Robot Motion Control Using Sensor Feedback

Jim Zondag - DaimlerChrysler
Senior Software Engineer
Team #33 - Killer Bees

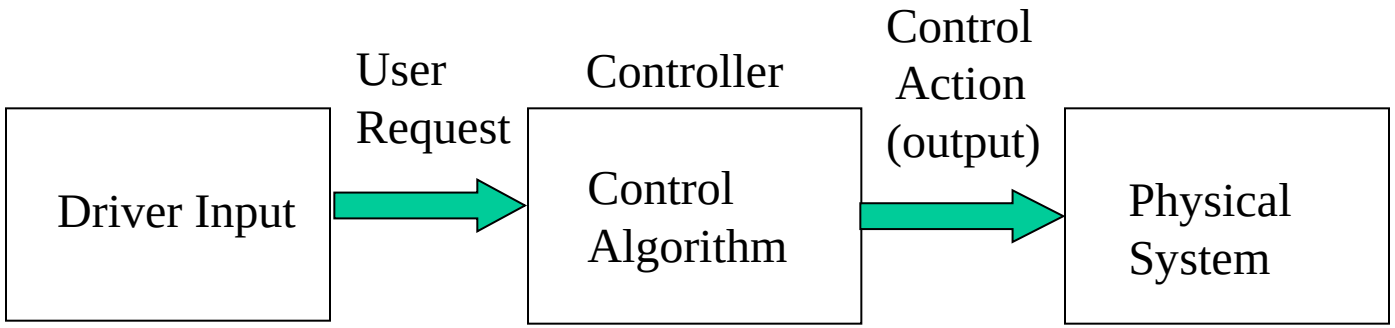
Agenda

- Simple Control Theory
- Proportional Control
  - Demonstration
  - Code Review
  - Stampplot Introduction
- Proportional Control with Variable Biasing
  - Demonstration
  - Code Review
- Accurate Time Delays
- Questions

Basic Controls Philosophy - Why do this?

- **The goal of any controls you add should be to make you robot simpler!** It may become slightly more complex in its creation, but it should get simpler to drive, operate, and adjust as a result.
- Intuitive easy to use controls can make up for a lack of practice time, something we all seem to need.
- Controls cannot make up for a poor mechanical design. Controls can make a good design better, but will not add any value if the system is not capable or controllable. Be sure to do a good job of sizing motors, picking gear ratios, tensioning chains, etc.
- Garbage in = garbage out. Always use good quality sensors and potentiometers, and protect your sensors and wires from damage. your entire control system depends on good sensor data.
- Remember that there is often a good mechanical solution to many problems which requires no controls.

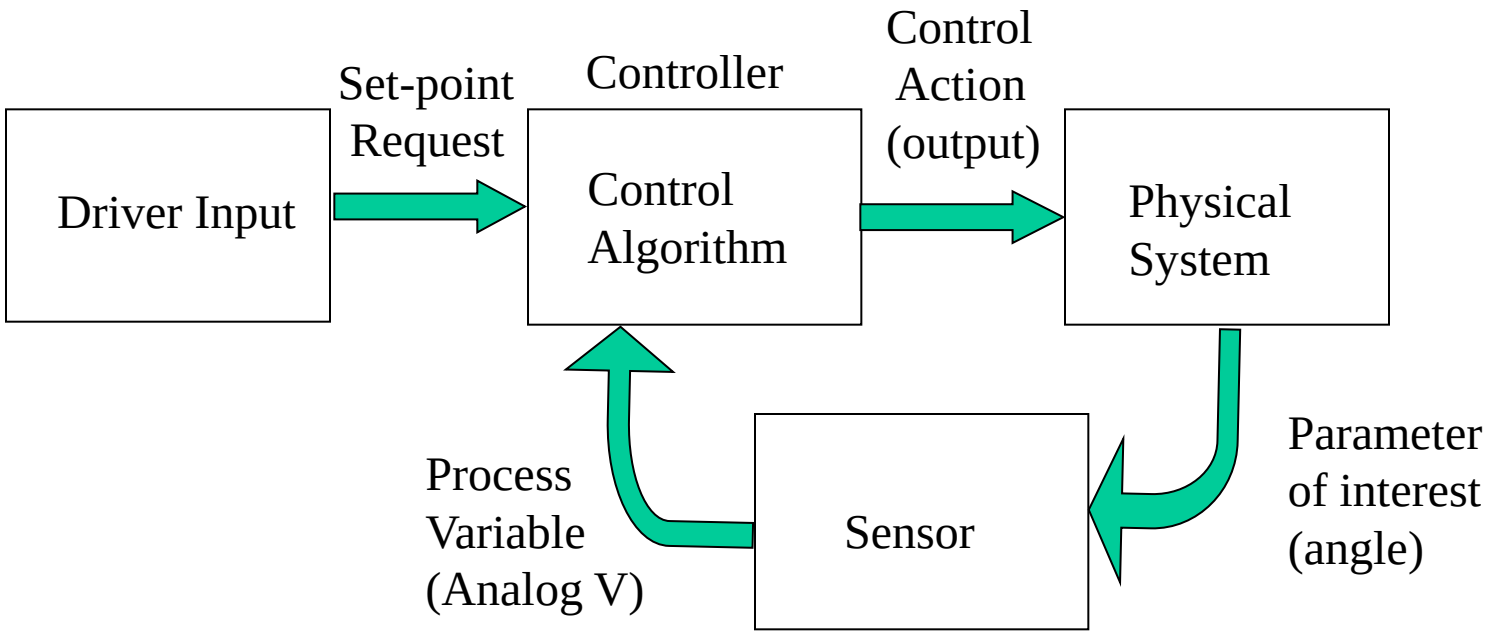
Open Loop Control System
Default Control



In an open loop system such as the default program, the user is in direct control of the physical system. There is nothing in the system to allow the robot to correct its position without the user's input.

Such systems often require skill to operate and practice to master.

Simple Feedback Control System
Closed Loop

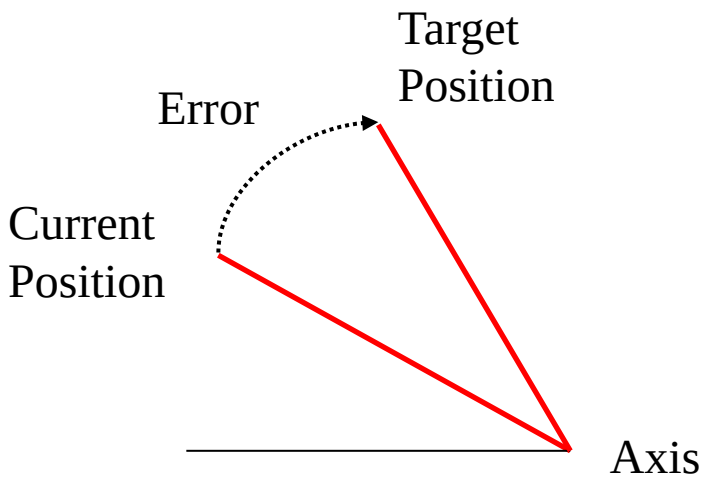


In a feedback control system the user does not have direct control of the controller outputs. Instead the user provides set-point change requests to the controller and the control algorithm(s) manages the system output based upon the readings taken from the sensor measurements of the system.

Potentiometers

- Use only good sensors/potentiometers - garbage in = garbage out. Your entire system depends upon good sensor data, never use cheap sensors. I like Bourns PN 3852A-282-104A (Newark PN 01F9202).
- Positioning accuracy is determined by sensor resolution. The robot controller can only resolve differences of about 1 degree (.0196 V) on a standard sensor. Always try to maximize the amount of rotation in your sensor by using small gears or pulleys if you are controlling device with a small rotation angle.
- Most standard potentiometers have about 270^{o} of rotation, try to use at least 180^{o}. Avoid allowing motion to come close to the end stops.
- Consider having default manual control if sensor data is lost.
- Guard your sensors and wiring from damage.
- Make your sensors easy to change.
- Depending on your application, a digital data filter can be a very good thing (see Mike Gray's 2002 kickoff presentation) to help with noise rejection.

Proportional Error Term

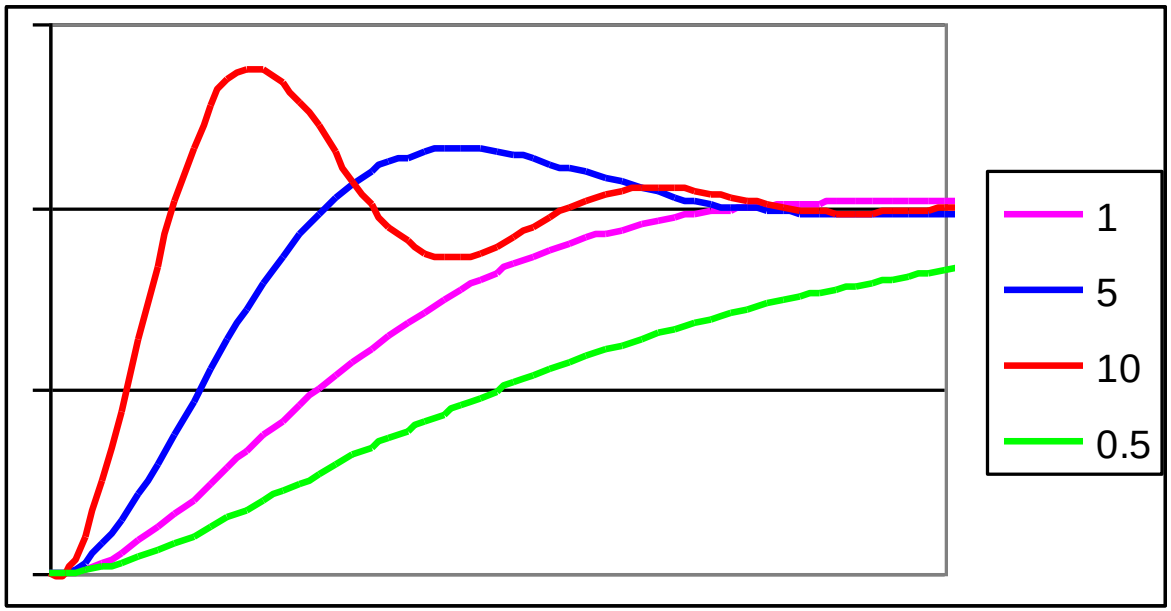


Error = difference between current position and the desired position.
Ideally, Error is always zero.
Practically, the laws of physics require finite time to make position changes.
Your robot controller operates in discrete time increments of 25ms (1/40 sec).
The complexity of your code will determine how often you can make adjustments to your system.

Proportional Control

- Error = (target position - current position)
- Error Drive = Gain * Error
- If Error is big, Error Drive to the motor is fast. The Controller slows the motor as position approaches target.
- The Gain value determines how much the system will respond to Errors from the target position.
- The bigger the Gain value, the more aggressively the system will react.
- Use care when tuning your gain value, too much can be destructive. Start small and work up.
- Be sure to remove all debug statements and rerun your system after you get it tuned. Your system will be faster without debug statements and this can often change your results.

Gain Examples



- More gain means faster response but more overshoot.
- In most robots, minimizing overshoot is important.
- For gains less than 1, use a 10X multiplier or similar.
- Beware that very small gains will cause round-off losses.
- Remember that overshoot will get worse the slower your program gets.

PBasic Implementation

- Remember, no negative numbers and no fractional numbers in Pbasic.

```
    pwmout = 127
    error       = abs(shoulder-target)
    if target = shoulder then done_prop
     pwmout = ((error) * gain)/10 MIN 0 MAX 127
     if target > shoulder then calc_prop
Neg_prop:
    pwmout = 127 - pwmout MIN 10 MAX 127
    goto done_prop
calc_prop:
    pwmout = 127 + pwmout MIN 127 MAX 244
Done_prop:
```

- Or can be done with all one equation:

pwmout = (((((2000 + shoulder - target + 127)Min 2000 Max 2254) - 2000) * gain)/10) Min 0 Max 254

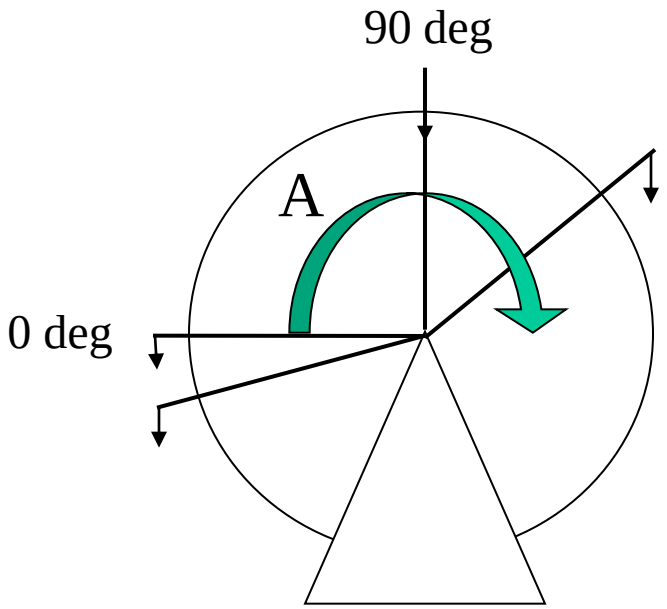
Do Prop_demo here

- File prop_demo1.bsx
- run stampplot demo

Biasing Theory

- A large load mass changes the system controller requirements.
- Motor will back-drive if drive current gets too low due to high torque applied by load; thus a constant error will always remain and target position cannot be reached.
- Trying to use large gains to reach the needed drive levels is a bad idea, there is too much inertia in the system to allow high velocities.
- Proportional-only system cannot be made to work well, another control parameter is needed to bring the system under control.
- In a simple system with a small arc, a fixed bias number can be added to the calculated proportional drive.
- In our system, the amount of biasing needed changes by up to 200% so more sophistication is needed.
- The new control scheme is: pwmout = Error Drive + Bias Drive

In our system, the torque required to keep the arm at a constant position is a function of the angle A from Horizontal.



Bias Drive = $Bias_0$ * COS A

$Bias^0$ = maximum bias current at 0 degrees.

Bias goes to zero at vertical, goes 100% negative at 180^o

Trig Functions in Pbasic

- Pbasic supports SIN and COS functions (no TAN or inverse trig).
- The SIN and COS operators break a circle into 0 to 255 units (brads) instead of the 0 to 359 degrees used in traditional decimal math.
- One brad = 1.406 degrees
    - 90^o = 64 brads, 180^o = 128 brads
- In decimal math, the results of trig functions are always a value of -1 to +1. This can't be done in digital math, all numbers must be integers.
- For this reason these functions return a **signed number** which is scaled against a unit circle of 127. I prefer to take the absolute value of the result and then check against the vertical axis to determine whether to add or subtract the bias.

Some Trig Examples

- In decimal:

  100 * Cos (45º)
  = 100 * 0.707
  = 70.7

  20 * Cos (150º)
  = 20 * -0.866
  = -17.32

In Pbasic:

(100 COS (32))/127
= (100 * 89)/ 127
= 70 **(No Problem)**

(20 COS (106))/127
= (20 *(-109))/ 127
Note: ‘(237 = -109)
in Binary 109 = %01101101
-109 = %11101101 = 237
MSB of 1 indicates negative sign in 2’s compement
The Division operator does not support negative numbers!
So in the negative case:
= (20 * 237)/ 127
= 37 **(Wrong Answer!! Does not work this way!!!)**

Implementation of Bias Calculation

```
        bias    VAR      p2_y
bias_calc:
        bias = bias / 4  MIN 1        'knob input
allows easy adjustment
        bias =(bias * abs(cos(brads)))/127
        if brads =< 64 or brads > 192 then pos_bias
 neg_bias:
        pwmout = pwmout - bias  MIN 10 MAX 244
        goto done_b
 pos_bias:
        pwmout = pwmout + bias   MIN 10 MAX 244
 done_b:
```

for above example: bias = 17, pwmout = 127 - 17 = 110

(bias = -17 : CORRECT)

Do Bias_demo here

- File bias_demo1.bsx
- run stampplot demo

Some Tips for Biasing

- Only certain motors are suitable for use in this type of application.
  - Automotive style motors (window, seat, van door) have large field magnets and low speed armatures and are designed to be stalled for short periods.
  - High speed motors (Drill, Fisher Price) are not designed to be stalled and will fail quickly if you try to stall them.
- Always try to include mechanical assist (counter-weights, counter-springs) wherever possible.
- Some motors are stronger in one direction than the other. A directional scalar is can be used to compensate for this.
- Be aware that it is easy to overheat motors during calibration sessions and extended practices if you use this approach. Be careful.
- You can expand this approach to control multiple axis arms. Calculate the horizontal position of the CG from the angle of each element and back calculate the effective system angle.

Creating Accurate Time Delays

- Making an accurate time delay in a FIRST Robot Controller takes some understanding of how the controller works.
- Using a simple For..Next like in the Basic Stamp Manual will not work. The Master Controller must receive slave data in the form of a Serout command roughly every 150ms or less or the system will reset.
- Simply counting loops works but is inaccurate. The system loop time is not fixed, but is a function of code length and can vary greatly.
- A solution is to use the Delta_t data, available in the Serin data.
- Delta_t tells you how many packets the user processor missed during the execution of the user code. Thus if Delta_t = 0, T = 25 ms, if Delta_t = 1, T = 50 ms, and so on.
- Using this method accurate time delays can be managed with no risk of crashing the system.

Do Timer_demo here

- File Timer_demo1.bsx
- 2 second cyclic with asynchronous 10 sec
  - uses scratch pad and tiered aliasing

Questions & Answers

- Please begin with related topics but I am happy to discuss any Robotics topics if time permits.

Good luck to all teams in 2003