บทที่ 1
บทนำ

1.1 หลักการและเหตุผลของโครงการวิจัย

ระบบการผลิตในปัจจุบันต้องการรอบเวลาการผลิตที่สั้น (Shot Production Cycle) และคุณภาพการผลิตสูง ทำให้มีความจำเป็นในการนำหุ่นยนต์มาใช้ในอุตสาหกรรมเพิ่มขึ้นอย่างต่อเนื่องเนื่องจากหุ่นยนต์มีการทำงานที่ยืดหยุ่น ซึ่งทำให้สามารถเปลี่ยนแปลงการผลิตได้ง่ายและรวดเร็ว อย่างไรก็ตาม การโปรแกรมหุ่นยนต์ให้ทำงานตามที่ต้องการต้องอาศัยความรู้และทักษะค่อนข้างสูงจากผู้ปฏิบัติงาน ซึ่งถือว่าเป็นบุคลากรที่ยังขาดแคลนในปัจจุบัน และบัณฑิตที่จบใหม่จากสถาบันอุดมศึกษายังขาดทักษะในการใช้งานหุ่นยนต์อุตสาหกรรม ถึงแม้ว่าจะมีการบรรจุวิชาหุ่นยนต์อุตสาหกรรมไว้ในหลักสูตรแล้วก็ตาม สาเหตุหลักของปัญหาเนื่องจากหุ่นยนต์เป็นเครื่องจักรที่มีราคาแพง ทำให้สถาบันอุดมศึกษาหลายๆ แห่งไม่มีหุ่นยนต์ให้นักศึกษาได้ลองปฏิบัติจริง หรือมีก็เพียงแต่รุ่นขนาดเล็กที่มีขีดความสามารถต่ำเท่านั้น อีกทั้งการบำรุงรักษาให้หุ่นยนต์อยู่ในสภาพที่พร้อมใช้งานต้องใช้ค่าใช้จ่ายสูง ทำให้หน่วยงานในสถาบันอุดมศึกษาหลายแห่งมีหุ่นยนต์แต่ไม่สามารถใช้ในการเรียนการสอนเชิงปฏิบัติได้

สาขาวิชาวิศวกรรมอุตสาหการ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังได้มีการเปิดสอนวิชา *ปฏิบัติการทางวิศวกรรมอุตสาหการ* ซึ่งได้บรรจุหุ่นยนต์ไว้ในหัวข้อของการปฏิบัติการ เพื่อเปิดโอกาสให้นักศึกษาได้ทดลองใช้งานหุ่นยนต์ในเชิงปฏิบัติ โดยหุ่นยนต์ที่ใช้เป็นหุ่นยนต์ขนาดเล็กรุ่น SCORBOT-ER 4u ปัญหาที่เกิดขึ้นในการเรียนการสอนคือหุ่นยนต์มีไม่เพียงพอเนื่องจากมีเพียงเครื่องเดียวเท่านั้น ทำให้ต้องแบ่งนักศึกษาออกเป็นกลุ่มและนักศึกษาไม่สามารถใช้งานได้อย่างทั่วถึง ปัญหาอีกอย่างหนึ่งคือมีการบำรุงรักษาและซ่อมแซมหุ่นยนต์บ่อยครั้งเนื่องจากนักศึกษาขาดทักษะในการโปรแกรมหุ่นยนต์ ทำให้หุ่นยนต์เกิดการชนกับวัตถุที่อยู่รอบข้างในระหว่างการเคลื่อนที่จนทำให้ชิ้นส่วนหรืออุปกรณ์ชำรุดเสียหาย

การโปรแกรมหุ่นยนต์สามารถทำได้ทั้งแบบเชื่อมตรง (On-line) และแบบไม่เชื่อมตรง (Off-line) ซึ่งการโปรแกรมแบบไม่เชื่อมตรงทำให้นักศึกษาสามารถโปรแกรมหุ่นยนต์ที่เครื่องคอมพิวเตอร์อื่นในขณะที่นักศึกษาอีกคนกำลังทดลองกับหุ่นยนต์ได้ ซึ่งทำให้ประหยัดเวลาในการปฏิบัติงาน อย่างไรก็ตาม เนื่องจากนักศึกษายังขาดประสพการณ์ในการเขียนโปรแกรม ทำให้ต้องมีการแก้ไขโปรแกรมในขณะที่เชื่อมต่อกับหุ่นยนต์โดยตรง ซึ่งทำให้เสียเวลาในการรอคอยเป็นเวลานานอย่างหลีกเลี่ยงไม่ได้

คอมพิวเตอร์ได้ถูกนำมาใช้ในการจำลองการทำงานของหุ่นยนต์เพื่อช่วยลดปัญหาที่เกิดขึ้นกับการใช้งานหุ่นยนต์ที่ต้องอาศัยทักษะในการปฏิบัติงานสูง โดยคอมพิวเตอร์จะทำการสร้างภาพเคลื่อนไหวสามมิติเพื่อแสดงการเคลื่อนที่ของหุ่นยนต์ตามโปรแกรมที่ได้กำหนดไว้ ดังนั้น ผู้ปฏิบัติงานสามารถตรวจสอบการทำงานของหุ่นยนต์ก่อนที่จะนำโปรแกรมไปใช้กับหุ่นยนต์จริง เช่น การตรวจสอบเส้นทางการเคลื่อนที่ การตรวจสอบการชนกับวัตถุรอบข้าง เป็นต้น นอกจากนี้ การจำลองการทำงานของหุ่นยนต์ยังสามารถใช้ในการตรวจสอบตำแหน่งที่หุ่นยนต์สามารถเข้าถึงได้ และใช้ในการหา รอบเวลาการทำงานของหุ่นยนต์ ประโยชน์ที่เห็นได้ชัดของการใช้โปรแกรมจำลองการทำงานของหุ่นยนต์คือการที่นักศึกษาสามารถเขียนโปรแกรมไปพร้อมๆ กับการแก้ไข

โปรแกรมบนเครื่องคอมพิวเตอร์ได้โดยไม่จำเป็นต้องเชื่อมต่อกับหุ่นยนต์โดยตรงจนกว่าโปรแกรมที่เขียนขึ้นจะไม่มีข้อผิดพลาด ทำให้ช่วยลดเวลาในการปฏิบัติงานกับหุ่นยนต์จริงและเพิ่มโอกาสในการใช้งานหุ่นยนต์ให้ทั่วถึงได้ นอกจากนี้ การใช้โปรแกรมจำลองการทำงานของหุ่นยนต์ยังเปิดโอกาสให้นักศึกษาได้ลองใช้หุ่นยนต์รุ่นอื่นๆ ได้โดยไม่จำเป็นต้องมีตัวหุ่นยนต์จริง ซึ่งเป็นการเสริมทักษะของนักศึกษาในการใช้งานหุ่นยนต์และเตรียมความพร้อมในการออกไปทำงานจริงในอนาคต

ดังนั้น ผู้วิจัยจึงได้เสนองานวิจัยนี้ในการพัฒนาโปรแกรมจำลองการทำงานของหุ่นยนต์เพื่อแก้ปัญหาจำนวนหุ่นยนต์ไม่เพียงพอ รวมทั้งลดปัญหาความเสี่ยงของการชำรุดของหุ่นยนต์อันเกิดจากการขาดทักษะในการใช้งานของนักศึกษา

1.2 วัตถุประสงค์ของโครงการวิจัย

- เพื่อพัฒนาโปรแกรมคอมพิวเตอร์ที่สามารถจำลองการทำงานของหุ่นยนต์รุ่น SCORBOT-ER 4u และรองรับหุ่นยนต์อุตสาหกรรมรุ่นอื่นๆ ได้
- เพื่อใช้เป็นสื่อการเรียนการสอน

1.3 ขอบเขตของโครงการวิจัย

- พัฒนาโปรแกรมคอมพิวเตอร์สำหรับหุ่นยนต์รุ่น SCORBOT-ER 4u
- โปรแกรมคอมพิวเตอร์ที่พัฒนาขึ้นมีโครงสร้างที่สามารถรองรับหุ่นยนต์อุตสาหกรรมรุ่นอื่นๆ ได้ โดยผู้ใช้สามารถป้อนข้อมูลและแบบของหุ่นยนต์เองได้ภายหลัง
- สามารถจำลองการเคลื่อนที่ของหุ่นยนต์โดยใช้ภาพเคลื่อนไหวสามมิติ

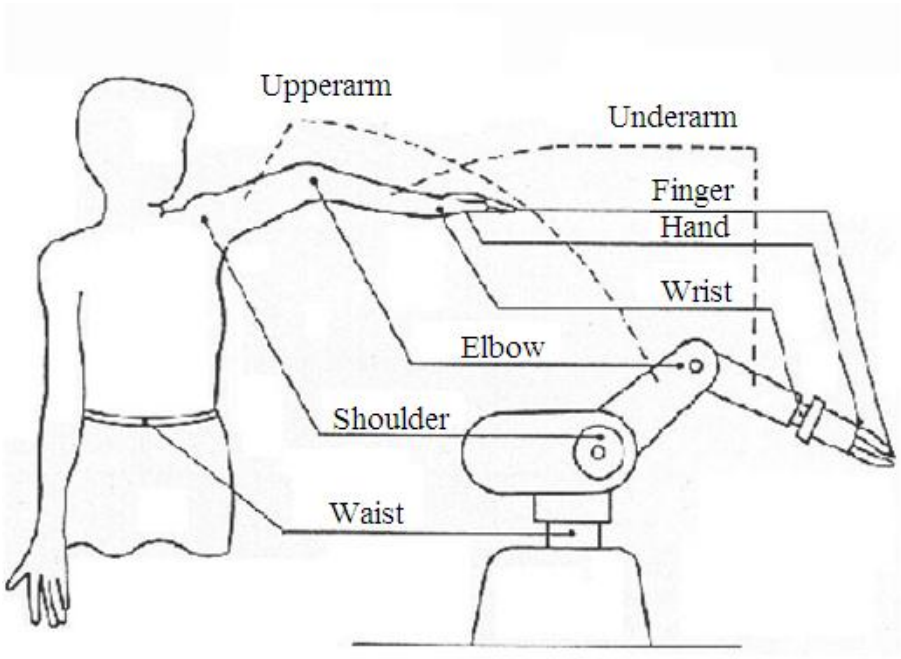
1.4 ประโยชน์ที่คาดว่าจะได้รับ

- เสริมทักษะการเรียนรู้ของนักศึกษาในการใช้งานหุ่นยนต์อุตสาหกรรม
- ลดความเสียหายที่อาจจะเกิดขึ้นในการใช้งานหุ่นยนต์และค่าใช้จ่ายในการบำรุงรักษาหุ่นยนต์
- ช่วยยืดอายุการใช้งานของหุ่นยนต์

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 หุ่นยนต์อุตสาหกรรม (Industrial Robot)

การทำงานของหุ่นยนต์อุตสาหกรรมจะเลียนแบบร่างกายของมนุษย์ โดยจะเลียนแบบเฉพาะส่วนของร่างกายที่จะนำไปใช้ประโยชน์ในอุตสาหกรรมเท่านั้น นั่นคือช่วงแขนของมนุษย์ ดังนั้นคำว่า “แขนกล” จึงเป็นอีกชื่อหนึ่งของหุ่นยนต์อุตสาหกรรม การทำงานของหุ่นยนต์อุตสาหกรรมเปรียบเทียบกับแขนของมนุษย์ แสดงดังรูปที่ 2.1



รูปที่ 2.1 แสดงส่วนต่างๆ ของหุ่นยนต์เปรียบเทียบกับสรีระของมนุษย์

2.1.1 นิยามของหุ่นยนต์ (Robotics definition)

คำจำกัดความของหุ่นยนต์ตามมาตรฐาน ISO 8373 คือ “An automatically controlled, reprogrammable, multipurpose, manipulator programmable in three or more axes which may be either fixed in place or mobile for use in industrial automation application” ซึ่งหมายถึงเครื่องจักรที่ถูกควบคุมอัตโนมัติ สามารถเขียนโปรแกรมใหม่ได้

ใช้งานเอนกประสงค์ โปรแกรมการเคลื่อนที่จะต้องสามารถโปรแกรมให้เคลื่อนที่ได้อย่างน้อย 3 แกนหรือมากกว่า หุ่นยนต์อาจจะยึดอยู่กับที่หรือเคลื่อนที่ได้ เพื่อใช้ในงานอุตสาหกรรม

2.1.2 การทำงานของข้อต่อของหุ่นยนต์

ข้อต่อ (Joint) ของหุ่นยนต์ อุตสาหกรรมสามารถแบ่งเป็น 2 ประเภทตามลักษณะการทำงาน ดังนี้

ตารางที่ 2.1 การทำงานของข้อต่อ

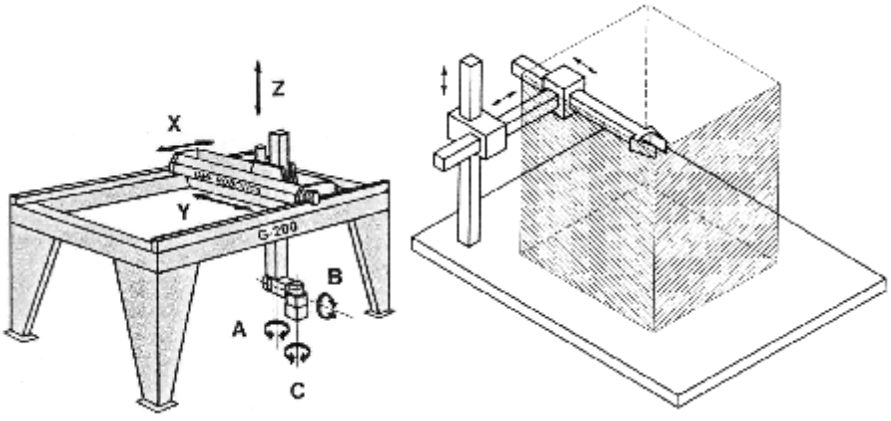
ชนิด	สัญลักษณ์	หมายเหตุ
Revolute (R)		การหมุนรอบแกน (Rotary)
Prismatic (P)		การเคลื่อนที่เชิงเส้น (Linear motion)

2.1.3 การแบ่งชนิดของหุ่นยนต์

การแบ่งชนิดของหุ่นยนต์ จะแบ่งตามลักษณะรูปทรงของพื้นที่ทำงาน (Envelope Geometric) ดังนี้

2.1.3.1 Cartesian Robot (Gantry Robot)

แกนทั้ง 3 ของหุ่นยนต์จะเคลื่อนที่เป็นแบบเชิงเส้น (Prismatic) ถ้าโครงสร้างมีลักษณะคล้าย Overhead Crane จะเรียกว่าเป็นหุ่นยนต์ชนิด gantry แต่ถ้าหุ่นยนต์ไม่มีขาตั้งหรือขาเป็นแบบอื่น เรียกว่า ชนิด Cartesian



a) Gantry Robot b) Work envelope of Gantry Robot

รูปที่ 2.2 Cartesian Robot (Gantry Robot)

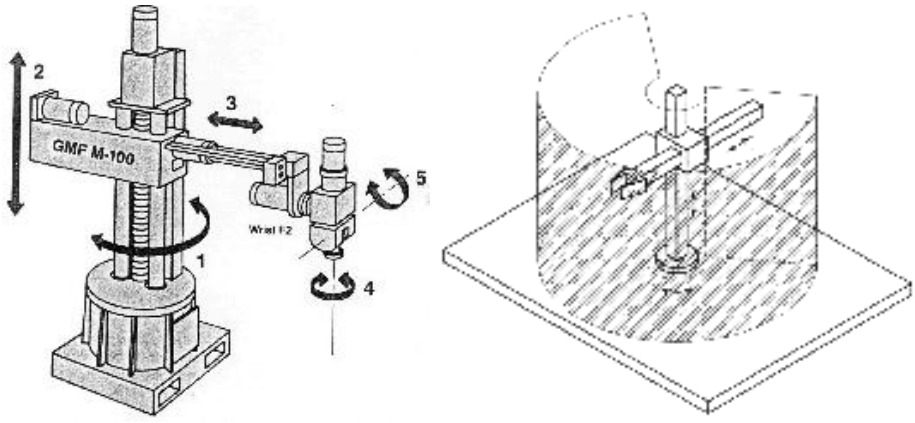
ตารางที่ 2.2 ข้อดี และข้อเสีย ของ Cartesian Robot

ข้อดี	ข้อเสีย
- เคลื่อนที่เป็นแนวเส้นตรงทั้ง 3 มิติ	- ต้องการพื้นที่ติดตั้งมาก
- การเคลื่อนที่สามารถทำความเข้าใจง่าย	- บริเวณที่หุ่นยนต์เข้าไปทำงานได้ จะเล็กกว่าขนาดของตัวหุ่นยนต์
- มีส่วนประกอบง่าย ๆ	- ไม่สามารถเข้าถึงวัตถุจากทิศทางข้างใต้ได้
- โครงสร้างแข็งแรงตลอดการเคลื่อนที่	- แกนแบบเชิงเส้นจะ Seal เพื่อป้องกันฝุ่นและของเหลวได้ยาก

เนื่องจากโครงสร้างของ Cartesian Robot มีความแข็งแรงตลอดแนวการเคลื่อนที่ ดังนั้นจึงเหมาะกับงานเคลื่อนย้ายของหนักๆ หรือเรียกว่างาน Pick – and – Place เช่นใช้โหลดชิ้นงานเข้าเครื่องจักร (Machine loading), ใช้จัดเก็บชิ้นงาน (Stacking) นอกจากนี้ยังสามารถใช้ในงานประกอบ (Assembly) ที่ไม่ต้องการเข้าถึงในลักษณะที่มีมุมหมุน เช่น ประกอบอุปกรณ์อิเล็กทรอนิกส์ และงานทดสอบต่างๆ

2.1.3.2 Cylindrical Robot

หุ่นยนต์ประเภทนี้จะมีแกนที่ 2 (ไหล่) และแกนที่ 3 (ข้อศอก) เป็นแบบ prismatic ส่วนแกนที่ 1 (เอว) จะเป็นแบบหมุน (revolute) ทำให้การเคลื่อนที่ได้พื้นที่การทำงานเป็นรูปทรงกระบอก ดังรูปที่ 2.3b



a) Cylindrical Robot b) Work envelope of Cylindrical Robot

รูปที่ 2.3 Cylindrical Robot

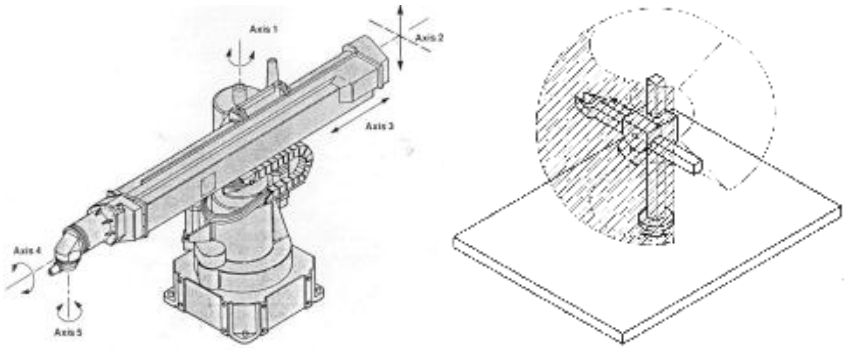
ตารางที่ 2.3 ข้อดี และข้อเสีย ของ Cylindrical Robot

ข้อดี	ข้อเสีย
- มีส่วนประกอบไม่ซับซ้อน	- มีพื้นที่ทำงานจำกัด
- การเคลื่อนที่สามารถเข้าใจได้ง่าย	- แกนที่เป็นเชิงเส้นมีความยุ่งยากในการ seal เพื่อป้องกันฝุ่นและของเหลว
- สามารถเข้าถึงเครื่องจักรที่มีการเปิด-ปิด หรือเข้าไปในบริเวณที่เป็นช่องหรือโพรงได้ง่าย (Loading) เช่น การโหลดชิ้นงานเข้าเครื่อง CNC	

Cylindrical Robot โดยทั่วไปจะใช้ในการหยิบยกชิ้นงาน (Pick-and-Place) หรือป้อนชิ้นงานเข้าเครื่องจักร เพราะสามารถเคลื่อนที่เข้าออกบริเวณที่เป็นช่องโพรงเล็กๆ ได้สะดวก

2.1.3.3 Spherical Robot (Polar)

Spherical Robot มีสองแกนที่เคลื่อนในลักษณะการหมุน (Revolute Joint) คือแกนที่ 1 (เอว) และแกนที่ 2 (ไหล่) ส่วนแกนที่ 3 (ข้อศอก) จะเป็นลักษณะของการเคลื่อนที่แนวเส้นตรง ดังรูปที่ 2.4 a) ซึ่งทำให้ได้พื้นที่การทำงานเป็นรูปทรงกลม ดังรูปที่ 2.4 b)



a) Spherical Robot b) Work envelope of Spherical Robot

รูปที่ 2.4 Spherical Robot

ตารางที่ 2.4 ข้อดี และข้อเสีย ของ Spherical Robot

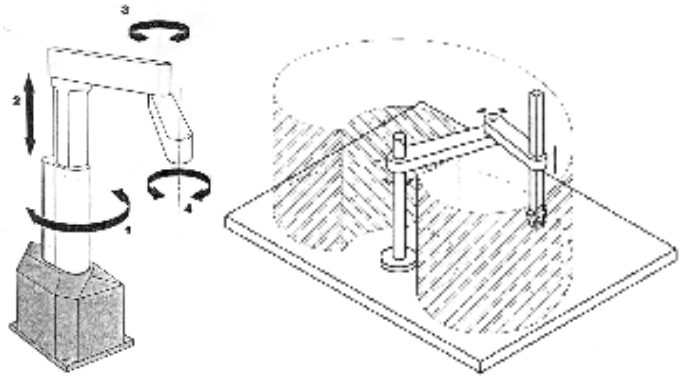
ข้อดี	ข้อเสีย
- มีปริมาตรการทำงานมากขึ้น เนื่องจากการหมุนของแกนที่ 2 (ไหล่)	- มีระบบพิกัด (Coordinate) และส่วนประกอบ ที่ซับซ้อน

- สามารถที่จะก้มลงมาจับชิ้นงานบนพื้นได้สะดวก	- การเคลื่อนที่และระบบควบคุมมีความซับซ้อนขึ้น

Spherical Robot ใช้ในงานที่มีการเคลื่อนที่ในแนวตั้ง (Vertical) เพียงเล็กน้อย เช่น การโหลดชิ้นงานเข้าออกจากเครื่องปั๊ม (Press) หรืออาจจะใช้งานเชื่อมจุด (Spot Welding)

2.1.3.4 SCARA Robot

หุ่นยนต์ SCARA (Selective Compliance Assembly Robot Arm) จะมีลักษณะแกนที่ 1 (เอว) และแกนที่ 3 (ข้อศอก) หมุนรอบแกนแนวตั้ง ส่วนแกนที่ 2 จะเป็นลักษณะการเคลื่อนที่ขึ้นลง (Prismatic) ดังรูปที่ 2.5a ทำให้ได้พื้นที่การทำงานดัง รูปที่ 2.5b หุ่นยนต์ SCARA จะเคลื่อนที่ได้รวดเร็วในแนวระนาบ และมีความแม่นยำสูง



a) Scara Robot b) Work envelope of Scara Robot

รูปที่ 2.5 SCARA Robot

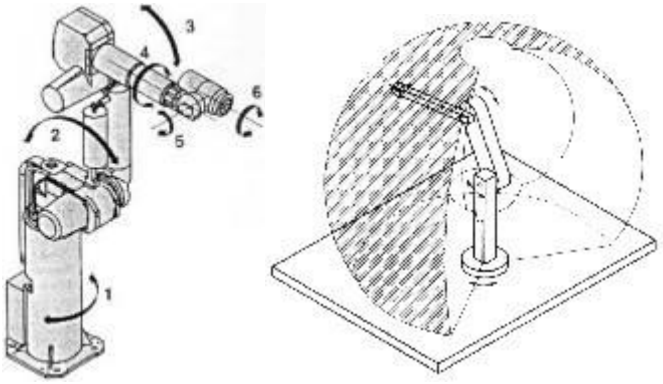
ตารางที่ 2.5 ข้อดี และข้อเสีย ของ SCARA Robot

ข้อดี	ข้อเสีย
- มีความแม่นยำสูง	- มีพื้นที่ทำงานจำกัด
- สามารถเคลื่อนที่ในแนวระนาบ และขึ้นลงได้รวดเร็ว	- ไม่สามารถหมุน (rotation)ในลักษณะมุมต่างๆได้
	- สามารถยกน้ำหนัก (Payload) ได้ไม่มากนัก

การประยุกต์การนำไปใช้งาน เนื่องจากการเคลื่อนที่ในแนวระนาบและขึ้นลงได้รวดเร็วจึงเหมาะกับงานประกอบชิ้นส่วนทางอิเล็กทรอนิกส์ที่ต้องการความรวดเร็ว และการเคลื่อนที่ไม่ต้องการการหมุนมากนัก แต่จะไม่เหมาะกับงานประกอบชิ้นส่วนทางกล (Mechanical part) ซึ่งส่วนใหญ่การประกอบจะอาศัยการหมุน (rotation) ในลักษณะมุมต่างๆ นอกจากนี้ SCARA Robot ยังเหมาะกับงานตรวจสอบ (Inspection) งานบรรจุภัณฑ์ (Packaging)

2.1.3.5 Articulated Arm (Revolute)

แกนการเคลื่อนที่ของ Articulated Arm จะเป็นแบบหมุน (Revolute) ทั้งหมด รูปแบบการเคลื่อนที่จะคล้ายกับแขนคน ซึ่งจะประกอบด้วยช่วงเอว ท่อนแขนบน ท่อนแขนล่าง ข้อมือ การเคลื่อนที่ทำให้ได้พื้นที่การทำงานดังรูปที่ 2.6b



a) Articulated Robot b) Work envelope of Articulated Robot

รูปที่ 2.6 Articulated Arm

ตารางที่ 2.6 ข้อดี และข้อเสีย ของ Articulated Arm

ข้อดี	ข้อเสีย
- เนื่องจากทุกแกนจะเคลื่อนที่ในลักษณะของการหมุนทำให้มีความยืดหยุ่นสูงในการเข้าไปยังจุดต่างๆ	- มีระบบพิกัด (Coordinate) ที่ซับซ้อน
- บริเวณข้อต่อ (Joint) สามารถ Seal เพื่อป้องกันฝุ่น ความชื้นหรือน้ำได้ง่าย	- การเคลื่อนที่และระบบควบคุมทำความ เข้าใจได้ยากขึ้น
- มีพื้นที่การทำงานมาก	- ควบคุมให้เคลื่อนที่ในแนวเส้นตรง (Linear) ได้ยาก
- สามารถเข้าถึงชิ้นงานทั้งจากด้านบน ด้านล่าง	- โครงสร้างไม่มั่นคงตลอดช่วงการเคลื่อนที่ เพราะบริเวณพื้นที่การทำงาน (Work envelope) ปลายแขนจะมีการสั่น ทำให้ความแม่ยำลดลง
- เหมาะกับการใช้มอเตอร์ไฟฟ้า เป็นชุดขับเคลื่อน	

การประยุกต์การนำไปใช้งาน หุ่นยนต์ชนิดนี้สามารถใช้งานได้กว้างขวาง เพราะสามารถเข้าถึงตำแหน่งต่างๆ ได้ดี เช่น งานเชื่อม Spot Welding, Path Welding, งานยกของ, งานตัด, งานทากาว, งานที่มีการเคลื่อนที่ยากๆ เช่น งานพ่นสี งาน sealing ฯลฯ

การนำข้อต่อ (Joint) ทั้งสองแบบมาต่อเข้าด้วยกันอย่างน้อย 3 แกนหลักจะได้พื้นที่ทำงาน (Work envelope) ที่มีลักษณะแตกต่างกันไป สามารถนำมาสรุปเป็นตาราง เมื่อแบ่งตามชนิดของหุ่นยนต์ได้ดังต่อไปนี้

ตารางที่ 2.7 สรุปการนำข้อต่อทั้งสองแบบมาต่อเข้าด้วยกัน

ชนิดของหุ่นยนต์	แกนที่ 1 (เอว)	แกนที่ 2 (ไหล่)	แกนที่ 3(ข้อศอก)
Cartesian (gantry)	P	P	P
Cylindrical	R	P	P
Spherical (Polar)	R	R	P
SCARA Robot	R	P	R
Articulated Arm	R	R	R
R = Revolute, P = Prismatic			

(สัมพันธ์ แหล่งป่าหมุ้น : http://www.elecnet.chandra.ac.th/learn/courses/ELTC2401/unit4/robot/robot2.htm)

2.2 การเขียนโปรแกรมเชิงวัตถุ (Object-oriented programming, OOP)

การเขียนโปรแกรมเชิงวัตถุ คือการเขียนโปรแกรมโดยการมองว่าส่วนประกอบของโปรแกรมเป็นเสมือนวัตถุชิ้นหนึ่งที่ประกอบไปด้วย คุณสมบัติ (property) ซึ่งสามารถอธิบายได้ว่าวัตถุนี้คืออะไร และวิธีการหรือที่เรียกว่าเมธอด (Method) ซึ่งสามารถอธิบายพฤติกรรมของวัตถุนั้นว่าสามารถทำอะไรได้ การเขียนโปรแกรมเชิงวัตถุเป็นการแบ่งซอฟท์แวร์หรือโปรแกรมออกเป็นส่วนๆ เรียกว่า คลาส (Class) โดยการนิยามคลาสและออบเจ็กต์ เพื่อทำให้สามารถนำส่วนของซอฟท์แวร์หรือโปรแกรม ส่วนนั้นกลับมาเรียกใช้งานได้อีก เพื่อลดความซ้ำซ้อนและเวลาในการพัฒนาโปรแกรมลง การทำงานของคลาส จะถูกกำหนดโดยส่วนอินเตอร์เฟสของเมธอด ส่วนการทำงานของส่วนที่เป็นโค้ด (Code) จะไม่ถูกคำนึงถึงมากนักในการออกแบบภาษา การเขียนโปรแกรมเชิงวัตถุสนใจเฉพาะข้อมูลที่จะถูกประมวลผลมากกว่าฟังก์ชันที่ทำการประมวลข้อมูลนั้นๆ

2.2.1 หลักสำคัญของการเขียนโปรแกรมเชิงวัตถุ

- Class and Subclass
- Encapsulation
- Inheritance
- Polymorphism
- Abstract Data Type

2.2.1.1 คลาส

คลาส คือกลุ่ม (category) ของออบเจ็กต์ที่มีคุณสมบัติและพฤติกรรมที่เหมือนกัน โดยคลาสจะต้องประกอบไปด้วย data, behavior และ interface หรือหมายถึงต้นแบบ (prototype) หรือพิมพ์เขียว ที่กำหนดตัวแปรและวิธีการเพื่อนำไปใช้ได้ในทุกออบเจ็กต์ของคลาส

2.2.1.2 ออบเจ็กต์

ออบเจ็กต์ คือ สิ่งใดๆ ซึ่งมีคุณลักษณะ (State) บ่งบอกถึงความเป็นตัวของมันเองในขณะนั้น และสามารถแสดงพฤติกรรม (Behavior) ของตัวเองออกมาได้ เช่น รถยนต์สีน้ำเงิน มีความหมายคือ วัตถุประเภทรถยนต์ มีคุณลักษณะของสีเป็นสีน้ำเงิน และมีพฤติกรรมที่แสดงถึงการเคลื่อนที่ และหยุดได้ หรือกล่าวได้ว่าออบเจ็กต์ คือข้อมูลของคลาส (เป็น entities ของคลาส) ซึ่งทุกๆ อย่างจะจัดเป็นออบเจ็กต์ โดยต้องประกอบไปด้วย

- ชื่อ (Identity)
- สถานะ (State) คุณสมบัติ หรือค่าของข้อมูล ซึ่งแทนด้วย value
- พฤติกรรม (Behavior) ที่ระบุว่าสามารถทำอะไรได้บ้าง ซึ่งแทนด้วยเมธอด

2.2.1.3 เมธอด

เมธอด คือ ฟังก์ชันที่บ่งบอกพฤติกรรมของออบเจ็กต์ว่าทำอะไรได้บ้าง เมธอดถูกกำหนดไว้ในคลาส ซึ่งประกอบด้วย ชื่อของเมธอด เรียกว่า Identifier ตามด้วยเครื่องหมายวงเล็บ () โดยในวงเล็บอาจมี parameter list อยู่หรือไม่ก็ได้ เช่น

getBalance()

raiseSalary(float Salary, float Percent)

2.2.1.4 Constructor Method

คือเมธอดที่ใช้สาหรับสร้าง instance object ของคลาสนั้นๆ โดยที่ชื่อเมธอดนี้ต้องเหมือนกับชื่อคลาส และใช้สำหรับ initialize ข้อมูลให้กับ instance variable โดยจะไม่มีการถ่ายทอดให้กับ subclass และไม่มีการ return ค่า

2.2.1.5 Message

คือคำสั่งหรือข้อความที่จะให้ข้อมูลหรือตัวแปรใดทำงาน ก็คือ parameter ในภาษาอื่นที่ไม่ใช่ OOP คือใช้เพื่อนำส่งค่าข้อมูลระหว่างออบเจ็กต์ โดยใน message นั้นต้องประกอบด้วย

- Destination (ชื่อของออบเจ็กต์)
- เมธอด
- Parameters

2.2.1.6 Accessibility

เป็นการกำหนดการเข้าถึงของข้อมูลหรือเมธอดภายในคลาส โดยใช้คีย์เวิร์ด ต่อไปนี้ในการกำหนดระดับการเข้าถึง

- public: เข้าถึงได้ในทุกที่
- private: เข้าถึงได้เฉพาะภายในคลาส เท่านั้น ไม่รวม subclass
- protected: เข้าถึงได้เฉพาะภายในคลาส และ subclass ที่สืบทอดกันมา (Inherit)

- default: ถ้าไม่ระบุ จะเข้าถึงข้อมูลภายในคลาส และอยู่แพ็กเกจเดียวกัน

2.2.1.7 Encapsulation

จากแนวทางเชิงวัตถุได้นำเอา Attribute (Data) และเมธอดเก็บรวมกันไว้ภายแต่ละออบเจ็กต์ นับว่าเป็นการห่อหุ้มรายละเอียดของออบเจ็กต์ (ข้อมูลและพฤติกรรมของออบเจ็กต์) ไว้ เปรียบเสมือนการนำตัวยาหลายๆ ชนิดมาผสมรวมกัน แล้วบรรจุในแคปซูล (Capsule) ซึ่งในความเป็นจริงแล้วถ้าไม่ใช่แคปซูลใส จะไม่สามารถรู้ได้ว่าภายในประกอบด้วยอะไรบ้าง OOP ได้นำเอาหลักการดังกล่าวมาประยุกต์ใช้กับการมองเห็นและการเข้าถึงรายละเอียดของคลาส ซึ่งมุมมองดังกล่าวเป็นที่มาของหลักการ Encapsulation

Encapsulation หมายถึง การห่อหุ้ม Attribute และเมธอดของออบเจ็กต์เข้าไว้ด้วยกัน เพื่อป้องกันการมองเห็นหรือเข้าถึงข้อมูลจากออบเจ็กต์อื่น สำหรับประโยชน์ของ Encapsulation คือ ทำให้เกิดการนำกลับมาใช้ใหม่ (reuse) การใช้งานออบเจ็กต์กระทำโดยการส่งข้อความ (Message) เพื่อเรียกใช้วิธีการของออบเจ็กต์ ซึ่งทำให้เกิดผลดีคือ เป็นการป้องกันสิ่งที่อยู่ในออบเจ็กต์ไม่ให้ได้รับผลกระทบที่เกิดจากการเปลี่ยนแปลงภายนอก และรักษาระบบไม่ให้ได้รับผลกระทบจากการเปลี่ยนแปลงภายในออบเจ็กต์

หลักการ Encapsulation ก่อให้เกิดการมองออบเจ็กต์ได้ใน 2 มุมคือ การมองออบเจ็กต์จากภายใน (Internal View) และการมองออบเจ็กต์จากภายนอก (Outside View) ซึ่งถ้ามองออบเจ็กต์จากภายในตัวเอง จะเห็นรายละเอียดทั้งหมดของออบเจ็กต์ (ทั้ง Attribute และเมธอด) แต่ถ้ามองออบเจ็กต์จากภายนอก จะเห็นเฉพาะสิ่งที่ออบเจ็กต์เปิดเผยทาง Public Interface เท่านั้น ซึ่งเรียกว่าเป็นการซ่อนข้อมูล (Information Hiding)

Information Hiding หรือการซ่อนรายละเอียดของคลาสหรือออบเจ็กต์นั้น มีหลายระดับแตกต่างกันออกไป ในบางรายละเอียดของคลาส ซึ่งอาจเปิดเผยให้ภายนอกสามารถมองเห็นและใช้งานได้โดยตรง (เมธอด) ในทางตรงกันข้าม ในบางรายละเอียดเราอาจต้องการปกปิดไม่ยอมให้ภายนอกได้เห็นเลยก็ได้ (Properties) ซึ่งระดับในการมองเห็นรายละเอียดต่างๆ ของคลาสจากภายนอกนี้ เรียกว่า “Visibility”

ทั้งนี้เรายังสามารถกำหนดให้พร็อพเพอร์ตี้ หรือเมธอดของคลาสมี Visibility เป็นระดับใดก็ได้ ขึ้นอยู่กับความต้องการและระดับของความจำเป็นในการปกปิดรายละเอียด แบ่งออกเป็น 3 ระดับ คือ Private, Protected และ Public

2.2.1.8 Inheritance

คือ การสร้างคลาสใหม่ ซึ่งสืบทอดคุณลักษณะ และพฤติกรรมของอีกคลาสหนึ่ง ดังนั้นคลาสที่สร้างขึ้นใหม่จึงมีเมธอด และพร็อพเพอร์ตี้เหมือนในคลาสต้นแบบทุกประการ โดยคลาสที่เป็นคลาสต้นแบบเรียกว่า “Superclass” และคลาสที่สืบทอดคุณสมบัติเรียกว่า “Subclass” นอกจากนี้ Subclass ยังสามารถแก้ไขหรือเพิ่มเติมเมธอดและพร็อพเพอร์ตี้ได้ด้วย

2.2.1.9 Polymorphism

การทำให้ message อันหนึ่งสามารถส่งให้ออบเจ็กต์แต่ละตัวในคลาส และ subclass ตอบสนองต่อ message อันเดียวกัน ในลักษณะที่เหมาะสมกับคลาส ของตัวเอง เช่น method print นี้สามารถส่งให้ทุกออบเจ็กต์ของคลาส และ

subclass ที่ทำให้ออบเจ็กต์นั้นรู้จัก method print และแต่ละออบเจ็กต์ที่ต่างกันจะตอบสนองต่อ message นี้ต่างกันออกไป ตามความสามารถในการใช้

2.2.1.10 Abstract Data Type (ADT)

รูปแบบชนิดของข้อมูล ผู้พัฒนาเป็นผู้กำหนดขึ้นมาเอง

2.2.1.11 เหตุผลที่ OOP มีบทบาทมากขึ้น

- ง่าย และรวดเร็ว ทำให้ลดเวลาในการพัฒนาลงไปได้
- เพิ่มปริมาณงานที่ได้ และมีความน่าเชื่อถือมากกว่า
- สามารถนำโค้ดกลับมาใช้ได้อีก (เรียกใช้คลาส)
- ทำต้นแบบ (Prototyping) ได้รวดเร็วกว่า
- ลดต้นทุนในการสร้าง และบำรุงรักษาซอฟท์แวร์
- การเปลี่ยนแปลงแก้ไข ไม่ทำให้เกิดผลกระทบไปยังภายนอกคลาส

(สุธี พงศาสกุลชัย และ หทัยชนก งามอินทร์, 2550)

2.3 OpenGL

2.3.1 ความหมายของ OpenGL

OpenGL (Open Graphics Library) เป็นซอฟต์แวร์ไลบรารี (Software Library) ที่ใช้ติดต่อกับฮาร์ดแวร์เพื่อแสดงภาพกราฟิก โดย OpenGL จะมีคำสั่งสำหรับการวาดภาพพื้นฐาน คือ จุด เส้น และรูปเหลี่ยมต่างๆ ซึ่งคำสั่งพื้นฐานมีประมาณ 120 คำสั่ง ที่สามารถใช้กำหนดคุณลักษณะและควบคุมการทำงานของแอพพลิเคชั่น 3 มิติ ซึ่งผู้พัฒนาโปรแกรมสามารถใช้ไลบรารี OpenGL ได้โดยไม่มีค่าลิขสิทธิ์ ทำให้มีการนำไลบรารีของของ OpenGL ไปใช้งานอย่างแพร่หลายในงานกราฟิก

ภาษาที่สามารถใช้กับ OpenGL มีดังนี้ C/C++ (VC++, Borland C++ C++ Builder, C Complier on UNIX), Delphi, Visual Basic, Java, Perl, Python, Fortran และ Ada เป็นต้น

เนื่องจากโครงสร้างของ OpenGL เป็นอินเตอร์เฟซที่เป็นอิสระจากฮาร์ดแวร์ (Hardware – independent interface) และสามารถใช้ได้กับระบบปฏิบัติการหลายๆ แบบ ไม่ว่าจะเป็น Window, UNIX เป็นต้น และด้วยเหตุที่ OpenGL ถูกออกแบบให้ทำงานโดยไม่ยึดติดกับระบบ สามารถทำงานได้บนทุกๆ แพลตฟอร์ม (Independent Platform) ทำให้สามารถเคลื่อนย้ายโค้ดที่สร้างเรียบร้อยแล้วไปใช้แพลตฟอร์มอื่นได้อย่างสะดวก (Portability) โดยไม่ต้องเปลี่ยนแปลงโค้ดโปรแกรมเลย การที่ OpenGL สามารถใช้ได้กับระบบปฏิบัติการที่หลากหลายนี้เอง ทำให้ OpenGL ไม่มีคำสั่งที่จัดการกับระบบปฏิบัติการเลย อีกทั้งยังไม่มีคำสั่งเพื่อรับอินพุตจากผู้ใช้อีกด้วย หน้าที่ทั้งสองอย่างนี้เป็นของผู้เขียนโปรแกรมที่จะต้องออกแบบและเขียนโค้ดเพื่อให้การทำงานเป็นไปอย่างมีประสิทธิภาพ แต่อย่างไรก็ตามยังมียูทิลิตี้ (Utility) ที่ช่วยจัดการงานทั้งสองนี้ หากพัฒนาโปรแกรมบนระบบปฏิบัติการแบบ Windows ยูทิลิตี้ดังกล่าวคือ GULT (OpenGL Utility Toolkit) อย่างไรก็ตาม OpenGL ยังไม่มีคำสั่งระดับสูงที่จะใช้วาดวัตถุ 3 มิติแบบซับซ้อน เช่น

รถยนต์ อวัยวะ หรือโมเลกุลต่างๆ สิ่งที่ OpenGL เตรียมไว้มีเพียงการสร้างรูปจำลองสามมิติ คือรูปทรงเรขาคณิตอย่างง่าย ได้แก่ จุด เส้น และรูปหลายเหลี่ยม ซึ่งผู้ใช้งานจะต้องนำรูปทรงเหล่านี้มาประกอบกันเพื่อให้เกิดรูปทรงสามมิติที่ซับซ้อน (ไลบรารี Open Inventor ถูกสร้างขึ้นจากชุดคำสั่งของ OpenGL เพื่อช่วยให้ผู้ใช้สามารถกำหนดสร้างรูปทรงที่ซับซ้อนได้โดยง่าย)

2.3.2 เหตุผลที่นิยมใช้ OpenGL ในระบบกราฟิก

ในการออกแบบและการทำงานในระบบกราฟิกมีเครื่องมือที่ช่วยงานหลายเครื่องมือ แต่ละเครื่องมือมีคุณสมบัติ ฟังก์ชัน หรือเครื่องมือที่แตกต่างกันไป เครื่องมือเหล่านั้นอาจมีจุดเด่น – จุดด้อยที่แตกต่างกันไป แต่ปัจจุบันเครื่องมือที่นิยมนำมาใช้ในการสร้างระบบกราฟิกคือ OpenGL ซึ่งเหตุผลสำคัญที่นิยมนำ OpenGL มาใช้งานมีดังนี้

- มีประสิทธิภาพสูงในการเร่งความเร็ว Application 3 มิติ และเกมต่างๆ ในปัจจุบัน
- สามารถใช้ข้อมูลจำนวนมาก สร้าง effect 3 มิติ ในแบบ real time ได้อย่างมีประสิทธิภาพ
- เพิ่มการสนับสนุนอุปกรณ์ใหม่ๆ ลงไปใน OpenGL ที่ทำได้ง่ายและรวดเร็ว
- ทำงานได้บนหลายแพลตฟอร์ม ทำให้การย้ายโปรแกรมประยุกต์ระหว่างแต่ละแพลตฟอร์มนั้นทำได้ง่าย และประหยัด
- มีเสถียรภาพในการทำงานสูง สามารถทำงานกับเครื่องเวิร์คสเตชันแบบ High End 3D และซุปเปอร์คอมพิวเตอร์ได้
- ใช้งานร่วมกับคอมไพเลอร์ได้หลากหลาย เช่น C/C++, Delphi, Visual Basic, Java, Perl, Fortran, Ada เป็นต้น

2.3.3 โครงสร้างไลบรารีพื้นฐานของ OpenGL

ไลบรารีพื้นฐานของ OpenGL จะมีฟังก์ชันเก็บอยู่ใน GL ฟังก์ชันเหล่านี้ชื่อจะขึ้นต้นด้วย gl หลังจากนั้นมีชื่อฟังก์ชันที่ขึ้นต้นตัวแรกด้วยอักษรพิมพ์ใหญ่ เช่น glBegin, glClear เป็นต้น ส่วนฟังก์ชันเฉพาะ ต้องใช้อาร์กิวเมนต์ 1 ตัว หรือมากกว่า 1 ตัว ซึ่งขึ้นอยู่กับฟังก์ชัน อาร์กิวเมนต์อาจเป็นสัญลักษณ์เฉพาะ เช่น ค่าคงที่ ชื่อพารามิเตอร์ ค่าพารามิเตอร์ หรือโหมดของพารามิเตอร์ เป็นต้น ค่าคงที่ทั้งหมดนี้จะขึ้นต้นด้วยอักษรพิมพ์ใหญ่ GL นอกจากนี้ยังมีเครื่องหมายขีดล่าง (Underscore; _) เพื่อคั่นระหว่างคอมโพเนนต์ เช่น GL_2D, GL_RGB เป็นต้น

ฟังก์ชันของ OpenGL อาจจะกำหนดประเภทข้อมูลได้ เช่น ใช้กำหนดประเภทข้อมูลตัวเลขจำนวนเต็ม 32 บิต แต่การกำหนดขนาดของตัวเลขจำนวนเต็มอาจจะแตกต่างกันไปตามเครื่อง การกำหนดค่าประเภทข้อมูล OpenGL จะใช้ชื่อประเภทข้อมูลที่มีในฟังก์ชัน โดยชื่อประเภทข้อมูลนี้จะขึ้นต้นด้วยอักษรพิมพ์ใหญ่ GL ต่อด้วยชื่อประเภทข้อมูลมาตรฐานที่เป็นตัวพิมพ์เล็ก (หรืออาจใช้ชื่อธรรมดา เช่น int, float) เช่น GLbyte, GLfloat, GLdouble เป็นต้น

บางอาร์กิวเมนต์ของฟังก์ชัน OpenGL สามารถกำหนดค่าโดยใช้อาร์เรย์ที่เป็นลิสต์ของค่าข้อมูลได้ มีออปชันในการกำหนดลิสต์ของค่าข้อมูลเป็นพอยเตอร์ของอาร์เรย์ อีกทั้งยังสามารถกำหนดลิสต์ในลักษณะ explicit ให้เป็นพารามิเตอร์ของอาร์กิวเมนต์ได้อีกด้วย

2.3.4 ไลบรารีที่เกี่ยวข้องกับ OpenGL

นอกจากไลบรารีหลักของ OpenGL แล้ว ยังมีไลบรารีที่เกี่ยวข้องอีกเป็นจำนวนมาก เพื่อจัดการกับงานเฉพาะได้อย่างมีประสิทธิภาพ ไลบรารีที่เกี่ยวข้องกับ OpenGL และเป็นไลบรารีเฉพาะมีดังนี้

2.3.4.1 OpenGL Utilities (GLU)

OpenGL Utilities (GLU) เป็นไลบราลีที่ประกอบด้วยรูทีนมากมายในการจัดการมุมมองเพื่อแสดงรูปพื้นฐานและออบเจ็กต์ที่ซับซ้อนที่ประกอบขึ้นจากเส้นและรูปหลายเหลี่ยม, แสดงรูปลูกบาศก์ เป็นต้น ตัวอย่างการสร้างวัตถุพื้นฐาน เช่น ฟังก์ชันในการสร้างทรงกลม ผู้ใช้ไม่จำเป็นต้องทราบว่าทรงกลมสร้างได้อย่างไรแต่สามารถเรียกใช้ฟังก์ชันได้เลย GLU นี้เป็นฟังก์ชันที่มีใน OpenGL อยู่แล้ว ซึ่งฟังก์ชัน GLU จะขึ้นต้นด้วยคำว่า glu เสมอ การเรียกใช้จะต้อง include ไฟล์ header ที่ชื่อ glu.h ในตอนต้นของโค้ดโปรแกรม

2.3.4.2 OpenGL Utility Toolkit (GLUT)

OpenGL Utility Toolkit (GLUT) คือไลราลีของระบบกราฟิกที่ช่วยในการติดต่อกับการแสดงผลทางจอภาพ ทั้งนี้เนื่องจากในการใช้งาน OpenGL เพื่อใช้งานด้านกราฟิก สิ่งแรกที่มีความจำเป็นต้องทำคือการกำหนดวินโดว์สำหรับการแสดงผล (display window) บนจอภาพ ซึ่งวินโดว์ดังกล่าวนี้คือพื้นที่สี่เหลี่ยมผืนผ้าของจอภาพที่ใช้แสดงกราฟิก ซึ่งเราไม่อาจสร้างวินโดว์นี้ได้โดยตรงจากฟังก์ชันของ OpenGL เนื่องจากไลบราลีนี้ประกอบเพียงฟังก์ชันทางด้านกราฟิกที่ไม่ขึ้นกับอุปกรณ์ใดๆ นอกจากนี้การจัดการเกี่ยวกับวินโดว์ขึ้นอยู่กับคอมพิวเตอร์ที่ใช้งานอยู่อีกด้วย อย่างไรก็ตามยังคงมีไลบราลีจำนวนหนึ่งที่สนับสนุนฟังก์ชันของ OpenGL สำหรับเครื่องที่แตกต่างกันไป ซึ่ง GLUT เป็นชุดเครื่องมือที่มีไลบราลีของฟังก์ชันสำหรับการใช้งานกับระบบวินโดว์ของจอภาพทั่วๆ ไป เนื่องจากการเขียนโปรแกรมโดยใช้ OpenGL มีความซับซ้อนน้อยลง เหมาะสำหรับการพัฒนาโปรแกรมขนาดเล็กถึงขนาดกลาง คำสั่งของ GLUT จะขึ้นต้นด้วยคำว่า glut เสมอ การเรียกใช้จะต้อง include ไฟล์ header ที่ชื่อ gult.h ในตอนต้นของโค้ดโปรแกรมเช่นกัน ถ้าในตอนต้นของโค้ดโปรแกรมเป็น glut.h แล้ว ไม่จำเป็นต้องใช้ gl.h และ glu.h อีก (ไพศาล โมลิสกุลมงคล, 2550)

2.4 กลศาสตร์การเคลื่อนไหวของหุ่นยนต์

กลศาสตร์การเคลื่อนไหวของหุ่นยนต์คือ การศึกษาการเคลื่อนไหว โดยวิเคราะห์จากตำแหน่งของหุ่นยนต์ ซึ่งจะคำนวณโดยปราศจากแรงที่ทำให้เกิดการเคลื่อนไหว การกำหนดให้แขนกลแต่ละส่วนมีความสัมพันธ์กับแต่ละเมตริกซ์ที่มีลักษณะเฉพาะและเชื่อมโยงไปสู่ข้อต่อต่างๆ ที่ได้รับผลกระทบ เรียกอีกอย่างว่ากลศาสตร์การเคลื่อนไหวโดยตรง มีการคำนวณตำแหน่งของจุดต่างๆ ในการทำงานของหุ่นยนต์ กลศาสตร์การเคลื่อนไหวหุ่นยนต์สามารถแบ่งได้เป็น กลศาสตร์การเคลื่อนไหวหุ่นยนต์แบบอนุกรม กลศาสตร์การเคลื่อนไหวหุ่นยนต์แบบขนาน กลศาสตร์การเคลื่อนไหวหุ่นยนต์เคลื่อนที่ และกลศาสตร์การเคลื่อนไหวที่เลียนแบบมนุษย์

2.4.1 Transformation matrix

ในการออกแบบหุ่นยนต์ ต้องมีการกำหนดโครงร่างในแต่ละข้อต่อของหุ่นยนต์ และในแต่ละวัตถุของหน่วยงาน ดังนั้นการเปลี่ยนแปลงของโครงร่างเป็นแนวคิดพื้นฐานในการสร้างแบบจำลองและการเขียนโปรแกรมของหุ่นยนต์ ซึ่งจะมีส่วนช่วยดังนี้

- คำนวณที่ตั้ง ตำแหน่ง และการกำหนดทิศทางของหุ่นยนต์
- อธิบายถึงตำแหน่งและการกำหนดทิศทางของหุ่นยนต์

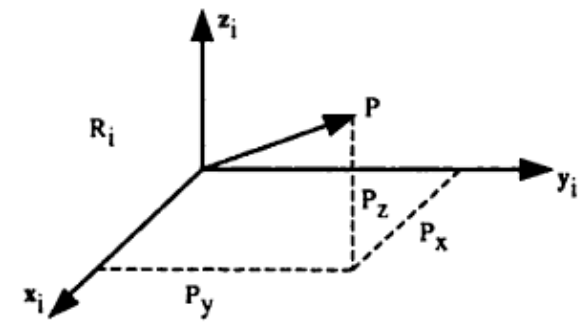
- ระบุเส้นทางการเคลื่อนที่และความความเร็วของ End – effector ของหุ่นยนต์ สำหรับงานที่ต้องการ
- อธิบายและควบคุมแรง เมื่อหุ่นยนต์มีการตอบสนองกับสภาพแวดล้อม
- ดำเนินการควบคุมทางประสาทสัมผัสที่ใช้ โดยใช้ข้อมูลจากเซ็นเซอร์ต่างๆ และกรอบอ้างอิงของตัวมันเอง

2.4.2 พิกัดเดี่ยว (Homogeneous coordinates)

2.4.2.1 ตัวแทนของจุด (Representation of a point)

ให้ $({}^{i}P_x, {}^{i}P_y, {}^{i}P_z)$ เป็นพิกัดคาร์ทีเซียนของจุด P ในกรอบ R_i ซึ่งจากรูปที่ 2.7 ได้บอกถึงจุดกำเนิด O_i และแกน x_i, y_i, z_i พิกัดเดี่ยวของ P ในกรอบของ R_i ที่ถูกกำหนดด้วย $(w{}^{i}P_x, w{}^{i}P_y, w{}^{i}P_z, w)$ เมื่อ w คือ ปัจจัยการปรับขนาด (Scaling Factor) ในการออกแบบหุ่นยนต์จะให้ w มีค่าเท่ากับ 1 ดังนั้นเราสามารถเขียนพิกัดของจุด P ได้ดังนี้

$$ {}^{i}P = \begin{bmatrix} {}^{i}P_x \\ {}^{i}P_y \\ {}^{i}P_z \\ 1 \end{bmatrix} \tag{2.1} $$



รูปที่ 2.7 ตัวแทนของเวคเตอร์ของจุด

2.4.2.2 ตัวแทนของทิศทาง (Representation of a direction)

ทิศทางของเวคเตอร์ใดๆ จะถูกแทนด้วยส่วนประกอบ 4 ส่วน ในส่วนที่สี่จะมีค่าเป็น 0 (ศูนย์) ซึ่งแสดงถึงเวคเตอร์ที่อินฟินิตี้ สมมติให้พิกัดคาร์ทีเซียนของหนึ่งหน่วยของเวคเตอร์ ในกรอบของ R_i เป็น $({}^{i}u_x, {}^{i}u_y, {}^{i}u_z)$ ซึ่งสามารถเขียนได้เป็น

$$ {}^{i}u = \begin{bmatrix} {}^{i}u_x \\ {}^{i}u_y \\ {}^{i}u_z \\ 0 \end{bmatrix} \tag{2.2} $$

2.4.2.3 ตัวแทนของระนาบ (Representation of a plane)

พิกัดเดี่ยวของระนาบ Q ซึ่งสมการในกรอบของ R_i เป็น ${}^{i}\alpha x + {}^{i}\beta y + {}^{i}\gamma z + {}^{i}\delta = 0$ ที่กำหนดโดย

$$ {}^{i}Q = [{}^{i}\alpha \quad {}^{i}\beta \quad {}^{i}\gamma \quad {}^{i}\delta] \tag{2.3} $$

ถ้าจุด P อยู่บนระนาบ Q แล้ว เมตริกซ์ของผลลัพธ์ของ ${}^{i}Q\,{}^{i}P$ มีค่าเท่ากับ 0 (ศูนย์)

$$ {}^{i}Q\,{}^{i}P = [{}^{i}\alpha \quad {}^{i}\beta \quad {}^{i}\gamma \quad {}^{i}\delta] \begin{bmatrix} {}^{i}P_x \\ {}^{i}P_y \\ {}^{i}P_z \\ 1 \end{bmatrix} = 0 \tag{2.4} $$

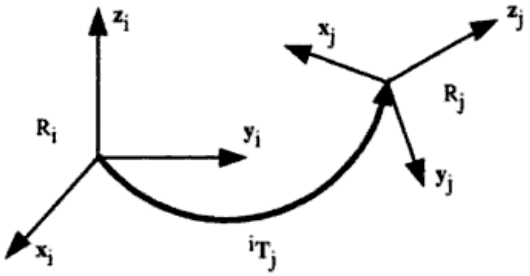
2.4.3 การเปลี่ยนแปลงเชิงเดี่ยว

2.4.3.1 การเปลี่ยนแปลงของกรอบ

การแทนที่ และ/หรือ การหมุนของกรอบ R_i ที่อยู่ในกรอบของ R_j ดังรูปที่ 2.8 ซึ่งสามารถเขียนเป็นเมตริกซ์การเปลี่ยนแปลงเชิงเดี่ยว ${}^{i}T_j$ มิติ 4×4 ดังนี้

$$ {}^{i}T_j = [{}^{i}s_j \quad {}^{i}n_j \quad {}^{i}a_j \quad {}^{i}P_j] = \begin{bmatrix} s_x & n_x & a_x & P_x \\ s_y & n_y & a_y & P_y \\ s_z & n_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5} $$

เมื่อ ${}^{i}s_j$, ${}^{i}n_j$ และ ${}^{i}a_j$ ถูกแทนค่าด้วยเวคเตอร์หนึ่งหน่วยตามแนวแกน x_j, y_j และ z_j ตามลำดับ ที่แสดงในกรอบ R_i และเมื่อ ${}^{i}P_j$ คือเวคเตอร์ที่เป็นตัวแทนของพิกัดของจุดกำเนิดของกรอบ R_j ที่แสดงในกรอบ R_i



รูปที่ 2.8 การเปลี่ยนแปลงของกรอบ

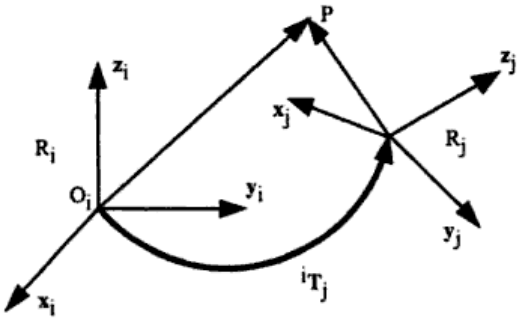
อีกทั้งยังสามารถกล่าวได้ว่าเมตริกซ์ ${}^{i}T_j$ กำหนดความสัมพันธ์ของกรอบ R_j กับกรอบ R_i หลังจากนั้นการเปลี่ยนแปลงของเมตริกซ์ในสมการที่ 5 ในบางครั้งจะถูกเขียนเป็นรูปแบบของพาร์ติชันเมตริกซ์ ดังนี้

$$ {}^{i}T_j = \begin{bmatrix} & {}^{i}A_j & & {}^{i}P_j \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^{i}s_j & {}^{i}n_j & {}^{i}a_j & {}^{i}P_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6} $$

2.4.3.2 การเปลี่ยนแปลงของเวคเตอร์

ให้เวคเตอร์ ${}^{j}P$ เป็นพิกัดเดี่ยวของจุด P ในกรอบ R_j ดังรูปที่ 4 ดังนั้นพิกัดเดี่ยวของ P ในกรอบของ R_i สามารถเขียนได้เป็น

$$ {}^{i}P = {}^{i}(O_iP) = {}^{i}s_j\,{}^{j}P_x + {}^{i}n_j\,{}^{j}P_y + {}^{i}a_j\,{}^{j}P_z + {}^{i}P_j = {}^{i}T_j\,{}^{j}P \qquad (2.7) $$

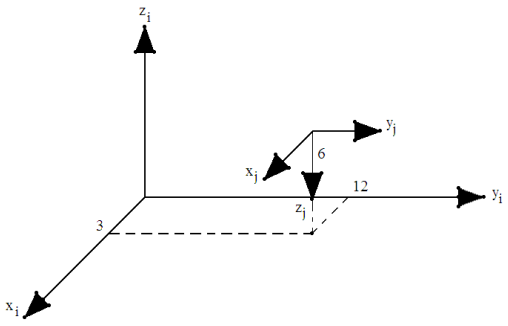


รูปที่ 2.9 การเปลี่ยนแปลงของเวคเตอร์

ดังนั้นเมตริกซ์ ${}^{i}T_j$ ช่วยให้เราสามารถคำนวณพิกัดของเวคเตอร์ในกรอบ R_i ในเทอมของพิกัดในกรอบ R_j

ตัวอย่างที่ 1 สมมติให้เมตริกซ์ ${}^{i}T_j$ และ ${}^{j}T_i$ จากรูปที่ 2.10 ใช้สมการที่ 5 จะได้เป็น

$$ {}^{i}T_j = \begin{bmatrix} 0 & 0 & 1 & 3 \\ 0 & 1 & 0 & 12 \\ -1 & 0 & 0 & 6 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^{j}T_i = \begin{bmatrix} 0 & 0 & -1 & 6 \\ 0 & 1 & 0 & -12 \\ 1 & 0 & 0 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$



รูปที่ 2.10 ตัวอย่างที่ 1

2.4.3.3 การเปลี่ยนแปลงของระนาบ

ตำแหน่งความสัมพันธ์ของจุดที่เกี่ยวกับระนาบจะไม่เปลี่ยนแปลงไปจากการนำไปใช้ของเซต {point, plane} ดังนั้น

$${}^{j}Q\,{}^{j}P = {}^{i}Q\,{}^{i}P = {}^{i}Q\,{}^{i}T_{j}\,{}^{j}P$$

จะได้

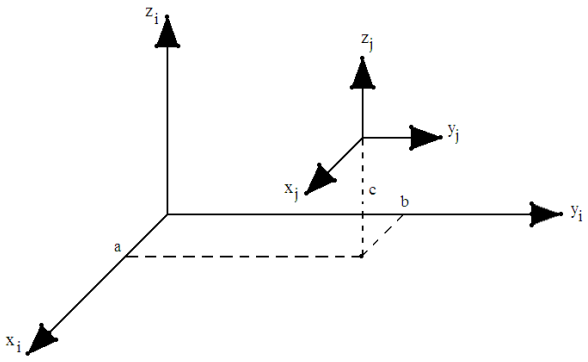
$${}^{j}Q = {}^{i}Q\,{}^{i}T_{j} \tag{2.8}$$

2.4.4 การเปลี่ยนแปลงเมตริกซ์ของการย้ายแบบ pure

ให้ **Trans**(a, b, c) เป็นการเปลี่ยนแปลงชนิดข้างต้น เมื่อ a, b และ c แสดงถึงการย้ายตามแนวแกน x, y และ z ตามลำดับ การเปลี่ยนแปลงของ **Trans**(a, b, c) จะแสดงดังรูปที่ 2.11

$${}^{i}T_{j} = \mathbf{Trans}(a, b, c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

ต่อจากนี้ เรายังคงใช้สัญลักษณ์ **Trans**(u, d) ในการแสดงการย้ายตามแกน u โดยค่า d ดังนั้นเมตริกซ์ **Trans**(a, b, c) สามารถแตกออกเป็นเมตริกซ์ 3 ส่วน คือ **Trans**(**x**, a) **Trans**(y, b) **Trans**(z, c) ซึ่งใช้ในการคูณใดๆ



รูปที่ 2.11 การเปลี่ยนแปลงของการย้ายแบบ pure

2.4.5 การแปลงเมตริกซ์ของการหมุนของแกน

2.4.5.1 การแปลงเมตริกซ์ของการหมุนของแกน x ด้วยมุม θ

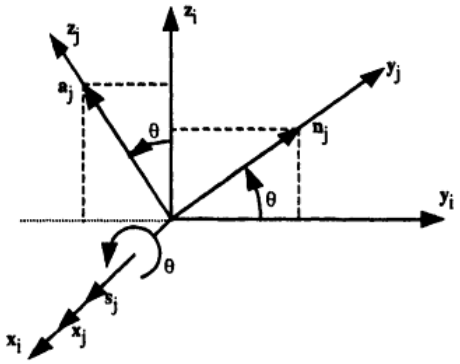
ให้ **Rot**(x, θ) เป็นการเปลี่ยนแปลงชนิดข้างต้น จากรูปที่ 2.12 เราสมมติว่าส่วนประกอบของเวคเตอร์ ${}^{i}s_j, {}^{i}n_j, {}^{i}a_j$ ขนาดหนึ่งหน่วย ตามแนวแกน x_j, y_j, z_j ของกรอบ R_j ที่แสดงในกรอบ R_i มีดังต่อไปนี้

$$\begin{cases} {}^{i}s_j = [1 \quad 0 \quad 0 \quad 0]^T \\ {}^{i}n_j = [0 \quad C\theta \quad S\theta \quad 0]^T \\ {}^{i}a_j = [0 \quad -S\theta \quad C\theta \quad 0]^T \end{cases} \tag{2.10}$$

เมื่อ $S\theta$ และ $C\theta$ คือ $\sin(\theta)$ และ $\cos(\theta)$ ตามลำดับ และ ตัวยก T เป็นการทรานสโพสของเวคเตอร์

$${}^{i}T_j = \mathbf{Rot}(x,\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\theta & -S\theta & 0 \\ 0 & S\theta & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & & & 0 \\ & \mathbf{rot}(x,\theta) & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

เมื่อ $\mathbf{rot}(x,\theta)$ เป็นเมตริกซ์ มิติ 3×3



รูปที่ 2.12 การแปลงของการหมุนของแกน x

2.4.5.2 การแปลงเมตริกซ์ของการหมุนของแกน y ด้วยมุม θ

ด้วยวิธีเดียวกับแกน x จะได้เป็น

$${}^{i}T_j = \mathbf{Rot}(y,\theta) = \begin{bmatrix} C\theta & 0 & S\theta & 0 \\ 0 & 1 & 0 & 0 \\ -S\theta & 0 & C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & & & 0 \\ & \mathbf{rot}(y,\theta) & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

2.4.5.3 การแปลงเมตริกซ์ของการหมุนของแกน z ด้วยมุม θ

ด้วยวิธีเดียวกับแกน x จะได้เป็น

$$ {}^{i}T_j = \mathbf{Rot}(z,\theta) = \begin{bmatrix} C\theta & -S\theta & 0 & 0 \\ s\theta & C\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & \mathbf{rot}(z,\theta) & & \begin{matrix}0\\0\\0\end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.13) $$

(Khali and Dombre,2002)

2.5 รูปแบบของแฟ้มข้อมูล Ply

Ply file format (Polygon File Format) เป็นรูปแบบการเก็บข้อมูลสามมิติ โดยในวัตถุหนึ่งจะเป็นความสัมพันธ์ของโครงตาข่ายรูปสามเหลี่ยม ซึ่งข้อมูลดังกล่าวประกอบด้วยส่วนของ Header ตามด้วยส่วนของจุดยอด และส่วนของรูปหลายเหลี่ยม ส่วนของ header จะระบุจำนวนพิกัด และจำนวนระนาบของโครงตาข่ายรูปสามเหลี่ยม และยังบอกถึงคุณสมบัติของแต่ละจุดยอด เช่น พิกัด x, y, z จากนั้นสามารถนำข้อมูลดังกล่าวไปใช้ในการเขียนโปรแกรมเพื่อแสดงรูปทรงสามมิติใน OpenGL Library (Georgia Institute of Technology) ตัวอย่าง Ply file format มีดังนี้

```
ply
format ascii 1.0
comment VCGLIB generated
element vertex 76
property float x
property float y
property float z
element face 148
property list uchar int vertex_indices
end_header
```

ในส่วนนี้เป็นการบอกข้อมูลของรูปทรงที่ทำการแปลงไฟล์เป็น PLY ได้แก่ จำนวนพิกัด โดยแบ่งเป็น x, y และ z และจำนวนระนาบ

```
-65.25 0 -4.7
65.25 0 -4.7
-65.25 0.39859 -4.6830681
65.25 0.39859 -4.6830681
        :
65.25 1.1843 -4.548345
```

ในส่วนนี้เป็นการแสดงให้เห็นว่า มีตัวเลข 3 กลุ่ม มีการเรียงเป็น x y z ตามลำดับ

```
3 67 0 1
3 1 0 3
3 0 2 3
```

ในส่วนนี้เป็นการแสดงให้เห็นว่า ตัวเลข 4 กลุ่ม กลุ่มแรกคือ เลข 3 หมายความว่า เป็นรูปสามเหลี่ยมที่มีจุดทั้งสามตัวหลังเป็นจุดมุม

3 3 2 66

:

3 3 66 4

2.6 การทบทวนวรรณกรรมที่เกี่ยวข้อง (Literature review)

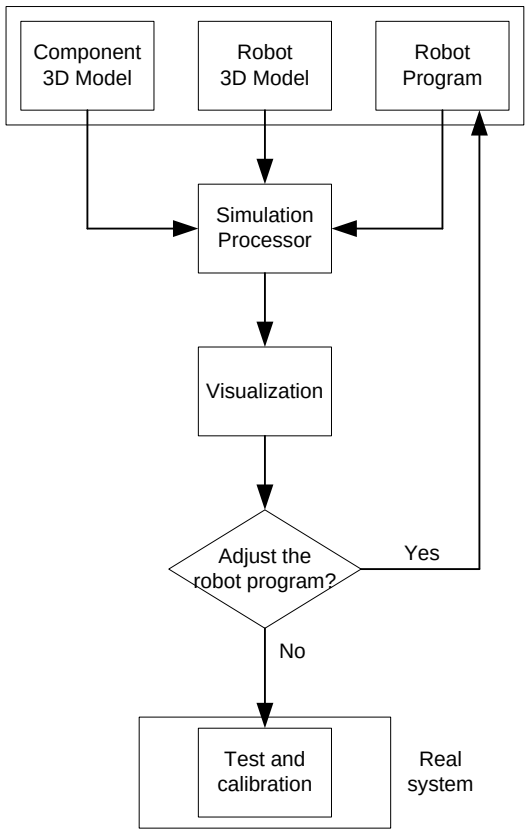
การจำลองการทำงานของหุ่นยนต์เป็นหัวข้องานวิจัยที่ได้รับความสนใจเพิ่มขึ้นอย่างมากในระยะหลังเนื่องจากความจำเป็นในการใช้ระบบการผลิตแบบยืดหยุ่น (Flexible Manufacturing Systems หรือ FMS) เพื่อเพิ่มผลผลิตให้เพียงพอต่อความต้องการของตลาด หุ่นยนต์อุตสาหกรรมได้ถูกนำไปใช้งานในด้านต่างๆ อย่างกว้างขวาง [6] และการจำลองการทำงานของหุ่นยนต์ก็ได้ถูกพัฒนาไปในทิศทางต่างๆ ตามประเภทของงานที่ใช้

ซอฟต์แวร์สำเร็จที่มีอยู่ในปัจจุบัน เช่น Roboguide, RoboSim, MotoSim ฯลฯ มีทั้งแบบที่เป็นซอฟต์แวร์ที่สามารถใช้ได้กับหุ่นยนต์ชนิดต่างๆ และแบบที่เป็นของบริษัทผู้ผลิตหุ่นยนต์เองซึ่งสามารถแสดงการทำงานของหุ่นยนต์ในเชิงภาพเคลื่อนไหวสามมิติและสามารถใช้กับงานในอุตสาหกรรมได้หลากหลาย ทำให้ผู้ใช้สามารถทดลองการทำงานของเซลล์หุ่นยนต์ (Robotic cell) ได้ก่อนการตัดสินใจลงทุน ซึ่งเป็นการเพิ่มทางเลือกในการออกแบบผังโรงงาน ช่วยลดค่าใช้จ่ายและเวลาในการติดตั้ง ตลอดจนการเพิ่มประสิทธิภาพการทำงานของระบบให้สูงขึ้น

การจำลองการทำงานของหุ่นยนต์ได้ถูกนำไปประยุกต์ใช้งานในการหาเส้นทางการเคลื่อนที่ของหุ่นยนต์ [1] โดยการจำลองเส้นทางการเคลื่อนที่ (Path Simulation) ที่อาศัยคอมพิวเตอร์ช่วยในการออกแบบ (Computer-aided design หรือ CAD) ซึ่งคณะผู้วิจัยได้ทดลองกับระบบขนถ่ายวัสดุและผลการทดลองแสดงให้เห็นถึงความง่ายและความยืดหยุ่นในการนำการจำลองการทำงานของหุ่นยนต์มาใช้ ตัวอย่างการประยุกต์ใช้งานอื่นที่นำซอฟแวร์จำลองการทำงานของหุ่นยนต์มาใช้ได้แก่การจำลองการเชื่อมด้วยหุ่นยนต์ที่นำเสนอใน [7] ซึ่งใช้การจำลองการทำงานของหุ่นยนต์ร่วมกับการสร้างโปรแกรมของหุ่นยนต์แบบอัตโนมัติ การประยุกต์ใช้งานอื่นๆ ได้แก่ การใช้การจำลองการทำงานของหุ่นยนต์ในการออกแบบเซลล์การทำงานสำหรับอุตสาหกรรมอากาศยาน [2] เพื่อช่วยในการวิเคราะห์การเข้าถึงของตำแหน่งต่างๆ บนส่วนลำตัวเครื่องบิน

บทที่ 3
วิธีการดำเนินวิจัย

การจำลองการทำงานของหุ่นยนต์ประกอบด้วยขั้นตอนหลักๆ ดังแสดงไว้ในรูปที่ 3.1 โดยในส่วนแรกเป็นการรับค่าข้อมูลซึ่งประกอบด้วย 1) แบบสามมิติของวัตถุและอุปกรณ์ที่ใช้ในการจำลองการทำงาน 2) แบบสามมิติของหุ่นยนต์ และ 3) โปรแกรมหุ่นยนต์ โดยแบบสามมิติในส่วนที่ 1) และ 2) ได้จากการนำวัตถุอุปกรณ์และชิ้นส่วนของหุ่นยนต์ไปทำการวัดและเขียนแบบด้วยโปรแกรมคอมพิวเตอร์ช่วยในการออกแบบ (CAD) ส่วนโปรแกรมในส่วนที่ 3) เป็นโปรแกรมที่ผู้ใช้ป้อนเข้าเพื่อจำลองการทำงานของหุ่นยนต์ หลังจากนั้น ข้อมูลจะถูกประมวลผลโดย Simulation Processor เพื่อคำนวณตำแหน่งและเส้นทางการเคลื่อนที่ของหุ่นยนต์ตามโปรแกรมหุ่นยนต์ที่ผู้ใช้ป้อนเข้า หลังจากนั้น ผลที่ได้จากการคำนวณจะถูกนำไปสร้างเป็นภาพสามมิติโดยใช้คลังโปรแกรมจาก OpenGL โดยผู้ใช้จะเป็นผู้ตรวจสอบผลลัพธ์ที่ได้และทำการแก้ไขโปรแกรมหุ่นยนต์ก่อนที่จะนำไปเชื่อมต่อกับระบบจริง

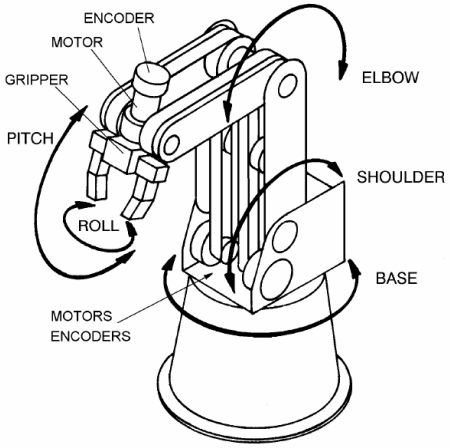


รูปที่ 3.1 แผนผังแสดงขั้นตอนการจำลองการทำงานของหุ่นยนต์

3.1 การทำงานของแขนกลรุ่น SCORBOT-ER 4u

SCORBOT – ER 4u เป็นหุ่นยนต์ ชนิด Articulated Arm (Revolute) สำหรับใช้ในหยิบจับวัตถุ ซึ่งมีรูปลักษณ์ดังแสดงไว้ในรูปที่ 3.2 ตัวหุ่นยนต์มีขนาดเล็กและสามารถติดตั้งบนฐานที่เคลื่อนที่ได้ ซึ่งเหมาะสำหรับการใช้งานเพื่อการศึกษาและจำลองการทำงานใน workcell อัตโนมัติ เช่น หุ่นยนต์ที่ใช้เชื่อม หุ่นยนต์ที่ใช้งานร่วมกับเครื่อง CNC และการดำเนินงานอื่นๆ โดยมีโปรแกรม SCORBASE ในการควบคุมการทำงานของแขนกล SCORBOT – ER 4u

การจำลองการทำงานของแขนกลจะเริ่มจากการวัดขนาดชิ้นส่วนของแขนกลจริง เพื่อใช้ในการเขียนแบบชิ้นส่วนด้วยโปรแกรม SolidWorks และศึกษาของเขตการเคลื่อนที่ของแขนกล โดยแต่ละแกนและขอบเขตการเคลื่อนที่ของแขนกล SCORBOT – ER 4u ได้แสดงในตารางที่ 3.1



รูปที่ 3.2 แขนกลรุ่น SCORBOT – ER 4u

ตารางที่ 3.1 ข้อมูลจำเพาะของแขนกล

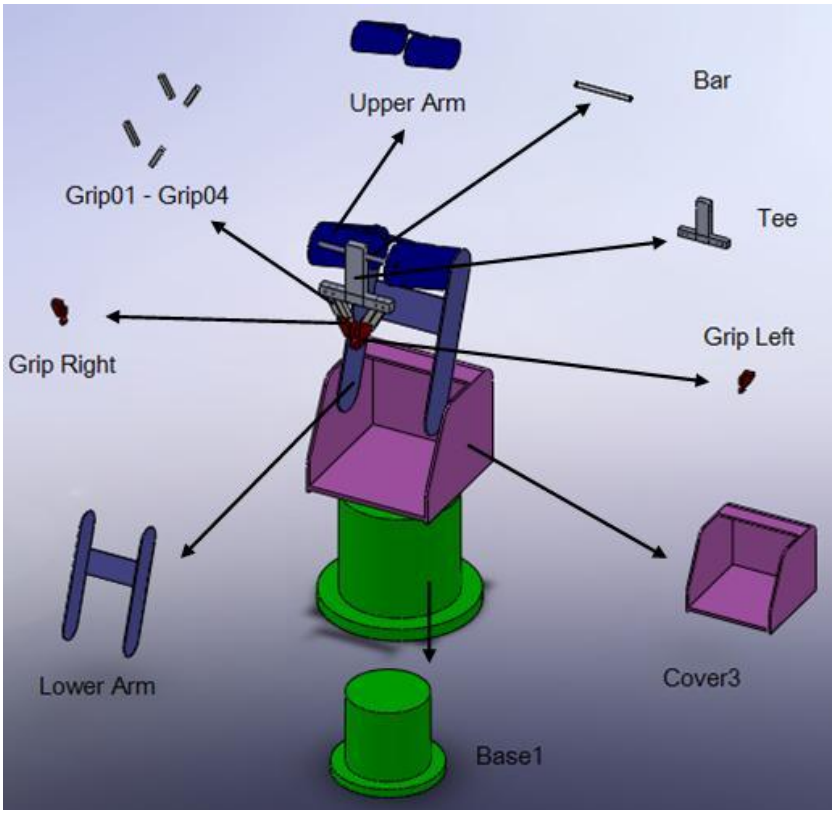
ลักษณะเชิงกล	โครงสร้างแนวตั้งแบบข้อต่อ โครงร่างแบบเปิด
องศาความเป็นอิสระ	มี 5 แกน และ gripper ที่สามารถหมุนได้
รับน้ำหนักได้	2.1 kg (4.6 lb)
ช่วงแกน	
แกนที่ 1:	Base หมุนได้ 310°
แกนที่ 2:	Shoulder หมุนได้ 130°/ -35°
แกนที่ 3:	Elbow หมุนได้ 130°
แกนที่ 4:	Wrist (ข้อมือ) มีองศากวาด (Pitch) เท่ากับ 130°
แกนที่ 5:	หมุนข้อต่อ (ROLL) Wrist ± 570
ระยะยืดจาก gripper	610 mm (24")
ความเร็ว	700 mm/s (27.6 inch/sec)

การทำซ้ำ	± 0.18 mm (0.007 inch)
การกลับไปยังตำแหน่งเดิม	การอ่านตำแหน่ง แบบที่บวกเพิ่มจากตำแหน่งเดิม
Gripper Jaw เปิด	0 - 65/75 mm (2.6"/ 3") ทั้งที่มีและไม่มีชิ้นงาน
ตัวส่งกำลัง	เฟือง สายพาน สกรู
น้ำหนัก	10.8 kg (23.8 lb)

3.2 การสร้างแบบสามมิติและเก็บข้อมูลของแขนกล

ขนาดชิ้นส่วนต่างๆ ของแขนกลได้วัดได้จะนำไปใช้ในการเขียนแบบด้วยโปรแกรม SolidWorks แล้วแปลงให้เป็นไฟล์รูปแบบ PLY เพื่อให้เป็นโครงร่างตาข่ายรูปสามเหลี่ยม ซึ่งมีการบอกพิกัดและระนาบของรูปสามเหลี่ยมที่ประกอบกันเป็นรูปทรงสามมิติของแต่ละชิ้นส่วน

ชิ้นส่วนหลักของแขนกล SCORBOT – ER 4u ประกอบด้วยชิ้นส่วนทั้งหมด 12 ชิ้น คือ Base1, Cover3, Lower Arm, Upper Arm, Bar, Tee, Grip01, Grip02, Grip03, Grip04, Grip Left และ Grip Right ดังรูปที่ 3.3

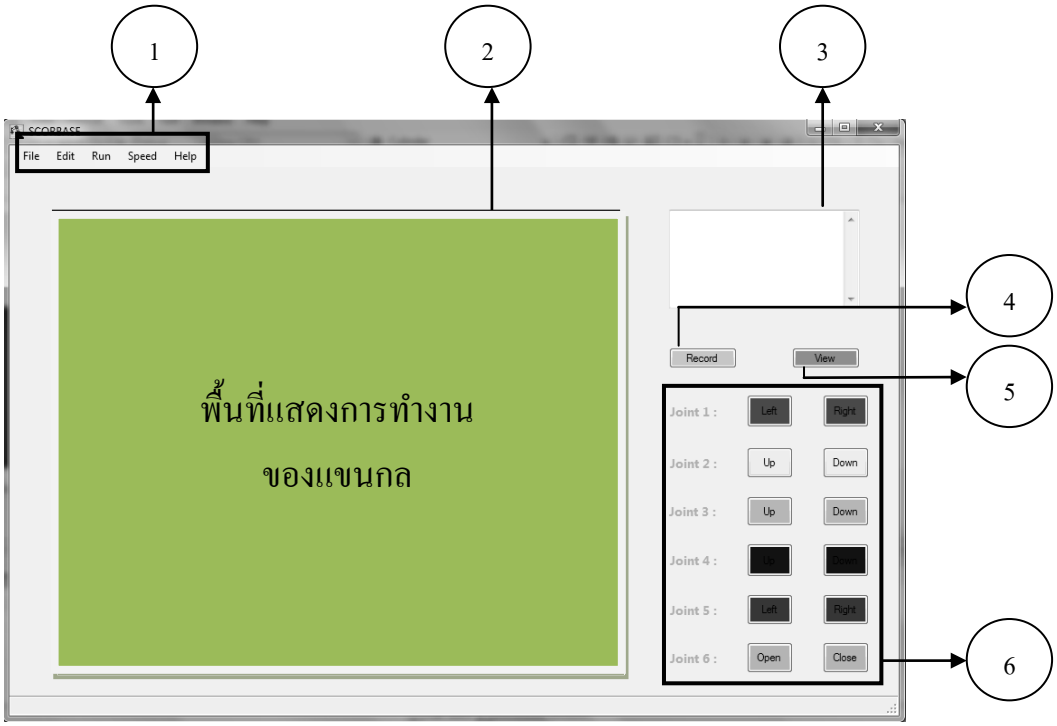


รูปที่ 3.3 ชิ้นส่วนหลักของแขนกล SCORBOT – ER 4u

3.3 การออกแบบโปรแกรม

3.3.1 ออกแบบส่วนติดต่อกับผู้ใช้ (User Interface)

การติดต่อกับผู้ใช้งานหรือ User Interface เป็นส่วนที่ผู้ใช้จะเข้าถึงการทำงานของโปรแกรม รวมไปถึงการแสดงรูปสามมิติของแขนกลในขณะทำงาน ซึ่งมีส่วนประกอบสำคัญอันประกอบไปด้วยปุ่มควบคุมการเคลื่อนที่หน้าต่างบันทึกตำแหน่งที่เปลี่ยนไป และเรียกดูการเคลื่อนที่จากคำสั่งทั้งหมดได้ ส่วนประกอบของหน้าจอ User Interface แสดงดังรูปที่ 3.4 โดยมีรายละเอียดดังนี้

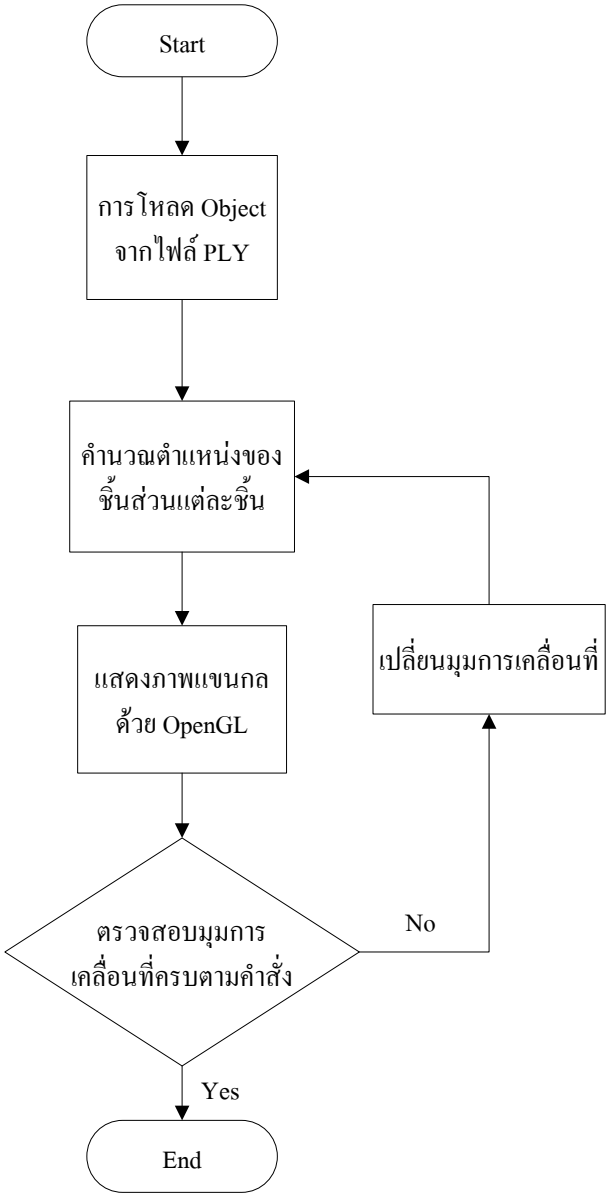


รูปที่ 3.4 หน้าจอส่วนติดต่อกับผู้ใช้งาน

1. แถบเมนู เป็นส่วนที่เก็บคำสั่งทั้งหมดของโปรแกรม เช่นการบันทึกไฟล์ สั่งรันโปรแกรม และกำหนดความเร็วในการเคลื่อนที่ของแขนกล
2. OpenGLControl เป็นส่วนที่แสดงภาพเคลื่อนไหวของแขนกล
3. หน้าต่างบันทึกตำแหน่ง เป็นส่วนที่แสดงตำแหน่งที่เปลี่ยนไปของการบันทึกแต่ละครั้ง
4. ปุ่มบันทึกคำสั่ง เป็นปุ่มที่ทำให้เกิดการจดจำตำแหน่งโดยจะแสดงอยู่ในหน้าต่างบันทึกตำแหน่ง
5. ปุ่มเลือกมุมมอง เป็นปุ่มที่ใช้เลือกมุมมองของแขนกลไม่ว่าจะเป็นด้านหน้า ด้านข้างหรือด้านหลัง
6. ปุ่มควบคุมแขนกล เป็นปุ่มที่ใช้ในการควบคุมแขนกล แบ่งออกเป็น 6 Joints

3.3.2 ขั้นตอนการทำงานของโปรแกรม

การทำงานของโปรแกรม เริ่มจากการโหลดข้อมูลจากไฟล์ PLY มาเก็บไว้ในออบเจ็กต์ที่สร้างไว้ และคำนวณตำแหน่งของแต่ละชิ้นส่วนว่าอยู่ตำแหน่งใดของแขนกล โดยเรียงลำดับจากชิ้นส่วนที่อยู่ด้านปลายของแขนกลลงมาที่ฐานของแขนกล จากนั้นจึงแสดงภาพโดยการนำจุดแต่ละจุดมาต่อกันเป็นระนาบสามเหลี่ยมและนำระนาบสามเหลี่ยมมาต่อกันเป็นภาพสามมิติ ในการแสดงภาพเคลื่อนไหว โปรแกรมจะตรวจสอบมุมการเคลื่อนที่ของแขนกลแต่ละชิ้น และทำการแบ่งมุมการเคลื่อนที่ออกเป็นช่วงเล็กๆ เพื่อให้ภาพเคลื่อนไหวที่ได้มีความต่อเนื่อง ขั้นตอนการทำงานของโปรแกรมแสดงดังรูปที่ 3.5



รูปที่ 3.5 แผนภูมิการทำงานของโปรแกรม

3.3.3 ฟังก์ชันการทำงานของโปรแกรม

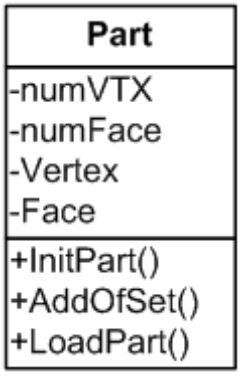
โปรแกรมจำลองการทำงานของแขนกลมีฟังก์ชันการดึงข้อมูล เพื่อดึงข้อมูลไฟล์ PLY มาเก็บไว้ในออบเจ็กต์ ฟังก์ชันการคำนวณตำแหน่งของแขนกล เพื่อคำนวณตำแหน่งของชิ้นส่วนแต่ละชิ้นก่อนการเคลื่อนที่ ฟังก์ชันการคำนวณการเคลื่อนที่ของแขนกล เพื่อคำนวณการเคลื่อนที่โดยการนำพิกัดคูณกับทรานฟอเมชันเมตริกซ์ ฟังก์ชันการเพิ่มความเร็ว เพื่อเพิ่มหรือลดความเร็วในการเคลื่อนไหว ฟังก์ชันการแสดงภาพการเคลื่อนไหว โดยแบ่งภาพการเคลื่อนไหวเป็น 150 ภาพต่อคำสั่ง เพื่อใช้ภาพเคลื่อนไหวมีความต่อเนื่อง และฟังก์ชันการเปิดหรือบันทึกข้อมูล เพื่อเก็บข้อมูลตำแหน่งการเคลื่อนไหวที่ต้องการ หรือเปิดคำสั่งที่ได้บันทึกไว้

เมื่อสามารถแสดงรูปสามมิติได้แล้ว ในส่วนการควบคุมเช่น คำสั่งที่ใช้สำหรับการเคลื่อนที่ของแขนกลจะต้องสามารถคำนวณตำแหน่งที่แขนกลเคลื่อนที่ไปซึ่งใช้สำหรับการแสดงภาพสามมิติ คำสั่งที่ใช้สำหรับการเก็บค่าตำแหน่งเป็นลำดับ เพื่อนำค่าตำแหน่งที่ได้ไปแสดงภาพสามมิติของขั้นตอนการทำงานอย่างเป็นลำดับ

3.3.3.1 การสร้างออบเจ็กต์จากไฟล์ PLY

การสร้างออบเจ็กต์ของชิ้นส่วนแขนกลสามารถทำได้ โดยการโหลดข้อมูลตำแหน่งและพื้นผิวของแต่ละชิ้นส่วนจากไฟล์ PLY โดยใช้ Part ซึ่งเป็นคลาสที่ใช้เก็บคุณสมบัติต่างๆ ของชิ้นส่วน ได้แก่ ค่าพิกัด (Vertex) พื้นผิว (Face) จำนวนพิกัด (numVTX) จำนวนพื้นผิว (numFace) และตำแหน่งของชิ้นส่วนนั้น (EndCo) ซึ่งมีค่าเป็น Vertex, Face, numVTX, numFace, EndCo และมีเมธอดที่ใช้สร้างออบเจ็กต์เริ่มต้นตำแหน่งของชิ้นส่วน (InitPart ()) และจัดข้อมูลของชิ้นส่วนลงในอาร์เรย์ (Array) ซึ่งมีค่าเป็น

- InitPart() ทำหน้าที่สร้างออบเจ็กต์เริ่มต้นตำแหน่งของชิ้นส่วน
- AddOfSet() ทำหน้าที่เก็บตำแหน่งของชิ้นส่วน
- LoadPart() ทำหน้าที่เก็บค่าพิกัด และพื้นผิวของชิ้นส่วน



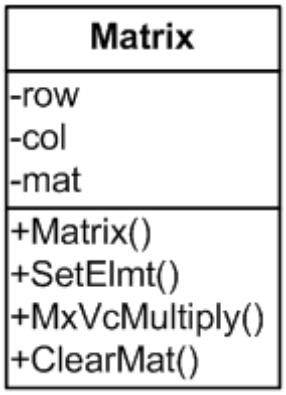
รูปที่ 3.6 คลาสของ ชิ้นส่วนของแขนกล

ออบเจ็กต์แต่ละชิ้นมีคุณสมบัติและหน้าที่เหมือนกัน โดยจะใช้ Coordinate เป็นโครงสร้างของการเก็บค่าพิกัดและพื้นผิว ในการสร้างแขนกลจึงเป็นการนำออบเจ็กต์ของแต่ละชิ้นส่วนมาเรียงต่อกัน

3.3.3.2 การคำนวณตำแหน่งชิ้นส่วนของแขนกล

การคำนวณตำแหน่งของแขนกลเริ่มจากการรับข้อมูลตำแหน่งจากผู้ใช้ ส่งข้อมูลไปยังฟังก์ชันการคำนวณและมุมที่เปลี่ยนไปจะถูกคำนวณโดยใช้ทรานฟอร์เมชันเมตริกซ์ เพื่อเปลี่ยนตำแหน่งของแขนกล หลังจากนั้นจึงสั่งให้ฟังก์ชันการแสดงภาพแสดงการเคลื่อนไหว ซึ่งมีคุณสมบัติเก็บค่าจำนวนแถว (row) จำนวนหลัก (col) และค่าต่างๆ และมีเมธอดต่างๆ ดังนี้

- Matrix() ทำหน้าที่สร้างเมตริกซ์ขึ้นมา
- SetElmt() ทำหน้าที่กำหนดค่าในเมตริกซ์
- MxVcMultiply() หน้าที่นำเมตริกซ์คูณกับเวคเตอร์
- ClearMat() ทำหน้าที่กำจัดเมตริกซ์ที่ใช้เรียบร้อยแล้วให้เป็นเมตริกซ์ศูนย์



รูปที่ 3.7 คลาสของเมตริกซ์

โดยที่

$[x, y, z]^T$ = ตำแหน่งพิกัดเดิม

$[x', y', z']^T$ = ตำแหน่งพิกัดใหม่

การหมุนรอบแกน x

$$[rot(x, \theta)] \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \quad (3.1)$$

การหมุนรอบแกน y

$$[\mathrm{rot}(y,\theta)]\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}x'\\y'\\z'\end{bmatrix} \quad (3.2)$$

การหมุนรอบแกน z

$$[\mathrm{rot}(z,\theta)]\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}x'\\y'\\z'\end{bmatrix} \quad (3.3)$$

กำหนดให้ θ เป็นอินพุตที่รับค่าจากการกดปุ่ม ซึ่งการกดปุ่ม UP 1 ครั้ง จะบวกมุมไป 10 องศา และการกด Down 1 ครั้ง จะลบมุมไป 10 องศา จากนั้นเมื่อคำนวณเสร็จ จะนำค่า x', y', z' ที่ได้ไปแสดงภาพ

การหาตำแหน่งของชิ้นส่วนแต่ละชิ้นทำได้โดยการนำพิกัดที่ระบุตำแหน่งของชิ้นส่วนมารวมกัน เช่น ถ้าต้องการต่อ Cover3 กับ Base1 พิกัดที่บอกตำแหน่งว่า Cover3 กับ Base1 ต่อกันตำแหน่งไหน จะอยู่ที่ Base1 ซึ่งเป็นตัวฐานของ Cover3 ส่วนการขยับแขนกลที่ต่อกันเสร็จเรียบร้อยแล้ว จะมีการคำนวณใหม่ทุกครั้งก่อนที่จะแสดงภาพ

3.3.4 การใช้ OpenGL ในการแสดงภาพการเคลื่อนไหวของแขนกล

การแสดงภาพการเคลื่อนไหวของแขนกลโดยใช้ OpenGL เป็นการนำผลจากการคำนวณในส่วนฟังก์ชันการทำงานมาสร้างรูปสามมิติที่แสดงตำแหน่งปัจจุบันขณะที่มีการควบคุม และค่าตำแหน่ง ซึ่งเก็บเป็นลำดับมาแสดงภาพสามมิติเป็นขั้นตอน โดยใช้ OpenGL ในการแสดงภาพสามมิติ และการเคลื่อนที่ของแขนกล

การแสดงภาพเกิดจากการนำออบเจ็กต์ของแขนกล ซึ่งเกิดจากการนำตำแหน่งทุกจุดของแขนกลมาสร้างระนาบสามเหลี่ยมต่อกันบน OpenGLControl หรือฉากสีดำ โดยภาพที่ปรากฏบนหน้าจอแสดงภาพเป็นท่าเริ่มต้น จากนั้นจะเป็นการแสดงภาพการเคลื่อนที่ โดยตำแหน่งที่เปลี่ยนแปลงไปจะถูกคำนวณแล้วนำมาแสดงใหม่ แล้วลบภาพก่อนหน้านั้นออก ทำอย่างนี้ซ้ำไปเรื่อยๆ ก็จะเกิดเป็นการเคลื่อนไหวของแขนกล

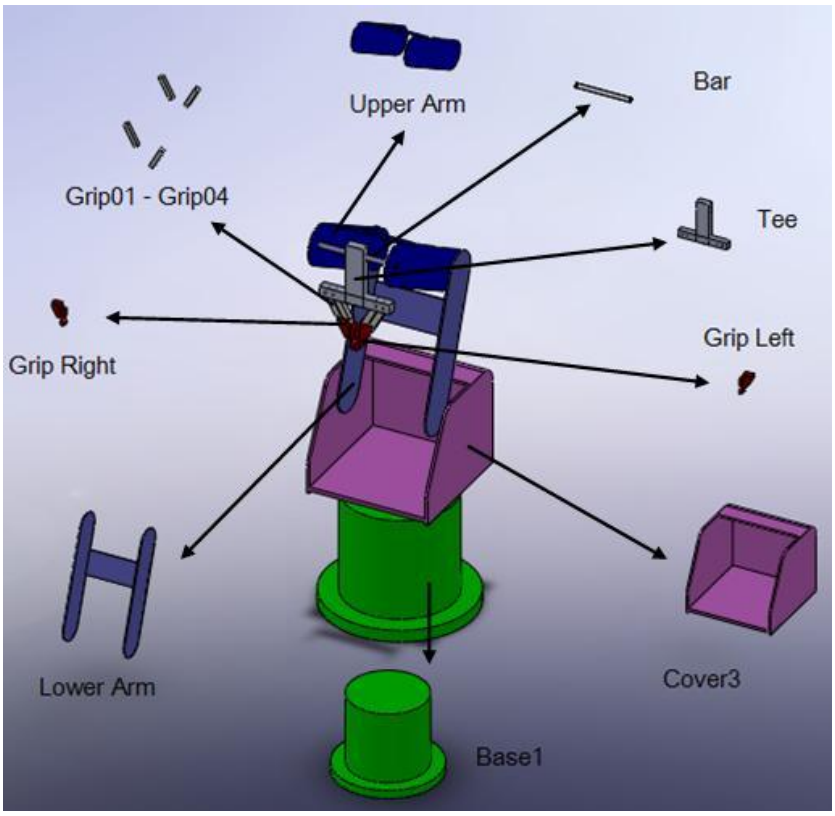
3.3.4.1 การกำหนดสี แสง ตำแหน่งและมุมมองของแขนกล

การกำหนดสีของแต่ชิ้นส่วน ทำเพื่อช่วยให้ง่ายต่อการควบคุม โดยกำหนดให้ชิ้นส่วนที่ขยับต่างกันมีสีต่างกัน เช่น ให้ Cover3 มีสีม่วง แต่ให้ Lower Arm มีสีเหลือง เพราะการเคลื่อนที่เป็นคนละทิศทางกัน ส่วนการกำหนดแสง ทำเพื่อให้เห็นรูปลักษณ์ของแขนกลเป็นแบบ 3 มิติ และให้เข้าใจง่ายต่อการมองแขนกลขณะเคลื่อนที่ โดยจะกำหนดให้แสงฉายจากด้านบนของแขนกล เพื่อให้เห็นภาพโดยรวม และเกิดเงาน้อยที่สุด และการกำหนดตำแหน่งและมุมมองของแขนกลทำเพื่อให้สามารถมองเห็นแขนกลในทิศทางอื่นได้ แม้ว่าจะเป็นการทำงานแบบเดียวกัน เช่น ต้องการดูการเคลื่อนที่จากด้านหลังของแขนกล เป็นต้น สามารถทำได้โดยการกดปุ่ม View เพื่อเปลี่ยนมุมมอง

บทที่ 4
ผลการวิจัย

4.1 การสร้างแบบสามมิติ

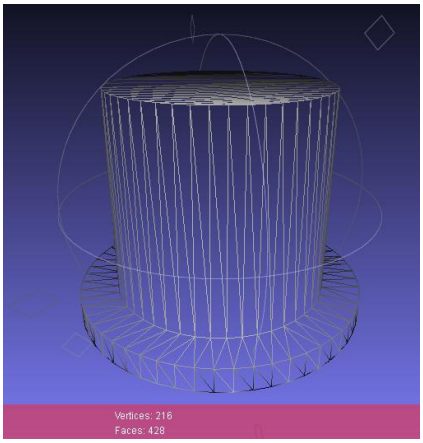
ข้อมูลที่ได้จากการวัดขนาดของแขนกลจะถูกนำไปเขียนแบบด้วยโปรแกรม SolidWorks รูปที่ 4.1 แสดงถึงส่วนประกอบหลักของแขนกล



รูปที่ 4.1 ส่วนประกอบของแขนกลที่เขียนด้วยโปรแกรม SolidWorks

4.2 การเก็บข้อมูลของแขนกล

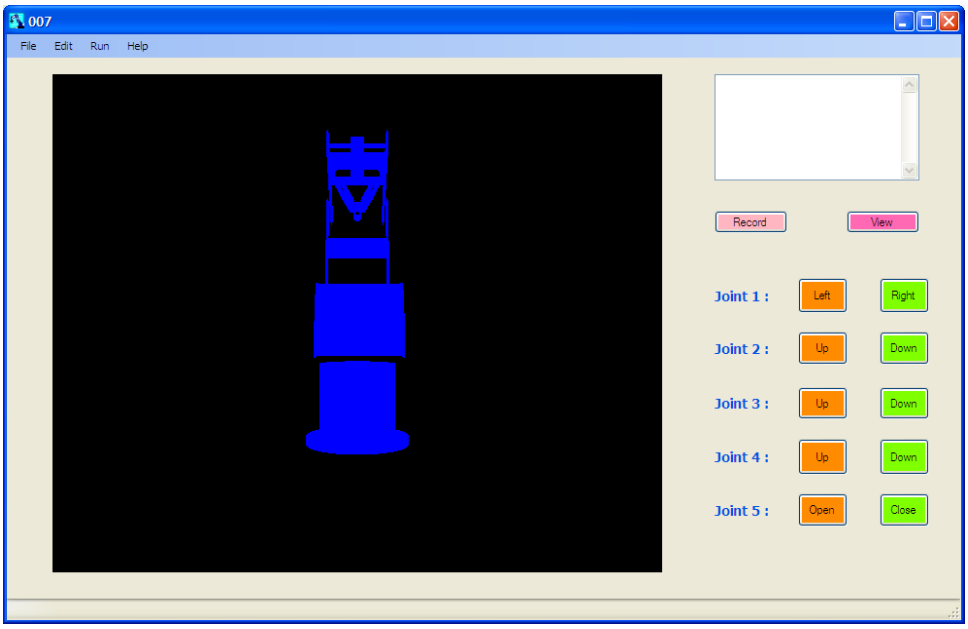
เมื่อวาดส่วนประกอบของแขนกลด้วยด้วยโปรแกรม SolidWorks แล้วแปลงเป็นไฟล์ PLY ซึ่งส่วนประกอบจะกลายเป็นโครงร่างตาข่ายรูปสามเหลี่ยม ยกตัวอย่างส่วนประกอบของแขนกลดังรูปที่ 4.2



รูปที่ 4.2 ส่วนประกอบ base ของแขนกล ที่เป็นโครงร่างตาข่ายรูปสามเหลี่ยม

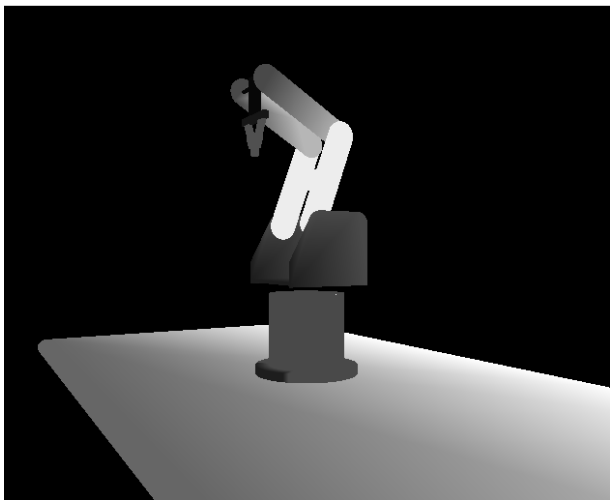
4.3 โปรแกรมจำลองการทำงานของแขนกล

เมื่อเขียนโปรแกรมเสร็จแล้ว หน้าต่างโปรแกรมจะแสดงการจำลองแขนกลในตำแหน่งเริ่มต้น (Home position) ดังรูปที่ 4.3 ในส่วนของโค้ด (Source code) ของโปรแกรม ผู้จัดทำได้แสดงไว้ในภาคผนวกของรายงานการวิจัยฉบับนี้



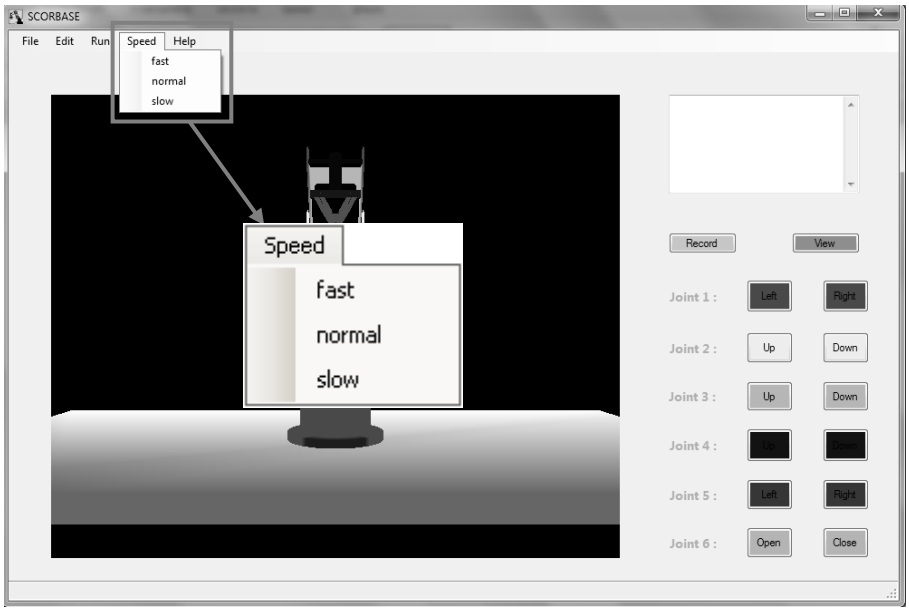
รูปที่ 4.3 โปรแกรมจำลองแขนกล

การเปลี่ยนมุมมองของการจำลอง โดยการกดปุ่ม View ดังรูปที่ 4.4



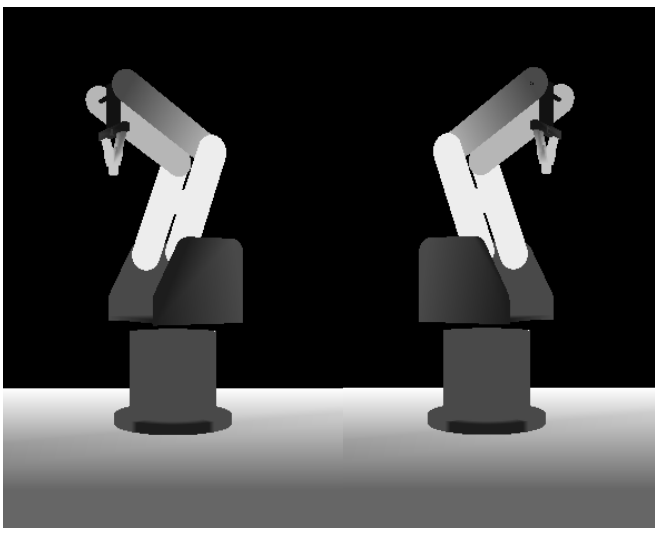
รูปที่ 4.4 แสดงมุมมองที่เปลี่ยนไปเมื่อกดปุ่ม View

การปรับความเร็วของการจำลอง ซึ่งมีให้เลือก 3 ระดับ นั่นคือ fast, Normal และ slow ดังแสดงในรูปที่ 4.5

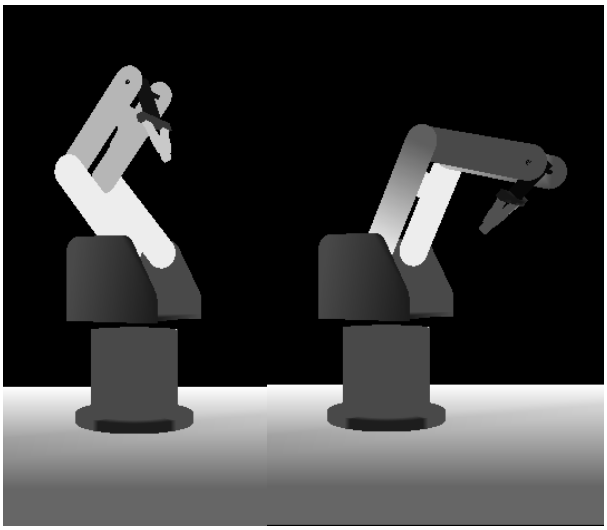


รูปที่ 4.5 แสดงขั้นตอนการเลือกความเร็ว

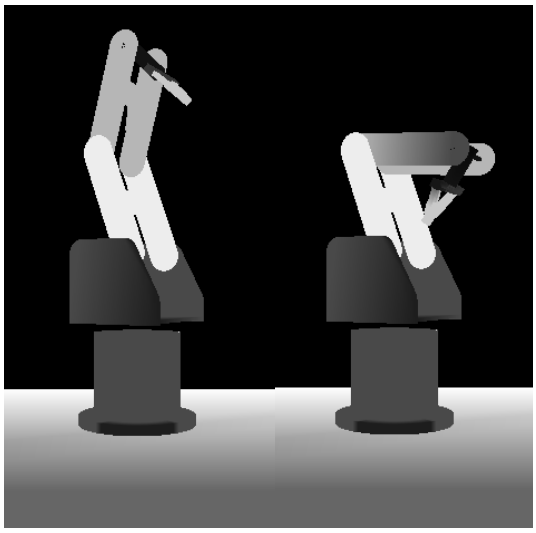
หรือกดปุ่มต่างๆ เพื่อให้หุ่นยนต์เคลื่อนที่ ดังรูปที่ 4.6 – 4.11



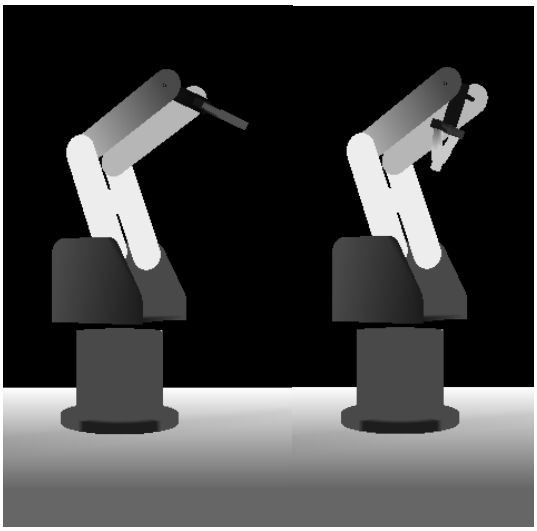
รูปที่ 4.6 แสดงภาพเมื่อกดปุ่ม Left และ Right ตามลำดับ ที่ Joint 1



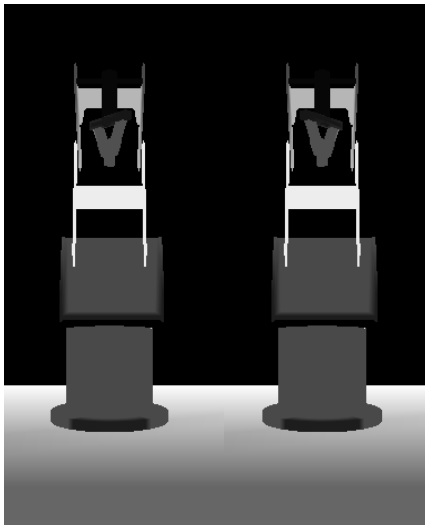
รูปที่ 4.7 แสดงภาพเมื่อกดปุ่ม Up และ Down ตามลำดับ ที่ Joint 2



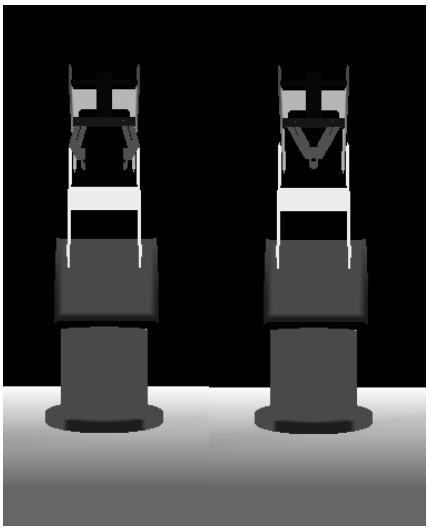
รูปที่ 4.8 แสดงภาพเมื่อกดปุ่ม Up และ Down ตามลำดับ ที่ Joint 3



รูปที่ 4.9 แสดงภาพเมื่อกดปุ่ม Up และ Down ตามลำดับ ที่ Joint 4

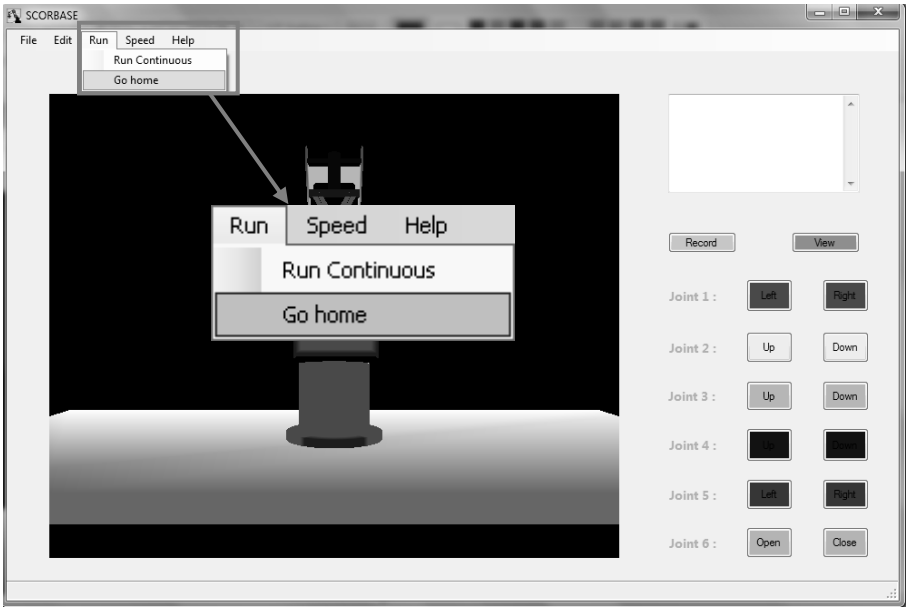


รูปที่ 4.10 แสดงภาพเมื่อกดปุ่ม Left และ Right ตามลำดับ ที่ Joint 5

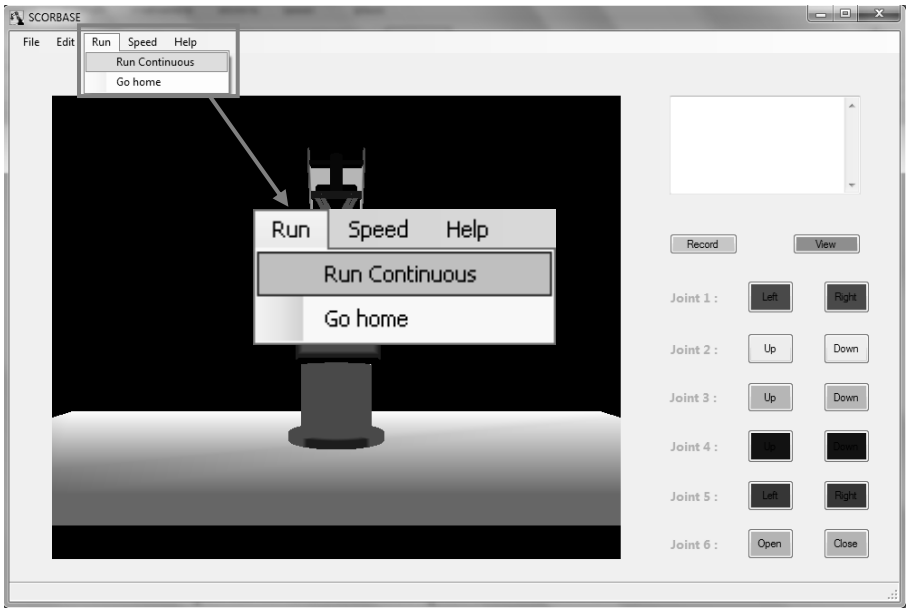


รูปที่ 4.11 แสดงภาพเมื่อกดปุ่ม Open และ Close ตามลำดับ ที่ Joint 6

การบันทึกตำแหน่งการเคลื่อนไหวของแขนกล เพื่อเรียกดูภายหลัง สามารถทำได้โดยกดปุ่ม Go home เพื่อกลับมายังท่าเริ่มต้น และกดปุ่ม Run Continuous เพื่อให้โปรแกรมเริ่มแสดงการเคลื่อนที่ตามตำแหน่งที่ได้บันทึกไว้ แสดงไว้ในรูปที่ 4.12 และ 4.13

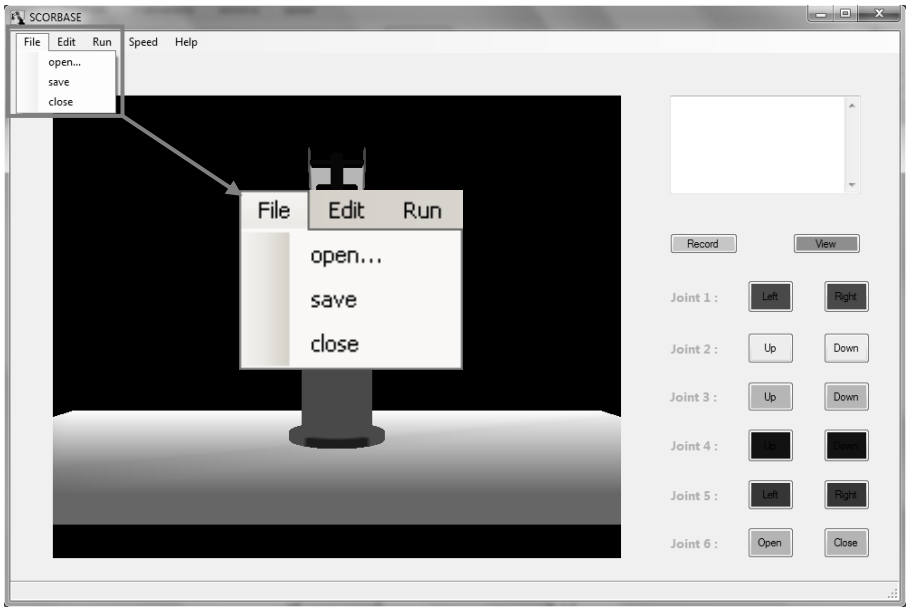


รูปที่ 4.12 แสดงขั้นตอนการเลือกตำแหน่งเริ่มต้น



รูปที่ 4.13 แสดงขั้นตอนการเรียกดูการเคลื่อนไหวทั้งหมดของแขนกล

วิธีการบันทึกตำแหน่งเป็นไฟล์ข้อมูล สามารถทำได้โดยการกด File เลือก save จะได้ไฟล์ข้อมูลเป็น .txt จะได้ชุดคำสั่งที่สามารถเรียกดูในภายหลังได้ โดยกดปุ่ม open เพื่อเรียกดูชุดคำสั่งดังกล่าว ดังรูปที่ 4.14



รูปที่ 4.14 แสดงขั้นตอนการบันทึกและเปิดไฟล์

บทที่ 5
สรุปและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

งานวิจัยนี้มีวัตถุประสงค์เพื่อสร้างโปรแกรมจำลองการทำงานของแขนกล เพื่อใช้ในการเรียนรู้การทำงานของแขนกลเบื้องต้น โดยการเขียนโปรแกรมการจำลองการทำงานของแขนกลรุ่น SCORBOT-ER 4u เริ่มจากการวัดขนาดของแขนกลแต่ละชิ้นส่วน นำขนาดดังกล่าวมาเขียนแบบด้วยโปรแกรม SolidWorks และแปลงเป็นไฟล์ PLY เพื่อทราบจำนวนจุดและจำนวนระนาบของรูปทรงต่างๆ และศึกษาการเขียนโปรแกรมเชิงวัตถุ, Visual C#, OpenGL Library, Transformation matrix และข้อมูลจำเพาะของแขนกลในแต่ละชิ้นส่วน โดยนำจำนวนพิกัดและระนาบที่ทราบจากไฟล์ PLY มาแสดงเป็นภาพสามมิติของชิ้นส่วนทั้งหมดที่ประกอบเข้าด้วยกันจนเป็นแขนกล โดยรับคำสั่งการเคลื่อนที่ในแต่ละชิ้นส่วนจากผู้ใช้มาประมวลผลและแสดงเป็นภาพเคลื่อนไหวสามมิติ

5.1.1 ผลที่ได้รับ

โปรแกรมจำลองการทำงานของแขนกลที่สร้างขึ้นสามารถรับคำสั่งจากผู้ใช้ และบันทึกคำสั่งได้ 100 คำสั่ง พร้อมทั้งเรียกดูการทำงานตั้งแต่คำสั่งแรกโดยเริ่มจากท่าเริ่มต้น (Home position) เป็นโปรแกรมที่สามารถเพิ่มทักษะการใช้แขนกลรุ่น SCORBOT – ER 4u เพื่อลดความสูญเสียที่เกิดขึ้นจากผู้ใช้งานแขนกลที่ขาดประสบการณ์ ลดเวลาในการศึกษาแขนกลเข้าใจลักษณะและข้อมูลจำเพาะในข้อต่อต่างๆ ของแขนกล โดยโปรแกรมดังกล่าวสามารถใช้เป็นสื่อการเรียนการสอนในสาขาวิชาวิศวกรรมอุตสาหการ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังได้

5.2 ข้อเสนอแนะ

ควรมีการปรับปรุงโปรแกรมหลังจากการนำไปทดลองใช้งานจริงกับผู้ที่สนใจหรือนักศึกษา เพื่อให้สอดคล้องกับการใช้งานจริงและเพิ่มประสิทธิภาพการทำงานของโปรแกรมให้มากยิ่งขึ้น และง่ายต่อการใช้งาน ผู้ใช้งานควรศึกษาคู่มือก่อนใช้งานจริง โปรแกรมจำลองการทำงานของแขนกลที่สร้างขึ้นเป็นเพียงการจำลองการเคลื่อนไหวของแขนกลเท่านั้น สิ่งที่ควรเพิ่มเติม คือ การสร้างสภาพแวดล้อมของแขนกล เช่น บริเวณที่ปฏิบัติงาน อุปกรณ์เสริม ชิ้นงานที่เกี่ยวข้อง มุมมองภาพในอิริยาบถต่างๆ ที่สามารถเปลี่ยนแปลงได้โดยง่าย เช่น การเลื่อนเมาส์ การเคลื่อนไหวแบบ Invert Kinematic นอกจากนี้ควรแสดงผลที่เกิดขึ้นจากการเคลื่อนที่ เช่น เกิดการชน เกิดความเสียหาย

บรรณานุกรม

[1] Pedro Neto, J. Norberto Pires, A. Paulo Moreira, “Robot Path Simulation: A Low Cost Solution Based on CAD”, IEEE Conference on Robotics Automation and Mechatronics, June 2010.

[2] A. Aguiar, A. Silva, E. Villani, “Graphic robot simulation for the design of work cells in the aeronautic industry”, ABCM Symposium Series in Mechatronics, Vol. 3, pp.346-354, 2008.

[3] J. Swider, K. Foit, G. Wszolek, D. Mastrowski, “ The off-line programming and simulation software for the Mitsubishi Movemaster RV-M1 robot”, Journal of Achievements in Materials and Manufacturing Engineering, Vol. 20, Issues 1-2, January 2007.

[4] P.C. Patic, L. Pascale, M. Ardeleanu, F. Popa, “Simulation and Modeling of a Column Industrial Robot Used in some Different Mounting Processes”, International Journal of Systmes Applications, engineering & Development, Vol. 4, Issue 4, 2010.

[5] P.I. Corke, “A computer tool for simulation and analysis: the Robitics Toolbox for MATLAB”, Proceedings of the 1995 National Conference of the Australian Robot Association, Melbourne, Australia, pp. 319-330, July 1995.

[6] R. Bernhardt, G. Schreck, and C. Willnow, “Realistic robot simulation,” Computing & Control Engineering Journal, vol. 6, no. 4, pp. 174–176, 1995.

[7] M. Bruccoleri, C. D’Onofrio, and U. La Commare, “Off-line programming and imulation for automatic robot control software generation,” in Proceedings of the 5th IEEE International Conference on Industrial Informatics, pp. 491–496, Vienna, Austria, 2007.

[8] Khali, W., and Dombre, E., 2002. Modeling, Identification & Control of Robots. London : Kogan Page Science.

[9] บัญชา ปะสีละเตสัง, 2552. พัฒนาแอปพลิเคชั่นด้วย Visual C# 2008. กรุงเทพฯ : สำนักพิมพ์ ซีเอ็ดยูเคชั่น.

[10] สุธี พงศาสกุลชัย และ หทัยชนก งามอินทร์, 2550. คัมภีร์ Visual C# 2005. พิมพ์ครั้งที่ 2. กรุงเทพฯ : สำนักพิมพ์ เคที พี คอมพ์ แอนด์ คอนซัลท์.

[11] ศูนย์รวมความรู้วิศวกรรม. หุ่นยนต์อุตสาหรรมประเภทต่างๆ. < http://www.mceengineer.com>

[12] ไพศาล โมลิสกุลมงคล, 2550. คอมพิวเตอร์กราฟิกใช้ OpenGL. กรุงเทพฯ : โรงพิมพ์ ไทยเจริญการพิมพ์.

[13] Intelitek. SCORBOT – ER 4u. <http://www.intelitek.com>