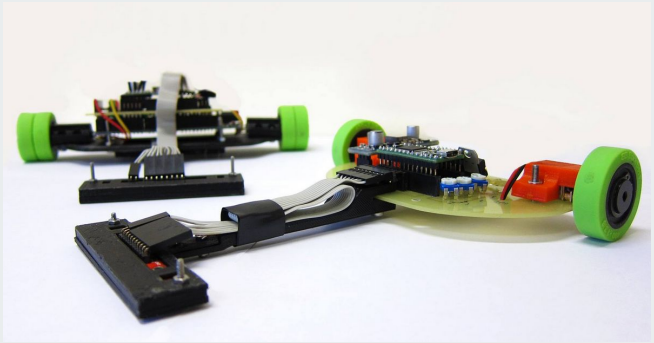
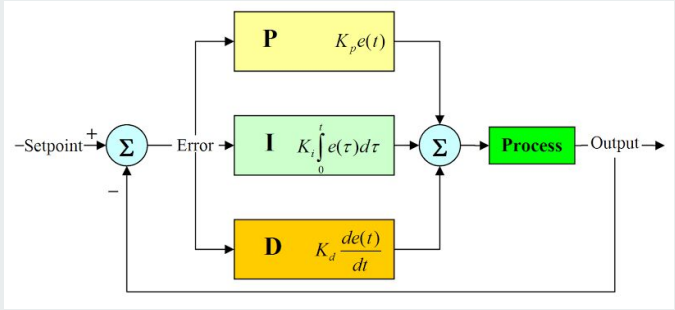
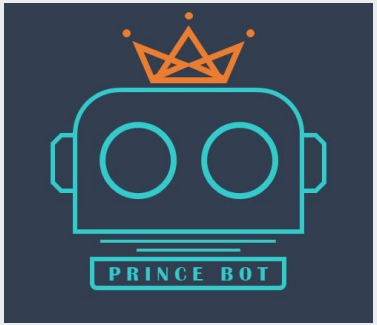
การเขียนโปรแกรมหุ่นยนต์วิ่งตามเส้นแบบ PID (เบื้องต้น)







ก่อนที่จะเข้า PID มารู้จักกับระบบควบคุมก่อนนะ

การควบคุมระบบพลศาสตร์ ให้มีค่าเอาต์พุตที่ต้องการ โดยการป้อนค่าอินพุตที่เหมาะสมให้กับระบบ

ระบบควบคุมยังแบ่งออกได้เป็น

1. ระบบควบคุมวงเปิด (open-loop control)
2. ระบบควบคุมวงปิด (closed-loop control)

ระบบควบคุมวงเปิด และ ระบบควบคุมวงปิด คือ

ระบบควบคุมวงเปิด (open-loop control) คือ ระบบควบคุมที่ไม่ได้ใช้สัญญาณจากเอาต์พุต มาบ่งชี้ถึงลักษณะการควบคุม

ระบบควบคุมวงปิด (closed-loop control) หรือ ระบบป้อนกลับ (feedback control) นั้นจะใช้ค่าที่วัดจากเอาต์พุตมาคำนวณค่าการควบคุม

ความแตกต่างระหว่าง ระบบควบคุมแบบเปิด และ ปิด

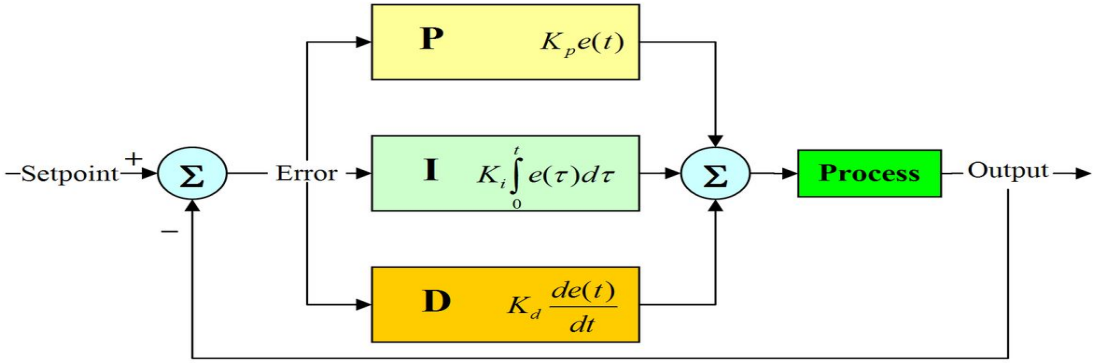
Open loop control system	Closed loop control system
The feedback element is absent.	The feedback element is always present.
An error detector is not present.	An error detector is always present.
It is stable one.	It may become unstable.
Easy to construct.	Complicated construction.
It is an economical.	It is costly.
Having small bandwidth.	Having large bandwidth.
It is inaccurate.	It is accurate.
Less maintenance.	More maintenance.
It is unreliable.	It is reliable.
Examples: Hand drier, tea maker	Examples: Servo voltage stabilizer, perspiration

ระบบควบคุมแบบเปิด	ระบบควบคุมแบบปิด
ไม่มีการการป้อนกลับหรือตรวจสอบของชุดข้อมูล	มีการป้อนกลับหรือตรวจสอบข้อมูลแบบตลอดเวลา
โครงสร้างเข้าใจง่าย	โครงสร้างมีความซับซ้อน
การดูแลรักษาง่าย และถูก	ดูแลรักษายากและแพง

PID คืออะไร

เป็นระบบควบคุมแบบป้อนกลับ ซึ่งค่าที่นำไปใช้ในการคำนวณเป็นค่าความผิดพลาดที่หามาจากความแตกต่างของตัวแปรในกระบวนการและค่าที่ต้องการ ตัวควบคุมจะพยายามลดค่าผิดพลาดให้เหลือน้อยที่สุดด้วยการปรับค่าสัญญาณขาเข้าของกระบวนการ

PID คืออะไร



อธิบายตามรูปคือ นำ Setpoint มาเปรียบเทียบกับ Output
เมื่อมี Error ก็นำ Error ไปเข้ากับ สมการ P สมการ I สมการ P
เมื่อคำนวณเสร็จแล้ว นำมารวมกัน เพื่อนำไป ควบคุมการทำงาน
ต่าง ๆ และส่ง Output ไปคำนวณอีกรอบ
ทำแบบนี้ไปเรื่อย ๆ

การประยุกต์ PID กับหุ่นยนต์วิ่งตามเส้น

สิ่งที่จะต้องรู้ก่อนคือ

1. หุ่นยนต์ใช้งานกี่ Sensor
2. โอกาสที่จะเกิดขึ้นกับหุ่นยนต์
3. รวมสมการหุ่นยนต์
4. ปรับจูนยังไง

ทำไมต้องรู้ว่า หุ่นยนต์ใช้งานกี่ Sensor

คือว่า ในการทำระบบแบบ PID นั้นเราจะต้องรู้ค่า Input ที่เราจะใส่เข้าไปก่อน ซึ่ง Input นั้นจะต้องมี 1 Input แต่ถ้าเรามีหลาย ๆ Sensor ก็ต้องยุบรวมกันให้เป็น Input เดียว แล้วจะยุบยังไง ก็คือว่านำค่าต่าง ๆ ของ Sensor แต่ละตัว มาคำนวณหาตำแหน่งของเส้น หรือ Position

การหา Position

การหา Position เราจะใช้สมการหาค่าเฉลี่ยแบบถ้วงน้ำหนัก

การหาค่าเฉลี่ยเลขคณิตถ่วงน้ำหนัก นี้ใช้ในกรณีที่ข้อมูลแต่ละค่ามีความสำคัญไม่เท่ากัน เช่น

วิธีทำ $$\mu = \frac{\sum_{i=1}^{N} W_i X_i}{\sum_{i=1}^{N} W_i} = \frac{0.5(4) + 1(3) + 1.5(2) + 2(2.5)}{0.5 + 1 + 1.5 + 2} = 2.6$$

ดังนั้น เกรดเฉลี่ยของนางสาวดาเป็น 2.6 Ans

การหา Position

ทีนี้ ตัวอย่างจะใช้ Sensor 4 ตัว ใช้จับตำแหน่งของเส้น โดยจะกำหนดว่า

Sensor 1 จับเส้นได้ให้มีค่า 0

Sensor 2 จับเส้นได้ให้มีค่า 100-200

Sensor 3 จับเส้นได้ให้มีค่า 201-300

Sensor 4 จับเส้นได้ให้มีค่า 300

ตัวอย่างนี้จะสาธิตให้จำเส้นดำนะครับ

ทำไมค่าที่อ่านได้มันแปลกๆ อย่างพึ่ง งง ไปดูก่อน

การหา Position

Sensor 1 จับเส้นได้ให้มีค่า 0 Sensor 2 จับเส้นได้ให้มีค่า 100-200
Sensor 3 จับเส้นได้ให้มีค่า 201-300 Sensor 4 จับเส้นได้ให้มีค่า 300

มาลองเขียนสมการกัน

A = (Sensor(1) * 0 + Sensor(2) * 100 + Sensor(3) * 200 + Sensor(4)*300)

B = (Sensor(1) + Sensor(2) + Sensor(3) + Sensor(4))

Position = A / B

เราจะได้ตำแหน่งของเส้นดำ ที่หุ่นยนต์เราอ่านได้ แต่!!!!

การหา Position

แต่ทีนี้จะเจอปัญหาที่ว่า Sensor แต่ละตัว นั้นหาค่าได้ไม่เท่ากัน
พอนำมาคำนวณแล้วมันอาจจะมีการผิดพลาดได้

ดังนั้นวิธีการแก้ไขก็คือว่า
เราจะต้อง หา % ของ Sensor แต่ละตัว เช่น
ค่าที่อ่านได้ของ
Sensor(1) อ่านได้ น้อย สุดคือ 200
Sensor(1) อ่านได้ มากสุด สุดคือ 900

ถ้าอ่านได้ 200 % ของตัวนั้นจะเป็น 100
แต่ถ้า อ่านได้ 900 % ของตัวนั้นจะเป็น 0

ทำไมจะต้องให้
ค่าน้อยเป็น 100 %
แต่ค่ามากเป็น 0 %
ก็เพราะว่า
เราให้ความสำคัญกับ
เส้นสีดำนั้นเอง

การหา Position

ปรับเปลี่ยนค่า Sensor ให้เป็น % ได้ดังนี้

((x - Sensor_min) * (100) / (Sensor_max - Sensor_min))-100

โดยที่ X คือค่าที่อ่านได้ เช่น 500
Sensor_min = 200
Sensor_max = 900

ค่าที่ได้คือ ((500 - 200) *100 / (900-200))-100 = 57.1 %

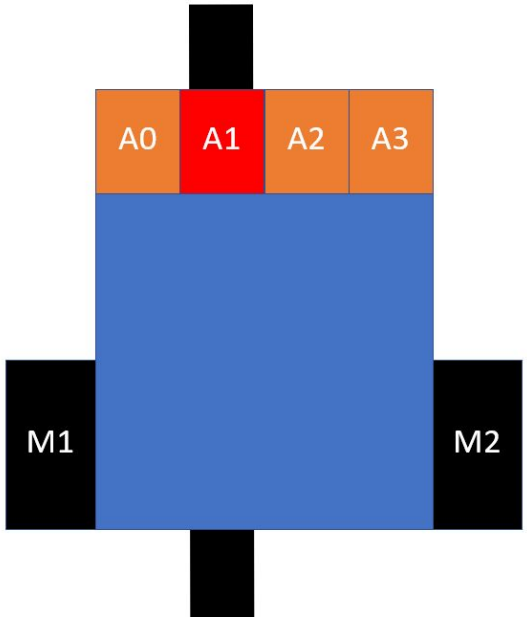
การหา Position

แต่รู้หรือไม่ ใน Arduino IDE นั้นมีฟังก์ชั้นหา % มาให้แล้วนะ ตัวอย่าง

```
int set_min[] = {200,100,200,300};   // ค่าที่อ่านได้ สีดำ  ของแต่ละ Sensor
int set_max[] = {900,900,900,900};   // ค่าที่อ่านได้ สีขาว ของแต่ละ Sensor
int sensor(int pin){
  return map(analog(pin),set_min[pin],set_max[pin],100,0);
}
```

อธิบาย โค้ด เขียนฟังก์ชั้น sensor(pin) เพื่อให้คำนวณ % ของ Sensor แต่ละตัว โดยจะต้องมี ตัวแปร set_min และ set_max เพื่อทำการคำนวณ % ออกมา
หลังจากนั้นก็สามารถนำไปแทนในสมการ หาค่าเฉลี่ยถ่วงน้ำหนักได้แล้ว

การหา Position (ตัวอย่าง)



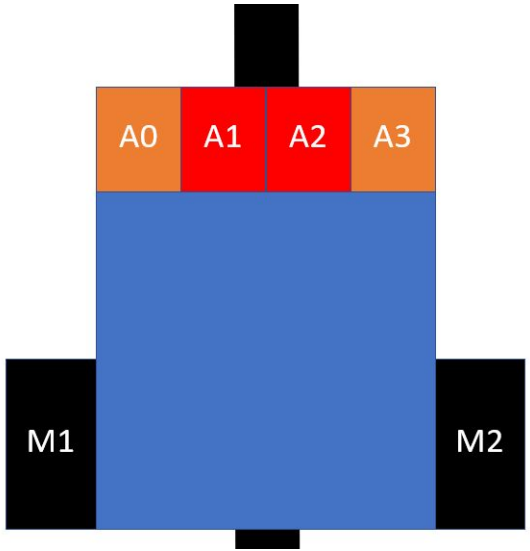
หุ่นยนต์อยู่บนเส้น แล้ว Sensor ตัวที่ A1 อยู่บนเส้นดำ
ลองมาคำนวณคร่าว ๆ ดู
ให้
Sensor(A0) อ่านได้ 0 % Sensor(A1) อ่านได้ 89 %
Sensor(A2) อ่านได้ 0 % Sensor(A3) อ่านได้ 0 %

ให้
A = (Sensor(A0) * 0 + Sensor(A1) * 100 + Sensor(A2) * 200 +Sensor(A3) * 300);
B = (Sensor(A0) + Sensor(A1) +Sensor(A2) +Sensor(A3));

จะได้ A = (0*0 + 89 * 100 + 0*200 + 0*300); = 8900
B = (0 + 89 +0 + 0); = 89

position A / B = 8900 / 89 = 100

การหา Position (ตัวอย่าง)



หุ่นยนต์อยู่บนเส้น แล้ว Sensor ตัวที่ A1 และ A2 อยู่บนเส้นดำ
ลองมาคำนวณคร่าว ๆ ดู
ให้
Sensor(A0) อ่านได้ 0 % Sensor(A1) อ่านได้ 90%
Sensor(A2) อ่านได้ 95 % Sensor(A3) อ่านได้ 0 %

ให้
A = (Sensor(A0) * 0 + Sensor(A1) * 100 + Sensor(A2) * 200 +Sensor(A3) * 300);
B = (Sensor(A0) + Sensor(A1) +Sensor(A2) +Sensor(A3));

จะได้ A = (0*0 + 90* 100 + 95*200 + 0*300); = 28000
B = (0 + 90+95+ 0); = 185

position A / B = 2800/ 185 = 151.3

โอกาสที่จะเกิดขึ้นกับหุ่นยนต์

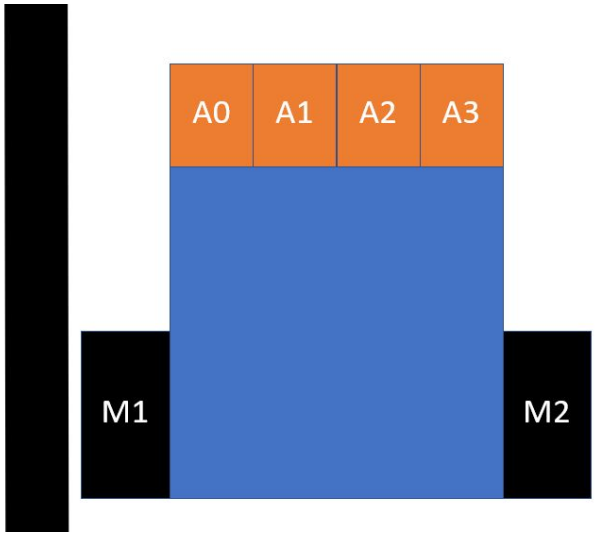
มาดูโอกาสที่จะเกิดขึ้นในระหว่างทำหุ่นยนต์วิ่งตามเส้น

1. หุ่นยนต์วิ่งอยุ่บนเส้นสีดำ
2. หุ่นยนต์หลุดออกจากเส้นทางด้าน ซ้าย
3. หุ่นยนต์หลุดออกจากเส้นทางด้าน ขวา

ถ้าหุ่นยนต์อยู่บนเส้น หรืออยู่ในกรณีที่ 1 เราก็เขียนการหา Position ตามปกติ
แต่ถ้ามันหลุดออกจากเส้นหละ ???

โอกาสที่จะเกิดขึ้นกับหุ่นยนต์

ลองคำนวณหุ่นยนต์เมื่อออกจากเส้นดูกันครับ



Sensor(A0) อ่านได้ 1 % Sensor(A1) อ่านได้ 1 %
Sensor(A2) อ่านได้ 1 % Sensor(A3) อ่านได้ 1 %

ให้
A = (Sensor(A0) * 0 + Sensor(A1) * 100 + Sensor(A2) * 200 +Sensor(A3) * 300);
B = (Sensor(A0) + Sensor(A1) +Sensor(A2) +Sensor(A3));

จะได้ A = (1*0 + 1* 100 + 1*200 + 1*300); = 600
B = (1 + 1+1+ 1); = 4

position A / B = 600/ 4 = 150

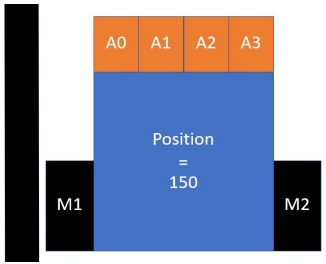
อ่าว ถ้าหุ่นยนต์เจอแบบนี้ มันก็แย่นะสิ

โอกาสที่จะเกิดขึ้นกับหุ่นยนต์

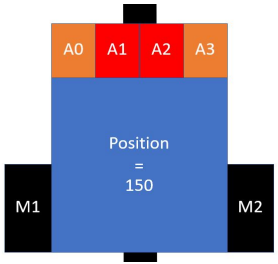
จากกรณีด้านบนนั้นจะเห็นได้ว่า เมื่อหุ่นยนต์ออกจากเส้นแล้วพอนำมาเข้า สมการการหาค่าเฉลี่ยแบบถ่วงน้ำหนัก มันจะทำงานไม่ได้

เพราะอะไร

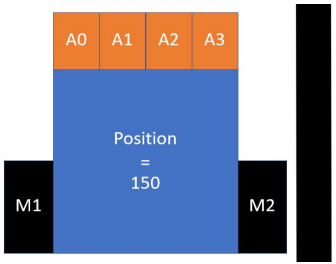
ก็เพราะว่า ถ้าตำแหน่งหรือ Position เมื่อคำนวณมันจะอยู่ที่ 150 เมื่อนำไปเข้าสมการ PID มันก็จะวิ่งตรงเหมือน กับ ว่า หุ่นยนต์พบเส้นดำอยู่กลางหุ่นยนต์เหมือนกัน



มีค่าเท่ากัน



มีค่าเท่ากัน



โอกาสที่จะเกิดขึ้นกับหุ่นยนต์

ดังนั้นก่อนที่เราจะเข้าสมการ หาค่า Position เราจะต้องเช็คก่อน
ว่าหุ่นยนต์ของเราอยู่บนเส้นหรือไม่
โดยกำหนดตัวแปร ขึ้นมา 1 ตัว ชื่อว่า On_line ให้มีค่า = 0

ถ้า Sensor ตัวใดตัวหนึ่งเจอเส้นดำให้ ตัวแปร On_line มีค่า = 1

ถ้าตัวแปร On_line มีค่า = 1 ให้คำนวณสมการปกติ
แต่ถ้าตัวแปร On_line มีค่า = 0 ถ้าไม่อยู่ในเส้นจะทำยังไง

โอกาสที่จะเกิดขึ้นกับหุ่นยนต์

มาเขียนโปรแกรมเช็คกันว่าหุ่นยนต์ของเราอยู่บนเส้นหรือไม่
โดยการ เช็คว่า Sensor (ทุกตัว) ที่เราอ่านได้มีค่ามากกว่า 50 %
หรือไม่
ถ้ามากกว่า 50 % แสดงว่าหุ่นยนต์อยู่บนเส้น ให้ On_line = 1;
แต่ถ้า ไม่อยู่บนเส้นให้ On_line = 0;

```
long readline(){
  int On_line = 0;
  if(sensor(0) > 50 || sensor(1) > 50 || sensor(2) > 50 || sensor(3) > 50){
    On_line = 1;
  }
  else{
    On_line = 0;
  }
}
```

โอกาสที่จะเกิดขึ้นกับหุ่นยนต์

ถ้า ตัวแปร On_line == 0 หรือ หุ่นยนต์ไม่อยู่ในเส้น

เราก็จะต้องให้หุ่นยนต์มันจำสิว่า มันหลุดไปจากตำแหน่งไหน ทางซ้าย หรือ ขวา

เราจะต้องมีตัวแปร อีก 1 ตัว ที่ไว้เก็บค่าก่อนหน้านั้นที่หุ่นยนต์จะหลุดออกจากเส้น จะกำหนดให้ชื่อว่า last_position

```
int set_min[] = {200,100,200,300};  // ค่าที่อ่านได้ สีดำ  ของแต่ละ Sensor
int set_max[] = {900,900,900,900};  // ค่าที่อ่านได้ สีขาว ของแต่ละ Sensor
int last_position = 0;  // ไว้เก็บตำแหน่งล่าสุด
```

โอกาสที่จะเกิดขึ้นกับหุ่นยนต์

เมื่อเรามีค่าก่อนหน้านั้นแล้วคือ last_position
เราก็นำมาเช็คว่า
ค่า ล่าสุดก่อนที่หุ่นยนต์จะหลุดออกคือตำแหน่งที่เท่าไร
เช่น
ถ้าตำแหน่งหลุดออกทางด้านซ้าย คือค่าที่น้อยกว่า 150
แต่ถ้าแหน่งหลุดออกทางด้านขวา คือค่าที่มากกว่า 150
เราก็นำ คำที่ว่า มากกว่า น้อยกว่า นั้น มาเช็ค
แล้วก็กำหนด Position ล่าสุดเข้าไป เอ้างงละสิ ไปดู ๆ !!!

ตัวเลข 150 มาจากไหน ตัวเลข 150 ก็มาจากว่า เมื่อหุ่นยนต์อยู่ตรงกลางค่า Position จะเท่ากับ 150 นั้นเอง

โอกาสที่จะเกิดขึ้นกับหุ่นยนต์

ดูเนื้อหามันจะมึน ๆ ลองมาดูโปรแกรมกันบ้างดีกว่า โดยเราจะสร้างฟังก์ชั้นในการอ่านค่าชื่อว่า readline()

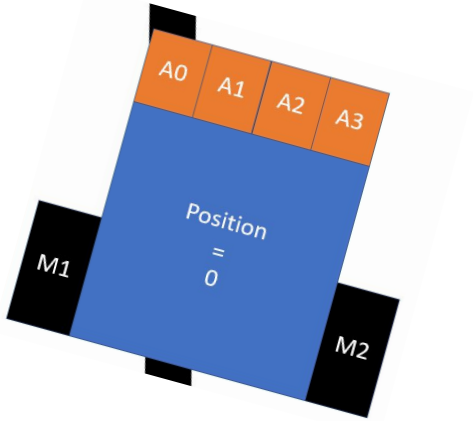
```
long readline(){
  int On_line = 0;
  if(sensor(0) > 50 || sensor(1) > 50 || sensor(2) > 50 || sensor(3) > 50){On_line = 1;}
  else{On_line = 0;}

  if(On_line == 1){/*เขียนสมการคำนวนค่า Position*/}
  else{
    if(last_position > 150){  //ค่าล่าสุดก่อนที่จะหลุดเส้นคือทางด้านขวา
      return 300;               //ให้มันมีค่า มากที่สุดของทางด้านขวา
    }
    else{return 0;}
  }
  return last_position = Position;
}
```

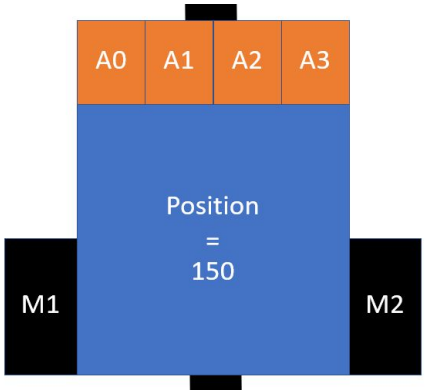
รวมสมการ PID กัน

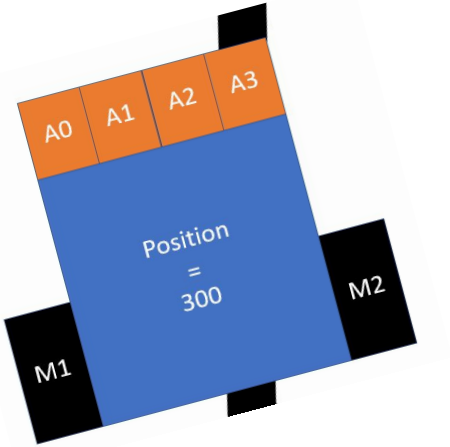
ก่อนจะเข้าสมการ PID มาลองใช้กันก่อนจะดีมั๊ย จะได้เข้าใจการทำงานของมันจริง ๆ

ค่า Position ของเรานั้นสามารถอ่านได้ ตั้งแต่ 0-300 ใช่ม่ะ
ดังนั้นลองเขียน เล่น ๆ ดู

รวมสมการ PID กัน







ถ้า Position > 200 ให้เลี้ยวขวา
ถ้า Position < 100 ให้เลี้ยวซ้าย
แต่ถ้า Position > 100 and Position < 200 ให้เดินตรง

รวมสมการ PID กัน

การควบคุมแบบ PID ได้ชื่อตามการรวมกันของเทอมของตัวแปรทั้งสามตามสมการ:

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

เมื่อ

P_{out}, I_{out}, และ D_{out} เป็นผลของสัญญาณขาออกจากระบบควบคุม PID จากแต่ละเทอมซึ่งนิยามตามรายละเอียดด้านล่าง

ปกติการใช้ระบบควมคุมก็จะต้องมี
เทอม ทั้ง 3 อัน คือ
เทอม P (สัดส่วน หรือ เรียกการ อัตราขยาย)
เทอม I (ปริพันธ์ หรือ รีเซ็ต) ส่วนของวิ่งตามเส้นไม่จำเป็นต้องใช้ก็ได้
เทอม D (อนุพันธ์)

รวมสมการ PID กัน

เทอม P ในส่วนของหุ่นยนต์วิ่งตามเส้น

เทอม P คือ เทอมที่มีหน้าที่ขยายอัตราส่วนของ Error โดยจะนำค่า Error คูนด้วย ค่าคงที่คือ Kp

เช่น Position ของหุ่นยนต์เมื่ออยู่ตรงกลางจะเท่ากับ 150
แต่ถ้าหุ่นยนต์วิ่งเอียงทำให้ Position อยู่ที่ 100
จะทำให้มีค่า Error = 150 - 100 = 50 แล้วนำมาคูณกับ Kp
เช่น ค่า Kp = 2 แล้วค่า Error = 50 ดังนั้น เทอมนี้จะมีค่าเท่ากับ 100

รวมสมการ PID กัน

เทอม I ในส่วนของหุ่นยนต์วิ่งตามเส้น (ไม่ต้องใช้ก็ได้)

เทอม I คือ เทอมที่มีหน้าที่ลดขนาดของความผิดพลาด ในทุก ๆช่วงเวลา

เช่น Position ของหุ่นยนต์วิ่งไปยังไงก็ไม่เข้าที่ตำแหน่งตรงกลาง (position 150) อาจจะได้ 152 ตลอดตามเส้นทาง เทอม I จะช่วยให้เข้าใกล้ 150 มากขึ้น แต่ถ้าปรับ เทอม I ผิดพลาด มันก็จะทำให้หุ่นยนต์ของเราวิ่งมั่วไปเลยหรือเขาเรียกว่า ระบบมันระเบิดไปเลย

รวมสมการ PID กัน

เทอม D ในส่วนของหุ่นยนต์วิ่งตามเส้น

เทอม D คือเทอมที่มีหน้าที่ชะลออัตราการเปลี่ยนแปลงของ Error หรือลดการสะบัด (ส่าย) ไปมาของหุ่นยนต์

เช่น หุ่นยนต์วิ่ง สายไปมา เนื่องจากปรับค่า Kp ที่สูง เมื่อปรับ Kd ขึ้นมา มันก็จะช่วยให้หุ่นยนต์ลดการสะบัด (ส่าย) ทำให้หุ่นยนต์วิ่งเข้าตำแหน่งตรงการได้เนียนขึ้น

รวมสมการ PID กัน

ทีนี้มาดู ตัวอย่าง สมการคร่าว ๆ ของ PID กันครับ

```
previous_error = setpoint - actual_position
integral = 0
start:
  error = setpoint - actual_position
  integral = integral + (error*dt)
  derivative = (error - previous_error)/dt
  output = (Kp*error) + (Ki*integral) + (Kd*derivative)
  previous_error = error
  wait(dt)
  goto start
```

จากตัวอย่าง จะมีคำต่าง ๆ เพิ่มมามากมายเลย มันก็มาจาก ทฤษฎี ระบบควบคุมแบบ PID นั้นแหละ

รวมสมการ PID กัน

ในส่วนโปรแกรมนี้จะให้เป็นตัวอย่างไปนะครับ

```
void loop() {
  int Kp=2,Ki=0,kp=10;
  int speed_max = 70;
  present_position = readline()/3;  // 300 / 3    = 100
  setpoint = 50;   // 50 คือค่าตรงกลางหลังจากหาร 3 มา  150 /3 = 50
  errors = setpoint - present_position;
  integral = integral + errors ;
  derivative = (errors - previous_error) ;
  output = Kp * errors + Ki * integral + Kd * derivative;
  previous_error = errors;
  motor(1, speed_max + output);  //ถ้า output เป็น บวก แสดงว่าเส้นอยู่ทางขวาของหุ่นยนต์
                                  //ให้ล้อ ซ้ายวิ่งเร็วขึ้น
  motor(2, speed_max - output);  //ถ้า output เป็น ลบ แสดงว่าเส้นอยู่ทางซ้ายของหุ่นยนต์
                                  //ให้ล้อ ขวา วิ่งเร็วขึ้น
  delay(1);
}
```

รวมสมการ PID กัน

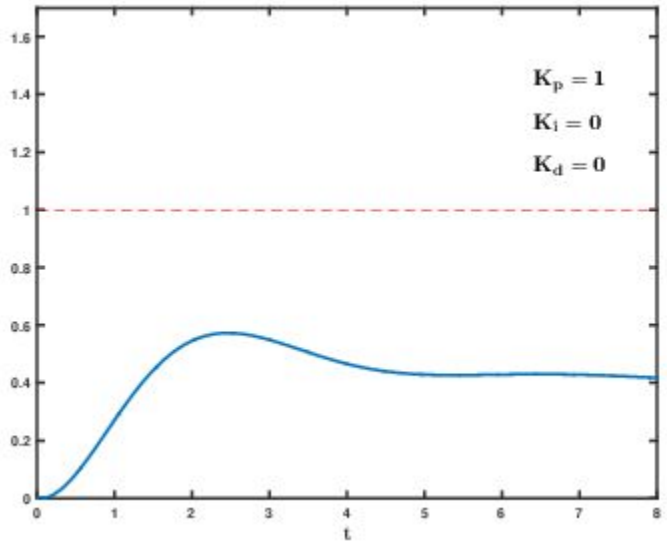
วิธีการปรับจูน เบื้องต้น

1. เริ่มจากให้ Kp Ki Kd =0 ให้หมด
2. ปรับค่า Kp เพิ่มขึ้น จะทำให้หุ่นยนต์กลับไปจุดกลางเร็วที่สุด
3. ปรับค่า Kd เพิ่มขึ่น จะทำให้หุ่นยนต์ลดอาการแกว่งลง
4. เมื่อ Kd เพิ่มขึ้น ให้ลด Kp ลง
5. ปรับค่า Ki เพิ่มขึ้นที่ละน้อยยยยยยยยยย

หรือดูตารางการปรับด้วยมือดังนี้

รวมสมการ PID กัน

วิธีการปรับค่า PID เบื้องต้น



รวมสมการ PID กัน

วิธีการปรับจูน เบื้องต้น

ผลของการเพิ่มค่าตัวแปรอย่างอิสระ

ตัวแปร	ช่วงเวลาขึ้น (Rise time)	โอเวอร์ชูต (Overshoot)	เวลาสู่สมดุล (Settling time)	ความผิดพลาดสถานะคงตัว (Steady-state error)	เสถียรภาพ[1]
K_p	ลด	เพิ่ม	เปลี่ยนแปลงเล็กน้อย	ลด	ลด
K_i	ลด[2]	เพิ่ม	เพิ่ม	ลดลงอย่างมีนัยสำคัญ	ลด
K_d	ลดลงเล็กน้อย	ลดลงเล็กน้อย	ลดลงเล็กน้อย	ตามทฤษฎีไม่มีผล	ดีขึ้นถ้า K_d มีค่าน้อย

เข้าไปดูตัวอย่างโค้ดได้ที่

http://www.princebot.net/article/12/

กดติดตามได้ที่

YouTube

PRINCE BOT