รายงานผู้เชี่ยวชาญ: การพัฒนาระบบหุ่นยนต์ Vision สำหรับหยิบของตามสีแบบอัตโนมัติบน Raspberry Pi

บทสรุปสำหรับผู้บริหาร

รายงานฉบับนี้เป็นคู่มือผู้เชี่ยวชาญที่ครอบคลุมและละเอียดถี่ถ้วนสำหรับการพัฒนาระบบหุ่นยนต์แขนกลที่ขับเคลื่อนด้วยวิชันซิสเต็ม เพื่อการหยิบจับวัตถุตามสีโดยเฉพาะ โดยเน้นการติดตั้งใช้งานบนแพลตฟอร์ม Raspberry Pi รายงานนี้จะอธิบายหลักสูตรการเรียนรู้ของผู้ใช้งานอย่างเป็นระบบ ตั้งแต่แนวคิดพื้นฐานของ Robot Operating System (ROS) และการจัดการพื้นที่ทำงาน ไปจนถึงกลไกที่ซับซ้อนของแขนกล การควบคุมการเคลื่อนไหวขั้นสูงด้วย MoveIt! การรวมกล้องเข้ากับระบบ การประมวลผลภาพที่ซับซ้อนเพื่อตรวจจับสี และการปรับปรุงประสิทธิภาพที่สำคัญสำหรับการติดตั้งใช้งานบนระบบฝังตัวที่มีทรัพยากรจำกัด รายงานเน้นการนำไปใช้งานจริง โดยให้ข้อมูลเชิงลึกโดยละเอียดเกี่ยวกับสถาปัตยกรรมระบบ การเปรียบเทียบระหว่าง ROS 1 และ ROS 2 โซลูชันจลนศาสตร์ และกลยุทธ์เพื่อให้การทำงานอัตโนมัติมีความแข็งแกร่ง ซึ่งจะช่วยเตรียมความพร้อมให้ผู้อ่านสำหรับความท้าทายในโลกแห่งความเป็นจริงของวิทยาการหุ่นยนต์

1. บทนำสู่ระบบหุ่นยนต์หยิบจับอัตโนมัติที่ขับเคลื่อนด้วยวิชันซิสเต็ม

1.1 บริบทของโครงการ: ความท้าทายในการหยิบจับวัตถุตามสี

ในปัจจุบัน ความต้องการระบบอัตโนมัติในอุตสาหกรรมต่างๆ ตั้งแต่การผลิตและโลจิสติกส์ไปจนถึงการใช้งานในครัวเรือนกำลังเพิ่มขึ้นอย่างต่อเนื่อง การหยิบจับวัตถุตามสีเป็นงานพื้นฐานในวิทยาการหุ่นยนต์ที่ต้องอาศัยการประสานงานที่แม่นยำระหว่างการรับรู้และการจัดการ การเลือกใช้การหยิบจับตามสีช่วยลดความซับซ้อนของปัญหาการรับรู้เบื้องต้นเมื่อเทียบกับการจดจำวัตถุที่ซับซ้อน (เช่น การใช้การเรียนรู้เชิงลึก) ทำให้เป็นจุดเริ่มต้นที่เหมาะสมอย่างยิ่งสำหรับผู้เรียนในขณะที่ยังคงแสดงให้เห็นถึงขั้นตอนการทำงานที่สมบูรณ์ตั้งแต่การรับรู้ไปจนถึงการกระทำ [1] การเรียนรู้ในลักษณะนี้จะช่วยให้ผู้เรียนสร้างรากฐานที่มั่นคงสำหรับงานวิชันซิสเต็มที่ซับซ้อนยิ่งขึ้นในอนาคต การเรียนรู้การหยิบจับตามสีจะให้ความเข้าใจที่จำเป็นเกี่ยวกับสถาปัตยกรรมและทักษะการปฏิบัติก่อนที่จะจัดการกับงานวิชันซิสเต็มที่ซับซ้อนมากขึ้น เช่น การจดจำรูปร่าง การวิเคราะห์พื้นผิว หรือการตรวจจับวัตถุทั่วไปโดยใช้โครงข่าย

ประสาทเทียม

1.2 วัตถุประสงค์และขอบเขตของรายงาน

รายงานฉบับนี้มีวัตถุประสงค์เพื่อเป็นคู่มือผู้เชี่ยวชาญที่ครอบคลุมสำหรับการพัฒนาระบบหุ่นยนต์หยิบจับที่ขับเคลื่อนด้วยวิชันซิสเต็ม โดยนำทางผู้อ่านตลอดวงจรการพัฒนาทั้งหมดตั้งแต่ต้นจนจบ รายงานจะครอบคลุมหัวข้อที่ระบุไว้อย่างชัดเจน ได้แก่ พื้นฐานของ ROS กลไกของหุ่นยนต์ การควบคุมการเคลื่อนไหว การรวมกล้อง การประมวลผลภาพ และการติดตั้งใช้งานบน Raspberry Pi นอกจากนี้ยังเน้นการประยุกต์ใช้งานจริงและการเพิ่มประสิทธิภาพสำหรับระบบฝังตัว

1.3 ภาพรวมสถาปัตยกรรมระบบและการพึ่งพาอาศัยกัน

ระบบหุ่นยนต์หยิบจับที่ขับเคลื่อนด้วยวิชันซิสเต็มประกอบด้วยส่วนประกอบหลักหลายส่วนที่ทำงานร่วมกันอย่างเป็นระบบ ได้แก่ กล้อง (Camera) หน่วยประมวลผลภาพ (Image Processing Unit) ซึ่งในที่นี้คือ Raspberry Pi ตัวควบคุมแขนกล (Robotic Arm Controller) และตัวแขนกลเอง การทำงานของระบบเริ่มต้นจากการป้อนข้อมูลภาพจากกล้อง ซึ่งจะถูกส่งต่อไปยังหน่วยประมวลผลภาพเพื่อทำการวิเคราะห์และระบุวัตถุเป้าหมาย จากนั้นข้อมูลตำแหน่งของวัตถุจะถูกส่งไปยังตัวควบคุมแขนกลเพื่อวางแผนและสั่งการให้แขนกลเคลื่อนที่ไปยังตำแหน่งที่ถูกต้องเพื่อทำการหยิบจับ

การทำงานร่วมกันนี้อาศัยหลักการของ Robot Operating System (ROS) ซึ่งเป็นเฟรมเวิร์กที่ส่งเสริมการออกแบบแบบโมดูลาร์ [3] ความเป็นโมดูลาร์นี้เป็นสิ่งสำคัญอย่างยิ่งในการจัดการความซับซ้อนของระบบ หุ่นยนต์ที่ซับซ้อนสามารถแบ่งออกเป็นส่วนประกอบย่อยๆ ที่จัดการโดย "โหนด" (Nodes) ที่แตกต่างกันใน ROS ตัวอย่างเช่น โหนดหนึ่งอาจจัดการการรับภาพจากกล้อง อีกโหนดหนึ่งทำการประมวลผลภาพ และอีกโหนดหนึ่งควบคุมการเคลื่อนไหวของแขนกล การแยกส่วนประกอบเหล่านี้ช่วยให้แต่ละส่วนสามารถพัฒนาและทดสอบได้อย่างอิสระ ก่อนที่จะนำมารวมเข้าด้วยกันอย่างราบรื่น ซึ่งช่วยลดความซับซ้อนในการแก้ไขข้อผิดพลาดและส่งเสริมการนำโค้ดกลับมาใช้ใหม่ได้ สิ่งนี้มีความสำคัญอย่างยิ่งสำหรับโครงการของผู้เรียน เนื่องจากช่วยให้สามารถแยกปัญหาและแก้ไขได้อย่างมีประสิทธิภาพมากขึ้น

2. พื้นฐานของระบบปฏิบัติการหุ่นยนต์ (ROS)

2.1 สถาปัตยกรรม ROS: โหนด, หัวข้อ, บริการ, การกระทำ และ ROS Master/DDS

ROS (Robot Operating System) เป็นเฟรมเวิร์กโอเพนซอร์สที่รวมไลบรารีและเครื่องมือต่างๆ เพื่อช่วยให้นักพัฒนาสร้างแอปพลิเคชันหุ่นยนต์ได้ [4] ซึ่งครอบคลุมตั้งแต่การจัดการฮาร์ดแวร์ ไดรเวอร์อุปกรณ์ เครื่องมือแสดงภาพ การส่งผ่านข้อความ และการจัดการแพ็กเกจ [4] ROS ถือเป็นสภาพแวดล้อมที่จำเป็นสำหรับการทำงานด้านวิทยาการหุ่นยนต์ในปัจจุบัน [6]

สถาปัตยกรรมของ ROS แบ่งออกเป็นสามระดับหลัก:

- **ระดับระบบไฟล์ (Filesystem Level):**
  - **แพ็กเกจ (Packages):** เป็นหน่วยย่อยที่สุดของ ROS ซึ่งประกอบด้วยโหนด ไฟล์การกำหนดค่า และอื่นๆ [3]
  - **ไฟล์ Manifest ของแพ็กเกจ (package.xml):** ให้ข้อมูลเกี่ยวกับแพ็กเกจ เช่น ใบอนุญาต การพึ่งพา และแฟล็กการคอมไพล์ [3]
  - **เมตาแพ็กเกจ (Metapackages):** ใช้รวบรวมแพ็กเกจหลายๆ แพ็กเกจเข้าด้วยกันเป็นกลุ่ม เช่น navigation stack [3]
  - **ประเภทข้อความ (Message (msg) types):** กำหนดโครงสร้างข้อมูลสำหรับการสื่อสารระหว่างกระบวนการ [3]
- **ระดับกราฟการคำนวณ (Computation Graph Level):**
  - **โหนด (Nodes):** เป็นกระบวนการที่ทำการคำนวณ [3] โดยทั่วไปแล้ว ระบบหนึ่งจะมีโหนดจำนวนมากเพื่อควบคุมฟังก์ชันต่างๆ แยกกัน การมีโหนดขนาดเล็กจำนวนมากที่ทำงานเพียงฟังก์ชันเดียวมักจะดีกว่าการมีโหนดขนาดใหญ่ที่ทำงานทุกอย่างในระบบ [3] โหนดถูกเขียนด้วยไลบรารีไคลเอนต์ ROS เช่น
    roscpp หรือ rospy [3]
  - **Master (The Master):** ทำหน้าที่ลงทะเบียนชื่อและให้บริการค้นหาแก่โหนดอื่นๆ รวมถึงตั้งค่าการเชื่อมต่อระหว่างโหนด [3] หากไม่มี Master โหนดจะไม่สามารถสื่อสารกันได้ ในระบบแบบกระจาย Master สามารถอยู่ในคอมพิวเตอร์เครื่องหนึ่ง และโหนดสามารถทำงานในคอมพิวเตอร์เครื่องเดียวกันหรือเครื่องอื่นๆ ได้ [3]
  - **เซิร์ฟเวอร์พารามิเตอร์ (Parameter Server):** ช่วยให้สามารถจัดเก็บข้อมูลโดยใช้คีย์ในตำแหน่งส่วนกลาง [3] ด้วยพารามิเตอร์นี้ สามารถกำหนดค่าโหนดขณะทำงานหรือเปลี่ยนพารามิเตอร์การทำงานของโหนดได้ [3]
  - **ข้อความ (Messages):** โหนดสื่อสารกันผ่านข้อความ ซึ่งมีข้อมูลที่ให้ข้อมูลแก่โหนดอื่นๆ [3]
  - **หัวข้อ (Topics):** แต่ละข้อความต้องมีชื่อเพื่อส่งผ่านเครือข่าย ROS [3] เมื่อโหนดส่งข้อมูล เราเรียกว่าโหนดกำลัง "เผยแพร่หัวข้อ" (publishing a topic) [3] โหนดสามารถรับหัวข้อจากโหนดอื่นได้โดยการ "สมัครรับหัวข้อ" (subscribing to the topic) [3] โหนดสามารถสมัครรับหัวข้อได้แม้ว่าจะไม่มีโหนดอื่นกำลังเผยแพร่หัวข้อนั้นอยู่ก็ตาม [3] สิ่งนี้ช่วยให้สามารถ "แยกส่วนการผลิตจากการบริโภค" (decouple the production from the consumption) ซึ่ง

เป็นคุณสมบัติที่สำคัญอย่างยิ่งในการออกแบบระบบหุ่นยนต์ [3] การแยกส่วนนี้หมายความว่าโหนดที่ผลิตข้อมูล (เช่น โหนดกล้องที่เผยแพร่ภาพ) ไม่จำเป็นต้องรู้ว่าใครจะใช้ข้อมูลนั้น และโหนดที่บริโภคข้อมูล (เช่น โหนดประมวลผลภาพที่สมัครรับภาพ) ก็ไม่จำเป็นต้องรู้ว่าใครเป็นผู้ผลิตข้อมูล [3] ความยืดหยุ่นนี้เป็นกุญแจสำคัญในการปรับขนาดและกำหนดค่าระบบหุ่นยนต์ใหม่ได้อย่างง่ายดาย [3] ชื่อหัวข้อต้องไม่ซ้ำกันเพื่อหลีกเลี่ยงปัญหาและความสับสน [3]
    - **บริการ (Services):** เมื่อต้องการร้องขอหรือรับคำตอบจากโหนดหนึ่ง โหนดสามารถใช้บริการได้ [3] บริการช่วยให้สามารถโต้ตอบกับโหนดได้ในลักษณะแบบขอ-ตอบ (request-reply) [3] บริการก็ต้องมีชื่อที่ไม่ซ้ำกันเช่นกัน [3]
    - **ถุงข้อมูล (Bags):** เป็นรูปแบบสำหรับบันทึกและเล่นข้อมูลข้อความ ROS ซ้ำ [3] ถุงข้อมูลเป็นกลไกสำคัญในการจัดเก็บข้อมูล เช่น ข้อมูลเซ็นเซอร์ ซึ่งอาจเก็บรวบรวมได้ยากแต่จำเป็นสำหรับการพัฒนาและทดสอบอัลกอริทึม [3]
- **ระดับชุมชน (Community Level):** เน้นลักษณะโอเพนซอร์ส เอกสารประกอบ [4] ฟอรัม [5] และทรัพยากรที่ใช้ร่วมกันซึ่งส่งเสริมการพัฒนา

ตาราง 2.1: กลไกการสื่อสารหลักของ ROS

กลไก	คำอธิบายโดยย่อ	รูปแบบการสื่อสาร	กรณีการใช้งานทั่วไป
โหนด (Node)	กระบวนการที่ทำการคำนวณในระบบ ROS [3]	-	ส่วนประกอบของซอฟต์แวร์ที่ทำงานเฉพาะอย่าง เช่น ไดรเวอร์กล้อง, ตัวควบคุมมอเตอร์
หัวข้อ (Topic)	ช่องทางที่มีชื่อสำหรับการส่งข้อความแบบหลาย-ต่อ-หลาย [3]	หนึ่ง-ต่อ-หลาย, หลาย-ต่อ-หนึ่ง	ข้อมูลเซ็นเซอร์ (ภาพ, LIDAR), สถานะหุ่นยนต์ (ตำแหน่งข้อต่อ), คำสั่งควบคุมต่อเนื่อง [3]
บริการ (Service)	กลไกแบบขอ-ตอบสำหรับการโต้ตอบแบบหนึ่ง-ต่อ-หนึ่ง [3]	หนึ่ง-ต่อ-หนึ่ง (ขอ-ตอบ)	การเรียกใช้ฟังก์ชันแบบไม่ต่อเนื่อง เช่น "รับตำแหน่งวัตถุ", "เปิด/ปิดกริปเปอร์" [3]
พารามิเตอร์ (Parameter)	การจัดเก็บข้อมูลแบบคีย์-ค่าในส่วนกลาง [3]	-	การกำหนดค่าโหนดขณะทำงาน, การปรับค่า PID [3]
การกระทำ (Action)	กลไกแบบขอ-ตอบที่สามารถถูกยกเลิกได้และให้ข้อเสนอแนะความคืบหน้า (ใน ROS 2)	หนึ่ง-ต่อ-หนึ่ง (เป้าหมาย-ผลลัพธ์พร้อมข้อเสนอแนะ)	งานที่ใช้เวลานาน เช่น การเคลื่อนที่ไปยังเป้าหมายที่ซับซ้อน, การหยิบจับวัตถุ

2.2 พื้นที่ทำงาน ROS และการจัดการแพ็กเกจ: Catkin และ Colcon

พื้นที่ทำงาน (Workspaces) คือชุดของไดเรกทอรีที่ใช้จัดเก็บโค้ด ROS [8] ใน ROS 1 จะเรียกว่า

catkin workspaces และมักจะตั้งชื่อว่า catkin_ws [8] แพ็กเกจ ROS ที่ผู้ใช้สร้างขึ้นทั้งหมดจะต้องอยู่ในพื้นที่ทำงาน Catkin [8]

การสร้างพื้นที่ทำงาน Catkin:

1. mkdir -p ~/catkin_ws/src: สร้างไดเรกทอรี src ภายใน catkin_ws [8]
2. cd ~/catkin_ws/: นำทางไปยังรากของพื้นที่ทำงาน [8]
3. catkin_make: สร้างพื้นที่ทำงาน ซึ่งจะสร้างโฟลเดอร์ build และ devel รวมถึงไฟล์ setup.*sh [8]

การตั้งค่าสภาพแวดล้อม:
หลังจากสร้างแล้ว จำเป็นต้อง source devel/setup.bash เพื่อตั้งค่าพาธให้สามารถค้นหาแพ็กเกจและโค้ดได้ 8 เพื่อหลีกเลี่ยงการ
source ซ้ำทุกครั้งที่เปิดเทอร์มินัลใหม่ ควรสั่งให้เพิ่มบรรทัด source ~/catkin_ws/devel/setup.bash ไปยังไฟล์ ~/.bashrc [8] การดำเนินการนี้เป็นแนวปฏิบัติที่สำคัญอย่างยิ่ง ซึ่งมักถูกมองข้ามโดยผู้เริ่มต้น หากไม่ดำเนินการนี้จะนำไปสู่ข้อผิดพลาด "command not found" และความหงุดหงิดอย่างมาก ซึ่งเน้นย้ำถึงความสำคัญของการจัดการสภาพแวดล้อมที่เหมาะสมใน ROS [8] สามารถตรวจสอบการตั้งค่าสภาพแวดล้อมได้โดยใช้คำสั่ง

echo $ROS_PACKAGE_PATH หรือ printenv | grep ROS [8]

Colcon:
Colcon เป็นเครื่องมือสร้างที่ใช้ใน ROS 2 11 ใช้สำหรับสร้างไฟล์และนำการเปลี่ยนแปลงไปใช้ 11 คำสั่ง colcon build --symlink-install เป็นที่นิยมใช้สำหรับโครงการขนาดใหญ่ [11]

2.3 ROS 1 กับ ROS 2: ทางเลือกเชิงกลยุทธ์สำหรับโครงการหุ่นยนต์สมัยใหม่

การตัดสินใจเลือกระหว่าง ROS 1 และ ROS 2 เป็นสิ่งสำคัญสำหรับโครงการหุ่นยนต์ใหม่ๆ เนื่องจากแต่ละเวอร์ชันมีข้อดีและข้อจำกัดที่แตกต่างกัน

ภาพรวม ROS 1 (Noetic):
ROS 1 ยังคงมีการใช้งานอย่างแพร่หลาย โดยเฉพาะสำหรับผู้เริ่มต้น เนื่องจากมีไลบรารีและโหนดจำนวนมาก 13 อย่างไรก็ตาม ROS 1 Noetic จะสิ้นสุดการสนับสนุน (End-of-Life, EOL) ในเดือนพฤษภาคม 2568 14

หลังจากนั้นจะได้รับการแก้ไขข้อผิดพลาดเท่านั้น ไม่มีการพัฒนาคุณสมบัติใหม่ 15 ROS 1 ได้รับการทดสอบหลักบน Ubuntu และใช้ C++03 และ Python 2 เป็นหลัก 17 ไม่ได้ออกแบบมาสำหรับระบบเรียลไทม์โดยเฉพาะ แม้ว่าจะสามารถทำงานได้รวดเร็ว 16 ใช้รูปแบบการจัดเรียงข้อมูล (serialization) โปรโตคอลการส่งข้อมูล และกลไกการค้นหาแบบรวมศูนย์ที่เป็นแบบกำหนดเอง 17 และใช้ระบบสร้าง
catkin ซึ่งสามารถสร้างแบบไม่แยกส่วนได้ [17]

ภาพรวม ROS 2 (Humble/Iron):
ROS 2 ได้รับการแนะนำสำหรับโครงการใหม่ๆ เนื่องจากมีการพัฒนาอย่างต่อเนื่องและรองรับในระยะยาว 14 โดยมีเวอร์ชัน Long-Term Support (LTS) เช่น Humble Hawkbill 14 ROS 2 ได้รับการออกแบบโดยคำนึงถึงความสามารถแบบเรียลไทม์ แม้ว่าจะไม่ใช่ค่าเริ่มต้น 16 รองรับหลายแพลตฟอร์ม (Ubuntu, OS X, Windows 10) 17 ใช้ C++11 (และอาจใช้ C++17 ในอนาคต) อย่างกว้างขวาง และ Python 3.5 ขึ้นไป 17 สิ่งสำคัญคือ ROS 2 ใช้ DDS (Data Distribution Service) สำหรับอินเทอร์เฟซมิดเดิลแวร์แบบนามธรรม ซึ่งช่วยให้สามารถกำหนดนโยบายคุณภาพบริการ (Quality of Service, QoS) ต่างๆ เพื่อปรับปรุงการสื่อสารผ่านเครือข่ายที่แตกต่างกัน 17 การเปลี่ยนผ่านจากมิดเดิลแวร์แบบกำหนดเองของ ROS 1 ไปสู่สถาปัตยกรรมแบบ DDS ของ ROS 2 เป็นการเปลี่ยนแปลงสถาปัตยกรรมพื้นฐานที่มีผลกระทบอย่างมากต่อการสื่อสารเครือข่าย ประสิทธิภาพแบบเรียลไทม์ และการรองรับหลายแพลตฟอร์ม ซึ่งหมายถึงรากฐานที่แข็งแกร่งและปรับขนาดได้มากขึ้นสำหรับวิทยาการหุ่นยนต์ในอนาคต 17 ROS 2 ใช้ระบบสร้าง
colcon ซึ่งรองรับการสร้างแบบแยกส่วนและระบบสร้างอื่นๆ [17] คำสั่ง

colcon build --symlink-install เป็นที่นิยมใช้ [11] นอกจากนี้ยังรวมถึงการสนับสนุน

MoveIt2 และ Navigation2 [15] และ

micro-ROS เป็นข้อได้เปรียบที่สำคัญสำหรับระบบฝังตัว เช่น Raspberry Pi [16]

ทางเลือกเชิงกลยุทธ์สำหรับโครงการนี้:
เนื่องจาก ROS 1 Noetic จะสิ้นสุดการสนับสนุนในปี 2568 14 การเริ่มต้นด้วย ROS 2 จึงเป็นสิ่งสำคัญอย่างยิ่งสำหรับความยั่งยืนของโครงการในระยะยาวและการพัฒนาทักษะ 14 แม้ว่า ROS 1 อาจช่วยให้เริ่มต้นได้เร็วขึ้นสำหรับผู้เริ่มต้นอย่างแท้จริงเนื่องจากมีบทเรียนจำนวนมาก 13 แต่ความซับซ้อนเพิ่มเติมของการใช้ ROS 1 bridge เพื่อความเข้ากันได้กับ ROS 2 13 ทำให้การเริ่มต้นด้วย ROS 2 โดยตรงมีประสิทธิภาพมากกว่าสำหรับโครงการใหม่ ความสามารถของ ROS 2 ในการทำงานแบบเรียลไทม์ 16 และความเข้ากันได้กับ
micro-ROS [16] มีประโยชน์อย่างยิ่งสำหรับการติดตั้งใช้งานบน Raspberry Pi

ตาราง 2.2: การเปรียบเทียบคุณสมบัติ ROS 1 และ ROS 2 สำหรับระบบฝังตัว

คุณสมบัติ	ROS 1 (Noetic)	ROS 2 (Humble/Iron)	ความเกี่ยวข้องกับ Raspberry Pi/ระบบฝังตัว
แพลตฟอร์ม	Ubuntu เป็นหลัก, รองรับ Linux/OS X โดยชุมชน [17]	Ubuntu, OS X, Windows 10 [17]	ROS 2 รองรับ Raspberry Pi ได้ดีกว่า, มี micro-ROS [16]

ภาษา (C++/Python)	C++03, Python 2 [17]	C++11/14/17, Python 3.5+ [17]	Python 3 ใน ROS 2 เป็นมาตรฐานที่ทันสมัยกว่า
มิดเดิลแวร์	กำหนดเอง (Custom) [17]	DDS (Data Distribution Service) [17]	DDS ให้ QoS (Quality of Service) ที่ดีกว่าสำหรับการสื่อสารบนเครือข่ายจำกัดทรัพยากร
ระบบสร้าง	Catkin [17]	Colcon [17]	Colcon มีความยืดหยุ่นและประสิทธิภาพที่ดีขึ้นในการสร้างโครงการขนาดใหญ่ [12]
ความสามารถเรียลไทม์	ไม่ใช่เรียลไทม์ (Soft real-time) [16]	ออกแบบมาสำหรับเรียลไทม์ (ต้องตั้งค่า) [16]	สำคัญสำหรับงานควบคุมที่ต้องการความแม่นยำสูงบนฮาร์ดแวร์จำกัด
การสนับสนุน/EOL	EOL พ.ค. 2568 (บำรุงรักษาเท่านั้น) [14]	LTS (Long-Term Support) สำหรับ Humble, พัฒนาต่อเนื่อง [14]	แนะนำ ROS 2 สำหรับความยั่งยืนของโครงการระยะยาว [14]
ระบบหลายหุ่นยนต์/กระจาย	ทำได้แต่ซับซ้อนกว่า	ออกแบบมาเพื่อรองรับโดยตรง [17]	เหมาะสำหรับการแบ่งงานประมวลผลระหว่าง Pi และคอมพิวเตอร์หลัก [19]
รองรับ Windows/macOS	รองรับโดยชุมชน/ผ่าน WSL [13]	รองรับอย่างเป็นทางการ [17]	เพิ่มความยืดหยุ่นในการพัฒนาบนเครื่องคอมพิวเตอร์ส่วนตัว
Micro-ROS	ไม่มี	มี [16]	ช่วยให้ ROS ทำงานบนไมโครคอนโทรลเลอร์ได้โดยตรง ลดภาระ Pi

3. กลไกแขนกลและจลนศาสตร์

3.1 หลักการออกแบบกลไกสำหรับหุ่นยนต์หยิบจับ

การออกแบบกลไกของหุ่นยนต์หยิบจับเป็นรากฐานสำคัญที่ส่งผลโดยตรงต่อความสามารถในการทำงานของระบบโดยรวม

- **องศาอิสระ (Degrees of Freedom, DOF):** โดยทั่วไปแล้ว แขนกล 6 องศาอิสระ (3 สำหรับตำแหน่ง และ 3 สำหรับการวางแนว) มักจะจำเป็นสำหรับงานหยิบและวางที่หลากหลาย [20] การมี DOF ที่เพียงพอช่วยให้หุ่นยนต์สามารถเข้าถึงตำแหน่งและทิศทางต่างๆ ได้อย่างยืดหยุ่น การทำงานร่วมกันระหว่างการออกแบบกลไกและความสามารถของซอฟต์แวร์เป็นสิ่งสำคัญอย่างยิ่ง แขนกลที่ออกแบบมาอย่างดี (เช่น มี DOF เพียงพอ, รับน้ำหนักได้เหมาะสม) จะช่วยให้ซอฟต์แวร์ (เช่น การวางแผนการเคลื่อนไหว) ทำงานได้ง่ายขึ้น ในทางกลับกัน ข้อจำกัดของซอฟต์แวร์ (เช่น ข้อจำกัดของตัวแก้ IK) สามารถเผยให้เห็นจุดอ่อนในการออกแบบกลไก (เช่น การเข้าถึงที่ไม่เพียงพอในบางทิศทาง) [21]
- **ระยะเอื้อมและขีดความสามารถในการรับน้ำหนัก (Reach and Payload Capacity):** พารามิเตอร์เหล่านี้มีความสำคัญอย่างยิ่งในการออกแบบ โดยขึ้นอยู่กับลักษณะการใช้งาน [20] ตัวอย่าง เช่น แขนกล Arctos มีระยะเอื้อม 600 มม. และสามารถรับน้ำหนักได้ 1 กก. [20]
- **ความแข็งแกร่งของโครงสร้างและการเลือกวัสดุ (Structural Rigidity and Material Choice):** การใช้วัสดุที่แข็งแรง (เช่น อะคริลิก ไม้ หรือชิ้นส่วนที่พิมพ์ด้วย 3D) เป็นสิ่งสำคัญในการลดการสั่นสะเทือนและการโก่งตัว [20]
- **การออกแบบกริปเปอร์ (Gripper Design):** การเลือกกริปเปอร์ที่เหมาะสมกับวัตถุที่จะหยิบจับเป็นสิ่งสำคัญ [22]

3.2 การเลือกแอคทูเอเตอร์และการวิเคราะห์แรงบิดสำหรับข้อต่อหุ่นยนต์

การเลือกประเภทมอเตอร์ที่เหมาะสมเป็นสิ่งสำคัญสำหรับประสิทธิภาพของแขนกล

- **การเปรียบเทียบประเภทมอเตอร์:**
    - **เซอร์โวมอเตอร์ (Servo Motors):** ได้รับการแนะนำสำหรับแขนกลเนื่องจากสามารถควบคุมมุมได้อย่างแม่นยำ มีเอนโค้ดเดอร์ในตัวสำหรับป้อนกลับข้อมูล และมีอัตราส่วนแรงบิดต่อราคาที่เหมาะสม [22] เป็นระบบวงปิดที่สามารถแก้ไขความคลาดเคลื่อนได้ [22] การเลือกใช้เซอร์โวมอเตอร์ที่มีการป้อนกลับข้อมูล [22] มีผลโดยตรงต่อความเป็นไปได้ในการนำเฟรมเวิร์กควบคุม ROS ขั้นสูง เช่น ros_control และ MoveIt! มาใช้ หากไม่มีข้อมูลสถานะข้อต่อที่แม่นยำ การดำเนินการวิถีการเคลื่อนที่ที่แม่นยำและการหลีกเลี่ยงการชนจะทำได้ยากขึ้นมากหรือเป็นไปไม่ได้ [12]
    - **มอเตอร์ DC มาตรฐาน (Standard DC Motors):** ไม่เหมาะเนื่องจากควบคุมมุมและความเร็วได้ไม่ดี และมีแรงบิดต่ำที่ความเร็วต่ำ [22]

    - **สเต็ปเปอร์มอเตอร์ (Stepper Motors):** ให้การเคลื่อนที่แบบขั้นบันได แต่ขาดระบบป้อนกลับ ทำให้เกิดข้อผิดพลาดในการวางตำแหน่งได้เนื่องจากความเฉื่อย [22]
- **หลักการวิเคราะห์แรงบิด:**
    - **สูตร:** แรงบิด (T) คำนวณจาก T=F·L (แรงคูณระยะทางตั้งฉาก) หรือ T=m·L เมื่อแรงบิดมีหน่วยเป็น kg.cm [22]
    - **กรณีวิกฤต:** การคำนวณแรงบิดควรพิจารณาแขนกลในตำแหน่งแนวนอน เนื่องจากจะทำให้เกิดระยะทางตั้งฉากสูงสุดและทำให้ความต้องการแรงบิดสูงสุด [22]
    - **ปัจจัยความปลอดภัย (Safety Factor):** ควรใช้ปัจจัยความปลอดภัย (เช่น 1.5) เพื่อให้การเคลื่อนที่ราบรื่นโดยไม่มีการสั่นสะเทือน [22]
    - **วิธีการคำนวณ:** แรงบิดที่แต่ละมอเตอร์คือผลรวมของแรงบิดที่เกิดจากมวลรวมของมอเตอร์และข้อต่อที่อยู่ก่อนหน้า [22]
- **แหล่งจ่ายไฟ (Power Supply):** ควรใช้แหล่งจ่ายไฟภายนอกสำหรับมอเตอร์ เนื่องจาก Arduino หรือ Raspberry Pi ไม่สามารถจ่ายกระแสไฟฟ้าได้เพียงพอ และต้องแน่ใจว่ามีการต่อสายดินร่วมกัน [22]

3.3 ความเข้าใจในจลนศาสตร์ไปข้างหน้าและจลนศาสตร์ผกผัน

- **จลนศาสตร์ไปข้างหน้า (Forward Kinematics, FK):**
    - **คำจำกัดความ:** การหาตำแหน่งและการวางแนวของปลายแขนกล (end-effector) เมื่อทราบพารามิเตอร์ข้อต่อทั้งหมด [25]
    - **วัตถุประสงค์:** ใช้เพื่อทำความเข้าใจว่าปลายแขนกลของหุ่นยนต์อยู่ที่ใดในปัจจุบัน โดยอิงจากมุมข้อต่อของมัน
- **จลนศาสตร์ผกผัน (Inverse Kinematics, IK):**
    - **คำจำกัดความ:** การหาสภาพข้อต่อ (มุม) ที่จำเป็นเพื่อให้ปลายแขนกลไปถึงตำแหน่งและการวางแนวเป้าหมายที่ต้องการ [25]
    - **วัตถุประสงค์:** สำคัญสำหรับการควบคุมในพื้นที่งาน (task-space control) เช่น การเคลื่อนย้ายกริปเปอร์ไปยังจุดเฉพาะในอวกาศเพื่อหยิบจับ
    - **ความท้าทาย:** อาจต้องใช้การคำนวณที่เข้มข้น มีหลายโซลูชัน (redundancy) หรือไม่มีโซลูชันเลย (unreachable pose/singularity) [21] ความยากของ IK โดยเฉพาะสำหรับแขนกลที่มี DOF สูงขึ้นและในสภาวะที่มีจุดเอกฐาน [27] เป็นเหตุผลสำคัญที่ทำให้เฟรมเวิร์กอย่าง MoveIt! เป็นสิ่งจำเป็น MoveIt! ช่วยลดความซับซ้อนนี้ ทำให้นักพัฒนาสามารถมุ่งเน้นไปที่การวางแผนระดับงาน แทนที่จะต้องคำนวณข้อต่อระดับต่ำ
- **แนวทาง IK ทั่วไป:**
    - **แนวทางเชิงพีชคณิต (Algebraic Approach):** การแก้สมการที่ได้จากการเทียบเมทริกซ์การแปลงเป้าหมายกับเมทริกซ์จลนศาสตร์ไปข้างหน้าของหุ่นยนต์ [25]
    - **แนวทางเชิงเรขาคณิต (Geometric Approach):** การหาสมการตรีโกณมิติโดยการสังเกตโครงสร้างทางกายภาพของหุ่นยนต์/แขนกล [25] มีประโยชน์สำหรับแขนกลที่เรียบง่าย หรือใช้

ร่วมกับการแยกส่วนจลนศาสตร์ (kinematic decoupling) สำหรับ DOF ที่สูงขึ้น [25]
    - **เทคนิค Jacobian Inverse (Jacobian Inverse Technique):** วิธีที่ใช้กันอย่างแพร่หลาย ซึ่งเกี่ยวข้องกับการคำนวณและผกผันเมทริกซ์ Jacobian [25] สิ่งนี้เป็นพื้นฐานสำหรับจลนศาสตร์เชิงอนุพันธ์ (differential kinematics) และการควบคุมแบบคาร์ทีเซียน [27]
    - **CCD (Cyclic Coordinate Descent):** แนวทางที่ใช้การปรับให้เหมาะสม (optimization-based) โดยการปรับข้อต่อทีละน้อยจากปลายแขนกลไปยังโคนแขนกล เพื่อให้ปลายแขนกลเข้าใกล้เป้าหมายมากที่สุด [25]
- **บทบาทของ MoveIt!:** MoveIt! มีอินเทอร์เฟซสำหรับตัวแก้ IK ซึ่งจัดการการคำนวณตรีโกณมิติที่ซับซ้อน [23] สามารถใช้ตัวแก้เช่น IKfast ได้ [21]

ตาราง 3.1: ภาพรวมจลนศาสตร์ของแขนกลหุ่นยนต์

แนวคิด	คำจำกัดความ	วัตถุประสงค์	อินพุต/เอาต์พุต	วิธีการ/อัลกอริทึมทั่วไป	ความท้าทาย/ข้อควรพิจารณา
จลนศาสตร์ไปข้างหน้า (FK)	การหาตำแหน่งและการวางแนวของปลายแขนกล เมื่อทราบพารามิเตอร์ข้อต่อทั้งหมด [25]	การทำความเข้าใจตำแหน่งปัจจุบันของปลายแขนกล	อินพุต: มุมข้อต่อ; เอาต์พุต: ตำแหน่ง/การวางแนวปลายแขนกล	การแปลงโฮโมจีเนียส (Homogeneous Transformation), D-H Parameters	-
จลนศาสตร์ผกผัน (IK)	การหามุมข้อต่อที่จำเป็นเพื่อให้ปลายแขนกลไปถึงตำแหน่งและการวางแนวเป้าหมายที่ต้องการ [25]	การควบคุมปลายแขนกลไปยังตำแหน่งเฉพาะในพื้นที่งาน	อินพุต: ตำแหน่ง/การวางแนวปลายแขนกล; เอาต์พุต: มุมข้อต่อ	พีชคณิต, เรขาคณิต, Jacobian Inverse, CCD [25]	จุดเอกฐาน (Singularities), หลายโซลูชัน, การคำนวณที่ซับซ้อน [21]

4. การเคลื่อนที่และการควบคุมหุ่นยนต์ด้วย ROS

4.1 เฟรมเวิร์ก ROS Control: มาตรฐานสำหรับการสั่งการหุ่นยนต์

ros_control (สำหรับ ROS 1) และ ros2_control (สำหรับ ROS 2) เป็นแพ็กเกจมาตรฐานสำหรับอินเทอร์เฟซตัวควบคุม ซึ่งช่วยให้หุ่นยนต์สามารถเคลื่อนที่และทำงานต่างๆ ได้ [1][2] เฟรมเวิร์กเหล่านี้ช่วยลดความซับซ้อนของฮาร์ดแวร์

ส่วนประกอบหลัก:

- **อินเทอร์เฟซฮาร์ดแวร์ (hardware_interface::RobotHW):** ให้การเข้าถึงระดับ API เพื่ออ่านและสั่งการคุณสมบัติข้อต่อจาก/ไปยังหุ่นยนต์จริง [1][2] ตัวอย่างเช่น JointStateInterface, EffortJointInterface, VelocityJointInterface, PositionJointInterface [2][4]
- **ตัวควบคุม (Controllers):** รับข้อความ ROS (เช่น คำสั่ง Joint Trajectory) ประมวลผล และส่งคำสั่งไปยังอินเทอร์เฟซฮาร์ดแวร์ [1][2] ตัวอย่างเช่น joint_state_controller, joint_position_controller [2][4]
- **ตัวจัดการตัวควบคุม (Controller Manager):** จัดการการโหลด การเริ่มต้น การหยุด และการสลับตัวควบคุม [2][4]
- **การส่งกำลัง (Transmissions):** เชื่อมโยงแอคทูเอเตอร์กับข้อต่อใน URDF กำหนดวิธีการส่งแรง/ความเร็ว/ตำแหน่ง [2][4]

การตั้งค่าในสภาพแวดล้อมจำลอง (Gazebo):

- เพิ่มองค์ประกอบ <transmission> ไปยัง URDF [2][4]
- เพิ่มปลั๊กอิน gazebo_ros_control ไปยัง URDF เพื่อแยกวิเคราะห์แท็ก transmission และโหลดอินเทอร์เฟซฮาร์ดแวร์/ตัวจัดการตัวควบคุม [2][4]
- สร้างไฟล์การกำหนดค่า .yaml สำหรับค่า PID และการตั้งค่าตัวควบคุม [2][4]
- สร้างไฟล์ roslaunch เพื่อโหลดตัวควบคุมผ่านโหนด controller_spawner [2][4]

การควบคุมด้วยตนเองและการปรับแต่ง: สามารถส่งคำสั่งด้วยตนเองผ่าน rostopic pub หรือใช้ปลั๊กอิน RQT (Message Publisher, Plot, Dynamic Reconfigure) สำหรับการแสดงภาพและการปรับแต่ง PID [2][4]

ชั้นนามธรรมของ ros_control มีความสำคัญอย่างยิ่ง [1][2] ซึ่งช่วยให้โค้ดควบคุมระดับสูงเดียวกัน (เช่น จาก MoveIt!) สามารถทำงานกับฮาร์ดแวร์หุ่นยนต์ที่แตกต่างกันได้ เพียงแค่เปลี่ยนการใช้งานอินเทอร์เฟซฮาร์ดแวร์ สิ่งนี้ส่งเสริมการนำโค้ดกลับมาใช้ใหม่และลดความซับซ้อนในการรวมระบบกับหุ่นยนต์ใหม่ๆ การออกแบบนี้ช่วยลดเวลาและความพยายามในการพัฒนาเมื่อต้องย้ายโค้ดไปยังแพลตฟอร์มหุ่นยนต์ใหม่หรืออัปเกรดฮาร์ดแวร์ ซึ่งเป็นปัจจัยสำคัญที่ทำให้ ROS สามารถทำงานแบบ "plug-and-play" ได้

4.2 การควบคุมในพื้นที่ข้อต่อเทียบกับพื้นที่คาร์ทีเซียนสำหรับการจัดการ

การเลือกระหว่างการควบคุมในพื้นที่ข้อต่อ (Joint Space) และพื้นที่คาร์ทีเซียน (Cartesian Space) เป็นการตัดสินใจพื้นฐานในการออกแบบที่ขับเคลื่อนโดยความต้องการของงาน

- **การวางแผนในพื้นที่ข้อต่อ (Joint Space Planning):**
  - **คำจำกัดความ:** เป้าหมายถูกกำหนดเป็นตำแหน่งข้อต่อที่แน่นอน (เช่น ข้อต่อ 1 = 90 องศา) [26]
  - **ข้อดี:** พลวัตมักจะง่ายกว่า รับประกันเสถียรภาพและประสิทธิภาพด้วยการกำหนดสูตร Lagrangian [27] การเคลื่อนที่ในพื้นที่ข้อต่อสามารถคาดเดาได้
  - **ข้อเสีย:** วิถีการเคลื่อนที่ของปลายแขนกลในพื้นที่คาร์ทีเซียนอาจคาดเดาไม่ได้หรือไม่เป็นเชิงเส้น ทำให้งานที่ต้องการการเคลื่อนที่แบบเส้นตรงทำได้ยาก
- **การวางแผนในพื้นที่คาร์ทีเซียน (Cartesian Space Planning):**
  - **คำจำกัดความ:** เป้าหมายถูกกำหนดโดยตำแหน่ง (position) และการวางแนว (orientation) ที่ต้องการของปลายแขนกล [26]
  - **ข้อดี:** ใช้งานง่ายสำหรับผู้ปฏิบัติงาน (เช่น "ย้ายกริปเปอร์ไปที่จุดนี้") อนุญาตให้มีข้อจำกัดของเส้นทาง (เช่น ให้ปลายแขนกลตั้งตรง, ลากเส้นทางการเชื่อม) [26] มีประโยชน์สำหรับงานหยิบและวาง, การประกอบ, การเชื่อม, การพ่นสี [30]
  - **ข้อเสีย:**
    - **ความสมบูรณ์ (Completeness):** ตัววางแผนคาร์ทีเซียนที่ใช้ Jacobian มักจะไม่สมบูรณ์ อาจไม่พบโซลูชันแม้ว่าจะมีอยู่จริง โดยติดอยู่ในจุดต่ำสุดเฉพาะที่ (joint limits, singularities) [26]
    - **จุดเอกฐาน (Singularities):** การกำหนดค่าหุ่นยนต์ที่ Jacobian สูญเสียอันดับ ทำให้ไม่สามารถเคลื่อนย้ายปลายแขนกลในบางทิศทางได้โดยไม่ต้องใช้ความเร็วข้อต่อที่สูงมาก [27]
    - **การวางแผนแบบไม่จำกัด (Underconstrained Planning):** ตัววางแผนบางตัวอนุญาตให้ระบุการวางแนวปลายแขนกลบางส่วน ซึ่งช่วยเพิ่มพื้นที่ทำงานสำหรับงานที่ยืดหยุ่น [26]
    - **เรียลไทม์ (Real-time):** อาจขัดแย้งกับการวางแผนล่วงหน้า เนื่องจากต้องมีการคำนวณในอนาคตมากขึ้น ซึ่งลดการตอบสนองแบบเรียลไทม์ [26]
- **การรวม MoveIt!:** MoveIt! รองรับการวางแผนเส้นทางทั้งในพื้นที่ข้อต่อและพื้นที่คาร์ทีเซียน [30]

การเลือกใช้การควบคุมในพื้นที่ข้อต่อหรือพื้นที่คาร์ทีเซียนเป็นการตัดสินใจในการออกแบบพื้นฐานที่ขึ้นอยู่กับความต้องการของงาน [26] ในขณะที่การควบคุมแบบคาร์ทีเซียนใช้งานง่ายกว่าสำหรับงานที่เน้นมนุษย์ (เช่น การหยิบจับ) แต่ความอ่อนไหวต่อจุดเอกฐานและจุดต่ำสุดเฉพาะที่ [26] ทำให้จำเป็นต้องมีอัลกอริทึมการวางแผนที่แข็งแกร่ง (เช่น ที่มีอยู่ใน MoveIt!) และการพิจารณาขีดจำกัดจลนศาสตร์ของหุ่นยนต์อย่างรอบคอบ สิ่งนี้แสดงให้เห็นว่าแม้เครื่องมือระดับสูงจะช่วยลดความซับซ้อนในการพัฒนา แต่ความเข้าใจอย่างลึกซึ้งในหลักการหุ่นยนต์พื้นฐาน (จลนศาสตร์, จุดเอกฐาน) ยังคงมีความสำคัญ

ต่อการออกแบบระบบอัตโนมัติที่เชื่อถือได้และแข็งแกร่ง

4.3 การใช้ประโยชน์จาก MoveIt! สำหรับการวางแผนการเคลื่อนไหวขั้นสูง

MoveIt! เป็นซอฟต์แวร์ที่ทันสมัยสำหรับการจัดการหุ่นยนต์เคลื่อนที่ ซึ่งรวมการวางแผนการเคลื่อนไหว การจัดการ การรับรู้ 3 มิติ จลนศาสตร์ การควบคุม และการนำทาง [21] เป็นซอฟต์แวร์โอเพนซอร์สที่ใช้กันอย่างแพร่หลาย [5]

คุณสมบัติหลัก:

- **การวางแผนการเคลื่อนไหว (Motion Planning):** สร้างวิถีการเคลื่อนที่ที่มีองศาอิสระสูง หลีกเลี่ยงจุดต่ำสุดเฉพาะที่ [24]
- **จลนศาสตร์ผกผัน (Inverse Kinematics):** แก้หาตำแหน่งข้อต่อที่ต้องการเมื่อกำหนดท่าทาง [24]
- **การควบคุม (Control):** ดำเนินการวิถีการเคลื่อนที่ข้อต่อที่กำหนดเวลาไปยังฮาร์ดแวร์ระดับต่ำ [24]
- **การรับรู้ 3 มิติ (3D Perception):** เชื่อมต่อกับเซ็นเซอร์ความลึกและ Point Cloud ด้วย Octomaps สำหรับการตรวจสอบการชน [24]
- **การตรวจสอบการชน (Collision Checking):** หลีกเลี่ยงสิ่งกีดขวางโดยใช้รูปทรงเรขาคณิต, Meshes หรือข้อมูล Point Cloud [24]
- **เครื่องมือแสดงภาพแบบโต้ตอบ 3 มิติ (RViz):** แสดงการสาธิตภาพแบบสำเร็จรูป ช่วยให้สามารถทดลองอัลกอริทึมการวางแผนต่างๆ ได้ [24]
- **การรวม Gazebo Simulation:** รวมกับ Gazebo และ ros_control เพื่อสร้างแพลตฟอร์มการพัฒนาและทดสอบหุ่นยนต์ที่มีประสิทธิภาพ [24]
- **Setup Assistant:** ตัวช่วยสร้างการกำหนดค่าแบบทีละขั้นตอนที่ช่วยให้ผู้ใช้สามารถตั้งค่าหุ่นยนต์ใดๆ เพื่อทำงานกับ MoveIt! ได้อย่างรวดเร็ว [24]
- **MoveIt Task Constructor (MTC):** วิธีที่ยืดหยุ่นและโปร่งใสในการกำหนดและวางแผนการกระทำที่ประกอบด้วยงานย่อยที่พึ่งพาอาศัยกันหลายงาน [24]

การตั้งค่า MoveIt!:

- ติดตั้ง MoveIt! [34]
- สร้างไฟล์ URDF (Unified Robot Description Format) เพื่อกำหนดลิงก์และข้อต่อของหุ่นยนต์ [34]
- ใช้ MoveIt! Setup Assistant เพื่อสร้าง SRDF (Semantic Robot Description Format) และไฟล์การกำหนดค่าอื่นๆ [23] ซึ่งรวมถึงการกำหนดกลุ่มการวางแผน (เช่น arm_group_name) ปลายแขนกล และตัวแก้จลนศาสตร์ [23]
- แสดงภาพใน RViz โดยใช้ roslaunch urdf_tutorial display.launch model:=arm.urdf [34]

การใช้งาน API: สามารถควบคุมได้ด้วยกราฟิกผ่าน RViz (การลาก orb สีน้ำเงิน) หรือด้วยโปรแกรมผ่าน API [21]

การวางแผนเป้าหมาย: แปลง tf วัตถุจากลิงก์กล้องไปยัง base_footprint สำหรับ MoveIt! [21]

การเพิ่มโอกาสความสำเร็จ: ใช้ตัวแก้ IKfast [21] หรือลองเป้าหมายที่แตกต่างกันรอบๆ เป้าหมายเดิม [21]

MoveIt! ไม่ได้เป็นเพียงตัววางแผนการเคลื่อนไหว แต่เป็นระบบนิเวศที่ครอบคลุมซึ่งเชื่อมโยงการวางแผนงานระดับสูง (เช่น "หยิบวัตถุ") เข้ากับการควบคุมหุ่นยนต์ระดับต่ำ การรวม MoveIt! เข้ากับ URDF/SRDF การตรวจสอบการชน และการจำลอง (Gazebo/RViz) สร้างสภาพแวดล้อมการพัฒนาที่มีประสิทธิภาพสำหรับการจัดการที่ซับซ้อน [21] สำหรับหุ่นยนต์หยิบจับที่ขับเคลื่อนด้วยวิชันซิสเต็ม MoveIt! มีความสำคัญอย่างยิ่งในการจัดการ: การหลีกเลี่ยงการชน, การวางแผนเส้นทาง, IK และการจำลอง/การแสดงภาพ สิ่งนี้ช่วยให้ MoveIt! สามารถจัดการการคำนวณที่ซับซ้อนและปัญหาการชนได้อย่างมีประสิทธิภาพ ซึ่งช่วยเร่งการพัฒนาและทำให้พฤติกรรมของหุ่นยนต์มีความแข็งแกร่งมากขึ้น

4.4 การใช้ระบบควบคุมกริปเปอร์และการจัดการปลายแขนกล

การควบคุมกริปเปอร์เป็นส่วนสำคัญของระบบการหยิบจับที่แม่นยำ

- **การรวมกริปเปอร์ใน URDF/MoveIt!:** เพิ่มลิงก์และข้อต่อกริปเปอร์ไปยังไฟล์ URDF [23] อัปเดตการกำหนดค่า MoveIt! โดยใช้ Setup Assistant เพื่อรวมกลุ่มการวางแผนใหม่สำหรับกริปเปอร์และกำหนดให้เป็นปลายแขนกล [23]
- **อินเทอร์เฟซควบคุม:** กริปเปอร์สามารถควบคุมได้โดยใช้อินเทอร์เฟซเช่น GripperActionController (เผยแพร่ภายใต้ /gripper_controller/gripper_cmd) [21]
- **MoveIt! Task Constructor สำหรับการจับ (Grasping):** MTC มีขั้นตอนต่างๆ เช่น GenerateGraspPose เพื่อคำนวณท่าทางการจับ โดยพิจารณาตำแหน่งก่อนการจับและหลังการจับ [32] สามารถสร้างการวางแนวการจับได้หลายแบบ [32]
- **การติด/ถอดวัตถุ:** MoveIt! สามารถจำลองการติดและถอดวัตถุกับปลายแขนกล โดยอัปเดตฉากการวางแผนสำหรับการเคลื่อนที่ครั้งต่อไป [31]

การสร้างแบบจำลองกริปเปอร์ใน URDF/SRDF และการรวมเข้ากับ MoveIt! อย่างเหมาะสมไม่ได้เป็นเพียงแค่การเปิดและปิดเท่านั้น แต่เป็นสิ่งสำคัญสำหรับการวางแผนที่คำนึงถึงการชนในระหว่างการจับและสำหรับการแสดงสถานะของหุ่นยนต์ได้อย่างถูกต้องเมื่อถือวัตถุ [23] สิ่งนี้ทำให้มั่นใจว่าหุ่นยนต์จะไม่ชนกับวัตถุที่เพิ่งหยิบหรือองค์ประกอบอื่นๆ ในสภาพแวดล้อม การกำหนดรูปทรงของกริปเปอร์ใน URDF/SRDF อย่างถูกต้อง และการใช้ฟังก์ชัน

attachObject ของ MoveIt! จะอัปเดตฉากการวางแผนแบบไดนามิก ทำให้ MoveIt! สามารถตรวจสอบการชนกับรูปทรงหุ่นยนต์รวมกับวัตถุที่ถืออยู่ได้ สิ่งนี้มีความสำคัญอย่างยิ่งต่อการหยิบและวางที่

แข็งแกร่ง เนื่องจากหากไม่มีการจำลองนี้ หุ่นยนต์อาจหยิบวัตถุได้สำเร็จ แต่ชนกับโต๊ะหรือตัวมันเองทันทีเมื่อพยายามเคลื่อนย้ายวัตถุ

5. การรวมวิชันซิสเต็ม: การตั้งค่ากล้องและการประมวลผลภาพ

5.1 การเลือกฮาร์ดแวร์กล้องและการรวม ROS

การเลือกกล้องที่เหมาะสมเป็นขั้นตอนแรกที่สำคัญในการพัฒนาระบบวิชันซิสเต็มสำหรับหุ่นยนต์

- **การเลือกกล้อง:**
  - **เว็บแคม USB:** ง่ายต่อการรวมเข้ากับระบบ มีการใช้งานทั่วไป และรองรับโดยแพ็กเกจ usb_cam ของ ROS [35] ให้ภาพ RGB มาตรฐาน
  - **กล้องวัดความลึก (Depth Cameras) (เช่น Microsoft Kinect):** ให้ข้อมูล RGB-D (สี + ความลึก) ซึ่งมีความสำคัญต่อการรับรู้ 3 มิติและการระบุตำแหน่งวัตถุ [21] สิ่งนี้จำเป็นสำหรับการประมาณท่าทาง 3 มิติที่แม่นยำสำหรับการหยิบและวาง แม้ว่ากล้อง USB ทั่วไปจะเพียงพอสำหรับการตรวจจับสี 2 มิติ แต่กล้องวัดความลึกจะช่วยเพิ่มขีดความสามารถของระบบได้อย่างมากโดยการให้ข้อมูล 3 มิติ ซึ่งมีความสำคัญต่อการระบุตำแหน่งวัตถุที่แม่นยำและการวางแผนการจับในสถานการณ์การหยิบและวางในโลกแห่งความเป็นจริง สิ่งนี้แสดงถึงเส้นทางการอัปเกรดที่เป็นธรรมชาติสำหรับโครงการ
- **การรวม ROS (usb_cam package):**
  - ติดตั้ง ros-kinetic-usb-cam (สำหรับ ROS 1) หรือ ros-<ros2-distro>-usb-cam (สำหรับ ROS 2) [35]
  - เรียกใช้ usb_cam_node [35]
  - เผยแพร่ข้อความ sensor_msgs/Image (ภาพที่เข้ากันได้กับ OpenCV ที่ไม่ได้บีบอัด) และ sensor_msgs/CameraInfo (เมทริกซ์การสอบเทียบ) ไปยังหัวข้อต่างๆ เช่น /usb_cam/image_raw และ /usb_cam/camera_info [35]
  - รองรับพารามิเตอร์ที่กำหนดค่าได้ (เช่น ชื่อกล้อง, ID เฟรม, โหมด I/O, รูปแบบพิกเซล, การควบคุมภายใน เช่น การเปิดรับแสง, โฟกัส) [35]
  - สามารถเรียกใช้ด้วยการตั้งค่าเริ่มต้นหรือไฟล์พารามิเตอร์ [36]
- **การรวมกล้องวัดความลึก (ตัวอย่าง Gazebo):**
  - ต้องเพิ่มปลั๊กอินกล้องวัดความลึก ROS เฉพาะ (เช่น libgazebo_ros_openni_kinect.so) ไปยังไฟล์ SDF ของโมเดล Gazebo เพื่อเผยแพร่ Point Cloud และภาพไปยังหัวข้อ ROS [39]

- สำคัญที่ต้องตั้งค่า updateRate เป็น 0 เพื่อใช้ Rate ของเซ็นเซอร์หลัก [39]
- ชื่อหัวข้อควรตรงกับแพ็กเกจ ROS ที่ใช้กันทั่วไปเพื่อให้ง่ายต่อการสลับระหว่างกล้องจริงและกล้องจำลอง [39]

5.2 การสอบเทียบกล้องเพื่อการรับรู้ที่แม่นยำ

- **ความสำคัญ:** การสอบเทียบกล้องจะประมาณค่าพารามิเตอร์ภายใน (ความยาวโฟกัส, จุดศูนย์กลางออปติคอล, ค่าสัมประสิทธิ์การบิดเบือน) และภายนอก (การหมุน, การแปลเทียบกับโลก) [40] สิ่งนี้จำเป็นสำหรับการรับรู้ 3 มิติที่แม่นยำและการแก้ไขภาพ [37] การสอบเทียบกล้องเป็นขั้นตอนที่จำเป็นอย่างยิ่งสำหรับหุ่นยนต์ที่ขับเคลื่อนด้วยวิชันซิสเต็มที่ต้องการความแม่นยำ [37] กล้องที่ไม่ได้รับการสอบเทียบจะนำไปสู่การบิดเบือนและการประมาณท่าทาง 3 มิติที่ไม่ถูกต้อง ซึ่งส่งผลโดยตรงต่อการจับที่ผิดพลาดและการชน ทำให้ระบบการหยิบและวางทั้งหมดทำงานผิดพลาด
- **กระบวนการสอบเทียบโดยใช้แพ็กเกจ camera_calibration:**
  - **ข้อกำหนดเบื้องต้น:** ติดตั้ง ROS, usb_cam (หรือไดรเวอร์กล้องอื่น) ทำงานและเผยแพร่ภาพ [37] ติดตั้ง
    ros-noetic-camera-calibration (สำหรับ ROS 1) หรือ
    ros-<ros2-distro>-camera-calibration [37]
  - **รูปแบบการสอบเทียบ:** ใช้รูปแบบกระดานหมากรุก (checkerboard) หรือวงกลมไม่สมมาตร [37]
  - **การเรียกใช้เครื่องมือ:** rosrun camera_calibration cameracalibrator.py --size <rows>x<cols> --square <size_in_meters> image:=/my_camera/image_raw camera:=/my_camera [37] ปรับ
    --size และ --square ให้ตรงกับรูปแบบจริง [37]
  - **การเก็บข้อมูล:** เคลื่อนย้ายรูปแบบไปรอบๆ เพื่อครอบคลุมตำแหน่งต่างๆ (ซ้าย, ขวา, บน, ล่าง), การหมุน (ทั้ง 3 แกน) และระยะทาง (ใกล้, ไกล) [37] ตรวจสอบให้แน่ใจว่าแถบ X, Y, Size, Skew กลายเป็นสีเขียว [43]
  - **การสอบเทียบและบันทึก:** คลิก "CALIBRATE" (อาจใช้เวลาหลายนาทีถึงหลายชั่วโมง) จากนั้นคลิก "SAVE" [37] ผลลัพธ์จะถูกบันทึกไปยัง
    ~/.ros/camera_info/<camera_name>.yaml [37]
  - **การแก้ไขภาพ (Rectification):** หลังจากการสอบเทียบ ควรแก้ไขภาพ (undistorted) ก่อนใช้งาน [37] สามารถใช้แพ็กเกจ
    image_proc สำหรับการนี้ได้ [37]
- **ค่า Intrinsics แบบกำหนดเอง:** หากทราบค่า intrinsics สามารถป้อนด้วยตนเองลงในไฟล์ .yaml ได้ [37]

5.3 พื้นฐานการประมวลผลภาพด้วย OpenCV ใน ROS

- **ภาพรวม OpenCV:** เป็นไลบรารีคอมพิวเตอร์วิชันโอเพนซอร์สที่มีประสิทธิภาพ [1]
- **cv_bridge:** จำเป็นสำหรับการแปลงระหว่างข้อความ sensor_msgs/Image ของ ROS และออบเจกต์ภาพ cv::Mat (หรือ NumPy arrays ใน Python) ของ OpenCV [1]
  cv_bridge เป็นส่วนประกอบมิดเดิลแวร์ที่สำคัญ ซึ่งช่วยลดความซับซ้อนของการจัดเรียงข้อมูลและการแปลงรูปแบบระหว่างระบบส่งข้อความของ ROS และการแสดงภาพของ OpenCV สิ่งนี้ช่วยให้นักพัฒนาสามารถใช้ประโยชน์จากอัลกอริทึม OpenCV ที่มีประสิทธิภาพได้โดยตรงภายในระบบนิเวศ ROS
  - bridge.imgmsg_to_cv2(imagemsg): แปลงภาพ ROS เป็นภาพ OpenCV [1]
  - bridge.cv2_to_imgmsg(img, "bgr8"): แปลงภาพ OpenCV กลับเป็นภาพ ROS [45]
- **การเผยแพร่/สมัครรับภาพ:**
  - สร้างผู้เผยแพร่ ROS สำหรับหัวข้อภาพ (เช่น /image) [45]
  - สร้างผู้สมัครรับ ROS สำหรับหัวข้อภาพ พร้อมฟังก์ชัน callback (process_image) เพื่อจัดการข้อความที่เข้ามา [45]
  - ใช้ rospy.spin() เพื่อให้โหนดทำงานและประมวลผล callback [45]
- **การจัดการภาพพื้นฐาน:**
  - **การอ่าน/แสดงภาพ:** cv2.imread(), cv2.imshow(), cv2.waitKey() [45]
  - **การปรับขนาด:** cv2.resize() เพื่อเร่งความเร็วในการประมวลผล [45]
  - **การแปลงพื้นที่สี:** cv2.cvtColor() (เช่น BGR เป็น HSV, BGR เป็น Grayscale) [1]

5.4 การตรวจจับและแบ่งส่วนวัตถุตามสี

- **พื้นที่สี HSV:**
  - **ข้อดี:** HSV (Hue, Saturation, Value) มักถูกเลือกใช้มากกว่า RGB สำหรับการตรวจจับตามสี เนื่องจากแยกข้อมูลสี (Hue) ออกจากสภาพแสง (Saturation, Value) ทำให้มีความทนทานต่อการเปลี่ยนแปลงของแสง [1]
  - **ช่วงค่า:** ค่า Hue ใน OpenCV โดยทั่วไปอยู่ในช่วง 0-180 ในขณะที่ Saturation และ Value อยู่ในช่วง 0-255 [1] สิ่งนี้สำคัญเมื่อเปรียบเทียบกับเครื่องมือเลือกสีอื่นๆ [1]
  - **การหาค่า HSV:** ใช้ rqt_image_view และ rqt_reconfigure (Dynamic Reconfigure) ร่วมกับโหนด cvexample.py เพื่อปรับแถบเลื่อน HSV แบบโต้ตอบ [1]
- **การกรองสี (cv2.inRange()):**
  - **กระบวนการ:** แปลงภาพเป็น HSV จากนั้นใช้ cv2.inRange(hsv, lower_bound, upper_bound) เพื่อสร้างมาสก์ไบนารี แยกสีเป้าหมาย [1]

    - **การมาสก์:** ใช้มาสก์กับภาพต้นฉบับโดยใช้ cv2.bitwise_and() เพื่อให้ได้เฉพาะสีเป้าหมาย [1]
- **การแปลงภาพเป็นไบนารี (Image Binarization):** แปลงพิกเซลเป็น 0 (ดำ) และ 1 (ขาว) โดยใช้การ Thresholding [46]
- **การกัดกร่อนและการขยาย (Erosion and Dilation):**
    - **การกัดกร่อน (Erosion):** ลบขอบหยักหรือสัญญาณรบกวน ทำให้วัตถุในภาพหน้าหดตัว [47]
    - **การขยาย (Dilation):** ขยายขอบภาพ เติมพิกเซลที่ไม่ใช่เป้าหมาย เชื่อมต่อส่วนที่ขาดของวัตถุ [47]
    - **การเปิด/ปิด (Opening/Closing):** การรวมกันของการกัดกร่อน/การขยายสำหรับการลบสัญญาณรบกวนและการเติมช่องว่าง [47]

กระบวนการปรับปรุงซ้ำๆ ของการมาสก์สี การแปลงเป็นไบนารี และการดำเนินการทางสัณฐานวิทยา (กัดกร่อน/ขยาย) มีความสำคัญอย่างยิ่งต่อการตรวจจับวัตถุที่แข็งแกร่งในสถานการณ์จริง เนื่องจากเกณฑ์สีดิบไม่เพียงพอเนื่องจากสัญญาณรบกวนและการเปลี่ยนแปลงของแสง [1] สิ่งนี้แสดงให้เห็นถึงความท้าทายในทางปฏิบัติของคอมพิวเตอร์วิชันที่นอกเหนือจากแนวคิดทางทฤษฎี

ตาราง 5.1: ตัวอย่างช่วงสี HSV สำหรับวัตถุทั่วไป

สี	ช่วง HSV ล่างทั่วไป	ช่วง HSV บนทั่วไป	หมายเหตุ
แดง	$$	$$	ต้องระวัง Hue ที่ "ห่อหุ้ม" (wrap-around) โดยใช้ 2 ช่วง: [0,10,...] และ [170,180,...]
เขียว	$$	$$	ช่วงค่าอาจแตกต่างกันไปตามสภาพแสง
น้ำเงิน	$$	$$	สำคัญอย่างยิ่งที่ต้องปรับแต่งแบบโต้ตอบด้วย rqt_reconfigure
เหลือง	$$	$$	-

5.5 การตรวจจับโครงร่างและการระบุตำแหน่งวัตถุ

- **การตรวจจับโครงร่าง (cv2.findContours()):**
    - **กระบวนการ:** ใช้กับมาสก์ไบนารี (หลังจากการดำเนินการทางสัณฐานวิทยา) เพื่อระบุขอบเขตของวัตถุ [1]

cv2.RETR_EXTERNAL จะดึงเฉพาะโครงร่างภายนอก [1]
    - **ผลลัพธ์:** ส่งคืนรายการโครงร่าง [1]
- **การระบุโครงร่างที่ใหญ่ที่สุด:**
    - ใช้ max(contours, key = cv2.contourArea) เพื่อหาโครงร่างที่มีพื้นที่ใหญ่ที่สุด โดยสมมติว่าวัตถุเป้าหมายคือกลุ่มสีที่ใหญ่ที่สุด [1]
- **การระบุตำแหน่งวัตถุ:**
    - **กล่องล้อมรอบ (Bounding Box) (cv2.boundingRect()):** คำนวณสี่เหลี่ยมผืนผ้าที่ตั้งตรงที่เล็กที่สุดที่ล้อมรอบโครงร่าง ให้ค่า x, y, w, h (มุมบนซ้าย, ความกว้าง, ความสูง) [1]
    - **การคำนวณจุดศูนย์กลาง:** cx = x + (w/2), cy = y + (h/2) จากกล่องล้อมรอบ [1]
    (cx, cy) นี้คือพิกัดพิกเซล 2 มิติของจุดศูนย์กลางของวัตถุ
    - **การวาดผลลัพธ์:** cv2.drawContours(), cv2.rectangle() เพื่อแสดงภาพวัตถุที่ตรวจพบ [1]

การระบุ "โครงร่างที่ใหญ่ที่สุดตามพื้นที่" [1] เป็นวิธีการทั่วไปในการเลือกวัตถุในสถานการณ์ที่เรียบง่าย แต่ก็มีความเสี่ยง หากมีวัตถุอื่นที่มีขนาดใหญ่กว่าและมีสีเดียวกัน ระบบอาจหยิบวัตถุผิด สิ่งนี้ชี้ให้เห็นถึงข้อจำกัดของการตรวจจับตามสีแบบง่ายๆ และความจำเป็นในการจดจำวัตถุที่ซับซ้อนยิ่งขึ้นในสภาพแวดล้อมที่ซับซ้อน การใช้ฮิวริสติกนี้จะสมมติว่าวัตถุเป้าหมายเป็นกลุ่มสีที่ใหญ่ที่สุดในมุมมองของกล้องเสมอ ซึ่งอาจนำไปสู่ความล้มเหลวหากมีวัตถุรบกวนขนาดใหญ่หรือวัตถุเป้าหมายถูกบดบังบางส่วน ข้อจำกัดนี้เน้นย้ำว่าวิธีการตรวจจับตามสีและโครงร่างที่ใหญ่ที่สุดเพียงอย่างเดียวนั้นไม่แข็งแกร่งพอสำหรับการหยิบจับวัตถุทั่วไป และเป็นจุดเริ่มต้นสำหรับการพัฒนาไปสู่การจดจำวัตถุที่ซับซ้อนยิ่งขึ้นในอนาคต

6. การพัฒนาระบบการหยิบจับอัตโนมัติ

6.1 สถาปัตยกรรมระบบสำหรับการจัดการที่ขับเคลื่อนด้วยวิชันซิสเต็ม

- **กราฟ ROS แบบโมดูลาร์:** ย้ำแนวคิดกราฟ ROS ซึ่งโหนดที่แตกต่างกันจะจัดการอินพุตกล้อง การประมวลผลภาพ การระบุตำแหน่งวัตถุ การวางแผนการเคลื่อนไหว และการควบคุมหุ่นยนต์ [3]
- **การไหลของข้อมูล:**
    - โหนดกล้องเผยแพร่ภาพดิบ (sensor_msgs/Image) และข้อมูลกล้อง (sensor_msgs/CameraInfo)
    - โหนดประมวลผลภาพสมัครรับภาพ เผยแพร่ท่าทางวัตถุที่ตรวจพบ (เช่น ประเภทข้อความที่กำหนดเอง หรือ geometry_msgs/PoseStamped)
    - โหนด MoveIt! สมัครรับท่าทางวัตถุ วางแผนวิถีการเคลื่อนที่ และส่งคำสั่งไปยัง

ros_control
    - ros_control เชื่อมต่อกับฮาร์ดแวร์หุ่นยนต์
- **การประสานงาน:** ใช้หัวข้อ ROS สำหรับการสตรีมข้อมูลต่อเนื่อง (เช่น ภาพ, สถานะข้อต่อ) และบริการ ROS สำหรับการร้องขอแบบไม่ต่อเนื่อง (เช่น "รับท่าทางวัตถุ", "ดำเนินการหยิบจับ") [3]

ความสำเร็จของระบบการหยิบจับอัตโนมัติขึ้นอยู่กับความแม่นยำและความแข็งแกร่งของห่วงโซ่การแปลงข้อมูลอย่างมาก [1] ตั้งแต่พิกัดพิกเซล 2 มิติไปจนถึงพิกัดกล้อง 3 มิติ และจากนั้นไปยังพิกัดฐานหุ่นยนต์ 3 มิติ ข้อผิดพลาดจะสะสมในแต่ละขั้นตอน ซึ่งเน้นย้ำถึงความจำเป็นในการสอบเทียบที่แม่นยำและอัลกอริทึมที่แข็งแกร่ง หากมีการคำนวณตำแหน่ง 3 มิติของวัตถุผิดพลาด แขนกลจะพยายามเคลื่อนที่ไปยังตำแหน่งที่ไม่ถูกต้อง ส่งผลให้การหยิบจับล้มเหลว เกิดการชนกับวัตถุ หรือแม้แต่การชนกับสภาพแวดล้อม ซึ่งส่งผลโดยตรงต่อเป้าหมาย "การทำงานอัตโนมัติ" และ "การหยิบจับ"

6.2 การแปลงการตรวจจับวัตถุเป็นท่าทางหุ่นยนต์

- **เฟรมพิกัด:** เน้นความสำคัญของการทำความเข้าใจเฟรมพิกัดที่แตกต่างกัน (เฟรมกล้อง, เฟรมฐาน, เฟรมโลก) และการแปลงระหว่างเฟรมเหล่านั้น [21]
- **ขั้นตอนการแปลง:**
    1. **พิกเซลเป็นพิกัดกล้อง:** ใช้ค่า intrinsics ของกล้อง (จากการสอบเทียบ) และข้อมูลความลึก (จากกล้องวัดความลึก) แปลงพิกัดพิกเซล 2 มิติ (u, v) และความลึก Z เป็นพิกัดกล้อง 3 มิติ (X_c, Y_c, Z_c)
    2. **กล้องเป็นพิกัดฐานหุ่นยนต์:** ใช้การแปลง extrinsics ของกล้อง (ท่าทางของกล้องเทียบกับเฟรมฐานของหุ่นยนต์) เพื่อแปลง (X_c, Y_c, Z_c) เป็น (X_b, Y_b, Z_b) เทียบกับฐานของหุ่นยนต์ ไลบรารี ROS tf (Transform Frame) มีความสำคัญอย่างยิ่งสำหรับขั้นตอนนี้ [21]
    3. **การวางแนววัตถุ:** สำหรับการหยิบจับ อาจไม่จำเป็นต้องใช้เพียงแค่ตำแหน่งเท่านั้น แต่ยังรวมถึงการวางแนวของวัตถุด้วย ซึ่งต้องใช้เทคนิควิชันซิสเต็มขั้นสูงขึ้น หรือการสมมติฐานเกี่ยวกับการวางแนววัตถุ
- **ไลบรารี ROS tf:** จำเป็นสำหรับการจัดการเฟรมพิกัดและการแปลง [21] MoveIt! ต้องการเป้าหมายที่สัมพันธ์กับ
  base_footprint [21] ระบบ
  tf (Transform Frame) ใน ROS เป็นรากฐานสำคัญสำหรับการจัดการความสัมพันธ์เชิงพื้นที่ในระบบหุ่นยนต์ ความสามารถในการให้การแปลงระหว่างเฟรมพิกัดใดๆ [21] มีความสำคัญอย่างยิ่งในการแปลข้อมูลเซ็นเซอร์ (เช่น ท่าทางวัตถุในเฟรมกล้อง) ไปเป็นคำสั่งที่เน้นหุ่นยนต์เป็นศูนย์กลาง (เช่น ท่าทางเป้าหมายสำหรับปลายแขนกลในเฟรมฐาน) หากไม่มี
  tf ทุกโหนดจะต้องฮาร์ดโค้ดการแปลง ทำให้ระบบเปราะบางและยากต่อการแก้ไข tf ให้วิธีการที่แข็งแกร่ง มีพลวัต และยืดหยุ่นในการจัดการความสัมพันธ์เชิงพื้นที่ที่ซับซ้อนภายในหุ่นยนต์

6.3 การวางแผนงานหยิบและวางด้วย MoveIt! Task Constructor

- **MoveIt! Task Constructor (MTC):** เป็นวิธีที่ยืดหยุ่นและโปร่งใสในการกำหนดและวางแผนการกระทำที่ประกอบด้วยงานย่อยที่พึ่งพาอาศัยกันหลายงาน [24] มีความสำคัญอย่างยิ่งสำหรับลำดับการจัดการที่ซับซ้อน แนวทางที่ใช้ขั้นตอนของ MTC [32] มีความสำคัญอย่างยิ่งต่อการสร้างระบบหยิบและวางที่แข็งแกร่ง เนื่องจากบังคับให้มีการแบ่งงานออกเป็นส่วนย่อยๆ ที่มีโครงสร้าง ทำให้สามารถวางแผนและจัดการข้อผิดพลาดได้อย่างอิสระในแต่ละขั้นตอนย่อย (เช่น การเข้าใกล้, การจับ, การยก) ความเป็นโมดูลาร์นี้ช่วยปรับปรุงการแก้ไขข้อผิดพลาดและการนำโค้ดกลับมาใช้ใหม่ได้อย่างมาก เมื่อเทียบกับการวางแผนการเคลื่อนที่แบบรวมศูนย์
- **การกำหนดขั้นตอน (Stages):** MTC แบ่งงานที่ซับซ้อน (เช่น การหยิบและวาง) ออกเป็นขั้นตอนแบบโมดูลาร์ [32]
    - **ขั้นตอนการเชื่อมต่อ (Connect Stages):** เชื่อมโยงผลลัพธ์ของขั้นตอนก่อนหน้าและขั้นตอนถัดไป (เช่น move_to_pick) [32]
    - **ขั้นตอนการสร้าง (Generator Stages):** คำนวณผลลัพธ์อย่างอิสระ (เช่น CurrentState, GenerateGraspPose) [32]
    - **SerialContainer:** จัดกลุ่มงานย่อยหลายงานสำหรับการกระทำเฉพาะ (เช่น "pick object") [32]
- **ตัวอย่างขั้นตอนการหยิบ (Pick Stages):**
    - move_to_pick: ย้ายแขนกลไปยังตำแหน่งก่อนการจับ [32]
    - generate_grasp_pose: สุ่มท่าทางการจับหลายแบบรอบๆ วัตถุ [32]
    - open hand/close hand: การกระทำของกริปเปอร์ [32]
- **การวางแผนและการดำเนินการ:** MTC สามารถสร้างแผนที่สำเร็จได้หลายแผน (task.plan(5)) และจากนั้นดำเนินการแผนเหล่านั้น (task.execute()) [32]
- **ฉากการวางแผนแบบไดนามิก:** MTC สามารถรวมเข้ากับการรับรู้แบบเรียลไทม์ (เช่น กล้องวัดความลึก) เพื่ออัปเดตฉากการวางแผนแบบไดนามิก ทำให้สามารถหยิบวัตถุที่ตรวจพบใหม่ได้ [33]

6.4 การจัดการข้อผิดพลาดและความแข็งแกร่งในการทำงานอัตโนมัติ

- **การคาดการณ์ความล้มเหลว:** อภิปรายจุดที่มักเกิดความล้มเหลว: การตรวจจับวัตถุล้มเหลว, การประมาณท่าทาง 3 มิติไม่ถูกต้อง, ท่าทางการจับที่เข้าไม่ถึง, ความล้มเหลวในการวางแผน (การชน, จุดเอกฐาน), ข้อผิดพลาดในการดำเนินการ (มอเตอร์หยุดทำงาน, การสื่อสารขาดหาย)
- **กลยุทธ์เพื่อความแข็งแกร่ง:**
    - **การลองใหม่ (Retries):** สำหรับความล้มเหลวในการวางแผน ให้ลองเป้าหมายที่แตกต่างกันเล็กน้อยรอบๆ เป้าหมายเดิม [21]

- **วงจรป้อนกลับ (Feedback Loops):** ใช้การป้อนกลับจากเซ็นเซอร์ (เช่น สถานะข้อต่อ, เซ็นเซอร์แรงกริปเปอร์) เพื่อยืนยันการกระทำ
- **สถานะเครื่อง (State Machines):** นำสถานะเครื่องที่แข็งแกร่งมาใช้เพื่อจัดการกระบวนการหยิบจับ โดยเปลี่ยนสถานะ (ตรวจจับ, วางแผน, ดำเนินการ, จับ, วาง) และจัดการข้อผิดพลาดอย่างเหมาะสม
- **การบันทึก (Logging):** การบันทึกสถานะระบบและข้อผิดพลาดอย่างครอบคลุม [28]
- **การดีบักด้วยภาพ (Visual Debugging):** ใช้ RViz สำหรับการแสดงภาพฉากการวางแผน, สถานะหุ่นยนต์ และวิถีการเคลื่อนที่ [21]
- **การปรับแต่งพารามิเตอร์ (Parameter Tuning):** ใช้ RQT Dynamic Reconfigure สำหรับการปรับแต่งค่า PID และพารามิเตอร์อื่นๆ แบบเรียลไทม์ [24]

การจัดการข้อผิดพลาดที่แข็งแกร่งเป็นสิ่งสำคัญยิ่งสำหรับระบบอัตโนมัติ [21] การวางแผนและดำเนินการเส้นทางเพียงอย่างเดียวไม่เพียงพอ ระบบจะต้องตอบสนองอย่างชาญฉลาดต่อเหตุการณ์ที่ไม่คาดคิดหรือความล้มเหลว โดยมักจะลองใหม่ วางแผนใหม่ หรือเปลี่ยนไปสู่สถานะที่ปลอดภัย สิ่งนี้จะเปลี่ยนระบบจาก "การสาธิต" ไปสู่ "ตัวแทนอัตโนมัติที่เชื่อถือได้" ความแข็งแกร่งนี้เองที่ทำให้หุ่นยนต์เป็นอิสระและมีประโยชน์อย่างแท้จริงนอกสภาพแวดล้อมห้องปฏิบัติการที่ควบคุมอย่างสมบูรณ์แบบ

7. การติดตั้งใช้งานและการเพิ่มประสิทธิภาพบน Raspberry Pi

7.1 การติดตั้ง ROS บน Raspberry Pi

- **การเลือกระบบปฏิบัติการ:** Ubuntu Server 20.04 ได้รับการแนะนำสำหรับ Raspberry Pi (Pi 3B/3B+/4) สำหรับ ROS Noetic [10] Raspbian ไม่เหมาะสำหรับ ROS เนื่องจากความยากในการติดตั้ง [19] คำแนะนำในการติดตั้ง Ubuntu Server และ ros-base [10] แทน Raspbian หรือ Desktop-Full เป็นการเพิ่มประสิทธิภาพที่สำคัญสำหรับ Raspberry Pi ซึ่งช่วยแก้ไขข้อจำกัดด้านทรัพยากรโดยตรง วิธีการเชิงรุกนี้ช่วยป้องกันปัญหาคอขวดด้านประสิทธิภาพและปัญหาความเสถียรที่พบบ่อย
- **ขั้นตอนการติดตั้ง (ตัวอย่าง ROS Noetic, สามารถปรับใช้กับ ROS 2 ได้):**
  1. **ติดตั้ง Ubuntu Server:** ใช้ Raspberry Pi Imager [10]
  2. **เข้าถึงเทอร์มินัล:** ผ่านเดสก์ท็อปหรือ SSH [10]
  3. **กำหนดค่า Repository:** เปิดใช้งาน universe, restricted, multiverse [10]
  4. **ตั้งค่าแหล่งข้อมูล:** เพิ่ม ROS package repository และ public key [10] อัปเดตแหล่งข้อมูล apt (

sudo apt update) [10]

5. **ติดตั้งแพ็กเกจหลักของ ROS:** ติดตั้ง ros-noetic-ros-base [10] หลีกเลี่ยง Desktop-Full Install เนื่องจากทรัพยากรของ Pi มีจำกัด [10] ติดตั้งแพ็กเกจเพิ่มเติมตามความจำเป็น (เช่น robot-state-publisher) [10]
6. **ตั้งค่า rosdep:** ติดตั้ง python3-rosdep, rosdep init, rosdep update [10]
7. **ตั้งค่าสภาพแวดล้อม:** Source setup.bash (source /opt/ros/noetic/setup.bash) และเพิ่มลงใน ~/.bashrc เพื่อให้คงอยู่ [10]
8. **สร้าง Workspace:** mkdir -p ~/catkin_ws/src, cd ~/catkin_ws, catkin_make [8] Source ไฟล์ setup ของ workspace [8]

- **อิมเมจที่ติดตั้งมาล่วงหน้า:** กล่าวถึงความพร้อมใช้งานของอิมเมจ ROS ที่ติดตั้งมาล่วงหน้าสำหรับ Raspberry Pi ซึ่งเป็นทางเลือกสำหรับผู้เริ่มต้น [48]

7.2 กลยุทธ์การเพิ่มประสิทธิภาพสำหรับสภาพแวดล้อมที่มีทรัพยากรจำกัด

- **การจัดการ RAM:**
  - **จำกัดจำนวนโหนด:** ลดจำนวนโหนด ROS ที่ทำงานอยู่ รวมฟังก์ชันการทำงานเข้าด้วยกันในโหนดที่น้อยลงแต่ใหญ่ขึ้นตามความเหมาะสม [10] ตัวอย่างเช่น จัดการเซ็นเซอร์หลายตัวในโหนดเดียว [10]
  - **Nodelets (C++):** ใช้ nodelets ใน C++ เพื่อเรียกใช้อัลกอริทึมหลายตัวในกระบวนการเดียว ลดโอเวอร์เฮดการสื่อสารระหว่างกระบวนการและการใช้หน่วยความจำ [19]
  - **การอัปเกรดฮาร์ดแวร์:** พิจารณา Raspberry Pi รุ่นที่มี RAM มากขึ้น (เช่น Pi 4/5 ที่มี RAM 4GB หรือ 8GB) หากเป็นไปได้ [19]
- **การเพิ่มประสิทธิภาพ CPU:**
  - **การคอมไพล์:** ใช้ catkin_make -j2 (หรือ colcon build --executor-args --workers 2) เพื่อจำกัดการคอมไพล์ไว้ที่ 2 คอร์ ป้องกันการค้างเนื่องจากการใช้ทรัพยากรมากเกินไปในระหว่างการสร้าง C++ ขนาดใหญ่ [10]
  - **การเลือกภาษา:** C++ โดยทั่วไปให้ประสิทธิภาพที่ดีกว่าสำหรับโหนดที่ต้องการการคำนวณสูงเมื่อเทียบกับ Python แม้ว่า Python จะง่ายกว่าสำหรับการสร้างต้นแบบอย่างรวดเร็ว [19]
  - **การถ่ายโอนงานคำนวณที่หนักหน่วง:** ถ่ายโอนงานที่ต้องใช้การคำนวณสูง (เช่น การวางแผนการเคลื่อนไหวที่ซับซ้อน, อัลกอริทึมการรับรู้ขั้นสูง เช่น SLAM หรือการอนุมานการเรียนรู้เชิงลึก) ไปยังคอมพิวเตอร์หลักที่มีประสิทธิภาพมากกว่า (การตั้งค่า ROS แบบหลายเครื่อง) [10] Pi สามารถเรียกใช้ลูปควบคุมแบบเรียลไทม์ได้ [19] การตั้งค่า ROS แบบ "หลายเครื่อง" [10] เป็นกลยุทธ์ที่ซับซ้อนที่ช่วยให้สามารถใช้ประโยชน์จากต้นทุนที่ต่ำและความสามารถในการฝังตัวของ Pi สำหรับการควบคุมแบบเรียลไทม์ ในขณะที่ถ่ายโอนงานที่ต้องใช้การคำนวณสูงไปยังเครื่องที่มีประสิทธิภาพมากกว่า สิ่งนี้แสดงถึงแนวทางที่ใช้งานได้จริงในการเอาชนะข้อจำกัดของฮาร์ดแวร์ในโครงการหุ่นยนต์ที่ซับซ้อน

- **สถาปัตยกรรมโปรเซสเซอร์ (ARM vs. x86):** ควรทราบว่าแพ็กเกจ ROS บางแพ็กเกจอาจมีการพึ่งพาที่ได้รับการทดสอบ/ปรับให้เหมาะสมสำหรับ x86 เท่านั้น ซึ่งอาจทำให้เกิดปัญหากับ ARM ได้ [10]
- **หลีกเลี่ยงเครื่องมือจำลอง 3 มิติบน Pi:** อย่าติดตั้งหรือใช้ RViz หรือ Gazebo โดยตรงบน Raspberry Pi [10] ใช้การตั้งค่าแบบหลายเครื่องที่เครื่องมือเหล่านี้ทำงานบนคอมพิวเตอร์หลักที่มีประสิทธิภาพ [10]

ตาราง 7.1: การเปรียบเทียบรุ่น Raspberry Pi สำหรับวิทยาการหุ่นยนต์

รุ่น	CPU	ตัวเลือก RAM	การเชื่อมต่อ	ความเหมาะสมทั่วไปสำหรับ ROS Robotics
Raspberry Pi 3B+	Quad-core 1.4 GHz ARM Cortex-A53	1GB	USB 2.0, Ethernet, Wi-Fi	เหมาะสำหรับ ROS พื้นฐาน, การควบคุมแบบฝังตัวขั้นต่ำ
Raspberry Pi 4 (1GB)	Quad-core 1.5 GHz ARM Cortex-A72	1GB	USB 3.0, Gigabit Ethernet, Wi-Fi	เหมาะสำหรับ ROS พื้นฐาน, ดีกว่า 3B+ สำหรับการประมวลผลบางอย่าง
Raspberry Pi 4 (2GB)	Quad-core 1.5 GHz ARM Cortex-A72	2GB	USB 3.0, Gigabit Ethernet, Wi-Fi	ดีสำหรับ ROS ทั่วไป, การประมวลผลบนบอร์ดขนาดเล็ก
Raspberry Pi 4 (4GB)	Quad-core 1.5 GHz ARM Cortex-A72	4GB	USB 3.0, Gigabit Ethernet, Wi-Fi	แนะนำสำหรับโครงการที่ซับซ้อนมากขึ้น, การประมวลผลวิชันซิสเต็มบนบอร์ด
Raspberry Pi 4 (8GB)	Quad-core 1.5 GHz ARM Cortex-A72	8GB	USB 3.0, Gigabit Ethernet, Wi-Fi	ดีที่สุดสำหรับ Pi สำหรับการประมวลผลบนบอร์ดที่หนักหน่วง, การทดลอง AI ขนาดเล็ก
Raspberry Pi 5	Quad-core 2.4 GHz ARM	4GB, 8GB	USB 3.0, Gigabit	ประสิทธิภาพสูงที่สุด, เหมาะสำหรับ

	Cortex-A76		Ethernet, Wi-Fi 6	งานที่ต้องการพลังประมวลผลมากที่สุดบน Pi

7.3 การเชื่อมต่อฮาร์ดแวร์และข้อควรพิจารณาแบบเรียลไทม์

- **การเชื่อมต่ออุปกรณ์ต่อพ่วง:** Raspberry Pi เชื่อมต่อกับไดรเวอร์มอเตอร์ (เช่น ผ่าน Arduino UNO เป็นไมโครคอนโทรลเลอร์ [23]) และเซ็นเซอร์อื่นๆ [50] ผ่าน USB, GPIO หรืออินเทอร์เฟซอื่นๆ
- **ROS-Arduino Bridge:** สำหรับการควบคุมมอเตอร์ระดับต่ำ Arduino สามารถทำหน้าที่เป็นไมโครคอนโทรลเลอร์ สื่อสารกับ Raspberry Pi ผ่านการสื่อสารแบบอนุกรม และจากนั้นรวมเข้ากับ ROS [23]
- **เรียลไทม์ใน ROS:** ROS เองเป็น "soft real-time" [16] ซึ่งหมายความว่ามีเป้าหมายที่ความหน่วงต่ำ แต่ไม่รับประกันเวลาที่เข้มงวด ROS 2 ได้รับการออกแบบโดยคำนึงถึงเรียลไทม์ แต่ต้องมีการตั้งค่าเฉพาะ [16]
- **การบรรลุประสิทธิภาพที่คาดการณ์ได้:**
  - ลดการดำเนินการที่ไม่เป็นไปตามกำหนด
  - ใช้การตั้งค่า QoS ที่เหมาะสมใน ROS 2 [17]
  - รักษาวงจรควบคุมที่สำคัญบน Pi ถ่ายโอนงานที่ไม่สำคัญ [19]
  - พิจารณาระบบปฏิบัติการเรียลไทม์เฉพาะ (RTOS) หรือเคอร์เนลเรียลไทม์สำหรับ Linux หากต้องการเรียลไทม์แบบแข็ง (hard real-time) (นอกเหนือจากขอบเขตของรายงานนี้ แต่ควรกล่าวถึงสำหรับระดับผู้เชี่ยวชาญ)

แม้ว่า Raspberry Pi จะมีราคาที่คุ้มค่า แต่ลักษณะ "soft real-time" ของมัน [16] หมายความว่าสำหรับงานหยิบและวางที่แม่นยำสูงหรือพลวัตสูง การออกแบบวงจรควบคุมอย่างรอบคอบ และอาจรวมถึงการถ่ายโอนการควบคุมป้อนกลับความถี่สูงไปยังไมโครคอนโทรลเลอร์เฉพาะ (เช่น Arduino [23]) เป็นสิ่งสำคัญอย่างยิ่ง สิ่งนี้เป็นสถาปัตยกรรมไฮบริดที่พบบ่อยในหุ่นยนต์สำหรับงานอดิเรกและการวิจัย การใช้ Arduino ช่วยให้สามารถควบคุมมอเตอร์ได้อย่างแม่นยำในระดับต่ำ ในขณะที่ Raspberry Pi สามารถจัดการงานระดับสูง เช่น วิชันซิสเต็มและการวางแผน

8. ความท้าทาย แนวปฏิบัติที่ดีที่สุด และแนวโน้มในอนาคต

8.1 ข้อผิดพลาดทั่วไปและการแก้ไขปัญหาใน ROS Robotics

- **ปัญหาการตั้งค่าสภาพแวดล้อม:** การ source setup.bash ไม่ถูกต้อง, การพึ่งพาที่ขาดหายไป [9]
- **การกำหนดค่าเครือข่าย:** ปัญหาการค้นหา ROS Master/DDS ในการตั้งค่าแบบหลายเครื่อง [19]
- **ข้อผิดพลาดในการคอมไพล์:** การใช้ทรัพยากรมากเกินไปบน Pi (catkin_make -j2), การพึ่งพา x86 กับ ARM [10]
- **ปัญหาเกี่ยวกับกล้อง:** กล้องที่ไม่ได้รับการสอบเทียบ, ชื่อหัวข้อไม่ถูกต้อง, ปัญหาไดรเวอร์ [37]
- **การประมวลผลภาพ:** ช่วง HSV ไม่ถูกต้อง (ความแปรปรวนของแสง), สัญญาณรบกวน, การปรับแต่งการดำเนินการทางสัณฐานวิทยา
- **จลนศาสตร์/การวางแผนการเคลื่อนไหว:** เป้าหมายที่เข้าไม่ถึง, จุดเอกฐาน, ปัญหาการชน, URDF/SRDF ไม่ถูกต้อง [21]
- **เครื่องมือดีบัก:** เน้น rostopic list/info/echo/hz, rosnode list/info, rqt_graph, rqt_image_view, RViz สำหรับการดีบักด้วยภาพ

8.2 แนวปฏิบัติที่ดีที่สุดสำหรับการพัฒนา ROS ที่แข็งแกร่งบนระบบฝังตัว

- **เริ่มต้นอย่างเรียบง่าย:** สร้างส่วนประกอบทีละน้อยและทดสอบอย่างละเอียด [1]
- **การควบคุมเวอร์ชัน:** ใช้ Git สำหรับการจัดการโค้ด [34]
- **การออกแบบแบบโมดูลาร์:** ใช้ประโยชน์จากโหนดและหัวข้อ ROS เพื่อการแยกส่วนที่ชัดเจน [3]
- **การบันทึกและการแสดงภาพ:** จำเป็นสำหรับการทำความเข้าใจพฤติกรรมของระบบและการดีบัก [21]
- **กำหนดพารามิเตอร์ทุกอย่าง:** ใช้พารามิเตอร์ ROS สำหรับค่าที่กำหนดค่าได้ (เช่น ช่วง HSV, ค่า PID) [3]
- **การสอบเทียบอย่างเป็นระบบ:** สอบเทียบเซ็นเซอร์ทั้งหมด (กล้อง, แขนกลหุ่นยนต์หากจำเป็น) อย่างพิถีพิถัน
- **จำลองก่อน:** พัฒนาและทดสอบอัลกอริทึมใน Gazebo/RViz ก่อนติดตั้งใช้งานบนฮาร์ดแวร์ [21]
- **การตระหนักถึงทรัพยากร:** เพิ่มประสิทธิภาพโค้ดสำหรับข้อจำกัดของ Raspberry Pi [10]
- **การมีส่วนร่วมกับชุมชน:** ใช้ ROS Answers, Discourse, GitHub สำหรับการแก้ไขปัญหาและการเรียนรู้ [5]

8.3 หัวข้อขั้นสูงและการปรับปรุงในอนาคต

- **การเรียนรู้เชิงลึกสำหรับการจดจำวัตถุ:** เปลี่ยนจากการตรวจจับตามสีไปสู่การจดจำวัตถุทั่วไปโดยใช้โมเดลเช่น YOLO หรือ Mask R-CNN เพื่อเพิ่มความแข็งแกร่งและความสามารถในการ

แยกแยะประเภทวัตถุที่นอกเหนือจากสี [33] สิ่งนี้จะต้องใช้พลังการประมวลผลที่มากขึ้น (เช่น NVIDIA Jetson หรือการถ่ายโอนไปยังเดสก์ท็อป)

- **การควบคุมแรงและการปฏิบัติตามข้อกำหนด:** สำหรับการจัดการที่ละเอียดอ่อน การรวมเซ็นเซอร์แรง/แรงบิดและการนำกลยุทธ์การควบคุมที่ยืดหยุ่นมาใช้
- **การติดตามวัตถุ:** ใช้เทคนิคเช่น Kalman filters หรือ optical flow เพื่อติดตามวัตถุที่ตรวจพบ ปรับปรุงความแข็งแกร่งต่อการบดบังชั่วคราวหรือสภาพแวดล้อมแบบไดนามิก
- **การปฏิสัมพันธ์ระหว่างมนุษย์กับหุ่นยนต์:** การเพิ่มคำสั่งเสียง [50] การควบคุมด้วยท่าทาง หรือการควบคุมระยะไกล
- **ระบบหุ่นยนต์หลายตัว:** การปรับขนาดสถาปัตยกรรมสำหรับงานที่ประสานกันด้วยแขนกลหลายตัว
- **การรวมเข้ากับมาตรฐานอุตสาหกรรม:** การสำรวจ ROS-Industrial สำหรับแอปพลิเคชันที่แข็งแกร่งและได้รับการรับรองความปลอดภัยมากขึ้น

อ้างอิง

- [6] ROS TUTORIALS. Tiziano Fiorenzani. Playlist on YouTube.
- [7] ROS Beginner Tutorials - Projects - Open Robotics Discourse.
- [3] Packtpub - ROS Architecture and Concepts.
- [4] ROS (Robot Operating System) Documentation.
- [8] AutomaticAddison - How to Create a ROS Workspace.
- [9] ROS Wiki - Installing and Configuring Your ROS Environment.
- [24] GazeboSim - Tutorial: ROS Control.
- [29] The Construct - ROS Control (YouTube).
- [22] Instructables - DIY Robotic Arm.
- [20] Arctos Robotics - DIY 3D printed robotic arm with AI.
- [21] ROS Wiki - Robotican/Tutorials/Arm manipulation.
- [34] Instructables - ROS MoveIt Robotic Arm.
- [26] Picknik AI - Guide to Cartesian Planners in MoveIt.
- [27] Reddit - Cartesian position controller or joint position.
- [38] Isaac Sim Omniverse NVIDIA Docs - ROS 2 Camera Tutorial.
- [39] GazeboSim - Tutorial: ROS Depth Camera.
- [42] Robotics StackExchange - Camera calibration algorithm.
- [43] ModalAI Docs - ROS Calibrate Cameras.
- [35] ROS Wiki - usb_cam.
- [36] ROS Index - usb_cam.
- [1] Campus Rover Gitbook - Object recognition based on color and size.

- [2] ROS Discourse - Process of achieving color and image recognition by myPalletizer AI Kit.
- [45] Industrial Training Master - OpenCV-in-Python.
- [44] Learn Robotics with ROS - OpenCV for robotics in ROS environment (YouTube).
- [11] Reddit - Color Detection with OpenCV.
- [46] Industrial Training Master - OpenCV-in-Python (Color Filter).
- [32] MoveIt! Docs - Pick and Place with MoveIt Task Constructor.
- [33] AutomaticAddison - How to Build a Full Pick and Place Pipeline in ROS 2 Jazzy (YouTube).
- [44] Learn Robotics with ROS - ROS Robotic Arm Vision System (YouTube).
- [5] ROS - Robot Operating System.
- [47] Hiwonder Docs - ArmPi Pro ROS OpenCV.
- [23] Instructables - ROS MoveIt Robotic Arm Part 2 : Robot Controller.
- [10] RoboticsBackend - Install ROS on Raspberry Pi 3.
- [48] Instructables - Raspberry Pi and ROS Robotic Operating System.
- [51] MathWorks - Getting Started with Robot Operating System (ROS) on Raspberry Pi.
- [19] RoboticsBackend - Using ROS on Raspberry Pi: Best Practices.
- [49] ROS Answers - Raspberry Pi optimization.
- [50] Revistas UDistrital - Configuration and programming of the SERB robot.
- [13] ROS Answers - ROS or ROS2 for robotics beginner?
- [14] YouTube - Should you start with Ross one and then learn ROSS 2 or should you directly learn Ross. 2.
- [17] ROS 2 Design - Platforms and dependencies.
- [18] Micro-ROS Docs - ROS 2 Feature Comparison.
- [15] Hello Robot Forum - Is ROS2 developed enough or should I stick with ROS1?
- [16] Reddit - Should we use ros1 or ros2?
- [37] NVlabs Deep Object Pose - Camera Tutorial.
- [35] ROS Wiki - usb_cam (Dependencies).
- [52] Medium - Camera Calibration through Kalibr.
- [40] GeeksForGeeks - Camera Calibration with Python OpenCV.
- [41] Clover Coex Tech - Camera calibration.
- [53] Medium - Camera Calibration through Kalibr (Workspace).
- [54] YouTube - ROS robotic arm joint control tutorial.
- [12] Mike Likes Robots - 6DOF Arm ROS2 Control.
- [31] MoveIt! Tutorials - Move Group Python Interface Tutorial.
- [30] AutomaticAddison - Cartesian Path Planning ROS 2 Jazzy (YouTube).
- [25] ROS Robotics Learning - Inverse Kinematics.
- [28] YouTube - ROS Robotic Arm Inverse Kinematics Tutorial.

- [10] RoboticsBackend - Step-by-step guide to install ROS on Raspberry Pi and best practices for performance optimization.
- [8] AutomaticAddison - How to create a catkin workspace step-by-step.
- [24] GazeboSim - Explain ROS control and its use for robot movement with MoveIt.
- [22] Instructables - Step-by-step guide for building a DIY robotic arm, including motor selection and torque calculation.
- [26] Picknik AI - What are the differences between joint space and cartesian space control in ROS MoveIt.
- [43] ModalAI Docs - Step-by-step guide to connect a USB camera to ROS and perform camera calibration with cameracalibrator.py.
- [1] Campus Rover Gitbook - Python code example for color-based object detection using OpenCV and cv_bridge in ROS.
- [32] MoveIt! Docs - Tutorial on creating a pick and place pipeline using MoveIt! Task Constructor in ROS.
- [10] RoboticsBackend - Step-by-step guide to install ROS on Raspberry Pi and best practices for performance optimization.

ผลงานที่อ้างอิง

1. Recognizing Objects Based on Color and Size using OpenCV ..., เข้าถึงเมื่อ สิงหาคม 6, 2025 https://campus-rover.gitbook.io/lab-notebook/fiiva/object_recognition_based_on_color_and_size
2. Process of achieving color and image recognition by myPalletizer AI Kit - ROS Discourse, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://discourse.ros.org/t/process-of-achieving-color-and-image-recognition-by-mypalletizer-ai-kit/26369
3. ROS Architecture and Concepts - Packt, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.packtpub.com/en-us/learning/how-to-tutorials/ros-architecture-and-concepts
4. ROS Wiki: Documentation, เข้าถึงเมื่อ สิงหาคม 6, 2025 http://wiki.ros.org/Documentation
5. ROS: Home, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.ros.org/
6. ROS TUTORIALS - YouTube, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.youtube.com/playlist?list=PLuteWQUGtU9BU0sQIVqRQa24p-pSBCYNv
7. ROS Beginner Tutorials - Projects - Open Robotics Discourse, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://discourse.openrobotics.org/t/ros-beginner-tutorials/3032
8. How to Create a ROS Workspace - Automatic Addison, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://automaticaddison.com/how-to-create-a-ros-workspace/
9. Installing and Configuring Your ROS Environment - ROS Wiki, เข้าถึงเมื่อ สิงหาคม 6, 2025 http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment
10. Install ROS Noetic on Raspberry Pi 4 - The Robotics Back-End, เข้าถึงเมื่อ สิงหาคม 6,

2025 https://roboticsbackend.com/install-ros-on-raspberry-pi-3/
11. Color Detection With OpenCV : r/ROS - Reddit, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.reddit.com/r/ROS/comments/1maslkm/color_detection_with_opencv/
12. Controlling a 6DOF Robot Arm with ros2_control | Mike Likes Robots, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://mikelikesrobots.github.io/blog/6dof-arm-ros2-control/
13. ROS or ROS2 for Robotics Beginner? - ROS Answers archive, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://answers.ros.org/question/315595/ros-or-ros2-for-robotics-beginner/
14. Should I learn ROS 1 or ROS 2? - YouTube, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.youtube.com/shorts/FWookNen2MM
15. Is ROS2 developed enough or should I stick with ROS1? - Ask - Stretch Forum, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://forum.hello-robot.com/t/is-ros2-developed-enough-or-should-i-stick-with-ros1/916
16. Should we use ROS1 or ROS2 : r/ROS - Reddit, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.reddit.com/r/ROS/comments/14db3ip/should_we_use_ros1_or_ros2/
17. Changes between ROS 1 and ROS 2 - ROS 2 Design, เข้าถึงเมื่อ สิงหาคม 6, 2025 http://design.ros2.org/articles/changes.html
18. ROS 2 Feature Comparison, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://micro.ros.org/docs/overview/ROS_2_feature_comparison/
19. Using ROS on Raspberry Pi: Best Practices - The Robotics Back-End, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://roboticsbackend.com/using-ros-on-raspberry-pi-best-practices/
20. DIY 3D printed robotic arm with AI - ARCTOS, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://arctosrobotics.com/
21. robotican/Tutorials/Arm manipulation - ROS Wiki, เข้าถึงเมื่อ สิงหาคม 6, 2025 http://wiki.ros.org/robotican/Tutorials/Arm%20manipulation
22. DIY Robotic Arm : 8 Steps (with Pictures) - Instructables, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.instructables.com/DIY-Robotic-Arm/
23. ROS MoveIt Robotic Arm Part 2 : Robot Controller : 6 Steps - Instructables, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.instructables.com/ROS-MoveIt-Robotic-Arm-Part-2-Robot-Controller/
24. Tutorial : ROS control - Gazebo, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://classic.gazebosim.org/tutorials?tut=ros_control
25. Inverse Kinematics - Robotics and ROS, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.rosroboticslearning.com/inverse-kinematics
26. Guide to Cartesian Planners in MoveIt | PickNik, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://picknik.ai/cartesian%20planners/moveit/motion%20planning/2021/01/07/guide-to-cartesian-planners-in-moveit.html
27. Cartesian Position Controller or Joint Position Controller for robotics manipulator - Reddit, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.reddit.com/r/robotics/comments/1gys7kt/cartesian_position_controller_or_joint_position/
28. Inverse Kinematics – ROS 2 Jazzy MoveIt Task Constructor - YouTube, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.youtube.com/watch?v=9oYhOwOKi94

29. [ROS Tutorials] ROS Control #Unit 1: Introduction to ROS Control - YouTube, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.youtube.com/watch?v=DX6mDcX7_4A
30. Cartesian Path Planning – ROS 2 Jazzy MoveIt Task Constructor - YouTube, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.youtube.com/watch?v=aENo0zrxrnA
31. Move Group Python Interface — moveit_tutorials Noetic documentation - GitHub Pages, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://moveit.github.io/moveit_tutorials/doc/move_group_python_interface/move_group_python_interface_tutorial.html
32. Pick and Place with MoveIt Task Constructor - MoveIt Documentation, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://moveit.picknik.ai/humble/doc/tutorials/pick_and_place_with_moveit_task_constructor/pick_and_place_with_moveit_task_constructor.html
33. Pick and Place Using MoveIt 2 and Perception – ROS 2 Jazzy - YouTube, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.youtube.com/watch?v=SqVqyGrzpag
34. ROS MoveIt Robotic Arm : 4 Steps - Instructables, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.instructables.com/ROS-MoveIt-Robotic-Arm/
35. usb_cam - ROS Wiki, เข้าถึงเมื่อ สิงหาคม 6, 2025 http://wiki.ros.org/usb_cam
36. ROS Package: usb_cam, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://index.ros.org/p/usb_cam/
37. Deep_Object_Pose/doc/camera_tutorial.md at master - GitHub, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://github.com/NVlabs/Deep_Object_Pose/blob/master/doc/camera_tutorial.md
38. ROS 2 Cameras — Isaac Sim Documentation - NVIDIA, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://docs.isaacsim.omniverse.nvidia.com/latest/ros2_tutorials/tutorial_ros2_camera.html
39. Tutorial : ROS Depth Camera Integration - Gazebo, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://classic.gazebosim.org/tutorials?tut=ros_depth_camera&branch=ros_depth_camera_preview%3C/p%3E
40. Camera Calibration with Python - OpenCV - GeeksforGeeks, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.geeksforgeeks.org/python/camera-calibration-with-python-opencv/
41. Camera calibration - Clover, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://clover.coex.tech/en/camera_calibration.html
42. camera calibration algorithm - ros - Robotics Stack Exchange, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://robotics.stackexchange.com/questions/90132/camera-calibration-algorithm
43. ROS Calibrate Cameras | ModalAI Technical Docs, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://docs.modalai.com/ros-calibrate-cameras/
44. OpenCV Basics for Robotics with ROS - How to read an image from Camera - YouTube, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.youtube.com/watch?v=nKu8w3gT6s8
45. OpenCV Image Processing (Python) - ROS Industrial Training - Read the Docs, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://industrial-training-master.readthedocs.io/en/melodic/_source/session5/Op

enCV-in-Python.html
46. OpenCV Image Processing (Python) - ROS Industrial Training - Read the Docs, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://industrial-training-master.readthedocs.io/en/latest/_source/session5/OpenCV-in-Python.html
47. 8. ROS+OpenCV Course — ArmPi Pro v1.0 documentation - Hiwonder Docs, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://docs.hiwonder.com/projects/ArmPi_Pro/en/latest/docs/14_ros_opencv.html
48. Raspberry Pi and ROS (Robotic Operating System) : 5 Steps - Instructables, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.instructables.com/Raspberry-Pi-and-ROS-Robotic-Operating-System/
49. Raspberry Pi optimization - ROS Answers archive, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://answers.ros.org/question/41815/
50. Performance evaluation of ROS on the Raspberry Pi platform as OS for small robots - Universidad Distrital, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://revistas.udistrital.edu.co/index.php/tekhne/article/download/14746/14702/72656
51. Get Started with Robot Operating System on Raspberry Pi - MATLAB & Simulink Example, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.mathworks.com/help/simulink/supportpkg/raspberrypi_ref/getting-started-with-robot-operating-system-ros-on-raspberry-pi-r.html
52. Camera Calibration with Example in Python | by Neeraj Krishna | TDS Archive | Medium, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://medium.com/data-science/camera-calibration-with-example-in-python-5147e945cdeb
53. Camera calibration through Kalibr | by SAJID HUSSAIN - Medium, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://medium.com/@smilesajid14/camera-calibration-through-kalibr-616d334b5a15
54. Controlling a 6DOF Robot Arm with ros2_control - YouTube, เข้าถึงเมื่อ สิงหาคม 6, 2025 https://www.youtube.com/watch?v=ZRrC6Hss01Y