หุ่นยนต์ส่งเอกสาร

DOCUMENT SENDING ROBOT

ณรงค์เกียรติ นิระเคน
NARONGKIAT NIRAKEN
ณัฐกุล แต่งสกุล
NATTAKUN TANGSAKUL
ณัฐวุฒิ ธนะทวี
NATTAWUT THANATAWEE

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2558

หุ่นยนต์ส่งเอกสาร

DOCUMENT SENDING ROBOT

จัดทำโดย

ณรงค์เกียรติ นิระเคน รหัส 55010325

ณัฐกุล แต่งสกุล รหัส 55010353

ณัฐวุฒิ ธนะทวี รหัส 55010404

อาจารย์ที่ปรึกษา

รศ.ดร.สุรพันธุ์ เอื้อไพบูลย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2558

ปริญญานิพนธ์ปีการศึกษา 2558

สาขาวิชา วิศวกรรมอิเล็กทรอนิกส์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง หุ่นยนต์ส่งเอกสาร

DOCUMENT SENDING ROBOT

ผู้จัดทำ นายณรงค์เกียรติ นิระเคน รหัสประจาตัว 55010325

นายณัฐกุล แต่งสกุล รหัสประจาตัว 55010353

นายณัฐวุฒิ ธนะทวี รหัสประจาตัว 55010404

ปริญญานิพนธ์นี้ผ่านการตรวจสอบโดยอาจารย์ที่ปรึกษาแล้ว

(รศ.ดร.สุรพันธุ์ เอื้อไพบูลย์)

อาจารย์ที่ปรึกษา

หัวข้อปริญญานิพนธ์	หุ่นยนต์ส่งเอกสาร
นักศึกษา	นายณรงค์เกียรติ นิระเคน 55010325
	นายณัฐกุล แต่งสกุล 55010353
	นายณัฐวุฒิ ธนะทวี 55010404
ปริญญา	วิศวกรรมศาสตร์บัณฑิต
สาขาวิชา	วิศวกรรมอิเล็กทรอนิกส์
ปีการศึกษา	2558
อาจารย์ที่ปรึกษาปริญญานิพนธ์	รศ.ดร.สุรพันธุ์ เอื้อไพบูลย์

บทคัดย่อ

ในยุคของแรงงานคนเริ่มถูกแทนที่ด้วยหุ่นยนต์ทั้งในโรงงานอุตสาหกรรมและบริษัทต่างๆ กลุ่มของข้าพเจ้าจึงเล็งเห็นประโยชน์และความสาคัญของหุ่นยนต์โดยเฉพาะหุ่นยนต์ส่งเอกสาร เพื่อความสะดวกสบายของมนุษย์ในการช่วยทำงานหรือเรื่องส่วนตัวหุ่นตัวนี้สามารถตอบโจทย์ได้เป็นอย่างดี เราใช้บอร์ด STM32F3 DISCOVERY ในการควบคุม ข้อดีของหุ่นยนต์ตัวนี้คือเมื่อคุณต้องการจะส่งเอกสารไปตามแผนกต่างๆในบริษัทหรือส่งเอกสารด่วนในขณะที่คุณกำลังคุมเครื่องจักรทำงานในโรงงานอุตสาหกรรม คุณสามารถทำได้อย่างง่ายดายภายในเวลาที่เร่งรีบ เพียงแค่นำเอกสารใส่หุ่นตัวนี้และตั้งโปรแกรมปลายทางที่ต้องการให้หุ่นตัวนี้ไป โดยเราใช้เอ็นโคดเดอร์ตรวจรู้ตาแหน่งแบบเพิ่มค่าเพื่อตั้งค่าตาแหน่งต่างๆ และป้องการการชนสิ่งกีดขวางของหุ่นยนต์ด้วย IR Sensor มีระบบระบบควบคุมแบบสัดส่วน-ปริพันธ์-อนุพันธ์ ที่จะคำนวณค่าผิดพลาดต่างๆ ของตัวแปรและปรับลดเพื่อให้หุ่นยนต์สมดุลและธรรมชาติยิ่งขึ้น ส่วนการขับเคลื่อนของหุ่นนั้นเราใช้ MOSFET h-bridge to drive dc motor ความเร็วของหุ่นยนต์เทียบเท่ากับความเร็วของมนุษย์เดินหรืออาจจะเร็วกว่าเราจึงตั้งชื่อหุ่นยนต์ตัวนี้ว่า “Document Sending Robot”

Thesis Title	Document Sending Robot
Student	NARONGKIAT NIRAKEN 55010325
	NATTAKUN TANGSAKUL 55010353
	NATTAWUT THANATAWEE 55010404
Degree	Bachelor of Engineering
Program	Electronics Engineering
Year	2015
Thesis Advisor	Assoc. Prof. Dr.SURAPAN AIRPHAIBOON

ABSTRACT

In the age of technology, human was replaced by robotic in manufactory and corporation. Therefore my group foresee useful and important of this. Then, the Document Sending robots is very respond for ours because they will help facilitate. In this robot, we use STM32F3 Discovery Board for controlling. The advantage of this robot is created to deliver your document while you were busy. By put your document to this robot and set the destination that is very easy for you. In this case, we use Incremental Encoder to set position and protect from obstacle by IR SENSOR. The control system has PID Controller to calculate the error of variable for natural balance. In part of propel, we use MOSFET H-Bridge to drive dc motor and the velocity is as same as human or faster. So we named it "Document Sending Robot".

กิตติกรรมประกาศ

การทำโครงงานนี้ สำเร็จลุล่วงไปได้ด้วยดีในทุกๆด้านก็ด้วยความร่วมมือและการช่วยเหลือจาก อาจารย์ รศ.ดร.สุรพันธุ์ เอื้อไพบูลย์ ที่อนุญาตให้ใช้อุปกรณ์ต่างๆสำหรับการทำโครงงานนี้ และช่วยให้ความรู้ ขอบคุณพี่ๆ ทุกคนที่คอยให้คาแนะนำ ขอบคุณชุมนุมโรบอทที่ให้ใช้สถานที่และอุปกรณ์ในการทาหุ่นยนต์ Document sending robot ขอขอบคุณเพื่อนๆ ที่คอยให้กาลังใจและให้ความช่วยเหลือเป็นอย่างดี และสุดท้ายขอขอบคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ให้การศึกษากับผู้จัดทำ

ณงค์เกียรติ นิระเคน
ณัฐกุล แต่งสกุล
ณัฐวุฒิ ธนะทวี

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญ

ในยุคของแรงงานคนเริ่มถูกแทนที่ด้วยหุ่นยนต์ทั้งในโรงงานอุตสาหกรรมและบริษัทต่างๆ กลุ่มของข้าพเจ้าจึงเล็งเห็นประโยชน์และความสำคัญของหุ่นยนต์โดยเฉพาะหุ่นยนต์ส่งของ เพื่อความสะดวกสบายของมนุษย์ในการช่วยทำงาน หุ่นยนต์ตัวนี้สามารถตอบโจทย์ได้เป็นอย่างดี ข้อดีคือสะดวกสบายและประหยัดเวลาในการส่งเอกสาร เพียงแค่นำเอกสารใส่หุ่นตัวนี้และตั้งโปรแกรมปลายทางที่ต้องการ ความเร็วของหุ่นเทียบเท่ากับความเร็วของมนุษย์เดินหรือเร็วกว่า เราจึงตั้งชื่อหุ่นยนต์ตัวนี้ว่า “Document sending robot”

1.2 วัตถุประสงค์ของการศึกษา

1. เพื่อนำอุปกรณ์ที่ได้ออกแบบไปใช้งานได้ และสามารถพัฒนาขึ้นไประดับสูงขึ้น
2. เพื่อฝึกฝนทักษะการทำงานขั้นพื้นฐานในด้านอิเล็กทรอนิกส์
3. เพื่อเป็นการประยุกต์ใช้ความรู้ที่ได้ศึกษาในด้านทฤษฎี มาปรับใช้ในการทำงานจริง

1.3 ขอบเขตการศึกษา

ศึกษาอุปกรณ์และโปรแกรมที่ใช้ในการออกแบบ Document sending robot โดยมีโปรแกรม Altium Design 10, keil uvision 5, STM32CubeMX, Docklight และ FreeRTOS เมื่ออุปกรณ์ทำการประกอบเสร็จสิ้นจะต้องทดสอบอุปกรณ์เครื่องนี้ว่าทำงานได้ตามที่โจทย์ตั้งเอาไว้หรือไม่ ถ้าไม่เป็นไปตามต้องหาสาเหตุว่าทำไมอุปกรณ์เครื่องนี้ถึงไม่เป็นไปตามที่โจทย์วางเอาไว้

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถนำความรู้ไปประยุกต์ใช้ในการศึกษาต่อๆไป
2. เครื่องมือที่เราสร้างขึ้นสามารถนำไปใช้งานได้จริง
3. เข้าใจในหลักการทำงานของหุ่นยนต์

1.5 ส่วนประกอบของรายงาน

รายงานฉบับนี้ได้รวบรวมแนวคิดการทำงาน ทฤษฎีต่างๆที่เกี่ยวข้อง การทดลองและผลที่ได้โดยรวบรวมไว้เป็นบทต่อไปนี้

บทที่ 1 กล่าวถึงความเป็นมา วัตถุประสงค์ และขั้นตอนการทำงาน

บทที่ 2 กล่าวถึงองค์ประกอบและหลักการทำงานของหุ่นยนต์

บทที่ 3 กล่าวถึงหลักการออกแบบหุ่นยนต์

บทที่ 4 กล่าวถึงการทดสอบการใช้งานของอุปกรณ์และผลการทดสอบ

บทที่ 5 กล่าวถึงบทสรุปทั้งหมดของการทำปริญญานิพนธ์ทั้งหมดที่ผ่านมา

บทที่2
ทฤษฎี

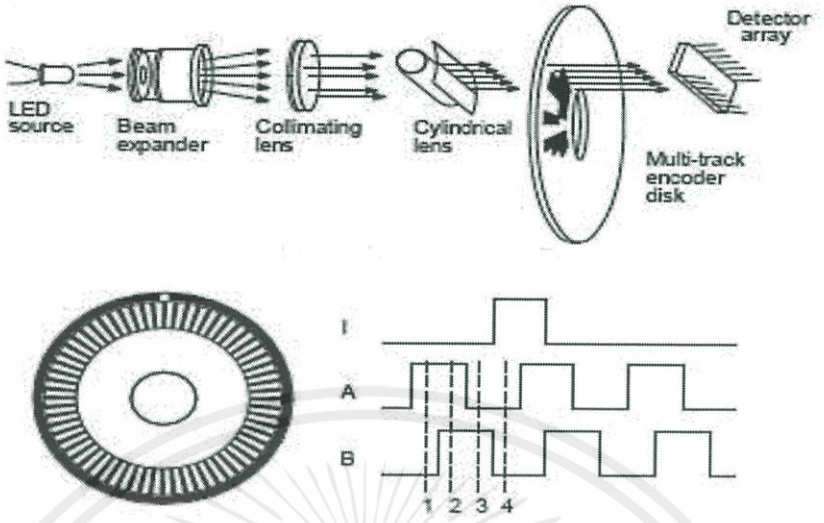
2.1 IR sensor

IR Sensor คืออุปกรณ์ที่นำโฟโต้ไดโอด หรือโฟโต้ทรานซิสเตอร์ มารวมเข้ากับวงจรควบคุมภายใน เพื่อใช้สำหรับความถี่สูงโดยเฉพาะ IR Sensor นั้น จะตอบสนองกับแสงอินฟาเรดเท่านั้น ใช้งานร่วมกับ LED อินฟาเรด นิยมใช้ส่งข้อมูลที่อยู่ในระยะไกล เครื่องใช้ไฟฟ้าที่ใช้งาน IR Sensor ก็จำพวก โทรทัศน์ เครื่องเล่น DVD หรือวิทยุในรถยนต์ กล่องรับดาวเทียม เป็นต้น

รูปที่ 2.1 อุปกรณ์ IR Sensor

2.2 Incremental Encoder

Incremental Encoder หรือโดยทั่วไปเรียกว่า Rotary Encoder จะสร้างสัญญาณพัลส์ (Pulse) ที่แปรผันตรงกับการหมุน ของเพลามอเตอร์ หรือจะหมุนด้วยความเร็วเท่ากับเพลาของมอเตอร์นั่นเอง โดย Rotary Encoder จะประกอบด้วยจานหมุน (Rotary Disk) และอุปกรณ์ตรวจจับ (Sensor) โดยจานหมุนจะมีช่องเล็กๆ (Slit) เมื่อเพลาของมอเตอร์หมุนจะทำให้จานหมุนไปตัดลำแสงของ Sensor ทำให้ชุดรับแสงมีการรับสัญญาณเป็นช่วงๆ จึงทำให้ สัญญาณเอาพุต (Output) มีลักษณะ pulse โดยแสดงดังรูปที่ 2.2



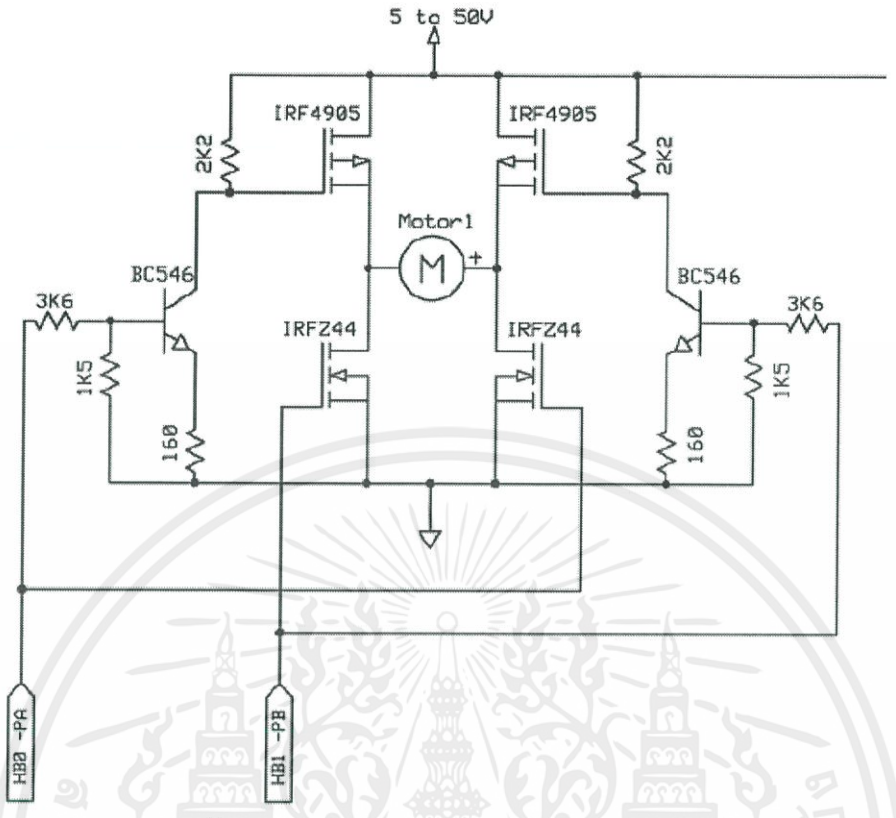
รูปที่ 2.2 Incremental Encode และสัญญาณ Pulse

2.3 motor and driver

กลไกการควบคุมทั่วไปสำหรับหุ่นยนต์ขนาดเล็กคือ การควบคุมความเร็วของมอเตอร์ โดยทั่วไปหุ่นยนต์ จะมีวงจร Drive ที่ใช้สำหรับขับเคลื่อน เพื่อที่จะให้หุ่นยนต์ทา ตามคำสั่งได้เพลา ของมอเตอร์ทั้งสองจะต้องมีการเปลี่ยนที่แม่นยำ อัตราเดียวกันซึ่งขนาดของล้อจะต้องมีขนาด เส้นผ่าศูนย์กลางที่เท่ากันด้วย

MOSFET h-bridge to drive dc motor

หลักการทำงานจะคล้ายกับการสร้างวงจร H-Bridge Switching จาก Transistor โดยที่ใช้ Mosfet เป็นตัว switching ซึ่งสามารถทนกระแสได้สูงกว่า Transistor ทั่วไปและมี switching time ที่ต่ำ ทำให้เหมาะสมกับการใช้การ ควบคุมแบบ PWM ในวงจรจะประกอบด้วย Mosfet p-channel และ n-channel โดยทำงานเป็นคู่ complement โดย 1 และ 4 จะทา งานพร้อมกันเมื่อมีสัญญาณ เข้าที่ PA, 2 และ 3 จะทำงานพร้อมกันเมื่อเราส่งสัญญาณที่ PB โดยจะทำหน้าที่เหมือน switch เปิด และปิดทำให้กระแสไหลผ่านมอเตอร์ในทิศทางกลับด้านเพื่อที่จะควบคุมมอเตอร์ให้กลับทิศทางและ จะมี Freewheeling diode เพื่อป้องกันกระแสไหลย้อนกลับจากมอเตอร์ต่อคร่อม Mosfet แต่ละตัว



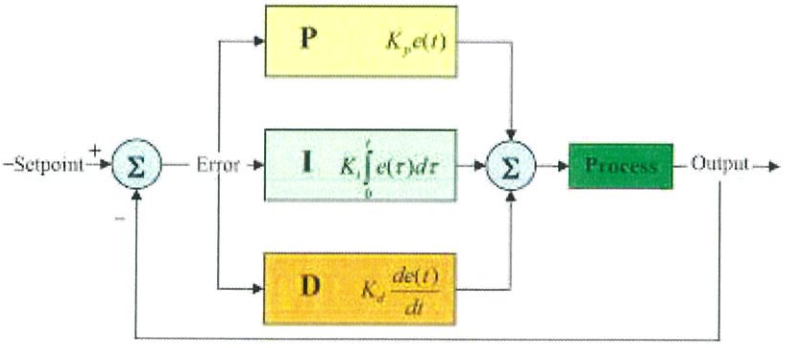
รูปที่ 2.3 Mosfet H-bridge Circuit

ข้อดีของการใช้ Mosfet

1. การทนกระแสที่สูงกว่า เมื่อเปรียบเทียบกับการต่อโดยใช้ Transistor
2. แรงดันตกคร่อมน้อยกว่าเมื่อเปรียบเทียบกับการต่อโดยใช้ Transistor
3. ความเร็วในการสวิทช์เร็วกว่า Transistor

2.4 ระบบควบคุมแบบสัดส่วน-ปริพันธ์-อนุพันธ์ (PID controller)

เป็นระบบควบคุมแบบป้อนกลับที่ใช้กันอย่างกว้างขวาง ซึ่งค่าที่นำไปใช้ในการคำนวณเป็นค่าความผิดพลาดที่หามาจากความแตกต่างของตัวแปรในกระบวนการและค่าที่ต้องการ ตัวควบคุมจะพยายามลดค่าผิดพลาดให้เหลือน้อยที่สุดด้วยการปรับค่าสัญญาณขาเข้าของกระบวนการ ค่าตัวแปรของ PID ที่ใช้จะปรับเปลี่ยนตามธรรมชาติของระบบ



รูปที่ 2.4 แผนผังการทำงานของระบบควบคุมแบบสัดส่วน-ปริพันธ์-อนุพันธ์ (PID controller)

การควบคุมแบบ PID ได้ชื่อตามการรวมกันของเทอมของตัวแปรทั้งสามตามสมการ:

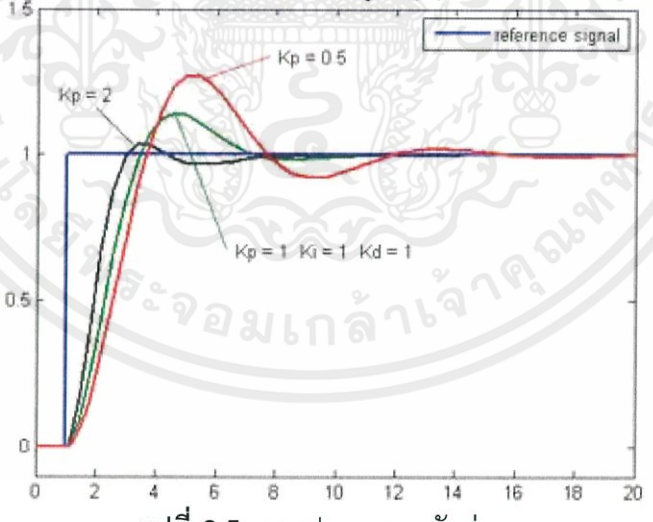
$$\mathrm{MV}(t) = P_{\mathrm{out}} + I_{\mathrm{out}} + D_{\mathrm{out}}$$

(Pout ,Iout ,Dout หมายถึง ผลของสัญญาณขาออกของระบบ)

2.4.1 เทอมของสัดส่วน หรือ P :

บางครั้งบางคราวเราเรียกว่า "อัตราการขยาย" จะเปลี่ยนแปลงเป็นสัดส่วนของค่าความผิดพลาด การตอบสนองของสัดส่วนสามารถทำได้โดยการคูณค่าความผิดพลาดด้วยค่าคงที่ Kp, หรือที่เรียกว่าอัตราขยายสัดส่วน

$$P_{\mathrm{out}} = K_p e(t)$$



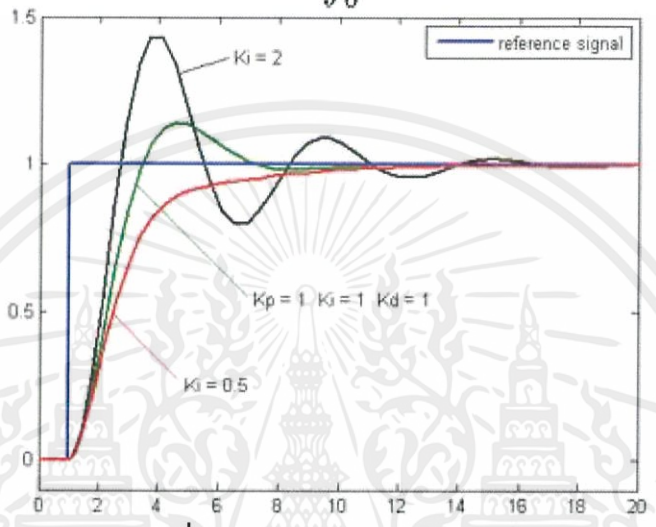
รูปที่ 2.5 กราฟเทอมของสัดส่วน

Pout: สัญญาณขาออกของเทอมสัดส่วน
Kp: อัตราขยายสัดส่วน, ตัวแปรปรับค่าได้
e: ค่าความผิดพลาด = SP − PV
t: เวลา

2.4.2 เทอมของปริพันธ์ หรือ I :

บางครั้งบางคราวเราเรียกว่า "reset" เป็นสัดส่วนของขนาดความผิดพลาดและระยะเวลาของความผิดพลาด ผลรวมของความผิดพลาดในทุกช่วงเวลา (ปริพันธ์ของความผิดพลาด) จะให้ออฟเซตสะสมที่ควรจะเป็นในก่อนหน้า ความผิดพลาดสะสมจะถูกคูณโดยอัตราขยายปริพันธ์ ขนาดของผลของเทอมปริพันธ์จะกำหนดโดยอัตราขยายปริพันธ์, Ki.

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$



รูปที่ 2.6 กราฟเทอมของปริพันธ์

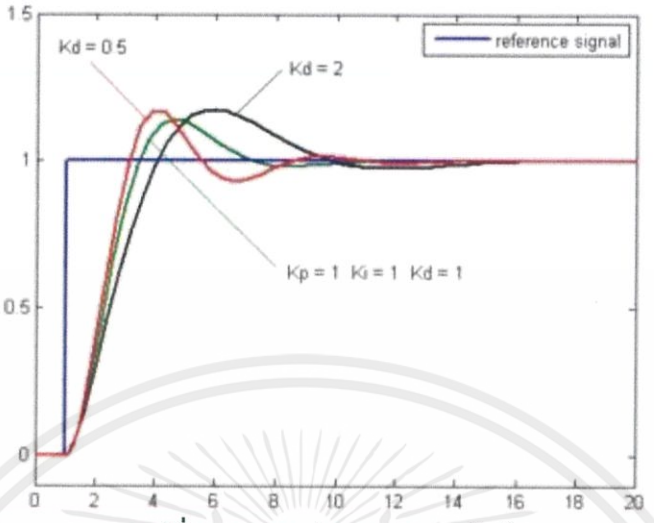
Iout: สัญญาณขาออกของเทอมปริพันธ์
Ki: อัตราขยายปริพันธ์, ตัวแปรปรับค่าได้
e: ความผิดพลาด = SP – PV
t: เวลา
τ: ตัวแปรปริพันธ์หุ่น

เทอมปริพันธ์ (เมื่อรวมกับเทอมสัดส่วน) จะเร่งกระบวนการให้เข้าสู่จุดที่ต้องการและขจัดความผิดพลาดที่เหลืออยู่ที่เกิดจากการใช้เพียงเทอมสัดส่วน แต่อย่างไรก็ตาม เทอมปริพันธ์เป็นการตอบสนองต่อความผิดพลาดสะสมในอดีต จึงสามารถทำให้เกิดโอเวอร์ชูตได้ (หมายถึง ข้ามจุดที่เราต้องการ)

2.4.3 เทอมของอนุพันธ์ หรือ D :

อัตราการเปลี่ยนแปลงของความผิดพลาดจากกระบวนการนั้นคำนวนหาจากความชันของความผิดพลาดทุกๆเวลา (นั่นคือ เป็นอนุพันธ์อันดับหนึ่งสัมพันธ์กับเวลา) และคูณด้วยอัตราขยายอนุพันธ์ Kd ขนาดของผลของเทอมอนุพันธ์ (บางครั้งเรียก อัตรา) ขึ้นกับอัตราขยายอนุพันธ์ Kd

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$



รูปที่ 2.7 กราฟเทอมของอนุพันธ์

Dout: สัญญาณขาออกของเทอมอนุพันธ์
Kd: อัตราขยายอนุพันธ์, ตัวแปรปรับค่าได้
e: ความผิดพลาด = SP – PV
t: เวลา

เทอมอนุพันธ์จะชะลออัตราการเปลี่ยนแปลงของสัญญาณขาออกของระบบควบคุมและด้วยผลนี้จะช่วยให้ระบบควบคุมเข้าสู่จุดที่ต้องการ ดังนั้นเทอมอนุพันธ์จะใช้ในการลดขนาดของโอเวอร์ชูตที่เกิดจาเทอมปริพันธ์และทำให้เสถียรภาพของการรวมกันของระบบควบคุมดีขึ้น แต่อย่างไรก็ตามอนุพันธ์ของสัญญาณรบกวนที่ถูกขยายในระบบควบคุมจะไวมากต่อการรบกวนในเทอมของความผิดพลาดและสามารถทำให้กระบวนการไม่เสถียรได้ถ้าสัญญาณรบกวนและอัตราขยายอนุพันธ์มีขนาดใหญ่เพียงพอ

2.4.4 การนำ PID ไปใช้กับเซนเซอร์

ถ้าเกิดเรามีจำนวนของเซนเซอร์หลายตัวในการที่เราจะใช้กับ PID เราจำเป็นต้องกำหนดให้ตำแหน่งของเซนเซอร์ที่เราต้องการเป็นตำแหน่งสมดุล และให้น้ำหนักของเซนเซอร์แต่ละตัวมีค่าไม่เท่ากัน จากนั้นทำการหาค่าเฉลี่ยของเซนเซอร์ทุกตัว ซึ่งเมื่อนำค่าเฉลี่ยของเซนเซอร์มาหาความแตกต่างกับตำแหน่งเซนเซอร์ที่สมดุลจะทำให้เราได้ค่าความผิดพลาดออกมา จากนั้นทำการเอาค่าความผิดพลาดที่ได้ไปเข้าสมการ PID เพื่อความคุมระบบต่อไป

2.4.5 การปรับแต่งค่า PID เพื่อให้ระบบควบคุมมีประสิทธิภาพ

จากสมการ PID เรามีค่าคงที่อยู่ 3 ตัวคือ Kp , Ki และ Kd ที่ต้องกำหนดค่าเข้าไป การเลือกค่าที่เหมาะสมจะทำให้ระบบสมดุลและได้ผลลัพธ์ตรงตามต้องการมากที่สุด ซึ่งก็คือ

การปรับแต่งค่า PID ส่วนวิธีการปรับ PID สามารถแบ่งได้สองแบบใหญ่ๆ คือ Close loop tuning และ Open loop tuning

- Closed loop คือการจูน PID controller ในโหมด Auto และกำลังคอนโทรลระบบอยู่
- Open loop อาศัย Process model และการตอบสนองของระบบต่อการเปลี่ยนแปลงของ CV แบบ Step ขณะที่ PID controller อยู่ในโหมด Manual

Trial & Error Close-Loop Tuning

วิธีนี้คือการทดลองป้อน Kp, Ki เข้าคอนโทรล แล้วสังเกตุค่าที่ทำให้ระบบสมดุล

ขั้นตอนการปรับค่า

1. เริ่มพล็อตกราฟของ Process variable (PV)
2. เซ็ต Ki และ Kd เป็นศูนย์
3. เซ็ต Kp ค่าน้อยๆ
4. เปลี่ยน PID Controller ให้อยู่ในโหมด Auto
5. เทสระบบโดยเปลี่ยนค่า Set point แล้วสังเกตุว่า damping ของระบบ
6. ปรับค่า Kp เพิ่มขึ้นเรื่อยๆ จนระบบมี damping ตามต้องการ หรือ ให้เป็น quarter-amplitude decay (Decay Ratio ~ 0.25) ดังรูป (b)
7. ถ้าระบบยังมี Offset ระหว่าง Set point และ Process variable ให้ปรับ Ki เพิ่มขึ้นจนไม่มี Offset

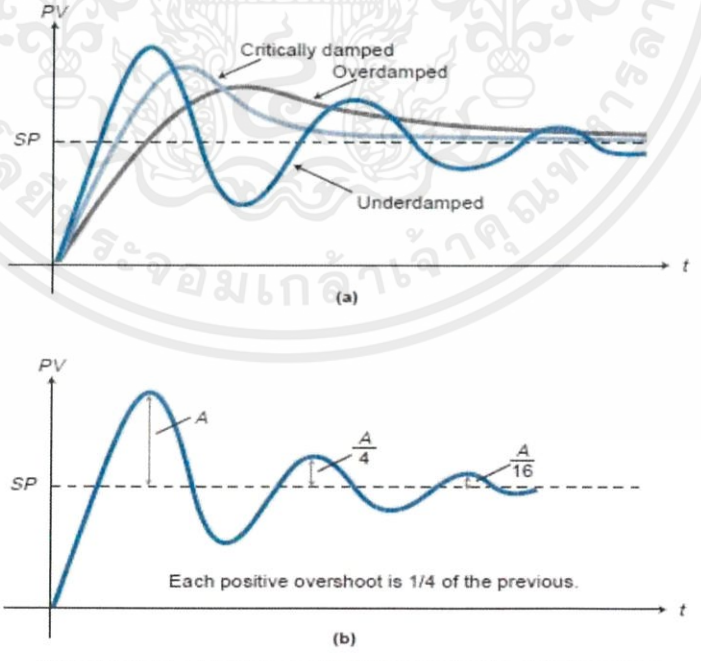


Figure 15-77. Process variable reponses: **(a)** overdamped, critically damped, underdamped, and **(b)** quarter-amplitude.

รูปที่ 2.8 Trial & Error Close-Loop Tuning

หลักของวิธีนี้คือหาค่า gain ที่ทำให้ระบบเกิด Oscillation แบบแอมปริจูดคงที่ ซึ่งเรียกว่าUltimate proportional gain (Kpu) และคาบการสั่น ซึ่งเรียกว่า Ultimate period (Tu) จากนั้นนำทั้งสองค่านี้ไปคำนวณหา Kp, Ki และ Kd ตามตารางด้านล่าง

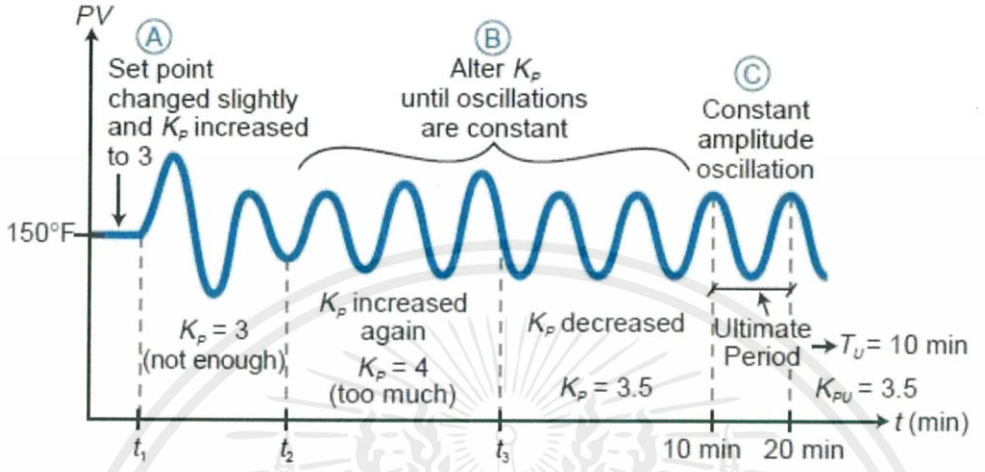


Figure 15-86. System tuned using the Ziegler-Nichols closed-loop tuning method.

รูปที่ 2.9 กราฟการปรับค่าแบบ Ziegler-Nichols Close-Loop Tuning

Type of Controller	Loop Tuning Constant	Tuning Equation *Dependent*	Tuning Equation *Independent*
Proportional (P)	Kp	Kp = 0.5*Kpu	Kp = 0.5*Kpu
Proportional-Integral (PI)	Kp Ki	Kp = 0.45*Kpu Ki = 1.2/Tu	Kp = 0.45*Kpu Ki = (1.2*Kp)/Tu
Proportional-Integral-Derivative (PID)	Kp Ki Kd	Kp = 0.6*Kpu Ki = 2/Tu Kd = Tu/8	Kp = 0.6*Kpu Ki = (2*Kp)/Tu Kd = (Tu*Kp)/8

รูปที่ 2.10 ตารางการปรับค่า Ziegler-Nichols

ขั้นตอนการปรับค่า

1. เริ่มพล็อตกราฟของ Process variable (PV)
2. เซ็ต Ki และ Kd เป็นศูนย์
3. เซ็ต PID Controller ให้อยู่ในโหมด Auto
4. ปรับค่า Kp เริ่มจากค่าน้อยๆ
5. จดค่า Kp และคาบเวลาที่ทำให้เกิด Oscillation แบบแอมปริจูดคงที่
6. คำนวณค่า Kp,Ki และ Kd จากตาราง

ตัวอย่าง

จากรูป Kpu = 3.5 , Tu = 10 นาที และมีสมการ PID เป็นแบบ Dependent

ถ้าใช้ P คอนโทรลอย่างเดียว ค่า Kp = (0.5*3.5) = 1.75

ถ้าใช้ PI คอนโทรล ค่า Kp = (0.45*3.5) = 1.575 , Ki = (1.2/10) = 0.12

ถ้าใช้ PID คอนโทรล ค่า Kp = (0.6*3.5) = 2.1 , Ki = (2/10) = 0.2, Kp = (10/8) = 1.25

Damped Oscillation Close-Loop Tuning

บางระบบไม่ต้องการให้เกิด Oscillation แบบแอมปริจูดคงที่ วิธีนี้จึงหาค่า gain ที่ทำให้ระบบมี damping เป็น quarter-amplitude decay ซึ่งเรียกว่า Damping proportional gain (Kdp) และคาบการสั่น ซึ่งเรียกว่า Damping period (Tdp) แล้วคำนวณค่า Kp, Ki และ Kd จากตารางด้านล่าง

Type of Controller	Loop Tuning Constant	Tuning Equation *Dependent*	Tuning Equation *Independent*
Proportional (P)	Kp	Kp = 1.1*Kdp	Kp = 1.1*Kdp
Proportional-Integral (PI)	Kp Ki	Kp = 1.1*Kdp Ki = 2.6/Tdp	Kp = 1.1*Kdp Ki = (2.6*Kp)/Tdp
Proportional-Integral-Derivative (PID)	Kp Ki Kd	Kp = 1.1*Kpu Ki = 3.6/Tdp Kd = Tdp/9	Kp = 1.1*Kpu Ki = (3.6*Kp)/Tdp Kd = (Tdp*Kp)/9

รูปที่ 2.11 ตารางการปรับค่า Damped Oscillation Close-Loop Tuning

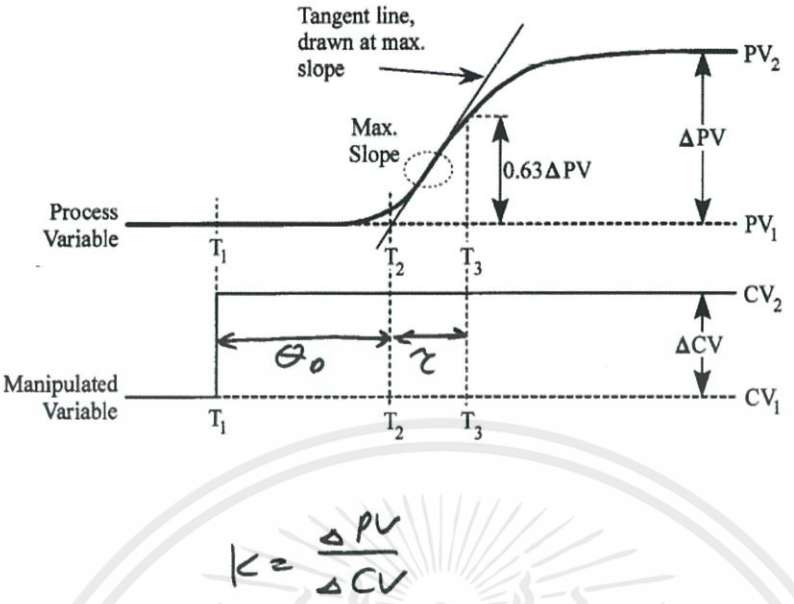
ขั้นตอนการปรับค่า

1. เริ่มพล็อตกราฟของ Process variable (PV)
2. เซ็ต Ki และ Kd เป็นศูนย์
3. เซ็ต PID Controller ให้อยู่ในโหมด Auto
4. ปรับค่า Kp เริ่มจากค่าน้อยๆ เปลี่ยนค่า Set point แล้วสังเกตุว่า damping ของระบบจดค่า Kp ที่ทำให้เกิด quarter-amplitude ratio
5. คำนวณค่า Kp,Ki และ Kd จากตาราง

Open Loop Tuning of Self-Regulating process

ในการจูนแบบ Open loop สำหรับระบบที่มี Steady state (Self-regulation) เราจูนโดยอาศัยการคำนวณค่าจากกราฟการตอบสนองของระบบต่อการเปลี่ยนแปลงของ Control variable (CV) ซึ่งมีอยู่หลายวิธีได้แก่

- Ziegler-Nichols open-loop ให้ผลลัพธ์ของระบบเป็นแบบ Quarter amplitude damping
- Cohen-Coon ให้ผลลัพธ์ของระบบเป็นแบบ Quarter amplitude damping



รูปที่ 2.12 กราฟการปรับค่าแบบ Open Loop Tuning of Self-Regulating process

ขั้นตอนการปรับค่า

1. เริ่มพล็อตกราฟของ Process variable (PV) และ Control variable (CV) เทียบกับเวลา
2. เซ็ต PID Controller ให้อยู่ในโหมด Manual
3. เปลี่ยนค่า CV เพิ่มขึ้น
4. รอจนระบบเข้าสู่ Steady state
5. หยุดพล็อตกราฟ PV และ CV
6. วาดเส้นสัมผัสกราฟ PV จุดที่มีความชันมากสุด
7. T1 คือเวลาที่ CV เริ่มเปลี่ยนแปลง, T2 คือเวลาที่เส้นสัมผัสตัดแกนเวลา คำนวณค่า process dead time θd = T2-T1
8. T3 คือเวลาที่ระบบตอบสนองที่ 63% คำนวณค่า process time constant τ = T2-T1
9. Process gain K = (PV2-PV1)/(CV2-CV1)
10. Process controllability $\alpha = \theta d / \tau$
11. คำนวณค่า Kp,Ki และ Kd จากตาราง

Type of Controller	Loop Tuning Constant	Tuning Equation *Dependent Ziegler-Nichols*	Tuning Equation *Dependent Cohen-Coon*
Proportional (P)	Kp	Kp = 1/Kα	Kp =(1/K)*(0.333+1/α)
Proportional-Integral (PI)	Kp Ki	Kp = 0.9/Kα Ki = 1/3.33θd	Kp = (1/K)*(0.082+0.9/α) Ki = (1+2.2α)/(0.333α+3.33)θd
Proportional-Integral-Derivative (PID)	Kp Ki Kd	Kp = 1.2/Kα Ki = 1/2θd Kd = 0.5θd	Kp = (1/K)*(0.27+1.35/α) Ki = (1+0.6α)/(0.5α+2.5)θd Kd = θd*0.37/(1+0.2α)

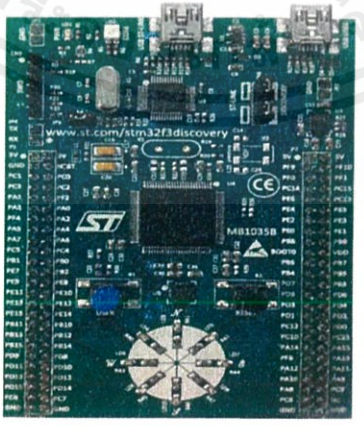
รูปที่ 2.13 ตารางการปรับค่า Open Loop Tuning of Self-Regulating process

2.5 STM32F3DISCOVERY

บอร์ด STM32F3DISCOVERY เป็นบอร์ดไมโครคอนโทรลเลอร์ราคาถูกที่บริษัท STMicroelectronics ได้ผลิตออกมา เพื่อให้นักพัฒนาได้ทดลองใช้งานไมโครคอนโทรลเลอร์ตระกูล STM32 F3 Series และที่มีอยู่บนบอร์ดก็เป็นชิป STM32F303VCT6 (256 KB Flash memory, 48 KB SRAM) ซึ่งเป็นซีพียูแบบ 32-bit ARM Cortex-M4 (DSP, FPU) ทำงานด้วยความถี่ได้สูงถึง 72MHz บอร์ด STM32F3 DISCOVERY ได้รวมวงจรในส่วนที่เรียกว่า ST-LINK/V2 embedded debug tool interface ไว้แล้ว ทำให้ผู้ใช้สามารถโปรแกรมเฟิร์มแวร์ไปยังชิปเป้าหมายและทำขั้นตอนดีบัก (Debug) ผ่านทาง USB ได้อย่างสะดวก และสามารถใช้แรงดันไฟเลี้ยง (5V) ผ่านทางพอร์ต USB ได้เช่นกัน

มีคุณสมบัติดังนี้

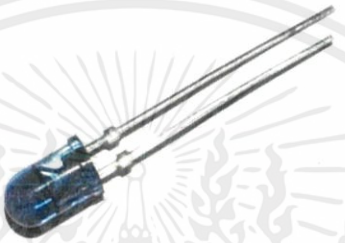
1. ส่วนST-LINK/V2 ใช้สำหรับ DOWNLOAD ข้อมูลจาก PORT USB ของคอมพิวเตอร์, ขั้วต่อ SWD สำหรับต่อภายในบอร์ดและต่อใช้งานออกนอกบอร์ด
2. ส่วนใช้งานจะเป็น MCU เบอร์ STM32F303VCT6 ขนาด 256KB FLASH, 48KB RAM, LQFP 100 PIN
3. ใช้ POWER จากขั้วต่อ USB หรือจากไฟ DC ภายนอก 3V หรือ 5V
4. มี GYROSCOPE แบบ 3-AXIS DIGITAL เบอร์ L3GD20 ของ ST บนบอร์ด
5. มี ACCELERTION SENSOR และ MAGNETIC SENSOR เบอร์ LSM303DLHC ของ ST บนบอร์ด
6. 8 LED แสดงสถานะการทำงานของ GYROSCOPE และ ACCELERTION ในแบบเข็มทิศอีเล็กทรอนิคส์
7. ขั้วต่อใช้งานแบบ USB MINI ต่อกับคอมพิวเตอร์ 1 PORT (สาย USB ไม่มีให้ในชุด ต้องซื้อเพิ่มถ้าไม่มี CABLE USB TO 5P MINI (A-CB-A-00044) * 75.-) และขั้วต่อใช้งานจาก MCU เป็นแบบ USB MINI 1 PORT
8. ตัวบอร์ดทำเป็นขั้วต่อแบบ PIN HEADER ต่อใช้งานได้ PCB ขนาด 25x2 PIN จำนวน 2 ชุด



รูปที่ 2.14 บอร์ด STM32F3 DISCOVERY

2.6 LED Infrared

ไดโอดเปล่งแสงอินฟราเรดหรือ IR LED เป็นอุปกรณ์สำคัญของตัวส่ง ให้แสงในช่วงคลื่นอินฟราเรด (มองด้วยตาเปล่าไม่เห็น) และให้ความเข้มแสงสูงสุดที่เฉพาะค่าความถี่เท่านั้น LED ประเภทนี้มีลักษณะเหมือน LED ทั่วไป มี 2 ขา คือ แอโนด กับ แคโทด ดังนั้นการต่อใช้งาน ก็เหมือนกรณี LED ทั่วไป LED ที่ให้แสงอินฟราเรดแต่ละชนิด สามารถทนกระแสสูงสุด (mA) ได้แตกต่างกัน



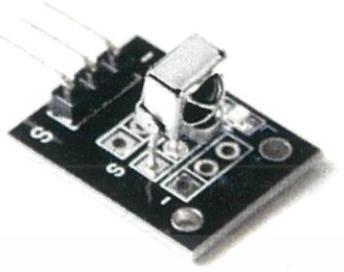
รูปที่ 2.15 LED Infrared

คุณสมบัติ

1. มีความน่าเชื่อถือสูง
2. มีความข้นของการแผ่รังสีสูง
3. ความยาวคลื่น 940 นาโนเมตร (λp=940nm)
4. ขนาด 2.54 มิลลิเมตร
5. แรงดันไฟฟ้าต่ำ
6. ไม่มีสารจะกั่ว
7. ผลิตภัณฑ์อยู่ในมาตรฐานของ RoHS

2.7 Infrared receiver module KY-022

โมดูลรับอินฟราเรด KY-022 ตรงส่วนของตัวรับอินฟราเรดมีตัวครอบหัวเพื่อป้องกันการรบกวนจากสนามไฟฟ้า สามารถทำงานได้ภายใต้ความเข้มแสง 500 lux ใช้กันอย่างแพร่หลายใน: สเตอริโอ, ทีวี, เครื่องวิดีโอ, เครื่องแผ่น, set- topbox, กรอบรูปดิจิตอล, สเตอริโอรถ, ของเล่นควบคุมระยะไกล, เครื่องรับสัญญาณดาวเทียม, ผู้เล่นฮาร์ดดิสก์, เครื่องปรับอากาศ, เครื่องทำความร้อน, พัดลมไฟฟ้า, ให้แสงสว่างและเครื่องใช้ภายในบ้านอื่นๆ



รูปที่ 2.16 Infrared receiver

คุณสมบัติ

1. ระยะการควบคุม: ไกลกว่า 8 เมตร
2. ความยาวคลื่น: 940 นาโนเมตร
3. Crystal frequency: 455 kHZ crystal
4. ตอบสนองที่ความถี่ 38 kHZ
5. เป็นการเข้ารหัส NEC
6. ขนาด 86*40*6 มิลลิเมตร
7. Power: CR2025/1600mAH

2.8 Atmega168

ATMEGA168 เป็นหนึ่งในไมโครคอนโทรลเลอร์อนุกรมAVR ที่ผลิตโดย Atmel Corporation สหรัฐอเมริกา

คุณสมบัติเด่นของ ATmega8 สามารถสรุปได้ดังนี้

- เป็นไมโครคอนโทลเลอร์ 8 บิต ในอนุกรม AVR มีสถาปัตยกรรมแบบ Advance Risc มีความเร็วในการทำงานสูง โดยสามารถประมวลผล 1 คาสั่งในเวลา 1 สัญญาณนาฬิกา สามารถทำงานกับความถี่สัญญาณนาฬิกาสูงสุด 20 MHz จึงสามารถประมวลคำสั่งได้สูงถึง 20 ล้านคำสั่งต่อวินาที
- มีหน่วยความจำ 3 แบบเพื่อรองรับการทำงานประกอบด้วย

1. หน่วยความจำแบบแฟลช (Flase Memory) ความจุ 16 กิโลไบต์ ที่สามารถรักษาข้อมูลโปรแกรมไว้ได้แม้ไม่มีไฟเลี้ยง และสามารถป้องกันการอ่านได้ สามารถลบ-เขียนใหม่ได้ 10,000 รอบ
2. หน่วยความจำข้อมูลอีอีพรอม (EEPROM Memory) ความจุ 512 ไบต์ สามารถรักษาข้อมูลไว้ได้แม้ไม่มีไฟเลี้ยง สามารถลบ -เขียนใหม่ได้ 100,000 รอบ
3. หน่วยความจำข้อมูลแรม (RAM) ความจุ 1 กิโลไบต์ ใช้ประมวลผลหลักร่วมกับซีพียู

- มีพอร์ตอินพุตเอาท์พุตอิสระที่สามารถโปรแกรมได้ 23 ขา แบ่งเป็น 3 กลุ่มคือ

1. พอร์ต B ใช้งานได้สูงสุด 8 ขา (PB0-PB7) โดยมี 2 ขา ใช้งานร่วมกับคริสตอลเพื่อกำเนิดสัญญาณนาฬิกาให้แก่ไมโครคอนโทลเลอร์
2. พอร์ต C ใช้งานได้ 7 ขา(PC0-PC6) โดย PC0-PC5 สามารถใช้เป็นอินพุตอนาล็อกได้

3. พอร์ต D 8 ขา (PD0 ถึง PD7)

- มีไทเมอร์/เคาน์เตอร์ ขนาด 16 บิต 1 ชุด รองรับการทำงานสมบูรณ์แบบทั้งในโหมดตั้งเวลา (Timer),ตัวนับ (Counter),เปรียบเทียบสัญญาณ (Compare) และตรวจจับสัญญาณ (Capture)
- มีโมดูลกำเนิดสัญญาณ PWM 3 ชุด
- มีโมดูลแปลงสัญญาณอนาล็อกเป็นดิจิตอล(Analog to Digital Converter : ADC) ความละเอียด 10 บิต จานวล 6 ช่อง
- มีอินพุตเปรียบเทียบสัญญาณอนาล็อก 2 ช่อง
- มีโมดูลสื่อสารข้อมูลอนุกรม 2 สาย รองรับการทางานกับบัส I2C
- มีโมดูลเชื่อมต่ออุปกรณ์อนุกรมหรือ SPI (Serial Peripheral Interface) ใช้สาหรับการโปรแกรมหน่วยความจาแบบแฟลช
- มีโมดูลสื่อสารอนุกรม UART (Universal asynchronous Receiver Transmitter)
- มีวอตช์ด็อกไทเมอร์เพื่อช่วยตรวจสอบการทางานของระบบ
- รองรับการอินเทอร์รัปต์ ทั้งจากสัญญาณภายนอกและการทางานของโมดูลต่างๆภายในไมโครคอนโทลเลอร์
- สามารถใช้ไฟเลี้ยง 1.8-5.5 V

รูปที่ 2.17 Atmega168

2.9 STM32F429DISCOVERY

1. ไมโครคอนโทรลเลอร์ STM32F429ZIT6 มีหน่วยความจำแฟลช 2 MB, แรม 256 KB ในแพคเกจ LQFP144

2. บนบอร์ด ST-LINK / V2 กับสวิตช์โหมดการเลือกใช้ชุดเป็นแบบสแตนด์อโลน STLINK / V2 (ที่มีการเชื่อมต่อ SWD สำหรับการเขียนโปรแกรมและแก้จุดบกพร่อง)

3. แหล่งจ่ายไฟในบอร์ด : การจ่ายแรงดันผ่านบัส USB หรือจากภายนอก 3 V หรือ 5 V

4. L3GD20, เซ็นเซอร์การเคลื่อนไหว ST MEMS, สัญญาณดิจิตอล 3 แกนหมุน

5. TFT LCD (ฟิล์มบางทรานซิสเตอร์จอภาพผลึกเหลว) 2.4 "262K สี RGB, 240 x 320 จุด

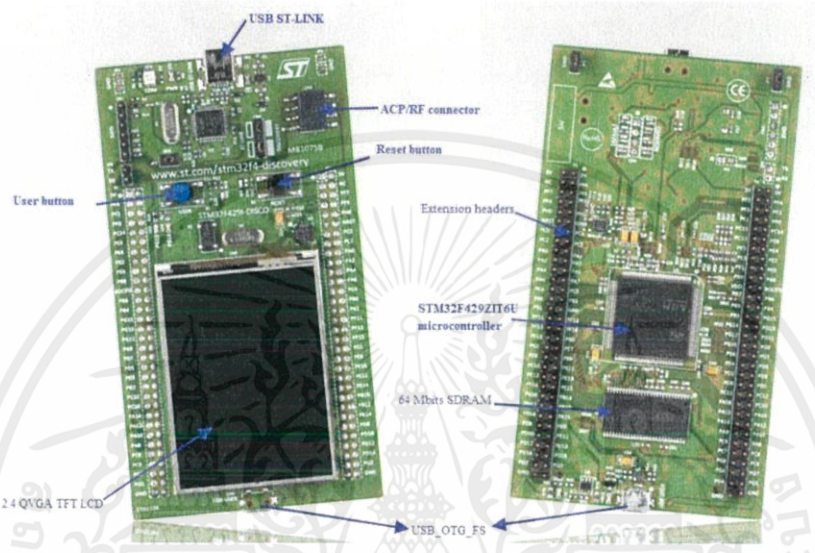
6. SDRAM 64 Mbits (1 Mbit x 16-bit x 4-bank) ซึ่งรวมถึงโหมดรีเฟรชอัตโนมัติและประหยัดพลังงาน

7. Six LEDs:

- LD1 (แดง / สีเขียว) สำหรับการสื่อสาร USB
- LD2 (สีแดง) 3.3 V เปิดเครื่องสำหรับการใช้งาน
- LED สองตัวบอกสถานะการใช้ :
- LD3 (สีเขียว), LD4 (สีแดง)

- OTG LED สองตัว:
- LD5 (สีเขียว) และ VBUS LD6 (สีแดง) OC (กระแสเกิน)

8. ปุ่มกดสองปุ่ม (ผู้ใช้และรีเซ็ท)
9. OTG USB ต่อกับช่องเสียบ Micro-AB
10. Extension header

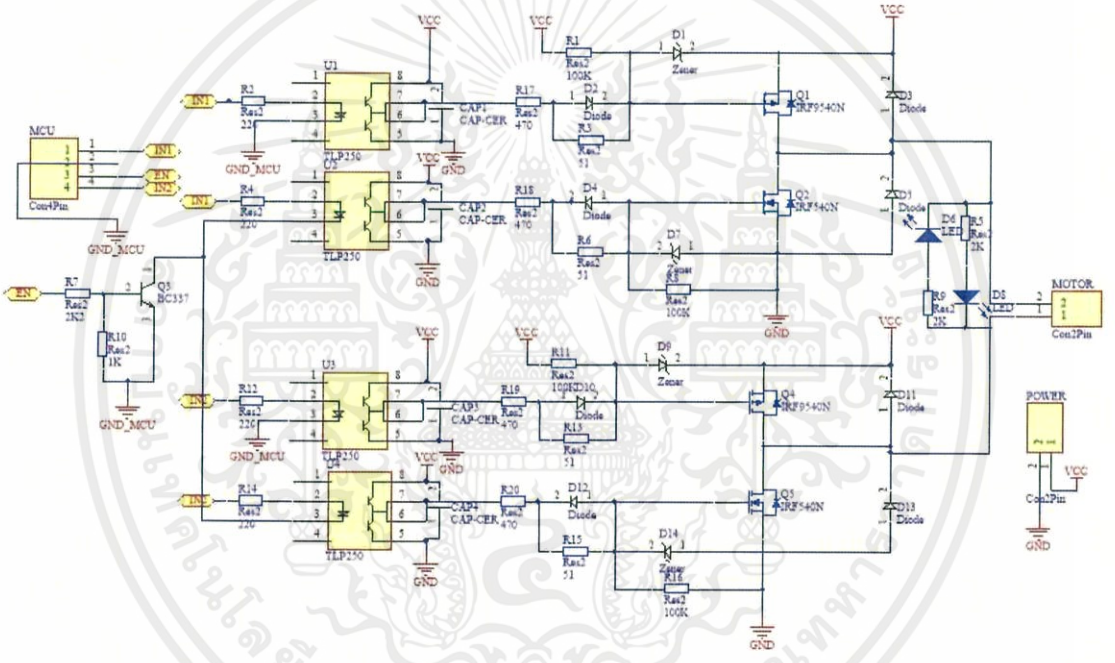


รูปที่ 2.18 STM32F429Discovery

บทที่ 3
การออกแบบ

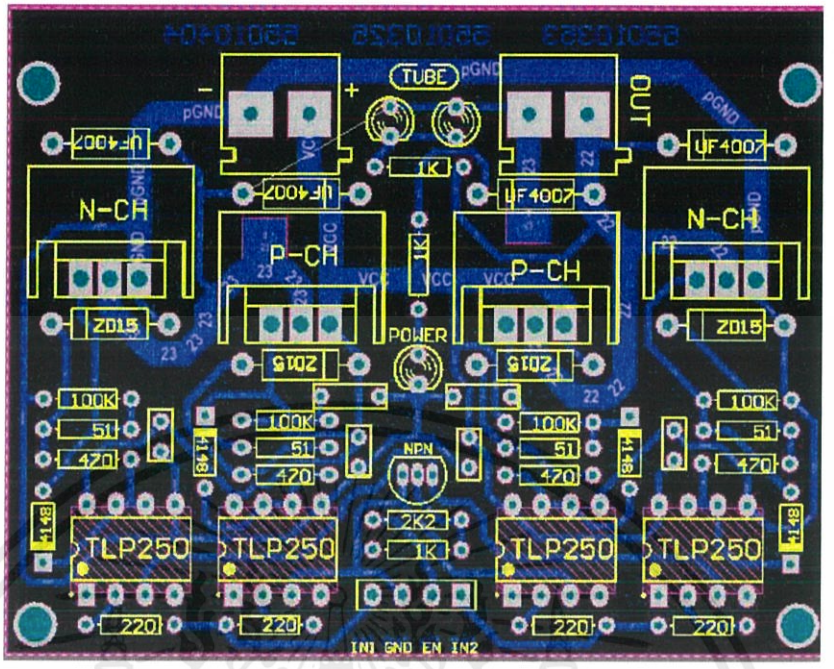
3.1 การออกแบบวงจร Motor

ทำการออกแบบ Schematic ของ Motor Driver โดยใช้ Mosfet 4 ตัวในในการขับ และใช้ Opto isolator เพื่อทำการแยก กราวน์ ของ มอเตอร์ กับ ไมโครคอนโทรลเลอร์ เพื่อให้เกิดความปลอดภัยแก่ ไมโครคอนโทรลเลอร์ และ คอมพิวเตอร์ของผู้ใช้งาน



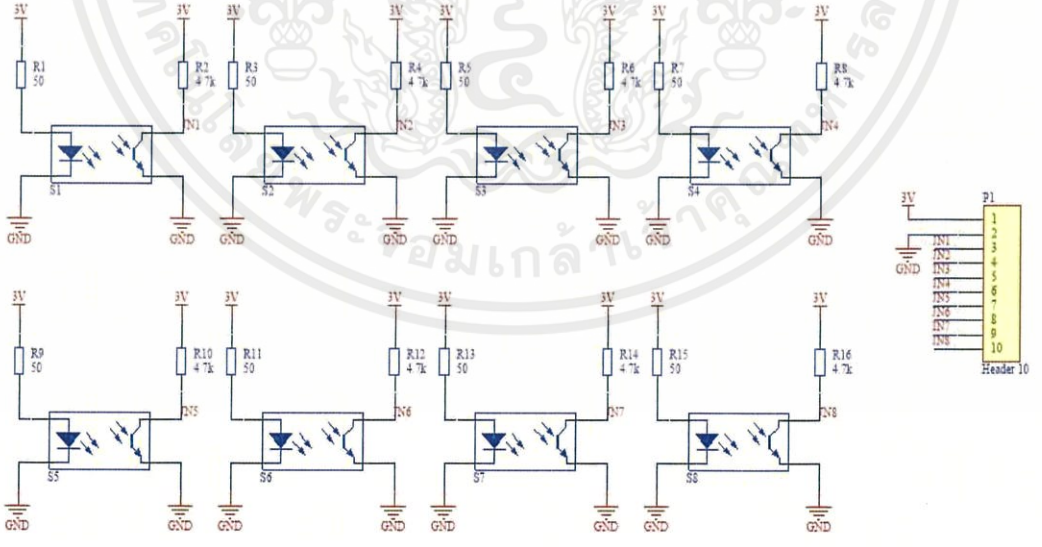
รูปที่ 3.1 Schematic Motor

ทำการออกแบบ PCB โดยคำนึงถึงกระแสที่ไหลในแต่ละจุด

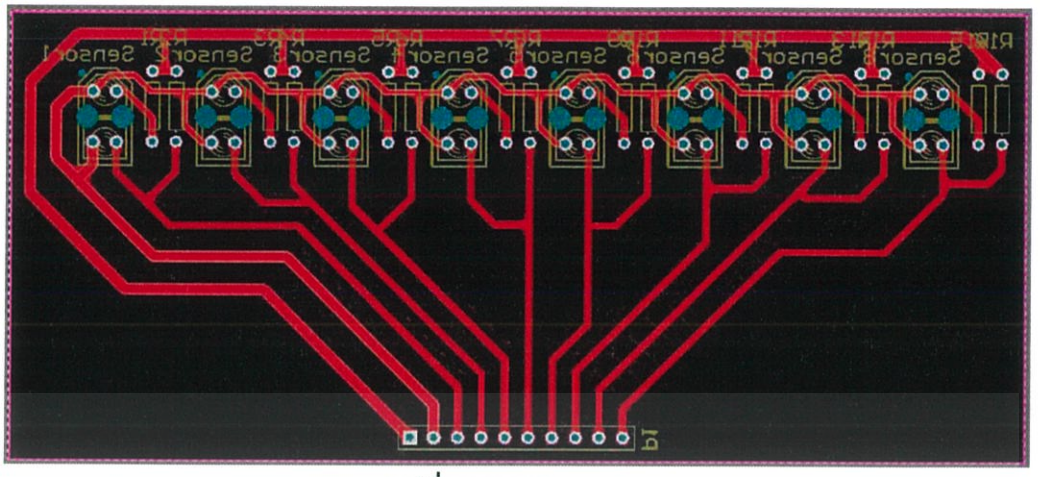


รูปที่ 3.2 PCB Motor

3.2 การออกแบบวงจร Sensor

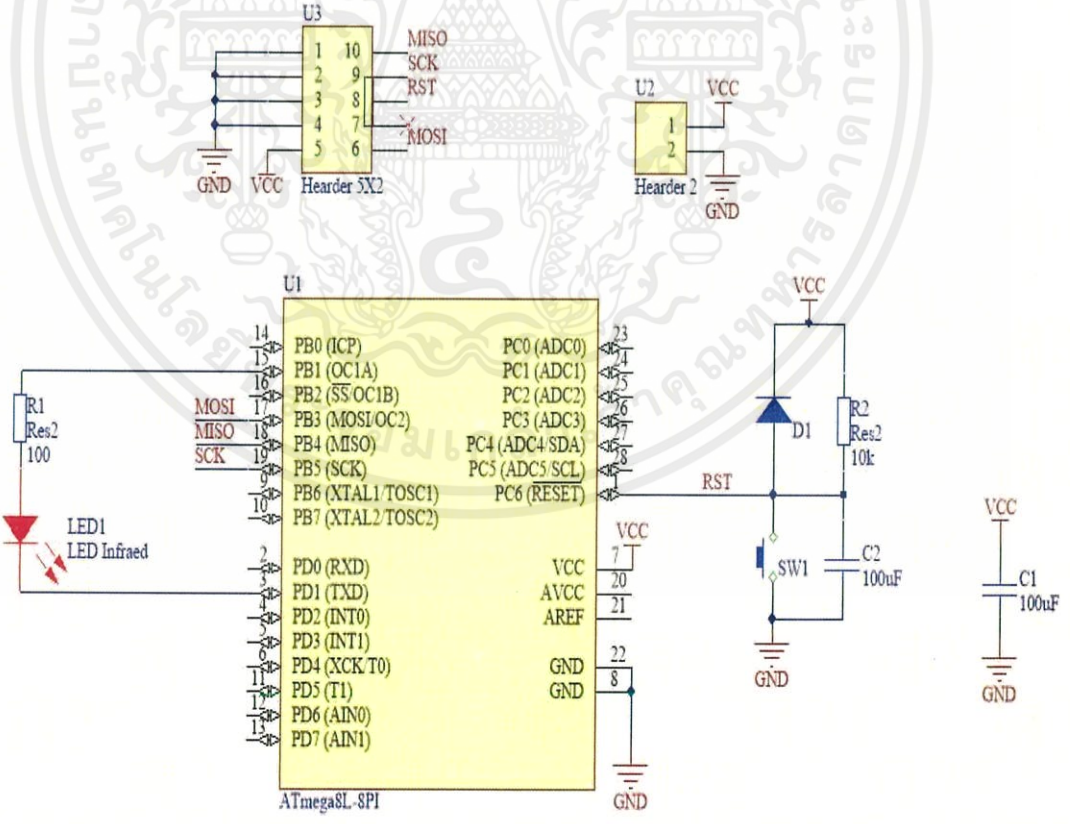


รูปที่ 3.3 Schematic sensor

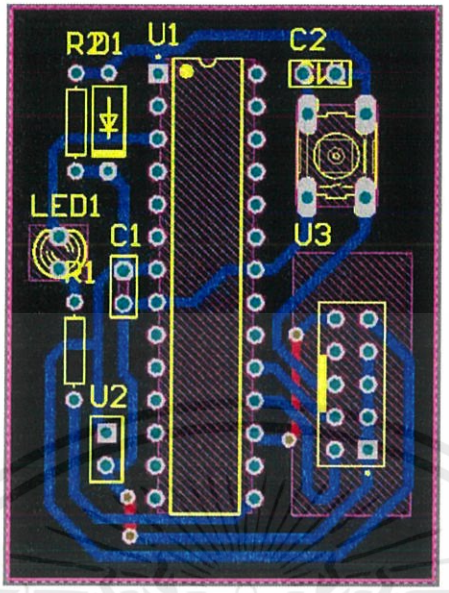


รูปที่ 3.4 PCB sensor

3.3 การออกแบบวงจรฝั่งตัวส่ง



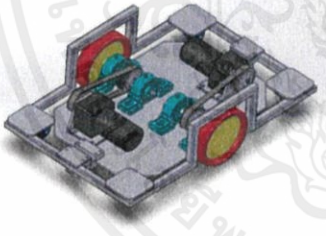
รูปที่ 3.5 Schematic ฝั่งตัวส่ง



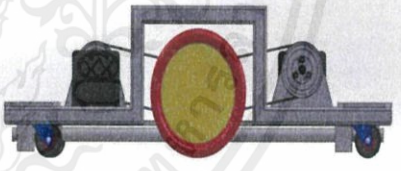
รูปที่ 3.6 PCB ฝั่งตัวส่ง

3.4 การออกแบบหุ่นยนต์ในโปรแกรม

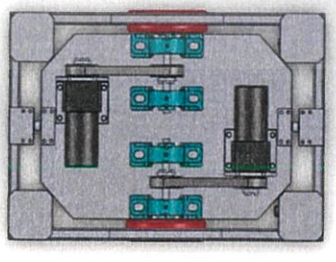
ทำการออกแบบตัวหุ่นยนต์โดยใช้โปรแกรม Solid work ดังรูป 3.7 – 3.12



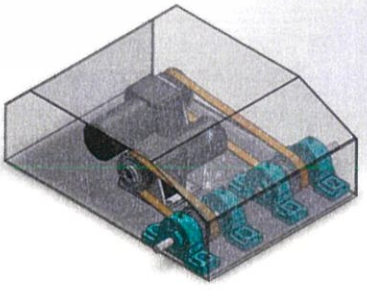
รูปที่ 3.7



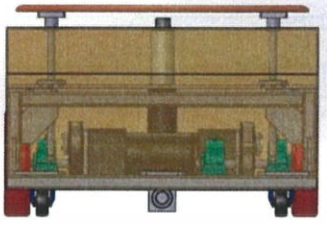
รูปที่ 3.8



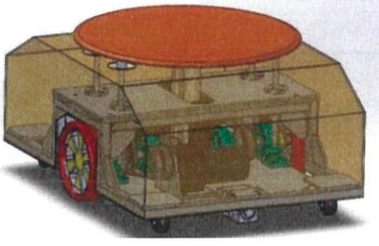
รูปที่ 3.9



รูปที่ 3.10



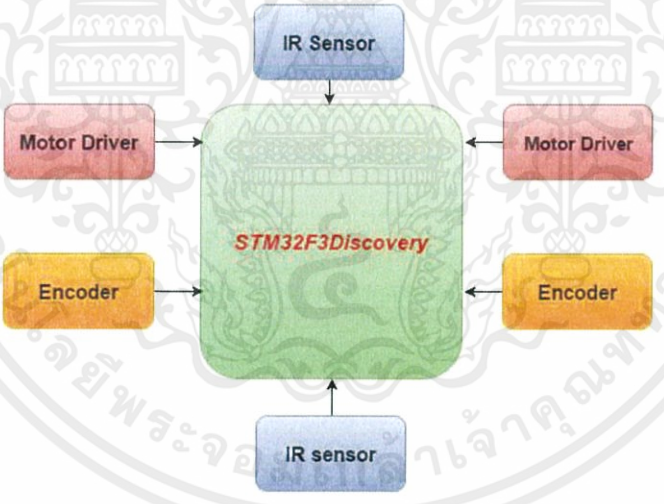
รูปที่ 3.11



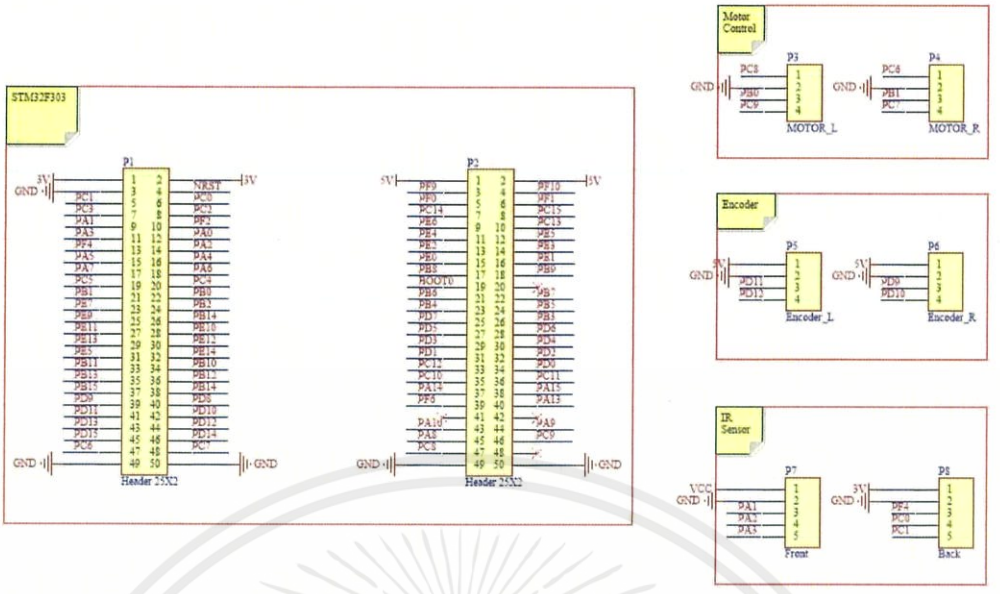
รูปที่ 3.12

3.5 การออกแบบบอร์ด STM32F3Discovery

ในการออกแบบการทำงานส่วนของระบบ ควบคุมหุ่นยนต์ โดยอ่านค่าจาก เซนเซอร์ และ Encoder โดยใช้บอร์ด ST32F3Discovery เป็นไมโครคอนโทรลเลอร์เพื่อใช้ควบคุม



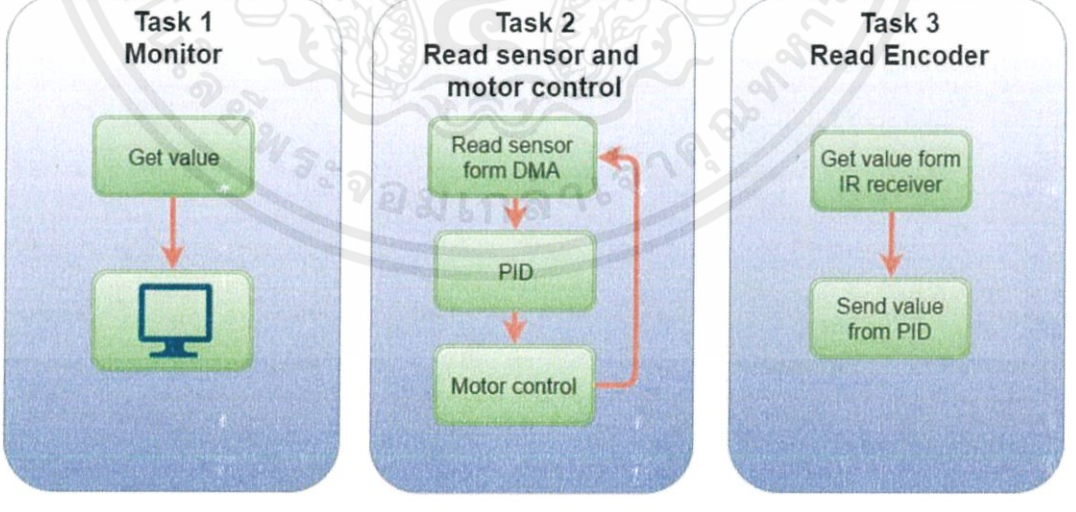
รูปที่ 3.13 แผนภาพการทำงานของ STM32F3Discovery



รูปที่ 3.14 Schematic บอร์ด STM32F3Discovery

3.6 การออกแบบโปรแกรม

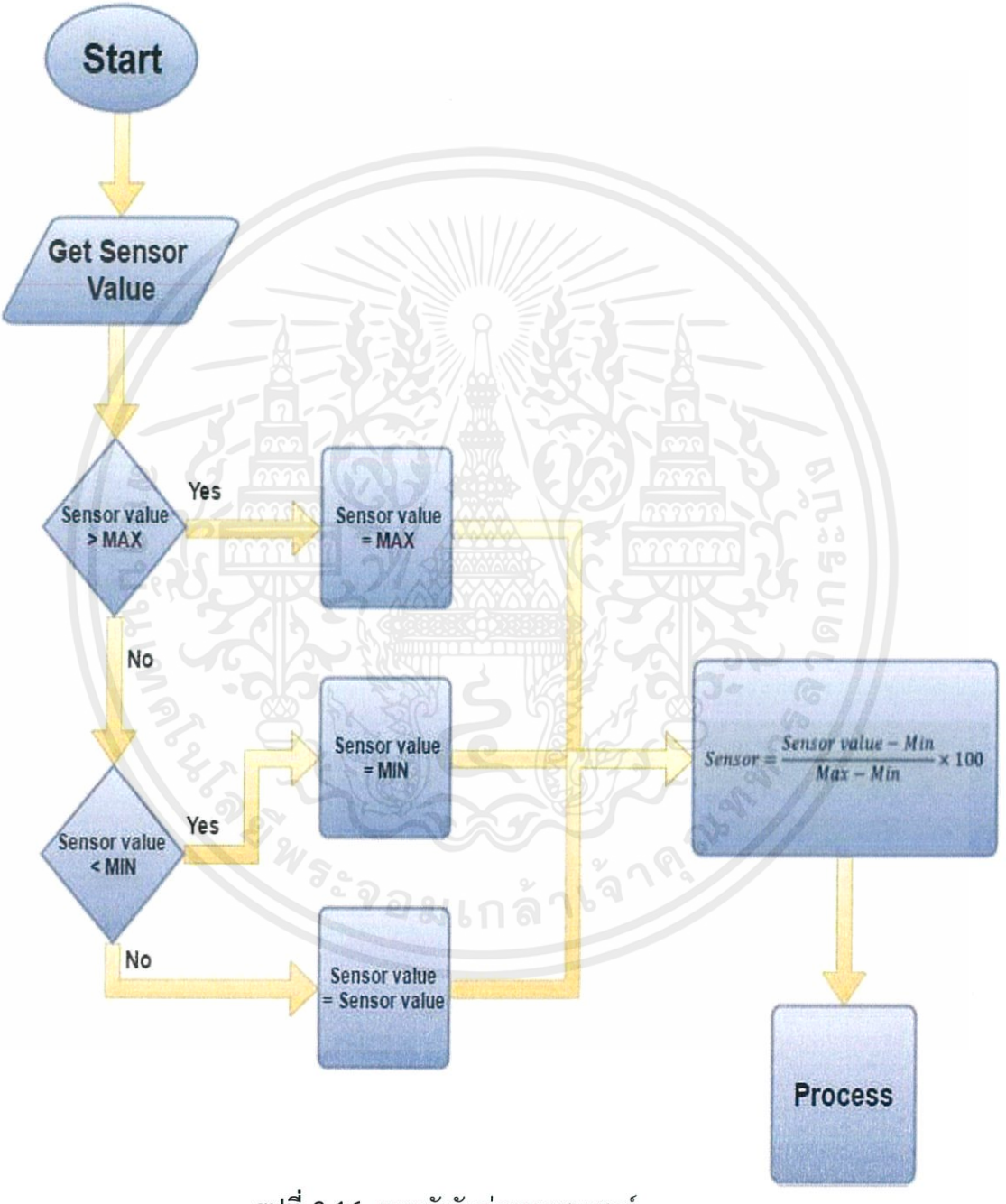
ทำการเขียนโปรแกรมเพื่อควบคุมตัวหุ่นยนต์ ในการเขียนโปรแกรมจะแบ่งเป็น 3 Task ในแต่ละ Task จะทำงานต่างกัน โดยที่ Task 1 ทำงานดึงค่า เซนเซอร์แต่ละตัวแล้วส่งมาแสดงผลที่คอมพิวเตอร์ ส่วนใน Task 2 ทำการอ่านค่าของ เซนเซอร์ผ่าน DMA และ ดึงค่า Encoder ที่อ่านได้มาใช้ใน PID control เพื่อควบคุมมอเตร์ ส่วนใน Task 3 ทำการอ่านค่า IR Receiver Module แล้วเก็บค่าไว้ในตัวแปรเพื่อทำการหยุดหรือทำงานต่อของหุ่นยนต์



รูปที่ 3.15 แผนผังการออกแบบโปรแกรม

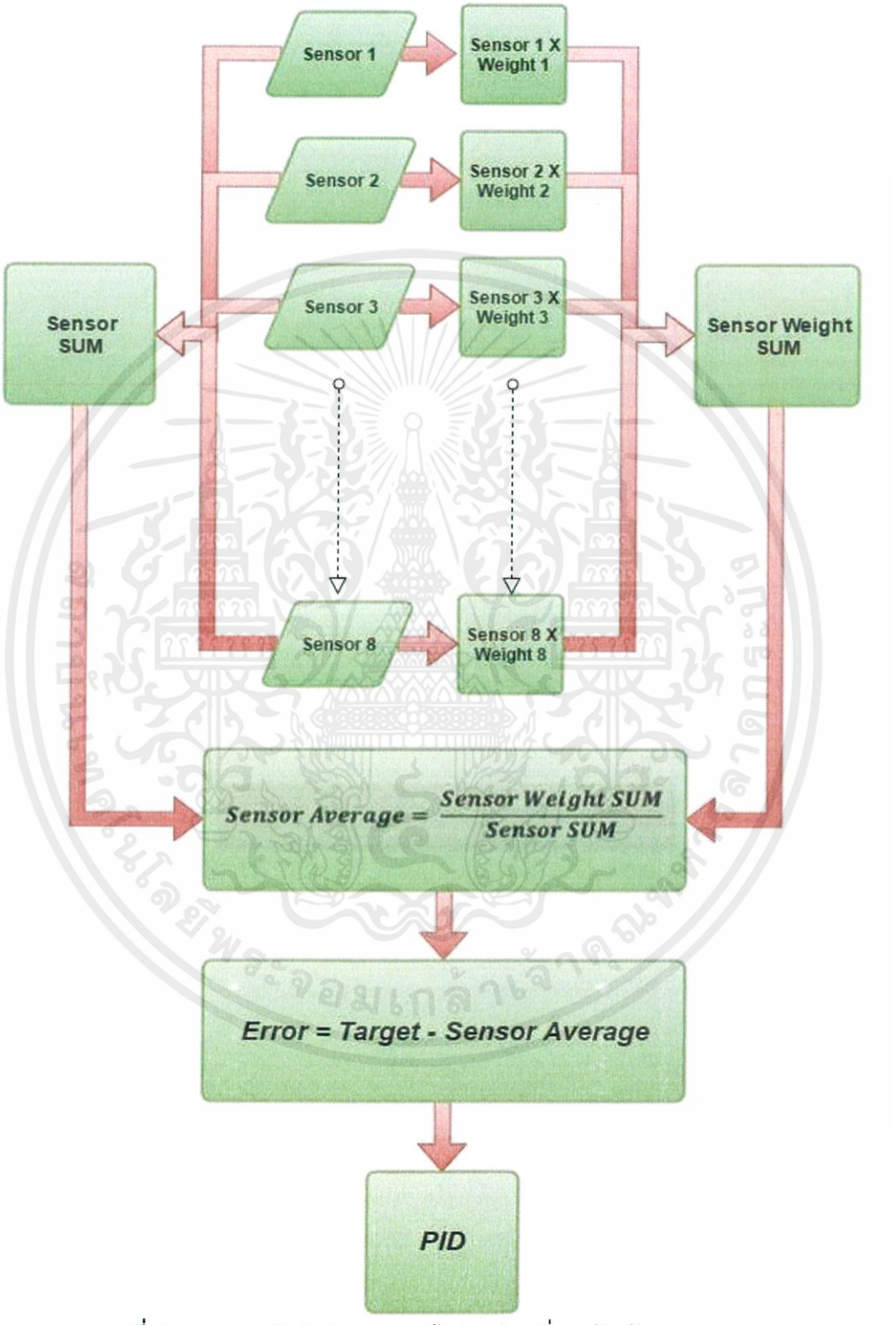
3.7 อัลกอริทึมของการใช้เซนเซอร์ในระบบความคุมPID

ช่วงแรกเป็นการรับค่าของเซนเซอร์ก่อนเข้าระบบการทำงาน โดยช่วงนี้จะรับค่าเซนเซอร์ 8 ตัว นำแต่ละตัวมาหาค่าสูงสุด (MAX) และค่าต่ำสุด (MIN) เนื่องจากเซนเซอร์แต่ละตัวให้ค่าออกมาไม่เท่ากัน เราจึงทำการแปลงให้อยู่ในหน่อยเดียวกัน คือ 100 เปอร์เซ็น เพื่อเพิ่มประสิทธิภาพก่อนที่จะเข้าระบบการทำงาน



รูปที่ 3.16 แผนผังรับค่าจากเซนเซอร์

ช่วงที่สองเป็นการนำค่าของเซนเซอร์แต่ละตัวมาหาค่าเฉลี่ย จากนั้นนำค่าที่ได้ไปลบออกจากค่าที่เราต้องการจะได้ค่าความผิดพลาดมา จากนั้นนำค่าความผิดพลาดไปเข้าระบบ PID



รูปที่ 3.17 แผนผังนำค่าเซนเซอร์ไปหาค่าเฉลี่ย แล้วเข้าระบบ PID

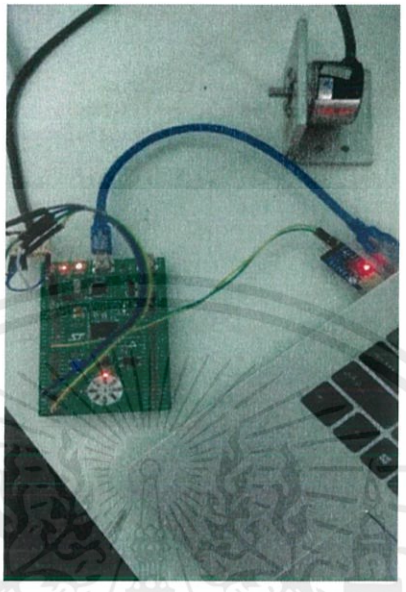
บทที่ 4
การทดลอง

4.1 อ่านค่า Encoder

อ่านค่าที่ได้จาก Encoder โดยใช้โปรแกรม Docklight โดยในรูปแสดงเลขฐาน 16 ซึ่งถ้าหมุนไปข้างหน้าจะทำให้ค่าเพิ่มขึ้น ส่วนถ้าหมุนกลับหลังจะทำให้ค่าลดลง

รูปที่ 4.1 ค่าที่ได้จาก Encoder

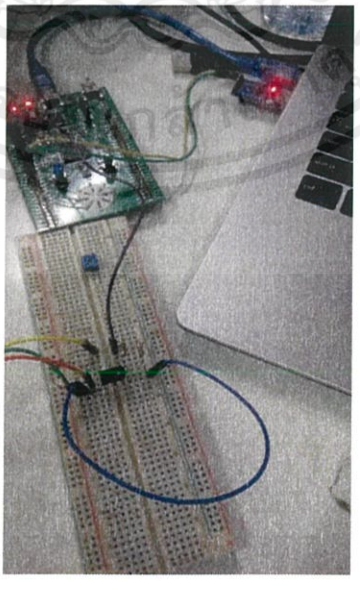
ต่อวงจรอ่านค่าจาก Encoder โดยใช้ STM32F3Discovery ต่อเข้ากับ Encoder และส่งค่าที่ได้แสดงผลที่คอมพิวเตอร์ผ่าน USB to serial โดยสื่อสารผ่าน USART1



รูปที่ 4.2 การ Test Read Encoder

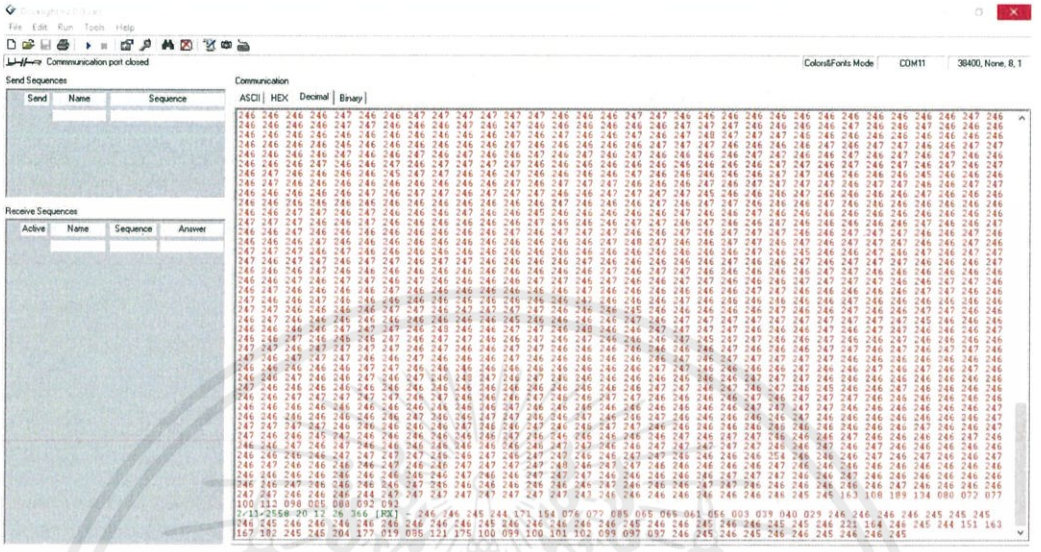
4.2 อ่านค่า IR Sensor

อ่านค่าจาก IR Sensor โดยเชื่อมต่อกับบอร์ด STM32F3Discovery ทาง ADC1 Channel 2 และส่งค่าที่ได้แสดงผลที่คอมพิวเตอร์ผ่าน USB to serial โดยสื่อสารผ่าน USART1



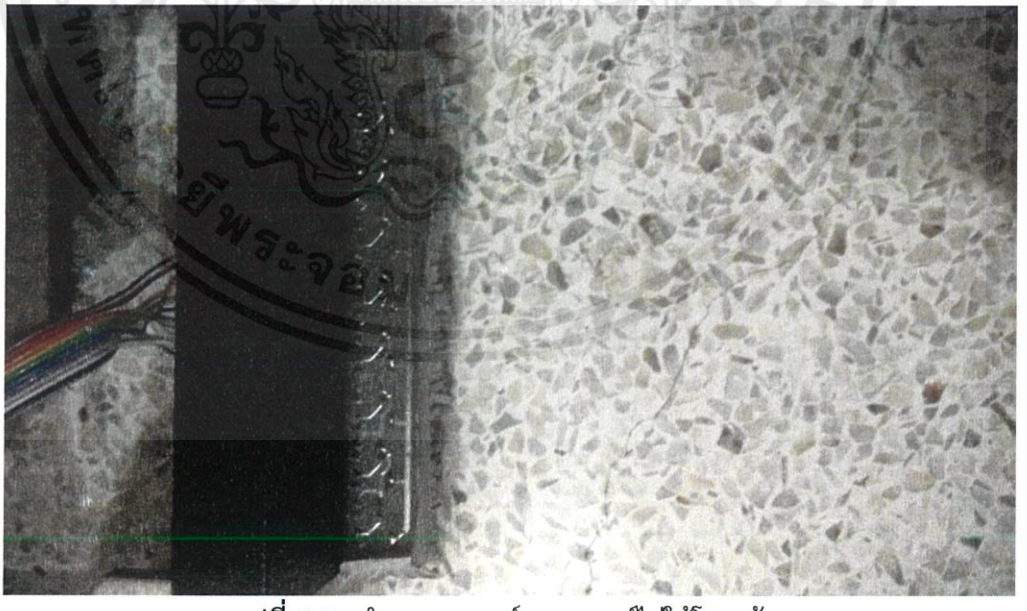
รูปที่ 4.3 การอ่านค่าจาก IR Sensor โดยเชื่อมต่อกับบอร์ด STM32F3Discovery

อ่านค่าที่ได้จาก IR Sensor โดยใช้โปรแกรม Docklight โดยในรูปแสดงเลขฐาน 10 ซึ่งค่าที่ได้อยู่ระหว่าง 70 – 247 ซึ่งจะในไปใช้ในระบบคอนโทรลต่อไป

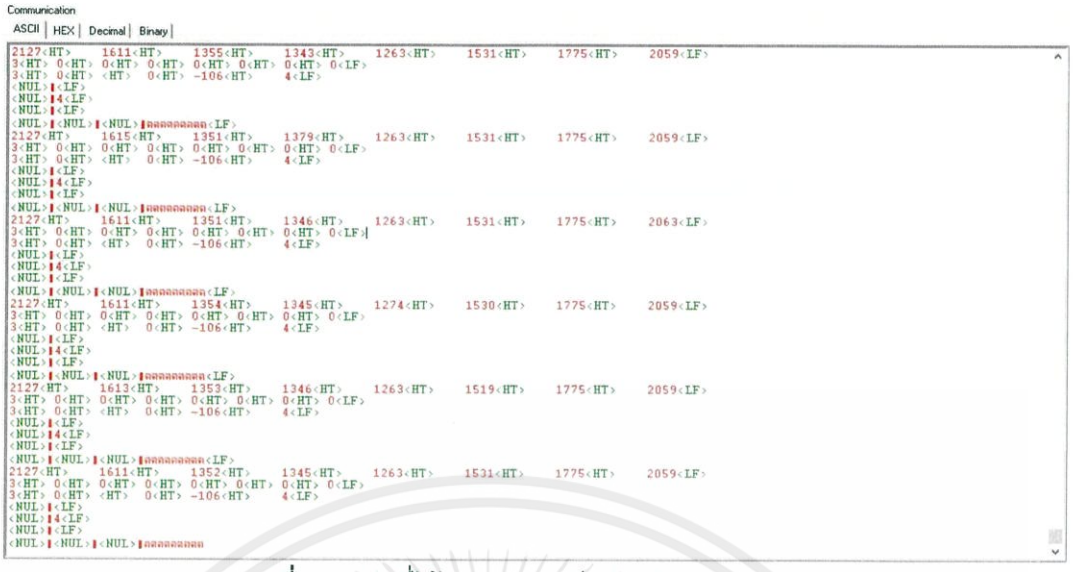


รูปที่ 4.4 ค่าที่ได้จาก เซนเซอร์อินฟาเรด ตอนที่ทดสอบกับโปรโตบอร์ด

ทำการอ่านค่าของเซอร์อินฟาเรด จากบอร์ดเซนเซอร์ที่ติดอยู่กับตัวหุ่นยนต์ แบบแรกทำการอ่านค่าของเซนเซอร์โดยที่ตัวของเซนเซอร์ไม่โดนเส้นสีดำเลย

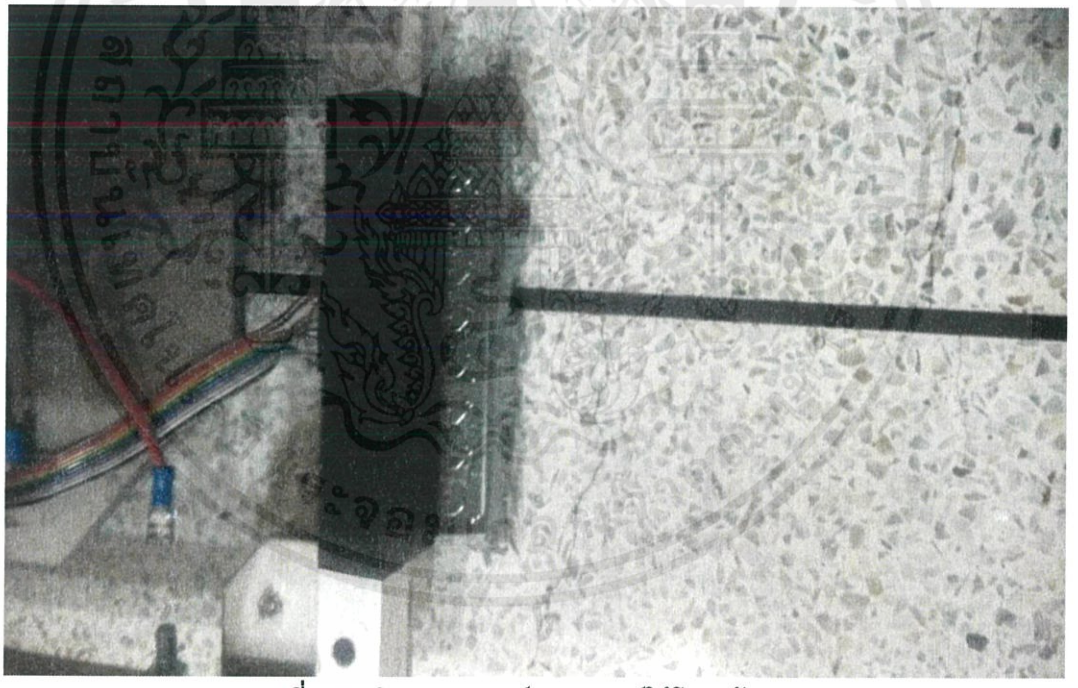


รูปที่ 4.5 ทำการวางบอร์ดเซนเซอร์ไม่ให้โดนเส้น

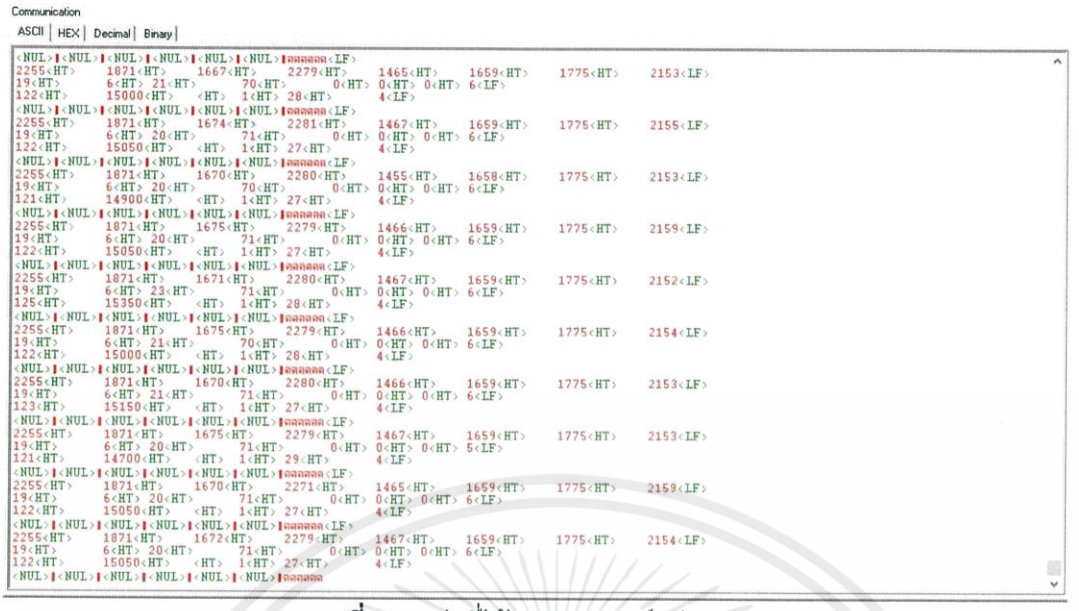


รูปที่ 4.6 ค่าที่ได้จากเซนเซอร์แต่ละตัว

แบบที่สองทำการอ่านค่าเซนเซอร์เมื่อให้เซนเซอร์บางตัวอยู่บนเส้น



รูปที่ 4.7 ทำการวางบอร์ดเซนเซอร์ให้โดนเส้น



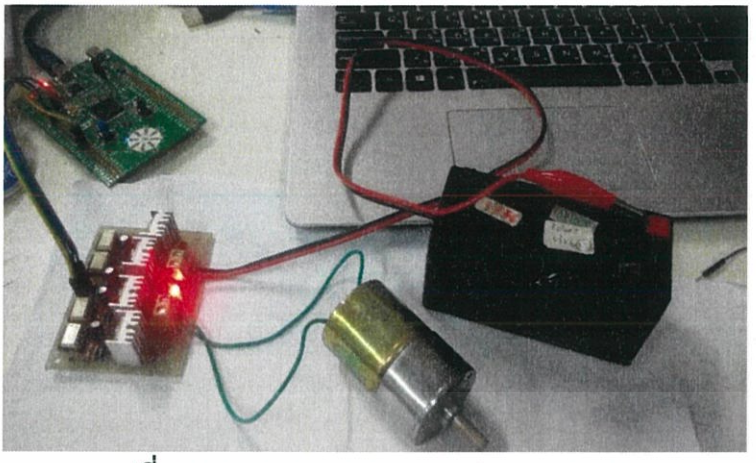
รูปที่ 4.8 ค่าที่ได้จากเซนเซอร์แต่ละตัว

เนื่องจากค่าของเซนเซอร์แต่ละตัวให้ค่ามาไม่เท่ากัน เราจึงได้ทำการปรับปรุงค่าของของเซนเซอร์ เพื่อให้ค่าของเซนเซอร์แต่ละตัวมีค่าใกล้เคียงกันในหน่อยของร้อยเปอร์เซน ตามสมการ

$$Sensor = \frac{Sensor\ value - Min}{Max - Min} \times 100$$

4.3 ทดสอบวงจร Mosfet motor driver

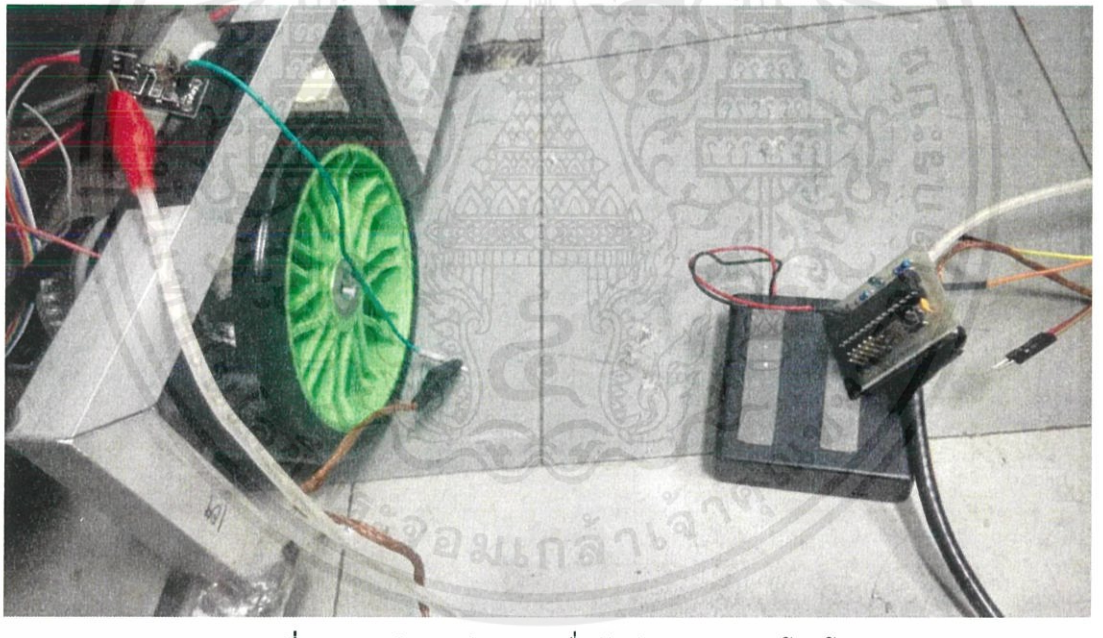
ทดสอบวงจร Mosfet motor driver โดยใช้ขับมอเตอร์ DC ผ่านบอร์ STM32F3Discovery ทำการจ่าย PWM ความคุมความเร็วของมอเตอร์ ใช้ความถี่ 1 kHz ทดสอบโดยจ่าย Duty cycle 100% และ 50% ปรากฏว่าความเร็วของมอเตอร์หมุนต่างกัน โดยที่ Duty cycle 100% มอเตอร์จะหมุนด้วยความเร็วเต็มที่ ส่วน Duty cycle 50% มอเตอร์จะหมุนด้วยความเร็มประมาณครึ่งนึง



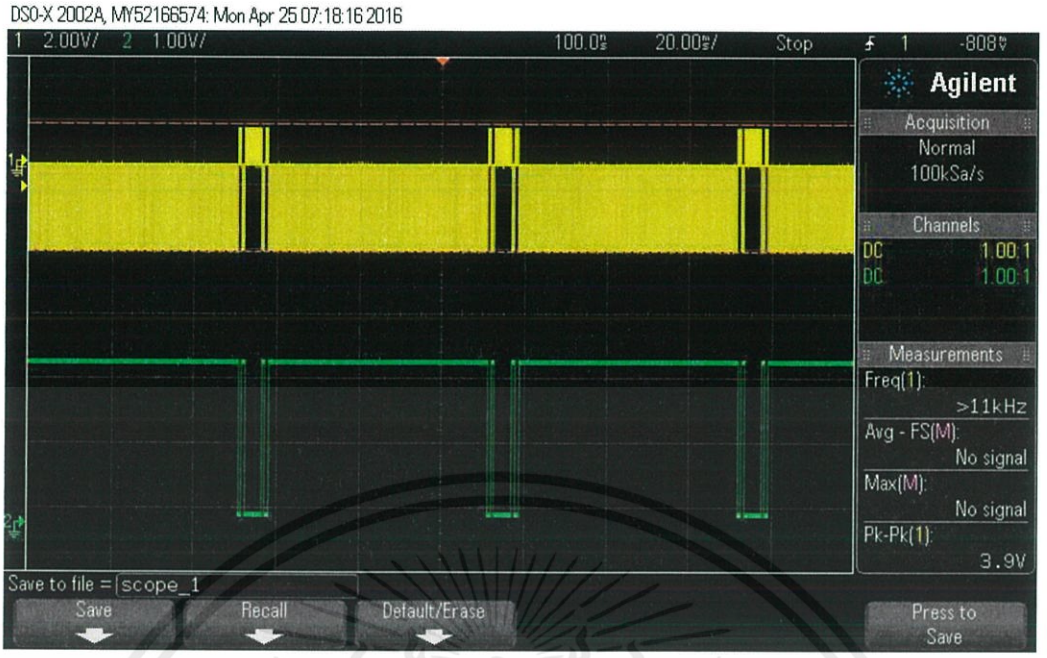
รูปที่ 4.9 การทดสอบวงจร Mosfet motor driver

4.4 อ่านค่าตัวรับ และตัวส่ง อินฟราเรด

ทำการทดสอบโดยใช้วงจรที่ทำการออกแบบไว้ แล้วใช้ออสซิลโลสโคปวัด และเปรียบเทียบค่าของตัวรับและตัวส่งว่าสามารถใช้งานได้หรือไม่



รูปที่ 4.10 ทำการต่อวงจรเพื่อวัดค่าจากออสซิลโลสโคป

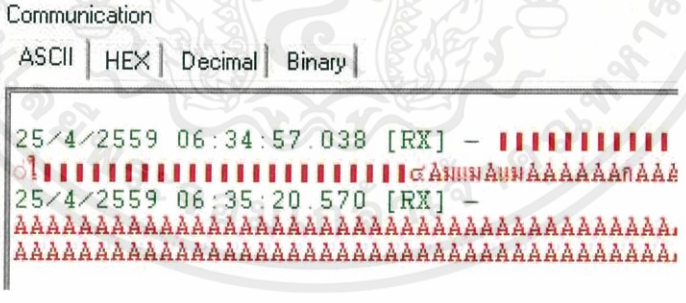


1

2

รูปที่ 4.11 ภาพจากออสซิลโลสโคป

ภาพที่ 1 เป็นภาพสัญญาณของตัวส่ง
ภาพที่ 2 เป็นภาพสัญญาณของตัวรับ
จะเห็นว่ากราฟของตัวรับและตัวส่งมีค่าใกล้เคียงกัน
จากนั้นทำการทดสอบอ่านค่าผ่านทาง USB to Serial และแสดงผลที่หน้าจอคอมพิวเตอร์ผ่านโปรแกรม Docklight

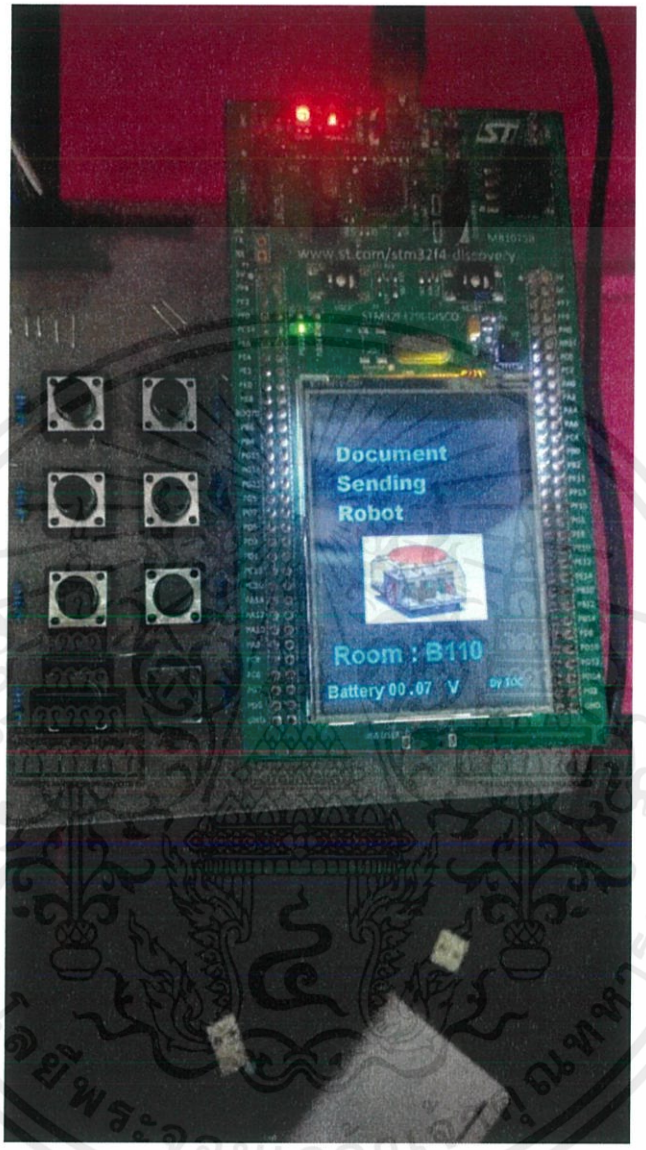


รูปที่ 4.12 ค่าที่อ่านจากตัวรับอินฟราเรด

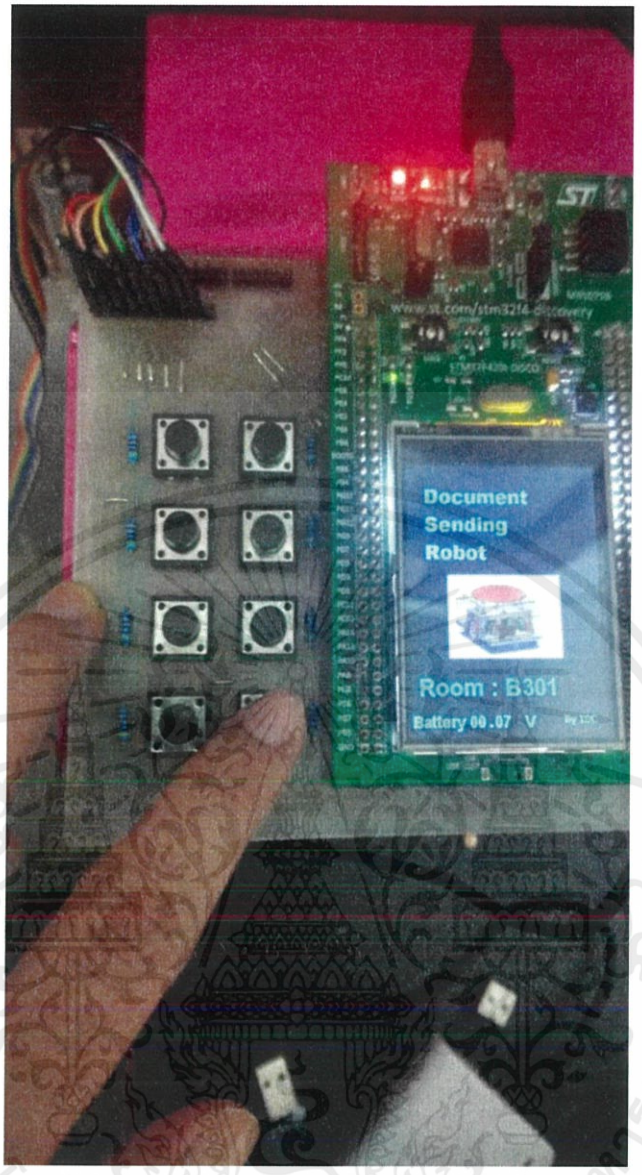
จะสังเกตุได้ว่า ค่าที่อ่านได้นั้นคือตัวอักษร A ซึ่งตรงกับที่โปรแกรมไว้ในตัวส่งสัญญาณอินฟราเรด

4.5 รับค่าจากสวิตซ์แล้วแสดงผลผ่านบอร์ด STM32F429

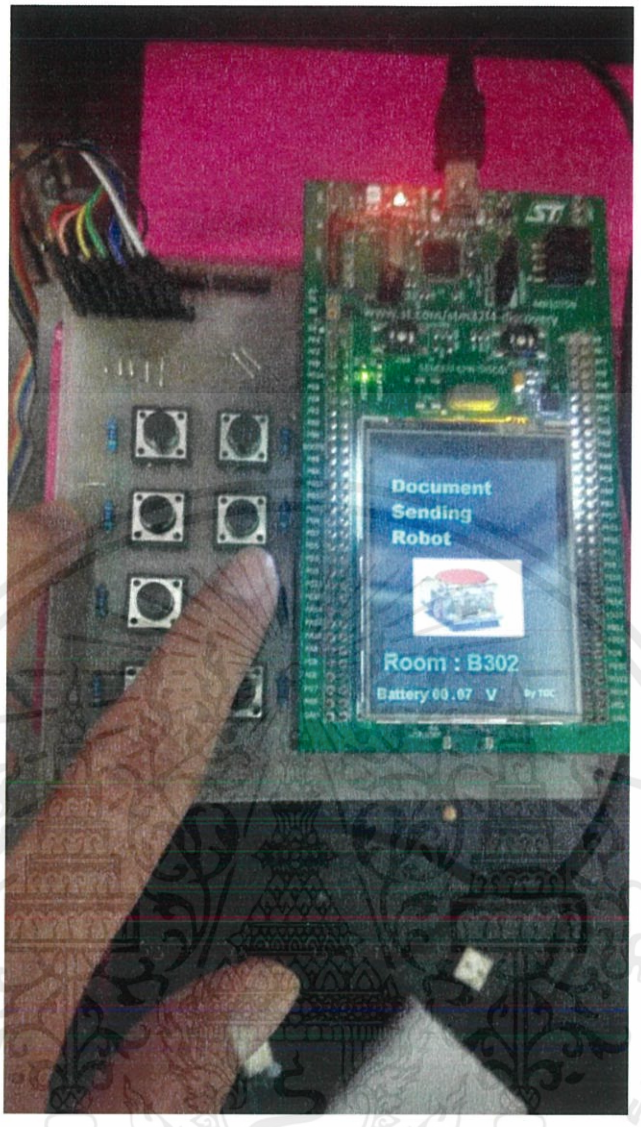
ทำการทดสอบโดยการกดสวิตซ์แล้วสังเกตุการเปลี่ยนแปลงที่จอของ STM32F429



รูปที่ 4.13 เมื่อยังไม่กดสวิตซ์ใดๆ จะสังเกตุเห็นว่าเมื่อยังไม่กดสวิตซ์ใดๆ จอจะแสดงผลเป็นตัวเลข 101 ตามที่โปรแกรมไว้



รูปที่ 4.14 แสดงผลเมื่อกดสวิตซ์ที่ 1 จะสังเกตุเห็นว่าเมื่อกดสวิตซ์ที่ 1 จอจะแสดงผลเป็นตัวเลข 301 ตามที่โปรแกรมไว้



รูปที่ 4.15 แสดงผลเมื่อกดสวิตซ์ที่ 2 จะสังเกตุเห็นว่าเมื่อกดสวิตซ์ที่ 2 จอจะแสดงผลเป็นตัวเลข 302 ตามที่โปรแกรมไว้

บทที่ 5
สรุปผลและข้อเสนอแนะ

ผลงานชิ้นนี้เป็นการศึกษาเกี่ยวกับทฤษฎีคอนโทรล และนำมาประยุกต์ใช้ในการทำหุ่นยนต์ ซึ่งเป็นหุ่นยนต์สำหรับส่งของตามบ้าน หรือตามโรงงาน โดยผู้จัดทำได้ใช้ บอร์ด STM32F3Discovery ซึ่งเป็นไมโครคอนโทรลเลอร์สถาปัตยกรรม ARM ซึ่งมีความเร็วในการประมวลผลสูงกว่าไมโครคอนโทรลเลอร์ 8 บิต ทั่วไปมาก และมีฟังก์ชั่นในการใช้งานมากกว่าในหลายๆ ด้าน และทางผู้จัดทำได้ใช้เซนเซอร์อินฟาเรต และ Encoder ในการควบคุมมอเตอร์โดยใช้ระบบควบคุม PID

ในการทดสอบวงจร ส่วนของระบบควบคุม ผู้จัดทำได้ทำการทดลอง โดยอ่านค่าของ Encoder และ เซนเซอร์ ต่างๆ และเขียนโปรแกรมควบคุมมอเตอร์จากค่าที่อ่านได้ และสามารถทำงานได้ตามบทที่ 4

ในส่วนการทดสอบวงจรส่วนของ Mosfet motor driver ผู้จัดทำได้ทำการทดสอบโดยการจ่ายอินพุตและ Pulse เพื่อควบคุมทิศทางและความเร็วของมอเตอร์ ตามการทดลองในบทที่ 4

5.1 ปัญหาที่และอุปสรรค

1. เนื่องจากการออกแบบตัวหุ่นยนต์ล่าช้า ทำให้การทำงานส่วนอื่นๆ ไม่สะดวกเท่าที่ควร
2. ค่าของ Encoder ตัวใหม่ให้ค่าต่างจากตัวเดิม
3. ซอฟแวร์ STM32CubeMX มีปัญหาในการ Generate บางครั้ง
4. คอมพิวเตอร์แสดงผลค่าของ Encoder ไม่ทัน เมื่อทำการอ่านด้วยความเร็ว
5. ผู้จัดทำยังใหม่กับ STM32 พอสมควร จึงทำให้การเขียนโปรแกรมล่าช้าในบางครั้ง

5.2 แนวทางการแก้ไขปัญหา

1. ทำการวางตารางออกแบบหุ่นยนต์ให้ดี
2. ปรับปรุงโปรแกรมให้สามารถอ่านค่าได้
3. ทำการเซคโค้ดโปรแกรมก่อนโปรแกรมทุกครั้ง
4. ทำการดีเลย์ตัวแปรเพื่อให้แสดงผลแทน
5. ทำการศึกษาข้อมูลให้มากขึ้น

5.3 สิ่งที่ได้รับจากการทำโครงงาน

1. ได้รับความรู้เกี่ยวกับ ระบบควบคุม PID
2. ได้รับความรู้เกี่ยวกับ วงจรควบคุมมอเตอร์
3. ได้ฝึกการแบ่งเวลาที่ดี
4. ได้ฝึกทักษะในการแก้ปัญหาวงจรอิเล็กทรอนิกส์
5. ได้ฝึกการเขียนโปรแกรมความคุม ไมโครคอนโทรลเลอร์ 32 บิต

เอกสารอ้างอิง

[1] **ระบบควบคุมแบบป้อมกลับและการควบคุมแบบ PID.** ค้นเมื่อ 10 มกราคม 2559,จาก http://bme- 4inone.blogspot.com/2010/07/pid.html

[2] **เอ็นโคดเดอร์ตรวจรู้ตำแหน่ง.** ค้นเมื่อ 10 มกราคม 2559,จาก http://dk.coe.psu.ac.th/assign/encoder/intro/Intro01.htm

[3] **H-Bridge.** ค้นเมื่อ 10 มกราคม 2559,จาก: https://en.wikipedia.org/wiki/H_bridge

[4] Lewis Loflin. **H-Bridge Motor Control with Power MISFETS** . ค้นเมื่อ 12 มกราคม 2559,จาก http://www.bristolwatch.com/ele/h_bridge.htm

[5] **Advancing Technology for Humanity.** ค้นเมื่อ 15 มกราคม 2559,จาก http://www.ieee.org/index.html

[6] **PID Tuning.** ค้นเมื่อ 22 มกราคม 2559,จาก https://thaicontrol.wordpress.com/2011/11/27/pid-tuning/

[7] **ARM KEIL.** ค้นเมื่อ 22 มกราคม 2559,จาก http://www.keil.com

ภาคผนวก ก.

Code ในส่วนของการทำงาน

ก.1 Code ในส่วนของการตั้งค่า ADC 1 เพื่อใช้ในการอ่านเซนเซอร์ด้านหน้า

```
void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig;

  /**Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC;
  hadc1.Init.Resolution = ADC_RESOLUTION12b;
  hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 8;
  hadc1.Init.DMAContinuousRequests = ENABLE;
  hadc1.Init.EOCSelection = EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.Overrun = OVR_DATA_OVERWRITTEN;
  HAL_ADC_Init(&hadc1);

  /**Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_2;
  sConfig.Rank = 1;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_3;
  sConfig.Rank = 2;
  HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure Regular Channel
```

```
  */
 sConfig.Channel = ADC_CHANNEL_4;
 sConfig.Rank = 3;
 HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure Regular Channel
  */
 sConfig.Channel = ADC_CHANNEL_5;
 sConfig.Rank = 4;
 HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure Regular Channel
  */
 sConfig.Channel = ADC_CHANNEL_6;
 sConfig.Rank = 5;
 HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure Regular Channel
  */
 sConfig.Channel = ADC_CHANNEL_7;
 sConfig.Rank = 6;
 HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure Regular Channel
  */
 sConfig.Channel = ADC_CHANNEL_8;
 sConfig.Rank = 7;
 HAL_ADC_ConfigChannel(&hadc1, &sConfig);

  /**Configure Regular Channel
  */
 sConfig.Channel = ADC_CHANNEL_9;
 sConfig.Rank = 8;
 HAL_ADC_ConfigChannel(&hadc1, &sConfig);
}
```

ก.2 Code ในส่วนของการตั้งค่า TIM3 เพื่อใช้ในการคุมความเร็วของมอเตอร์

```
/* TIM3 init function */
void MX_TIM3_Init(void)
{

        TIM_ClockConfigTypeDef sClockSourceConfig;
        TIM_MasterConfigTypeDef sMasterConfig;
        TIM_OC_InitTypeDef sConfigOC;

        htim3.Instance = TIM3;
        htim3.Init.Prescaler = 48;
        htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
        htim3.Init.Period = 1000;
        htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
        HAL_TIM_Base_Init(&htim3);

        sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
        HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig);

        HAL_TIM_PWM_Init(&htim3);

        sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
        sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
        HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig);
}
```

ก.3 Code ในส่วนของการตั้งค่า USART1 เพื่อใช้ในการแสดงผลต่างๆ

```
/* USART1 init function */
void MX_USART1_UART_Init(void)
{
        huart1.Instance = USART1;
        huart1.Init.BaudRate = 115200;
        huart1.Init.WordLength = UART_WORDLENGTH_8B;
        huart1.Init.StopBits = UART_STOPBITS_1;
        huart1.Init.Parity = UART_PARITY_NONE;
        huart1.Init.Mode = UART_MODE_TX_RX;
        huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        huart1.Init.OverSampling = UART_OVERSAMPLING_16;
        huart1.Init.OneBitSampling = UART_ONEBIT_SAMPLING_DISABLED ;
        huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
        HAL_UART_Init(&huart1);
}
```

ก.4 Code ในส่วนของการตั้งค่า USART2 เพื่อใช้ในการอ่านค่าจากตัวรับอินฟราเรด กำหนดให้ค่า BaudRate = 1200 เนื่องจากตัวรับอินฟราเรดตอบสนองที่ BaudRate 800 – 1200

```
/* USART2 init function */
void MX_USART2_UART_Init(void)
{
        huart2.Instance = USART2;
        huart2.Init.BaudRate = 1200;
        huart2.Init.WordLength = UART_WORDLENGTH_8B;
        huart2.Init.StopBits = UART_STOPBITS_1;
        huart2.Init.Parity = UART_PARITY_NONE;
        huart2.Init.Mode = UART_MODE_TX_RX;
        huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        huart2.Init.OverSampling = UART_OVERSAMPLING_16;
        huart2.Init.OneBitSampling = UART_ONEBIT_SAMPLING_DISABLED ;
        huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
        HAL_UART_Init(&huart2);
}
```

ก.5 Code ในส่วนของการใช้งาน DMA เพื่อใช้ในการเก็บค่าของเซนเซอร์เข้าไปที่แรมโดยตรง

```
void MX_DMA_Init(void)
{
        /* DMA controller clock enable */
        __DMA2_CLK_ENABLE();
        __DMA1_CLK_ENABLE();
}
```

ก.6 Code แสดงฟังชันก์ในการควบคุมความเร็วของมอเตอร์ซ้ายและขวา

```
//**********************Control PWM Left ************************************
void PWM_L(uint16_t en)
{
        TIM_OC_InitTypeDef sConfigOC;
        sConfigOC.OCMode = TIM_OCMODE_PWM1;
        sConfigOC.Pulse = en;
        sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
        sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
        HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3);
        HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_3);
}
//**********************Control PWM Right ************************************
void PWM_R(uint16_t en)
{
        TIM_OC_InitTypeDef sConfigOC;
        sConfigOC.OCMode = TIM_OCMODE_PWM1;
        sConfigOC.Pulse = en;
        sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
        sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
        HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_4);
        HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_4);
}
```

ก.7 Code แสดงฟังชันก์ที่ใช้ในการทดสอบมอเตอร์ซ้ายและขวา และความเร็วของมอเตอร์

```
//**********************Control Motor Left ************************************
void motor_L(uint8_t in1 ,uint8_t in2 , uint16_t en)
{
        /************ PC8 = in1 , PC9 = in2 ********************/
        if(in1 == 1 && in2 == 1)
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_8 , GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_9 , GPIO_PIN_SET);
        }
        else if(in1 == 0 && in2 == 1)
```

```
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_8 , GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_9 , GPIO_PIN_SET);
        }
        else if(in1 == 1 && in2 == 0)
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_8 , GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_9 , GPIO_PIN_RESET);
        }
        else
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_8 , GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_9 , GPIO_PIN_RESET);
        }
        PWM_L(en);
        HAL_Delay(1);
}
//**********************Control Motor Right ************************************
void motor_R(uint8_t in1 ,uint8_t in2 , uint16_t en)
{
 /********************** PC7 = in1 , PC6 = in2 ****************************/
        if(in1 == 1 && in2 == 1)
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_6 , GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_7 , GPIO_PIN_SET);
        }
        else if(in2 == 0 && in1 == 1)
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_6 , GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_7 , GPIO_PIN_SET);
        }
        else if(in2 == 1 && in1 == 0)
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_6 , GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_7 , GPIO_PIN_RESET);
        }
        else
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_6 , GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_7 , GPIO_PIN_RESET);
        }
        PWM_R(en);
```

```
        HAL_Delay(1);
}
```

ก.8 Code แสดงฟังชันก์การควบคุมมอเตอร์ด้วยระบบ PID

```
////////////////////// PID Line Follower Forward ////////////////////////////////
uint32_t sensor_sum = 0;
uint32_t sensor_weight_sum = 0;
uint8_t sensor_on_line = 0;
uint16_t sensor_pid = 0;
uint16_t weight[9] = {0,50,100,150,200,250,300,350};        //ให้น้ำหนักของเซนเซอร์เท่ากับ 0 - -350
int8_t error = 0;
uint32_t last_error = 0;
long integral = 0;
uint8_t target = 150;
uint8_t i=0;
uint8_t kp = 1/20;
uint8_t ki = 1/100;
uint8_t kd = 3/2;
int16_t PID = 0;
#define min1 1700
#define max1 2200
#define min2 1600
#define max2 2300
#define min3 1400
#define max3 2000
#define min4 1300
#define max4 1900
#define min5 1500
#define max5 2100
#define min6 1500
#define max6 2200
#define min7 1200
#define max7 2200
#define min8 2000
#define max8 2400
void PID_line_forward(uint16_t sml, uint16_t smr, uint8_t run_state)
```

```
{
        HAL_GPIO_WritePin(GPIOC , GPIO_PIN_8 , GPIO_PIN_SET);                ///
forward
        HAL_GPIO_WritePin(GPIOC , GPIO_PIN_9 , GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOC , GPIO_PIN_6 , GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOC , GPIO_PIN_7 , GPIO_PIN_SET);

        /**********************************      Sensor 1        ***************************/
        if(ss[0] > max1)
        {
                HAL_GPIO_WritePin(GPIOE,GPIO_PIN_9,GPIO_PIN_SET);
                ssp[0] = max1;
        }
        else if(ss[0] < min1)
        {
                HAL_GPIO_WritePin(GPIOE,GPIO_PIN_9,GPIO_PIN_RESET);
                ssp[0] = min1;
        }
        else
        {
                ssp[0] = ss[0];
        }
        ssp[0] = 100*(ssp[0]-min1)/(max1-min1);
        /**********************************      Sensor 2        ***************************/
        if(ss[1] > max2)
        {
                HAL_GPIO_WritePin(GPIOE,GPIO_PIN_10,GPIO_PIN_SET);
                ssp[1] = max2;
        }
        else if(ss[1] < min2)
        {
                HAL_GPIO_WritePin(GPIOE,GPIO_PIN_10,GPIO_PIN_RESET);
                ssp[1] = min2;
        }
        else
        {
                ssp[1] = ss[1];
        }
        ssp[1] = 100*(ssp[1]-min2)/(max2-min2);
        /**********************************      Sensor 3        ***************************/
        if(ss[2] > max3)
```

```
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_11,GPIO_PIN_SET);
        ssp[2] = max3;
}
else if(ss[2] < min3)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_11,GPIO_PIN_RESET);
        ssp[2] = min3;
}
else
{
        ssp[2] = ss[2];
}
ssp[2] = 100*(ssp[2]-min3)/(max3-min3);
/**********************************      Sensor 4      ***************************/
if(ss[3] > max4)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_12,GPIO_PIN_SET);
        ssp[3] = max4;
}
else if(ss[3] < min4)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_12,GPIO_PIN_RESET);
        ssp[3] = min4;
}
else
{
        ssp[3] = ss[3];
}
ssp[3] = 100*(ssp[3]-min4)/(max4-min4);
/**********************************      Sensor 5      ***************************/
if(ss[4] > max5)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_13,GPIO_PIN_SET);
        ssp[4] = max5;
}
else if(ss[4] < min5)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_13,GPIO_PIN_RESET);
        ssp[4] = min5;
}
```

```
else
{
        ssp[4] = ss[4];
}
ssp[4] = 100*(ssp[4]-min5)/(max5-min5);
/*********************************      Sensor 6      ***************************/
if(ss[5] > max6)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_14,GPIO_PIN_SET);
        ssp[5] = max6;
}
else if(ss[5] < min6)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_14,GPIO_PIN_RESET);
        ssp[5] = min6;
}
else
{
        ssp[5] = ss[5];
}
ssp[5] = 100*(ssp[5]-min6)/(max6-min6);
/*********************************      Sensor 7      ***************************/
if(ss[6] > max7)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_15,GPIO_PIN_SET);
        ssp[6] = max7;
}
else if(ssc[6] < min7)
{
        HAL_GPIO_WritePin(GPIOE,GPIO_PIN_15,GPIO_PIN_RESET);
        ssp[6] = min7;
}
else
{
        ssp[6] = ss[6];
}
ssp[6] = 100*(ssp[6]-min7)/(max7-min7);
/*********************************      Sensor 8      ***************************/
if(ss[7] > max8)
{
        ssp[7] = max8;
```

```
        }
        else if(ss[7] < min8)
        {
                ssp[7] = min8;
        }
        else
        {
                ssp[7] = ss[7];
        }
        ssp[7] = 100*(ssp[7]-min8)/(max8-min8);

        sensor_sum = 0;
        sensor_weight_sum = 0;

        for(i=0;i<8;i++)
        {
                sensor_sum += ssp[i];
                sensor_weight_sum += ssp[i]*weight[i];
        }

        if(sensor_sum > 20  && sensor_sum < 600)
        {
                HAL_GPIO_WritePin(GPIOE,GPIO_PIN_8,GPIO_PIN_SET);
                sensor_on_line = 1;
        }

        else
        {
                sensor_on_line = 0;
                HAL_GPIO_WritePin(GPIOE,GPIO_PIN_8,GPIO_PIN_RESET);
        }

        if(sensor_on_line == 1)
        sensor_pid = sensor_weight_sum/sensor_sum;          //ทำการหาค่าเฉลี่ยของเซนเซอร์
        error = target - sensor_pid;                        //ทำการหาค่าความผิดะพลาด
ของเซนเซอร์

        uint32_t derivative = error - last_error;           //กำหนดตัวแปรเพื่อใช้ในเทอม D
        integral += error;                                  //คำนวนค่าตัวแปรเพื่อใช้ใน
เทอม I
        error = last_error;

        PID = error*kp + integral*ki + derivative*kd;       // PID control
```

```
        if(run_state == 1)
        {
                if(sensor_on_line == 1)
                {
                        if(PID < 0 )

                        {
                                PWM_R(smr-((-PID)+40));
                                PWM_L(sml+150);
                        }
                        else if(PID > 0)
                        {
                                PWM_R(smr+100);
                                PWM_L(sml-(PID+100));
                        }
                        else
                        {
                                PWM_R(smr/2);
                                PWM_L(sml/2);
                        }
                }
                else
                {
                        PWM_L(smr);
                        PWM_R(sml);
                }
        }
        else    //ทำการหยุดโดยการ short break เพื่อให้มอเตอร์หยุดหมุนในทันที
        {
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_8 , GPIO_PIN_RESET);                 //
break
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_9 , GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_6 , GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOC , GPIO_PIN_7 , GPIO_PIN_RESET);
                PWM_R(0);
                PWM_L(0);
        }
}
```

ก.9 Code แสดงในส่วนของ Task1 ทำการเปิดใช้งาน DMA และแสดงค่าของเซนเซอร์ผ่านทาง USART1

```
void StartDefaultTask(void const * argument)
{

  /* USER CODE BEGIN 5 */
////////////////////////////////  Read Sensor  ////////////////////////////////////
        uint8_t read_sensor[200];
        uint8_t read_sensor_b[200];
        HAL_ADC_Start_DMA(&hadc1,(uint32_t*)ss,8);          //เปิดใช้ DMA สำหรับเซนเซอร์
ด้านหน้า
        HAL_ADC_Start_DMA(&hadc4,(uint32_t*)ss_b,8);        //เปิดใช้ DMA สำหรับเซนเซอร์
ด้านหลัง
  /* Infinite loop */
  for(;;)
  {
        sprintf((char*)read_sensor,"\n%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n%d\t%d\t%d\t
%d\t%d\t%d\t%d\t%d\n%d\t%d\t\t%d\t%d\t%d\n"

                        ,ss[0],ss[1],ss[2],ss[3],ss[4],ss[5],ss[6],ss[7]

        ,ssp[0],ssp[1],ssp[2],ssp[3],ssp[4],ssp[5],ssp[6],ssp[7]

        ,sensor_sum,sensor_weight_sum,sensor_on_line,PID,mstate);
         HAL_UART_Transmit(&huart1,read_sensor_b,90,100);
   osDelay(100);
        }
  /* USER CODE END 5 */
}
```

ก.10 Code แสดงในส่วนของ Task2 ทำการรับค่าจากสวิตซ์ เพื่อเป็นการเลือกห้อง

```
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
        osDelay(3000);
  for(;;)
  {
//*************************Begin Sw Input*********************************/
                if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_0)==0)
        // sw1
                {
                        room_selec = 'A';
                }
                else if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_1)==0)          // sw2
```

```
        {
                room_selec = 'B';
        }
        else if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_2)==0)          // sw3
        {
                room_selec = 'C';
        }
        else if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_3)==0)          // sw4
        {
                room_selec = 'D';
        }
        else if(HAL_GPIO_ReadPin(GPIOD,GPIO_PIN_4)==0)          // sw5
        {
                room_selec = 'E';
        }
        else if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_10)==0)         // sw6
        {
                room_selec = 'F';
        }
        else if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_11)==0)         // sw7
        {
                room_selec = 'G';
        }
        else if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_12)==0)         // sw8
        {
                room_selec = 'H';
        }
**************************End Sw Input*********************************/

**************************Begin State*********************************/
        if(room_selec == 'A')
        {
                if(room_selec != room)
                {
                        mstate = 4;
                }
                else
                {
                        mstate = 0;
                        room_selec = 'Z';
                }
        }
        else if(room_selec == 'B')
        {
                if(room_selec != room)
                {
                        mstate = 1;
```

```
                }
                else
                {
                        mstate = 0;
                        room_selec = 'Z';
                }
        }
        else if(room_selec == 'C')
        {
                if(room_selec != room)
                {
                        mstate = 1;
                }
                else
                {
                        mstate = 0;
                        room_selec = 'Z';
                }
        }
        else if(room_selec == 'D')
        {
                if(room_selec != room)
                {
                        mstate = 1;
                }
                else
                {
                        mstate = 0;
                        room_selec = 'Z';
                }
        }
        else if(room_selec == 'E')
        {
                if(room_selec != room)
                {
                        mstate = 1;
                }
                else
                {
                        mstate = 0;
                        room_selec = 'Z';
                }
        }
else if(room_selec == 'F')
        {
                if(room_selec != room)
                {
                        mstate = 1;
```

```
                }
                else
                {
                        mstate = 0;
                        room_selec = 'Z';
                }
        }
        else if(room_selec == 'G')
        {
                if(room_selec != room)
                {
                        mstate = 1;
                }
                else
                {
                        mstate = 0;
                        room_selec = 'Z';
                }
        }
        else if(room_selec == 'Z')
        {
                mstate = 3;
        }
        else if(room_selec == 'K')
        {
                mstate = 4;
        }
**************************End State**********************************/

        if(mstate == 1)
        {
                //HAL_GPIO_WritePin(GPIOE , GPIO_PIN_13 , GPIO_PIN_SET);
                //PID_line_backward(800,800,1);
                PID_line_forward(500,500,1);
        }
        else if(mstate == 4)
        {
                PID_line_backward(500,500,1);
        }
        else if(mstate == 0)
        {
                //HAL_GPIO_WritePin(GPIOE , GPIO_PIN_15 , GPIO_PIN_RESET);
                //PID_line_backward(0,0,0);
                PID_line_forward(0,0,0);
```

```
                PID_line_backward(0,0,0);
            }
            else if(mstate == 2)
            {
                    //PID_line(0,0);
            }

            //PID_line(500,500);
    osDelay(10);
  }
  /* USER CODE END StartTask02 */
}
```

ก.11 Code แสดงในส่วนของ Task3 ทำงานในส่วนของการรับค่าจากอินฟราเรด

```
void StartTask03(void const * argument)
{
  /* USER CODE BEGIN StartTask03 */
  /* Infinite loop */
  for(;;)
  {
            HAL_UART_Receive(&huart2,&room,1,10);
            HAL_UART_Transmit(&huart2,&room,1,10);
    osDelay(1);
  }
  /* USER CODE END StartTask03 */
}
```

ก.12 Code ในส่วนของตัวส่งสัญญาณอินฟราเรด

```
#include <avr/io.h>
#include <util/delay.h>
#define FOSC 8000000 // Clock Speed
#define BAUD 1200
#define MYUBRR FOSC/16/BAUD-1

void USART_Init( unsigned int ubrr)
{
        /*Set baud rate */
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;
        UCSR0B = (1<<RXEN0)|(1<<TXEN0);
        /* Set frame format: 8data, 2stop bit */
        UCSR0C = (1<<USBS0)|(3<<UCSZ00);
}

void USART_Transmit( unsigned char data )
```

```
{
        /* Wait for empty transmit buffer */
        while ( !( UCSR0A & (1<<UDRE0)) )
        ;
        /* Put data into buffer, sends the data */
        UDR0 = data;
}

int main(void)
{
        uint8_t i = 0;
        uint8_t j = 0;
        DDRB = 0xff;
        DDRD = 0xff;

        TCCR1A = 0b10000010;
        TCCR1B = 0b00011001;
        ICR1 = 211;
        OCR1A = ICR1/2;
        USART_Init(MYUBRR);
  while(1)
  {
            USART_Transmit('C');
            PORTB = 0b00000010;
            PORTD = 0b00000010;
            _delay_ms(500);
  }
}
```