หุ่นยนต์หกขาควบคุมด้วยบลูทูธ

BLUETOOTH-CONTROLLED HEXAPOD ROBOT

โดย

นายทศพร เตชะศักดิ์

นายพันธกานต์ โนนจุ้ย

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมแมคคาทรอนิกส์ ภาควิชาวิศวกรรมระบบควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

BLUETOOTH-CONTROLLED HEXAPOD ROBOT

TODSAPORN TACHASAK

PANTAKAN NONJUI

THIS PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT

FOR THE BACHELOR DEGREE IN MECHATRONIC ENGINEERING

DEPARTMENT OF CONTROL ENGINEERING FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2020

ปีการศึกษา 2563

หุ่นยนต์หกขาควบคุมด้วยบลูทูธ

BLUETOOTH-CONTROLLED HEXAPOD ROBOT

โดย

นายทศพร เตชะศักดิ์

นายพันธกานต์ โนนจุ้ย

อาจารย์ที่ปรึกษา

ผศ.ดร.คงศักดิ์ อนันตหิรัญรัตน์

ผศ.เทพจิตร์ เชยโภคา

โครงงานพิเศษปีการศึกษา 2563

สาขาวิศวกรรมแมคคาทรอนิกส์

ภาควิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง หุ่นยนต์หกขาควบคุมด้วยบลูทูธ

ผู้จัดทำ

1. นายทศพร เตชะศักดิ์
2. นายพันธกานต์ โนนจุ้ย

..... อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ ดร.คงศักดิ์ อนันตหิรัญรัตน์)

..... อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ เทพจิตร์ เชยโภคา)

หุ่นยนต์หกขาควบคุมด้วยบลูทูธ

นาย ทศพร เตชะศักดิ์
นาย พันธกานต์ โนนจุ้ย
ผศ.ดร. คงศักดิ์ อนันตหิรัญรัตน์ อาจารย์ที่ปรึกษา
ผศ. เทพจิตร์ เชยโภคา
ปีการศึกษา 2563

บทคัดย่อ

เนื่องจากในปัจจุบันมีงานใหม่ๆ มากมายเกิดขึ้น บางงานที่เกิดขึ้นอาจจะอันตรายพอสมควร อย่างเช่น งานที่ต้องจัดการในที่ขรุขระเกินกว่าที่คนจะสามารถเดินผ่านได้และไม่สามารถผ่านได้ด้วยหุ่นยนต์แบบล้อ เช่น หน่วยกู้ภัยที่ต้องไปช่วยคนที่ติดบนภูเขา หรือ ในถ้ำ หุ่นยนต์แมงมุมถูกออกแบบมาให้สามารถเดินทางไปในสถานที่ที่ไม่เรียบเนียนได้ โดยที่ตัวของหุ่นยนต์นั้นยังอยู่ในสภาพเสถียรอยู่

เป้าหมายของโปรเจคนี้คือการสร้างหุ่นยนต์หกขาที่สามารถทำกิริยาพื้นฐานได้ เช่น การเดินไปข้างหน้า การเดินถอยหลัง การเลี้ยงตัว และการเดินไปด้านข้าง โดยหุ่นยนต์นี้จะเป็นหุ่นยนต์แมงมุมที่มี 18 Degrees of Freedom ที่มี 6 ขา โดยแต่ละขาจะมี 3 Degrees of Freedom

โปรเจคนี้ถูกสร้างขึ้นมา เพื่อเรียนรู้ลักษณะ และความสามารถทางฟิสิกส์ของหุ่นยนต์แมงมุม สำหรับนำไปต่อยอดในอนาคตต่อไป

BLUETOOTH-CONTROLLED HEXAPOD ROBOT

Mr. TODSAPORN TACHASAK
Mr. PANTAKAN NONJUI
Asst. Prof. Dr.KONGSAK ANUNTAHIRUNRAT Advisor
Asst. Prof. Thepjit Cheypoca
Year 2020

ABSTRACT

Due to a lot of new jobs that happen in the present, there are some jobs that might be a little too dangerous for human to do it themselves such as, jobs that work in the rough ground such as rescuer. The Hexapod is designed to be able to travel in the rough place which wheeled-robot is unable to do due to its rough ground.

The main aim of this project is to create hexapod that can perform basic task, such as walking forward, backward, rotation and sidewalk. This robot would serve as an 18-DOF spider robot consist of 6 legs each one have three degrees of freedom.

The project is created to learn the nature and physical capability of hexapod robot for the future improvement.

กิตติกรรมประกาศ

โครงงานพิเศษฉบับนี้สำเร็จลุล่วงด้วยดีเนื่องจากได้รับความกรุณาอย่างสูงจาก ผู้ช่วยศาสตราจารย์ ดร.คงศักดิ์ อนันตหิรัญรัตน์ และ ผู้ช่วยศาสตราจารย์เทพจิตร์ เชยโภคา อาจารย์ที่ปรึกษาที่กรุณาให้คำแนะนำปรึกษา สนับสนุนในเรื่องอุปกรณ์และพื้นที่สำหรับใช้ในการทำโครงงาน ตลอดจนปรับปรุงแก้ไขข้อบกพร่องต่างๆ ด้วยความเอาใจใส่อย่างดียิ่ง คณะผู้จัดทำได้ตระหนักถึงความทุ่มเทและความตั้งใจจริงของอาจารย์และขอกราบขอบพระคุณเป็นอย่างสูงไว้ ณ ที่นี้

ขอขอบคุณ เหล่าเพื่อนๆ ในคณะ ที่ช่วยให้คำปรึกษาเกี่ยวกับโครงงานพิเศษนี้ รวมทั้งเป็นกำลังใจที่ดีเสมอมา

ท้ายนี้คณะผู้จัดทำขอกราบขอบพระคุณ บิดามารดาและครอบครัว ซึ่งให้การช่วยเหลือทางด้านทุนทรัพย์และให้กำลังใจตลอดมา สำหรับข้อบกพร่องต่างๆ ที่อาจจะเกิดขึ้นนั้น คณะผู้จัดทำขอน้อมรับผิดและยินดีที่จะรับฟังคำแนะนำจากทุกท่านที่ได้เข้ามาศึกษา เพื่อเป็นประโยชน์ในการพัฒนางานวิจัยต่อไป

คณะผู้จัดทำ

นายทศพร เตชะศักดิ์

นายพันธกานต์ โนนจุ้ย

สารบัญ

หน้า

หน้า

สารบัญรูป

หน้า

บทที่ 1
บทนำ

ความเป็นมาและความสำคัญ

ปัจจุบันโลกได้มีการเปลี่ยนแปลงเป็นอย่างมาก การงานอาชีพเองนั้นก็มีหลากหลายขึ้น และมีการระบาดของโรคติดเชื้อไวรัสโคโรนาสายพันธุ์ใหม่ 2019 ด้วย การทำงานหลายๆ อย่างจึงยุ่งยากขึ้น ในงานที่ผู้ปฏิบัติงานนั้นต้องเสี่ยงอันตราย เช่น อาสากู้ภัย หรือ นักสำรวจที่ต้องเดินทางไปในพื้นที่อันตรายจากสภาพพื้นที่แย่ หรือ สภาพพื้นที่ที่อาจมีอันตรายอย่างอื่นอยู่ การเข้าไปทำงานในพื้นที่อันตรายเหล่านั้นด้วยตัวเองนั้นอาจเป็นเรื่องที่สุ่มเสี่ยงเกินไป และอาจเกิดการสูญเสียคนงานหรืออุปกรณ์ได้

จากสถานการณ์ที่เกิดขึ้นนี้ เราจึงเลือกที่จะลองสร้างหุ่นยนต์แมงมุม หรือ Hexapod ขึ้นมา เนื่องจากว่าโครงสร้างของหุ่นยนต์แมงมุมนี้ ทำให้ตัวหุ่นยนต์สามารถเดินทางไปในสถานที่วิบากได้โดยไม่เสียสมดุล เหมาะกับการทำงานในสถานที่ที่ขรุขระ และอันตรายได้ เนื่องจากขาแบบพิเศษของหุ่นยนต์แมงมุมนี้ซึ่งมีสองข้อต่อ ต่างจากหุ่นยนต์แบบล้อที่สามารถเสียสมดุลได้ง่ายๆ ในพื้นที่ที่ขรุขระ นอกจากจะออกแบบเป็นพิเศษสำหรับงานนั้น

ในโครงงานพิเศษที่เราทำนี้เป็นเพียงแค่รูปแบบหุ่นยนต์พื้นฐานเท่านั้น แต่หุ่นยนต์แมงมุมนี้ยังคงสามารถต่อยอดได้อีก เพื่อใช้ในสถานการณ์เฉพาะต่างๆ อาจใช้เป็นหุ่นยนต์ส่งของ หุ่นยนต์หยิบจับ เป็นต้น ซึ่งหุ่นยนต์แมงมุมนี้อาจทำให้งานของผู้ที่ไปเสี่ยงอันตรายมีความปลอดภัยมากยิ่งขึ้น

วัตถุประสงค์การวิจัย

เพื่อสร้างหุ่นยนต์หกขาที่สามารถเดินได้ และถูกควบคุมด้วยรีโมตบังคับจากระยะไกล

ขอบเขตการวิจัย

ความสามารถในการเคลื่อนที่ของหุ่นยนต์ในช่วงเวลา 1 ปีการศึกษา โดยใช้โปรแกรม Arduino และ Solidworks

วิธีการดำเนินการวิจัย

1.4.1 กำหนดหัวข้อ และ วางแผนงาน

1.4.2 ศึกษาหาความรู้ที่จำนำมาทำโปรเจค

1.4.3 ออกแบบอุปกรณ์ของหุ่นยนต์หกขา

1.4.4 ออกแบบโปรแกรมการเคลื่อนที่ของหุ่นยนต์หกขา

1.4.5 ทดสอบโปรแกรมการเคลื่อนที่กับอุปกรณ์ที่ออกแบบไว้แล้ว

1.4.6 ทดสอบหุ่นยนต์หกขา

1.4.7 รวบรวมข้อมูลสรุปงานวิจัยและจัดทำรูปเล่ม

ประโยชน์ที่คาดว่าจะได้รับ

1.5.1 สามารถสร้างหุ่นยนต์ที่สามารถเคลื่อนที่ได้อย่างยืดหยุ่น

1.5.2 สามารถออกแบบหุ่นยนต์ให้ควบคุมด้วยรีโมตจากระยะไกลได้

1.5.3 เข้าใจ Inverse Kinematics และทฤษฎีที่เกี่ยวข้องในตอนออกแบบโปรแกรมนี้

บทที่ 2
แนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้อง

2.1 การจำลองการเคลื่อนที่

การจำลองการเคลื่อนที่ คือ การสร้างระบบการเคลื่อนที่ผ่านสมการคณิตศาสตร์ในโปรแกรมที่กำหนด เพื่อแสดงรูปแบบการทำงาน และการเคลื่อนที่ของหุ่นยนต์

2.1.1 ทฤษฎีที่เกี่ยวข้อง

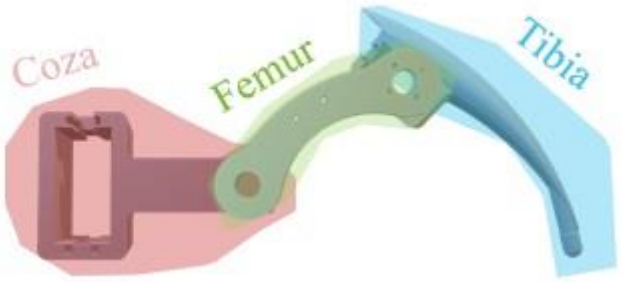
2.1.1.1 Kinematics of mechanisms หรือ จลนศาสตร์ของเครื่องกล คือ การสนใจการเคลื่อนที่ของวัตถุ โดยไม่สนใจอิทธิพลใดๆ ที่ส่งผลต่อวัตถุนั้น (เช่น แรงหรือมวล) เพราะฉะนั้นจลนศาสตร์นั้นจะสนใจเพียงแค่คอนเซปต์พื้นฐานเท่านั้น

2.1.1.2 Dynamics of mechanisms หรือ พลศาสตร์ของเครื่องกล คือ การสนใจแรงที่เกิดขึ้นต่อวัตถุจากแรงทั้งแบบสมดุล และไม่สมดุล โดยสนใจทั้งน้ำหนักของวัตถุ และความเร่งที่เกิดขึ้นกับวัตถุ รวมไปถึงแรงจากภายนอกด้วย

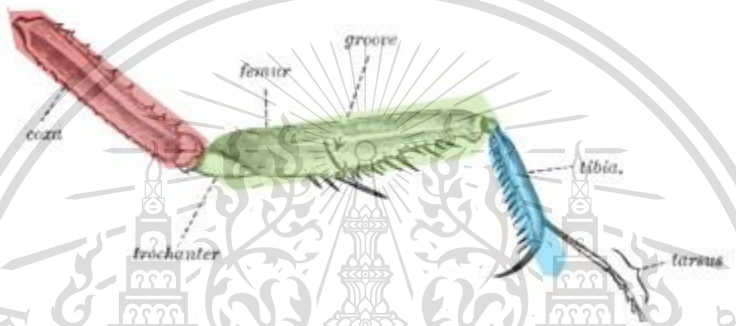
2.1.1.3 Inverse Kinematics คือ กระบวนการทางคณิตศาสตร์สำหรับคำนวณค่าของมุมของข้อต่อที่ต้องการ เพื่อจัดวางตำแหน่งปลายของจุดเชื่อมต่อทางจลนศาสตร์ เช่น การควบคุมแขนหรือขาของหุ่นยนต์ หรือการควบคุมกระดูกของตัวละครในอนิเมชันในตำแหน่งที่เหมาะสมกับจุดเริ่มต้นของจุดเชื่อมต่อนั้น โดยการหา Inverse Kinematics นั้นสามารถหาได้ เมื่อมีการให้ความยาวของขาแต่ละส่วนที่เชื่อมต่อกัน

รูปที่ 2.1 ตัวอย่างรูปแบบขาของหุ่นยนต์แมงมุม

จากรูปที่ 2.1 แสดงให้เห็นถึงตัวอย่างรูปแบบขาของหุ่นยนต์หกขานี้ ซึ่งโดยทั่วไปแล้วสามารถแบ่งออกได้เป็นสามส่วน โดยเราต้องหาความยาวของของชิ้นส่วนทั้งสาม เพื่อใช้หา Inverse Kinematics

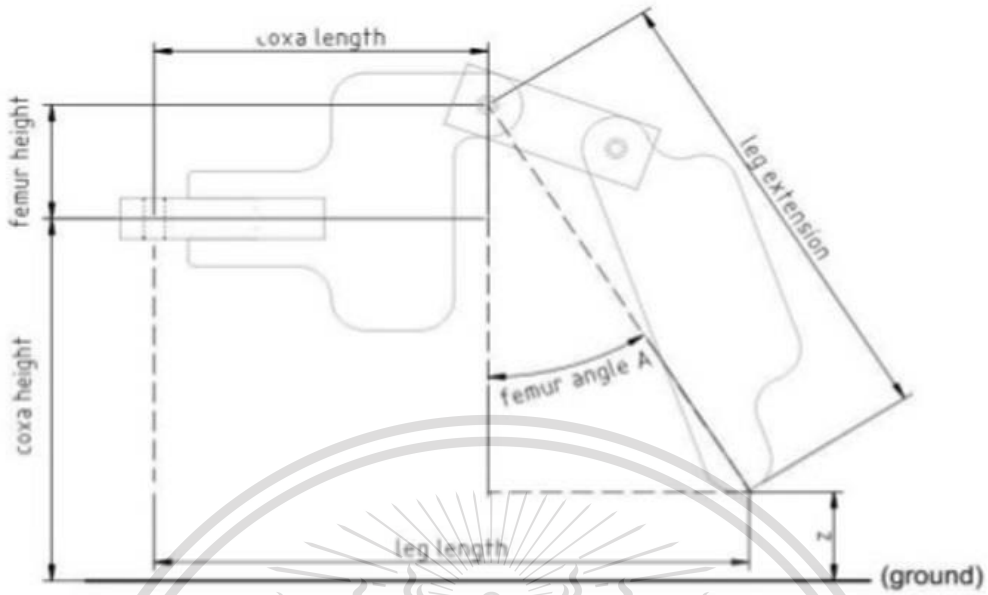


รูปที่ 2.2 แสดงโครงสร้างของขาหุ่นยนต์



รูปที่ 2.3 แสดงโครงสร้างขาของแมลง

รูปที่ 2.2 และรูปที่ 2.3 แสดงการเปรียบเทียบกันระหว่างขาของแมลงกับขาของหุ่นยนต์แมงมุม ขาของหุ่นยนต์มีต้นแบบมาจากขาของแมลง โดย Coza คือส่วนของต้นขาที่ติดอยู่กับลำตัวหลักของตัวหุ่นยนต์ Femur คือส่วนที่เชื่อมต่อระหว่างขาส่วนบน Coza กับ ขาส่วนล่าง Tibia โดย Femur นี้ ทำให้ตัวหุ่นยนต์สามารถเดินได้อย่างยืดหยุ่นมากกว่า เนื่องจากมีข้อต่ออยู่ถึง 2 ข้อ เมื่อเทียบกับขาของคนซึ่งมีข้อต่อแค่ข้อเดียว ส่วนสุดท้ายคือ Tibia ซึ่งเป็นส่วนที่สัมผัสอยู่กับพื้น

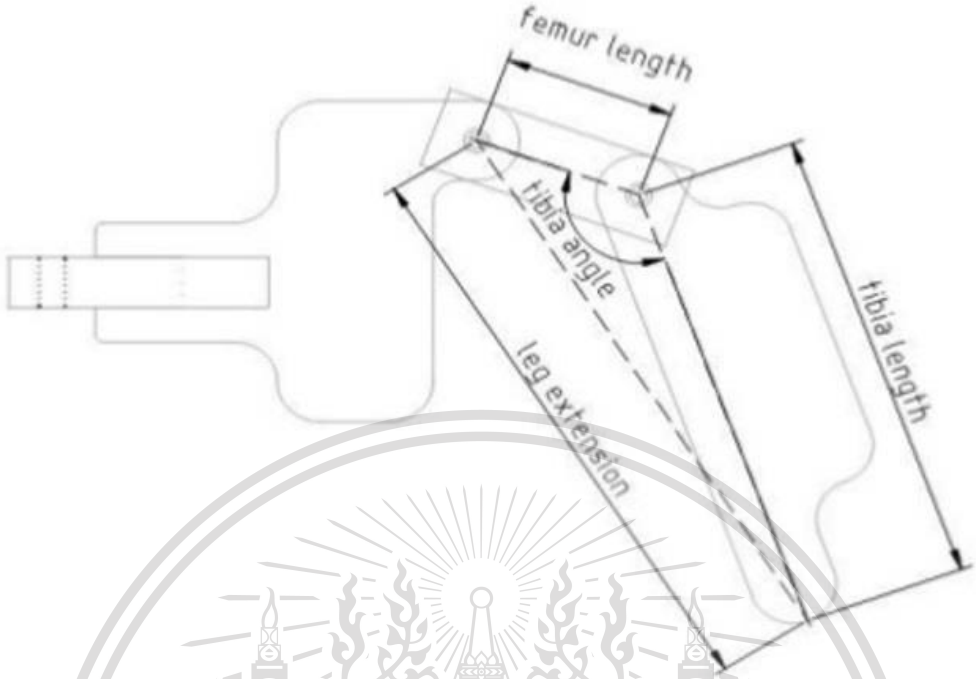


รูปที่ 2.4

รูปที่ 2.4 แสดงข้อมูลต่างๆ สำหรับการหามุม Femur A เพื่อหาความสามารถในการกางขาของหุ่นยนต์ โดยใช้สูตร ดังนี้

$$\text{Leg Extension} = \sqrt{(femur\ height + coxa\ height\ - z)^2 + (leg\ length - coxa\ length)^2}$$

$$\text{A} = tan^{-1}\left(\frac{Leg\ length - coxa\ length}{Coxa\ height + femur\ height - Z}\right)$$



รูปที่ 2.5

รูปที่ 2.5 แสดงข้อมูลต่างๆ สำหรับการหามุม Tibia เพื่อหาความกว้างที่ขาส่วน tibia จะสามารถกางออกไปได้ โดยหาได้จากสูตร ดังนี้

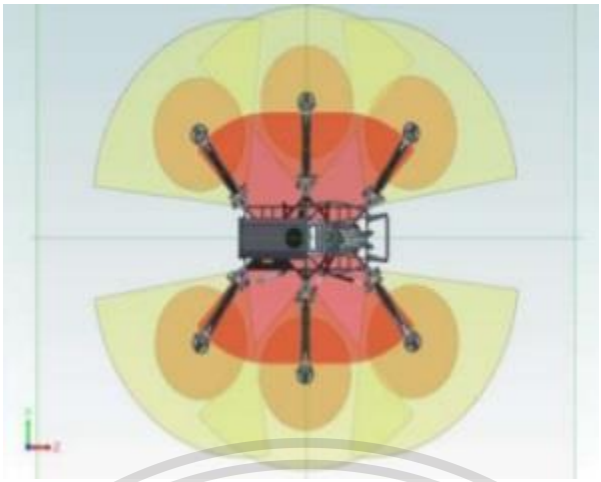
$$\mathrm{B} = tan^{-1}\left(\frac{Tibia\ length}{Leg\ extension}\right)$$

$$\text{Tibia angle} = tan^{-1}\left(\frac{Leg\ extension}{Femur\ length}\right)$$

2.1.1.4 Degree of Freedom หรือ องศาเสรี คือ จำนวนตัวแปรที่จำเป็นสำหรับการอธิบายระบบนั้นๆ ได้อย่างครบถ้วน โดยในทางกลศาสตร์หมายถึงความสามารถในการเคลื่อนที่ของวัตถุว่าอิสระแค่ไหน โดยสามารถหาได้จากสมการของ Grubler นั่นคือ

$$n = 3(l-1)-2j-h$$

2.1.1.5 Work Envelope คือ ความสามารถในการเคลื่อนที่ของหุ่นยนต์



รูปที่ 2.6 ความสามารถในการเคลื่อนที่ของหุ่นยนต์แมงมุม

จากรูปที่ 2.6 พื้นที่สีแดงแสดงให้เห็นพื้นที่ที่ขาของหุ่นยนต์ไม่สามารถเดินทางไปได้ พื้นที่สีส้มแสดงถึงพื้นที่ที่ขาของหุ่นยนต์สามารถเดินไปถึงได้ และพื้นที่สีเหลืองแสดงถึงพื้นที่ที่ขาของหุ่นยนต์มีปฏิกิริยาต่อสิ่งรอบข้าง

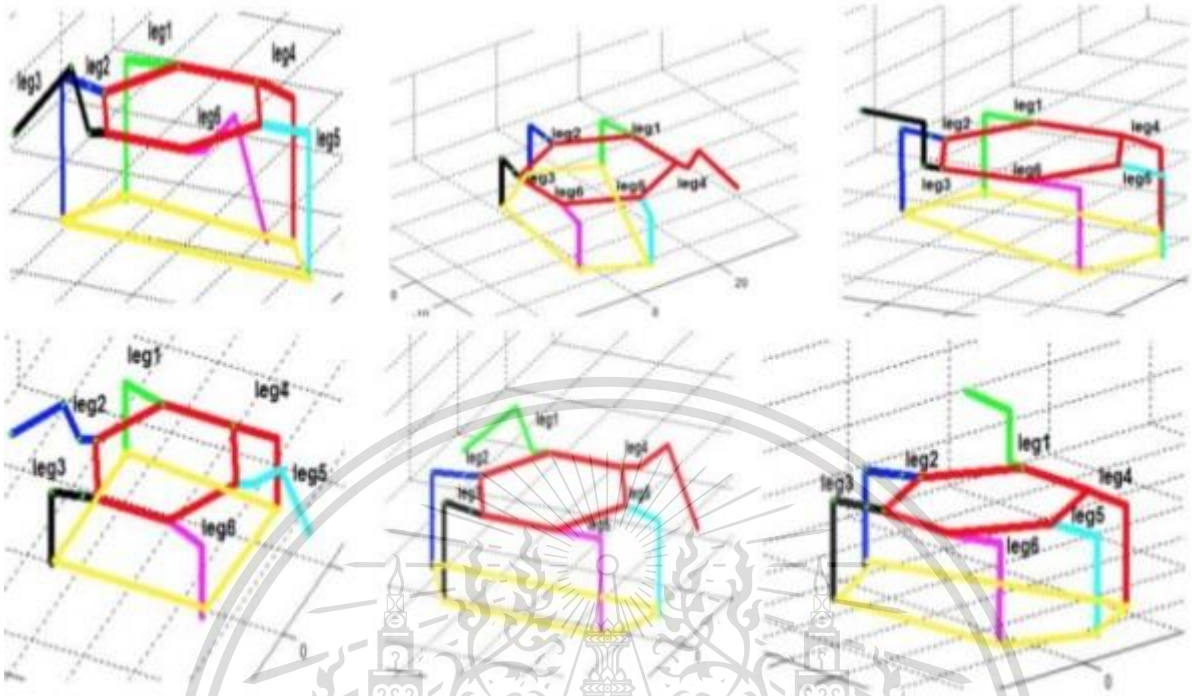
2.1.1.6 Dynamic Stability หุ่นยนต์ที่เดินอยู่นั้นต้องมีความสมดุลทางพลศาสตร์ เพื่อที่จะให้หุ่นยนต์นั้นไม่ล้มลงมา ถ้าหุ่นยนต์นั้นจะหยุดเดิน ศูนย์กลางของมวลหุ่นยนต์นั้นจะทำให้หุ่นยนต์นั้นล้มลงมา

2.1.1.7 Static Stability หุ่นยนต์ที่มีความสมดุลทางจลนศาสตร์จะสามารถหยุดตอนไหนก็ได้ระหว่างการเดินของมัน ทำให้หุ่นยนต์นั้นจะไม่ล้มลงมา ในกรณีของหุ่นยนต์แมงมุมนี้ ตราบเท่าที่ขา 3 ขาของหุ่นยนต์นี้ยังอยู่บนพื้น และศูนย์กลางของมวลอยู่นำตำแหน่งสามเหลี่ยมที่ถูกสร้างจากจุด 3 จุดที่ขาหุ่นยนต์แต่ละข้างนี้ หุ่นยนต์จะอยู่ในสภาพสมดุลสถิตตลอด ดังแสดงในรูปที่ 2.7

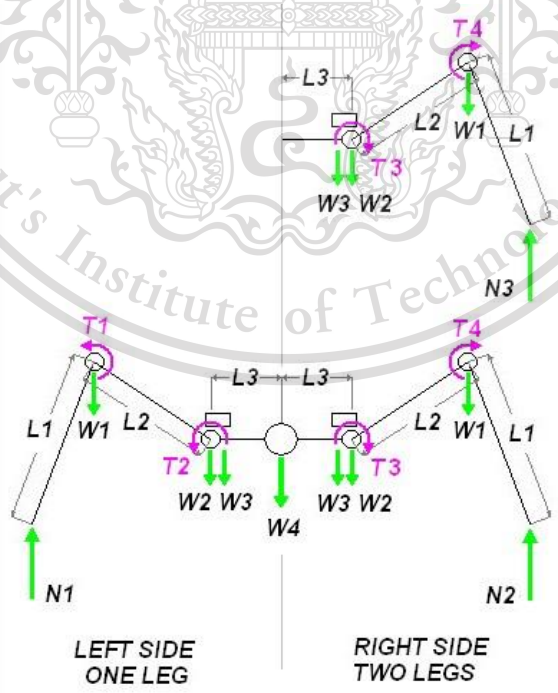
จากรูปที่ 2.9 แสดงตัวอย่างความสมดุลสถิตที่เกิดขึ้น โดยในรูปแรกมีขาสี่ขาอยู่บนพื้นซึ่งสร้างเป็นรูปสี่เหลี่ยมขึ้นมา และศูนย์กลางของมวลอยู่ในสี่เหลี่ยมนั้น หุ่นยนต์นั้นจึงสมดุล

รูปที่ 2 ถึงหุ่นยนต์จะยืนสามขาก็จริง แต่ว่าศูนย์กลางมวลของหุ่นยนต์นั้นไม่ได้อยู่บนพื้นที่ที่สร้างมาบนพื้นหุ่นยนต์นั้นจึงอยู่ในสภาพไม่สมดุลสถิต และทำให้หุ่นยนต์นั้นล้มได้

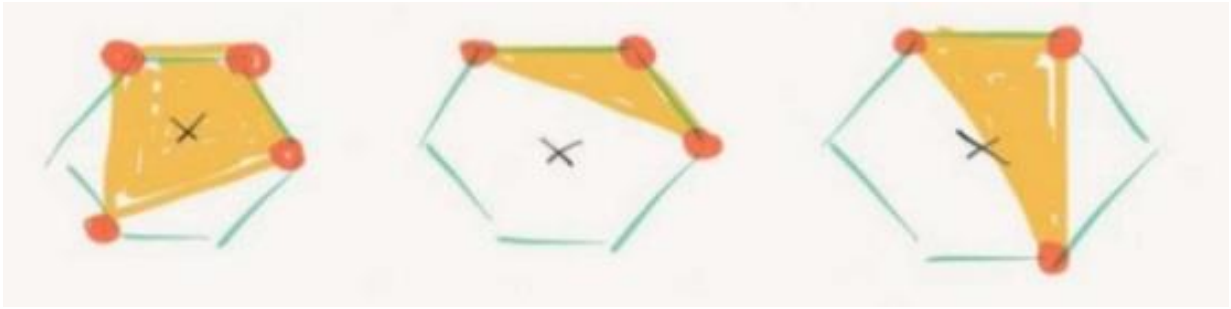
รูปที่ 3 ศูนย์กลางของหุ่นยนต์อยู่ตรงขอบของพื้นที่ขาของหุ่นยนต์นั้นพอดี ยังมีเกิดความสมดุลสถิตอยู่



รูปที่ 2.7 เงื่อนไขของสภาพสมดุลสถิตของหุ่นยนต์แมงมุม



รูปที่ 2.8 แรงที่เกิดขึ้นบนตัวของหุ่นยนต์



รูปที่ 2.9 ตัวอย่างของความสมดุลของหุ่นยนต์

บทที่ 3
วิธีดำเนินงานวิจัย

3.1 กำหนัดหัวข้อ และ วางแผนงาน

ในสัปดาห์แรกได้ประชุมปรึกษาหาหัวข้อโครงงานพิเศษ และวางแผนทำงานเบื้องต้น โดยหารูปแบบหุ่นยนต์แมงมุมที่ต้องการจะทำ และสังเกตรูปแบบการทำงานของหุ่นยนต์แมงมุมนั้นๆ

3.2 ศึกษาหาความรู้ที่เพื่อนำมาใช้ทำโปรเจค

โปรแกรมหลักๆ ที่ต้องศึกษาในการทำโปรเจคนี้ คือ โปรแกรม Solidworks และ Arduino โดย Solidworks นั้นมีไว้สำหรับออกแบบส่วนประกอบของหุ่นยนต์แมงมุมนี้ และโปรแกรม Arduino ใช้สำหรับเขียนโปรแกรมการเคลื่อนไหวของหุ่นยนต์

3.3 ออกแบบส่วนประกอบของหุ่นยนต์

ใช้โปรแกรม Solidworks ในการออกแบบชิ้นส่วนต่างๆ ของหุ่นยนต์ โดยศึกษาจากต้นแบบหุ่นยนต์แมงมุมต่างๆ ในฐานข้อมูล เพื่อให้ได้หุ่นยนต์ที่มีรูปร่างไม่ผิดหลักกลศาสตร์

3.4 เขียนโปรแกรมการเคลื่อนไหวของหุ่นยนต์

เขียนโปรแกรมใน Arduino โดยใช้พื้นฐานที่เรียนรู้จากในอินเตอร์เน็ตมาช่วย ในส่วนของโค้ดนั้นจะพูดถึงในภาคผนวก ก.

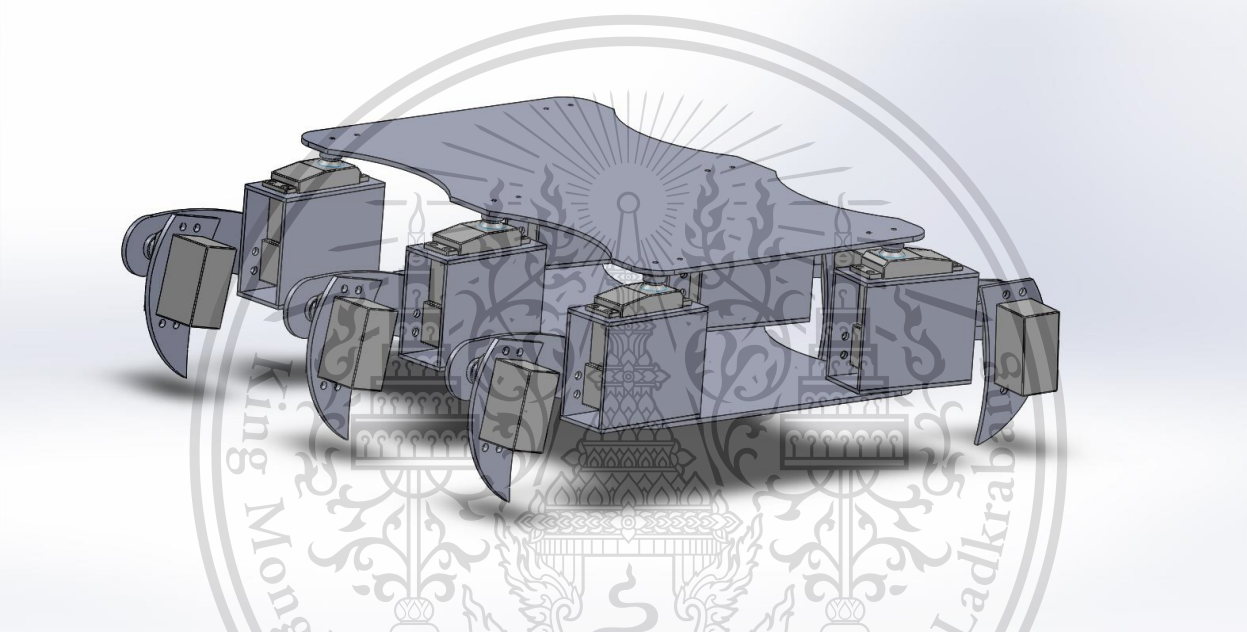
บทที่ 4

ผลการดำเนินงาน

ส่วนของ Software

ส่วนของ Software จะมีการพูดถึงในภาคผนวก ก.

ส่วนของ Mechanics



รูปที่ 4.1

รูปที่ 4.1 นี้ คือ เป้าหมายหุ่นยนต์แมงมุมที่เราต้องการจะสร้าง แต่เนื่องจากสถานการณ์ที่ไม่ปกติ ทำให้ทางเราไม่สามารถหารูปของหุ่นยนต์ที่เราทำไว้ได้

บทที่ 5

สรุปผลการดำเนินงานและข้อเสนอแนะ

5.1 สรุปผลการดำเนินการ

จากสถานการณ์ไม่ปกติที่เกิดขึ้น ทำให้ผมไม่สามารถทดสอบตัวโปรแกรมกับหุ่นยนต์ได้ ในสรุปผลการดำเนินการนี้จึงจะพูดถึงเพียงเรื่องของตัวโปรแกรม

ส่วนแรก โปรแกรมหลักของ Hexapod คือ โปรแกรมที่เริ่มต้นปฏิบัติการทุกอย่างของระบบ

ส่วนที่สอง คือ ตัววัดค่าก่อนที่ระบบโปรแกรมหลักของ Hexapod จะทำงาน ตัวโปรแกรมนี้จะทำการวัดหาค่าเริ่มต้นก่อนเราจะเริ่มการใช้โปรแกรมหลักของ Hexapod โดยสำหรับหุ่นยนต์ที่มีเซอร์โวมอเตอร์ 18 ตัวที่เชื่อมต่อกับบอร์ด Arduino นั้น การวัดค่านี้จะถูกเก็บค่าไว้ในเมม EEPROM ภายในของบอร์ดนั้น โดยในการทดสอบนั้น ได้มีการตั้งค่าฟังก์ชันเพื่อให้รู้ค่าจริงของเมม โดยฟังก์ชันนั้นคือ

currentCalibration() ซึ่งถูกส่งโดยพอร์ตต่อเนื่องจากมุมทั้งหมดที่ถูกวัด
writeCalibrationCompleted() ใช้เขียนไบต์ใน EEPROM เพื่อกำหนดการวัดที่สมบูรณ์แล้ว
tryCalibrationCompleted() จะส่งกลับว่าจริงถ้าการวัดนั้นเสร็จสมบูรณ์

ส่วนที่สาม คือ การควบคุม ROS สำหรับหุ่นยนต์แมงมุม โดยเมื่อโปรแกรมนี้ถูกใส่ไว้รวมกับโปรแกรมหลักของ Hexapod โปรแกรมจะอนุญาตให้เราควบคุมตัวหุ่นยนต์ได้โดยการใช้พอร์ตต่อเนื่องของบอร์ด Arduino เช่น โนด ROS แต่ว่าการควบคุมนี้ทำงานขัดกับระบบควบคุมต่อเนื่อง

ส่วนที่สี่ คือ ระบบควบคุมต่อเนื่องสำหรับหุ่นยนต์แมงมุม โดยเมื่อโปรแกรมนี้ถูกใส่ไว้รวมกับโปรแกรมหลักของ Hexapod โปรแกรมจะอนุญาตให้เราควบคุมหุ่นยนต์ได้โดยการใช้พอร์ตต่อเนื่องของบอร์ด Arduino แต่ว่าการควบคุมนี้ทำงานขัดกับระบบควบคุม ROS

ส่วนที่ห้า คือ ส่วนหลักของการควบคุมหุ่นยนต์แมงมุม โปรแกรมส่วนนี้สนับสนุนการควบคุมต่อเนื่องโดยการใช้พอร์ตสื่อสารต่อเนื่องหรือคุณสามารถแนบหุ่นยนต์แมงมุมเป็นโนดของระบบ ROS และควบคุมมันด้วยระบบปฏิบัติการนั้น

ส่วนที่หก คือ ส่วนควบคุมขาหุ่นยนต์แมงมุม ส่วนโปรแกรมของขานี้สามารถใช้เดี่ยวๆ หรือ ใช้ควบคุมทั้งหกขาก็ได้

ส่วนที่เจ็ด คือ ฟังก์ชัน log ทั่วไป กลุ่มของฟังก์ชัน log ทั่วไปที่ถูกแบ่งอย่างเป็นลำดับขั้นโดยสิ่งที่เกี่ยวข้องนั่นคือ ข้อผิดพลาด คำเตือน ข้อมูล และข้อความดีบัก ยิ่งค่า ‘level_log’ มากขึ้น ฟังก์ชันที่เราเปิดไว้ก็ยิ่งมากขึ้น

เช่นกัน ‘level_log=1’ หมายความว่าข้อความแสดงข้อผิดพลาดจะแสดงขึ้นเท่านั้น ขณะที่ ‘level_log=3’ หมายความว่า ความผิดพลาด คำเตือน และข้อความแสดงข้อมูลเปิดขึ้นมา

ส่วนที่แปด คือ ฟังก์ชันตรีโกณมิติปรับค่าสำหรับ Inverse Kinematics ของหุ่นยนต์แมงมุม

5.2 ข้อเสนอแนะ

5.2.1 ควรนำไปพัฒนาให้มีความสามารถมากกว่านี้ เนื่องจากตัวหุ่นยนต์ที่สร้างมานี้เป็นเพียงต้นแบบเท่านั้น จึงไม่มีความสามารถพิเศษใดๆ

เอกสารอ้างอิง

[1] cbenson. (2561). Robot Leg Torque Tutorial, สืบค้นเมื่อ 16 ธันวาคม 2563 จาก https://www.robotshop.com/community/tutorials/show/robot-leg-torque-tutorial

[2] Al Mostafa Mohamed. (2560). Hexapod presentation-robot, สืบค้นเมื่อ 18 ธันวาคม 2563 จาก https://www.slideshare.net/elmostafamohamed9/hexa-pod-presentationrobot

[3] pando85. (2558). Antdroid, hexapod open source robot, สืบค้นเมื่อ 19 ธันวาคม 2563 จาก https://www.slideshare.net/elmostafamohamed9/hexa-pod-presentationrobot

[4] Andrew Beaupre and co. (2558). Design and Production of a 3-D Printed Wireless Hexapod, สืบค้นเมื่อ 5 มีนาคม 2564 จาก https://web.wpi.edu/Pubs/E-project/Available/E-project-042615-150727/unrestricted/Hexapod_MQP_Final_MQP_Report_4-26-2015.pdf

[5] Topics For Seminar (2561). Hexapod Robot Project Report with Coding and PPT, สืบค้นเมื่อ 10 เมษายน 2564 จาก https://www.topicsforseminar.com/2018/06/hexapod-robot-project-report-with.html?m=1

ภาคผนวก

ภาคผนวก ก

โค้ดโปรแกรม

1. โปรแกรมหลักของ Hexapod

```
Copyright (C) 2014 Alexander Gil and Javier Román

#include "Antfirm.h"

byte level_log = 0;

#ifdef ControlRos
    ros::NodeHandle arduino;
#endif

Hexapod Antdroid;

Control DefaultControl(&Antdroid);

void setup() {

    DefaultControl.Start();
      Antdroid.Start();

}

void loop() {
      DefaultControl.ReadInput();
```

2. การวัดก่อนเริ่มใช้งานโปรแกรมหลัก

```
Copyright (C) 2014 Alexander Gil and Javier Román

/*
 * Allows us to have a default calibration before we start to run the
 * Antfirm code for the 18 servos que have attached to the Arduino
 * MEGA board. This calibration is saved in the internal EEPROM memory
 * of the board. For debug and testing proves, we have added a function
 * to know the actual values of the memory

 * The functions are:

 * currentCalibration(): send by serial port all the calibration angles.
 * writeCalibrationCompleted(): write byte in eeprom to define calibration
 * completed.
 * tryCalibrationCompleted(): return true if calibration completed
 */
#include "calibration.h"

#include <avr/eeprom.h>

void currentCalibration(void)
{
    log("Current calibration of the servos: ", Info);

    for(byte i=0; i<6; i++)
    {
        log("Leg:", i, Info);
```

```
        log("-------------------------------------------------", Info);
        for(byte j=0; j<3; j++)
        {
            if(j==0)
                log("Coxa:", Info);
            else if(j==1)
                log("Femur:", Info);
            else
                log("Tibia:", Info);

            log(" ", eeprom_read_byte((unsigned char *)(j+3*i)), Info);
            log("EEPROM dir:", j+3*i, Debug);
        }
    log(" ", Info);
    }
}

void writeCalibrationCompleted(void)
{
    eeprom_write_byte((unsigned char *) (CalibrationCompletedDir),
        CalibrationCompletedValue);
    log("Calibration completed", Info);
    log("You must restart your Arduino now", Info);
}

bool tryCalibrationCompleted(void)
{
    if(eeprom_read_byte((unsigned char *) (CalibrationCompletedDir)) ==
        CalibrationCompletedValue)
        return true;

    log("Calibration not completed yet",Error);
    log("To complete calibration send calibrate with angle = 255",Info);

    return false;
}
```

3. ROS control สำหรับ Hexapod

```
Copyright (C) 2014 Alexander Gil and Javier Román

/*
 * If this library is included in the Antfirm program, it allows us
 * to control a hexapod by using the serial port of the Arduino MEGA
 * board like a ROS node. Warn: It is not compatible with serial control
 * system.
 *
 *The methos are:
 *
 * Control: Class for manipulating a hexapod by using the serial port
 * Start(): create node and subscribers
 * ReadInput(): wait to read orders publicated in topics.
 *
 * RunCommand(): main menu for the rest of the methods.
 *
 * ControlWalk(antdroid_msgs::Walk& ): moves the hexapod to the selected
```

```
 * (X, Y) coordinate.
 *
 * ControlBalance(antdroid_msgs::Balance& ): balances the hexapod
 *
 * ControlRotate(antdroid_msgs::Rotate& ): spins the hexapod the selected
 * number of degrees.
 *
 * ControlChangeSpeed(antdroid_msgs::Speed& ): changes the movement speed of
 * the hexapod [0 - 250]
 *
 * ControlChangeHeight(antdroid_msgs::Height& ): changes the current height
 * of the hexapod.
 *
 * ControlChangeFootDistance(antdroid_msgs::Foot& ): changes foot distance of
 * hexapod's legs.
 *
 * ControlChangeLogLevel(const antdroid_msgs::Log& msg): changes log level
 *
 * ControlChangeCalibration(const antdroid_msgs::Calibrate& msg): changes
 * calibration values
 *
 */

#include "Configuration.h"

#ifdef ControlRos
#include "controlRos.h"

ros::Subscriber<antdroid_msgs::Walk> walk("/antfirm/walk", &ControlWalk);
ros::Subscriber<antdroid_msgs::Balance> balance("/antfirm/balance",
&ControlBalance);
ros::Subscriber<antdroid_msgs::Rotate> rotate("/antfirm/rotate",
&ControlRotate);
ros::Subscriber<antdroid_msgs::Speed> speed ("/antfirm/speed",
&ControlChangeSpeed);
ros::Subscriber<antdroid_msgs::Height> height("/antfirm/height",
&ControlChangeHeight);
ros::Subscriber<antdroid_msgs::Foot> footDistance("/antfirm/foot",
&ControlChangeFootDistance);
ros::Subscriber<antdroid_msgs::Log> logLevel("/antfirm/log",
&ControlChangeLogLevel);
ros::Subscriber<antdroid_msgs::Calibrate> calibration("/antfirm/calibrate",
&ControlChangeCalibration);
ros::Subscriber<antdroid_msgs::Gait> gait("/antfirm/gait",
&ControlChangeGait);
ros::Subscriber<antdroid_msgs::MoveLeg> moveLeg("/antfirm/move_leg",
&ControlMoveLeg);
ros::Subscriber<std_msgs::Bool> attack("/antfirm/attack", &ControlAttack);
ros::Subscriber<std_msgs::Bool> sayHello("/antfirm/say_hello",
&ControlSayHello);

std_msgs::Bool is_new_message;
ros::Publisher pub_is_new_message("/antfirm/new_message", &is_new_message);

Control::Control(Hexapod* Antdroid)
{
    is_new_message.data = true;
```

```
}

void Control::Start(void)
{
    arduino.initNode();

    arduino.subscribe(walk);
    arduino.subscribe(balance);
    arduino.subscribe(rotate);
    arduino.subscribe(speed);
    arduino.subscribe(height);
    arduino.subscribe(footDistance);
    arduino.subscribe(logLevel);
    arduino.subscribe(calibration);
    arduino.subscribe(gait);
    arduino.subscribe(moveLeg);
    arduino.subscribe(attack);
    arduino.subscribe(sayHello);

    arduino.advertise(pub_is_new_message);

    level_log = 0;

}

void Control::ReadInput(void)
{
    arduino.spinOnce();
}

void ControlWalk(const antdroid_msgs::Walk& msg)
{
    Antdroid.Walk(msg.x, msg.y);

    is_new_message.data = true;
    pub_is_new_message.publish(&is_new_message);
    pub_is_new_message.publish(&is_new_message);
}


void ControlBalance(const antdroid_msgs::Balance& msg)
{
    Antdroid.Balance(msg.pitch, msg.roll, msg.yaw);
}

void ControlRotate(const antdroid_msgs::Rotate& msg)
{
    Antdroid.Rotate(msg.yaw);

    is_new_message.data = true;
    pub_is_new_message.publish(&is_new_message);
    pub_is_new_message.publish(&is_new_message);
}

void ControlChangeSpeed(const antdroid_msgs::Speed& msg)
{
    Antdroid.ChangeSpeed(msg.speed);
```

```
}

void ControlChangeHeight(const antdroid_msgs::Height& msg)
{
    Antdroid.ChangeHeight(msg.height);
}

void ControlChangeFootDistance(const antdroid_msgs::Foot& msg)
{
    Antdroid.ChangeFootDistance(msg.footDistance);
}

void ControlChangeLogLevel(const antdroid_msgs::Log& msg)
{
    level_log = msg.level;
}

void ControlChangeCalibration(const antdroid_msgs::Calibrate& msg)
{
    Antdroid.CalibrateLeg(msg.leg, msg.member, msg.angle);
}

void ControlChangeGait(const antdroid_msgs::Gait& msg)
{
    const uint8_t sequence[6]= { msg.leg0, msg.leg1, msg.leg2, msg.leg3,
msg.leg4,
        msg.leg5};

    Antdroid.ChangeGait(msg.type, sequence);

    is_new_message.data = false;
    pub_is_new_message.publish(&is_new_message);
}


void ControlMoveLeg(const antdroid_msgs::MoveLeg& msg)
{
    Antdroid.MoveLeg(msg.leg, msg.x, msg.y, msg.z);
}

void ControlAttack(const std_msgs::Bool& msg)
{
     Antdroid.Attack();
}

void ControlSayHello(const std_msgs::Bool& msg)
{
     Antdroid.SayHello();
}

#endif
```

4. การควบคุมแบบอนุกรมของหุ่นยนต์แมงมุม

```
/*
  * If this library is included in the Antfirm program, it allows us
  * to control a hexapod by using the serial port of the Arduino MEGA
  * board. Warn: It is not compatible with ROS control system.
  *
  *The methos are:
  *
  * Control: Class for manipulating a hexapod by using the serial port
  * Start(): begins the serial communication (baud rate:115200)
  * ReadInput(): reads the commands introduced by the user by serial port
  *
  * RunCommand(): main menu for the rest of the methods.
  * TryRunMove(byte ): moves the hexapod to the selected (X, Y) coordinate.
  * TryRunTurn(byte ): spins the hexapod the selected number of degrees.
  * TryRunGait(byte ): selects the gait that the hexapod is going to use.
  * TryRunBalance(byte ): balances the hexapod
  * TryRunSpeed(byte ): changes the movement speed of the hexapod [0 - 250]
  * TryRunHeight(byte ): changes the current height of the hexapod.
  * TryRunFootDistance(byte ): changes foot distance of hexapod's legs
  * TryChangeLog(byte ): changes the amount of info given by the log
  * TryChangeCalibration(byte ): runs calibration mode.
  */

#include "Configuration.h"

#ifdef ControlSerial
#include "controlSerial.h"

Control::Control(Hexapod* Antdroid)
{
    log("In Control::Control", Debug);

    _antdroid = Antdroid;
    _command = "";
}

void Control::Start(void)
{
    level_log = 3;

    Serial.begin(115200);
    log("Serial.begin = 115200", Info);
}

void Control::ReadInput(void)
{
    log("In Control::ReadInput", Debug);

    while(Serial.available() >0 )
    {
        _current_char = Serial.read();
        if(_current_char == '\n' ||
            _current_char == '\r'||
            _current_char == ';'
            )
        {
            RunCommand();
```

```
            _command = "";
        }
        else
        {
            _command += _current_char;
        }
    }

}

void Control::RunCommand(void)
{
    log("In Control::RunCommand", Debug);

    byte endCommandName = 0;

    while(endCommandName < _command.length() &&
    _command.charAt(endCommandName) != '(' &&
    _command.charAt(endCommandName) != ' ')
    {
        endCommandName++;

    }

    if(_command.substring(0, endCommandName) == "move")
    {
        if(!TryRunMove(endCommandName))
            RunDefault();
    }
    else if (_command.substring(0, endCommandName) == "turn")
    {
        if(!TryRunTurn(endCommandName))
            RunDefault();
    }
    else if (_command.substring(0, endCommandName) == "gait")
    {
        if(!TryRunGait(endCommandName))
            RunDefault();
    }
    else if (_command.substring(0, endCommandName) == "balance")
    {
        if(!TryRunBalance(endCommandName))
            RunDefault();
    }
    else if (_command.substring(0, endCommandName) == "speed")
    {
        if(!TryRunSpeed(endCommandName))
            RunDefault();
    }
    else if (_command.substring(0, endCommandName) == "height")
    {
        if(!TryRunHeight(endCommandName))
            RunDefault();
    }
    else if (_command.substring(0, endCommandName) == "foot_distance")
    {
        if(!TryRunFootDistance(endCommandName))
```

```
            RunDefault();
    }
     else if (_command.substring(0, endCommandName) == "calibrate")
    {
        if(!TryChangeCalibration(endCommandName))
            RunDefault();
    }
    else if (_command.substring(0, endCommandName) == "log")
    {
        if(!TryChangeLog(endCommandName))
            RunDefault();
    }
    else if (_command == "help")
        RunHelp();
    else
        RunDefault();

}


bool Control::TryRunMove(byte endCommandName)
{
    log("In Control::TryRunMove", Debug);

    String number;

    byte initString;
    byte endString;
    short x;
    short y;

    endString = endCommandName;


    if(!_command.startsWith("(", endString))
        return false;

    initString = ++endString;

    while(endString < _command.length() &&
    _command.charAt(endString) != ',' )
    {
        endString++;
    }

    if(!_command.startsWith(",", endString))
        return false;

    number = _command.substring(initString, endString);

    if( number == "0")
        x = 0;
    else if(number.toInt() != 0)
        x = number.toInt();
    else
        return false;
```

```
    initString = ++endString;


    while(endString < _command.length() &&
    _command.charAt(endString) != ')' )
    {
        endString++;
    }

    if(!_command.startsWith(")", endString))
        return false;

   number = _command.substring(initString, endString);

    if( number == "0")
        y = 0;
    else if(number.toInt() != 0)
        y = number.toInt();
    else
        return false;

    _antdroid->Walk(x,y);

    return true;
}



bool Control::TryRunTurn(byte endCommandName)
{
    log("In Control::TryRunTurn", Debug);

    String number;

    byte initString;
    byte endString;
    short degree;

    endString = endCommandName;


    if(!_command.startsWith("(", endString))
        return false;

    initString = ++endString;

    while(endString < _command.length() &&
    _command.charAt(endString) != ')' )
    {
        endString++;
    }

    if(!_command.startsWith(")", endString))
        return false;

    number = _command.substring(initString, endString);
```

```
    if( number == "0")
        degree = 0;
    else if(number.toInt() != 0)
        degree = number.toInt();
    else
        return false;

    _antdroid->Rotate(degree);
    return true;
}

bool Control::TryRunGait(byte endCommandName)
{
    log("In Control::TryRunGait", Debug);

    byte initString;
    byte endString;

    endString = endCommandName;

    while(endString < _command.length() &&
    _command.charAt(endString) != '-' )
    {
        endString++;
    }

    if(!_command.startsWith("-", endString))
        return false;

    initString = ++endString;

    if(_command.startsWith("-", endString))
    {
        initString = ++endString;

        while(endString < _command.length() &&
        _command.charAt(endString) != ' ' )
        {
            endString++;
        }

        if(_command.substring(initString, endString) == "tripod")
        {
            _antdroid->EnableTripodGait();
            return true;
        }
        if(_command.substring(initString, endString) == "ripple")
        {
            _antdroid->EnableRippleGait();
            return true;
        }
        else
            return false;
    }
    else if (_command.charAt(endString) == 't' && endString ==
_command.length()-1)
```

```
    {
        _antdroid->EnableTripodGait();
        return true;
    }

    else if (_command.charAt(endString) == 'r' && endString ==
_command.length()-1)
    {
        _antdroid->EnableRippleGait();
        return true;
    }


    return false;

}
bool Control::TryRunBalance(byte endCommandName)
{
    log("In Control::TryRunBalance", Debug);

    String number;

    byte initString;
    byte endString;
    short pitch;
    short roll;
    short yaw;

    initString = endCommandName;


    if(!_command.startsWith("(", initString))
        return false;

    initString++;

    endString = initString;

    while(endString < _command.length() &&
    _command.charAt(endString) != ',' )
    {
        endString++;
    }

    if(!_command.startsWith(",", endString))
        return false;

    number = _command.substring(initString, endString);

    if( number == "0")
        pitch = 0;
    else if(number.toInt() != 0)
        pitch = number.toInt();
    else
        return false;

    initString = ++endString;
```

```
    while(endString < _command.length() &&
    _command.charAt(endString) != ',' )
    {
        endString++;
    }

    if(!_command.startsWith(",", endString))
        return false;

    number = _command.substring(initString, endString);

    if( number == "0")
        roll = 0;
    else if(number.toInt() != 0)
        roll = number.toInt();
    else
        return false;

    initString = ++endString;

    while(endString < _command.length() &&
    _command.charAt(endString) != ')' )
    {
        endString++;
    }

    if(!_command.startsWith(")", endString))
        return false;

   number = _command.substring(initString, endString);

    if( number == "0")
        yaw = 0;
    else if(number.toInt() != 0)
        yaw = number.toInt();
    else
        return false;

    _antdroid->Balance(pitch, roll, yaw);

    return true;
}

bool Control::TryRunSpeed(byte endCommandName)
{
    log("In Control::TryRunSpeed", Debug);

    String number;

    byte initString;
    byte endString;

    endString = endCommandName;
    while(endString < _command.length() &&
    _command.charAt(endString) != '-' )
```

```
    {
        endString++;
    }

    if(!_command.startsWith("-", endString))
        return false;

    initString = ++endString;

    if (_command.charAt(endString) == 'r')
    {
        _antdroid->RiseSpeed();
        return true;
    }

    else if(_command.charAt(endString) == 'd')
    {
        _antdroid->DecreaseSpeed();
        return true;
    }

    else if(_command.charAt(endString) == 'c')
    {
        initString = ++endString;

        while(endString < _command.length() &&
        _command.charAt(endString) != ' ' )
        {
            endString++;
        }

        number = _command.substring(initString, endString);

        if(number.toInt() != 0)
        {
            _antdroid->ChangeSpeedStep(number.toInt());
            return true;
        }
    }
    return false;
}

bool Control::TryRunHeight(byte endCommandName)
{
    log("In Control::TryRunHeight", Debug);

    String number;

    byte initString;
    byte endString;

    endString = endCommandName;

    while(endString < _command.length() &&
    _command.charAt(endString) != '-' )
    {
        endString++;
```

```
    }

    if(!_command.startsWith("-", endString))
        return false;

    initString = ++endString;

    if (_command.charAt(endString) == 'r')
    {
        _antdroid->RiseHeight();
        return true;
    }

    else if(_command.charAt(endString) == 'd')
    {
        _antdroid->DecreaseHeight();
        return true;
    }

    else if(_command.charAt(endString) == 'c')
    {
        initString = ++endString;

        while(endString < _command.length() &&
        _command.charAt(endString) != ' ' )
        {
            endString++;
        }

        number = _command.substring(initString, endString);

        if(number.toInt() != 0)
        {
            _antdroid->ChangeHeightStep(number.toInt());
            return true;
        }
    }
    return false;
}

bool Control::TryRunFootDistance(byte endCommandName)
{
    log("In Control::TryRunDistance", Debug);

    String number;

    byte initString;
    byte endString;

    endString = endCommandName;

    while(endString < _command.length() &&
    _command.charAt(endString) != '-' )
    {
        endString++;
    }
```

```
    if(!_command.startsWith("-", endString))
        return false;

    initString = ++endString;

    if (_command.charAt(endString) == 'r')
    {
        _antdroid->RiseFootDistance();
        return true;
    }

    else if(_command.charAt(endString) == 'd')
    {
        _antdroid->DecreaseFootDistance();
        return true;
    }

    else if(_command.charAt(endString) == 'c')
    {
        initString = ++endString;

        while(endString < _command.length() &&
        _command.charAt(endString) != ' ' )
        {
            endString++;
        }

        number = _command.substring(initString, endString);

        if(number.toInt() != 0)
        {
            _antdroid->ChangeFootDistanceStep(number.toInt());
            return true;
        }
    }
    return false;
}

bool Control::TryChangeLog(byte endCommandName)
{
    log("In Control::TryChangeLog", Debug);

    String number;

    byte initString;
    byte endString;

    endString = endCommandName;

    while(endString < _command.length() &&
    _command.charAt(endString) != '-' )
    {
        endString++;
    }
    if(!_command.startsWith("-", endString))
        return false;
```

```
    initString = ++endString;

    if (_command.charAt(endString) == '1')
    {
        level_log = 1;
        Serial.println("Error messages activated");
        return true;
    }
    else if(_command.charAt(endString) == '2')
    {
        level_log = 2;
        Serial.println("Errors and warns messages activated");
        return true;
    }
    else if(_command.charAt(endString) == '3')
    {
        level_log = 3;
        Serial.println("Errors, warns and info messages activated");
        return true;
    }
    else if(_command.charAt(endString) == '4')
    {
        level_log = 4;
        Serial.println("Errors, warns, info and debug messages activated");
        return true;
    }
    return false;
}

bool Control::TryChangeCalibration(byte endCommandName)
{
    log("In Control::TryChangeCalibration", Debug);

    String number;

    byte initString;
    byte endString;
    short leg;
    short member;
    short angle;

    initString = endCommandName;


    if(!_command.startsWith("(", initString))
        return false;

    initString++;

    endString = initString;

    while(endString < _command.length() &&
    _command.charAt(endString) != ',' )
    {
        endString++;
    }
```

```
    if(!_command.startsWith(",", endString))
        return false;

    number = _command.substring(initString, endString);

    if( number == "0")
        leg = 0;
    else if(number.toInt() != 0)
        leg = number.toInt();
    else
        return false;

    if(leg > 5 || leg < 0)
        return false;

    initString = ++endString;

    while(endString < _command.length() &&
    _command.charAt(endString) != ',' )
    {
        endString++;
    }

    if(!_command.startsWith(",", endString))
        return false;

    number = _command.substring(initString, endString);

    if( number == "0")
        member = 0;
    else if(number.toInt() != 0)
        member = number.toInt();
    else
        return false;

    initString = ++endString;

    while(endString < _command.length() &&
    _command.charAt(endString) != ')' )
    {
        endString++;
    }

    if(!_command.startsWith(")", endString))
        return false;

   number = _command.substring(initString, endString);

    if( number == "0")
        angle = 0;
    else if(number.toInt() != 0)
        angle = number.toInt();
    else
        return false;

    _antdroid->CalibrateLeg(leg, member, angle);
```

```
    return true;
}

void Control::RunHelp(void)
{
    Serial.println("Command list:");
    RunHelpMove();
    RunHelpTurn();
    RunHelpGait();
    RunHelpBalance();
    RunHelpSpeed();
    RunHelpHeight();
    RunHelpFootDistance();
    RunHelpCalibration();
    RunHelpLog();
}

void Control::RunHelpMove(void)
{
    Serial.println("move(X,Y)");
    Serial.println("Move hexapod with (X,Y) step with selected gait.");
    Serial.println("X: distance forward in milimeters.");
    Serial.println("Y: distance left side in milimeters.");
    Serial.println(DefaultGaitMsg);
    Serial.println("Example of use: move(30,-20);");
    Serial.println();
}

void Control::RunHelpTurn(void)
{
    Serial.println("turn(Angle)");
    Serial.println("Turn hexapod with selected gait.(positive value:
clockwork side)");
    Serial.println("Angle: degrees with one decimal.");
    Serial.println(DefaultGaitMsg);
    Serial.println("Example of use: turn(-150);");
    Serial.println();
}

void Control::RunHelpGait(void)
{
    Serial.println("gait [OPTIONS]");
    Serial.println("Select gait.");
    Serial.println("-t, --tripod");
    Serial.println("    select tripod gait.");
    Serial.println("-r, --ripple");
    Serial.println("    select ripple gait.");
    Serial.println(DefaultGaitMsg);
    Serial.println("Example of use: gait -t;");
    Serial.println();
}

void Control::RunHelpBalance(void)
{
    Serial.println("balance(pitch, roll, yaw)");
    Serial.println("Balance hexapod mode without move.");
```

```
    Serial.println("Pitch: pitch angle in degrees with one decimal.");
    Serial.println("Roll: pitch angle in degrees with one decimal.");
    Serial.println("Yaw: pitch angle in degrees with one decimal.");
    Serial.println("Example of use: balance(50,-60,150); ");
    Serial.println();
}

void Control::RunHelpSpeed(void)
{
    Serial.println("speed [OPTIONS]");
    Serial.println("Change speed of hexapod movements.");
    Serial.println("-r");
    Serial.println("    rise speed");
    Serial.println("-d");
    Serial.println("    decrease speed");
    Serial.println("-c");
    Serial.println("    change step of rise/decrease (speed is [0,255]
range)");
    Serial.println("Example of use: speed -c30; ");
    Serial.println();
}

void Control::RunHelpHeight(void)
{
    Serial.println("height [OPTIONS]");
    Serial.println("Change height of hexapod.");
    Serial.println("-r");
    Serial.println("    rise height");
    Serial.println("-d");
    Serial.println("    decrease height");
    Serial.println("-c");
    Serial.println("    change step of rise/decrease (height is in
milimeters)");
    Serial.println("Example of use: height -c30; ");
    Serial.println();
}

void Control::RunHelpFootDistance(void)
{
    Serial.println("foot_distance [OPTIONS]");
    Serial.println("Change foot distance of hexapod legs.");
    Serial.println("-r");
    Serial.println("    rise foot distance");
    Serial.println("-d");
    Serial.println("    decrease foot distance");
    Serial.println("-c");
    Serial.println("    change step of rise/decrease (foot distance is in
milimeters)");
    Serial.println("Example of use: foot distance -c30; ");
    Serial.println();
}

void Control::RunHelpLog(void)
{
    Serial.println("log [OPTIONS]");
    Serial.println("Change information given by the log.");
    Serial.println("-1 \t");
```

```
    Serial.println("    error messages");
    Serial.println("-2\t");
    Serial.println("    error and warn messages");
    Serial.println("-3\t");
    Serial.println("    error, warn and info messages");
    Serial.println("Example of use: log -2; ");
    Serial.println("-4\t");
    Serial.println("    error, warn, info and debug messages");
    Serial.println("Example of use: log -2; ");
    Serial.println();
}

void Control::RunHelpCalibration(void)
{
    Serial.println("calibrate(Leg,Member,Angle)");
    Serial.println("Calibration of the indicated leg.");
    Serial.println("Leg: number of the leg [0 - 5].");
    Serial.println("Member: Coxa = 0, Femur = 1, Tibia = 2 [0 - 2].");
    Serial.println("Angle: angle to default position [0 - 120].");
    Serial.println("NOTE: to complete calibration send angle 255.");
    Serial.println("Example of use: calibrate(3,0,90);");
    Serial.println();
}

void Control::RunDefault(void){
    Serial.print(_command);
    Serial.println(": not correct.Try help.");
}

#endif
```

5. ส่วนหลักของการควบคุมหุ่นยนต์แมงมุม

```
Copyright (C) 2014 Alexander Gil and Javier Román
/*
  * Main library to control the hexapod. It supports serial control by
  * using the serial communication port or you can attach the hexapod
  * object as a node of the ROS system and controll it using that OS.
  *
  */


#include "hexapod.h"

const byte Pin[] PROGMEM = {
      LeftFrontCoxaPin, LeftFrontFemurPin, LeftFrontTibiaPin,
      RightFrontCoxaPin, RightFrontFemurPin, RightFrontTibiaPin,
      LeftMiddleCoxaPin, LeftMiddleFemurPin, LeftMiddleTibiaPin,
      RightMiddleCoxaPin, RightMiddleFemurPin, RightMiddleTibiaPin,
      LeftRearCoxaPin, LeftRearFemurPin, LeftRearTibiaPin,
      RightRearCoxaPin, RightRearFemurPin, RightRearTibiaPin
};


const short CoxaOffset[] PROGMEM ={
      CoxaOffsetX, CoxaOffsetY, CoxaOffsetX, -CoxaOffsetY,
      0, CoxaOffsetY, 0, -CoxaOffsetY,
```

```
      -CoxaOffsetX, CoxaOffsetY, -CoxaOffsetX, -CoxaOffsetY
};


Hexapod::Hexapod(void)
:     _floorHeight(FootHeight),
      _rotate(false),
      _speed(DefaultSpeed),
      _footDistance(FootDistance),

      _footDistanceStep(FootDistanceStep),
      _speedStep(SpeedStep),
      _floorHeightStep(FloorHeightStep),
      _is_blocked(0),
      _mode(1)
{
      log("In Hexapod::Hexapod", Debug);

      _legs = new Leg*[6];

      _legs[0] = new Leg(LeftFrontCoxaDefaultAngle, LeftFrontCoxaMin,
            LeftFrontCoxaMax, LeftFemurMin, LeftFemurMax, LeftTibiaMin,
            LeftTibiaMax, 0);
      _legs[1] = new Leg(RightFrontCoxaDefaultAngle, RightFrontCoxaMin,
            RightFrontCoxaMax,     RightFemurMin, RightFemurMax,
RightTibiaMin,
            RightibiaMax, 1);
      _legs[2] = new Leg(LeftMiddleCoxaDefaultAngle, LeftMiddleCoxaMin,
            LeftMiddleCoxaMax, LeftFemurMin, LeftFemurMax, LeftTibiaMin,
             LeftTibiaMax, 2);
      _legs[3] = new Leg(RightMiddleCoxaDefaultAngle, RightMiddleCoxaMin,
            RightMiddleCoxaMax, RightFemurMin, RightFemurMax, RightTibiaMin,
            RightibiaMax, 3);
      _legs[4] = new Leg(LeftRearCoxaDefaultAngle, LeftRearCoxaMin,
            LeftRearCoxaMax, LeftFemurMin, LeftFemurMax, LeftTibiaMin,
            LeftTibiaMax, 4);
      _legs[5] = new Leg(RightRearCoxaDefaultAngle, RightRearCoxaMin,
            RightRearCoxaMax, RightFemurMin, RightFemurMax, RightTibiaMin,
            RightibiaMax, 5);
}

void Hexapod::Start(void)
{
      log("In Hexapod::Start", Debug);

      GoStartingPostion();

      for(byte i = 0;i < 6; i++)
      {
            _legs[i]->Attach(pgm_read_byte(&Pin[3*i]),
pgm_read_byte(&Pin[3*i+1]),
                  pgm_read_byte(&Pin[3*i+2]));
      }

      CalculateMaxFloorHeight();

      if(!tryCalibrationCompleted())
```

```
    {
      for (byte i = 0; i < 6; i++)
      {
            _legs[i]->SaveDefaultCalibration();
      }
    }

      EnableDefaultGait();
}

void Hexapod::GoStartingPostion(void)
{
      for(byte i = 0;i < 6; i++)
      {
            if(!_legs[i]->TryCalDefaultPosition(_footDistance ,
_floorHeight))
                  {
                        if(tryCalibrationCompleted())
                              return;
                  }
      }

      for(byte i = 0;i < 6; i++)
      {
            if(!_legs[i]->WriteStartPosition())
                  return;
      }
}

void Hexapod::EnableDefaultGait(void)
{
      EnableTripodGait();
}

void Hexapod::EnableTripodGait( void)
{
      const uint8_t sequence[6] = {0, 1, 1, 0, 0, 1};

      for (byte i = 0; i < 6; i++)
      {
            _sequence[i] = sequence[i];
      }

      log("Tripod Gait Enabled", Info);

      GoDefaultPosition();
}

void Hexapod::EnableRippleGait(void)
{
      const uint8_t sequence[6] = {0, 2, 3, 1, 2, 0};

      for (byte i = 0; i < 6; i++)
      {
            _sequence[i] = sequence[i];
      }
```

```
	log("Ripple Gait Enabled", Info);

	GoDefaultPosition();
}


void Hexapod::GoDefaultPosition()
{
	uint16_t timeMove = 0;
	_is_blocked = 0;

	for(byte i = 0;i < 6; i++)
	{
		if(!_legs[i]->TryCalDefaultPosition(_footDistance ,
_floorHeight))
			{
				if(tryCalibrationCompleted())
					return;
			}
	}

	for(byte i = 0;i < 6; i++)
	{
		if (timeMove < _legs[i]->CalculateTimeMove(_speed))
			timeMove = _legs[i]->CalculateTimeMove(_speed);
	}

	for(byte i = 0;i < 6; i++)
	{
		if(!_legs[i]->TryUpdatePosition(timeMove))
			return;
	}
}

void Hexapod::CalculateMaxFloorHeight()
{
	log("In Hexapod::CalculateMaxFloorHeight", Debug);

	_MaxFloorHeight = - (TibiaLength +
		((long)GetSin(LeftFemurMax) * FemurLength) / Shift4Decimal);

	log("_MaxFloorHeight", _MaxFloorHeight, Debug);
}



void Hexapod::Balance(const short pitch, const short roll, const short yaw)
{
	log("In Hexapod::Balance", Debug);

	uint16_t timeMove = 0;
	uint8_t balanceSpeed = _speed * BalanceSpeedProportion;

	const short sinRoll = GetSin(roll);
	const short cosRoll = GetCos(roll);

	const short sinPitch = GetSin(pitch);
```

```
      const short cosPitch = GetCos(pitch);

      const short sinYaw = GetSin(yaw);
      const short cosYaw = GetCos(yaw);

      short position[6][3] = {0};

      short aux[3];

      short rotationMatrix[3][3] = {
            {((long)cosPitch * cosYaw)/Shift4Decimal,
            (-((long)cosPitch * sinYaw)/Shift4Decimal),
            sinPitch} ,
            {((long)cosRoll*sinYaw)/ Shift4Decimal +
                  ((((long)sinRoll*sinPitch)/ Shift4Decimal)*cosYaw)/
Shift4Decimal,
            (((long)cosRoll*cosYaw)/ Shift4Decimal) -
                  ((((long)sinRoll*sinPitch)/ Shift4Decimal)* sinYaw)/
Shift4Decimal,
             -((long)sinRoll*cosPitch)/ Shift4Decimal} ,
            {((((long)sinRoll*sinYaw)/ Shift4Decimal) -
                  ((((long)cosRoll*sinPitch)/ Shift4Decimal)* cosYaw)/
Shift4Decimal,
            (((long)sinRoll*cosYaw)/ Shift4Decimal) +
                  ((((long)cosRoll*sinPitch)/ Shift4Decimal)* sinYaw)/
Shift4Decimal,
            ((long)cosRoll*cosPitch)/ Shift4Decimal}
      };

      if(_is_blocked)
            GoDefaultPosition();

      // Rotate in absolute coordinates
      for(byte f = 0; f < 6; f++)
      {
            for(byte i = 0; i < 3; i++)
            {
                  aux[0] = ((long)rotationMatrix[i][0] *
                  ((short)pgm_read_word(&CoxaOffset[2*f]) + _legs[f]-
>currentX))
                  / Shift4Decimal;
                  aux[1] = ((long)rotationMatrix[i][1] *
                  ((short)pgm_read_word(&CoxaOffset[2*f+1]) + _legs[f]-
>currentY))
                  / Shift4Decimal;
                  aux[2] = ((long)rotationMatrix[i][2] * _floorHeight)
                  / Shift4Decimal;
                  position[f][i] = aux[0] + aux[1] + aux[2];
            }
      }

      // Relative coordinates
      for(byte f = 0; f < 6; f++)
      {
            position[f][0] = position[f][0]-
(short)pgm_read_word(&CoxaOffset[2*f]);
```

```
            position[f][1] = position[f][1]-
(short)pgm_read_word(&CoxaOffset[2*f+1]);
      }

      for(byte f = 0; f < 6; f++)
      {
            if(!_legs[f]->TryCalRotationPosition(position[f]))
                  return;
      }

      for(byte f = 0; f < 6; f++)
      {
            if(_legs[f]->CalculateTimeMove(balanceSpeed) > timeMove)
                  timeMove = _legs[f]->CalculateTimeMove(balanceSpeed);
      }

      for(byte f = 0; f < 6; f++)
      {
            if(!_legs[f]->TryUpdatePosition(timeMove))
                  return;
      }
}

void Hexapod::Rotate(const short yawAngle)
{
      log("In Hexapod::Rotate", Debug);

      const short sinYaw = GetSin(yawAngle);
      const short cosYaw = GetCos(yawAngle);

      short aux[2];

      short rotationMatrix[2][2]={
            {cosYaw, -sinYaw},
            {sinYaw, cosYaw}
      };

      for(byte f = 0; f < 6;f++)
      {
            for (byte i = 0; i < 2; i++)
            {
                  aux[0] = ((long)rotationMatrix[i][0] * _legs[f]->currentX)
                  / Shift4Decimal;
                  aux[1] = ((long)rotationMatrix[i][1] * _legs[f]->currentY)
                  / Shift4Decimal;
                  if(i == 0)
                        _position[f][i] = aux[0] + aux[1] - _legs[f]-
>currentX;
                  else
                        _position[f][i] = aux[0] + aux[1] - _legs[f]-
>currentY;
            }
            _position[f][2] = _floorHeight;
      }
      for (byte f = 0; f < 2; f++)
      {
```

```
                for (byte i = 0; i < 2; i++)
                {
                        if (f == i)
                                rotationMatrix[f][i]=GetCos(-yawAngle);
                        else if (i == 0)
                                rotationMatrix[f][i]=GetSin(-yawAngle);
                        else
                                rotationMatrix[f][i]=-GetSin(-yawAngle);
                }
        }

        for (byte f = 0; f < 6; f++)
        {
                for(byte i = 0; i < 2; i++)
                {
                        aux[0] = ((long)rotationMatrix[i][0] * _legs[f]->currentX)
                        / Shift4Decimal;
                        aux[1] = ((long)rotationMatrix[i][1] * _legs[f]->currentY)
                        / Shift4Decimal;
                        if(i == 0)
                                _negativePosition[f][i] = aux[0] + aux[1] - _legs[f]-
>currentX;
                        else
                                _negativePosition[f][i] = aux[0] + aux[1] - _legs[f]-
>currentY;
                }
                _negativePosition[f][2] = _floorHeight;
        }

        _rotate = true;

        Walk(0,0);
}

void Hexapod::Walk(const short x, const short y)
{
        log("In Hexapod::Walk", Debug);

        uint16_t timeMoveTransfer = 0;
        uint16_t timeMove = 0;

        const short gap = Gap;

        InitPosition( x, y);

        if(IsCollising())
                return;

        if(_is_blocked)
                GoDefaultPosition();

        const byte steps = ReturnSteps();

        const uint8_t speed_body = _speed / (steps -1);
        for (byte current_step = 0; current_step < steps; current_step++)
        {
```

```
            byte previous = (current_step == 0) ? steps -1 : current_step -1;

            for(byte i = 0; i < 6; i++)
            {
                  if(_sequence[i] == current_step)
                  {
                        if(!_legs[i]->TryCalTransferTrajectory(_position[i],
gap))
                              return;
                  }
                  if(_sequence[i] == previous)
                        if(!_legs[i]-
>TryCalRelativePosition(_negativePosition[i]))
                              return;
            }
            for(byte i=0;i<6;i++)
            {
                  if((_sequence[i] == current_step) || (_sequence[i]==
previous) )
                        _legs[i]->WaitUntilStop();
            }

            for(byte i = 0; i < 6; i++)
            {
                  if(_sequence[i] == current_step)
                  {
                        if(timeMoveTransfer < _legs[i]-
>CalculateTimeMove(_speed))
                              timeMoveTransfer = _legs[i]-
>CalculateTimeMove(_speed);
                  }
                        if(_sequence[i] == previous && current_step == 0)
                              if( timeMove < _legs[i]-
>CalculateTimeMove(speed_body))
                              timeMove = _legs[i]-
>CalculateTimeMove(speed_body);
            }

            timeMove = (timeMove > timeMoveTransfer * 2 * (steps -1)) ?
                  timeMove : timeMoveTransfer * 2 * (steps -1) ;

            for(byte i = 0; i < 6; i++)
            {
                  if(_sequence[i] == current_step)
                  {
                        if(!_legs[i]->TryUpdatePosition(timeMoveTransfer))
                              return;
                  }

                  if(_sequence[i] == previous)
                        if(!_legs[i]->TryUpdatePosition(timeMove))
                              return;
            }

            for(byte i=0;i<6;i++)
            {
                  if(_sequence[i] == current_step)
```

```
                        if(!_legs[i]->TryCalRelativePosition(_position[i]))
                              return;
                }
                for(byte i=0;i<6;i++)
                {
                        if(_sequence[i] == current_step)
                              _legs[i]->WaitUntilStop();

                }
                for(byte i = 0; i < 6; i++)
                {
                        if(_sequence[i] == current_step)
                              if(timeMoveTransfer < _legs[i]-
>CalculateTimeMove(_speed))
                                    timeMoveTransfer = _legs[i]-
>CalculateTimeMove(_speed);
                }

                for(byte i = 0; i < 6; i++)
                {
                        if(_sequence[i] == current_step)
                              if(!_legs[i]->TryUpdatePosition(timeMoveTransfer))
                                    return;
                }
        }
}

void Hexapod::InitPosition(const short x, const short y)
{
        log("In Hexapod::InitPosition", Debug);

        if(!_rotate)
        {
                for(byte f = 0; f < 6; f++)
                {
                        _position[f][0] = x;
                        _position[f][1] = y;
                        _position[f][2] = _floorHeight;

                        for (byte i=0;i<2;i++)
                        {
                              _negativePosition[f][i] = - _position[f][i];
                        }
                        _negativePosition[f][2] = _position[f][2];
                }
        }

        _rotate = false;
}

byte Hexapod::ReturnSteps(void)
{
        log("In Hexapod::ReturnSteps", Debug);

        byte steps = 0;

        for(byte i = 0; i < 6; i++)
```

```
        {
                if(_sequence[i] > steps)
                        steps = _sequence[i];
        }
        return ++steps;
}

bool Hexapod::IsCollising(void)
{
        log("In Hexapod::IsCollising", Debug);

        const short max_distance = (_position[0][0] > 0) ? _position[0][0]
                : - _position[0][0];

        if(max_distance >= (CoxaOffsetX +
                FootDistance * GetCos(LeftFrontCoxaDefaultAngle))/2)
        {
                log("Hexapod::IsCollising == true", Error);
                return true;
        }

        return false;
}

void Hexapod::RiseSpeed(void)
{
        log("In Hexapod::RiseSpeed", Debug);

        if ( ((short)_speed + _speedStep) < 255)
                _speed += _speedStep;

        log("_speed", _speed, Debug);
}

void Hexapod::DecreaseSpeed(void)
{
        log("In Hexapod::DecreaseSpeed", Debug);

        if ( ((short)_speed - _speedStep) > 0)
                _speed -= _speedStep;

        log("_speed", _speed, Debug);
}

void Hexapod::ChangeSpeedStep(uint8_t speedStep)
{
        log("In Hexapod::ChangeSpeedStep", Debug);
        _speedStep = speedStep;
}

void Hexapod::ChangeSpeed(uint8_t speed)
{
        log("In Hexapod::ChangeSpeed", Debug);

        if(speed == 1)
                RiseSpeed();
        else if(speed == 0)
```

```
            DecreaseSpeed();
      else if((speed >1) && (speed <= 255))
            _speed = speed;

      log("\% Speed", ((float)_speed*100)/255, Info);
}

void Hexapod::RiseHeight(void)
{

      if(!ControlMode)
      {
            RiseMode();
            return;
      }

      log("In Hexapod::RiseHeight", Debug);
      if ( _floorHeight + _floorHeightStep > _MaxFloorHeight)
            _floorHeight -= _floorHeightStep;

}

void Hexapod::DecreaseHeight(void)
{
      if(!ControlMode)
      {
            DecreaseMode();
            return;
      }

      log("In Hexapod::DecreaseHeight", Debug);
      if ( _floorHeight - _floorHeightStep < 0)
            _floorHeight += _floorHeightStep;

}

void Hexapod::RiseMode(void)
{
      if(_mode < 2)
      {
            _mode += 1;
            ChangeMode(_mode);
      }
}

void Hexapod::DecreaseMode(void)
{
      if(_mode > 0)
      {
            _mode -= 1;
            ChangeMode(_mode);
      }
}

void Hexapod::ChangeMode(uint8_t mode)
{
      if(mode == 0)
```

```
        {
                _floorHeight = BasicLowHeigh;
                _footDistance = BasicLowFootDistance;
                log("Change to mode Low.", Info);
        }

        else if(mode == 1)
        {
                _floorHeight = BasicMiddleHeigh;
                _footDistance = BasicMiddleFootDistance;
                log("Change to mode Middle.", Info);
        }

        else if(mode == 2)
        {
                _floorHeight = BasicHighHeigh;
                _footDistance = BasicHighFootDistance;
                log("Change to mode High.", Info);
        }

        else
        {
                log("Change mode fail.", Error);
        }
}

void Hexapod::ChangeHeightStep(uint8_t heightStep)
{
        log("In Hexapod::ChangeHeightStep", Debug);
        _floorHeightStep = heightStep;
}

void Hexapod::ChangeHeight(short height)
{
        log("In Hexapod::ChangeHeight", Debug);

        if(height == 1)
                RiseHeight();

        else if(height == 0)
                DecreaseHeight();

        else if((height < -1) && (height < _MaxFloorHeight ))
                _floorHeight = -height;

        log("Height distance:", -_floorHeight , Info);
}

void Hexapod::RiseFootDistance(void)
{
        if(!ControlMode)
                return;

        log("In Hexapod::RiseFootDistance", Debug);
        if (_footDistance < CoxaLength + FemurLength + TibiaLength - 30)
                _footDistance += _footDistanceStep;
}
```

```
void Hexapod::DecreaseFootDistance(void)
{
	if(!ControlMode)
		return;

	log("In Hexapod::DecreaseFootDistance", Debug);
	if (_footDistance > CoxaLength - 20)
		_footDistance -= _footDistanceStep;
}

void Hexapod::ChangeFootDistanceStep(uint8_t footDistanceStep)
{
	log("In Hexapod::ChangeFootDistanceStep", Debug);
	_footDistanceStep = footDistanceStep;
}

void Hexapod::ChangeFootDistance(short footDistance)
{
	log("In Hexapod::ChangeFootDistance", Debug);

	if(footDistance == 1)
		RiseFootDistance();
	else if(footDistance == 0)
		DecreaseFootDistance();
	else if((footDistance >1) && (footDistance < CoxaLength + FemurLength
		+ TibiaLength - 30) )
		_footDistance = footDistance;

	log("FootDistance:",_footDistance , Info);
}

void Hexapod::CalibrateLeg(const byte legNumber, const uint8_t member,
	const uint8_t angle)
{
	log("In Hexapod::CalibrateLeg", Debug);

	log("Ang [255]: validates FIRST calibration.", Info);
	log("Ang [254]: shows current calibration.", Info);
	log("Ang [253]: servos will move to the calibration value", Info);
	log("Ang [252]: servos won't move to the calibration value", Info);

	static uint8_t moveLegs = 0;

	if(legNumber > 5 || legNumber < 0)
	{
		log("LegNumber range [0-5]", Error);
		return;
	}

	if(angle == 252)
	{
		moveLegs = 0;
		log("Move and calibrate OFF", Warn);
		return;
	}
```

```
        if(angle == 253)
        {
              moveLegs = 1;
              LegsToCalibrationAngles();
              log("Move and calibrate ON", Warn);
              return;
        }

        if(angle == 255)
        {
              writeCalibrationCompleted();
              return;
        }

        if(angle == 254)
        {
              currentCalibration();
              return;
        }
        if(!moveLegs)
        {
              _legs[legNumber]->TrySaveValueCalibration(member, angle);
        }
        else
        {
              _legs[legNumber]->SaveCalibrationMovePosition(member, angle);
        }

  }


  void Hexapod::ChangeGait(uint8_t type, const uint8_t sequence[6])
  {
        if(type ==1)
        {
              EnableTripodGait();
              return;
        }

        if(type == 2)
        {
              EnableRippleGait();
              return;
        }

        if(type == 0)
        {
              EnableCustomGait(sequence);
              return;
        }

        log("Gait type error",Error);

  }

  void Hexapod::EnableCustomGait(const uint8_t sequence[6])
  {
```

```
	for (byte i = 0; i < 6; i++)
	{
		_sequence[i] = sequence[i];
	}

	GoDefaultPosition();
}

uint16_t Hexapod::MoveLeg(const byte legNumber, const uint16_t x,
	const uint16_t y, const uint16_t z)
{
	short foot_position[3];
	foot_position[0]= x;
	foot_position[1]= y;
	foot_position[2]= z;

	if(legNumber >5)
	{
		log("Leg number must be into 0 and 5",Error);
		return 0;
	}

	if(!_legs[legNumber]->TryCalculatePosition(foot_position))
		return 0;

	uint16_t timeMove = _legs[legNumber]->CalculateTimeMove(_speed);

	if(!_legs[legNumber]->TryUpdatePosition(timeMove))
		return 0;

	return timeMove;
}

void Hexapod::Attack()
{
	short foot_position[6][3];
	_is_blocked = 1;

	foot_position[0][0] = (short ATTACK_LEFT_FRONT_X);
	foot_position[0][1] = (short ATTACK_LEFT_FRONT_Y);
	foot_position[0][2] = (short ATTACK_LEFT_FRONT_Z);

	foot_position[2][0] = (short ATTACK_LEFT_MIDDLE_X);
	foot_position[2][1] = (short ATTACK_LEFT_MIDDLE_Y);
	foot_position[2][2] = (short ATTACK_LEFT_MIDDLE_Z);

	foot_position[4][0] = (short ATTACK_LEFT_REAR_X);
	foot_position[4][1] = (short ATTACK_LEFT_REAR_Y);
	foot_position[4][2] = (short ATTACK_LEFT_REAR_Z);

	for(byte i = 0;i < 3; i++)
	{
		for(byte f = 0;f < 3; f++)
		{
			if( f == 1)
				foot_position[i * 2 + 1][f] = -foot_position[i *
2][f];
```

```
                else
                        foot_position[i * 2 + 1][f] = foot_position[i *
2][f];
            }
        }

        for(byte i = 0;i < 6; i++)
        {
            if(!_legs[i]->TryCalculatePosition(foot_position[i]))
                return;
        }

        uint16_t timeMove = 0;

        for(byte i = 0;i < 6; i++)
        {
            uint16_t legTimeMove = _legs[i]->CalculateTimeMove(_speed);
            if(timeMove < legTimeMove)
                timeMove = legTimeMove;
        }

        for(byte i = 0;i < 6; i++)
        {
            if(!_legs[i]->TryUpdatePosition(timeMove))
                return;
        }
}

void Hexapod::SayHello()
{
        _is_blocked = 1;

        delay(MoveLeg(0, SAY_HELLO_X_A, SAY_HELLO_Y_A, SAY_HELLO_Z) + 200);
        delay(MoveLeg(0, SAY_HELLO_X_B, SAY_HELLO_Y_B, SAY_HELLO_Z) + 200);

}

void Hexapod::LegsToCalibrationAngles()
{
        for(byte i = 0;i < 6; i++)
        {
            _legs[i]->ServosToCalibrationAngles();
        }
}

Hexapod::~Hexapod(){
        delete _legs;
}
```

6. ส่วนขาของหุ่นยนต์แมงมุม

```
Copyright (C) 2014 Alexander Gil and Javier Román
/*
 * Library to create and control a leg object. The leg objects can be used
 * alone or throw the hexapod library, which is in charge of create and
 * control the six of them.
 *
 * The methods are:
```

```
 *
 * Leg: constructor. Needs as initial attributes the min and max angles
 *      for each servo and the number of the new leg.
 *
 * Attach(byte , byte , byte ): attaches the three servos of the leg.
 * TryCalDefaultPosition(short , short ): moves the leg to def. position
 * TryCalculatePosition(short ): calculates the new servo angles using IK
 * TryCalRelativePosition(short ):
 * TryCalTransferTrajectory(short , short ):
 * TryCalRotationPosition(short ):
 * CalculateTimeMove(uint8_t ): calculates the needed time for the next move
 *                     taking into account the current and the next position
 *                     and the maximum allowed speed.
 * TryUpdatePosition(uint16_t ): moves the servos to its new angles.
 * TryMoveServos(short,short,short,uint16_t ): moves the servos to its
 *                                             new location.
 * SaveCalibrationMovePosition(uint8_t, uint8_t ): allows us to calibrate the
three servos
 *                     of the leg and moves them to their locations.
 */

/*
 * Axis system:
 *     X    forwards
 *     Y    left
 *     Z    up
 *
 * Default angle values:
 *    Degree    One decimal. Example: 1800 = 180.0 °
 *    Radians   Four decimal. Example: 31416 = 3,1416 rad
 */

#include "leg.h"

#include <avr/eeprom.h>
#include <ServoEx.h>


Leg::Leg(short CoxaDefaultAngle, short coxa_min, short coxa_max,
    short femur_min, short femur_max, short tibia_min, short tibia_max,
    uint8_t legNumber)
    : _coxaDefaultAngle(CoxaDefaultAngle),
    _coxaMin(coxa_min),
    _coxaMax(coxa_max),
    _femurMin (femur_min),
    _femurMax(femur_max),
    _tibiaMin (tibia_min),
    _tibiaMax(tibia_max),
    _legNumber(legNumber),
    _coxaCalibrationAngle(eeprom_read_byte((unsigned char
*)(legNumber*3+Coxa))),
    _femurCalibrationAngle(eeprom_read_byte((unsigned char
*)(legNumber*3+Femur))),
    _tibiaCalibrationAngle(eeprom_read_byte((unsigned char
*)(legNumber*3+Tibia)))
{
    log("In Leg::Leg", Debug);
```

```
    log("New leg object", _legNumber, Info);
}

void Leg::Attach(byte pinServoCoxa, byte pinServoFemur, byte pinServoTibia)
{
    log("In Leg::Attach", Debug);

    _servoCoxa.attach(pinServoCoxa);
    _servoFemur.attach(pinServoFemur);
    _servoTibia.attach(pinServoTibia);

    log("New LEG object attached", Info);
}

bool Leg::TryCalDefaultPosition(const short footDistance, const short
footHeight)
{
    log("In Leg::TryCalDefaultPosition", Debug);

    short default_position[3];

    default_position[0] = ((long)footDistance * GetCos(_coxaDefaultAngle))
        /Shift4Decimal;
    default_position[1] = ((long)footDistance * GetSin(_coxaDefaultAngle))
        /Shift4Decimal;
    default_position[2] = footHeight;

    if(!TryCalculatePosition(default_position))
        return false;

    return true;
}

bool Leg::WriteStartPosition(void)
{
    if(_alarm)
    {
        log("_alarm == true", Error);
        return false;
    }

    const short coxaAngle = _coxaAngle/Shift1Decimal + _coxaCalibrationAngle;
    const short femurAngle = _femurAngle/Shift1Decimal +
_femurCalibrationAngle;
    const short tibiaAngle = _tibiaAngle/Shift1Decimal +
_tibiaCalibrationAngle;

    if((coxaAngle <= 0) || (femurAngle <= 0) || (tibiaAngle <= 0)
        || (coxaAngle >= 180) || (tibiaAngle >= 180) || (femurAngle >= 180))
    {
        log("Angles are not in range", Error);
        log("Leg:", _legNumber, Debug);
        log("Coxa", coxaAngle, Debug);
        log("Femur", femurAngle, Debug);
        log("Tibia", tibiaAngle, Debug);
```

```
            return false;
        }

        currentX = _foot_position[0];
        currentY = _foot_position[1];


        _servoCoxa.write(coxaAngle);
        _servoFemur.write(femurAngle);
        _servoTibia.write(tibiaAngle);

        _coxaCurrentAngle = (coxaAngle - _coxaCalibrationAngle)*Shift1Decimal;
        _femurCurrentAngle = (femurAngle - _femurCalibrationAngle)*Shift1Decimal;
        _tibiaCurrentAngle = (tibiaAngle - _tibiaCalibrationAngle)*Shift1Decimal;

        return true;

    }

    bool Leg::TryCalculatePosition(short foot_position[3])
    {
        log("In Leg::TryCalculatePosition", Debug);

        for(byte i=0;i<3;i++)
            _foot_position[i] = foot_position[i];


        if(!TryCalculateInverseKinematic())
            return false;
        _updateXY = true;

        return true;
    }

    bool Leg::TryCalRelativePosition(short foot_position[3])
    {
        log("In Leg::TryCalRelativePosition", Debug);

        _foot_position[0] = foot_position[0] + currentX;
        _foot_position[1] = foot_position[1] + currentY;
        _foot_position[2] = foot_position[2];

        if(!TryCalculateInverseKinematic())
            return false;
        _updateXY = false;

        return true;
    }

    bool Leg::TryCalTransferTrajectory(short foot_position[3], const short gap)
    {
        log("In Leg::TryCalTransferTrajectory", Debug);

        _foot_position[0] = currentX;
        _foot_position[1] = currentY;
        _foot_position[2] = foot_position[2] + gap ;
```

```
    if(!TryCalculateInverseKinematic())
    {
        log("Trying to calculate new position", Warn);

        if (_coxaDefaultAngle > 0)
            _foot_position[1] = foot_position[1]/2 + currentY + 20;
        else
            _foot_position[1] = foot_position[1]/2 + currentY - 20;

        if(!TryCalculateInverseKinematic())
            return false;
    }
    _updateXY = false;

    return true;
}

bool Leg::TryCalRotationPosition(short foot_position[3])
{
    log("In Leg::TryCalRotationPosition", Debug);

    for(byte i=0;i<3;i++)
        _foot_position[i] = foot_position[i];

    if(!TryCalculateInverseKinematic())
        return false;
    _updateXY = false;

    return true;
}

bool Leg::TryCalculateInverseKinematic(void)
{
    log("In Leg::TryCalculateInverseKinematic", Debug);

      long XYhyp;
      long aux[3];
      short hip_foot;

    log("Leg:", _legNumber, Debug);
    log("_foot_position[0]", _foot_position[0], Debug);
    log("_foot_position[1]", _foot_position[1], Debug);
    log("_foot_position[2]", _foot_position[2], Debug);

    if(!tryCalibrationCompleted())
    {
        return false;
    }

    XYhyp = Hypotenuse (_foot_position[0], _foot_position[1]);
      _coxaAngle = Rad2Deg(GetArcTan(_foot_position[1], _foot_position[0],
XYhyp))
                 - _coxaDefaultAngle;
    _coxaAngle = - _coxaAngle;
```

```
    log("_CoxaAngle: ", _coxaAngle, Debug);

    if ((_coxaAngle < _coxaMin) || (_coxaAngle > _coxaMax) )
    {
        log("Coxa angle not in range", Error);
        _alarm = true;
        return false;
    }

      XYhyp = XYhyp - CoxaLength;

      hip_foot = Hypotenuse(_foot_position[2], XYhyp);
    log("hip_foot: ", hip_foot, Debug);

    if (hip_foot > (FemurLength + TibiaLength) )
    {
        log("Leg is too short", Error);
        _alarm = true;
        return false;
    }

      aux[2] = (GetArcTan( XYhyp, -_foot_position[2], hip_foot));

      //Law of cosines to get Femur-hip_foot Angle
      aux[0] = (((long)FemurLength * FemurLength) - ((long)TibiaLength *
            TibiaLength)) + ((long)hip_foot * hip_foot);
      aux[1] = ((long) 2 * FemurLength * hip_foot);
      aux[0] = GetArcCos((aux[0] * Shift4Decimal) /aux[1]);//Femur-hip_foot
Angle
      _femurAngle = - Rad2Deg((aux[2] - Pi/2) + aux[0]);

    if(_coxaDefaultAngle < 0)
        _femurAngle = - _femurAngle;

    log("_femurAngle", _femurAngle, Debug);

    if ((_femurAngle < _femurMin)|| (_femurAngle > _femurMax) )
    {
        log("Femur angle not in range", Error);
        _alarm = true;
        return false;
    }

      //Law of cosines to get Femur-Tibia Angle
      aux[0] = (((long)FemurLength * FemurLength) - ((long)hip_foot *
hip_foot))
      + ((long)TibiaLength * TibiaLength);
      aux[1] = 2 * FemurLength * TibiaLength;

      //Femur-Tibia Angle
      aux[0] = GetArcCos((aux[0] * Shift4Decimal) / aux[1]);
      _tibiaAngle = Rad2Deg(aux[0] - Pi/2);
    if(_coxaDefaultAngle > 0)
        _tibiaAngle = - _tibiaAngle;
    log("_tibiaAngle", _tibiaAngle, Debug);
```

```
    if ((_tibiaAngle < _tibiaMin)|| (_tibiaAngle > _tibiaMax) )
    {
        log("Tibia angle not in range", Error);
        _alarm = true;
        return false;
    }

    _alarm = false;
      return true;
}

bool Leg::TryUpdatePosition(const uint16_t timeMove)
{
    log("In Leg::TryUpdatePosition", Debug);

    if(_alarm)
    {
        log("_alarm == true", Error);
        return false;
    }

    if(!TryMoveServos(_coxaAngle/Shift1Decimal + _coxaCalibrationAngle,
        _femurAngle/Shift1Decimal + _femurCalibrationAngle,
        _tibiaAngle/Shift1Decimal + _tibiaCalibrationAngle,timeMove))
        return false;


    return true;
}

uint16_t Leg::CalculateTimeMove(const uint8_t speed)
{
    log("In Leg::CalculateTimeMove", Debug);

    uint16_t maxDif = 0;
    uint16_t timeMove;
    uint16_t realSpeed;
    short difAngle[3];

    difAngle[0] = _coxaCurrentAngle - _coxaAngle;
    difAngle[1] = _femurCurrentAngle - _femurAngle;
    difAngle[2] = _tibiaCurrentAngle - _tibiaAngle;

    for(byte i=0; i<3; i++)
    {
        if (difAngle[i] < 0)
            difAngle[i] = -difAngle[i];

        if(difAngle[i]>maxDif)
            maxDif=difAngle[i];
    }

    maxDif = maxDif / Shift1Decimal;
    realSpeed = ((long)speed * MaxRealSpeed)/MaxSpeed;
    timeMove = ((uint32_t)(maxDif*Shift4Decimal))/realSpeed; // miliseconds

    log("realSpeed", realSpeed, Debug);
```

```
    log("timeMove", timeMove, Debug);

    return timeMove;
}

bool Leg::TryMoveServos(const short coxaAngle, const short femurAngle,
    const short tibiaAngle, const unsigned int mTime)
{

    log("In Leg::TryMoveServos", Debug);

    if((coxaAngle <= 0) || (femurAngle <= 0) || (tibiaAngle <= 0)
        || (coxaAngle >= 180) || (tibiaAngle >= 180) || (femurAngle >= 180))
    {
        log("Angles are not in range", Error);
        log("Leg:", _legNumber, Debug);
        log("Coxa", coxaAngle, Debug);
        log("Femur", femurAngle, Debug);
        log("Tibia", tibiaAngle, Debug);

        return false;
    }

    if(_updateXY)
    {
        currentX = _foot_position[0];
        currentY = _foot_position[1];
    }

    _servoCoxa.move(coxaAngle, mTime);
    _servoFemur.move(femurAngle, mTime);
    _servoTibia.move(tibiaAngle, mTime);

    _coxaCurrentAngle = (coxaAngle - _coxaCalibrationAngle)*Shift1Decimal;
    _femurCurrentAngle = (femurAngle - _femurCalibrationAngle)*Shift1Decimal;
    _tibiaCurrentAngle = (tibiaAngle - _tibiaCalibrationAngle)*Shift1Decimal;

    log("Leg:", _legNumber, Debug);
    log("Time move:", mTime, Debug);
    log("_coxaCurrentAngle", _coxaCurrentAngle, Debug);
    log("_femurCurrentAngle", _femurCurrentAngle, Debug);
    log("_tibiaCurrentAngle", _tibiaCurrentAngle, Debug);

    return true;
}

void Leg::WaitUntilStop(void){
    while(IsMoving());
}

bool Leg::IsMoving(){

      if (_servoCoxa.moving() || _servoFemur.moving() ||
_servoTibia.moving())
          return true;
      return false;
```

```
}

void Leg::SaveCalibrationMovePosition(uint8_t member, uint8_t angle)
{
    if(!TrySaveValueCalibration(member, angle))
        return;

    log("Moving servo to default position", Info);

    if(member == Coxa)
        _servoCoxa.move(_coxaCalibrationAngle, DefaultTime);

    else if (member == Femur)
        _servoFemur.move(_femurCalibrationAngle, DefaultTime);

    else
        _servoTibia.move(_tibiaCalibrationAngle, DefaultTime);
}

bool Leg::TrySaveValueCalibration(uint8_t member, uint8_t angle)
{
    log("In Leg::TrySaveValueCalibration", Debug);

    if(angle > 180 || angle < 0 || member > 2 || member < 0)
    {
        log("Calibrate values not in range", Error);
        return false;
    }

    if(eeprom_read_byte((unsigned char *)(_legNumber*3+member)) != angle)
    {
        log("Writing to EEPROM memory. Do not turn off the power source",
Warn);
        eeprom_write_byte((unsigned char *)(_legNumber*3+member), angle);
        log("Save completed.", Warn);
        _coxaCalibrationAngle = eeprom_read_byte((unsigned char
*)(_legNumber*3+Coxa));
        _femurCalibrationAngle = eeprom_read_byte((unsigned char
*)(_legNumber*3+Femur));
        _tibiaCalibrationAngle = eeprom_read_byte((unsigned char
*)(_legNumber*3+Tibia));
    }

    return true;
}


void Leg::SaveDefaultCalibration(void)
{
    log("Saving default calibration", Info);
    uint8_t defaultAngle;
    defaultAngle = CalculateCalibrationPosition(_coxaMin, _coxaMax);
    SaveCalibrationMovePosition(Coxa, defaultAngle);
    defaultAngle = CalculateCalibrationPosition(_femurMin, _femurMax);
    SaveCalibrationMovePosition(Femur, defaultAngle);
    defaultAngle = CalculateCalibrationPosition(_tibiaMin, _tibiaMax);
    SaveCalibrationMovePosition(Tibia, defaultAngle);
```

```
}

uint8_t Leg::CalculateCalibrationPosition(short minAngle, short maxAngle)
{
    uint8_t defaultAngle = 0;

    defaultAngle = (900 -(maxAngle - minAngle) /2 - minAngle)/10;

    log("Default angle = ", defaultAngle, Info);
    return defaultAngle;
}

void Leg::ServosToCalibrationAngles(void)
{
    _servoCoxa.write(_coxaCalibrationAngle);
    _servoFemur.write(_femurCalibrationAngle);
    _servoTibia.write(_tibiaCalibrationAngle);
}
```

7. ฟังก์ชัน Log

```
Copyright (C) 2014 Alexander Gil and Javier Román
/*
* A generic set of log functions divided hierarchically by its
* relevance: error, warn, info anf bebug messages. The higher the
* value of the 'level_log' global parameter, the higher number of
* functions we turn on. 'level_log=1' means only error messages turned
* on while 'level_log=3' means error, warn anf info messages turned on
*
* The functions:
*
* log(String, int): sends by serial port a message. The int parameter sets
*                   the level of the message we send.
* log(String, int, int): log overloaded. Allows us to send the value of
*                   one parameter to the printDebug function.
*
* printMsg(String, int): sends by serial port a message if level_log = [1-4]
* printMsg(String, int, int): printMsg overloaded. Allows us to send a
*                   message and a parameter if level_log >=4
*/

#include "log.h"

void log(String msg, int code)
{
    if(level_log >= code)
        printMsg(msg, code);
}

void log(String msg, int attribute, int code)
{
    if(level_log >= code)
        printMsg(msg, attribute, code);
}

#ifdef ControlSerial
    void printMsg(String msg, int code)
    {
```

```
        switch(code)
        {
            case 1:
                Serial.print("Error: ");
                break;
            case 2:
                Serial.print("Warn: ");
                break;
            case 3:
                Serial.print("Info: ");
                break;
            case 4:
                Serial.print("Debug: ");
                break;
            default:
                Serial.print("log.cpp. printMsg.");
                break;
        }
        Serial.println(msg);
    }

    void printMsg(String msg, int attribute,int code)
    {
        switch(code)
        {
            case 1:
                Serial.print("Error: ");
                break;
            case 2:
                Serial.print("Warn: ");
                break;
            case 3:
                Serial.print("Info: ");
                break;
            case 4:
                Serial.print("Debug: ");
                break;
            default:
                Serial.print("log.cpp. printMsg.");
                break;
        }
        Serial.print(msg);
        Serial.print(" = ");
        Serial.println(attribute);
    }

#else

    #ifdef ControlRos

        void printMsg(String msg, int code)
        {
            char* charMsg = StringToChar(msg);
            switch(code)
            {
                case 1:
                    arduino.logerror(charMsg);
```

```
                break;
            case 2:
                arduino.logwarn(charMsg);
                break;
            case 3:
                arduino.loginfo(charMsg);
                break;
            case 4:
                arduino.logdebug(charMsg);
                break;
            default:
                arduino.logerror("log.cpp. printMsg.");
                break;
        }
        delete charMsg;
    }

    void printMsg(String msg, int attribute, int code)
    {
        char* charMsg = StringToChar(msg);
        String stringAttribute = String(attribute);
        char* charAtrribute = StringToChar(stringAttribute);

        switch(code)
        {
            case 1:
                arduino.logerror(charMsg);
                arduino.logerror(charAtrribute);
                break;
            case 2:
                arduino.logwarn(charMsg);
                arduino.logwarn(charAtrribute);
                break;
            case 3:
                arduino.loginfo(charMsg);
                arduino.loginfo(charAtrribute);
                break;
            case 4:
                arduino.logdebug(charMsg);
                arduino.logdebug(charAtrribute);
                break;
            default:
                arduino.logerror("log.cpp. printMsg.");
                break;
        }

        delete charMsg;
        delete charAtrribute;
    }

#else
    void printMsg(String msg, int code){}
    void printMsg(String msg, int attribute, int code){}
#endif
#endif

inline char* StringToChar(String msg)
```

```
{
    char* charBuff = new char[msg.length()+1];
    msg.toCharArray(charBuff, msg.length()+1);
    return charBuff;
}
```

8. ฟังก์ชันตรีโกณมิติปรับค่าสำหรับ Inverse Kinematics ของหุ่นยนต์แมงมุม

```
Copyright (C) 2014 Alexander Gil and Javier Román
#include "trig.h"

//**************************************************************************
**
// Tables
// ArcCosinus Table
// Table build in to 3 part to get higher accuracy near cos = 1.
// The biggest error is near cos = 1 and has a biggest value of 3*0.012098rad
// = 0.521 deg.
//  -    Cos 0 to 0.9 is done by steps of 0.0079 rad. [1/127]
//  -    Cos 0.9 to 0.99 is done by steps of 0.0008 rad [0.1/127]
//  -    Cos 0.99 to 1 is done by step of 0.0002 rad [0.01/64]
// Since the tables are overlapping the full range of 127+127+64 is not
// necessary. Total bytes: 277

static const byte GetACos[] PROGMEM = {

255,254,252,251,250,249,247,246,245,243,242,241,240,238,237,236,234,233,232,

231,229,228,227,225,224,223,221,220,219,217,216,215,214,212,211,210,208,207,

206,204,203,201,200,199,197,196,195,193,192,190,189,188,186,185,183,182,181,

179,178,176,175,173,172,170,169,167,166,164,163,161,160,158,157,155,154,152,

150,149,147,146,144,142,141,139,137,135,134,132,130,128,127,125,123,121,119,
  117,115,113,111,109,107,105,103,101,98,96,94,92,89,87,84,81,79,76,73,73,73,
  72,72,72,71,71,71,70,70,70,70,69,69,69,68,68,68,67,67,67,66,66,66,65,65,65,
  64,64,64,63,63,63,62,62,62,61,61,61,60,60,59,59,59,58,58,58,57,57,57,56,56,
  55,55,55,54,54,53,53,53,52,52,51,51,51,50,50,49,49,48,48,47,47,47,46,46,45,
  45,44,44,43,43,42,42,41,41,40,40,39,39,38,37,37,36,36,35,34,34,33,33,32,31,
  31,30,29,28,28,27,26,25,24,23,23,23,23,22,22,22,22,21,21,21,21,20,20,20,19,
  19,19,19,18,18,18,17,17,17,17,16,16,16,15,15,15,14,14,13,13,13,12,12,11,11,
  10,10,9,9,8,7,6,6,5,3,0 };//

// Sin table 90 deg, persision 0.5 deg [180 values]
static const word Get_Sin[] PROGMEM = {
  0, 87, 174, 261, 348, 436, 523, 610, 697, 784, 871, 958, 1045, 1132, 1218,
  1305, 1391, 1478, 1564, 1650, 1736, 1822, 1908, 1993, 2079, 2164, 2249,
  2334, 2419, 2503, 2588, 2672, 2756, 2840, 2923, 3007, 3090, 3173, 3255,
  3338, 3420, 3502, 3583, 3665, 3746, 3826, 3907, 3987, 4067, 4146, 4226,
  4305, 4383, 4461, 4539, 4617, 4694, 4771, 4848, 4924, 4999, 5075, 5150,
  5224, 5299, 5372, 5446, 5519, 5591, 5664, 5735, 5807, 5877, 5948, 6018,
  6087, 6156, 6225, 6293, 6360, 6427, 6494, 6560, 6626, 6691, 6755, 6819,
  6883, 6946, 7009, 7071, 7132, 7193, 7253, 7313, 7372, 7431, 7489, 7547,
  7604, 7660, 7716, 7771, 7826, 7880, 7933, 7986, 8038, 8090, 8141, 8191,
  8241, 8290, 8338, 8386, 8433, 8480, 8526, 8571, 8616, 8660, 8703, 8746,
  8788, 8829, 8870, 8910, 8949, 8987, 9025, 9063, 9099, 9135, 9170, 9205,
  9238, 9271, 9304, 9335, 9366, 9396, 9426, 9455, 9483, 9510, 9537, 9563,
```

```
  9588, 9612, 9636, 9659, 9681, 9702, 9723, 9743, 9762, 9781, 9799, 9816,
  9832, 9848, 9862, 9876, 9890, 9902, 9914, 9925, 9935, 9945, 9953, 9961,
  9969,9975, 9981, 9986, 9990, 9993, 9996, 9998, 9999, 10000 };

//*************************************************************************
**
// GetCos: Get cosinus from the angle +/- multiple circles
// Input: Angle in degree with one decimal.
// Return: Cosine of angle input with four decimals.
//*************************************************************************
**

short GetCos(short AngleDeg1)
{
  short cos4;
  short ABSAngleDeg1;    //Absolute value of the Angle in Degrees, decimals =
1
  //Get the absolute value of AngleDeg
  if (AngleDeg1 < 0)
  {
      ABSAngleDeg1 = AngleDeg1 * -1;
  }
  else
  {
      ABSAngleDeg1 = AngleDeg1;
  }
  //Shift rotation to a full circle of 360 deg -> AngleDeg // 360
  if (AngleDeg1 < 0)     //Negative values
      AngleDeg1 = 3600-(ABSAngleDeg1-(3600*(ABSAngleDeg1/3600)));
  else                   //Positive values
      AngleDeg1 = ABSAngleDeg1-(3600*(ABSAngleDeg1/3600));

  if (AngleDeg1>=0 && AngleDeg1<=900)     // 0 to 90 deg
  {
      cos4 = pgm_read_word(&Get_Sin[(900-(AngleDeg1))/5]);
      return cos4;
  }

  else if (AngleDeg1>900 && AngleDeg1<=1800)     // 90 to 180 deg
  {
      cos4 = -pgm_read_word(&Get_Sin[(AngleDeg1-900)/5]);
      return cos4;
  }
  else if (AngleDeg1>1800 && AngleDeg1<=2700) // 180 to 270 deg
  {
      cos4 = -pgm_read_word(&Get_Sin[(2700-AngleDeg1)/5]);
      return cos4;
  }

  else if(AngleDeg1>2700 && AngleDeg1<=3600) // 270 to 360 deg
  {
      cos4 = pgm_read_word(&Get_Sin[(AngleDeg1-2700)/5]);
      return cos4;
  }
}
```

```
//*************************************************************************
**
// GetSin: Get sinus from the angle +/- multiple circles
// Input: Angle in degree with one decimal.
// Return: Sinus of angle input with four decimals.
//*************************************************************************
**

short GetSin(short AngleDeg1)
{
  short sen4;
  sen4 = GetCos(900-AngleDeg1);
  return sen4;
}

//*************************************************************************
**
// GetArcCos: Get arc cosine from the cosine with 4 decimals
// Input: Cosine with four decimals.
// Return: Angle in radians with four decimals.
//*************************************************************************
**

short GetArcCos(short cos4)
{
      short AngleRad4;
  bool NegativeValue;

      if (cos4 < 0)
      {
            cos4 = -cos4;
            NegativeValue = true;
      }
      else
            NegativeValue = false;

  //Limit cos4 to his maximal value
      cos4 = min(cos4,Shift4Decimal);

      if ((cos4>=0) && (cos4<9000))
      {
            AngleRad4 = (byte)pgm_read_byte(&GetACos[cos4/79]);
    AngleRad4 = ((long)AngleRad4 * 616/Shift1Decimal);
//616=((Pi/2)*Shift1Decimal)/255)
    }
      else if ((cos4>=9000) && (cos4<9900))
      {
        AngleRad4 = (byte)pgm_read_byte(&GetACos[(cos4-9000)/8+114]);
    AngleRad4 = (long)((long)AngleRad4 * 616/Shift1Decimal);
      }
      else if ((cos4>=9900) && (cos4<=10000))
      {
        AngleRad4 = (byte)pgm_read_byte(&GetACos[(cos4-9900)/2+227]);
    AngleRad4 = (long)((long)AngleRad4 * 616/Shift1Decimal);
  }
  //Add negative sign
```

```
      if (NegativeValue)
            AngleRad4 = 31416 - AngleRad4;

return AngleRad4;
}

//*************************************************************************
**
// GetArcTan: Get simplyfied arc tangent function based on fixed point arc
cos
// GetArcTan(X,Y,hip)= ArcTan(X/Y)
// Input: X and Y.
// Return: Angle in radians with four decimals.
//*************************************************************************
**

short GetArcTan (short x, short y, long XYhyp)
{
      short Atan4;
      short AngleRad4;

      AngleRad4 = GetArcCos (((long)y * Shift4Decimal) / XYhyp);

  if (x < 0)     // removed overhead... Atan4 = AngleRad4 * (y / abs(y));
    Atan4 = -AngleRad4;
  else
      Atan4 = AngleRad4;
  return Atan4;
}

//*************************************************************************
**
// Hypotenuse: Calculate hypotenuse
// Input: X and Y.
// Return: Hypotenuse.
//*************************************************************************
**
short Hypotenuse (short x, short y)
{
  long hyp;

  hyp = Isqrt(((long)x * x ) + ((long)y * y));
  return hyp;
}

//*************************************************************************
**
// Isqrt: Calculate squared.
// Input: Number.
// Return: Square.
//*************************************************************************
**
unsigned long Isqrt (unsigned long n)
{
      unsigned long root;
      unsigned long remainder;
      unsigned long  place;
```

```
        root = 0;
        remainder = n;
    place = 0x40000000; // OR place = 0x4000; OR place = 0x40; - respectively

    while (place > remainder)
        place = place >> 2;
    while (place)
    {
        if (remainder >= root + place)
        {
              remainder = remainder - root - place;
              root = root + (place << 1);
        }
        root = root >> 1;
        place = place >> 2;
    }
    return root;
}

//*************************************************************************
**
// Rad2Deg: Transform ange in radians to angle in degree.
// Input: Angle in radians with four decimals.
// Return: Angle in degree with one decimal.
//*************************************************************************
**

short Rad2Deg(short AngleRad4)
{
    short AngleDeg1;

    AngleDeg1 = ((long)AngleRad4 * 180)/3141;

    return AngleDeg1;
}
```

ภาคผนวก ข
ข้อมูลอุปกรณ์

อุปกรณ์	รูปภาพประกอบ
เซอร์โวมอเตอร์ TOWERPRO 996R	
แบตเตอรี่ ลิโพ 1500mAh	
Arduino MEGA 2560 R3	
บอร์ดเซนเซอร์ MEGA Sensor Shield	
YPG 20A SBEC	

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายทศพร เตชะศักดิ์
วัน เดือน ปีเกิด	25 มกราคม 2542
ที่อยู่	225/139 ซ.หมู่บ้านเศรษฐกิจ22 แขวงบางแคเหนือ เขตบางแค กรุงเทพฯ 10160
E-mail	thodsarpron@hotmail.com
ประวัติการศึกษา	2560 มัธยมศึกษา (วิทย์-คณิต) โรงเรียนอัสสัมชัญธนบุรี จ.กรุงเทพฯ
	ปัจจุบัน 2560-2564 หลักสูตรวิศวกรรมศาสตร์บัณฑิต
	สาขาวิชาวิศวกรรมแมคคาทรอนิกส์ ภาควิชาวิศวกรรมการวัดและควบคุม
	คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อ-นามสกุล	นายพันธกานต์ โนนจุ้ย
วัน เดือน ปีเกิด	18 ตุลาคม 2540
ที่อยู่	119/44 หมู่ 1 ตำบลเนินมะกอก อำเภอบางมูลนาก พิจิตร 66120
E-mail	lancer.ziggs150@gmail.com
ประวัติการศึกษา	พ.ศ. 2559 มัธยมศึกษาตอนปลาย (วิทย์+คณิต) โรงเรียนบางมูลนากภูมิวิทยาคม
	ปัจจุบัน 2560-2564 หลักสูตรวิศวกรรมศาสตร์บัณฑิต
	สาขาวิชาวิศวกรรมแมคคาทรอนิกส์ ภาควิชาวิศวกรรมการวัดและควบคุม
	คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง