PID Controls
(Part II)

Howie Choset
(thanks to George Kantor and Wikipedia)

http://www.library.cmu.edu/ctms/ctms/examples/motor/motor.htm
Overview

• Mass-Spring-Damper System
• Second order ODE
  – Definition
  – Vary parameters
  – Forcing functions
• Different feedback meaning
  – Proportional
  – Derivative
• Control for Error – block diagram
• Integral Control
• Different Affects of Varying PID
• Feed Forward Term
• Vehicle Controls
Big Dog Quadruped

Boston Dynamics
Nathan Michael Quadrotors

Controls

Estimation
RC Airplane - Adaptive Control

Chowdhary G., Johnson E., Chandramohan R., Kimbrell M. S., Calise A.
Mass Spring

\[ F_s = -kx \]

\[ F_{\text{tot}} = ma = m\frac{d^2x}{dt^2} = m\ddot{x} \]

\[ F_{\text{tot}} = F_s \]

\[ m\ddot{x} = -kx \]

\[ \ddot{x} + \frac{k}{m}x = 0. \]
Solutions/Responses

Critical damping ($\zeta = 1$)

$$x(t) = (A + Bt)e^{-\omega_0 t}$$

$$A = x(0)$$

$$B = \dot{x}(0) + \omega_0 x(0)$$

Over-damping ($\zeta > 1$)

$$x(t) = Ae^{\gamma_+ t} + Be^{\gamma_- t}$$

$$A = x(0) + \frac{\gamma_+ x(0) - \dot{x}(0)}{\gamma_- - \gamma_+}$$

$$B = -\frac{\gamma_+ x(0) - \dot{x}(0)}{\gamma_- - \gamma_+}.$$ 

Under-damped ($0 < \zeta < 1$)

$$x(t) = e^{-\zeta \omega_0 t}(A \cos(\omega_d t) + B \sin(\omega_d t))$$

$$\omega_d = \omega_0 \sqrt{1 - \zeta^2}$$

$$A = x(0)$$

$$B = \frac{1}{\omega_d} (\zeta \omega_0 x(0) + \dot{x}(0)).$$

Let $\gamma = \omega_0 \left(-\zeta \pm \sqrt{\zeta^2 - 1}\right)$
Step Response

\[ \frac{d^2 x}{dt^2} + 2\zeta\omega_0 \frac{dx}{dt} + \omega_0^2 x = \frac{F(t)}{m}, \]

\[ \frac{F(t)}{m} = \begin{cases} \omega_0^2 & t \geq 0 \\ 0 & t < 0 \end{cases} \]

\[ x(t) = 1 - e^{-\zeta\omega_0 t} \frac{\sin \left( \sqrt{1 - \zeta^2} \omega_0 t + \varphi \right)}{\sin(\varphi)} \]

\[ \cos \varphi = \zeta. \]

As time goes on, \( x(t) \) goes to 1.
controller tells your system to do something, but doesn’t use the results of that action to verify the results or modify the commands to see that the job is done properly.
Closed Loop Controller

Give it a velocity command and get a velocity output

Controller Evaluation
Steady State Error
Rise Time (to get to ~90%)
Overshoot
Settling Time (Ring) (time to steady state)
Stability

Ref + error voltage Ref Controller Plant \( \dot{\theta} \)
Closed Loop Response (Proportional Feedback)

Proportional Control $K_p$

Easy to implement
Input/Output units agree
Improved rise time

Steady State Error (true)

$\uparrow P$: $\downarrow$ Rise Time vs. $\uparrow$ Overshoot*
$\uparrow P$: $\downarrow$ Rise Time vs. $\downarrow$ Settling time*
$\uparrow P$: $\downarrow$ Steady state error vs. other problems

$\text{Voltage} = K_p \text{ error}$

*In some other systems, not mass-spring
Closed Loop Response (PI Feedback)

Proportional/Integral Control

\[ K_p + \frac{1}{s} K_i \]

No Steady State Error

Bigger Overshoot and Settling
Saturate counters/op-amps

\( \uparrow P: \downarrow \text{Rise Time vs. } \uparrow \text{Overshoot} \)
\( \uparrow P: \downarrow \text{Rise Time vs. } \downarrow \text{Settling time} \)

\( \uparrow I: \downarrow \text{Steady State Error vs. } \uparrow \text{Overshoot} \)

\[ \text{Voltage} = (K_p + \frac{1}{s} K_i) \text{ error} \]
Closed Loop Response (PID Feedback)

Proportional/Integral/Differential

Quick response
Reduced Overshoot

Sensitive to high frequency noise
Hard to tune

\[ K_p + \frac{1}{s} K_I + sK_D \]

\[ V_{\text{o}} = (K_p + \frac{1}{s} K_i + sK_d) \text{error} \]

\[ \uparrow \text{P:} \downarrow \text{Rise Time vs.} \uparrow \text{Overshoot} \]
\[ \uparrow \text{P:} \downarrow \text{Rise Time vs.} \downarrow \text{Settling time} \]
\[ \uparrow \text{I:} \downarrow \text{Steady State Error vs.} \uparrow \text{Overshoot} \]
\[ \uparrow \text{D:} \downarrow \text{Overshoot vs.} \uparrow \text{Steady State Error} \]
Quick and Dirty Tuning

- Tune P to get the rise time you want
- Tune D to get the setting time you want
- Tune I to get rid of steady state error
- Repeat

- More rigorous methods – Ziegler Nichols, Self-tuning,
- Scary thing happen when you introduce the I term
  – Wind up (example with brick wall)
  – Instability around set point
Feed Forward
Decouples Damping from PID

To compute $K_b$
Try different open loop inputs and measure output velocities
For each trial $i$,
**Tweak** from there.

$$K_b^i = \frac{u_i}{\dot{\theta}_i}, \quad K_b = \text{avg } K_b^i$$
Mobile Robot

- planar workspace
Mobile Robot

- planar workspace
- position of robot and goal are known
Mobile Robot

- planar workspace
- position of robot and goal are known
- omni-directional robot
Mobile Robot

- planar workspace
- position of robot and goal are known
- omni-directional robot
- control input is velocity:

\[
\begin{bmatrix}
  u_x \\
  u_y
\end{bmatrix}
= \begin{bmatrix}
  v_x \\
  v_y
\end{bmatrix}
\]
Mobile Robot

- planar workspace
- position of robot and goal are known
- omni-directional robot
- control input is velocity:

\[
\begin{bmatrix}
  u_x \\
  u_y
\end{bmatrix}
= \begin{bmatrix}
  v_x \\
  v_y
\end{bmatrix}
\]

(boldface lie, we’ll relax this later, too)
Proportional (P) Control:

- The equation above is called a control law.
- $k_p$ is called the proportional gain.
- $k_p$ is a tunable parameter.
- Physically, $k_p$ is the stiffness of the spring.
Proportional-Derivative (PD) control:

Fill the world with honey!

\[
\begin{bmatrix}
 u_x \\
 u_y 
\end{bmatrix} =
-k_p \begin{bmatrix}
 x_g - x_r \\
 y_g - y_r 
\end{bmatrix} - k_d \begin{bmatrix}
 v_x \\
 v_y 
\end{bmatrix}
\]

- \( k_d \) is called the **derivative gain**
- \( k_p \) and \( k_d \) are tunable parameters
- physically, \( k_d \) is the damping term
- all of the stuff about P control still applies
Robot Inputs

So far we’ve assumed something like

\[
\begin{bmatrix}
\dot{x}_r \\
\dot{y}_r
\end{bmatrix}
= 
\begin{bmatrix}
u_x \\
u_y
\end{bmatrix}
\]

But really, we control the velocities of the left and right wheels, which can easily be mapped to forward and turning velocities:

\[
\begin{bmatrix}
v_l \\
v_r
\end{bmatrix}
\Rightarrow 
\begin{bmatrix}
v_f \\
\omega
\end{bmatrix}
\]
Nonholonomic Constraints

The equations of motion using these controls are:

\[
\begin{bmatrix}
\dot{x}_r \\
\dot{y}_r \\
\dot{\theta}_r
\end{bmatrix} =
\begin{bmatrix}
v_f \cos \theta \\
v_f \sin \theta \\
\omega
\end{bmatrix}
\]

The fact that the robot can’t move sideways is a nonholonomic constraint (we will see this again).
The Problem:

P or PD control won’t work.

No smooth control law will!
A Simple Solution:

Like a rigid trailer hitch (not driving to point)

\[
\begin{bmatrix}
    x_p \\
    y_p \\
    \theta_r
\end{bmatrix} =
\begin{bmatrix}
    x_r + \ell \cos \theta \\
    y_r + \ell \sin \theta \\
    \theta_r
\end{bmatrix}
\]

\[
\Rightarrow
\begin{bmatrix}
    \dot{x}_p \\
    \dot{y}_p \\
    \dot{\theta}_r
\end{bmatrix} =
\begin{bmatrix}
    \dot{x}_r - \ell \dot{\theta}_r \sin \theta_r \\
    \dot{y}_r + \ell \dot{\theta}_r \cos \theta_r \\
    \omega_r
\end{bmatrix} =
\begin{bmatrix}
    v_f \cos \theta_r - \omega \ell \sin \theta_r \\
    v_f \sin \theta_r + \omega \ell \cos \theta_r \\
    \omega
\end{bmatrix}
\]
A Simple Solution (cont.):

If we ignore orientation:

\[
\begin{bmatrix}
\dot{x}_p \\
\dot{y}_p \\
\end{bmatrix} =
\begin{bmatrix}
\cos \theta_r & -\sin \theta_r \\
\sin \theta_r & \cos \theta_r \\
\end{bmatrix}
\begin{bmatrix}
\nu_f \\
\omega \ell \\
\end{bmatrix}
\]

\[
\Rightarrow
\begin{bmatrix}
\nu_f \\
\omega \ell \\
\end{bmatrix} =
\begin{bmatrix}
\cos \theta_r & \sin \theta_r \\
-\sin \theta_r & \cos \theta_r \\
\end{bmatrix}
\begin{bmatrix}
\dot{x}_p \\
\dot{y}_p \\
\end{bmatrix}
\]

so we can implement the PD control law as:

\[
\begin{bmatrix}
\nu_f \\
\omega \ell \\
\end{bmatrix} =
\begin{bmatrix}
\cos \theta_r & \sin \theta_r \\
-\sin \theta_r & \cos \theta_r \\
\end{bmatrix}
\left(-k_p \begin{bmatrix} x_g - x_p \\ y_g - y_p \end{bmatrix} - k_d \begin{bmatrix} v_x \\ v_y \end{bmatrix}\right)
\]

Did not get rid of nh constraint, but moved it to something we don’t care about (theta, angular and linear velocities) - trailer hitch story
Follow a straight line with differential drive or at least get to a point

Error can be difference in wheel velocities or accrued distances

Make both wheels spin the same speed
  asynchronous – false start
  wheels can have slight differences (radius, etc)
Make sure both wheels spin the same amount and speed
  false start
Line following

More complicated control laws – track orientation

\[ m1vref = vref + K1 \times \thetaerror + K2 \times \text{offset error} \]
\[ m2vref = vref - K1 \times \thetaerror - K2 \times \text{offset error} \]
Really, there is a sensor
Encoders
Encoders – Incremental

- Photodetector
- Encoder disk
- LED Photoemitter
Encoders - Incremental

fixed sensors

A  
B  
INDEX

0°  ➡️  360°

direction of positive track motion

A  
1
0

B  
1
0

INDEX  
1
0

©2000 HowStuffWorks
Encoders - Incremental

- Quadrature (resolution enhancing)
To be continued

- Maps
- Bayesian Localization