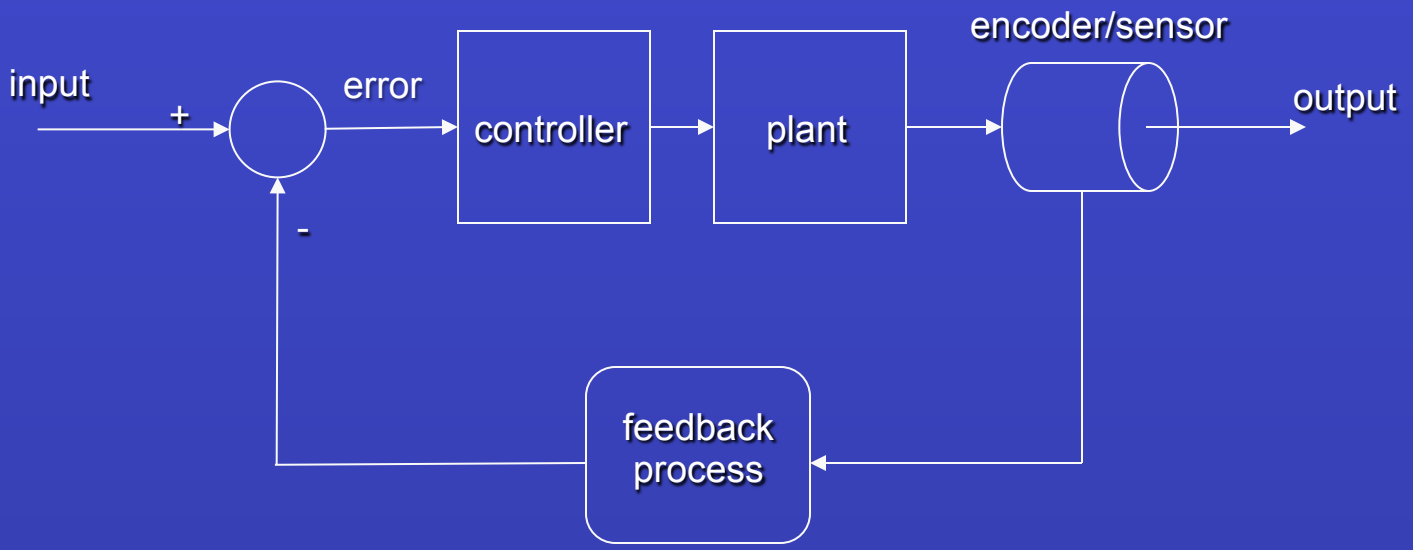
Control Using Simple Feedback

S&C - 1

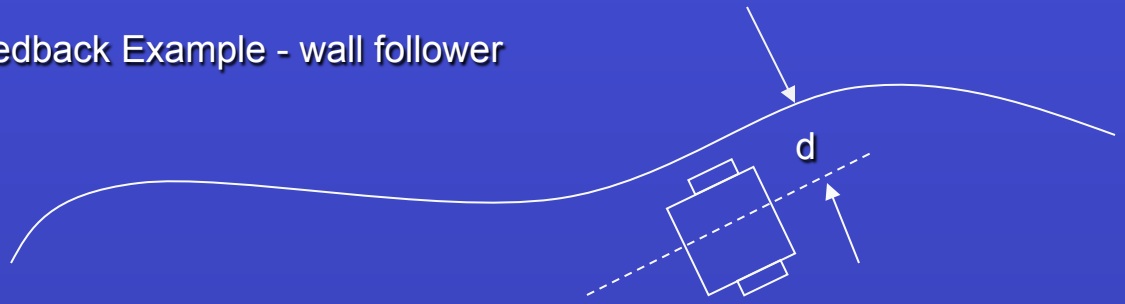
General model for a feedback system



McGill

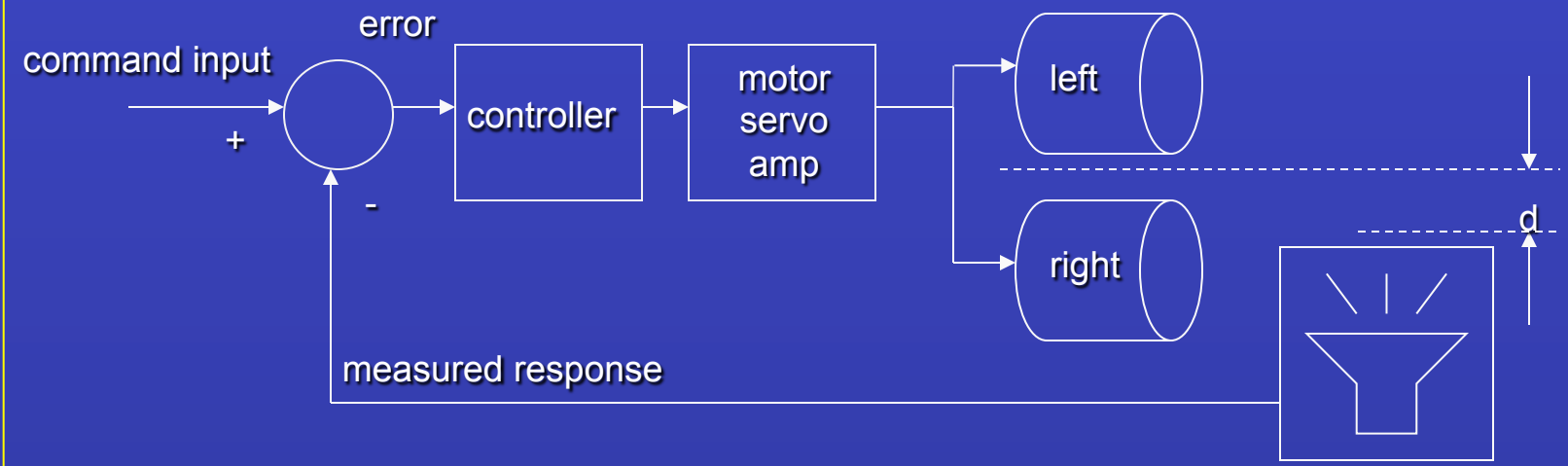
S&C - 2

Simple Feedback Example - wall follower



set up

Systems View



- Difference in motor speeds controls lateral distance - d

ultrasonic sensor

McGill

S&C - 3

Designing the Controller

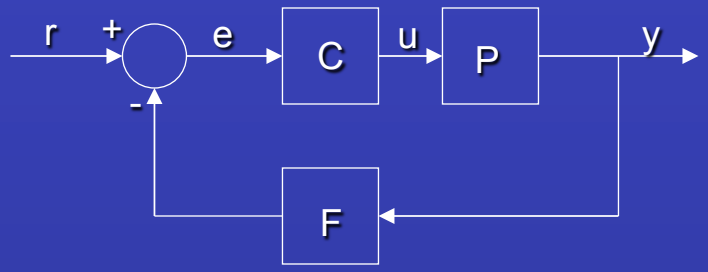
- An intuitive view (you will study formal methods in ECSE 404, Control Systems)

Known Entities

- The plant model can be measured and/or suitably approximated
- The range of inputs
- The desired response of the system to the inputs

Design Problem

- Figure out appropriate controller and feedback models to achieve the desired response



if CP large, then y ≈ r / F

$$y = CPe$$
$$e = r - Fy$$
$$y = CP(r - Fy) = CPr - CPFy$$
$$y(1 + FCP) = CPr$$
$$y = \frac{CP}{(1 + FCP)} r$$

S&C - 4

Designing the Controller cont.

- C, P, and F can be arbitrarily complex functions
- Key aspects of their design include
  - Stability, the output must follow the input in a predictable manner
  - Fidelity, the output must be sufficiently responsive as a function of time
  - Control Theory deals with these and many other aspects of the design
- For the simple case considered here, the output is directly proportional to the input

Designing the Wall Follower

r = set offset distance in cm

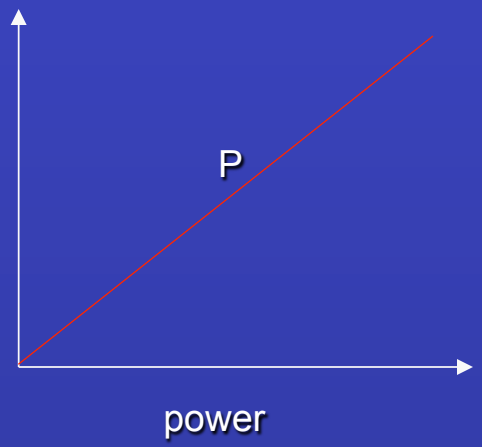
y = measured offset distance in cm

P = constant $\propto$ speed (deg/sec)

F = constant = 1 (error in cm)

What is C?

$d = 2\pi v_\theta t$

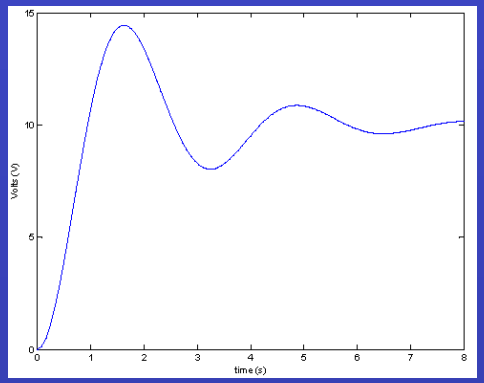


McGill

S&C - 5

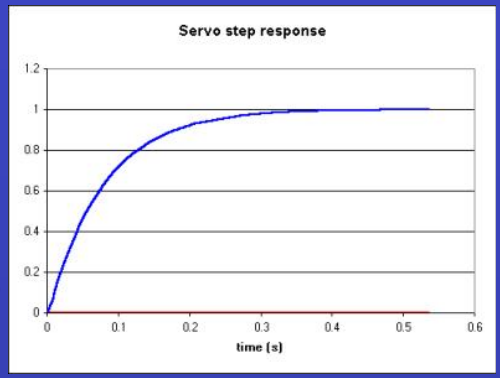
Designing the Controller cont.

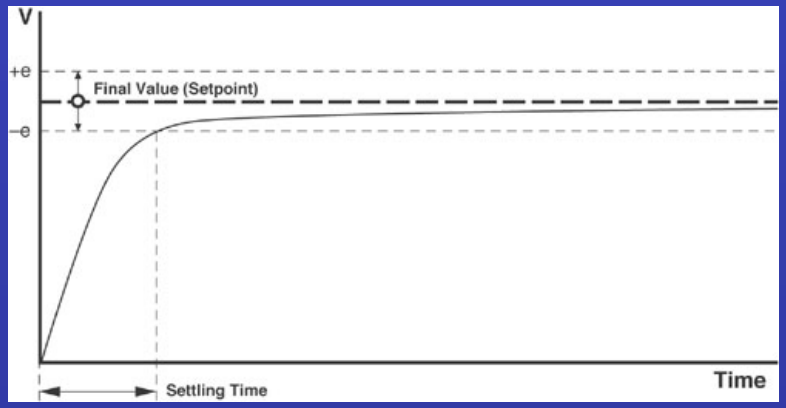
- C is a *gain* constant proportional to the error (r - y).
- Too large a value of C results in an *underdamped* system
- Too small a value of C results in an *overdamped* system
- A correct choice of C results in a *critically damped* system



under

over





critical

Designing the Controller cont.

- Knowing the analytical forms of P and F, one can determine the proportional *gain*, C, that results in an appropriate response.
- More often than not in simple designs such as this one, C can be adjusted by trial and error (so-called tweaking).
- An even more simplied approach is possible, so-called BANG BANG Control.

The BANG BANG Controller

- Consider a digital control loop that periodically samples the output at a rate S.
- At each sample interval, compute error, e, and apply a correction in the appropriate direction (+/-) using a fixed step.
- If the correction is applied often enough, the output will eventually catch up and follow the input as desired.
- At convergence, the controller will “hunt” about the zero error point applying successive +/- corrections.
- We eliminate hunting by suppressing any correction if | r – y | is less than threshold.

S&C - 7

Back to the Wall Follower

Steering the cart:

- Error = reference control value - measured distance from the wall.
- If abs(Error) < Threshold, we consider the cart to be on a correct heading.
- if Error < 0, the cart is too far from the wall. Increase rotation of outside wheel; (decrease rotation of inside wheel).
- If Error > 0, the cart is too close to the wall. Decrease rotation of outside wheel; (increase rotation of inside wheel).
- Magnitude of change in rotation is proportional to the magnitude of the error.
- In our example, we follow the BANG BANG approach and define 2 speeds:

  FWDSPEED – speed at which left and right wheel rotate to go straight.

  DELTASPD – amount by which speed increased/decreased to effect motion towards/away from wall.

- A more elaborate approach would scale the correction, i.e., DELTASPD according to Error (this is called proportional control).


```
package wf1EV3;

import lejos.hardware.Button;                    // All references to classes in your code
import lejos.hardware.ev3.LocalEV3;              // need to be included here.  Eclipse
import lejos.hardware.lcd.TextLCD;               // can do this for you.
import lejos.hardware.motor.Motor;
import lejos.hardware.port.Port;
import lejos.hardware.sensor.EV3TouchSensor;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.hardware.sensor.SensorModes;
import lejos.robotics.RegulatedMotor;
import lejos.robotics.SampleProvider;

// A direct, procedural implementation of the simple wall follower.
// Lab 1 introduces a more object-oriented approach.
//

public class SimpleWF {

// Class Constants

        public static final int WALLDIST = 30;       // Standoff distance to wall
        public static final int DEADBAND = 2;        // Error threshold
        public static final int FWDSPEED = 200;   // Default rotational speed of wheels
        public static final int DELTASPD = 100;     // Bang-bang constant
```


```
// Class Variables

        public static int wallDist=0;        // Measured distance to wall
        public static int distError=0;       // Error

// Objects instantiated once in this class

        static TextLCD t = LocalEV3.get().getTextLCD();
        static RegulatedMotor leftMotor = Motor.A;
        static RegulatedMotor rightMotor = Motor.D;

// Sensor set-up

// 1. Allocate a port for each sensor

        static Port portUS = LocalEV3.get().getPort("S1");
        static Port portTouch = LocalEV3.get().getPort("S2");

// 2. Create an instance for each sensor

        static SensorModes myUS = new EV3UltrasonicSensor(portUS);
        static SensorModes myTouch = new EV3TouchSensor(portTouch);
```

```
/ 3. Each sensor needs a sample provided which actually fetches the data

        static SampleProvider myDistance = myUS.getMode("Distance");
        static SampleProvider myTouchStatus = myTouch.getMode(0);

// 4. Sensors return real-valued data; need to allocate buffers for each

        static float[] sampleUS = new float[myDistance.sampleSize()];
        static float[] sampleTouch = new float[myTouchStatus.sampleSize()];

//
//   Main entry point - set display, start motors, enter polling loop.
//   (this is a very inefficient way to do things)
//

        public static void main(String[] args) {

                t.clear();                              // Clear display
                t.drawString("Simple Wall F", 0, 0);    // Print banner
                t.drawString("Distance: ", 0, 1);

                leftMotor.setSpeed(FWDSPEED);     // Start moving forward
                rightMotor.setSpeed(FWDSPEED);
                leftMotor.forward();
                rightMotor.forward();
```

```
//
//   Main control loop: read distance, determine error, adjust speed, and repeat

                        boolean traveling=true;
                        int status=0;

// Check for stop command or collisions

                        while(traveling){
                          status=Button.readButtons();                          // Check for abort
                          myTouchStatus.fetchSample(sampleTouch, 0);    // Check for collision
                         if ((status==Button.ID_ALL)||(sampleTouch[0]==1))  // Abort if keypad pressed
                                  System.exit(0);                                // or touch sensor
                                                                                 // tripped.
// Get sensor reading and update display

                          myDistance.fetchSample(sampleUS, 0);     // Get latest reading
                          wallDist=(int)(sampleUS[0]*100.0);            // Scale to integer
                          t.drawInt(wallDist,4,11,1);                    // Print current
                                                                                 // sensor reading
```


```
// Controller
                        distError=WALLDIST-wallDist;       // Compute error

                        if (Math.abs(distError) <= DEADBAND) {        // Within limits, same speed
                                leftMotor.setSpeed(FWDSPEED);  // Start moving forward
                                rightMotor.setSpeed(FWDSPEED);
                                leftMotor.forward();
                                rightMotor.forward();
                                }

                        else if (distError > 0) {                              // Too close to the wall
                                leftMotor.setSpeed(FWDSPEED);
                                rightMotor.setSpeed(FWDSPEED-DELTASPD);
                                leftMotor.forward();
                                rightMotor.forward();
                                }

                        else if (distError < 0) {
                                leftMotor.setSpeed(FWDSPEED-DELTASPD);
                                rightMotor.setSpeed(FWDSPEED);
                                leftMotor.forward();
                                rightMotor.forward();
                                }
                        }
                }
}
```

Improving the Simple Wall Follower

1. Control both wheels, i.e., operate wheels differentially:

```
if (Math.abs(error) <= DEADBAND) {          /* Within tolerance */
leftMotor.setSpeed(FWDSPEED);               /* 0 bias           */
rightMotor.setSpeed(FWDSPEED);
}
else if (error > 0) {                       /* Too close        */
leftMotor.setSpeed(FWDSPEED+DELTA);          /* Speed up inner wheel */
rightMotor.setSpeed(FWDSPEED-DELTA);        /* Slow outer wheel */
}
else {                                      /* Too far          */
leftMotor.setSpeed(FWDSPEED-DELTA);         /* Exactly opposite to above  */
rightMotor.setSpeed(FWDSPEED+DELTA);
}
```

n.b. the leftMotor.forward() and rightMotor.forward() methods are not shown for brevity.

2. Use a timer to control servo updates (also use touch to detect collisions)

```
package myPackage;

import lejos.hardware.Button;
import lejos.hardware.ev3.LocalEV3;
import lejos.hardware.lcd.TextLCD;
import lejos.hardware.motor.Motor;
import lejos.hardware.port.Port;
import lejos.hardware.sensor.EV3TouchSensor;
import lejos.hardware.sensor.EV3UltrasonicSensor;
import lejos.hardware.sensor.SensorModes;
import lejos.robotics.RegulatedMotor;
import lejos.robotics.SampleProvider;
import lejos.utility.Timer;
import lejos.utility.TimerListener;

//
// This is the Timed Wall Follower from the NXT updated to run on EV3 hardware.
// Differences between the two implementations are indicated in the comments.
//
```


```
// Class Constants

        public static final int SINTERVAL=100;          // A 10Hz sampling rate
        public static final double PROPCONST=1.0;   // Proportionality constant
        public static final int WALLDIST=30;            // Distance to wall * 1.4 (cm)
        public static final int FWDSPEED=100;           // Forward speed (deg/sec)
        public static final int MAXCORRECTION=50;  // Bound on correction to prevent stalling
        public static final long SLEEPINT=500;          // Display update 2Hz
        public static final int ERRORTOL=1;             // Error tolerance (cm)
        public static final int MAXDIST=200;            // Max value of valid distance
        public static final int ID_ESCAPE=32;          // Value returned when ESCAPE key pressed

// Class Variables
        public static int wallDist=0;                   // Measured distance to wall
        public static int distError=0;                  // Error
        public static int leftSpeed=FWDSPEED;           // Vehicle speed
        public static int rightSpeed=FWDSPEED;

// Objects instanced once by this class

        static TextLCD t = LocalEV3.get().getTextLCD();
        static RegulatedMotor leftMotor = Motor.A;
        static RegulatedMotor rightMotor = Motor.D;
```

Using a timer cont.

```
// leJOS EV3 uses a new scheme for doing sensor I/O

// 1. Get a port instance for each sensor used

        static Port portUS = LocalEV3.get().getPort("S1");
        static Port portTouch = LocalEV3.get().getPort("S2");

// 2. Get an instance for each sensor

        static SensorModes myUS = new EV3UltrasonicSensor(portUS);
        static SensorModes myTouch = new EV3TouchSensor(portTouch);

// 3. Get an instance of each sensor in measurement mode

        static SampleProvider myDistance = myUS.getMode("Distance");
        static SampleProvider myTouchStatus = myTouch.getMode(0);

// 4. Allocate buffers for data return

        static float[] sampleUS = new float[myDistance.sampleSize()];
        static float[] sampleTouch = new float[myTouchStatus.sampleSize()];
```

Using a timer cont.

```
public static void main(String[] args) throws InterruptedException {
        boolean noexit;
        int status;

// Set up the display area
        t.clear();
        t.drawString("Wall Follower", 0, 0);
        t.drawString("Distance:", 0, 4);
        t.drawString("L Speed:", 0, 5);
        t.drawString("R Speed:", 0, 6);
        t.drawString("Error:", 0, 7);

// Set up timer interrupts
        Timer myTimer = new Timer(SINTERVAL, new TimedWF());

// Start the cart rolling forward at nominal speed
        leftMotor.setSpeed(leftSpeed);
        rightMotor.setSpeed(rightSpeed);
        leftMotor.forward();
        rightMotor.forward();
        distError=0;
```

Using a timer cont.

```
// Enable the exception handler
                        myTimer.start();

// There are two threads in operation, Main and the timer exception
// handler.  Main continuously updates the display and checks for
// an abort from the user.
                        noexit=true;

                        while(noexit) {
                          status=Button.readButtons();        // Check for press on console
                          myTouchStatus.fetchSample(sampleTouch,0);
                          if ((status==32)||(sampleTouch[0]==1)) {
                                  System.exit(0);
                          }
// Update status on LCD
                        t.drawInt(wallDist,5,11,4);             // Display key parameters on LCD
                        t.drawInt(leftSpeed,4,11,5);
                        t.drawInt(rightSpeed,4,11,6);
                        t.drawInt(distError,4,11,7);

                       Thread.sleep(SLEEPINT);
                                                                // Have a short nap
                        }
                }
```


```
//
// The servo (control) loop is implemented in the timer handler (listener).  Version 0.90 of
// leJOS EV3 has a bug in the servo code.  A motion command has to follow setSpeed in
// order for the new set point to register.  Hopefully this will get fixed in later versions.
//
                public void timedOut() {
                  int diff;
                  myDistance.fetchSample(sampleUS,0);       // Read latest sample in buffer
                  wallDist=(int)(sampleUS[0]*100.0);        // Convert from MKS to CGS; truncate to int
                  if (wallDist <= MAXDIST)
                    distError = WALLDIST-wallDist;          // Compute error term

// Controller Actions

                  if (Math.abs(distError) <= ERRORTOL) {    // Case 1: Error in bounds, no correction
                    leftSpeed=FWDSPEED;
                    rightSpeed=FWDSPEED;
                    leftMotor.setSpeed(leftSpeed);          // If correction was being applied on last
                    rightMotor.setSpeed(rightSpeed);        // update, clear it
                    leftMotor.forward();                    // Hack - leJOS bug
                    rightMotor.forward();
                  }
```


```
    else if (distError > 0) {                  // Case 2: positive error, move away from wall
      diff=calcProp(distError);                // Get correction value and apply
      leftSpeed=FWDSPEED+diff;
      rightSpeed=FWDSPEED-diff;
      leftMotor.setSpeed(leftSpeed);
      rightMotor.setSpeed(rightSpeed);
      leftMotor.forward();                     // Hack - leJOS bug
      rightMotor.forward();
    }

    else if (distError < 0) {                  // Case 3: negative error, move towards wall
      diff=calcProp(distError);                // Get correction value and apply
      leftSpeed=FWDSPEED-diff;
      rightSpeed=FWDSPEED+diff;
      leftMotor.setSpeed(leftSpeed);
      rightMotor.setSpeed(rightSpeed);
      leftMotor.forward();                     // Hack - leJOS bug
      rightMotor.forward();
    }
}
```

```
//
// This method is used to implement your particular control law.  The default
// here is to alter motor speed by an amount proportional to the error.  There
// is some clamping to stay within speed limits.


                int calcProp (int diff) {

                  int correction;

// PROPORTIONAL:  Correction is proportional to magnitude of error

                  if (diff < 0) diff=-diff;
                  correction = (int)(PROPCONST *(double)diff);
                  if (correction >= FWDSPEED) correction = MAXCORRECTION;
                  return correction;
                }
}
```