CENG 4480

Embedded System Development & Applications

Lecture 07: PID Control

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Latest update: August 9, 2021)

Fall 2021

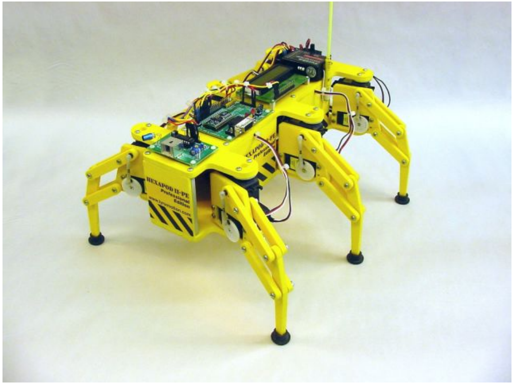
Overview

Overview

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

DC Motor and Servo Motor

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

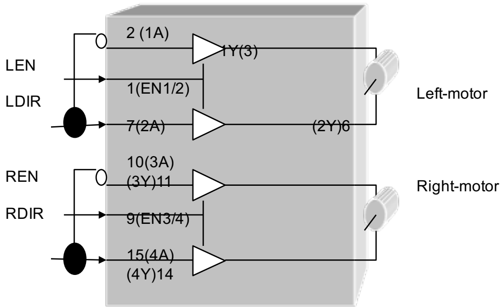


Servo motors for making robot legs
http://www.lynxmotion.com/

Small Direct Current D.C. Motors

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

Open-loop Motor Control and its Problems

Change motor supply power change speed

Problem: How much power is right?

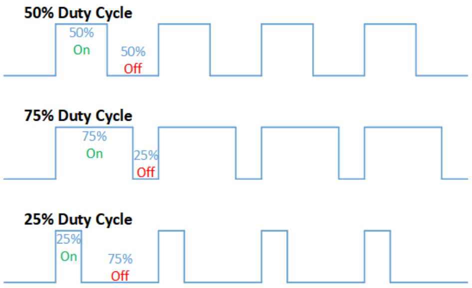
Ans: don't know , depends on internal/external frictions of individual motors.

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

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

PWM Signal

- 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.

LPC2138 PWM Configuration (Optional)

- 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.

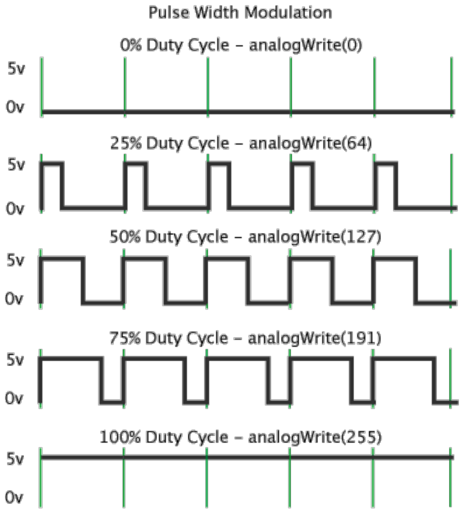
Table 181. Set and reset inputs for PWM Flip-Flops

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)



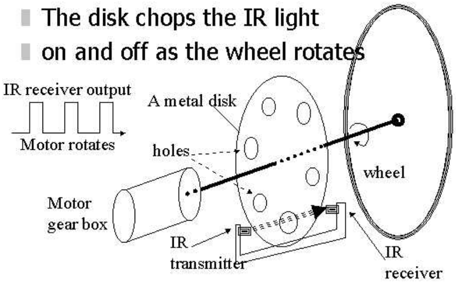
- The real solution to real speed control is feedback control
- Require speed encoder to read back the real speed of the wheel at real time.

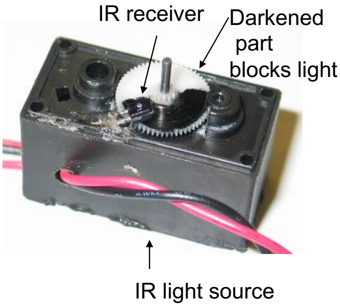
First you need to have speed encoders

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

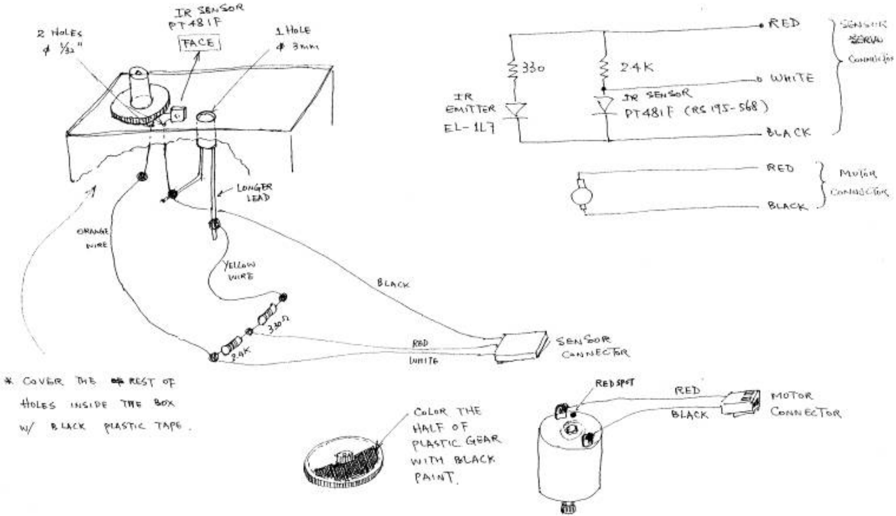
Wheel Encoder

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

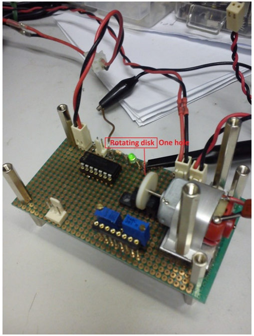


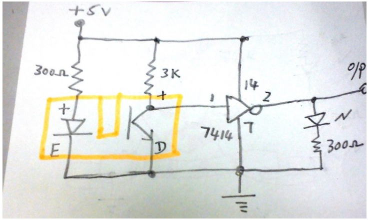


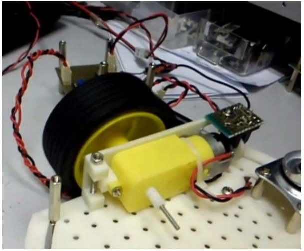
SERVO MOTOR MODIFICATION



New Speed

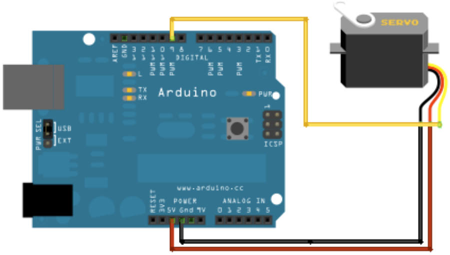






https://youtu.be/7qf_ypIGn_0

Servo library in Arduino

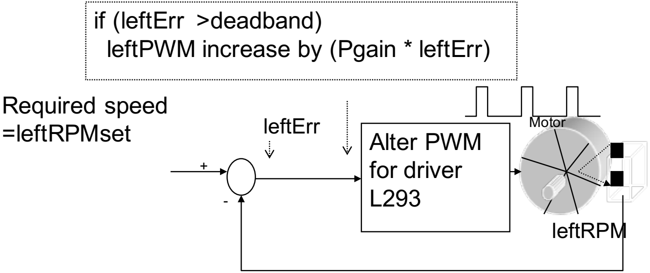



```
#include <Servo.h>

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_ql3Fg

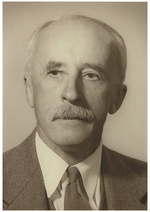
Overview

Closed-loop feed back control



Note: Show the left motor control only

PID Control

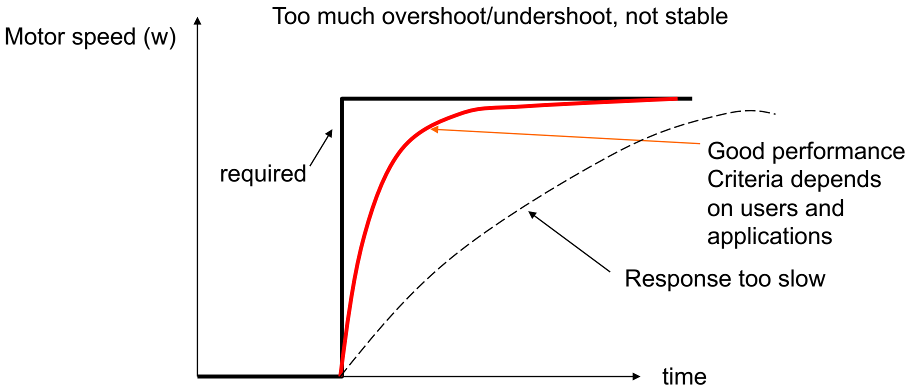


- PID: Proportional-Integral-Derivative
- A more formal and precise method used in most modern machines

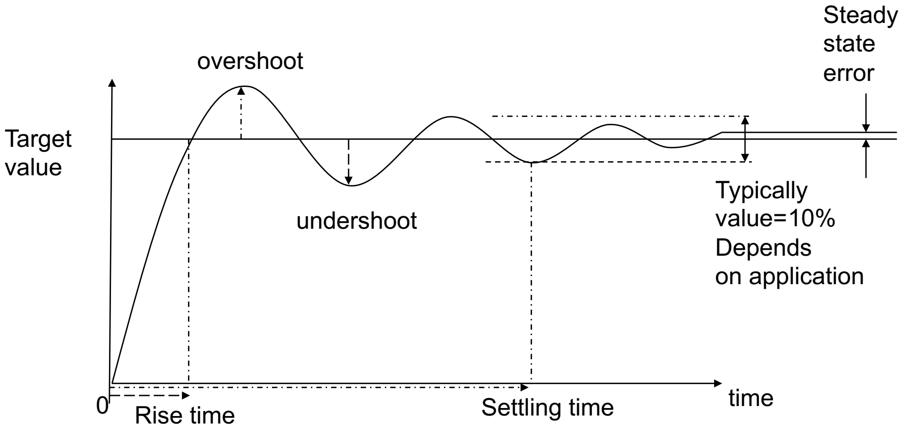
History of PID

- 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 n the following diagrams:

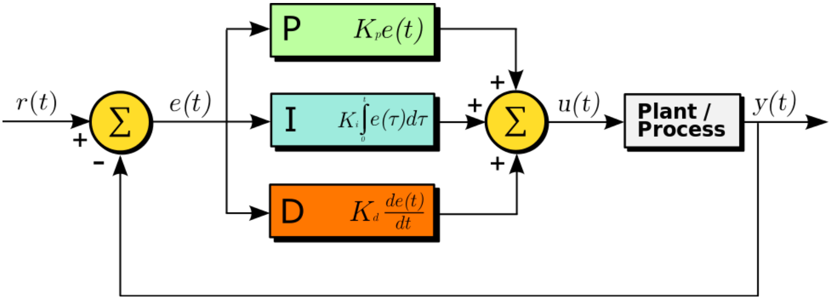


PID Control

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

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)



$r(t)$
Σ
+
-
$e(t)$
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau) d\tau$
D
$K_d \frac{de(t)}{dt}$
+
+
+
Σ
$u(t)$
Plant / Process
$y(t)$

PID – Control Terms Are Intertwined

Proportional Gain K_p

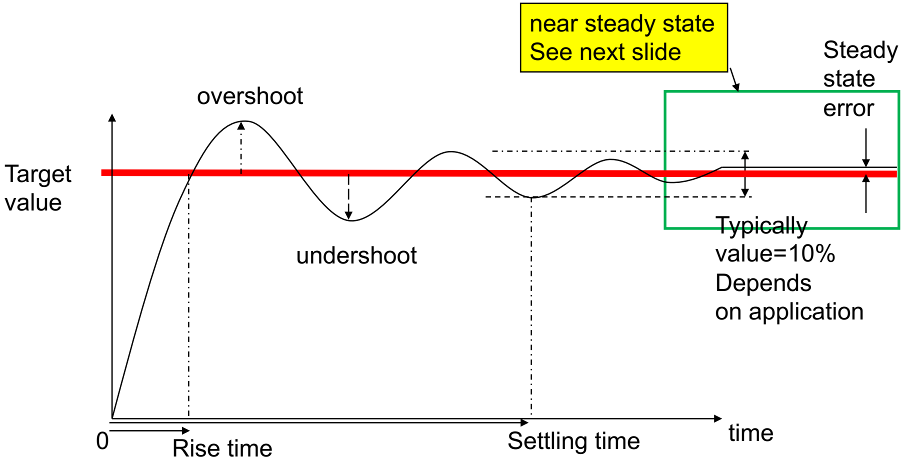
Larger K_p typically means faster response since the larger the error, the larger the Proportional term compensation. An excessively large proportional gain will lead to process instability and oscillation.

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.


near steady state
See next slide
Steady
state
error
overshoot
Target
value
undershoot
Typically
value=10%
Depends
on application
0
Rise time
Settling time
time

Effects of Increasing Parameters

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

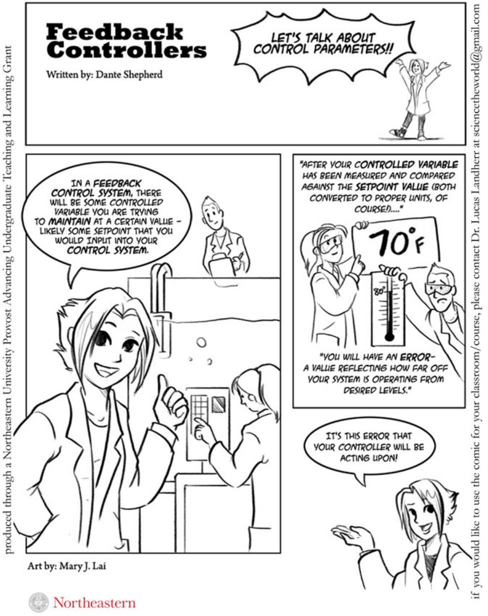
Exercise

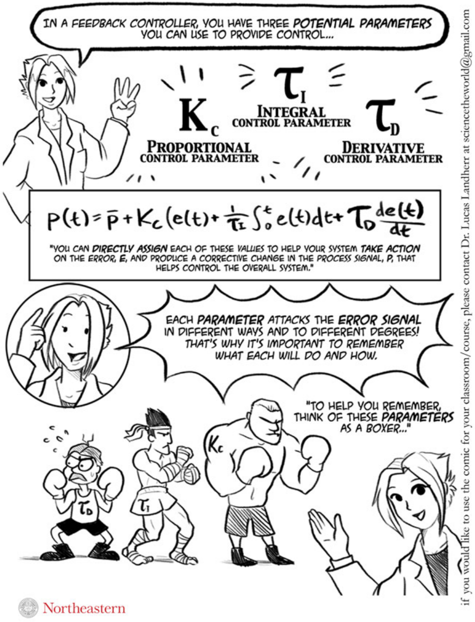
Please try to give the discrete incremental PID formulations. Some notations are given:

- $u(t)$ is the output of a controller in the tth measurement interval.
- $e(t)$ is the error between the target value and measurement value in the tth 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.

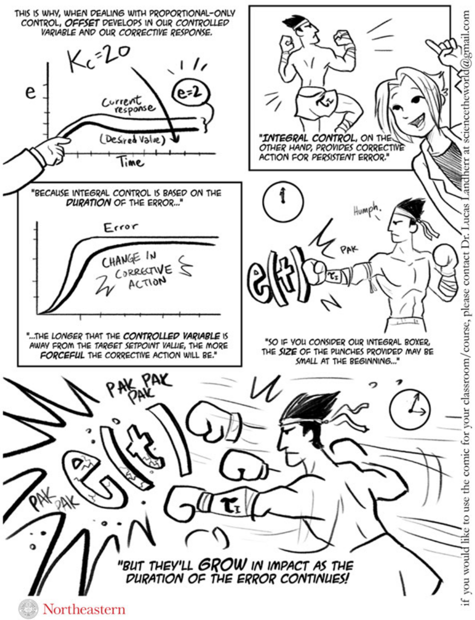
(Hint: incremental means $\Delta u(t) = u(t) - u(t-1)$.)

Easter egg 彩蛋



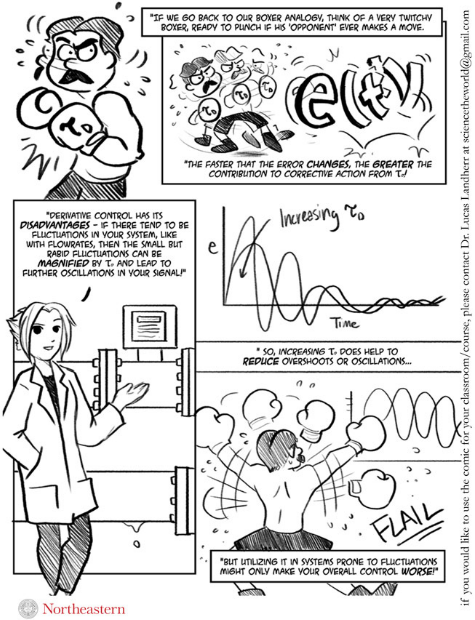


Source: http://survivingtheworld.net/ScienceComic3.html



Source: http://survivingtheworld.net/ScienceComic3.html

Source: http://survivingtheworld.net/ScienceComic3.html



Source: http://survivingtheworld.net/ScienceComic3.html

Overview

Overview

https://youtu.be/Lym2UxUh81Q

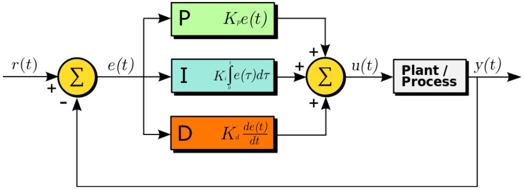
```
int  main(void)
{
+-- 23 lines: ------------------------------------------------------------------------

 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
    init_timer();               // Init TIMER 0
+-- 34 lines: ------------------------------------------------------------------------

    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
}
```

Algorithm for PID Core

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

Pay attention to the following variables:

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

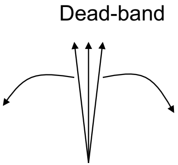


```
if (tmpl>=(MIDL+50)) {
    deltal = (tmpl - (MIDL+50))/200;
    ......
}
```

Dead-band

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

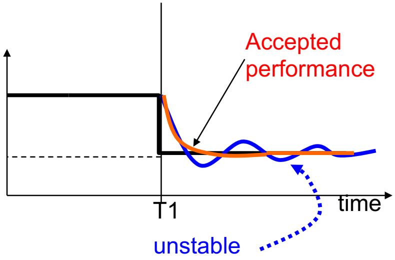
- Only enable motor when `tmpl` > a small value (deadband, ie = 50)
- Otherwise may oscillate when tmpl is small



PID Tuning

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



Arduino PID Library

```
#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);
}
```

Summary

- Studies PID control theory
- PID implementation