



CENG 4480

Embedded System Development & Applications

Lec 09: PID Control

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

(Latest update: October 14, 2024)

2024 Fall

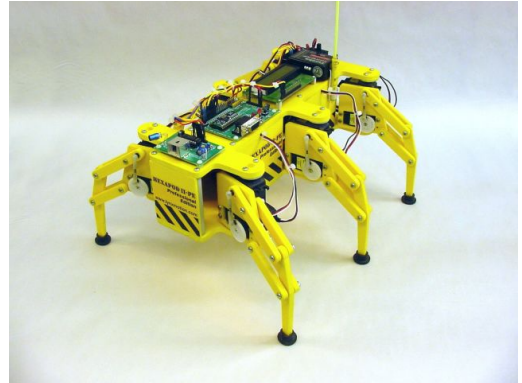


- ① Motors
- ② Open-loop and Closed-loop Control
- ③ Control Methods
- ④ Software



- 1 Motors
- 2 Open-loop and Closed-loop Control
- 3 Control Methods
- 4 Software

DC Motors: Direct current motor, easy to control and use. For making wheeled robots



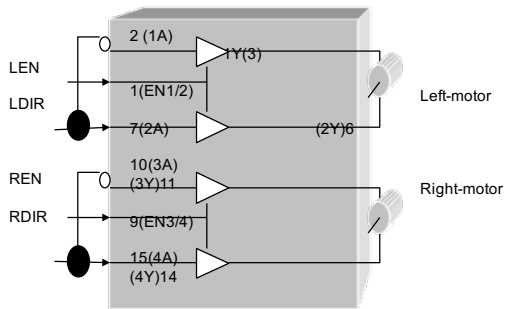
Servo motors for making robot legs
<http://www.lynxmotion.com/>



- Speed (≈ 1200 – 2000 rpm).
- Operates on a $3\sim 5$ Volt, Can use gear box (e.g. ratio 58:1) to increase torque
- Use H-bridge circuit to boost up current from the TTL level to motor driving level.



[Taobao link](#)



H-bridge Chips

- **L293D**: H-bridge circuit, up 2A
- **LDIR**: left motor direction
- **RDIR**: right motor direction
- **LEN**: left motor enable
- **REN**: right motor enable



- 1 Motors
- 2 Open-loop and Closed-loop Control
- 3 Control Methods
- 4 Software



Change motor supply power change speed

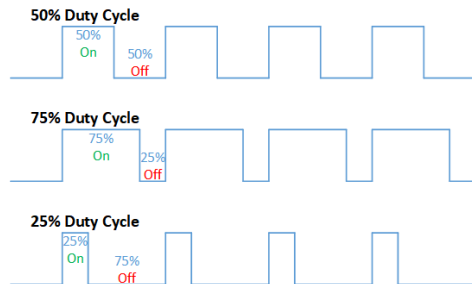
Problem: How much power is right?

Ans: don't know , depends on internal/external frictions of individual motors.

Problem: How to control power (Ton) by MCU?

- Solution: Use feedback control to read actual wheel:
- Slower, increase power (+ Ton)
- Faster, reduce power (- Ton)

- Pulse Width Modulation
- Analog results with digital means
- a square signal switched between on and off
- changing the portion the signal on





Exercise

When using the open-loop control method with a constant PWM signal for both wheels, explain why the robot would slow down when climbing up hill.



- Supports single edge controlled and/or double edge controlled PWM outputs.
- Seven match registers allow up to 6 single edge controlled or 3 double edge controlled PWM outputs, or a mix of both types.

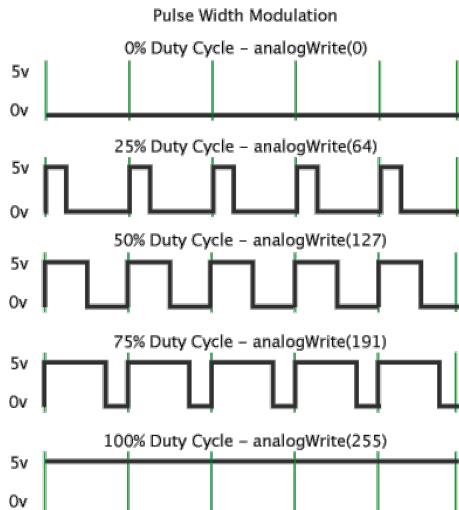
Table 181. Set and reset inputs for PWM Flip-Flops

PWM Channel	Single Edge PWM (PWMSELn = 0)		Double Edge PWM (PWMSELn = 1)	
	Set by	Reset by	Set by	Reset by
1	Match 0	Match 1	Match 0 ^[1]	Match 1 ^[1]
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2 ^[2]	Match 3 ^[2]
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4 ^[2]	Match 5 ^[2]
6	Match 0	Match 6	Match 5	Match 6

[1] Identical to single edge mode in this case since Match 0 is the neighboring match register. Essentially, PWM1 cannot be a double edged output.

[2] It is generally not advantageous to use PWM channels 3 and 5 for double edge PWM outputs because it would reduce the number of double edge PWM outputs that are possible. Using PWM 2, PWM4, and PWM6 for double edge PWM outputs provides the most pairings.

- Call `analogWrite()`
- On a scale of 0 – 255
- `analogWrite(255)` requests a 100% duty cycle (always on)
- `analogWrite(127)` is a 50% duty cycle (on half the time)





- The real solution to real speed control is feedback control
- Require speed encoder to read back the real speed of the wheel at real time.

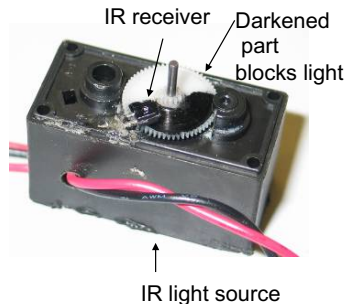
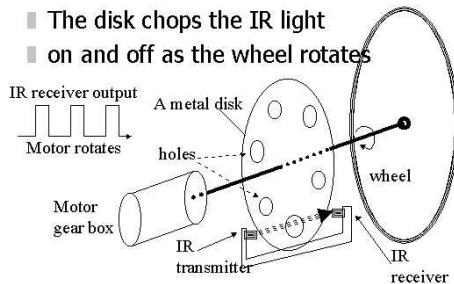
First you need to have speed encoders



- Read wheel speed.
- Use photo interrupter
- Use reflective disk to save space
- Based on interrupts

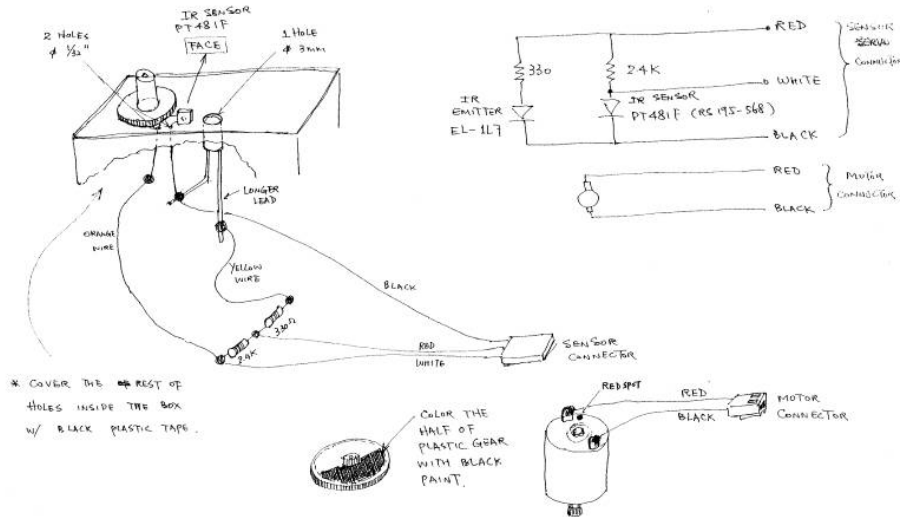


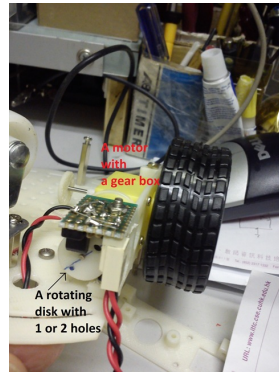
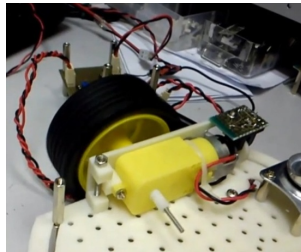
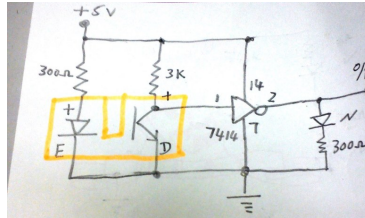
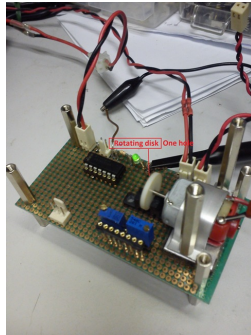
- Our motor and speed encoder
- Each wheel rotation = 88 on/off changes





SERVO MOTOR MODIFICATION





Servo library in Arduino



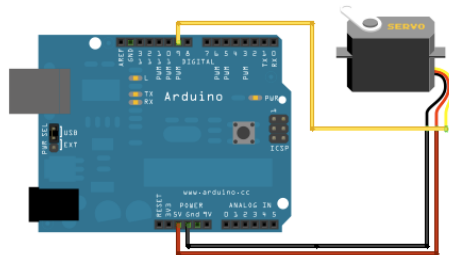
```
#include <Servo.h>

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);                // tell servo to go to position in variable 'pos'
    delay(15);                          // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);                // tell servo to go to position in variable 'pos'
    delay(15);                          // waits 15ms for the servo to reach the position
  }
}
```

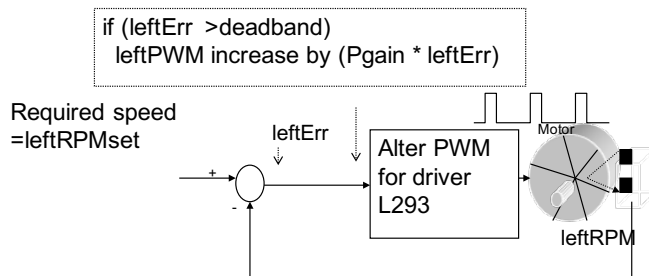


https://youtu.be/VvHg6_q13Fg



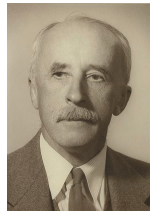
- ① Motors
- ② Open-loop and Closed-loop Control
- ③ Control Methods
- ④ Software

Closed-loop feed back control



Note: Show the left motor control only

- PID: Proportional-Integral-Derivative
- A more formal and precise method used in most modern machines



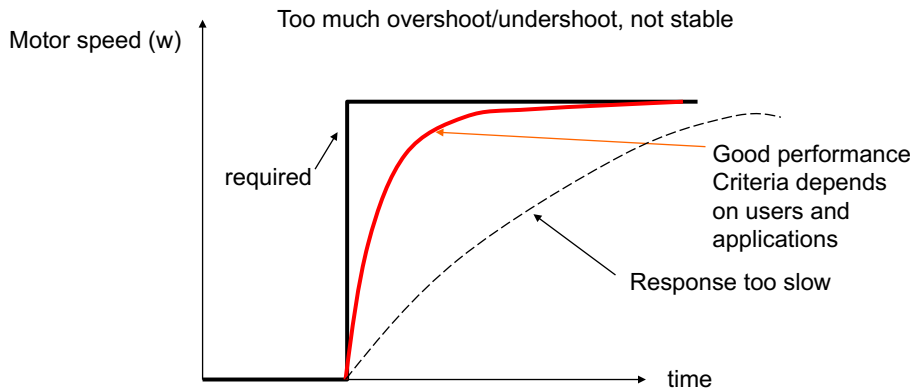
History of PID

- By Nicolas Minorsky in 1922
- Observations of a helmsman
- Steered the ship based on
 - the current course error
 - past error
 - the current rate of change

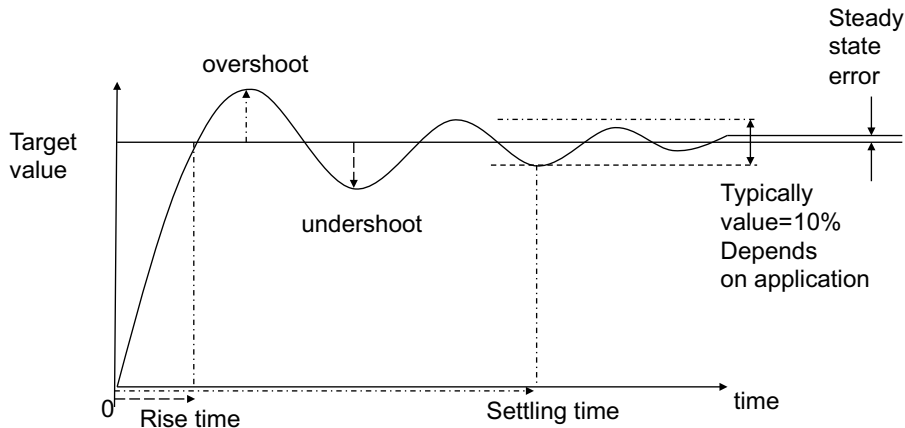




- Control for better performance
- Use PID, choose whatever response you want



Describe the terms in the following diagrams:

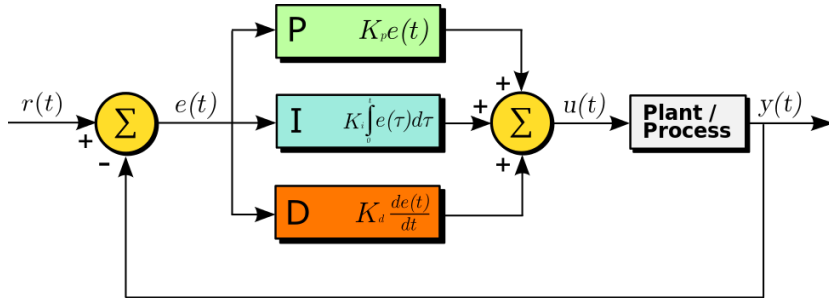




$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt},$$

where

- $e(t)$: error value
- $u(t)$: control variable
- K_p : coefficient for the proportional (**P**)
- K_i : coefficient for the integral (**I**)
- K_d : coefficient for the derivative (**D**)





Proportional Gain K_p

Larger K_p typically means faster response since the larger the error, the larger the Proportional term compensation. An excessively large proportional gain will lead to process instability and oscillation.

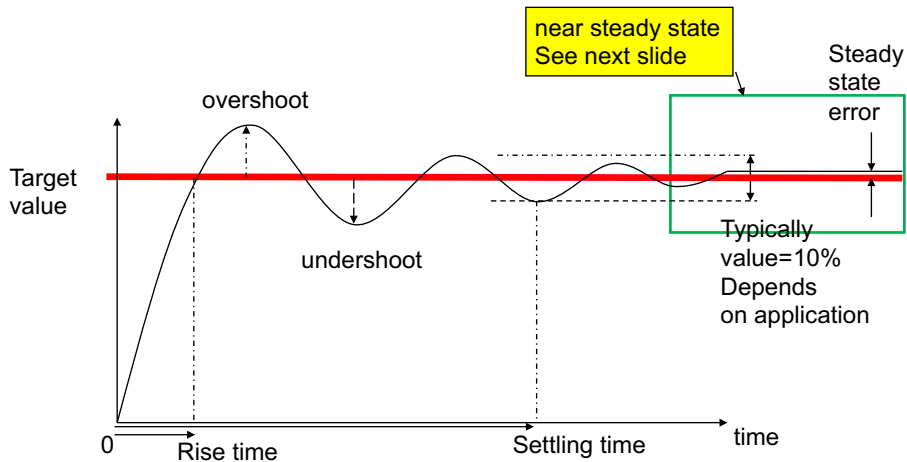
Integral Gain K_i

Larger K_i implies steady state errors are eliminated quicker. The trade-off is larger overshoot: any negative error integrated during transient response must be integrated away by positive error before we reach steady state.

Derivative Gain K_d

Larger K_d decreases overshoot, but slows down transient response and may lead to instability due to signal noise amplification in the differentiation of the error.

Parameters for Evaluating a Control System





Parameter	Rise Time	Overshoot	Settling Time	Steady state error
Kp (Pgain)	Decrease <u>step1</u>	Increase	Small Change	Decrease
Ki (Igain)	Decrease	Increase	Increase	Eliminate <u>step3</u>
Kd (Dgain)	Small Change	Decrease <u>step2</u>	Decrease	Small Change

Exercise



Please try to give the discrete incremental PID formulations. Some notations are given:

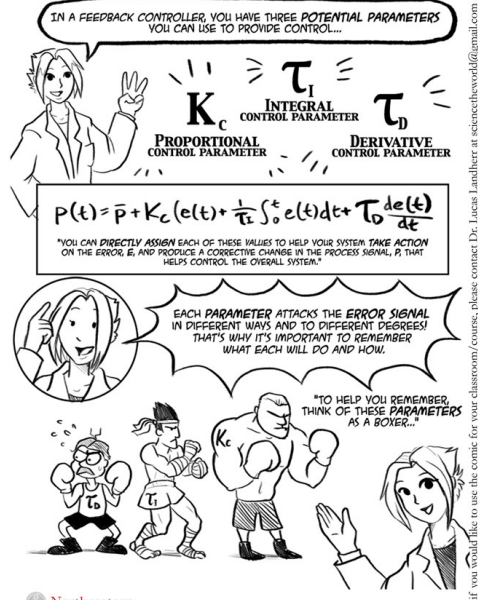
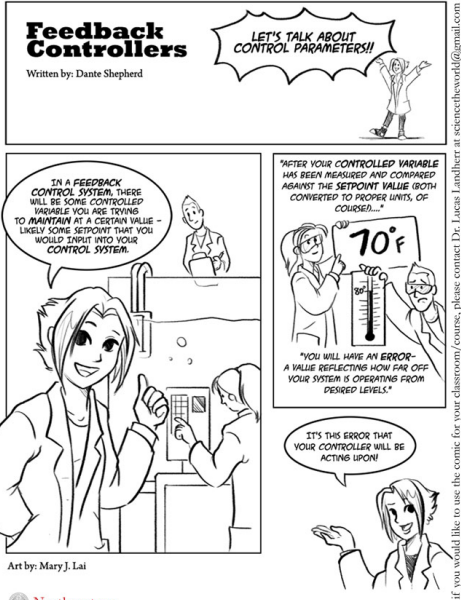
- $u(t)$ is the output of a controller in the t th measurement interval.
- $e(t)$ is the error between the target value and measurement value in the t th measurement interval. And the error is measured every T time interval (T is small enough).
- The PID parameters, K_p , K_i and K_d , are all set.

(Hint: incremental means $\Delta u(t) = u(t) - u(t-1)$.)



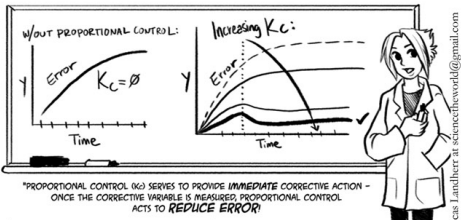
Easter egg 彩蛋







if you would like to use the comic for your classroom/course, please contact Dr. Lucas Landherr at sciencetheworld@gmail.com



"PROPORTIONAL CONTROL (K_c) SERVES TO PROVIDE IMMEDIATE CORRECTIVE ACTION - ONCE THE CORRECTIVE VARIABLE IS MEASURED, PROPORTIONAL CONTROL ACTS TO **REDUCE ERROR**!"

"THE SIZE OF THE PROPORTIONAL CONTROL ACTION IS DIRECTLY DEPENDENT ON THE SIZE OF THE PROPORTIONAL CONTROL PARAMETER!"

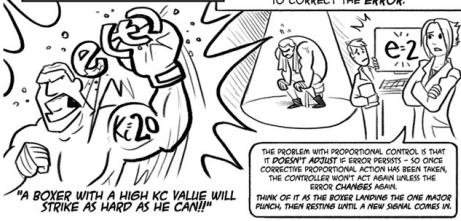


"THE LARGER THE PARAMETER VALUE, THE LARGER THE ACTION IT CAN TAKE!"

"SO IF WE HAVE OUR PROPORTIONAL BOXER, AND THE BOXER HAS A LOW K_c VALUE..."



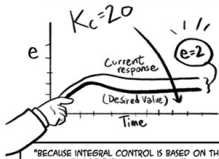
"...HE WILL ONLY GIVE A SMALL PUNCH TO CORRECT THE ERROR!"



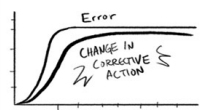
"A BOXER WITH A HIGH K_c VALUE WILL STRIKE AS HARD AS HE CAN!!!"

THE PROBLEM WITH PROPORTIONAL CONTROL IS THAT IT DOESN'T ADJUST IF ERROR PERSISTS - SO ONCE CORRECTIVE PROPORTIONAL ACTION HAS BEEN TAKEN, THE CONTROLLER WON'T ACT AGAIN UNLESS THE ERROR CHANGES AGAIN. THINK OF IT AS THE BOXER LANDING THE ONE MAJOR PUNCH, THEN RESTING UNTIL A NEW SIGNAL COMES IN.

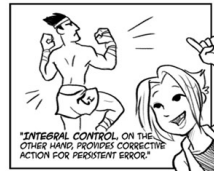
THIS IS WHY, WHEN DEALING WITH PROPORTIONAL-ONLY CONTROL, OFFSET DEVELOPS IN OUR CONTROLLED VARIABLE AND OUR CORRECTIVE RESPONSE.



"BECAUSE INTEGRAL CONTROL IS BASED ON THE DURATION OF THE ERROR..."



"...THE LONGER THE CONTROLLED VARIABLE IS AWAY FROM THE TARGET SETPOINT VALUE, THE MORE FORCEFUL THE CORRECTIVE ACTION WILL BE."



"INTEGRAL CONTROL, ON THE OTHER HAND, PROVIDES CORRECTIVE ACTION FOR PERSISTENT ERROR."



"SO IF YOU CONSIDER OUR INTEGRAL BOXER, THE SIZE OF THE PUNCHES PROVIDED MAY BE SMALL AT THE BEGINNING..."



"BUT THEY'LL GROW IN IMPACT AS THE DURATION OF THE ERROR CONTINUES!"



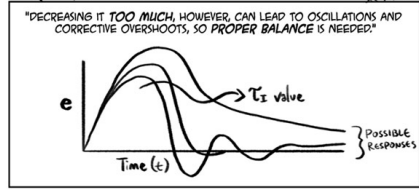
if you would like to use the comic for your classroom/course, please contact Dr. Lucas Landherr at sciencetheworld@gmail.com

SO, CHANGING THE **STRENGTH** OF THE INTEGRAL CONTROL CAN AFFECT BOTH THE **STRENGTH** AND **SPEED** OF THE CORRECTIVE RESPONSE.

$$p(t) = \bar{p} + K_c (e(t) + \frac{1}{T_I} \int_0^t e(t) dt + T_D \frac{de(t)}{dt})$$

"IT CAN BE A LITTLE COUNTER-INTUITIVE, THOUGH, ABOUT HOW TO FINE TUNE INTEGRAL CONTROL. BECAUSE THE T_I PARAMETER IS IN THE DENOMINATOR, INCREASING T_I ACTUALLY **REDUCES** THE INTEGRAL CONTROL CONTRIBUTION."

REDUCING T_I IS NECESSARY TO **REDUCE THE OFFSET** AND IMPROVE RESPONSE TIME WITH INTEGRAL CONTROL.



if you would like to use the comic for your classroom/course, please contact Dr. Lucas Landherr at sciencetheworld@gmail.com

"AT THIS POINT, WE HAVE MEANS OF CONTROLLING THE **SIZE** OF THE INITIAL CORRECTIVE RESPONSE TIME AND **OFFSET** FOR PROLONGED ERROR."

THE ONE ELEMENT OF POTENTIAL ERROR REMAINING IS IF DEVIATIONS IN THE CONTROLLED VARIABLE OCCUR **SUDDENLY** AND NEED MORE CORRECTIVE ACTION -

ACTION THAT IS FASTER THAN CAN BE PROVIDED BY INTEGRAL CONTROL!

"AFTER ALL, A DEVIATION OF 3" MAY NOT MEAN MUCH IF IT OCCURS OVER AN HOUR, BUT IF IT OCCURS OVER A FEW MINUTES?"

"YOU'LL PROBABLY HOPE YOUR POWER PLANT HAS SOME GOOD PROCESS CONTROL AT WORK!!"

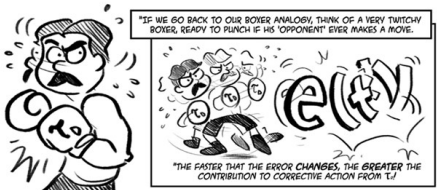
"DERIVATIVE CONTROL SERVES TO DETECT THE **RATE** THAT ERROR DEVELOPS (HENCE, DERIVATIVE), AND THEN PROVIDES CORRECTIVE ACTION IN RESPONSE."

WHICH LEADS US TO OUR FINAL PARAMETER, T_D , FOR DERIVATIVE CONTROL.

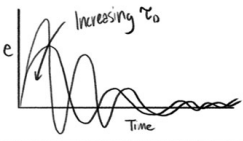
T_I T_D K_c



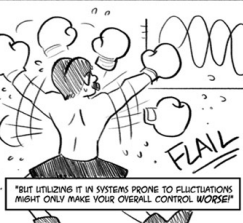
if you would like to use the comic for your classroom/course, please contact Dr. Lucas Landherr at sciencetheworld@gmail.com



"DERIVATIVE CONTROL HAS ITS DISADVANTAGES - IF THERE TEND TO BE FLUCTUATIONS IN YOUR SYSTEM, LIKE WITH FLOWRATES, THEN THE SMALL BUT RAPID FLUCTUATIONS CAN BE MAGNIFIED BY T_d AND LEAD TO FURTHER OSCILLATIONS IN YOUR SIGNAL!"

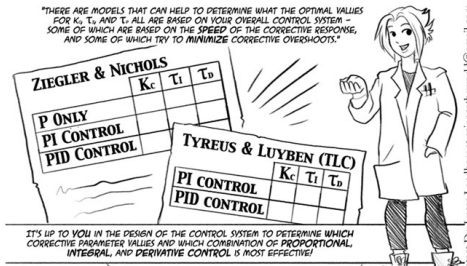


"SO, INCREASING T_d DOES HELP TO REDUCE OVERSHOOTS OR OSCILLATIONS..."



Northeastern

if you would like to use the comic for your classroom/course, please contact Dr. Lucas Landherr at sciencetheworld@gmail.com



Northeastern

survivingtheworld.net

sciencetheworld.com

drawn by Mary J. Lai



- ① Motors
- ② Open-loop and Closed-loop Control
- ③ Control Methods
- ④ Software



<https://youtu.be/Lym2UxUh81Q>

```
int main(void)
{
+-- 23 lines: -----

    tmpjp = IO0PIN & JUMPER;           // check function selection jumper
    if(tmpjp==0) {                     // if jumper is set then print X, Y value
+-- 15 lines: -----
    }
    else {                             // else run self balancing demo
+-- 34 lines: -----
        init_timer();                 // Init TIMER 0
    }
}

while(1) {
}

}

void __irq IRQ_Exception()
{
+-- 62 lines: -----
}

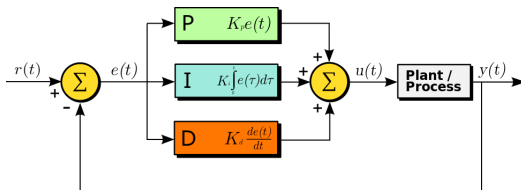
/* Setup the Timer Counter 0 Interrupt */
void init_timer (void) {
    T0PR = 0;                          // set prescaler to 0
    T0MR0 = 27648;                     // set interrupt interval to 1mS
                                        // Pclk/500Hz = (11059200 x 5)/(4 x 1000)
    T0MCR = 3;                        // Interrupt and Reset on MR0
    T0TCR = 1;                        // Timer0 Enable
    VICVectAddr0 = (unsigned long)IRQ_Exception; // set interrupt vector in 0
    VICVectCntl0 = 0x20 | 4;          // use it for Timer 0 Interrupt
    VICIntEnable = 0x00000010;        // Enable Timer0 Interrupt
}
```



```
void __irq IRQ_Exception()
{
    //{{{
    tmp1 = read_sensor(0);
    if (tmp1 >= (MIDL+50)) {
        delta1 = (tmp1 - (MIDL+50))/200;
        diff1 = delta1 - last1;
        if (diff1 < maxdiff) {
            last1 = delta1;
            leftPWM = leftPWM - (P*delta1 - I*accu1 + D*diff1);
            if (leftPWM < MINOUTPUT) leftPWM = MINOUTPUT;
            if (accu1 < maxaccu) accu1 += delta1/200;
            PWMMR2 = leftPWM;
            PWMLER = 0x44;
        }
    }
    // read X-axis value
    // if X-axis value >= setpoint plus 50
    // calculate the error and normalize it
    // calculate the different between current and last error
    // ignore if the error different > max. difference
    // this prevent the noise due to undesired movement of accelerometer
    // save error as the last error
    // update the left PWM value by PID
    // limit the PWM value to its minimum
    // ensure the integral not exceed the maximum
    // set the left PWM output
    // enable match 2,6 latch to effective
}
```

Pay attention to the following variables:

- P, I, D: **to tuned**
- PWMMR2, PWMLER

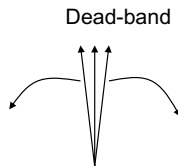


```
if (tmp1 >= (MIDL+50)) {  
    delta1 = (tmp1 - (MIDL+50)) / 200;  
    .....  
}
```

Dead-band

A Dead-band (sometimes called a neutral zone) is an area of a signal range or band where no action occurs.

- Only enable motor when $tmp1 >$ a small value (deadband, ie = 50)
- Otherwise may oscillate when $tmp1$ is small



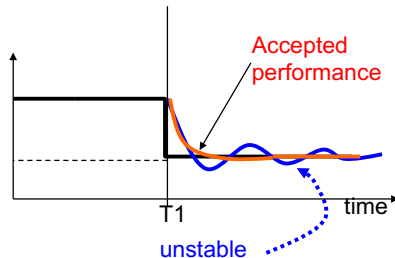
Usually done by trail and error

① Tune (adjust manually)

- step1: K_p
- step2: K_d
- mstep3: K_i

② Record the angle by the computer to see if the performance is ok or not

- Yes, then done.
- If no, go to first step again





```
#include <PID_v1.h>
double Setpoint, Input,      Output;
double aggKp=4,  aggKi=0.2,  aggKd=1;
double consKp=1, consKi=0.05, consKd=0.25;

PID myPID(&Input, &Output, &Setpoint, consKp, consKi, consKd, DIRECT);

void setup() {
    Input      = analogRead(0);
    Setpoint = 100;
    myPID.SetMode(AUTOMATIC); //turn the PID on
}

void loop() {
    Input = analogRead(0);
    double gap = abs(Setpoint-Input); //distance away from setpoint
    if(gap<10) { //we're close to setpoint, use conservative tuning parameters
        myPID.SetTunings(consKp, consKi, consKd);
    }
    else { //we're far from setpoint, use aggressive tuning parameters
        myPID.SetTunings(aggKp, aggKi, aggKd);
    }
    myPID.Compute();
    analogWrite(3,Output);
}
```




- Studies PID control theory
- PID implementation