Feedback Controller Implementation

- The sampling process
- Approximation of continuous time controllers
- Aliasing and new Frequencies: Anti-aliasing filters
- PID Control: Discretization, integrator windup
- Real time implementation

Feedback control systems

Quantization in time and magnitude

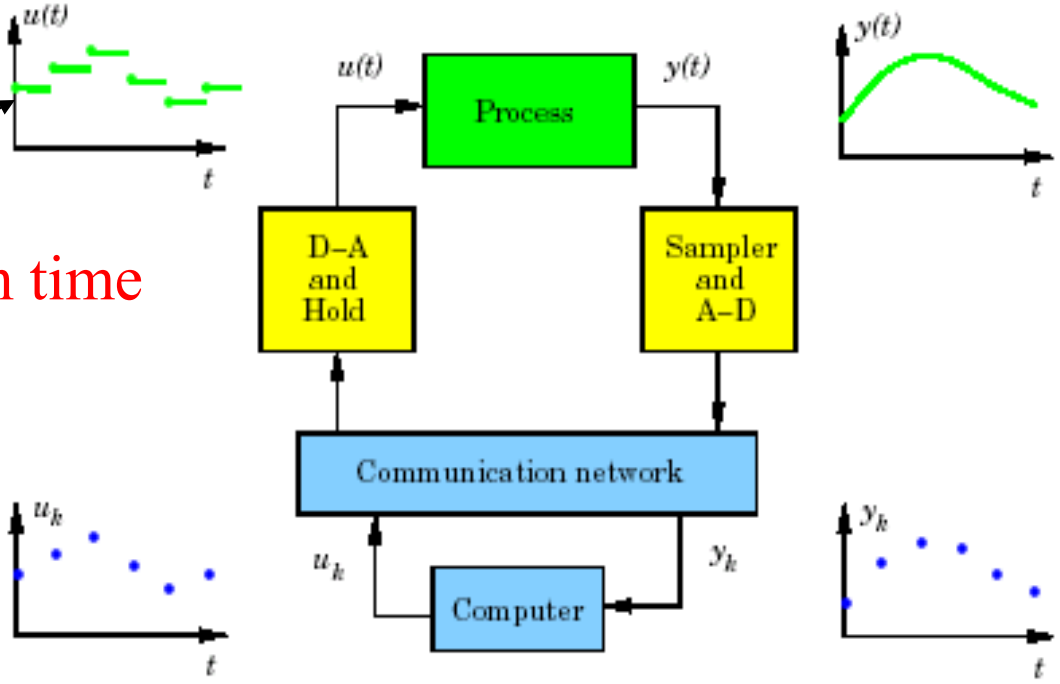


Figure 1 Schematic diagram of a computer-controlled system.

The sampling process

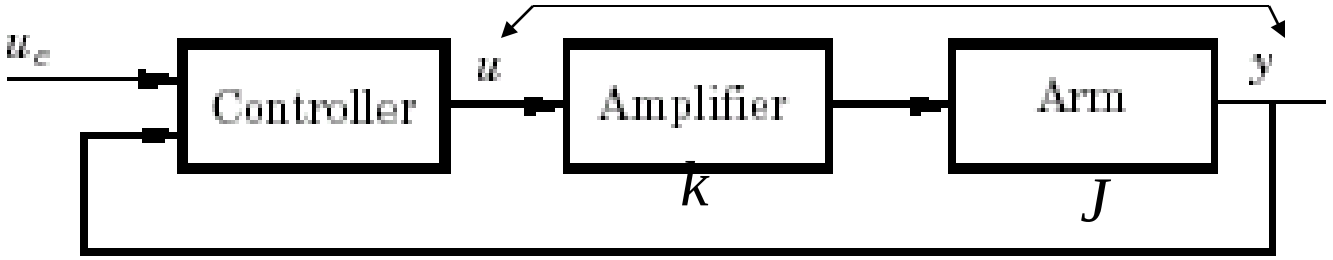
- ***Periodic sampling***
- ***Sampling instants:*** times when the measured physical variables are converted into digital form
- ***Sampling period:*** time between two sampling instants (denoted by h or T)

Sampling and Reconstruction

- ***Sampler:*** A device that converts a continuous time signal into a sequence of numbers, e.g., A/D converter
  - A/D: 8-16 bits → 2^8 to 2^16 levels of quantization, normally higher than the precision of physical sensor
- Reconstructor: Converts numbers from the computer to analog signals
  - D/A converter combined with a *hold circuit (Zero Order Hold)*
  - *PWM:* Plant responds to the average PWM signal

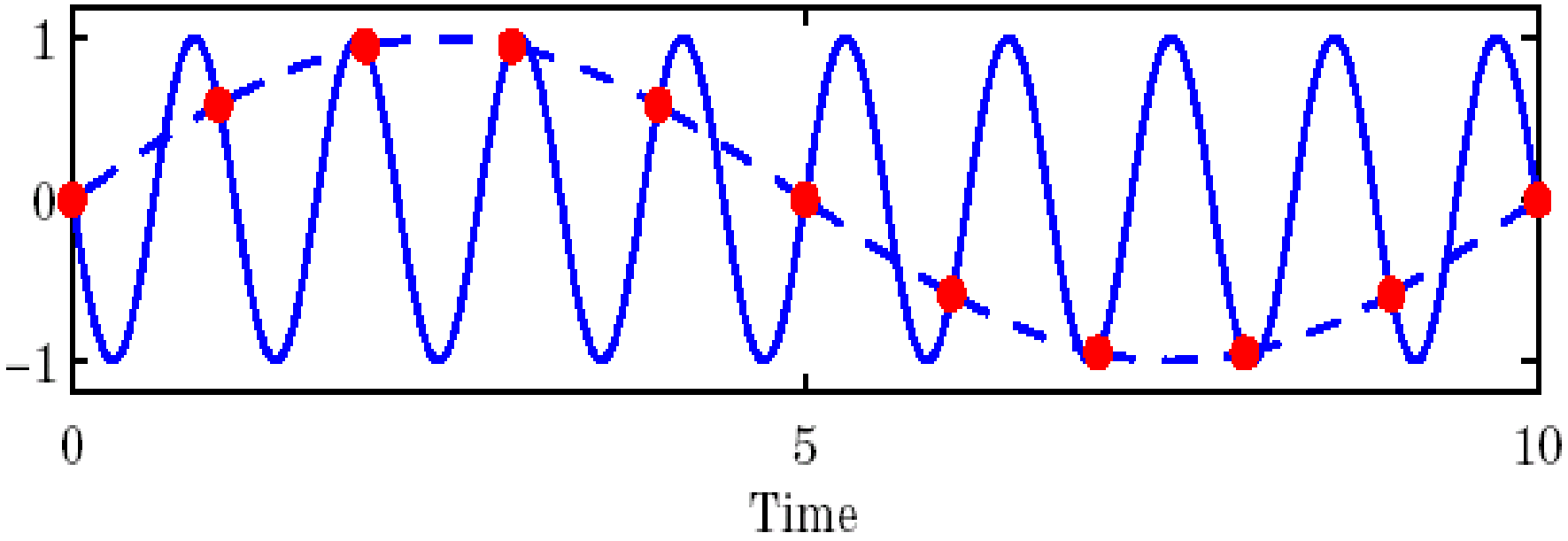
Discretization of continuous controllers

$$G(s) = \frac{k}{Js^2}$$



Disk drive system controller can be discretized by backward/forward differences

Sampling continuous signals



Aliasing and new frequencies

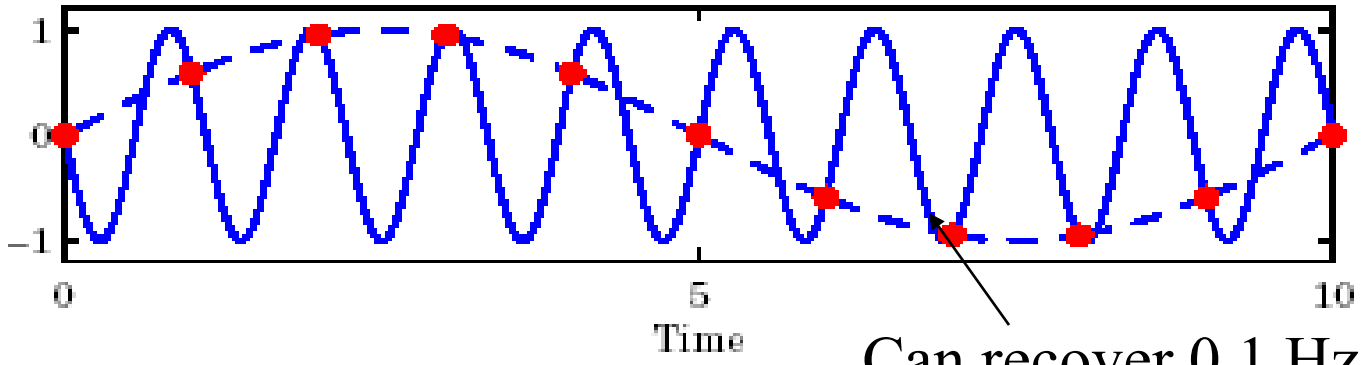
- Can information be lost by sampling a continuous time signal?

→ $\sin(2\pi\ 0.9\ t)$ & $\sin(2\pi\ 0.1\ t)$

- Are sampled values obtained from a low frequency signal or a high frequency signal? → It may be possible that some continuous signal CANNOT be recovered from the sampled signal→ May have loss of information

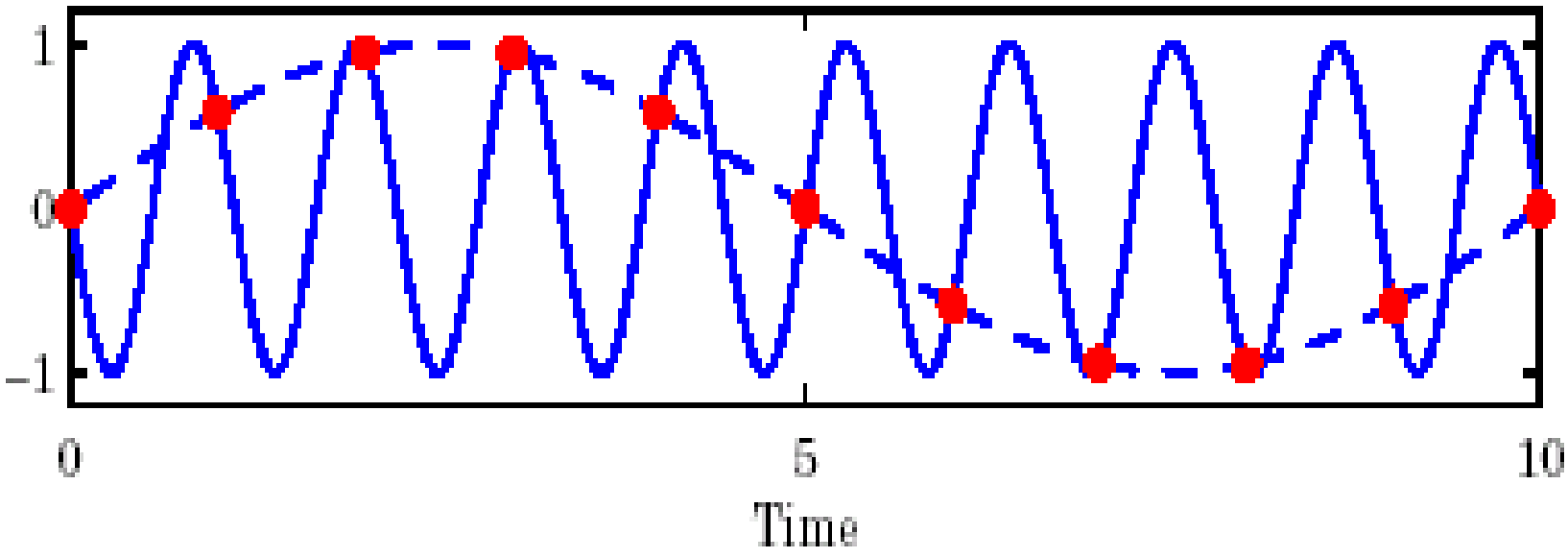
Shannon's Sampling Theorem (1949)

- If a signal contains no frequencies above $\omega 0$, then the continuous time signal can be uniquely reconstructed from a periodically sampled sequence provided the sampling frequency is higher than $2\omega 0$ (Nyquist rate)



Can recover 0.1 Hz but not 0.9 Hz

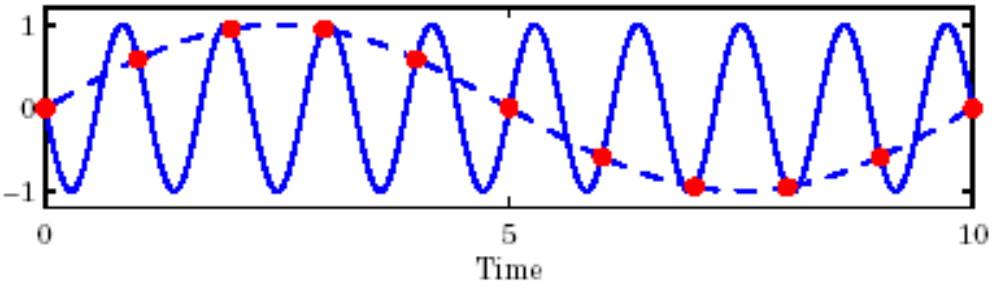
Sampling Theory- Example

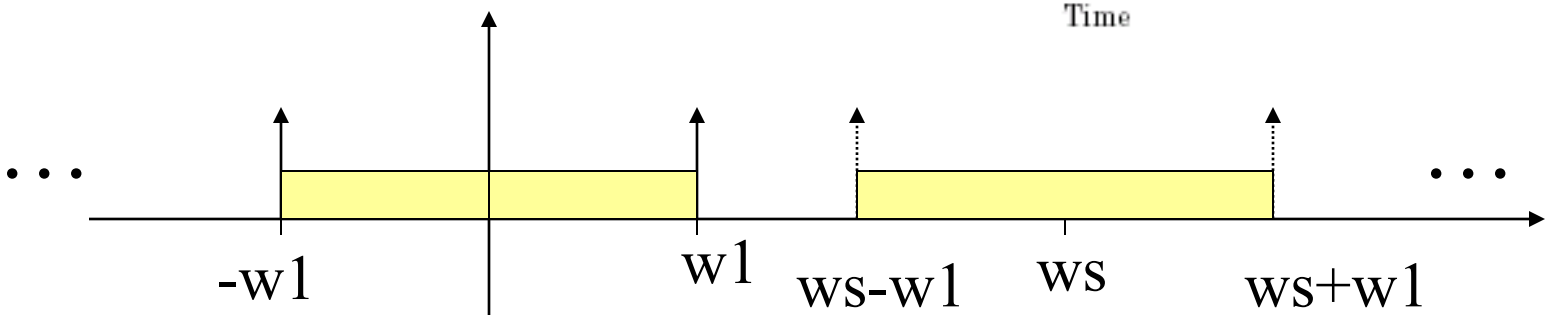


- Can recover 0.1 Hz but not 0.9 Hz
- What should be the sampling frequency for recovering the 0.9 Hz sinusoidal?

Aliasing & Anti-aliasing Filters

- Aliasing (or frequency folding): High frequency signal interpreted as a low frequency signal when sampled at lower than the Nyquist rate (the 0.1=1-0.9 Hz signal)





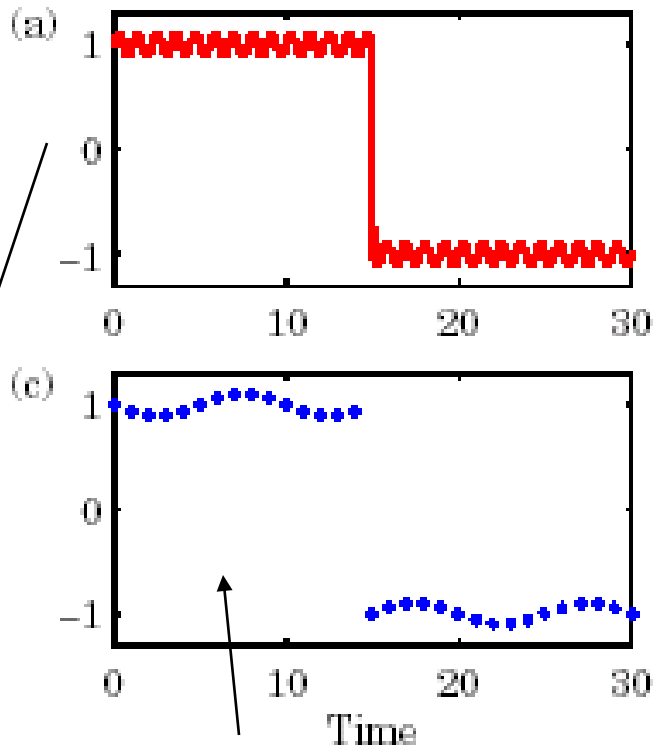
… anti-aliasing filters

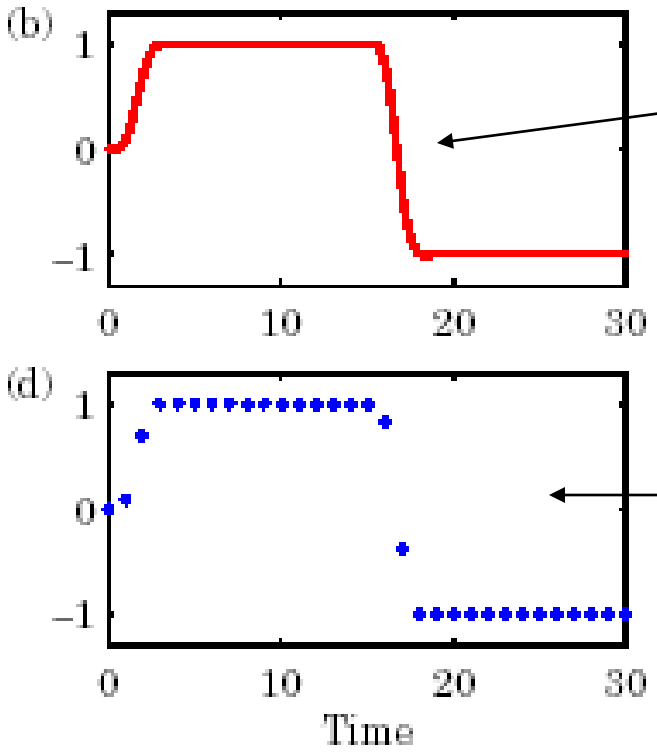
- To prevent aliasing:
  - Remove all frequencies above the Nyquist frequency before sampling the signal.
- Antialiasing filter (low pass analog filter)
  - High attenuation at and above Nyquist frequency

$$\frac{\omega^2}{(s/\omega_B)^2 + 2\zeta\omega(s/\omega_B) + \omega^2}$$

Example: Pre-filtering

- Square wave + 0.9 Hz sine disturbance





Filtered by a 0.25Hz LPF.

Sampled values

Sampled at 1 Hz (alias freq 0.1 Hz)

Summary

- Information can be lost through sampling if the signal contains frequencies higher than the Nyquist frequency.
- Sampling creates new frequencies (aliases).

… summary

- Aliasing makes it necessary to filter the signals before they are sampled.

- Effect of the anti-aliasing filters must be considered when designing controllers.

→ The dynamics can be neglected if the sampling period is sufficiently short.

A Typical Controller

- PID Control: Discretization, saturation and windup
- Other practical issues

PID Control

- Most common controller:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int^t e(s)\,ds + T_d\,\frac{de(t)}{dt}\right)$$

- e: error, K: gain, Ti: integration time, Td: derivative time
- System error ↓ as K & 1/Ti ↑, stability improves as Td ↑, increasing 1/Ti reduces stability
- Originally implemented using analog technology

PID Control & Discretization

- Modifications to PID:

$$U(s) = K\left(bU_c(s) - Y(s) + \frac{1}{sT_i}\left(U_c(s) - Y(s)\right) - \frac{sT_d}{1 + sT_d/N}\,Y(s)\right)$$

- Derivative term:

$$sT_d \approx \frac{sT_d}{1 + sT_d/N}$$

N: gain at high freq: 3-20

- Derivative not acting on the command signal

Discretization of PID Controller

- Proportional part needs no approximation:

$$P(t) = K\Big(bu_c(t) - y(t)\Big)$$

- Integral part: $I(t) = \dfrac{K}{T_i}\int^t e(s)\,ds$

→ can be approximated as

$$I(kh+h) = I(kh) + \frac{Kh}{T_i}\,e(kh)$$

.. Discretization of PID: P+I+D

- Derivative part: D(t)

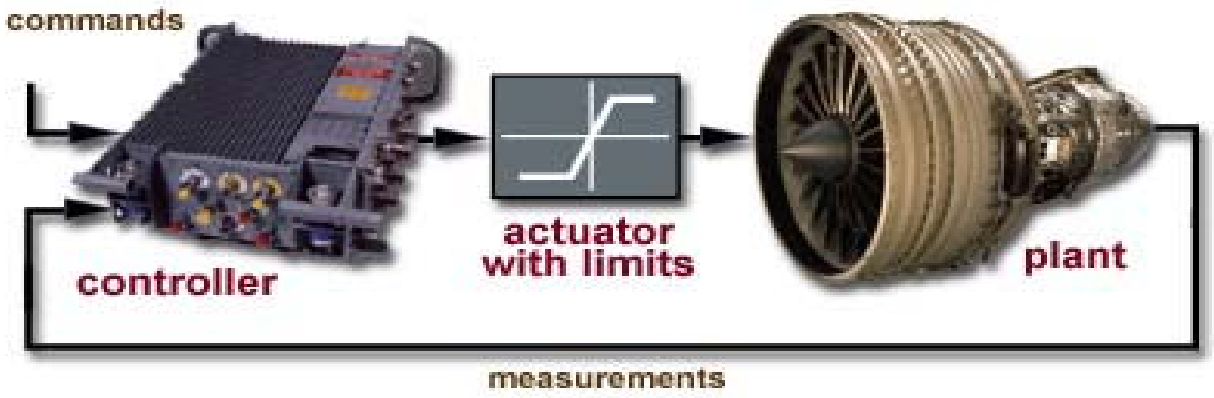
$$\frac{T_d}{N}\frac{dD}{dt} + D = -KT_d\frac{dy}{dt}$$

→ use backward differences for approximation:

$$D(kh) = \frac{T_d}{T_d + Nh}D(kh-h) - \frac{KT_dN}{T_d + Nh}\Big(y(kh) - y(kh-h)\Big)$$

Integrator windup

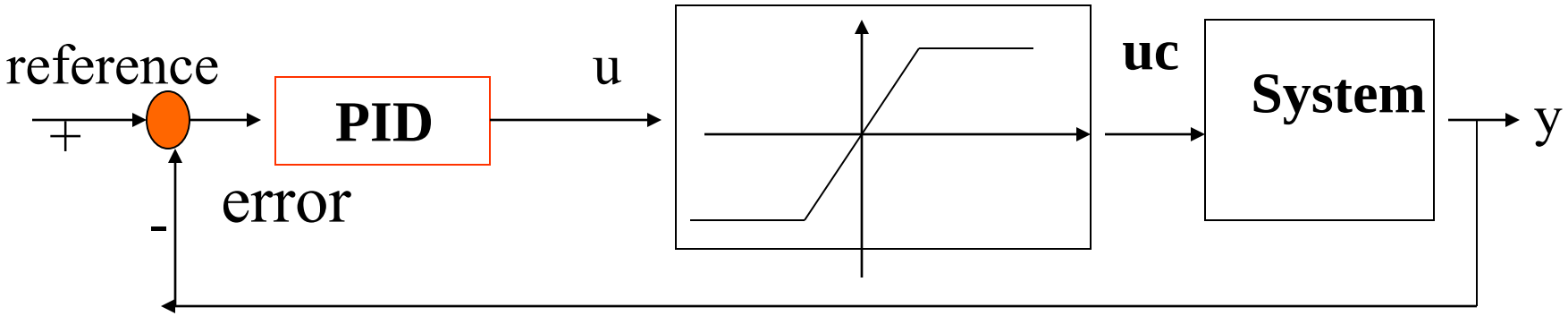
- Actuators can become saturated due to large inputs
- Examples of saturated actuators:
  - throttle position in an automobile
  - amplifier output voltage
  - aircraft control surfaces



- Pushing actuator limits

A taste of the practical world

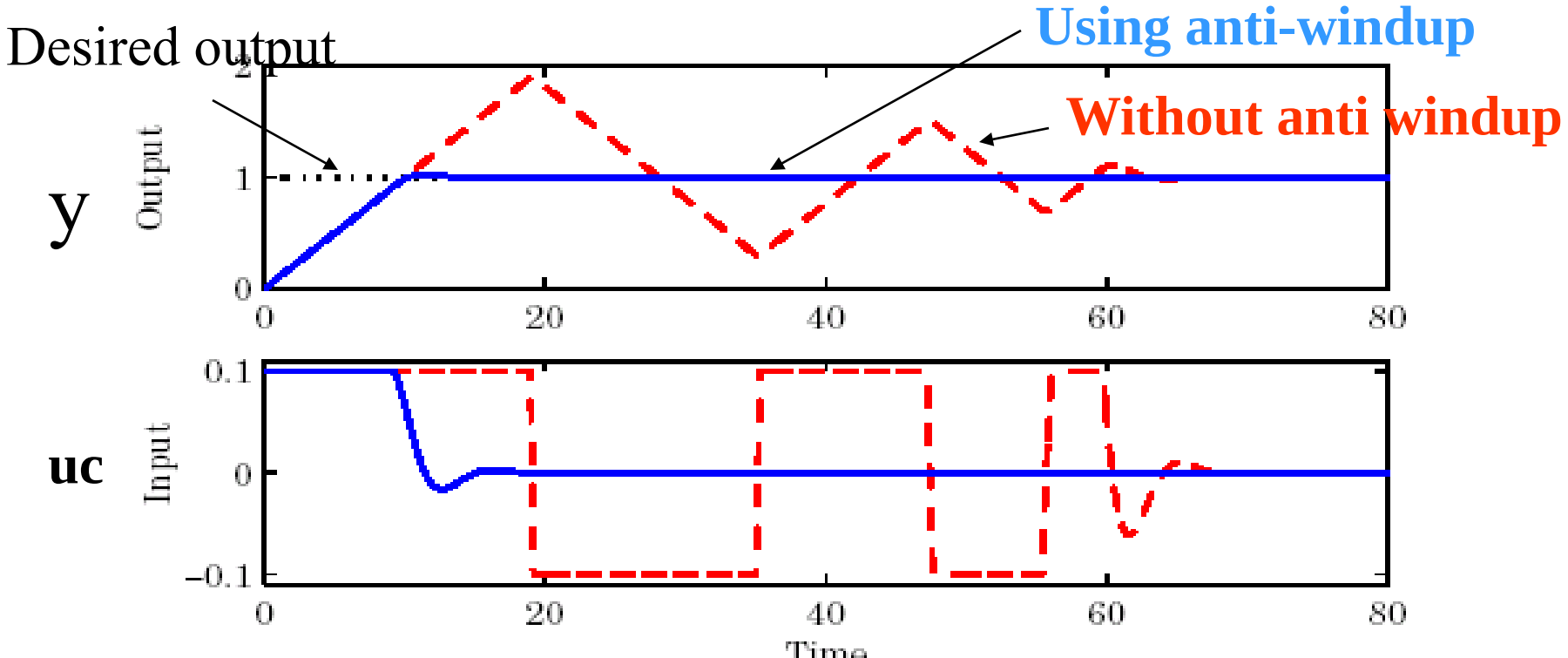
- If error is large, Integral Control can saturate the actuator.



- Integrator output can become large (windup) with no effect on the output (due to saturation).

… integrator windup

- Integrator output ***winds up*** until the sign of $e(t)$ changes and the integration turns the other way around → Solution: Reset integrator when actuator is saturated



Summary

- Things to consider:
  - Filtering of the derivative term
  - Updating the integrator
  - Anti windup compensation

Other practical issues

- **Numerical accuracy is improved if more bits are used:**
  - **float coeff; vs double coeff; (64 bits)**
- **A/D should be accurate but D/A can have much less accuracy**
  - Typical resolutions: 8, 12, 16 bits
  - Better to have a good resolution at the A/D converter than D/A
    - A control system is less crucial for a quantized signal applied to the input of the plant

… practical issues

- Nonlinearities such as quantization by A/D can result in oscillations
  - Do simulations to find out if A/D, D/A resolutions can improve performance

- Selection of the sampling period
  - **A common rule: ωT = 0.1 to 0.6,**

    ω: desired BW of the closed-loop system

... practical issues

- Anti-aliasing filters
  - Must provide attenuation at Nyquist frequency
- Anti-reset windup is important to include in the controller.

Next: Design of a typical control system

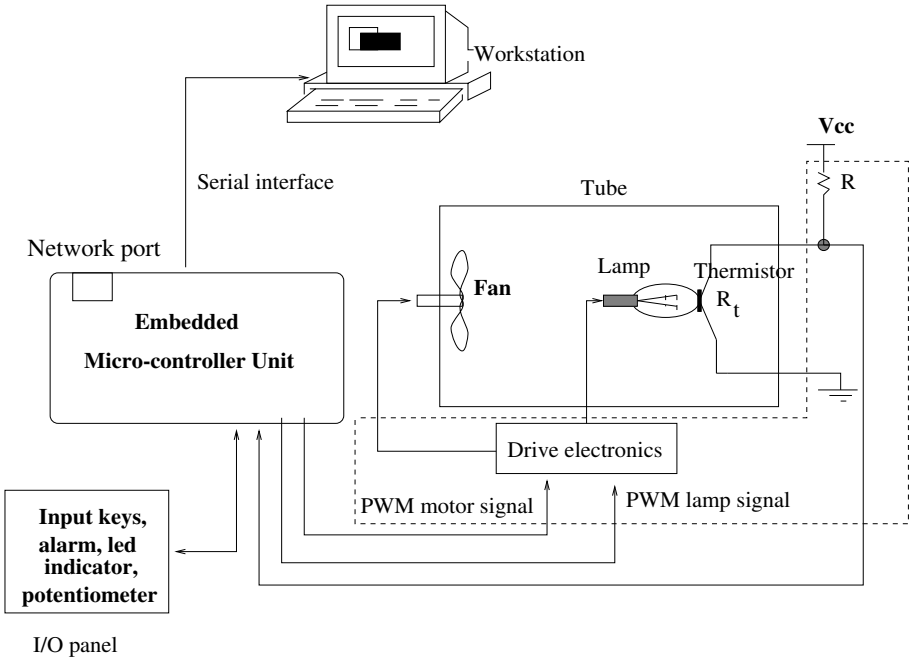


Figure 1: Components of the temperature control system.

Laboratory Setup

- Objective: Design the software and build a sensor-based feedback control system, using a PID controller and existing components

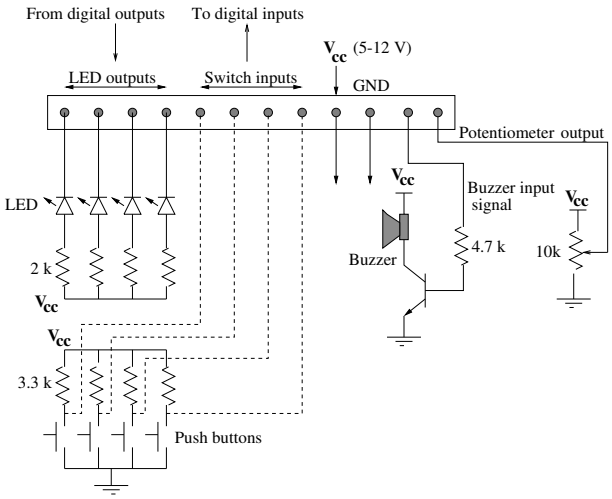


Figure 2: Circuit diagram of the I/O panel board.

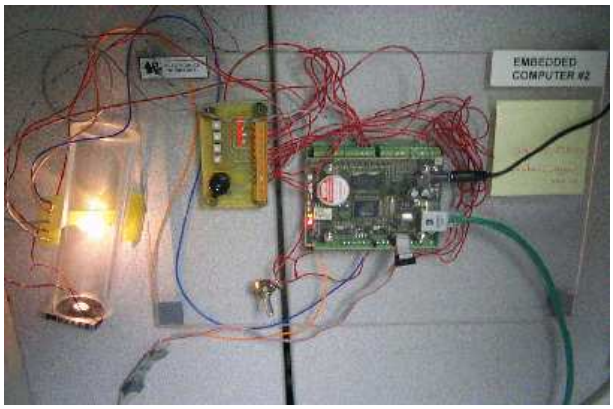


Figure 3: Embedded system hardware components.

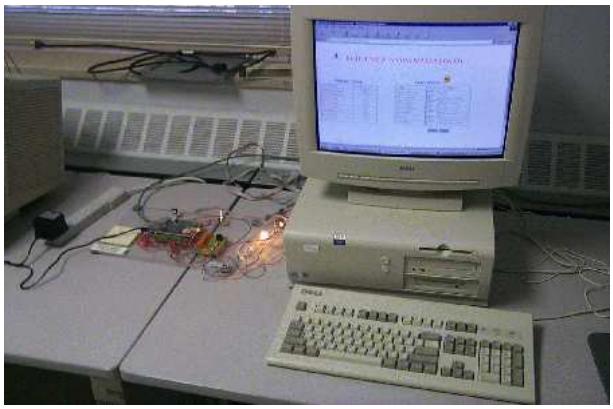


Figure 4: Embedded system controlled and monitored using an Internet browser.

- System components:

  — tube $\rightarrow$ DC fan, lamp, 10 $k\Omega$ thermistor, drive electronics

  — I/O panel $\rightarrow$ push button switches, buzzer, potentiometer,

  — microcontroller development kit

  — host-target configuration

Design of the Control System Software

Coding a feedback algorithm to work properly is not always sufficient from a product design point of view

- Software design must take into account:

  — functionality

  — user interface

  — safety and reliability

  — upgradability

  — power consumption

  — cost

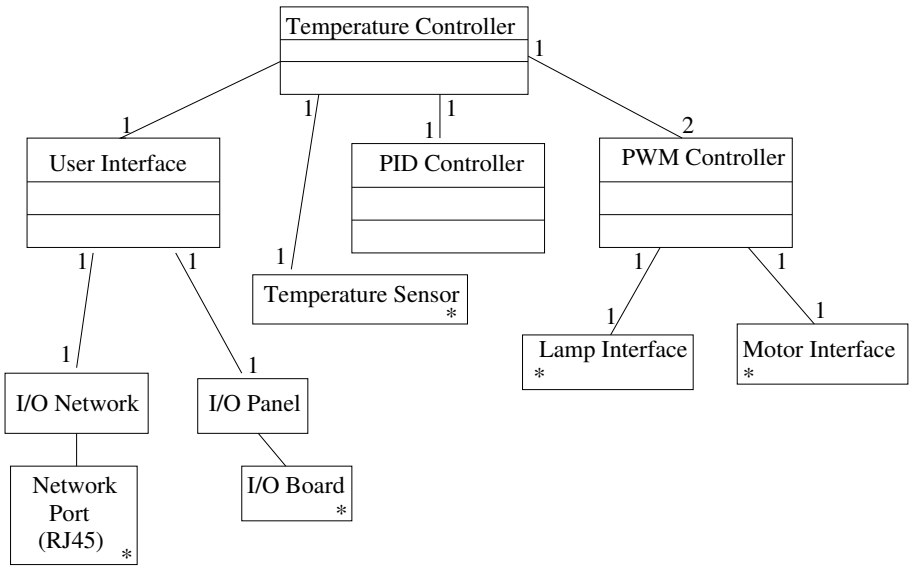


Figure 5: A Unified Modeling Language (UML) class diagram showing composition of subsystems.

- Design steps:

  — requirements, specifications, architecture, components, system integration

  — top-down, bottom-up approach

- Class diagram:

  — UML represantation

  — classes: *Temperature Controller*, *User Interface*, *PID Controller*, *PWM Controller*

  — *Temperature Controller* is the main module → initializes other modules, links modules together, and runs systems' tasks

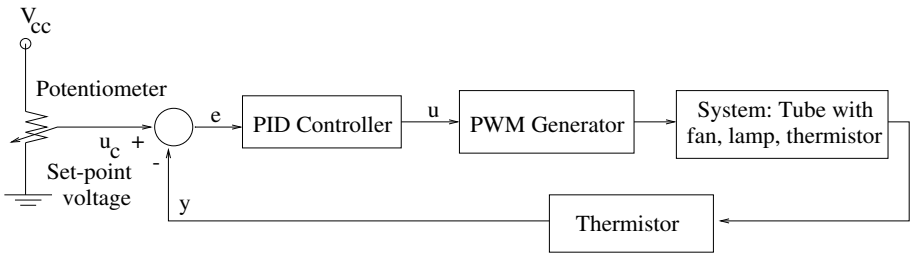


Figure 6: Block diagram of the temperature feedback control system.

PID Controller

- Modified PID (B. Wittenmark, *et. al*):

$$U(s) = K(bU_c(s) - Y(s) + \frac{1}{sT_i}(U_c(s) - Y(s)) - \frac{sT_d}{1 + sT_d/N}Y(s)) \quad (1)$$

- Discretized algorithm:

$$u(kT) = P(kT) + I(kT) + D(kT) \tag{2}$$

where

$$\begin{aligned} P(kT) &= K(bu(kT) - y(kT)) \\ I(kT) &= I((k-1)T) + \frac{KT}{T_i} e((k-1)T) \\ D(kT) &= \frac{T_d}{T_d + NT} D((k-1)T) \\ &- \frac{KT_d N}{T_d + NT}(y(kT) - y((k-1)T)). \end{aligned} \tag{3}$$

- Modular software in C $\rightarrow$ Abstact Data Types (ADT)
  — *PID Controller* module in Figure 5

```
typedef struct {
 float K, b, Ti, T, Td;
 unsigned short N; /* parameters */
 float i, ilast; /* integral term */
 float y, ylast; /* output */
 float d, dlast; /* derivative term */
 float uc;       /* input */
 float u;        /* output */
 float elast;    /* error term */
```

```
 float ulow, uhigh;/* low/high inputs
            for windup compensation */
} pid_t;
```

— routines operating on the `pid_t` data type:

```
void initPid(pid_t *p, ...);
void updatePid(pid_t *p, ...);
void calcPid(pid_t *p, ...);
```

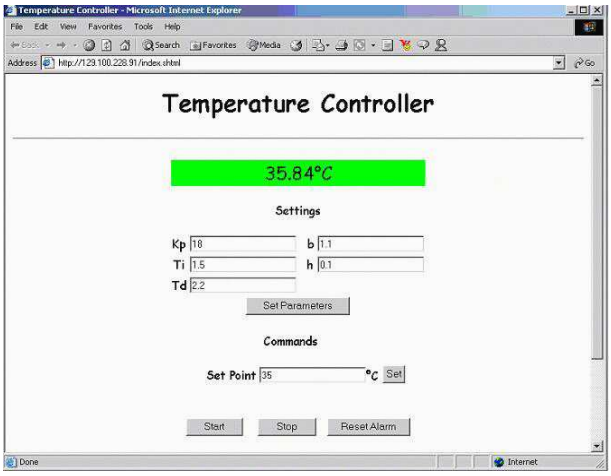


Figure 7: Web browser interface to the embedded system.

Other Modules

- *PWM Controller*

  — uses a digital output pin

- *User Interface*: Scans I/O panel for

  — new inputs from the push button switches and the potentiometer

  — provides outputs to the LED's and the buzzer

  — handles *I/O Network* (for devices with a network port): email, ftp, http

- *Temperature Sensor* consists of a thermistor in series with a resistor

$$T = \frac{\beta T_0}{\beta - T_0 ln\left(\frac{v_t R}{(V_{cc} - v_t) R(T_0)}\right)}. \tag{4}$$

Execution of Tasks

- Concurrent programming
  - — allows for separation of modules into separate tasks
  - — a key element in modular software development
  - — schedulability analysis is required
- Cooperative multitasking

```
main ( ) {
/* variable declarations */
 for (; ;){
   costate Task_1 {
      waitfor(delay(period of task 1);
   /* code for task 1 */
      }
   costate Task_2 {
      waitfor(delay(period of task 2);
   /* code for task 2 */
      }
            ......
   costate Task_N {
      waitfor(delay(period of task N);
   /* code for task N */
      }
   }
 }
```

— *costates* are blocks of code that can suspend themselves

— timing analysis can be performed on the system

Conclusion

- Control systems are one of the main application domains for embedded computing
- There is a growing need to train control engineers who are aware of the design procedures and challenges in embedded computing technology
- An experimental testbed to serve the above purpose was introduced

M. Moallem, "A Laboratory Testbed for Embedded Computer Control," IEEE Transactions on Education, Vol. 47, No. 3, pp.340-347, 2004.