรงโน้มถ่วง (Gravity Compensation), การใช้ตัวควบคุม PID เพื่อเพิ่มความแม่นยำ, หรือการปรับปรุงโครงสร้างกลไกของแขนกล เพื่อให้ได้ผลลัพธ์ที่ใกล้เคียงกับการจำลองมากที่สุด

บทที่ 5 สรุปและอภิปราย

5.1 สรุปผลการดำเนินงาน

จากการดำเนินงานทางผู้จัดทำได้ดำเนินการตามขั้นตอนการดำเนินงานที่ได้วางแผนไว้ คือการกำหนดขอบเขตของแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ, การออกแบบโมเดลจำลองของหุ่นยนต์แขนกล 6 องศาอิสระ และการสร้างแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ ซึ่งเป็นขั้นตอนพื้นฐานในการสร้างแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ และการออกแบบแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ โดยหุ่นยนต์แขนกลที่ทำการออกแบบจะคำนึงถึงขอบเขตของงานที่เราตั้งไว้, ความยาวของแขนหุ่นยนต์ (Link), น้ำหนักโดยประมาณ การประยุกต์ใช้แบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ ที่ถูกสร้างขึ้นเพื่อศึกษาส่วนประกอบ กลไก ทฤษฎีที่เกี่ยวข้อง และหลักการทำงานของหุ่นยนต์อุตสาหกรรม ตลอดจนศึกษาการควบคุมแขนหุ่นยนต์อุตสาหกรรม เพื่อที่จะสามารถพัฒนาและต่อยอดองค์ความรู้ในระดับอุตสาหกรรมได้ในอนาคต

นอกจากนี้ ได้มีการจำลองการเคลื่อนที่แบบเสมือนด้วยโปรแกรม RViz ซึ่งทำให้ผู้ใช้งานสามารถกำหนดค่าพิกัดที่ต้องการในแต่ละจุดหมุน (Joint) และทำการแสดงผลผ่านโปรแกรม Gazebo เพื่อให้เห็นภาพการเคลื่อนที่ของแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระ

เพื่อให้สามารถนำแบบจำลองที่พัฒนาขึ้นไปใช้งานกับหุ่นยนต์จริง ได้มีการพัฒนาระบบที่สามารถเชื่อมต่อระหว่าง ROS และ Arduino ซึ่งทำหน้าที่ควบคุมมอเตอร์สเต็ปเปอร์ของแขนกลจริง โดยมีการใช้ rosserial เพื่อสื่อสารค่ามุมของข้อต่อจาก ROS ไปยัง Arduino ผ่านพอร์ตอนุกรม และใช้ A4988 stepper driver ในการขับเคลื่อนมอเตอร์แต่ละตัวเมื่อมีการกำหนดค่ามุมของข้อต่อผ่าน MoveIt หรือการกำหนดค่าผ่าน ROS Topic ระบบจะส่งข้อมูลไปยัง Arduino เพื่อให้สเต็ปเปอร์มอเตอร์เคลื่อนที่ไปยังตำแหน่งที่ต้องการ

จากการดำเนินงานพบว่า แบบจำลองที่พัฒนาขึ้นสามารถเชื่อมต่อและควบคุมแขนหุ่นยนต์จริงได้สำเร็จ โดยสามารถรับค่ามุมจาก ROS และส่งไปยัง Arduino เพื่อสั่งงานมอเตอร์สเต็ปเปอร์ให้เคลื่อนที่ได้อย่างถูกต้อง นอกจากนี้ยังสามารถแสดงผลการเคลื่อนที่ใน RViz และ Gazebo ได้แบบเรียลไทม์ ซึ่งช่วยให้สามารถตรวจสอบและปรับปรุงการทำงานของแขนกลได้สะดวกยิ่งขึ้น

5.2 ปัญหาในการดำเนินงานและแนวทางการแก้ไข

จากการดำเนินงานทำให้พบปัญหาและมีแนวทางการแก้ไขปัญหาดังกล่าวดังนี้

1. การใช้งานการออกแบบแบบจำลองหุ่นยนต์แขนกล 6 องศาอิสระด้วยโปรแกรม ROS Rviz Movit Gazebo ซึ่งเป็นโปรแกรมที่พึ่งใช้งานเป็นครั้งแรก จึงทำให้ใช้เวลาในการเรียนรู้และใช้งานที่เป็นระยะเวลานาน

2. การสร้างชิ้นส่วนโมเดล เพื่อนำมาใช้ในการจำลองผ่าน Rviz ไม่สามารถใช้ไฟล์โมเดลเก่าจากโครงงาน [4] ทำให้มีปัญหาต้องสร้างขึ้นมาใหม่ทั้งหมด อาจทำให้โมเดลมีส่วนที่ไม่คล้ายกับหุ่นยนต์จริง

5.3 ข้อเสนอแนะ

จากการพัฒนาแบบจำลองและระบบควบคุมแขนกล 6 องศาอิสระ พบว่าระบบสามารถทำงานได้ตามที่กำหนด อย่างไรก็ตาม ยังมีประเด็นที่สามารถพัฒนาและปรับปรุงเพิ่มเติมเพื่อเพิ่มประสิทธิภาพของระบบ ดังนี้

1. การปรับปรุงความแม่นยำของมอเตอร์สเต็ปเปอร์ เนื่องจากมอเตอร์สเต็ปเปอร์มีความละเอียดจำกัดจากการใช้ไดรเวอร์ A4988 ซึ่งอาจทำให้เกิดการกระตุกขณะเคลื่อนที่
2. การเพิ่มระบบป้อนกลับ (Feedback System) ปัจจุบันการควบคุมหุ่นยนต์เป็นแบบ Open-loop โดยไม่มีการตรวจสอบตำแหน่งจริงของข้อต่อ ควรพิจารณาเพิ่ม เอนโค้ดเดอร์ (Encoder) หรือ IMU Sensor เพื่อให้สามารถตรวจสอบตำแหน่งที่แท้จริงของแขนกล และทำให้ระบบควบคุมมีความแม่นยำมากขึ้น
3. การเพิ่มความสามารถในการปรับความเร็วของแต่ละข้อต่อ เนื่องจากปัจจุบันค่าความเร็วของมอเตอร์ถูกกำหนดแบบคงที่ในโค้ด Arduino ควรเพิ่มฟังก์ชันให้สามารถส่งค่าความเร็วจาก ROS ไปยัง Arduino เพื่อให้สามารถปรับเปลี่ยนความเร็วในระหว่างการทำงานได้
4. การพัฒนา GUI เพื่อการควบคุมที่ง่ายขึ้น เนื่องจากปัจจุบันการควบคุมแขนหุ่นยนต์ต้องใช้ RViz หรือ Terminal ซึ่งอาจไม่สะดวกสำหรับผู้ใช้งานทั่วไป ควรพัฒนา Graphical User Interface (GUI) ที่ช่วยให้สามารถป้อนค่ามุมของข้อต่อ และควบคุมแขนหุ่นยนต์ได้ง่ายขึ้น
5. การทดสอบและนำไปใช้งานจริงในอุตสาหกรรม ควรดำเนินการทดสอบกับงานจริงในอุตสาหกรรม เช่น งานประกอบชิ้นส่วนอัตโนมัติ เพื่อประเมินประสิทธิภาพของแขนกลและทำการปรับปรุงให้เหมาะสมกับการใช้งานจริง
6. ปรับปรุงโมเดลหุ่นยนต์แขนกลให้มีรายละเอียดให้มีความสมจริงมากยิ่งขึ้น

สรุปข้อเสนอแนะ:

การพัฒนาแขนกล 6 องศาอิสระในโครงงานนี้สามารถใช้งานได้ตามที่วางแผนไว้ อย่างไรก็ตาม ยังสามารถปรับปรุงเพิ่มเติมในด้านโมเดลหุ่นยนต์แขนกล ความแม่นยำ ความยืดหยุ่น และความสะดวกในการใช้งาน เพื่อให้แขนกลสามารถนำไปใช้ในงานจริงได้อย่างมีประสิทธิภาพมากขึ้น

การอ้างอิง

[1] แผนกวิชาเมคคาทรอนิกส์ วิทยาลัยเทคนิคชลบุรี, “Articulated Robot”, 2020. [ออนไลน์]. Available: http://ctc.chontech.ac.th/external_newsblog.php?links=606. [เข้าถึงเมื่อ: 10 ส.ค. 2024].

[2] ROS, “Wiki.ROS.org,” [ออนไลน์]. Available: https://wiki.ros.org/. [เข้าถึงเมื่อ: 10 ส.ค. 2024].

[3] A. A. Pinouts, “Arduino Duemilanove Pinout”, 2016. [ออนไลน์]. Available: https://arduino.pinout.guide/arduino_duemilanove_pinout.png. [เข้าถึงเมื่อ: 10 ส.ค. 2024].

[4] พัชรพงศ์ ลำต้น และ เกียรติสุริยา บุราเลข, “การออกแบบชุดจำลองแขนหุ่นยนต์อุตสาหกรรม (Design of an Industrial Robot Arm),” ในรายวิชา ELECTRONIC SYSTEMS ENGINEERING PROJECT 2022.

[5] Age of Robotics, “Simulate Your Robot Arm In ROS Noetic,” 9 มิ.ย. 2024. [ออนไลน์]. Available: https://github.com/ageofrobotics/Simulate_Your_Robot_Arm_In_ROS_Noetic. [เข้าถึงเมื่อ: 15 ส.ค. 2024].

[6] สฤษฎ์พงศ์ มีบุญ, “Project Robot Arm 6DOF,” [ออนไลน์]. Available: https://github.com/sttpp58/robot_arm_6DOF. [เข้าถึงเมื่อ: 20 ก.พ. 2025].

ภาคผนวก

ภาคผนวก ก

ภาคผนวก ก.1 โปรแกรมควบคุมหุ่นยนต์แขนกล 6 องศาอิสระ ผ่าน ROS

```
#include <ros.h>
#include <std_msgs/Int32MultiArray.h>

#define NUM_MOTORS 6
const int STEP_PINS[NUM_MOTORS] = {53, 51, 49, 47, 45, 43};
const int DIR_PINS[NUM_MOTORS]  = {52, 50, 48, 46, 44, 42};

ros::NodeHandle nh;

long target_angle[NUM_MOTORS] = {0};
long current_angle[NUM_MOTORS] = {0};
const float steps_per_degree[NUM_MOTORS] = {80.0 / 1.8, 100.0 / 1.8, 27.0 / 1.8, 4.0 / 1.8, 4.0 / 1.8, 1.0 /
1.8};

//  ปรับความเร็วแต่ละมอเตอร์ (ค่ามาก = ช้ากว่า, ค่าน้อย = เร็วกว่า)
const int motor_speed[NUM_MOTORS] = {300, 450, 800, 850, 900, 950};

void messageCallback(const std_msgs::Int32MultiArray& msg) {
    if (msg.data_length < NUM_MOTORS) return;

    for (int i = 0; i < NUM_MOTORS; i++) {
        target_angle[i] = msg.data[i];
    }

    Serial.print("Angles: ");
    for (int i = 0; i < NUM_MOTORS; i++) {
        Serial.print(target_angle[i]); Serial.print(" ");
    }
    Serial.println();
}

ros::Subscriber<std_msgs::Int32MultiArray> sub("/target_joint_degrees", &messageCallback);

void setup() {
    Serial.begin(115200);
    nh.getHardware()->setBaud(115200);
    nh.initNode();
    nh.subscribe(sub);

    for (int i = 0; i < NUM_MOTORS; i++) {
        pinMode(STEP_PINS[i], OUTPUT);
        pinMode(DIR_PINS[i], OUTPUT);
    }
}

void moveAllMotors() {
    long steps[NUM_MOTORS];
    long max_steps = 0;

    for (int i = 0; i < NUM_MOTORS; i++) {
```

```
        steps[i] = (target_angle[i] - current_angle[i]) * steps_per_degree[i];
        steps[i] = abs(steps[i]);
        if (steps[i] > max_steps) max_steps = steps[i];
        digitalWrite(DIR_PINS[i], (target_angle[i] > current_angle[i]) ? HIGH : LOW);
    }

    for (long step = 0; step < max_steps; step++) {
        for (int i = 0; i < NUM_MOTORS; i++) {
            if (step < steps[i]) {
                digitalWrite(STEP_PINS[i], HIGH);
            }
        }
        delayMicroseconds(500);
        for (int i = 0; i < NUM_MOTORS; i++) {
            if (step < steps[i]) {
                digitalWrite(STEP_PINS[i], LOW);
            }
        }

        // ปรับความเร็วแต่ละมอเตอร์
        int min_delay = 80;
        int max_delay = 800;
        int step_delay = map(motor_speed[step % NUM_MOTORS], 300, 800, min_delay, max_delay);
        delayMicroseconds(step_delay);
    }

    for (int i = 0; i < NUM_MOTORS; i++) {
        current_angle[i] = target_angle[i];
    }
}

void loop() {
    nh.spinOnce();
    moveAllMotors();
}
```

ภาคผนวก ก.2 โปรแกรมรับค่ามุมจาก ROS โดยใช้ Python

```
1  #!/usr/bin/env python3
2  import rospy
3  from moveit_msgs.msg import DisplayTrajectory
4  from std_msgs.msg import Int32MultiArray  # Use Int32MultiArray for integer values
5  import numpy as np
6
7  # Publisher for integer joint angles
8  pub = rospy.Publisher("/target_joint_degrees", Int32MultiArray, queue_size=10)
9
10 def callback(msg):
11     if not msg.trajectory:
12         rospy.logwarn("No trajectory received from MoveIt!")
13         return
14
15     # Extract last joint positions in the trajectory
16     joint_values_rad = msg.trajectory[0].joint_trajectory.points[-1].positions
17     joint_values_deg = np.degrees(joint_values_rad)  # Convert radians to degrees
18     joint_values_int = [int(round(angle)) for angle in joint_values_deg]  # Convert to integers
19
20     # Publish as Int32MultiArray
21     joint_msg = Int32MultiArray()
22     joint_msg.data = joint_values_int[:6]  # Take only 6 values
23     pub.publish(joint_msg)
24
25     rospy.loginfo(f"Published target joint angles (int degrees): {joint_msg.data}")
26
27 def listener():
28     rospy.init_node("moveit_joint_publisher", anonymous=True)
29     rospy.Subscriber("/move_group/display_planned_path", DisplayTrajectory, callback)
30     rospy.spin()
31
32 if __name__ == "__main__":
33     listener()
34
```

ภาคผนวก ก.3 การเชื่อมต่อ ROS กับ Arduino Mega 2560

การเชื่อมต่อ Robot Operating System (ROS) กับ Arduino Mega 2560 เป็นขั้นตอนสำคัญในการควบคุมหุ่นยนต์จากคอมพิวเตอร์ โดยใช้ ROS เป็นแพลตฟอร์มสำหรับสื่อสารและส่งคำสั่งไปยัง Arduino เพื่อควบคุมมอเตอร์และเซ็นเซอร์ต่างๆ ในโครงงานนี้มีการใช้ rosserial เพื่อให้ Arduino สามารถสื่อสารกับ ROS ได้อย่างมีประสิทธิภาพ

1. การติดตั้งและตั้งค่า rosserial บน ROS

1.1 เปิด Terminal และติดตั้ง **rosserial** โดยใช้คำสั่ง:

```
$ sudo apt-get install ros-noetic-rosserial ros-noetic-rosserial-arduino ros-noetic-rosserial-python
```

1.2 ตั้งค่า Arduino เพื่อรองรับ rosserial:

```
$ cd ~/Arduino/libraries
$ rm -rf ros_lib
$ rosrun rosserial_arduino make_libraries.py
```

ไฟล์ไลบรารีของ ROS จะถูกสร้างในโฟลเดอร์ Arduino/libraries/ros_lib

2. อัปโหลดโค้ด ไปยัง Arduino Mega 2560

2.1 อัปโหลดโค้ดโปรแกรมควบคุมหุ่นยนต์แขนกล 6 องศาอิสระ ผ่าน ROS ไปยัง Arduino

2.2 เชื่อมต่อ Arduino Mega 2560 กับคอมพิวเตอร์ผ่านสาย USB และอัปโหลดโค้ดไปยังบอร์ด

3. การเชื่อมต่อ ROS กับ Arduino ผ่าน rosserial

3.1 ตรวจสอบพอร์ตของ Arduino Mega 2560 โดยใช้คำสั่ง:

```
$ ls /dev/ttyUSB*
```

หรือ

```
$ ls /dev/ttyACM*
```

สมมติว่า Arduino ถูกเชื่อมต่อที่ /dev/ttyACM0

3.2 เริ่มต้นการสื่อสารระหว่าง ROS และ Arduino โดยใช้คำสั่ง:

```
$ rosrun rosserial_python serial_node.py /dev/ttyACM0
```

ภาคผนวก ข

Credit by: สามารถดูเพื่อเติมได้ในช่อง YouTube: https://www.youtube.com/@Age.of.Robotic

ไฟล์ PDF ภาษาอังกฤษ:
https://github.com/ageofrobotics/Simulate_Your_Robot_Arm_In_ROS_Noetic/blob/main/Importing_Your_ROS_Package_Generated_from_SolidWorks_in_ROS_Noetic.pdf

1 Install ROS Noetic and Create CATKIN Workspace

1.1 ROS installation

หากไม่ได้ติดตั้ง ROS ให้ไปที่ https://wiki.ros.org/noetic/Installation/Ubuntu หน้านี้ โดยในโครงงานนี้ใช้ Ubuntu 20.04 และ ROS Noetic

หมายเหตุสำคัญ: หากใช้ ROS เวอร์ชั่นอื่น Code ที่ให้ไว้ในบทช่วยสอนนี้อาจใช้งานไม่ได้

2.2 Setup a CATKIN Workspace

หากใช้ ROS มาเป็นเวลานานและได้สร้าง CATKIN Workspace ไว้แล้ว สามารถใช้เวิร์กสเปซนั้นสำหรับบทช่วยสอนนี้ได้เลย หากแต่เป็นผู้ใช้ใหม่หรือต้องการสร้างเวิร์กสเปซแยกต่างหากสำหรับโปรเจ็กต์นี้ เพียงคัดลอกและรันคำสั่งที่ระบุไว้ด้านล่างใน Ubuntu Terminal ทีละคำสั่ง

การติดตั้ง CATKIN Tools และแพ็กเกจ std_msg

1. อัปเดตแพ็กเกจ Ubuntu

```
$ sudo apt-get update
```

1. ติดตั้ง CATKIN Tools สำหรับ ROS Noetic

```
$ sudo apt-get install ros-noetic-catkin python3-catkin-tools
```

2. ติดตั้งแพ็กเกจ std_msg

```
$ sudo apt install ros-noetic-std-msgs
```

สร้าง CATKIN Workspace

1. อัปเดตแพ็กเกจ Ubuntu

```
$ sudo apt-get update
```

2. ไปที่โฮมไดเรกทอรี

```
$ cd ~/
```

3. สร้างโฟลเดอร์ catkin workspace พร้อมกับโฟลเดอร์ src ในนั้น ฉันกำลังสร้างเวิร์กสเปซที่มีชื่อว่า “moveit_ws”

```
$ mkdir --parents moveit_ws/src
```

สามารถใช้ชื่อใดๆ ก็ได้สำหรับพื้นที่ทำงาน เช่น "catkin_ws"

4. ไป catkin workspace ที่สร้างขึ้น

```
$ cd ~/moveit_ws
```

5. เริ่มต้นการใช้งาน Catkin workspace

```
$ catkin init
```

6.ไปที่ไดเรกทอรี catkin workspace ที่สร้างขึ้น หากยังไม่มีอยู่ในนั้น

```
$ cd ~/moveit_ws
```

7. สร้าง workspace.

```
$ catkin build
```

หลังจากเริ่มต้นและสร้างพื้นที่ทำงาน CATKIN แล้ว ระบบจะสร้างโฟลเดอร์ต่างๆ เช่น devel, src , .etc ในโฟลเดอร์พื้นที่ทำงาน

8. แหล่งที่มาของไฟล์ “setup.bash” ซึ่งสร้างขึ้นโดยอัตโนมัติในโฟลเดอร์ “devel” ของพื้นที่ทำงาน catkin หากอยู่ในพื้นที่ทำงาน catkin ของแล้ว:

```
$ source devel/setup.bash
```

หากไม่ได้อยู่ใน catkin workspace ให้ระบุเส้นทางแบบเต็ม:

```
$ source ~/moveit_ws/devel/setup.bash
```

สำคัญการหาแหล่งที่มาของไฟล์ setup.bash จะทำให้เวิร์กสเปซทำงานและมาสเตอร์ ROS ทราบถึงไฟล์และแพ็กเกจที่มีอยู่ในเวิร์กสเปซ ต้องทำการหาแหล่งที่มาของไฟล์นี้ทุกครั้งที่คุณเปิดเทอร์มินัลใหม่

2 Add the URDF Package to Your Workspace and Add the Dependencies

2.1 คัดลอกแพ็กเกจ ROS ไปยังพื้นที่ทำงาน catkin

สามารถดาวน์โหลดแพ็คเกจพร้อมใช้งานของแขนหุ่นยนต์ได้ที่นี่:

https://github.com/ageofrobotics/Simulate_Your_Robot_Arm_In_ROS_Noetic/blob/main/robot_arm_urdf.zip

ในโครงงานนี้จะเป็น: https://github.com/sttpp58/robot_arm_6DOF

ชื่อแพ็กเกจคือ robot_arm_urdf คัดลอกไปไว้ในโฟลเดอร์ต่อไปนี้: ~/moveit_ws/src/robot_arm_urdf

2.2 แก้ไข CmakeLists.txt

Full YouTube Tutorial Link: https://youtu.be/CB_lwesDkqg

ไฟล์ CMakeLists.txt เริ่มต้นเป็นไฟล์พื้นฐานและไม่มีการอ้างอิงที่สำคัญอื่นๆ ที่จำเป็นสำหรับการจำลองของหุ่นยนต์ ให้ทำการแก้ไขสคริปต์ในไฟล์นี้ตามที่ระบุไว้ด้านล่าง สีเขียวคือส่วนที่เพิ่ม

CmakeLists.txt [หลังการแก้ไข]

```
cmake_minimum_required(VERSION 2.8.3)

project(robot_arm_urdf)

find_package(catkin REQUIRED COMPONENTS
 message_generation
 roscpp
 rospy
 std_msgs
 geometry_msgs
 urdf
 xacro
 message_generation
)

catkin_package(
CATKIN_DEPENDS
 geometry_msgs
 roscpp
 rospy
 std_msgs
)

find_package(roslaunch)

foreach(dir config launch meshes urdf)
          install(DIRECTORY ${dir}/
                  DESTINATION
${CATKIN_PACKAGE_SHARE_DESTINATION}/${dir})
endforeach(dir)
```

2.3 แก้ไขไฟล์ package.xml

ไฟล์ package.xml เริ่มต้นมีการเพิ่มการอ้างอิงเพียงเล็กน้อย จำเป็นต้องเพิ่มการอ้างอิงเพิ่มเติม เพื่อวัตถุประสงค์ในการจำลองของหุ่นยนต์ ส่วนแก้ไขสีเขียวคือส่วนที่เพิ่ม

package.xml [After editing]

```
<package format="2">
 <name>robot_arm_urdf</name>
 <version>1.0.0</version>
 <description>
 <p>URDF Description package for robot_arm_urdf</p>
 <p>This package contains configuration data, 3D models and launch files for
robot_arm_urdf robot</p>
 </description>
 <author>TODO</author>
 <maintainer email="TODO@gmail.com" />
 <license>BSD</license>

 <buildtool_depend>catkin</buildtool_depend>

 <build_depend>message_generation</build_depend>
 <build_depend>roscpp</build_depend>
 <build_depend>rospy</build_depend>
 <build_depend>std_msgs</build_depend>
 <build_depend>geometry_msgs</build_depend>
 <build_depend>urdf</build_depend>
 <build_depend>xacro</build_depend>

 <depend>roslaunch</depend>
 <depend>robot_state_publisher</depend>
 <depend>rviz</depend>
 <depend>joint_state_publisher</depend>
 <depend>joint_state_publisher_gui</depend>
 <depend>gazebo</depend>
 <depend>moveit_simple_controller_manager</depend>

 <build_export_depend>roscpp</build_export_depend>
 <build_export_depend>rospy</build_export_depend>
 <build_export_depend>std_msgs</build_export_depend>
 <build_export_depend>geometry_msgs</build_export_depend>
 <build_export_depend>urdf</build_export_depend>
 <build_export_depend>xacro</build_export_depend>

 <exec_depend>roscpp</exec_depend>
 <exec_depend>rospy</exec_depend>
 <exec_depend>std_msgs</exec_depend>
 <exec_depend>geometry_msgs</exec_depend>
 <exec_depend>urdf</exec_depend>
 <exec_depend>xacro</exec_depend>
 <exec_depend>message_runtime</exec_depend>


 <export>
  <architecture_independent />
 </export>
</package>
```

2.4 แก้ไขไฟล์ URDF เพื่อให้เหมาะกับการจำลอง

Full YouTube Tutorial: https://youtu.be/gfXUqaj2Xm0

เปิดไฟล์ URDF ที่มีอยู่ในโฟลเดอร์ URDF ของแพ็คเกจ และเริ่มทำการเปลี่ยนแปลงตามที่ระบุไว้ด้านล่าง:

1. เราต้องเพิ่มลิงก์ “world” และแก้ไข “base_link” ให้กับลิงก์นั้น เพิ่มโค้ดสั้นๆ ด้านล่างนี้ใน URDF ระหว่างแท็ก <robot name=“robot_arm_urdf”> และ <link name=“base_link”> ดังที่ระบุไว้ด้านล่าง

```
<robot
name="robot_arm_urdf">

<link name="world"/>
<joint name="base_joint" type="fixed">
     <parent link="world"/>
     <child link="base_link"/>
     <origin rpy="0 0 0" xyz="0.0 0.0 0.1"/>
</joint>

<link
name="base_link">
```

2. ตรวจสอบแท็ก <joint> แต่ละแท็กในไฟล์ URDF ตรวจสอบให้แน่ใจว่าขีดจำกัด “ล่าง” ของข้อต่อจะน้อยกว่าขีดจำกัด “บน” เสมอ ไม่ว่าในกรณีใด หากขีดจำกัด “บน” มีขนาดเล็กกว่าขีดจำกัด “ล่าง” ให้สลับตำแหน่งของทั้งสองแท็กนี้ ตัวอย่างเช่น

ถูก:

```
<limit
   lower="-1.57"
   upper="1.57"
   effort="300"
   velocity="3" />
```

ผิด:

```
<limit
   lower="1.57"
   upper="-1.57"
   effort="300"
   velocity="3" />
```

3. เพิ่มแท็กการส่งผ่านสำหรับแต่ละข้อต่อที่มีอยู่ใน URDF คัดลอกโค้ดสั้นๆ ด้านล่างนี้และคัดลอกไว้ที่ท้าย URDF ก่อนแท็ก </robot> ต้องเพิ่มโค้ดสั้นๆ นี้สำหรับแต่ละข้อต่อใน URDF

```
<transmission name="link_n_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="joint_n">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    </joint>
    <actuator name="link_n_motor">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
    </actuator>
</transmission>
```

หมายเหตุ:

- แทนที่ส่วน “link_n” ด้วยชื่อของลิงก์ย่อยของข้อต่อและ “joint_n” ด้วยชื่อข้อต่อที่แน่นอน
- ชื่อการส่งข้อมูลและแอคทูเอเตอร์สามารถกำหนดได้ตามที่ต้องการ
- แต่ชื่อข้อต่อจะต้องเป็นชื่อที่แน่นอนของข้อต่อ
- ตั้งชื่อข้อต่อของโครงงานนี้ว่า “joint_1”,”joint_2”....”joint_n” และลิงก์เป็น “link_1”, “link_2”....”link_n” (เฉพาะลิงก์พื้นฐานเท่านั้นที่ตั้งชื่อว่า “base_link” ควรทำแบบเดียวกันสำหรับลิงก์พื้นฐานด้วย)
- URDF ของบทสอนนี้มีข้อต่อ 7 ข้อ ดังนั้น จึงคัดลอกและวางสไนปเป็ตด้านบนทีละข้อและเปลี่ยนค่า n ในแต่ละสไนปเป็ตตามที่ระบุในวิดีโอ

4. เพิ่ม Gazebo Controller

หลังจากแท็กการส่งสัญญาณครั้งสุดท้าย ให้เพิ่มแท็กตัวควบคุมศาลาตามที่ระบุไว้ด้านล่าง

```
<gazebo>
    <plugin name="control" filename="libgazebo_ros_control.so">
        <robotNamespace>/</robotNamespace>
    </plugin>
</gazebo>
```

หมายเหตุ: - หากเขียนตัวควบคุมในไฟล์ yaml ของตัวควบคุมเส้นทางร่วมภายในเนมสเปซ จะต้องเปลี่ยน “/” ระหว่างแท็ก <robot Namespace> ด้วย “/Namespace ของคุณ”

5. เพิ่มแท็ก Self Collision คล้ายกับแท็ก transmission ให้สร้างแท็ก self-collision สำหรับแต่ละลิงก์ที่เคลื่อนย้ายได้ใน URDF ในกรณีนี้ “link_1” ถึง “link_7” สามารถเคลื่อนย้ายได้ ดังนั้น ฉันจะทำ 7 ครั้ง

```
<gazebo reference="link_n">
    <selfCollide>true</selfCollide>
</gazebo>
```

หมายเหตุ: เปลี่ยน “link_n” เป็นชื่อของลิงก์

2.5 เขียนไฟล์ .yaml เพื่อกำหนดตัวควบคุม ROS ที่จะใช้สำหรับข้อต่อต่างๆและการเผยแพร่สถานะข้อต่อ

YouTube Tutorial Link: https://youtu.be/zoc5I-VVlUg

สร้างไฟล์ .yaml ในโฟลเดอร์ “config” ของแพ็กเกจ ROS ไฟล์นี้จะมีตัวควบคุมข้อต่อสำหรับ: ข้อต่อแขนหุ่นยนต์ ข้อต่อเอฟเฟกเตอร์ปลาย การเผยแพร่สถานะของข้อต่อ และตัวควบคุมอื่นๆ ตามความจำเป็น

ในกรณีของโครงงานนี้ ข้อต่อ_1 ถึงข้อต่อ_5 เป็นของแขนหุ่นยนต์ ข้อต่อ_6 และข้อต่อ_7 เป็นข้อต่อของเอฟเฟกเตอร์ปลายแขน

ทำตามขั้นตอนด้านล่างเพื่อสร้างไฟล์ (.yaml) ที่มีคำจำกัดความของตัวควบคุม ROS สำหรับแขนหุ่นยนต์:

1. เปิดเทอร์มินัล

2. ไปที่โฟลเดอร์ config ของแพ็คเกจ robot arms

```
$ cd ~/YourWorkSpaceName/src/YourURDFPackageName/config
```

สำหรับกรณีนี้คือ:

```
$ cd ~/moveit_ws/src/robot_arm_urdf/config
```

หากมีพื้นที่ทำงานและชื่อแพ็คเกจเดียวกัน ให้ใช้คำสั่งด้านบน

3. สร้างไฟล์ “joint_trajectory_controller.yaml” ในโฟลเดอร์ config ของแพ็กเกจ URDF โดยใช้เทอร์มินัล

```
$ touch joint_trajectory_controller.yaml
```

หมายเหตุ: สามารถตั้งชื่อไฟล์นี้ได้ตามต้องการ ควรใช้ชื่อไฟล์เดียวกันขณะโหลดคอนโทรลเลอร์ในไฟล์เรียกใช้งาน

4. เปิดไฟล์ “joint_trajectory_controller.yaml” ในโปรแกรมแก้ไขข้อความ “gedit”

```
$ gedit joint_trajectory_controller.yaml
```

5. คัดลอกและวางโค้ดด้านล่างลงไปตามที่เป็นอยู่

```
#Instead of using TAB for indentation, use two spaces at the place of one TAB
#Controller to control robot arm joints
robot_arm_controller:
  type: "position_controllers/JointTrajectoryController"
  joints: [joint_1, joint_2, joint_3, joint_4, joint_5]
#Controller to control end effector joints
hand_ee_controller:
  type: "position_controllers/JointTrajectoryController"
  joints: [joint_6, joint_7]
#Controller to continuously publish joint states/positions
joint_state_controller:
  type: "joint_state_controller/JointStateController"
  publish_rate: 50
```

หมายเหตุ:

-อาจสูญเสียการเยื้องบรรทัดหลังจากวางโค้ดนี้ลงในไฟล์ YAML ดังนั้นให้ใช้ช่องว่างสม่ำเสมอในการเยื้องบรรทัดดังที่แสดงในโค้ดตัวอย่าง

- สามารถเปลี่ยนชื่อคอนโทรลเลอร์ได้ตามต้องการ ตัวอย่างเช่น คุณสามารถตั้งชื่อ

“robot_arm_controller” เช่น “my_arm_controller” หรือ “yourArmName_arm_controller” เป็นต้น

- แต่ในขณะที่สร้างคอนโทรลเลอร์ในไฟล์เรียกใช้ ต้องใช้ชื่อที่แน่นอนในไฟล์นั้น

- นอกจากนี้ ต้องเพิ่มชื่อข้อต่ออื่นๆ ของแขน (ถ้ามี) และเอฟเฟกเตอร์ปลาย (ถ้ามี) ในคอนโทรลเลอร์ที่เกี่ยวข้อง โดยคั่นด้วยเครื่องหมายจุลภาค (“,”)

- สามารถใช้ “ประเภท” ของคอนโทรลเลอร์ที่แตกต่างกันตามความต้องการ

- นอกจากนี้ ให้รักษาระยะเยื้องให้เหมาะสม ไม่ยอมรับแท็บ ใช้ระยะห่างที่สม่ำเสมอสำหรับระดับการเยื้อง สามารถใช้ช่องว่างสองช่องที่ตำแหน่งแท็บเดียว (ดูวิดีโอเพื่อดูรายละเอียดhttps://youtu.be/zoc5I-VVlUg)

6. หากพบปัญหาในการสร้างไฟล์นี้หรือรับไวยากรณ์สำหรับไฟล์ yaml ให้ดาวน์โหลดไฟล์ โดยตรงจากที่นี่:

https://github.com/ageofrobotics/Simulate_Your_Robot_Arm_In_ROS_Noetic/blob/main/joint_trajectory_controller.yaml

7. บันทึกและปิดไฟล์

2.6 สร้างไฟล์ .launch เพื่อสร้าง/เปิดแขนหุ่นยนต์ของคุณใน Gazebo

Full YouTube Video Tutorial: https://youtu.be/1k9Jfb25rbw

ทำตามขั้นตอนด้านล่างเพื่อสร้างไฟล์เปิดใช้สำหรับแขนหุ่นยนต์ของเรา:

1. เปิดเทอร์มินัล
2. ไปที่โฟลเดอร์เปิดใช้ของแพ็คเกจแขนหุ่นยนต์

```
$ cd ~/YourWorkSpaceName/src/YourURDFPackageName/launch
```

สำหรับกรณีนี้คือ:

```
$ cd ~/moveit_ws/src/robot_arm_urdf/launch
```

หากมีพื้นที่ทำงานและชื่อแพ็คเกจเดียวกัน ให้ใช้คำสั่งด้านบน

3. สร้างไฟล์ “arm_urdf.launch” ในโฟลเดอร์เปิดใช้ของแพ็คเกจ URDF ของคุณโดยใช้เทอร์มินัล

```
$ touch arm_urdf.launch
```

หมายเหตุ: คุณสามารถตั้งชื่อไฟล์นี้ได้ตามต้องการ

4. เปิดไฟล์ “arm_urdf.launch” ในโปรแกรมแก้ไขข้อความ “gedit”

```
$ gedit arm_urdf.launch
```

5. คัดลอกและวางโค้ดด้านล่างลงในไฟล์

```
<launch>
	<arg name="arg_x" default="0.00" />
	<arg name="arg_y" default="0.00" />
	<arg name="arg_z" default="0.00" />
	<arg name="arg_R" default="0.00" />
	<arg name="arg_P" default="0.00" />
	<arg name="arg_Y" default="0.00" />
	<!--Urdf file path-->
	<param name="robot_description" textfile="$(find
robot_arm_urdf)/urdf/robot_arm_urdf.urdf"/>
	<!--spawn a empty gazebo world-->
	<include file="$(find gazebo_ros)/launch/empty_world.launch" />
	<node name="tf_footprint_base" pkg="tf" type="static_transform_publisher" args="0 0
0 0 0 0
base_link base_footprint 40" />
	<!--spawn model-->
	<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-x $(arg arg_x) -
y $(arg
arg_y) -z $(arg arg_z) -Y $(arg arg_Y) -param robot_description -urdf -model
robot_arm_urdf -J joint_1 0.0 -J
joint_2 0.0 -J joint_3 0.0 -J joint_4 0.0 -J joint_5 0.0 -J joint_6 0.0 -J joint_7 0.0" />
	<!--Load and launch the joint trajectory controller-->
	<rosparam file ="$(find robot_arm_urdf)/config/joint_trajectory_controller.yaml"
command="load"/>
	<node name= "controller_spawner" pkg= "controller_manager" type="spawner"
respawn="false"
output="screen" args="joint_state_controller robot_arm_controller hand_ee_controller"/>
	<!-- Robot State Publisher for TF of each joint: publishes all the current states
of the joint, then RViz
can visualize -->
	<node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"
respawn="false" output="screen"/>

</launch>
```

7. สามารถดาวน์โหลดไฟล์พร้อมใช้งานได้จากที่นี่:
https://github.com/ageofrobotics/Simulate_Your_Robot_Arm_In_ROS_Noetic/blob/main/arm_urdf.launch

8. บันทึกและปิดไฟล์

2.7 สร้าง catkin workspace

1. เปิดเทอร์มินัล
2. ไปที่เวิร์กสเปซของคุณ

    $ cd ~/moveit_ws

3. ซอร์สไฟล์ setup.bash ของเวิร์กสเปซ catkin

    $ source devel/setup.bash

4. สร้างเวิร์กสเปซ

    $ catkin build

5. ซอร์สไฟล์ setup.bash อีกครั้ง

    $ source devel/setup.bash

2.8 เปิดไฟล์เปิดตัวหุ่นยนต์เพื่อโหลดครั้งแรกใน GAZEBO

Full YouTube Video Tutorial: https://youtu.be/1k9Jfb25rbw

1. เปิดเทอร์มินัลและดำเนินการคำสั่งต่อไปนี้เพื่อเริ่ม ROS master

    $ roscore

หลังจากรันคำสั่งนี้แล้ว ให้เปิดเทอร์มินัลนี้ไว้

2. เปิดเทอร์มินัลอื่น
3. ไปที่เวิร์กสเปซของคุณ

    $ cd ~/moveit_ws

4. ซอร์สไฟล์ setup.bash ของเวิร์กสเปซ catkin

    $ source devel/setup.bash

5. สร้างเวิร์กสเปซหากยังไม่ได้สร้างหลังจากทำการเปลี่ยนแปลง

    $ catkin build

6. ซอร์สไฟล์ setup.bash อีกครั้ง

    $ source devel/setup.bash

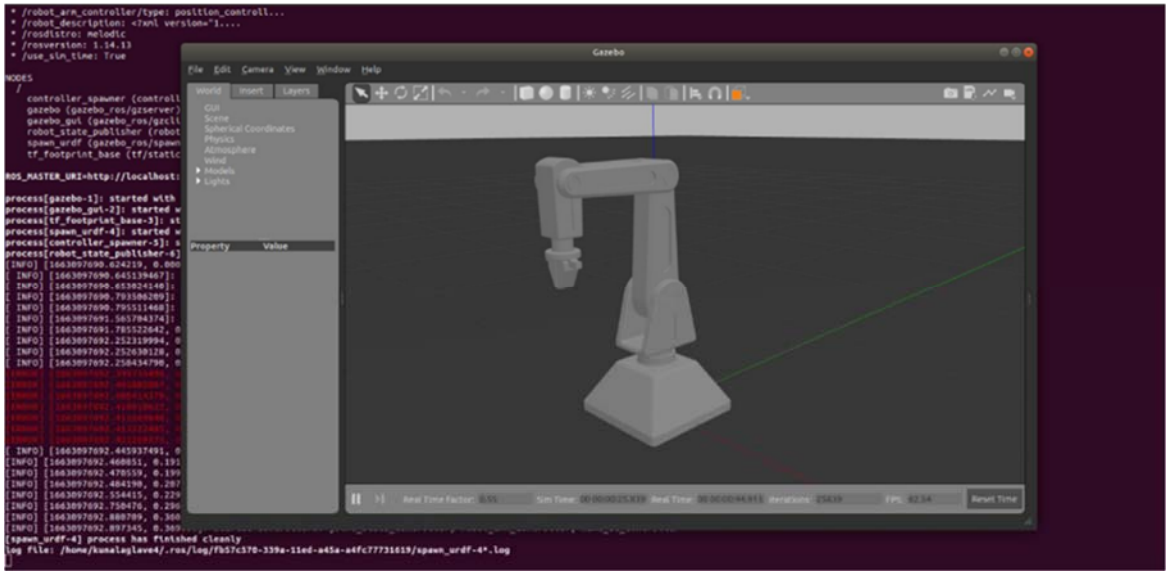
7. เปิดไฟล์ “arm_urdf.launch” ที่เราสร้างในขั้นตอนก่อนหน้านี้

    $ roslaunch your_package_name launch_file_name.launch

ในกรณีนี้จะเป็น:

    $ roslaunch robot_arm_urdf arm_urdf.launch

หากทำทุกอย่างถูกต้องหุ่นยนต์จะเปิดใน Gazebo โดยไม่มีข้อผิดพลาดใดๆ และข้อผิดพลาดที่เกี่ยวข้องกับ PID เท่านั้นที่จะเกิดขึ้นได้ ไม่ควรมีข้อผิดพลาดอื่นๆ เกิดขึ้น นอกจากนี้ ตรวจสอบว่าตัวควบคุมทั้งหมดถูกสร้างขึ้นอย่างถูกต้องหรือไม่



3 Creating a new Manipulator Package to control the Robotic Arm URDF using Moveit Setup Assistant

Moveit เป็นเครื่องมือโอเพ่นซอร์สที่ยอดเยี่ยมสำหรับควบคุมหุ่นยนต์ซึ่งใช้สร้างแพ็คเกจซึ่งสามารถใช้ในการควบคุมหุ่นยนต์ใดๆ ใน ROS ได้ ในบทช่วยสอนนี้ เราจะใช้ไฟล์ URDF ที่เรากำหนดค่าไว้ในบทที่ 2 ของภาคผนวก ข นี้

เว็บไซต์อย่างเป็นทางการของ Moveit: https://moveit.ros.org/

ใช้ ROS 1 และ Moveit 1 สำหรับบทช่วยสอนนี้ Moveit จะถูกติดตั้งตามค่าเริ่มต้นพร้อมกับเวอร์ชันเดสก์ท็อปแบบเต็มของ ROS รุ่นใดก็ได้สามารถดำเนินการตามคำสั่งด้านล่างเพื่อให้แน่ใจว่าทุกอย่างติดตั้งอย่างถูกต้อง นอกจากนี้ โปรแกรมจะติดตั้งสิ่งที่จำเป็นหากยังไม่มีอยู่

คำสั่งในการติดตั้ง Moveit ใน ROS melodic:

```
$ sudo apt install ros-noetic-moveit
```

ติดตั้งตัวควบคุม ROS:

```
$ sudo apt-get install ros-noetic-ros-control ros-noetic-ros-controllers
```

3.1 การสร้างแพ็กเกจ Moveit โดยใช้ Moveit Setup Assistant เพื่อควบคุมไฟล์ URDF

Full YouTube Tutorial: https://youtu.be/M2yiVbJmzKY

หมายเหตุสำคัญ: ควรดูวิดีโอโดยตรง

1. เปิดเทอร์มินัลใหม่และไปที่ Catkin Workspace ซึ่งแพ็กเกจ URDF จะถูกบันทึกไว้ (พื้นที่ทำงาน Catkin ของโครงงานนี้ถูกสร้างในโฮมไดเร็กทอรีโดยใช้ชื่อ “moveit_ws”)

```
$ cd ~/moveit_ws
```

2. สร้างพื้นที่ทำงานหากยังไม่ได้สร้าง

คำสั่ง build:

```
$ catkin build
```

หรือคำสั่ง make:

```
$ catkin_make
```

3. ซอร์สไฟล์ setub.bash ใหม่ในโฟลเดอร์ devel ของเวิร์กสเปซ

```
$ source devel/setup.bash
```

หรือ

```
$ source ~/moveit_ws/devel/setup.bash
```

4. เปิด Moveit Setup Assistant

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

คำเตือน: อย่าเปิด Moveit setup assistance ก่อนที่จะซอร์สไฟล์ setup.bash ของเวิร์กสเปซ

5. จากนั้นทำตามขั้นตอนตามวิดีโอ

6. ขณะสร้างแพ็กเกจจาก moveit ตั้งชื่อว่า “moveit_robot_arm_sim”

3.2 การทดสอบแพ็คเกจ Moveit ที่สร้างโดยใช้ Moveit Setup Assistance

Full YouTube Tutorial: https://youtu.be/M2yiVbJmzKY

ตรวจสอบก่อนว่าสร้างอย่างถูกต้องหรือไม่ เนื่องจากไฟล์ ros_controller.yaml เริ่มต้นที่สร้างโดย Moveit Setup Assistant นั้นไม่สมบูรณ์หรือไม่สามารถใช้งานได้เนื่องจากเป็นไฟล์สำหรับควบคุมหุ่นยนต์

Moveit Setup Assistant ได้ส่งออกแพ็กเกจที่มีข้อมูลทั้งหมดที่จำเป็นสำหรับการควบคุมแขนหุ่นยนต์ และได้สร้างไฟล์สำหรับการเปิดตัวสำหรับการสาธิตครั้งแรก

• **demo.launch**: เปิดหุ่นยนต์ใน Rviz เท่านั้น

• **gazebo.launch**: เปิดหุ่นยนต์ใน gazebo เท่านั้น

• **demo_gazebo.launch**: เปิดหุ่นยนต์ใน Rviz และ Gazebo เช่นกัน แต่การจำลอง Rviz และ Gazebo ไม่เชื่อมต่อถึงกันเนื่องจากตัวควบคุม Moveit หรือตัวควบคุม ROS ไม่สมบูรณ์

ทำตามขั้นตอนด้านล่าง:

1. ก่อนอื่นต้องสร้างเวิร์กสเปซ catkin ของเราก่อน เนื่องจากมีการเพิ่มแพ็คเกจใหม่เข้าไป

   a. เปิดเทอร์มินัลใหม่

   b. ไปที่เวิร์กสเปซของคุณ

   ```
   $ cd ~/moveit_ws
   ```

   c. ซอร์สไฟล์ setup.bash

   ```
   $ source devel/setup.bash
   ```

   d. สร้างเวิร์กสเปซ

   ```
   $ catkin build
   ```

   หรือ

   ```
   $ catkin_make
   ```

e. ซอร์สไฟล์ setup.bash ใหม่อีกครั้ง

```
$ source devel/setup.bash
```

2. ตอนนี้ เปิดไฟล์ demo_gazebo.launch

```
$ roslaunch YourMoveitPackagename demo_gazebo.launch
```

หากตั้งชื่อแพ็คเกจเหมือนกับแพ็คเกจ (moveit_robot_arm_sim) คำสั่งจะเป็นดังนี้:

```
$ roslaunch moveit_robot_arm_sim demo_gazebo.launch
```

3. รอสักครู่ จนกว่าทุกอย่างจะเปิดขึ้น

4. ใน ROS Noetic จะพบกับปัญหาบางอย่างกับไฟล์เริ่มต้นที่สร้างโดย Moveit Setup Assistant เราจำเป็นต้องทำการเปลี่ยนแปลงในไฟล์บางไฟล์และต้องสร้างไฟล์เริ่มต้นของเราเองเพื่อแก้ไขปัญหา ในบทต่อไปเราจะกำหนดค่าแพ็คเกจเพื่อแก้ไขปัญหา

4 การกำหนดค่าแพ็คเกจ Moveit ให้ทำงานอย่างถูกต้อง

แพ็กเกจที่สร้างโดยใช้ Moveit มีข้อมูลของตัวควบคุมบางส่วนที่ไม่สมบูรณ์ เราจำเป็นต้องกำหนดค่าก่อนเริ่มใช้งานเราจำเป็นต้องสร้างตัวควบคุม ROS ใหม่สำหรับแพ็กเกจใหม่ของเรา “moveit_robot_arm_sim” เนื่องจาก “ros_controller” สำหรับการจำลอง moveit ในแพ็กเกจ “moveit_robot_arm_sim” ไม่เชื่อมต่อกับ joint_trajectory_controller ในแพ็กเกจ “robot_arm_urdf”

4.1 การเขียน ROS Controller ใหม่สำหรับการเชื่อมต่อ joint_trajectory_controller กับ ROS manipulator หรือแพ็กเกจ moveit

Full YouTube Tutorial: https://youtu.be/_Js85DDrnvw

1. เปิดเทอร์มินัลใหม่

2. ไปที่โฟลเดอร์ config ของแพ็กเกจ moveit ที่เราสร้างชื่อแพ็กเกจ moveit ในโครงงงนานี้คือ “moveit_robot_arm_sim” (สังเกตการสะกดให้ดี) และชื่อเวิร์กสเปซคือ “moveit_ws” ดังนั้นคำสั่งจะเป็นดังนี้:

```
$ cd ~/moveit_ws/src/moveit_robot_arm_sim/config
```

หากชื่อเวิร์กสเปซหรือแพ็กเกจของแตกต่างกัน คำสั่งจะเป็นดังนี้:

```
$ cd ~/YourWorkSpaceName/src/YourMoveitPackageName/config
```

3. พิมพ์คำสั่งด้านล่างเพื่อสร้างไฟล์ “.yaml” ที่ชื่อ “new_ros_controllers.yaml” แล้วกด Enter

```
$ touch new_ros_controllers.yaml
```

4. ไฟล์ใหม่จะถูกสร้างขึ้น

5. พิมพ์คำสั่งด้านล่างเพื่อเปิดในโปรแกรมแก้ไขข้อความ gedit (สามารถใช้คำสั่งนี้ได้โดยตรงโดยไม่ต้องใช้ “touch” คำสั่งนี้จะสร้างไฟล์ขึ้นมาหากยังไม่มีอยู่และเปิดในโปรแกรมแก้ไข)

```
$ gedit new_ros_controllers.yaml
```

6. คัดลอกและวางโค้ดบรรทัดด้านล่างตามที่เป็นอยู่

new_ros_controllers.yaml file

```
#This is a movit contoller connecting follow_joint_trajectory
controller with JointTrajectoryController
controller_list:
 - name: robot_arm_controller
 action_ns: follow_joint_trajectory
 type: FollowJointTrajectory
 default: true
 joints:
 - joint_1
 - joint_2
 - joint_3
 - joint_4
 - joint_5
 - name: hand_ee_controller
 action_ns: follow_joint_trajectory
 type: FollowJointTrajectory
 joints:
 - joint_6
 - joint_7
```

ด้านล่างนี้คือไฟล์ joint_trajecotry_controller.yaml ที่เขียนขึ้นสำหรับแพ็กเกจ robot_arm_urdf

```
#Instead of using TAB for indentation, use two spaces at the place of one TAB
#Controller to control robot arm joints
robot_arm_controller:
  type: "position_controllers/JointTrajectoryController"
  joints: [joint_1, joint_2, joint_3, joint_4, joint_5]
#Controller to control end effector joints
hand_ee_controller:
  type: "position_controllers/JointTrajectoryController"
  joints: [joint_6, joint_7]
#Controller to continuously publish joint states/positions
joint_state_controller:
  type: "joint_state_controller/JointStateController"
  publish_rate: 50
```

7. หากพบปัญหาในการสร้างไฟล์ ให้ดาวน์โหลดไฟล์ที่พร้อมใช้งานได้จากที่นี่:

https://github.com/ageofrobotics/Simulate_Your_Robot_Arm_In_ROS_Noetic/blob/main/new_ros_controllers.yaml

8. เมื่อเสร็จแล้ว ให้บันทึกและปิดไฟล์

4.2 แก้ไขไฟล์ “simple_moveit_controller_manager.launch.xml” ที่มีอยู่ในโฟลเดอร์เปิดตัวของแพ็คเกจ moveit

เมื่อเราสร้างตัวควบคุม ros ใหม่ เราจำเป็นต้องส่งชื่อตัวควบคุมใหม่นี้ไปยังไฟล์ controller manager ซึ่งมีอยู่ในโฟลเดอร์ “launch” ของแพ็กเกจ moveit ที่สร้างขึ้นโดยใช้ moveit setup assistant สำหรับ ROS Noetic ชื่อของไฟล์คือ: simple_moveit_controller_manager.launch.xml

1. ใช้ตัวจัดการไฟล์เพื่อไปที่โฟลเดอร์ “launch” ของแพ็คเกจ moveit โฟลเดอร์ของบทนี้อยู่ในโฟลเดอร์ src ของ “moveit_ws” workspace (~/moveit_ws/src/moveit_robot_arm_sim/launch)
2. ค้นหาไฟล์ที่ชื่อ “simple_moveit_controller_manager.launch.xml”
3. เปลี่ยนชื่อ “ros_controllers.yaml” เป็น “new_ros_controller.yaml”

```
<launch>
 <!-- Define the MoveIt controller manager plugin to use for
trajectory execution -->
 <param name="moveit_controller_manager"
value="moveit_simple_controller_manager/MoveItSimpleControllerManager"
/>
 <!-- Load controller list to the parameter server -->
 <rosparam file="$(find
moveit_robot_arm_sim)/config/simple_moveit_controllers.yaml" />
 <rosparam file="$(find
moveit_robot_arm_sim)/config/new_ros_controllers.yaml" />
</launch>
```

4. หากพบปัญหาใดๆ ให้ใช้ไฟล์ที่พร้อมใช้งานได้จากที่นี่:

   https://github.com/ageofrobotics/Simulate_Your_Robot_Arm_In_ROS_Noetic/blob/main/simple_moveit_controller_manager.launch.xml

5. บันทึกและปิดไฟล์

4.3 สร้างไฟล์เปิดตัวใหม่เพื่อเปิดการจำลอง Moveit ใน Rviz และ Gazebo

Full YouTube Tutorial: https://youtu.be/eXntnMTr6tg

ทำตามขั้นตอนด้านล่างเพื่อสร้างไฟล์เรียกใช้งานเพื่อโหลดโหนดกลุ่ม moveit, Raviz และการจำลองศาลาของแขนหุ่นยนต์ของเรา:

1. เปิดเทอร์มินัล
2. ไปที่โฟลเดอร์เรียกใช้งานของแพ็คเกจ moveit ของแขนหุ่นยนต์

   $ cd ~/YourWorkSpaceName/src/YourMoveitPackageName/launch

สำหรับกรณีนี้คือ:

```
$ cd ~/moveit_ws/src/moveit_robot_arm_sim/launch
```

หากมีพื้นที่ทำงานและชื่อแพ็คเกจเดียวกัน เพียงใช้คำสั่งด้านบน

3. สร้างไฟล์ “full_robot_arm_sim.launch” ในโฟลเดอร์เรียกใช้งานของแพ็คเกจ URDF โดยใช้เทอร์มินัล

```
$ touch full_robot_arm_sim.launch
```

หมายเหตุ: สามารถตั้งชื่อไฟล์นี้ได้ตามต้องการ

4. เปิดไฟล์ “arm_urdf.launch” ในโปรแกรมแก้ไขข้อความ “gedit”

```
$ gedit full_robot_arm_sim.launch
```

5. คัดลอกและวางโค้ดด้านล่างลงในไฟล์

full_robot_arm_sim.launch

```
<launch>

 <!-- Launch Your robot arms launch file which loads the robot in
Gazebo and spawns the controllers -->
 <include file = "$(find robot_arm_urdf)/launch/arm_urdf.launch" />
 <!-- Launch Moveit Move Group Node -->
 <include file = "$(find
moveit_robot_arm_sim)/launch/move_group.launch" />
 <!-- Run Rviz and load the default configuration to see the state of
the move_group node -->
 <arg name="use_rviz" default="true" />
 <include file="$(find
moveit_robot_arm_sim)/launch/moveit_rviz.launch" if="$(arg use_rviz)">
 <arg name="rviz_config" value="$(find
moveit_robot_arm_sim)/launch/moveit.rviz"/>
 </include>
</launch>
```

6. หากคุณพบปัญหาใดๆ ให้ใช้ไฟล์ที่พร้อมใช้งานได้จากที่นี่:

https://github.com/ageofrobotics/Simulate_Your_Robot_Arm_In_ROS_Noetic/blob/main/full_robot_arm_sim.launch

7. บันทึกและปิดไฟล์

4.4 สร้าง catkin workspace

1. เปิดเทอร์มินัล

2. ไปที่พื้นที่ทำงานของคุณ

```
$ cd ~/moveit_ws
```

3. ซอร์สไฟล์ setup.bash ของพื้นที่ทำงาน catkin ของคุณ

```
$ source devel/setup.bash
```

4. บิวด์พื้นที่ทำงาน

```
$ catkin build
```

5. ซอร์สไฟล์ setup.bash อีกครั้ง

```
$ source devel/setup.bash
```

4.5 เปิดไฟล์เปิดตัว moveit ของหุ่นยนต์เพื่อโหลดการจำลองการเคลื่อนที่ด้วย Rviz และ Gazebo

Full YouTube Tutorial: https://youtu.be/eXntnMTr6tg

1. เปิดเทอร์มินัลและดำเนินการคำสั่งต่อไปนี้เพื่อเริ่ม ROS master

```
$ roscore
```

หลังจากรันคำสั่งนี้แล้ว ให้เปิดเทอร์มินัลนี้ไว้

2. เปิดเทอร์มินัลอื่น

3. ไปที่เวิร์กสเปซของคุณ

```
$ cd ~/moveit_ws
```

4. ซอร์สไฟล์ setup.bash ของเวิร์กสเปซ catkin ของคุณ

```
$ source devel/setup.bash
```

5. สร้างเวิร์กสเปซหากยังไม่ได้สร้างหลังจากทำการเปลี่ยนแปลง

```
$ catkin build
```

6. ซอร์สไฟล์ setup.bash อีกครั้ง

```
$ source devel/setup.bash
```

7. เปิดไฟล์ “arm_urdf.launch” ที่เราสร้างในขั้นตอนก่อนหน้า

```
$ roslaunch your_Moveit_package_name moveit_launch_file.launch
```

ในกรณีนี้จะเป็น:

```
$ roslaunch moveit_robot_arm_sim full_robot_arm_sim.launch
```

หากทำทุกอย่างถูกต้อง หุ่นยนต์จะเปิดใน Gazebo โดยไม่มีข้อผิดพลาดใดๆ ข้อผิดพลาดที่เกี่ยวข้องกับ PID เท่านั้นที่จะเกิดขึ้นได้ ไม่ควรมีข้อผิดพลาดอื่นๆ เกิดขึ้น นอกจากนี้ ตรวจสอบว่าตัวควบคุมทั้งหมดถูกสร้างขึ้นอย่างถูกต้องหรือไม่ และสามารถวางแผนการเคลื่อนที่ของหุ่นยนต์ตามที่ระบุในวิดีโอ: https://youtu.be/eXntnMTr6tg