

Controls and Signals

Maslab IAP 2011

Ellen Yi Chen

yichen@mit.edu

Agenda

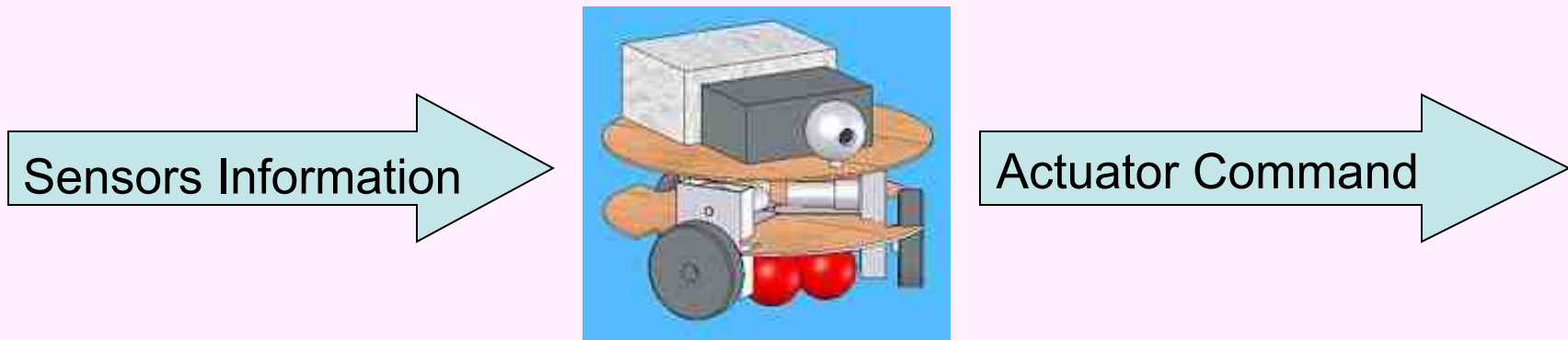
- What do we mean by controls?
- Simple PID Controller
- Robot Drive Controller
- Examples
- Kalman, ALS, Filters
- Extensions

What are Controls?

- “High” Level Control Paradigms
 - Model/Plan/Act
 - Emergent
 - FSM (Finite State Machine)
- “Low” Level Control Loops
 - Motor Velocity
 - Robot Angular Position
 - Etc...

Why can't we just tell the robot to go at 0.2m/s in a straight line?

What are Controls?



Sensors are far from perfect

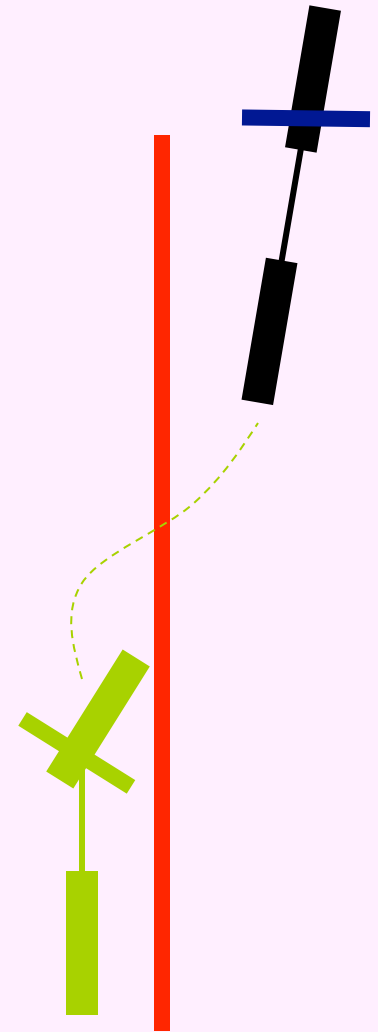
- Camera white balance
- Encoder quantization error
- Ultrasound reflections
- Infrared sensors noisy
- Etc...

Actuators are far from perfect

- Motor velocity changes with time/ terrain/torque
- Wheels/gears slip
- Servos get stuck
- Etc...

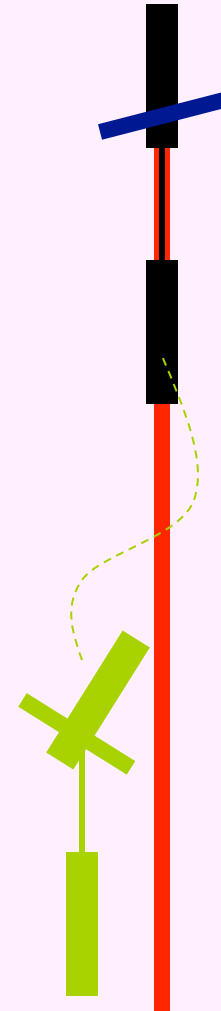
Example: Bike in straight line

- Steer the bike in a straight line blindfolded
- Open loop \rightarrow no sensor feedback
- What if you hit a rock?
- What if the handle bars aren't perpendicular to the wheels?



Example: Bike in a straight line

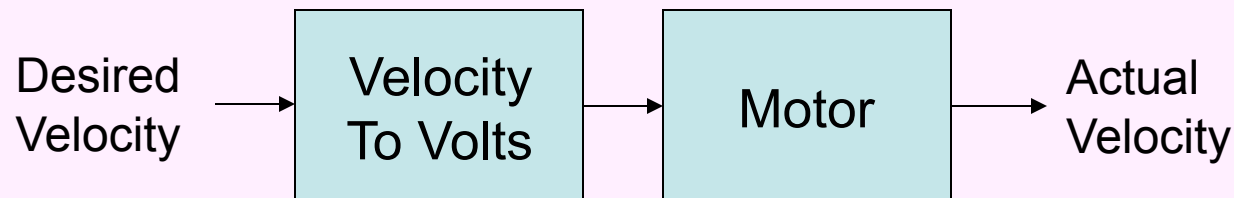
- If you can see the pavement → Closed Loop Approach
- Control based on error: PID
 - **Proportional** : Change handle angle proportional to the current error
 - **Derivative** : Large handle corrections when error is changing slowly, and small handle corrections when error is changing quickly
 - **Integral** : Handle corrections based on the cumulative error



Problem: Set Motor Velocity

Open Loop Controller

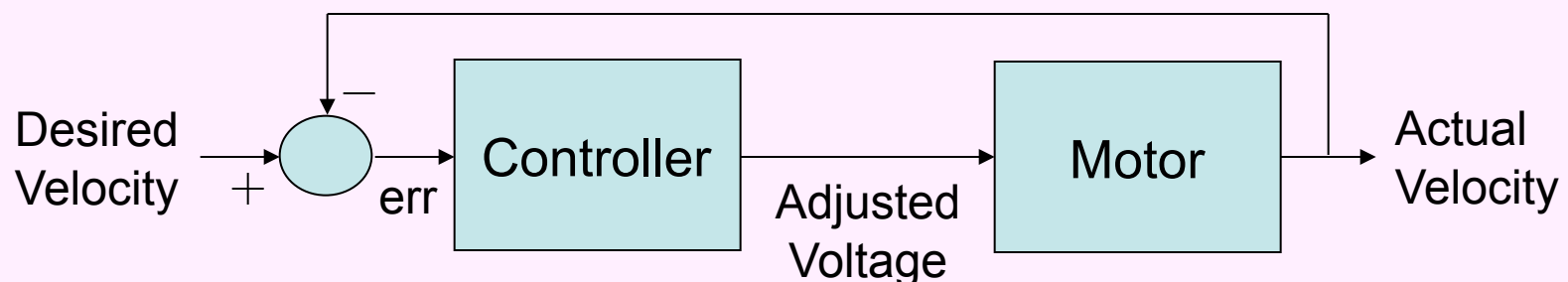
- Use trial and error to create relationship between velocity and voltage
- Problems
 - Supply voltage change
 - Bumps in carpet
 - Motor Transients



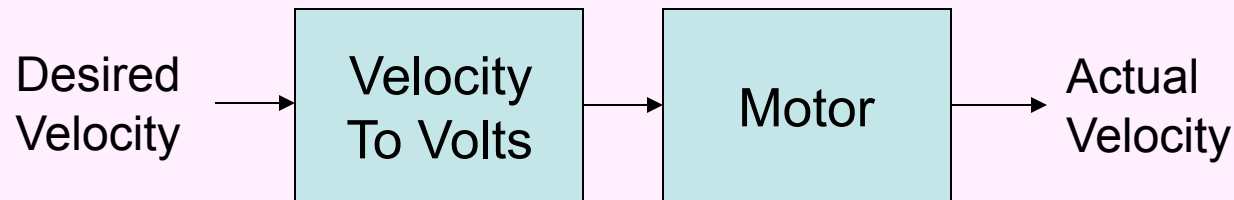
Problem: Set Motor Velocity

Closed Loop Controller

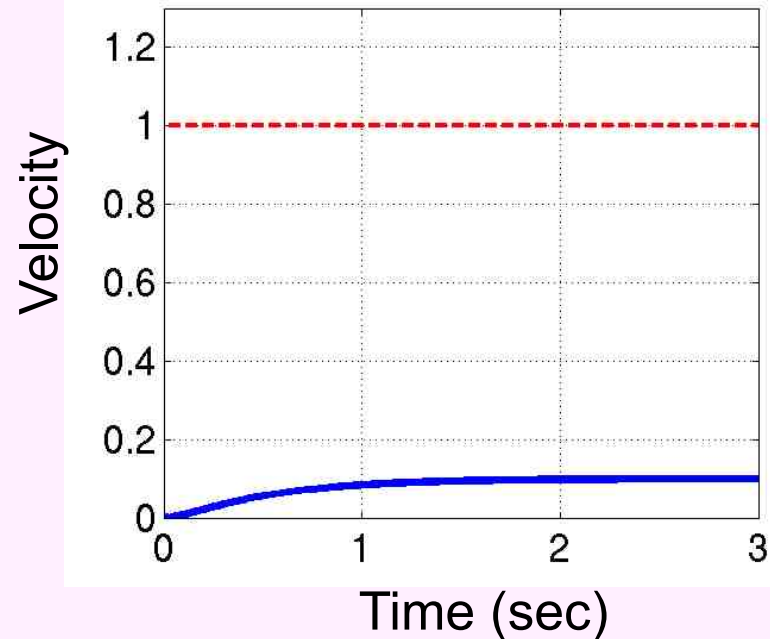
- Feedback is used so that the actual velocity equals the desired velocity
- Can use an optical encoder to measure actual velocity



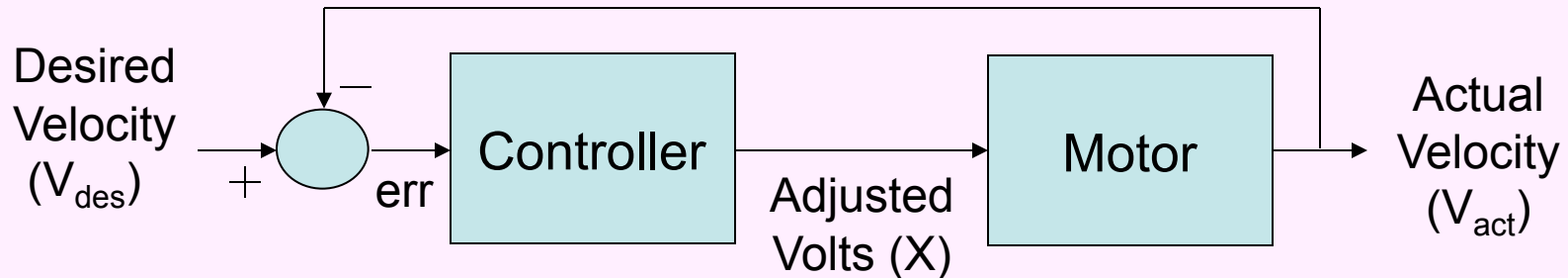
Step response with **no controller**



- Naive velocity to volts
- Model motor with several differential equations
- Slow rise time
- Stead-state offset

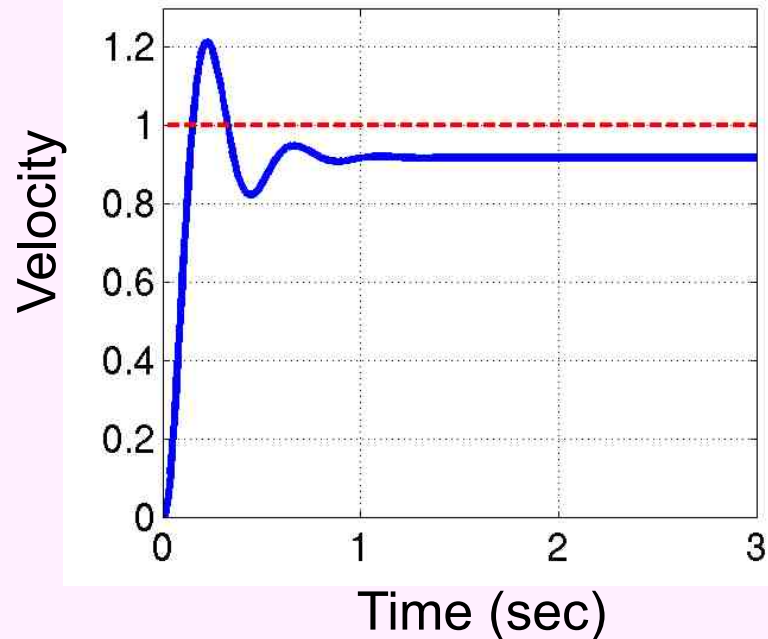


Step response with proportional controller

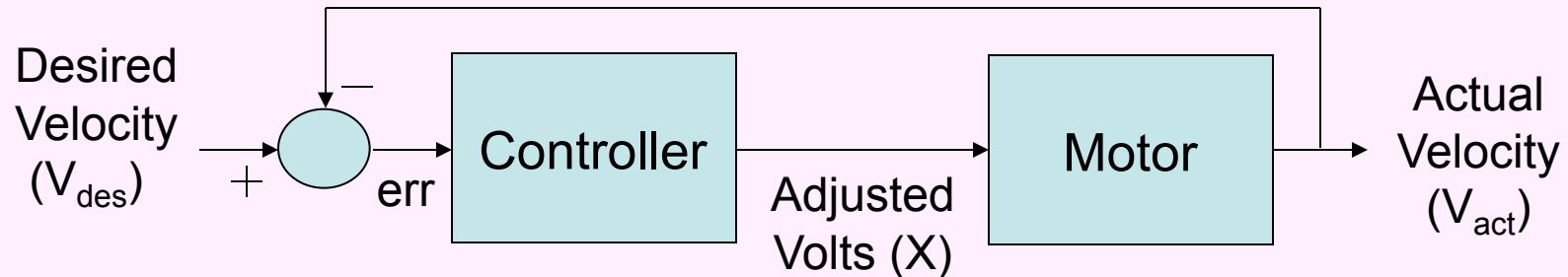


$$X = V_{des} + K_P \cdot (V_{des} - V_{act})$$

- Big error big = big adj
- Faster rise time
- Overshoot
- Steady-state offset
(there is still an error
but it is not changing!)

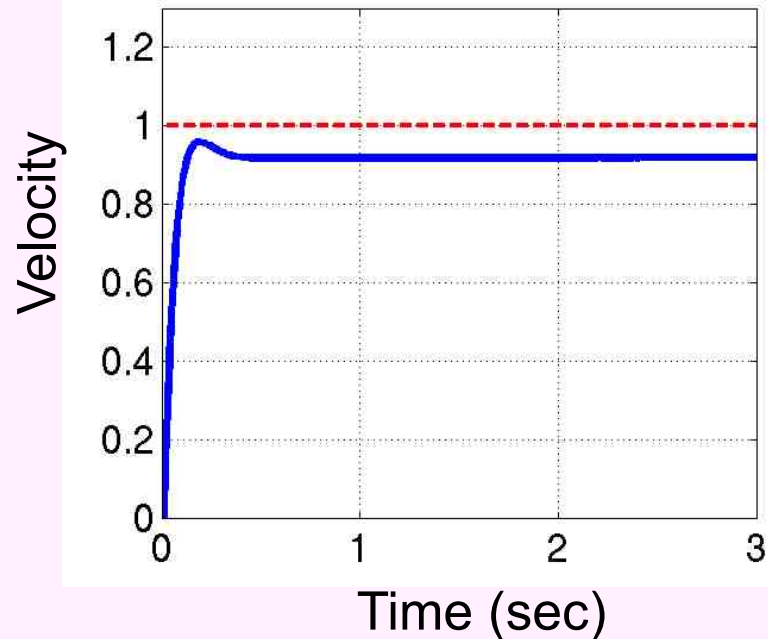


Step response with **PD** controller

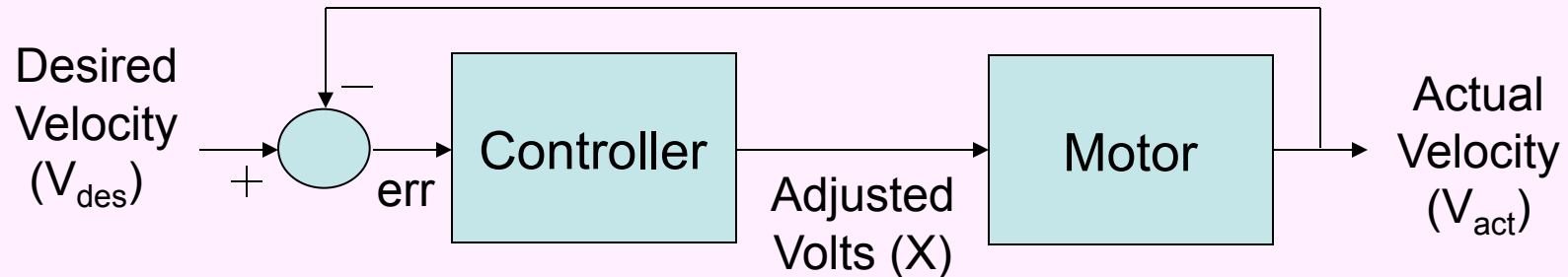


$$X = V_{des} + K_P e(t) - K_D \frac{de(t)}{dt}$$

- When approaching desired velocity quickly, de/dt term counteracts proportional term slowing adjustment
- Faster rise time
- Reduces overshoot

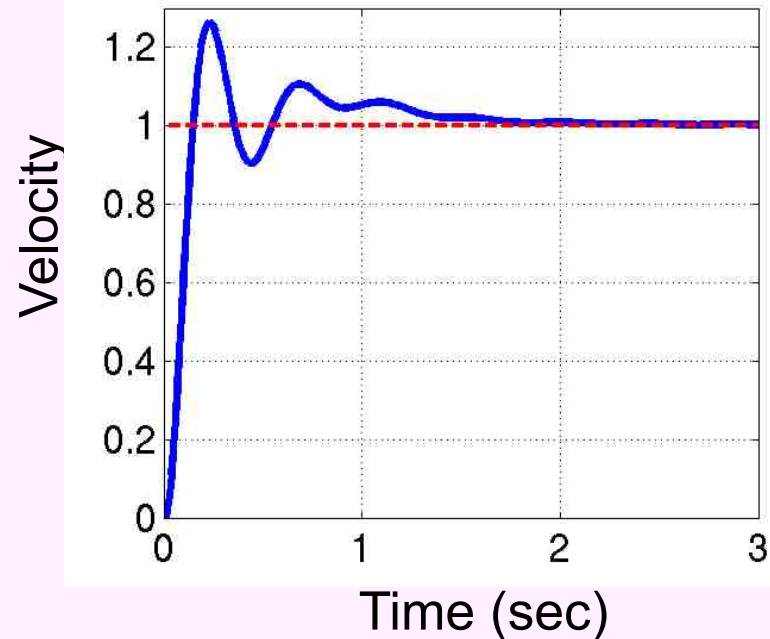


Step response with **PI** controller

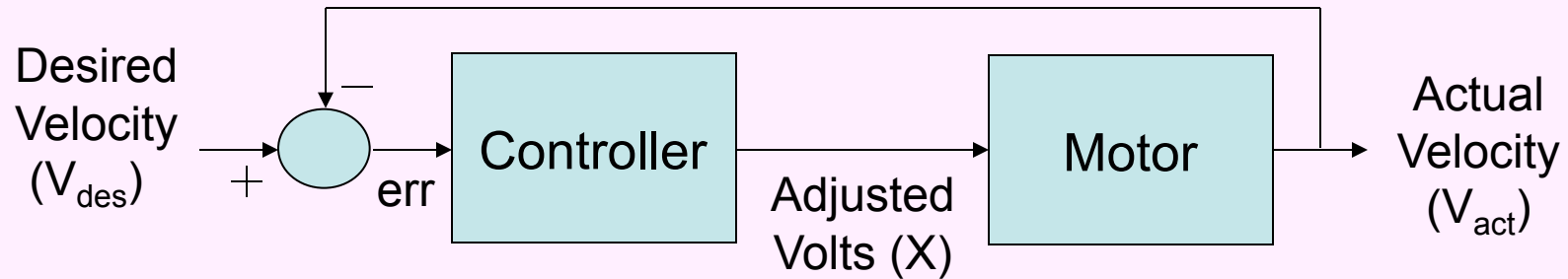


$$X = V_{des} + K_P e(t) - K_I \int e(t) dt$$

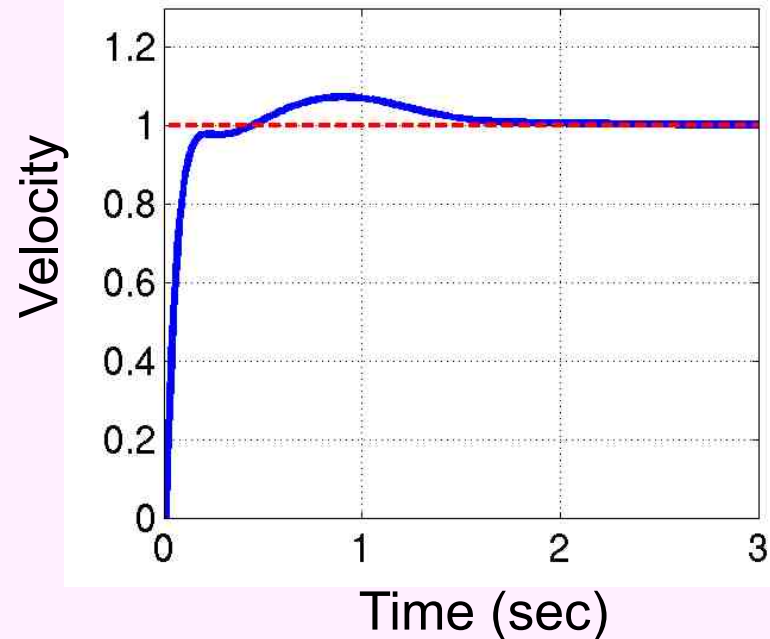
- Integral term eliminates accumulated error
- Increases overshoot



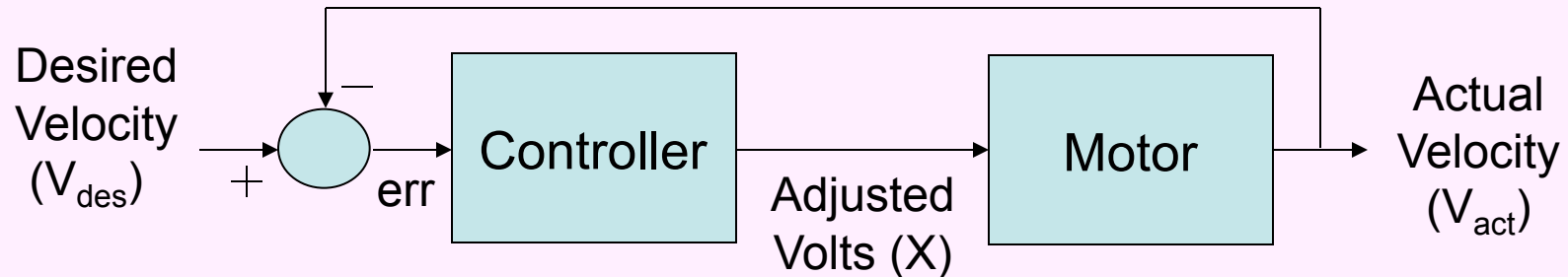
Step response with **PID** controller



$$\begin{aligned} X = & V_{des} + K_P e(t) \\ & + K_I \int e(t) dt \\ & - K_D \frac{de(t)}{dt} \end{aligned}$$



Choosing and tuning a controller

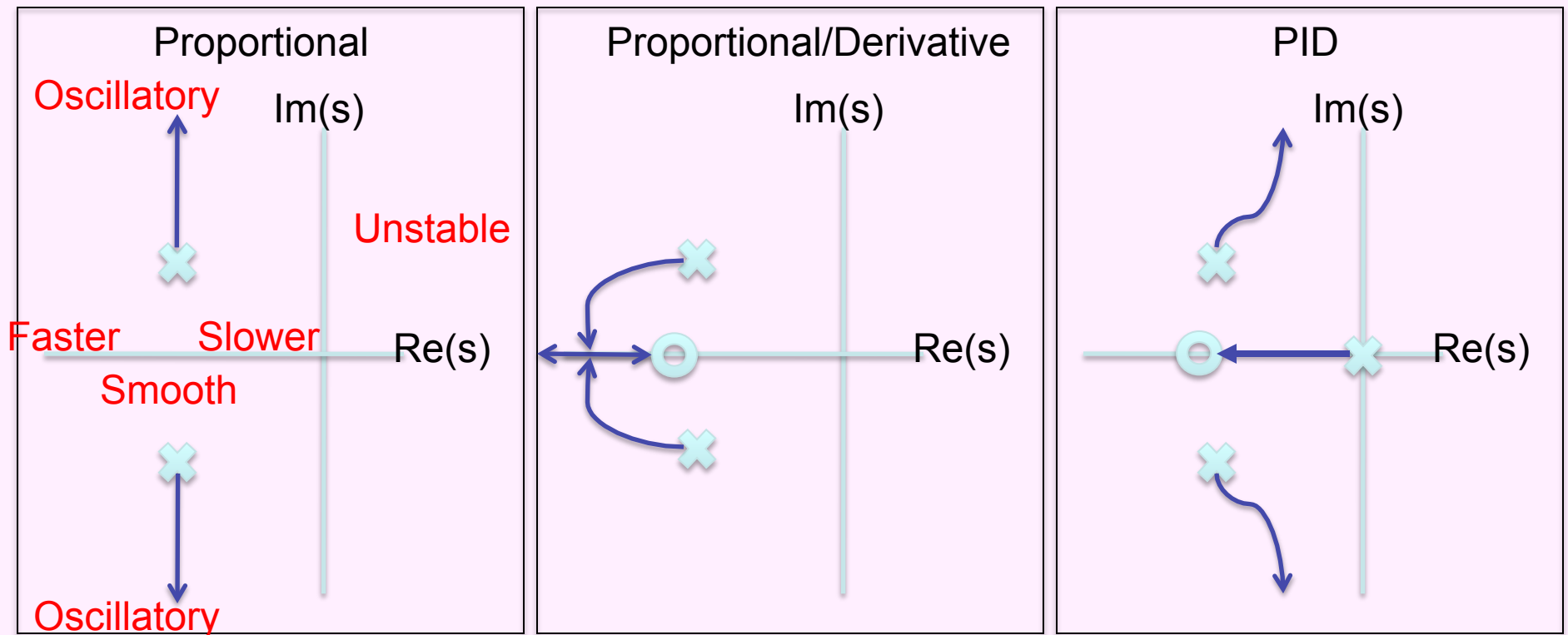


	Rise Time	Overshoot	SS Error
Proportional	Decrease	Increase	Decrease
Integral	Decrease	Increase	Eliminate
Derivative	~	Decrease	~

Controller Design: Root Locus

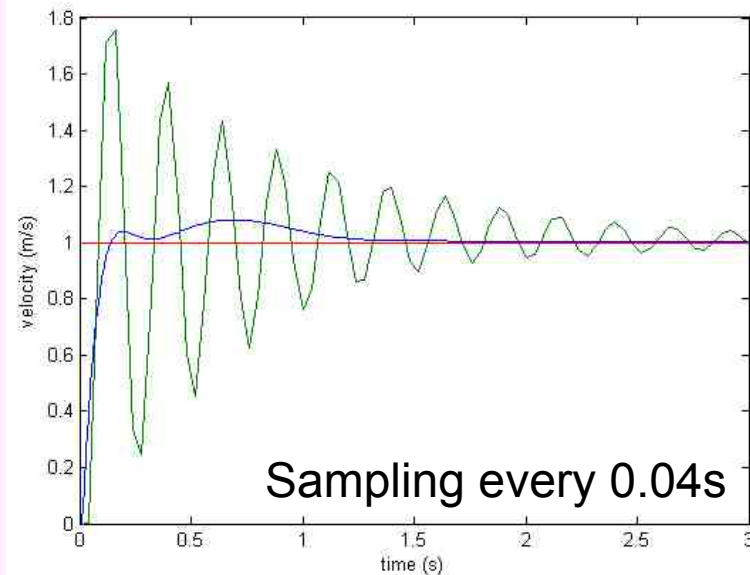
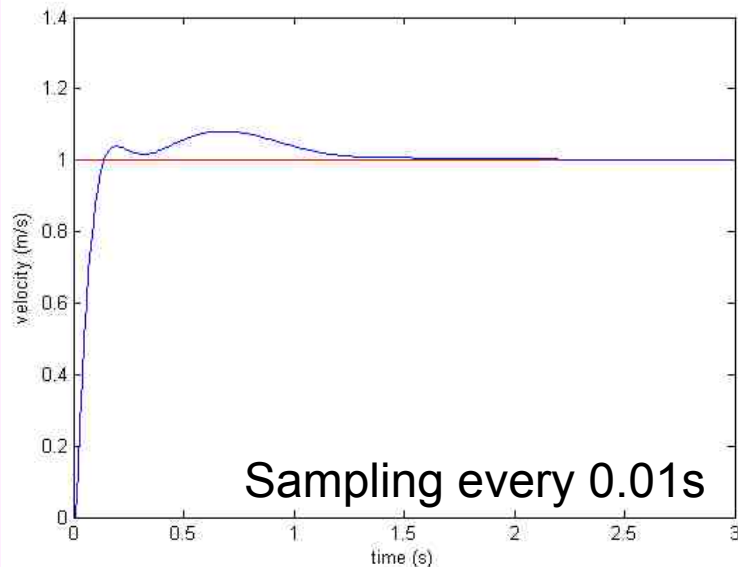
Current controlled motor

$$J \frac{d^2 x}{dt^2} + B \frac{dx}{dt} + Kx = \tau(t) = Ci(t)$$



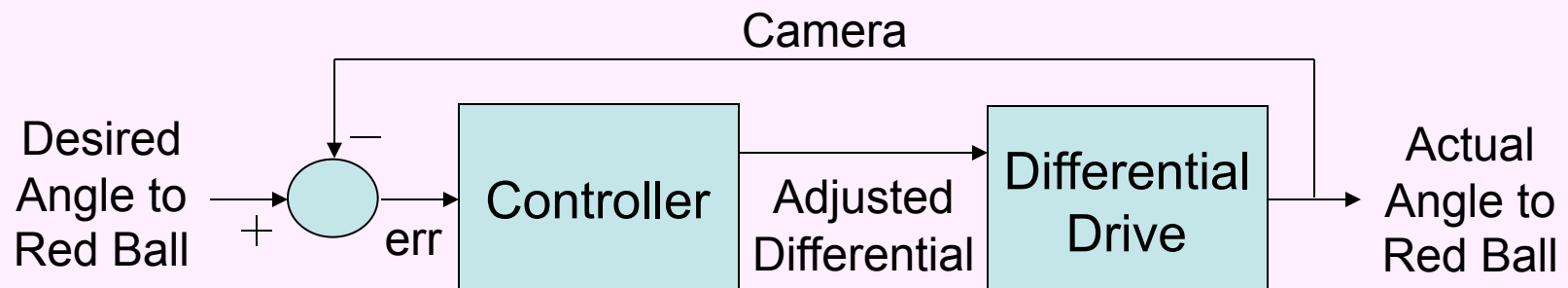
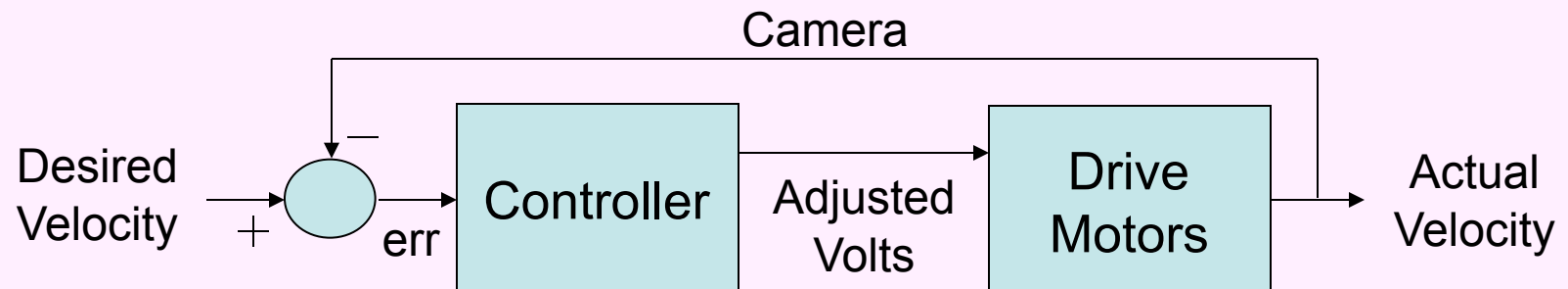
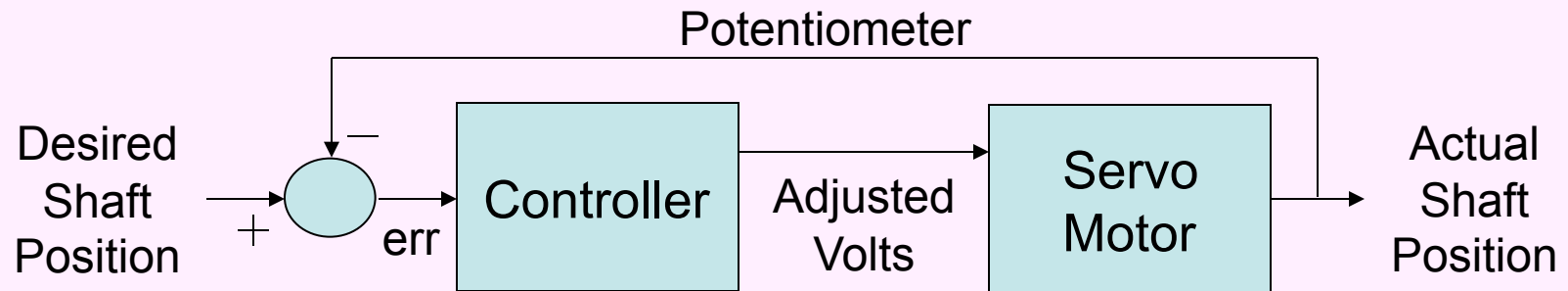
Sampling Time, Noise, Limits

- When you learn PID, you learn it in continuous models
- For the discrete world, sampling time is another variable!
- Say you tune your PID and you sample every 0.01 seconds
- Then you write more code, add more threads
- At the end, you sample every 0.04 seconds. This affects your system and you may have to retune your PID!
- Take aways:
 - Set a constant sampling time* and stick with it!
 - Controller unstable due to noise? Low pass filter signal before controlling!
 - Response speed is limited by slew rate and max output of electronics



*Nyquist dictates 2x, but in practice at least 5x greater than fastest characteristic

Other Control Loop Uses

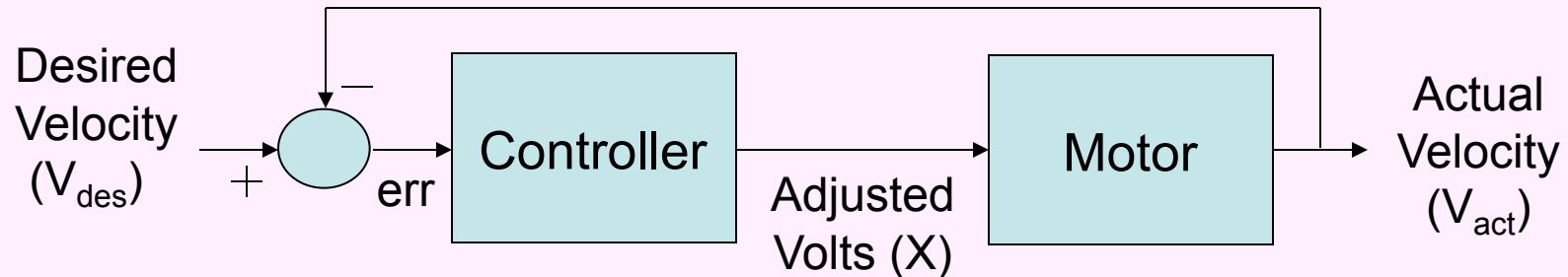


Matlab Examples

- `motorConstructor` → Create a basic motor structure
- `motorSetVoltage` → Set the motor voltage
- `motorStepResponse` → Find unit step response for a motor
- `motorPID` → Find unit step response for a motor with PID
- `robotPID` → differential drive robot with two independent PID loops
- `plotRobotTrajectory` → plot the trajectory of robotPID

*Thanks to Christopher Batten for the code

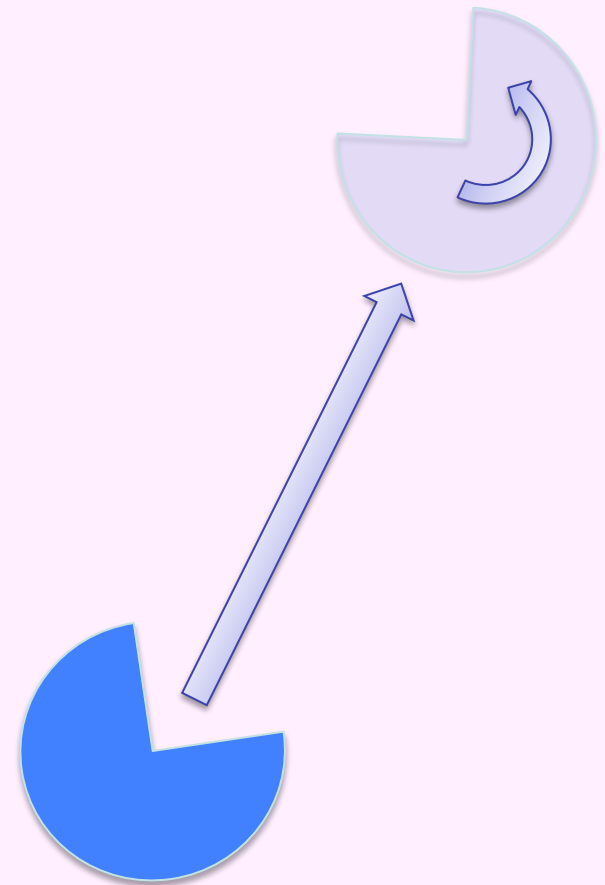
Choosing and tuning a controller



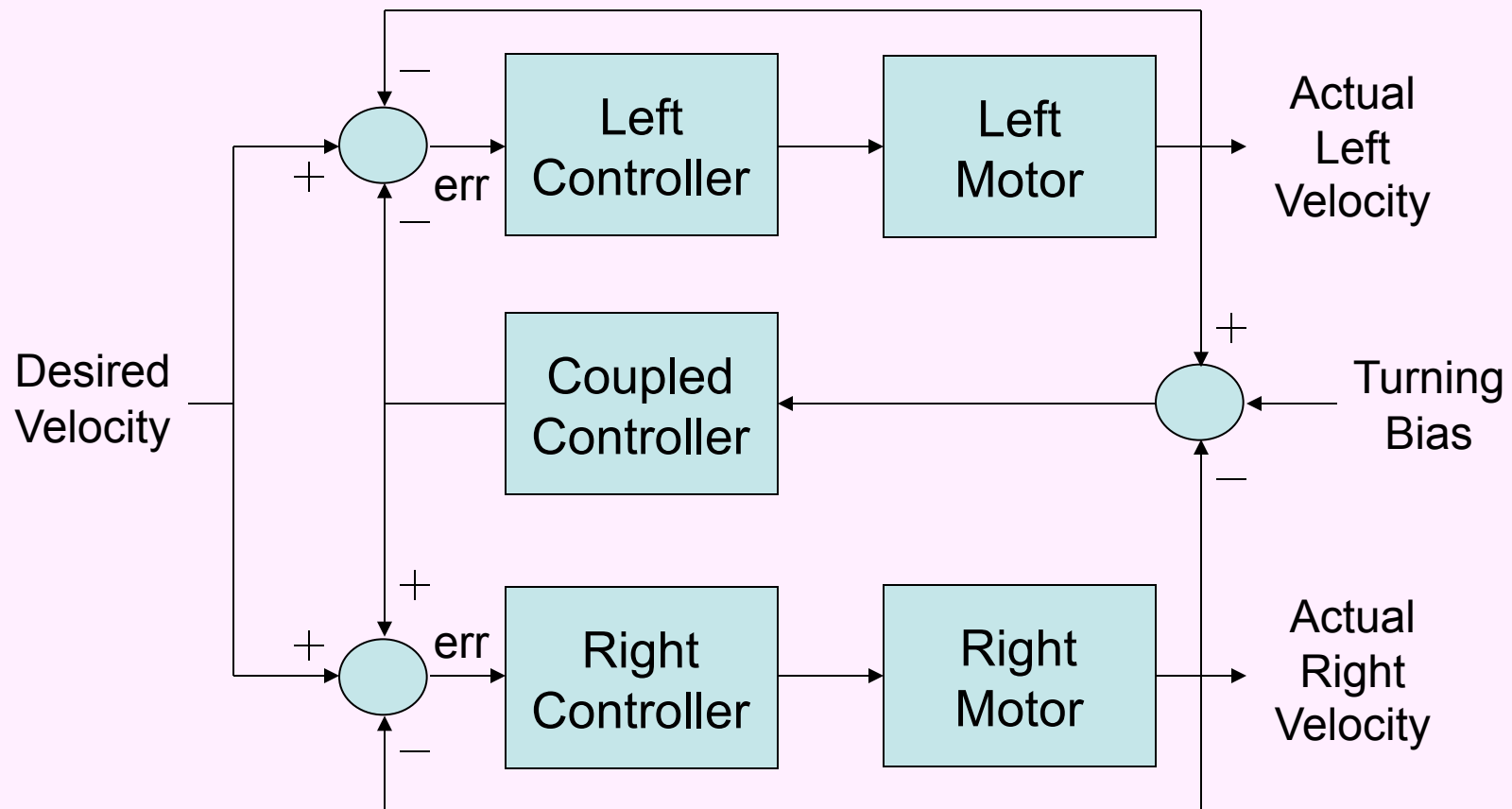
- Set constant sampling time
- Tuning PID constants can be tricky
 - Use control system theory as a guide!
 - Guess system parameters and simulate.
- Use gain scheduling for nonlinearities
 - Use different PID constants for different situations.
- Make PID parameters tunable without reuploading code
 - Use an interactive tuning program.
 - Once decided, then hard code constants in.

MIMO Systems

- Multiple Input (gyro and two encoders) / Multiple Output (two motors)
- Want to control displacement and rotation
- Method 1 (easiest method)
 - 1. **Decouple the system**
 - 2. Build linear single input / single output controllers around each decoupled parameter.
 - 3. Execute displacement
 - 4. Execute rotation (executing simultaneously could be buggier)
- Easy method for driving straight
 - Set a moderate speed for one wheel
 - Have PID running on the other wheel
 - Use the gyroscope to drive straight.



We can synchronize the motors with a third PID controller



We can synchronize the motors with a third PID controller

What should the coupled controller use as its error input?

Velocity Differential

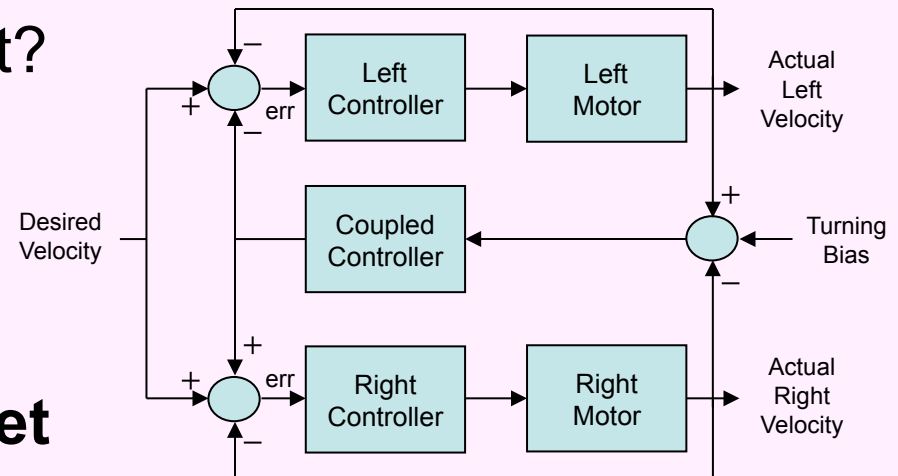
- Will simply help the robot go straight but not necessarily straight ahead

Cumulative Centerline Offset

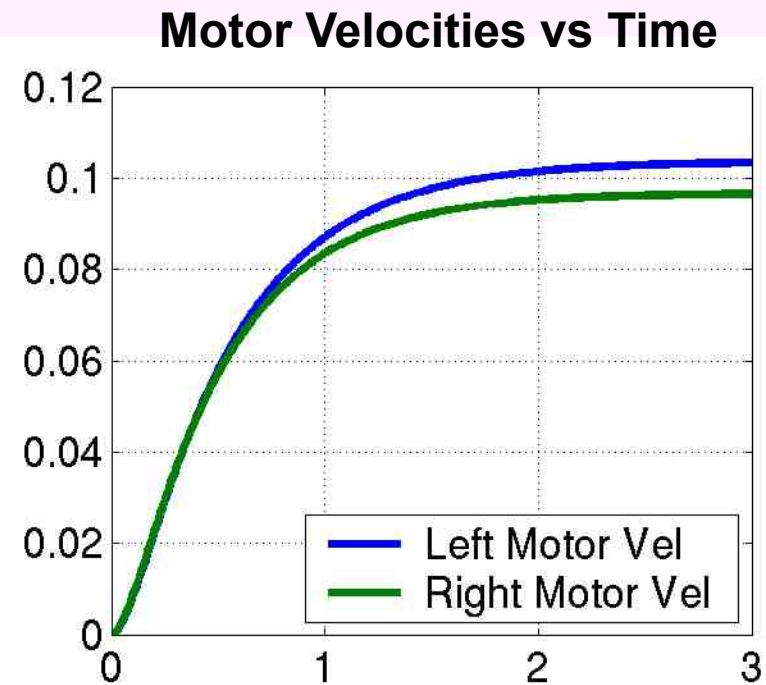
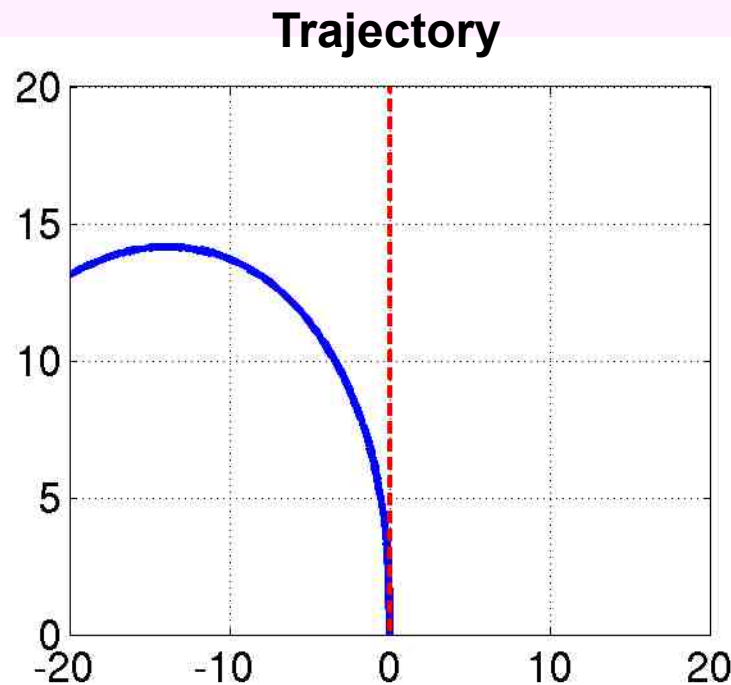
- Calculate by integrating motor velocities and assuming differential steering model for the robot
- Will help the robot go straight ahead

Alternatives:

- Gyro
- Camera



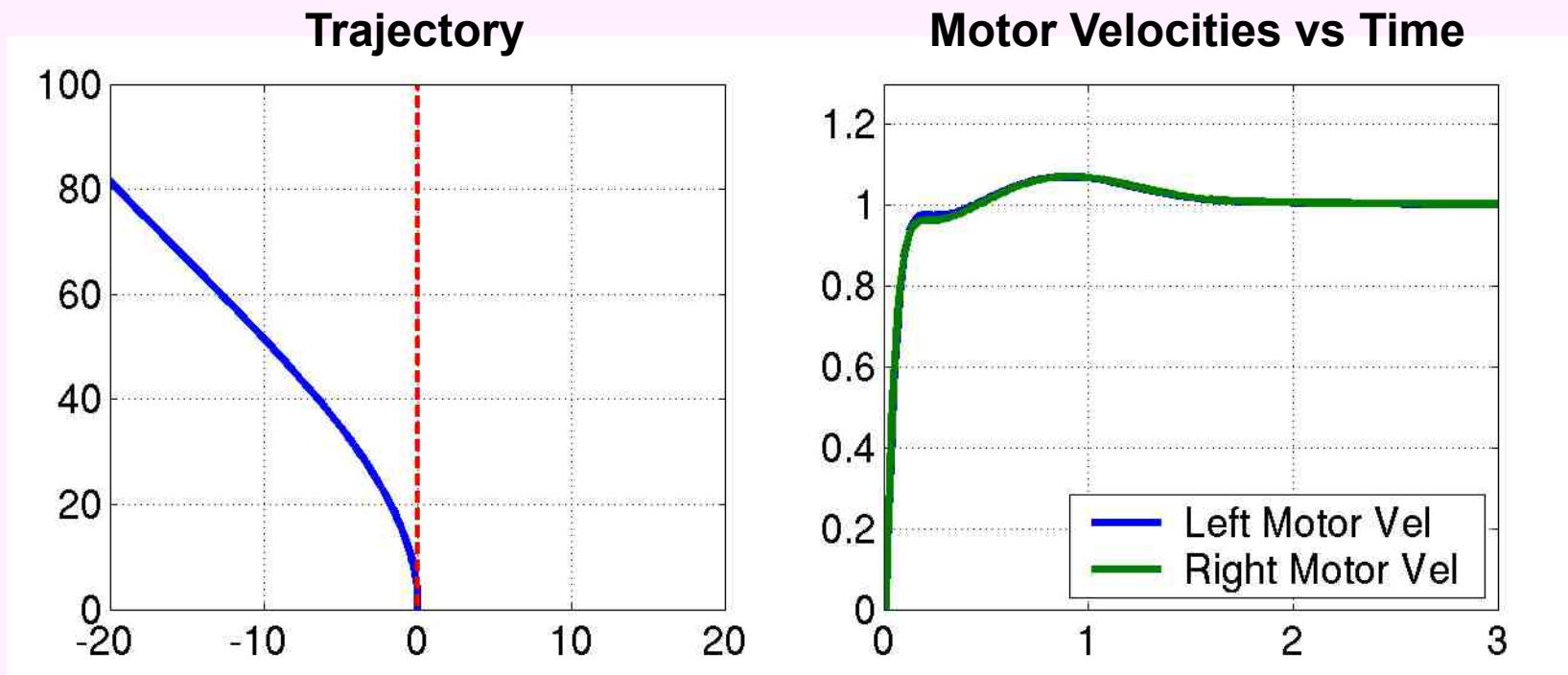
Robot driving in a straight line



Model differential drive with slight motor mismatch

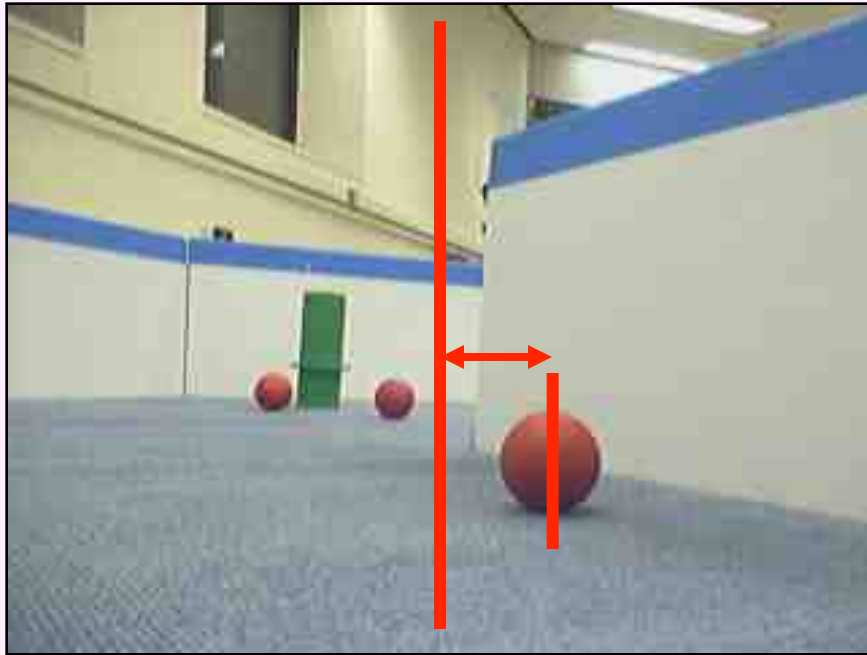
With an open loop controller, setting motors to same velocity results in a less than straight trajectory

Robot driving in a straight line



With an independent PID controller for each motor, setting motors to same velocity results in a straight trajectory but not necessarily **straight ahead**!

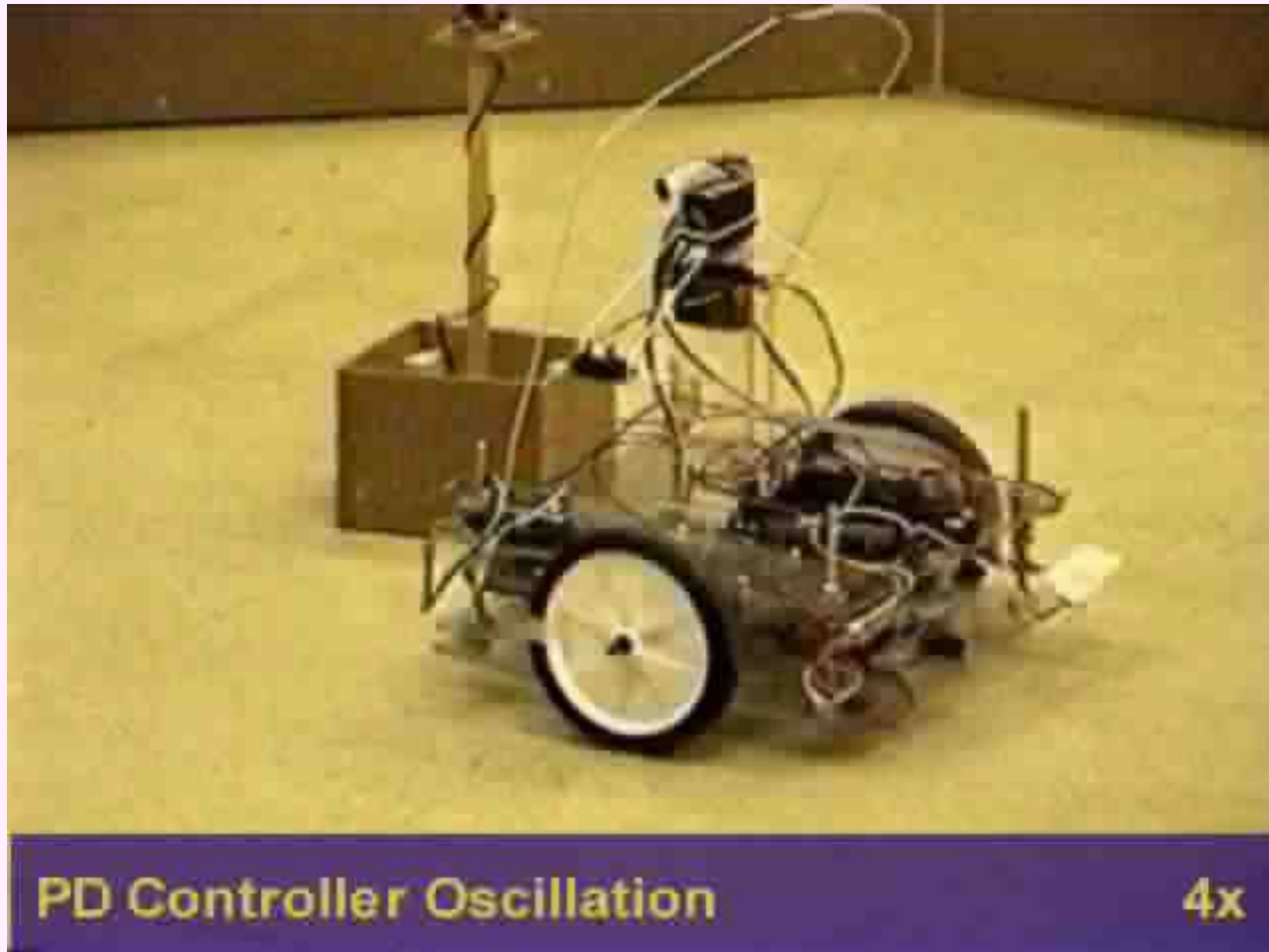
Alternatives: Gyro or Camera



- Track how far ball center is from center of image
- Use analytical model of projection to determine an orientation error
- Push error through PID controller

What if we just used a simple proportional controller? Could lead to steady-state error if motors are not perfectly matched!

Example Videos



Java Examples

Wall Following without PID

<http://web.mit.edu/6.186/2008/lectures/pid/wallfollow/index.html>

Wall Following with PID

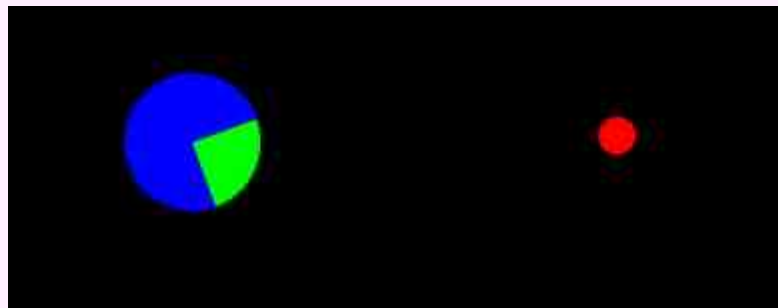
<http://web.mit.edu/6.186/2008/lectures/pid/wallfollowpid/index.html>

Driving Straight without PID

<http://web.mit.edu/6.186/2008/lectures/pid/towardball/index.html>

Driving Straight with PID

<http://web.mit.edu/6.186/2008/lectures/pid/towardballpid/index.html>



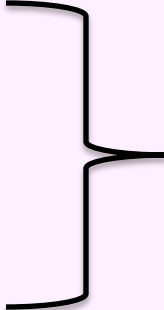
*Thanks to Dany Qumsiyeh

5 Lines of PID Code

```
while(true) {  
    ba = Camera.getballangle;    //Get the ball angle from some other function  
    if (abs(ba) > ANGLETOLERANCE) //Drive Straight  
    {  
        ml = -1; //Left Motor Command  
        mr = 1; //Right Motor Command  
    }  
    else {  
        float adj = anglePID(ba, 2, 0.2, 0.2); //Call PID controller to adjust heading  
        ml = (1 - adj);  
        mr = (1 + adj);  
    }  
}
```

```
float lasterr = 0; //Variables to be saved between calls  
float integral = 0;
```

```
float anglePID(float err, float Kp, float Ki, float Kd) {  
    integral += err;  
    float deriv = err - lasterr;  
    float output = Kp*err + Ki*integral + Kd*deriv;  
    lasterr = err;  
    return output;  
}
```



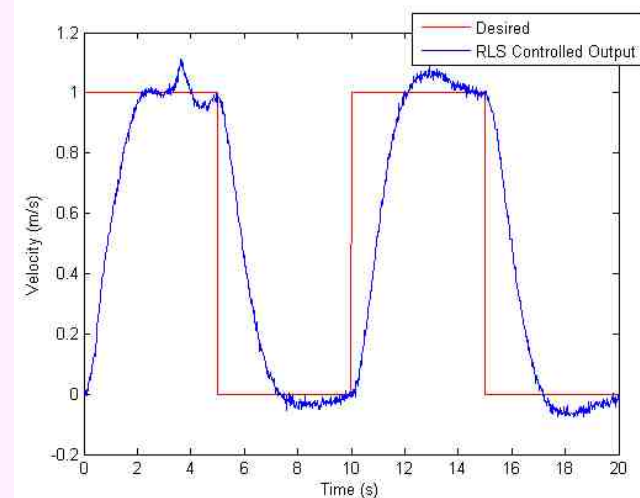
This code for driving
towards a ball has 5 lines
of PID code!

More Advanced Controllers

- There is more to controls than PID!
 - Lead/lag controllers
 - Kalman and Adaptive filters
 - Full state feedback
 - Observers
 - Feedforward
 - Nonlinear Systems
 - Etc...

Kalman Filtering

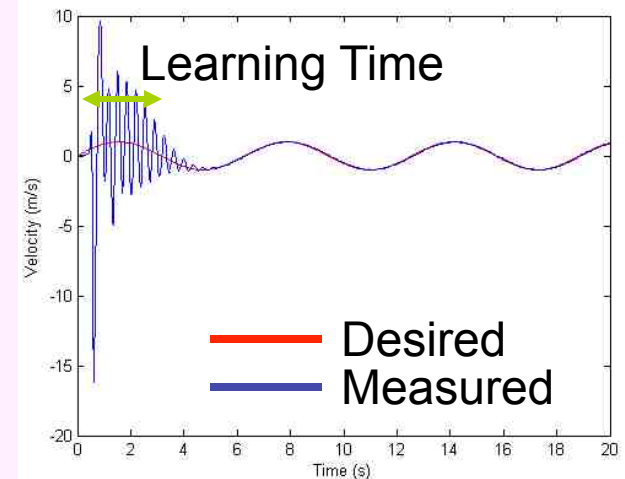
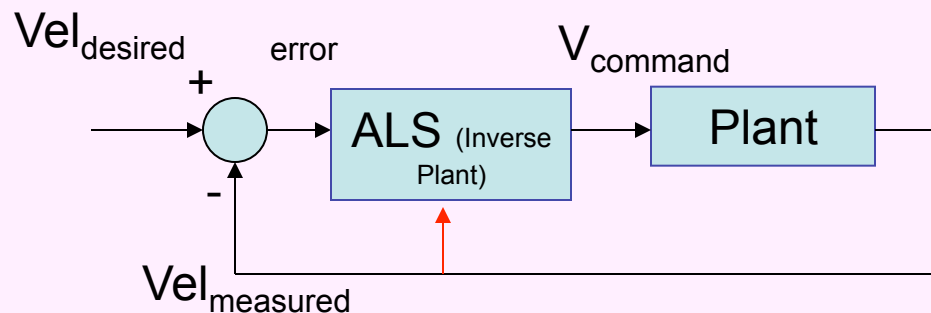
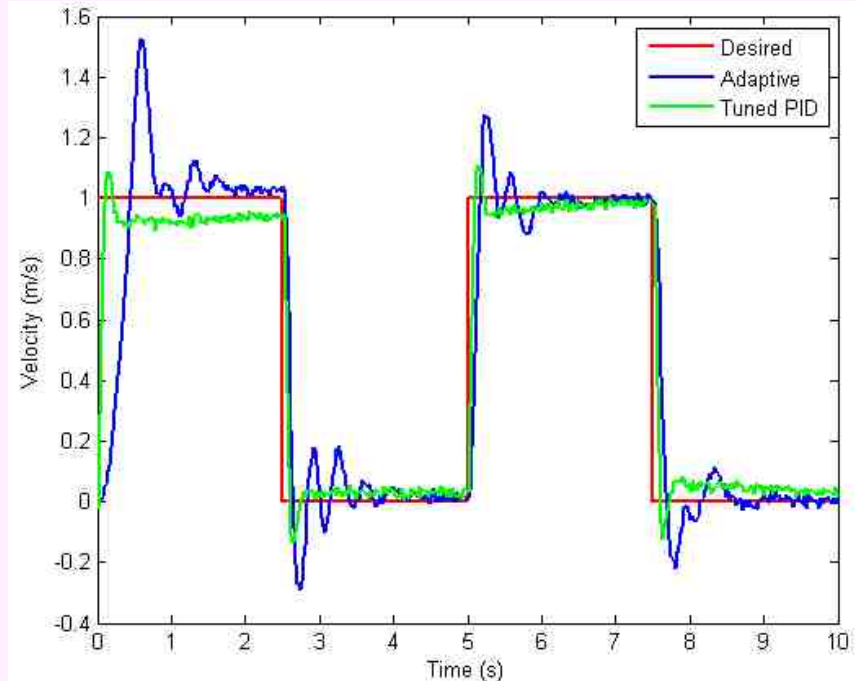
- Recursive method of estimating linear system dynamics in a noisy environment
- Can simultaneously determine system parameters and be used to control the system.



- How does it work?
 - Use a vector to represent system dynamics (impulse response)
 - Collect input and output information and solve for system dynamics
 - Every time a new data point is obtained, we can recursively add this information to our system representation vector (known as update).
- Drawbacks
 - Computational power to invert matrices (time and resources)
 - Needs forgetting factor

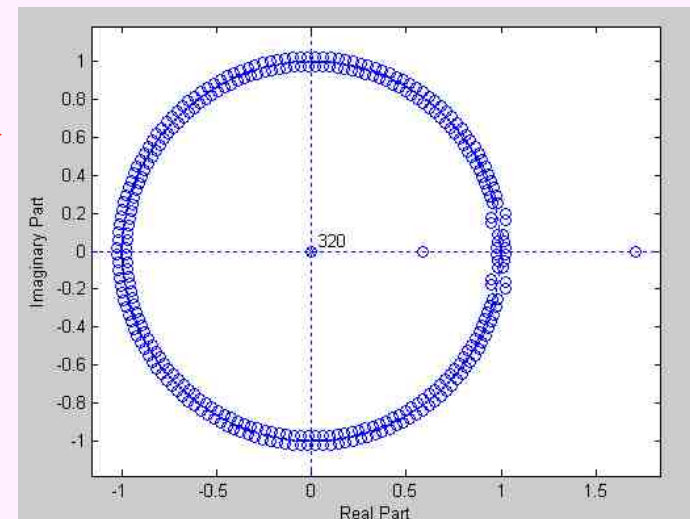
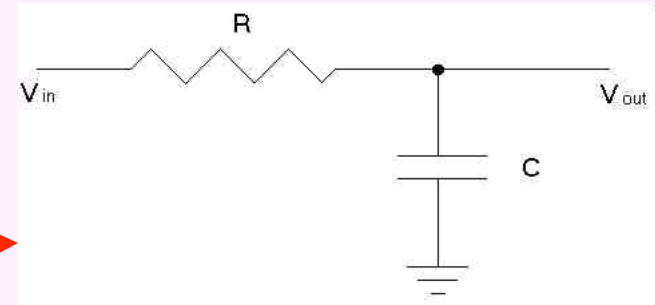
Adaptive Controller

- Self-adjusts estimation of system parameters (vector)
- Slightly faster run time
- Only remembers the most recent data on system dynamics
- Learning time when the program starts



Filter Design

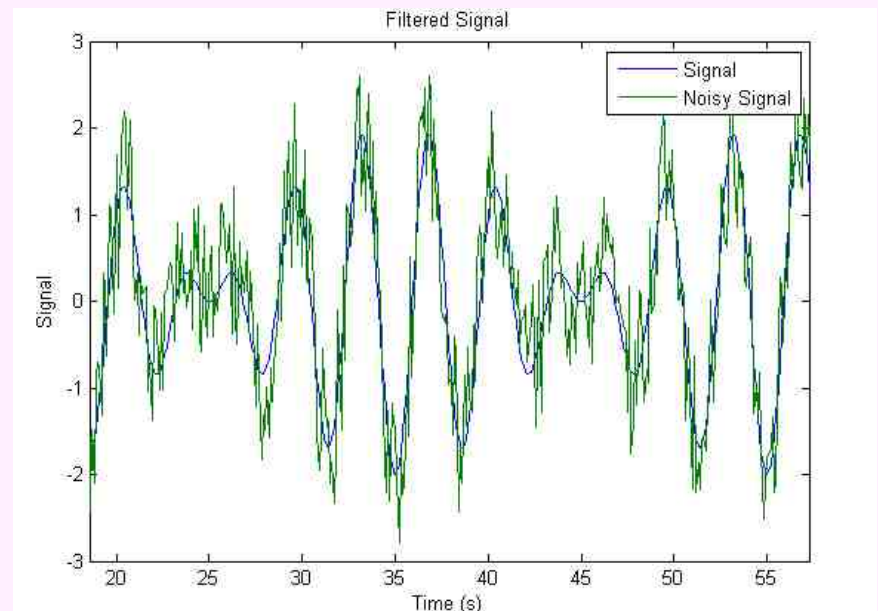
- Continuous Filters
 - In the real world, time is continuous.
 - We are constantly getting inputs and giving outputs
 - Analog circuits
- Discrete Filters
 - When using computers, we get discrete samples at a given sampling rate
 - FIR Filters (Finite Impulse Response)
 - IIR Filters (Infinite Impulse Response)
- Filter Types
 - Low Pass –allows low frequencies to pass through
 - High Pass – allows high frequencies
 - Band Pass- allows a bands of frequencies to pass



Pole/Zero plot for FIR filter

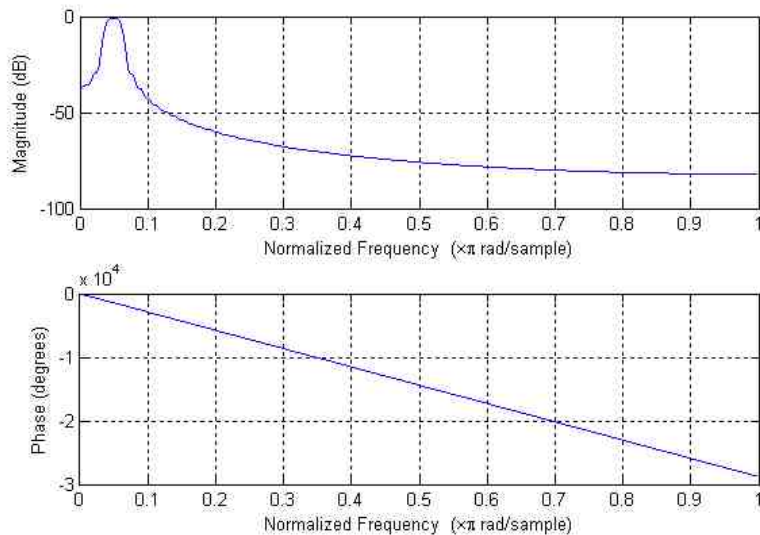
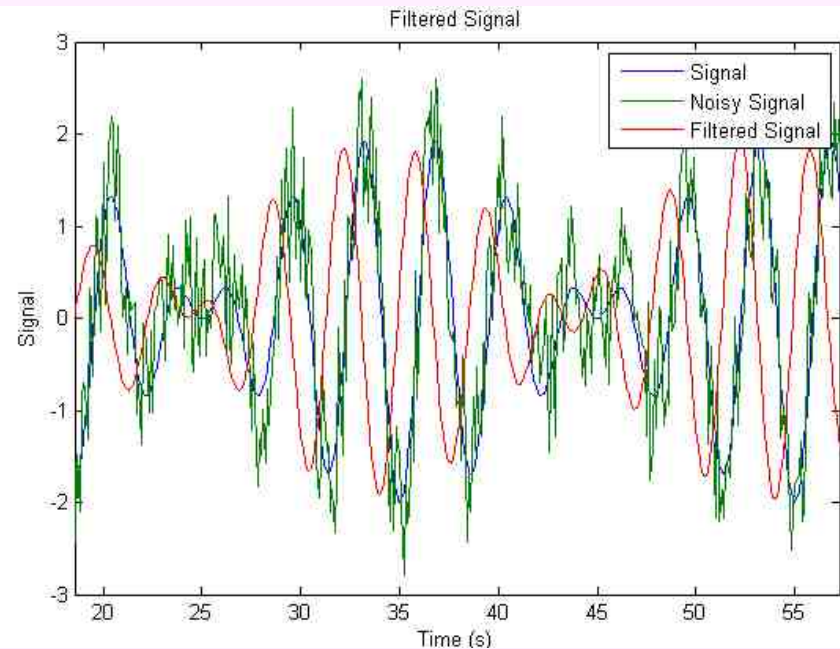
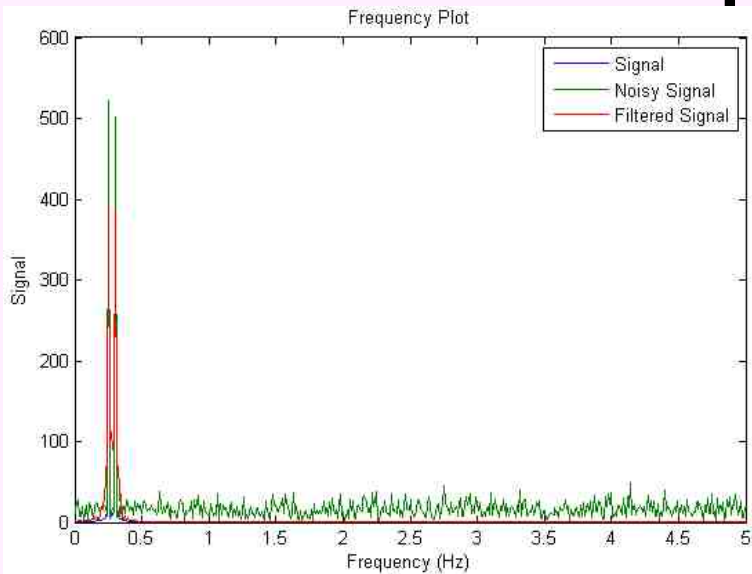
Example: FIR Filter

- Lets say you have a signal and your sensor is very noisy
- Could be IR sensor, ultrasound, or even an image
- How do you separate actual signal from the noise?
- Use an FIR digital filter (in your code)
- $y(n) \rightarrow$ filter output at time n
- $x(n-k) \rightarrow$ sensor input at time $n-k$
- $b \rightarrow$ weighting constants given by Matlab
- $N \rightarrow$ filter order given by Matlab



$$y[n] = b_0x[n] + b_1x[n-1] + \cdots + b_Nx[n-N]$$

Example: FIR filter



- Create band pass filter
- Recover the band of frequencies where the actual signal is
- Special Notes
 - The better the filter, the higher the order (N)
 - The lag in the filter is approximately $N/2$ samples

Matlab Code

- PIDController.m → Script for testing a simple PID controller with arbitrary desired inputs.
- RLSController.m → Kalman filter controls example
- ALSController.m → Simple Adaptive controls example
- Filter.m → Create and test any signal filter

*Code written by Ellen Yi Chen

Extensions

- Controls and signal processing are powerful tools (6.003, 2.004, etc...)
 - Modeling of physical systems
 - Given parameters of a system, how do we determine how it will act to a given input
 - Etc...
 - Control schemes
 - Deterministic control schemes
 - **PID controllers**
 - Fuzzy logic controllers
 - Etc...
 - Signal processing
 - Discrete and continuous methods
 - Filters: **Low-pass**, high-pass, band-pass, notch
 - Frequency domain techniques
 - **Echo removal**
 - **Autocorrelation techniques**
 - Etc...
 - System identification
 - For an unknown black box system, how do we find the transfer function?
 - Impulse invariant, swept sine, stochastic methods
 - Parametric techniques, nonparametric techniques
 - Etc...

Take Aways

- Why do we need controllers?
 - Motors are not matched
 - Your center of mass is not in the middle of your robot
 - Signals are noisy
- Use a PID Controller to simplify driving code
 - Motor Speed: Encoders
 - Robot angle: Gyro
 - Robot trajectory: Gyro and Camera
- Controllers will:
 - Make your robot move and respond faster
 - Make motions smoother
 - Help abstract physics away from desired response
 - Save you from headaches!

References

- Christopher Batten, “Controls for Mobile Robot,” 2007, http://maslab.mit.edu/2007/wiki/Control_lecture.
- Dany Qumsiyeh, Controls scripts 2008, <http://maslab.mit.edu/2008/wiki/PID>.