การควบคุมแขนหุ่นยนต์
ARM ROBOT MANIPULATOR

ปาณัสม์ หมู่โพธิ์
Panus Moopho
พิทักษ์ ภู่เรือง
Pituk Pu-raeng
พีรกิตติ์ ตันติวรานุรักษ์
Peerakit Tuntiwaranuruk

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2557

การควบคุมแขนหุ่นยนต์

ARM ROBOT MANIPULATOR

โดย

ปาณัสม์ หมู่โพธิ์

พิทักษ์ ภู่เรือง

พีรกิตติ์ ตันติวรานุรักษ์

อาจารย์ที่ปรึกษา

ผศ.ดร.ยุทธนา คิดใจเดียว

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2557

ปริญญานิพนธ์ปีการศึกษา 2557

สาขาวิชา	วิศวกรรมอิเล็กทรอนิกส์
คณะ	วิศวกรรมศาสตร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
เรื่อง	การควบคุมแขนหุ่นยนต์
	Arm Robot Manipulator

ผู้จัดทำ	นายปาณัสม์	หมู่โพธิ์	รหัสนักศึกษา	54010810
	นายพิทักษ์	ภู่เรือง	รหัสนักศึกษา	54010923
	นายพีรกิตติ์	ตันติวรานุรักษ์	รหัสนักศึกษา	54010937

รายงานฉบับนี้ผ่านการตรวจสอบโดยอาจารย์ที่ปรึกษาแล้ว

ลงชื่อ.....

(ผศ.ดร.ยุทธนา คิดใจเดียว)

หัวข้อปริญญานิพนธ์	การควบคุมแขนหุ่นยนต์			
นักศึกษา	นายปาณัสม์	หมู่โพธิ์	รหัสนักศึกษา	54010810
	นายพิทักษ์	ภู่เรือง	รหัสนักศึกษา	54010923
	นายพีรกิตติ์	ตันติวรานุรักษ์	รหัสนักศึกษา	54010937
ปริญญา	วิศวกรรมศาสตรบัณฑิต			
สาขา	วิศวกรรมอิเล็กทรอนิกส์			
ปีการศึกษา	2557			
อาจารย์ที่ปรึกษาปริญญานิพนธ์	ผศ.ดร.ยุทธนา คิดใจเดียว			

บทคัดย่อ

ปริญญานิพนธ์นี้ นำเสนอการออกแบบและการทำงานของแขนหุ่นยนต์ที่มีการสั่งการทำงานจากระยะไกล และมีการตอบสนองอย่างรวดเร็ว ระบบนี้จะศึกษาและวิจัยการเคลื่อนไหวของมนุษย์ โดยเซ็นเซอร์ของกล้อง Kinect เป็นอุปกรณ์ตรวจสอบภาพการเคลื่อนไหวของมนุษย์ แล้วใช้โปรแกรม Processing ในการรับภาพและประมวลผลแล้วทำการส่งข้อมูลแบบทันทีโดยใช้โมดูลไร้สาย XBee เป็นตัวกลาง ให้ไมโครคอนโทรลเลอร์ Arduino ประมวลผล ควบคุมการทำงานของแขนหุ่นยนต์ ระบบนี้จะประหยัดต้นทุนและหุ่นยนต์มีการตอบสนองที่รวดเร็วสำหรับใช้ในโรงงาน หรือใช้ในพื้นที่ที่อันตรายไม่สามารถใช้คนในการทำงานได้

Thesis Title	Arm Robot Manipulator		
Student	Panus	Moopho	ID.54010810
	Pituk	Pu-raeng	ID.54010923
	Peerakit	Tuntiwaranuruk	ID.54010937
Degree	Bachelor of Engineering		
Program	Electronics Engineering		
Year	2014		
Thesis Advisor	Assist. Prof. Dr. Yutthana Kitjaidure		

ABSTRACT

This project presents about design and operation of the robot arm, the command works remotely and quickly responds. This system will analyse human movement, by using the Kinect through the Processing. The Processing Program takes images and send the data via XBee modules. The micro controller Arduino is used to control the robot arm. This system is very economic. The robot responds very suitable for using in the factory or in areas that cannot work by human-being.

กิตติกรรมประกาศ

ในการทำโครงงานครั้งนี้ ทางเราขอขอบคุณ อาจารย์ที่ปรึกษา ผศ.ดร.ยุทธนา คิดใจเดียว ที่คอยให้คำปรึกษาในทุกๆเรื่อง ขอขอบคุณพ่อและแม่ ที่คอยให้กำลังใจพวกเราเสมอมา ขอบคุณพี่อัธ พี่พิม พี่ต่าย พี่เชฟ ที่ค่อยช่วยเหลือและให้คำปรึกษาในการทำโครงงานครั้งนี้ และขอบใจเพื่อนในห้องโปรเจคทุกคนทั้ง มอส เปม และโย ที่ ค่อยอยู่ทำงานเป็นเพื่อนกัน ถึงดึก เกือบทุกวัน

ปาณัสม์ หมู่โพธิ์

พิทักษ์ ภู่เรือง

พีรกิตติ์ ตันติวรานุรักษ์

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

หน้า

สารบัญรูป

หน้า

สารบัญรูป(ต่อ)

หน้า

สารบัญรูป(ต่อ)

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เนื่องด้วยปัจจุบัน เทคโนโลยีการตรวจจับภาพเคลื่อนไหว มีก้าวหน้าและทันสมัยอย่างมาก และยังมีการนำมาประยุกต์ใช้งานหลากหลายแบบ เช่น การสร้างภาพทางการแพทย์ การวิเคราะห์ทางวิทยาศาสตร์การกีฬา การสร้างภาพยนตร์หรือการ์ตูนสามมิติ และการนำมาใช้ในเล่นเกมส์ต่างๆ เป็นต้น เราจึงได้มีการนำเทคโนโลยีการตรวจจับภาพการเคลื่อนไหวมาประยุกต์ใช้กับการสั่งให้แขนหุ่นยนต์ทำงานตามการเคลื่อนไหว จากระยะไกล เพื่อนำไปใช้ในโรงงานอุตสาหกรรม หรือใช้ในพื้นที่ที่อันตรายไม่สามารถใช้คนในการทำงานได้

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

ปริญญานิพนธ์ ฉบับนี้มุ่งหวังการศึกษาและวิจัยการเคลื่อนไหวของมนุษย์ จะใช้เซ็นเซอร์ของกล้อง Kinect เป็นอุปกรณ์ตรวจสอบภาพการเคลื่อนไหวของมนุษย์ โดยใช้โปรแกรม Processing ในการรับภาพและประมวลผลแล้วทำการส่งข้อมูลแบบทันทีโดยใช้โมดูลไร้สาย ให้ไมโครคอนโทรลเลอร์ Arduino ประมวลผลและควบคุมการทำงานของแขนหุ่นยนต์ให้แขนกลทำงานตามการเคลื่อนไหวอย่างถูกต้อง

1.3 ขอบเขตของโครงงาน

ทำการรับภาพแบบวิดีโอด้วยโปรแกรม Processing จากกล้อง Kinect แล้ว

ทำการ capture นำข้อมูลที่ได้มาคำนวณหาพิกัดการเคลื่อนไหว แล้วส่งข้อมูลจากระยะไกลให้ไมโครคอนโทรลเลอร์ Arduino ประมวลผลและควบคุมการทำงานของแขนหุ่นยนต์

1.4 ประโยชน์ที่คาดว่าจะได้รับรายละเอียดและเนื้อหาในรายงาน

1.ได้รับความรู้ความเข้าใจเกี่ยวกับเทคนิคการประมวลผลภาพ

2.ได้รับความรู้และความเข้าใจในการใช้กล้อง Kinect ในการรับภาพระดับความลึกของวัตถุ

3.ได้รับความรู้ในการเขียนโปรแกรม Processing, Arduino และการส่งข้อมูลด้วย XBee

4.เรียนรู้การวัดมุมแบบ Absolute Angle และ Relative Angle

1.5 รายละเอียดของเนื้อหาในรายงาน

เนื้อหาในรายงานจะแบ่งออกเป็นบทต่างๆตามรายละเอียด ดังนี้

บทที่ 1 บทนำ

บทที่ 2 ทฤษฎีที่เกี่ยวข้อง

บทที่ 3 หลักการทำงานของระบบ

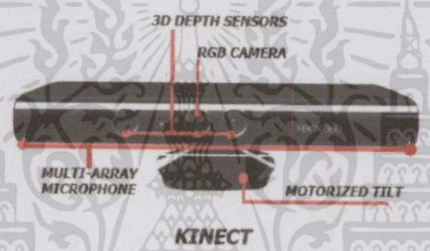
บทที่ 4 ขั้นตอนการทดลองและผลการทดลอง

บทที่ 5 สรุปผลการทดลองและปัญหาที่เกิด

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 Kinect

เป็นอุปกรณ์รับรู้การเคลื่อนไหวที่ใช้การทำงานผสมผสานกันระหว่างฮาร์ดแวร์และซอฟท์แวร์ ฟังก์ชันหลักของ Kinect มีสองฟังก์ชันคือ สร้างภาพเคลื่อนไหวสามมิติของวัตถุในมุมมองที่กำหนด และแยกแยะมนุษย์ออกจากวัตถุเหล่านั้นได้ ในปัจจุบัน Kinect มีความละเอียดของภาพอยู่ที่ 640x480 pixel และสามารถทำงานได้ที่ 30 frames ต่อวินาที



รูปที่ 2.1 อุปกรณ์ Kinect

2.1.1 ส่วนประกอบของและหลักการทำงานของ Kinect

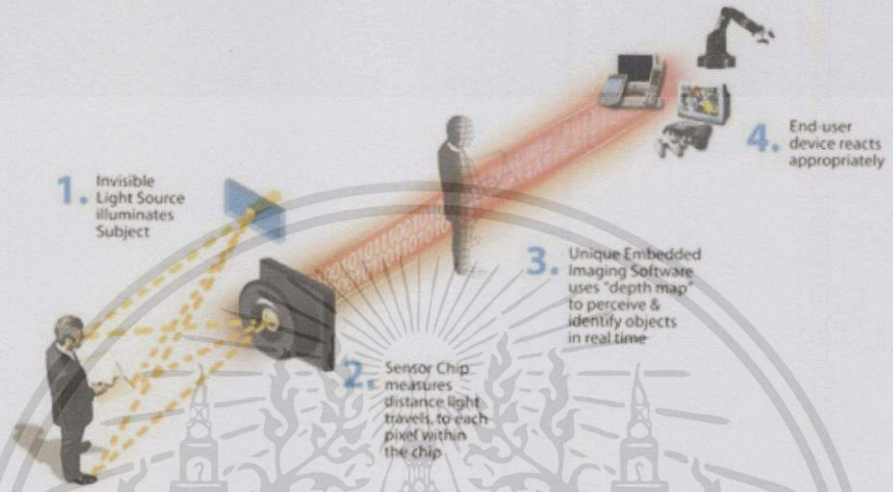
สามารถแบ่งออกเป็น 3 ส่วนประกอบสำคัญ ได้แก่

Color VGA video camera เป็นกล้องถ่ายภาพเคลื่อนไหวซึ่งช่วยในการจดจำใบหน้า (Face Recognition) และการตรวจจับลักษณะเด่นอื่นๆโดยใช้ความสามารถในการตรวจจับองค์ประกอบของสีทั้งสามสีอันได้แก่ สีแดง สีเขียว และสีน้ำเงิน

Depth sensor ประกอบด้วยการทำงานร่วมกันของต้นกำเนิดแสงอินฟราเรด (Infrared Projector)

ตัวรับรู้แบบ monochrome CMOS (Complimentary Metal-Oxide Semiconductor) ซึ่งทำหน้าที่รับแสงอินฟราเรดที่ถูกสะท้อนกลับมาจากวัตถุ จากนั้นทำการวัดเวลาในการเดินทาง

(Time of Flight) แสงอินฟราเรดนี้ใช้หลักการทำงานเช่นเดียวกับโซนาร์คือ หากรู้ระยะเวลาที่แสงอินฟราเรดใช้ในการเดินทางไปกลับก็จะสามารถคำนวณระยะห่างระหว่างตัวรับรู้ความลึกกับวัตถุได้ โดยการทำงานด้วยความเร็วแสงหลายๆรอบทำให้สามารถระบุระยะห่างที่แน่นอนได้โดยไม่ต้องคำนึงถึงสภาพแสงสว่าง

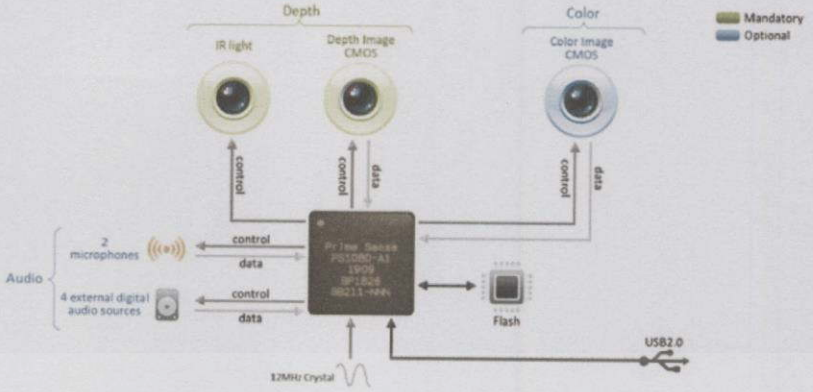


รูปที่ 2.2 แสดงการทำงานของตัวรับรู้ความลึกใน Kinect

การทำงานเริ่มจากการฉายแสง อินฟาเรดออกจากตัว Kinect ซึ่งไม่สามารถมองเห็นได้ด้วยตาเปล่า แสงที่ถูกฉายออกมาจะมีลักษณะเป็นจุดๆ ตามแนวตั้ง 480 จุด แนวนอน 640 จุด แต่ละจุดห่างกัน 3 มิลลิเมตร (ที่ระยะสองเมตรจากแหล่งกำเนิดแสง)

หลังจากนั้น กล้องวัดความลึกจะรับภาพระดับความสว่างของแสงอินฟาเรดที่ตกกระทบลงบนวัตถุ ส่งไปให้เซนเซอร์เพื่อทำการวัดความลึกตามแนวแกน Z (Axis-Z) ทำให้สามารถจำลองสภาพแวดล้อมเป็นสามมิติได้ หากความสว่างมีมากแสดงว่าวัตถุนั้นอยู่ใกล้ ในทางตรงกันข้ามหากมีความสว่างน้อยลงแสดงว่าวัตถุนั้นอยู่ไกลออกไป

รายละเอียดของไดอะแกรม Kinect แสดงดังรูปที่ 2.3 นอกจากนี้ Kinect ยังทำการบันทึกใบหน้าของคนและยังสามารถใช้เสียงในการควบคุมการใช้งานได้อีกด้วย



รูปที่ 2.3 แสดงไดอะแกรมของ kinect

เมื่อได้ระดับความลึกของภาพแล้ว ทำให้เซนเซอร์ของ Kinect สามารถแยกตัวคนออกจากสภาพแวดล้อมภายในห้องได้ เช่น ผนัง ที่นั่งเล่น หรือแม้แต่การจำแนกว่ามือของคนอยู่ข้างหน้าหรือข้างหลัง

ตารางที่ 2.1 แสดงรายละเอียดของ Kinect

Property	Spec
Field of View (Horizontal, Vertical, Diagonal)	58° H, 45° V, 70° D
Depth image size	VGA (640x480)
Spatial x/y resolution (@ 2m distance from sensor)	3mm
Depth z resolution (@ 2m distance from sensor)	1cm
Maximum image throughput (frame rate)	60fps
Operation range	0.8m – 3.5m
Color image size	UXGA (1600x1200)
Audio: built-in microphones	Two microphones
Audio: digital inputs	Four inputs
Data interface	USB 2.0
Power supply	USB 2.0
Power consumption	2.25W

ตารางที่ 2.1 แสดงรายละเอียดของ Kinect (ต่อ)

Dimensions (Width x Height x Depth)	14cm x 3.5cm x 5cm
Operation environment (every lighting condition)	Indoor
Operating temperature	0°C – 40°C

รูปที่ 2.4 แสดงภาพการฉาย Infrared

2.1.2 การรับรู้การเคลื่อนไหวของผู้เล่น

Kinect มีระบบการรับรู้การเคลื่อนไหวของมนุษย์อยู่ภายในเซนเซอร์เอง ซึ่งเป็นการนำเทคโนโลยีปัญญาประดิษฐ์เข้ามาช่วย โดยจะมีการส่งข้อมูลการเคลื่อนไหวของมนุษย์ในลักษณะต่างๆ เข้าไปเป็นจำนวนมาก ไม่ว่าจะเป็นท่าทางการนั่ง ยืน การเอียงตัว การขว้างลูกบอล การหยิบสิ่งของ หรือแม้แต่กระทั่งการตวัดมือ ข้อมูลการเคลื่อนไหวเหล่านี้จะถูกประมวลเหลือเพียงการเคลื่อนไหวของโครงกระดูก ซึ่ง Kinect จะวิเคราะห์ลักษณะการเคลื่อนไหวของข้อต่อแต่ละข้อรวมทั้งสิ้น 20 ข้อต่อ เพื่อนำไปวิเคราะห์อีกครั้งว่าขณะนี้มนุษย์กำลังแสดงท่าทางอะไรอยู่

สิ่งที่ระบบการตรวจจับการเคลื่อนไหวของ Kinect ต่างจากระบบอื่น คือ Kinect สามารถที่จะแยกแยะผู้เล่นออกจากสภาพแวดล้อมที่เป็นฉากหลังได้ดีกว่า เนื่องจาก Kinect มองภาพที่รับมาเป็นสามมิติ ไม่ใช่ระบบสองมิติ ซึ่งจะต้องใช้อัลกอริธึมอีกจำนวนมากในการแยกมนุษย์ออกจากฉากหลัง หรือยากต่อการวิเคราะห์ว่าสิ่งใดบ้างที่เคลื่อนไหวในฉากนั้น โดยส่วนใหญ่จะแสดงออกมาในรูปข้อต่อของร่างกายที่ตัวรับสัญญาณสามารถวิเคราะห์ได้

รูปที่ 2.5 แสดงข้อต่อที่ตัวรับสัญญาณวิเคราะห์ได้

2.2 เซอร์โวมอเตอร์

2.2.1 หลักการทำงานของเซอร์โวมอเตอร์

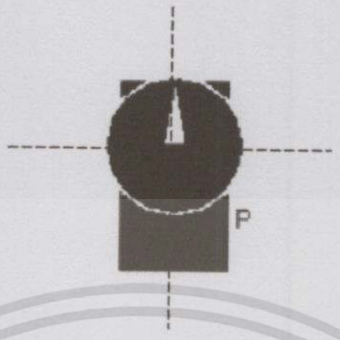
เซอร์โวมอเตอร์ ประกอบด้วย มอเตอร์ความเร็วสูง ภายในมีเฟืองทดรอบ ให้หมุนช้าลงเพื่อจะได้มีกำลังแรงบิดสูงขึ้น นอกจากนี้ยังมีวงจรควบคุมมอเตอร์ ซึ่งวงจรนี้จะนำค่าแรงดันเฉลี่ยของพัลส์รูปสี่เหลี่ยม เข้าไปเปรียบเทียบกับค่าแรงดันค่าหนึ่งที่มีอยู่ในวงจร ถ้าค่าต่างกัน วงจรควบคุมจะสั่งให้มอเตอร์หมุนไปตามทิศทาง ซึ่งขึ้นอยู่กับขนาดความกว้างของพัลส์ โดยที่แกนเฟืองทดรอบจะถูกพ่วงไปขับแกนของตัวต้านทานปรับค่าได้ (Potentiometer) ซึ่งอยู่ในวงจรควบคุมมอเตอร์ในขณะที่มอเตอร์หมุนตัวต้านทานปรับค่าได้ จะถูกปรับค่าทำให้ค่าแรงดันเปรียบเทียบของวงจรควบคุมมอเตอร์เปลี่ยนไปด้วย จนกระทั่งค่าเฉลี่ยของพัลส์ในวงจรควบคุมมอเตอร์ เท่ากับค่าเฉลี่ยของ พัลส์ที่เข้ามาจึงทำให้มอเตอร์หยุดหมุนได้

เซอร์โวมอเตอร์จะมีสายไฟ 3 เส้นคือ สายไฟเลี้ยง สายกราวด์ และสายสัญญาณพัลส์ควบคุม ซึ่งลักษณะของสัญญาณ พัลส์ที่ใช้ควบคุมตำแหน่งของเซอร์โวมอเตอร์ จะเป็นการส่งพัลส์ที่มีความกว้างต่างกัน เพื่อควบคุมให้เซอร์โวมอเตอร์ หมุนไปยังตำแหน่งที่ต้องการ โดยที่มีความกว้างของพัลส์ จะเป็นตัวกำหนดขนาดและทิศทางการหมุนของแกนเซอร์โวมอเตอร์สำหรับคาบเวลา หรือระยะห่างระหว่างลูกพัลส์แต่ละลูก จะเป็นตัวกำหนดแรงบิดมอเตอร์

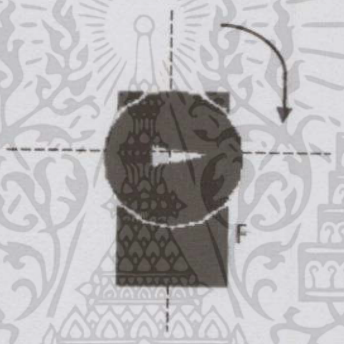
ถ้ากำหนดให้ในสภาวะปกติ เมื่อป้อนพัลส์ สี่เหลี่ยม ที่มีขนาด 1.5 มิลลิวินาที (1.5 ms) ให้กับเซอร์โวมอเตอร์ แขนของเซอร์โวมอเตอร์จะอยู่ตำแหน่งกลาง

เมื่อป้อนพัลส์สี่เหลี่ยม ที่มีความกว้างขนาด 1 มิลลิวินาที(1ms) ให้กับเซอร์โวมอเตอร์ แขนของเซอร์โวมอเตอร์จะหมุนตามเข็มนาฬิกา

เมื่อป้อนพัลส์สี่เหลี่ยม ที่มีความกว้างขนาด 2 มิลลิวินาที(2ms) ให้กับเซโวมอเตอร์ แขนของเซอร์โวมอเตอร์จะหมุนทวนเข็มนาฬิกา



รูปที่ 2.6 แสดงการตอบสนองของเซอร์โว เมื่อจ่ายพัลส์ขนาด 1.5ms



รูปที่ 2.7 แสดงการตอบสนองของเซอร์โว เมื่อจ่ายพัลส์ขนาด 1ms

รูปที่ 2.8 แสดงการตอบสนองของเซอร์โว เมื่อจ่ายพัลส์ขนาด 2ms

ดังนั้น ถ้าจ่ายสัญญาณพัลส์ที่มีความกว้างมากหรือน้อยกว่าความกว้างของพัลส์ 1.5ms ก็จะทำให้เซอร์โวมอเตอร์หมุนต่างทิศกัน ทั้งตามเข็มและทวนเข็มนาฬิกา โดยตำแหน่งของแขนหมุน เซอร์โวมอเตอร์จะเบี่ยงเบนออกจากจุดกึ่งกลางเป็นสัดส่วนกับความกว้างของพัลส์ที่จ่ายให้

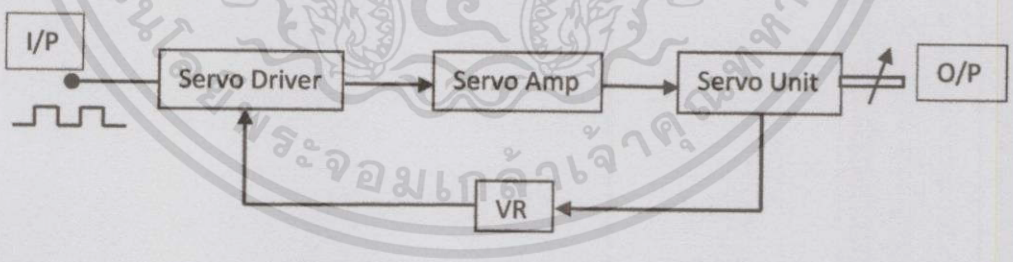
2.2.2 ภาคการทำงานของเซอร์โวมอเตอร์

ในเซอร์โวมอเตอร์หนึ่งตัวจะประกอบด้วย 3 ภาคการทำงานแต่ละภาคมีหน้าที่และ การทำงานดังนี้คือ

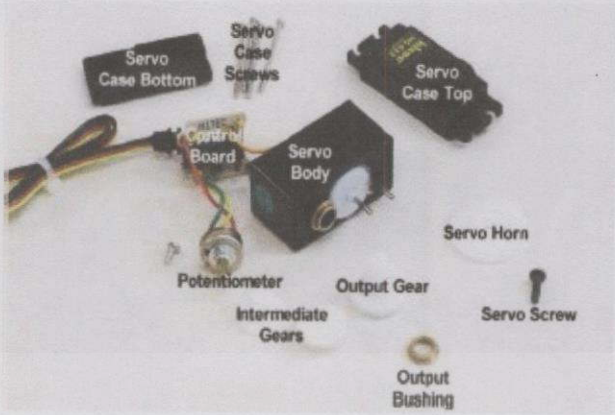
ภาคขับเซอร์โว ประกอบด้วย วงจรสร้างสัญญาณพัลส์ และวงจรเปรียบเทียบสัญญาณพัลส์ที่สร้างขึ้น กับสัญญาณพัลส์ I/P ที่รับเข้ามา

ภาคขยายเซอร์โว ประกอบด้วย RC Network ที่ช่วยหน่วงสัญญาณให้เซอร์โว สามารถทำงานได้ตลอดช่วงคาบเวลา จนกระทั่งมีสัญญาณลูกต่อไปมารวมถึงวงจร กลับขั้วแรงดันไฟฟ้าควบคุมทิศทางการหมุนของมอเตอร์

หน่วยเซอร์โว ประกอบด้วย มอเตอร์ความเร็วสูง เฟืองทดรอบ แกนหมุน อุปกรณ์ต่างๆ และตัวต้านทานปรับค่าได้ หน้าที่ป้อนกลับตำแหน่ง (Position Feedback) ซึ่งในขณะที่มอเตอร์หมุนตัวต้านทานปรับค่าได้จะเปลี่ยนแปลงค่า ซึ่งส่งผลให้ค่า ป้อนกลับเปลี่ยนแปลงไป จากนั้นนำค่าป้อนกลับมาปรับและเปรียบเทียบค่าความกว้างของพัลส์ ที่ภาคขับเซอร์โว เมื่อขนาดความกว้างของพัลส์มีค่าแรงดันเฉลี่ยเท่ากันมอเตอร์จะหยุดหมุนทันที



รูปที่ 2.9 แสดงภาคการทำงานของเซอร์โวมอเตอร์



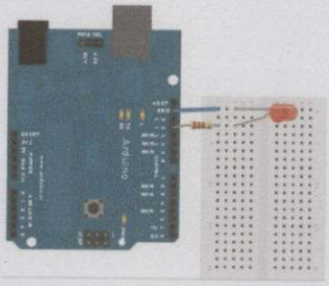
รูปที่ 2.10 ส่วนประกอบต่างๆ ของเซอร์โวมอเตอร์

2.3 Arduino

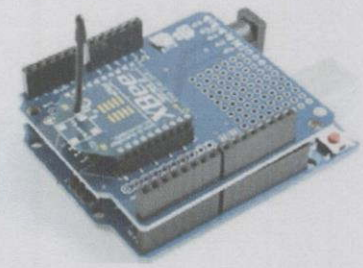
2.3.1 Arduino

เป็นบอร์ดไมโครคอนโทรเลอร์ตระกูล AVR ที่มีการพัฒนาแบบ Open Source คือมีการเปิดเผยข้อมูลทั้งด้าน Hardware และ Software ตัว บอร์ด Arduino ถูกออกแบบมาให้ใช้งานได้ง่าย ดังนั้นจึงเหมาะสำหรับผู้เริ่มต้นศึกษา ทั้งนี้ผู้ใช้งานยังสามารถดัดแปลง เพิ่มเติม พัฒนาต่อยอดทั้งตัวบอร์ด หรือโปรแกรมต่อได้อีกด้วย

ความง่ายของบอร์ด Arduino ในการต่ออุปกรณ์เสริมต่างๆ คือผู้ใช้งานสามารถต่อวงจรอิเล็กทรอนิคส์จากภายนอกแล้วเชื่อมต่อเข้ามาที่ขา I/O ของบอร์ด หรือเพื่อความสะดวกสามารถเลือกต่อกับบอร์ดเสริม (Arduino Shield) ประเภทต่างๆ เช่น Arduino XBee Shield, Arduino Music Shield, Arduino Relay Shield, Arduino Wireless Shield, Arduino GPRS Shield เป็นต้น มาเสียบกับบอร์ดบนบอร์ด Arduino แล้วเขียนโปรแกรมพัฒนาได้โดยมี Library ของแต่ละ Arduino Shield ให้ดาวน์โหลด

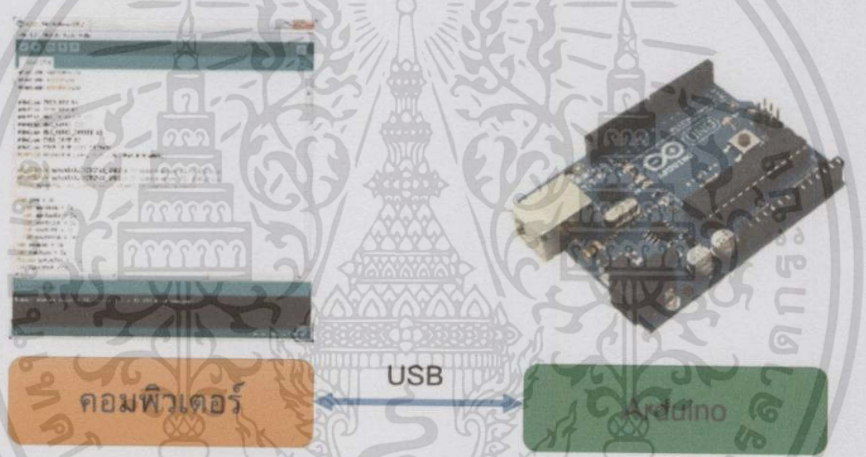


รูปที่ 2.11 บอร์ด Arduino ต่อกับ LED



รูปที่ 2.12 บอร์ด Arduino ต่อกับ XBee Shield

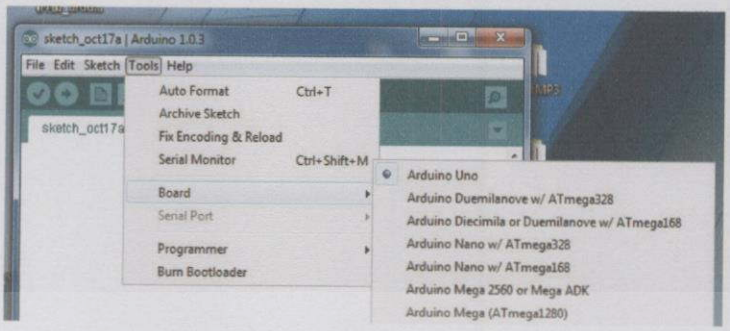
2.3.2 รูปแบบการเขียนโปรแกรมบน Arduino



รูปที่ 2.13 การเขียนโปรแกรมบน Arduino

1. เขียนโปรแกรมบนคอมพิวเตอร์ ผ่านทางโปรแกรม ArduinoIDE ซึ่งสามารถดาวน์โหลดได้จาก Arduino.cc/en/main/software

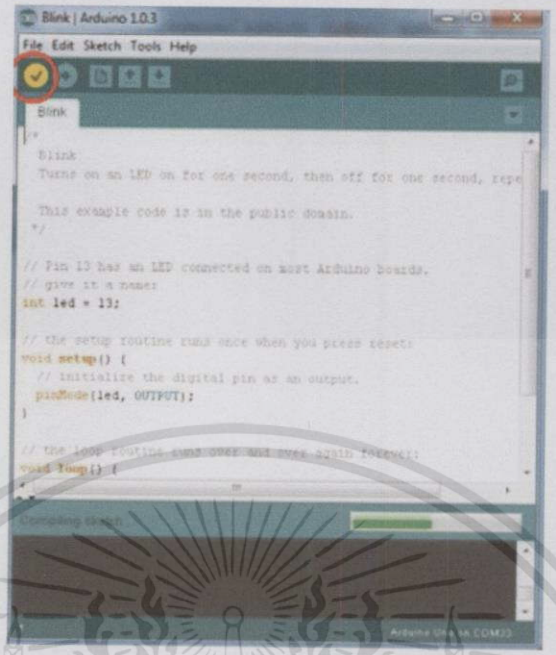
2. หลังจากที่เขียนโค้ดโปรแกรมเรียบร้อยแล้ว ให้ผู้ใช้งานเลือกรุ่นบอร์ด Arduino ที่ใช้และหมายเลข Com port



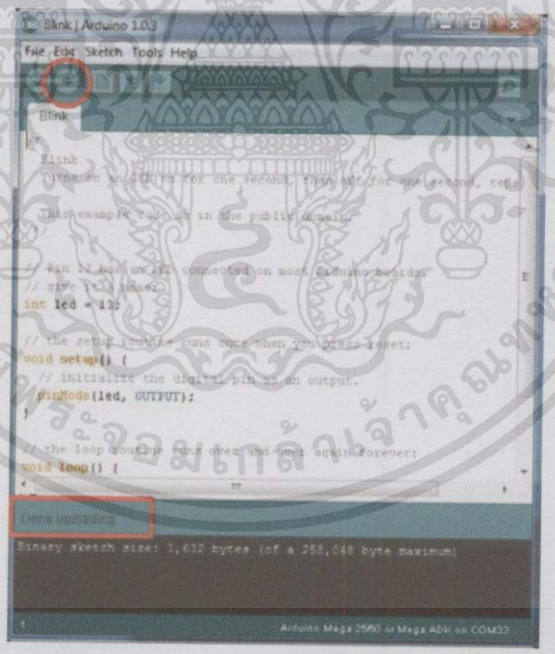
รูปที่ 2.14 เลือกรุ่นบอร์ด Arduino ที่ต้องการ upload

รูปที่ 2.15 เลือกหมายเลข Comport ของบอร์ด

3. กดปุ่ม Verify เพื่อตรวจสอบความถูกต้องและ Compile โค้ดโปรแกรม จากนั้นกดปุ่ม Upload โค้ด โปรแกรมไปยังบอร์ด Arduino ผ่านทางสาย USB เมื่ออัปโหลดเรียบร้อยแล้ว จะแสดงข้อความแถบข้างล่าง “Done uploading” และบอร์ดจะเริ่มทำงานตามที่เขียนโปรแกรมไว้ได้ทันที

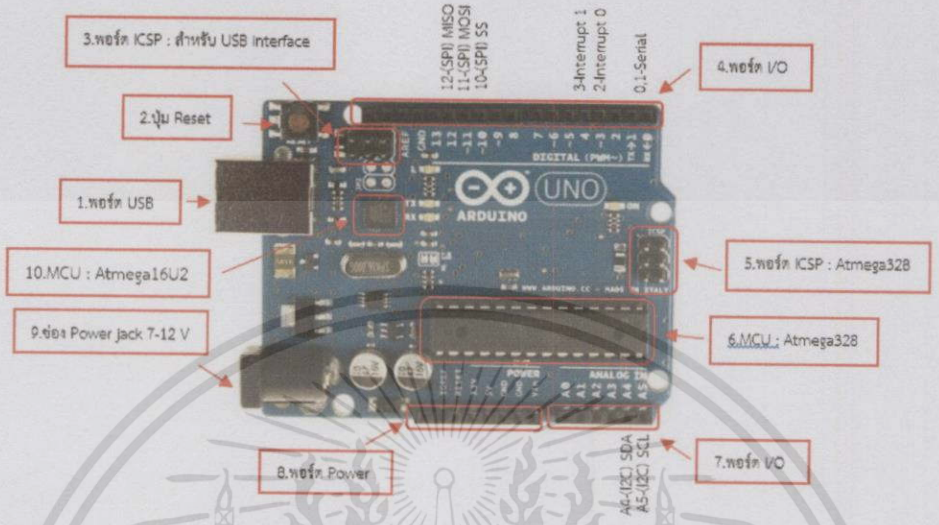


รูปที่ 2.16 กดปุ่ม Verify เพื่อตรวจสอบความถูกต้อง



รูปที่ 2.17 Upload โค้ดโปรแกรมและ Compile โค้ดโปรแกรม

2.3.3 Layout & Pin out Arduino Board (Model: Arduino UNO R3)



รูปที่ 2.18 บอร์ด Arduino UNO R3

1.USBPort: ใช้สำหรับต่อกับ Computer เพื่ออับโหลดโปรแกรมเข้า MCU และจ่ายไฟให้กับบอร์ด

2.Reset Button: เป็นปุ่ม Reset ใช้กดเมื่อต้องการให้ MCU เริ่มการทำงานใหม่

3.ICSP Port ของ Atmega16U2 เป็นพอร์ตที่ใช้โปรแกรม Visual Com port บน Atmega16U2

4. I/OPort:Digital I/O ตั้งแต่ขา D0 ถึง D13 นอกจากนี้ บาง Pin จะทำหน้าที่อื่นๆ เพิ่มเติมด้วย เช่น Pin0,1 เป็นขา Tx,Rx Serial, Pin3,5,6,9,10 และ 11 เป็นขา PWM

5.ICSP Port: Atmega328 เป็นพอร์ตที่ใช้โปรแกรม Bootloader

6. MCU: Atmega328 เป็น MCU ที่ใช้บนบอร์ด Arduino

7. I/OPort: นอกจากจะเป็น Digital I/O แล้ว ยังเปลี่ยนเป็น ช่องรับสัญญาณอนาล็อก ตั้งแต่ขา A0-A5

8.Power Port: ไฟเลี้ยงของบอร์ดเมื่อต้องการจ่ายไฟให้กับวงจรภายนอก ประกอบด้วยขาไฟเลี้ยง +3.3 V, +5V, GND, Vin

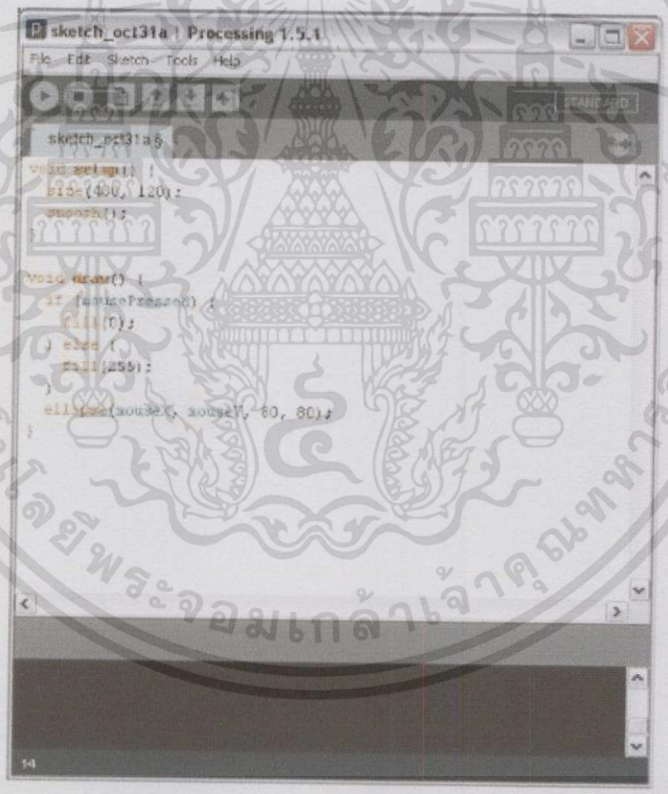
9. Power Jack: รับไฟจาก Adapter โดยที่แรงดันอยู่ระหว่าง 7-12 V

10. MCU ของ Atmega16U2 เป็น MCU ที่ทำหน้าที่เป็น USB to Serial โดย Atmega328 จะติดต่อกับ Computer ผ่าน Atmega16U2

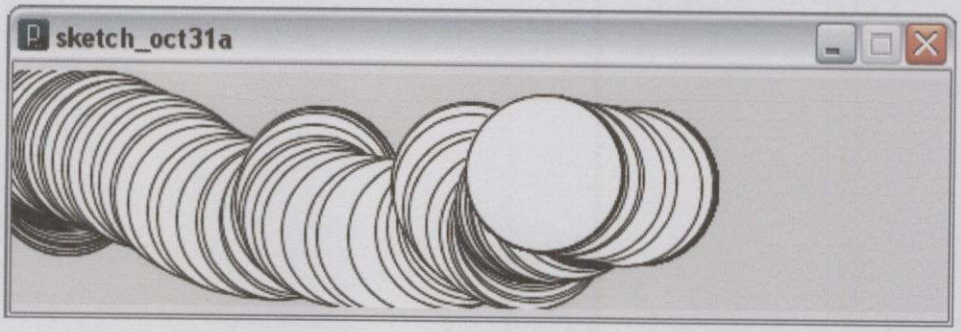
2.4 Processing

Processing เป็นซอฟท์แวร์ระบบเปิด เหมาะสำหรับผู้ต้องการพัฒนาโปรแกรมเกี่ยวกับการสร้างภาพเคลื่อนไหวและการมีปฏิสัมพันธ์สำหรับผู้ที่เคยใช้ชุดพัฒนาโปรแกรม Arduino

ทั้ง Processing และ Arduino ใช้หลักการในการเขียนโปรแกรมเหมือนกัน โดยมีพื้นฐานมาจากภาษา C/C++ รวมถึงการติดตั้งชุดพัฒนาก็เหมือนกันด้วย



รูปที่ 2.19 แสดงอินเตอร์เฟสโปรแกรม Processing



รูปที่ 2.20 แสดงการทำงานเบื้องต้นของโปรแกรม Processing

2.5 Accelerometers & Gyroscope GY-521

2.5.1 ข้อมูลเบื้องต้น (Introduction / Overview)

GY-521 เป็นโมดูล Accelerometers & Gyroscope ซึ่งสามารถทำงานได้ทั้ง 2 อย่างในเวลาเดียวกัน ใช้ ในการตรวจสอบทิศการเคลื่อนที่ และ สามารถใช้ในการตรวจสอบความเร็วในการเปลี่ยนแปลงทิศทาง ของแกน XYZ ได้

ยกตัวอย่าง ถ้าวัตถุเกิดการเคลื่อนที่หรือเอียง Output ของ Accelerometer จะบอกค่าของการเอียงว่า สถานะปัจจุบันค่าของ XYZ อยู่ที่เท่าไร แต่ Gyroscope จะวัดค่าได้ตอนที่กำลังเอียงหรือตอนกำลังเคลื่อนไหว เท่านั้น เมื่อวัตถุหยุดยิ่ง ค่างของ Gyroscope จะวัดไม่ได้เพราะไม่มีการเคลื่อนไหว

2.5.2 คุณสมบัติ (Features)

ใช้ไฟเลี้ยง +3.3 ถึง+5 V

ชิป MPU6050

เชื่อมต่อผ่านบัส I2C

2.5.3 การนำไปประยุกต์ใช้งาน (Application Ideas)

ตรวจสอบทิศทางการเคลื่อนที่ เคลื่อนไหวต่างๆของวัตถุ

2.5.4 ข้อควรระวังในการใช้งาน (Caution / Warning)

ควรหลีกเลี่ยงการต่อวงจรให้เกิดการลัดวงจร

ควรอ่านเอกสารก่อนการต่อวงจรจริง

ไม่ควรใช้ไฟเกินตามที่เอกสารกำหนด

2.5.5 คุณลักษณะ (Specification)

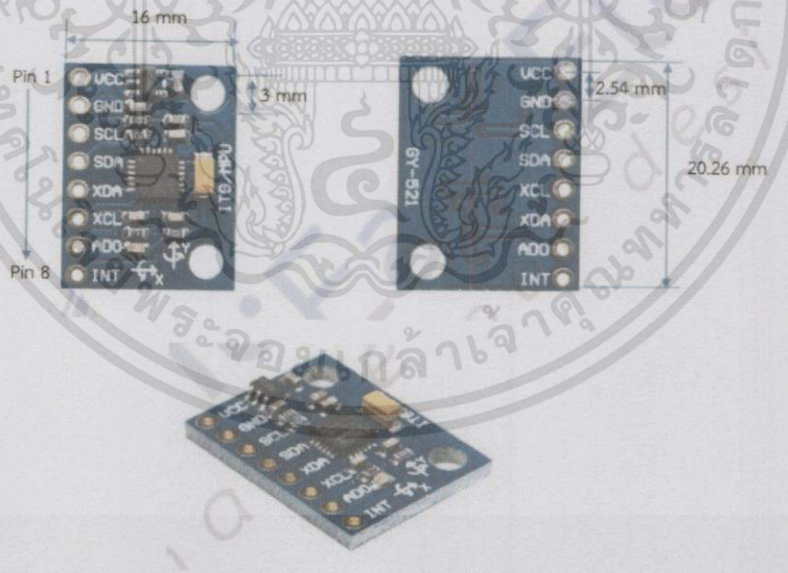
อุณหภูมิที่รองรับ -40 to +85 °C

รองรับแรงดัน 3.3 – 5 V

ทดสอบการตกกระแทกที่ 1.8 เมตร

ขนาด: 16 mm * 20 mm

2.5.6 โครงสร้าง (Structure)

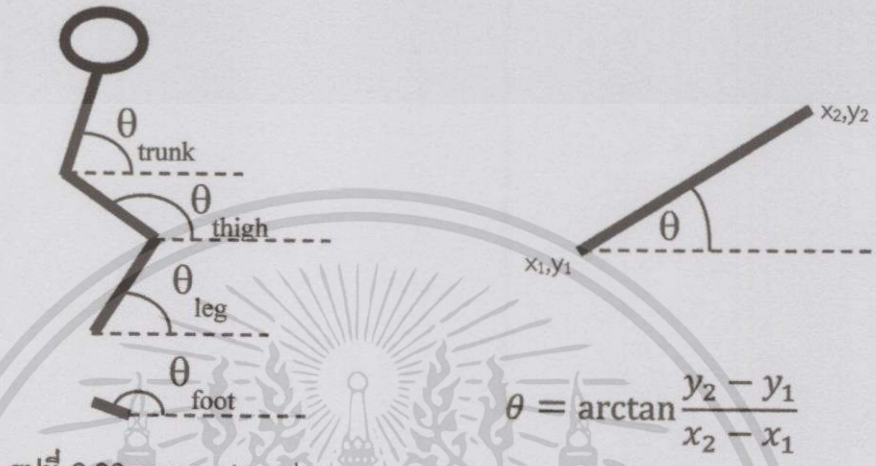


รูปที่ 2.21 โครงสร้างของ Accelerometers & Gyroscope GY-521

2.6 Absolute & Relative Angles

2.6.1 Absolute Angles (segment angles)

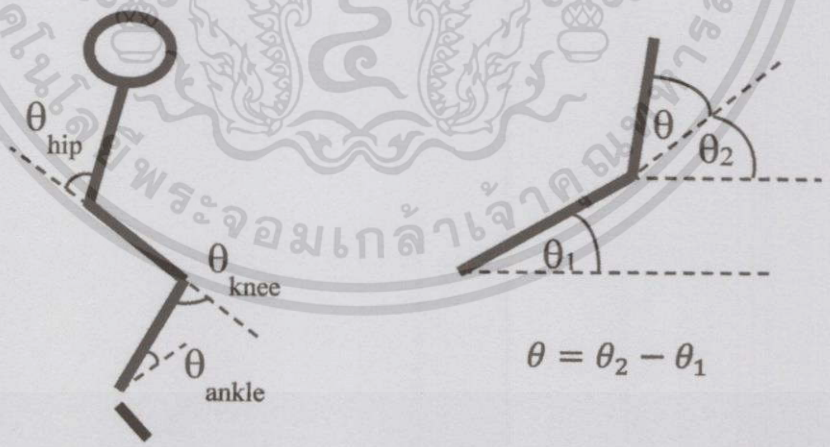
คือ มุมที่อยู่ระหว่างเซ็กเมนต์กับจุดอ้างอิง



รูปที่ 2.22 segment angles

2.6.2 Relative Angles (joint angles)

คือ มุมที่อยู่ระหว่างข้อต่อของสองเซ็กเมนต์ที่ติดกัน



รูปที่ 2.23 joint angles

2.7 XBee

ในโปรเจคนี้ เราได้ใช้ XBee-PRO โมดูลสื่อสารข้อมูลอนุกรมไร้สาย 2.4 GHz

2.7.1 คุณสมบัติทางเทคนิค

คุณสมบัติโดยทั่วไป

ความถี่ในการทำงาน 2.4 GHz

มีสายอากาศแบบ Whip

ระยะทำการในร่ม สูงสุด 300 ฟุต หรือประมาณ 100 เมตร

ระยะทำการกลางแจ้ง (แบบ light-of-sight) สูงสุดถึง 1 ไมล์ หรือประมาณ 1500 เมตร

กำลังส่ง 60 mW (18dBm)

ความไวในการรับสัญญาณ -100 dBm (1% packet error late)

การทำงานของขาพอร์ต สามารถกำหนดทางซอฟแวร์ X-CTU เพื่อทำงานเป็น

-อินพุตอนาลอกสำหรับวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล ความละเอียด 10 บิต

-อินพุตเอาต์พุตดิจิตอล

ขนาด 0.96×1.297 นิ้ว หรือ 2.438×3.294 เซนติเมตร

ไฟเลี้ยง 2.8 ถึง 3.4 V

กระแสไฟฟ้า เมื่อส่งข้อมูล 215 mA รับข้อมูล 55 mA น้อยกว่า 10 μA ในโหมดลดพลังงาน ที่ไฟเลี้ยง +3.3V

อุณหภูมิใช้งาน -40 ถึง 85 ˚C

คุณสมบัติด้านการสื่อสารข้อมูล

สามารถทำงานเป็นอุปกรณ์มาสเตอร์และสเลฟได้

อัตราการถ่ายทอดข้อมูลผ่านคลื่นวิทยุ 250,000 บิตต่อวินาที

อัตราการถ่ายทอดข้อมูลอนุกรม (บอดเรต) 1,200 ถึง 115,200 บิตต่อวินาที

รูปแบบเครือข่ายข้อมูลที่รองรับ จุดต่อจุด(point to point), จุดต่อหลายจุด(point to multipoint) และ เข้ากันได้กับอุปกรณ์ตามมาตรฐานรหัส 802.15.4

ทางเลือกแอดเดรส PAN ID, ช่อง Channel และเอดเดรส (Addresses) สำหรับแอดเดรส สามารถกำหนดรหัสแอดเดรสได้มากถึง 65,000 รหัส

เทคโนโลยีในการกระจายคลื่น DSSS (Direct Sequence Spread Spectrum)

รองรับการทำงานทั้งแบบ API และ AT command สามารถกำหนดได้ผ่านทางซอฟแวร์ X-CTU

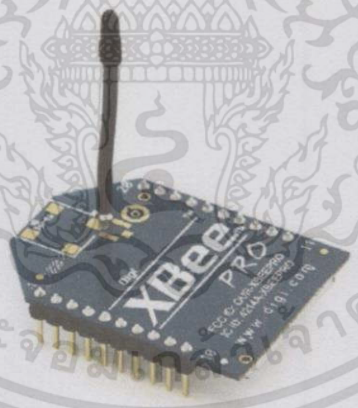
การรับรองมาตราฐาน

-สหรัฐอเมริกา (FCC Part 15.247) OUR-XBEEPRO

-แคนาดา (IC) 4214A XBEEPRO

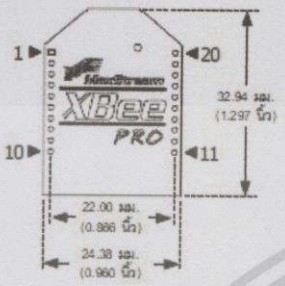
-ยุโรป (CE) ETSI (ที่กำลังส่งสูงสุด 10dBm)

-ญี่ปุ่น 005NYCA0378c(ที่กำลังส่งสูงสุด 10dBm)



รูปที่ 2.24 XBee

2.7.2 การจัดขา

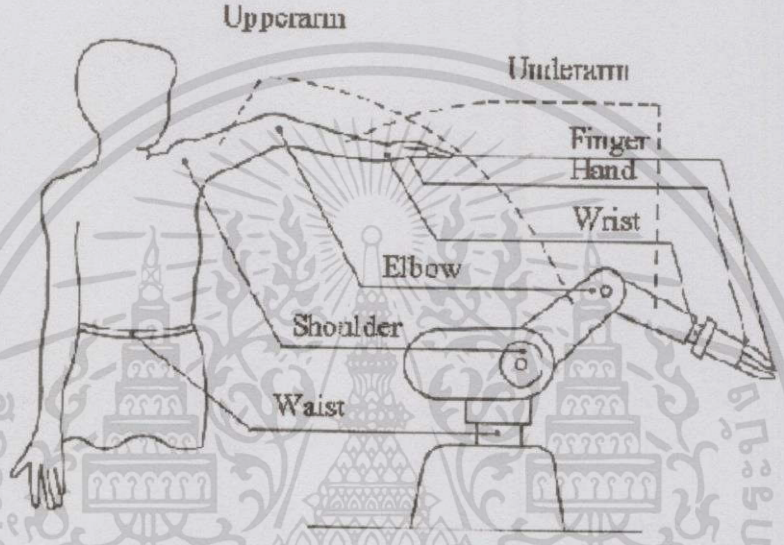


ขาที่	ชื่อขา/การทำงาน
1	Vcc : ขาต่อไฟเลี้ยง +3.3V
2	DOUT : ขาเอาต์พุตส่งข้อมูลอนุกรม
3	DIN : ขาอินพุตรับข้อมูลอนุกรม
4	DO8 : ขาเอาต์พุตดิจิตอล ช่อง 8
5	RESET : ขารีเซตหลัก (แอกตีฟ "0")
6	PWM0/RSSI : ขาเอาต์พุต PWM ช่อง 0 และขาเอาต์พุตแสดงความแรงของการรับสัญญาณ
7	PWM1 : ขาเอาต์พุต PWM ช่อง 1
8	ไม่ใช้งาน
9	DTR/SLEEP_RQ/DI8 : ขาอินพุตรับสัญญาณให้หยุดทำงานเข้าสู่โหมดสลีป หรือเป็นขาอินพุตดิจิตอลช่อง 8
10	GND : ขาต่อกราวด์
11	AD4/DIO4 : ขาอินพุตอะนาลอก 4 หรือ ขาอินพุตเอาต์พุตดิจิตอล 4
12	CTS/DIO7 : อินพุตรับสัญญาณแจ้งการส่งข้อมูลจากโฮสต์ (Clear-To-Send) ใช้ในการควบคุมจังหวะการรับส่งข้อมูล หรือเป็นขาอินพุตเอาต์พุตดิจิตอล 7
13	ON/SLEEP : ขาแสดงสถานะการทำงาน "1" : อยู่ในโหมดทำงานปกติ "0" : อยู่ในโหมดสลีป
14	VREF : ขาต่อแรงดันอ้างอิงสำหรับโมดูลแปลงสัญญาณอะนาลอกเป็นดิจิตอล ภายใน XBee-PRO
15	Associated/AD5/DIO5 : ขาแสดงสถานะการเชื่อมต่อ หรือ ขาอินพุตอะนาลอก 5 หรือ ขาอินพุตเอาต์พุตดิจิตอล 5
16	RTS/AD6/DIO6 : ขาเอาต์พุตแจ้งความพร้อมในการส่งข้อมูล (Ready-To-Send) ใช้ควบคุมจังหวะการรับส่งข้อมูล หรือเป็นขาอินพุตอะนาลอก 6 หรือเป็นขาอินพุตเอาต์พุตดิจิตอล 6
17	AD3/DIO3 : ขาอินพุตอะนาลอก 3 หรือ ขาอินพุตเอาต์พุตดิจิตอล 3
18	AD2/DIO2 : ขาอินพุตอะนาลอก 2 หรือ ขาอินพุตเอาต์พุตดิจิตอล 2
19	AD1/DIO1 : ขาอินพุตอะนาลอก 1 หรือ ขาอินพุตเอาต์พุตดิจิตอล 1
20	AD0/DIO0 : ขาอินพุตอะนาลอก 0 หรือ ขาอินพุตเอาต์พุตดิจิตอล 0

ตารางที่ 2.2 การจัดขาของ XBee-PRO และการฟังก์ชันทำงาน

2.8 หุ่นยนต์อุตสาหกรรม

การทำงานของหุ่นยนต์อุตสาหกรรมจะเลียนแบบร่างกายมนุษย์ โดยจะเลียนแบบเฉพาะส่วนของร่างกายที่จะนำไปใช้ประโยชน์ในอุตสาหกรรมเท่านั้น นั่นคือช่วงแขนของมนุษย์ ดังนั้นคำว่า “แขนกล” จึงเป็นอีกชื่อหนึ่งของหุ่นยนต์อุตสาหกรรม การทำงานของหุ่นยนต์สามารถเปรียบเทียบกับแขนของมนุษย์ แสดงดังรูปที่ 2.25



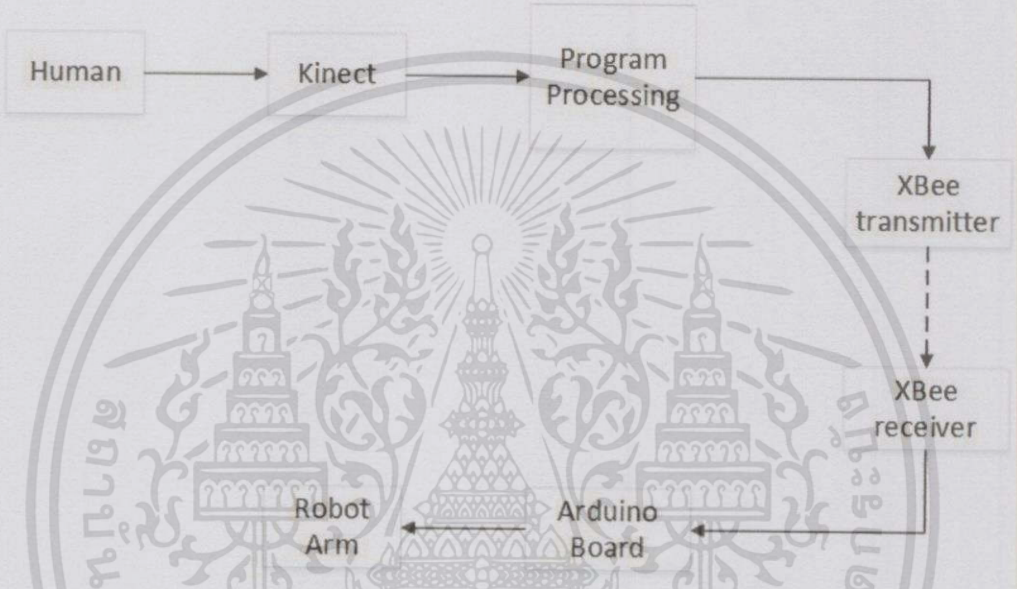
รูปที่ 2.25 แสดงส่วนต่างๆ ของหุ่นยนต์เปรียบเทียบกับสรีระของมนุษย์

บทที่ 3

หลักการทำงานของระบบ

3.1 หลักการทำงานของระบบ การควบคุมแขนหุ่นยนต์

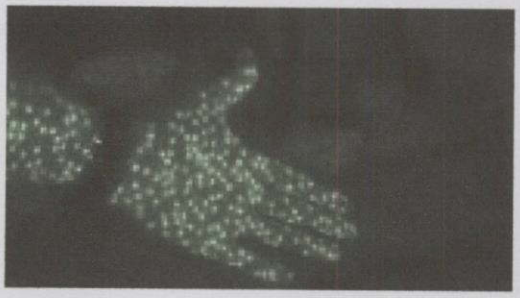
Block Diagram



รูปที่ 3.1 แสดง Block Diagram ของระบบ

3.2 Kinect

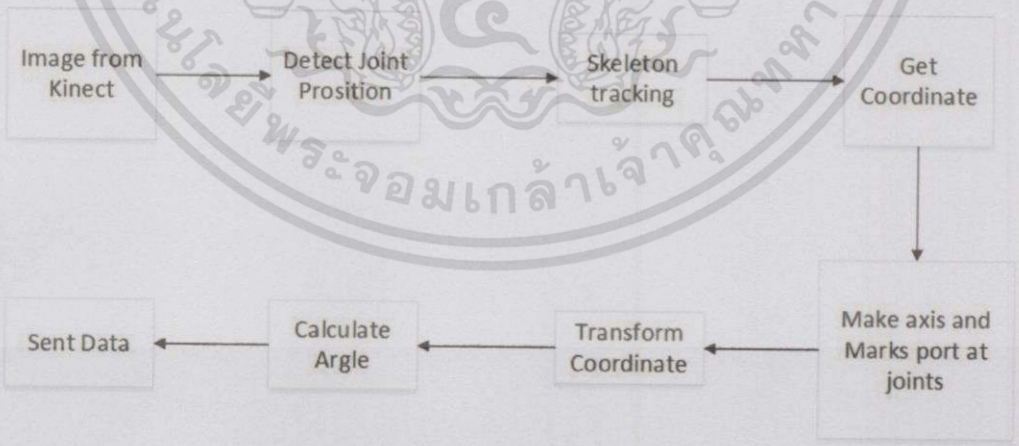
Kinect จะฉายแสงอินฟาเรดออกมา โดยแสงที่ฉายออกมาจะมีลักษณะเป็นจุด 640x480 และ Sensor ระดับความลึกของภาพ จะรับภาพระดับความสว่างของแสงอินฟาเรดที่ตกกระทบลงบนวัตถุ ส่งไปประมวลผลและสร้างภาพด้วยความเร็ว 30 เฟรมต่อวินาที



รูปที่ 3.2 แสดงแสงอินฟาเรดจาก Kinect

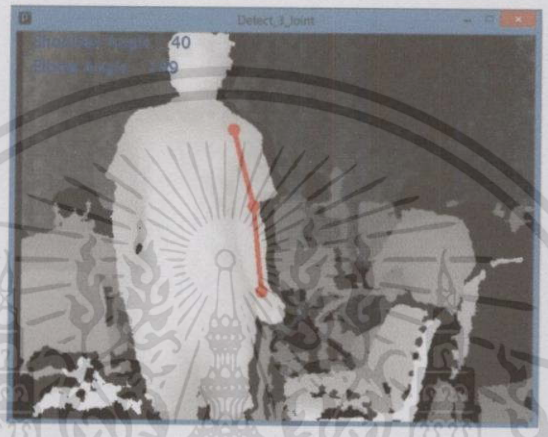
รูปที่ 3.3 แสดงภาพระดับความลึกของวัตถุ

3.3 Program Processing



รูปที่ 3.4 แสดงการทำงานของ Processing

การทำงานของโปรแกรม Processing เริ่มต้นจากการรับภาพจาก Kinect เพื่อทำการค้นหาข้อต่อ (หัวไหล่ ศอก และข้อมือและมือ) หลังจากจับข้อต่อทั้ง 4 ได้ จะสร้างเส้น Skeleton ระหว่างข้อต่อทั้ง 3 แล้วดึงข้อมูลพิกัด (x,y) ของข้อต่อต่างๆ มาใช้ในการวาดแกน,จุดตามข้อต่อ และแปลงพิกัด (x,y)ของ Processing เช่น พิกัด (x,y) ปกติเพื่อนำไปคำนวณมุมที่แขนของมนุษย์หนุนไปเพื่อส่งต่อไปยัง Xbee ผ่านทาง Serial port



รูปที่ 3.5 แสดง Skeleton ของแขน

3.4 XBee

ทำหน้าที่เป็นตัวกลางในการส่งข้อมูลระหว่างโปรแกรม Processing และโปรแกรม Arduino

รูปที่ 3.6 การส่งข้อมูลของ XBee

3.5 Arduino

Arduino จะทำการรับค่ามุมมาจากโปรแกรม Processing ผ่านทาง XBee และนำค่ามุมที่ได้ไปควบคุม หุ่นยนต์ให้เคลื่อนที่ตามมุมที่ได้รับมา

บทที่ 4

ขั้นตอนการทดลองและผลการทดลอง

4.1 ขั้นตอนการทดลอง

ตอนที่ 1 การวัดค่า Duty Cycle (มุม 0,30,45,60,90,120,135,180)

เขียนโปรแกรมสั่งให้บอร์ด Arduino สร้างสัญญาณพัลส์ออกทางขา PWM แล้วใช้ Oscilloscope วัดค่า Duty Cycle (%,ms)

ตอนที่ 2 การควบคุม Servo Motor ด้วย Arduino Board

เขียนโปรแกรมสั่งให้ Arduino Board จ่ายสัญญาณ Pulse ออกทาง Port Digital เพื่อควบคุม Servo Motor ที่ละตัวโดยใช้ Gyro Motor gy-521 วัดมุมที่ เคลื่อนที่ไปแล้วคำนวณค่า error

ตอนที่ 3 การควบคุม Servo Motor ด้วยแขน

ให้กล้อง Kinect จับภาพแขนของผู้ทดลอง แล้วขยับแขนทีละส่วนเพื่อควบคุมแขนกลให้เคลื่อนที่ และใช้ Gyro Motor gy-521 วัดมุมที่แขนกลเคลื่อนที่ไปได้ บันทึกลงในตารางแล้วคำนวณหาค่า error

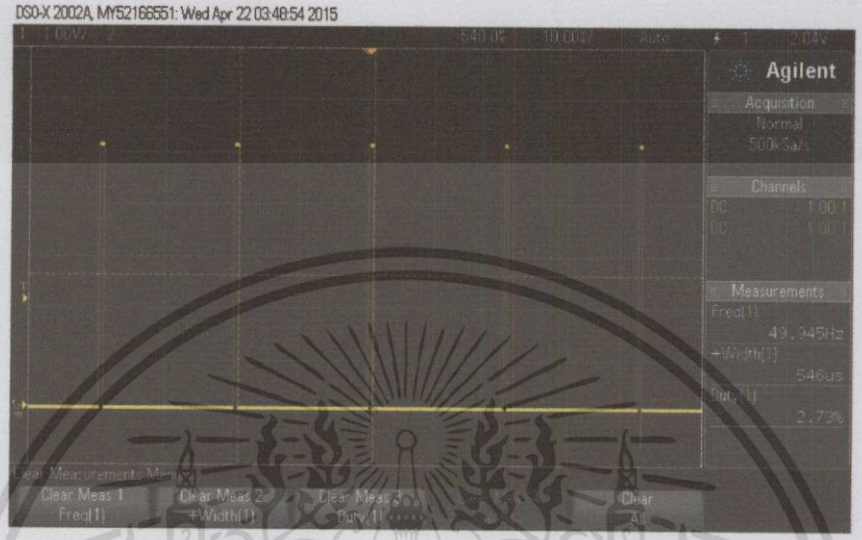
ตอนที่ 4 ควบคุมแขนกลแบบอิสระ โดยใช้การส่งข้อมูลแบบ Serial Port และแบบไร้สาย

ใช้กล้อง Kinect จับภาพ แขนของผู้ทดลอง และให้ผู้ทดลองเคลื่อนที่แขนเพื่อบังคับแขนกลให้หยิบของ โดยในครั้งแรกใช้การส่งข้อมูลแบบ Serial Port โดยตรง และครั้งที่สองใช้การส่งข้อมูลไร้สายโดยใช้ โมดูล XBee แล้วสรุปผลการทดลอง

4.2 ผลการทดลอง

ตอนที่ 1 การวัดค่า Duty Cycle

ทำการวัดค่า Duty Cycle เมื่อทำการสั่งค่ามุม 0 , 30 , 45 ,60 ,90 ,135 ,180 องศา

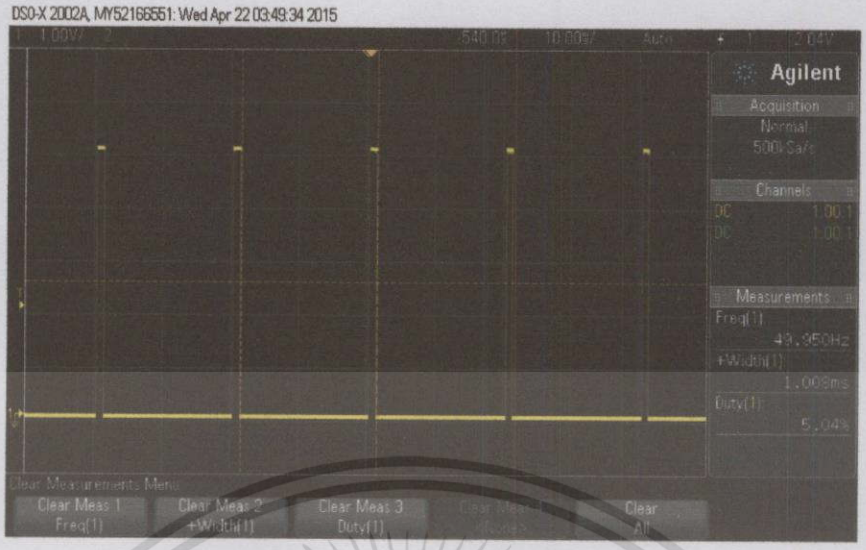


รูปที่ 4.1 กราฟค่า Duty Cycle ที่มุม 0 องศา

มีความถี่ 49.945 Hz Duty cycle 2.73% ,546 us

รูปที่ 4.2 กราฟค่า Duty Cycle ที่มุม 30 องศา

มีความถี่ 49.945 Hz Duty cycle 4.28% ,856 us



รูปที่ 4.3 กราฟค่า Duty Cycle ที่มุม 45 องศา

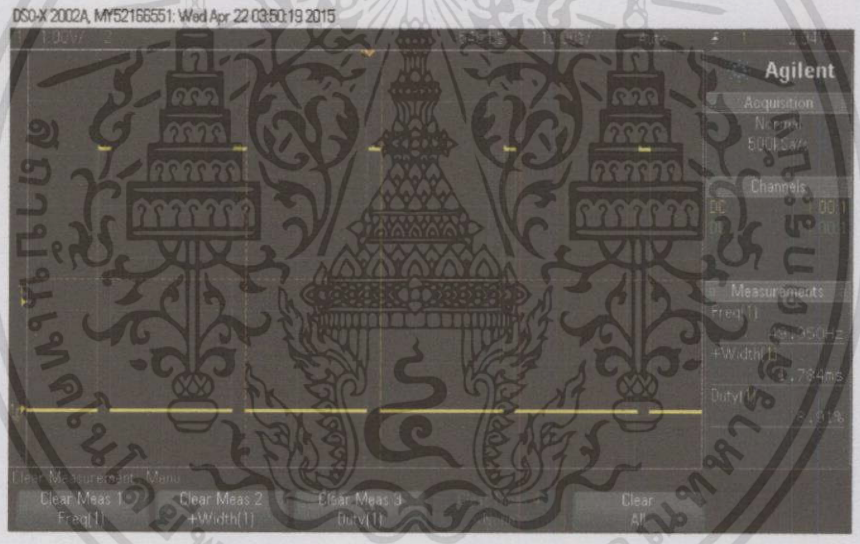
มีความถี่ 49.950 Hz Duty cycle 5.04% ,1.008 ms

รูปที่ 4.4 กราฟค่า Duty Cycle ที่มุม 60 องศา

มีความถี่ 49.945 Hz Duty cycle 5.81% ,1.164 ms

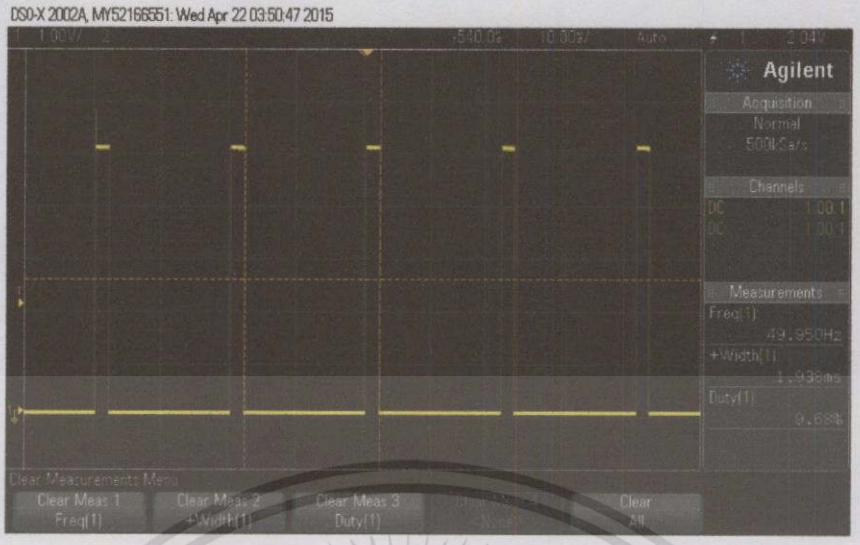
รูปที่ 4.5 กราฟค่า Duty Cycle ที่มุม 90 องศา

มีความถี่ 49.950 Hz Duty cycle 7.35% ,1.472 ms



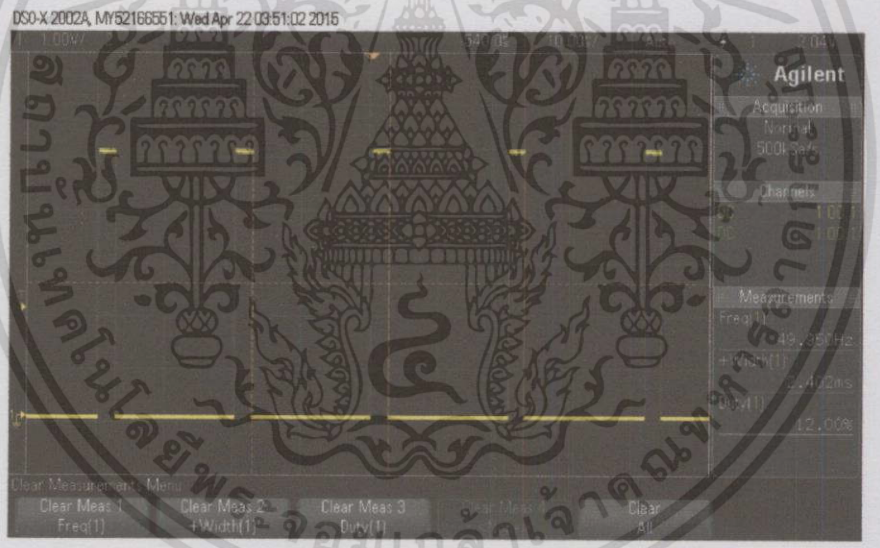
รูปที่ 4.6 กราฟค่า Duty Cycle ที่มุม 120 องศา

มีความถี่ 49.950 Hz Duty cycle 8.91% ,1.784 ms



รูปที่ 4.7 กราฟค่า Duty Cycle ที่มุม 135 องศา

มีความถี่ 49.950 Hz Duty cycle 9.68% ,1.983 ms



รูปที่ 4.8 กราฟค่า Duty Cycle ที่มุม 180 องศา

มีความถี่ 49.950 Hz Duty cycle 12.00% ,2.402 ms

ตอนที่ 2 การควบคุม Servo Motor ด้วย Arduino Board

ตารางที่ 4.1 ผลการทดลองส่วน Shoulder

มุมที่สั่ง	มุมที่เคลื่อนที่ได้	%error
0	0	0.00
45	35	28.57
90	82	9.75
135	150	10.00
180	196	8.16

ตารางที่ 4.2 ผลการทดลองส่วน Elbow

มุมที่สั่ง	มุมที่เคลื่อนที่ได้	%error
0	0	0.00
45	45	0.00
90	72	25.00
135	160	15.63
180	222	18.92

หมายเหตุ กำหนดให้มุมส่วน Shoulder อยู่ที่ 90 องศา

ตารางที่ 4.3 ผลการทดลองส่วน Wrist

มุมที่สั่ง	มุมที่เคลื่อนที่ได้	%error
0	0	0
45	44	2.27
90	76	18.42
135	159	15.09
180	210	14.29

หมายเหตุ กำหนดให้มุมส่วน Shoulder และ Elbow อยู่ที่ 90 องศา

ตอนที่ 3 การควบคุม Servo Motor ด้วย แขนของผู้ทดลอง

ตารางที่ 4.4 ผลการทดลองส่วน Shoulder

มุมที่สั่ง (องศา)	มุมที่เคลื่อนที่ได้ (องศา)	%error
0	0	0.00
45	35	14.29
90	82	9.76
135	160	15.63
180	200	15.00

ตารางที่ 4.5 ผลการทดลองส่วน Elbow

มุมที่สั่ง (องศา)	มุมที่เคลื่อนที่ได้ (องศา)	%error
0	0	0.00
45	46	2.17
90	82	9.76
135	160	15.63
180	190	5.26

หมายเหตุ กำหนดให้มุมส่วน Shoulder อยู่ที่ 90 องศา

ตารางที่ 4.6 ผลการทดลองส่วน Wrist

มุมที่สั่ง (องศา)	มุมที่เคลื่อนที่ได้ (องศา)	%error
45	65	30.77
90	80	12.50
135	190	28.95

หมายเหตุ กำหนดให้มุมส่วน Shoulder และ Elbow อยู่ที่ 90 องศา

บทที่ 5

สรุปผลการทดลองและปัญหาที่เกิด

5.1 สรุปผลการทดลอง

ตอนที่ 1 Duty Cycle ของสัญญาณพัลส์ที่จ่ายออกมาจากพอร์ทดิจิตอลของ Arduino board มีค่าแปรผันตามขนาดของมุมที่สั่งให้กับ Arduino board แต่ ความถี่ของสัญญาณ จะมีค่าคงที่

ตอนที่ 2 ที่มุม 90 องศา ส่วนของแขนกลไม่ตั้งฉากกับพื้น และ มุม 180 ส่วนของแขนกลไม่ขนานกับพื้น ในการทดลองได้ ใช้จุดที่ Servo Motor หมุนไปเมื่อสั่ง มุม 0 องศาให้กับ Arduino board เป็นจุดอ้างอิงของ Accelerometers & Gyroscope GY-521 ให้มีค่า 0 องศา แต่ Servo Motor ไม่ได้ขนานกับพื้นตรงเป๊ะ ทำให้มุมต่างๆ ที่สั่งไป มีความคลาดเคลื่อน และ Servo motor เองก็มีค่าความผิดพลาดอยู่แล้ว ทำให้ความผิดพลาดค่อนข้างมาก

ตอนที่ 3 ในการทดลองการใช้แขนของผู้ทดลองควบคุมแขนกลที่มุมต่างๆ แขนของผู้ทดลองไม่สามารถหยุดนิ่งอยู่ที่มุมที่ต้องการได้ตลอดเวลา และ ในการวัดมุมของแขนท่อนล่าง (elbow) ไม่สามารถทำให้ มุมของแขนท่อนบน (Shoulder) อยู่ที่ 90 องศาได้ตลอดเวลา ทำให้เกิดความผิดพลาดจากการวัด รวมทั้งในการทดลองวัดมุม แขนท่อนบน(shoulder) ของผู้ทดลองไม่สามารถลงมาถึงที่มุม 180 องศาได้ และ ข้อมือ (Wrist) ของผู้ทดลองไม่สามารถหมุนไปที่มุม ตามที่ต้องการได้ทั้งหมด

ตอนที่ 4 เมื่อใช้แขนของผู้ทดลองควมคุมแขนกลอย่างอิสระ พบว่า ภาพจาก กล้อง Kinect เมื่อนำมาประมวลผลกับโปรแกรม Processing ในการทำ Skeleton Tracking เส้นของ Skeleton ที่สร้างขึ้นไม่สามารถเคลื่อนที่ตามแขนของผู้ทดลองได้อย่างราบรื่นมีการกระตุก และในส่วนของแขนท่อนข้อมือ (Wrist) มีความยาวค่อนข้างน้อยทำให้ในการทดลองสังเกตการเคลื่อนที่ของ Skeleton ส่วนนี้ได้ยาก และมุมที่คำนวณได้ของท่อนข้อมือมีค่าความผิดพลาดค่อนข้างสูง

ในการทดลองทั้งการส่งข้อมูลแบบไร้สายโดยใช้ XBee และไม่ใช้ XBee แขนกลสามารถเคลื่อนที่ได้คล้ายกับแขนของผู้ทดลอง แต่แขนกลไม่สามารถเคลื่อนที่ได้เร็วเท่าแขนของผู้ทดลอง และทั้ง 2 กรณีผู้ทดลองสามารถควบคุมให้แขนกลหยิบของได้ โดยมือจับใช้คีย์บอร์ดในการควบคุม

ในการส่งข้อมูลแบบไร้สาย แขนกลจะเคลื่อนที่ไม่ราบเรียบเท่าการส่งข้อมูลผ่าน Serial Port แขนกลมีอาการเคลื่อนที่แบบสั่นและกระตุก

5.2 ปัญหาที่เกิดขึ้นและแนวทางการแก้ไข

จากการพัฒนาชิ้นงานทั้งหมดถึงแม้ว่าแขนกลจะสามารถทำตามแขนของมนุษย์ได้บางส่วน และสามารถหยิบของได้ในระยะหนึ่ง แต่ยังคงประสบกับปัญหาอีกหลายปัญหาที่ยังไม่สามารถแก้ไขได้สมบูรณ์ ดังนี้

5.2.1 ในการพัฒนาชิ้นงาน ได้ใช้กล้อง Kinect ในการจับการเคลื่อนไหวแขนของมนุษย์ และกล้อง Kinect มีข้อจำกัดในระยะการใช้งานอยู่ที่ 0.8 ถึง 1.5 เมตร ทำให้ต้องใช้พื้นที่ในการทดลอง

5.2.2 ผู้จัดทำขาดความรู้ความเข้าใจทางด้านกลอย่างลึกซึ้งทำให้ไม่สามารถแก้ปัญหาทั้งหมดที่เกิดขึ้นได้

5.2.3 ในการทดลองมุมของแขนผู้ทดลองใช้การสังเกตด้วยตาประกอบกับการคำนวณผ่านโปรแกรม Processing ทำให้การบันทึกผลการทดลองมีความคลาดเคลื่อน และ Servo Motor แต่ละตัว จะมีความผิดพลาดในตัวเอง

5.2.4 ในการเขียนโค้ดให้กับโปรแกรม Processing ประมวลผลภาพที่ได้ และทำ Skeleton Tracking ผู้จัดทำไม่ได้เขียน libraries ขึ้นมาใหม่เอง แต่ใช้ libraries ของโปรแกรม Processing และฐานข้อมูลของกล้อง Kinect ในการพัฒนาทำให้มีข้อจำกัดในการพัฒนา

5.2.5 เส้น Skeleton ที่สร้างขึ้นไม่สามารถเคลื่อนที่ตามแขนของมนุษย์ได้อย่างราบลื่น และ Skeleton ในส่วนของแขนส่วนข้อมือ(Wrist) ใช้ข้อต่อส่วนข้อมือกับมือ ในการสร้าง skeleton ทำให้เส้นมีความยาวน้อยมาก และเกิดข้อผิดพลาดสูงทำให้มุมที่แขนส่วนข้อมือ (Wrist) มีข้อจำกัดและความผิดพลาดสูง

5.2.6 ในส่วนของข้อต่อหัวไหล่ (Shoulder) ไม่สามารถเคลื่อนที่แบบหมุนได้ ทำให้เกิดข้อจำกัดในการใช้แขนกลหยิบของ และมือจับยังต้องใช้คีย์บอร์ดในการควบคุมทำให้เกิดความลำบากในการใช้งาน

5.2.7 ในการควบคุมแขนกลผู้ควบคุมต้องมีความชำนาญในการควบคุม และเคลื่อนที่แขนด้วยความเร็วไม่สูงมากเพื่อให้แขนกลสามารถทำงานได้

ซึ่งปัญหาในข้างต้นที่ที่กล่าวมาจะขอเสนอแนวทางแก้ไขบางส่วนดังนี้

ในการพัฒนา ผู้จัดทำอาจจะใช้อุปกรณ์เสริมในการลดระยะการทำงานของ Kinect ให้สามารถทำงานได้ในระยะที่ใกล้ขึ้น และในการทำ Skeleton Tracking ผู้พัฒนาอาจจะเขียน libraries ขึ้นมาใหม่เอง และศึกษาการทำงานของแขนมนุษย์เพิ่มเติม เพื่อให้แขนกลสามารถเลียนแบบพฤติกรรมของแขนมนุษย์ได้ดีขึ้น ในส่วนของ ความผิดพลาดที่เกิดขึ้นจาก Servo Motor สามารถใช้เซนเซอร์ วัดมุมที่ Servo Motor หมุนไป เพื่อทำการ Feedback ข้อมูล มาปรับค่าความผิดพลาด แต่อาจจะต้องใช้เวลาในการพัฒนา

เอกสารอ้างอิง

[1] Casey Reas, Ben Fry "Getting Started with Processing," 2010.

[2] Greg Borenstein "Making Things See," 2012.

[3] ศรีสุดา อภัยพนันธ์ "การรู้จำท่าทางการเคลื่อนไหวด้วยวิธีการ Adaptive RBFNNs (ACTION RECOGNITION WITH ADAPTIVE RBFNNs)" วิทยานิพนธ์วิศวกรรมมหาบัณฑิต สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2557.

[4] ด๊าซ ณ สงขลา "หุ่นยนต์ 6 ขา (HEXRAPOD)" ปริญญานิพนธ์วิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2551.

ภาคผนวก

โค้ดโปรแกรม Processing

```
//libralies Kinect SDK
import kinect4WinSDK.*;
import kinect4WinSDK.SkeletonData;

Kinect kinect;

ArrayList <SkeletonData> bodies;
SkeletonData _s ;

int js ;
int je ;
int jw ;
int jh ;

//for calculate true coordinate
float Width  = 640;
float Height = 480;

//libralies serial port
import processing.serial.*;
Serial port;

//coordinate from kinect 640*480
PVector kinectShoulder;
PVector kinectElbow;
PVector kinectWrist;
PVector kinectHand;

//coordinate main axis 0,0 at shoulder
PVector shoulder;
PVector elbow;
PVector wrist;
PVector hand;

//delta x,y
PVector delta1; //delta first segment
PVector delta2; //delta second segment
PVector delta3; //delta third segmentground

//angle from processing calculate
float shoulderAngleTrue;  //first segment angle
float elbowAngleTrue;     //second segment angle
float wristAngleTrue;     //third segment angle
```

```
//angle from calculate follow condition all
positive

float shoulderAngleCalculate;  //segment 1
angle

float elbowAngleCalculate;     //segment 2
angle

float wristAngleCalculate;     //segment 3
angle


//angle for calculate relative angle follow
condition

float shoulderAngleCalculateNew;  //for
calculate relative elbow angle

float elbowAngleCalculateNew1;    //for
calculate relative elbow angle

float elbowAngleCalculateNew2;    //for
calculate relative wrist angle

float wristAngleCalculateNew;     //for
calculate relative wrist angle


//relative angle segment follow condition

float elbowAngleRelative;

float wristAngleRelative;


//servo angle

float shoulderAngle;

float elbowAngle;

float wristAngle;

float handAngle;
```

```
void setup()

{

 //display 1200*700

 size(1200,480);


 //about libralies SDK for skeleton tracking

 kinect = new Kinect(this);

 bodies = new ArrayList<SkeletonData>();


 //serial port

 println(Serial.list());

 String portName = Serial.list()[0];

 port = new Serial(this, portName, 9600);


 //initial variable vector for calculate

 kinectShoulder  = new PVector();

 kinectElbow     = new PVector();

 kinectWrist     = new PVector();

 kinectHand      = new PVector();

 shoulder        = new PVector();

 elbow           = new PVector();

 wrist          = new PVector();

 hand            = new PVector();

 delta1          = new PVector();

 delta2          = new PVector();
```

```
  delta3        = new PVector();
}

void draw ()
{
  //fill white background
  background(255);

  //image from depth image
  image(kinect.GetDepth(), 0, 0);

  //frame of depth image
  noFill();
  stroke(21,241,255);
  strokeWeight(4);
  line(640,0,640,480);

  for(int i=0 ; i<bodies.size() ; i++)
  {
    _s = bodies.get(i);

    //Kinect Tracking
    js =
Kinect.NUI_SKELETON_POSITION_SHOULDER_R
IGHT;
    je =
Kinect.NUI_SKELETON_POSITION_ELBOW_RIGH
T;
    jw =
Kinect.NUI_SKELETON_POSITION_WRIST_RIGHT
;
    jh =
Kinect.NUI_SKELETON_POSITION_HAND_RIGHT;

    if( _s.skeletonPositionTrackingState[js] !=
Kinect.NUI_SKELETON_POSITION_NOT_TRACKE
D && _s.skeletonPositionTrackingState[je] !=
Kinect.NUI_SKELETON_POSITION_NOT_TRACKE
D && _s.skeletonPositionTrackingState[jw] !=
Kinect.NUI_SKELETON_POSITION_NOT_TRACKE
D && _s.skeletonPositionTrackingState[jh] !=
Kinect.NUI_SKELETON_POSITION_NOT_TRACKE
D )
    {
      //get and tranfrom coordinate
      kinectShoulder.x =
_s.skeletonPositions[js].x*Width ;
      kinectShoulder.y =
_s.skeletonPositions[js].y*Height ;
      kinectElbow.x    =
_s.skeletonPositions[je].x*Width ;
      kinectElbow.y    =
_s.skeletonPositions[je].y*Height ;
      kinectWrist.x    =
_s.skeletonPositions[jw].x*Width ;
      kinectWrist.y    =
_s.skeletonPositions[jw].y*Height ;
```

```
        kinectHand.x    =
_s.skeletonPositions[jh].x*Width ;

        kinectHand.y    =
_s.skeletonPositions[jh].y*Height ;


        //new coordinate for calculate

        shoulder.x  = kinectShoulder.x -
kinectShoulder.x;

        shoulder.y  = kinectShoulder.y -
kinectShoulder.y;

        elbow.x     = kinectElbow.x -
kinectShoulder.x;

        elbow.y     = kinectShoulder.y -
kinectElbow.y;

        wrist.x     = kinectWrist.x -
kinectShoulder.x;

        wrist.y     = kinectShoulder.y -
kinectWrist.y;

        hand.x      = kinectHand.x -
kinectShoulder.x;

        hand.y      = kinectShoulder.y -
kinectHand.y;


        //draw main axis or shoulder axis

        fill(255);

        stroke(255);

        strokeWeight(2);


line(kinectShoulder.x,kinectShoulder.y,kinectS
houlder.x+100,kinectShoulder.y);  //x-axis right

line(kinectShoulder.x,kinectShoulder.y,kinectS
houlder.x-100,kinectShoulder.y);  //x-axis left


line(kinectShoulder.x,kinectShoulder.y,kinectS
houlder.x,kinectShoulder.y-100);  //y-axis up


line(kinectShoulder.x,kinectShoulder.y,kinectS
houlder.x,kinectShoulder.y+100);  //y-axis
down


                //draw elbow axis

line(kinectElbow.x,kinectElbow.y,kinectElbow.x
+50,kinectElbow.y);  //x-axis right

line(kinectElbow.x,kinectElbow.y,kinectElbow.x
-50,kinectElbow.y);  //x-axis left

line(kinectElbow.x,kinectElbow.y,kinectElbow.x
,kinectElbow.y-50);  //y-axis up

line(kinectElbow.x,kinectElbow.y,kinectElbow.x
,kinectElbow.y+50);  //y-axis down


                //draw wrist axis

line(kinectWrist.x,kinectWrist.y,kinectWrist.x+50
,kinectWrist.y);  //x-axis right

line(kinectWrist.x,kinectWrist.y,kinectWrist.x-
50,kinectWrist.y);  //x-axis left
```

```
line(kinectWrist.x,kinectWrist.y,kinectWrist.x,kin
ectWrist.y-50);  //y-axis up

line(kinectWrist.x,kinectWrist.y,kinectWrist.x,kin
ectWrist.y+50);  //y-axis down


        //mark point at joint

        fill(255,0,0);

        stroke(255,0,0);

        strokeWeight(3);


ellipse(kinectShoulder.x,kinectShoulder.y,5,5);
//mark shoulder joint

ellipse(kinectElbow.x,kinectElbow.y,5,5);
//mark elbow joint

        ellipse(kinectWrist.x,kinectWrist.y,5,5);
//mark wrist joint

        ellipse(kinectHand.x,kinectHand.y,5,5);
//mark hand joint


        //draw segment between joint


line(kinectShoulder.x,kinectShoulder.y,kinectEl
bow.x,kinectElbow.y);  //segment between
shoulder and elbow (1)


line(kinectElbow.x,kinectElbow.y,kinectWrist.x,k
inectWrist.y);        //segment between elbow
and wrist (2)

line(kinectWrist.x,kinectWrist.y,kinectHand.x,kin
ectHand.y);          //segment between wrist
and hand (3)


            //calculate delta x,y

            delta1.x = elbow.x - shoulder.x;
//delta x first segment

            delta1.y = elbow.y - shoulder.y;
//delta y first segment

            delta2.x = wrist.x - elbow.x;     //delta
x second segment

            delta2.y = wrist.y - elbow.y;     //delta
y second segment

            delta3.x = hand.x - wrist.x;      //delta x
third segment

            delta3.x = hand.y - wrist.y;      //delta y
third segment


            //process first segment

            //shoulderAngleTrue

            shoulderAngleTrue =
degrees(atan(delta1.y/delta1.x));

            //shoulderAngleCalculate

            if(delta1.x==0 || delta1.y==0)

            {

                if(delta1.x>0) {
shoulderAngleCalculate = 0; }

                else if(delta1.x<0) {
shoulderAngleCalculate = 180; }
```

```
            else if(delta1.y>0) {
shoulderAngleCalculate = 90; }

            else if(delta1.y<0) {
shoulderAngleCalculate = 270; }

        }

      else if(delta1.y>0)

        {

            if(delta1.x>0) {
shoulderAngleCalculate = shoulderAngleTrue;
}

            else if(delta1.x<0) {
shoulderAngleCalculate = 180 +
shoulderAngleTrue; }

        }

      else if(delta1.y<0)

        {

            if(delta1.x>0) {
shoulderAngleCalculate = 360 +
shoulderAngleTrue; }

            else if(delta1.x<0) {
shoulderAngleCalculate = 180 +
shoulderAngleTrue; }

        }

        //shoulderAngle (servo)

        if(kinectShoulder.x <= kinectElbow.x)

        {

          shoulderAngle = 90 -
shoulderAngleTrue;

        }

      else if(kinectShoulder.x > kinectElbow.x)

        {

          shoulderAngle = 90 +
shoulderAngleTrue;

        }

        //constrain shoulderAngle

        shoulderAngle =
constrain(shoulderAngle,0,180);


        //process second segment

        //elbowAngleTrue

        elbowAngleTrue =
degrees(atan(delta2.y/delta2.x));

        //elbowAngleCalculate

        if(delta2.x==0 || delta2.y==0)

        {

            if(delta2.x>0) {
elbowAngleCalculate = 0; }

            else if(delta2.x<0) {
elbowAngleCalculate = 180; }

            else if(delta2.y>0) {
elbowAngleCalculate = 90; }

            else if(delta2.y<0) {
elbowAngleCalculate = 270; }

        }

      else if(delta2.y>0)

        {
```

```
            if(delta2.x>0) {
elbowAngleCalculate = elbowAngleTrue; }

            else if(delta2.x<0) {
elbowAngleCalculate = 180 +
elbowAngleTrue;}

        }

      else if(delta2.y<0)

        {

            if(delta2.x>0) {
elbowAngleCalculate = 360 +
elbowAngleTrue; }

            else if(delta2.x<0) {
elbowAngleCalculate = 180 +
elbowAngleTrue; }

        }

        //elbowAngleRelative

        if(abs(elbowAngleCalculate -
shoulderAngleCalculate) < 180)

        {

          elbowAngleRelative =
elbowAngleCalculate -
shoulderAngleCalculate;

        }

      else if(abs(elbowAngleCalculate -
shoulderAngleCalculate) >= 180)

        {

          if((elbowAngleCalculate -
shoulderAngleCalculate) > 0)

          {

            elbowAngleCalculateNew1 =
elbowAngleCalculate - 360;

            elbowAngleRelative =
elbowAngleCalculateNew1 -
shoulderAngleCalculate;

          }

        else if((elbowAngleCalculate -
shoulderAngleCalculate) < 0)

          {

            shoulderAngleCalculateNew =
shoulderAngleCalculate - 360;

            elbowAngleRelative =
elbowAngleCalculate -
shoulderAngleCalculateNew;

          }

        }

        //elbowAngle (Servo)

        elbowAngle = 90 +
elbowAngleRelative;

        //constrain elbowangle

        elbowAngle =
constrain(elbowAngle,0,180);


      //process third segment

        //wristAngleTrue

        wristAngleTrue =
degrees(atan(delta3.x/delta3.y));

        //wristAngleCalculate

        if(delta3.x==0 || delta3.y==0)
```

```
        {
            if(delta3.x>0) {
wristAngleCalculate = 0; }
        else if(delta3.x<0) {
wristAngleCalculate = 180; }
        else if(delta3.y>0) {
wristAngleCalculate = 90; }
        else if(delta3.y<0) {
wristAngleCalculate = 270; }
        }
    else if(delta3.y>0)
        {
            if(delta3.x>0) {
wristAngleCalculate = wristAngleTrue; }
        else if(delta3.x<0) {
wristAngleCalculate = 180 + wristAngleTrue;}
        }
    else if(delta3.y<0)
        {
            if(delta3.x>0) { wristAngleCalculate
= 360 + wristAngleTrue; }
        else if(delta3.x<0) {
wristAngleCalculate = 180 + wristAngleTrue; }
        }
        //wristAngleRelative
        if(abs(wristAngleCalculate -
elbowAngleCalculate) < 180)
        {
            wristAngleRelative =
wristAngleCalculate - elbowAngleCalculate;
        }
    else if(abs(wristAngleCalculate -
elbowAngleCalculate) >= 180)
        {
            if((wristAngleCalculate -
elbowAngleCalculate) > 0)
            {
                wristAngleCalculateNew =
elbowAngleCalculate - 360;
                wristAngleRelative =
wristAngleCalculateNew -
elbowAngleCalculate;
            }
        else if((wristAngleCalculate -
elbowAngleCalculate) < 0)
            {
                elbowAngleCalculateNew2 =
elbowAngleCalculate - 360;
                wristAngleRelative =
wristAngleCalculate -
elbowAngleCalculateNew2;
            }
        }
        //wristAngle (Servo)
        wristAngle = 90 - wristAngleRelative;
        //constrain wristAngle
```

```
        wristAngle =
constrain(wristAngle,0,180);


      //process hand open and close
      textSize(20);
      fill(255,0,0);
      stroke(255,0,0);
      if (keyPressed)
      {
        if (key == 'a' || key == 'A')
        {
          handAngle = 180;
        }
        else
        {
          handAngle = 120;
        }
      }
      else
      {
        handAngle = 120;
      }


      //show new coordinate at joint
      fill(255);
      stroke(255);
            textSize(12);
            text( "(" +int(shoulder.x)+ ","
+int(shoulder.y)+ ")" ,kinectShoulder.x-
55,kinectShoulder.y);     // Show New (x,y)
Shoulder
            text( "(" +int(elbow.x)+ ","
+int(elbow.y)+ ")"
,kinectElbow.x+10,kinectElbow.y+10);
// Show New (x,y) Elbow
            text( "(" +int(wrist.x)+ "," +int(wrist.y)+ ")"
,kinectWrist.x+10,kinectWrist.y+10);
// Show New (x,y) Wrist
            text( "(" +int(hand.x)+ "," +int(hand.y)+
")" ,kinectHand.x+10,kinectHand.y+10);
// Show New (x,y) Hand

            //show angle
            textSize(20);
            //show shoulderAngle
            fill(90,255,32);
            stroke(90,255,32);
            text("Shoulder Angle : ",700,30);
            text(+int(shoulderAngle),900,30);
            //show elbowAngle
            fill(255,183,29);
            stroke(255,183,29);
            text("Elbow Angle : ",700,60);
            text(+int(elbowAngle),900,60);
            //show wristAngle
```

```
    fill(239,82,222);
    stroke(239,82,222);
    text("Wrist Angle : ",700,90);
    text(+int(wristAngle),900,90);

  //show condition of hand
  fill(255,0,0);
  stroke(255,0,0);
  if(handAngle == 180)
  {
    text("Hand Close",700,120);
  }
  else
  {
    text("Hand Open",700,120);
  }

  //serial port send value of servo angle
to arduino
  byte[] outAngle = new byte[4];
  outAngle[0] = byte(int(shoulderAngle));
  outAngle[1] = byte(int(elbowAngle));
  outAngle[2] = byte(int(wristAngle));
  outAngle[3] = byte(int(handAngle));
  port.write(outAngle);
 }
  }
}

//function's WinKinectSDK libralies for skeleton
tracking
void appearEvent(SkeletonData _s)
{
 if (_s.trackingState ==
Kinect.NUI_SKELETON_NOT_TRACKED)
 {
  return;
 }
 synchronized(bodies)
 {
  bodies.add(_s);
 }
}

void disappearEvent(SkeletonData _s)
{
 synchronized(bodies)
 {
```

```
      for (int i=bodies.size ()-1; i>=0; i--)
      {
        if (_s.dwTrackingID ==
bodies.get(i).dwTrackingID)
        {
          bodies.remove(i);
        }
      }
    }
  }
}
```

```
void moveEvent(SkeletonData _b,
SkeletonData _a)
{
 if (_a.trackingState ==
Kinect.NUI_SKELETON_NOT_TRACKED)
 {
   return;
 }

 synchronized(bodies)
 {
   for (int i=bodies.size ()-1; i>=0; i--)
   {
     if (_b.dwTrackingID ==
bodies.get(i).dwTrackingID)
     {
       bodies.get(i).copy(_a);
       break;
     }
   }
 }
}
```

โค้ดโปรแกรม Arduino

```
#include <Servo.h>

//variable of servo
Servo shoulderRotate; //Fix
Servo shoulder1;
Servo shoulder2;
Servo elbow;
```

```
Servo wrist;
Servo handRotate; //Fix
Servo hand;

//variable recive data from serial port and
angle of servo
int nextServo = 0;
int servoAngles[] = {95,90,90,120};
//0=shoulderAngle 1=elbowAngle
2=wristAngle 3=handAngle

int shoulderRotateAngle = 0;
int handRotateAngle = 180;

void setup()
{
 //pin of servo
 shoulderRotate.attach(4);
 shoulder1.attach(5);
 shoulder2.attach(6);
 elbow.attach(8);
 wrist.attach(9);
 handRotate.attach(7);
 hand.attach(10);

//serial port
Serial.begin(9600);
}

void loop()
{

 //Servo fix
 shoulderRotate.write(shoulderRotateAngle);
 handRotate.write(handRotateAngle);

 if(Serial.available())
 {
 int servoAngle = Serial.read();
 servoAngles[nextServo] = servoAngle;
 nextServo++;
 if(nextServo > 3)
 {
  nextServo = 0;
 }

 shoulder1.write(servoAngles[0]);
 shoulder2.write(servoAngles[0]);
 elbow.write(servoAngles[1]);
```

```
    wrist.write(servoAngles[2]);
    hand.write(servoAngles[3]);
  }
}
```