การสร้างฟังก์ชันบล็อกของพีแอลซีเพื่อประมวลผลชุดคำสั่ง RAPID สำหรับการควบคุมหุ่นยนต์อุตสาหกรรม

PLC Function Block Creation to Execute RAPID Instructions for Industrial Robot Control

นายณัฐนนท์ รอดสุทธิ์

นายสิรดนัย สิงหา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมอัตโนมัติ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2562

PLC Function Block Creation to Execute RAPID Instructions for Industrial Robot Control

NATTHANON RODSUT
SIRADANAI SINGHA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF ENGINEERING IN AUTOMATION ENGINEERING
FACULTY OF ENGINEERING
KING MONGKUT'INSTITUTE OF TECHNALOGY LADKRABANG
ACADEMIC YEAR 2019

หัวข้อปริญญานิพนธ์	การสร้างฟังก์ชันบล็อกของพีแอลซีเพื่อประมวลผลชุดคำสั่ง RAPID สำหรับการควบคุมหุ่นยนต์อุตสาหกรรม
นักศึกษาผู้จัดทำ	นายณัฐนนท์ รอดสุทธิ์ **รหัสนักศึกษา** 59010433
	นายสิรดนัย สิงหา **รหัสนักศึกษา** 59011401
ชื่ออาจารย์ที่ปรึกษา	รศ.ประภาษ อุคคกิมาพันธุ์
	รศ.ดร.พิทยา ปานนิล
	ผศ.ดร.กฤษณ์ เสมอพิทักษ์
ปีการศึกษา	2562

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอเทคนิคการสร้างฟังก์ชันบล็อกสำหรับการประมวลผลคำสั่ง RAPID ที่ได้จากโปรแกรม RobotStudio เพื่อให้ง่ายต่อการประยุกต์ใช้พีแอลซีเป็นตัวควบคุมหุ่นยนต์อุตสาหกรรม โดยใช้ 3 คำสั่ง (ได้แก่ MoveL, MoveJ, และ MoveC) เพื่อกำหนดเคลื่อนที่ของหุ่นยนต์สการารุ่น Mitsubishi MELFA RH-P2 เป็นกรณีศึกษาสำหรับขั้นตอนการโปรแกรม RAPID ใน RobotStudio และใช้ Microsoft Excel ในการจัดเรียงและแสดงข้อมูลของคำสั่ง RAPID ในรูปแบบ 32 ตัวอักษร ซึ่งเป็นรูปแบบที่เหมาะสมกับการประมวลผลในพีแอลซีรุ่น Mitsubishi Q03UDECPU ด้วย MELSOFT Gx Work2 จากกรณีศึกษาที่ได้กำหนด มีการสร้าง 4 ฟังก์ชันบล็อก (ที่เรียกว่า Text_Rapid, Read_Rapid, Check_Move, และ Check_Target) นอกจากกนี้ ยังมีการสร้างส่วนหน้าจอเอชเอ็มไอโดยใช้ MELSOFT GT Designer อีกด้วย ผลการทดสอบแสดงให้เห็นว่าฟังก์ชันบล็อกของพีแอลซีที่สร้างขึ้น สามารถประมวลผลชุดคำสั่ง RAPID ได้อย่างถูกต้องเป็นไปตามความต้องการ

คำสำคัญ: ฟังก์ชันบล็อก, หุ่นยนต์ในอุตสาหกรรม, พีแอลซี, RobotStudio, RAPID

Thesis Title	PLC Function Block Creation to Execute RAPID Instructions for Industrial Robot Control	
Students	Mr. Natthannon Rodsut	**Student ID** 59010433
	Mr. Siradanai Singha	**Student ID** 59011401
Advisors	Assoc.Prof. Prapart Ukakimaparn	
	Assoc.Prof.Dr. Pittaya Pannil	
	Asst.Prof.Dr. Krit Smerpitak	
Academic Year	2019	

ABSTRACT

In order to provide ease of applying a programmable logic controller (PLC) to perform as an industrial robot controller, this thesis presents a technique to create function blocks for executing RAPID instructions from RobotStudio. To create robot movements for a selective compliance assembly robot arm (SCARA) modeled Mitsubishi MELFA RH-P2, three move instructions (including MoveL, MoveJ, and MoveC) are defined as an illustrative case study for RAPID programming in RobotStudio. The Microsoft Excel is employed to arrange and present the RAPID instructions in 32-character format, which is suitable for further processing in the PLC modeled Mitsubishi Q03UDECPU by using MELSOFT Gx Work2. Four function blocks (named Text_Rapid, Read_Rapid, Check_Move, and Check_Target) are created for the PLC to execute the RAPID Instructions from the specified case study. In addition, the human-machine interface (HMI) screens are also built by using MELSOFT GT Designer. Test results confirm that the proposed PLC function blocks can correctly execute the RAPID instructions in accordance with the requirements.

Keywords: Function Block, Industrial Robot, PLC, RobotStudio, RAPID

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงได้ด้วยดี เนื่องด้วยความอนุเคราะห์จากหลาย ๆ ฝ่าย โดยเฉพาะยิ่ง รศ.ประภาษ อุคคกิมาพันธุ์ รศ.ดร.พิทยา ปานนิล และผศ.ดร.กฤษณ์ เสมอพิทักษ์ รวมไปถึงอาจารย์ประจำสาขาวิศวกรรมอัตโนมัติ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังทุก ๆ ท่าน

ขอกราบขอบพระคุณสำหรับ คุณพ่อ คุณแม่ ครอบครัว ศิษย์เก่าของหลักสูตรวิศวกรรมอัตโนมัติทุกท่านและเพื่อน ๆ ที่คอยช่วยเหลือ ห่วงใยและเป็นกำลังใจมาโดยตลอด จนทำให้โครงงานนี้สำเร็จลุล่วงไปได้ด้วยดี

ความรู้ใด ๆ ที่จะเป็นประโยชน์ต่อการศึกษาค้นคว้าวิจัยต่อไปของผู้รักการศึกษาค้นคว้าวิจัยนั้น ซึ่งนับว่าเป็นความดีประการหนึ่ง ข้าพเจ้าขอมอบอุทิศให้แด่ท่านเจ้าคุณทหารและคุณหญิงเลี่ยมผู้ซึ่งมอบที่ดินสำหรับสร้างสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังแห่งนี้

นายณัฐนนท์ รอดสุทธิ์

นายสิรดนัย สิงหา

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญของปริญญานิพนธ์

ในปัจจุบันงานด้านอุตสาหกรรม ได้มีการนำเอาแขนกลมาใช้ในกระบวนการผลิตต่าง ๆ อย่างแพร่หลาย เพื่อใช้ปฏิบัติงานแทนมนุษย์ ในลักษณะงานที่มนุษย์ไม่สามารถปฏิบัติได้ หรือปฏิบัติได้ยาก ยกตัวอย่าง เช่น งานที่ต้องมีกระบวนการผลิตอย่างต่อเนื่องตลอด 24 ชั่วโมง หรือในงานที่ต้องทำซ้ำ ๆ กันตลอดเวลา นอกจากนี้ยังมีลักษณะงานที่อันตราย และหนักเกินกว่าที่มนุษย์จะทำได้ เช่น งานที่เกี่ยวข้องกับสารเคมี งานยกและจัดเรียงสินค้าที่มีน้ำหนักมาก งานเชื่อมโลหะ งานผลิตและประกอบชิ้นส่วนที่ต้องการคุณภาพ และมาตรฐานการผลิตแบบเดียวกัน เป็นต้น ด้านอุตสาหกรรมจึงได้มีการนำเทคโนโลยีระบบอัตโนมัติ (Automation Technology) เข้ามาใช้งาน เพื่อเพิ่มคุณภาพการผลิตและทำงานในส่วนที่มนุษย์ไม่สามารถทำได้ หนึ่งในเทคโนโลยีนั้น ได้แก่ หุ่นยนต์อุตสาหกรรม [1] หุ่นยนต์อุตสาหกรรมจะเป็นการทำงานเลียนแบบร่างกายของมนุษย์ โดยจะเลียนแบบ เฉพาะส่วนของร่างกายที่จะนำไปใช้ประโยชน์ในอุตสาหกรรม นั่นคือช่วงแขนของมนุษย์ ดังนั้นจึงมีการเรียก หุ่นยนต์อุตสาหกรรมด้วยคำว่า “แขนกล” ซึ่งก็หมายถึงหุ่นยนต์อุตสาหกรรมที่ทำงานเปรียบเสมือนกับแขนของมนุษย์ และในปัจจุบันยังสามารถออกแบบการเคลื่อนที่ของหุ่นยนต์ได้อย่างถูกต้องแม่นยำ โดยออกแบบผ่านโปรแกรม RobotStudio [2]

ดังนั้นในปริญญานิพนธ์เล่มนี้จึงทำขึ้น เพื่อที่จะนำรูปแบบการเคลื่อนที่ ที่อยู่ในรูป RAPID Instructions [3] ที่ได้จากการออกแบบโดยใช้โปรแกรม RobotStudio มาประยุกต์ใช้งานกับพีแอลซี (Programmable Logic Control: PLC) [4]-[6] ในการควบคุมหุ่นยนต์อุตสาหกรรม โดยการเขียนฟังก์ชันบล็อกขึ้นมา เพื่อที่จะประมวลผลชุดคำสั่ง RAPID ให้สามารถทำงานได้บนพีแอลซี เพื่อที่จะใช้ในการควบคุมหุ่นยนต์อุตสาหกรรมต่อไป

1.2 วัตถุประสงค์ของโครงการ

การสร้างฟังก์ชันบล็อกของพีแอลซีรุ่น Mitsubishi Q03UDECPU ด้วยโปรแกรม MELSOFT Gx Work2 [7] เพื่อประมวลผลชุดคำสั่ง RAPID ที่ได้จากการออกแบบการเคลื่อนที่ของหุ่นยนต์สการา (Selective Compliance Assembly Robot Arm: SCARA) โดยใช้โปรแกรม RobotStudio

1.3 ขอบเขตของปริญญานิพนธ์

1. ทำการออกแบบการเคลื่อนที่ของหุ่นยนต์สการารุ่น Mitsubishi MELFA RH-P2 ซึ่งเป็นหุ่นยนต์ที่มีจำนวน Joint และ Link เท่ากับ 4 และ 2 ตามลำดับ โดยเน้นการเคลื่อนที่ 3 รูปแบบ ได้แก่ การเคลื่อนที่แบบเส้นตรง (MoveL), การเคลื่อนที่อย่างรวดเร็วโดยไม่ใช่เส้นตรง (MoveJ) และ

การเคลื่อนที่เป็นเส้นโค้ง (MoveC) เพื่อให้หุ่นยนต์เคลื่อนที่ตามข้อกำหนดโดยใช้โปรแกรม RobotStudio

2. นำ RAPID Instruction ที่ได้จากการออกแบบรูปแบบการเคลื่อนที่ด้วยโปรแกรม RobotStudio มาจัดการข้อมูลด้วยโปรแกรม Microsoft Excel เพื่อให้ขนาดของข้อมูลมีขนาดเท่ากับ 32 ตัวอักษร เพื่อเตรียมบันทึกค่าลงในโปรแกรม MELSOFT Gx Work2 โดยสิ่งสำคัญที่ได้จากข้อมูล RAPID ได้แก่ robtarget, coordinate, move, speed, zone และ tool เพื่อนำไปใช้ในการควบคุมหุ่นยนต์

3. สร้างฟังก์ชันบล็อกในพีแอลซี Mitsubishi รุ่น Q03UDECPU โดยการใช้โปรแกรม MELSOFT Gx Work2 ในการสร้าง ซึ่งจะใช้วิธีการเขียนแบบ Structured text [8] โดยจะสร้างฟังก์ชันบล็อกเพื่อใช้ในการประมวลผลขึ้นมา 4 ฟังก์ชันบล็อกได้แก่ ฟังก์ชันบล็อกที่ใช้ในการเก็บข้อมมูล RAPID ทั้งหมด(Text_Rapid), ฟังก์ชันบล็อกที่ใช้ในการอ่านค่าการประกาศตัวแปรทั้งหมดของข้อมูล RAPID (Read_Rapid), ฟังก์ชันบล็อกที่ใช้ในการอ่านค่าที่ใช้ในการเคลื่อนที่ (Check_Move) และ ฟังก์ชันบล็อกที่ใช้ในการจัดเรียงลำดับการเคลื่อนที่ (Check_Target) เพื่อใช้ในการประมวลผลชุดคำสั่ง RAPID

4. สร้างหน้าจอเอชเอ็มไอ (Human-Machine Interface: HMI) [9] โดยใช้โปรแกรม MELSOFT GT Designer เพื่อตรวจสอบการประมวลผลของโปรแกรม ซึ่งจะใช้ในการตรวจสอบความถูกต้องของพิกัดการเคลื่อนที่ ที่ได้จากการประมวลผลข้อมูล RAPID

5. สามารถตรวจสอบความถูกต้องในการประมวลผลชุดคำสั่ง RAPID ในฟังก์ชันบล็อก และข้อมูลที่ได้ โดยการจำลองการทำงานในพีแอลซี โดยใช้ MELSOFT Gx Work2 (Simulation) และสามารถแสดงค่าออกมาได้

1.4 วิธีดำเนินงาน

1. ศึกษาข้อมูลเกี่ยวกับหุ่นยนต์อุตสาหกรรม
2. ศึกษารายละเอียดและการทำงานของโปรแกรม RobotStudio
3. ศึกษารูปแบบโครงสร้างของ RAPID Programming Language
4. สร้างฟังก์ชันบล็อกในพีแอลซี เพื่อประมวลชุดคำสั่ง RAPID
5. สร้างหน้าจอเอชเอ็มไอ เพื่อตรวจสอบและแสดงผลข้อมูล
6. ทดสอบการทำงานของโปรแกรมพร้อมแสดงค่าของข้อมูล

จากขั้นตอนการดำเนินงานข้างต้นสามารถแสดงเป็นแผนงานได้ดังตารางที่ 1.1

ตารางที่ 1.1 แผนการดำเนินงาน

ลำดับ	หัวข้องาน	สิงหาคม				กันยายน				ตุลาคม				พฤศจิกายน			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1.	ศึกษาข้อมูลเกี่ยวกับหุ่นยนต์อุตสาหกรรม																
2.	ศึกษารายละเอียดและการทำงานของโปรแกรม RobotStudio																
3.	ศึกษารูปแบบโครงสร้างของ RAPID Programming Language																
4.	สร้างฟังก์ชันบล็อกในพีแอลซี เพื่อประมวลชุดคำสั่ง RAPID																
5.	สร้างหน้าจอเอชเอ็มไอ เพื่อตรวจสอบและแสดงผลข้อมูล																
6.	ทดสอบการทำงานของโปรแกรมพร้อมแสดงค่าของข้อมูล																

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ฟังก์ชันบล็อกที่สร้างขึ้นมาเพื่อจัดการข้อมูล RAPID จะช่วยเพิ่มความสะดวกในการออกแบบรูปแบบการเคลื่อนที่โดยใช้โปรแกรม RobotStudio ได้

บทที่ 2
แนวคิดและหลักการที่เกี่ยวข้อง

2.1 กล่าวนำ

ในบทนี้กล่าวถึงแนวคิดและหลักการที่เกี่ยวข้องกับโครงงานนี้ได้แก่ RobotStudio, RAPID Instructions, พีแอลซี, Gx Work2, เอชเอ็มไอและหุ่นยนต์ในอุตสาหกรรม โดยทฤษฎีและหลักการที่ได้กล่าวถึงนี้เป็นการศึกษาข้อมูล เพื่อนำมาประกอบการทำโครงงานครั้งนี้

2.2 RobotStudio

RobotStudio เป็นโปรแกรมเกี่ยวกับระบบหุ่นยนต์ และเป็นโปรแกรมที่เป็น offline Programming ของเอบีบีกรุ๊ป (ABB Limited) โดยที่โปรแกรมสามารถเขียนโปรแกรมการทำงานให้เสร็จก่อนที่จะลงสู่หุ่นยนต์ของจริง ช่วยทำให้หุ่นยนต์เพิ่มประสิทธิภาพการทำงานมากยิ่งขึ้น โดยภายในโปรแกรมจะมีเครื่องมือที่ช่วยดำเนินงานต่างๆ อาทิเช่น การอบรม การเขียนโปรแกรม และการเพิ่มประสิทธิภาพโดยไม่ต้องรบกวนกระบวนการทำงาน ซึ่งโปรแกรมนี้ช่วยลดความเสี่ยงในการทำงาน เริ่มการทำงานได้รวดเร็วขึ้นและเวลาในการเปลี่ยนแปลงน้อยลง

รูปที่ 2.1 โปรแกรม RobotStudio

2.2.1 ระดับการใช้งานของ RobotStudio

คุณสมบัติของ RobotStudio แบ่งออกเป็นระดับพื้นฐานและระดับพรีเมียม สิทธิ์ของผู้ดูแลระบบในพีซีเป็นสิ่งจำเป็นสำหรับการติดตั้ง RobotStudio

1. **พื้นฐาน** อนุญาตให้การใช้ฟังก์ชันพื้นฐานได้สามารถใช้ Virtual Controller ได้และสามารถตรวจสอบข้อมูลออนไลน์ของ Real Controller
2. **พรีเมียม** อนุญาตให้เข้าถึงฟังก์ชันการทำงานขั้นสูงได้ สามารถใช้ฟังก์ชันการทำงานแบบออฟไลน์ได้เต็มรูปแบบ มีฟังก์ชันการทำงานของหุ่นยนต์ให้เลือกมากขึ้นเพื่อให้เหมาะกับการใช้งาน แต่สำหรับการใช้งานในระดับนี้จำเป็นที่จะต้องซื้อใบอนุญาตการใช้งานจากเอบีบี

ตารางที่ 2.1 รายละเอียดของการใช้งานแบบพื้นฐานและพรีเมียม

Feature	Basic	Premium
Necessary features for commissioning a real or virtual robot, such as: • System Builder • Event Log Viewer • Configuration Editor • RAPID Editor • Backup / Restore • IO Window	Yes	Yes
Productivity fratures, such as: • RAPID Data Editor • RAPID Compare • Adjust robtargets • RAPID Watch • RAPID Breakpoints • Signal Analyzer • MultiMove tool • Screenmaker Jobs		Yes

Elementary offline features, such as: • Open station • Unpack & Work • Run Simulation • Go Offline • Robot jogging tools • Gearbox heat prediction • ABB Library of robots	Yes	Yes
Advanced offline features, such as: • Graphical programming • Save station • Pack & Go • Import / Export Geometry • Import Library • Create station viewer and movies • Transfer • AutoPath • 3D operations		Yes
Add-Ins		Yes

2.2.2 ข้อกำหนดของ RobotStudio ในการติดตั้ง

ก่อนการติดตั้ง RobotStudio ตรวจสอบให้แน่ใจว่าคอมพิวเตอร์มีคุณสมบัติตรงตามข้อกำหนดของฮาร์ดแวร์และซอฟต์แวร์ต่อไปนี้

ตารางที่ 2.2 ข้อกำหนดของ RobotStudio ในการติดตั้งในส่วนของฮาร์ดแวร์

Part	Requirement
CPU	2.0 GHz or faster processor, multiple cores recommended
Memory	8GB minimum. 16GB or more if working with heavy CAD models
Disk	10+ GB free space, solid state drive (SSD) recommended.
Graphics card	High-performance, DirectX 11 compatible, gaming graphics card from any of the leading vendors. For the Advanced lightning mode Direct3D feature level 10_1 or higher is required
Display settings	1920x1080 pixels or higher resolution is recommended
Dots per inch (dpi)	Only Normal size supported for Integrated Vision.
Mouse	Three-button mouse
3D Mouse [optional]	Any 3D mouse from 3DConnexion. See http://www.3dconnexion.com.

ตารางที่ 2.3 ข้อกำหนดของ RobotStudio ในการติดตั้งในส่วนของซอฟต์แวร์

Operating system	Description
Microsoft Windows 7SP1	64-bit edition
Windows 10 Anniversary Update or later	64-bit edition

2.3 RAPID

ภาษาดั้งเดิมของคอมพิวเตอร์ประกอบด้วยเลขศูนย์และหนึ่ง นี่เป็นไปไม่ได้ที่มนุษย์จะเข้าใจ ดังนั้นคอมพิวเตอร์จึงได้รับการสอนให้เข้าใจภาษาที่ค่อนข้างเข้าใจง่ายเป็นภาษาโปรแกรมระดับสูง RAPID เป็นภาษาการเขียนโปรแกรมระดับสูงใช้คำภาษาอังกฤษบางคำเช่น IF และ FOR เพื่อให้เข้าใจได้ง่ายสำหรับมนุษย์

ตัวอย่างโปรแกรม RAPID อย่างง่าย

```
MODULE MainModule

VAR num length;

VAR num width;

VAR num area;

PROC main()

        length := 10;

        width := 5;

        area := length * width;

        TPWrite "The area of the rectangle is " \Num:=area;

END PROC

ENDMODULE
```

จากโปรแกรมดังกล่าวโปรแกรมนี้จะคำนวณพื้นที่ของสี่เหลี่ยมผืนผ้า ซึ่งพื้นที่ของสี่เหลี่ยมผืนผ้าคือ 50

2.3.1 ข้อมูลตัวแปรของ RAPID

1. ตัวแปร

ประเภทตัวแปรมีหลากหลายใน RAPID แต่ในหัวข้อนี้จะเน้นไปที่ตัวแปร 3 ประเภทดังต่อไปนี้

ตารางที่ 2.4 ประเภทของตัวแปร

Data type	Description
num	Numerical data, can be both integer and decimal number. E.g. 10 or 3.14159.
string	A text string. E.g. “This is a string”. Muximum of 80 characters.
bool	A Boolean (logical) variable. Can only have the values True or False

ชนิดข้อมูลอื่น ๆ ทั้งหมดจะยึดตาม 3 ตัวแปรสิ่งนี้ หากเข้าใจวิธีการดำเนินการและวิธีที่ 3 ตัวแปรนี้สามารถรวมเข้ากับชนิดข้อมูลที่ซับซ้อนมากขึ้นก็จะสามารถเข้าใจประเภทข้อมูลทั้งหมดได้อย่างง่ายดาย

2. ลักษณะของตัวแปร

ตัวแปรที่มีค่าข้อมูล หากโปรแกรมหยุดทำงานและเริ่มทำงานตัวแปรจะเก็บค่าไว้แต่ถ้าตัวชี้โปรแกรมถูกย้ายไปยังค่าหลักข้อมูลของตัวแปรจะหายไป

3 ประกาศตัวแปร

การประกาศตัวแปรเป็นวิธีการกำหนดชื่อตัวแปรและระบุข้อมูลว่าเป็นชนิดใดตัวแปรจะถูกประกาศโดยใช้คีย์เวิร์ด VAR รูปแบบตามหลักภาษาเป็นดังนี้

```
VAR datatype identifier;
```

ตัวอย่างเช่น

```
VAR num length;
VAR string name;
VAR bool finished;
```

4 การกำหนดค่า

การกำหนดค่าให้กับตัวแปรจะใช้คำสั่ง :=

```
length := 10;
name := "John"
finished := TRUE;
```

โปรดทราบว่า := ไม่ใช่เครื่องหมายเท่ากับ แต่หมายความว่าข้อมูลทางด้านขวาถูกส่งผ่านไปยังตัวแปรด้านซ้าย โดยตัวแปรจำเป็นต้องอยู่ด้านซ้ายของเครื่องหมาย := เท่านั้น

ตัวอย่างเช่น

```
reg1 := 2;
reg1 := reg1 + 1;
```

จากโปรแกรมด้านบนเป็นการเขียนโปรแกรมตามหลักภาษาที่ถูกต้องส่งผลให้ reg1 มีค่าเท่ากับ 3

```
VAR num length := 10;

VAR string name := "John";

VAR bool finished := TRUE;
```

จากโปรแกรมด้านบนจะเห็นว่าการกำหนดค่าสามารถทำได้ในเวลาเดียวกันกับการประกาศตัวแปร

5. ตัวแปรถาวร

ตัวแปรถาวรนั้นโดยทั่วไปเหมือนกับตัวแปรธรรมดา แต่มีความสำคัญอย่างหนึ่ง คือ ตัวแปรถาวรจะจดจำค่าสุดท้ายที่ได้รับการกำหนดแม้ว่าโปรแกรมจะหยุดและเริ่มการทำงานของโปรแกรมใหม่

การประกาศตัวแปรถาวรสามารถทำได้โดยใช้คีย์เวิร์ด PERS ในการประกาศค่าเริ่มต้นต้องมีการประกาศตามรูปแบบ

```
PERS num nbr := 1;

PERS string string1 := "Hello";
```

ตัวอย่างเช่น

```
PERS num nbr := 1;

PROC main()

    nbr := 2;

ENDPROC
```

จากโปรแกรมข้างต้นได้มีการประกาศใช้ตัวแปรถาวร ซึ่งได้มีการบันทึกตัวแปร nbr ให้มีค่าเท่ากับ 1 ในตอนเริ่มโปรแกรม หลังจากนั้นได้เข้าเมื่อโปรแกรมทำงานในโปรแกรมหลักได้มีการบันทึกค่าใหม่ในตัวแปร nbr ให้มีค่าเท่ากับ 2

```
PERS num nbr := 2;

PROC main()

    nbr := 2;

ENDPROC
```

จากโปรแกรมข้างต้นจะเห็นได้ว่าเมื่อให้โปรแกรมทำงานใหม่ค่าที่บันทึกไว้ในตัวแปร nbr ก็จะไม่เปลี่ยนแปลงค่าที่บันทึกไว้ในตัวแปล nbr ก็จะมีค่าเท่ากับ 2 เท่าเดิม

6. ค่าคงที่

ค่าคงที่มีค่าแบบเดียวกับตัวแปร แต่ค่าจะถูกกำหนดใหม่เสมอเมื่อมีการประกาศและหลังจากนั้นค่าจะไม่สามารถเปลี่ยนแปลงได้ ค่าคงที่สามารถใช้ในโปรแกรมเช่นเดียวกับตัวแปรยกเว้นว่าไม่ได้รับอนุญาตให้กำหนดค่าใหม่ให้กับมัน

ค่าคงที่จะประกาศโดยใช้คีย์เวิร์ด CONST ตามด้วยระบุชนิดตัวข้อมูลและกำหนดค่า

```
CONST num gravity := 9.81;

CONST string greating := "Hello"
```

โดยการใช้ค่าคงที่แทนตัวแปรจะสามารถมั่นใจได้ว่าค่าจะไม่เปลี่ยนแปลงในโปรแกรม การใช้ค่าคงที่แทนการเขียนค่าโดยตรงในโปรแกรมจะดีกว่าถ้าต้องการอัพเดตโปรแกรมด้วยค่าอื่น ลงในค่าคงที่ เพราะฉะนั้นจึงสามารถเปลี่ยนค่าในที่เดียวก็สามารถมั่นใจได้ว่าจะไม่ได้ลืมค่าที่เกิดขึ้น

2.3.2 เครื่องหมายและสัญลักษณ์ในการใช้งาน

1. ตัวดำเนินงานเชิงตัวเลข

ตัวดำเนินการดังต่อไปนี้จะทำงานก็ต่อเมื่อข้อมูลเป็นข้อมูลประเภท num และจะส่งค่าคืนเป็นข้อมูลประเภท num เช่นกัน

ตัวอย่างเช่น ตารางที่ 2.5 ตัวแปร reg1, reg2 และ reg3 เป็นข้อมูลประเภท num

ตารางที่ 2.5 ตัวดำเนินงานเชิงตัวเลข

Operator	Description	Example
+	Addition	reg1 := reg2 + reg3 ;
-	Subtraction Unary minus	reg1 := reg2 - reg3 ; reg1 := -reg2 ;
*	Multiplication	reg1 := reg2 * reg3 ;
/	Division	reg1 := reg2 / reg3 ;

2. ตัวดำเนินแบบความสัมพันธ์

ตัวดำเนินการดังต่อไปนี้ส่งข้อมูลแบบ Bool ตัวอย่าง reg1 และ reg2 เป็นข้อมูลประเภท num ในขณะที่ flag1 เป็นข้อมูลประเภท bool ซึ่งตัวดำเนินการเหล่านี้มักใช้ร่วมกับคำสั่ง IF

ตารางที่ 2.6 ตัวดำเนินงานแบบความสัมพันธ์

Operator	Description	Example
=	equal to	flag1 := reg1 = reg2 ; flag1 is True if reg1 Equals reg2
<	less then	flag1 := reg1 < reg2 ; flag1 is True if reg1 is less then reg2
>	greater then	flag1 := reg1 > reg2 ; flag1 is True if reg1 is greater then reg2
<=	less then or equal to	flag1 := reg1 <= reg2 ; flag1 is True if reg1 is less then or equal to reg2
>=	greater then or equal to	flag1 := reg1 >= reg2 ; flag1 is True if reg1 is greater then or equal to reg2
<>	not equal to	flag1 := reg1 <> reg2 ; flag1 is True if reg1 is not equal to reg2

ตารางที่ 2.7 ตัวดำเนินงานแบบความสัมพันธ์ในการบวก String

Operator	Description	Example
+	String concatenation	VAR string firstname := “John”; VAR string lastname := “Smith”; VAR string fullname; Fullname := firstname +””+ lastname; The variable fullname will contain the string “John Smith”.

2.3.3 คำสั่งที่ในการกำหนดเงื่อนไข

ตัวอย่างโปรแกรมที่กล่าวมานั้นถูกดำเนินการตามลำดับจากบนลงล่าง สำหรับโปรแกรมที่ซับซ้อนมากขึ้นอาจต้องการควบคุมว่าจะประมวลผลรหัสตามลำดับแบบใดและกี่ครั้ง ในส่วนต่อไปจะกล่าวถึงวิธีการตั้งค่าเงื่อนไขสำหรับโปรแกรมควรดำเนินการตามลำดับหรือทำในกระบวนการที่ต้องการ

1. IF

คำสั่ง IF สามารถใช้ได้เมื่อมีการดำเนินการชุดคำสั่งเท่านั้นหากมีการระบุไว้เป็นไปตามเงื่อนไขเมื่อเงื่อนไข logic ในคำสั่ง IF เป็นจริง ดังนั้นโค้ดโปรแกรมระหว่างคำหลัก THEN และ ENDIF จะถูกดำเนินการ หากเงื่อนไขเป็นเท็จชุดคำสั่งนั้นจะไม่ถูกเรียกใช้และจะไปการดำเนินการต่อไปหลังจาก ENDIF

ตัวอย่างเช่น string1 เป็นข้อมูลประเภท String ถูกเขียนบน FlexPendant โดยหลักการทำงานของโปรแกรมนี้คือหาก string1 ไม่ใช่ตัวแปรที่ไม่มีข้อมูลโปรแกรมจะเขียนข้อมูลที่อยู่ใน string1 ออกมาแสดงที่หน้าจอแต่ถ้าข้อมูลใน string1 เป็นข้อมูลว่างโปรแกรมก็จะไม่ทำงานในส่วนของ IF และข้ามไปทำงานหลัง ENDIF ทันที

```
VAR string string1 := "Hello";

IF string1 <> "" THEN

        TPWrite string1;

ENDIF
```

2. ELSE

คำสั่ง IF ยังสามารถมีชุดคำสั่งโปรแกรมที่จะดำเนินการหากเงื่อนไขเป็นเท็จ คือหากเงื่อนไข logic ในคำสั่ง IF เป็นจริงดังนั้นชุดคำสั่งโปรแกรมระหว่างคีย์เวิร์ด THEN และ ELSE จะถูกดำเนินการ หากเงื่อนไขเป็นเท็จดังนั้นชุดคำสั่งโปรแกรมระหว่างคีย์เวิร์ด ELSE และ ENDIF ถูกเรียกใช้งาน

ตัวอย่างเช่น string1 เป็นข้อมูลประเภท String ถูกเขียนบน FlexPendant โดยหลักการทำงานของโปรแกรมนี้คือหาก string1 ไม่ใช่ตัวแปรที่ไม่มีข้อมูลโปรแกรมจะเขียนข้อมูลที่อยู่ใน string1 ออกมาแสดงที่หน้าจอแต่ถ้าข้อมูลใน string1 เป็นข้อมูลว่างโปรแกรมจะสั่งเห็นแสดงผลคำว่า “The string is empty”

```
VAR string string1 := "Hello";
IF string1 <> "" THEN
        TPWrite string1;
ELSE
        TPWrite "The string is empty";
  ENDIF
```

3. ELSEIF

บางครั้งโปรแกรมจะมีลำดับของโปรแกรมให้เลือกมากกว่าสองรายการ เพราะฉะนั้นจะสามารถใช้ ELSEIF เพื่อตั้งค่าหลายทางเลือกได้

ตัวอย่างเช่น โปรแกรมนี้จะแสดงข้อความที่แตกต่างกัน โดยขึ้นอยู่กับค่าในตัวแปร time

```
VAR num time := 38.7;
IF time < 40 THEN
        TPWrite "Part produced at fast rate";
ELSEIF time < 60 THEN
        TPWrite "Part produced at average rate";
ELSE    TPWrite "Part produced at slow rate";                ENDIF
```

4. FOR loop

อีกวิธีหนึ่งในการควบคุมการทำงานของโปรมแกรมให้โปรแกรมมีการทำงานซ้ำในชุดคำสั่งที่ต้องการโดยสามารถกำหนดจำนวนการทำงานซ้ำได้

ตัวอย่างเช่น โปรแกรมดังต่อไปนี้จะมีการทำงานซ้ำ 5 ครั้งนั้นคือจะมีการแสดงผลคำว่า “Hello” 5 ครั้งนั้นเอง

```
FOR i FROM 1 TO 5 DO
        TPWrite "Hello";
ENDFOR
```

โครงสร้างหลักในการนำ FOR loop ไปใช้

```
FOR counter FROM startvalue TO endvalue DO
        program code to be repeated
ENDFOR
```

ตัวแปร counter ไม่จำเป็นต้องประกาศ แต่ทำหน้าที่เป็นตัวแปรตัวเลขภายในลูปวนของ FOR ครั้งแรกที่โค้ดถูกเรียกใช้ตัวนับจะมีค่าที่ระบุโดยค่าเริ่มต้นค่าของตัวนับจะเพิ่มขึ้น 1 สำหรับแต่ละครั้งที่เรียกใช้โค้ดครั้งสุดท้าย โค้ดดำเนินการคือเมื่อการนับมีค่าเท่ากับค่าที่กำหนดไว้ หลังจากนั้นการดำเนินการจะดำเนินต่อไป ด้วยโค้ดการเขียนโปรแกรมหลังจาก ENDFOR

ตัวอย่างเช่น การคำนวณผลรวมของตัวเลขทั้งหมดตั้งแต่ 1 ถึง 50 (1 + 2 + 3 + ... + 49 + 50) สามารถตั้งโปรแกรมดังนี้

```
VAR num sum := 0;
FOR i FROM 1 TO 50 DO
        sum := sum + i;
ENDFOR
```

5. WHILE loop

WHILE loop เป็นอีกหนึ่งคำสั่งที่ใช้ในการวนลูปตามเงื่อนไขของชุดคำสั่งโดยใน WHILE loop จะทำการวนลูปไปเรื่อยตราบใดที่เงื่อนไขยังคงเป็นจริงอยู่จะหลุดการวนลูปได้ก็ต่อเมื่อเงื่อนไขเป็นเท็จ โครงสร้างหลักภาษาสำหรับ WHILE คือ

```
WHILE condition DO
        program code to be repeated
ENDWHILE
```

หากเงื่อนไขเป็นเท็จตั้งแต่เริ่มต้นชุดคำสั่งต่างๆในโปรแกรมจะไม่ถูกดำเนินการ แต่หากว่าเงื่อนไขเป็นจริงชุดคำสั่งในโปรแกรม จะถูกดำเนินการซ้ำ ๆ จนกว่าเงื่อนไขจะไม่จริงอีกต่อไป

ตัวอย่างเช่น ตามโปรแกรมด้านล่างนี้จะเป็นการเพิ่มตัวเลข (1+2+3+...) จนผลบวกมีค่าเท่ากับ 100

```
VAR num sum := 0;
VAR num i := 0;
WHILE sum <= 100 DO
        i := i + 1;
        sum := sum + i;
ENDWHILE
```

6. กฎการทำงาน RAPID ทั่วไป

กฎทั่วไปในการเขียนชุดคำสั่งในการเขียนทุกคำสั่งจะต้องลงท้ายด้วยเครื่องหมาย semicolon

ตัวอย่าง การประกาศตัวแปร

```
VAR num length;
```

ตัวอย่าง การกำหนดค่าตัวแปร

```
area := length * width;
```

ตัวอย่าง คำสั่งส่วนใหญ่

```
MoveL p10,v1000,fine,tool0;
```

แต่ก็ยังมีคีย์เวิร์ดบางคำที่ไม่จำเป็นต้องมีเครื่องหมาย semicolon ตามหลังรูปภาพด้านล่างจะเป็นตัวอย่างคำบางคำที่จำเป็นต้องรู้

ตารางที่ 2.8 คีย์เวิร์ดของตัวกำหนดเงื่อนไข

Instruction keyword	Terminating keyword
IF	ENDIF
FOR	ENDFOR
WHILE	ENDWHILE
PROC	ENDPROC

2.3.4 ประเภทของ Move

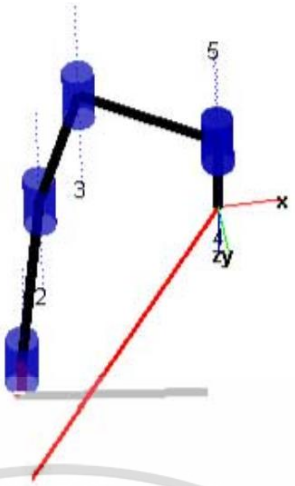
ข้อดีของ RAPID คือเป็นภาษาที่ใช้ในการเขียนโปรแกรมระดับสูง มีฟังก์ชันพิเศษต่าง ๆ ที่รองรับการเขียนโปรแกรมเพื่อควบคุมหุ่นยนต์โดยเฉพาะ และยังมีฟังก์ชันมากมาย ในการเขียนรูปแบบการเคลื่อนที่ของหุ่นยนต์

1. MoveL

โครงสร้างหลักภาษาของ MoveL

```
MoveL p10, v1000, fine, tool0;
```

MoveL เป็นรูปแบบการเคลื่อนที่ของหุ่นยนต์ ในลักษณะเส้นตรงจากพิกัดปัจจุบันไปที่พิกัดที่ได้ทำการกำหนดไว้



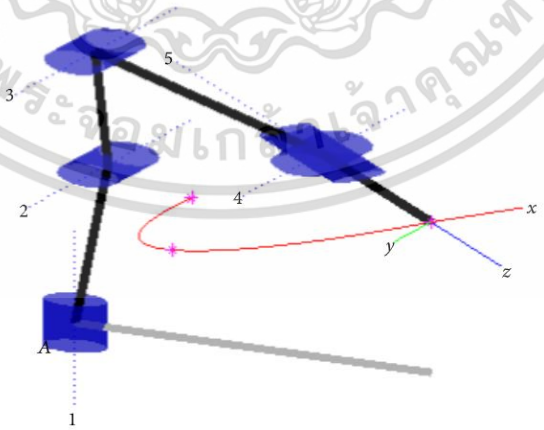
รูปที่ 2.2 รูปแบบจำลองการเคลื่อนที่แบบ MoveL ของหุ่นยนต์

2. MoveJ

โครงสร้างหลักภาษาของ MoveJ

```
MoveJ p10, v1000, fine, tPen;
```

MoveJ เป็นรูปแบบการเคลื่อนที่แบบรวดเร็วที่สุดโดยจะเคลื่อที่จากพิกัดเริ่มต้นไปยังพิกันที่ได้กำหนดไว้โดยรูปแบบการเคลื่อนที่แบบ MoveJ จะไม่เคลื่อนที่เป็นเส้นตรง



รูปที่ 2.3 รูปแบบจำลองการเคลื่อนที่แบบ MoveJ ของหุ่นยนต์

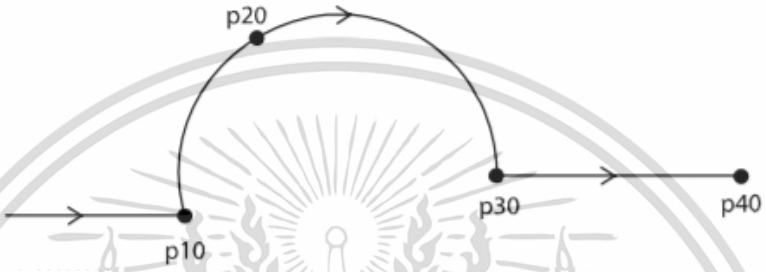
3. MoveC

โครสร้างหลักภาษาของ MoveC

MoveL p10, v500, fine, tPen;

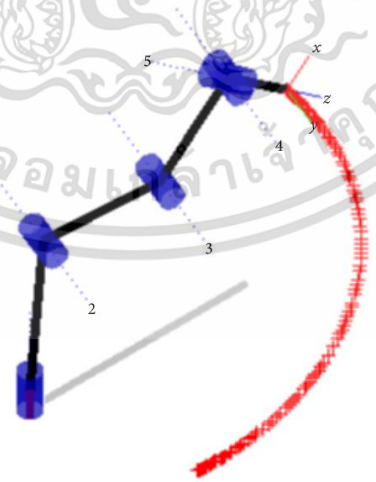
MoveC p20, p30, v500, fine, tPen;

MoveL p40, v500, fine, tPen;



รูปที่ 2.4 การกำหนดพิกัดสำหรับการเคลื่อนที่ด้วย MoveC

MoveC เป็นรูปแบบการเคลื่อนที่เป็นวงโดยที่จะต้องกำหนดพิกัด 3 พิกัด โดยพิกัดที่ 1 คือพิกัดเริ่มต้น พิกัดที่ 2 คือพิกัดส่วนของเส้นโค้งวงกลม พิกัดที่ 3 คือพิกัดสุดท้ายของการเคลื่อนที่โดยที่ทั้ง 3 พิกัดนี้คือพิกัดที่ตั้งอยู่บนส่วนโค้งของวงกลม



รูปที่ 2.5 รูปแบบจำลองการเคลื่อนที่แบบ MoveC ของหุ่นยนต์

2.3.5 โครงสร้างภายใน Move

1. ToPoint

การกำหนดพิกัดปลายทาง โดยการปะกาศตัวแปรประเภท robtarget ซึ่งสามารถกำหนดพิกัดหรือเปลี่ยนแปลงพิกัดได้ในภายหลัง ซึ่งในตัวอย่างต่อจากนี้จะเป็นการกำหนดพิกัด x = 600, y = -100, z = 800 ซึ่งจะเขียนได้ในลักษณะดังตัวอย่างด้านล่าง

```
CONST robtarget p10 := [ [600, -100, 800], [1, 0, 0, 0], [0, 0, 0,
0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
```

2. Speed

ความเร็วในการเคลื่อนที่นั้น สามารถกำหนดโดยการประกาศตัวแปร speeddata ซึ่งมีค่าที่ได้กำหนดไว้ให้แล้วในโปรแกรม มีดังนี้

ตารางที่ 2.9 ค่าความเร็วในการเคลื่อนไหว

Predefined speeddata	Value
v5	5 mm/s
v100	100 mm/s
v1000	1000 mm/s
vmax	Maximum speed for the robot

ค่าความเร็วในตัวแปร speeddata ที่ได้ถูกกำหนดไว้นั้นสามารถเปิดหาข้อมูลได้เพิ่มเติมจากคู่มือ RAPID หากใช้ค่าความเร็วที่ได้ถูกกำหนดไว้แล้วก็ไม่ควรที่จะไปประกาศหรือกำหนดค่าใหม่ลงไปเพื่อป้องกันหากเกิดกรณีเข้าใจผิดในกรณีเปลี่ยนผู้ใช้งาน

3. Zone

ค่า Zone นั้น สามารถกำหนดโดยการประกาศตัวแปร zonedata ซึ่งมีค่าที่ได้กำหนดไว้ให้แล้วในโปรแกรม มีดังนี้

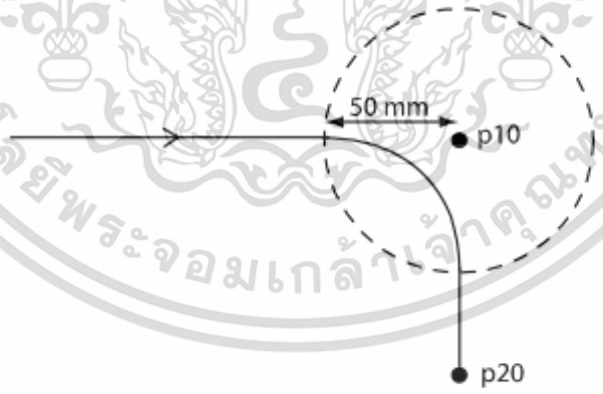
ตารางที่ 2.10 ค่า Zone รัศมีของการเคลื่อนที่

Predefined zonedata	Value
Fine	The robot will go to exactly the specified position
Z10	The robot path can cut comers when it is less then 10 mm from ToPoint
Z50	The robot path can cut comers when it is less then 50 mm from ToPoint

ค่าความเร็วในตัวแปร zonedata ที่ได้ถูกกำหนดไว้นั้นสามารถเปิดหาข้อมูลได้เพิ่มเติมจากคู่มือ RAPID หากใช้ค่าความเร็วที่ได้ถูกกำหนดไว้แล้วก็ไม่ควรที่จะไปประกาศหรือกำหนดค่าใหม่ลงไปเพื่อป้องกันหากเกิดกรณีเข้าใจผิดในกรณีเปลี่ยนผู้ใช้งาน

ตัวอย่างเช่น ชุดคำสั่ง RAPID ด้านล่างนี้จะส่งผลให้เส้นทางหุ่นยนต์ทำงาน

```
MoveL p10, v1000, z50, tool0;
MoveL p20, v1000, fine, tool0;
```



รูปที่ 2.6 ตัวอย่าง Zone รัศมีของการเคลื่อนที่

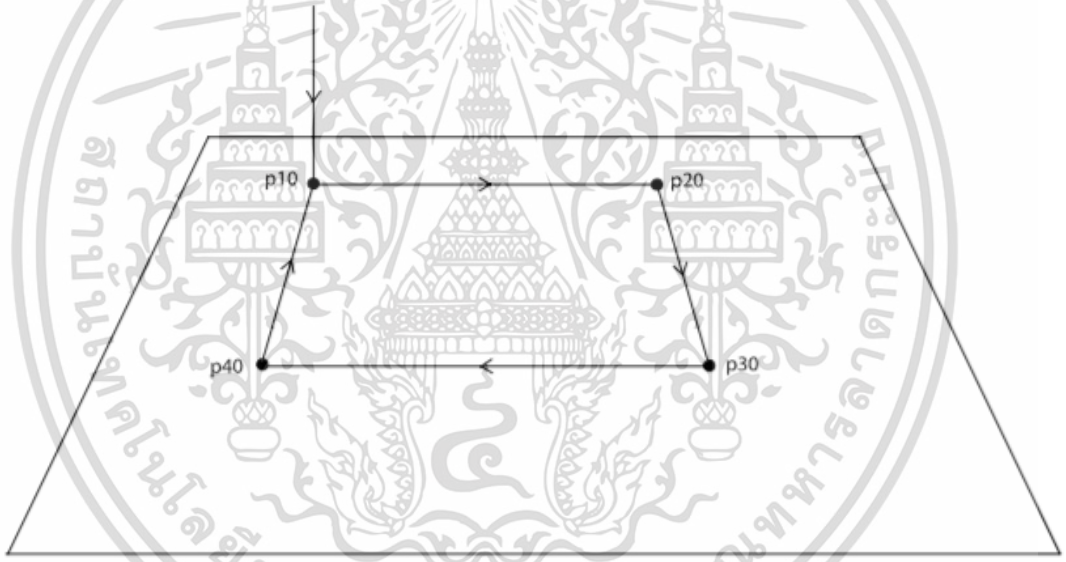
4. Tool

การระบุเครื่องมือที่หุ่นยนต์ใช้โดยจะกำหนดโดยการประกาศตัวแปรถาวรของประเภทข้อมูล tooldata เช่นหากมีปืนเชื่อม, ปืนกาว หรือปากกา ติดอยู่กับหุ่นยนต์ จะต้องทำการการเปลี่ยนโปรแกรม ToPoint เพื่อเปลี่ยนส่วนเครื่องมือของหุ่นยนต์ กระบวนการเหล่านี้จะทำโดยอัตโนมัติหากมีการประกาศ tooldata ไว้แล้ว tool0 เป็นเครื่องมือที่กำหนดไว้ล่วงหน้าซึ่งแสดงถึงหุ่นยนต์ที่ไม่มีเครื่องมือใด ๆ ติดตั้งอยู่

2.3.6 ตัวอย่างการเคลื่อนที่อย่างง่ายในคำสั่ง Move

ตัวอย่างการเคลื่อนที่เป็นสี่เหลี่ยมแบบไม่ใช่ Zone

หุ่นยนต์กำลังถือปากกาเหนือแผ่นกระดาษบนโต๊ะ ตัวอย่างด้านล่างต้องการให้หุ่นยนต์ขยับปลาย ของปากกาลงไปที่กระดาษแล้วจึงวาดสี่เหลี่ยม



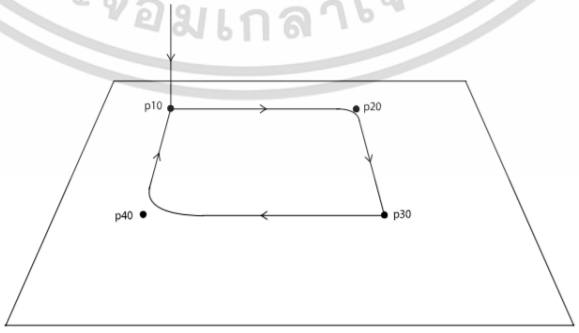
รูปที่ 2.7 ตัวอย่างการเคลื่อนที่เป็นสี่เหลี่ยมแบบไม่ใช่ Zone

```
PERS tooldata tPen := [ TRUE, [[200, 0, 30], [1, 0, 0 ,0]], [0.8,
        [62, 0, 17], [1, 0, 0, 0], 0, 0, 0]];
CONST robtarget p10 := [ [600, -100, 800], [0.707170, 0, 0.707170,
        0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
```

```
CONST robtarget p20 := [ [600, 100, 800], [0.707170, 0, 0.707170,
        0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
CONST robtarget p30 := [ [800, 100, 800], [0.707170, 0, 0.707170,
        0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
CONST robtarget p40 := [ [800, -100, 800], [0.707170, 0, 0.707170,
        0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
PROC main()
        MoveL p10, v200, fine, tPen;
        MoveL p20, v200, fine, tPen;
        MoveL p30, v200, fine, tPen;
        MoveL p40, v200, fine, tPen;
        MoveL p10, v200, fine, tPen;
ENDPROC
```

ตัวอย่างการเคลื่อนที่เป็นสี่เหลี่ยมแบบใช้ Zone

วาดรูปเดียวกันในตัวอย่างก่อนหน้า แต่มีโซนมุม 20 มม.ที่ p20 และ มุม 50 มม.ที่ p40



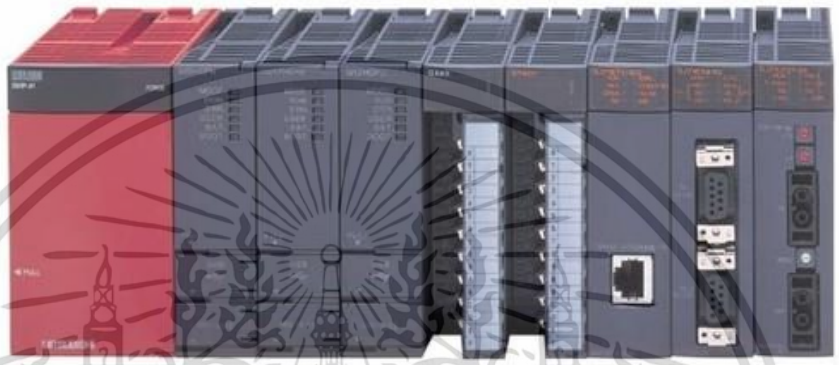
รูปที่ 2.8 ตัวอย่างการเคลื่อนที่เป็นสี่เหลี่ยมแบบใช้ Zone

```
PERS tooldata tPen := [ TRUE, [[200, 0, 30], [1, 0, 0 ,0]], [0.8,
        [62, 0, 17], [1, 0, 0, 0], 0, 0, 0]];
CONST robtarget p10 := [ [600, -100, 800], [0.707170, 0, 0.707170,
        0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
CONST robtarget p20 := [ [600, 100, 800], [0.707170, 0, 0.707170,
        0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
CONST robtarget p30 := [ [800, 100, 800], [0.707170, 0, 0.707170,
        0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
CONST robtarget p40 := [ [800, -100, 800], [0.707170, 0, 0.707170,
        0], [0, 0, 0, 0], [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
PROC main()
        MoveL p10, v200, fine, tPen;
        MoveL p20, v200, z20, tPen;
        MoveL p30, v200, fine, tPen;
        MoveL p40, v200, z50, tPen;
        MoveL p10, v200, fine, tPen;
ENDPROC
```

2.4 พีแอลซีที่ใช้ในกรณีศึกษา

พีแอลซีย่อมาจาก "Programmable Logic Controller: PLC" เป็นอุปกรณ์ควบคุมอิเล็กทรอนิกส์ ที่มีหน่วยความจำในการเก็บโปรแกรมสำหรับควบคุมการทำงานของอุปกรณ์ต่าง ๆ ที่ต่อกับขั้วเข้า และขั้วออกของมัน พีแอลซีนี้ยังมีชื่อเรียกอย่างอื่นเช่น พีซีซึ่งย่อมาจาก "Programmable Controller: PC" และ เอ็ซซีซึ่งย่อมาจาก "Sequence Controller: SC" พีแอลซีขนาดเล็กอาจเรียกว่า ซีเควนเซอร์ (Squencer) ก็มี พีแอลซีถือเป็นอุปกรณ์ควบคุมสำคัญมากตัวหนึ่ง

ในการควบคุมเครื่องจักรต่าง ๆ ใน โรงงานอุตสาหกรรมให้ทำงานแบบอัตโนมัติในระบบ เอฟเอ (Factory Automation: FA) พีแอลซีจะถูกใช้ในการเพิ่มประสิทธิภาพการทำงานของเครื่องจักรทำให้เครื่องจักรสามารถทำงานได้เองโดยอัตโนมัติ เป็นการลดภาระหน้าที่ของคนงาน พีแอลซีนั้นมีทั้งที่มีขนาดใหญ่หรืออาจเป็นระบบควบคุมสายพานลำเลียงในโรงงานจนกระทั่งถึง พีแอลซีขนาดเล็กซึ่งใช้ในการควบคุมเครื่องจักรแต่ละเครื่อง



รูปที่ 2.9 พีแอลซี

โครงสร้างของพีแอลซีโครงสร้างภายในของพีแอลซีแต่ละส่วนนั้นจะประกอบกันทำงานเป็นระบบควบคุม หรือที่เราเรียกว่าพีแอลซีซึ่งประกอบไปด้วยส่วนสำคัญคือยูนิตทั้ง 5 ส่วน เมื่อประกอบเข้าด้วยกันแล้ว ก็จะกลายเป็นพีแอลซีชุดหนึ่งที่สามารถทำงานได้ แต่ละยูนิตจะมีหน้าที่และคุณสมบัติดังนี้

ก) ซีพียู (Central Processing Unit: CPU)

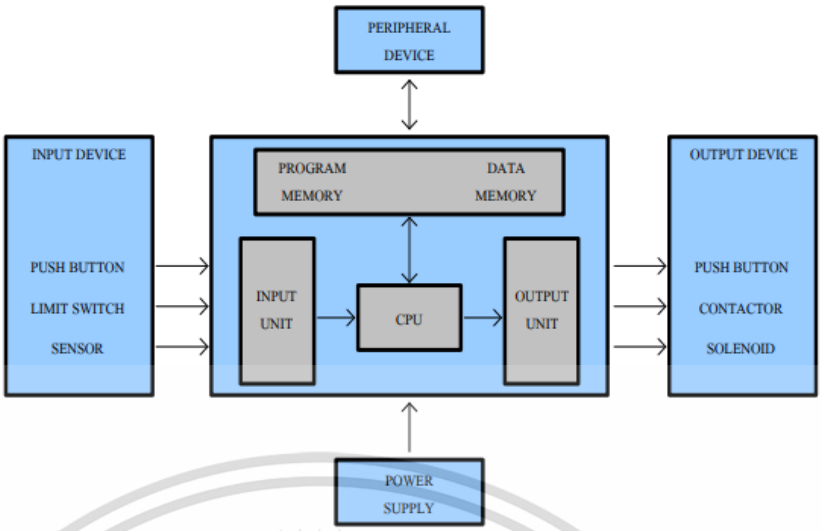
ข) หน่วยความจำ (Memory Unit)

ค) ภาคอินพุท (Input Unit)

ง) ภาคเอาท์พุท (Output Unit)

จ) ภาคแหล่งจ่ายพลังงาน (Power Supply Unit)

ลักษณะโครงสร้างภายในของพีแอลซีประกอบด้วย



รูปที่ 2.10 โครงสร้างภายในพีแอลซี

ซีพียู (Central Processing Unit: CPU) ซีพียูหรือหน่วยประมวลผลกลางทำหน้าที่ประมวลผล กลาง จะทำงานตามคำสั่งของส่วนต่าง ๆ ตามที่ได้รับมา ผลจากการประมวลผลก็จะถูกส่งออกไปยังส่วนต่าง ๆ ตามที่ได้ระบุไว้ด้วยคำสั่งนั่นเอง ซีพียูจะใช้เวลาในการประมวลผลช้าหรือเร็วขึ้นอยู่กับการเลือกขนาดของซีพียูและความยาวของโปรแกรมที่เขียนด้วย ปกติแล้วซีพียูจะใช้ไมโครโปรเซสเซอร์ขนาดตั้งแต่ 4 บิต 8 บิต 16 บิต 32 บิต 64 บิต 128 บิต มาทำงาน โดยทำให้ซีพียูในแต่ละขนาดก็จะมีความสามารถไม่เท่ากันจึงทำให้พีแอลซีในแต่ละรุ่น แต่ละยี่ห้อนั้น จะมีความสามารถแตกต่างกันนั่นเอง หรือแม้กระทั่งว่าภายในพีแอลซีบางรุ่นจะใช้ไมโครโปรเซสเซอร์มากถึง 2 ตัวมาช่วยกันทำงาน จึงทำให้เวลาประมวลผลก็จะเร็วกว่าพีแอลซีที่ใช้ไมโครโปรเซสเซอร์เพียงแค่ตัวเดียว โดยปกติแล้วในการเลือกใช้งานพีแอลซีนั้น จะเลือกจากการประยุกต์ใช้งานจึงทำให้ ผู้ใช้งาน (User) ไม่รู้ว่าผู้ผลิตใช้ไมโครโปรเซสเซอร์รุ่นหรือเบอร์อะไรในการสร้างเครื่องพีแอลซีเวลาพิจารณาเลือกใช้พีแอลซีซึ่งไม่มีการระบุเบอร์หรือรุ่นของไมโครโปรเซสเซอร์ ดังนั้น ผู้ที่ใช้งานสามารถเลือกจากคุณสมบัติอื่น ๆ เช่น จำนวนอินพุท/เอาท์พุท ความเร็วในการประมวลผลของคำสั่ง ขนาดความจุของโปรแกรม และข้อมูลเป็นต้น

หน่วยความจำ (Memory Unit) หน่วยความจำเป็นอุปกรณ์ที่ใช้เก็บโปรแกรมข้อมูลต่าง ๆ ของพีแอลซีกรณีที่ต้องการสั่งให้พีแอลซีทำงาน (RUN) โดยพีแอลซีจะนำเอาโปรแกรมและข้อมูลในหน่วยความจำมาประมวลผลการทำงาน สำหรับหน่วยความจำที่ใช้งานมีด้วยกัน 2 ชนิด คือ

หน่วยความจำชั่วคราว (Random Access Memory: RAM)

หน่วยความจำถาวร (Read Only Memory: ROM)

หน่วยความจำชั่วคราว (Random Access Memory: RAM) โปรแกรมและข้อมูลที่ถูกสร้างขึ้นโดยผู้ใช้ก็จะถูกจัดเก็บในส่วนนี้ คุณสมบัติของ RAM นั้นเมื่อไม่มีไฟเลี้ยงจะทำให้โปรแกรมและข้อมูลหายไปทันที ดังนั้นภายในพีแอลซีจะพบว่า มีแบตเตอรี่สำรองข้อมูล (Backup Battery) เอาไว้สำรองข้อมูล (Backup Data) กรณีที่ไฟหลัก (Main Power Supply) ไม่จ่ายไฟให้กับพีแอลซีข้อควรระวังคือ ไม่ควรที่จะถอดแบตเตอรี่สำรอง (Backup Battery) ในกรณีที่ไม่มีไฟจ่ายให้พีแอลซี

หน่วยความจำถาวร (Read Only Memory: ROM) เป็นหน่วยความจำอีกชนิดหนึ่งโดยที่ข้อมูลใน ROM นั้นไม่จำเป็นต้องมีแบตเตอรี่สำรองข้อมูล แต่ก็มีปัญหาเรื่องเวลาในการเข้าถึงข้อมูล (Time Access) ช้ากว่า RAM จึงปรากฏให้ผู้ใช้เห็นพีแอลซีจะมีหน่วยความจำใช้งานทั้ง RAM และ ROM ร่วมกันอยู่ ROM แบ่งออกเป็น 3 ชนิดดังนี้

-PROM (Programmable ROM)

- EPROM (Erasable Programmable ROM)

- EEPROM (Electrical Erasable Programmable ROM)

-PROM จัดเป็น ROM รุ่นแรก ๆ ที่สามารถเขียนข้อมูลลงชิฟได้เพียงครั้งเดียว ถ้าเขียนแล้วข้อมูลไม่สมบูรณ์ชิฟก็จะเสียทันทีโดยไม่สามารถนำกลับมาเขียนใหม่ได้อีก

- EPROM (Erasable Programmable Read Only Memory) หน่วยความจำชนิด EPROM นี้ จะต้องใช้เครื่องมือพิเศษในการเขียนโปรแกรม การลบโปรแกรมทำได้โดยใช้แสงอัลตราไวโอเลตหรือตากแดดร้อน ๆ นาน ๆ มีข้อดีตรงที่โปรแกรมจะไม่สูญหาย แม้ไฟดับจึงเหมาะกับการใช้งานที่ ไม่ต้องการเปลี่ยนโปรแกรม

- EEPROM (Electrical Erasable Programmable Read Only Memory) หน่วยความจำชนิด นี้ไม่ต้องใช้เครื่องมือพิเศษในการเขียน หรือลบโปรแกรม โดยจะใช้วิธีการทางไฟฟ้าเหมือนกับ RAM นอกจากนั้นก็ไม่จำเป็นต้องมีแบตเตอรี่สำรองไฟเมื่อไฟดับ ซึ่ง EEPROM จะรวมคุณสมบัติที่ดีของทั้ง RAM และ EPROM เอาไว้ด้วยกัน

การใช้งานหน่วยความจำในพีแอลซี

- RAM จะใช้เก็บโปรแกรมและข้อมูลที่ทำงานจากการสั่ง RUN/STOP พีแอลซี

- ROM จะใช้เก็บซอฟต์แวร์ระบบ (System Software) และใช้เป็นชุดสำรองโปรแกรมหรือข้อมูล (Backup Program and Data) เพื่อป้องกันในกรณีที่โปรแกรมหรือข้อมูลใน RAM หายไป ผู้ใช้สามารถที่จะถ่ายโปรแกรมและข้อมูลเข้าไปที่ RAM ใหม่ได้

ภาคอินพุท (Input Unit) ภาคอินพุตของพีแอลซีจะทำหน้าที่รับสัญญาณอินพุทเข้ามาแล้วแปลงสัญญาณเพื่อที่จะส่งเข้าไปภายในพีแอลซีอุปกรณ์อินพุท (Input Device) ต่าง ๆ ที่จะนำมาต่อกับภาคอินพุทได้เช่น Relay, Limit Switch, Inverter, Encoder, Temperature Controller, Photoelectric Sensor เพื่อส่งไป ยังซีพียูเพื่อประมวลผลตามโปรแกรมคำสั่งของผู้ใช้ โดยปกติแล้วหน้าที่ของหน่วยอินพุทคือ

- แปลงระดับสัญญาณเข้าไปให้เป็นระดับสัญญาณที่เหมาะสมให้กับระบบการทำงานของซีพียู

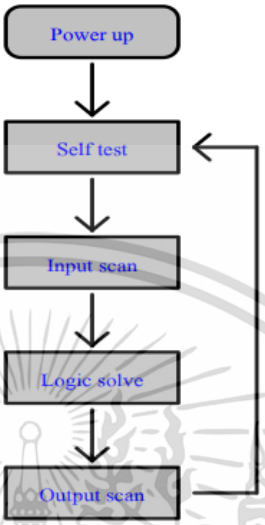
- แบ่งสัญญาณภายนอกและภายในออกจากกัน (Isolate) เพื่อที่จะต้องการป้องกัน ไม่ให้หน่วยประมวลผลได้รับความเสียหาย

– แก้ปัญหาการสั่นสะเทือนของหน้าสัมผัส

ภาคเอาท์พุท (Output Unit) ภาคเอาท์พุทของพีแอลซีทำหน้าที่สงสัญญาณออกไปขับโหลดชนิดต่าง ๆ ตามเงื่อนไขที่ได้เขียนโปรแกรมเอาไว้ ซึ่งหน่วยเอาท์พุททำหน้าที่รับข้อมูลจากตัวประมวลผลแล้วส่งต่อข้อมูลไป ควบคุมอุปกรณ์ภายนอกเช่น ควบคุมหลอดไฟ มอเตอร์ และวาล์ว เป็นต้น

การทำงานของพีแอลซีส่วนใหญ่จะมีลำดับการทำงานพื้นฐานอยู่4 ขั้นตอนและจะทำงานซ้ำ ๆ กันหลายครั้งภายในเวลาหนึ่งวินาที และเมื่อเริ่มต้นจ่ายไฟให้กับพีแอลซีมันจะเริ่มตรวจสอบการทำงานของฮาร์ดแวร์และซอฟต์แวร์ เพื่อที่จะหาข้อบกพร่อง แต่ถ้าไม่มีปัญหาใด ๆ มันจะนำเอาข้อมูลในอินพุต (สัญญาณอินพุทต่าง ๆ) เข้ามาเก็บไว้ในหน่วยความจำซึ่งเราจะเรียกว่า สแกนอินพุท (Input Scan) หลังจากนั้น พีแอลซีจะประมวลผล ตามโปรแกรมแลดเดอร์ (Ladder Program) โดยจะใช้ข้อมูล จากหน่วยความจำการประมวลผล ดังกล่าวนี้เรียกว่า สแกนลอจิก (Logic Scan) ในขณะที่ พีแอลซีประมวลผลตามโปรแกรมแลดเดอร์นั้นค่าเอาท์พุทของโปรแกรมแลดเดอร์ ก็จะเปลี่ยนแปลงไปตาม เงื่อนไขต่าง ๆ ของโปรแกรม แต่การเปลี่ยนแปลงนี้จะ อยู่ในหน่วยความจำชั่วคราว (Temporary Memory) เท่านั้น เมื่อการสแกนแลดเดอร์ทำงานเสร็จแล้วข้อมูลด้านเอาท์พุทในหน่วยความจำชั่วคราวนี้ จะถูกส่งไปที่ยูนิตเอาท์พุททำให้อุปกรณ์ที่ต่ออยู่ภายนอกทำงาน หรือไม่ทำงานตาม ผลลัพธ์ที่ได้จากการประมวลผลซึ่งเรียกว่า สแกนเอาท์พุท (Output Scan) เมื่อสิ้นสุดการ

สแกนเอาท์พุท จะกลับไปเริ่มต้นการทำงานใหม่ ซึ่งกระบวนการดังกล่าวนี้ จะใช้เวลา 5 –10 ms หรือเร็วกว่าขึ้นอยู่กับความเร็วในการทำงานของซีพียู



รูปที่ 2.11 วงรอบการสแกนของพีแอลซี

การสแกนอินพุทและเอาท์พุท เมื่ออินพุทต่าง ๆ ที่ต่อเข้ากับพีแอลซีนั้นจะถูกสแกน มันจะเก็บค่าหรือสถานะต่าง ๆ ไว้ในหน่วยความจำและเมื่อเอาท์พุทที่ต่อกับพีแอลซีถูกสแกนมันก็จะทำการคัดลอก (Copy) ข้อมูลจาก หน่วยความจำส่งออกไปให้เอาท์พุท และเมื่อทำการสแกนแลดเดอร์ หรือประมวลผลแลดเดอร์พีแอลซีมันจะใช้ค่าหรือข้อมูลในหน่วยความจำเท่านั้น โดยไม่สนใจค่าหรือข้อมูลจริงของอินพุทและเอาท์พุทในขณะนั้นในทำนองเดียวกันถ้าเอาท์พุทต้องเปลี่ยนแปลงอยู่ตลอดเวลาเมื่อประมวลผลในแต่ละคำสั่งของโปรแกรมแลดเดอร์แทนที่จะประมวลผลให้จบทั้งโปรแกรม มันจะทำให้ พีแอลซีทำงานได้ช้ามาก เพราะต้องคัดลอกข้อมูลไปที่ยูนิตเอาท์พุททุกครั้งที่ค่ามันเปลี่ยนแปลงจากการประมวลผล สัญญาณอินพุทที่ดีจะต้องมีคุณสมบัติและหน้าที่ดังนี้

- ทำให้สัญญาณที่เข้าได้ระดับที่เหมาะสมกับพีแอลซี

- การส่งสัญญาณระหว่างอินพุทกับซีพียูจะติดต่อกันด้วยลำแสงโดยอาศัยอุปกรณ์ประเภทโฟโตทรานซิสเตอร์เพื่อต้องการจะแยกสัญญาณ (Isolate) ทางไฟฟ้าให้ออกจากกันเป็นการป้องกันไม่ให้ซีพียูเสียหายเมื่ออินพุทเกิดลัดวงจร

- หน้าสัมผัสจะต้องไม่สั่นสะเทือน (Contact Chattering) ในส่วนของเอาท์พุทจะทำหน้าที่รับค่าสภาวะที่ได้จากการประมวลผลของซีพียูแล้วนำค่าเหล่านี้ไปควบคุมอุปกรณ์ต่าง ๆ เช่น รีเลย์ หรือหลอดไฟเป็นต้น นอกจากนั้นแล้วยังทำหน้าที่แยกสัญญาณของหน่วยประมวลผลกลาง (CPU) ออกจากอุปกรณ์ด้านเอาท์พุท ซึ่งปกติแล้วเอาท์พุทนี้จะสามารถขับโหลดได้ด้วยกระแสไฟฟ้าประมาณ 1–2 แอมแปร์แต่ถ้าโหลดนั้นต้องการกระแสไฟฟ้ามากกว่านี้ จะต้องต่อเข้ากับอุปกรณ์ขับอื่นเพื่อขยายให้รับกระแสไฟฟ้ามากขึ้น เช่น รีเลย์ แม็กเนติกคอนแทกเตอร์ เป็นต้น

ภาษาที่ใช้ในพีแอลซี

- Ladder Diagram Language
- Sequential Flow Chart Language
- Function Block Diagram Language
- Instruction List Language (Statement List Language)
- Structure Text Language

- **ชนิดพีแอลซี ที่ใช้ในการเขียนการสร้างฟังก์ชันบล็อก**

ในการสร้างฟังก์ชันบล็อกของพีแอลซีเพื่อประมวลผลชุดคำสั่ง RAPID สำหรับการควบคุมหุ่นยนต์อุตสาหกรรม สามารถใช้พีแอลซีได้หลากหลายชนิดตามความเหมาะสมของงาน แต่ชนิดที่เลือกมาใช้ในโครงงานคือรุ่น Mitsubishi Q03UDECPU ซึ่งเป็นรุ่นซีรี่ย์ Q เป็นซีรี่ย์ของเครื่องขนาดใหญ่

ตารางที่ 2.11 ความสามารถและ I/O point ของพีแอลซี

CPU type	Program capacity	I/O points
Q00UJCPU	10 k steps	256/8192
Q00UCPU	10 k steps	1024/8192
Q01UCPU	15 k steps	1024/8192
Q02UCPU	20 k steps	2048/8192
Q03UDCPU	30 k steps	4096/8192
Q03UDECPU	30 k steps	4096/8192
Q04UDHCPU	40 k steps	4096/8192
Q04UDEHCPU	40 k steps	4096/8192

Q06UDHCPU	60 k steps	4096/8192
Q06UDEHCPU	60 k steps	4096/8192
Q10UDHCPU	100 k steps	4096/8192
Q10UDEHCPU	100 k steps	4096/8192
Q13UDHCPU	130 k steps	4096/8192
Q13UDEHCPU	130 k steps	4096/8192
Q20UDHCPU	200 k steps	4096/8192
Q20UDEHCPU	200 k steps	4096/8192
Q26UDHCPU	260 k steps	4096/8192
Q26UDHCPU	260 k steps	4096/8192

ซึ่งแต่ละรุ่นในซีรี่ย์ของ Q จะมีขนาดและความสามารถในการทำงานที่แตกต่างกันออกไปตามราคาของเครื่อง ดูความแตกต่างของแต่ละรุ่นได้

ตารางที่ 2.12 เปรียบเทียบ Specifications ของพีแอลซีแต่ละรุ่น

Specifications		Q00UJCPU	Q00UCPU	Q01UCPU	Q02UCPU	Q03UDCPU, Q03UDECPU
Type		Multi processor CPU module				
I/O points		256/ 8169	1024/ 8192	1024/ 8192	2048/ 8192	4096/ 8192
CPU self-diagnostic functions		CPU error detection, Watch Dog, battery error detection, memory error detection, program check, power supply error detection, fuse error detection				
Battery buffer		All CPU modules are fitted with a lithium-battery with a life expectancy of 5 years.				
Memory type		RAM, ROM, FLASH	RAM, ROM, FLASH	RAM, ROM, FLASH	RAM, ROM, FLASH	RAM, ROM, FLASH
Memory capacity	Overall	<=32MByte	<=32MByte	<=32MByte	<=32MByte	<=32MByte

	Max.for PLC program	10 k steps (40 kByte)	10 k steps (40 kByte)	15 k steps (60 kByte)	20 k steps (80 kByte)	30 k steps (120 kByte)
Program cycle period		120 ns/log.instruction	80 ns/log.instruction	60 ns/log.instruction	40 ns/log.instruction	20 ns/log.instruction
Dimensions (WxHxD) mm		245x98x98	27.4x98x89.3	27.4x98x89.3	27.4x98x89.3	27.4x98x89.3
Order information Art.no.		221575	221576	221577	207604	207605, 217899

2.5 เอชเอ็มไอ

การใช้งานร่วมกันระหว่างพีแอลซี กับเครื่องคอมพิวเตอร์เรียกว่าเอชเอ็มไอ (Human Machine Interface: HMI) โดยนำคอมพิวเตอร์มาเป็นอุปกรณ์ที่ใช้ในการติดต่อระหว่างผู้ใช้งานกับเครื่องจักร เพื่อควบคุมและเป็นจอแสดงผลเอชเอ็มไอรวมไปถึงสกาดา (Supervisory Control And Data Acquisition: SCADA) เกิดจากความต้องการของผู้ใช้งานที่ต้องการเข้าไปควบคุมระบบที่พีแอลซีเป็นตัวควบคุมอยู่โดยเอชเอ็มไอ นั้นจะเป็นการนำข้อมูลจากพีแอลซีส่งผ่านโครงข่ายของการสื่อสารแบบต่าง ๆ และทำการรวบรวมข้อมูลในรูปแบบต่าง ๆ เข้าด้วยกัน และสามารถสั่งการได้โดยผู้เชี่ยวชาญงานอุตสาหกรรมในปัจจุบันเกือบทุกประเภท จะมีระบบควบคุมอัตโนมัติที่ใช้พีแอลซีเป็นตัวควบคุมและจะต้องใช้งานร่วมกันกับเอชเอ็มไอโดยใช้เอชเอ็มไอเป็นตัวสื่อสารระหว่างผู้ใช้งานกับระบบ Module พีแอลซีหรือจอแสดงผลต่าง ๆ โดยให้พีแอลซีสั่งงานไปที่เครื่องจักรอีกที

- **คุณสมบัติของเอชเอ็มไอ**

  1. การสื่อสาร (Communication)

สามารถสื่อสารข้อมูลกับอุปกรณ์อื่นๆ ในลักษณะแบบดิจิตอล โดยมีรูปแบบของสัญญาณให้เลือกหลายแบบ และสามารถสื่อสารข้อมูลกับอุปกรณ์ต่าง ๆ ทุกยี่ห้อได้อย่างมีประสิทธิภาพ สามารถต่อได้ทั้งอุปกรณ์ พีแอลซี, Meter, Controller และอีกมากมายตามการใช้งานประเภทต่าง ๆ โดยอุปกรณ์เอชเอ็มไอเพียงตัวเดียวก็สามารถควบคุม หรืออ่านค่าตัวอุปกรณ์ฮาร์ดแวร์อื่น ๆ ที่ต่อเชื่อมอยู่ได้อย่างง่ายดาย ผ่านการเชื่อมต่อทางเครือข่ายอินเตอร์เน็ต, Lan หรือ Wireless

2. การเก็บค่า (Collection)

สามารถเก็บข้อมูลกระบวนการผลิตต่าง ๆ ในรูปแบบไฟล์ Excel รวมไปถึงการเข้าถึงข้อมูล (Data logger) ผ่านทาง Web Browser ได้อย่างง่ายดาย ทำให้สะดวกในการทราบข้อมูล แม้ไม่ได้อยู่ที่หน้างานไลน์ผลิต สามารถอำนวยความสะดวกให้กับผู้ใช้งานในการดูค่าหรือควบคุมกระบวนการผลิตจากระยะไกล โดยการเชื่อมต่อผ่านมือถือหรือแท็บเล็ตใช้เว็บบราวเซอร์มาตรฐานตัวใดก็ได้ในการดูค่าหรือควบคุม โดยหน้าจอแสดงผลโชว์หน้าตาเสมือนว่าอยู่ตรงหน้า สามารถส่งข้อความ SMS หรือ email แจ้งเตือนให้กับบุคคลที่เกี่ยวข้อง

3. การเชื่อมต่อ (Connnection)

สามารถสื่อสารข้อมูลกับอุปกรณ์อื่น ๆ ในลักษณะแบบดิจิตอล โดยมีรูปแบบของสัญญาณให้เลือกหลายแบบ และสามารถสื่อสารข้อมูลกับอุปกรณ์ต่าง ๆ ทุกยี่ห้อได้อย่างมีประสิทธิภาพ สามารถต่อได้ทั้งอุปกรณ์พีแอลซี, Meter, Controller และอีกมากมายตามการใช้งานประเภทต่าง ๆ โดยอุปกรณ์เอชเอ็มไอเพียงตัวเดียวก็สามารถควบคุม หรืออ่านค่าตัวอุปกรณ์ฮาร์ดแวร์อื่น ๆ ที่ต่อเชื่อมอยู่ได้อย่างง่ายดาย ผ่านการเชื่อมต่อทางเครือข่ายอินเตอร์เน็ต, Lan หรือ Wireless

2.6 MELSOFT GX Work2

GX Works2 คือซอฟต์แวร์ประยุกต์สำหรับใช้งานกับ PLC MELSEC, GX Works2 เป็นซอฟต์แวร์รุ่นใหม่ที่รวมความสามารถของ GX developer ,GX explorer และ GX simulator ไว้ในซอฟต์แวร์ตัวเดียว เมื่อติดตั้ง GX works2 สามารถใช้งานได้ครบทุกอย่างโดยไม่ต้องใช้ซอฟต์แวร์อื่น ๆ อีก โปรแกรมที่เขียนโดยใช้ GX developer เมื่อเขียนไปที่พีแอลซีแล้ว ก็สามารถอ่านหรือแก้ไขโดยใช้ GX works2 ได้ การเขียนโปรแกรมสำหรับพีแอลซีรุ่นใหม่เช่น FX3S จะต้องใช้ GX works2

2.6.1 การสร้าง Simple Project และ Structured Project

1. Simple Project

Simple Project ใน Simple Project คุณสามารถสร้างโปรแกรมแบบลำดับโดยใช้คำสั่งสำหรับซีพียูระบบพีแอลซีของ Mitsubishi Simple Project มีความสามารถในการทำงานสำหรับการสร้างโปรแกรมแบบเดียวกับ GX Developer รุ่นเดิม คุณสามารถสร้างโปรแกรมแบบลำดับโดยใช้ภาษาการโปรแกรมต่อไปนี้

- **ภาษากราฟิก**

Ladder ใช้ภาษากราฟิกนี้ เพื่ออธิบายโปรแกรมในลักษณะเป็นแลดเดอร์ที่ประกอบด้วยหน้าสัมผัสคอยล์ ฯลฯ โดยใช้ขั้นตอน การทำงานเดียวกันกับ GX Developer รุ่นเดิม

SFC ใช้ภาษากราฟิกนี้เพื่ออธิบายการควบคุมลำดับในแบบที่เข้าใจได้ง่าย อธิบายขั้นตอนที่ระบุเงื่อนไขการประมวลผล และการเปลี่ยนแปลงซึ่งระบุเงื่อนไขต่าง ๆ สำหรับการดำเนินการต่อในขั้นตอนถัดไป สามารถอธิบายขั้นตอนและเงื่อนไขการเปลี่ยนแปลง โดยใช้ภาษาแลดเดอร์

- **ภาษาข้อความ**

ST (Structured Text) ภาษาข้อความนี้ทำให้คุณสามารถอธิบายการควบคุมโดยไวยากรณ์คำสั่งที่มีลำดับแบบตัวเลือกที่เสนอโดยประโยคแบบเงื่อนไข และการซ้ำกันที่เสนอโดยประโยคซ้ำกันในลักษณะเดียวกันกับภาษาระดับสูงอย่างภาษาซีด้วยเหตุนี้ จึงสามารถสร้างโปรแกรมที่ตรวจสอบได้ง่ายในเวลาสั้น ๆ

2. Structured Project

Structured Project สามารถสร้างโปรแกรมด้วยโปรแกรมเชิงโครงสร้าง ด้วยการแบ่งตัวควบคุมออกเป็นส่วนย่อย และทำให้เป็นส่วนต่าง ๆ ของเนื้อหาร่วม จึงสามารถสร้างโปรแกรมที่เข้าใจได้ง่าย และสามารถใช้งานได้ในหลายกรณีตามโปรแกรมเชิงโครงสร้าง สามารถสร้างโปรแกรมแบบลำดับโดยใช้ภาษาการโปรแกรมต่อไปนี้

- **ภาษากราฟิก**

Ladder ใช้ภาษากราฟิกนี้ เพื่ออธิบายโปรแกรมในลักษณะเป็นแลดเดอร์ ที่ประกอบด้วยหน้าสัมผัส คอยล์ ฯลฯ โดยใช้ขั้นตอนการทำงานเดียวกันกับ GX Developer รุ่นเดิม

Structured Ladder/FBD Structured Ladder สร้างตามเทคโนโลยี การออกแบบวงจรเลยเนื่องจากภาษานี้สามารถเข้าใจได้ง่าย จึงมีการนำมาใช้กัน โดยทั่วไปสำหรับโปรแกรมแบบลำดับ ทุกแลดเดอร์เริ่มต้นจากเส้นฐานทางด้านซ้ายเสมอ Structured Ladder ประกอบด้วยหน้าสัมผัสคอยล์ ฟังก์ชันบล็อกและฟังก์ชันที่มีการเชื่อมต่อกับฟังก์ชันอื่นด้วยเส้นแนวตั้งและเส้นแนวนอน FBD เชื่อมต่อฟังก์ชันและฟังก์ชันบล็อกด้วยเส้นที่เป็นกฎในการอธิบายแลดเดอร์

SFC ใช้ภาษากราฟิกนี้เพื่ออธิบายการควบคุมลำดับในแบบที่เข้าใจได้ง่าย อธิบายขั้นตอนที่ระบุเงื่อนไขการประมวลผล และการเปลี่ยนแปลงซึ่งระบุเงื่อนไขต่าง ๆ สำหรับการดำเนินการต่อในขั้นตอนถัดไป สามารถอธิบายขั้นตอนและเงื่อนไขการเปลี่ยนแปลงโดยใช้ภาษาแลดเดอร์

- **ภาษาข้อความ**

ST (Structured Text) ภาษาข้อความนี้ สามารถอธิบายการควบคุมโดยไวยากรณ์คำสั่งที่มีลำดับแบบตัวเลือกที่เสนอโดยประโยค แบบเงื่อนไขและการซ้ำกันที่เสนอโดยประโยคซ้ำกันในลักษณะเดียวกันกับภาษาระดับสูงอย่างภาษา C ด้วยเหตุนี้ จึงสามารถสร้างโปรแกรมที่ตรวจสอบได้ง่ายในเวลาสั้น ๆ

2.6.2 การเขียนโปรแกรม Structured text

ภาษาโปรแกรมสำหรับพีแอลซี นั้นถูกกำหนดมาตรฐานโดย IEC 61131-3ST ก็เป็นหนึ่งในภาษาโปรแกรมมาตรฐานนั้น แต่ละภาษาก็มีลักษณะพิเศษที่มีความแตกต่างกันตามการใช้งานและทักษะของผู้เขียนโปรแกรม ตารางต่อไปนี้แสดงลักษณะพิเศษของภาษาโปรแกรมของมาตรฐาน IEC 61131-3

ตารางที่ 2.13 ลักษณะพิเศษของภาษาโปรแกรมตามมาตฐาน IEC 61131-3

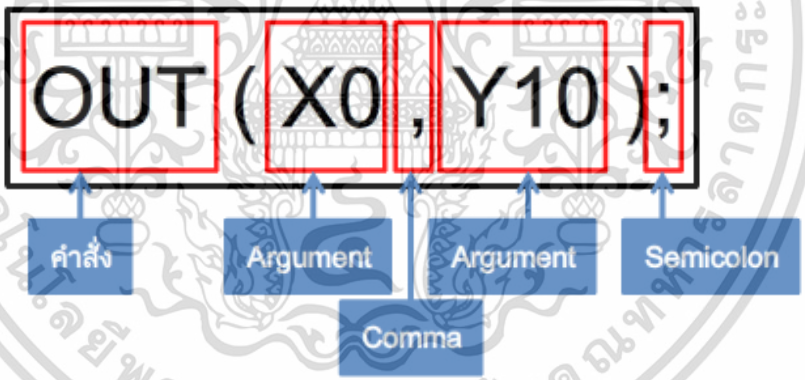
ภาษาโปรแกรม	ลักษณะพิเศษ
Ladder Diagram (LD)	● ใช้สัญลักษณ์หน้าสัมผัสและขดลวดในการสร้างโปรแกรมประกอบมาเป็นวงจรไฟฟ้า ● การไหลของโปรแกรม ง่ายต่อการติดตามและการทำความเข้าใจ แม้แต่สำหรับผู้เริ่มต้น
Structured Text (ST)	● โปรแกรมถูกเขียนด้วยข้อความ (อักขระ) ● ST นั้นง่ายต่อการเรียนรู้สำหรับผู้ที่มีประสบการณ์ในการเขียนโปรแกรมด้วยภาษา C หรือภาษา BASIC ● สูตรคำนวณต่าง ๆ นั้นเหมือนนิพจน์ทางคณิตศาสตร์ ซึ่งเข้าใจได้ง่าย ● ST เหมาะกับการใช้งานข้อมูล
Function Block Diagram (FBD)	● โปรแกรมเขียนโดยการเรียงบล็อกที่มีฟังก์ชันต่าง ๆ กันและระบุความสัมพันธ์ระหว่างบล็อกต่าง ๆ

	• FBD เพิ่มความสามารถในการอ่านได้ดีกว่าภาษาอื่น เนื่องจากการทำงานทั้งหมดสามารถมองเห็นได้โดยง่าย
Sequential Function Chart (SFC)	• เงื่อนไขและการดำเนินการต่าง ๆ ถูกเขียนเป็น Flowcharts • การไหลของโปรแกรม ง่ายต่อการทำความเข้าใจ
Instruction List (IL)	• IL เหมือนกับภาษาเครื่องจักร • IL ไม่ค่อยได้ใช้แล้วในปัจจุบัน

1.พื้นฐานในการเขียนโปรแกรม

- **คำสั่งควบคุม I/O**

ในส่วนเป็นการแสดงตัวอย่างของโปรแกรม ST พื้นฐาน จากโปรแกรมตัวอย่างต่อไปนี้ เอาท์พุท Y10 ON เมื่ออินพุท X0 ทำงาน และ Y10 OFF เมื่อ X0 ปิดการทำงาน

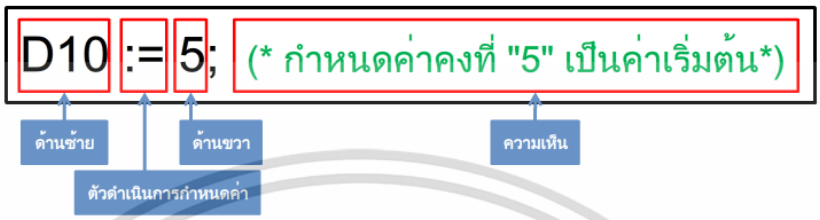


รูปที่ 2.12 ตัวอย่างคำสั่งควบคุม I/O

จำนวน Argument ขึ้นอยู่กับคำสั่ง Argument หลายตัวแบ่งโดยใช้ Comma (,) บรรทัดด้านบนแสดงคำสั่งหนึ่งคำสั่ง แต่ละคำสั่งจะสิ้นสุดด้วย Semicolon (;) โปรแกรมหนึ่งโปรแกรมประกอบด้วยการรวมคำสั่งต่าง ๆ เข้าด้วยกัน

- **คำสั่งการกำหนดค่า**

ตัวอย่างต่อไป แสดงโปรแกรมที่ใช้การกำหนดค่า คำสั่งต่อไปนี้ กำหนดค่าคงที่ เลขฐานสิบ "5" ไปยังตัวแปร "D10"



รูปที่ 2.13 ตัวอย่างคำสั่งการกำหนดค่า

ตัวดำเนินการกำหนดค่า (=) ใช้สำหรับคำสั่งการกำหนดค่า หมายเหตุ เครื่องหมาย colon (:) จะวางทางด้านซ้ายของเครื่องหมายเท่ากับ (=) ตัวดำเนินการการกำหนดค่าจะกำหนดค่าทางด้านขวาไปยังค่าทางด้านซ้าย การเพิ่ม Comment ไปยังโปรแกรม ทำให้การทำงานเข้าได้ใจง่ายมากยิ่งขึ้น กรอก comments เข้าไประหว่างดอกจันทั้งสองด้าน (**)

2.6.3 การระบุสัญลักษณ์เชิงตัวเลข

จากโปรแกรมตัวอย่างที่ผ่านมา ได้กำหนดค่าเลขฐานสิบไปยังตัวแปร บางครั้งจำเป็นต้องใช้ค่าที่ไม่ใช่เลขฐานสิบเช่น เลขฐานสองและเลขฐานสิบหกในการควบคุม ตารางต่อไปนี้แสดงประเภทของสัญลักษณ์เชิงตัวเลขที่ใช้ในพีแอลซีของ Mitsubishi

ตารางที่ 2.14 ตัวอย่างคำสั่งควบคุม I/O

ประเภทของสัญลักษณ์เชิงตัวเลข	วิธีการระบุ	ตัวอย่าง
เลขฐานสอง	เพิ่ม “2#” ที่ด้านหน้า	2#11010
เลขฐานแปด	เพิ่ม “8#” ที่ด้านหน้า	8#32
เลขฐานสิบ	กรอก input โดยตรง	26
	เพิ่ม “K” ที่ด้านหน้า	K26
เลขฐานสิบหก	เพิ่ม “16#” ที่ด้านหน้า	26#1A
	เพิ่ม “H” ที่ด้านหน้า	H1A

1. การระบุสัญลักษณ์ bit

Bits แสดงถึงสถานะ true/false เช่น สถานะ on/off ของสัญญาณ อีกทั้งยังแสดงถึงสถานการณ์มีอยู่การไม่มีอยู่ อีกด้วย ในภาษา ST Bits ไม่สามารถเขียนเป็น "ON" และ "OFF" ได้ แต่ต้องเขียนเป็น "1" (ON) และ "0" (OFF) แทน และยังสามารถเขียนเป็น "TRUE" และ "FALSE" ได้อีกด้วย

ตารางที่ 2.15 ตัวอย่างสัญลักษณ์ของสถานะ

สถานะ	ON	OFF
	True	False
สัญลักษณ์ตัวเลข	1	0
สัญลักษณ์ true/false	TRUE	FALSE

2.6.4 ลำดับการดำเนินการโปรแกรม

```
Y10 := (X0 OR X1) AND X2;      (* ดำเนินการลำดับแรก *)
Y11 := X3 AND X4;              (* ดำเนินการลำดับที่สอง *)
Y12 := X3 AND X5;   (* ดำเนินการลำดับที่สาม ไม่ต้องมีคำสั่ง END ในการสิ้นสุด*)
```

ดำเนินการซ้ำ

รูปที่ 2.14 ตัวอย่างสัญลักษณ์ของสถานะ

แม้ว่าคำสั่ง END จะจำเป็นในการสิ้นสุดโปรแกรมในภาษา Ladder แต่ไม่จำเป็นในภาษา Structured text

- **การดำเนินการเชิงคณิตศาสตร์**

  อธิบายถึงวิธีการสร้างโปรแกรมการทำงานเชิงคณิตศาสตร์

  - การอธิบายการดำเนินการเชิงคณิตศาสตร์

โปรแกรมตัวอย่างต่อไปนี้รวมปริมาณการผลิตของสายการผลิตสองสาย ด้านขวาของสมการเป็นการดำเนินการทางคณิตศาสตร์ประกอบด้วยตัวแปรและตัวดำเนินการทางคณิตศาสตร์

รูปที่ 2.15 ตัวอย่างโปรแกรมที่มีการคำนวณทางคณิตศาสตร์

○ การกำหนดประเภทข้อมูลตามช่วงจำนวน

ประเภทข้อมูลจะต้องกำหนดให้แต่ละตัวแปรในการระบุช่วงของค่าในการใช้งาน ประเภทข้อมูลตัวเลขที่ใช้งานในภาษา ST ได้แก่ บิต จำนวนเต็ม และจำนวนจริง ภาษา ST มีประเภทข้อมูลหลายชนิด ตารางด้านล่างแสดงรายการประเภทข้อมูลที่ใช้งานในหลักสูตรนี้

ตารางที่ 2.16 ตัวอย่างประเภทข้อมูล

ประเภทข้อมูล		ช่วงข้อมูล
บิต		สถานะ ON/OFF (เปิด/ปิด) ของอุปกรณ์บิตและสถานะ true/false (จริง/เท็จ) ของผลลัพธ์การดำเนินการ
จำนวนเต็ม	เวิร์ด (ไม่มีเครื่องหมาย)	0 – 65,535
	เวิร์ด (มีเครื่องหมาย)	-32,768 – 32,767
	ดับเบิล (ไม่มีเครื่องหมาย)	0 – 4,294,967,295
	ดับเบิล (มีเครื่องหมาย)	-2,147,483,648 - 2,147,483,647

เมื่อใช้ตัวแปรประเภทจำนวนเต็ม โปรดเลือกประเภทเป็นเวิร์ดหรือดับเบิลตามช่วงของข้อมูลที่ต้องการใช้ และเลือกว่าจะมี เครื่องหมายหรือไม่ตามความจำเป็นในการใช้งานจำนวนติดลบ ระบุประเภทข้อมูลของตัวแปรขณะตั้งชื่อลาเบลโดยใช้ซอฟต์แวร์ MELSOFT

○ การกำหนดชื่อตัวแปรเพื่อหลีกเลี่ยงความไม่สอดคล้องกันของประเภทข้อมูล

การใช้งานประเภทข้อมูลต่างกันทางด้านซ้ายและขวาของสมการ อาจก่อให้เกิดข้อผิดพลาดหรือผลลัพธ์ที่ไม่คาดคิดได้ ด้านล่างเป็นตัวอย่างของกรณีดังกล่าว

```
ValueA := ValueB;      (* ValueA: จำนวนเต็มประเภทเวิร์ด  ValueB: จำนวนเต็มประเภทดับเบิล *)
```

รูปที่ 2.16 ตัวอย่างโปรแกรมความไม่สอดคล้องกันของประเภทข้อมูล

จำนวนเต็มประเภทดับเบิล ไม่สามารถกำหนดค่าไปยังจำนวนเต็มประเภทเวิร์ดได้ อย่างไรก็ตาม ในกรณีนี้ประเภทข้อมูลจะ ไม่สามารถกำหนดได้ การเพิ่มตัวอักษรด้านหน้าชื่อตัวแปรเพื่อระบุประเภทข้อมูลจะทำให้ระบุประเภทข้อมูลได้ง่ายยิ่งขึ้น วิธีการระบุชื่อตัวแปรแบบนี้เรียกว่าการระบุสัญลักษณ์แบบฮังกาเรียน

ตารางที่ 2.17 การระบุสัญลักษร์แบบฮังกาเรียน

ประเภทข้อมูล		ช่วงข้อมูล	ตัวอักษรด้านหน้า	คำเต็มของตัวอักษรด้านหน้า
บิต		สถานะ ON/OFF (เปิด/ปิด) ของอุปกรณ์บิตและสถานะ true/false (จริง/เท็จ) ของผลลัพธ์การดำเนินการ	b	Bit (บิต)
จำนวนเต็ม	เวิร์ด (ไม่มีเครื่องหมาย)	0 – 65,535	u	unsigned word (เวิร์ดแบบไม่มีเครื่องหมาย)
	เวิร์ด (มีเครื่องหมาย)	-32,768 – 32,767	w	Signed word (เวิร์ดแบบมีเครื่องหมาย)
	ดับเบิล (ไม่มีเครื่องหมาย)	0 – 4,294,967,295	ud	Unsigned double-word (ดับเบิลแบบไม่มีเครื่องหมาย)
	ดับเบิล (มีเครื่องหมาย)	-2,147,483,648 - 2,147,483,647	d	Signed double-word (ดับเบิลแบบมีเครื่องหมาย)

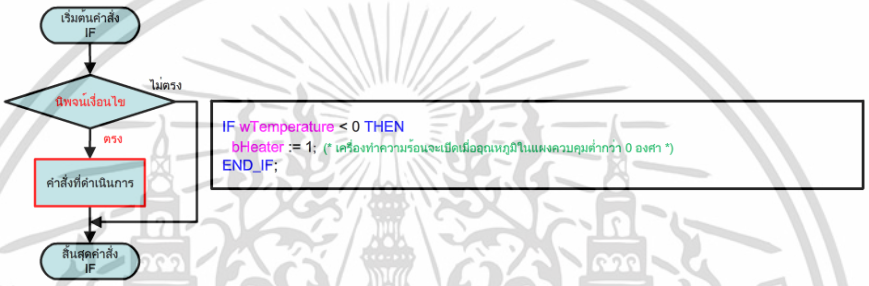
2.6.5 การแยกเงื่อนไข

โปรแกรมควบคุมจะมีส่วนของโค้ดซึ่งจะเปลี่ยนรูปแบบการทำงานตามเงื่อนไขที่กำหนด บทนี้อธิบายถึงการแตกสาขาแบบมีเงื่อนไข

```
IF conditional expression THEN
    Execution statement;          (* คำสั่งจะดำเนินการหากนิพจน์เงื่อนไขถูกต้อง *)
END_IF;                           (* END_IF; ต้องถูกวางในการสิ้นสุดคำสั่ง IF *)
```

รูปที่ 2.17 วิธีการใช้คำสั่ง IF

ในโปรแกรมตัวอย่างนี้คำสั่งจะทำงานเมื่อเงื่อนไขถูกต้อง แต่หากเงื่อนไขไม่ถูกต้องคำสั่งจะไม่ทำงาน

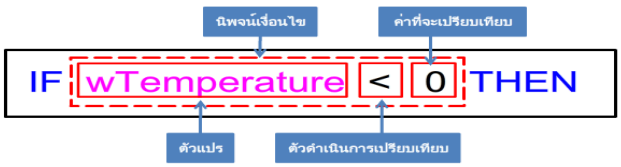


รูปที่ 2.18 ขั้นตอนการทำงานในโปรแกรมตัวอย่าง IF

ตัวอย่างด้านบนแสดงการแยกของโปรแกรมโดยการเปรียบเทียบค่าของตัวแปร ในโปรแกรมตัวอย่าง เครื่องทำความร้อนจะเปิดเมื่ออุณหภูมิที่แผงควบคุมต่ำกว่า 0 องศา

1. การเขียนนิพจน์เงื่อนไข

หน้าที่ผ่านมานั้น ใช้นิพจน์เงื่อนไข "wTemperature < 0" ซึ่งหมายถึง "เมื่อค่าของตัวแปร WTemperature น้อยกว่า 0" นิพจน์แบบมีเงื่อนไขใช้ตัวดำเนินการเปรียบเทียบในการแสดงความสัมพันธ์ระหว่างตัวแปรและค่าต่าง ๆ ในการเปรียบเทียบ



รูปที่ 2.19 ขั้นตอนการทำงานในโปรแกรมตัวอย่าง

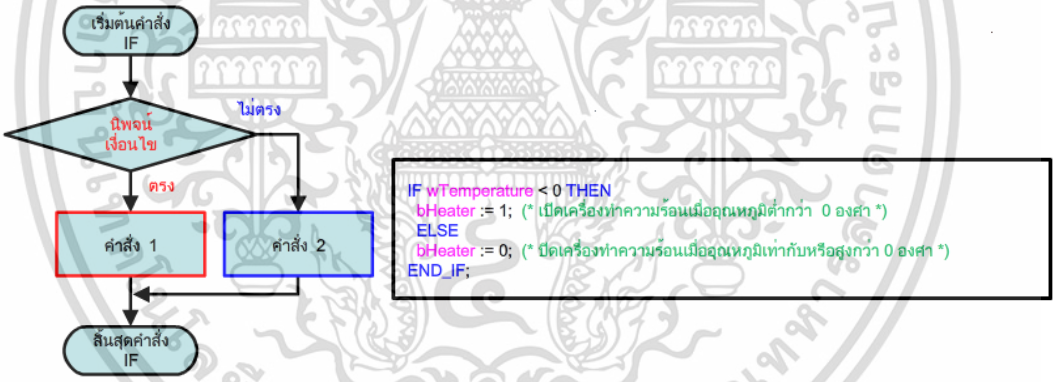
ที่ด้านซ้ายและขวามือของตัวดำเนินการเปรียบเทียบ ค่าจะเขียนเป็นตัวแปรหรือค่าคงที่ในการเปรียบเทียบ นอกจากจะใช้ในการเปรียบเทียบตัวแปรและค่าคงที่แล้ว นิพจน์เงื่อนไขยังสามารถใช้เปรียบเทียบผลลัพธ์ของการดำเนินการ เชิงตรรกะหรือตัวแปรประเภทบิตได้อีกด้วย

2. การแยกเงื่อนไขเพิ่มเติมสำหรับคำสั่ง ELSIF

คำสั่ง IF โดยทั่วไปแล้ว ใช้ในการดำเนินการคำสั่งเมื่อนิพจน์เงื่อนไขถูกต้อง ในการดำเนินการคำสั่งอื่นๆ เมื่อนิพจน์เงื่อนไขไม่ถูกต้อง จะใช้คำสั่ง ELSE

```
IF conditional expression THEN
  Execution statement 1;        (* คำสั่ง 1 จะดำเนินการหากนิพจน์เงื่อนไขถูกต้อง *)
ELSE
  Execution statement 2;        (* คำสั่ง 2 จะดำเนินการหากนิพจน์เงื่อนไขไม่ถูกต้อง *)
END_IF;
```

รูปที่ 2.20 ตัวอย่างการแยกเงื่อนไขแบบสำหรับคำสั่ง ELSIF



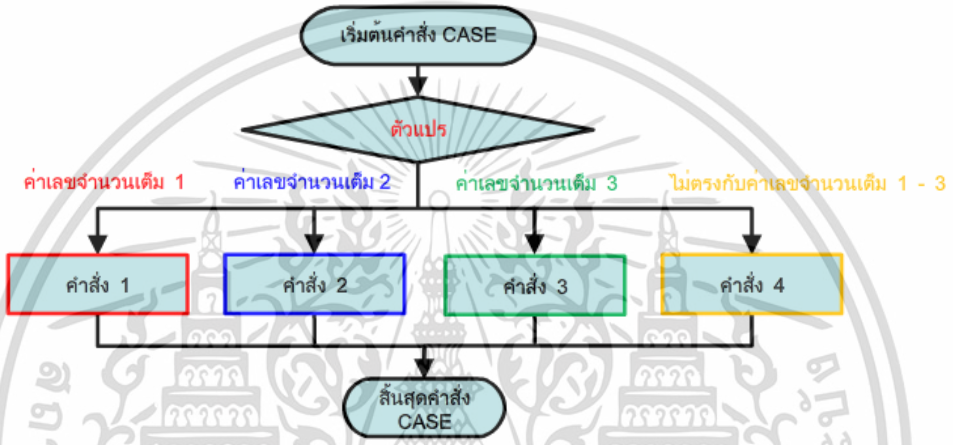
รูปที่ 2.21 ขั้นตอนการทำงานในโปรแกรมตัวอย่าง ELSE

3. การแยกเงื่อนไขตามค่าเลขจำนวนเต็ม CASE

คำสั่ง IF ใช้ในการแยกเงื่อนไขขึ้นอยู่กับว่านิพจน์เงื่อนไขนั้นถูกต้องหรือไม่ คำสั่ง CASE จะใช้ในการแยกเงื่อนไขตามค่าเลขจำนวนเต็ม รูปภาพต่อไปนี้แสดงวิธีการใช้งานคำสั่ง CASE

```
CASE Variable OF
    Integer value 1: Execution statement 1;        (* คำสั่ง 1 จะดำเนินการเมื่อตัวแปรตรงกับค่าจำนวนเต็ม 1 *)
    Integer value 2: Execution statement 2;        (* คำสั่ง 2 จะดำเนินการเมื่อตัวแปรตรงกับค่าจำนวนเต็ม 2 *)
    Integer value 3: Execution statement 3;        (* คำสั่ง 3 จะดำเนินการเมื่อตัวแปรตรงกับค่าจำนวนเต็ม 3 *)
    ELSE Execution statement 4;        (* คำสั่ง 4 จะดำเนินการหาก ตัวแปรไม่ตรงกับค่าจำนวนเต็ม r ใดๆ *)
END_CASE;                              (* "END_CASE;" ต้องถูกวางในการสิ้นสุดคำสั่ง CASE *)
```

รูปที่ 2.22 ตัวอย่างการแยกเงื่อนไขตามค่าเลขจำนวนเต็ม CASE



รูปที่ 2.23 ขั้นตอนการทำงานในโปรแกรมเลขจำนวนเต็ม CASE

จุดประสงค์ของแต่ละประเภท

ตารางที่ 2.18 สรุปจุดประสงค์ของแต่ละประเภท

คำสั่ง IF	• โปรแกรมจะถูกแยกเงื่อนไขด้วยคำสั่ง IF เมื่อการ แสดงเงื่อนไขถูกต้อง • คำสั่ง ELSE ใช้ในการแยกเงื่อนไขเมื่อการแสดงเงื่อนไขไม่ถูกต้อง • คำสั่ง ELSE ใช้ในการเพิ่มการแยกเงื่อนไขเมื่อการแสดงเงื่อนไขในคำสั่ง IF ไม่ถูกต้อง
การแสดงเงื่อนไข	• การแสดงเงื่อนไขแสดงความสัมพันธ์ระหว่างตัวแปรและค่าในการเปรียบเทียบโดยใช้ตัวดำเนินการเปรียบเทียบ
คำสั่ง CASE	• คำสั่ง CASE จะใช้ในการแยกเงื่อนไขตามค่าเลขจำนวนเต็ม

2.6.6 การจัดเก็บและการใช้งานข้อมูล

ตัวควบคุมแบบตั้งโปรแกรมได้นั้น นอกจากจะใช้ในงานควบคุม i/o ก็ใช้ในการประมวลผลข้อมูลปริมาณมากเป็นแกนของระบบ การผลิต ในการประมวลผลข้อมูลปริมาณมาก ข้อมูลจำเป็นจะต้องถูกเก็บและนำมาอ่าน บทนี้จะอธิบายถึงวิธีการเขียนโปรแกรมเพื่อจัดเก็บและประมวลผลข้อมูลอย่างแม่นยำ

- Arrays ใช้ในการจัดลำดับและจัดการตัวแปร
- โครงสร้างข้อมูลใช้ในการจัดการตัวแปรที่เกี่ยวข้อง
- การใช้งานลูปทำให้ประมวลผล Arrays ได้อย่างมีประสิทธิภาพมากขึ้นโดยใช้คำสั่ง FOR

โปรแกรมที่จัดเก็บและจัดการข้อมูลได้อย่างแม่นยำสามารถทำได้โดยใช้ Arrays โครงสร้างข้อมูล และคำสั่ง FOR

1. การจัดลำดับและการจัดเก็บข้อมูล (Array)

ค่าหลายค่าสามารถจัดการได้ในตัวแปรเดียวโดยใช้ Arrays ในตัวอย่างในรูปที่ 2.24 ข้อมูลปริมาณการผลิตของโรงงานผลิตรถยนต์ถูกเก็บค่าไว้ตามประเทศเป้าหมาย

เป้าหมาย	ประเทศ A	ประเทศ B	ประเทศ C
ปริมาณการผลิต	35	75	65

รูปที่ 2.24 ตัวอย่างข้อมูลปริมาตรในการผลิต

ปริมาณการผลิตตามประเทศเป้าหมายถูกกำหนดไปยังตัวแปรหนึ่งตัว หากไม่ได้ใช้ Arrays จะต้องสร้างตัวแปรหนึ่งตัวสำหรับแต่ละเป้าหมาย แต่หากใช้ Arrays ปริมาณการผลิตสำหรับหลายเป้าหมายสามารถกำหนดและเก็บไว้ในตัวแปรเดียว ดังตัวอย่างในรูปที่ 2.25

รูปที่ 2.25 ตัวแปรแต่ละตัว Array โดยใช้หมายเลของค์ประกอบ

```
uShowProductionPlan := uProduction[0];
(* กำหนดจำนวนองค์ประกอบสำหรับประเทศ A *)
```

รูปที่ 2.26 ตัวอย่างการกำหนดตัวแปรของปริมาณ

- **ตัวแปร Array (Array matrix)**

สีรถ (column)

เป้าหมาย (แถว)	แดง	เหลือง	น้ำเงิน
ประเทศ A	[0,0] 10	[0,1] 5	[0,2] 20
ประเทศ B	[1,0] 15	[1,1] 40	[1,2] 20
ประเทศ C	[2,0] 25	[2,1] 30	[2,2] 10

รูปที่ 2.27 ตัวอย่างข้อมูลปริมาตรในการผลิตแบบเมทริกซ์

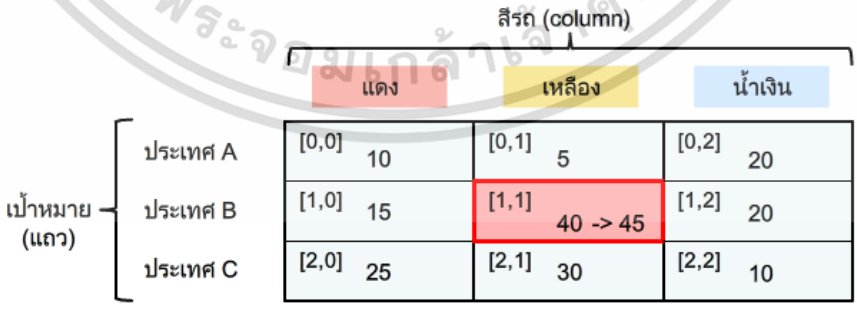
รูปที่ 2.28 ตัวอย่างการกำหนดตัวแปร Array matrix

- **การกำหนด matrix array**

โปรแกรมตัวอย่างต่อไปนี้กำหนดจำนวนรถที่ต้องผลิตเพิ่มเติมจากปริมาณการผลิตที่วางแผนไว้อย่างเร่งด่วนสำหรับรถสีเหลืองที่ส่งไปยัง ประเทศ B โดยใช้ Array matrix

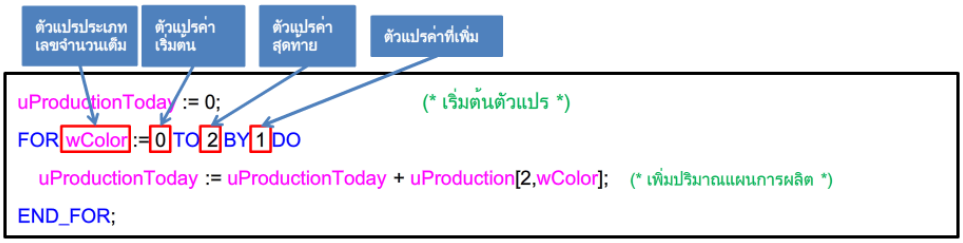
```
uAdditionalProduction := 5;
uProduction[1,1] := uProduction[1,1] + uAdditionalProduction;
(* เพิ่มจำนวนการผลิตเพิ่มเติม (5 หน่วย) ไปยังปริมาณแผนการผลิตขั้นต้น *)
```

รูปที่ 2.29 การกำหนดตัวแปร Array matrix



รูปที่ 2.30 ตัวอย่างข้อมูลปริมาตรในการผลิตที่ถูกกำหนดแบบ Array matrix

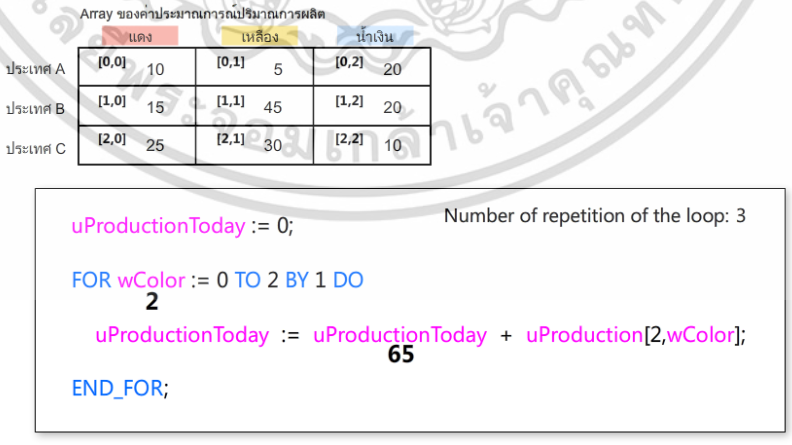
2.6.7 การวนลูป



รูปที่ 2.31 ตัวอย่างการเก็บค่าโดยใช้คำสั่งการวนลูป FOR

ในการใช้คำสั่ง FOR ตัวแปร "wColor" เพิ่มค่าทีละหนึ่งโดยเริ่มจากค่าเริ่มต้นที่เป็นศูนย์ และคำสั่งจะทำซ้ำไปเรื่อยๆ จนกระทั่งตัวแปรเป็นค่า สองตัวแปร "wColor" ถูกกำหนดให้เป็นองค์ประกอบที่สองในอาร์เรย์ "uProduction" ในคำสั่งที่ดำเนินการ ค่าของตัวแปร "wColor" จะเพิ่มทุกครั้งที่คำสั่งกลับมาทำซ้ำ ปริมาณการผลิตที่วางแผนไว้สำหรับแต่ละสีรถได้นำมาบวกกันในแต่ละครั้ง เพื่อให้ได้ค่ารวม โปรแกรมตัวอย่างนี้ได้ดำเนินการในลูปสามครั้ง ครั้งแรก: สีแดง [0] => ครั้งที่สอง: สีเหลือง [1] => ครั้งที่สาม: สีน้ำเงิน (21)

การดำเนินการของคำสั่ง FOR ถูกอธิบายโดยใช้การทำงานของตัวอย่างโปรแกรม



รูปที่ 2.32 ตัวอย่างโปรแกรมการวนลูป FOR

2.7 หุ่นยนต์ในอุตสาหกรรม

2.7.1 ความหมายของหุ่นยนต์ในอุตสาหกรรม

สถาบันหุ่นยนต์อเมริกา (The Robotics Institute Of America) ได้ให้ความหมายของหุ่นยนต์ว่า หุ่นยนต์คือเครื่องจักรที่ถูกออกแบบให้สามารถทำงานได้หลากหลายหน้าที่เพื่อใช้เคลื่อนย้ายวัสดุ ชิ้นงานเครื่องมือ หรืออุปกรณ์พิเศษ ผ่านโปรแกรมควบคุมการเคลื่อนที่ต่าง ๆ สำหรับงานต่าง ๆ ที่หลากหลาย อย่างมีประสิทธิภาพหรือหุ่นยนต์คือ เครื่องจักรกลทุกชนิดที่ออกแบบมาให้สามารถทำงานแทนมนุษย์ได้ทุกประเภทที่มนุษย์ไม่สามารถปฏิบัติงานได้ และเป็นการทำงานอัตโนมัติ สามารถทำงานในรูปแบบที่ ซับซ้อนและมีความยืดหยุ่น หุ่นยนต์หรือภาษาอังกฤษเขียนว่า Robot มาจากบทละครของ นายคาเรล คาเปก (Karal Kapek) นักแต่งนิยายชาวเช็ก เรื่อง R.U.R (Rossum's Universal Robots) ซึ่งหมายถึงคนงานหุ่นยนต์คือเครื่องจักรที่ถูกควบคุมอัตโนมัติ สามารถเขียนโปรแกรมใหม่ได้ ใช้งานเอนกประสงค์ โปรแกรมการเคลื่อนที่จะต้องสามารถโปรแกรมให้เคลื่อนที่ได้อย่างน้อย 3 แกนหรือมากกว่าหุ่นยนต์ อาจจะยึดอยู่กับที่หรือย้ายตำแหน่ง (Mobile)

2.7.2 ชนิดของหุ่นยนต์

โดยทั่วไปการแบ่งชนิดของหุ่นยนต์จะแบ่งตามลักษณะรูปทรงของพื้นที่ทำงาน (Envelope Geometric) แต่ก่อนจะอธิบายชนิดของหุ่นยนต์มาอธิบายการทำงานของจุดต่อ (Joint) ของหุ่นยนต์อุตสาหกรรมซึ่งในขั้นพื้นฐานมี 2 ชนิดด้วยกันดังนี้

ตารางที่ 2.19 การแยกชนิดของหุ่นยนต์ตามลักษณะการหมุน

ชนิด	สัญลักษณ์	หมายเหตุ
Revolute		เป็นการหมุนรอบแกน
Prismatic		การเคลื่อนที่เชิงเส้น

จุดต่อ (Joint) ทั้งสองแบบเมื่อนำมาต่อเข้าด้วยกันอย่างน้อย 3 แกนหลักจะได้พื้นที่ทำงาน (Work envelope) ที่มีลักษณะแตกต่างกันไป ซึ่งสามารถนำมาแบ่งชนิดของหุ่นยนต์ได้ดังนี้

ตารางที่ 2.20 การแบ่งแยกกลุ่มหุ่นยนต์ตามลักษณะการหมุน

ชนิดของหุ่นยนต์	แกนที่ 1 (เอว)	แกนที่ 2 (ไหล่)	แกนที่ 3 (ข้อศอก)
Cartesian	P	P	P

Cylindrical	P	P	P
Spherical	R	R	P
SCARA	R	P	R
Articulated	R	R	R

1. หุ่นยนต์ชนิดคาร์ทีเซียน (Cartesian (gantry) Robot)

แกนทั้ง 3 ของหุ่นยนต์จะเคลื่อนเป็นแบบเชิงเส้น (Prismatic) ถ้าโครงสร้างมีลักษณะ คล้าย Overhead Crane จะเรียกว่าเป็นหุ่นยนต์ชนิด gantry แต่ถ้าหุ่นยนต์ไม่มีขาตั้งหรือขาเป็น แบบอื่น เรียกว่าชนิด Cartesian

รูปที่ 2.33 หุ่นยนต์ Cartesian Robot

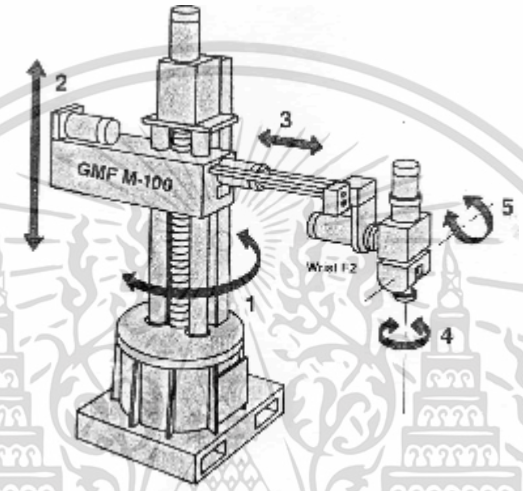
การประยุกต์ใช้งาน เนื่องจากโครงสร้างมีความแข็งแรงตลอดแนวการเคลื่อนที่ ดังนั้นจึงเหมาะกับงานเคลื่อนย้าย หนักๆหรือเรียกว่างาน Pick-and-Plane เช่นใช้โหลดชิ้นงานเข้าเครื่องจักรกล (Machine loading), ใช้จัดเก็บชิ้นงาน (Stacking) นอกจากนี้ยังสามารถใช้ในการงานประกอบ (Assembly) ที่ไม่ต้องการ เข้าถึงในลักษณะที่มีมุมหมุน เช่น ประกอบอุปกรณ์อิเล็กทรอนิกส์ และงาน Test ต่างๆ

ข้อดี เคลื่อนที่เป็นแนวเส้นตรงทั้ง 3 มิติ การเคลื่อนที่สามารถทำความเข้าใจง่าย มีส่วนประกอบง่ายๆ โครงสร้างแข็งแรงตลอดการเคลื่อนที่

ข้อเสีย ต้องการพื้นที่ติดตั้งมาก บริเวณที่หุ่นยนต์เข้าไปทำงานได้จะเล็กกว่าขนาดของตัวหุ่นยนต์ ไม่สามารถเข้าถึงวัตถุจากทิศทางข้างได้ แกนแบบเชิงเส้นจะ Seal เพื่อป้องกันฝุ่นและของเหลวได้ยาก

2. หุ่นยนต์ชนิดทรงกระบอก (Cylindrical Robot)

หุ่นยนต์ประเภทนี้จะมีแกนที่ 2 (ไหล่) และแกนที่ 3 (ข้อศอก) เป็นแบบหมุน (revolute) ทำให้การเคลื่อนที่ได้พื้นที่การทำงานเป็นรูปทรงกระบอก



รูปที่ 2.34 หุ่นยนต์ Cylindrical Robot

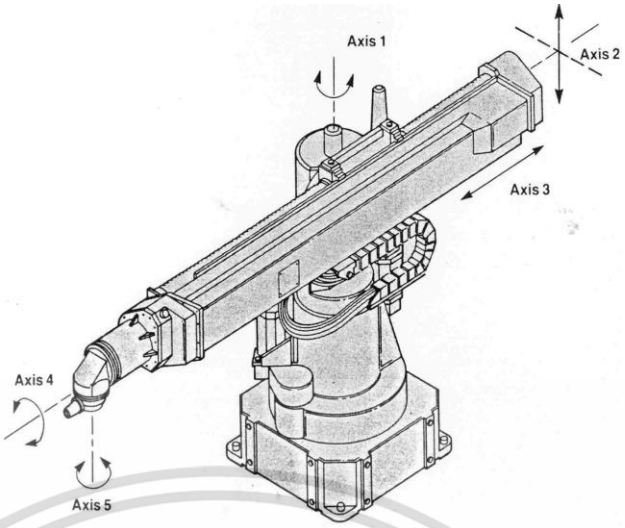
การประยุกต์ใช้งาน โดยทั่วไปจะใช้ในการหยิบยกชิ้นงาน (Pick-and-Place) หรือป้อนชิ้นงานเข้าเครื่องจักร เพราะ สามารถเคลื่อนที่เข้าออกบริเวณที่เป็นโพรงเล็กๆได้สะดวก

ข้อดี มีส่วนประกอบไม่ซับซ้อน การเคลื่อนที่สามารถเข้าใจง่าย สามารถเข้าถึงเครื่องจักรกลที่มีการเปิด- ปิด หรือเข้าไปในบริเวณที่เป็นช่องหรือโพรงได้ง่าย (Loading) เช่นการโหลดชิ้นงานเข้าเครื่อง CNC

ข้อเสีย มีพื้นที่ทำงานจำกัด แกนที่เป็นเชิงเส้นมีความยุ่งยากในการ seal เพื่อป้องกันฝุ่นและของเหลว

3. หุ่นยนต์ชนิดทรงกลม (Spherical Robot(Polar))

มีสองแกนที่เคลื่อนที่ในลักษณะการหมุน (Revolute Joint) คือแกนที่ 1 (เอว) และ แกนที่ 2 (ไหล่) ส่วนแกนที่ 3 (ข้อศอก) จะเป็นลักษณะของการเคลื่อนที่แนวเส้นตรง ดังรูปที่ 2.46



รูปที่ 2.35 หุ่นยนต์ Spherical Robot

การประยุกต์ใช้งาน ใช้ในงานที่มีการเคลื่อนที่ในแนวตั้ง (Vertical) เพียงเล็กน้อย เช่น การโหลดชิ้นงานเข้าออกจาก เครื่องปั้ม (Press) หรืออาจจะใช้งานเชื่อมจุด (Spot Welding)

ข้อดี มีปริมาตรการทำงานมากขึ้นเนื่องจากการหมุนของแกนที่ 2 (ไหล่) สามารถที่จะก้มลงมาจับชิ้นงานบนพื้นได้สะดวก

ข้อเสีย มีระบบพิกัด (Coorinate) และส่วนประกอบที่ซับซ้อน การเคลื่อนที่และระบบควบคุมมีความซับซ้อนขึ้น

4. หุ่นยนต์ชนิดสการา (SCARA Robot)

หุ่นยนต์ SCARA (Selective Compliance Assembly Robot Arm) จะมี ลักษณะแกนที่ 1 (เอว) และแกนที่ 3 (ข้อศอก) หมุนรอบแกนแนวตั้งและแกนที่ 2 จะเป็นลักษณะ การเคลื่อนที่ขึ้นลง (Prismatic) ดังรูปที่ 2.47 หุ่นยนต์SCARA จะเคลื่อนที่ได้รวดเร็วในแนวระนาบ และมีความแม่นยำสูง

รูปที่ 2.36 หุ่นยนต์ SCARA Robot

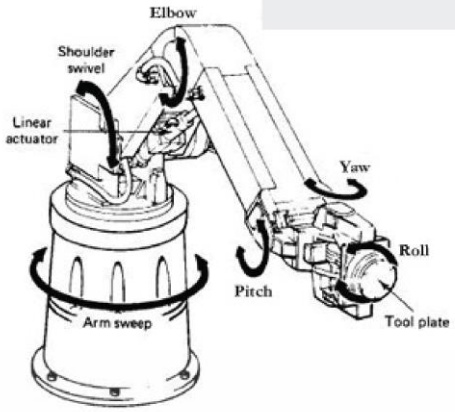
การประยุกต์ใช้งาน เนื่องจากการเคลื่อนที่ในแนวระนาบและขึ้นลงได้รวดเร็วจึงเหมาะกับงานประกอบชิ้นส่วนทาง อิเล็กทรอนิกส์ซึ่งต้องการความรวดเร็วและความเคลื่อนที่ก็ไม่ต้องการการหมุนมากนัก แต่จะไม่เหมาะกับงานประกอบชิ้นส่วนทางกล (Mechanical part) ซึ่งส่วนใหญ่การประกอบจะอาศัยการหมุนจะอาศัยการหมุน (rotation) ในลักษณะมุมต่าง ๆ นอกจากนี้ Scara robot ยังเหมาะกับงาน ตรวจสอบ (Inspection) งานบรรจุภัณฑ์ (Packaging)

ข้อดี สามารถเคลื่อนที่ในแนวระนาบและขึ้นลงได้รวดเร็ว มีความแม่นยำสูง

ข้อเสีย มีพื้นที่ทำงานจำกัด ไม่สามารถหมุน (rotation) ในลักษณะมุมต่างๆได้ สามารถยกน้ำหนัก (Payload) ได้มามากนัก

5. หุ่นยนต์ชนิดข้อต่อ (Articulated Arm (Revolute))

ทุกแกนการเคลื่อนที่จะเป็นแกนหมุน (Revolute) รูปแบบการเคลื่อนที่จะคล้ายกับ แขนคน ซึ่งจะประกอบด้วยช่วงเอว ท่อนแขนบน ท่อนแขนล่าง ข้อมือ การเคลื่อนที่ทำให้ได้พื้นที่การทำงาน



รูปที่ 2.37 หุ่นยนต์ Articulated Robot

การประยุกต์ใช้งาน หุ่นยนต์ชนิดนี้สามารถใช้งานได้กว้างขวางเพราะสามารถเข้าถึงตำแหน่งต่างๆได้ดี เช่นงานเชื่อม Spot Welding, Path Welding, งานยกของ, งานตัด, งานทากาว, งานที่มีการเคลื่อนที่ยาก ๆ เช่น งานพ่นสีงาน sealing ฯลฯ

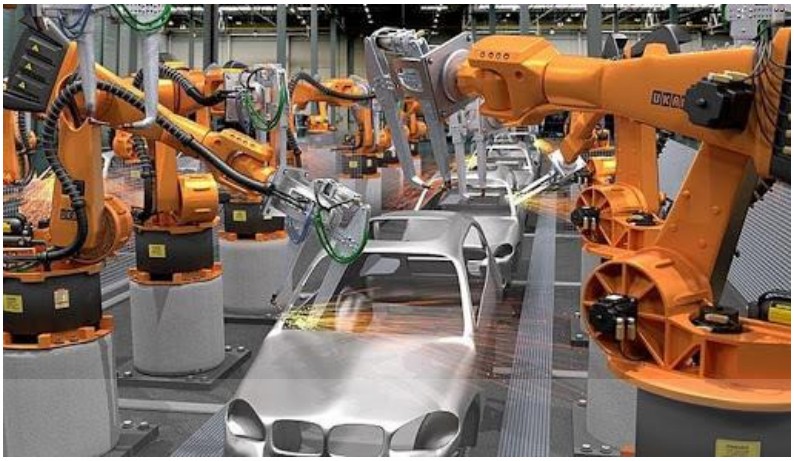
ข้อดี เนื่องจากทุกแกนจะเคลื่อนที่ในลักษณะของการหมุนทำให้มีความยืดหยุ่นสูงในการเข้าไปยัง จุดต่าง ๆ บริเวณข้อต่อ(Joint) สามารถ Seal เพื่อป้องกันฝุ่น ความชื้น หรือน้ำได้ง่าย มีพื้นที่การทำงานมาก สามารถเข้าถึงชิ้นงานทั้งจากด้านบน ด้านล่าง เหมาะกับการใช้มอเตอร์ไฟฟ้า เป็นชุดขับเคลื่อน

ข้อเสีย มีระบบพิกัด (Coordinate) ที่ซับซ้อน การเคลื่อนที่และระบบควบคุมทำความเข้าใจได้ยากขึ้น ควบคุมให้เคลื่อนที่ในแนวเส้นตรง (Linear) ได้ยาก โครงสร้างไม่มั่นคงตลอดช่วงการเคลื่อนที่เพราะบริเวณขอบ Work envelope ปลายแขนจะมีการสั่น

2.8 ประเภทของหุ่นยนต์ แบ่งตามลักษณะการใช้งาน

1. หุ่นยนต์ชนิดที่ติดตั้งอยู่กับที่ (Fixed robot)

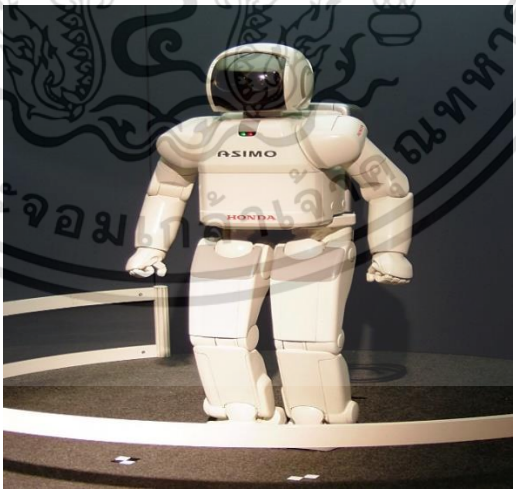
เป็นหุ่นยนต์ที่ไม่สามารถเคลื่อนที่ไปไหนได ด้วยตนเอง มีลักษณะเป็นแขนกล สามารถขยับและเคลื่อนที่ได้เฉพาะแต่ละข้อต่อ ภายในตัวเอง เท่านั้น มักนำมาใช้ในโรงงานอุตสาหกรรม เช่น โรงงานประกอบรถยนต์



รูปที่ 2.38 หุ่นยนต์ชนิดติดตั้งอยู่กับที่ Fixed Robot

2. หุ่นยนต์ชนิดเคลื่อนที่ได้ (mobile robot)

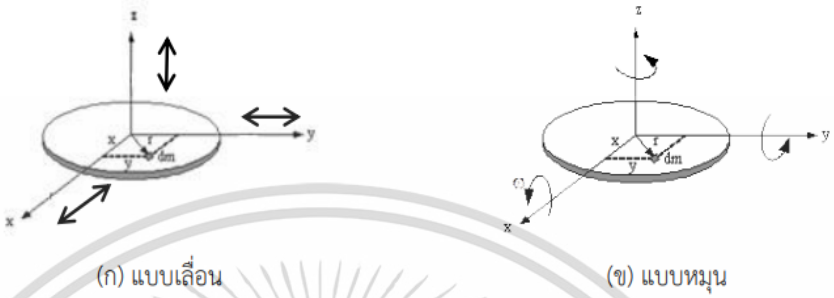
หุ่นยนต์ประเภทนี้จะแตกต่างจากหุ่นยนต์ที่ ติดตั้งอยู่กับที่เพราะสามารถเคลื่อนที่ไปไหนมาไหนได้ด้วยตนเอง โดยใช้ล้อหรือการใช้ขา ซึ่งหุ่นยนต์ ประเภทนี้ปัจจุบันยังเป็นงานวิจัยที่ทำการศึกษาอยู่ภายในห้องทดลอง เพื่อพัฒนาออกมาใช้งานใน รูปแบบต่าง ๆ เช่นหุ่นยนต์สำรวจดาวอังคาร ขององค์กรนาซ่าปัจจุบันมีการพัฒนาหุ่นยนต์ให้มี ลักษณะเป็นสัตว์เลี้ยงอย่างสุนัข เพื่อให้เป็นเพื่อนเล่นมนุษย์



รูปที่ 2.39 หุ่นยนต์ชนิดเคลื่อนที่ได้ Mobile Robot

2.9 ลำดับขั้นความเป็นอิสระและการบังคับ หรือ Degrees Of Freedom

การเคลื่อนที่ของวัสดุเกร็ง ส่วนมากจะเป็นการเคลื่อนที่แบบเลื่อน (Prismatic or Translational) แบบแกนหมุน (Rotational) รวมกัน การเคลื่อนที่แบบเลื่อนสามารถเลื่อนที่หมุนได้ใน 1 แกน 2 แกน หรือ 3 แกน เช่นเดียวกัน



(ก) แบบเลื่อน (ข) แบบหมุน

รูปที่ 2.40 ชนิดของการเคลื่อนที่

ลำดับขั้นความเป็นอิสระ คือจำนวนของส่วนประกอบการเคลื่อนที่ที่ต้องการควบคุมการเคลื่อนที่ จากรูปด้านบน วัตถุมีการเคลื่อนที่ทั้งแบบเลื่อน (ตามแกน X,Y และ Z) จึงมี 3 DOF และแกนหมุน (รอบแกน X,Y และ Z) จึงมี 3 DOF รวมกันก็จะมี 6 ลำดับขั้นความเป็นอิสระ (6 DOF) จำนวนของลำดับขั้นความเป็นอิสระ คือจำนวนส่วนประกอบของการเคลื่อนที่ เพื่อ ต้องการควบคุมการเคลื่อนที่ ในรูป (ก) รอยต่อ (Joint) ของชิ้นส่วนกลไกถูกบังคับให้เคลื่อนที่ แบบเคลื่อนที่เพียงอย่างเดียว จึงมี 1 DOF และรูป (ข) รอยต่อของชิ้นส่วนกลไกถูกบังคับให้เคลื่อนที่ แบบหมุนด้วย จึงมี 2 DOF

(ก) 1 DOF (ข) 2 DOF

รูปที่ 2.41 รอยต่อ 1 DOF และ 2 DOF

2.10 โซ่จลนศาสตร์

จลนศาสตร์คือ การศึกษาถึงการเคลื่อนที่ของก้านต่อโดยไม่สนใจแรงกระทำ ดังนั้นโซ่จลนศาสตร์คือ ส่วนประกอบของก้านต่อและรอยต่อ ซึ่งก้านต่ออินพุตจะควบคุมการเคลื่อนที่ของก้านต่อเอาต์พุต โดยอย่างน้อยต้องมีก้านต่อ 1 ก้านต่อยึดติดอยู่กับที่

1. คำจำกัดความของส่วนประกอบหุ่นยนต์

2. ก้านต่อ (Links) คือชิ้นส่วนของกลไกที่มีการเคลื่อนที่สัมพันธ์กับชิ้นส่วนอื่น ก้านต่อเป็นวัสดุแข็งเกร็ง (Rigid Body)

3. โนด (Nodes) คือจุดต่อระหว่างก้านต่อ

4. รอยต่อ (Joints) เป็นรอยต่อระหว่างก้านต่อ 2 ก้านหรือมากกว่าต่อกันที่โนด และเป็นสิ่งที่ซึ่งมีการเคลื่อนที่ระหว่างรอยต่อนั้น

5. ก้านต่อโยง (Linkages) เป็นกลไกที่ประกอบขึ้นจากชิ้นส่วนของก้านต่อที่ เชื่อมต่อกันที่โนดเกิดเป็นรอยต่อ เพื่อทำให้เกิดการเคลื่อนที่แบบลูกโซ่หรือแบบต่อเนื่อง

6. ตัวอย่างของก้านต่อ ได้แก่ คาน ข้อเหวี่ยง ก้านสูบ ก้านต่อ สไลเดอร์ พูลเลย์ สายพาน เพลา ฯลฯ

บทที่ 3
การดำเนินงาน

3.1 กล่าวนำ

ในบทนี้จะกล่าวถึง รายละเอียดและวิธีการดำเนินงานในการสร้างและออกแบบจำลองการเคลื่อนที่ของหุ่นยนต์โดยการใช้โปรแกรม RobotStudio และขั้นตอนในการสร้างฟังก์ชันบล็อกเพื่อประมวลผลชุดคำสั่ง RAPID

3.2 การออกแบบการเคลื่อนที่ของหุ่นยนต์โดยใช้โปรแกรม RobotStudio

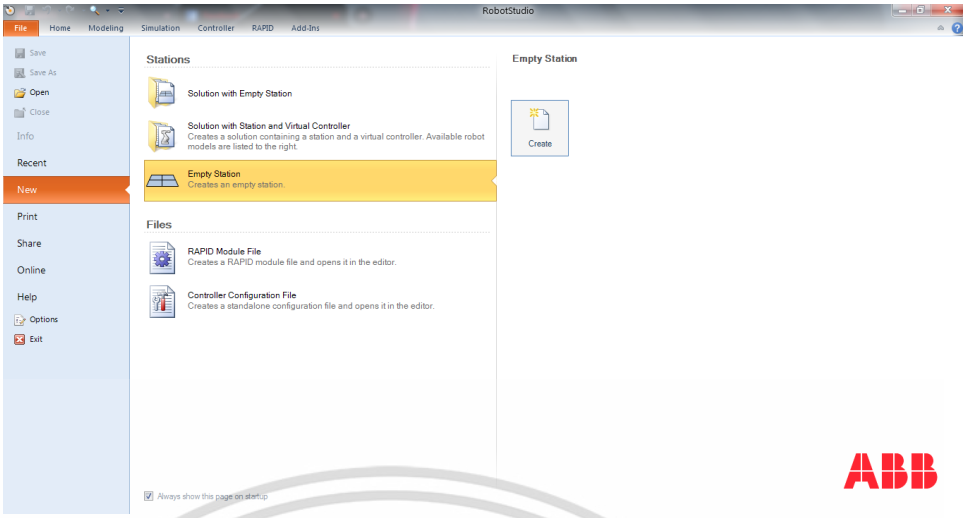
เมื่อต้องการทำให้หุ่นยนต์เคลื่อนที่ตามที่กำหนดต้องสร้างการจำลองเส้นทางการเดินให้หุ่นยนต์เดินตามที่สร้างไว้ เพื่อทำให้หุ่นยนต์ทำงานได้อย่างถูกต้องตามตำแหน่งที่ต้องการให้หุ่นยนต์เคลื่อนที่จะไป

โดยข้อมูลที่ได้จากการสร้างเส้นทางจำลองขึ้นมาจะมีด้วยกัน 5 อย่างซึ่งในแต่ละอย่างจะมีข้อมูลที่แตกต่างกันออกไปมีดังนี้

1. ตำแหน่งที่ต้องการจะเคลื่อนที่ไป ตำแหน่งเป็น Target ซึ่งใน Target จะบอก X,Y,Z
2. ความเร็ว
3. รัศมีการโค้งก่อนเราถึงจุดที่จะเคลื่อนที่ไป
4. อุปกรณ์ที่ใช้กับหุ่นยนต์ เช่น Gripper
5. ลักษณะการเคลื่อนที่

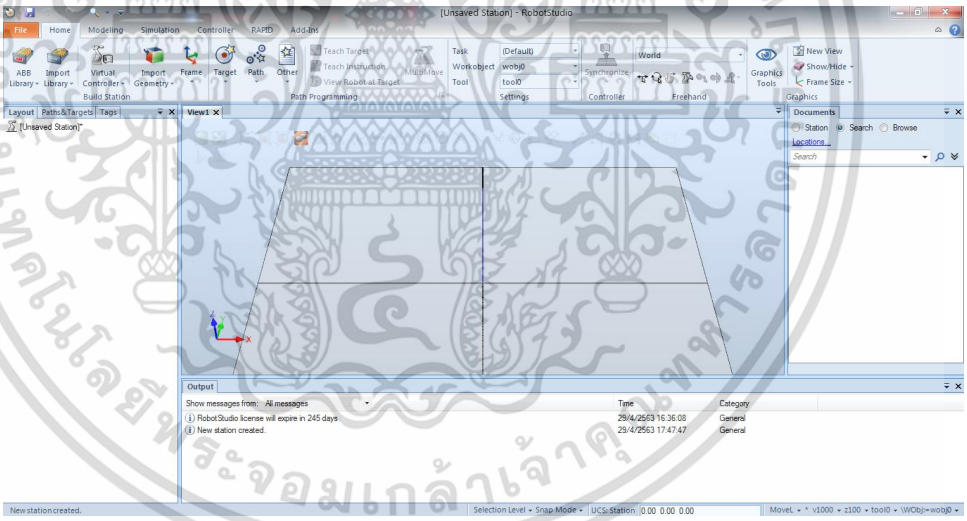
ก่อนที่จะเริ่มต้นเขียนส่วนส่วนประมวลผล RAPID เพื่อควบคุมการเคลื่อนที่ของแขนกลอันดับแรกคือต้องจำลองรูปแบบการเคลื่อนที่โดยใช้โปรแกรม RobotStudio ในการกำหนดจุดและรูปแบบการเคลื่อนที่รวมไปถึงการเลือกรุ่นและขนาดของหุ่นยนต์ โดยมีวิธีดังต่อไปนี้

1. สร้างโฟลเดอร์ โฟลเดอร์ที่สร้างขึ้นมาจะต้องสร้าง Station ขึ้นมาเพื่อสร้างรูปแบบการเคลื่อนที่ต่อไปดังรูปที่ 3.1



รูปที่ 3.1 การสร้างโฟลเดอร์

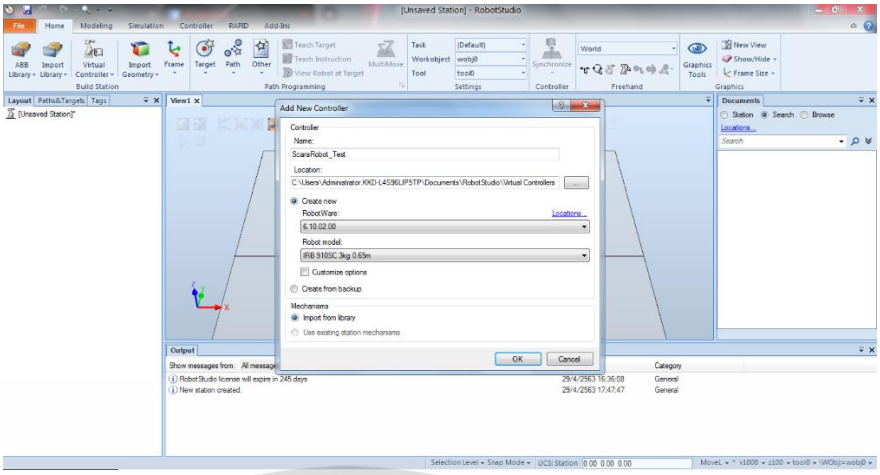
2. หลักจากกดสร้าง Station จะได้หน้าตาของ Station ดังรูปที่ 3.2



รูปที่ 3.2 หน้าของ Station

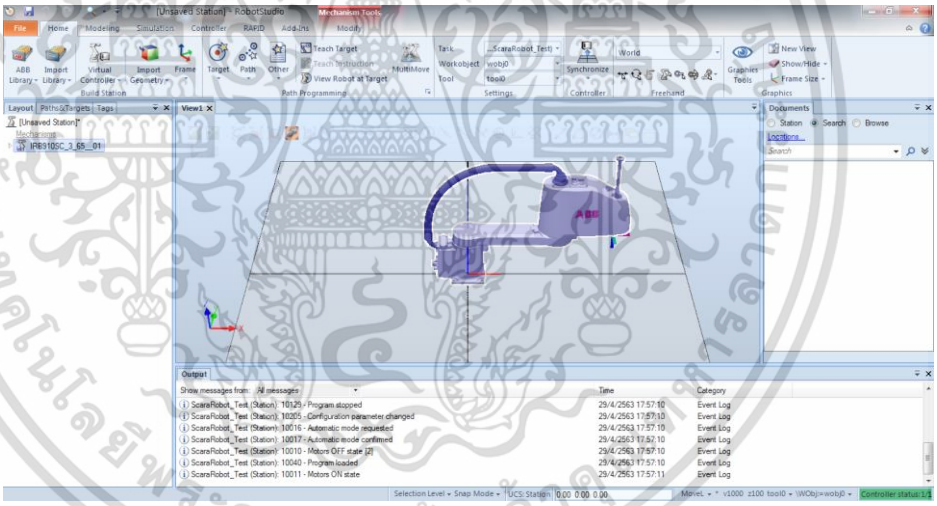
3. หลังจากสร้าง Station ให้กดไปที่ Virtual controller เพื่อทำการ Add New Controller กำหนดชื่อ และรุ่นของหุ่นยนต์ที่ต้องการใช้งานดังรูปที่ 3.3

โดยเลือก RobotWare : V.6.10.02.00 และเลือก Robot model : IRB 910SC 3kg 0.65m



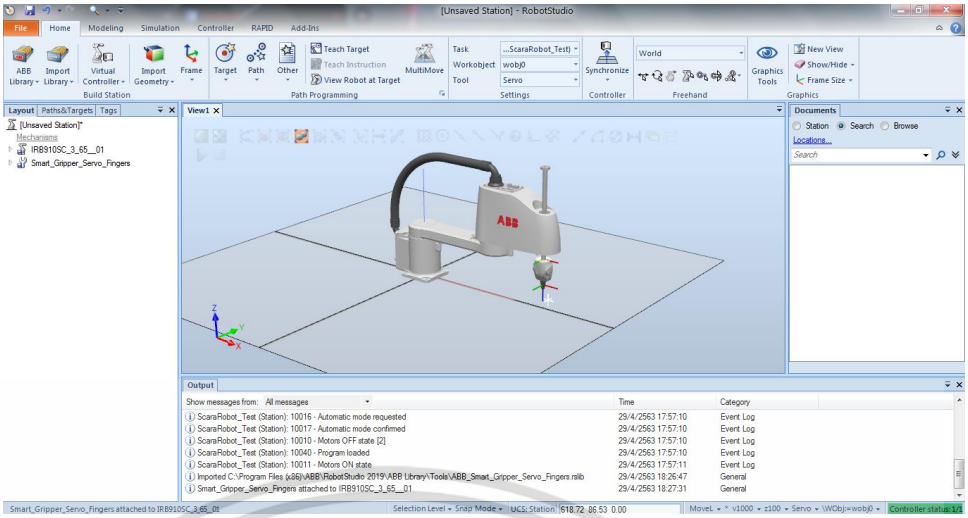
รูปที่ 3.3 การ Add New Controller

4. เมื่อกด Ok แล้วระบบจะทำการสร้าง Controller ดังรูปที่ 3.4



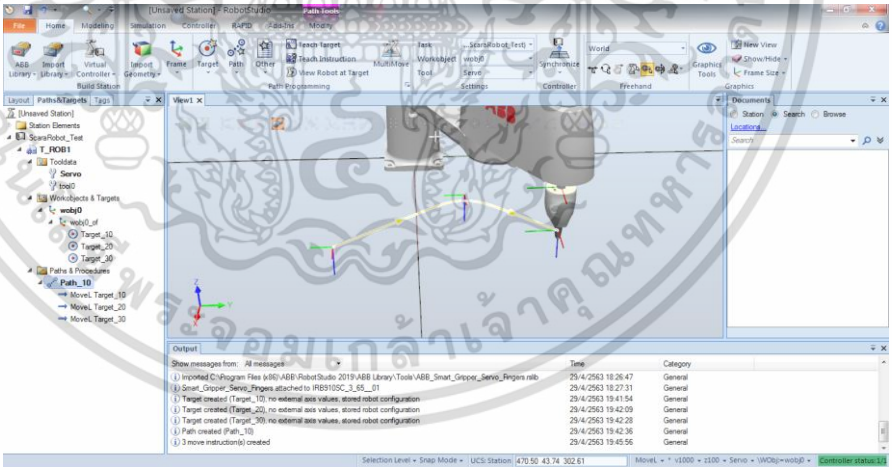
รูปที่ 3.4 การสร้าง Controller

5. ลำดับต่อไปเมื่อสร้าง แบบจำลองเสร็จแล้ว จะทำการติดตั้ง Tool ลงไป โดยเลือก Import Library เลือกไปที่ Equipment เลือกไปที่ ABB Smart Gripper เลือกรูปแบบการทำงานแบบ Servo,Fingers กด Ok



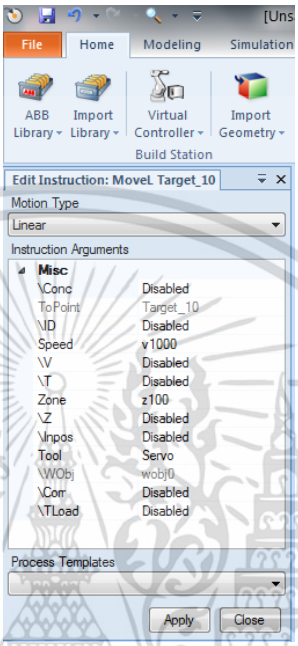
รูปที่ 3.5 การเลือก Gripper ที่ใช้งาน

6. หลังจากนั้นก็จะเป็นการกำหนดการเคลื่อนที่ Teach Target And Create Paths & Procedures



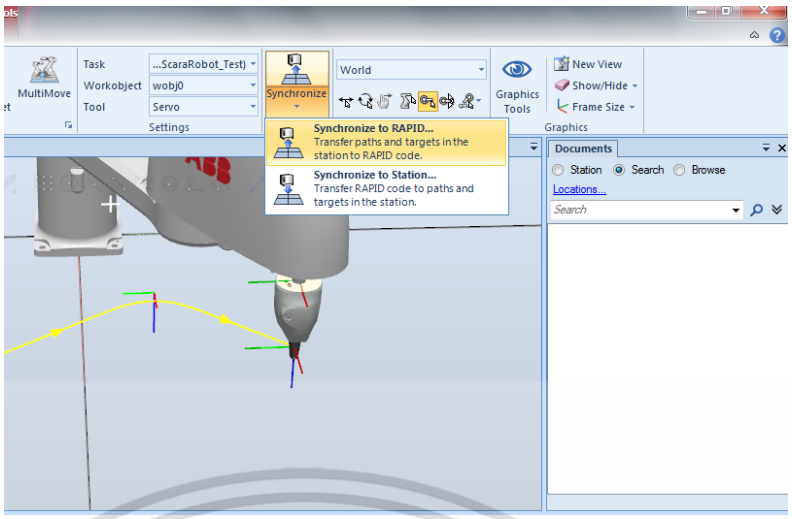
รูปที่ 3.6 การกำหนดการเคลื่อนที่

7. สามารถเลือกรูปแบบการเครื่องที่ว่าจะเป็นแบบไหนได้โดยการเลือกไปในส่วน Paths เลือก Move ที่ต้องการเปลี่ยนแล้วกดไปที่ Edit Instruction โดยสามารถเลือก MoveL, MoveC, MoveJ และสามารถเลือก Speed Zone หรือ Tool ได้ในนี้ด้วย



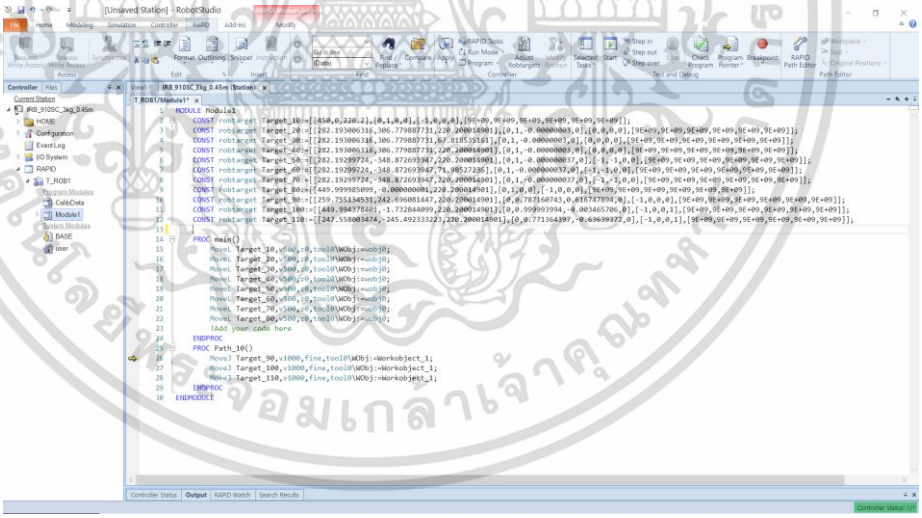
รูปที่ 3.7 การใส่ค่าที่ต้องการให้เครื่องทำงาน

8. หลังจากที่ได้กำหนดการเคลื่อนที่เรียบร้อยแล้วจะต้องทำการ Synchronize ข้อมูลไปเป็นข้อมูลภาษา RAPID โดยไปที่ Controller เลือก Synchronize เลือก Synchronize to RAPID



รูปที่ 3.8 การทำ Synchronize to RAPID

9.เมื่อกด Synchronize แล้วจะได้การเคลื่อนที่ในรูปของภาษา RAPID



รูปที่ 3.9 ตัวอย่างภาษา RAPID

3.3 การนำข้อมูล RAPID มาจัดการขนาดของข้อมูล

หลังจากที่เขียนการเคลื่อนที่โดยการใช้ RobotStudio แล้ว จะได้การเคลื่อนที่ในรูปแบบคำสั่ง RAPID ออกมา แต่ในพีแอลซีนั้นมีข้อจำกัดในด้านข้อมูลประเภท String ซึ่งสามารถรับข้อมูลได้เพียง 32 ตัวอักษรเท่านั้น จึงจำเป็นต้องใช้โปรแกรม Microsoft Excel ในการช่วยเพื่อจัดเรียงข้อมูลเป็นชุด ๆ ซึ่งข้อมูลแต่ละชุดนั้นกำหนดให้มีข้อมูลชุดละ 32 ตัวอักษร โดยใช้รูปแบบคำสั่งในการเขียนดังต่อไปนี้

```
=MID(text, start_num, num_chars)
```

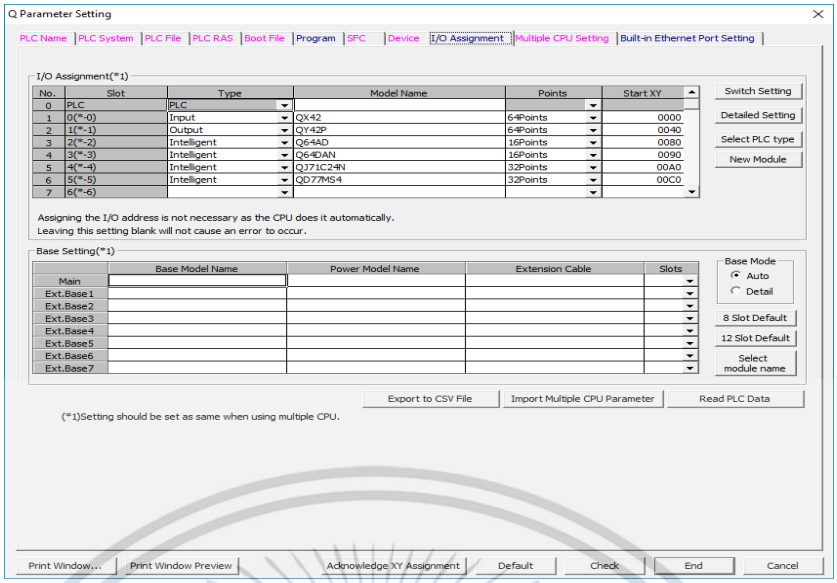
TEXT
MODULE Module1
CONST robtarget Target_30:=[[294.24332191,-273.691688556,220.200014901],[0,1,0.000000015,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_10:=[[450,0,220.2],[0,1,0,0],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_20:=[[294.243312829,203.405303065,220.200014901],[0,1,-0.00000003,0],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
PROC main()
ENDPROC
PROC Path_10()
MoveJ Target_30,v100,fine,tool0\WObj:=wobj0;
MoveJ Target_10,v100,fine,tool0\WObj:=wobj0;
MoveJ Target_20,v100,fine,tool0\WObj:=wobj0;
MoveJ Target_30,v100,fine,tool0\WObj:=wobj0;
ENDPROC
ENDMODULE

ข้อความที่แบ่งออกมาสูงสุดช่องละ 32 ตัว				
MODULE Module1				
CONST robtarget Target_30:=[[294	.24332191,-273.691688556,220.200	014901],[0,1,0.000000015,0],[-1,	-1,0,0],[9E+09,9E+09,9E+09,9E+09	,9E+09,9E+09]];
CONST robtarget Target_10:=[[450	,0,220.2],[0,1,0,0],[-1,0,0,0],[	9E+09,9E+09,9E+09,9E+09,9E+09,9E	+09]];	
CONST robtarget Target_20:=[[294	.243312829,203.405303065,220.200	014901],[0,1,-0.00000003,0],[-1,	0,0,0],[9E+09,9E+09,9E+09,9E+09,	9E+09,9E+09]];
PROC main()				
ENDPROC				
PROC Path_10()				
MoveJ Target_30,v100,fine,tool0\	WObj:=wobj0;			
MoveJ Target_10,v100,fine,tool0\	WObj:=wobj0;			
MoveJ Target_20,v100,fine,tool0\	WObj:=wobj0;			
MoveJ Target_30,v100,fine,tool0\	WObj:=wobj0;			
ENDPROC				
ENDMODULE				

รูปที่ 3.10 การจัดข้อมูลด้วยโปรแกรม Excel

3.4 การสร้างฟังก์ชันบล็อกเพื่อประมวลผล RAPID

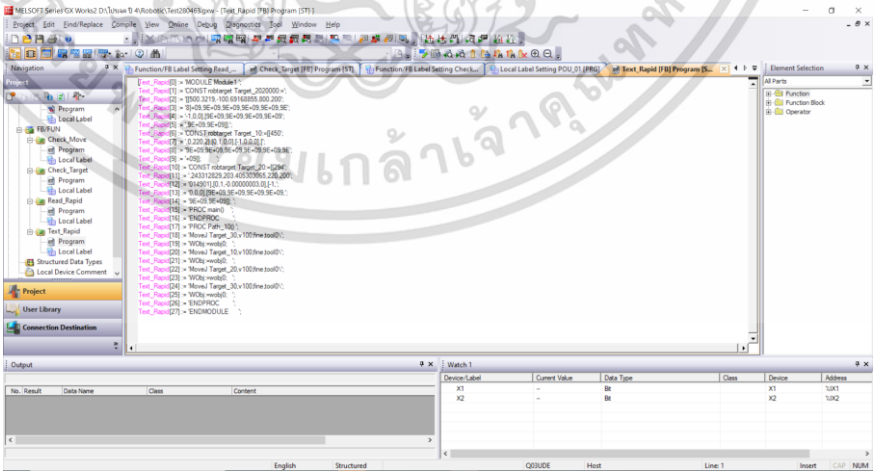
โมดูล หมายถึงอุปกรณ์หรือโมดูลต่างๆ ที่ไม่ได้มาพร้อมกับพีแอลซีแต่สามารถนำมาเสริมเพื่อเพิ่มความสามารถให้กับพีแอลซี ให้ทำงานได้หลากหลายมากขึ้น ดังนั้นเมื่อนำเอาอุปกรณ์ดังกล่าวมาต่อเพิ่มต้องมีการตั้งค่าทางซอฟต์แวร์ ให้สามารถเชื่อมต่อกันได้ โดยหมายถึงการตั้งค่า I/O Assignment ที่จะเป็นตัวกำหนด Address ของแต่ละโมดูลเพื่อเขียนโปรแกรมควบคุมการทำงาน



รูปที่ 3.11 การสร้าง I/O Assignment

จากรูปที่ 3.11 เป็นการตั้งค่า I/O Assignment ของอุปกรณ์ที่เชื่อมต่อกับพีแอลซีโดยจำเป็นต้องเรียงโมดูลตาม Slot ที่ต้องจัดเรียงตามฮาร์ดแวร์จริง

เมื่อทำการจัดการขนาดของข้อมูลเสร็จเรียบร้อยแล้ว จะนำข้อมูลส่วนที่ได้จัดขนาดไว้แล้วใส่เข้าไปในพีแอลซีโดยจะสร้างฟังก์ชันบล็อก ชื่อว่า Text_Rapid ซึ่งเป็นฟังก์ชันบล็อกที่เก็บข้อมูลประเภท String ซึ่งเก็บข้อมูลแบบ Array 1 มิติ



รูปที่ 3.12 ฟังก์ชันบล็อกที่เก็บข้อมูลประเภท String ซึ่งเก็บข้อมูลแบบ Array 1 มิติ

ซึ่งต่อมาเมื่อทำการบันทึกข้อมูลเสร็จเรียบร้อยแล้ว จะเป็นส่วนในการประมวลผลชุดคำสั่ง RAPID ซึ่งจะอธิบายในส่วนแรกก็คือส่วนที่ใน RAPID Instructions ซึ่งมีการประกาศตัวแปร ใน RAPID Instructions นั้นจะมีการประกาศโดยใช้คำว่า “CONST robtarget” โดยในพีแอลซีจะเขียนโปรแกรม Gx Work2 ให้มีการตรวจเช็คตัวอักษร “CONST robtarget” และให้มีการเก็บข้อมูลพิกัดต่างๆเอาไว้

ซึ่งกระบวนการทำงานของโปรแกรมคือจะทำการวนลูปตรวจหาคำว่า “CONST robtarget” เมื่อตรวจเจอแล้วตามรูปแบบของ RAPID Instructions จะได้ว่า

```
CONST robtarget Target_30:=[[294.24332191,-273.691688556,220.200014901],[0,1,0.000000015,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_10:=[[450,0,220.2],[0,1,0,0],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_20:=[[294.243312829,203.405303065,220.200014901],[0,1,-0.000000003,0],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

รูปที่ 3.13 ตัวอย่าง CONST robtarget

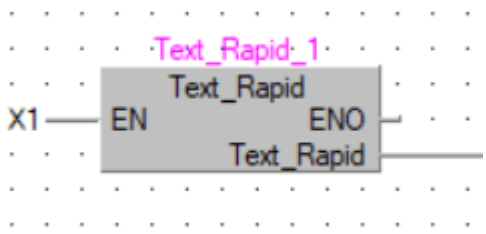
ซึ่งจะเห็นได้ว่าชื่อของตัวแปรที่ตามหลัง CONST robtarget นั้นจะมีเครื่องหมาย Colon หรือ “ : ” ตามหลังอยู่จะเขียนโปรแกรมออกมาเพื่อตรวจหา Colon เพื่อที่จะเก็บข้อมูลชื่อ CONST robtarget ได้

ส่วนต่อมาเมื่อเก็บค่าชื่อ Target ได้แล้วจำเป็นที่จะต้องเก็บค่าพิกัดต่าง ๆ ที่ภาษา RAPID ได้ประกาศออกมาแต่เนื่องจากได้รวบรวมปัจจัยต่างเพื่อนที่จะเก็บค่าพิกัด เพราะขีดจำกัดของพีแอลซี ที่ทำให้การเก็บข้อมูลประเภท String นั้นไม่สามารถทำได้อย่างต่อเนื่อง เนื่องจากต้องเก็บข้อมูลเป็นชุด ชุดละ 32 ตัวอักษรจึงทำให้การตรวจหาตัวอักษรต่าง ๆ นั้นทำได้ยาก จึงแยกออกเป็นหลาย ๆ กรณีเท่าที่จะมีความเป็นไปได้

ซึ่งในส่วนของการเก็บค่าพิกัด X, Y, Z นั้นได้เก็บค่าโดยใช้หลักการตรวจหาเครื่องหมาย Comma หรือ “ , ” ในการเก็บข้อมูลเป็นลำดับไปเรื่อย ๆ

ฟังก์ชันบล็อกที่เขียนขึ้นเพื่อจัดการกับข้อมูลมีดังต่อไปนี้

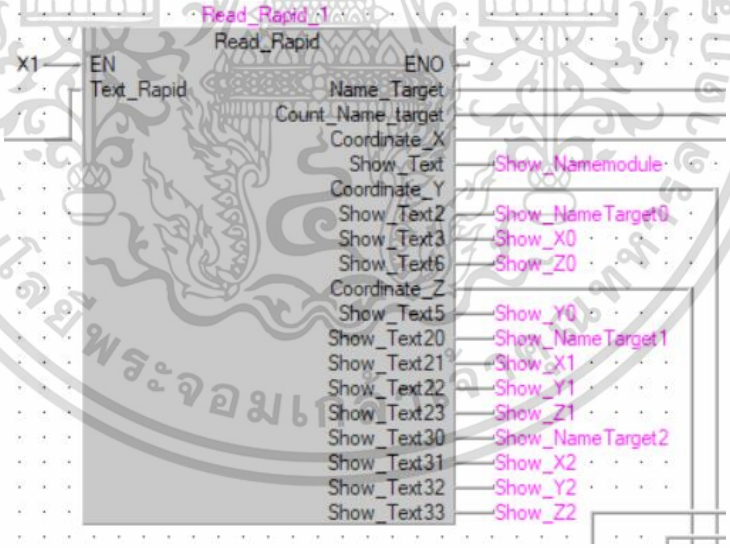
1. ส่วนแรกจะเป็นส่วนรับ RAPID Instructions มาใช้งานซึ่งมีการรับค่าของข้อมูลประเภท String Array 1 มิติ



รูปที่ 3.14 รูปฟังก์ชันบล็อกของ Text_Rapid

ในฟังก์ชันบล็อก นี้จะยังไม่มีการเขียนโปรแกรมสำหรับการทำงานมากนักมีหน้าที่จัดการข้อมูลเพื่อนำไปใช้กับฟังก์ชันบล็อก ในโปรแกรมการจัดการ

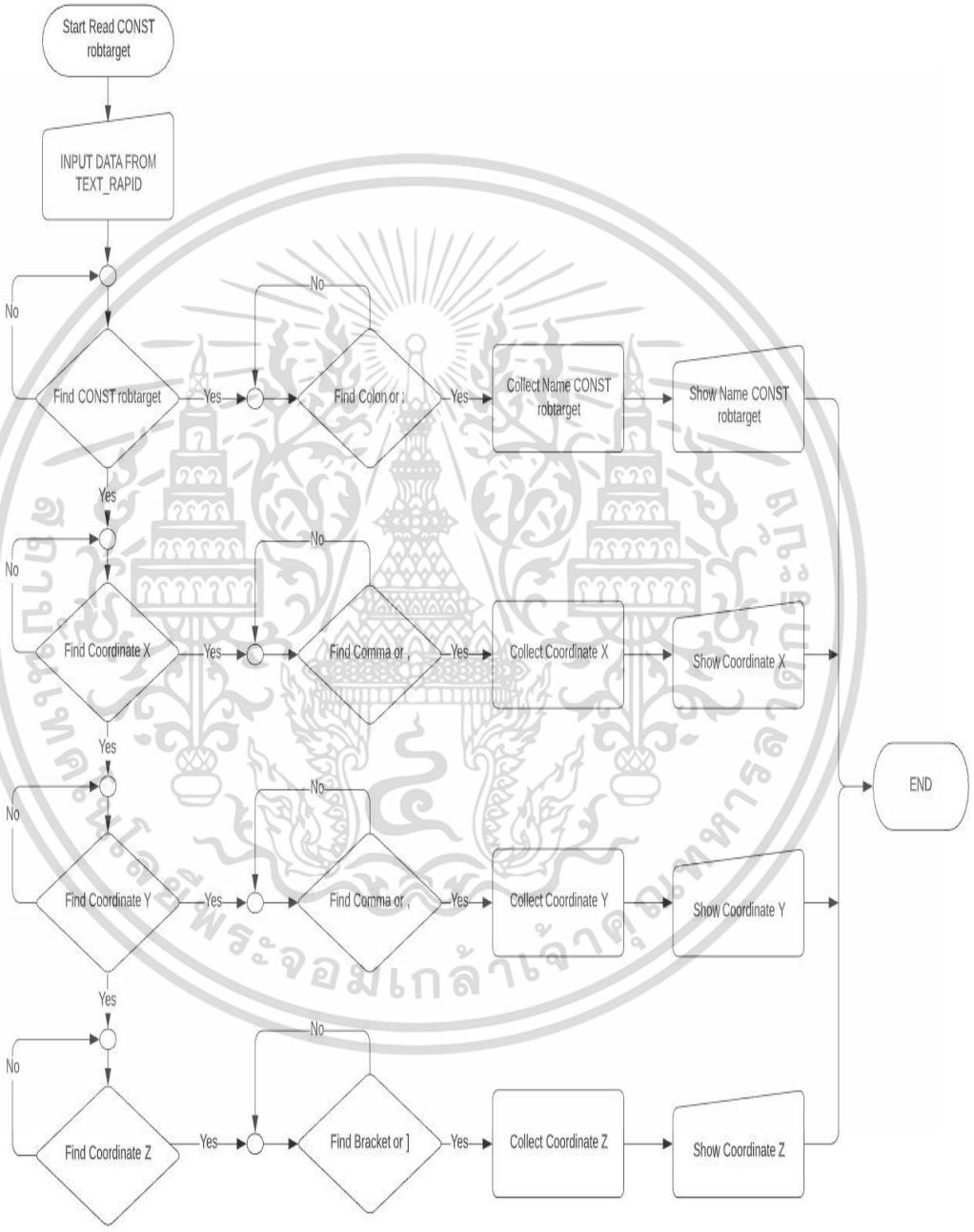
2. ในส่วนที่สองจะเป็นส่วนที่เกี่ยวกับการอ่านค่าตัวแปรที่ใน RAPID Instructions นั้นซึ่งใน RAPID Instructions ได้มีการประกาศตัวแปรโดยการใช้คำว่า “CONST robtarget” ซึ่งในฟังก์ชันบล็อก กล่องนี้จะทำหน้าที่จักเก็บข้อมูลพิกัดต่างๆและชื่อ Target



รูปที่ 3.15 รูปฟังก์ชันบล็อกของ Read_Rapid

ซึ่งโปรแกมในส่วนนี้จะเน้นความสำคัญไปที่ข้อมูลด้าน RAPID Instructions มากเป็นพิเศษ เนื่องจากข้อจำกัดของพีแอลซีที่ไม่สามารถเก็บข้อมูลประเภท String แบบชุดเดียวได้จึงจำเป็นต้อง

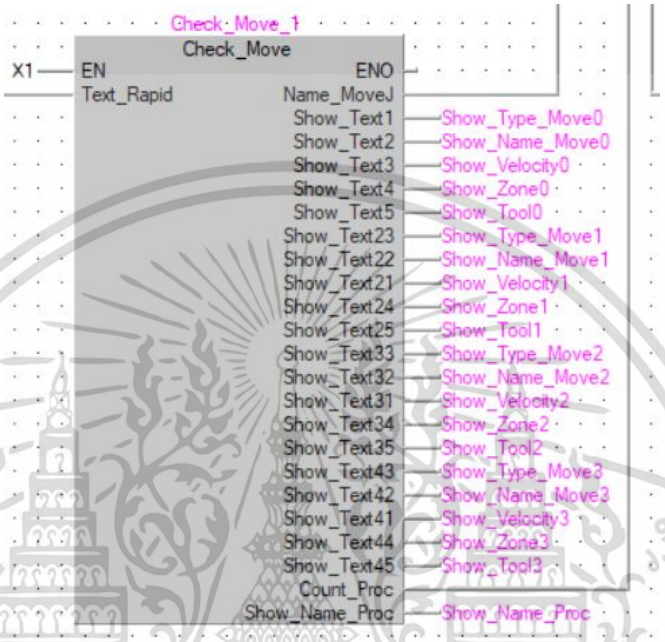
แบ่งข้อมูลออกเป็นชุด ชุดละ 32 ตัวอักษรเพราะฉะนั้นโปรแกรมที่ใช้ในการจัดเก็บข้อมูลจึงมีความจำเป็นที่จะต้องมีความซับซ้อมมากยิ่งขึ้นโดยการเขียนคำสั่งให้ครอบคลุมทุกกรณี ข้อความที่มีความยาวมากๆโปรแกรมก็ยังคงมีความสามารถในการอ่านได้



รูปที่ 3.16 Flowchart ของ Read_Rapid

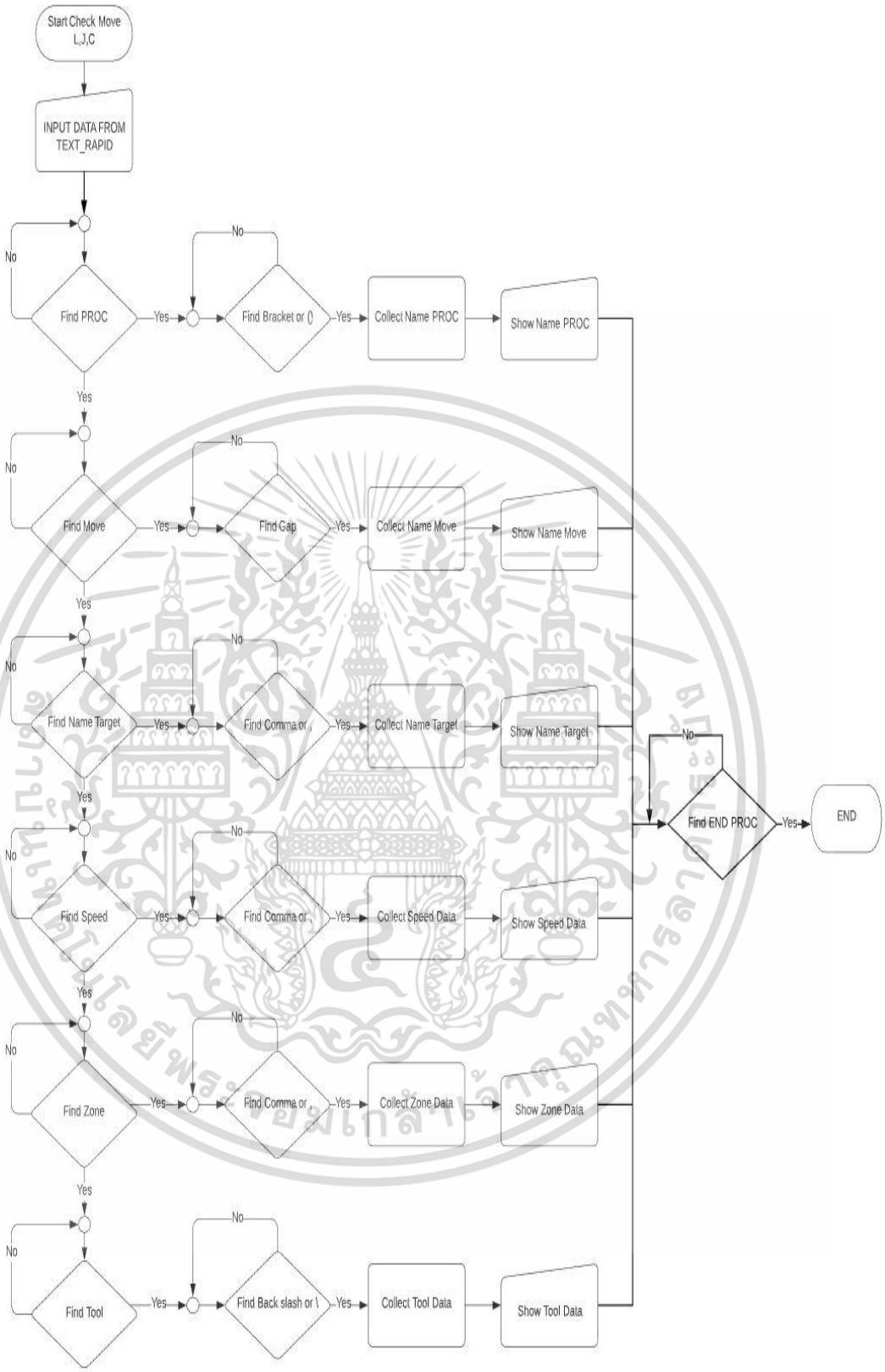
รูปที่ 3.16 แสดง Flowchart การทำงานของ Function box Read_Rapid

3. ในส่วนที่สามจะเป็นส่วนของฟังก์ชันบล็อก Check_Move ซึ่งเป็นส่วนสำคัญที่จะอ่าน RAPID Instructions ในส่วนของการเคลื่อนที่ ที่ได้ออกแบบเอาไว้แล้ว



รูปที่ 3.17 รูปฟังก์ชันบล็อกของ Check_Move

ซึ่งในส่วนนี้จะมีการทำงานจะมีรูปแบบที่คล้ายกับฟังก์ชันบล็อก Read_Rapid ในส่วนของกระบวนการจัดเก็บข้อมูลและปัญหาการจัดเก็บข้อมูลประเภท String ของพีแอลซี

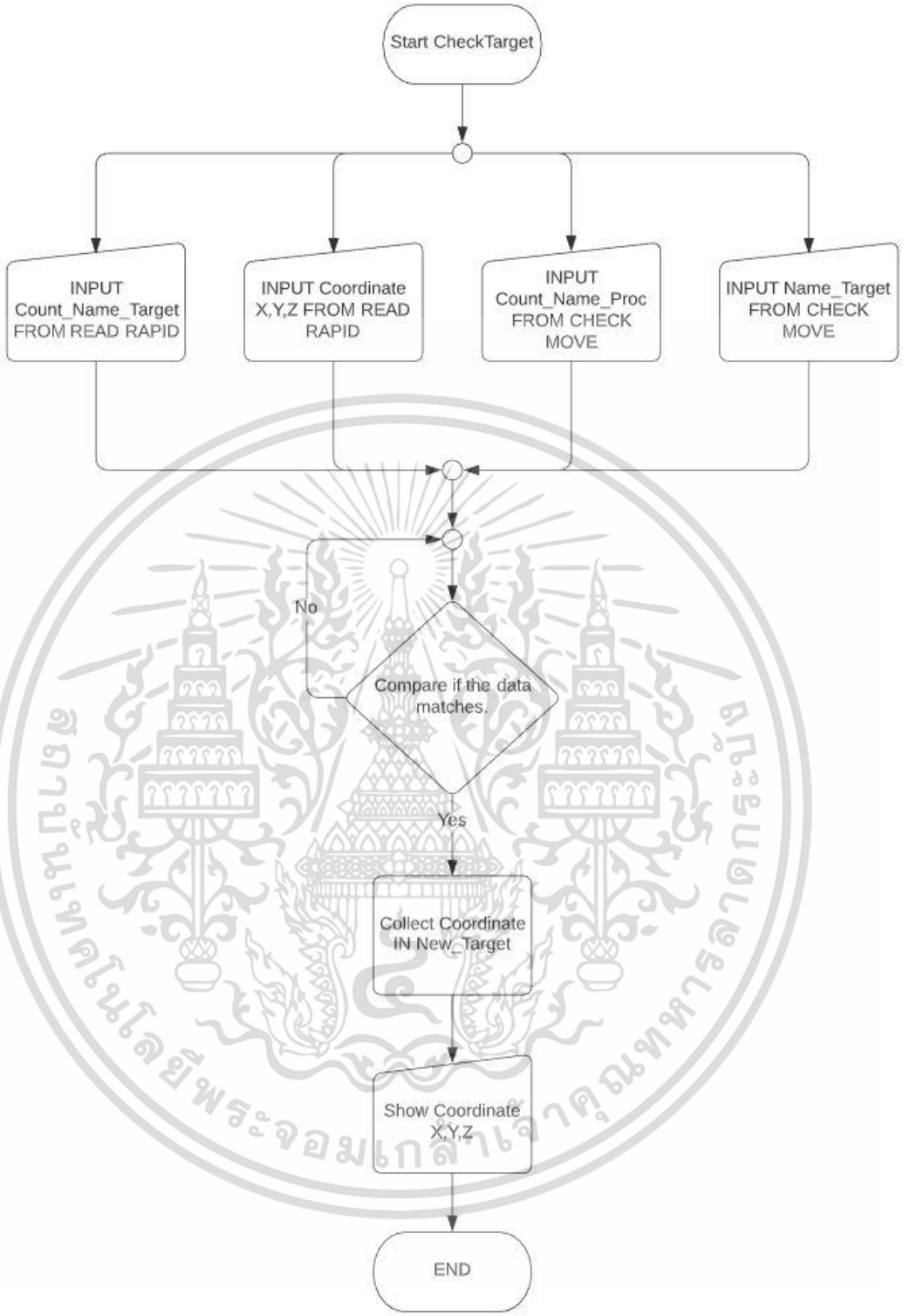


รูปที่ 3.18 Flowchart ของ Check_Move

รูปที่ 3.18 แสดง Flowchart แสดงการทำงานของฟังก์ชันบล็อก Check_Move ซึ่งจะมีลักษณะการทำงานคล้ายกับฟังก์ชันบล็อก Read_Rapid แต่จะมีส่วนสำคัญการจัดเก็บข้อมูล PROC ซึ่งจะเป็นตัวกำหนดการทำงานว่าโปรแกรมจะสามารถทำงานใน PROC ไหน

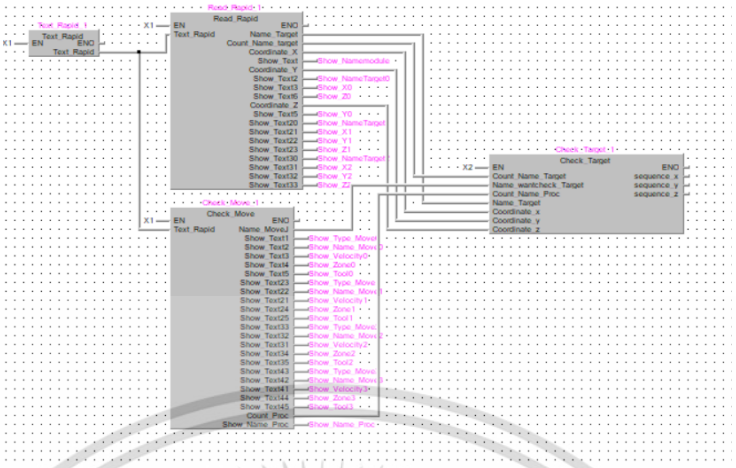
4.ในส่วนของโปรแกรมส่วนที่สี่จะเป็นส่วนของฟังก์ชันบล็อก Check_Target ซึ่งการทำงานของฟังก์ชันบล็อก กล่องนี้จะมีการทำงานผสานกับฟังก์ชันบล็อก Check_Move โดยการนำข้อมูลที่ได้จากฟังก์ชันบล็อก Check_Move ไปตรวจสอบกับข้อมูลจากฟังก์ชันบล็อก Read_Rapid ว่าตัว Target นั้นมีข้อมูลตรงกันหรือไม่ และเมื่อข้อมูลตรงกันฟังก์ชันบล็อก กล่องนี้ก็จะทำการจัดเรียงข้อมูลใหม่โดยโอนข้อมูลจากฟังก์ชันบล็อก Read_Rapid มาจัดลำดับพิกัดใหม่ให้สอดคล้องกับคำสั่ง Move ในฟังก์ชันบล็อก Check_Move

รูปที่ 3.19 ฟังก์ชันบล็อกของ Check_target



รูปที่ 3.20 Flowchart ของ Check_target

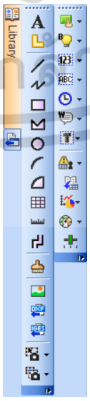
และเมื่อนำทุกส่วนมารวมกันโปรแกรมนี้จะมีลักษณะดังรูปที่ 3.21



รูปที่ 3.21 การทำงานของโปรแกรมทุกฟังก์ชันบล็อก

3.5 การสร้างหน้าจอเอชเอ็มไอ

ส่วนหน้าจอได้รับการเพิ่มมาจากหน้าจอเอชเอ็มไอเก่าที่มีอยู่แล้ว ซึ่งได้มีการเพิ่มส่วนที่เกี่ยวกับ RAPID ที่มีการบอกตำแหน่ง โดยถ้ากดทำงานจะเริ่มทำงานของโปรแกรมที่ถูกออกแบบเอาไว้ตามที่ออกแบบมาจากโปรแกรม RobotStuio ที่ไปประมวลผลออกมาเป็นภาษา RAPID เพื่อให้ได้ค่าพิกัดออกมาแสดงที่หน้าจอเพื่อ สะดวกต่อการตรวจสอบความถูกต้อง แถบเครื่องมือในการออกแบบใช้สำหรับการสร้างหน้าจอเอชเอ็มไอ

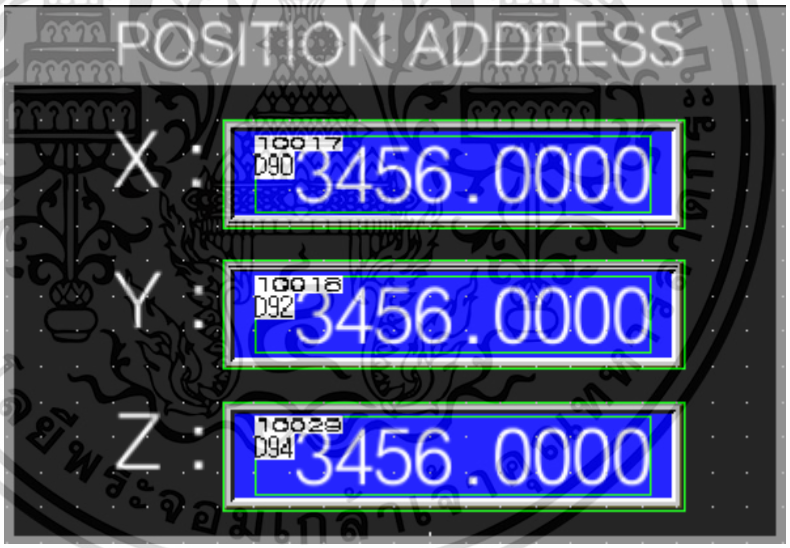


รูปที่ 3.22 แถบเครื่องมือใช้สร้างจอเอชเอ็มไอ

1. สร้างหน้า Base Screen ขึ้นมาโดยตั้งชื่อว่า RAPID เป็นหน้า Screen ของตัวฟังก์ชันบล็อกของโปรแกรม

รูปที่ 3.23 ปุ่มของ Base Screen หน้า RAPID

2. สร้าง Display สำหรับแสดงค่าของตำแหน่ง X, Y และ Z ที่อยู่ในตำแหน่งปัจจุบัน ค่าที่แสดงตำแหน่งของแกน X, Y และ Z ถูกแสดงเป็น Data type แบบ Real Display Range มีระยะอยู่ที่ -999 ถึง 999 แสดงจำนวนเต็มได้ 4 หลักและทศนิยม 4 ตำแหน่ง



รูปที่ 3.24 ค่าพิกัดปัจจุบันบนหน้า RAPID

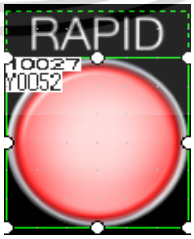
3. สร้างสวิตช์ start การทำงานของโปรแกรม เมื่อกดจะเริ่มทำงานโปรแกรม เป็นปุ่มทำงานแบบ Bit Momentary เมื่อกดทำงานแล้วจะทำงานจนกว่าจะกดสวิตช์ Stop การทำงานถึงจะหยุดการทำงานของโปรแกรม

รูปที่ 3.25 ปุ่มเริ่มกระบวนการทำงานของโปรแกรมบนหน้า RAPID

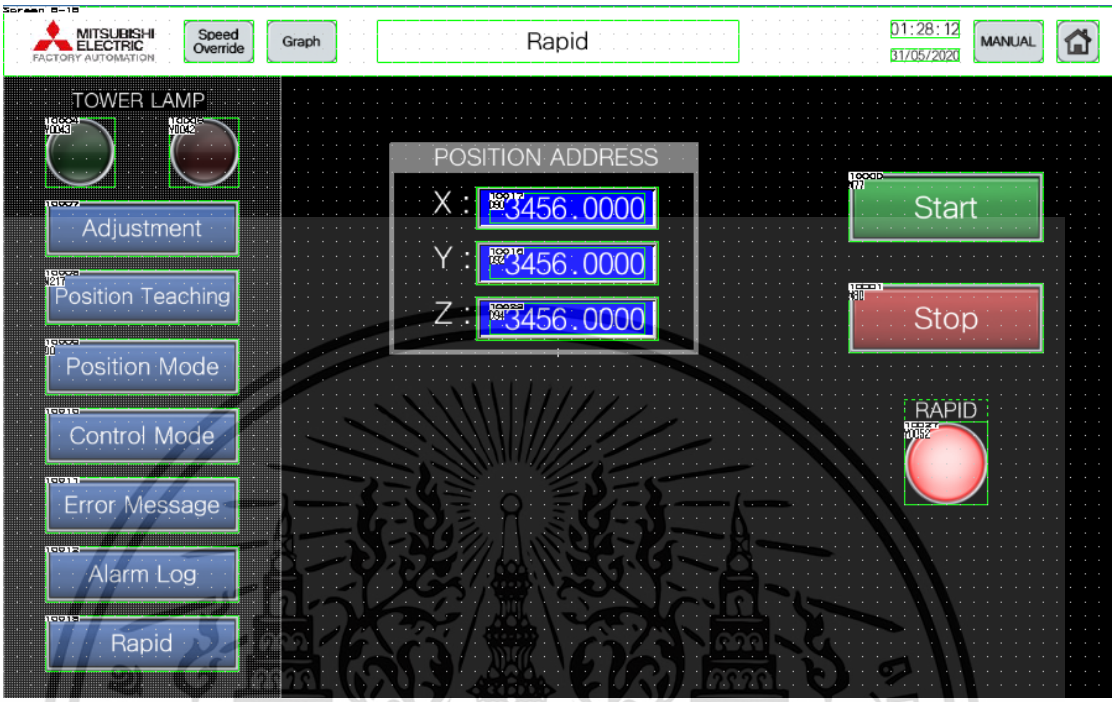
4. สวิตช์ stop การทำงานของโปรแกรม เมื่อกดจะหยุดการทำงานโปรแกรม เป็นปุ่มทำงานแบบ Bit Momentary การหยุดทำงานของโปรแกรมจะไม่สามารถเก็บค่า หากต้องการเก็บค่าใหม่ต้องทำการกดสวิตช์ Start เพื่อทำการเก็บค่า

รูปที่ 3.26 ปุ่มหยุดกระบวนการทำงานของโปรแกรมบนหน้า RAPID

5. สร้าง Lamp แสดงสถานะของการทำงาน เมื่อกดสวิตช์ Start Lamp จะแสดงสถานะสีเขียวซึ่งเป็นการแสดงว่าโปรแกรมกำลังทำงานแต่เมื่อ โปรแกรมถูกหยุดการทำงานหรือกดสวิตช์ Stop จะแสดงสถานะเป็นสีแดง



รูปที่ 3.27 หลอดไฟแสดงสถานะการทำงานของโปรแกรมบนหน้า RAPID



รูปที่ 3.28 หน้าจอเอชเอ็มไอ

ตารางที่ 3.1 รายละเอียดของหน้าจอเอชเอ็มไอ

Name	Device Tag	Description
X	D90	Numerical Display
Y	D92	Numerical Display
Z	D94	Numerical Display
Start	M77	Switch
Stop	M80	Switch
Rapid Lamp	Y0052	Bit Lamp

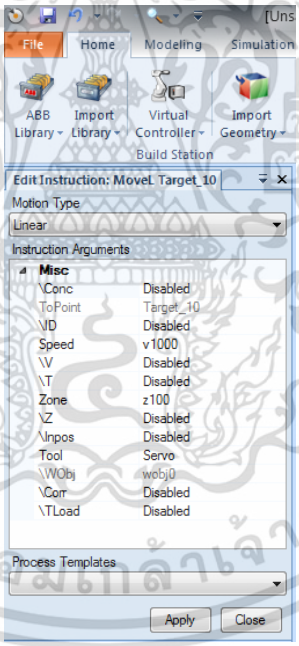
บทที่ 4
ผลการดำเนินงาน

4.1 กล่าวนำ

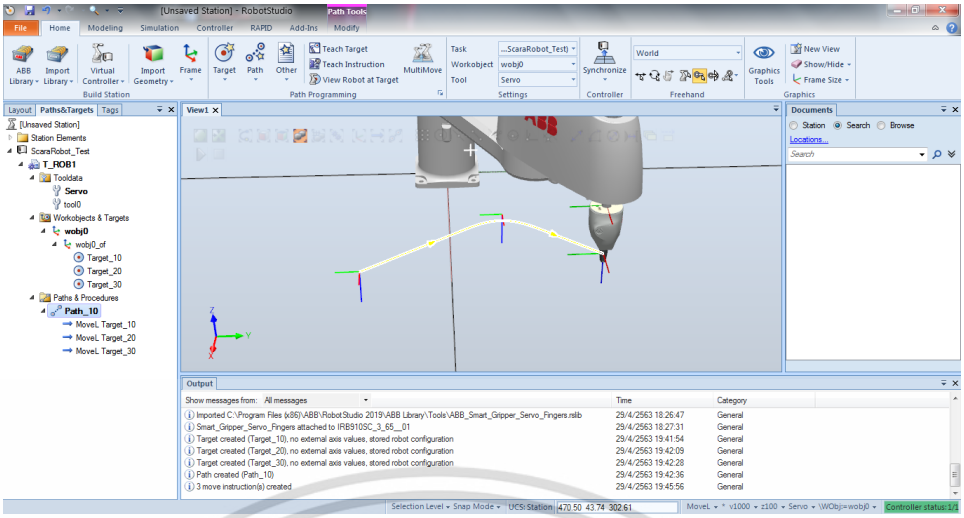
โดยบทนี้จะกล่าวถึงวิธีและผลทดสอบของการดำเนินงานทั้งหมด โดยมีผลของการทดสอบเพื่อตรวจงานว่า การทำงานโปรแกรมที่สร้างขึ้นใหม่สามารถทำงานได้ถูกต้องและครบถ้วน โดยการทดสอบจะแบ่งเป็น 3 ส่วน คือ ส่วนการออกแบบ RAPID ส่วนแปลงภาษาให้พีแอลซีและทดสอบสอบการทำงานของโปรแกรม

4.2 การทดสอบการกำหนดการเคลื่อนที่ของหุ่นยนต์ด้วยคำสั่ง Move

การทดสอบนี้เพื่อตรวจสอบว่าเส้นทางการเคลื่อนที่ถูกต้องหรือไม่ เส้นทางจะถูกจำลองสร้างขึ้นเพื่อนำไปใช้ในพีแอลซี โดยสามารถลองได้โดยการกรอกค่าลงไปในแต่ละค่า เช่น ความเร็ว โซนรัศมี สามารถกรอกค่าที่อยากทดลองลงไปโปรแกรมได้

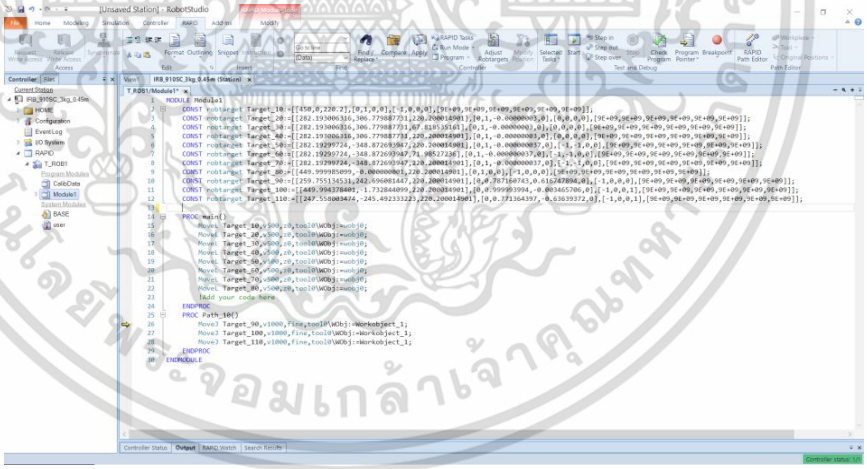


รูปที่ 4.1 การทดสอบใส่ค่าจำลองการทำงาน



รูปที่ 4.2 การทดสอบการจำลองเส้นทางการเดินของหุ่นยนต์

การทดสอบเส้นทางการเดินของหุ่นยนต์ สามารถทำได้ด้วยการจำลองแต่ละจุดที่ต้องการจะให้ไป โดยใช้ Target แต่ละ Target แบ่งเป็น Path ที่ต้องการทดสอบ แบ่งเป็นลำดับขั้นตอนการทำงานได้ โดยจะโดยทำการทดสอบดังรูปที่ 4.2



รูปที่ 4.3 ผลการทดสอบการทดลองเป็น RAPID

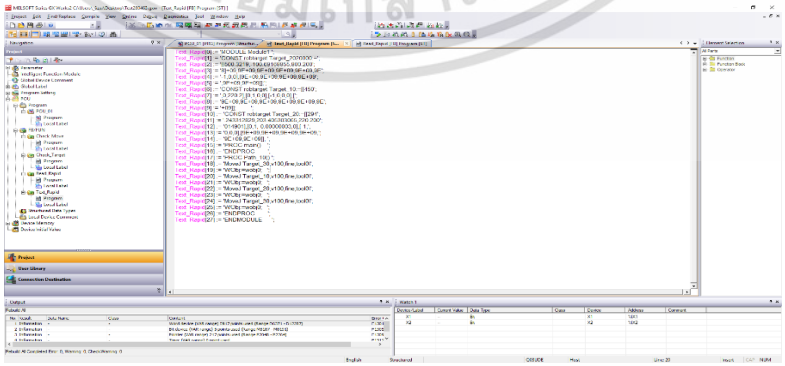
การทดสอบ RAPID ทำได้ด้วยการกด Synchronize แล้วเราจะได้การเคลื่อนที่ในรูปของ RAPID Instructions ต้องการทดสอบ โดยที่ภาษา RAPID ที่ได้สามารถนำไปใช้กับ Controller ตัวหุ่นยนต์ชนิดนั้นได้เลย แต่เราต้องการที่จะนำไปใช้ควบคุมโดยพีแอลซี จึงทำการประมวลผลชุดคำสั่ง RAPID เพื่อให้เข้าใจกัน

ตารางที่ 4.1 ผลทดสอบการกำหนดการเคลื่อนที่ของหุ่นยนต์

ผลทดสอบการกำหนดการเคลื่อนที่ของหุ่นยนต์			
รายละเอียดการทดสอบ	ผ่าน	ไม่ผ่าน	หมายเหตุ
1. ตรวจสอบเส้นทางการเคลื่อนที่ของหุ่นยนต์			
- การเคลื่อนที่ด้วยวิธีการ MoveL	✓		
- การเคลื่อนที่ด้วยวิธีการ MoveJ	✓		
- การเคลื่อนที่ด้วยวิธีการ MoveC	✓		
2. ตรวจสอบข้อมูลที่จำเป็นต่อการเคลื่อนที่ของหุ่นยนต์			
- ตรวจสอบพิกัดการเคลื่อนที่	✓		
- ตรวจสอบรูปแบบการเคลื่อนที่	✓		
- ตรวจสอบความเร็วที่ใช้ในการเคลื่อนที่	✓		
- ตรวจสอบ Zone ที่ใช้ในการเคลื่อนที่	✓		
- ตรวจสอบ Tool ที่จุดปลายของหุ่นยนต์	✓		
3. ตรวจสอบการ Synchronize to RAPID			
- ตรวจสอบการ Synchronize to RAPID	✓		

4.3 การทดสอบการการประมวลผลชุดคำสั่งภาษา RAPID

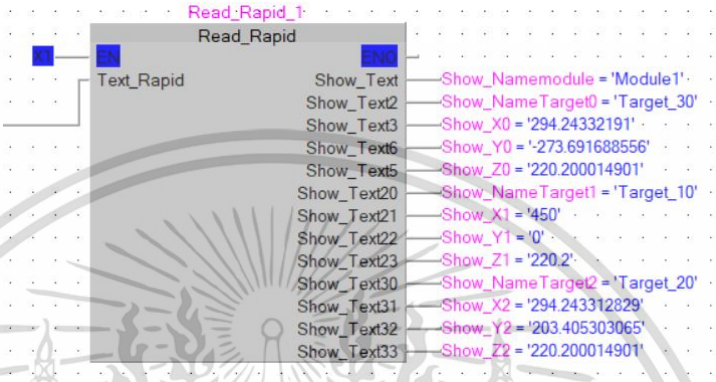
การทดสอบการแปลงภาษาจะทำหลังจากการทดสอบการสร้างการจำลองเส้นทางการเดินเป็น RAPID ทำการรับค่าเข้ามาจาก RAPID จากนั้นตรวจสอบความถูกต้องในโปรแกรม Microsoft Excel และใส่ข้อมูลไปยังพีแอลซี ดังรูปที่ 4.4 จากนั้นตรวจสอบว่าข้อความประเภท String ไม่เกิน 32 ตัวอักษร เพราะถ้าเกินที่กำหนด 32 ตัวอักษรจะขึ้นเตือนที่ Error code ว่า Too long string constant เพราะเป็นข้อจำกัดของการใช้งาน



รูปที่ 4.4 หน้าต่างฟังก์ชันบล็อกของ text_rapid

ทำการใส่ค่าจำลองเส้นทางการเดินที่กำหนดไว้ เพื่อให้เป็นไปตามเงื่อนไขที่สร้างไว้ และตรวจสอบว่าการ เมื่อการทดสอบมีความถูกต้องของข้อมูล จะทำการดึงข้อมูลใน text_rapid ที่ใส่ไว้ในฟังก์ชันบล็อก ดังรูปที่ 4.4 ทำการเปรียบเทียบข้อมูลได้

การเปรียบเทียบข้อมูลการประกาศค่าตัวแปรจากฟังก์ชันบล็อกที่ใช้ในการจัดเก็บข้อมูล

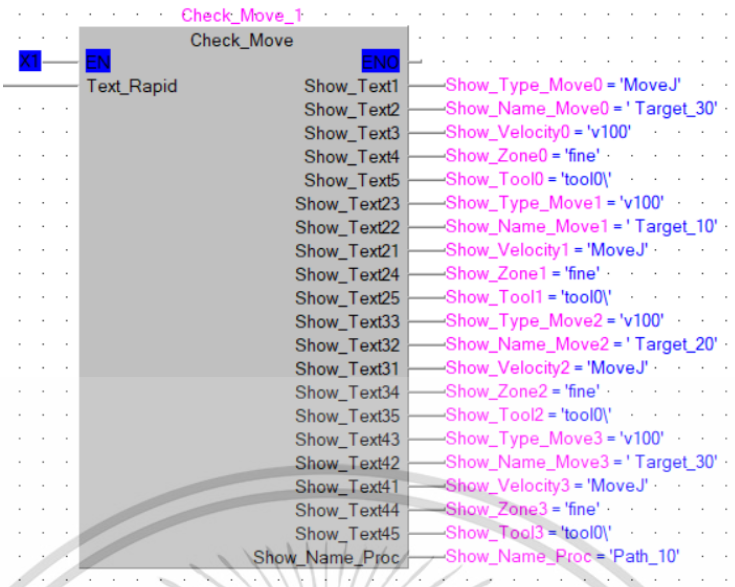


รูปที่ 4.5 ค่าที่แสดงจากการค่าข้อมูลจากฟังก์ชันบล็อก

```
MODULE Module1
CONST robtarget Target_30:=[[294.24332191,-273.691688556,220.200014901],[0,1,0.000000015,0],[-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_10:=[[450,0,220.2],[0,1,0,0],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_20:=[[294.243312829,203.405303065,220.200014901],[0,1,-0.00000003,0],[-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

รูปที่ 4.6 ข้อมูล RAPID ที่ใช้ในการตรวจสอบการประกาศตัวแปร

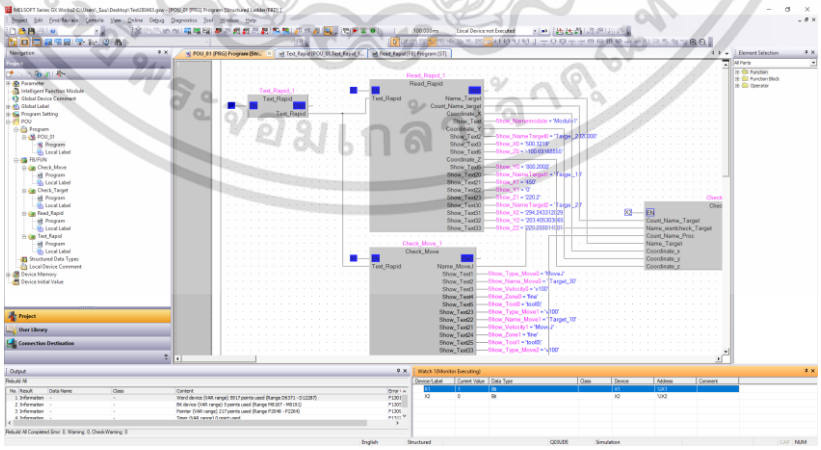
ต่อไปเป็นการตรวจสอบลำดับการเคลื่อนที่ให้มีความเป็นไปอย่างถูกต้องตามที่ได้รับการออกแบบไว้ ดังรูปที่ 4.6 จะเป็นการตรวจสอบข้อมูลที่ได้จากฟังก์ชันบล็อกจะอ่านค่ากระบวนการทำงานและการเคลื่อนที่ ซึ่งจะตรวจสอบว่าเป็นการเคลื่อนที่แบบใด ส่วนในรูปที่ 4.7 จะเป็นค่าที่ได้จากข้อมูล RAPID โดยตรง ส่วนในรูปที่ 4.8 รูปที่ 4.9 และ 4.10 จะเป็นส่วนอธิบายเพิ่มเติมในส่วนของการทำงานของโปรแกรม



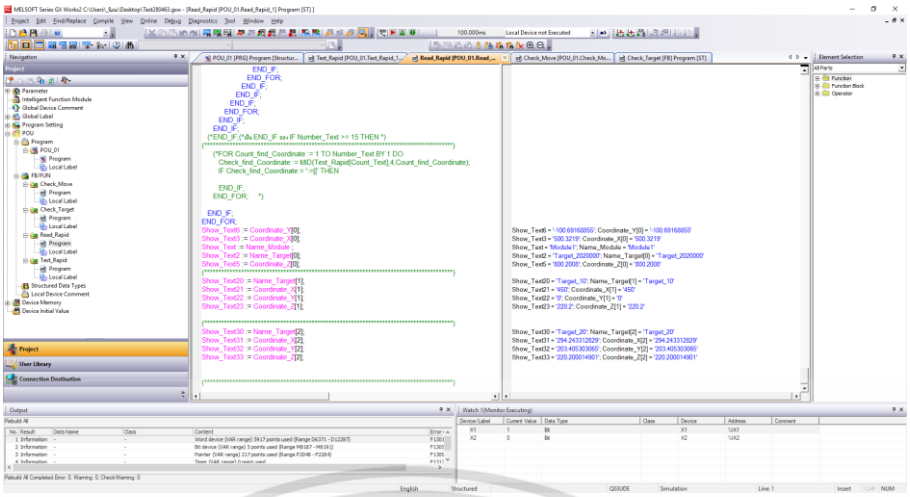
รูปที่ 4.7 ค่าที่ได้จากการประมวลผลข้อมูล RAPID จากกระบวนการเคลื่อนที่

PROC Path_10()	
MoveJ Target_30,v100,fine,tool0\	WObj:=wobj0;
MoveJ Target_10,v100,fine,tool0\	WObj:=wobj0;
MoveJ Target_20,v100,fine,tool0\	WObj:=wobj0;
MoveJ Target_30,v100,fine,tool0\	WObj:=wobj0;
ENDPROC	

รูปที่ 4.8 เป็นข้อมูลที่ได้จาก RAPID โดยตรง



รูปที่ 4.9 แสดงข้อมูลที่ได้มากจาก RAPID



รูปที่ 4.10 ค่าของ Target ตำแหน่ง X, Y, Z

TEXT
MODULE Module1
CONST robtarget Target_30:=[[294.24332191,-273.691688556,220.200014901],[0,1,0.000000015,0],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_10:=[[450,0,220.2],[0,1,0,0],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Target_20:=[[294.243312829,203.405303065,220.200014901],[0,1,-0.00000003,0],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
PROC main()
ENDPROC
PROC Path_10()
MoveJ Target_30,v100,fine,tool0\WObj:=wobj0;
MoveJ Target_10,v100,fine,tool0\WObj:=wobj0;
MoveJ Target_20,v100,fine,tool0\WObj:=wobj0;
MoveJ Target_30,v100,fine,tool0\WObj:=wobj0;
ENDPROC
ENDMODULE

รูปที่ 4.11 ข้อมูลที่ได้จาก RAPID โดยตรงที่ใช้ในการตรวจสอบ

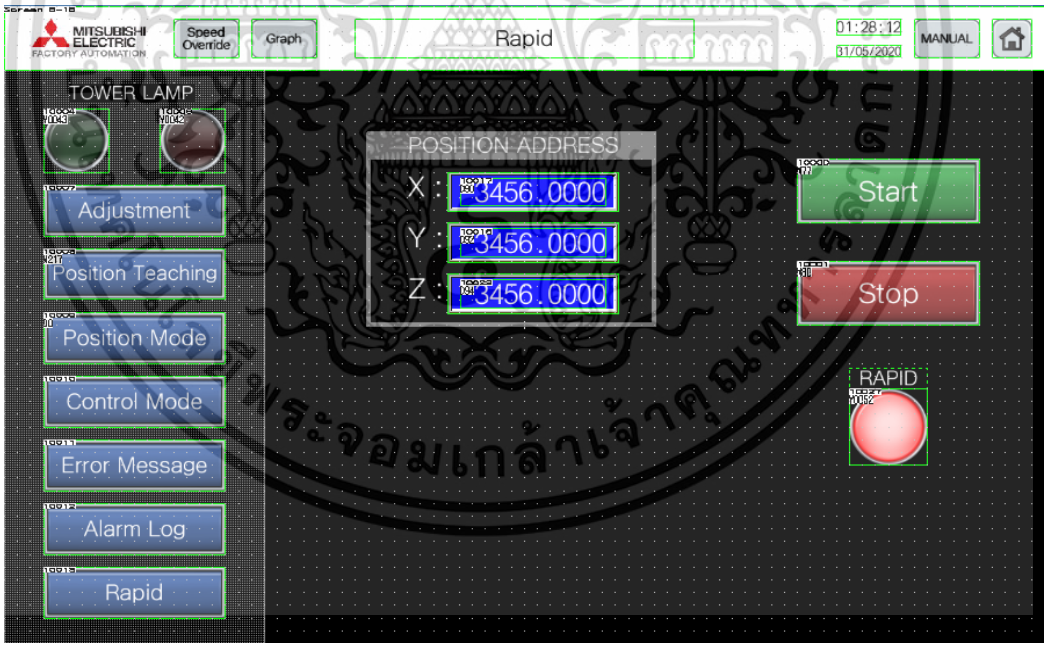
ตารางที่ 4.2 ผลทดสอบการประมวลผลชุดคำสั่ง RAPID

ผลทดสอบการประมวลผลชุดคำสั่ง RAPID			
รายละเอียดการทดสอบ	ผ่าน	ไม่ผ่าน	หมายเหตุ
1. ตรวจสอบข้อมูล RAPID (ก่อนการประมวลผล)			
- การจัดขนาดของข้อมูลเป็นขนาด 32 ตัวอักษร	✓		
- ข้อมูลมีความถูกต้องครบถ้วน	✓		
2. ตรวจสอบข้อมูล RAPID (หลังการประมวลผล)			
- ตรวจสอบพิกัดการเคลื่อนที่	✓		
- ตรวจสอบรูปแบบการเคลื่อนที่	✓		

- ตรวจสอบความเร็วที่ใช้ในการเคลื่อนที่ - ตรวจสอบ Zone ที่ใช้ในการเคลื่อนที่ - ตรวจสอบ Tool ที่จุดปลายของหุ่นยนต์	✓ ✓ ✓		
3. ตรวจสอบการจัดเก็บข้อมูลการเคลื่อนที่ใน Array - ตรวจสอบการจัดเก็บข้อมูลพิกัด X, Y และ Z - ตรวจสอบการจัดเก็บข้อมูลรูปแบบการเคลื่อนที่ - ตรวจสอบการจัดเก็บข้อมูลความเร็ว - ตรวจสอบการจัดเก็บข้อมูล Zone - ตรวจสอบการจัดเก็บข้อมูล Tool	 ✓ ✓ ✓ ✓ ✓		

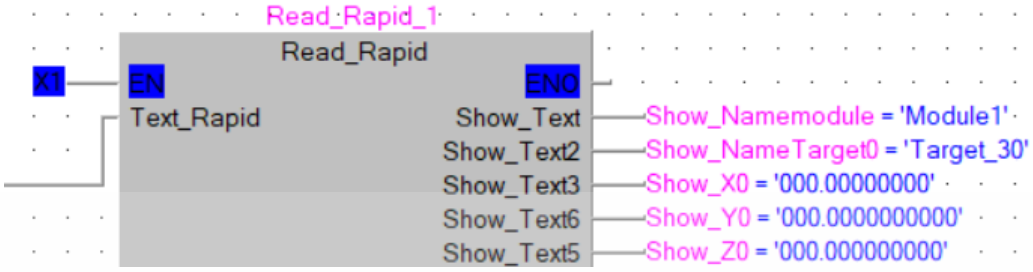
4.4 การทดสอบด้วยการสั่งการด้วยหน้าจอเอชเอ็มไอ

การทดสอบนี้เพื่อตรวจสอบฟังก์ชันและลำดับการทำงานของโปรแกรม โดยการจำลองการเคลื่อนที่ให้ตรงกับการทำงานของตามที่ได้กำหนดไว้ จากนั้นตรวจสอบเส้นทางการเคลื่อนที่



รูปที่ 4.12 การทดสอบการทำงานเอชเอ็มไอของ RAPID

ทดสอบการทำงานของโปรแกรม โดยการใช้หน้ากราฟิกใหม่ที่สร้างขึ้น เพื่อใช้ทดสอบการทำงานของโปรแกรม ดังรูปที่ 4.9 เป็นรูปที่แสดงค่าข้อมูลที่ได้จากการประมวลผลข้อมูล RAPID และรูปที่ 4.10 คือค่าข้อมูล RAPID โดยตรง



รูปที่ 4.13 ค่าข้อมูลที่แสดงจากฟังก์ชันบล็อกหลังจากการประมวลผล

```
Text_Rapid[0] := 'MODULE Module1';
Text_Rapid[1] := 'CONST robtarget Target_30:=[[000';
Text_Rapid[2] := ',00000000,000,0000000000,000,000';
Text_Rapid[3] := '000000],[0,1,0,000000015,0],[-1,';
Text_Rapid[4] := '-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09';
Text_Rapid[5] := ',9E+09,9E+09]];';
Text_Rapid[6] := 'CONST robtarget Target_10:=[[450';
Text_Rapid[7] := ',0,220,2],[0,1,0,0],[-1,0,0,0],[';
Text_Rapid[8] := '9E+09,9E+09,9E+09,9E+09,9E+09,9E+09,9E';
Text_Rapid[9] := '+09]];      ';
```

รูปที่ 4.14 ค่าข้อมูลที่ได้จาก RAPID โดยตรง

ตารางที่ 4.3 ผลทดสอบการสั่งงานด้วยหน้าจอเอชเอ็มไอ

ผลทดสอบการสั่งงานด้วยหน้าจอเอชเอ็มไอ			
รายละเอียดการทดสอบ	ผ่าน	ไม่ผ่าน	หมายเหตุ
1. ตรวจสอบการทำงานของโปรแกรม - เปิดการทำงานในโหมด RAPID - เมื่อกดปุ่ม Start - เมื่อกดปุ่ม Stop - สถานะการทำงานของหลอดไฟแสดงสถานะ	✓ ✓		
2. ตรวจสอบข้อมูลการเคลื่อนที่ณะพิกัดปัจจุบัน - พิกัดการเคลื่อนที่แกน X - พิกัดการเคลื่อนที่แกน Y - พิกัดการเคลื่อนที่แกน Z	✓ ✓ ✓		

บทที่ 5
สรุปผล ปัญหา และข้อเสนอแนะ

5.1 สรุปผลการดำเนินงาน

เนื่องจากทางผู้จัดทำต้องการประวลผลชุดคำสั่ง RAPID ให้พีแอลซีสามารถทำงานได้โดยทำการออกแบรูปแบบการเคลื่อนที่ของหุ่นยนต์อุตสาหกรรม จึงต้องทำการสร้างส่วนประมวลผลชุดคำสั่ง อีกทั้งยังสร้างหน้าจอเอชเอ็มไอที่ใช้ทดสอบการทำงานของส่วนประมวลผล เพื่อเตรียมส่งค่าข้อมูลที่ได้จากการประมวลผลไปควบคุมหุ่นยนต์ต่อไป อีกด้วย

โครงการนี้ได้ทำการสร้างส่วนประมวลผลชุดคำสั่ง RAPID โดยได้ทดสอบฟังก์ชันต่าง ๆ ด้วยการจำลองการทำงาน MELSOFT Gx Work2 (Simulator) โดยหลังจากนี้สามารถนำส่วนงานเหล่านี้ไปทดสอบ และปรับปรุงให้เป็นไปตามความต้องการของผู้ปฏิบัติงาน

5.2 ปัญหา และวิธีการแก้ไข

5.2.1 ปัญหาที่พบ

ในขั้นตอนการเขียนรูปแบบการเคลื่อนที่ของคำสั่ง ได้มีคำสั่งองค์ประกอบต่าง ๆ ที่ไม่สามารถหาข้อมูลได้จึงทำให้ไม่สามารถทราบข้อมูลเหล่านั้นได้

ปัญหาข้อจำกัดในการใช้งานของโปรแกรม MELSOFT GX work2 ที่ไม่สามารถใช้ข้อมูลประเภท String ที่ไม่สามารถใช้ได้เกิน 32 ตัวอักษร

5.2.2 วิธีการแก้ไข

จากปัญหาเป็นเรื่องรูปแบบการเคลื่อนที่ของคำสั่ง จึงแก้ไขโดยการสอบถามจากอาจารย์และหาข้อมูลจากแหล่งต่างๆ

ทำให้ต้องนำโปรแกรม Microsoft Excel เข้ามาช่วยในการทำงาน

5.3 ข้อเสนอแนะ

ควรมีขั้นตอนและข้อตกลงที่แน่นอนในการทำงาน เพื่อที่จะได้ดำเนินงานตามแผนงานที่กำหนดไว้และ ควรมีพื้นฐานในการเขียนโปรแกรมเช่น ภาษา C หรือภาษา BASIC เพราะจะเป็นสิ่งจำเป็นอย่างมาก ซึ่งจะช่วยทำให้ในการทำงานสะดวกมากยิ่งขึ้นและมีประสิทธิภาพ

เอกสารอ้างอิง

[1] หุ่นยนต์ในอุตสาหกรรม [ออนไลน์]. เข้าถึงเมื่อ 15 พฤศจิกายน 2562,จาก: https://www.applicadthai.com/articles/ industrial-robot-type/

[2] RoBotStudio [ออฟไลน์]. เข้าถึงเมื่อ 8 กุมภาพันธ์ 2563,จาก: file:///D:/โปรเจค%20ปี%204/3HAC032104%20OM%20RobotStudio-en.pdf

[3] Introduction Rapid [ออนไลน์]. เข้าถึงเมื่อ 30 มกราคม 2563,จาก: https://library.e.abb.com/public/Technicalreferencemanual_RAPID_3HAC16581-1_revJ_en.pdf

[4] MELSECQ Series [ออนไลน์]. เข้าถึงเมื่อ 15 มกราคม 2563,จาก: https://www.bpx.co.uk/dbdocument/128185/Q%20Datasheet.pdf

[5] พีแอลซีพื้นฐาน [ออนไลน์]. เข้าถึงเมื่อ 28 พฤศจิกายน 2562,จาก: https://thaiautomation.blogspot.com/2017/06/plc-plc.html

[6] ส่วนประกอบพีแอลซี [ออนไลน์]. เข้าถึงเมื่อ 30 พฤศจิกายน 2562,จาก: http://www.advance-electronic.com/blog/detail/112/th/PLC.html

[7] MELSOFT GX Works2 [ออนไลน์]. เข้าถึงเมื่อ 10 กุมภาพันธ์ 2563,จาก: https://www.mitsubishifa.co.th/files/dl/GX%20Works2%20Starting%20guide.pdf

[8] พื้นฐานการเขียน Structured Text [ออนไลน์]. เข้าถึงเมื่อ 10 กุมภาพันธ์ 2563,จาก: https://www.mitsubishifa.co.th/files/dl/GX%20Works2%20Starting%20guide.pdf

[9] เอชเอ็มไอ [ออนไลน์]. เข้าถึงเมื่อ 8 กุมภาพันธ์ 2563,จาก: http://dspace.spu.ac.th/bitstream/g.202.pdf