รายงาน Final Report

สถาบันวิทยาการหุ่นยนต์ภาคสนาม มหาวิทยาลัยพระจอมเกล้าธนบุรี

โครงงานโมดูล 6 – 7 กลุ่มที่ 9

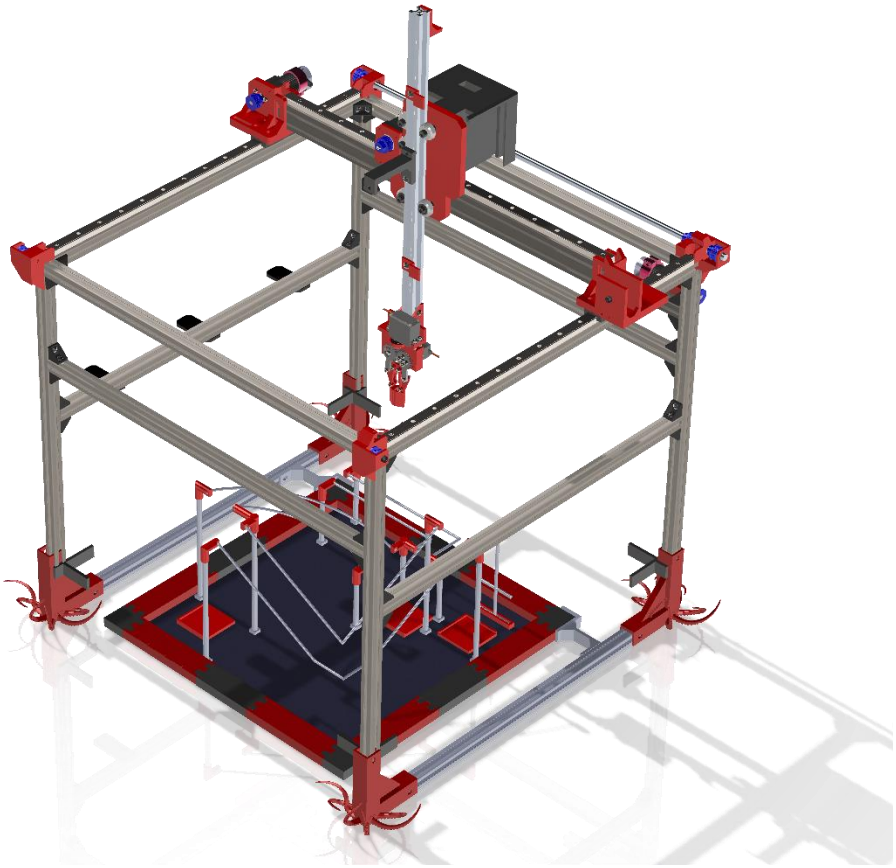
Project Module 6 – 7 Group 9

ผู้ดำเนินงาน

1. นางสาวณัฐชญา	ไพบูลย์สิทธิวงศ์	รหัสนักศึกษา 61340500020
2. นายนายเมธัส	มานวกุล	รหัสนักศึกษา 61340500053
3. นายสิทธิกร	วัฒนไชย	รหัสนักศึกษา 61340500067

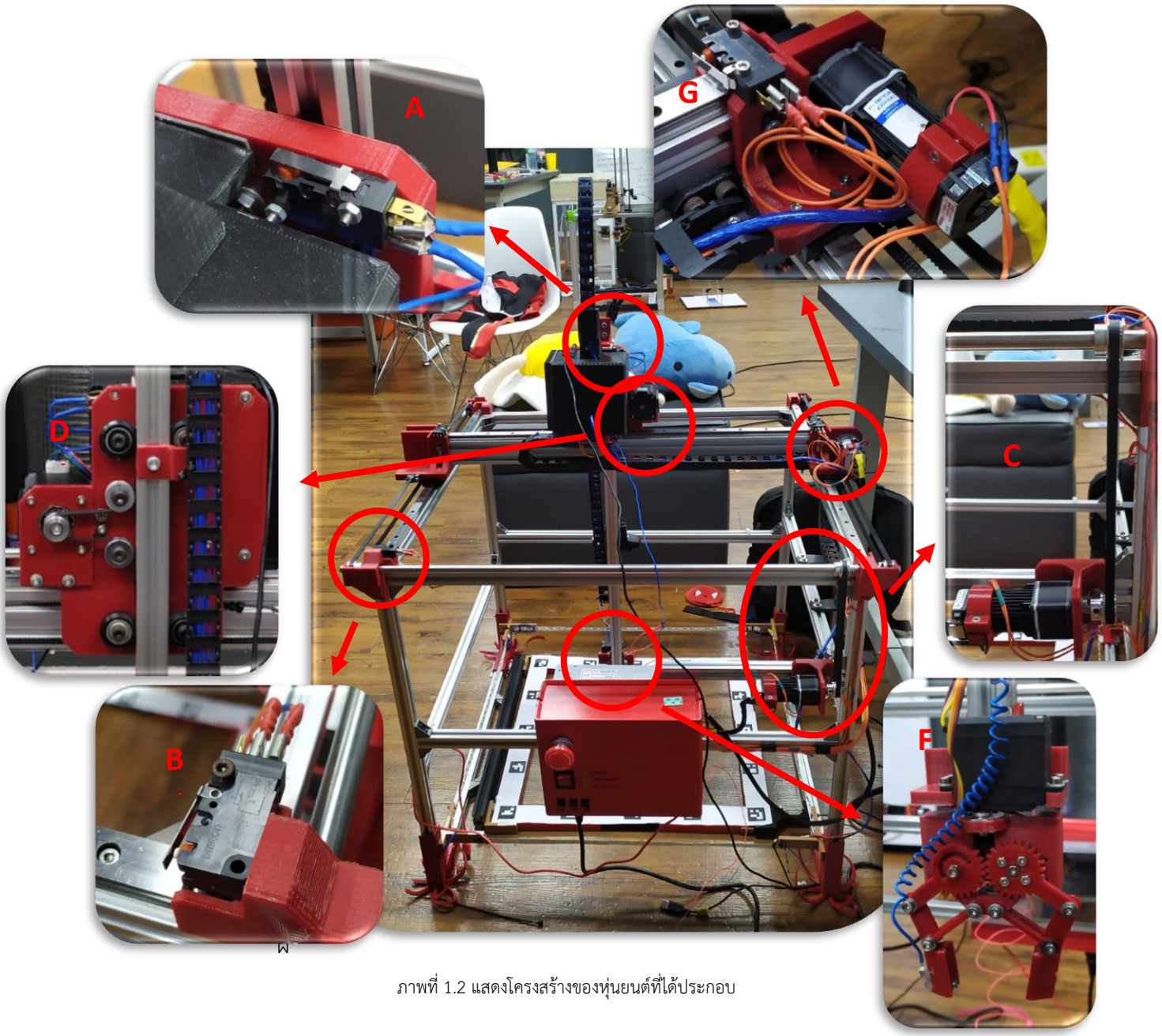
Structure + Gripper

1. **สรุปแบบโครงสร้าง และระบบส่งกำลังของหุ่นยนต์ที่ได้ออกแบบไว้**



ภาพที่ 1.1 แสดงโครงสร้างของหุ่นยนต์ที่ออกแบบในโปรแกรมจำลอง

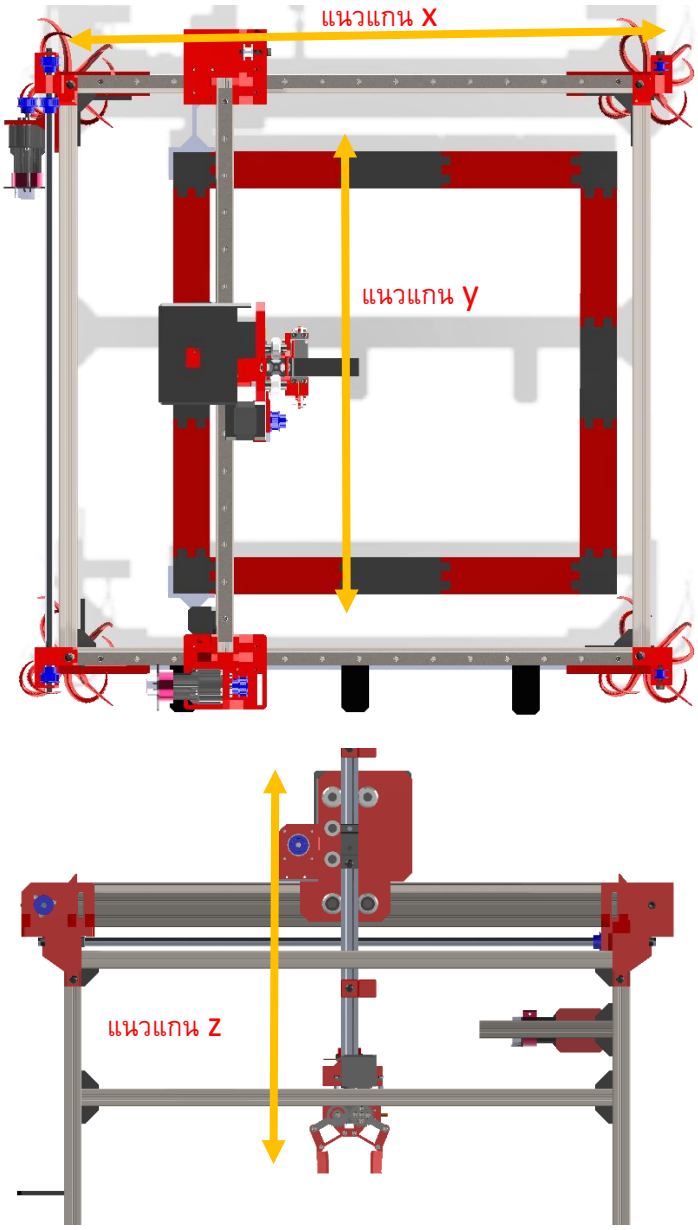
จากภาพที่ 1 จะเป็นการแสดงโครงสร้างและส่วนประกอบทั้งหมดของหุ่นยนต์ โดยผ่านโปรแกรมออกแบบ Soild Works ประมาณ 90 % ซึ่งมีบางชิ้นส่วนที่ไม่ได้ใส่เข้าไปในโปรแกรมจำลองเช่น ชิ้นส่วนอุปกรณ์อิเล็คทรอนิคส์ สายไฟ เป็นต้น ส่วนที่เป็นส่วนประกอบหลักได้มีแสดงอยู่ครบถ้วน เช่น ส่วนการขับเคลื่อนแนวแกน X Y และ Z ส่วนมือจับชิ้นงาน ตำแหน่งการติดตั้งกล้อง ตำแหน่งการติดตั้งไฟ LED และการยึดประกอบโครงสร้างต่างๆ



ภาพที่ 1.2 แสดงโครงสร้างของหุ่นยนต์ที่ได้ประกอบ

จากภาพที่ 1.2 จะเป็นการแสดงโครงสร้างของหุ่นยนต์ที่ได้ประกอบเสร็จสมบูรณ์เกือบ 100 เปอร์เซ็นต์ จากรูปภาพจะแสดงให้เห็นถึงส่วนประกอบหลัก ๆ เช่นระบบขับเคลื่อนในแนวแกน X , Y , Z ตามลำดับตัวอักษร C , G , D ระบบเซนเซอร์ที่ต้องอาศัยตำแหน่งติดตั้งแม่นยำ สามารถดูได้ตามแนวแกนซึ่งตามแนวแกน X , Y , Z จะอยู่ตามลำดับตัวอักษร B , G , A และส่วนประกอบกริปเปอร์

สรุประบบส่งกำลังของหุ่นยนต์



ภาพที่ 1.3 แสดงระบบส่งกำลังทั้งสามแนวแกน

ระบบการเคลื่อนที่ 3 แนวแกนรูปแบบนี้นิยมเรียกว่า หุ่นยนต์ **Cartesian** ซึ่งมีการเคลื่อนที่ 3 แนวแกนเหมาะกับทำงานที่สามารถสร้างหุ่นยนต์ให้มีขนาดใหญ่กว่า Workspace ได้ จึงสามารถทำงานกับการทำภารกิจ Challenger on Fires: Electric Stick ที่ต้องควบคุมแท่งโลหะให้เคลื่อนที่ไปตามรางได้โดยไม่เกิดการชน ซึ่งต้องใช้ความแม่นยำสูง ด้วยงบประงานที่จำกัดและวัสดุที่สามารถหาได้ตามท้องตลาดจึงได้เลือกใช้ ลิเนียร์ไกด์ ที่ทำให้เกิดการเคลื่อนที่ในแนวเส้นตรงแนวแกน x และ y เนื่องจากสามารถรับแรงได้หลายทิศทาง สัมประสิทธิ์ความเสียดทานต่ำ มีความแม่นยำสูง ดังนั้นลิเนียร์ไกด์จึงมีความเหมาะสมที่จะทำมาควบคุมให้เกิดการเคลื่อนที่ในแนวแกน x และ y ในส่วนแกน z เลือกใช้ล้อ V-Slot ในการควบคุมให้เกิดการเคลื่อนที่ในแนวแกน เนื่องจาก

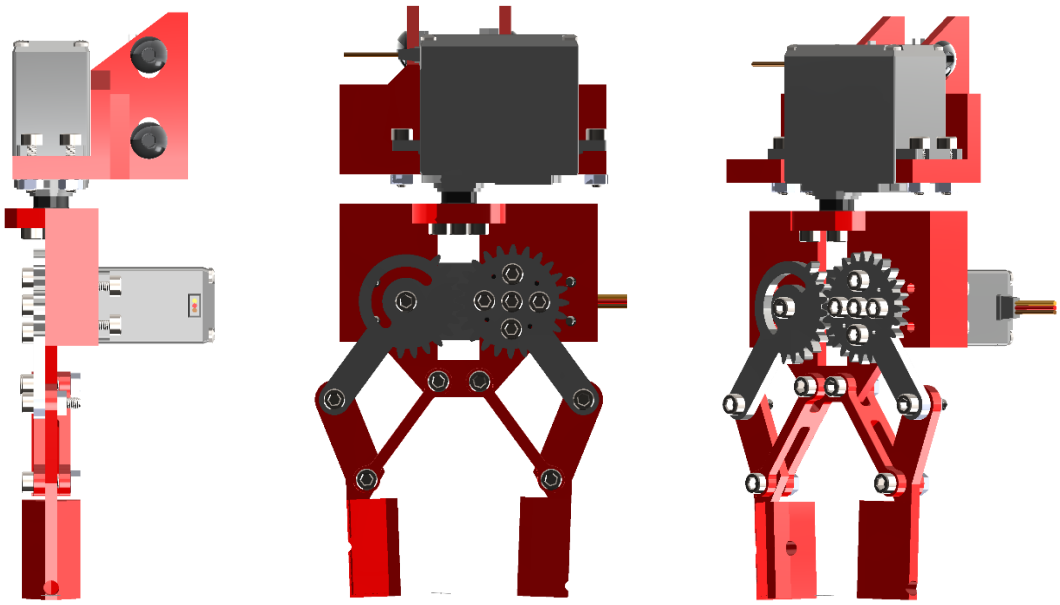
ไม่ได้รับแรงโดยตรงและสามารถปรับความแม่นยำได้ จึงเป็นเหตุผลเลือกใช้งานล้อ V-Slot ส่วนการส่งถ่ายแรงที่ทำให้เกิดการเคลื่อนที่นั้นเลือกใช้งาน สายพานไทม์มิ่ง เนื่องจากราคาอยู่ในงบประมาณที่พอเหมาะและอยู่ในระยะที่สามารถใช้งานได้แต่ก็มีข้อด้อยในด้านการทำงานซ้ำ ๆ และการใช้แรงดึงตึงเป็นเวลานานหรือบ่อย ๆ จะทำให้ความแม่นยำลดลง

แนวแกน x มีรูปแบบเคลื่อนที่เป็น มอเตอร์เป็นต้นกำลัง พูเลห์ถ่ายทอดกำลัง และสายพานไทม์มิ่งส่งกำลัง จากรูปที่ 1.3 จะเห็นได้ว่ามี shaft อยู่ 1 ชิ้นเนื่องจากในขณะขับเคลื่อนมีความจำเป็นที่จะต้องขับเคลื่อนทั้งสองฝั่งไปพร้อม ๆ เพื่อไม่ให้ฝั่งใดฝั่งหนึ่งต้องออกแรงมากเกินไป มีการใช้พูเลห์เพื่อสงกำลังจากมอเตอร์ไปที่ shaft เพื่อลดพื้นที่ในการทำงานไม่ให้ยืดออกไปมากเกินไป และยังทำให้งานดูเรียบร้อยขึ้นด้วย สังเกตได้ว่า พูเลห์ที่ shaft ตรงจุกที่รับแรงมาจากมอเตอร์จะอยู่ไกล้กับแบริ่งมากก็เพื่อไม่ให้ลดการโก่งงอของ shaft ด้วย

แนวแกน y รูปแบบการเคลื่อนที่เหมือนแกน x เพียงไม่ได้ใช้ พูเลห์ส่งกำลังไปที่ shaft ก่อนเท่านั้น

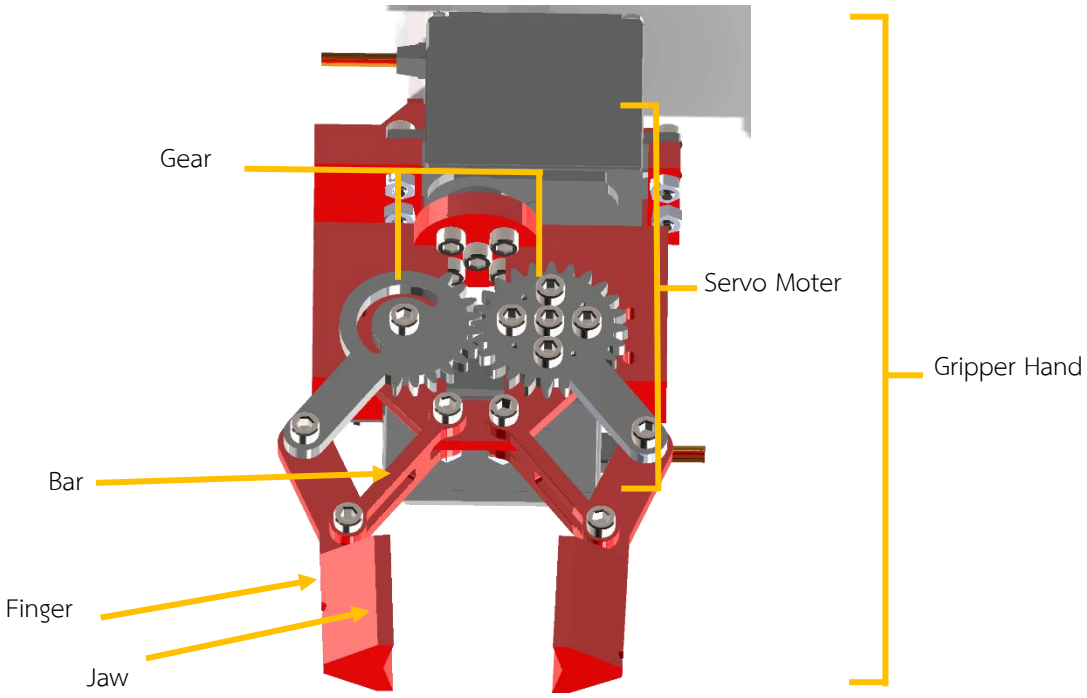
แนวแกน z มีรูปแบบเคลื่อนที่เป็น มอเตอร์เป็นต้นกำลัง พูเลห์ถ่ายทอดกำลัง และสายพานไทม์มิ่งส่งกำลัง เหมือนกันเพียงแต่จะแตกต่างก็ตรงที่ในแนวแกนนี้ใช้ล้อ V-Slot และพูเลห์รับน้ำหนักโดยตรงจึงมีโอกาศที่สายพานไทม์มิ่งมีโอกาศยืดได้กว่าแนวแกนอื่น

สรุประบบและส่วนต่าง ๆ ของ กริปเปอร์



รูปที่ 1.4 แสดงโครงสร้างของกริปเปอร์ที่ออกแบบในโปรแกรมจำลอง

จากรูปที่ 1.4 แสดงให้เห็นถึงภาพรวมของงกริปเปอร์ที่ออกแบบในโปรแกรมจำลอง โดยภารกิจคือต้องเคลื่อนที่แท่งเหล็กที่มีรัศมีประมาณ 5 มิลลิเมตร โดยมีการเคลื่อนที่ 3 แกน (แกน **x, y, z)** ไปตามเส้นทางที่กำหนดให้ซึ่งหากแท่งเหล็กที่ เคลื่อนที่ไปนั้นสัมผัสกับราง ภารกิจจะล้มเหลวทันทีและระยะห่างระหว่างรางมีความกว้างประมาณ 5 เซนติเมตร ดังนั้นจึงต้องคิดออกแบบมือจับที่สามารถหยิบจับแท่งเหล็กและเคลื่อนที่ไปตามรางได้อย่างปลอดภัย



รูปที่ 1.5 แสดงส่วนประกอบต่าง ๆ ของ กริปเปอร์

จากรูปที่ 1.5 ส่วนประกอบของ **Gripper** แบ่งเป็น 5 ส่วนหลัก ๆ

1. **Servo motor** คือภาพรวมทั้งหมดของ **Gripper** ซึ่งประกอบไปด้วย **servo motor** 2 ตัว โดยแบ่งหน้าที่เป็น แกนหมุน และหน้าที่ขยับเคลื่อนที่ **Finger** ให้หนีบวัตถุ

2. **Gear** ทำหน้าที่ส่งกำลังจาก **servo motor** ไปที่ **Finger** เพื่อให้เกิดการหนีบวัตถุ

3. **Bar** มีหน้าที่ช่วยรับ-ส่งแรง กำหนดทิศทางและระยะการเคลื่อนที่ของ **Finger**

4. **Finger** คือส่วนที่สามารถเคลื่อนที่เข้าออกใช้ในการหนีบจับวัตถุซึ่งมีลักษณะการเข้าออกรูปแบบขนานโดยมี **Gear** และ **Bar** เป็นชิ้นส่วนที่ช่วย รับ - ส่ง กำลังเพื่อให้เกิดการหนีบวัตถุได้

5. **Jaw** คือส่วนที่เชื่อมกับ **Finger** มีหน้าที่สัมผัสกับวัตถุส่วนนี้จะมีแรงเสียดทานมากเป็นพิเศษเพื่อให้สามารถหยิบจับชิ้นงานได้แม่นยำ ไม่หลุดออกจาก **Gripper** ระหว่างเคลื่อนที่ ซึ่งแต่ละ **Jaw** จะมี 2 เส้นสัมผัสกับวัตถุ ดังนั้นเมื่อเกิดการหนีบวัตถุเข้ามาทั้งสองด้านจะทำ ให้สามารถหยิบจับชิ้นงานได้แม่นยำ

2. **ปัญหาที่พบระหว่างการประกอบ**

ปัญหาที่ 1 ชิ้นส่วนที่ขึ้นรูปมาจากเครื่อง 3D printer มี error ในตัวดังนั้นสิ่งที่ออกแบบในโปรแกรมกับสิ่งที่ได้มาจริง ๆ ไม่เท่ากันจึงต้องใช้วิธีการตะไบ หรือเจาะขยายรูเพิ่มให้สามารถนำมาใช้งานได้

ปัญหาที่ 2 สายพานไทม์มิ่ง หย่อนทำให้เกิดการสลิปของสายพานกับพูเลห์ เนื่องจากสายพานเมื่อถูกใช้งานไปนาน ๆ จะยืดตัวได้เป็นเรื่องปกติจึงต้องมีการตรวจสอบและปรับค่าความตึงของสายพานอยู่เป็นระยะ ๆ

ปัญหาที่ 3 ไม่สามารถใส่ แบริ่งเข้ากับเพลาได้ เนื่องจากไม่ได้ตะไบบริเวณส่วนปลายของเพลาให้ละเอียดเนียนก่อน ดังนั้นเมื่อใส่แบร่งจะใส่ไม่เข้าเพราะบริเวณปลายมีเศษร่องรอยจากการตัดทำให้ขนาดเกิน

ปัญหาที่ 4 ประกอบชิ้นส่วน 3D Printer เข้ากับตัวบล็อกของลิเนียร์ไกด์แล้วมีความฝืดเกิดขึ้น เนื่องจากเกิด error จากชิ้นส่วน 3D Printer ที่ขึ้นรูปมาทำให้เมื่อขันน็อตเข้าไป น็อตไปขัดเส้นทางวิ่งของเม็ดลูกปืนทำให้เกิดความฝืด

ปัญหาที่ 5 บล็อกของลิเนียร์ไกด์เสีย ทำให้เกิดการติดขัดเมื่อติดตั้งและทดลองเคลื่อนที่ เนื่องจากเม็ดลูกปืนไม่สามารถวิ่งวนรอบได้ เม็ดลูกปืนค้าง

ปัญหาที่ 6 ชิ้นส่วน 3D print มีปัญหาด้านความแข็งแรงของวัสดุ จึงต้องเพิ่มองศาชดเชยและต้องถอดประกอบทดสอบหลายครั้ง

ปัญหาที่ 7 การจัด Alignment ค่อนข้างยากเนื่องจากพื้นที่ใช้ประกอบหุ่นยนต์ไม่เสมอกัน

3. **การปรับปรุงต่าง ๆ ที่แตกต่างจากแบบที่ได้ออกแบบไว้ พร้อมเหตุผลประกอบ**

การปรับปรุงที่ 1 เพิ่มความสูงของหุ่นยนต์โดยการสร้างขาเสริมความสูงให้กับหุ่นยนต์ทั้ง 4 ขาเนื่องจากพื้นที่การทำงานไม่เพียงพอ

การปรับปรุงที่ 2 ใช้วัสดุทดแทนยึดสนามให้อยู่กับที่ เนื่องจากชิ้นส่วนที่ใช้ยึดสนามมี error และไม่ได้ตรวจสอบสนามจริงทำให้ชิ้นส่วนนั้นใช้ไม่ได้ อีกทั้งยังต้องย้ายให้ตัวสนามมาอยู่บริเวณด้านหน้าของหุ่นยนต์ จึงจัสามารถถ่ายรู้ได้

การปรับปรุงที่ 3 คือการเปลี่ยนแปลงมอเตอร์ ในช่วง Milestone 2 จาก RSPro-454-0883 เป็น 120RPM Faulhaber Coreless 17w Encode Meter เนื่องจากมีปัญหาตอน drive มีประกายไฟเกิดขึ้นและค่าความถี่ที่คำนวณได้ ไม่สามารถนำมาใช้จริงได้ทำให้การ Contron ได้ยากขึ้นไปอีก จึงจำเป็นต้องเปลี่ยนมอเตอร์ส่งผลให่ที่ติดยึดมอเตอร์ต้องออกแบบใหม่ทั้งหมด

การปรับปรุงที่ 4 ในช่วง Milestone 1 คือเปลี่ยนจากการใช้งาน proximity sensor มาใช้งานเป็น Switch Sensor เนื่องจากมีความราคาถูกว่าและเหมากับการทำงานมากกว่า

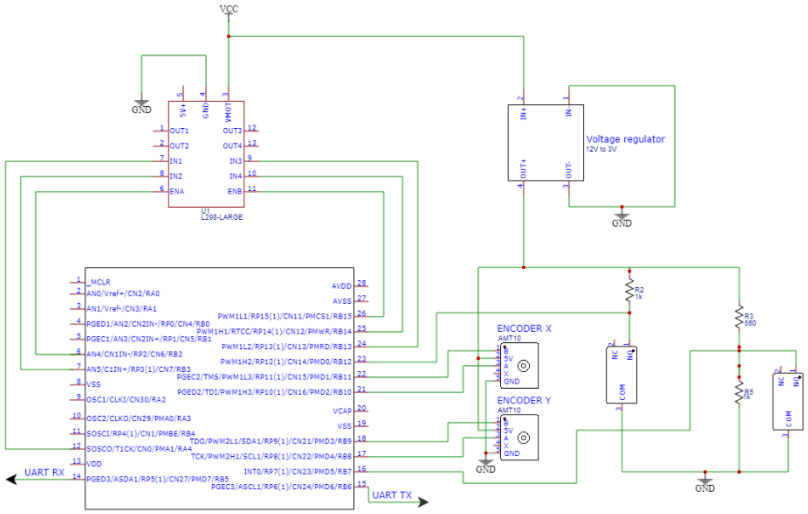
การปรับปรุงที่ 5 กริปเปอร์ที่มี finger กับ jaw ที่เป็นชิ้นเดียวกันอยู่เนื่องจากขึ้นรูปโดยใช้ 3D Printer วัสดุเป็นพลาสติกจึงมีการให้ตัวหรือยืดหยุ่นสูง ทำให้ต้องปรับองศาเผื่อด้วย

4. **สิ่งที่ยังแก้ไม่ได้ หรือสิ่งที่ต้องการปรับปรุงเพิ่มเติมหากมีโอกาส**

    1. การใช้วัสดุทดแทนยึดสนามให้อยู่กับที่ เปลี่ยนเป็นขึ้นรูปวัสดุ **3D Print** ที่นำมาใช้งานได้พอดี
    2. เพิ่มระบบตึงสายพานเพื่อป้องกันสายพานหย่อน
    3. ขาหุ่นยนต์ที่ต่อเพิ่มความสูงของหุ่นยนต์ เปลี่ยนไปใช้ อลูมิเนียมโปรไฟร์ 20**x20** แทน
    4. การ **black Drive** ตามแนวแกน **x , y** ของมอเตอร์

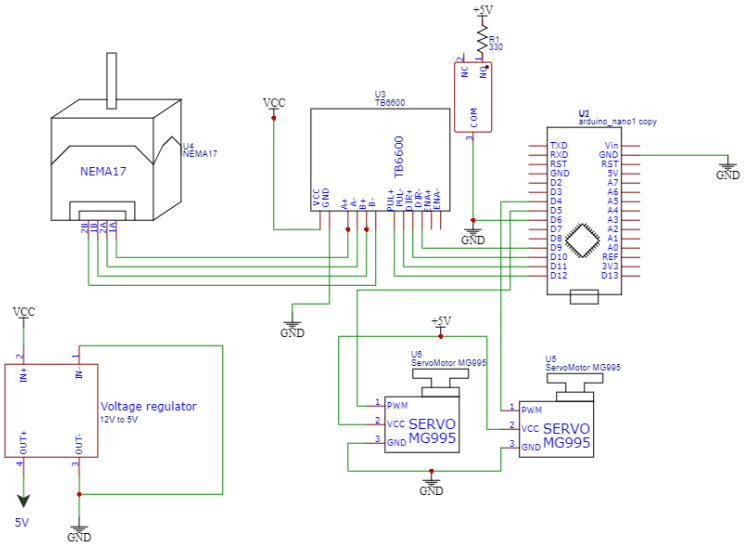
Electronics

1. **สรุปแบบวงจรไฟฟ้าของหุ่นยนต์ที่ได้ออกแบบไว้**



รูปที่ 2.1 วงจรสำหรับแกน **X Y**

ไฟเลี้ยงจะมาจาก power supply 12 โวลต์ สำหรับส่วนที่ต้องใช้ 5 โวลต์จะใช้ Voltage regulator ในการลดความต่างศักย์ให้พอกับการใช้งาน ในการออกแบบวงจรเพื่อให้ง่ายต่อการสร้างวงจรจึงเลือกให้ขาของ dsPIC ที่สามารถรับได้ถึง 5 โวลต์ต่อกับ encoder ซึ่งทำงานและมีสัญญาณ output ที่ 5 โวลต์ และขาที่เหลือต่อกับวงจรที่เหลือโดยดังรูป จะเห็นได้ว่าจากวงจรของ limits switch มีขาหนึ่งที่ไม่พบว่าไม่มีที่สำหรับ 5 โวลต์ จึงต้องต่อกับตัวต้านทานเพื่อให้ความต่างศักย์ลดลงจนเหลือต่ำกว่า 3.3 โวลต์ การต่อ limit switch จะต่อแบบ pull up เพื่อให้ตอนที่บอร์ดยังปิดอยู่แล้วมีคำสั่งให้ set home จะถือว่าทำเสร็จเรียบร้อยแล้ว และมอเตอร์จะไม่ถูกสั่งให้วิ่งในเวลาที่ไม่ต้องการให้วิ่ง



รูปที่ 2.2 วงจรแกน Z

สำหรับแกน **z** จะใช้ Arduino nano เป็น microcontroller สำหรับควบคุมการเคลื่อนที่ โดยสำหรับการควบคุมการเคลื่อนที่ของ Stepper motor และ servo motor จะเลือกขาที่สามารถใช้ PWM ทำให้ได้ผลตามวงจร สำหรับไฟเลี้ยงจะมาจาก power supply เช่นเดียวกัน และเข้าสู่บอร์ดที่ทำให้ความต่างศักย์เปลี่ยนเป็น 5 โวลต์ ในส่วนของไฟเลี้ยงของ Arduino จะมีไฟเลี้ยงมาจาก mini USB ซึ่งจะต่อไว้ตลอดเวลาสำหรับการสื่อสารทาง UART

2. ปัญหาที่พบระหว่างการนำไปใช้จริง

ปัญหาที่ 1 บอร์ดวงจรแรกที่ใช้เป็นบอร์ดไข่ปลา 2 ด้าน ซึ่งบัดกรียากกว่าเพราะทำให้วงจรเชื่อมกันได้ง่าย ทำให้ในระหว่างการทำงานจะต้องแก้ไขหลายครั้ง เมื่อเปลี่ยนเป็น 1 ด้านก็สามารถใช้ได้อย่างราบรื่น

ปัญหาที่ 2 ในบางครั้งการออกแบบไม่จำเป็นต้องทำถึงขนาดที่ได้คิดไว้ ซึ่งหากทำอย่างที่คิดทำให้เปลืองงบและพื้นที่จึงต้องคิดให้กระชับขึ้น

ปัญหาที่ 3 บางครั้งการบัดกรีและไม่ทำความสะอาดฟลัคซ์ให้ดีจะทำให้วงจรมีโอกาสช็อตกันได้และทำให้การสั่งการ logic แปลกไป

3. การปรับปรุงต่าง ๆ ที่แตกต่างจากแบบที่ได้ออกแบบไว้ พร้อมเหตุผลประกอบ

การปรับปรุงที่ 1 จาก milestone2 ลดจำนวน limit switch ที่ใช้เพราะเห็นว่าใช้เฉพาะในการ set home ก็พอแล้ว

การปรับปรุงที่ 2 ในช่วงแรกใช้มอเตอร์ RSPRO แต่หลังจากการทำ control พบว่า มอเตอร์ RSPRO มีค่าที่ estimate parameter แล้วได้ผลที่ไม่ดีและค่าที่ไม่ตอบสนองตามที่สั่ง จึงเลือกที่จะเปลี่ยนไปใช้ Faulhaber

การปรับปรุงที่ 3 ในช่วงแรกใช้บอร์ด VNH2SP30 เพราะใช้มอเตอร์ RSPRO ซึ่งในระหว่างทำ บอร์ดดังกล่าวเกิดการลัดวงจรทำให้บอร์ดไหม้และไม่สามารถใช้งานได้และในระหว่างที่พิจารณาที่จะเปลี่ยนได้เปลี่ยนมอเตอร์เป็น Faulhaber แล้ว จึงเลือกใช้ l298 เพราะกระแสที่รองรับได้อยู่ในช่วงที่ใช้งานกับมอเตอร์ดังกล่าวได้

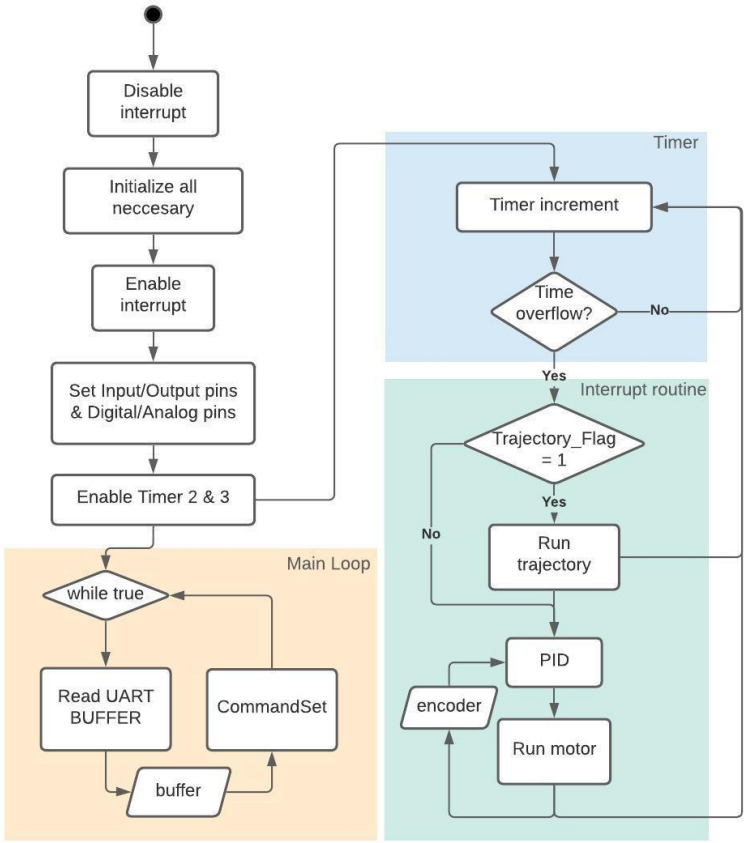
การปรับปรุงที่ 4 เพิ่ม limit switch แกน **z** ไปเพราะลืมใส่ในวงจร และเปลี่ยนลำดับขาที่เขาสู่ stepper driver เพราะเกิดข้อผิดพลาดระหว่างการบัดกรี

4. สิ่งที่ยังแก้ไม่ได้ หรือสิ่งที่ต้องการปรับปรุงเพิ่มเติมหากมีโอกาส

1. หากมีเวลาเพิ่มเติม คิดว่าการรวมทั้งสองแกนในบอร์ดเดียวมีโอกาสเป็นไปได้

Embedded System + Control

1. **สรุป Flow การทำงานของ Embedded System พร้อมอธิบายการทำงานแต่ละ Block พร้อมเหตุผลประกอบ**



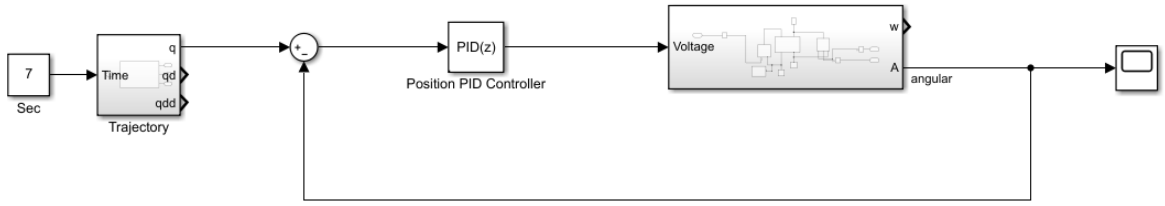
รูปที่3.1 การทำงานในส่วน embedded และ control

Embedded

การทำงานหากมองจากภาพกว้างจะเริ่มจากการ ปิด interrupt เพื่อทำการตั้งค่าการทำงานต่างๆ ได้แก่การตั้งค่า Output oscillator การสื่อสารทาง UART QEI1 QEI2 interrupt timer 1, 2 และ 3 ซึ่งมีไว้สำหรับ foreground interrupt timer สำหรับการควบคุมมอเตอร์ การขับเคลื่อนมอเตอร์แบบ PWM และการปริ้นค่าที่ใช้สำหรับการควบคุมมอเตอร์ตามลำดับ เมื่อการตั้งค่าเสร็จสิ้นจะนำไปสู่การเปิดให้ timer ทำงานซึ่งจะสั่งให้เฉพาะ timer 2 และ 3 ทำงานและสำหรับ timer 1 จะสั่งให้ทำงานหลังจากการ set home สุดท้ายแล้วจะเป็น background loop ที่มีไว้อ่านค่าที่ผ่านการสื่อสารทาง UART

1. สำหรับการตั้งค่า PLL หรือ output oscillator จะตั้งค่าให้มี output 40Mhz
2. Timer 2 กำหนดให้มอเตอร์ทำงานที่ 1300 ticks ต่อรอบการทำงานและให้ prescaler เป็น 8 เพื่อไม่ให้ความถี่ของมอเตอร์มากเกินความถี่ที่เหมาะสมซึ่งจากการคำนวณจะหาความถี่ดังกล่าวได้ 487.79 Hz
3. Timer 1 กำหนดให้มี prescaler 8 และมี period คือ 10417 ซึ่งจะได้ 479.98 Hz
4. Timer 3 กำหนดให้มี prescaler 8 และมี period คือ 50000 ซึ่งจะได้ 100 Hz
5. สำหรับ QEI1 และ 2 จะมีการตั้งค่าเหมือนกันที่ให้เป็นรูปแบบการทำงานแบบ 4x และมี clock divider ที่หารด้วย 1

Control



รูปที่3.2 block diagram การทำงานในส่วนของ control

ส่วนของ control จะทำงานอยู่ใน Interrupt เนื่องจากต้องการ Sampling เวลาที่คงที่ในการควบคุมผลที่จะออกมา มีลักษณะเป็น การที่ทำ Trajectory Planning ขึ้นมาเพื่อให้ได้ ตำแหน่งที่หุ่นควรจะต้องเคลื่อนที่ไปในแต่ละเวลา t จากนั้นจะไปทำงานใน ฟังก์ชัน PID เพื่อคำนวณค่าที่จะสั่งไปยังมอเตอร์แล้ว อ่านค่า position จาก encoder กลับมาคำนวณใน Position PID Controller ด้วย

- **PID**

  ทั้งแกน x และ y มีค่า gain ดังนี้ K_p = 8.0, K_i = 0, K_d = 5.0 และตั้งให้ตัดการทำงานเมื่อ error อยู่ที่ 10 มิลลิเมตร

- **Trajectory**

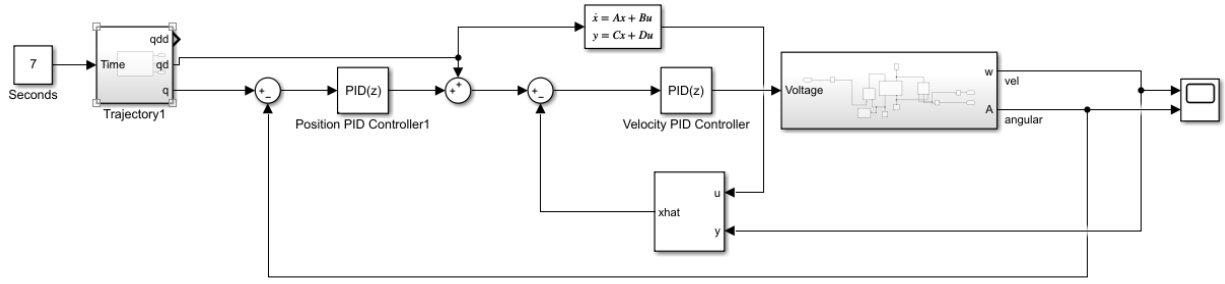
  กำหนดให้ทำงานใน 6 วินาทีและตัดการทำงานใน 6.5 วินาที

2. ปัญหาที่พบระหว่างการนำไปใช้จริง

เมื่อสั่งการให้เคลื่อนที่ด้วย PID ในระยะ 10 มิลลิเมตร จะไม่สามารถทำงานได้เพราะจะเข้าสู่การตัดการทำงานทันที

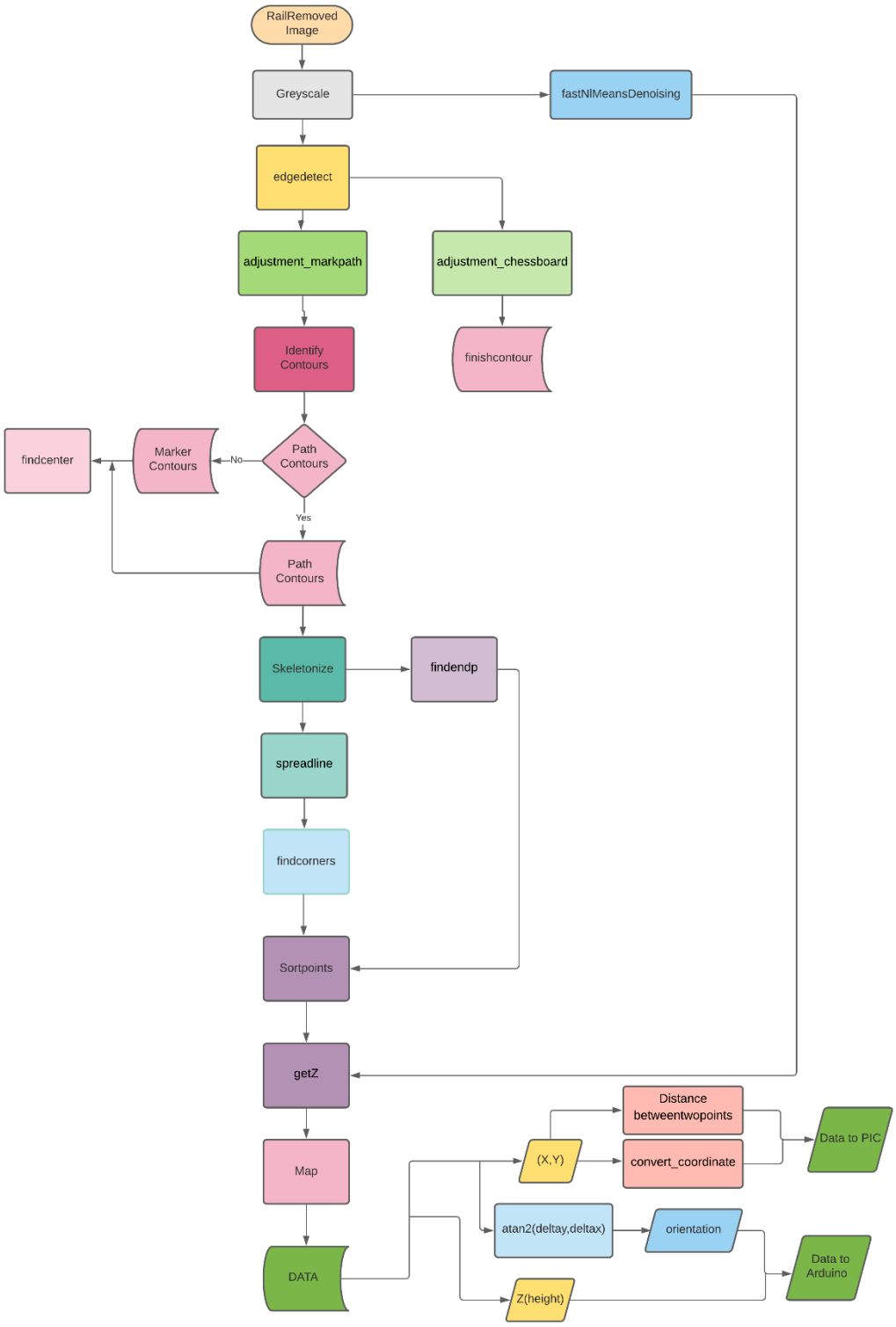
3. สิ่งที่ยังแก้ไม่ได้ หรือสิ่งที่ต้องการปรับปรุงเพิ่มเติมหากมีโอกาส

การปรับปรุงในส่วนของ control จะเป็นการปรับส่วนของ Control Policy โดยจะมีการ feed forward ค่าของ Velocity จาก Trajectory เข้าไปเพิ่มเติม โดยรวมจะเป็น การควบคุมแบบ Cascade Control โดยมี Velocity Control และ Position Control โดยค่าที่ได้จาก Position PID Controller จะไป รวมกับค่า Velocity Trajectory เพื่อปรับค่าความเร็วที่ต้องการสั่ง และ จะ feedback ความเร็ว กลับมาจะผ่าน State Estimator เช่น Kalman Filter เพื่อประมาณค่าที่ sensor จะอ่านได้ แม่นยำมากขึ้น ซึ่งคิดว่า วิธีนี้จะช่วยลดปัญหาการสั่นและกระตุกของการ ทำ Trajectory ที่มี แค่ Position Control เพียงอย่างเดียวได้



รูปที่3.3 block diagram การทำงานแบบ Cascade Control

Image Processing



รูปที่ 4.1 flow การทำงานในส่วนของ Image processing

1. สรุป **Flow** การทำงานของ **Image Processing** พร้อมอธิบายการทำงานแต่ละ **Block** พร้อมเหตุผลประกอบ

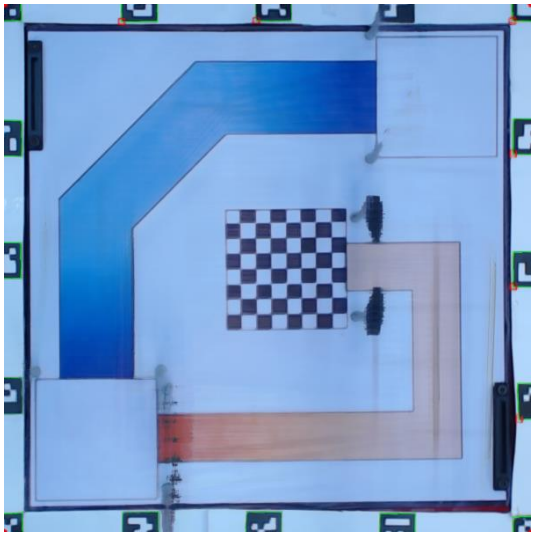
การถ่ายภาพใช้ สัญลักษณ์ที่เรียกว่า Aruco ในการถ่ายภาพ

เนื่องจากเป็นอุปกรณ์ที่สะดวกแล้วเข้าใจง่าย โดยทำการนำสัญลักษณ์จำนวน 16 ID มาครอบสนามเพื่อ จับส่วนที่เป็นสนามแล้วทำ ใช้ฟังก์ชัน perspective เพื่อ ตัดและตีกรอบส่วนที่เป็นสนามในภาพมาใช้งาน โดยภาพที่จะใช้งานได้ต้องสามารถ จับ aruco ได้อย่างน้อย ขอบละ 2 สัญลักษณ์ เพื่อนำสัญลักษณ์ มาหาจุดศูนย์กลางแล้วลากเส้นตรงเชื่อม แล้วหาจุดตัดของเส้นตรงแต่ละขอบ จะได้ มุม 4 มุม และหาขอบของรูปได้

รูปที่ 4.2 aruco ที่ทำการ perspective transform

การลบรางออกจากสนามเพื่อ นำไปประมวลผลต่อ

ภาพที่ถ่ายได้จะถูกนำมาทำการ median เพื่อนำรูปที่มีการเปลี่ยนแปลงตลอดเมื่อเทียบกับรูปอื่น ๆ ออกทำให้รูปที่เป็นส่วนของรางส่วนมากถูกนำออกไป



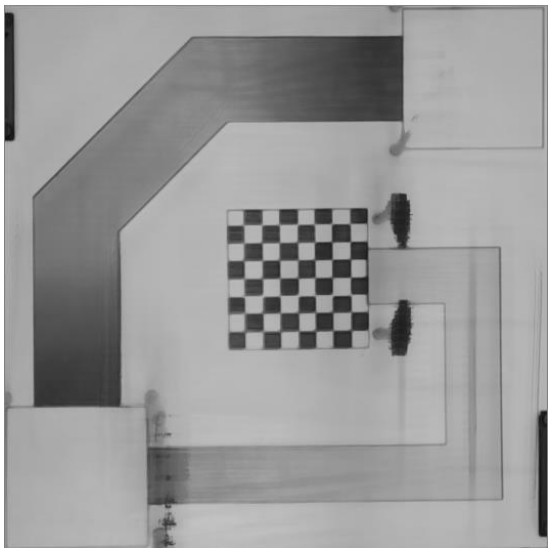
รูปที่ 4.3 รูปที่ทำการลบรางส่วนมากออกไป

การทำงานของ Image Processing ส่วนของการถ่ายภาพ

หลังจากระบบสั่งหุ่นถ่ายภาพทั้งหมด 45 ภาพ ซึ่งได้มากจากระยะที่หุ่นยนต์ขยับแล้วเห็น aruco ครบทุกด้าน ถ้าขยับมากกว่านี้จะทำให้ aruco ถูกบังมากเกินไปจนใช้งานไม่ได้

Greyscale

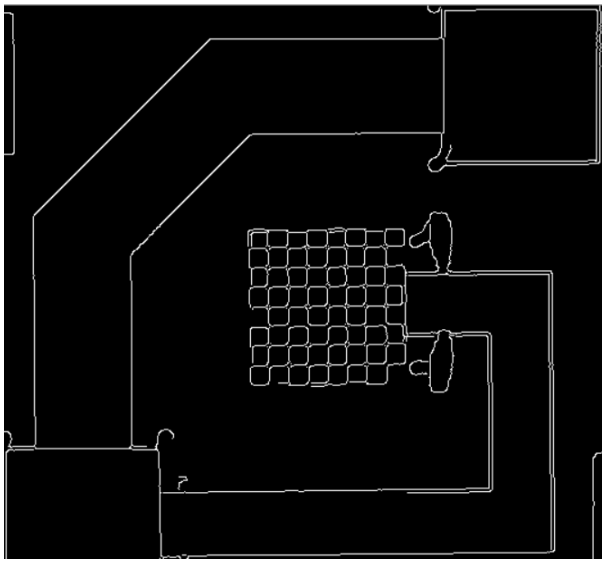
-นำรูปที่ลบรางออกแล้วมาทำเป็น grey scale เพื่อให้ทำงานได้สะดวก



รูปที่ 4.4 รูปที่ผ่านการอ่านแบบ greyscale

edgedetect

-ใช้ ฟังก์ชัน Canny เพื่อหาขอบภายในรูปภาพ โดยปรับ ค่า min และ max ของ intensity ได้



รูปที่ 4.5 ใช้ฟังก์ชัน Canny หาขอบภายในรูปสนาม

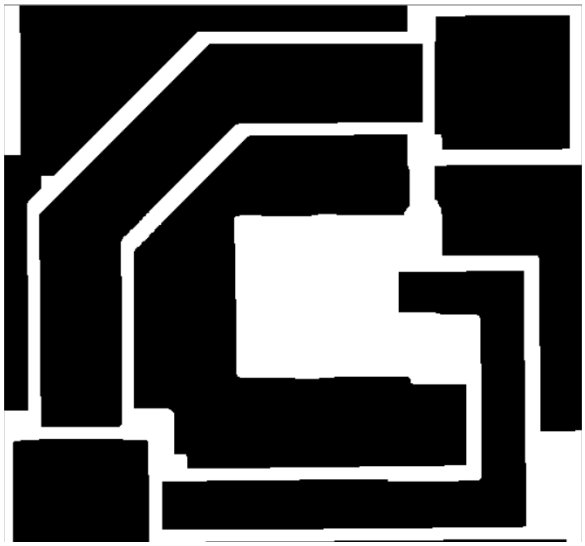
adjustment_markpath

-ใช้ ฟังก์ชัน Opening เพื่อทำให้ noise เล็กๆ หายไป

-จากนั้นทำการ dilation เพื่อให้เส้นขอบใหญ่ขึ้นและเชื่อมติดกัน

-จากนั้นทำการ erosion เพื่อลบขอบให้เล็กลงเพียงพอ และลบ noise ใน contours ให้หมด

-จะได้ contours ของ path และ marker มา โดยทำการแยกด้วยการเปรียบเทียบช่วงของพื้นที่ของ contours

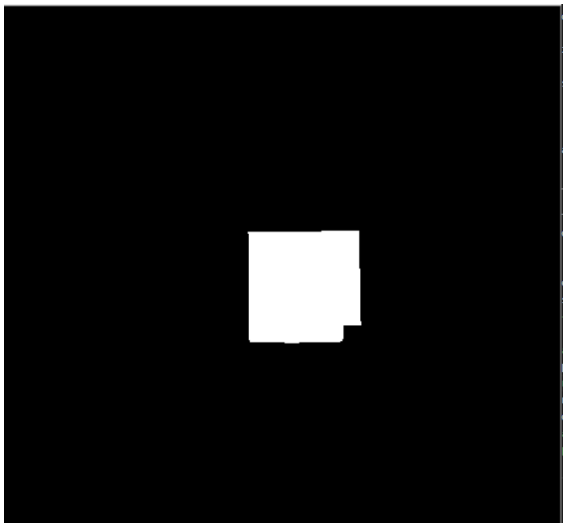


รูปที่ 4.6 ภาพสนามที่ผ่านการทำ Morphological Transformations

adjustment_chessboard

-ทำงานเช่นเดียวกับ ฟังก์ชัน adjustment_markpath แต่ทำเพื่อให้ เหลือแค่ contour ของเส้นชัยที่เป็นตารางหมากรุก

-จะได้ contours ของ ตารางหมากรุก มา



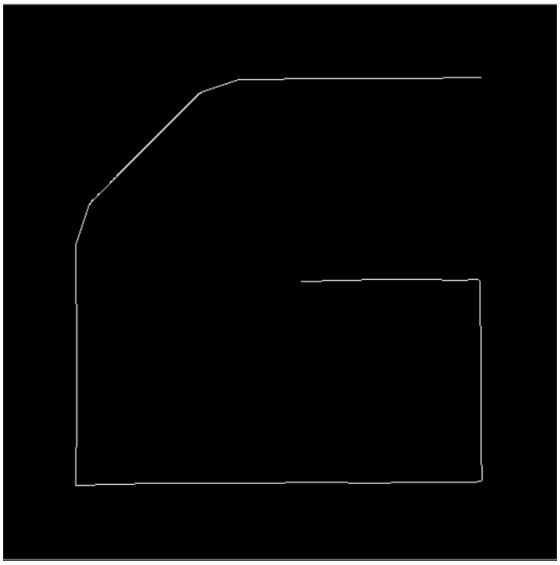
รูปที่ 4.7 ภาพสนามที่ผ่านการทำ Morphological Transformations

findcenter

-หาจุดศูนย์กลางของ contours โดยใช้ ฟังก์ชัน moments เพื่อทำการเฉลี่ยหาจุดตรงกลาง

Skeletonize

-ใช้ฟังก์ชัน ximproc.thinning เพื่อทำ skeletonize กับ pathcontours เพื่อให้ได้จุดของpath ออกมา



รูปที่ 4.8 ภาพสนามที่ผ่านการทำ skeletonize

findendp

-ใช้การนำ kernel ขนาด 3x3 ที่มี เลข 10 ตรงกลางแล้ว รอบข้างเป็น 1 มา convolution กับ contours ของ path จุดที่เป็นจุดปลาย จะได้ค่าเป็น 11

spreadline

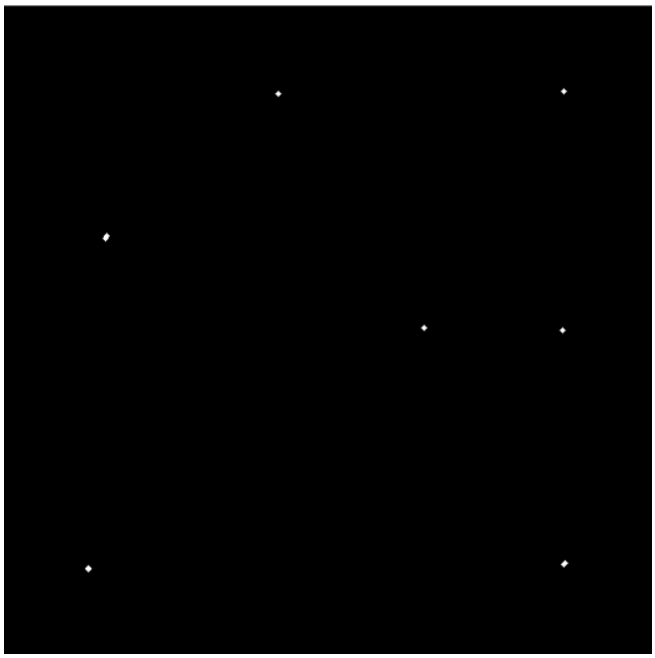
-นำ path ที่ผ่านการทำ skeletonize มา ใช้ฟังก์ชัน bitwise.and กับ contours ของpathนั้นก่อนเพื่อให้เส้นยาวเท่าปลาย contours ของpath

-จากนั้นทำการลากเส้นจาก path ขยายต่อไปถึงกลาง marker contours และ finishcontour ที่อยู่ติดกัน

findcorners

-ใช้ฟังก์ชัน approxPolyDP เพื่อหาว่ามุมตรงไหนบ้างในรูป skeleton ที่ผ่านการspreadlineแล้ว

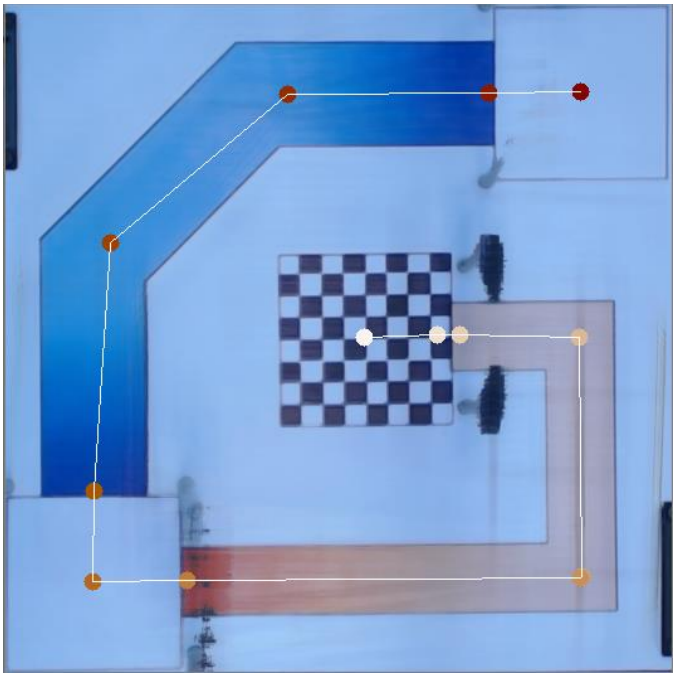
-จะได้ จุดที่เป็นมุมของ contours นั้นออกมา



รูปที่ 4.9 หามุมของรูป skeleton

Sortpoints

-นำจุดที่ได้ทั้งหมดมาเพื่อทำการเรียง จุดเป็นการเคลื่อนที่ทั้งหมดของหุ่น โดยเริ่มเรียงจากจุดแรกเป็นจุดศูนย์กลางของ finishcontour แล้วเรียงไปจนสุดปลาย marker ซึ่งหลังจากนั้นจะทำการเรียงกลับจุดภายใน list ทั้งหมด



รูปที่ 4.10 จุดที่ถูกเรียง

getZ

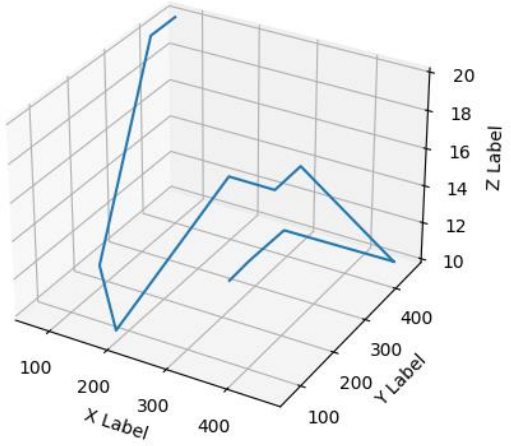
-นำจุดทั้งหมด ใน path แต่ละpath ไปหา ค่าIntensity จากภาพที่ผ่านการ denoise แล้ว แล้วหาค่า min max intensity ใน path นั้นเก็บไว้

-นำจุดที่อยู่ใน list ของ sortpoint มาหา ค่า Intensity จากภาพที่ denoise แล้วเช่นกัน

map

-นำค่า Intensity จากจุดใน list ของ sortpoint ซึ่งได้จากฟังก์ชัน getZ มา mapเทียบกับค่า min,max ของแต่ละ path ที่เก็บไว้

และค่าความสูงที่มีค่า 20 ซม. และ 10 ซม.ด้วยเพื่อให้ได้ค่า ความสูงจริงๆในช่วง 10-20 ซม.



รูปที่ 4.11 เส้นทางการเคลื่อนที่ใน แกน xyz

2.ปัญหาที่พบระหว่างการนำไปใช้จริง

ปัญหาที่ 1 แสงในแต่ละเวลามีผลต่อรูปที่ได้จากการถ่ายภาพของหุ่นยนต์จึงทำการนำไป LED มาติดให้ต่ำลงเพื่อให้สนามได้รับแสงเพียงพอแล้วลดเงาลง

ปัญหาที่ 2 gradient ที่ได้จากภาพที่ถ่ายจากหุ่นยนต์ เมื่อนำมาวาดกราฟในพิกัด xyz พบว่าค่อนข้างไม่ linear เลย จึงแก้ไขโดยการเลือกบางจุดมาแล้วทำการ threshold ด้วยค่าหนึ่งเพื่อปรับความสูงเอาเอง

3. สิ่งที่ยังแก้ไม่ได้ หรือสิ่งที่ต้องการปรับปรุงเพิ่มเติมหากมีโอกาส

การปรับปรุงที่ 1 ความสามารถในการทำซ้ำ เช่น การที่ถ่ายรูปมาแล้วใช้งานได้เช่นเดียวกับรูปก่อนน้อยครั้งที่สามารถทำได้ คิดว่าควรจะแก้จากการเลือกวิธีการในการทำ Image Processing ใหม่โดยหาฟังก์ชันที่ใช้parameter ที่ต้องปรับน้อยเนื่องจากตอนนี้มีฟังก์ชันที่ต้องการปรับ patameter เช่น canny, diration, erotion เป็นต้น

การปรับปรุงที่ 2 ตอนนี้หลังการทำ Image processing เสร็จ path ที่ได้บางครั้ง จะพบจุดที่วกกลับมาจากปลายทางบางครั้งสามารถแก้ได้และบางครั้งไม่สามารถแก้ไขได้

System Integration

1. สรุป Flow การรับส่งข้อมูลระหว่าง Image Processing กับ Microcontroller ทั้งหมด พร้อมอธิบายการทำงานแต่ละ Block พร้อมเหตุผลประกอบ

โดยเบื้องต้นการรับส่งข้อมูลระหว่าง Image processing และ Microcontroller จะใช้ Protocol ร่วมกันดังนี้

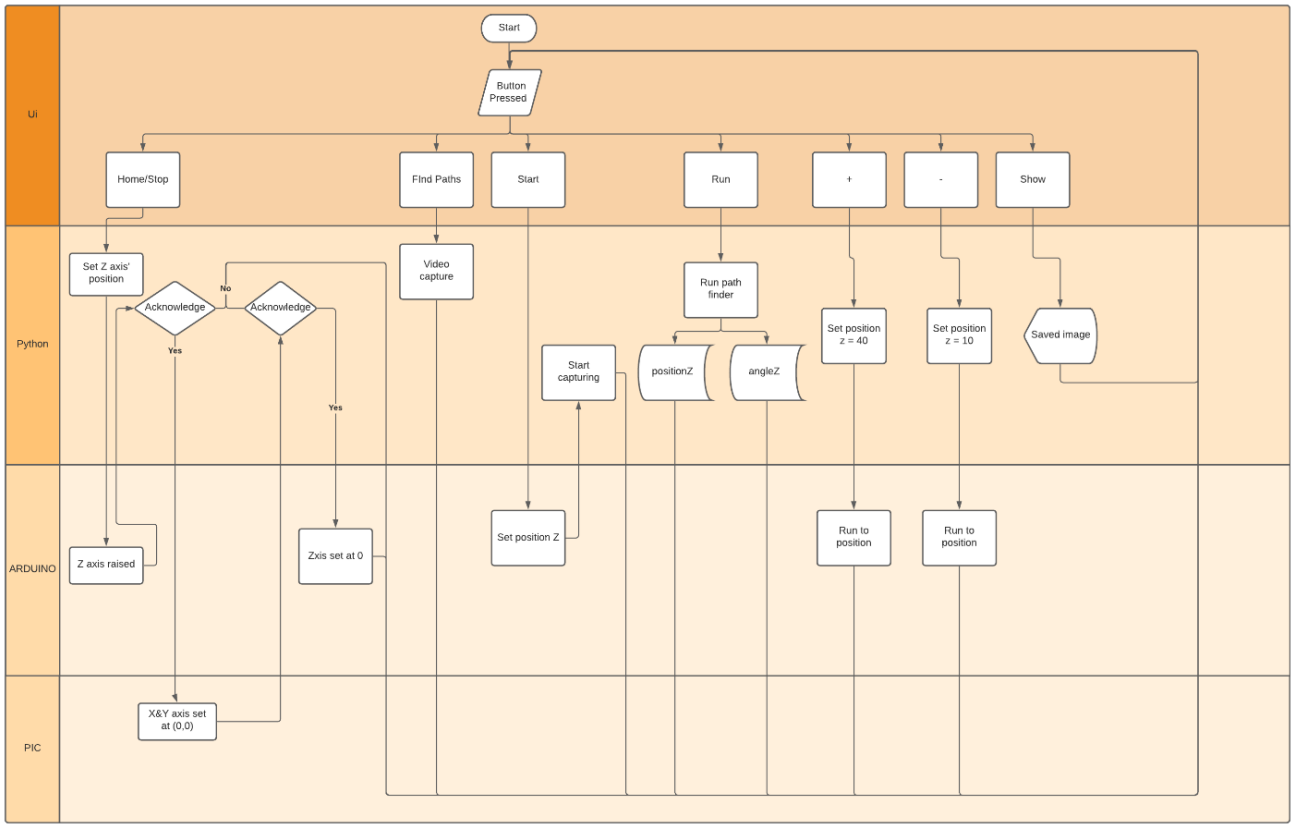
Header (2 bytes)	Data (8 bytes)	Check sum (1 bytes)

ตารางที่ 5.1 Protocol ที่ใช้ในการสื่อสาร

โดย Header คือ 0xFF ทั้ง 2 bytes และ check sum คือ การบวกแต่ละ byte เข้าด้วยกันและเอาแค่ low byte

- การส่ง set home จะส่ง data ทั้งหมดเป็น 0 ดังนั้น check sum ก็จะเป็น 0 ทั้งกับการส่งให้ Arduino และ PIC
- การส่ง position ให้แกน X & Y จะรับค่า position เป็นมิลลิเมตรและนำไปคูณ 100 เพื่อเก็บทศนิยมสองตำแหน่งสุดท้ายไปด้วย จะได้ data ไม่เกิน 2 byte เมื่อรับค่าไปแล้วจะนำค่าไปหาร 100 เพื่อนำไปใช้งานต่อไป
- การส่ง position ให้แกน Z จะรับค่า position(rf) เป็นมิลลิเมตรและนำไปคูณ 100 เพื่อเก็บทศนิยมสองตำแหน่งสุดท้ายไปด้วย จะได้ data ไม่เกิน 2 byte เมื่อรับค่าไปแล้วจะนำค่าไปหาร 100 เพื่อนำไปใช้งานต่อไป องศาการหมุนของกริปเปอร์ที่จะเก็บค่าเป็น int จะได้ค่าข้อมูลเพียง 1 byte และ คำสั่งหนีบหรือปล่อย ซึ่งถ้าสั่งเป็น 0 จะทำการหนีบและถ้ามากกว่า 1 จะสั่งให้ปล่อย
- การส่ง trajectory ให้แกน X & Y จะรับค่า ขนาดของ rf เป็นมิลลิเมตรและองศาของ rf นำไปคูณ 100 เพื่อเก็บทศนิยมสองตำแหน่งสุดท้ายไปด้วย ทั้งสองข้อมูลจะได้ data ไม่เกิน 2 byte เมื่อรับค่าไปแล้วจะนำค่าไปหาร 100 เพื่อนำไปใช้งานต่อไป
- การส่ง trajectory ให้แกน Z จะรับค่า ขนาดของ rf เป็นมิลลิเมตร นำไปคูณ 100 เพื่อเก็บทศนิยมสองตำแหน่งสุดท้ายไปด้วย ทั้งสองข้อมูลจะได้ data ไม่เกิน 2 byte เมื่อรับค่าไปแล้วจะนำค่าไปหาร 100 เพื่อนำไปใช้งานต่อไป และองศาการหมุนของกริปเปอร์ที่จะเก็บค่าเป็น int จะได้ค่าข้อมูลเพียง 1 byte

การรับข้อมูลของ **PIC** และ Arduino จะรับเข้ามาทีละ byte และนำมาเก็บเป็น array ไว้และอ่านทีเดียวเมื่อข้อมูลครบ โดยทั้งสองจะวนเช็คข้อมูลว่ามีเข้ามาไหมและอ่านข้อมูลใน background loop



รูปที่ 5.1 การทำงานในส่วนของ Integration

2. ปัญหาที่พบระหว่างการนำไปใช้จริง

ปัญหา 1 ในบางครั้งถ้าลืมทำการ set home ก่อนจะไม่สามารถใช้งานบางฟังก์ชันได้ และบางครั้งการกด set home ทำงานไม่ครบทุกขั้นตอน

ปัญหา 2 รูปที่ถ่ายระหว่างเปิดโปรแกรม บางครั้งไม่สามารถนำขึ้นมาแสดงผลได้

3. **สิ่งที่ยังแก้ไม่ได้ หรือสิ่งที่ต้องการปรับปรุงเพิ่มเติมหากมีโอกาส**

การปรับปรุงที่ 1 ในส่วนของ UI ต้องการให้สามารถแสดงตำแหน่ง point ที่ส่งไปขึ้นในรูปได้ว่าตอนนี้หุ่นทำงานไปยังตำแหน่งไหนของสนามแล้ว

การปรับปรุงที่ 2 ในส่วนของ UI ควรเพิ่มเติมอีกหลายฟังก์ชั่นให้สามารถ debug การทำงานต่างๆได้ง่ายขึ้น

การปรับปรุงที่ 3 ปรับปรุงการสื่อสารให้มีการตอบกลับในหลายส่วนมากขึ้นเพื่อให้รับรู้ว่าได้รับข้อมูลที่ครบถ้วนและถูกต้อง