ระบบควบคุมลูกบอลบนแผ่นระนาบ

BALL ON PLATE CONTROL SYSTEM

ภาสกร ไทรศักดิ์สิทธิ์

ภูบดี ไชยกาล

อภิสร วจนาพร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมแมคคาทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2557

ระบบควบคุมลูกบอลบนแผ่นระนาบ

BALL ON PLATE CONTROL SYSTEM

ภาสกร ไทรศักดิ์สิทธิ์

ภูบดี ไชยกาล

อภิสร วจนาพร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมแมคคาทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2557

BALL ON PLATE CONTROL SYSTEM

Pasakon Saisaksit

Phubodee Chaiyakan

Apisorn Wajanaporn

THIS THESIS IS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

BECHELOR OF ENGINEETING IN MECHATRONICS ENGINEERING

FACULTY OF ENGINEETING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2014

ปริญญานิพนธ์ปีการศึกษา 2557

ภาควิชาวิศวกรรมการวัดและควบคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบควบคุมลูกบอลบนแผ่นระนาบ

BALL ON PLATE CONTROL SYSTEM

ผู้จัดทำ นายภาสกร ไทรศักดิ์สิทธิ์ 54010998

นายภูบดี ไชยกาล 54011003

นายอภิสร วจนาพร 54011498

.....อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์สุมิตร พนาอุดมทรัพย์)

.....อาจารย์ที่ปรึกษา

(รองศาสตราจารย์ ดร.เกียรติศักดิ์ คมวัชระ)

ระบบควบคุมตำแหน่งลูกบอลบนระนาบ

โดย

นายภาสกร ไทรศักดิ์สิทธิ์ 54010998

นายภูบดี ไชยกาล 54011003

นายอภิสร วจนาพร 54011498

อาจารย์ที่ปรึกษา

ผู้ช่วยศาสตราจารย์สุมิตร พนาอุดมทรัพย์

รองศาสตราจารย์ดร. เกียรติศักดิ์ คมวัชระ

ปีการศึกษา 2557

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้จัดทำเพื่อศึกษาทฤษฎีในการออกแบบการควบคุมตำแหน่งของลูกบอลบนระนาบ โดยใช้กล้องดิจิตอลในการรับสัญญาณภาพ เพื่อนำมาประมวลผลในคอมพิวเตอร์ผ่านโปรแกรมด้านการประมวลผลภาพ ทำให้รักษาตำแหน่งของลูกบอลบนระนาบได้ จุดหมายของโครงงานนี้คือการบังคับให้ลูกบอลมีตำแหน่งที่ต้องการ โดยมีเซอร์โวมอเตอร์เป็นตัวควบคุมระนาบ

ขั้นตอนดำเนินการ ได้แก่ ออกแบบโครงสร้างกลไกทางแมคคานิกส์ที่จำเป็นสำหรับการควบคุมระนาบโดยใช้เซอร์โวมอเตอร์ จากนั้นศึกษาโปรแกรมรับค่าภาพด้วยกล้องดิจิตอล และการใช้โปรแกรมในการประมวลผลภาพ และนำไปควบคุมตำแหน่งของลูกบอลบนระนาบ ศึกษาทฤษฎีเกี่ยวกับระบบการเคลื่อนที่ของลูกบอลเพื่อนำไปใช้ในการควบคุมตำแหน่งของลูกบอลบนระนาบให้แม่นยำ และวิเคราะห์ค่าความผิดพลาดต่างๆ ที่มีผลทำให้สมการคลาดเคลื่อนจากตำแหน่งที่ควรจะเป็น พร้อมทั้งปรับแต่งสมการจนทำให้ค่าความผิดพลาดอยู่ในระบบที่ยอมรับได้

BALL ON PLATE CONTROL SYSTEM

By

Mr.Pasakorn Saisaksit 54010998

Mr.Phubodee Chaiyakan 54011003

Mr.Apisorn Wajanaporn 54011498

Advisors

Asst.Prof. Sumit Panaaudomsap

Assoc.Prof.Dr. Kiatisuk Komwatchara

Academic Year 2014

ABSTRACT

This thesis is produced for researching about theory related to controlling ball position on the plate. By using digital camera for receiving picture signal then, processing that signal with image processing program in computer, using this method, the position of the ball on the plate can be controlled. The objective of this project is to control position of the ball on the plate with servo motors controlling the angle of the surface, plate.

There are severalprocedures must be completed. First, design mechanical structure which is necessary for controlling the plate using servo motors. Second, find and study program that can receive the picture signal from digital camera and, image processing program which use to process the image by transforming picture signal into certain parameters to control position of the ball on plate. Finally, study in-depth mechanismequation required to improve positioning to be more accurately and calculate difference between equation from theory and real use equation then, mathematically analyze and formulize new equation until the result will be in range with acceptable margin of error.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ฉบับนี้ สามารถสำเร็จลุล่วงไปได้ด้วยดี โดยเฉพาะอาจารย์ที่ปรึกษา ผศ. สุมิตร พนาอุดมทรัพย์ และรศ.ดร. เกียรติศักดิ์ คมวัชระ ที่ได้ให้คำแนะนำอันเป็นประโยชน์ที่การทดลองค้นคว้านี้ พร้อมยังจัดสถานที่การค้นคว้า และอุปกรณ์ในการทำการทดลองบางส่วนที่จำเป็นต่อโครงงาน ผู้จัดทำรู้สึกยินได้อย่างยิ่งและขอกราบขอบคุณอย่างสูง

ขอขอบคุณชมรมโรบอทที่เอื้อเฟื้อเครื่องมือที่ขาดแคลน และสถานที่ทำการทดลอง

ขอบคุณรุ่นพี่ที่ให้คำปรึกษา และเพื่อนๆ ที่คอยให้คำติชม ให้กำลังใจ และติดตามผลงานเป็นอย่างดีรวมทั้งคอยไตร่ถามความเป็นไปอยู่สม่ำเสมอ

และขอขอบคุณบิดา มารดา ครอบครัว และญาติ ที่คอยให้กำลังใจ มอบแรงกระตุ้นในการทำงานตลอดจนให้คำปรึกษาในยามต้องการ และสนับสนุนงบประมาณ อุปกรณ์ที่ขาดเหลือยามจำเป็น ผู้จัดทำรู้สึกซาบซึ้งอย่างมากที่ได้รับการช่วยเหลือจนสามารถสำเร็จโครงงานได้

ผู้จัดทำ

นายภาสกร ไทรศักดิ์สิทธิ์

นายภูบดี ไชยกาล

นายอภิสร วจนาพร

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญตาราง

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมา และความสำคัญของปัญหา

การประเมินคุณภาพของเครื่องจักรนั้นมักจะพิจารณาจากหลายๆ องค์ประกอบ โดยหลักๆ คือ พิจารณาที่ประสิทธิภาพ และความแม่นยำของชิ้นงานที่ได้ ในยุคก่อนที่เทคโนโลยีดิจิตอลยังไม่ได้รับการพัฒนาเช่นปัจจุบัน เครื่องจักรที่มีคุณภาพดีมักถูกสร้างขึ้นด้วยกลที่ซับซ้อน และต้องมีผู้ใช้งานที่มีความเชี่ยวชาญสูง จึงจะสามารถสร้างชิ้นงานที่มีคุณภาพดีได้ ภายหลังเมื่อเข้าสู่ยุคดิจิตอล คอมพิวเตอร์ถูกพัฒนาจนมีประสิทธิภาพมากขึ้น และเข้ามามีส่วนช่วยในการควบคุมเครื่องจักร ทำให้เครื่องจักรที่มีโครงสร้างไม่ซับซ้อนสามารถทำงานที่ซับซ้อนได้ ชิ้นงานที่มีคุณภาพสูงจึงถูกผลิตได้ในจำนวนที่มากขึ้นแต่ใช้เวลาน้อยลง และสามารถได้โดยอัตโนมัติโดยไม่จำเป็นต้องใช้ช่างที่ชำนาญเฉพาะทางก็ได้ ดังนั้นจะเห็นได้ว่าระบบควบคุมมีความสำคัญในการพัฒนาเครื่องจักรต่างๆ เป็นอย่างยิ่ง การนำความรู้ด้านทฤษฎีการควบคุมมาประยุกต์ใช้ให้เชี่ยวชาญจึงเรื่องที่น่าสนใจอย่างยิ่ง

ปริญญานิพนธ์จัดทำขึ้นโดยการสร้างโมเดลที่มีโครงสร้างง่ายๆ ขึ้นมาโมเดลหนึ่ง เพื่อแสดงการประยุกต์ใช้ระบบควบคุมให้เห็นได้ชัดเจน รวมถึงยังมีการใช้เทคโนโลยี Image Processing มาใช้ในการตรวจวัดตำแหน่งของวัตถุเป้าหมาย โดยเหตุผลที่ใช้เทคโนโลยี Image Processing เนื่องจากมีความสนใจในเทคโนโลยีนี้เป็นพิเศษ เพราะสามารถสร้างอุปกรณ์การตรวจวัดที่มีความละเอียดสูงในโดยไม่ต้องใช้อุปกรณ์หรือวงจรอิเล็กทรอนิกส์ที่ซับซ้อน และสามารถประยุกต์ใช้งานได้หลากหลาย รวมทั้งกล้องที่มีคุณภาพสูงในปัจจุบันมีราคาที่ถูก จึงทำให้สามารถพัฒนาได้สะดวก และหลากหลาย

คาดหวังว่าจะสามารถนำความรู้ด้านระบบควบคุมมาประยุกต์ร่วมกับเทคโนโลยี Image Processing ได้อย่างมีประสิทธิภาพ ซึ่งจะทำให้เกิดความเชี่ยวชาญในการประยุกต์ใช้ทฤษฎีทางระบบควบคุม และนำไปสู่การพัฒนาอุปกรณ์ เครื่องจักร รวมถึงระบบอื่นๆ ต่อไปในภายภาคหน้า

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาทฤษฎีการควบคุมต่างๆ
2. เพื่อศึกษาเทคโนโลยี Image Processing
3. เพื่อประยุกต์ใช้ทฤษฎีการควบคุม

4. เพื่อประยุกต์ใช้เทคโนโลยี Image Processing ร่วมกับระบบควบคุมที่ออกแบบ

1.3 ขอบเขตของโครงงาน

1. สร้างระนาบที่สามารถปรับเอียงตัวเองเพื่อควบคุมตำแหน่งของลูกบอลที่อยู่บนระนาบ
2. ใช้ Image Processing ในการตรวจจับตำแหน่งของลูกบอลได้อย่างแม่นยำ
3. สามารถควบคุมลูกบอลให้เคลื่อนที่ตามเส้นทางที่ต้องการได้

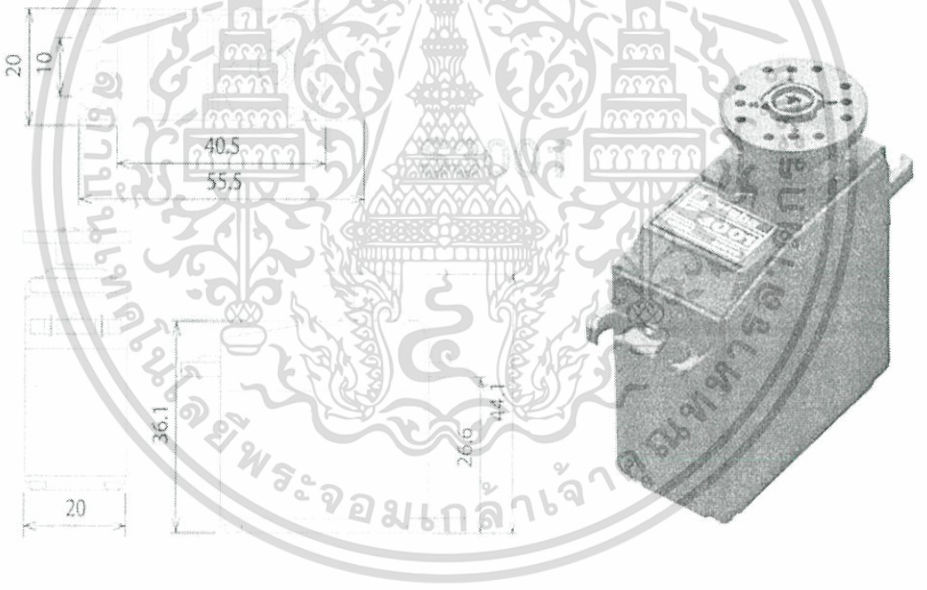
1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถนำความรู้ด้านทฤษฎีการควบคุมไปใช้ประยุกต์กับอุปกรณ์หรือระบบอื่นๆ ได้
2. พัฒนาทักษะด้านการประยุกต์ใช้ทฤษฎีระบบควบคุม และ Image Processing
3. สร้างอุปกรณ์ตัวอย่างที่แสดงการประยุกต์ใช้ความรู้ทฤษฎีทางระบบควบคุม และ Image Processing ที่สามารถเห็นภาพได้ชัดเจน และเข้าใจได้ง่าย

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

2.1 เซอร์โวมอเตอร์ (Servo Moter)

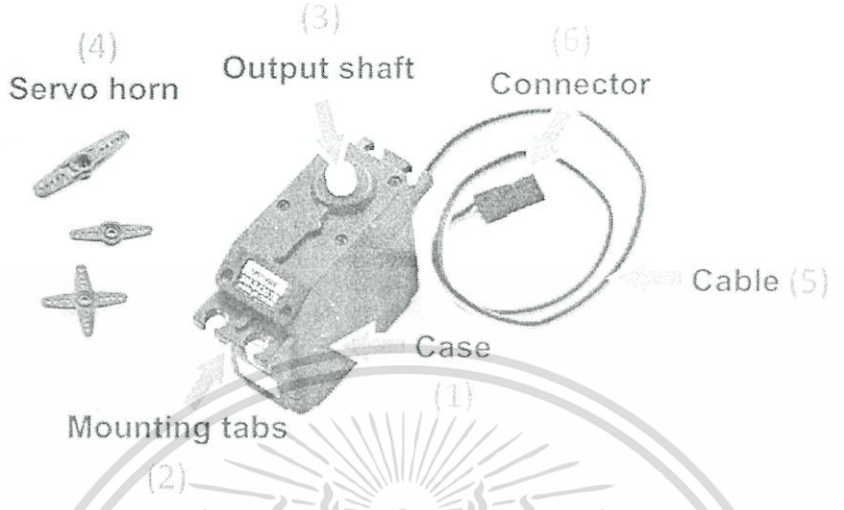
เซอร์โวมอเตอร์คือ มอเตอร์ไฟตรงขนาดเล็กที่ถูกประกบเข้ากับส่วนประกอบต่างๆ ได้แก่ ชุดเกียร์ทด ชุดวงจรควบคุมตำแหน่งการหมุนไว้ในโมดูลเดียวกัน โดยมีสายต่อใช้งาน 3 เส้น คือ V+ GND และ Control Line ซึ่งเป็นสายควบคุมที่ทำให้มอเตอร์หมุนซ้ายขวาโดยใช้สายสัญญาณ PWM เป็นตัวควบคุม ส่วนแรงเคลื่อน V+ ที่ป้อนให้ขับเซอร์โวมอเตอร์อยู่ที่ประมาณ 4 โวล์ ถึง 6 โวลต์ เซอร์โวมอเตอร์มีข้อดีคือ มีขนาดเล็ก น้ำหนักเบา แต่ให้แรงบิดสูง และกินพลังงานน้อย ใช้ระดับสัญญาณควบคุมแบบ TTL Level ต่อโดยตรงกับไมโครคอนโทรลเลอร์ได้เลยโดยไม่ต้องมีวงจรขับ



รูปที่ 2.1 เซอร์โวมอเตอร์

เซอร์โวมอเตอร์แบบนี้จะมีวงจรควบคุมอยู่ในตัวการควบคุมตำแหน่งการหมุนโดยทั่วไปจะมีช่วงประมาณ 180 องศา ถึง 210 องศา

2.1.1 ส่วนประกอบภายนอกของเซอร์โวมอเตอร์



รูปที่ 2.2 ส่วนประกอบภายนอกของเซอร์โวมอเตอร์

1. Case ตัวถัง หรือกรอบของตัวเซอร์โวมอเตอร์

2. Mounting Tab ส่วนจับยึดตัวเซอร์โวมอเตอร์กับชิ้นงาน

3. Output Shaft เพลาส่งกำลัง

4. Servo Horns ส่วนเชื่อมต่อกับ Output Shaft เพื่อสร้างกลไกล

5. Cable สายเชื่อมต่อเพื่อจ่ายไฟฟ้า และควบคุมเซอร์โวมอเตอร์จะประกอบด้วยสายไฟ 3 เส้น และ ในเซอร์โวมอเตอร์จะมีสีของสายแตกต่างกันไปดังนี้

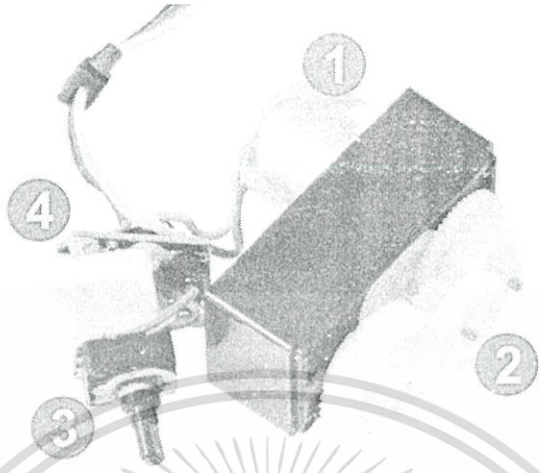
- สายสีแดง คือ ไฟเลี้ยง (4.8 - 6V)

- สายสีดำ หรือน้ำตาล คือ กราวด์

- สายสีเหลือง (ส้ม ขาว หรือฟ้า) คือ สายส่งสัญญาณพัลซ์ควบคุม (3-5 โวลต์)

6. Connector จุดเชื่อมต่อสายไฟ

2.1.2 ส่วนประกอบภายในของเซอร์โวมอเตอร์



รูปที่ 2.3 ส่วนประกอบภายในของเซอร์โวมอเตอร์

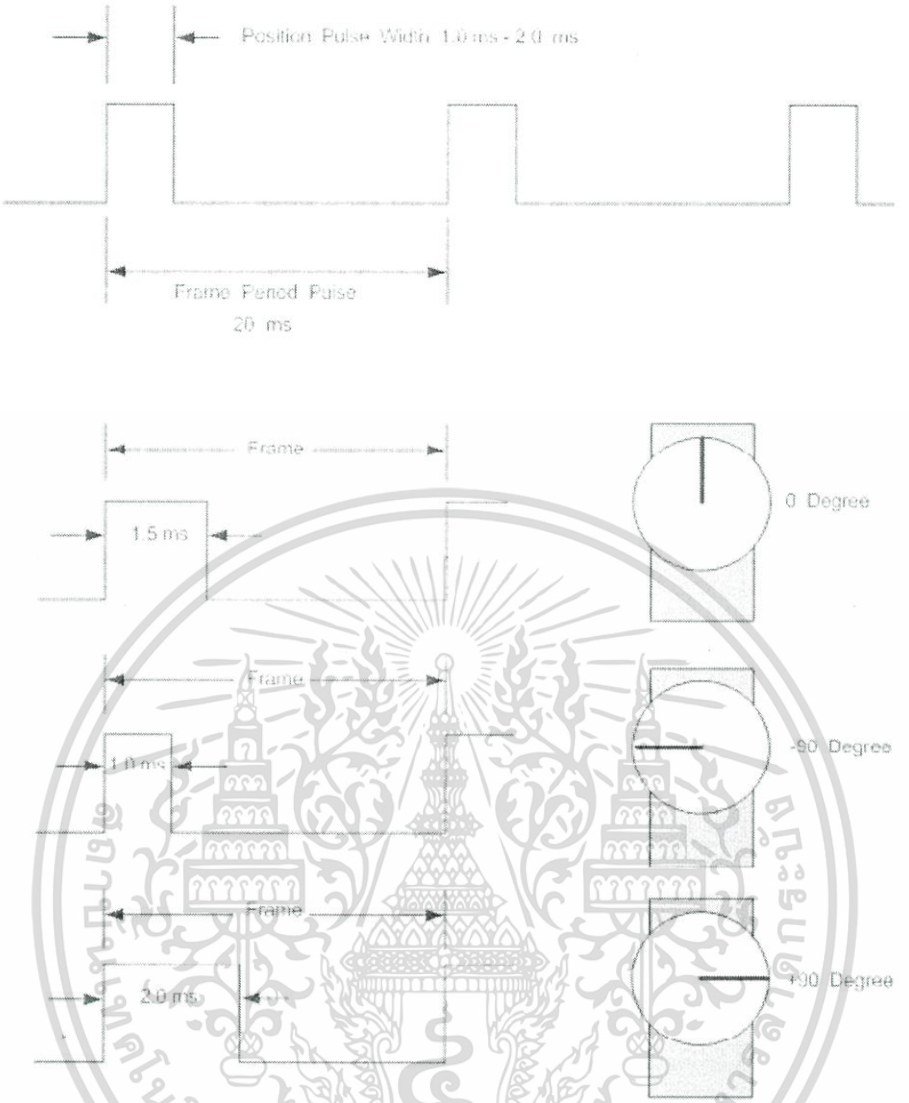
1. Motor เป็นส่วนของตัวมอเตอร์
2. Gear Train หรือ Gearbox เป็นชุดเกียร์ทดแรง
3. Position Sensor เป็นเซนเซอร์ตรวจจับตำแหน่งเพื่อหาค่าองศาในการหมุน
4. Electronic Control System เป็นส่วนที่ควบคุม และประมวลผล

2.1.3 หลักการทำงานของเซอร์โวมอเตอร์

การควบคุมตำแหน่งการหมุนของเซอร์โวมอเตอร์ ทำได้โดยการป้อนสัญญาณความกว้างพัลส์เข้าที่ขา Control Line ตำแหน่ง และทิศทางการหมุนของแกนเอาต์พุตจะขึ้นอยู่กับความกว้างของพัลส์สัญญาณควบคุมจะประกอบด้วย

- Frame Period Pulse เป็นสัญญาณพัลส์ต่อเนื่องโดยจะเริ่มต้นห่างกันทุกๆ 20 ms ตลอดเวลาเพื่อรักษาสภาพตำแหน่งการหมุนเอาไว้

- Position Pulse Width เป็นค่าความกว้างของยอดพัลส์ของ Frame Period Pulse ใช้เป็นค่าควบคุมตำแหน่ง และทิศทางการหมุน โดยจะมีค่าอยู่ระหว่าง 1.0 ms - 2.0 ms โดยจะมีจุดอ้างอิง 3 จุดที่สามารถควบคุม และรักษาตำแหน่งตั้งแต่ 0 – 180 องศา ตามรูปที่ 2.4



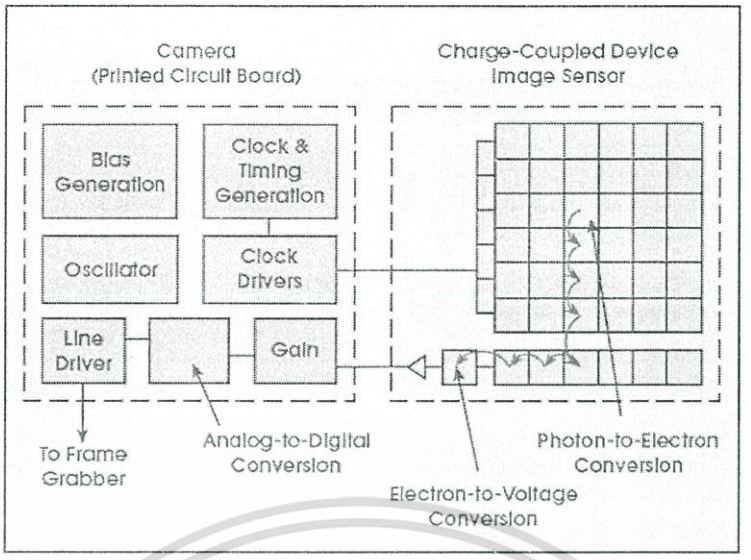
รูปที่ 2.4 Timing Diagram ของ Control Pulse

2.2 กล้องดิจิตอล

2.2.1 Camera Sensor

2.2.1.1 CCD

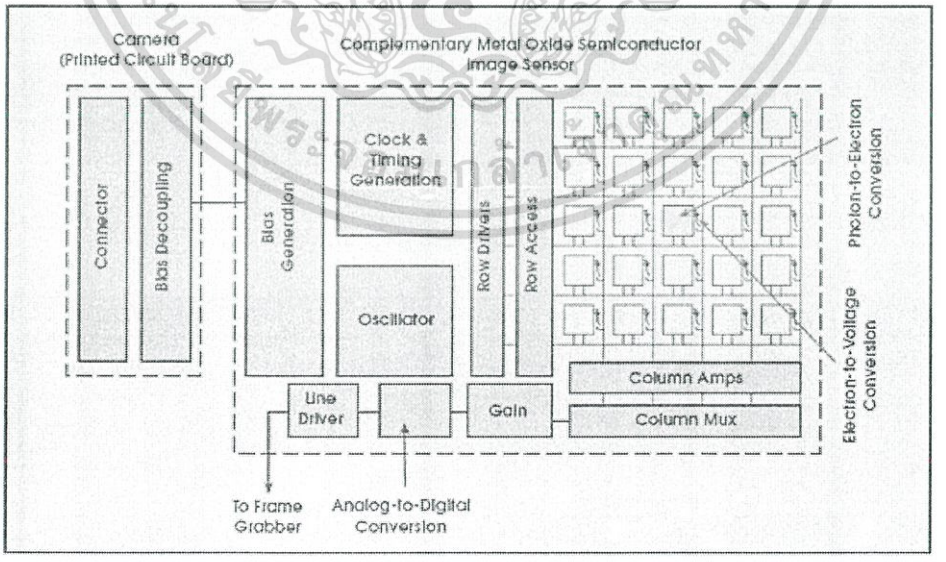
CCD ย่อมาจาก Charge Coupled Device เป็นเซนเซอร์ที่ทำงานโดยส่วนที่เป็นเซนเซอร์แต่ละพิกเซล (Pixel) จะทำหน้าที่รับแสง และเปลี่ยนค่าแสงเป็นสัญญาณอนาล็อก ส่งเข้าสู่วงจรเปลี่ยนค่าอนาล็อกเป็นสัญญาณดิจิตอลอีกที ซึ่งการรับแสงเป็นไปได้อย่างเต็มที่ไม่ต้องเสียพื้นที่ในการแปลงสัญญาณ ซึ่งตัวแปลงสัญญาณอยู่แยกกันทำให้สัญญาณรบกวนเกิดน้อย แต่ก็มีข้อเสียในด้านความร้อนและเปลืองพลังงาน



รูปที่ 2.5 การทำงานของ CCD Sensor

2.2.1.2 CMOS

CMOS ย่อมาจาก Complementary Metal Oxide Semiconductor เป็นเซนเซอร์ที่มีลักษณะการทำงานโดยแต่ละพิกเซลจะมีวงจรย่อยๆ เปลี่ยนค่าแสงที่เข้ามาเป็นสัญญาณดิจิตอลในทันทีไม่ต้องส่งออกไปแปลงเหมือน CCD



รูปที่ 2.6 การทำงานของ CMOS Sensor

CMOS ทำจากซิลิคอนทั่วไปเช่นเดียวกับอุปกรณ์ไฟฟ้าทั่วไปทำให้ราคาถูกกว่า CCD ด้วย CMOS ต้องมีพื้นที่ในการแปลงค่าสัญญาณในตัวทำให้พื้นที่ในการรับแสงนั้นศูนย์เสียไป ดังนั้นการแก้ปัญหานี้ คือ ได้เพิ่มขนาดเซนเซอร์ CMOS ให้มีขนาดที่ใหญ่ขึ้นเพื่อเพิ่มปริมาณพื้นที่รับแสงให้ดีขึ้น

ตารางที่ 2.1 แสดงการเปรียบเทียบคุณสมบัติระหว่าง CCD & CMOS Sensor

	CCD	CMOS
Signal Out of Pixel	Electron Packet	Voltage
Signal Out of Chip	Voltage (Analog)	Bits (Digital)
Signal Out of Camera	Bits (Digital)	Bits (Digital)
Fill Factor	High	Moderate
Amplifier Mismatch	N/A	Moderate
System Noise	Low	Moderate to High
System Complexity	High	Low
Sensor Complexity	Low	High
Camera Components	PCB + Multiple Chips + Lens	Chip + Lens
Responsibility	Moderate	Slightly Better
Dynamic Range	High	Moderate
Uniformity	High	Low to Moderate
Uniform Shuttering	Fast, Common	Poor
Speed	Moderate to High	Higher
Windowing	Limited	Extensive
Antiblooming	High to None	High
Biasing and Clocking	Multiple, Higher Voltage	Single, Low-Voltage

ถ้าเปรียบเทียบตารางด้านบนจะพบว่า CCD ให้คุณภาพที่ดีกว่า แต่กระบวนการผลิต และการนำไปใช้งานจะซับซ้อนกว่า กินพลังงานมากกว่า ซึ่งทำให้ CCD จะมีราคาแพงกว่า CMOS โดยทั่วไป

สรุปคือ CMOS จะมีตัวรับแสง และแปลงสัญญาณในแต่ละพิกเซลเลย ส่วน CCD ตัวรับแสงจะรับแสงอย่างเดียว และจะส่งค่าที่ได้ออกมาให้วงจรที่มีหน้าที่แปลงสัญญาณอีกทีสมัยก่อน CMOS มักนำไปใช้กับกล้องดิจิตอลในระดับล่าง เพราะคุณภาพของภาพยังไม่ดีนัก (แต่ในปัจจุบัน การพัฒนาเซนเซอร์ดีขึ้นอย่างมาก จึงไม่จำเป็นแล้วว่า CCD หรือ CMOS ที่ดีกว่ากัน เพราะการรับแสงจากเซนเซอร์มีหลายปัจจัยในการได้คุณภาพที่ดีออกมา)

2.3 ไมโครคอนโทรลเลอร์ (Microcontroller)

เป็นอุปกรณ์ไอซี (IC : Integrated Circuit) ที่สามารถโปรแกรมการทำงานได้ซับซ้อน สามารถรับข้อมูลในรูปสัญญาณดิจิตอลเข้าไปทำการประมวลผลแล้วส่งผลลัพธ์ข้อมูลดิจิตอลออกมา เพื่อนำไปใช้งานตามที่ต้องการได้

ไมโครคอนโทรลเลอร์ภายในชิพจะมีหน่วยความจำ, Port อยู่ในชิพเพียงตัวเดียว ซึ่งอาจจะเรียกได้ว่าเป็นคอมพิวเตอร์ชิพเดี่ยว ไมโครคอนโทรลเลอร์เป็นไมโครโพรเซสเซอร์ชนิดหนึ่งเช่นเดียวกับหน่วยประมวลผลกลาง (CPU : Central Processing Unit) ที่ใช้ในคอมพิวเตอร์ แต่ได้รับการพัฒนาแยกออกมาภายหลังเพื่อนำไปใช้ในวงจรทางด้านงานควบคุมคือ แทนที่ในการใช้งานจะต้องต่อวงจรภายนอกต่างๆ เพิ่มเติมเช่นเดียวกับไมโครโปรเซสเซอร์ ก็จะทำการรวมวงจรที่จำเป็น เช่น หน่วยความจำ, ส่วนอินพุต/เอาต์พุต บางส่วนเข้าไปในตัวไอซีเดียวกัน และเพิ่มวงจรบางอย่างเข้าไปด้วยเพื่อให้มีความสามารถเหมาะสมกับการใช้ในงานควบคุม เช่น วงจรตั้งเวลา, วงจรการสื่อสารอนุกรม, วงจรแปลงสัญญาณอนาล็อกเป็นดิจิตอล เป็นต้น สรุปคือ

Microcontroller = Microprocessor + Memory + I/O

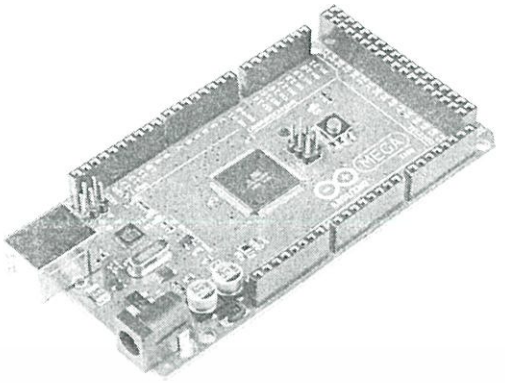
ไมโครคอนโทรลเลอร์สามารถนำไปประยุกต์ใช้งานอย่างกว้างขวาง โดยมักจะเป็นการนำไปใช้ฝังในระบบของอุปกรณ์อื่นๆ (Embeded Systems) เพื่อใช้ควบคุมการทำงานบางอย่าง เช่น ใช้ในรถยนต์, เตาอบไมโครเวฟ, เครื่องปรับอากาศ, เครื่องซักผ้าอัตโนมัติ เป็นต้น เพราะว่าไมโครคอนโทรลเลอร์มีข้อดีเหมาะสมต่อการใช้ในงานควบคุมหลายประการ เช่น

- ชิพไอซี และระบบที่ได้มีขนาดเล็ก
- ระบบที่ได้มีราคาถูกกว่าการใช้ชิพไมโครโพรเซสเซอร์
- วงจรที่ได้จะมีความซับซ้อนน้อย ช่วยลดข้อผิดพลาดที่อาจจะเกิดขึ้นได้ในการต่อวงจร
- มีคุณสมบัติเพิ่มเติมสำหรับงานควบคุมโดยเฉพาะ ซึ่งใช้งานได้ง่าย
- ช่วยลดระยะเวลาในการพัฒนาระบบได้

ไมโครคอนโทรลเลอร์มีหลายยี่ห้อ หลายตระกูล และหลายเบอร์ด้วยกัน ซึ่งแต่ละเบอร์ก็จะมีโครงสร้างภายในและความสามารถในการทำงานที่แตกต่างกันทำให้เลือกใช้กับงานได้อย่างเหมาะสม

2.3.1 ชุดโมดูลบอร์ดไมโครคอนโทรลเลอร์ Arduino รุ่น Arduino MEGA 2560 R3

เป็นไมโครคอนโทรลเลอร์ตระกูล AVR โดยใช้ไมโครคอนโทรลเลอร์เบอร์ ATmega2560 เป็น MCU ประจำบอร์ด โดย MCU รุ่นนี้มีขา Pin ทั้งหมด 54 ขา เพียบพร้อมไปด้วยทรัพยากรพื้นฐานต่างๆ อย่างครบถ้วน จึงมีความเหมาะสมเป็นอย่างยิ่งในการใช้งานทั่วไป



รูปที่ 2.7 ชุดโมดูลบอร์ดไมโครคอนโทรลเลอร์ Arduino MEGA 2560 R3

2.3.2 คุณสมบัติของบอร์ดไมโครคอนโทรลเลอร์

- ใช้ ATMEGA 2560 เป็น MCU ประจำบอร์ด Run ความถี่ 16MHz จาก Crystal Oscillator

- 256Kbyte Flash (สงวนไว้ 8KB สำหรับ Bootloader) / 8KB SRAM / 4KB EEPROM

- ใช้ USB สำหรับติดต่อสื่อสาร และ Download Code จากคอมพิวเตอร์ให้บอร์ด โดยไม่ต้องใช้เครื่องโปรแกรมจากภายนอก

- 1.0 Pinout เพิ่ม SDA และ SCL (อยู่ใกล้กับ AREF Pin) และอีก 2 Pin ใหม่คือ IOREF เป็น Pin ที่ใช้ในการเชื่อมต่อกับ Shields เพื่อแปลงเป็นแรงดันที่ได้จากบอร์ด ส่วนอีก 1 Pin ที่เหลือมีไว้สำหรับใช้ร่วมกับ AVR ในอนาคต

- ใช้ ATmega 16U2 แทน 8U2

- 54 Pin Digital I/O (15 Pin สามารถใช้เป็น PWM Output ได้)

- 16 Pin Analog Input

- 4 UART

- ICSP header : In-Circuit Serial Programming (ส่วนที่เป็น AVR ขนาดเล็กสำหรับการโปรแกรม Arduino ซึ่งประกอบด้วย MOSI, MISO, SCK, RESET, VCC, GND)

- สามารถเชื่อมรับพลังงานโดยการเชื่อมต่อ Micro USB Connector หรือจาก Power Supply จากภายนอกได้ โดยแหล่งพลังงานจะถูกเลือกโดยอัตโนมัติ

- แหล่งจ่ายจากภายนอกสามารถมาได้จาก AC-to-DC Adapter หรือจากแบตเตอรี่ โดยต่อเข้ากับ 2.1 มม. Center-Positive Plug ไปยังช่องเสียบแหล่งจ่าย และการต่อเข้ากับแบตเตอรี่สามารถทำได้โดยการต่อเข้ากับ GND และ Vin Pin Header ของ Power Connecter

- บอร์ดสามารถทำงานได้ในช่วงแรงดัน 6 โวลต์ ถึง 20 โวลต์ ถ้าแหล่งจ่ายมีค่าต่ำกว่า 7 โวลต์ อาจส่งผลให้ 5 โวลต์ Pin มีแรงดันที่ต่ำกว่า 5 โวลต์ และบอร์ดอาจจะไม่เสถียร แต่ถ้าหากแรงดันมีค่าสูงกว่า 12 โวลต์ อาจส่งผลให้บอร์ด Overheat และอาจทำให้บอร์ดเสียหายได้ ดังนั้นช่วงแรงดันที่เหมาะสมกับบอร์ดคือ 7 โวลต์ ถึง 12 โวลต์

- ในแต่ละ Digital Pins ทั้ง 54 pin บนบอร์ด Arduino MEGA 2560 สามารถเป็นได้ทั้ง Input และ Output โดยจะทำงานที่แรงดัน 5 โวลต์ และให้กระแสสูงสุด 40 mA

2.4 ระบบควบคุม (Control Systems)

ระบบควบคุม คือ ส่วนหรือหน่วยที่ได้จากการรวบรวมสิ่งต่างๆ เข้าด้วยกัน เพื่อใช้ในการบังคับควบคุมตัวเองหรือสิ่งอื่นๆ ให้เป็นไปตามที่ต้องการ

ระบบควบคุมแบ่งออกตามโครงสร้างได้ 2 ชนิด ได้แก่

1. ระบบควบคุมแบบวงเปิด (Open-Loop Control Systems) คือ ระบบควบคุมที่ไม่มีการป้อนกลับค่าเอาต์พุต (Output) ของระบบที่ได้กลับมาใช้ในการควบคุม การควบคุมแบบนี้ทำได้โดยการตั้งค่าอินพุตเริ่มต้นให้เหมาะสมทำให้ได้เอาต์พุตตามต้องการ ตัวอย่างระบบควบคุมแบบนี้ เช่น การควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรง (DC Motor) ซึ่งทำได้โดยการตั้งค่าแรงดันให้เหมาะสมจนกระทั่งขับให้มอเตอร์หมุนไปด้วยความเร็วรอบตามที่ต้องการ แล้วก็ใช้ค่าของแรงดันนั้นเป็นอินพุตให้กับระบบตลอดไป ระบบควบคุมประเภทนี้มีข้อดีที่ไม่ต้องมีอุปกรณ์มากมาย ประหยัดเวลา และค่าใช้จ่ายในการสร้าง แต่มีข้อเสียก็คือ ไม่สามารถชดเชยความผิดพลาดที่เกิดจากปัจจัยอื่นๆ ที่ไม่สามารถทำนายได้

2. ระบบควบคุมแบบวงปิด (Closed-Loop Control Systems) คือ ระบบที่นำเอาเอาต์พุตป้อนกลับมาใช้ในการควบคุมอยู่ตลอดเวลา ซึ่งเรียกอีกอย่างหนึ่งว่าระบบควบคุมแบบป้อนกลับ (Feedback Control Systems) ระบบควบคุมแบบวงปิดนี้มีข้อดีที่เหนือกว่าแบบวงเปิดตรงที่สามารถกำจัดผลของสัญญาณรบกวนที่มีต่อเอาต์พุต ซึ่งทำให้ควบคุมระบบได้อย่างแม่นยำ ระบบทั่วไปทางกายภาพ แม้กระทั่งระบบทางชีววิทยาของสัตว์ ต้นไม้ ก็มักจะมีกลไกในการป้อนกลับเพื่อควบคุมลักษณะทางกายภาพตามต้องการ เช่น การเคลื่อนที่ของสัตว์ ที่ต้องใช้ตามองทิศทางตลอดเวลา กลไกการควบคุม

ความร้อนในร่างกาย เมื่ออากาศภายนอกร้อนขึ้น ร่างกายมีกลไกในการขับเหงื่อเพื่อชดเชยให้อุณหภูมิที่ผิวหนังลดลง

2.4.1 ระบบควบคุมแบบป้อนกลับ (Feedback Control Systems)

ระบบควบคุมแบบป้อนกลับ (Feedback Control Systems) เป็นระบบควบคุมที่พยายามรักษาเอาต์พุตให้ได้ตามต้องการ โดยการนำเอาสัญญาณเอาต์พุตมาเปรียบเทียบกับสัญญาณอ้างอิงที่ต้องการแล้วนำค่าความแตกต่างไปใช้ในการควบคุมสัญญาณป้อนให้กับสิ่งที่ต้องการควบคุม

2.4.1.1 องค์ประกอบพื้นฐานของระบบควบคุมแบบป้อนกลับ

องค์ประกอบพื้นฐานของระบบควบคุมแบบป้อนกลับ แสดงได้ดังบล็อกไดอะแกรม (Block Diagram)ดังแสดงในรูปที่ 2.1

รูปที่ 2.8 องค์ประกอบพื้นฐานของระบบควบคุมแบบป้อนกลับ

ระบบควบคุมแบบป้อนกลับข้างต้นมีส่วนประกอบย่อย ดังต่อไปนี้

- แพลนท์ (Plant) หรือ โพรเซส (Process) คือ ระบบทางกายภาพใดๆ ที่ต้องการควบคุม เช่น ถ้าต้องการควบคุมอุณหภูมิของเตาอบ แพลนท์ ก็จะประกอบไปด้วยตู้อบ ตัวทำความร้อน (Heater)

- สัญญาณเอาต์พุต (Output) คือ ตัวแปรทางกายภาพของแพลนท์ (Process Variable) ที่ต้องการควบคุม เช่น อุณหภูมิที่ได้จากเตาอบ เป็นต้น

- อุปกรณ์วัด (Measurement Element) คือ เครื่องมือวัดที่ใช้ตรวจจับสัญญาณเอาต์พุต ซึ่งจะทำการแปลงตัวแปรทางกายภาพให้เป็นสัญญาณที่พร้อมจะนำไปเปรียบเทียบกับสัญญาณอินพุต เช่น การใช้เทอร์โมคัปเปิล (Thermo Couple) ในการวัดค่าอุณหภูมิโดยให้ค่าเป็นแรงดันที่สัมพันธ์กับอุณหภูมิที่วัด ถ้าการเปรียบเทียบ และการคำนวณต่างๆ กระทำบนคอมพิวเตอร์ อุปกรณ์การวัดอาจหมายรวมไปถึงอุปกรณ์ที่ใช้ แปลงสัญญาณแบบต่อเนื่อง (Analog) ไปเป็นสัญญาณที่สามารถประมวลผลได้ด้วยคอมพิวเตอร์ หรือที่ เรียกว่าสัญญาณ ดิจิตอล (Digital) เราจะเรียกอุปกรณ์แบบนี้ว่า Analog to Digital Converter (ADC หรือ A/D)

- สัญญาณป้อนกลับ (Feedback) คือ สัญญาณที่ได้จากอุปกรณ์วัด ซึ่งผ่านการแปลงให้อยู่ในรูปของสัญญาณ ที่พร้อมนำไปประมวลผลเพื่อการควบคุมระบบ

- สัญญาณอินพุต (Input) หรือคำสั่ง (Command) หรือค่าที่ต้องการ (Set Point) หรือบางครั้งอาจเรียกว่า สัญญาณอ้างอิงที่ต้องการ (Reference input) เป็นค่าหรือรูปแบบสัญญาณที่ต้องการให้ได้ในด้านเอาต์พุตของระบบ หรือนั่นก็คือ ต้องการให้เอาต์พุตมีค่าเท่ากับค่านี้นั่นเอง สัญญาณอินพุตนี้จะถูกนำไปเปรียบเทียบกับเอาต์พุต ดังนั้นรูปแบบของสัญญาณจะต้องเป็นเช่นเดียวกับสัญญาณป้อนกลับ เช่น ถ้าเราต้องการควบคุมอุณหภูมิของเตาอบที่ 200°C และใช้เทอร์โมคัปเปิลวัดค่าอุณหภูมิ ซึ่งให้ค่าแรงดันที่ 200°C เป็นค่า 5 โวลต์ ดังนั้นเราต้องตั้งค่าอินพุตเป็น 5 โวลต์

- สัญญาณความผิดพลาด (Error หรือ Deviation) คือ สัญญาณความแตกต่างของสัญญาณป้อนกลับเปรียบเทียบกับสัญญาณอินพุตที่ต้องการ เช่น ถ้าอุณหภูมิเอาต์พุตเป็น 150°C ก็จะเกิดความผิดพลาดไป 50°C

- ตัวควบคุม (Controller) คือ อุปกรณ์ที่ทำหน้าที่ประมวลผลหรือคำนวณ แล้วส่งสัญญาณที่เหมาะสมไปควบคุมระบบ เพื่อให้ระบบเกิดการตอบสนอง และได้ค่าเอาต์พุตตามต้องการ ตัวควบคุมอาจเป็นไปได้ทั้ง ระบบจักรกล ไฟฟ้า หรือคอมพิวเตอร์ ซึ่งจะใช้กันอย่างแพร่หลายในปัจจุบัน เพราะมีข้อดีในการประมวลผล ที่รวดเร็ว ถูกต้อง และแม่นยำ

2.4.2 การควบคุมแบบ PID

PID Controller (ระบบควบคุมแบบสัดส่วน-ปริพันธ์-อนุพันธ์) เป็นระบบควบคุมแบบป้อนกลับที่ใช้กันอย่างกว้างขวาง ซึ่งค่าที่นำไปใช้ในการคำนวณเป็นค่าความผิดพลาด ที่หามาจากความแตกต่างของตัวแปรในกระบวนการและค่าที่ต้องการ ตัวควบคุมจะพยายามลดค่าผิดพลาดให้เหลือน้อยที่สุดด้วยการปรับค่าสัญญาณขาเข้าของกระบวนการ ค่าตัวแปรของ PID ที่ใช้จะปรับเปลี่ยนตามธรรมชาติของระบบ

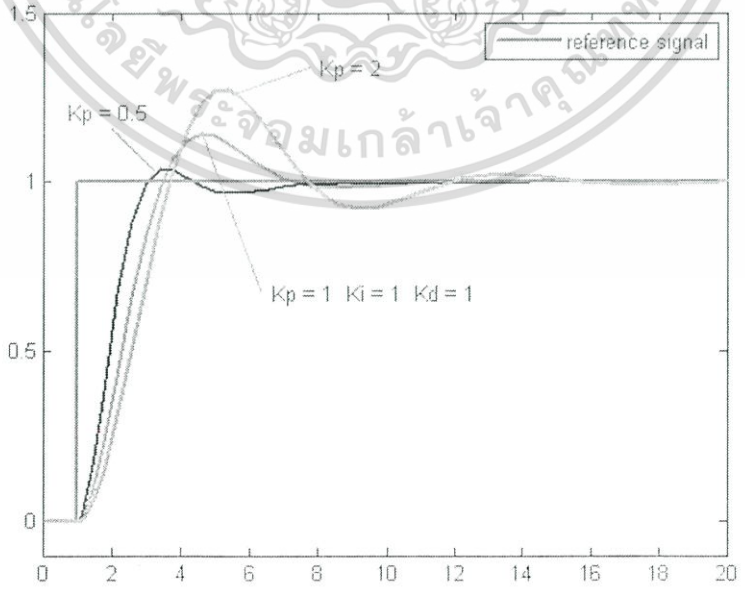
วิธีคำนวณของ PID ขึ้นอยู่กับสามตัวแปรคือ ค่าสัดส่วน, ปริพันธ์ และอนุพันธ์ ค่าสัดส่วนกำหนดจากผลของความผิดพลาดในปัจจุบัน, ค่าปริพันธ์กำหนดจากผลบนพื้นฐานของผลรวมความผิดพลาดที่ ซึ่งพึ่งผ่านพ้นไป และค่าอนุพันธ์กำหนดจากผลบนพื้นฐานของอัตราการเปลี่ยนแปลงของค่าความผิดพลาด น้ำหนักที่เกิดจากการรวมกันของทั้งสามนี้จะใช้ในการปรับกระบวนการโดยการปรับค่าคงที่ใน PID ตัวควบคุมสามารถปรับรูปแบบการควบคุมให้เหมาะกับที่กระบวนการต้องการได้ การตอบสนองของตัวควบคุมจะอยู่ในรูปของการไหวตัวของตัวควบคุมจนถึงค่าความผิดพลาด ค่าโอเวอร์ชูต (Overshoots) และค่าแกว่งของระบบ (Oscillation) วิธี PID ไม่รับประกันได้ว่าจะเป็นระบบควบคุมที่เหมาะสมที่สุด หรือสามารถทำให้กระบวนการมีความเสถียรแน่นอน การประยุกต์ใช้งานบางครั้งอาจใช้เพียงหนึ่งถึงสองรูปแบบ ขึ้นอยู่กับกระบวนการเป็นสำคัญ PID บางครั้งจะถูกเรียกว่าการควบคุมแบบ PI, PD, P, I หรือ D ขึ้นอยู่กับว่าใช้รูปแบบใดบ้าง

ทฤษฎีการควบคุมแบบ PID ได้ชื่อตามการรวมกันของเทอมของตัวแปรทั้งสามตามสมการที่ (2.1)

$$u(t) = P_{out} + I_{out} + D_{out} \tag{2.1}$$

เมื่อ P_{out} , I_{out} และ D_{out} เป็นผลของสัญญาณขาออกจากระบบควบคุม PID จากแต่ละเทอม ซึ่งนิยามตามรายละเอียดด้านล่าง

2.4.2.1 สัดส่วน Proportional Control Action (P - Action)



รูปที่ 2.9 กราฟ PV ต่อเวลา K_p กำหนดเป็น 3 ค่า (K_i และ K_d คงที่)

เทอมของสัดส่วน (บางครั้งเรียก อัตราขยาย) จะเปลี่ยนแปลงเป็นสัดส่วนของค่าความผิดพลาดการตอบสนองของสัดส่วน สามารถทำได้โดยการคูณค่าความผิดพลาดด้วยค่าคงที่ K_p หรือที่เรียกว่าอัตราขยายสัดส่วนเทอมของสัดส่วนจะเป็นไปตามสมการที่ (2.2)

$$P_{out} = K_p e(t) \tag{2.2}$$

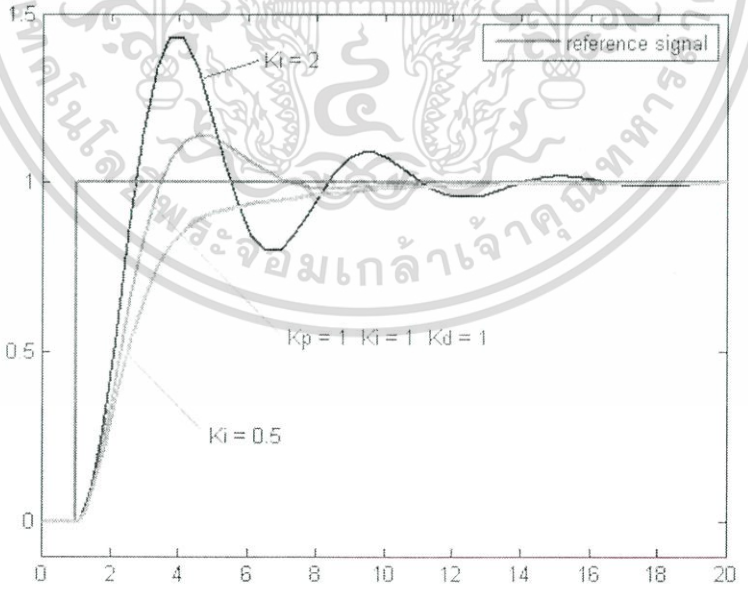
เมื่อ P_{out} : สัญญาณขาออกของเทอมสัดส่วน

K_p : อัตราขยายสัดส่วน, ตัวแปรปรับค่าได้

$e(t)$: ค่าความผิดพลาดในแต่ละช่วงเวลา

ผลอัตราขยายสัดส่วนที่สูงค่าความผิดพลาดก็จะเปลี่ยนแปลงมากเช่นกัน แต่ถ้าสูงเกินไประบบจะไม่เสถียรได้ ในทางตรงกันข้ามผลอัตราขยายสัดส่วนที่ต่ำระบบควบคุมจะมีผลตอบสนองต่อกระบวนการน้อยตามไปด้วย

2.4.2.2 ปริพันธ์ Integral Control Action (I-Action)



รูปที่ 2.10 กราฟ PV ต่อเวลา K_i กำหนดเป็น 3 ค่า (K_p และ K_d คงที่)

ผลจากเทอมปริพันธ์ (บางครั้งเรียก Reset) เป็นสัดส่วนของขนาดความผิดพลาด และระยะเวลาของความผิดพลาด ผลรวมของความผิดพลาดในทุกช่วงเวลา (ปริพันธ์ของความผิดพลาด) จะให้ออฟเซตสะสมที่ควรจะเป็นในก่อนหน้า ความผิดพลาดสะสมจะถูกคูณโดยอัตราขยายปริพันธ์ ขนาดของผลของเทอมปริพันธ์จะกำหนดโดยอัตราขยายปริพันธ์, K_i

เทอมปริพันธ์จะเป็นไปตามสมการที่ (2.3)

$$I_{out} = K_i \int_0^t e(t)\, dt \qquad (2.3)$$

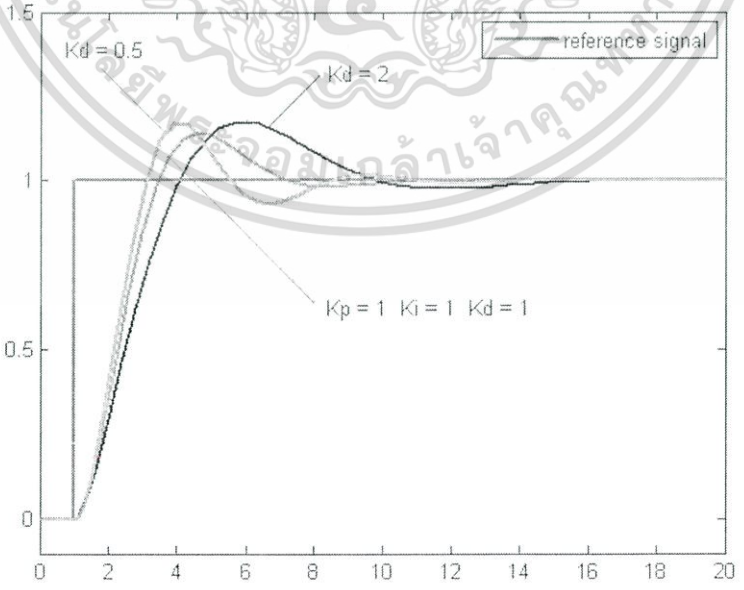
เมื่อ I_{out} : สัญญาณขาออกของเทอมปริพันธ์

K_i : อัตราขยายปริพันธ์, ตัวแปรปรับค่าได้

e(t) : ค่าความผิดพลาดในแต่ละช่วงเวลา

เทอมปริพันธ์ (เมื่อรวมกับเทอมสัดส่วน) จะเร่งกระบวนการให้เข้าสู่จุดที่ต้องการ และขจัดความผิดพลาดที่เหลืออยู่ที่เกิดจากการใช้เพียงเทอมสัดส่วน แต่อย่างไรก็ตามเทอมปริพันธ์เป็นการตอบสนองต่อความผิดพลาดสะสมในอดีต จึงสามารถทำให้เกิดโอเวอร์ชูตได้ (ข้ามจุดที่ต้องการ และเกิดการหันเหไปทางทิศทางอื่น)

2.4.2.3 อนุพันธ์ Derivative Control Action (D-Action)



รูปที่ 2.11 กราฟ PV ต่อเวลา สำหรับ K_d 3 ค่า (K_p และ K_i คงที่)

อัตราการเปลี่ยนแปลงของความผิดพลาดจากกระบวนการนั้นคำนวณหาจากความชันของความผิดพลาดทุกๆ เวลา (นั่นคือ เป็นอนุพันธ์อันดับหนึ่งสัมพันธ์กับเวลา) และคูณด้วยอัตราขยายอนุพันธ์ K_d ขนาดของผลของเทอมอนุพันธ์ (บางครั้ง เรียก อัตรา) ขึ้นกับอัตราขยายอนุพันธ์ K_d เทอมอนุพันธ์เป็นไปตามสมการที่ (2.4)

$$D_{out} = K_d \frac{d}{dt} e(t) \tag{2.4}$$

เมื่อ D_{out} : สัญญาณขาออกของเทอมอนุพันธ์

K_d : อัตราขยายอนุพันธ์, ตัวแปรปรับค่าได้

$e(t)$: คือค่าความผิดพลาดในแต่ละช่วงเวลา

เทอมอนุพันธ์จะชะลออัตราการเปลี่ยนแปลงของสัญญาณขาออกของระบบควบคุม และด้วยผลนี้จะช่วยให้ระบบควบคุมเข้าสู่จุดที่ต้องการ ดังนั้นเทอมอนุพันธ์จะใช้ในการลดขนาดของโอเวอร์ชูตที่เกิดจากเทอมปริพันธ์และทำให้เสถียรภาพของการรวมกันของระบบควบคุมดีขึ้น แต่อย่างไรก็ตามอนุพันธ์ของสัญญาณรบกวนที่ถูกขยายในระบบควบคุมจะไวมากต่อการรบกวนในเทอมของความผิดพลาด และสามารถทำให้กระบวนการไม่เสถียรได้ถ้าสัญญาณรบกวน และอัตราขยายอนุพันธ์มีขนาดใหญ่เพียงพอ ซึ่งลักษณะการทำงานของตัวควบคุมแบบ PID สามารถเขียนเป็นสมการทางคณิตศาสตร์ในเทอมของเวลาได้ดังนี้

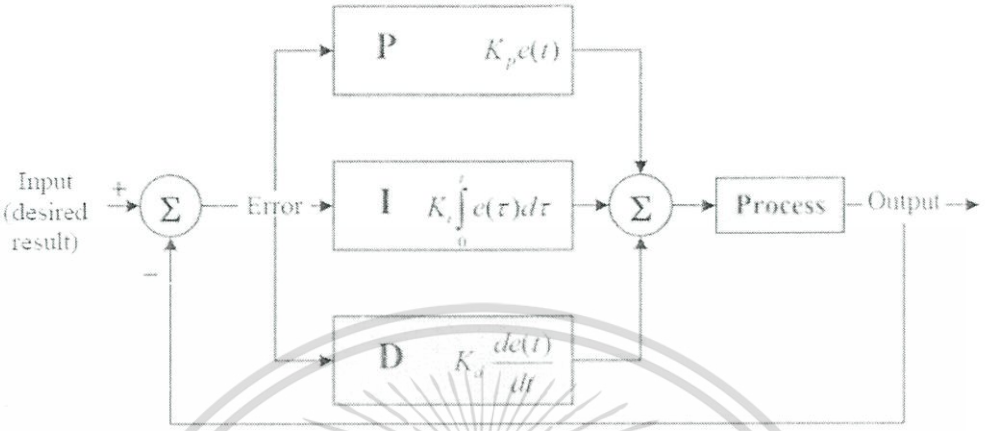
$$u(t) = K_p e(t) + K_i \int_0^t e(t)\,dt + K_d \frac{d}{dt} e(t) \tag{2.5}$$

เมื่อทำอยู่ในรูป Laplace Transform จะได้

$$U(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s\right) \tag{2.6}$$

และจากสมการที่ (2.5) สามารถแสดงในรูปของไดอะแกรมของการประยุกต์ใช้หลักการควบคุมแบบ PID ในการปรับค่าเกนของตัวควบคุมดังรูปที่ 2.12 นอกจากนี้ยังสามารถนำไปทำงานร่วมกับเทคนิคการ

เรียนรู้ และชดเชยค่าความไม่แน่นอน และวิธี Conventional Boundary Layer Technique (BL Technique) ได้



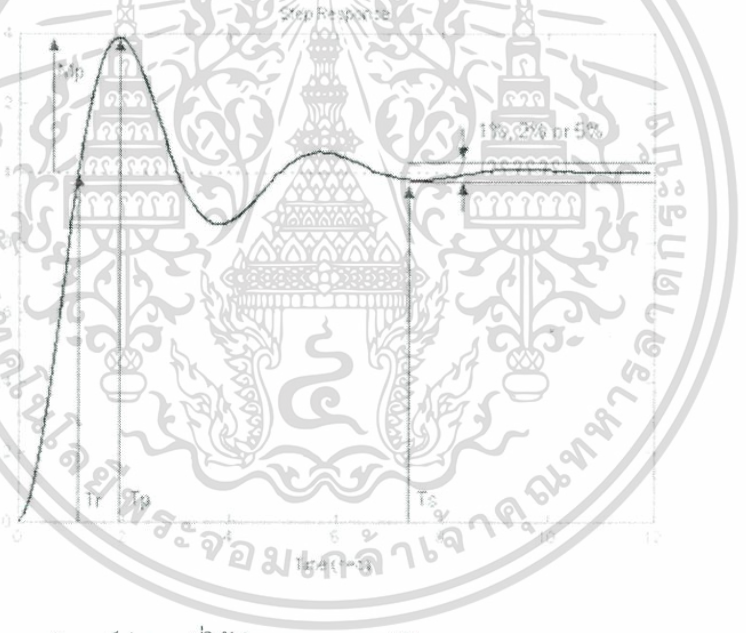
รูปที่ 2.12 ส่วนประกอบของระบบควบคุมอัตโนมัติ

จากรูปที่ 2.12 ระบบควบคุมจะเริ่มพิจารณาจากเอาต์พุตของกระบวนการ ซึ่งอาจจะเป็นอุณหภูมิ หรือความดัน เป็นต้น เอาต์พุตจะถูกวัด และแปลงสัญญาณโดยอุปกรณ์แปลงสัญญาณเพื่อแปลงปริมาณทางกายภาพที่วัดได้ไปเป็นปริมาณที่ต้องการ เช่น ปริมาณทางไฟฟ้า จากนั้นค่าที่วัดได้จะถูกนำไปเปรียบเทียบกับค่าอินพุตที่ตั้งไว้ ค่าผิดพลาดที่เกิดขึ้นจะถูกส่งไปให้ตัวควบคุมเพื่อสร้างสัญญาณควบคุมไปควบคุมกระบวนการต่อไป ขั้นตอนทั้งหมดนี้จะทำซ้ำไปเรื่อยๆ จนกระทั่งค่าอินพุตกับค่าที่วัดได้มีขนาดเท่ากันหรือใกล้เคียงกัน ส่วนประกอบหนึ่งที่ถือว่าเป็นหัวใจสำคัญของระบบควบคุมอัตโนมัติ คือ ตัวควบคุม (Controller) ซึ่งมีมากมายหลายชนิดให้เลือกใช้งาน แต่ตัวควบคุมที่ยังคงได้รับความนิยมอย่างสูงนับจากอดีตจนถึงปัจจุบันก็คือ ตัวควบคุมแบบ PID สาเหตุที่ทำให้ตัวควบคุมชนิดนี้เป็นที่นิยมใช้เนื่องจากความเรียบง่ายของโครงสร้างตัวควบคุม และความสามารถในการลดค่าความผิดพลาดได้หลายชนิดในตัวควบคุมเดียว ปัญหาของการใช้งานตัวควบคุมแบบ PID ก็คือ ค่าเกนของตัวควบคุม ซึ่งมีอยู่ด้วยกันถึง 3 ตัวคือ K_p, K_i และ K_d ควรจะมีค่าเป็นเท่าใดจึงจะเหมาะสมกับกระบวนการนั้นๆ ในทางทฤษฎีแล้วค่าเกนเหล่านี้จะสามารถหาได้อย่างถูกต้อง ถ้าทราบแบบจำลองทางคณิตศาสตร์ของกระบวนการ แต่ในทางปฏิบัติการหาแบบจำลองทางคณิตศาสตร์ของกระบวนการในอุตสาหกรรมไม่ใช่เรื่องง่ายอีกทั้งยังมีค่าใช้จ่ายสูง เนื่องจากตัวแปรต่างๆ ในกระบวนการมีการเปลี่ยนแปลงตลอดเวลา และเซนเซอร์ที่จะต้องนำมาวัดตัวแปรต่างๆ ก็มีราคาแพง

จากปัญหาดังกล่าวสามารถปรับค่าเกนตัวควบคุมแบบ PID ในกรณีที่ไม่ทราบแบบจำลองทางคณิตศาสตร์ของกระบวนการ โดยจะใช้วิธีการปรับค่าเกนตัวควบคุมแบบ PID ของซีเกลอร์-นิโคลส์ (Ziegler-Nichols) ร่วมกับตารางแสดงผลกระทบของค่าเกนในตัวควบคุมแบบ PID ต่อการตอบสนองของระบบ

2.4.2.4 ผลการตอบสนองของระบบ

กระบวนการ หรือระบบ เมื่อได้รับอินพุตจะต้องมีการตอบสนองออกมาเป็นเอาต์พุต ซึ่งรูปแบบการตอบสนองนั้นก็จะขึ้นอยู่กับชนิดของอินพุตที่ใส่เข้าไป ในการนิยามตัวแปรต่างๆ ในการตอบสนองของระบบ มักจะนิยมป้อนอินพุตเป็นฟังก์ชันหนึ่งหน่วย แล้วดูผลการตอบสนองของระบบพลศาสตร์โดยทั่วไปเมื่อได้รับอินพุตเป็นฟังก์ชันหนึ่งหน่วยก็จะมีการตอบสนองเป็นดังรูปที่ 2.13



รูปที่ 2.13 พารามิเตอร์ต่างๆ ที่ใช้กำหนดคุณสมบัติเฉพาะของการตอบสนองของระบบ

จากรูปที่ 2.13 พารามิเตอร์ต่างๆ สามารถอธิบายความหมายได้ดังนี้

1. ช่วงเวลาขึ้น (Rise Time, Tr) หมายถึง ช่วงเวลาที่ผลตอบสนองเมื่อสัญญาณเอาต์พุตเพิ่มจาก 10% จนถึง 90% หรือจาก 5% ถึง 95% หรือจาก 0% ถึง 100% ดังนั้นการกำหนดช่วงเวลาขึ้นจำเป็นต้องบอกด้วยว่าวัดโดยใช้ช่วงเวลาไหน

2. เวลาของค่ายอด (Peak Time, Tp) หมายถึง เวลาที่สัญญาณผลการตอบสนองมีค่าสูงสุดค่าแรกของผลการตอบสนองนั้นๆ

3. โอเวอร์ชูตสูงสุด (Maximum Overshoot, Mp) หมายถึง ค่าการตอบสนองสูงสุดที่วัดจากสถานะอยู่ตัวสุดท้าย (Final Steady State) การบอกค่าโอเวอร์ชูตสูงสุดมักจะบอกเป็นเปอร์เซ็นต์

4. เวลาเข้าที่ (Settling Time, Ts) หมายถึง เวลาที่ผลการตอบสนองลดลงจนเริ่มเข้าไปอยู่ในช่วงที่กำหนด ซึ่งจะวัดเทียบกับค่าสุดท้ายของผลการตอบสนองในสถานะอยู่ตัวสุดท้าย (Final Steady State) ค่าที่นิยมกำหนดสำหรับช่วงนี้มักจะบอกเป็นเปอร์เซ็นต์ เช่น 1%, 2% หรือ 5% เป็นต้น

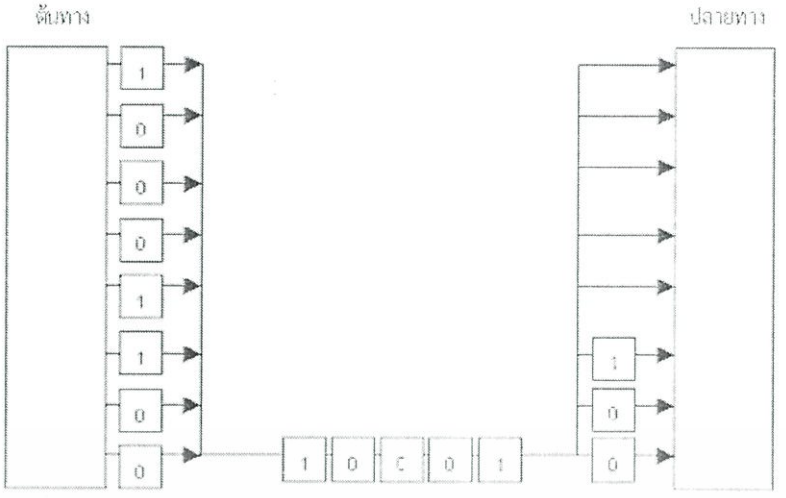
ตารางที่ 2.2 ผลกระทบของค่าเกนในตัวควบคุมแบบ PID ต่อการตอบสนองของระบบ

ค่าเกน	ช่วงเวลาขึ้น (Tr)	โอเวอร์ชูต (Mp)	เวลาเข้าที่ (Ts)	ค่าความผิดพลาด ณ สถานะคงตัว	เสถียรภาพ
K_p	ลดลง	เพิ่มขึ้น	เปลี่ยนแปลงเล็กน้อย	ลดลง	ลดลง
K_i	ลดลง	เพิ่มขึ้น	เพิ่มขึ้น	ลดลงจนหมดไป	ลดลง
K_d	ลดลงเล็กน้อย	ลดลง	ลดลง	เปลี่ยนแปลงน้อยมาก	ดีขึ้นถ้า K_d มีค่าน้อย

จากตารางที่ 2.2 จะแสดงถึงผลกระทบของค่าเกนในตัวควบคุมแบบ PID ต่อพารามิเตอร์ต่างๆ ที่ใช้กำหนดคุณสมบัติเฉพาะของการตอบสนองของระบบ ซึ่งจะพบว่าค่าเกน K_p จะทำให้ช่วงเวลาขึ้น (Rise Time) ลดลง และ ลดค่าความผิดพลาด ณ สถานะคงตัว แต่ไม่สามารถกำจัดค่าความผิดพลาด ณ สถานะคงตัวให้หมดได้ ค่าเกน K_i จะมีหน้าที่หลักในการลดค่าความผิดพลาด ณ สถานะคงตัวให้หมดไปแต่การเพิ่มค่าเกน K_i มากเกินไปจะทำให้ผลการตอบสนองชั่วครู่ของระบบเสียไปได้ส่วนค่าเกน K_d มีหน้าที่หลักในการลดโอเวอร์ชูตลง และทำให้ผลการตอบสนองชั่วครู่ของระบบดีขึ้น

2.5 การส่งผ่านข้อมูลแบบอนุกรม (Serial Transimission)

รูปแบบการส่งผ่านข้อมูลในลักษณะนี้ทุกบิตที่เข้ารหัสแทนข้อมูลหนึ่งตัวอักษร จะถูกส่งผ่านไปตามสายส่งเรียงลำดับกันไปทีละบิตในสายส่งเพียงเส้นเดียว ดังรูปที่ 2.14



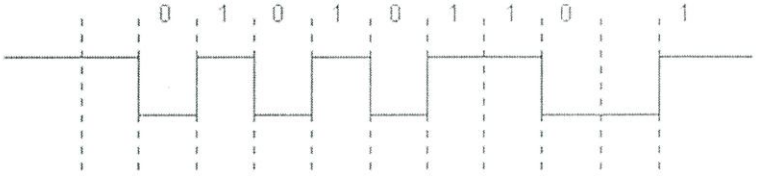
รูปที่ 2.14 การส่งข้อมูลแบบอนุกรม

จากรูปที่ 2.14 ตัวอักษรจะประกอบด้วย 8 บิต เรียงเป็นลำดับ ข้อมูลจะถูกส่งออกมาทีละบิต ระหว่างต้นทาง และปลายทาง ปลายทางจะรวบรวมบิตเหล่านี้ทีละบิตจนครบ 8 บิต เป็น 1 ตัวอักษร จะเห็นว่าการส่งข้อมูลแบบนี้จะช้ากว่าแบบขนาน แต่ค่าใช้จ่ายจะถูกกว่าแบบขนาน ซึ่งเหมาะสำหรับการส่งระยะทางไกลๆ

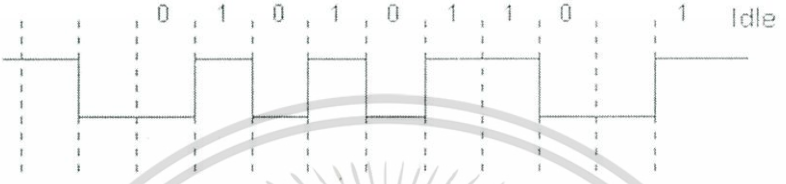
โดยทั่วไปแล้วการส่งข้อมูลนั้นจะประกอบไปด้วยกลุ่มของตัวอักษร ในการส่งข้อมูลแบบอนุกรมนี้จึงแบ่งวิธีการสื่อสารข้อมูลขึ้น 2 แบบคือ การสื่อสารแบบอะซิงโคนัส (Asynchronous Transmission) และการสื่อสารแบบซิงโคนัส (Synchronous Transmission)

2.5.1 การสื่อสารแบบอะซิงโคนัส (Asynchronous Transmission)

การสื่อสารแบบอะซิงโคนัส หรือเรียกอีกอย่างหนึ่งว่าเป็น การสื่อสารแบบระบุจุดเริ่มต้น และจุดสิ้นสุด (Start-Stop Transmission) ลักษณะของสัญญาณที่ใช้ในการติดต่อสื่อสารกันจะประกอบไปด้วย บิตเริ่มต้น (Start Bit) บิตของข้อมูลที่สื่อสาร (Transmission Data) จำนวน 8 บิต บิตตรวจข้อผิดพลาด (Parity Bit) และบิตสิ้นสุด (Stop Bit) สำหรับบิตตรวจสอบข้อผิดพลาดจะใช้หรือไม่ใช้ก็ได้ ดังนั้นสัญญาณจึงต้องประกอบด้วยส่วนประกอบอย่างน้อย 3 ส่วน ดังรูปที่ 2.15



รูปที่ 2.15 การสื่อสารแบบอะซิงโคนัสที่ไม่ได้ใช้พารีตี้บิต



รูปที่ 2.16 การสื่อสารแบบอะซิงโคนัสที่ใช้พารีตี้บิต

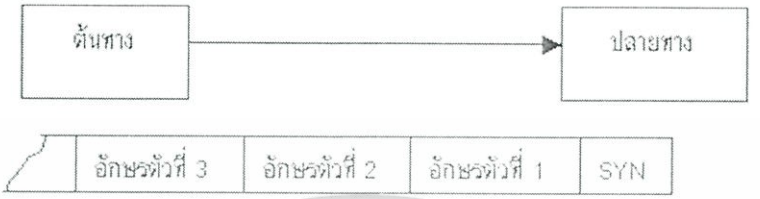
จากรูปที่ 2.15 และรูปที่ 2.16 จะเห็นว่าขณะที่ไม่มีข้อมูลส่งออกมาสถานะของการส่งจะเป็นแบบว่าง (Idle) ซึ่งจะมีระดับของสัญญาณเป็น 1 ตลอดเวลา เพื่อความแน่ใจว่าปลายทาง หรือฝ่ายรับยังคงติดต่อกับต้นทาง หรือฝ่ายส่งอยู่ เมื่อเริ่มจะส่งข้อมูลสัญญาณของอะซิงโคนัสจะเป็น 0 หนึ่งช่วงสัญญาณนาฬิกา ซึ่งบิตนี้เราเรียกว่า บิตเริ่มต้น ตามหลังของบิตเริ่มต้นจะเป็นบิตข้อมูลสำหรับ 1 ตัวอักษร ตามหลังบิตข้อมูลก็จะเป็นบิตตรวจข้อผิดพลาด แล้วจะตามด้วยบิตสิ้นสุด ถ้าไม่ใช่บิตตรวจข้อผิดพลาดตามหลังบิตข้อมูลก็จะเป็นบิตสิ้นสุดเลย หลังจากนั้นถ้าไม่มีข้อมูลส่งออกมาสัญญาณจะกลับไปอยู่ที่สถานะแบบว่างอีก เพื่อรอการส่งข้อมูลต่อไป

จะเห็นว่าการสื่อสารแบบอะซิงโคนัสนี้ มีลักษณะเป็นไปทีละตัวอักษร และสัญญาณที่ส่งออกมามีบางส่วนเป็นบิตเริ่มต้น บิตสิ้นสุด และบิตตรวจข้อผิดพลาด ทำให้ความเร็วในการส่งข้อมูลต่อวินาทีน้อยลงไป เนื่องจากต้อง สูญเสียช่องทางการสื่อสารให้กับบิตเริ่มต้น บิตสิ้นสุด และบิตตรวจข้อผิดพลาด (ถ้ามีใช้) ตลอดเวลาการสื่อสารแบบอะซิงโคนัสนี้มักใช้ในการติดต่อระหว่างคอมพิวเตอร์กับอุปกรณ์รอบข้าง

2.5.2 การสื่อสารแบบซิงโคนัส (Synchronous Transmission)

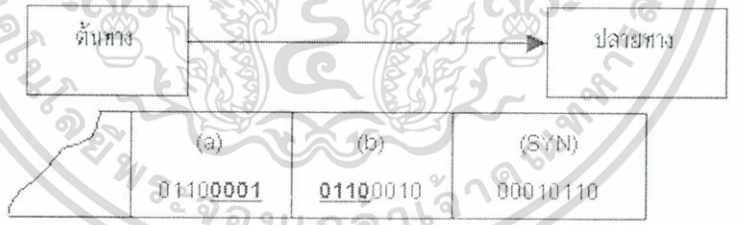
การสื่อสารแบบซิงโคนัส จะทำการจัดกลุ่มของข้อมูลเป็นกลุ่มๆ และทำการส่งข้อมูลทั้งกลุ่มไปพร้อมกันในทีเดียว เราเรียกกลุ่มของข้อมูลนี้ว่า บล็อกของข้อมูล (Block of Data) ซึ่งตัวอักษรตัวแรก และตัว

ถัดไปที่อยู่ในบล็อกเดียวกันจะไม่มีอะไรมาคั่นเหมือนอย่างแบบอะซิงโคนัส ที่ต้องใช้บิตเริ่มต้น และบิตสิ้นสุดคั่นทุกๆ ตัวอักษร แต่จะมีข้อมูลเริ่มต้น ซึ่งเป็นลักษณะของบิตพิเศษที่ส่งมาเพื่อให้รู้ว่านั้นคือจุดเริ่มต้นของกลุ่มตัวอักษรที่กำลังส่งเรียงกันเข้ามา เช่น อักขระซิง (SYN character) โดยที่อักขระซิงมีรูปแบบบิต คือ 00010110 ตัวอย่างของการส่งแสดงได้ดังรูปที่ 2.17



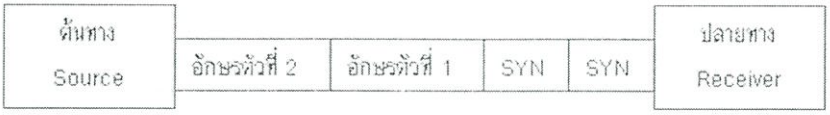
รูปที่ 2.17 การสื่อสารแบบซิงโคนัส

จากรูปที่ 2.17 เมื่อลายทางตรวจพบอักขระซิง หรือ 00010110 แล้วจะทราบได้ทันทีว่าบิตที่ตามมาคือ บิตตัวอักษรแต่ละตัว แต่การใช้อักขระซิงเพียงตัวเดียวอาจเกิดข้อผิดพลาดได้ เช่น ถ้าเราส่งตัวอักษร b และตัวอักษร a ติดต่อกันไป ซึ่งตัวอักษร b มีรูปแบบบิตคือ 01100010 และตัวอักษร a มีรูปแบบบิตคือ 01100001 การส่งจะแสดงได้ดังรูปที่ 2.18

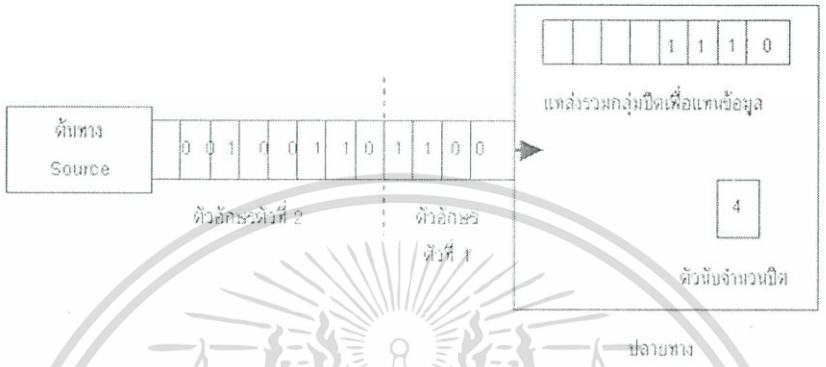


รูปที่ 2.18 ตัวอย่างการสื่อสารแบบซิงโคนัส

จะเห็นว่าเครื่องปลายทางจะตรวจพบอักขระซิงระหว่างบิตของตัวอักษร b และตัวอักษร a ทำให้เข้าใจว่าบิตต่อไปจะเป็นบิตของกลุ่มข้อมูล ซึ่งจะทำให้การรับข้อมูลนั้นเกิดผิดพลาดขึ้นได้ ดังนั้นจึงแก้ปัญหาด้วยการใช้อักขระซิง 2 ตัวต่อกันเป็นลักษณะของบิตพิเศษที่บอกให้ทราบว่าเป็นจุดเริ่มต้นบิตของกลุ่มข้อมูลตัวอย่างของการใช้อักขระซิง 2 ตัวในการสื่อสารแบบซิงโคนัส และการตัดแถวของบิตข้อมูลออกเป็นกลุ่มทีละ 8 บิต เพื่อแทนข้อมูลแสดงได้ดังรูปที่ 2.19



รูปที่ 2.19 ตัวอย่างการใช้อักขระซิง 2 ตัวในการสื่อสารแบบซิงโครนัส



รูปที่ 2.20 การตัดแถวของบิตออกเป็นกลุ่มๆ ละ 8 บิต

2.6 Image Processing

การประมวลผลภาพ (Image Processing) หมายถึงการนำภาพมาประมวลผลหรือคิดคำนวณด้วยคอมพิวเตอร์เพื่อให้ได้ข้อมูลที่ต้องการ ซึ่งค่าอินพุตที่เข้าไปนั้นเป็นภาพ หลังจากประมวลผลเสร็จสิ้นผลลัพธ์ที่ได้สามารถเป็นภาพ หรือค่าพารามิเตอร์ต่างๆ ที่เกี่ยวข้องกับภาพอย่างใดก็ได้

รูปที่ 2.21 หลักการทำงานของ Image Processing

การทำให้ภาพมีความคมชัดมากขึ้น การกำจัดสัญญาณรบกวนออกจากภาพการแบ่งส่วนของวัตถุที่สนใจออกมาจากภาพเพื่อนำภาพวัตถุที่ได้ไปวิเคราะห์หาข้อมูลเชิงปริมาณ เช่น ขนาด รูปร่าง และทิศ

ทางการเคลื่อนของวัตถุในภาพจากนั้นสามารถนำข้อมูลเชิงปริมาณเหล่านี้ไปวิเคราะห์ และสร้างเป็นระบบเพื่อใช้ประโยชน์ในงานด้านต่างๆ เช่น ระบบรู้จำลายนิ้วมือเพื่อตรวจสอบว่าภาพลายนิ้วมือที่มีอยู่นั้นเป็นของผู้ใด ระบบตรวจสอบคุณภาพของผลิตภัณฑ์ในกระบวนการผลิตของโรงงานอุตสาหกรรม ระบบคัดแยกเกรดหรือคุณภาพของพืชผลทางการเกษตรระบบอ่านรหัสไปรษณีย์อัตโนมัติเพื่อคัดแยกปลายทางของจดหมายที่มีจำนวนมากในแต่ละวัน โดยใช้ภาพถ่ายของรหัสไปรษณีย์ที่อยู่บนซองระบบเก็บข้อมูลรถที่เข้าและออกอาคารโดยใช้ภาพถ่ายของป้ายทะเบียนรถเพื่อประโยชน์ในด้านความปลอดภัยระบบดูแล และตรวจสอบสภาพการจราจรบนท้องถนนโดยการนับจำนวนรถบนท้องถนนในภาพถ่ายด้วยกล้องวงจรปิด ในแต่ละช่วงเวลาระบบรู้จำใบหน้าเพื่อเฝ้าระวังผู้ก่อการร้ายในอาคารสถานที่สำคัญๆ หรือในเขตคนเข้าเมืองเป็นต้น จะเห็นได้ว่าระบบเหล่านี้จำเป็นต้องมีการประมวลผลภาพจำนวนมาก และเป็นกระบวนการที่ต้องทำซ้ำๆ กันในรูปแบบเดิมเป็นส่วนใหญ่ ซึ่งงานในลักษณะเหล่านี้ หากให้มนุษย์วิเคราะห์เองมักต้องใช้เวลามาก และใช้แรงงานสูงอีกทั้งหากจำเป็นต้องวิเคราะห์ภาพเป็นจำนวนมาก ผู้วิเคราะห์ภาพเองอาจเกิดอาการล้า ส่งผลให้เกิดความผิดพลาดขึ้นได้ ดังนั้นคอมพิวเตอร์จึงมีบทบาทสำคัญในการทำหน้าที่เหล่านี้แทนมนุษย์ อีกทั้งเป็นที่ทราบโดยทั่วกันว่าคอมพิวเตอร์มีความสามารถในการคำนวณ และประมวลผลข้อมูลจำนวนมหาศาลได้ในเวลาอันสั้น จึงมีประโยชน์อย่างมากในการเพิ่มประสิทธิภาพการประมวลผลภาพ และวิเคราะห์ข้อมูลที่ได้จากภาพในระบบต่างๆ

2.6.1 OpenCV

OpenCV (Open Source Computer Vision Library) ได้สร้างมาเพื่อการใช้ร่วมกับคอมพิวเตอร์ในการประยุกต์ใช้กับการประมวลผลทางรูปภาพ โดยบริษัท Intel Corporation จำกัด มีมากกว่า 2500 คำสั่งคำสั่งเหล่านี้สามารถใช้ในการตรวจจับ และจัดหมวดหมู่ข้อมูลรูปภาพ และวีดีโอได้รวมทั้งการติดตามการเคลื่อนไหวของวัตถุแยกส่วนของโมเดล 3 มิติของวัตถุต่างๆ รวมภาพหลายๆ ภาพมาเป็นภาพเดียวค้นหาภาพที่คล้ายกันในฐานข้อมูลตัดแต่งภาพ และอื่นๆ มากมายใช้ได้บนระบบปฏิบัติการที่เป็น Linux และ Microsoft Windows และสามารถพัฒนาโปรแกรมได้หลากหลายภาษา ทั้งภาษา C++, C# และ Phyton

รูปที่ 2.22 Logo ของ OpenCV

2.6.2 รูปแบบสีแบบอาร์จีบี (RGB)

เป็นระบบสีพื้นฐานของคอมพิวเตอร์ที่ใช้ในการแสดงผล โดยจุดย่อยของภาพ (Pixel) จะประกอบด้วยค่าสี 3 ค่า คือ แดง (R) เขียว (G) และน้ำเงิน (B) การผสมสีทั้งสามนี้ด้วยค่าต่างๆ กัน จะก่อให้เกิดสีที่แตกต่างกัน โดยคอมพิวเตอร์จะเก็บค่าสีนี้แยกกัน โดยใช้ขนาดข้อมูล 1 ไบต์ต่อ 1 สี ทำให้ค่าของสีนั้นมีได้ 256 ระดับ และผสมได้สีทั้งหมด 16 ล้านสี สามารถแสดงโมเดลของสีได้ดังรูป 2.23

รูปที่ 2.23 สีแบบ RGB

2.6.3 รูปแบบสีแบบระดับสีเทา (Gray Scale)

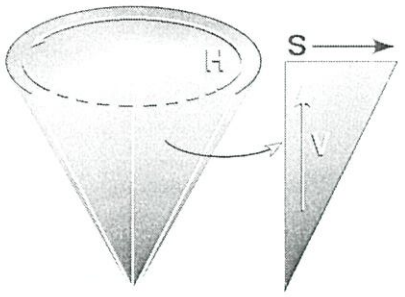
ภาพระดับสีเทา หรือ Gray Scale เป็นภาพที่ประกอบไปด้วยค่าของพิกเซลที่เป็นค่าเฉดสีของสีเทา ซึ่งได้จากระดับสีจากการเปลี่ยนสีจากสีดำไปเป็นสีขาว โดยจะถือสีขาวเป็นสีที่แก่ที่สุด และสีดำเป็นสีที่อ่อนที่สุด เนื่องจากค่าของพิกเซลที่เป็นสีขาวจะมีค่ามากกว่าค่าของสีดำ สิ่งที่ภาพระดับสีเทาแตกต่างกับภาพขาว-ดำคือ ในภาพขาว-ดำ จะเก็บข้อมูลโดยมีอยู่สองสีคือ สีขาว และสีดำ แต่ในภาพระดับสีเทาจะมีความละเอียดมากกว่า กล่าวคือจะเก็บค่าของเฉดสีเทาที่อยู่ระหว่างการเปลี่ยนจากสีดำเป็นสีขาว

การนำตัวเลขมาแทนค่าของพิกเซลในภาพระดับสีเทานั้น ใช้เป็นเปอร์เซ็นต์คือ 0% (สีดำ) ถึง 100% (สีขาว) แต่เมื่อนำมาใช้งานร่วมกับเครื่องพิมพ์จะมีการทำงานที่ตรงกันข้าม กล่าวคือ จะต้องทำการกลับค่าของเปอร์เซ็นต์จึงจะพิมพ์ได้ตรงตามที่ต้องการ เนื่องจากการอ่านค่าของเครื่องพิมพ์คือ 0% คือ ไม่ปล่อยหมึกออกมา ซึ่งหมายความว่าส่วนนั้นจะเป็นสีขาว ส่วน 100% เครื่องพิมพ์จะปล่อยหมึกออกมา ซึ่งนั่นคือสีดำ ในทางคอมพิวเตอร์ เดิมจะมีการเก็บในลักษณะของเลขฐานสองจำนวนสี่บิต สามารถเก็บความแตกต่างได้ 16 ระดับ แต่ในปัจจุบันนี้ได้มีการเพิ่มข้อมูลที่เก็บโดยเพิ่มเป็นเลขฐานสองจำนวน 8 บิต (0-255) ซึ่งจะสามารถเก็บค่าความแตกต่างได้ 256 ระดับ แสดงได้ดังรูป 2.24

รูปที่ 2.24 สีแบบระดับสีเทา

2.6.4 ระบบสี HSV

HSV เป็นระบบสีที่อาศัยหลักการใช้ ค่าของสี (Hue, H) ค่าความเข้มของเนื้อสี หรือค่าความบริสุทธิ์ของสี (Saturation, S) และ ค่าความสว่างของสี (Value, V) ซึ่งเสนอโดย A.R. Smith (1978) โดย Hue คือ ค่าสีหลัก เช่น สีแดง สีเขียว สำน้ำเงิน มีค่าอยู่ระหว่าง 0 ถึง 255 ถ้าเกิดค่าของสีมีค่าเท่ากับ 0 จะแทนให้เป็นสีแดง และเมื่อค่าของสีมีค่าเพิ่มขึ้นเรื่อยๆ สีก็จะเปลี่ยนไปตามความถี่สเปกตรัมของสีจนถึง 256 แล้วกลับมาเป็นสีแดงเช่นเดิมอีกครั้ง และความสามารถแทนให้อยู่ในรูปองศาได้ คือ สีแดง มีค่าเท่ากับ 0 องศา สีเขียว มีค่าเท่ากับ 120 องศา และสีน้ำเงิน มีค่าเท่ากับ 240 องศา สำหรับค่าความเข้มของเนื้อสีหรือค่าความเข้มของเนื้อสี เมื่อมีค่าเพิ่มขึ้น สีจะมีความเข้มมากขึ้นเรื่อยๆ และสุดท้ายค่าความสว่างของสี เมื่อมีค่าเพิ่มมากขึ้นภาพจะมีความสว่างเพิ่มขึ้น การบอกสีในระบบ HSV สามารถแสดงได้ดังรูปที่ 2.25



รูปที่ 2.25 การบอกค่าสีในระดับ HSV

จะสังเกตว่าการบอกสีในระบบ HSV จะมีความคงทนต่อการเปลี่ยนแปลงของแสงในสภาพแวดล้อมมากกว่าการใช้สีในระบบ RGB เนื่องจากหากภาพที่ความสว่างมาก ค่าของสี และค่าความเข้มของเนื้อสีจะไม่มีการเปลี่ยนแปลง มีเพียงค่าความสว่างของสีเปลี่ยนแปลงเพียงค่าเดียวเท่านั้น

2.6.5 การแปลงภาพแบบ RGB ไปสู่ ภาพแบบระดับสีเทา

ในการทำงานของไลบรารี่โอเพนซีวี เมื่อต้องการจะทำการเปลี่ยนรูปแบบสีของภาพจาก RGB ไปสู่ภาพระดับสีเทา จะมีการเรียกการทำงานของฟังก์ชัน cvCvtColor โดยการทำงานในฟังก์ชันนี้จะอ่านค่าทีละพิกเซลแล้วทำการคำนวณเพื่อให้ได้ค่าใหม่ที่อยู่ในช่วง 0 - 255 ซึ่งการคำนวณจะเป็นดังสมการที่ (2.7)

$$Y=(0.299 \times R) + (0.587 \times G) + (0.114 \times B) \quad (2.7)$$

โดยที่ Y คือ ค่าผลลัพธ์ในช่วง 0 - 255 ตัวแปร R, G และ B คือ ค่าของสีแดง สีเขียว และสีน้ำเงินตามลำดับ

2.6.6 การแปลงระบบสี RGB เป็น HSV

เนื่องจากภาพที่นำมาใช้ในการประมวลผลนั้นเก็บค่าสีในระบบ RGB ซึ่งค่าสีที่อยู่ในระบบ RGB นั้นจะประกอบไปด้วยค่าสี ค่าแสง และค่าความสว่าง ซึ่งจะมีความซับซ้อนในการแยกแยะสี เนื่องจากมีค่าแสงและค่าความสว่างผสมอยู่ด้วย ดังนั้นในงานวิจัยนี้จึงได้ทำการแปลงระบบสีแบบ RGB ให้เป็นแบบ HSV เพื่อให้สามารถแสดงความสัมพันธ์ได้ดังสมการที่ (2.8), สมการที่ (2.9) และสมการที่ (2.10)

โดยที่

R G B แทนค่าของสีในระบบ RGB มีค่าระหว่าง 0.0 – 1.0

H S V แทนค่าของสีในระบบ HSV

max = ค่าสูงสุดใน (R, G, B)

min = ค่าต่ำสุดใน (R, G, B)

และ

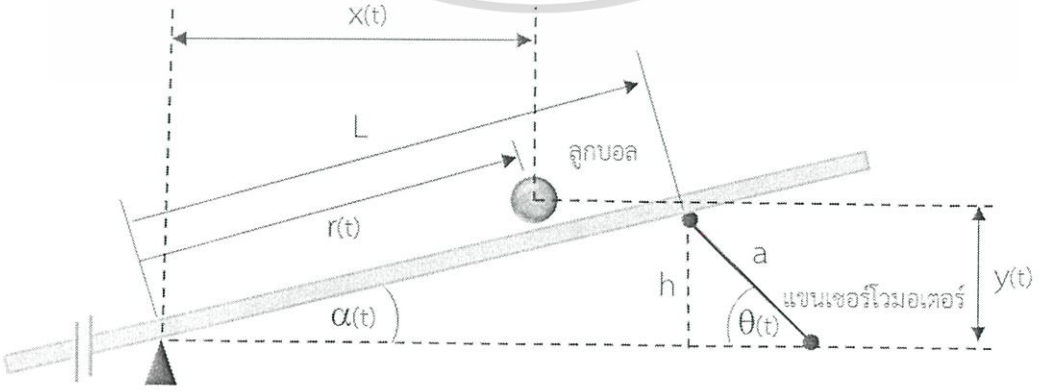
$$H \begin{cases} 60\times\dfrac{G-B}{max-min} & \text{เมื่อ } max = R \\ 60\times\dfrac{B-R}{max-min}+120 & \text{เมื่อ } max = G \\ 60\times\dfrac{R-G}{max-min}+240 & \text{เมื่อ } max = B \end{cases} \quad (2.8)$$

$$S = \frac{max-min}{max}\times 100 \quad (2.9)$$

$$V = max\times 100 \quad (2.10)$$

2.7 แบบจำลองทางคณิตศาสตร์ของระบบควบคุมตำแหน่งลูกบอลบนระนาบ

ระบบควบคุมตำแหน่งลูกบอลบนระนาบประกอบไปด้วย ลูกบอล และแผ่นระนาบ โดยลูกบอลสามารถเคลื่อนที่ไปมาบนแผ่นระนาบได้ 2 ทิศทาง คือ ในแนวตั้ง (แกน Y) และแนวนอน (แกน X) โดยตำแหน่งลูกบอลจะมีความสัมพันธ์กับมุมของแผ่นระนาบ ในการเขียนแบบจำลองทางคณิตศาสตร์จะพิจารณาลักษณะการเคลื่อนที่ในแนวแกนเดียว เนื่องจากการเคลื่อนที่ในของทั้งสองแนวแกนมีลักษณะเหมือนกันทุกประการ



รูปที่ 2.26 จำลองการเคลื่อนที่ของลูกบอลบนแผ่นระนาบ

การพิจารณาระบบการเคลื่อนที่นี้ จะถือว่าลูกบอลกลิ้งด้วยไม่มีการไถลละไม่มีแรงเสียดทานในการกลิ้งระหว่างลูกบอล และแผ่นระนาบ ตัวแปร และค่าคงตัวต่างๆ แสดงดังตารางที่ 2.3

ตารางที่ 2.3 ตัวแปร และค่าคงตัวในแบบจำลองระบบควบคุมตำแหน่งลูกบอลบนระนาบ

ชื่อตัวแปร และค่าคงตัว	ค่า	หน่วย
1. มวลลูกบอล(m)	0.006	kg
2. รัศมีลูกบอล	0.0055	m
3. รัศมีแขนเซอร์โวมอเตอร์	0.015	m
4. ความเร่งเนื่องจากแรงโน้มถ่วงโลก (g)	9.81	m/s^2
5. โมเมนต์ความเฉื่อยของลูกบอล (J)	7.26×10^{-8}	$kg\cdot m^2$
6. ความยาวของแขนเซอร์โวมอเตอร์ (a)	0.015	m
7.ระยะบนระนาบจากจุดหมุนกับปลายแขนเซอร์โซมอเตอร์	≈0.145	m
8. ตำแหน่งของลูกบอล (r)	-	m
9. มุมของแผ่นระนาบ (α)	-	เรเดียน
10. มุมของเซอร์โวมอเตอร์ (θ)	-	เรเดียน

จากระบบข้างต้น สามารถเขียนเป็นสมการของลากรางจ์ (Lagragian) ได้ดังนี้

$$L = K - U \tag{2.11}$$

$$K = \frac{1}{2}J\dot{\theta}_b + \frac{1}{2}mv_b^2 \tag{2.12}$$

เนื่องจาก

$$\dot{\theta}_b = \frac{\dot{r}}{d} \tag{2.13}$$

และ

$$v_b^2 = x_b^2 + y_b^2 \tag{2.14}$$

$$x = r\cos\alpha \tag{2.15}$$

$$\dot{x} = \dot{r}\cos\alpha - r\dot{\alpha}\sin\alpha \tag{2.16}$$

$$\dot{x}^2 = \dot{r}^2\cos^2\alpha - 2r\dot{r}\sin\alpha\cos\alpha + r^2\dot{\alpha}^2\sin^2\alpha \tag{2.17}$$

$$y = r\sin\alpha \tag{2.18}$$

$$\dot{y} = \dot{r}\sin\alpha + r\dot{\alpha}\cos\alpha \tag{2.19}$$

$$\dot{y}^2 = \dot{r}^2\sin^2\alpha + 2r\dot{r}\sin\alpha\cos\alpha + r^2\dot{\alpha}^2\cos^2\alpha \tag{2.20}$$

แทนสมการที่ (2.16) และสมการที่ (2.20) ลงในสมการที่ (2.14) จะได้

$$v_b^2 = \dot{r}^2 + r^2\dot{\alpha}^2 \tag{2.21}$$

แทนสมการที่ (2.13) และสมการที่ (2.21) ลงในสมการที่ (2.12) จะได้

$$K = \frac{1}{2}\left(\frac{J}{d^2} + m\right)\dot{r}^2 + \frac{1}{2}mr^2\dot{\alpha}^2 \tag{2.22}$$

พลังงานศักย์ของลูกบอลมีค่าเท่ากับ

$$U = mgr\sin\alpha \tag{2.23}$$

แทนสมการที่ (2.22) และสมการที่ (2.23) ลงในสมการที่ (2.11) จะได้

$$L = \frac{1}{2}\left(\frac{J}{d^2} + m\right)\dot{r}^2 + \frac{1}{2}mr^2\dot{\alpha}^2 - mgr\sin\alpha \tag{2.24}$$

ระบบควบคุมตำแหน่งลูกบอลบนระนาบเป็นระบบแบบ 1 Degree of Freedom จากสมการออยเลอร์-ลากรางจ์ หรือ สมการของลากรางจ์ที่ใช้อธิบายการเคลื่อนที่ต่างๆ

$$\frac{d}{dt}\left(\frac{\partial L}{\partial r}\right) - \frac{\partial L}{\partial r} = 0 \tag{2.25}$$

แทนสมการที่ (2.24) ในสมการที่ (2.25) จะได้

$$\left(\frac{J}{d^2} + m\right)\ddot{r} + mg\sin\alpha - mr\dot{\alpha}^2 = 0 \tag{2.26}$$

ทำการประมาณเชิงเส้นสมการที่ (2.26) โดยพิจารณาที่ αเข้าใกล้ 0 จะได้

$$\left(\frac{J}{d^2} + m\right)\ddot{r} + mg\alpha = 0 \tag{2.27}$$

โดยความสัมพันธ์ระหว่างมุมของระนาบกับมุมของเซอร์โวมอเตอร์คือ

$$\sin\alpha = \frac{h}{L} \tag{2.28}$$

$$\sin\alpha = \frac{a\sin\theta}{L} \tag{2.29}$$

เมื่อพิจารณาที่ α และ θ มีขนาดเข้าใกล้ศูนย์ จะได้

$$\alpha = \frac{a}{L}\theta \tag{2.30}$$

แทนค่าสมการที่ (2.30) ในสมการที่ (2.17) จะได้

$$\left(\frac{J}{d^2}+m\right)\ddot{r} + \frac{mga}{L}\theta = 0 \tag{2.31}$$

ทำการแปลงเป็นลาปาซ

$$\left(\frac{J}{d^2}+m\right)R(s)s^2 + \frac{mga}{L}\Theta(s) = 0 \tag{2.32}$$

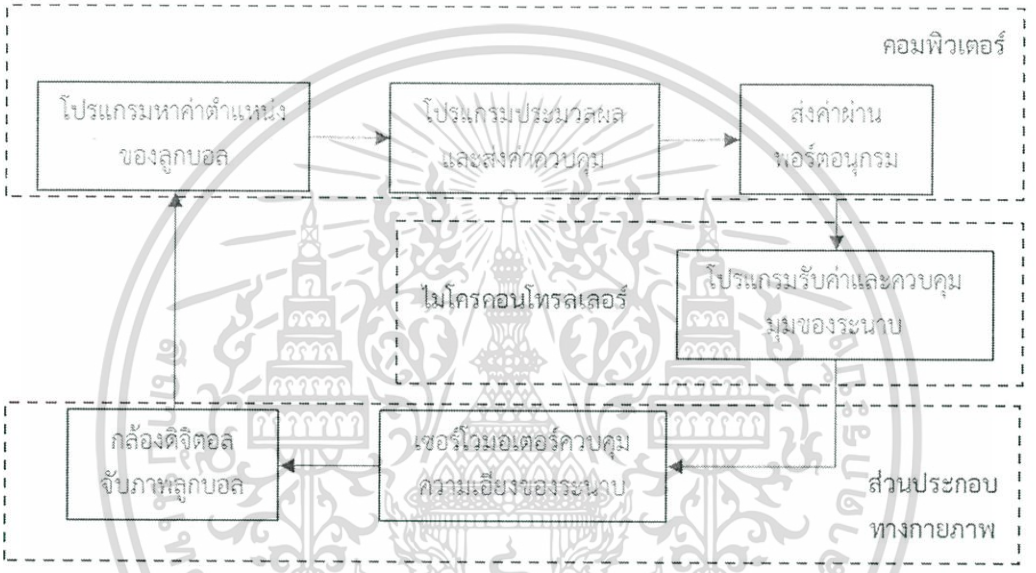
ซึ่งเขียนในรูปของฟังก์ชันถ่ายโอนได้เป็น

$$P(s) = \frac{R(s)}{\Theta(s)} = -\frac{mga}{L\cdot\left(\frac{J}{d^2}+m\right)}\frac{1}{s^2}\left[\frac{m}{rad}\right] \tag{2.33}$$

พบว่า ตำแหน่งของลูกบอลมีความสัมพันธ์กับค่ามุมของแผ่นระนาบที่เปลี่ยนไป ซึ่งการควบคุมมุมของแผ่นระนาบจะทำผ่านการหมุนเซอร์โวมอเตอร์ โดยความสัมพันธ์ของเซอร์โวมอเตอร์ และมุมของแผ่นระนาบกับมุมนั้นจะกล่าวในบทถัดไป

บทที่ 3
หลักการออกแบบ

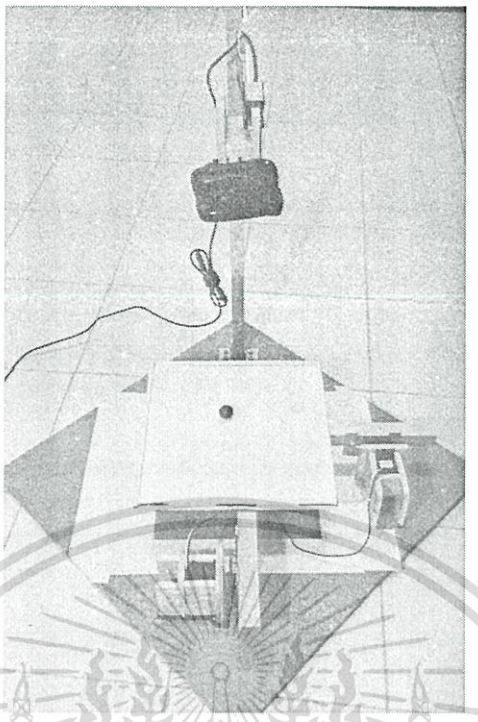
ระบบควบคุมตำแหน่งลูกบอลบนระนาบนั้น ประกอบด้วยส่วนสำคัญทั้งหมด 3 ระบบ การออกแบบรายละเอียดย่อยในแต่ละระบบนั้น จำเป็นต้องเข้าใจถึงภาพรวบของระบบควบคุมทั้งหมดก่อน รวมถึงความสัมพันธ์ระหว่างองค์ประกอบต่างๆ ของแต่ละระบบด้วย



รูปที่ 3.1 การทำงานของระบบควบคุมตำแหน่งบนระนาบ

ซึ่งหลักการทำงานของระบบควบคุมตำแหน่งลูกบอลบนระนาบในภาพรวม สามารถอธิบายได้ดังรูปที่ 3.1 ซึ่งจะมีประกอบด้วยส่วนสำคัญดังนี้

- โครงสร้างทางกายภาพ
- คอมพิวเตอร์
- ไมโครคอนโทรลเลอร์



รูปที่ 3.2 โครงสร้างของระบบควบคุมตำแหน่งลูกบอลบนระนาบ

3.1 โครงสร้างทางกายภาพของระบบ

โครงสร้างทางกายภาพของระบบควบคุมตำแหน่งลูกบอลบนระนาบนั้น ประกอบด้วยส่วนของแผ่นระนาบและฐานที่ทำจากไม้อัดหนา 9 มม. ติดสติกเกอร์สีขาวทับที่ผิวเพื่อเพิ่มการความสว่างของแสง ซึ่งจะช่วยให้กล้องทำงานได้เร็วขึ้น มีแท่งเหล็กปลายแหลมมนสำหรับรองแผ่นระนาบเพื่อเป็นจุดศูนย์กลางในการหมุน ใช้ดีซีเซอร์โวมอเตอร์ 2 ตัวในการควบคุมความเอียงของระนาบ ดังแสดงไว้ในรูปที่ 3.2

3.1.1 ลูกบอล

การเลือกใช้ลูกบอลนั้นจำเป็นต้องเลือกให้เหมาะกับการทำงานของระบบ โดยเลือกใช้ลูกบอลที่มีผิวเรียบ และแข็ง เพื่อให้การกลิ้งของลูกบอลเป็นไปตามสมการถ่ายโอนที่ออกแบบไว้มากที่สุด น้ำหนักของลูกบอลประมาณ 6 กรัม การใช้ลูกบอลที่มีน้ำหนักเบาจะลดความหน่วงในการเคลื่อน ซึ่งจะช่วยให้ลูกบอลเปลี่ยนความเร็ว และทิศในการเคลื่อนที่ได้เร็วขึ้น และทำให้เซอร์โวมอเตอร์รับภาระแรงบิดน้อยลง สีของลูกบอลเลือกใช้เป็นสีดำด้าน เพื่อให้มีค่าความอิ่มตัวของแสงแตกต่างจากพื้นหลังที่เป็นสีขาวมากที่สุด ซึ่งจะทำให้ง่ายในการประมวลผลด้วยภาพ ลูกบอลมีเส้นผ่าศูนย์กลาง 11 มม.

3.1.2 กล้อง

กล้องเป็นอุปกรณ์สำหรับจับภาพการเคลื่อนที่ของลูกเพื่อหาตำแหน่ง ถือเป็นเซนเซอร์ของระบบในการทำให้ระบบกลายเป็นระบบวงปิด กล้องที่ใช้เป็นกล้องดิจิตอลเว็บแคมทั่วไป โดยหลักการเลือกใช้จะพิจารณาคุณสมบัติสำคัญ 2 ประการ คือ Frame Rate และ Resolution โดยค่า Frame Rate จะบอกถึงจำนวนภาพที่กล้องสามารถบันทึกได้ภายใน 1 วินาที หากค่า Frame Rate สูงจะช่วยให้สามารถสุ่มค่าข้อมูลได้สูงขึ้นนั่นเอง ส่วนค่า Resolution จะบอกถึงความละเอียดของภาพที่บันทึก ซึ่งจะส่งผลต่อความละเอียดในการหาตำแหน่งของวัตถุ ค่าความละเอียดที่สูงจะทำให้ความคลาดเคลื่อนมีค่าลดลง โดยกล้องที่ใช้นั้น มีค่า Frame Rate เท่ากับ 30 fps (Frame Per Second) และมีค่า Resolution เท่ากับ 640x480 Pixels (ที่อัตราส่วนภาพ 4 : 3)

3.1.3 เซอร์โวมอเตอร์

เซอร์โวมอเตอร์เป็นอุปกรณ์สำหรับควบคุมองศาของแผ่นระนาบ เพื่อให้ลูกบอลเคลื่อนที่ไปยังตำแหน่งที่ต้องการ โดยเซอร์โวมอเตอร์เป็นมอเตอร์ที่สามารถควบคุมองศาการหมุนได้ โดยการปรับความกว้างของพัลส์ของสัญญาณควบคุมที่จ่ายให้เซอร์โวมอเตอร์ รวมทั้งมีวงจรขับเคลื่อนอยู่แล้วภายใน จึงไม่จำเป็นต้องสร้างวงจรขับมอเตอร์เอง สิ่งที่ต้องพิจารณาในการเลือกใช้คือ ความเร็วในการหมุน และทอร์คของเซอร์โวมอเตอร์ โดยเซอร์โวมอเตอร์จะต้องมีความเร็วที่มากพอที่จะตอบสนองการทำงานในแต่ละรอบได้ และต้องมีค่าทอร์คมากพอที่ยกระนาบที่ลูกบอลกลิ้งอยู่ให้เอียงได้ เซอร์โวมอเตอร์ที่เลือกใช้ในที่นี้เป็นเซอร์โวมอเตอร์แบบกระแสตรงขนาด 4.8 โวลต์ ถึง 6.0 โวลต์ ความเร็วในการการหมุนอยู่ที่ประมาณ 0.005 วินาที/องศา (ที่แรงดัน 4.8 โวลต์) และมีค่าทอร์คเท่ากับ 0.024 กิโลกรัม/เมตร

3.2 คอมพิวเตอร์

คอมพิวเตอร์ถือเป็นส่วนประมวลผลหลักของระบบนี้ โดยภาพที่บันทึกได้จากกล้องดิจิตอลจะถูกส่งมายังคอมพิวเตอร์เพื่อใช้หาตำแหน่งของลูกบอล จากนั้นจะทำการคำนวณหาองศาของเซอร์โวมอเตอร์ในการควบคุมความเอียงของแผ่นระนาบ และส่งค่าไปยังผ่านพอร์ตอนุกรมไปยังไมโครคอนโทรลเลอร์อีกที ในส่วนของการเขียนโปรแกรมนั้น ต้องคำนึงรูปแบบการประมวลที่เป็นแบบดิจิตอล หรือก็คือ การดำเนินการบนเวลาดีสครีต ซึ่งต่างจากระบบในความเป็นจริงที่เป็นระบบเวลาต่อเนื่อง เพื่อให้ระบบควบคุมทำงานได้อย่างมีประสิทธิภาพ โปรแกรมจำเป็นที่จะต้องทำงานได้เร็วพอในแต่ละรอบการทำงาน เพื่อที่จะ

สามารถสุ่มค่าได้มากพอนั่นเอง โดยในโครงงานนี้โปรแกรมที่ใช้เขียนขึ้นด้วยภาษา C++ ซึ่งเป็นภาษาที่ใช้งานง่าย รวมทั้งยังสามารถใช้งานร่วมกับไลบลารีฟังก์ชันของ OpenCV ในการประมวลด้วยภาพได้ด้วย โปรแกรมจะประกอบด้วยส่วนการทำงานหลักๆ 4 ส่วน ดังต่อนี้

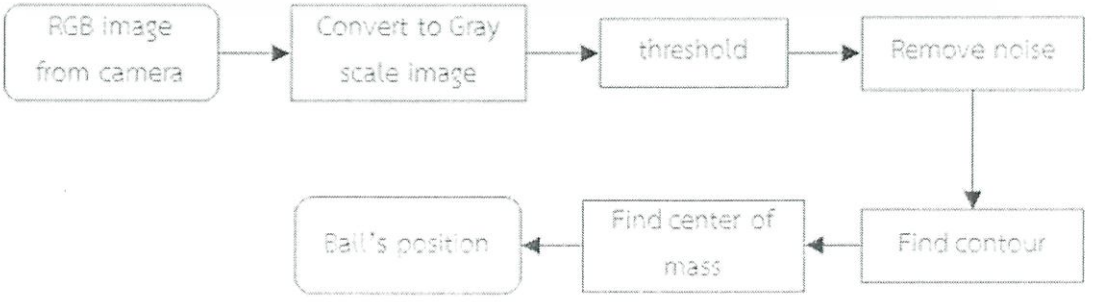
- การประมวลผลภาพ
- ตัวควบคุม PID
- การคำนวณมุมของเซอร์โวมอเตอร์
- การส่งค่าผ่านพอร์ตอนุกรม

3.2.1 การประมวลผลภาพ

ในโครงงานนี้จะใช้กล้องดิจิตอลเป็นเซนเซอร์ในการตรวจจับตำแหน่งของลูกบอล เพื่อใช้ในการป้อนกลับไป ซึ่งหลักการทำงานของกล้องดิจิตอลนั้นได้กล่าวไว้แล้วในบทที่ 2

ภาพที่ใช้ในการตรวจหาตำแหน่งของลูกบอลที่ใช้ในโครงงานนี้จะเป็นภาพในปริภูมิสีแบบ Gray Scale ซึ่งเป็นเมทริกซ์ขนาด $M \times N$ จำนวน 1 เมทริกซ์ โดยแต่ละตำแหน่งในเมทริกซ์ จะเก็บค่าความเข้มแสงของภาพ โดยมีระดับความละเอียดขึ้นอยู่กับจำนวนบิตของที่กำหนด เช่น หากกำหนดเป็น 8 บิท จะเก็บความความเข้มแสงได้ 256 ระดับ โดยสีดำคือ 0 และสีขาวคือ 255 เป็นต้น จากระบบทางกายภาพที่ได้กล่าวไปข้างต้น ภาพที่จะได้จากกล้องดิจิตอลจะเป็นภาพลูกบอลสีดำ (ความเข้มแสงน้อย) ที่มีพื้นหลังเป็นสีขาว (ความเข้มแสงมาก) ดังนั้นจึงสามารถระบุตำแหน่งของลูกบอลได้จากตำแหน่งของกลุ่มข้อมูลที่มีความความเข้มแสงน้อยนั่นเอง

การหาตำแหน่งของกลุ่มข้อมูลที่ต้องการนั้น จะใช้ฟังก์ชันต่างๆ ของ OpenCV ในการประมวลผล ซึ่งขั้นตอนในการประมวลภาพได้แสดงไว้ดังรูปที่ 3.3



รูปที่ 3.3 ขั้นตอนการประมวลภาพเพื่อระบุตำแหน่งของลูกบอลบนระนาบ

โดยค่าที่ได้ออกมาจะเป็นค่าตำแหน่งของพิกเซลบนภาพ ส่วนฟังก์ชันต่างๆ และการใช้งานได้แสดงไว้ในภาคผนวก

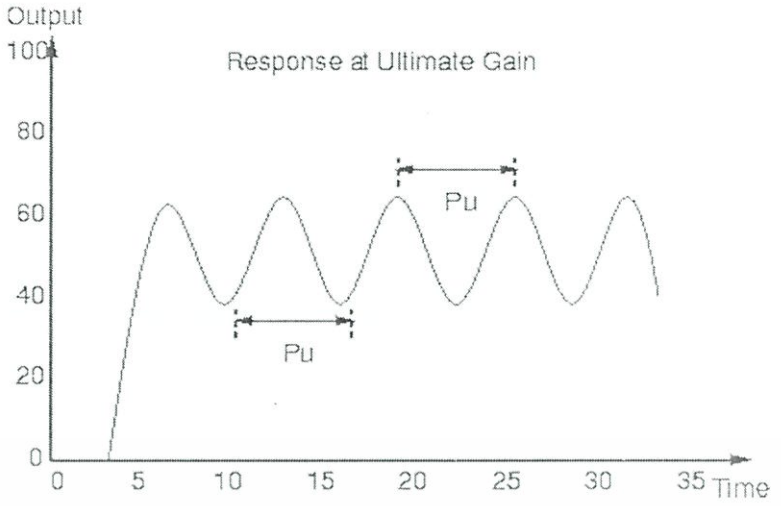
3.2.2 ตัวควบคุม PID

เนื่องจากระบบควบคุมตำแหน่งลูกบอลบนระนาบนั้น หากทำงานเป็นระบบวงเปิดจะไม่สามารถควบคุมตำแหน่งของลูกบอลได้เลย แต่ถึงจะให้ระบบทำงานเป็นระบบวงปิด (โดยไม่เพิ่มตัวควบคุม) ระบบก็ยังไม่สามรถทำงานได้ตามที่ต้องการ ดังนั้นจึงจำเป็นต้องมีตัวควบคุมเข้ามาเพื่อทำให้ระบบวงปิดสามารถทำงานได้ตามเป้าหมาย และหนึ่งในตัวควบคุมที่นิยมใช้กันมากที่สุดก็คือ ตัวควบคุมแบบ PID เนื่องจากมีการกำหนดค่าพารามิเตอร์ต่างๆ ที่ไม่ซับซ้อน เข้าใจง่าย และมีประสิทธิภาพสูง

การปรับค่าตัวควบคุม PID นั้นมีด้วยกันหลายวิธี วิธีหนึ่งที่ใช้กันอย่างแพร่คือ การปรับค่าแบบ Ultimate Method

3.2.2.1 Ultimate Method

วิธีการปรับค่าของตัวควบคุมแบบ Ultimate Method นั้นคิดค้นขึ้นโดย Ziegler และ Nicoles ในปี 1942 ที่เรียกว่า Ultimate เพราะว่าวิธีนี้จะใช้ Ultimate Gain ซึ่งเป็นค่า Gain สูงสุดก่อนที่ระบบจะไม่เสถียร (หรือก็คืออยู่ในสภาวะเสถียรวิกฤต) และ Ultimate Period ซึ่งเป็นคาบเวลาสุดท้ายของผลตอบสนองของระบบก่อนที่จะเข้าสู่สภาวะไม่เสถียร (นั่นคือ คาบเวลาของผลตอบสนองของระบบภายใต้ Gain สูงสุดนั่นเอง) มาใช้หาค่าต่างๆ ของตัวควบคุม โดยรูปที่ 3.4 แสดงถึงผลตอบสนองของระบบวงปิด ที่มีการควบคุมแบบ Proportional โดยที่ค่า K_u คือ ค่า Ultimate Gain ที่ทำให้เอาต์พุตของระบบเกิดการ Oscillate อย่างต่อเนื่อง และคาบเวลาของผลตอบสนองนี้คือ P_u (Ultimate Period)



รูปที่ 3.4 กราฟผลตอบสนองของระบบวงปิดเมื่อใช้ Ultimate Gain

Ziegler และ Nicoles ได้กำหนดความสัมพันธ์ในการปรับแต่งค่าต่างๆ ของตัวควบคุมไว้ดังตารางที่ 3.1

ตารางที่ 3.1 การกำหนดค่าต่างๆ ของตัวควบคุมจากวิธีการของผลตอบสนองของระบบวงปิดด้วยวิธี Ultimate Method

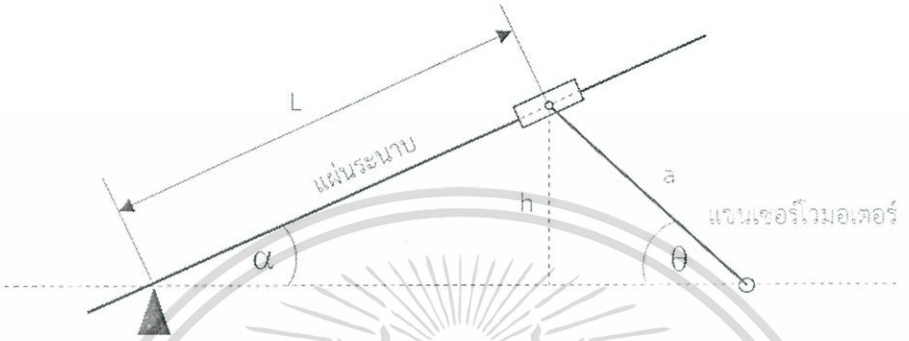
ชนิดของตัวควบคุม	ค่าที่กำหนด		
	K_p	T_i	T_d
P	0.50 K_u	-	-
PI	0.45 K_u	$P_u/1.2$	-
PD	0.60 K_u	-	$P_u/8.0$
PID	0.60 K_u	0.50 P_u	$P_u/1.2$

โดยผลตอบสนองที่เกิดจากตัวควบคุมที่ปรับด้วยวิธี Ultimate Method จะมีลักษณะส่ายกลับไปกลับมา แต่จะมีค่าพุ่งเกิน (Overshoot) ลดลงทุกครั้ง โดยจะลดลงเหลือประมาณ 1 ใน 4 ของค่าพุ่งเกินก่อนหน้า และจะลู่เข้าสู่ค่าเป้าหมายในที่สุด การปรับตัวควบคุมแบบ PID ด้วยวิธีนี้อาจไม่ได้ผลตอบสนองที่มีประสิทธิภาพตามที่ต้องการ ดังนั้นเพื่อให้ได้ได้ผลตอบสนองที่น่าพอใจ จึงจำเป็นตองทดลองปรับด้วย

ตัวเองไปซ้ำไปเรื่อยๆ จนกว่าจะได้ผลตอบสนองที่มีประสิทธิภาพตามต้องการ Ultimate Method เป็นเพียงวิธีที่ทำให้สามารถประมาณค่าตัวแปรของตัวควบคุมที่สามารถทำให้ระบบมีเสถียรภาพเท่านั้น

3.2.3 การคำนวณมุมของเซอร์โวมอเตอร์

มุมของระนาบ และมุมของเซอร์โวมอเตอร์ มีความสัมพันธ์กันดังรูปที่ 3.5



รูปที่ 3.5 ความสัมพันธ์ระหว่างมุมของระนาบ และมุมของเซอร์โวมอเตอร์

เนื่องจากระบบนี้จำกัดให้มุมของระนาบมีค่าไม่เกิน ±2 องศา ส่งผลให้ค่า L มีการเปลี่ยนแปลงน้อยมากๆ จึงสามารถประมาณให้ค่า L เป็นค่าคงตัวได้ จากรูปที่ 3.5 สามารถหาค่ามุมของเซอร์ได้ดังนี้

$$h = L\sin\alpha \tag{3.1}$$

$$\theta = \sin^{-1}\left(\frac{h}{a}\right) \tag{3.2}$$

แทนค่าสมการที่ (3.2) ในสมการที่ (3.1)

$$\theta = \sin^{-1}\left(\frac{L\sin\alpha}{a}\right) \tag{3.3}$$

โดยในระบบจริง ตัวแปร L มีค่าเท่ากับ 145 มม. เมื่อลองแทนค่า α เท่ากับ ±2 ลงในสมการที่ (3.3) จะได้ว่าเซอร์โวมอเตอร์จะหมุนด้วยมุมไม่เกิน ±20 องศา

3.2.4 การส่งค่าผ่านพอร์ตอนุกรม

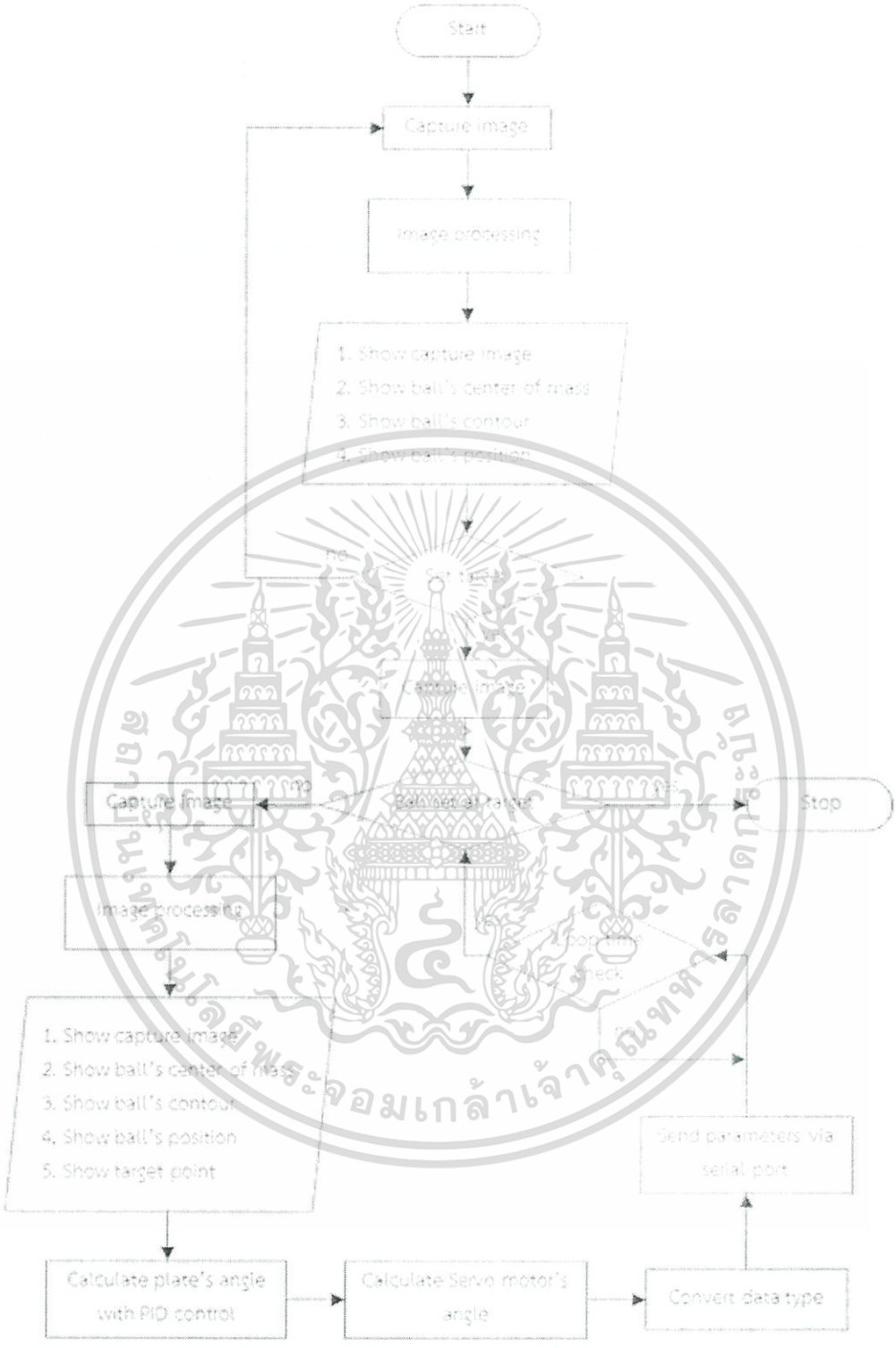
คอมพิวเตอร์ และไมโครคอนโทรลเลอร์จะสื่อสารกันผ่านพอร์ตอนุกรม ซึ่งในที่นี้คือ พอร์ตแบบ USB เนื่องจากส่วนของไมโครคอนโทรลเลอร์ที่ใช้นั้นจะรับค่าจากพอร์ตอนุกรมในรูปของข้อมูลชนิดตัวอักษร

(Character) เท่านั้น ดังนั้นในส่วนของโปรแกรมคอมพิวเตอร์จะต้องแปลงชนิดข้อมูลที่จะส่งเสียก่อน ซึ่งค่ามุมของเซอร์โวมอเตอร์นั้นเป็นชนิดตัวเลขจำนวนเต็ม (Integer) โดยจะถือให้เลข 1 หลักเป็น 1 ตัวอักษร

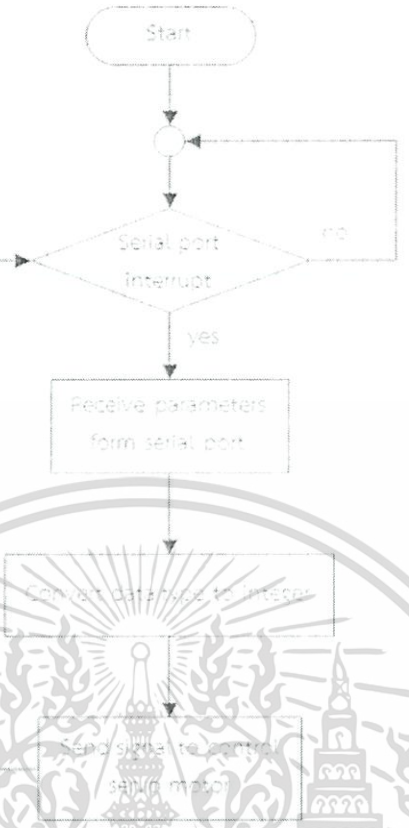
3.3 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์เป็นอุปกรณ์สำหรับสร้างสัญญาณในการสั่งงานเซอร์โวมอเตอร์เพื่อใช้ควบคุมองศาของระนาบ หน้าที่หลักมีเพียงแค่รับค่าจากคอมพิวเตอร์ผ่านพอร์ตอนุกรม จากนั้นจึงส่งสัญญาณพัลซ์ไปยังเซอร์โวมอเตอร์

โฟล์วชาร์ตแสดงการทำงานของส่วนคอมพิวเตอร์ และส่วนไมโครคอนโทรลเลอร์ได้แสดงไว้ดังรูปที่ 3.6 และรูปที่ 3.7



รูปที่ 3.6 โฟล์วชาร์ตการทำงานของส่วนโปรแกรมคอมพิวเตอร์



รูปที่ 3.7 โฟล์วชาร์ตการทำงานของส่วนไมโครคอนโทรลเลอร์

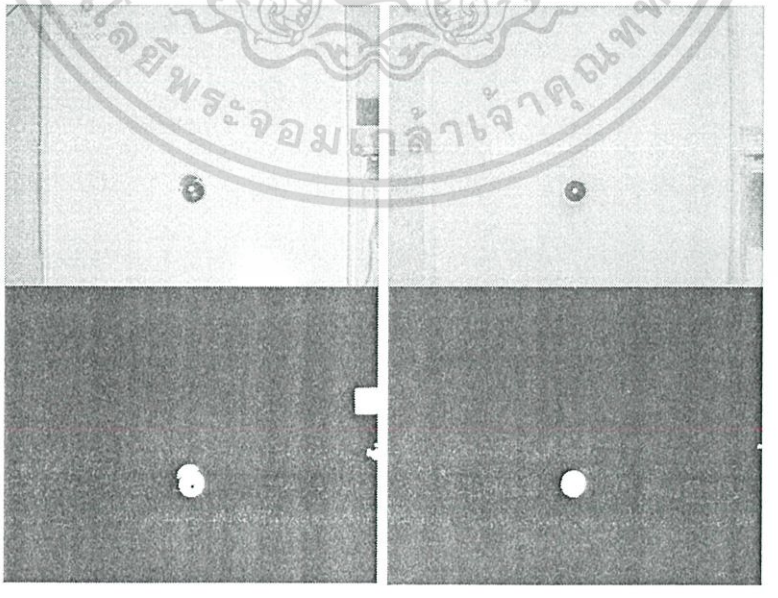
บทที่ 4
ผลการทดลอง

บทนี้จะเป็นการกล่าวถึงการทดลองต่างๆ ของระบบควบคุมตำแหน่งลูกบอลบนระนาบ โดยส่วนหลักจะแบ่งออกเป็น 2 ส่วนด้วยกัน ได้แก่ การทดลองระบุตำแหน่งลูกบอลด้วยกล้อง การควบคุมตำแหน่งลูกบอลด้วยตัวควบคุม PID โดยแต่ละส่วนมีรายละเอียดการทดลองดังนี้

4.1 การระบุตำแหน่งลูกบอลด้วยกล้อง

การทดลองในส่วนนี้จะเป็นการใช้กล้องจับภาพลูกบอลบนระนาบ เพื่อระบุตำแหน่งของลูกบอลเป็นพิกัดของพิกเซลในการหาระยะห่างระหว่างลูกบอลกับจุดที่กำหนด โดยการทดลองจะเน้นไปที่การปรับแต่งคุณภาพของภาพเพื่อให้เหมาะสมกับการประมวลผลภาพ รวมถึงการจัดสภาพแวดล้อมต่างเพื่อให้ได้ภาพที่มีคุณภาพ

สภาพแวดล้อมนั้นมีผลต่อภาพต่อภาพที่จะนำมาประมวลผลเป็นอย่างมาก โดยเฉพาะเรื่องแสง และเงา โดยหากเกิดเงาในภาพมากเกินไป จะทำให้โปแกรมคิดว่าบริเวณที่เป็นเงานั้นคือ ลูกบอล ซึ่งจะทำให้การระบุตำแหน่งผิดพลาด รูปที่ 4.1 เป็นการแสดงการจัดสภาพแสงในแบบต่างๆ รวมถึงผลที่ได้จากการประมวลผลภาพแล้ว

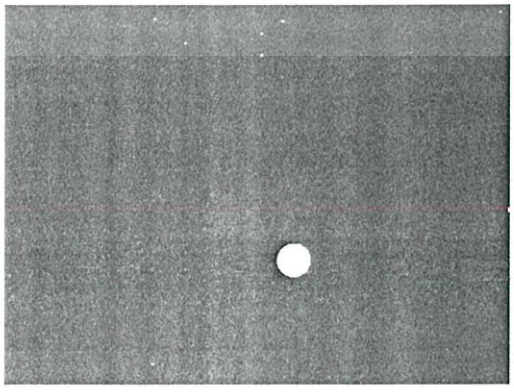


รูปที่ 4.1 การจัดแสงในลักษณะต่างๆ และผลที่ได้จากการประมวลภาพ

ภาพ Gray Scale ที่ผ่านการ Threshold แล้วนั้น อาจมีคุณภาพที่ไม่ดีที่สุด อันเนื่องมาจากจุดรบกวนที่เกิดขึ้นบนภาพ ดังแสดงในรูปที่ 4.2 ซึ่งจุดที่เกิดขึ้นนี้จะมี 2 ลักษณะ คือ จุดขาวที่เกิดจากเงากับจุดดำที่เกิดจากแสงสะท้อนบนผิวลูกบอล ซึ่งจุดดำที่เกิดจากแสงสะท้อนบนผิวลูกบอลนั้นจะส่งผลให้ได้ภาพลูกบอลไม่เต็มใบ ทำให้การหาตำแหน่งศูนย์กลางของลูกบอลคลาดเคลื่อน จึงต้องมีการปรับแต่งเพื่อกำจัดจุดขาว และดำดังกล่าวออกไป เพื่อเพิ่มความแม่นยำของผลลัพธ์

การปรับคุณภาพของภาพนั้นจะใช้ฟังก์ชันของ OpenCV สองฟังก์ชัน ได้แก่ ฟังก์ชัน Erode และฟังก์ชัน Dilate โดยฟังก์ชัน Erode จะทำการลดจุดขาวตามรูปแบบที่กำหนด ซึ่งจะช่วยในการกำจัดจุดขาวเล็กๆ ที่เกิดจากเงาบนภาพได้ ส่วนฟังก์ชัน Dilate จะช่วยเติมจุดขาวลงบนภาพตามรูปแบบที่กำหนด ซึ่งช่วยในการลบจุดดำที่เกิดจากแสงสะท้อนบนลูกบอลได้ (รายละเอียดการใช้งานฟังก์ชันทั้งสองตัวนี้อยู่ในภาคผนวก)

รูปที่ 4.2 เกิดการรบกวนจากจุดขาว และจุดดำ



รูปที่ 4.3 ผลของการประมวลผลภาพระหว่างภาพที่ปรับปรุงคุณภาพแล้ว

4.2 การควบคุมตำแหน่งลูกบอลด้วยตัวควบคุม PID

ในส่วนนี้จะเป็นการทดลองควบคุมตำแหน่งลูกบอลบนระนาบด้วยตัวควบคุมแบบ PID โดยจะแบ่งการทดลองออกเป็นสองส่วน ส่วนแรกคือ การจำลองการควบคุมตำแหน่งลูกบอลบนระนาบด้วยด้วยโปรแกรม Simulation เพื่อศึกษาถึงผลตอบสนองของระบบกับตัวควบคุม PID ชนิดต่างๆ ส่วนที่สองคือ การทดลองกับระบบจริงโดยใช้ผลจากทดลองจากส่วนแรกเป็นแนวทางในการกำหนดค่าตัวควบคุม จากนั้นจึงทำการเปรียบเทียบผลการทดลองที่ได้ระหว่างการทดลองทั้งสองส่วน

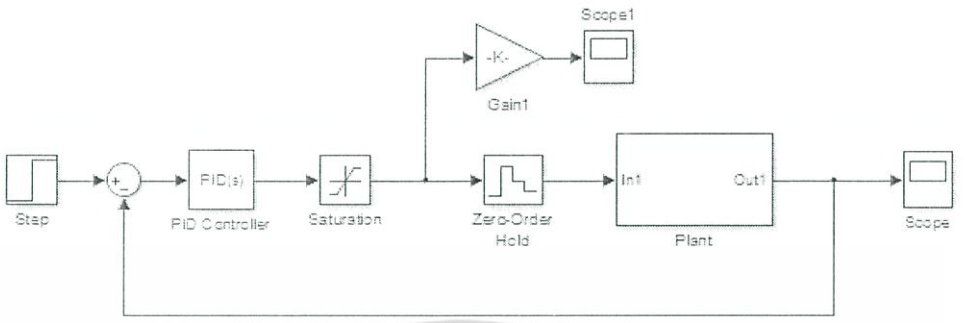
4.2.1 การจำลองการควบคุมตำแหน่งลูกบอลด้วยโปรแกรม Simulation

ในหัวข้อนี้จะเป็นการจำลองการควบคุมตำแหน่งลูกบอลบนระนาบด้วยโปรแกรม Simulation โดยจะใช้สมการถ่ายโอนที่ได้ออกแบบไว้ในบทที่ 2 ในการทดสอบ โดยการทดลองนี้มีวัตถุประสงค์เพื่อพิจารณาหาค่าที่เหมาะสมของตัวแปรของตัวควบคุมแต่ละชนิด (K_p, K_i, K_d) ในการทำให้ผลตอบสนองของระบบมีสมรรถนะตามที่ต้องการ รวมถึงพิจารณาถึงผลตอบสนองที่เปลี่ยนไปเมื่อเปลี่ยนค่าตัวแปรของตัวควบคุมแต่ละชนิด

การควบคุมตำแหน่งของลูกบอลนั้นจะกระทำผ่านการควบคุมการเอียงของระนาบในแกน X และแกน Y ซึ่งเป็นอิสระจากกัน และมีรูปแบบการทำงานที่เหมือนกันทุกประการ ในการจำลองการทำงานจึงสามารถทำการทดลองเพียงแกนเดียวก็เพียงพอ โปรแกรมที่ใช้ในการจำลองการทำงานในที่นี้คือ Matlab/Simulink แบบจำลองของระบบ (1แกน) แสดงไว้ดังรูปที่ 4.4

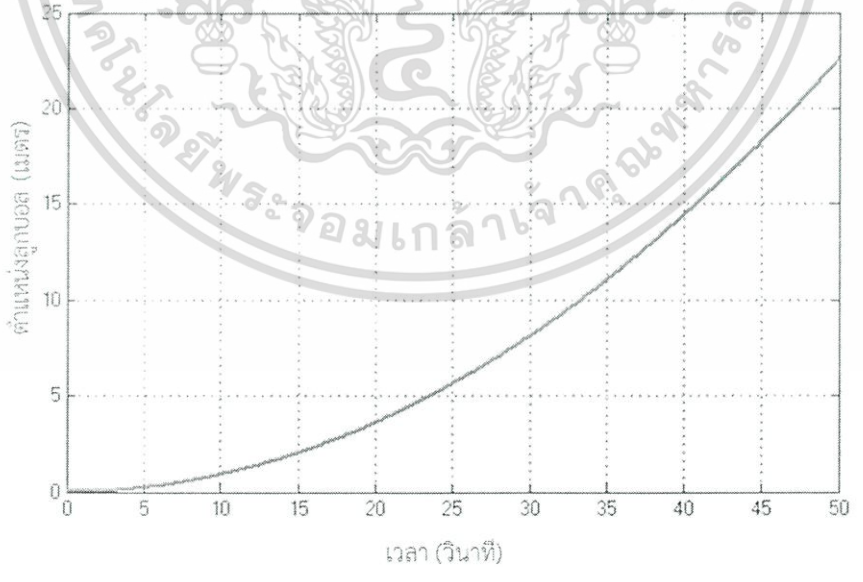
เนื่องจากอุปกรณ์ในระบบนั้นเป็นอุปกรณ์ดิจิตอล ซึ่งทำให้การทำงานของระบบไม่ได้อยู่บนโดเมนเวลาแบบต่อเนื่อง (Continuous Time Domain) แต่เป็นโดเมนเวลาแบบดิสครีต (Discrete Time Domain) ดังนั้นในการออกแบบ และจำลองระบบจำเป็นต้องกระทำบนโดเมนเวลาแบบดิสครีตคือ ต้องมีการพิจารณาถึงคาบเวลาในการชักตัวอย่าง ซึ่งในที่นี้จะคิดจากเวลาทั้งหมดในการทำงานครบ 1 รอบของระบบ เริ่มจากการทำงานของส่วนโปรแกรมคอมพิวเตอร์ จนกระทั่งเซอร์โวมอเตอร์หมุนไปยังตำแหน่งที่ต้องการ ซึ่งจากการทดลองให้ระบบทำงาน พบว่า ในส่วนของโปรแกรมนั้นมีรอบการทำงานอยู่ที่ประมาณ 0.03 – 0.04 วินาที ส่วนการทำงานของเซอร์โวมอเตอร์นั้นหากคำนวณจากคุณสมบัติของเซอร์โวมอเตอร์ที่กล่าวไว้ในหัวข้อ 3.1.3 โดยพิจารณาที่มุมเคลื่อนขนาดเล็กๆ (เนื่องจากในการทำงานจริง ส่วนใหญ่เซอร์โวมอเตอร์จะเคลื่อนที่เป็นขนาดมุมเล็กๆ เท่านั้น จึงพิจารณาที่มุมขนาดเล็กๆ เป็นหลัก) ประมาณไม่เกินหนึ่งองศา รอบการทำงานของเซอร์โวมอเตอร์จะอยู่ที่ประมาณ 0.005 วินาที เมื่อเผื่อเวลาสำหรับการ

ติดต่อสื่อสารระหว่างคอมพิวเตอร์กับไมโครคอนโทรลเลอร์แล้ว ได้ว่าควรเลือกใช้คาบเวลาในการชักตัวอย่างที่ 0.05 วินาที ซึ่งหมายความว่าระบบสามารถทำงานได้ 20 รอบต่อวินาทีนั่นเอง

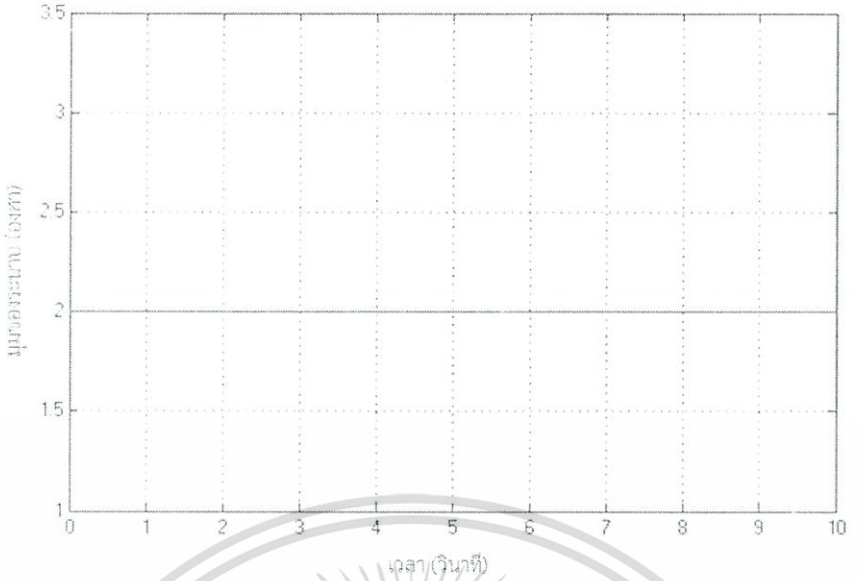


รูปที่ 4.4 แบบจำลองของระบบด้วยโปรแกรม Matlab/Simulink (พิจารณาใน 1 แกน)

เพื่อแสดงให้เห็นว่าระบบควบคุมตำแหน่งลูกบอลบนระนาบนั้น เป็นระบบที่ไม่มีเสถียรภาพหากไม่ควบคุมแบบป้อนกลับ (Feedback Control) จะทำการจำลองการทำงานของระบบดังแบบจำลองตามรูปที่ 4.3 โดยตัดส่วนป้อนกับออกไป และกำหนดให้ค่า K_p K_i K_d เท่ากับ 1, 0 และ 0 ตามลำดับ โดยให้ลูกบอลเคลื่อนที่ไปยังตำแหน่งอ้างอิงที่ 5 เซนติเมตร ได้ผลตอบสนองดังรูปที่ 4.5 และมุมของระนาบในรูปที่ 4.6

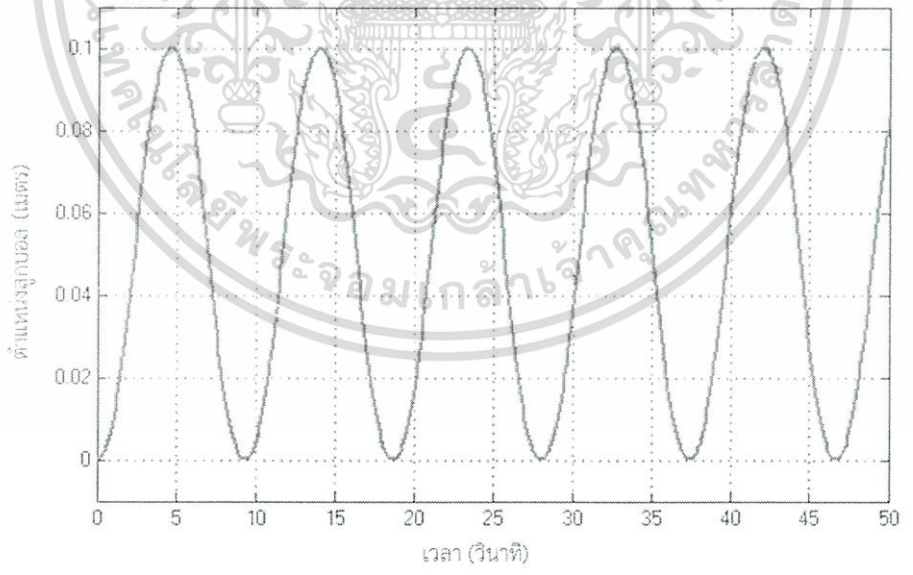


รูปที่ 4.5 ผลตอบสนองของระบบวงเปิด

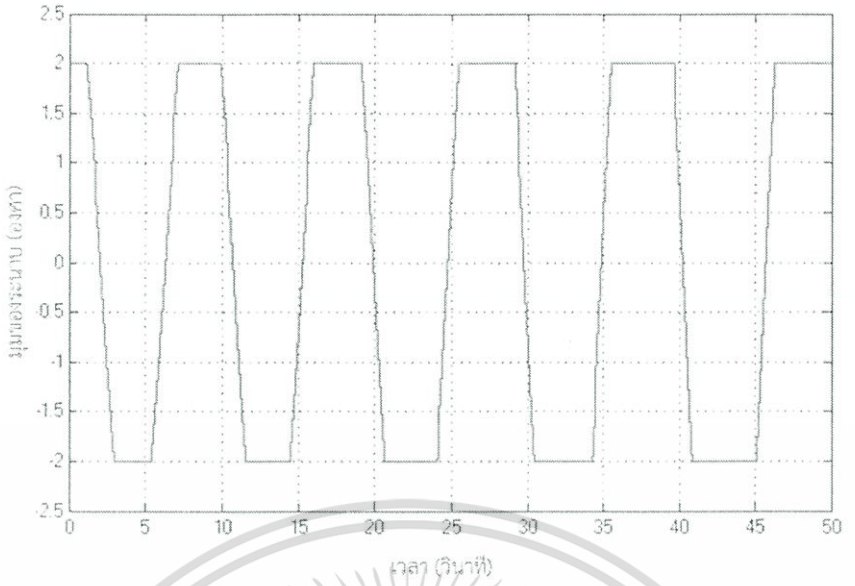


รูปที่ 4.6 มุมของระนาบเมื่อควบคุมด้วยระบบวงเปิด

เมื่อจำลองการทำงานด้วยแบบจำลองเดิม แต่เพิ่มส่วนป้อนกลับเข้าไป พบว่าได้ผลตอบสนองตามรูปที่ 4.7 และมุมของระนาบในรูปที่ 4.8



รูปที่ 4.7 ผลตอบสนองของระบบวงปิด

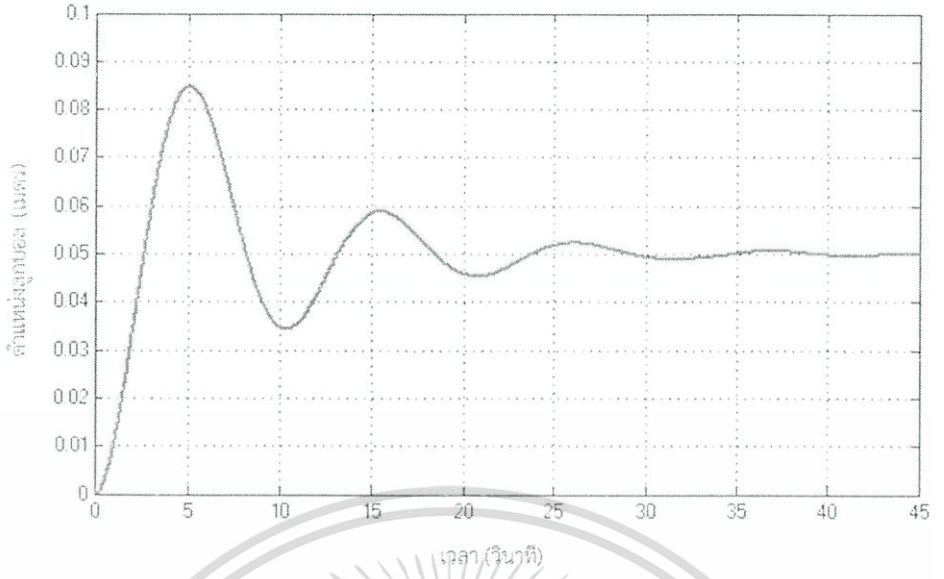


รูปที่ 4.8 มุมของระนาบเมื่อควบคุมด้วยระบบวงปิด

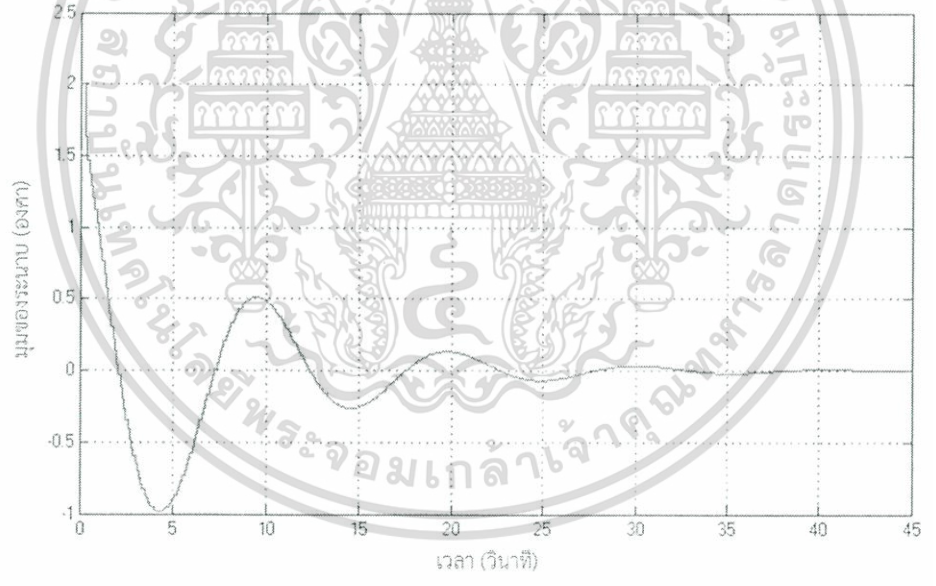
จากกราฟผลตอบสนองของระบบวงปิดพบว่า เมื่อป้อนกลับค่าเอาต์พุตโดยให้เกนมีค่าเท่ากับ 1 จะทำให้ระบบเสถียรวิกฤต ซึ่งการจะทำให้ระบบเสถียรสามารถทำได้โดยการปรับค่าตัวแปรของตัวควบคุมนั่นเอง

4.2.2 ปรับตัวควบคุมด้วยวิธี Ultimate Method

ดังที่ได้อธิบายไว้ในหัวข้อ 3.2.2 สามารถหาขอบเขตของตัวแปร K_p, K_i และ K_d ได้ด้วยวิธี Ultimate Method จากการทดลองก่อนหน้า พบว่าค่า K_u เท่ากับ 1 และค่า P_u เท่ากับ 9.35 วินาที เมื่อแทนค่าตัวแปรทั้งสองลงในตารางที่ 3.1 จะได้ว่าตัวแปร K_p K_i K_d มีค่าเท่ากับ 0.6, 0.128 และ 0.701 ตามลำดับ และเมื่อทดลองจำลองการทำงานของระบบด้วยค่าตัวแปรข้างต้น พบว่าได้ผลตอบสนองดังรูปที่ 4.9 และมุมของระนาบในรูปที่ 4.10



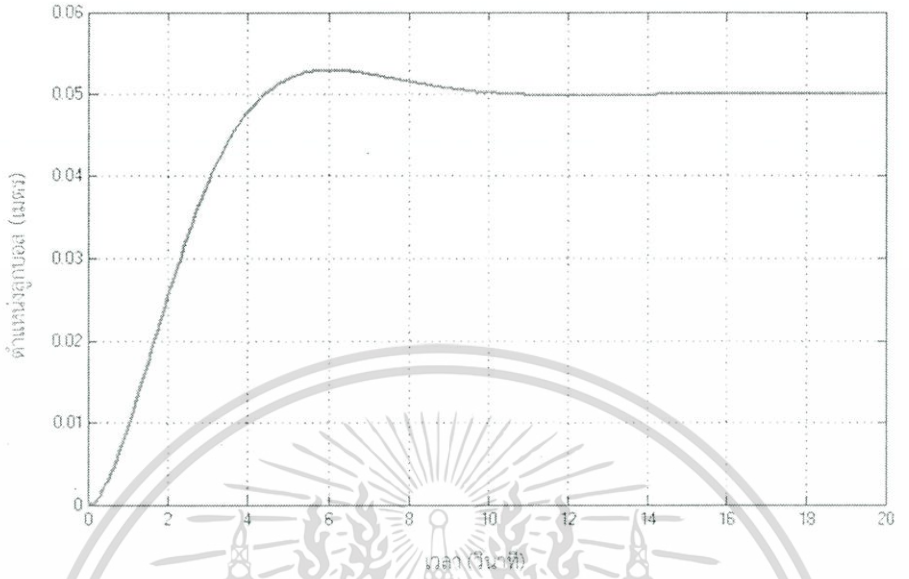
รูปที่ 4.9 ผลตอบสนองของระบบที่ใช้ตัวควบคุม PID ที่ปรับแต่งด้วยวิธี Ultimate Method



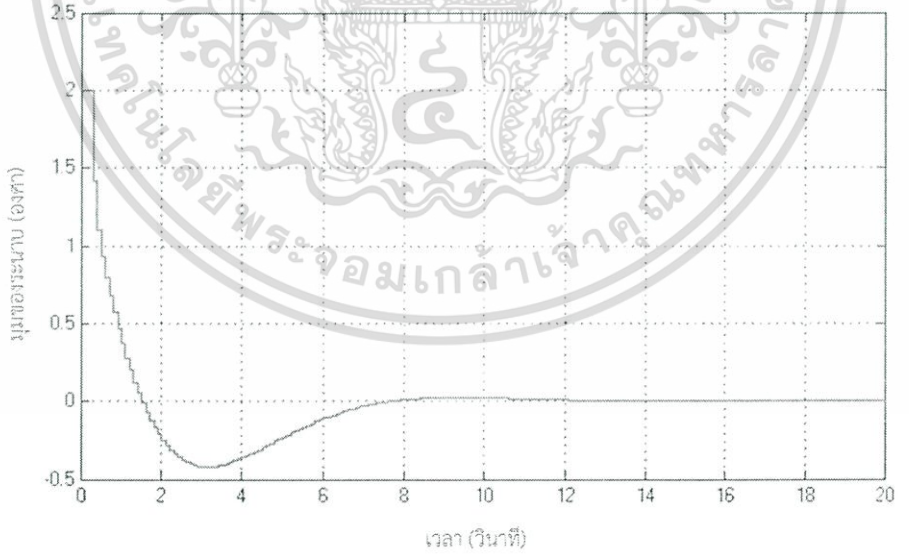
รูปที่ 4.10 มุมของระนาบเมื่อใช้ตัวควบคุม PID ที่ปรับแต่งด้วยวิธี Ultimate Method

จากกราฟผลตอบสนองของระบบจากการปรับแต่งตัวควบคุม PID ด้วยวิธี Ultimate Method พบว่ามีค่า Rising time ประมาณ 2.6 วินาที ค่า Overshoot ประมาณ 70% และใช้เวลามากกว่า 40 วินาที กว่าจะเข้าสู่ค่าเป้าหมาย ซึ่งถือว่าสมรรถนะยังไม่ดีพอ จึงจำเป็นต้องมีการปรับแต่งตัวควบคุม

เพิ่มเติมเพื่อให้ระบบมีสมรรถนะมากขึ้น ซึ่งจากการทดลองสุ่มปรับค่า พบว่าเมื่อกำหนดให้ K_p, K_i และ K_d มีค่าเท่ากับ 0.6, 0 และ 1.2 ตามลำดับ จะได้ผลตอบสนองดังรูปที่ 4.11 และมุมของระนาบในรูปที่ 4.12



รูปที่ 4.11 ผลตอบสนองของระบบเมื่อทำการปรับตัวตัวควบคุมเพื่อให้ได้สมรรถนะที่ดีขึ้น



รูปที่ 4.12 มุมของระนาบจากผลตอบสนองในรูปที่ 4.10

ซึ่งจากกราฟผลตอบสนองของระบบในรูปที่ 4.12 จะเห็นว่า ค่า Overshoot ลดลงเหลือเพียง 6% และระบบใช้เวลาเพียง 12 วินาทีในการเข้าสู่ค่าเป้าหมาย แต่ค่า Rising Time สูงขึ้นเพียงเล็กน้อย ซึ่งมีสมรรถนะดีกว่าผลการทดลองครั้งก่อนอย่างมาก

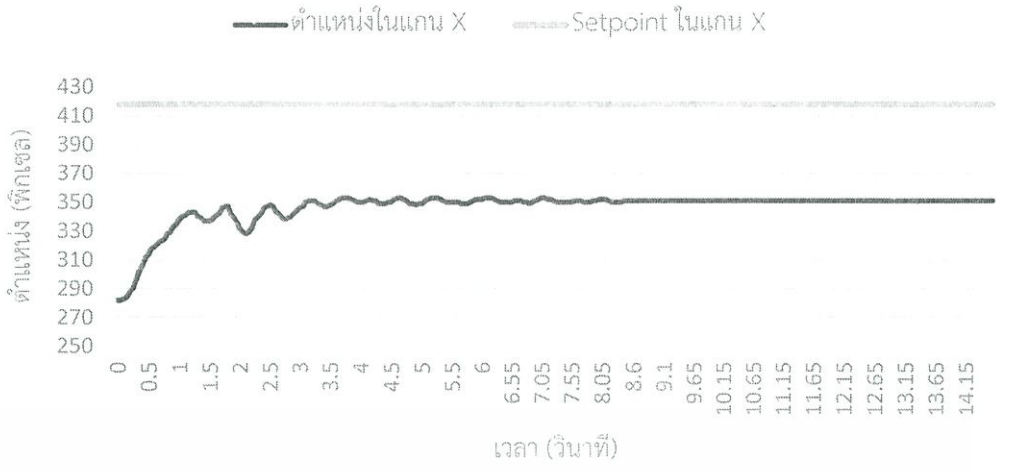
4.2.3 การควบคุมตำแหน่งลูกบอลในระบบจริง

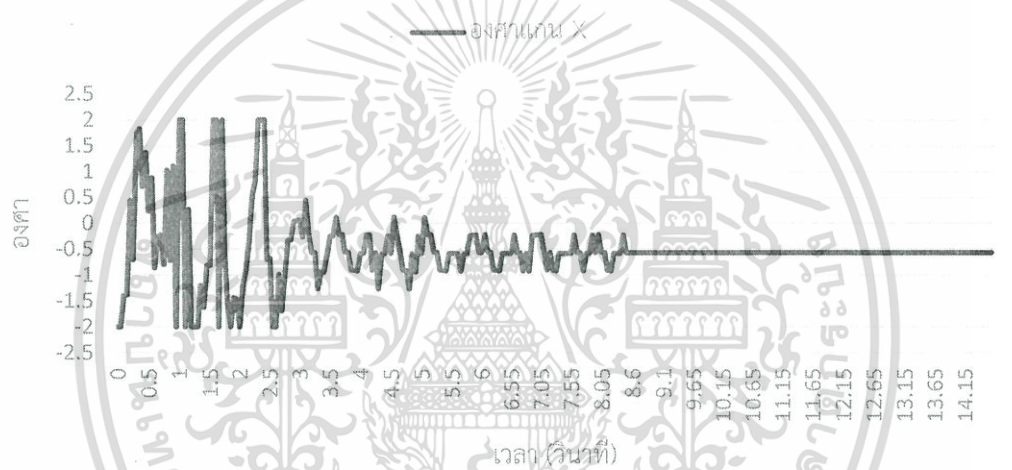
ในส่วนนี้จะเป็นการทดลองกับอุปกรณ์จริง โดยใช้ค่าที่ได้จากการทดลองด้วยการจำลองการเคลื่อนโดย Matlab มาเป็นตัวช่วยในการปรับแต่ง และอ้างอิง โดยการทดลองจะแบ่งเป็นการทดลองแบบแยกแกน X แกน Y การทดลองแบบทั้งสองแกน และการปรับปรุงระบบ

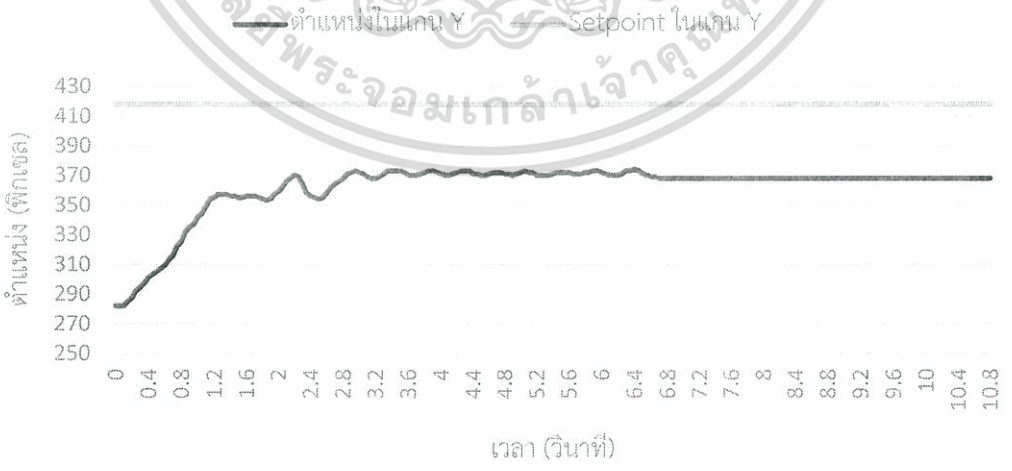
4.2.3.1 การทดลองแบบแยกแกน

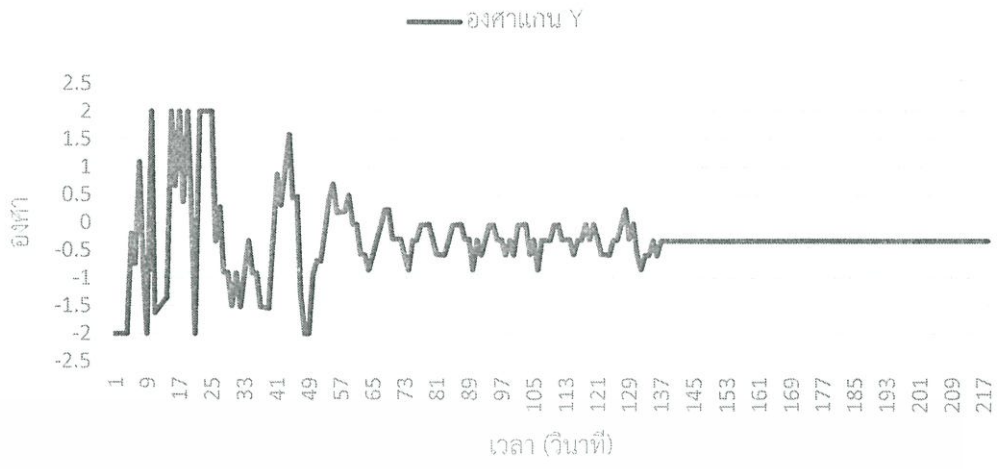
การทำงานของระบบควบคุมตำแหน่งลูกบอลบนระนาบนั้น จะประกอบไปด้วยการควบคุมความเอียงของระนาบในสองแนวแกน ซึ่งการทำงานของทั้งสองแนวแกนนั้นจะเป็นอิสระต่อกัน แต่เนื่องจากการประกอบโครงสร้างทางด้านกายภาพอาจมีข้อผิดพลาดหรืออาจไม่ละเอียดมากพอ ที่จะกล่าวได้ว่าการทำงานของทั้งสองแกนนั้นจะเป็นอิสระจากกันโดยสมบูรณ์ ดังนั้นจึงต้องทำการทดลองแบบแยกแกนเพื่อแสดงให้เห็นว่า เมื่อแต่ละแกนทำงานโดยเป็นอิสระจากกันโดยสมบูรณ์จริงๆ แล้ว จะให้ผลตอบสนองออกมาเป็นเช่นไร

เริ่มต้นการทดลองโดยการใช้ค่าที่ได้จากการจำลองการเคลื่อนที่ของลูกบอลด้วยโปรแกรม Matlab เป็นค่าเริ่มต้น พบว่าได้ผลตอบสนองในแต่ละแกนตามรูปที่ 4.13 จากผลการทดลองจะเห็นว่า ตำแหน่งสุดท้ายที่ลูกบอลหยุดนิ่งนั้นคลาดเคลื่อนจากตำแหน่งที่ต้องการอยู่มาก ค่าที่ได้จากการจำลองการเคลื่อนที่ของลูกบอลจึงไม่สามารถใช้กับระบบนี้ได้ทันที จึงต้องมีการปรับค่าตัวควบคุมเพื่อให้ระบบมีสมรรถนะตามที่ต้องการ



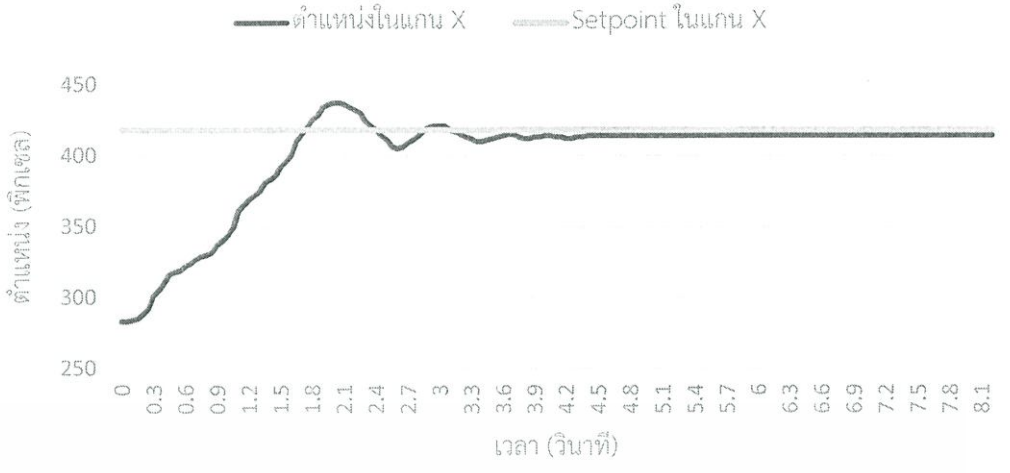


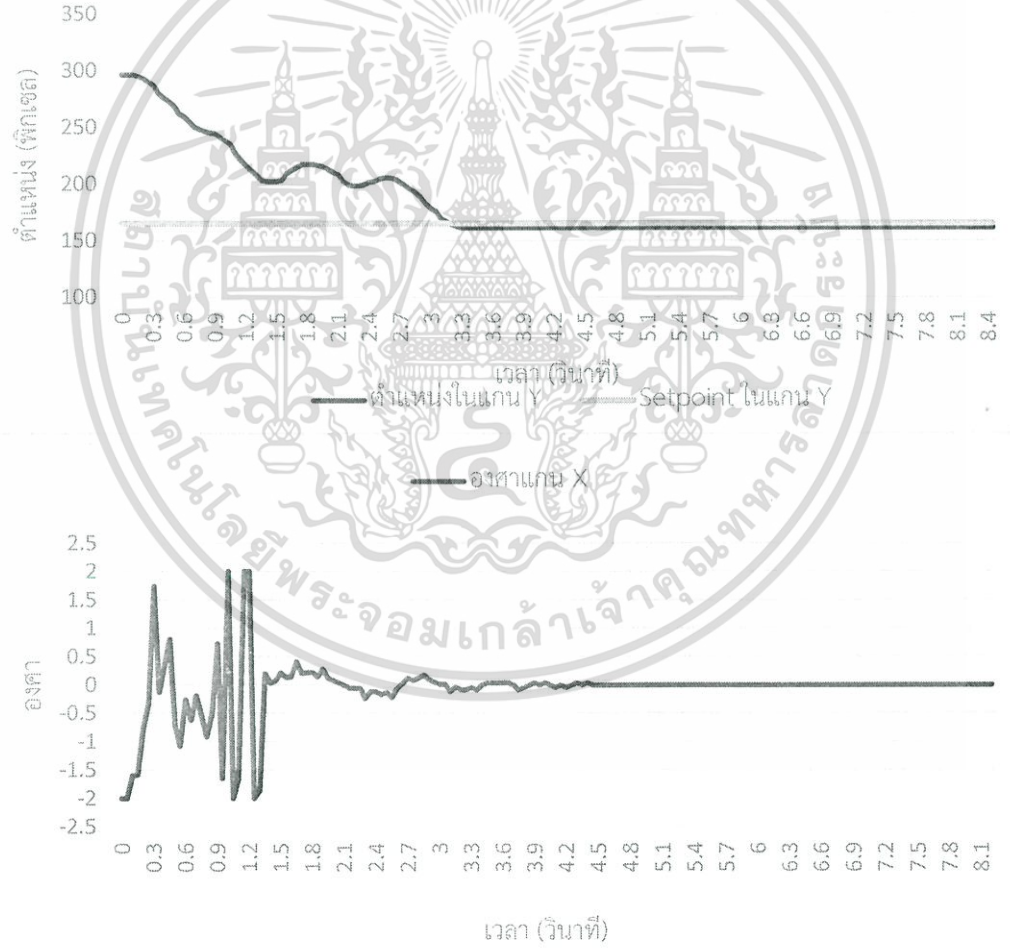


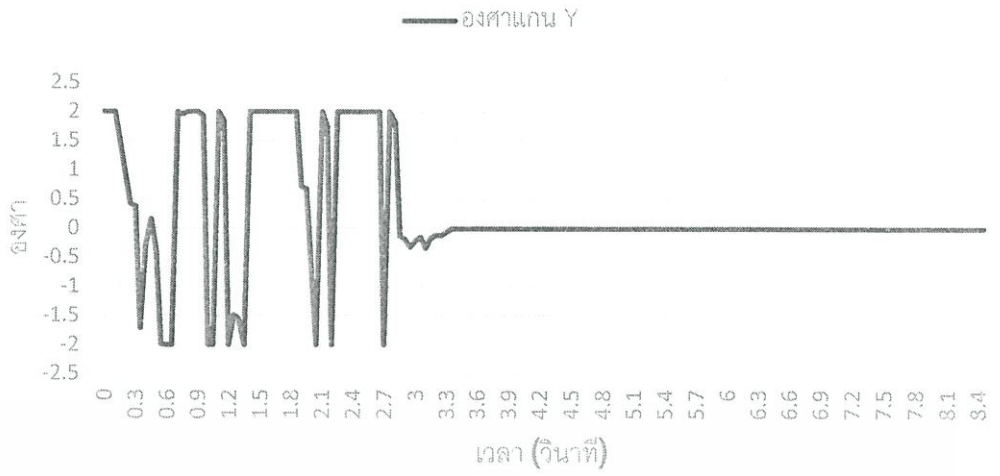


รูปที่ 4.13 ผลตอบสนองของระบบจริงแบบแยกแกนโดยใช้ค่าที่ได้จากการจำลองการเคลื่อนที่โดยโปรแกรม Matlab

เมื่อทดลองปรับค่าตัวควบคุม PID พร้อมกับสังเกตผลตอบสนองของระบบไปเรื่อยๆ พบว่า ในแกน X เมื่อปรับค่าตัวแปร K_p K_i K_d เป็น 1.0, 0 และ 1.3 ตามลำดับ ระบบสามารถควบคุมลูกบอลให้ไปอยู่ในตำแหน่งที่ต้องการได้ โดยมีค่าผิดพลาดประมาณ ไม่เกิน 10% และเช่นเดียวกันในแกน Y เมื่อปรับค่าตัวแปร K_p K_i K_d เป็น 0.8, 0 และ 1.3 ตามลำดับ ระบบสามารถควบคุมลูกบอลให้ไปอยู่ในตำแหน่งที่ต้องการได้โดยมีค่าผิดพลาดประมาณ ไม่เกิน 10% เช่นกัน





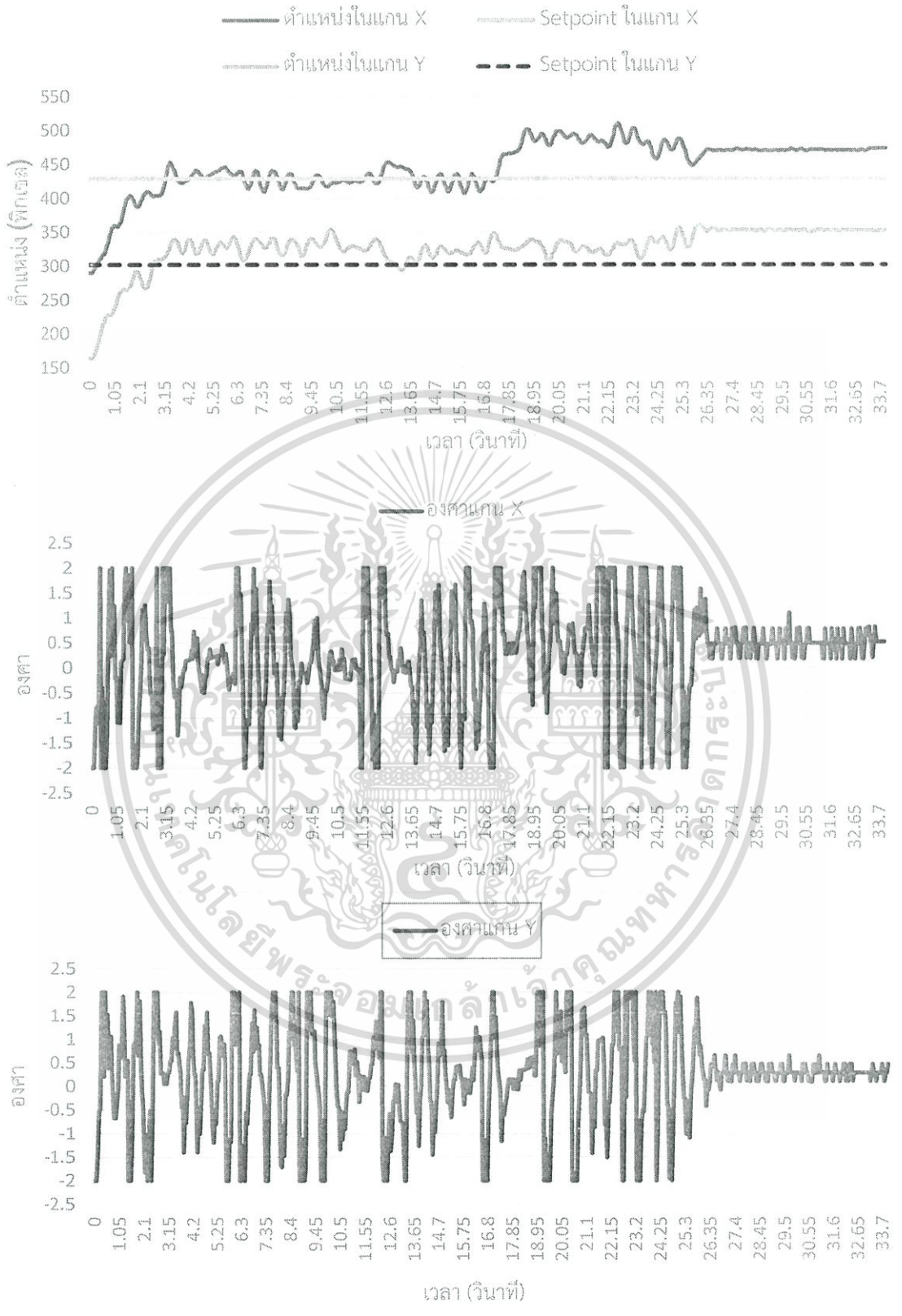


รูปที่ 4.14 ผลตอบสนองของระบบจริงแบบแยกแกน โดยใช้ค่าที่ได้จากการปรับแต่งเอง

4.2.3.2 การทดลองแบบทั้งสองแกนพร้อมกัน

การทดลองแบบแยกแกนแสดงให้เห็นแล้วว่า เมื่อแต่ละแกนทำงานอิสระจากกันอย่างสมบูรณ์ จะสามารถควบคุมตำแหน่งของลูกในแนวแกนได้ตามที่ต้องการ ดังนั้นเมื่อให้ระบบทำงานพร้อมกันทั้งสองแกน ย่อมสามารถควบคุมตำแหน่งลูกบอลบนระนาบได้เช่นกัน

เมื่อทดลองควบคุมตำแหน่งลูกบอลบนระนาบโดยให้ระบบทำงานพร้อมกันทั้งสองแกน โดยใช้ค่าตัวแปรควบคุมที่ได้จากการทดลองแบบแยกแกน พบว่าได้ผลตอบสนองตามรูปที่ 4.15 ซึ่งเมื่อพิจารณาจากผลตอบสนองจะเห็นว่า ระบบไม่สามารถควบคุมลูกบอลให้เคลื่อนที่ไปอยู่ในตำแหน่งที่ต้องการไม่ได้ จึงมีโอกาสเป็นไปได้สูงว่าโครงสร้างทางกายภาพไม่สมบูรณ์ตามเงื่อนไขในการตั้งสมมติฐานการเคลื่อนที่ของลูกบอลระนาบ ทำให้การทำงานแต่ละแกนไม่เป็นอิสระต่อกันโดยสมบูรณ์



รูปที่ 4.15 ผลตอบสนองของระบบจริงแบบทำงานพร้อมกันทั้งสองแกน

โดยใช้ค่าตัวควบคุมจากการทดลองแบบแยกแกน

ทั้งนี้ เมื่อลองปรับค่าตัวควบคุมในแต่ละแกน และสังเกตผลตอบสนองไปเรื่อยๆ พบว่าระบบยังคงไม่สามารถควบคุมลูกบอลไปยังตำแหน่งที่ต้องการได้

4.3 การปรับปรุงระบบ

จากการทดลองที่ผ่านๆ มาพบว่าระบบสามารถควบคุมตำแหน่งลูกบอลได้เมื่อทำงานแบบแยกแกน แต่เมื่อทำงานพร้อมกันทั้งสองแกนจะไม่สามารถควบคุมตำแหน่งลูกบอลให้เป็นไปตามที่ต้องการได้ แม้จะปรับค่าตัวควบคุมแล้วก็ตาม ดังนั้นเพื่อให้ระบบสามารถทำงานได้ตามที่ต้องการ จึงจำเป็นต้องมีการปรับแต่งระบบเพิ่มเติม ซึ่งในที่นี้จะปรับแต่งเฉพาะในส่วนของโปรแกรมคอมพิวเตอร์ที่ใช้ในการประมวลผล เนื่องจากสามารถทำได้ง่ายกว่าการปรับปรุงระบบทางกายภาพ อีกทั้งการปรับปรุงระบบทางกายภาพใหม่อาจทำให้ได้ผลตอบสนองที่มีลักษณะต่างออกไป ซึ่งทำให้ออกกลับไปปรับแต่งค่าตัวควบคุมใหม่ทั้งหมด

การปรับปรุงในส่วนของโปแกรมประมวลผล จะยังคงใช้ตัวควบคุมแบบ PID เป็นหลักเหมือนเดิม แต่จะปรับปรุงกระบวนการเก็บค่า และประมวลข้อมูลเพิ่มเติม เพื่อโปรแกรมทำงานได้สอดคล้องกับระบบทางกายภาพมากที่สุด จากการสังเกตค่าองศาของระนาบจากการทดลองแบบสองแกน พบว่าองศาของระนาบมีการเปลี่ยนแปลงองศามากเกินไปในช่วงเวลาสั้นๆ ซึ่งทำให้เซอร์โวมอเตอร์ตอบสนองไม่ทัน ทำให้ตำแหน่งลูกบอลเกิดการคลาดเคลื่อน ดังนั้นจึงต้องปรับปรุงการปรับเทียบค่า PID กับค่าองศาการหมุนของเซอร์โวมอเตอร์ใหม่

อีกจุดหนึ่งที่ปรับปรุงคือ ใช่ส่วนของช่วงเวลาการชักตัวอย่าง โดยแบบเดิมนั้นเมื่อทำงานเสร็จในหนึ่งรอบการทำงาน โปรแกรมจะรอจนกว่าเวลาในรอบนั้นจะครบ 0.05 วินาทีแล้วจึงเริ่มทำงานรอบใหม่ แต่ในความเป็นจริงนั้นแต่ละรอบการทำงาน (หากไม่คิดในส่วนของเซอร์โวมอเตอร์ที่ใช้เวลาน้อยมากเมื่อเทียบกับเวลาทั้งหมด) ใช้เวลาน้อยกว่า 0.05 วินาที หากตัดส่วนที่ต้องรอนี้ออกไป จะทำให้ระบบสามารถชักตัวอย่างได้มากขึ้น และทำการปรับปรุงระบบตามที่ได้กล่าวไป และทดลองร่วมกับการปรับค่าตัวควบคุมไปเรื่อยๆ พบว่าได้ผลการทดลองตามรูปที่ 4.16

รูปที่ 4.16 ผลตอบสนองของระบบจริงแบบทำงานพร้อมกันทั้งสองแกน หลังจากปรับปรุงระบบแล้ว

บทที่ 5
วิจารณ์ และสรุปผล

5.1 สรุปผลการทดลอง

จากการทดลองระบบควบคุมตำแหน่งลูกบอลบนระนาบ โดยแบ่งการทดลองออกเป็น 2 ส่วน คือ ส่วนการระบุตำแหน่งลูกบอลด้วยกล้อง และส่วนการควบคุมตำแหน่งลูกบอลด้วยตัวควบคุม PID ได้ผลดังนี้

การทดลองระบุตำแหน่งลูกบอลด้วยกล้องเป็นการทดลองจัดสภาพแวดล้อมเพื่อให้ได้แสง และเงาที่เหมาะสมเพื่อให้ภาพที่ได้มีคุณภาพ จากนั้นเมื่อผ่านกระบวนการขจัดจุดรบกวนบนภาพที่ผ่านการแปลงเป็น Gray Scale พบว่าได้ภาพภาพของลูกบอลที่มีความคมชัด และสามารถหาจุดศูนย์กลางของลูกบอลได้อย่างแม่นยำ

ในส่วนการทดลองควบคุมตำแหน่งลูกบอลด้วยตัวควบคุม PID แบ่งการทดลองออกเป็น 2 ส่วน ได้แก่ส่วนการจำลองการควบคุมตำแหน่งลูกบอลด้วยโปรแกรม Simulation และส่วนของการทดลองควบคุมตำแหน่งลูกบอลในระบบจริง โดยในส่วนแรกนั้นได้จำลองการทำงานของระบบโดยใช้โปรแกรม Matlab และทดลองระบบแบบแกนเดียว เนื่องจากการทำงานในแต่ละแกนนั้นเหมือนกัน การทดลองกำหนดให้ลูกบอลเคลื่อนที่เป็นระยะ 5 เซนติเมตร และมุมเอียงของระนาบอยู่ที่ ±2 องศา และคาบเวลาการชักตัวอย่างเท่ากับ 0.05 วินาที เมื่อทดลองปรับค่าตัวควบคุมด้วยวิธี Ultimate Method พบว่าระบบทำงานได้ตามที่ต้องการ แต่ยังมีสมรรถนะที่ไม่ดีพอ จึงทำการปรับปรุงค่าตัวควบคุมเพิ่มเติม พบว่าเมื่อกำหนดค่า K_p K_i K_d มีค่าเท่ากับ 0.6, 0 และ 1.2 ตามลำดับ จะได้ผลตอบสนองที่มีค่า Rising Time ประมาณ 4.5 วินาที ค่า Overshoot ประมาณ 6% และใช้เวลาประมาณ 12 วินาทีในการเข้าสู่ค่าเป้าหมาย ซึ่งถือเป็นสมรรถนะที่น่าพอใจ

ถัดมาในส่วนของการทดลองควบคุมตำแหน่งลูกบอลในระบบจริงนั้น จะแบ่งออกเป็น 3 ส่วน คือการทดลองแบบแยกแกน การทดลองแบบทั้งสองแกน และการปรับปรุงระบบ การทดลองแบบแยกแกนนั้นเริ่มด้วยการนำค่าตัวควบคุมที่ได้จากการจำลองการทำงานของระบบมาใช้ ผลคือตำแหน่งลูกบอลมีความคลาดเคลื่อนจากตำแหน่งที่ต้องการมากเกินไปในทั้งสองแกน เมื่อทำการปรับค่าตัวควบคุมใหม่โดยกำหนดให้ค่า K_p K_i K_d ในแกน X มีค่าเท่ากับ 1.0, 0 และ 1.3 ตามลำดับ และกำหนดให้ค่า K_p K_i K_d ในแกน Y มีค่าเท่ากับ 0.8, 0 และ 1.3 ตามลำดับ พบว่าระบบสามารถควบคุม

ลูกบอลไปยังตำแหน่งที่ต้องการได้ โดยมีค่าคลาดเคลื่อนไม่เกิน 10% ซึ่งสรุปได้ว่าเมื่อแต่ละแกนทำงานเป็นอิสระจากกันจะสามารถควบคุมตำแหน่งในแกนได้ตามต้องการ

ถัดมาเป็นการทดลองแบบทั้งสองแกนพร้อมกัน โดยใช้ค่าตัวควบคุมจากการทดลองแบบแยกแกน พบว่า ระบบไม่สามารถควบคุมลูกบอลไปยังตำแหน่งที่ต้องการได้ แม้จะทำการปรับค่าตัวควบคุมแล้วก็ตาม จึงมีความเป็นไปได้ว่าการทำงานของแต่ละแกนนั้นไม่เป็นอิสระจากกัน อาจเนื่องมาจากการออกแบบโครงสร้างทางกายภาพยังไม่ดีพอ

สุดท้ายในส่วนของการปรับปรุงนั้น ได้ทดลองปรับในส่วนของโปรแกรมคอมพิวเตอร์ โดยปรับปรุงของส่วนการเทียบค่า PID กับองศาการหมุนของเซอร์โวมอเตอร์ และคาบเวลาการชักตัวอย่าง เพื่อให้ระบบสามารถทำงานได้สอดคล้องกับโครงสร้างทางกายภาพมากขึ้น ผลที่ได้คือ ระบบสามารถควบคุมลูกบอลไปยังตำแหน่งที่ต้องการได้สำเร็จ

5.2 ปัญหาที่พบ

ปัญหาหลักที่พบในโครงงานนี้คือ ส่วนของโครงสร้างการกายภาพ เนื่องจากชิ้นส่วนทั้งหมดนั้นตัด และประกอบด้วยมือ ทำให้ขนาด และระยะต่างๆ ในการประกอบมีความคลาดเคลื่อน ในส่วนของเซอร์โวมอเตอร์นั้นตอบสนองได้ช้าไป และมักจะไม่ทำงานเมื่อต้องต้องหมุนในองศาที่มีขนาดเล็กมากๆ ทำให้ไม่สามารถควบคุมองศาของระนาบได้ตามที่โปรแกรมประมวลผลได้

ปัญหาต่อมาเป็นในส่วนของโปรแกรมคอมพิวเตอร์ เนื่องจากผู้เขียนโปรแกรมไม่มีความชำนาญเฉพาะในด้านการเขียนโปรแกรม จึงทำให้โปรแกรมอาจใช้เวลาในการทำงานแต่ละรอบมากเกินไป รวมถึงอัลกอริทึมในการเทียบค่าองศาของเซอร์โวมอเตอร์กับค่าที่ได้จาก PID ยังไม่ดีพอที่จะทำให้ระบบแสดงประสิทธิภาพได้สูงสุด

5.3 แนวทางการแก้ไข และข้อเสนอแนะ

1. เลือกใช้มอเตอร์ที่สามารถตอบสนองได้เร็ว และละเอียดกว่านี้ อาทิเช่น ดิจิตอลเซอร์โวมอเตอร์

2. ออกแบบโครงสร้างให้การทำงานของแต่ละแกนมีผลต่อกันน้อยลง ควรตัด และประกอบชิ้นงานด้วยอุปกรณ์ที่มีความแม่นยำสูง

3. หากเลือกใช้กล้องที่มีค่าเฟรมเรตสูงขึ้น จะทำให้สามารถประมวลผลได้ละเอียดขึ้น แต่ถึงกระนั้นก็ต้องลดระยะเวลาในการทำงานของโปรแกรมในแต่ละรอบไปพร้อมกันด้วย

เอกสารอ้างอิง

[1] กิตติ ตีรเศรษฐ. **พื้นฐานวิศวกรรมระบบควบคุม.** กรุงเทพฯ: สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2539.

[2] อรรถพล อยู่แสนสุข, ภิเษก เลิศวรรณการ. **"ระบบควบคุมตำแหน่งลูกบอลบนระนาบ."** ปริญญานิพนธ์ วิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมระบบควบคุม คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2551.

[3] ชวนากร จรดรัมย, ธีระพันธ์ ศรีแจ่ม. **"การศึกษาตัวควบคุมแบบรวมชนิดPIDสำหรับการควบคุมความดันอุณหภูมิและระดับน้ำโดยใช้ตัวควบคุมเชิงตรรกะ รุ่น SIEMENS S7-300."** ปริญญานิพนธ์วิศวกรรมศาสตรบัณฑิต ภาควิชาวิศวกรรมอุตสาหการ คณะวิศวกรรมศาสตร์, มหาวิทยาลัยขอนแก่น. 2555.

[4] **Arduino Mega 2560** (ออรไลน์). สืบค้นจาก : http://www.arduino.cc/en/Main/arduinoBoardMega2560

[5] **การส่งผ่านข้อมูลแบบอนุกรม** (ออนไลน์). สืบค้นจาก : http://irrigation.rid.go.th/rid15/ppn/Knowledge/Networks%20Technology/network4.htm

[6] **ระบบควบคุมแบบป้อนกลับ** (ออนไลน์). สืบค้นจาก : https://app.enit.kku.ac.th/mis/administrator/doc_upload/20110308123427.pdf

[7] **ระบบสี** (ออนไลน์). สืบค้นจาก : http://digi.library.tu.ac.th/thesis/st/0251/03CHAPTER2.pdf

[8] **ตัวอย่างการควบคุม RC Servo Motor ด้วย Arduino** (ออนไลน์). สืบค้นจาก : http://www.thaieasyelec.com/article-wiki/review-product-article/บทความตัวอย่างการควบคุม-rc-servo-motor-ด้วย-arduino.html

[9] **OpenCV** (ออนไลน์). สืบค้นจาก : http://docs.opencv.org/index.html

[10] **Ball & Beam: System Modeling** (ออนไลน์). สืบค้นจาก : http://ctms.engin.umich.edu/CTMS/index.php?example=BallBeam&section=SystemModeling

[11] **เทคโนโลยีการประมวลผลภาพ (Image processing)** (ออนไลน์). สืบค้นจาก :

https://silllovely.wordpress.com/2013/06/11/เทคโนโลยีการประมวลผลภา/

[12] Image processing (ออนไลน์). สืบค้นจาก :

http://www.nectec.or.th/index.php?option=com_content&view=article&id=289%3Aimage-processing

ภาคผนวก

ภาคผนวก ก

โปรแกรมการประมวลผล และควบคุมตำแหน่งลูกบอลบนระนาบ

ก.1 โปรแกรมประมวลผลในส่วนของคอมพิวเตอร์

โปรแกรมที่ใช้ในส่วนนี้คือ โปรแกรม Visual Studio 2012 โดยใช้ภาษา C++ ในการเขียน

ก.1.1 โปรแกรมประมวลผลในส่วนของคอมพิวเตอร์ (ก่อนปรับปรุงระบบ)

```
#include<opencv\cv.h>
#include<opencv\highgui.h>
#include<iostream>
#include<stdio.h>
#include<ctime>                    //for clock count
#include<fstream>                  //for save to txt

#include<sstream>
#include<string>
#include<stdlib.h>
#include<Windows.h>
#include<math.h>
#definePI 3.14159265

usingnamespace cv;
usingnamespace std;

void morphOps(Mat&);
MatRGB2GrayScale(Mat);
Point Contour_CM_Find_and_Display(Mat, Mat, Mat);
void CallBackFunc(int, int, int, int, void*);
void CallBackFunc(intevent, int, int, int, void*);
int Servo_angle(double, double);
void Serial2(int, int);

int iLastX = -1;
int iLastY = -1;
bool Target_Set = FALSE;
Point Setpoint = Point(-1,-1);
double saveBall_position [10000][9];        // Time : Xposition : Yposition
HANDLE hSerial = CreateFile(L"COM1", GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);      /// for serial
communication
DWORD btsIO;
int hue = 55;
int Max_angle = 200;
int Zero_angle = 1500;
double P_ang_max = 2*PI/180;
double P_ang_zero = 0;
double P_ang = 0;

int main(){

        ////////////////////// Initail Parameters ////////////////////////////////

        Mat img, gray_img;  // for video capture
        Mat imgLines = Mat::zeros(Size(640,480), CV_8UC3);          // for draw ball path
        Point Ball_Position;

        //***** for PID ******//
        double xoutput;                    double youtput;
```

```
double pre_xoutput = 0;                double pre_youtput = 0;
double Max_error_x = 0;                double Max_error_y = 0;
double xerr;                                   double yerr;
double xpre_error = 0;                         double ypre_error = 0;
double xInte = 0;                              double yInte = 0;
double xDiff = 0;                              double yDiff = 0;
double dt = 0.05; //sec
double Kpx = 1.0;                              double Kpy = 0.8;
double Kix = 0;                                double Kiy = 0;
double Kdx = 1.3;                              double Kdy = 1.3;
//********************//

int X_angle = 1500 ;                   int Y_angle = 1500;

clock_t T_present=0, T_start=0, Time=0;    // for time loop check

//////////////////////////////////////////////////////////////////////////
/////// Setup serial port connection and needed variables. ///////////////
if (hSerial!=INVALID_HANDLE_VALUE)
  {
printf("Port opened! \n");

DCB dcbSerialParams;
GetCommState(hSerial,&dcbSerialParams);

    dcbSerialParams.BaudRate = CBR_9600;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.Parity = NOPARITY;
    dcbSerialParams.StopBits = ONESTOPBIT;

SetCommState(hSerial, &dcbSerialParams);
        }
        else
        {
            if (GetLastError() == ERROR_FILE_NOT_FOUND)
            {
                printf("Serial port doesn't exist! \n");
            }

            printf("Error while setting up serial port! \n");
        }

        //////////////////////////////////////////////////////////////////////////


        /// Video Capture ///
        VideoCapture cap;
        cap.open(1);
        namedWindow("Result window",1);
        ////////////////////

        Serial2(Zero_angle,Zero_angle);
        waitKey(500);
        Serial2(Zero_angle+Max_angle,Zero_angle+Max_angle);
        waitKey(500);
        Serial2(Zero_angle-Max_angle,Zero_angle+Max_angle);
        waitKey(500);
        Serial2(Zero_angle+Max_angle,Zero_angle-Max_angle);
        waitKey(500);
        Serial2(Zero_angle-Max_angle,Zero_angle-Max_angle);
        waitKey(500);
        Serial2(Zero_angle,Zero_angle);
        waitKey(500);
```

```
for(int a=0; a<10000; a++)
        for(int b=0; b<9; b++)
                saveBall_position[a][b] = -1;

namedWindow("Result window",1);
namedWindow("Parameters");
createTrackbar("Hue","Parameters",&hue,255);

//-------------------- MAIN PROGRAME ----------------------//
while(1){

        cap>>img;
        gray_img = RGB2GrayScale(img);
        Ball_Position = Contour_CM_Find_and_Display(gray_img,img,imgLines);
        setMouseCallback("Result window", CallBackFunc, NULL);

        if(Target_Set){
                // open clock
                T_start = clock();              double tmp = 0;
                for(int i = 0; Target_Set ; i++){
                        cap>>img;
                        gray_img = RGB2GrayScale(img);
                        Ball_Position = Contour_CM_Find_and_Display(gray_img,img,imgLines);
                        saveBall_position[i][1] = Ball_Position.x;
                        saveBall_position[i][2] = Setpoint.x;
                        saveBall_position[i][3] = Ball_Position.y;
                        saveBall_position[i][4] = Setpoint.y;


                        /// Cal PID for x-axis


                        xerr = (Setpoint.x - Ball_Position.x)*(-1);

                        xInte = xInte + (xerr * dt);

                        xDiff = (xerr - xpre_error)/dt;

                        xoutput = (Kpx * xerr) + (Kix * xInte) + (Kdx * xDiff);

                        if((abs(pre_xoutput)/abs(xoutput) > 10)){
                                Max_error_x = abs(xoutput);
                                xpre_error = xerr;
                                pre_xoutput = xoutput;
                        }
                        elseif( abs(xoutput) > Max_error_x ){
                                Max_error_x = abs(xoutput);
                                //xpre_error = 0;
                                xpre_error = xerr;
                                pre_xoutput = xoutput;
                        }
                        else{
                                xpre_error = xerr;
                                pre_xoutput = xoutput;
                        }

                        X_angle = Servo_angle(xoutput, Max_error_x);
                        saveBall_position[i][5] = xoutput;
                        saveBall_position[i][6] = P_ang*180/PI;

                        /// Cal PID for y-axis
                        yerr = (Setpoint.y - Ball_Position.y)*(-1);
                        yInte = yInte + (yerr * dt);
```

```
                                        yDiff = (yerr - ypre_error)/dt;
                                        youtput = (Kpy * yerr) + (Kiy * yinte) + (Kdy * yDiff);


                                        if((abs(pre_youtput)/abs(youtput) > 10)){
                                                Max_error_y = abs(youtput);
                                                ypre_error = yerr;
                                                pre_youtput = youtput;
                                        }
                                        elseif( abs(youtput) > Max_error_y ){
                                                Max_error_y = abs(youtput);
                                                //ypre_error = 0;
                                                ypre_error = yerr;
                                                pre_youtput = youtput;
                                        }
                                        elsef
                                                ypre_error = yerr;
                                                pre_youtput = youtput;
                                        }

                                        Y_angle = Servo_angle(youtput, Max_error_y);
                                        saveBall_position[i][7] = youtput;
                                        saveBall_position[i][8] = P_ang*180/PI;

                                /// Send parameter to arduino
                                Serial2(X_angle,Y_angle);

                                /// loop time check
                                do{
                                T_present = clock();
                                Time = T_present - T_start;
                                }while(Time%650 != 0);
                                saveBall_position [i][0] = (static_cast<double>(Time)/CLOCKS_PER_SEC)-0.05;
                                tmp = Time;
                                waitKey(1);
                                setMouseCallback("Result window", CallBackFunc, NULL);
                        }
                        //// reset Target
                        Setpoint = Point(-1, -1);
                        Target_Set = FALSE;

                        //// save ball's path
                        ofstream save_path("Ball_path.txt",ios::out);
                        for(int k=0; saveBall_position[k][0] > -1; k++){
                                for(int j=0; j<9; j++){
                                        save_path << saveBall_position[k][j] ;
                                        if(j==8){
                                                save_path <<'\n';
                                        }
                                        else save_path <<'\t' ;
                                }
                        }
                        //// pause programe
                        waitKey(50);
                        Serial2(Zero_angle,Zero_angle);
                        waitKey(50);
                        CloseHandle(hSerial);
                }
                waitKey(10);
        }
        //-------------------------------------------------------------------------------//

}
```

```
///// Modify image function
void morphOps(Mat&thresh){

        Mat erodeElement = getStructuringElement(MORPH_RECT,Size(3,3));
        Mat dilateElement = getStructuringElement(MORPH_RECT,Size(3,3));

        erode(thresh,thresh,erodeElement);
        erode(thresh,thresh,erodeElement);

        dilate(thresh,thresh,dilateElement);
        dilate(thresh,thresh,dilateElement);
}

///// Convert to Gray Scale & Modify function
MatRGB2GrayScale(Matimg){
        Mat gray;
        cvtColor(img,gray,CV_BGR2GRAY);
        inRange(gray,0,hue,gray);
        morphOps(gray);

        return gray;
}

///// Find&Display Contour&Center of mass
Point Contour_CM_Find_and_Display(Matinputimg,Matdisplayimg, MatBallpath_img){

        int Xposition = -1;                                     //
        int Yposition = -1;                                     //  for find mass center
        int Ballarea = 800;
        //imshow("Gray Scale", inputimg);

        /// Find contours
        vector<vector<Point>> contours;
        vector<Vec4i> hierarchy;
        RNGrng(12345);
        findContours(inputimg, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

        if(contours.size() > 0){
                double tmp=0;
                double MaxArea=0;
                for(int j = 0; j < contours.size(); j++){
                        if(contourArea(contours[j]) > tmp){
                        tmp = contourArea(contours[j]);
                        MaxArea = j;
                        }
                }
                Moments mu;
                mu = moments(contours[MaxArea]);
                //Mass center
                if (mu.m00 > Ballarea){
                        //calculate the position of the ball
                        Xposition = mu.m10 / mu.m00;
                        Yposition = mu.m01 / mu.m00;
                        iLastX = Xposition;
                        iLastY = Yposition;
                }
                /// Draw contours
                Mat drawing = Mat::zeros(inputimg.size(), CV_8UC3 );
                if(contourArea(contours[MaxArea]) > Ballarea){
                        drawContours(displayimg, contours, MaxArea, Scalar(0,0,255), 2, 8, hierarchy, 0, Point() );
                        circle(displayimg, Point(Xposition,Yposition), 4, Scalar(0,255,0), -1, 8, 0 );
                        //draw target box
                        if(Setpoint.x > 0 && Setpoint.y > 0){
                                circle(displayimg, Setpoint, 4, Scalar(0,0,255), -1, 8, 0 );
```

```
                                        line(displayimg, Point(Setpoint.x-20,Setpoint.y-20), Point(Setpoint.x+20,Setpoint.y-20), Scalar(0,255,255), 2);
                                        line(displayimg, Point(Setpoint.x+20,Setpoint.y-20), Point(Setpoint.x+20,Setpoint.y+20), Scalar(0,255,255), 2);
                                        line(displayimg, Point(Setpoint.x+20,Setpoint.y+20), Point(Setpoint.x-20,Setpoint.y+20), Scalar(0,255,255), 2);
                                        line(displayimg, Point(Setpoint.x-20,Setpoint.y+20), Point(Setpoint.x-20,Setpoint.y-20), Scalar(0,255,255), 2);
                                }
                                string poX = to_string(Xposition);
                                string poY = to_string(Yposition);
                                string Area = to_string(mu.m00);
                                putText(displayimg,"Position X :" + poX,Point(20,20),FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,255),1,15,false);
                                putText(displayimg,"Position Y :" + poY,Point(20,40),FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,255),1,15,false);
                                putText(displayimg,"Area :" + Area,Point(20,60),FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,255),1,15,false);
                                poX = to_string(Setpoint.x);
                                poY = to_string(Setpoint.y);
                                putText(displayimg," (" + poX + "," + poY + ")",Setpoint,FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,255),1,15,false);
                                }
                        }

                        imshow("Result window", displayimg+Ballpath_img);

                        returnPoint(Xposition,Yposition);

}

///// Call back function for Get Mouse click position
void CallBackFunc(intevent, intx, inty, intflags, void* userdata)
{
if  (event == EVENT_LBUTTONDOWN )
    {
                        Setpoint.x = x;
                        Setpoint.y = y;
                        Target_Set = TRUE;
    }
                elseif  ( event == EVENT_RBUTTONDOWN )
    {
                        Target_Set = FALSE;
                }
}

int Servo_angle(doublepid, doubleMax_error){

                int Servo_angle = Zero_angle;
                P_ang = 0;

                P_ang = (pid/Max_error)*(P_ang_max);

                if(P_ang > P_ang_zero + P_ang_max)
                        P_ang = P_ang_zero + P_ang_max;
                elseif(P_ang < P_ang_zero - P_ang_max)
                        P_ang = P_ang_zero - P_ang_max;

                Servo_angle = Zero_angle + (int)((asin((145/15)*(sin(P_ang)))*180/PI)/0.10345);

                return Servo_angle;
}

void Serial2(intXangle, intYangle){

                        char outputChars[10];
                        for(int i = 4; i >= 0; i--){
                                if(i == 4){
                                        outputChars[i] = '-';
                                }
                                else{
                                        outputChars[i] = (Xangle % 10)+'0';
```

```
                        Xangle /= 10;
                }
        }
        for(int j = 9; j >= 5; j--){
                if(j == 9){
                        outputChars[j] = '*';
                }
                else{
                        outputChars[j] = (Yangle % 10)+'0';
                        Yangle /= 10;
                }
        }
        cout<< outputChars << endl;
        WriteFile(hSerial, outputChars, strlen(outputChars), &btsIO, NULL);
        fill_n(outputChars, 10, '\0');
}
```

ก.1.2 โปรแกรมประมวลผลในส่วนของคอมพิวเตอร์ (หลังปรับปรุงระบบ)

```
#include<opencv\cv.h>
#include<opencv\highgui.h>
#include<iostream>
#include<stdio.h>
#include<ctime>                         //for clock count
#include<fstream>                       //for save to txt

#include<sstream>
#include<string>
#include<stdlib.h>
#include<Windows.h>
#include<math.h>
#definePI 3.14159265

usingnamespace cv;
usingnamespace std;


void morphOps(Mat&);
MatRGB2GrayScale(Mat);
Point Contour_CM_Find_and_Display(Mat, Mat, Mat);
void CallBackFunc(int, int, int, int, void*);
void CallBackFunc(intevent, int, int, int, void*);
int Servo_angle(double, double);
void Serial2(int, int);

int iLastX = -1;
int iLastY = -1;
bool Target_Set = FALSE;
Point Setpoint = Point(-1,-1);
double saveBall_position [10000][9];          // Time : Xposition : Yposition
HANDLE hSerial = CreateFile(L"COM1", GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);      /// for serial
communication
DWORD btsIO;
int hue = 55;
double Max_error_x = 0, Max_error_y = 0;
double MEX = 0, MEY = 0;
int Max_angle = 200;
int Zero_angle = 1500;
double P_ang_max = 2*PI/180;
double P_ang_zero = 0;
double P_ang = 0;

int main(){
```

```
////////////////////// Inital Parameters ////////////////////////////////

Mat img, gray_img;   // for video capture
Mat imgLines = Mat::zeros(Size(640,480), CV_8UC3);          // for draw ball path
Point Ball_Position;

//***** for PID ******//
double xoutput;                                                    double youtput;
double xerr;                                        double yerr;
double prex = 0;                                    double prey = 0;
double xpre_error = 0;                    double ypre_error = 0;
double xInte = 0;                                   double yInte = 0;
double xDiff = 0;                                   double yDiff = 0;
double dt = 0.001; //sec
double Kp = 0.8;
double Ki = 0;
double Kd = 400;
//******************//

int X_angle = 1500 ;                        int Y_angle = 1500;

clock_t T_present=0, T_start=0, Time=0;     // for time loop check


////////////////////////////////////////////////////////////////////////
//////// Setup serial port connection and needed variables. ////////////
if (hSerial !=INVALID_HANDLE_VALUE)
  {
printf("Port opened! \n");

DCB dcbSerialParams;
GetCommState(hSerial,&dcbSerialParams);

    dcbSerialParams.BaudRate = CBR_9600;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.Parity = NOPARITY;
    dcbSerialParams.StopBits = ONESTOPBIT;

SetCommState(hSerial, &dcbSerialParams);
        }
        else
        {
                if (GetLastError() == ERROR_FILE_NOT_FOUND)
                {
                        printf("Serial port doesn't exist! \n");
                }

                printf("Error while setting up serial port! \n");
        }

        ////////////////////////////////////////////////////////////////////////////

        /// Video Capture ///
        VideoCapture cap;
        cap.open(1);
        namedWindow("Result window",1);
        ////////////////////

        Serial2(Zero_angle,Zero_angle);
        waitKey(500);
        Serial2(Zero_angle+Max_angle,Zero_angle+Max_angle);
        waitKey(500);
        Serial2(Zero_angle-Max_angle,Zero_angle+Max_angle);
```

```
waitKey(500);
Serial2(Zero_angle+Max_angle,Zero_angle-Max_angle);
waitKey(500);
Serial2(Zero_angle-Max_angle,Zero_angle-Max_angle);
waitKey(500);
Serial2(Zero_angle,Zero_angle);
waitKey(500);

for(int a=0; a<10000; a++)
        for(int b=0; b<9; b++)
                saveBall_position[a][b] = -1;

namedWindow("Result window",1);
namedWindow("Parameters");
createTrackbar("Hue","Parameters",&hue,255);

//---------------------------------- MAIN PROGRAME ----------------------------------//
while(1){

        cap>>img;
        gray_img = RGB2GrayScale(img);
        Ball_Position = Contour_CM_Find_and_Display(gray_img,img,imgLines);
        setMouseCallback("Result window", CallBackFunc, NULL);

        if(Target_Set){
                // open clock
                T_start = clock();              double tmp = 0;
                MEX = abs(Setpoint.x - Ball_Position.x);
                MEY = abs(Setpoint.y - Ball_Position.y);
                for(int i = 0; Target_Set ; i++){
                cap>>img;
                gray_img = RGB2GrayScale(img);
                Ball_Position = Contour_CM_Find_and_Display(gray_img,img,imgLines);
                saveBall_position[i][1] = Ball_Position.x;
                saveBall_position[i][3] = Ball_Position.y;
                saveBall_position[i][2] = Setpoint.x;
                saveBall_position[i][4] = Setpoint.y;


                /// Cal PID for x-axis
                xerr = (Setpoint.x - Ball_Position.x)*(-1);
                xInte = xInte + (xerr * dt);

                xDiff = (xerr - xpre_error)/dt;

                xoutput = (Kp * xerr) + (Ki * xInte) + (Kd * xDiff);
                xpre_error = xerr;
                if(abs(xoutput) > abs(prex)){
                        Max_error_x = abs(xoutput);
                }
                prex = xoutput;
                if(abs(xerr) < 40)
                        P_ang_max = 1*PI/180;
                else
                        P_ang_max = 2*PI/180;

                if(abs(xerr) < 15)
                        X_angle = Zero_angle;
                else
                        X_angle = Servo_angle(xoutput, Max_error_x);
                saveBall_position[i][5] = xoutput;
                saveBall_position[i][6] = P_ang*180/PI;
```

```
                    /// Cal PID for y-axis
                    yerr = (Setpoint.y - Ball_Position.y)*(-1);
                    yinte = yinte + (yerr * dt);
                    yDiff = (yerr - ypre_error)/dt;
                    youtput = (Kp * yerr) + (Ki * yinte) + (Kd * yDiff);

                    ypre_error = yerr;
                    if(abs(youtput) > abs(prey)){
                            Max_error_y = abs(youtput);
                    }
                    prey = youtput;
                    if(abs(yerr) < 40)
                            P_ang_max = 1*PI/180;
                    else
                            P_ang_max = 2*PI/180;

                    if(abs(yerr) < 15)
                            Y_angle = Zero_angle;
                    else
                            Y_angle = Servo_angle(youtput, Max_error_y);
                    saveBall_position[i][7] = youtput;
                    saveBall_position[i][8] = P_ang*180/PI;

                    /// Send parameter to arduino
                    Serial2(X_angle,Y_angle);

                    /// loop time check
                    T_present = clock();
                    Time = T_present - T_start;
                    saveBall_position [i][0] = (static_cast<double>(Time)/CLOCKS_PER_SEC) ;
                    dt = Time - tmp;
                    tmp = Time;
                    waitKey(1);
                    setMouseCallback("Result window", CallBackFunc, NULL);
                }
                //// reset Target
                Setpoint = Point(-1,-1);
                Target_Set = FALSE;

                //// save ball's path
                ofstream save_path("Ball_path.txt",ios::out);
                for(int k=0; saveBall_position[k][0] > -1; k++){
                    for(int j=0; j<9; j++){
                        save_path << saveBall_position[k][j] ;
                        if(j==8){
                                save_path <<'\n';
                        }
                        else save_path <<'\t' ;
                    }
                }
                //// pause programe
                waitKey(50);
                Serial2(Zero_angle,Zero_angle);
                waitKey(50);
                CloseHandle(hSerial);
            }
            waitKey(10);
        }
        //-------------------------------------------------------------------//

}

/////Modify image function
```

```
void morphOps(Mat&thresh){

        Mat erodeElement = getStructuringElement(MORPH_RECT,Size(3,3));
        Mat dilateElement = getStructuringElement(MORPH_RECT,Size(3,3));

        erode(thresh,thresh,erodeElement);
        erode(thresh,thresh,erodeElement);

        dilate(thresh,thresh,dilateElement);
        dilate(thresh,thresh,dilateElement);
}

////// Convert to Gray Scale & Modify function
MatRGB2GrayScale(Matimg){
        Mat gray;
        cvtColor(img,gray,CV_BGR2GRAY);
        inRange(gray,0,hue,gray);
        morphOps(gray);

        return gray;
}

////// Find&Display Contour&Center of mass
Point Contour_CM_Find_and_Display(Matinputimg,Matdisplayimg, MatBallpath_img){

        int Xposition = -1;                                     //
        int Yposition = -1;                                     // for find mass center
        int Ballarea = 800;
        //imshow("Gray Scale", inputimg);

        /// Find contours
        vector<vector<Point>> contours;
        vector<Vec4i> hierarchy;
        RNGrng(12345);
        findContours(inputimg, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

        if(contours.size() > 0){
                double tmp=0;
                double MaxArea=0;
                for(int j = 0; j < contours.size(); j++){
                        if(contourArea(contours[j]) > tmp){
                        tmp = contourArea(contours[j]);
                        MaxArea = j;
                        }
                }
                Moments mu;
                mu = moments(contours[MaxArea]);
                //Mass center
                if (mu.m00 > Ballarea){
                        //calculate the position of the ball
                        Xposition = mu.m10 / mu.m00;
                        Yposition = mu.m01 / mu.m00;
                        iLastX = Xposition;
                        iLastY = Yposition;
                }
                /// Draw contours
                Mat drawing = Mat::zeros(inputimg.size(), CV_8UC3 );
                if(contourArea(contours[MaxArea]) > Ballarea){
                        drawContours(displayimg, contours, MaxArea, Scalar(0,0,255), 2, 8, hierarchy, 0, Point() );
                        circle(displayimg, Point(Xposition,Yposition), 4, Scalar(0,255,0), -1, 8, 0 );
                        //draw target box
                        if(Setpoint.x > 0 && Setpoint.y > 0){
                                circle(displayimg, Setpoint, 4, Scalar(0,0,255), -1, 8, 0 );
                                line(displayimg, Point(Setpoint.x-20,Setpoint.y-20), Point(Setpoint.x+20,Setpoint.y-20), Scalar(0,255,255), 2);
```

```
                                        line(displayimg, Point(Setpoint.x+20,Setpoint.y-20), Point(Setpoint.x+20,Setpoint.y+20), Scalar(0,255,255), 2);
                                        line(displayimg, Point(Setpoint.x+20,Setpoint.y+20), Point(Setpoint.x-20,Setpoint.y+20), Scalar(0,255,255), 2);
                                        line(displayimg, Point(Setpoint.x-20,Setpoint.y+20), Point(Setpoint.x-20,Setpoint.y-20), Scalar(0,255,255), 2);
                                }
                                string poX = to_string(Xposition);
                                string poY = to_string(Yposition);
                                string Area = to_string(mu.m00);
                                putText(displayimg,"Position X :" + poX,Point(20,20),FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,255),1.15,false);
                                putText(displayimg,"Position Y :" + poY,Point(20,40),FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,255),1.15,false);
                                putText(displayimg,"Area :" + Area,Point(20,60),FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,255),1.15,false);
                                poX = to_string(Setpoint.x);
                                poY = to_string(Setpoint.y);
                                putText(displayimg," (" + poX + "," + poY + ")",Setpoint,FONT_HERSHEY_SIMPLEX,0.5,Scalar(0,0,255),1.15,false);
                                }
                        }

                        imshow("Result window", displayimg+Ballpath_img);

                        returnPoint(Xposition,Yposition);

}

///// Call back function for Get Mouse click position
void CallBackFunc(intevent, intx, inty, intflags, void* userdata)
{
if  (event == EVENT_LBUTTONDOWN )
    {
                        Setpoint.x = x;
                        Setpoint.y = y;
                        Target_Set = TRUE;
    }
                elseif  ( event == EVENT_RBUTTONDOWN )
    {
                        Target_Set = FALSE;
                }
}

int Servo_angle(doublepid, doubleMax_error){

                int Servo_angle = Zero_angle;
                P_ang = 0;

                P_ang = (pid/Max_error)*(P_ang_max);

                iffP_ang > P_ang_zero + P_ang_max)
                        P_ang = P_ang_zero + P_ang_max;
                elseif(P_ang < P_ang_zero - P_ang_max)
                        P_ang = P_ang_zero - P_ang_max;

                Servo_angle = Zero_angle + ((asin((145/15)*(sin(P_ang)))*180/PI)/0.10345);

                return Servo_angle;
}

void Serial2(intXangle, intYangle){

                        char outputChars[10];
                        for(int i = 4; i >= 0; i--){
                                if(i == 4){
                                        outputChars[i] = '-';
                                }
                                else{
                                        outputChars[i] = (Xangle % 10)+'0';
                                        Xangle /= 10;
```

```
                }
        }
        for(int j = 9; j >= 5; j--){
                if(j == 9){
                        outputChars[j] = '*';
                }
                else{
                        outputChars[j] = (Yangle % 10)+'0';
                        Yangle /= 10;
                }
        }
        cout<< outputChars << endl;
        WriteFile(hSerial, outputChars, strlen(outputChars), &btsIO, NULL);
        fill_n(outputChars, 10, '\0');
}
```

ก.2 โปรแกรมสั่งการเซอร์โวมอเตอร์ในส่วนของไมโครคอนโทรลเลอร์

โปรแกรมที่ใช้ในส่วนนี้คือ โปรแกรม Arduino IDE 1.5.8 ซึ่งเป็นโปรแกรมที่ใช้ร่วมกับบอร์ด Arduino Mega 2560 และบอร์ด Arduino รุ่นอื่นๆ โค้ดโปรแกรมมีดังต่อไปนี้

```
#include <Servo.h>

Servo Xservo, Yservo;
int Xangle = 1500;
int Yangle = 1500;
String inString = "";

void setup() {
Serial.begin(9600);
Xservo.attach(8);
Yservo.attach(12);
}

void loop() {
while (Serial.available() >0) {
int inChar = Serial.read();
if (isDigit(inChar)) {
inString += (char)inChar;
    }
if (inChar == '-') {
      Xangle = inString.toInt();
Xservo.writeMicroseconds(Xangle);
```

```
inString = "";
    }
if (inChar == '*') {
      Yangle = inString.toInt();
Yservo.writeMicroseconds(Yangle);
inString = "";
    }
 }
}
```

ภาคผนวก ข

รายละเอียดการใช้งานคำสั่งต่างๆ ของไลบรารี่ OpenCV ที่ใช้ในโปรแกรมประมวลผล

ในส่วนนี้จะอธิบายถึงฟังก์ชันต่างๆ ของ OpenCV ที่ใช้ในส่วนของโปรแกรมประมวลผลภาพ มีรายละเอียดดังนี้

VideoCapture::open

Open video file or a capturing device for video capturing

C++: bool VideoCapture::open(const string& filename)

C++: bool VideoCapture::open(int device)

Parameters:

- filename – name of the opened video file (eg. video.avi) or image sequence (eg. img_%02d.jpg, which will read samples like img_00.jpg, img_01.jpg, img_02.jpg, ...)
- device – id of the opened video capturing device (i.e. a camera index).

waitKey

Waits for a pressed key.

C++: int waitKey(int delay=0)

Parameters: delay – Delay in milliseconds. 0 is the special value that means “forever”.

namedWindow

Creates a window.

C++: void namedWindow(const string& winname, int flags=WINDOW_AUTOSIZE)

Parameters:

- name – Name of the window in the window caption that may be used as a window identifier.
- flags –Flags of the window. The supported flags are:
  - WINDOW_NORMAL If this is set, the user can resize the window

(no constraint).

- WINDOW_AUTOSIZE If this is set, the window size is automatically adjusted to fit the displayed image (see imshow()), and you cannot change the window size manually.
- WINDOW_OPENGL If this is set, the window will be created with OpenGL support.

createTrackbar

Creates a trackbar and attaches it to the specified window.

C++: int **createTrackbar**(const string& **trackbarname**, const string& **winname**, int* **value**, int **count**, TrackbarCallback **onChange**=0, void* **userdata**=0)

C: int **cvCreateTrackbar**(const char* **trackbar_name**, const char* **window_name**, int* **value**, int **count**, CvTrackbarCallback **on_change**=NULL)

Python: cv.**CreateTrackbar**(trackbarName, windowName, value, count, onChange) → None

Parameters

- **trackbarname** – Name of the created trackbar.
- **winname** – Name of the window that will be used as a parent of the created trackbar.
- **value** – Optional pointer to an integer variable whose value reflects the position of the slider. Upon creation, the slider position is defined by this variable.

:

- **count** – Maximal position of the slider. The minimal position is always 0.
- **onChange** – Pointer to the function to be called every time the slider changes position. This function should be prototyped as void Foo(int,void*); , where the first parameter is the trackbar position and the second parameter is the user data (see the next parameter). If the callback is the NULL pointer, no callbacks are called, but only value is

updated.

- **userdata** – User data that is passed as is to the callback. It can be used to handle trackbar events without using global variables.

cvtColor

Converts an image from one color space to another.

C++: void **cvtColor**(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)

Parameters:

- **src** – input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point.
- **dst** – output image of the same size and depth as src.
- **code** – color space conversion code (see the description below).
- **dstCn** – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from src and code .

erode

Erodes an image by using a specific structuring element.

C++: void **erode**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar&**borderValue**=morphologyDefaultBorderValue())

Parameters:

- **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV_8U, CV_16U, CV_16S, CV_32F` or ``CV_64F.
- **dst** – output image of the same size and type as src.
- **element** – structuring element used for erosion; if element=Mat() , a 3 x 3 rectangular structuring element is used.
- **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center.

- **iterations** – number of times erosion is applied.
- **borderType** – pixel extrapolation method (see borderInterpolate() for details).
- **borderValue** – border value in case of a constant border (see createMorphologyFilter() for details).

dilate

Dilates an image by using a specific structuring element.

C++: void **dilate**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar&**borderValue**=morphologyDefaultBorderValue())

Parameters:

- **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV_8U, CV_16U, CV_16S, CV_32F` or ``CV_64F.
- **dst** – output image of the same size and type as src.
- **element** – structuring element used for dilation; if element=Mat() , a 3 x 3 rectangular structuring element is used.
- **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center.
- **iterations** – number of times dilation is applied.
- **borderType** – pixel extrapolation method (see borderInterpolate() for details).
- **borderValue** – border value in case of a constant border (see createMorphologyFilter() for details).

inRange

Checks if array elements lie between the elements of two other arrays.

C++: void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst)

Parameters:

- src – first input array.
- lowerb – inclusive lower boundary array or a scalar.
- upperb – inclusive upper boundary array or a scalar.
- dst – output array of the same size as src and CV_8U ty

findContours

Finds contours in a binary image.

C++: void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())

C++: void findContours(InputOutputArray image, OutputArrayOfArrays contours, int mode, int method, Point offset=Point())

Parameters:

- image – Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary . You can use compare(), inRange() , threshold() , adaptiveThreshold() , Canny() , and others to create a binary image out of a grayscale or color one. The function modifies the imagewhile extracting the contours. If mode equals to CV_RETR_CCOMP or CV_RETR_FLOODFILL, the input can also be a 32-bit integer image of labels (CV_32SC1).
- contours – Detected contours. Each contour is stored as a vector of points.
- hierarchy – Optional output vector, containing information about the image topology. It has as many elements as the number of contours. For each i-th contourcontours[i] , the elements hierarchy[i][0] , hiearchy[i][1] , hiearchy[i][2] ,

and hiearchy[i][3] are set to 0-based indices in contours of the next and previous contours at the same hierarchical level, the first child contour and the parent contour, respectively. If for the contour i there are no next, previous, parent, or nested contours, the corresponding elements of hierarchy[i] will be negative.

- mode –Contour retrieval mode (if you use Python see also a note below).
    - CV_RETR_EXTERNAL retrieves only the extreme outer contours. It sets hierarchy[i][2]=hierarchy[i][3]=-1 for all the contours.
    - CV_RETR_LIST retrieves all of the contours without establishing any hierarchical relationships.
    - CV_RETR_CCOMP retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
    - CV_RETR_TREE retrieves all of the contours and reconstructs a full hierarchy of nested contours. This full hierarchy is built and shown in the OpenCV contours.cdemo.
- method –Contour approximation method (if you use Python see also a note below).
    - CV_CHAIN_APPROX_NONE stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, max(abs(x1-x2),abs(y2-y1))==1.
    - CV_CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
    - CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS a pplies one of the flavors of the Teh-Chin chain approximation algorithm. See [TehChin89]for details.

- **offset** – Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

moments

Calculates all of the moments up to the third order of a polygon or rasterized shape.

C++: Moments **moments**(InputArray **array**, bool **binaryImage**=false)

Parameters:

- **array** – Raster image (single-channel, 8-bit or floating-point 2D array) or an array (1xN or Nx1) of 2D points (Point or Point2f).
- **binaryImage** – If it is true, all non-zero image pixels are treated as 1's. The parameter is used for images only.
- **moments** – Output moments.

drawContours

Draws contours outlines or filled contours.

C++: void **drawContours**(InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray**hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point())

Parameters:

- **image** – Destination image.
- **contours** – All the input contours. Each contour is stored as a point vector.
- **contourIdx** – Parameter indicating a contour to draw. If it is negative, all the contours are drawn.
- **color** – Color of the contours.
- **thickness** – Thickness of lines the contours are drawn with. If it is negative (for example, thickness=CV_FILLED), the contour interiors are drawn.
- **lineType** – Line connectivity. See line() for details.

- **hierarchy** – Optional information about hierarchy. It is only needed if you want to draw only some of the contours (see maxLevel).
- **maxLevel** – Maximal level for drawn contours. If it is 0, only the specified contour is drawn. If it is 1, the function draws the contour(s) and all the nested contours. If it is 2, the function draws the contours, all the nested contours, all the nested-to-nested contours, and so on. This parameter is only taken into account when there ishierarchy available.
- **offset** – Optional contour shift parameter. Shift all the drawn contours by the specified offset = (dx, dy) .
- **contour** – Pointer to the first contour.
- **externalColor** – Color of external contours.
- **holeColor** – Color of internal contours (holes).

circle

Draws a circle.

C++: void **circle**(Mat& **img**, Point **center**, int **radius**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Parameters:

- **img** – Image where the circle is drawn.
- **center** – Center of the circle.
- **radius** – Radius of the circle.
- **color** – Circle color.
- **thickness** – Thickness of the circle outline, if positive. Negative thickness means that a filled circle is to be drawn.
- **lineType** – Type of the circle boundary. See the line() description.
- **shift** – Number of fractional bits in the coordinates of the center and in the radius value.

line

Draws a line segment connecting two points.

C++: void **line**(Mat& **img**, Point **pt1**, Point **pt2**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, int **shift**=0)

Parameters:

- **img** – Image.
- **pt1** – First point of the line segment.
- **pt2** – Second point of the line segment.
- **color** – Line color.
- **thickness** – Line thickness.
- **lineType** –Type of the line:
  - 8 (or omitted) - 8-connected line.
  - 4 - 4-connected line.
  - **CV_AA** - antialiased line.
- **shift** – Number of fractional bits in the point coordinates.

putText

Draws a text string.

C++: void **putText**(Mat& **img**, const string& **text**, Point **org**, int **fontFace**, double **fontScale**, Scalar **color**, int **thickness**=1, int **lineType**=8, bool **bottomLeftOrigin**=false)

Parameters:

- **img** – Image.
- **text** – Text string to be drawn.
- **org** – Bottom-left corner of the text string in the image.
- **font** – CvFont structure initialized using InitFont().
- **fontFace** – Font type. One of FONT_HERSHEY_SIMPLEX, FONT_HERSHEY_PLAIN, FONT_HERSHEY_DUPL

EX, FONT_HERSHEY_COMPLEX, FONT_HERSHEY_TRIPLEX,FONT_HERSHEY_COMPLEX_SMALL, FONT_HERSHEY_SCRIPT_SIMPLEX,
or FONT_HERSHEY_SCRIPT_COMPLEX, where each of the font ID's can be combined withFONT_ITALIC to get the slanted letters.

- **fontScale** – Font scale factor that is multiplied by the font-specific base size.
- **color** – Text color.
- **thickness** – Thickness of the lines used to draw a text.
- **lineType** – Line type. See the line for details.
- **bottomLeftOrigin** – When true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.

imshow

Displays an image in the specified window.

C++: void **imshow**(const string& **winname**, InputArray **mat**)

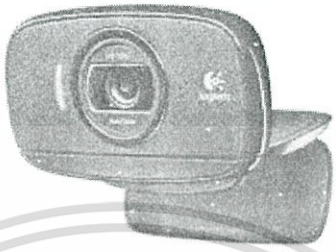
Parameters:

- **winname** – Name of the window.
- **image** – Image to be shown.

ภาคผนวก ค

เอกสารคู่มืออุปกรณ์ต่างๆ

ค.1 กล้องเว็บแคม



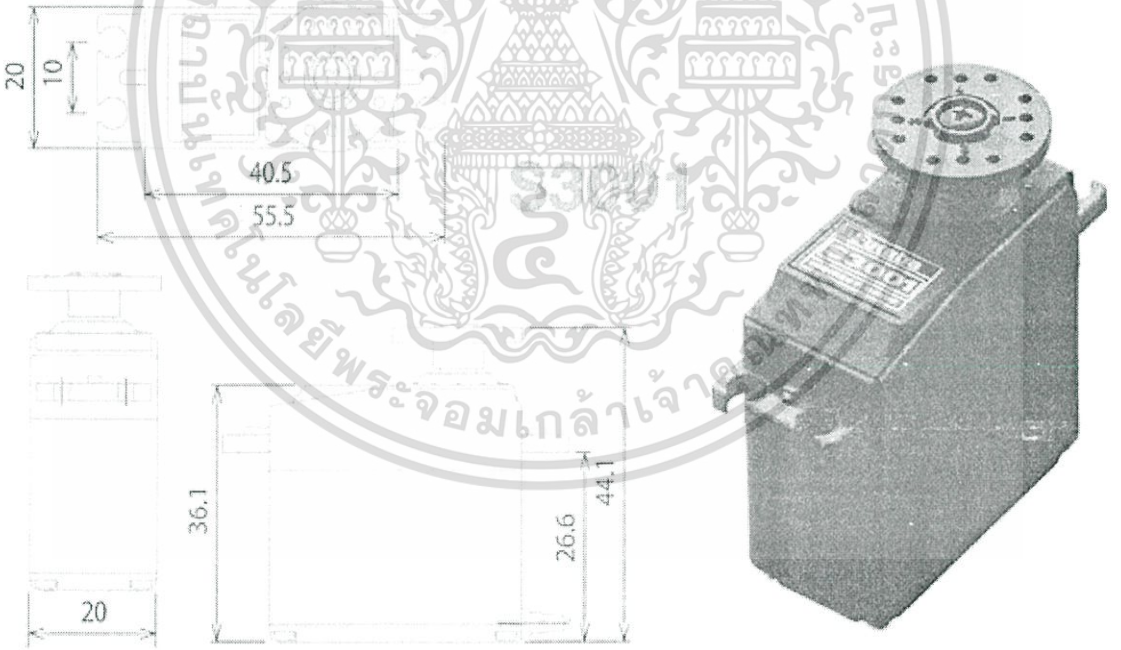
Specifications	
Product Code	960000721
Category	Webcam
OS Support (at release)	Windows XP x32, Windows XP x64, Windows Vista x32, Windows Vista x64, Windows 7 x32, Windows 7 x64
Connection Type	USB
USB Type	USB 2.0
USB VID_PID	081A
UVC Support	Yes
Microphone	Yes
Microphone Type	Mono
Lens and Sensor Type	Plastic
Focus Type	Auto
Optical Resolution	2MP, True, 8MP Software enhanced
Diagonal Field of View (FOV)	69°
Focal Length	N/A
Image Capture (4:3 SD)	320x240, 640x480, 2MP, 8.0MP
Image Capture (16:9 W)	360p, 480p, 720p
Video Capture (4:3 SD)	320x240, 640x480, 2MP
Video Capture (16:9 W)	360p, 480p, 720p*, 720p
Frame Rate (max)	30fps@640x480
Right Light	Yes
Video Effects (VFX)	Avatars, Fun Filters, Masks
Buttons	N/A
Indicator Lights (LED)	Yes
Privacy Shade	No
Tripod Mounting Option	No
Universal Clip Adjustability (range)	48.94mm
Cable Length	6 Feet, 183 CM
Width	40.4mm
Depth/Length	68.5mm
Height	31.75mm
Weight	88g

©2014 ComXpert International CC

ค.2 เซอร์โวมอเตอร์ Futaba S3001

Basic Information

Modulation:	Analog
Torque:	**4.8V:** 33.0 oz-in (2.38 kg-cm) **6.0V:** 42.0 oz-in (3.02 kg-cm)
Speed:	**4.8V:** 0.28 sec/60° **6.0V:** 0.22 sec/60°
Weight:	1.59 oz (45.0 g)
Dimensions:	Length: 1.57 in (39.9 mm) Width: 0.79 in (20.1 mm) Height: 1.42 in (36.1 mm)
Motor Type:	3-pole
Gear Type:	Plastic
Rotation/Support:	Single Bearing



ภาคผนวก ง

ตารางรหัสแอสกี้

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACK SPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[END OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]