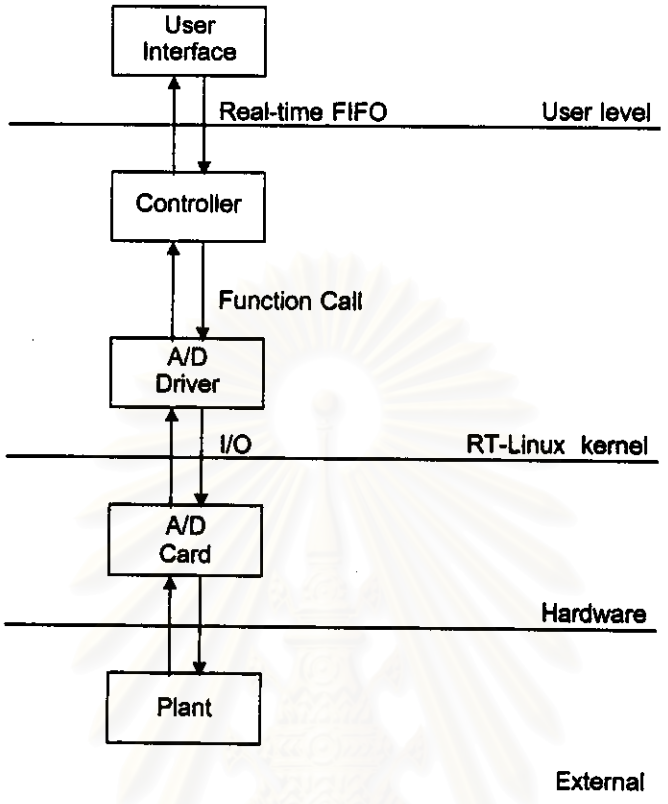
บทที่ 4
ซอฟท์แวร์ตัวควบคุมโดยตรง

หลังจากพิจารณาถึงความเป็นไปได้ในการใช้ RT-Linux เป็นระบบปฏิบัติการสำหรับใช้ในงานวิทยานิพนธ์ไปแล้วในบทที่ 3 บทนี้จะกล่าวถึงการออกแบบซอฟท์แวร์ตัวควบคุมโดยตรง การแบ่งองค์ประกอบหลัก และการพัฒนาโปรแกรมเพื่อใช้เป็นกรณีศึกษา ซึ่งในที่นี้คือโปรแกรมควบคุมกระบวนการแลกเปลี่ยนความร้อน (Heat Exchanger) สำหรับวิธีการที่ใช้ในการควบคุมเป็นแบบพีไอดีที่ใช้งานในอุตสาหกรรมทั่วไป ในตอนท้ายจะอธิบายเกี่ยวกับไลบรารีที่เขียนขึ้นสำหรับใช้งานกับกฎการควบคุมแบบพร็อพเพอร์ (proper)

4.1 การแบ่งองค์ประกอบหลัก

การพัฒนาซอฟท์แวร์ตัวควบคุมโดยตรง แบ่งงานได้เป็นสามส่วนใหญ่ๆ คือ 1) โปรแกรมควบคุมการทำงานของการ์ดแปลงสัญญาณ (A/D-D/A driver) 2) โปรแกรมควบคุมระบบ (controller) และ 3) โปรแกรมติดต่อกับผู้ใช้ (User Interface) โดยสองส่วนแรกเป็นงานแบบเวลาจริง สามารถแสดงเป็นแผนภาพในการเชื่อมต่อกับการ์ดแปลงสัญญาณและระบบจริง (plant) ได้ดังรูปที่ 4.1

การแบ่งงานเป็นสามส่วนทำให้สามารถพัฒนาแยกส่วนกันได้ รวมถึงการเปลี่ยนแปลงในภายหลังอีกด้วย โดยมีการกำหนดข้อตกลงในการเชื่อมต่อดังนี้คือ โปรแกรมควบคุมการ์ดแปลงสัญญาณ รับและส่งข้อมูลระหว่างโปรแกรมและตัวการ์ดโดยชุดคำสั่ง I/O โปรแกรมควบคุมระบบใช้งานโปรแกรมควบคุมการ์ดแปลงสัญญาณโดยการเรียกใช้ฟังก์ชัน (Function Call) และระหว่างโปรแกรมควบคุมระบบ และส่วนติดต่อกับผู้ใช้ รับและส่งข้อมูล (ในที่นี้คือคำสั่งจากผู้ใช้และสภาวะของระบบ) โดยผ่านไฟล์แบบท่อ (FIFO) ซึ่งมีลักษณะคือ ข้อมูลที่เข้ามาก่อนจะออกไปก่อน (First In First Out) ไฟล์ชนิดนี้เป็นส่วนหนึ่งของการพัฒนาระบบปฏิบัติการ RT-Linux เพื่อใช้ในการติดต่อระหว่างโปรแกรมซึ่งทำงานแบบเวลาจริง และโปรแกรมซึ่งทำงานแบบปกติ รายละเอียดสำหรับงานในแต่ละส่วนมีดังนี้คือ



รูปที่ 4.1 ระบบควบคุมโดยตรง

4.1.1 โปรแกรมควบคุมการ์ดแปลงสัญญาณ (A/D-D/A Driver)

เป็นโปรแกรมที่ทำหน้าที่ควบคุมการทำงานของการ์ดแปลงสัญญาณ (A/D-D/A Card) เขียนด้วยภาษา C มีลักษณะเป็นมอดูล (module) ซึ่งจะถูกบรรจุเข้าเป็นส่วนหนึ่งของระบบปฏิบัติการในเวลาที่จะใช้งาน ประกอบด้วยฟังก์ชันการทำงานซึ่งโปรแกรมอื่นสามารถเรียกใช้ได้ดังนี้

- int din(void)

ทำหน้าที่สั่งงานให้การ์ดรับค่าสัญญาณเชิงเลขเข้ามา ไม่รับอาร์กิวเมนต์ เนื่องจากการ์ดส่วนใหญ่มักจะมีพอร์ตสัญญาณเชิงเลขเพียงพอร์ตเดียว โดยรับสัญญาณได้มากที่สุด 16 บิต ส่งค่ากลับเป็นจำนวนเต็มซึ่งมีบิตน้อยที่สุด (LSB) เป็นค่าสัญญาณจากขาที่ 1 เรียงลำดับไปจนถึงบิตมากที่สุด (MSB) เป็นค่าสัญญาณจากขาที่ 16

- int dout(int data)

ทำหน้าที่สั่งงานให้การ์ดส่งสัญญาณเชิงเลข data ออกไป และส่งค่ากลับเพื่อบอกความสำเร็จในการทำงาน

- int adin(int ch)

ทำหน้าที่สั่งงานให้การ์ดรับค่าสัญญาณเข้าทางช่องสัญญาณอนาลอกที่ ch แล้วแปลงสัญญาณอนาลอกที่ได้เป็นสัญญาณเชิงเลขเพื่อส่งค่ากลับ

- int daout(int ch, int data)

ทำหน้าที่สั่งงานให้การ์ดแปลงสัญญาณเชิงเลข data ให้เป็นสัญญาณอนาลอกแล้วส่งออกทางช่องสัญญาณออกของอนาลอกที่ ch และส่งค่ากลับเป็นความสำเร็จในการทำงาน

- double itov(int i)

ทำหน้าที่แปลงค่าสัญญาณเชิงเลข i ให้เป็นค่าแรงดันเทียบเท่ากับสัญญาณอนาลอกจริงเพื่อส่งค่ากลับ

- int vtoi(double v)

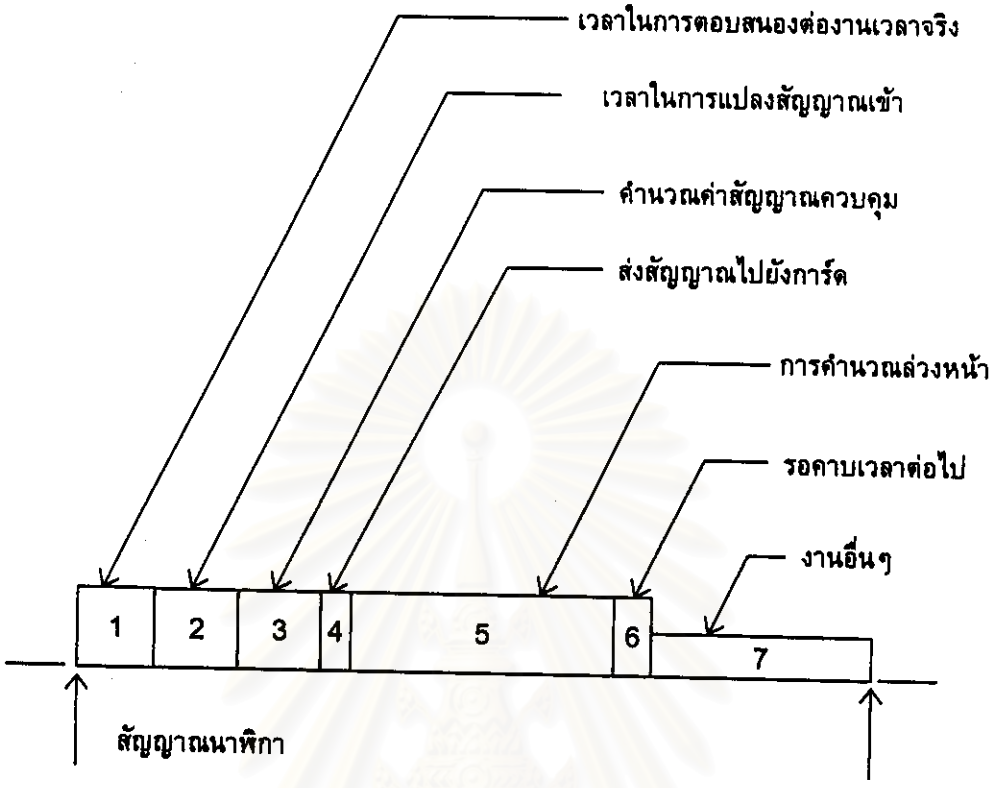
ทำหน้าที่แปลงค่าแรงดัน v ให้เป็นค่าสัญญาณเชิงเลขเพื่อส่งค่ากลับ

ในการใช้งานการ์ดแปลงสัญญาณ (A/D-D/A card) รุ่นต่าง ๆ ทำได้โดยการเขียนฟังก์ชัน การทำงานต่าง ๆ ข้างต้นให้ทำงานตามที่กำหนด การเขียนฟังก์ชันสำหรับการ์ดแต่ละรุ่นนั้น จะแตกต่างกันในรายละเอียด เช่นวิธีการรับ-ส่งข้อมูล, พอร์ตอินพุตเอาท์พุตและค่าอินเทอร์รัพที่ใช้ เป็นต้น

4.1.2 โปรแกรมควบคุมระบบ (controller)

เป็นโปรแกรมที่ทำหน้าที่ควบคุมการทำงานของแพลนต์ (Plant) โดยรับและส่งข้อมูลผ่านการ์ดแปลงสัญญาณ (A/D-D/A card) ซึ่งถูกควบคุมโดยโปรแกรมควบคุมการ์ด (A/D-D/A driver) โปรแกรมเขียนด้วยภาษา C และมีลักษณะเป็นมอดูล เช่นเดียวกัน ลักษณะของโปรแกรมจะประกอบด้วยวงรอบการทำงานหลัก ๆ 3 ส่วน ดังนี้

- วงรอบการควบคุม (Control Loop) มีลักษณะการทำงานเป็นคาบ สามารถแสดงเป็นแผนภาพได้ดังรูปที่ 4.2 [11] โดยแต่ละคาบจะเริ่มจาก (1) เวลาในการสับเปลี่ยนงานของตัวจัดการงาน (2) การป้อนกลับสถานะของระบบ (3) นำมาคำนวณค่าสัญญาณควบคุม แล้ว (4) ส่งค่าที่ได้กลับออกไป และ (5) ทำการคำนวณล่วงหน้าที่ไม่ได้อาศัยสถานะของระบบในการสุ่มตัวอย่างรอบถัดไป จากนั้น (6) รอจนกว่าจะถึงเวลาในคาบถัดไปจึงกลับไปเริ่มต้นใหม่ ซึ่งในระหว่างนี้ (7) ตัวจัดการงานจะทำการสับเปลี่ยนไปทำงานอื่นก่อน



รูปที่ 4.2 กรอบเวลาสำหรับการควบคุม

- วงรอบการรับคำสั่ง (Operation Loop) มีลักษณะการทำงานเป็นคาบ โดยในแต่ละคาบจะคอยเช็คไฟล์คำสั่ง (Operation FIFO) ว่ามีคำสั่งถูกส่งผ่านมาหรือไม่ ถ้ามีจะทำตามคำสั่งทีละหนึ่งคำสั่ง ตัวอย่างของคำสั่ง เช่น การสั่งเริ่มต้นการควบคุม, การเปลี่ยนค่าคงที่ต่างๆ ของระบบ (ได้แก่ค่า Pb, Ti, Td สำหรับวิธีการแบบพีไอดี), การสั่งหยุดการควบคุมเป็นต้น ถ้าไม่มีคำสั่งถูกส่งผ่านมาก็จะไม่มีการทำงานและรอจนกว่าจะถึงคาบเวลาถัดไป

- วงรอบการส่งค่าเพื่อแสดงผล (Display Loop) มีลักษณะการทำงานเป็นคาบเวลาเช่นเดียวกัน โดยจะคอยเก็บค่าสถานะของระบบลงสู่ไฟล์แสดงผล (Display FIFO) เพื่อให้โปรแกรมแสดงผลนำไปใช้ต่อไป

ไฟล์คำสั่ง (Operation Fifo) และไฟล์แสดงผล (Display Fifo) เป็นส่วนประกอบของ Real-time FIFO ในรูปที่ 4.1 ซึ่งเป็นทางส่งผ่านข้อมูลระหว่างโปรแกรมควบคุมระบบ (Controller) และโปรแกรมติดต่อกับผู้ใช้ (User Interface)

4.1.3 โปรแกรมติดต่อกับผู้ใช้ (User Interface)

เป็นโปรแกรมที่ทำหน้าที่ติดต่อกับผู้ควบคุม โดยจะแสดงค่าสถานะของระบบซึ่งถูกส่งผ่านมาทางไฟล์แสดงผล (Display FIFO) ในรูปแบบต่างๆ เช่นกราฟผลตอบของระบบ เป็นต้น ผู้ใช้สามารถเปลี่ยนแปลงค่าพารามิเตอร์ในการควบคุมต่าง ๆ ได้ ซึ่งโปรแกรมจะส่งคำสั่งเหล่านี้ต่อไปยังโปรแกรมควบคุมระบบผ่านทางไฟล์คำสั่ง (Operation FIFO)

ค่าสูงสุดของคิว (FIFO) ที่สามารถใช้งานได้สูงสุดปกติจะเท่ากับ 64 [5] ค่าสูงสุดนี้สามารถเปลี่ยนแปลงได้ โดยจะต้องกันเนื่อที่หน่วยความจำสำหรับใช้งานเพิ่มเติม

4.2 กรณีศึกษา

ในงานวิทยานิพนธ์รวมถึงการพัฒนาระบบควบคุมโดยตรง ที่มีโครงสร้างตามข้อ 4.1 ซึ่งสามารถควบคุมกระบวนการจริงได้ สำหรับส่วนประกอบต่าง ๆ มีดังต่อไปนี้

4.2.1 กระบวนการ (Plant)

กระบวนการที่เลือกใช้เป็นกระบวนการแลกเปลี่ยนความร้อน (Heat Exchanger) ซึ่งสามารถประมาณเป็นระบบอันดับหนึ่งที่มีความล้าหลังทางเวลาได้ และตัวจำลองกระบวนการ (Process Simulator) ซึ่งเป็นระบบอันดับสอง และทั้งสองระบบมีแรงดันช่วงทำงานอยู่ในช่วงประมาณ -10 ถึง 10 โวลท์

4.2.2 การ์ดแปลงสัญญาณ (A/D-D/A card)

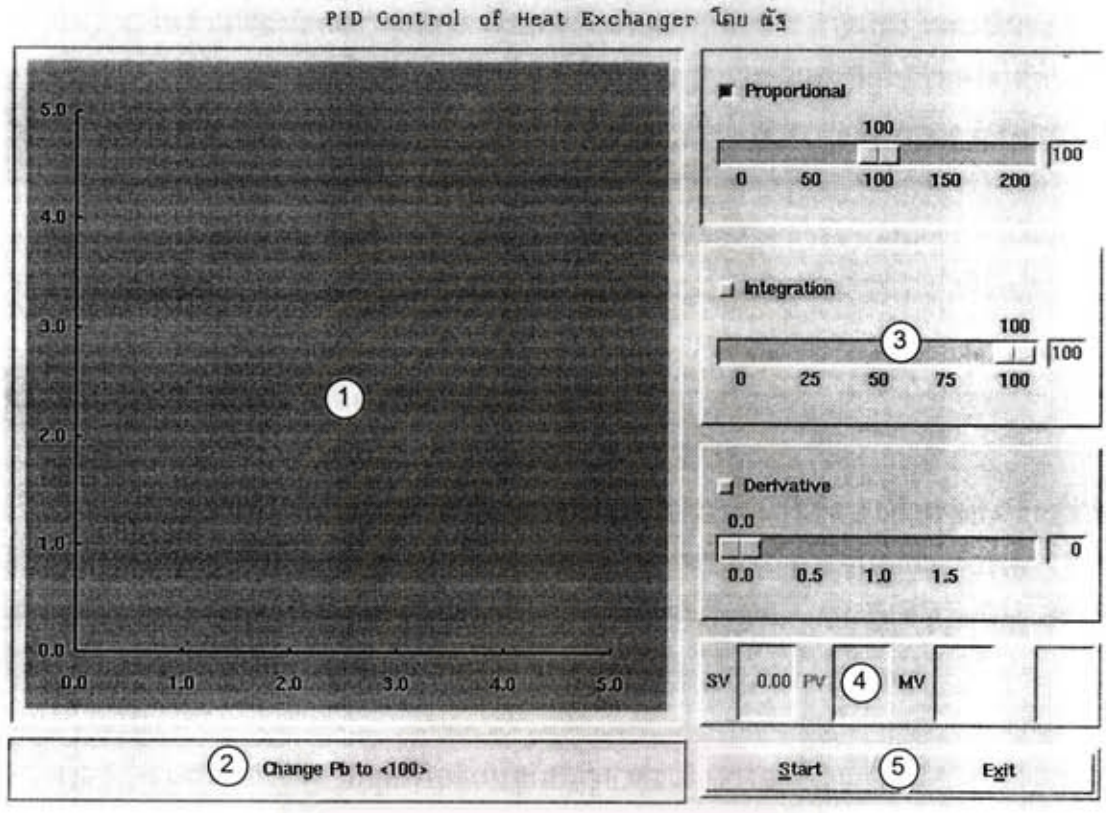
ในงานวิทยานิพนธ์ เลือกใช้การ์ดรุ่น PCLab-Card ทำหน้าที่แปลงสัญญาณระหว่างอนาลอก (analog) และสัญญาณเชิงเลข (digital) มีรายละเอียดดังนี้คือ มีความละเอียด 12 บิต รับสัญญาณเข้าได้ 16 ช่องแบบสองขั้ว (bipolar) เลือกขอบเขตของสัญญาณเข้าได้ 4 แบบ คือ +-1, +-2, +-5, +-10 โวลท์ ความถี่สูงสุดในการชักตัวอย่างเท่ากับ 30 กิโลเฮิร์ด สามารถส่งสัญญาณออกแบบขั้วเดียว (unipolar) ได้ 2 ช่อง ใช้เวลาเข้าที่ (settling time) 30 ไมโครวินาที และสามารถเลือกสัญญาณอ้างอิงจากภายในซึ่งให้สัญญาณออก 0-5 โวลท์ หรือใช้สัญญาณอ้างอิงจากภายนอกซึ่งให้สัญญาณออกค่าต่าง ๆ ได้

4.2.3 รายละเอียดโปรแกรม

โปรแกรมควบคุมการ์ดแปลงสัญญาณ เขียนด้วยภาษา C มีลักษณะเป็นมอดูลซึ่งจะต้องถูกบรรจุเข้าสู่ระบบปฏิบัติการก่อนการใช้งาน โดยมีฟังก์ชันการทำงานตามที่ได้กำหนดไว้ในหัวข้อ 4.1.1

โปรแกรมควบคุมระบบ เขียนด้วยภาษา C เช่นเดียวกัน โดยมีขั้นตอนวิธีตามที่กำหนดไว้ในหัวข้อ 4.1.2 สำหรับในส่วนของวิธีการควบคุมนั้นจะเป็นแบบพีไอดี (PID) แบบที่ใช้ในอุตสาหกรรมทั่วไป รายละเอียดของวิธีการควบคุมแบบพีไอดีแสดงไว้ในหัวข้อ 4.3

ส่วนโปรแกรมติดต่อกับผู้ใช้เป็นแบบกราฟฟิกเขียนด้วยภาษา Tcl/Tk ซึ่งทำงานบน X-windows มีลักษณะดังรูปที่ 4.3



รูปที่ 4.3 โปรแกรมส่วนติดต่อกับผู้ใช้

รายละเอียดของโปรแกรมสามารถอธิบายได้ดังต่อไปนี้ คือ

(1) ส่วนแสดงกราฟ

ในส่วนนี้จะแสดงกราฟสเตตของระบบซึ่งประกอบด้วย สัญญาณอ้างอิง (SV), สัญญาณออกของระบบ (PV), สัญญาณควบคุม (MV), สัญญาณความผิดผลาด (DV) ซึ่งเป็นผลต่างระหว่างสัญญาณอ้างอิงกับสัญญาณออกของระบบ ในช่วงเวลา 5 วินาทีล่าสุดที่อ่านได้จากไฟล์แสดงผล (Display FIFO)

(2) ส่วนข้อความแสดงการทำงาน

ในส่วนนี้จะเป็นแถบข้อความแสดงการทำงานต่าง ๆ เช่น การเปลี่ยนพารามิเตอร์ของตัวควบคุม การสั่งเริ่มต้นการควบคุม การสั่งหยุดการควบคุม เป็นต้น

(3) ส่วนการปรับค่าพารามิเตอร์

ในส่วนนี้จะเป็นแถบเครื่องมือที่ใช้สำหรับการเปลี่ยนแปลงพารามิเตอร์ของตัวควบคุมแบบสัดส่วน (P), ตัวควบคุมแบบอินทิเกรต (I) และตัวควบคุมแบบอนุพันธ์ (D)

(4) ส่วนแสดงค่าสเตตของระบบ

ในส่วนนี้จะแสดงค่าสเตตของระบบเป็นตัวเลขทศนิยม 2 ตำแหน่ง ในกรณีของ PV, MV, DV จะแสดงค่าซึ่งอ่านจากไฟล์แสดงผล ส่วน SV จะเป็นค่าที่ผู้ใช้กำหนดให้กับตัวควบคุม

(5) ส่วนควบคุมโปรแกรม

ประกอบด้วยปุ่มสั่งการ 2 ปุ่มคือ ปุ่มสั่งเริ่มต้นหรือหยุดการควบคุม และปุ่มออกจากโปรแกรม

4.3 วิธีการควบคุมแบบพีไอดี (PID algorithm)

การควบคุมแบบพีไอดีประกอบด้วยตัวควบคุม 3 ส่วนประกอบกันคือ 1) ตัวควบคุมสัดส่วน (Proportional part) ซึ่งมีผลต่อส่วนพุ่งเกินของผลตอบของระบบ 2) ตัวควบคุมอินทิเกรต (Integral part) ซึ่งลดความผิดพลาดในสภาวะอยู่ตัว และ 3) ตัวควบคุมอนุพันธ์ (Derivative part) ซึ่งจะช่วยให้ระบบมีผลตอบเร็วขึ้น

4.3.1 สมการของตัวควบคุมพีไอดีในโดเมนเวลาต่อเนื่อง [12]

$$U(s) = K [bUc(s) - Y(s) + (1/sTi)(Uc(s) - Y(s)) - (sTd/(1 + sTd/N))Y(s)]$$

โดย U = สัญญาณควบคุม

Uc = สัญญาณสั่งการ (command signal)

Y = สัญญาณออกของระบบที่วัดได้ (output signal)

K = อัตราขยายของตัวควบคุม = 100/Pb

Pb = อัตราส่วนของอัตราขยายต่อ 100

b = เศษส่วนของสัญญาณสั่งการซึ่งมีผลต่อตัวควบคุมสัดส่วน

Ti = ค่าคงที่ของตัวควบคุมอินทิเกรต

Td = ค่าคงที่ของตัวควบคุมอนุพันธ์

N = ค่าลิมิตของตัวควบคุมอนุพันธ์เมื่อความถี่สูง

4.3.2 การประมาณเป็นสมการไม่ต่อเนื่อง

- ตัวควบคุมสัดส่วน (Proportional part) ไม่จำเป็นต้องอาศัยการประมาณ

$$P(t) = K(bUc(t) - y(t))$$

- ตัวควบคุมอินทิเกรต (Integral part)

$$I(t) = (K/Ti)\int e(s)\,ds,\ e = Uc(t) - Y(t)$$

โดยใช้การประมาณแบบผลต่างสืบเนื่องไปข้างหน้า (forward differences)

$$I(kh+h) = I(kh) + (Kh/Ti)e(kh)$$

- ตัวควบคุมอนุพันธ์ (Derivative part)

$$(Td/N)(dD/Dt) + D = -KTd(dy/dt)$$

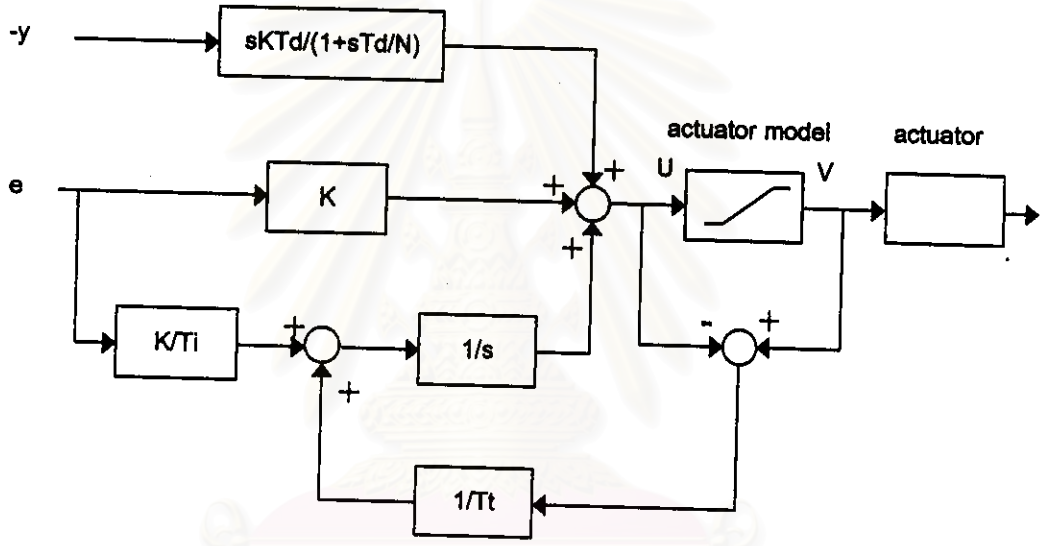
โดยใช้การประมาณแบบผลต่างสืบเนื่องย้อนกลับ (backward differences)

$$D(kh) = (Td/(Td+Nh))D(kh-h) - (KTdN/(Td+Nh))(y(kh)-y(kh-h))$$

4.3.3 ผลเสียของตัวอินทิเกรตเมื่อเกิดการอิ่มตัว (Integrator Windup)

เมื่อมีการใช้ตัวควบคุมแบบอินทิเกรต ควบคู่กับตัวกระตุ้น (actuator) ที่มีการอิ่มตัวได้ จะทำให้เกิดผลเสียซึ่งไม่สามารถทำนายได้ เนื่องจากถ้าสัญญาณผิดผลาดมีค่ามาก ตัวควบคุมอินทิเกรตจะมีค่ามากตามไปด้วย และจะส่งผลให้เกิดการอิ่มตัวของตัวกระตุ้น ตัวควบคุมอินทิเกรตซึ่งไม่มีเสถียรภาพอยู่แล้ว จะอินทิเกรตจนถึงค่าสูงมาก เมื่อสัญญาณผิดผลาดลดลง ตัวควบคุมอินทิเกรตจะมีค่าสูงมากและจะต้องใช้เวลานานกว่าจะคืนสู่ปกติ ปรากฏการณ์นี้เรียกว่า Integrator Windup มีหลายวิธีที่จะหลีกเลี่ยงเหตุการณ์นี้ วิธีหนึ่งคือการหยุดการอินทิเกรตเมื่อตัวกระตุ้นอิ่มตัว อีกวิธีหนึ่งซึ่งเป็นวิธีที่ใช้ในวิทยานิพนธ์ [12] แสดงได้ดังรูปที่ 4.4 คือ มีการป้อนกลับเพิ่มขึ้นอีกหนึ่งทาง โดยสัญญาณที่ป้อนกลับจะเป็นค่าผลต่างระหว่างสัญญาณออกของโมเดลตัวกระตุ้นและสัญญาณออกของตัวควบคุม มีเกณฑ์ในการป้อนกลับ 1/Tt ซึ่งค่าผลต่างนี้จะเป็นศูนย์เมื่อไม่เกิดการอิ่มตัวของตัวกระตุ้น และเมื่อเกิดการอิ่มตัว

การป้อนกลับที่เพิ่มขึ้นใหม่นี้จะพยายามทำให้ผลต่างมีค่าเป็นศูนย์ อาจกล่าวได้อีกอย่างหนึ่งว่า ตัวควบคุมอินทิเกรตจะถูกตั้งค่าใหม่จนกระทั่งตัวกระตุ้นไม่อิ่มตัวด้วยค่าคงที่ทางเวลา Tt หรือเรียกว่า tracking-time constant ข้อได้เปรียบของวิธีการนี้คือ ไม่เพียงแต่สามารถใช้กับตัวกระตุ้นที่มีการอิ่มตัวเท่านั้น ยังสามารถใช้ได้กับตัวกระตุ้นแบบอื่น ๆ ได้ เช่น ตัวกระตุ้นที่มีเขตไร้ผลสนอง เป็นต้น โดยมีข้อแม้ว่าจะต้องจำลองโมเดลของตัวกระตุ้นได้ หรืออาจทำการวัดสัญญาณออกของตัวกระตุ้นโดยตรงเพื่อป้อนกลับก็ได้ ซึ่งวิธีนี้จะไม่ต้องจำลองโมเดลของตัวกระตุ้น



รูปที่ 4.4 ตัวควบคุมพีไอดีซึ่งมีการแก้ไขผลเสียจากการอิ่มตัวของตัวกระตุ้น เมื่อค่า b = 1

การแก้ไขผลเสียจากการอิ่มตัวของตัวกระตุ้น ทำให้สมการของตัวควบคุมอินทิเกรตเปลี่ยนไปคือ

$$I(kh+h) = I(kh) + (Kh/Ti)e(kh) - (1/Tt)(U(kh) - V(kh))$$

โดย U = สัญญาณควบคุม

V = สัญญาณควบคุมที่ผ่านโมเดลจำลองของตัวกระตุ้น

Tt = tracking-time constant

4.4 ไลบรารีสำหรับตัวควบคุมแบบพร็อพเพอร์

ในวิทยานิพนธ์รวมถึงไลบรารีสำหรับใช้ในการควบคุมระบบที่มีกฎการควบคุมเป็นแบบพร็อพเพอร์ (proper control law) และแบบ สตริกท์ลีพร็อพเพอร์ (strictly proper control law) โดย กฎการควบคุมแบบพร็อพเพอร์สามารถนิยามได้ดังต่อไปนี้ คือ

4.4.1 กฎการควบคุมแบบพร็อพเพอร์และสตริกท์ลีพร็อพเพอร์

เราสามารถกล่าวได้ว่า กฎการควบคุมที่เป็นฟังก์ชันระหว่าง ลำดับของค่าชักตัวอย่างของสัญญาณออก y_k และ สัญญาณอ้างอิง y_k^* กับ ลำดับของสัญญาณเข้า u_k เป็นแบบพร็อพเพอร์ถ้าสามารถเขียนได้ในรูป

$$u_k = f(k, u_{k-1}, u_{k-2}, \ldots, y_k, y_{k-1}, \ldots, y_k^*, y_{k-1}^*, \ldots)$$

และกฎการควบคุมจะเป็นแบบสตริกท์ลีพร็อพเพอร์ถ้าฟังก์ชัน f ไม่ขึ้นกับสัญญาณออก y_k และสัญญาณอ้างอิง y_k^* ในกรณีของระบบเชิงเส้นที่มีพิกัดแน่นอน จะสามารถนิยามกฎการควบคุมแบบพร็อพเพอร์และสตริกท์ลีพร็อพเพอร์ได้ในลักษณะดังนี้คือ ตัวควบคุมใด ๆ ที่อยู่ในรูป

$$\delta x_c = A_c x_c + B_c y + B_c' y^*$$
$$u = C_c x_c + D_c y + D_c' y^*$$

จะเป็นแบบพร็อพเพอร์ และจะเป็นแบบสตริกท์ลีพร็อพเพอร์ถ้า $D_c = D_c' = 0$

4.4.2 ไลบรารี

ในกรณีกฎของการควบคุมแบบสตริกท์ลีพร็อพเพอร์สามารถโปรแกรมได้ง่ายกว่าดังนี้

```
initialize                          ตั้งค่าพารามิเตอร์ต่าง ๆ ของระบบ
while forever {
    wait(real_time_clock)           รอจนกว่าจะถึงคาบเวลาที่กำหนด
    send_to_D2A(u)                  ส่งสัญญาณควบคุมไปยังการ์ดแปลงสัญญาณ
    read_from_A2D(y, ystar)         อ่านค่า y และ y* ในขณะนั้น
    update_control(y, ystar, u, state)  ปรับปรุงค่าสเตตและคำนวณสัญญาณควบคุม
                                        สำหรับรอบต่อไป
}
```

Astrom และ Wittenmark [9] ได้แนะนำว่าควรทำการอ่านค่าสัญญาณออกก่อนที่จะส่งสัญญาณควบคุม เพื่อหลีกเลี่ยงความเสี่ยงจากการ coupling ในวงจรไฟฟ้า นั่นคือควรเรียงลำดับการทำงานใหม่ดังนี้

```
read_from_A2D(y, ystar)
send_to_D2A(u)
update_control(y, ystar, u, state)
```

การโปรแกรมในกรณีของกฎการควบคุมแบบพร็อพเพอร์จะแตกต่างออกไปคือ

```
initialize                          ตั้งค่าพารามิเตอร์ต่าง ๆ ของระบบ
while forever {
    wait(real_time_clock)           รอจนกว่าจะถึงคาบเวลาที่กำหนด
    read_from_A2D(y, ystar)         อ่านค่า y และ y* ในขณะนั้น
    update_control(y, ystar, u, state)  ปรับปรุงค่าสเตตและคำนวณสัญญาณควบคุม
                                        สำหรับรอบต่อไป
    send_to_D2A(u)                  ส่งสัญญาณควบคุมไปยังการ์ดแปลงสัญญาณ
}
```

ซึ่งการคำนวณสัญญาณควบคุมคือช่วง update_control() ในบางครั้งจะใช้เวลามากจนเกินไปทำให้ผลการควบคุมไม่ดีพอ สำหรับกฎการควบคุมแบบเชิงเส้นที่เป็นแบบพร็อพเพอร์ ควรมีการปรับปรุงโดยการแยกคำนวณสัญญาณควบคุมเป็นดังนี้ คือ

$$u = -K_y y + K_* y^* + H(\delta)[\, y \; y^* \,]^T$$

โดย $H(\delta)$ เป็นตัวดำเนินการสตริกท์ลีพร็อพเพอร์ (strictly proper operator) ขั้นตอนวิธีในการควบคุมสำหรับวิธีใหม่เป็นดังนี้

```
initialize                          ตั้งค่าพารามิเตอร์ต่าง ๆ ของระบบ
while forever {
    wait(real_time_clock)           รอจนกว่าจะถึงคาบเวลาที่กำหนด
    read_from_A2D(y, ystar)         อ่านค่า y และ y* ในขณะนั้น
    u = u – ky * y + kstar * ystar  ปรับปรุงค่าสเตตและคำนวณสัญญาณควบคุม
                                    สำหรับรอบต่อไป
    send_to_D2A(u)                  ส่งสัญญาณควบคุมไปยังการ์ดแปลงสัญญาณ
    pre_compute(y,ystar,u,state)    การคำนวณล่วงหน้า
}
```

ในกรณีนี้เวลาในการคำนวณสัญญาณควบคุมจะมีผลต่อข้อจำกัดทางเวลาน้อยลง เนื่องจากถูกแบ่งเป็นส่วนที่สตริกท์ลีพร็อพเพอร์และส่วนที่พร็อพเพอร์ ขั้นตอนวิธีแบบนี้จะช่วยให้ผลการควบคุมดีขึ้นมากสำหรับกฎการควบคุมที่ซับซ้อน เช่น กฎการควบคุมแบบปรับตัวเองได้ (adaptive control law) เป็นต้น

เมื่อรวมขั้นตอนวิธีทั้ง 3 แบบเข้าด้วยกันสามารถเขียนเป็นไลบรารีสำหรับใช้งานดังนี้

```
init_param;                         /* ตั้งค่าพารามิเตอร์ต่าง ๆ ของระบบ */
while (1) {
    read_in_data()                  /* อ่านค่าสัญญาณออกในขณะนั้น */
    #ifdef STRICTLY                 /* ตัวควบคุมแบบสตริกท์ลีพร็อพเพอร์*/
        write_out_data();           /* ส่งสัญญาณควบคุม */
        compute_control_law();      /* คำนวนค่าสัญญาณควบคุม */
    #else                           /* ตัวควบคุมแบบพร็อพเพอร์ /*
        compute_control_law();
        write_out_data();
    #endif
    #ifdef PRECOMPUTE               /* ตัวควบคุมที่มีการแยกส่วนการคำนวณ */
        pre_compute_control_law();  /* ส่วนการคำนวณล่วงหน้า */
    #endif
        rt_task_wait();             /* รอเวลาในคาบต่อไป */
}
```

ถ้ากฎการควบคุมเป็นแบบสตริกท์ลีพร็อพเพอร์ การส่งสัญญาณควบคุมออกไปจะทำก่อนการคำนวณสัญญาณควบคุมสำหรับรอบต่อไป แต่ถ้ากฎการควบคุมเป็นแบบพร็อพเพอร์จะต้องทำการคำนวณสัญญาณควบคุมก่อนที่จะส่งสัญญาณควบคุมออกไป เนื่องจากค่าสัญญาณออกของระบบที่อ่านได้ขณะนั้นมีผลต่อค่าสัญญาณควบคุมด้วย และถ้ามีการแยกคำนวณก็จะทำหลังจากส่งสัญญาณควบคุมออกไปแล้ว โดยรายละเอียดการใช้งานไลบรารีแสดงอยู่ในหัวข้อ 4.4.3

4.4.3 การใช้งานไลบรารี

ไลบรารีประกอบด้วยไฟล์ต่าง ๆ ดังต่อไปนี้ คือ

- proper.c ต้นฉบับของโปรแกรม
- proper.h ไฟล์กำหนดค่าคงที่ต่าง ๆ ในโปรแกรม
- Makefile ไฟล์สำหรับใช้แปลโปรแกรม

ไลบรารีประกอบด้วยงานแบบเวลาจริง 3 งานด้วยกันคือ

1) งานควบคุม

งานควบคุมมีขั้นตอนวิธีแสดงในตอนท้ายของหัวข้อ 4.4.2 โดยผู้ใช้จะต้องเขียนฟังก์ชันต่าง ๆ เหล่านี้คือ

- init_param ฟังก์ชันในการกำหนดค่าตัวแปรเริ่มต้น หรืองานบางอย่างที่ต้องการทำเป็นอันดับแรก ตัวอย่างของฟังก์ชันนี้คือ

```
void init_param(void){
    a = 0; b = 0; c = -1;
    daout( 1, 0);   /* ตั้งค่าสัญญาณออกของช่องที่ 1 ให้เป็น 0 */
}
```

- compute_control_law ฟังก์ชันในการคำนวณสัญญาณควบคุมขึ้นอยู่กับกฎการควบคุมที่เลือกใช้ ตัวอย่างเช่น

```
void compute_comtrol_law(void){
    b = b + a^2;
}
```

- read_in_data() ฟังก์ชันสำหรับอ่านค่าสัญญาณออกของระบบ ตัวอย่างเช่น

```
void read_in_data(void){
    a = adin(0);    /* อ่านค่าสัญญาณเข้าจากช่อง 0 เก็บไว้ในตัวแปร a */
}
```

- write_out_data() ฟังก์ชันสำหรับอ่านค่าสัญญาณออกของระบบ ตัวอย่างเช่น

```
void write_out_data(void){
    daout(1, b);    /* ส่งสัญญาณออกจากช่องที่ 1 เป็นค่า b */
}
```

- pre_compute_control_law ฟังก์ชันสำหรับการคำนวณล่วงหน้าในกรณีที่มีการแยกคำนวณ ตัวอย่างเช่น

```
void pre_compute_control_law(void){
    b = c * a^2;
}
```

2) งานรับคำสั่ง

งานรับคำสั่งมีขั้นตอนวิธีดังนี้

```
struct Command command;
while(1){
    get_command();
    do_command();
}
```

โดยฟังก์ชัน get_command จะทำหน้าที่อ่านค่าคำสั่งจากไฟล์คำสั่ง (Operation FIFO) มาเก็บไว้ใน struct command ซึ่งผู้ใช้จะต้องกำหนดไว้ในไฟล์ "proper.h" และฟังก์ชัน do_command จะทำหน้าที่นำคำสั่งใน struct command มาปฏิบัติ โดยผู้ใช้ต้องเขียนฟังก์ชันนี้เองด้วย

ตัวอย่างเช่นกำหนด struct command ประกอบด้วยตัวแปร integer 1 ตัว และกำหนดคำสั่งเพียง 2 คำสั่งคือ 1) ตั้งค่าตัวแปร c = 0 และ 2) ตั้งค่าตัวแปร c = -1 สามารถทำได้ดังนี้

```
struct Command {
    int code;
}command;
void do_command(){
    if (command.code == 0) {
        c = 0;
    } else if (command.code == 1) {
        c = 1;
```

```
    } /* else invalid command-code – do nothing*/
}
```

หมายเหตุ ค่าคงที่ต่าง ๆ เช่นขนาดของ FIFO ความถี่ในการรับคำสั่ง สามารถแก้ไขได้ในไฟล์ "proper.h"

3) งานแสดงผล

งานแสดงผลมีขั้นตอนวิธีดังนี้

```
struct State state;
while(1) {
    put_state();
}
```

โดยฟังก์ชัน put_state จะทำหน้าที่เก็บค่า struct state ลงสู่ FIFO ซึ่งผู้ใช้จะต้องกำหนด struct state ในไฟล์ "proper.h" ตัวอย่างเช่น

```
struct State {
    int  a, b, c;
} state;
```

สรุปขั้นตอนการใช้งานไลบรารีสำหรับตัวควบคุมแบบพร็อพเพอร์ มีดังนี้

1.แก้ไขไฟล์ "proper.h"

1.1 แก้ไขพารามิเตอร์เกี่ยวกับ RT-Linux เช่นค่าความสำคัญของงาน ค่าความถี่ และขนาดของ FIFO ที่ใช้เป็นต้น

1.2 ถ้าตัวควบคุมเป็นแบบสตริกท์ลีพร็อพเพอร์ให้ "#define STRICTLY" หรือถ้าตัวควบคุมเป็นแบบแยกส่วนการคำนวณให้ "#define PRECOMPUTE".

1.3 กำหนด struct State และ struct Command ตามต้องการ

2. เขียนโปรแกรมซึ่งประกอบด้วยสิ่งต่าง ๆ เหล่านี้

2.1 กำหนดตัวแปรที่ใช้งานร่วมกัน สำหรับทุก ๆ ฟังก์ชัน

2.2 ฟังก์ชันสำหรับงานควบคุม

```
init_param();
compute_control_law();
read_in_data();
write_out_data();
```

และ pre_compute_control_law(); ในกรณีที่แยกคำนวณ

2.3 ฟังก์ชันสำหรับงานรับคำสั่ง

do_command();

3. ใส่ #include "proper.h" และ #include "../adda/pclab.h" ในโปรแกรมที่เขียนขึ้น

4. แก้ไขไฟล์ "Makefile" ในบรรทัดที่เขียนว่า

all: proper.o yourfile.o แทนที่ "yourfile" ด้วยชื่อไฟล์ที่เขียนขึ้น

5. ทำคำสั่ง "make"

บทนี้ได้แสดงให้เห็นถึงการออกแบบและการพัฒนาระบบควบคุมโดยตรงเพื่อเป็นกรณีศึกษา และในตอนท้ายนำสิ่งที่ได้มาประยุกต์ใช้ในการออกแบบไลบรารีสำหรับกฎการควบคุมแบบพร็อพเพอร์ บทต่อไปจะกล่าวถึงการนำกรณีศึกษามาประยุกต์ใช้เพื่อออกแบบโปรแกรมที่มีฟังก์ชันการทำงานมากขึ้น และเป็นโปรแกรมสำหรับชุดทดลองในห้องปฏิบัติการวิจัยระบบควบคุม โดยจะตั้งชื่อว่าโปรแกรม RTheat