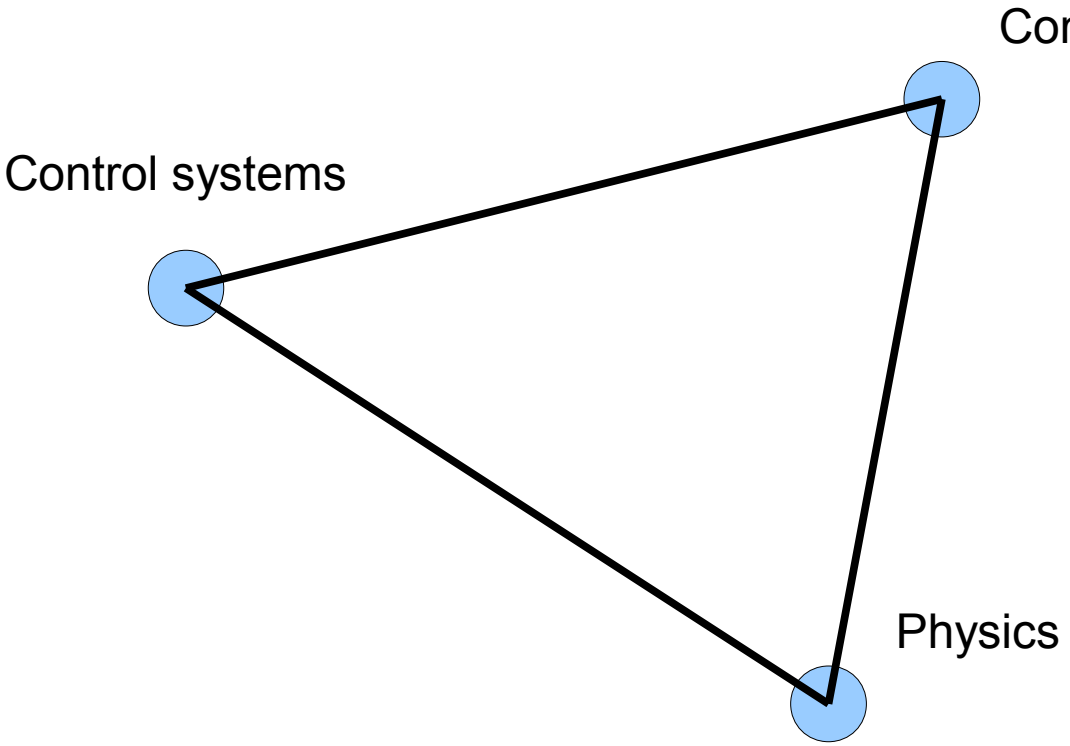
ScicosLab/Scicos for dummies like us

www.scicoslab.org

www.scicos.org

Digital control systems design and simulation: the Bermuda Triangle for engineers



What ScicosLab is?

- An interpreted language (“Scilab Language”, very similar _but_not_equal_ to Matlab)
- Full support for matrix computation (BLAS, LAPACK, single and multi cores support using ACML now, GPU support in the near future using OpenCL).
- Basic programming (just like Matlab)
- GUI development (using TCL/TK and UICONTROL)
- Linear algebra
- Polynomial calculation
- Control systems modelling and design (continuous, discrete and hybrid systems)
- Robust control toolbox
- Optimization and simulation (build-in solvers)
- Signal processing (filters design, etc.)
- ARMA modelling and simulation
- Basic statistics
- Basic identification
- PVM support
- TCL/TK and Java support
- Max-plus algebra toolbox

What you can do with ScicosLab?

- Interact with the command line
- Write a program, one line at time, using the internal or with an external editor
- Run the program in an user friendly interpreted environment (easy debug)
- Use the rich embedded library.
- You can develop your ScicosLab functions using C, Fortran, Java, etc.
- Produce nice graphics diagrams.

What Scicos is?

Scicos is a graphical dynamical system modeller and simulator developed inside the METALAU project at INRIA, Paris Rocquencourt centre.

Scicos is a **graphics**, object oriented tool where the user create block diagrams to model and simulate the dynamics (***time domain***) of **hybrid** dynamical systems and compile models into executable code (code generation for simulation and embedded applications).

Scicos is used for signal processing, systems control, queuing systems, and to study physical and biological systems.

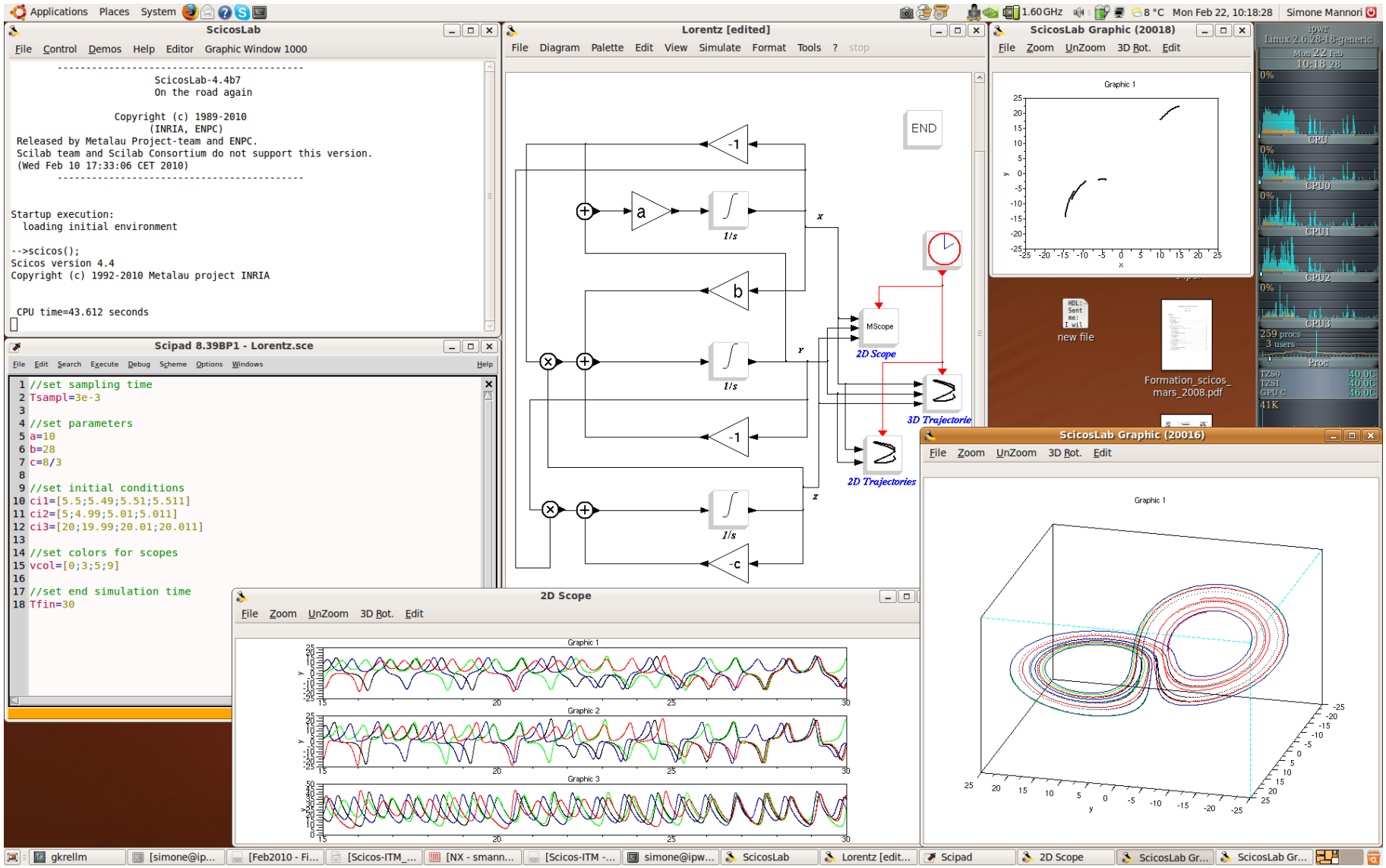
New extensions allow generation of component based modelling of electrical and hydraulic circuits using the **Modelica** language.

What you can do with Scicos?

- Graphically model, compile, and simulate dynamical systems
- Combine continuous and discrete-time behaviours in the same model
- Select model elements from “Palettes” of standard blocks
- Program new blocks in C, Fortran, or Scilab Language
- Run simulations in batch mode from ScicosLab command line
- Generate C code from Scicos model using the built in Code Generator
- Generate C/C++ libraries ready to be integrated in Kepler using FC2K
- Run simulations in real time with and real devices using **Scicos-HIL**
- Generate hard real-time control executables with **Scicos-RTAI**, **Scicos-FLEX** and **Scicos-ITM** code generators.
- Use implicit blocks developed in the Modelica language.
- Discover new Scicos capability using additional toolboxes like RTSS, Scicos-HDL, Coselica, etc.

ScicosLab

Basic ScicosLab and Scicos



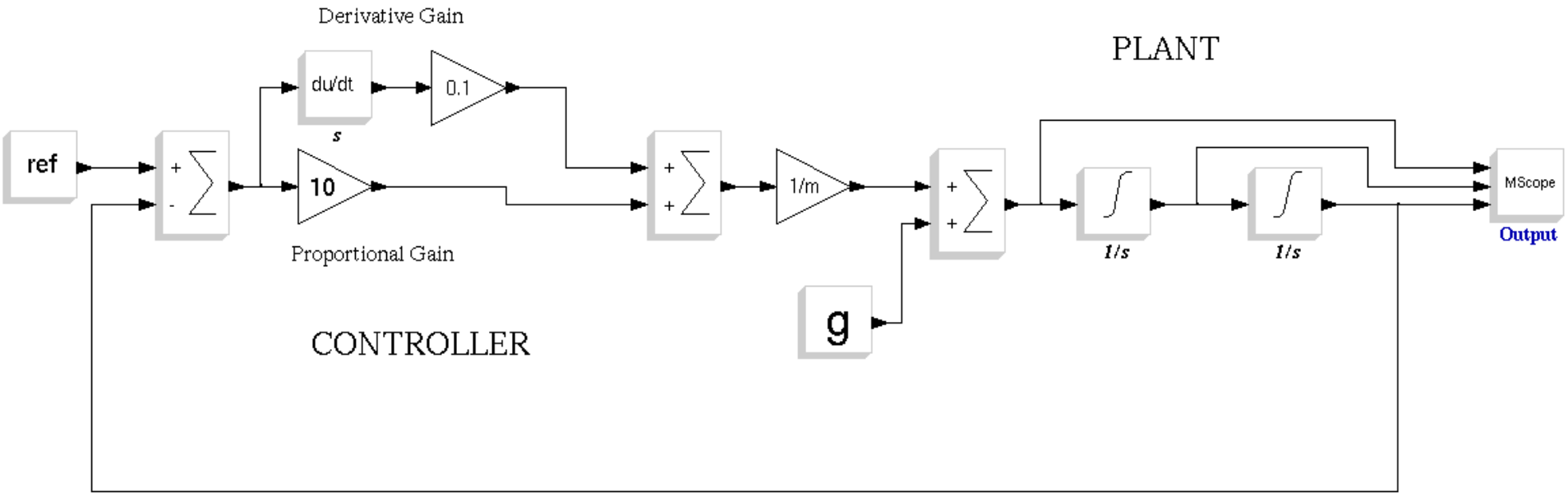
Modelling of complex dynamical systems

ScicosLab and Scicos are designed to handle explicit (ODE, Ordinary Differential Equation) or implicit (IDA, Implicit Differential Algebraic equation) differential equation problem (in the continuous domain) and difference equation (in the discrete domain).

ScicosLab and Scicos can handle also partial differential equation (PDE) and finite element problem, but the user must develop some code to implement features like meshing (not built in ScicosLab).

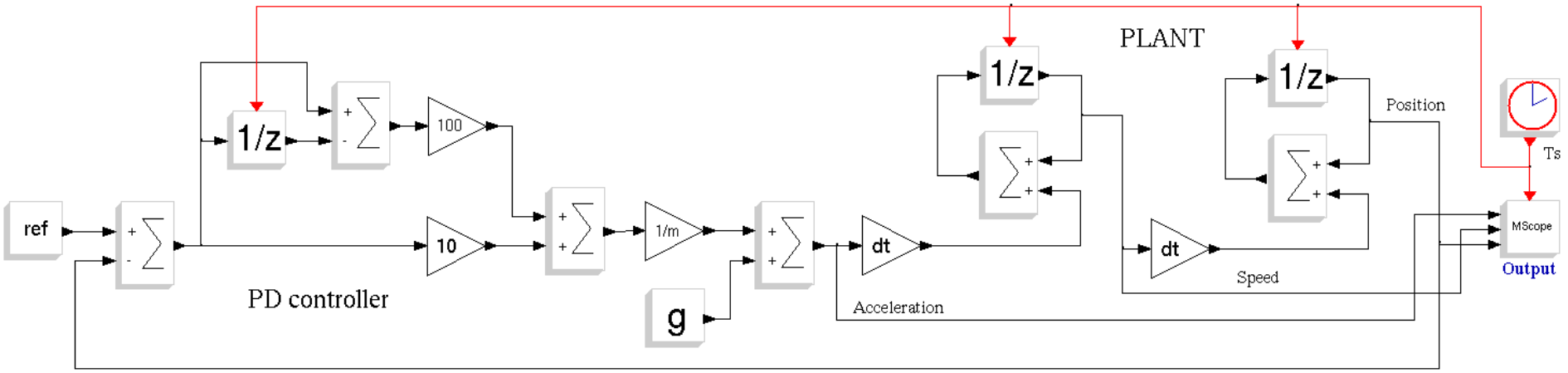
Continuous systems

Linear controller for a floating apple.



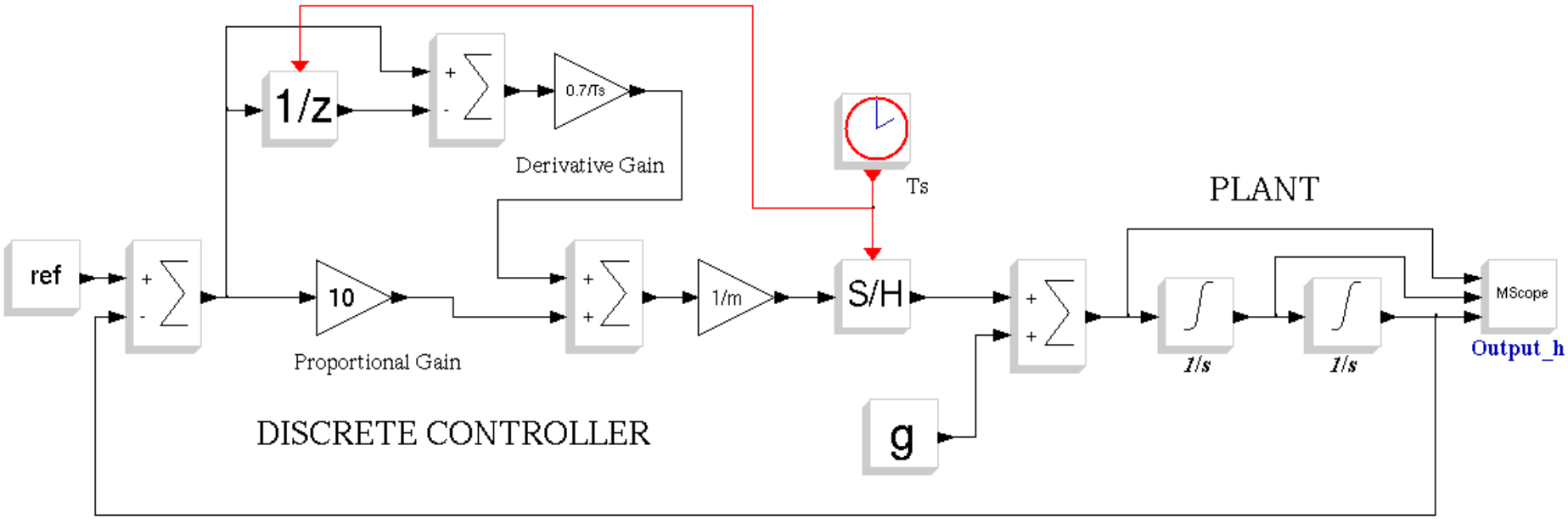
Discrete systems

Same diagram, but realized with (time) discrete blocks.



Hybrid systems

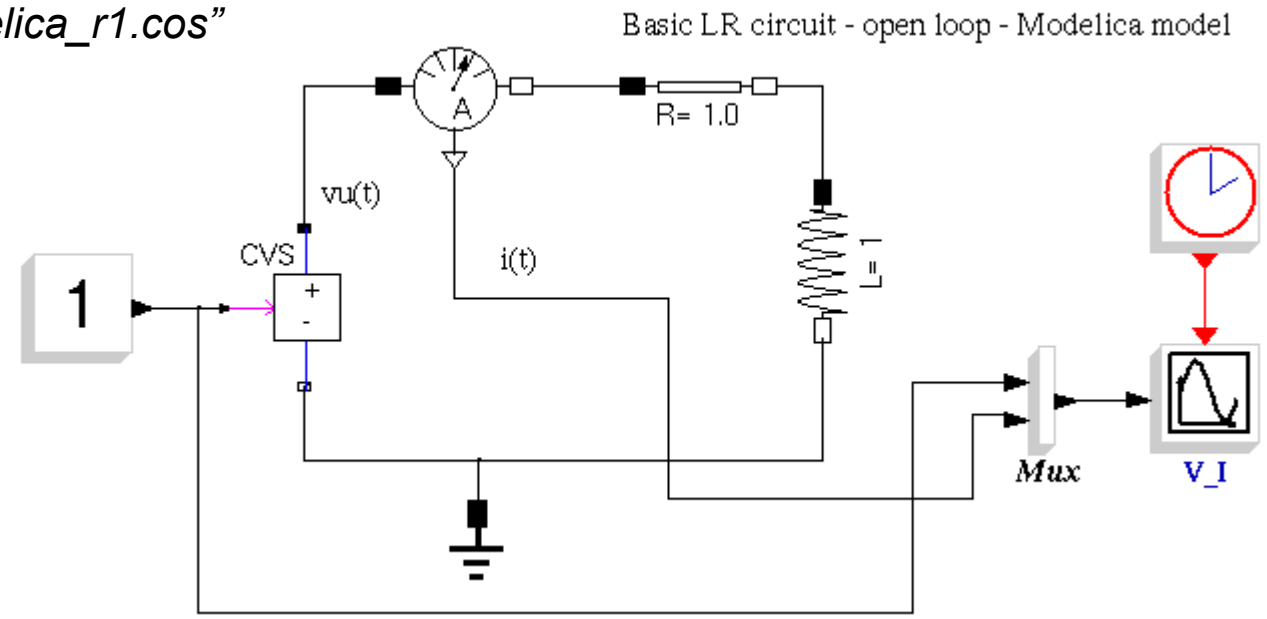
Welcome to the Real World: continuous system, discrete controller.



Implicit model: an easy example.

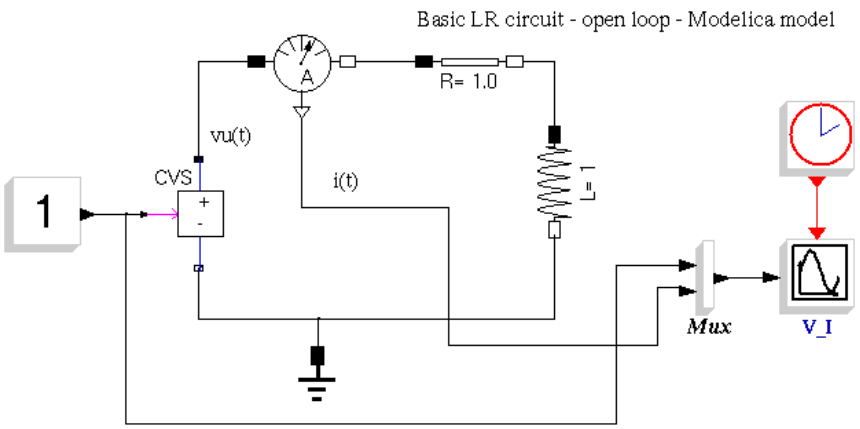
Consider the basic LR circuits show below.
You would like to control the current “i(t)” that flows inside the inductance using “vu(t)” (the voltage source that drive the coil).

“lr_cc_modelica_r1.cos”



Basic modelling

The equations are:



$$v_u(t) = R \cdot i(t) + \frac{d}{dt}\phi(t)$$

$$\phi = L \cdot i$$

$$v_u(t) = R \cdot i(t) + L\frac{d}{dt}i(t) + i(t)\frac{d}{dt}L(t)$$

$$v_u(t) = R \cdot i(t) + L\frac{d}{dt}i(t)$$

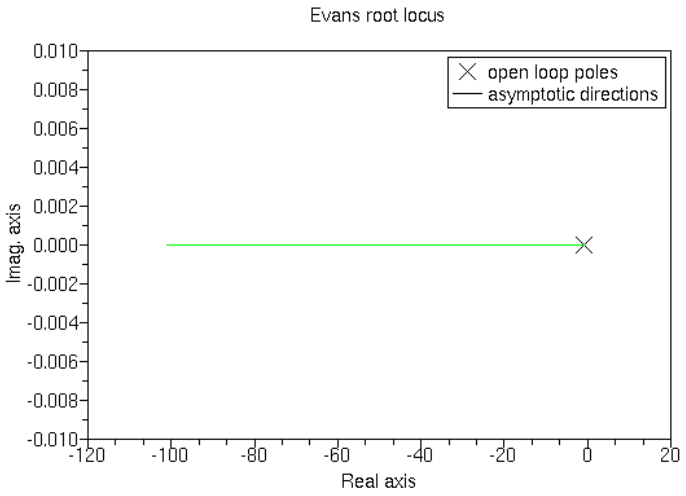
$$L = \mu \cdot K \cdot \frac{N^2 A}{l}$$

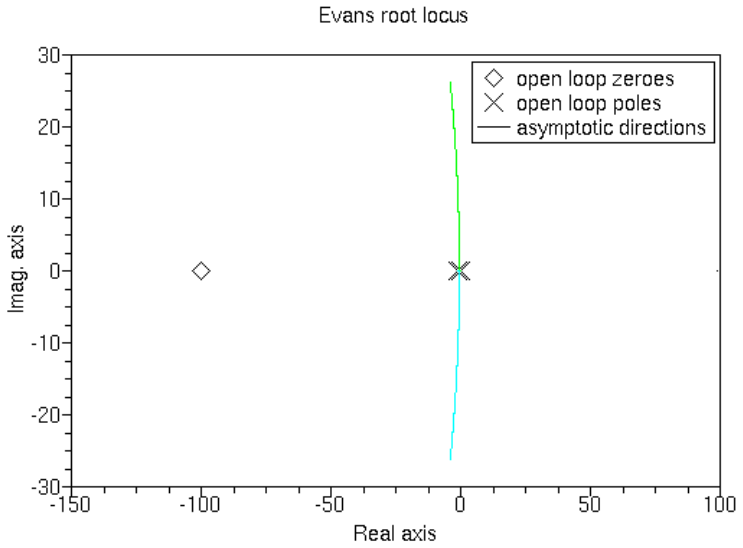
$$\mu = \text{<material>}$$

$$K = f_{Nagaoka}\left(\frac{d}{l}\right)$$

Root locus and Bode plots with ScicosLab

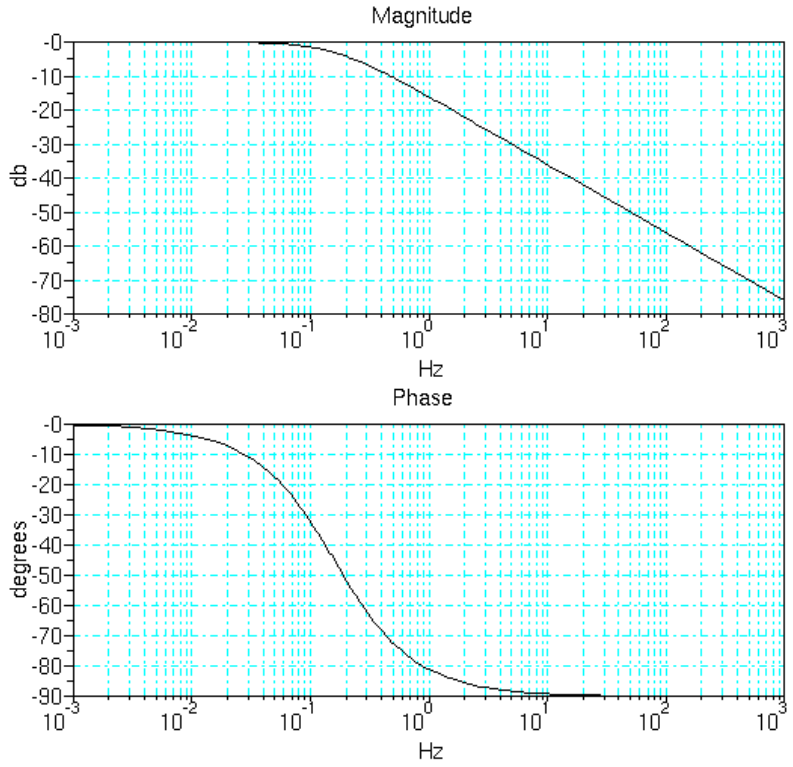
```
1  //** Plant parameters
2  R = 1.0; L = 1.0;
3  //** PI controller parameters
4  Kp = 1; Ki = 100 ;
5  //** sys plant build
6  N_plant = poly([1],"s", "coeff");
7  D_plant = poly([R L],"s", "coeff");
8  sys_ol = syslin("c", N_plant, D_plant);
9  //** plant root locus
10 Kmax = 100;
11 scf(0); evans(sys_ol,Kmax);
12 disp("With PI controller"); pause
13 //** sys PID build
14 N_pid = poly([Ki Kp],"s", "coeff");
15 D_pid = poly([0 1],"s", "coeff");
16 //** controller + plant
17 N = N_pid * N_plant;
18 D = D_pid * D_plant;
19 //** full system
20 sys_cl = syslin("c", N, D);
21 scf(1); evans(sys_cl);
22 //** plant Bode diagram
23 scf(2); bode(sys_ol);
24 //** controller + plant Bode diagram
25 scf(3); bode(sys_cl);
```



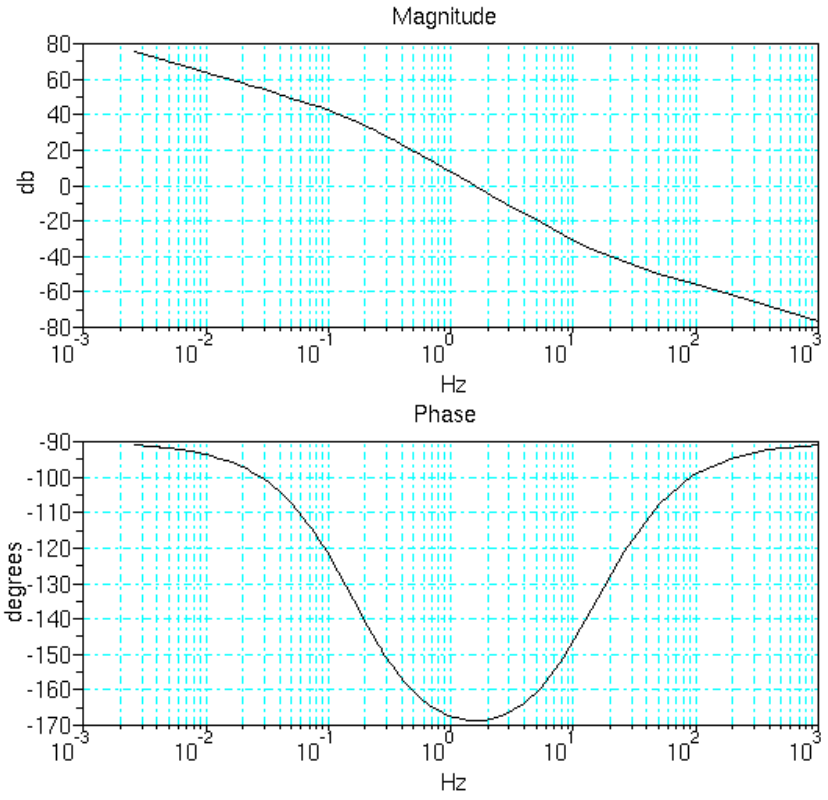


Root locus and Bode plots with ScicosLab

PLANT



PLANT + CONTROLLER



Why do you need control systems?

Control technology was born in order to improve the performances of the "PLANT", e.g. the system that makes the work.

Examples:

- furnace (temperature control)
- audio amplifier (distortion, bandwidth)
- motion control (precision, speed)
- AAA (Anti Aircraft Artillery)
- Flight control (efficiency, comfort, performances)
- Combustion engine (efficiency, emission's reduction)

Continuous systems

Linear, continuous and time invariant systems

Continuous: all the internal and external signals are “proper” time functions. They are unequivocally defined for all values of “t”.

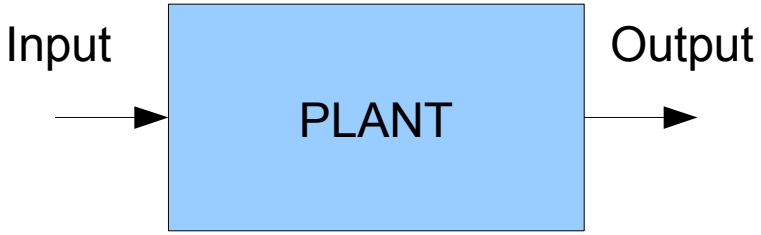
Linear: if u1, u2 are two inputs and y1 and y2 are the two corresponding outputs, the input u = u1 + u2, produces the output y = y1+ y2

Time invariant: the response of the systems does not vary in time (system parameters are time invariant, no “ageing effect”).

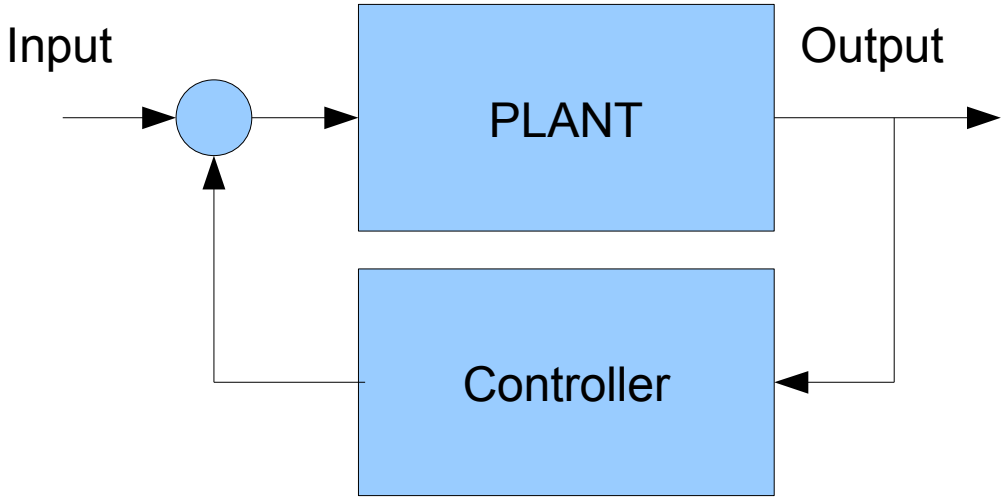
LTI continuous systems: the Paradise

Control system means feedback

NO FEEDBACK : open loop system

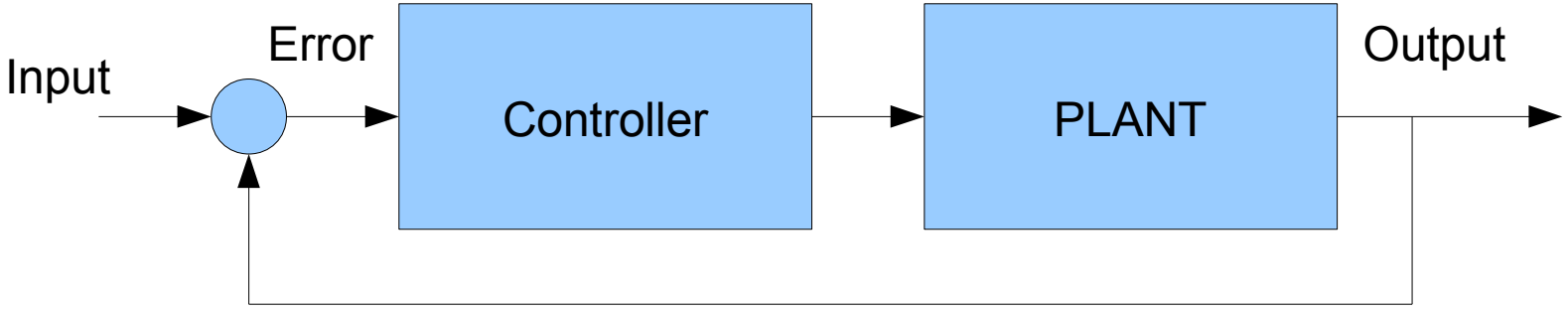


Control system means feedback: standard config.



Controller in the “feedback” path

Controller in the “direct” path



Example of feedback system: the floating apple

- I have an apple at y(0)=y0=1.0m.
- At t=0 I drop the apple, and the apple falls down.
- I'm not satisfied: I'd like to see the apple floating at a reference height (ref=0.5m).
- I need a controller to implement a closed loop feedback system.

To design a controller you need:

- a theory : Newton's gravity law

- a model of the “PLANT”

Open Loop "PLANT" model

The "PLANT" : a free falling apple

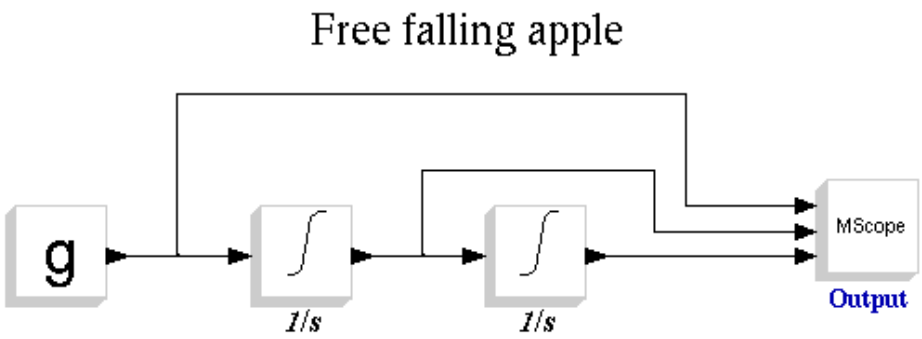
$$F = G\frac{m_a m_T}{r^2}$$

$$F = m\,a \qquad a = \frac{F}{m} \qquad a_a = G\frac{m_T}{r^2};\ g = -9.81\ m/s^2$$

$$v(t) = v_0 + \int a(t)\,dt \qquad y(t) = y_0 + \int v(t)\,dt$$

$$y(t) = y_0 + \frac{1}{2}g\,t^2$$

Open loop PLANT model: Scicos simulation

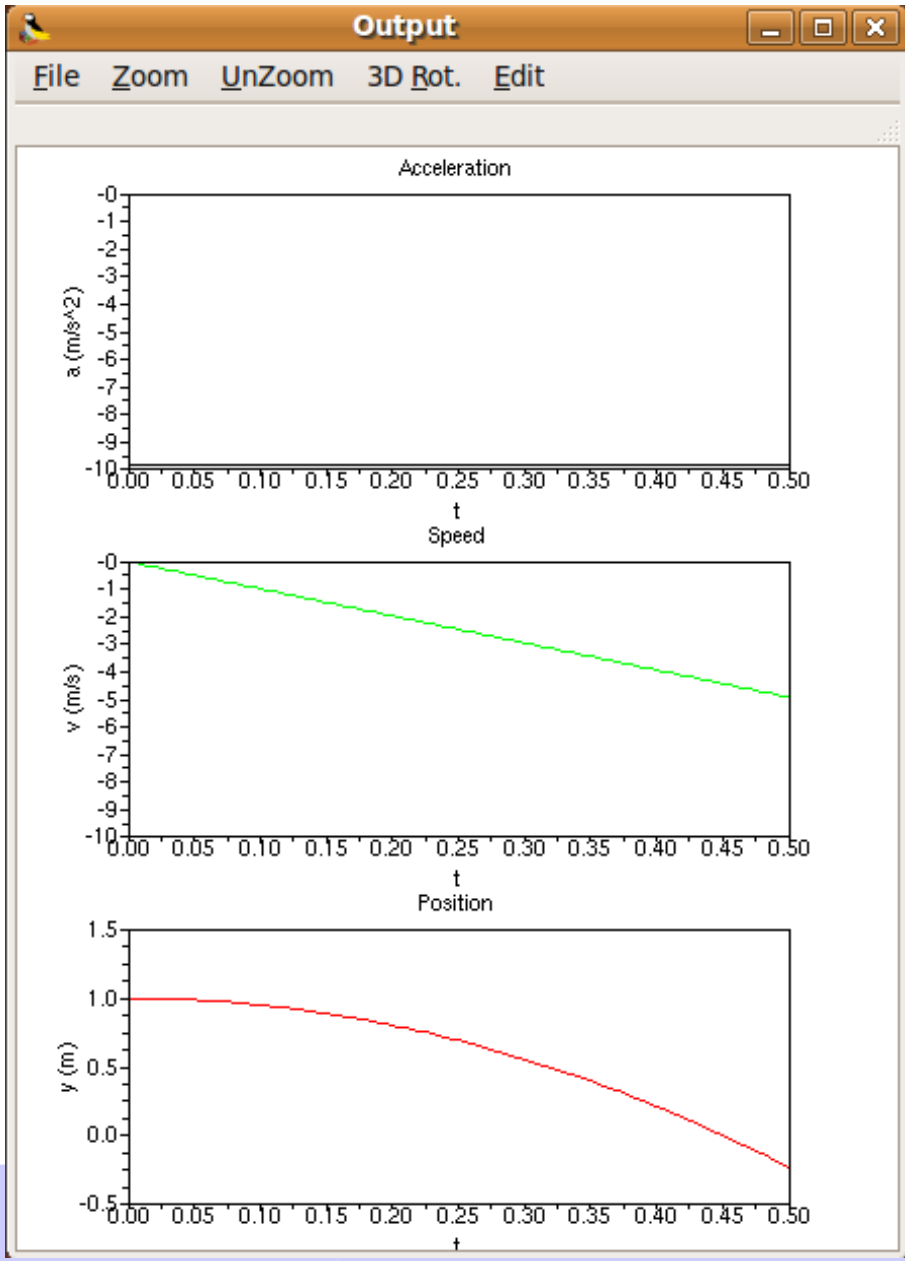


$$v(t) = v_0 + \int a(t)\,dt$$

$$y(t) = y_0 + \int v(t)\,dt$$

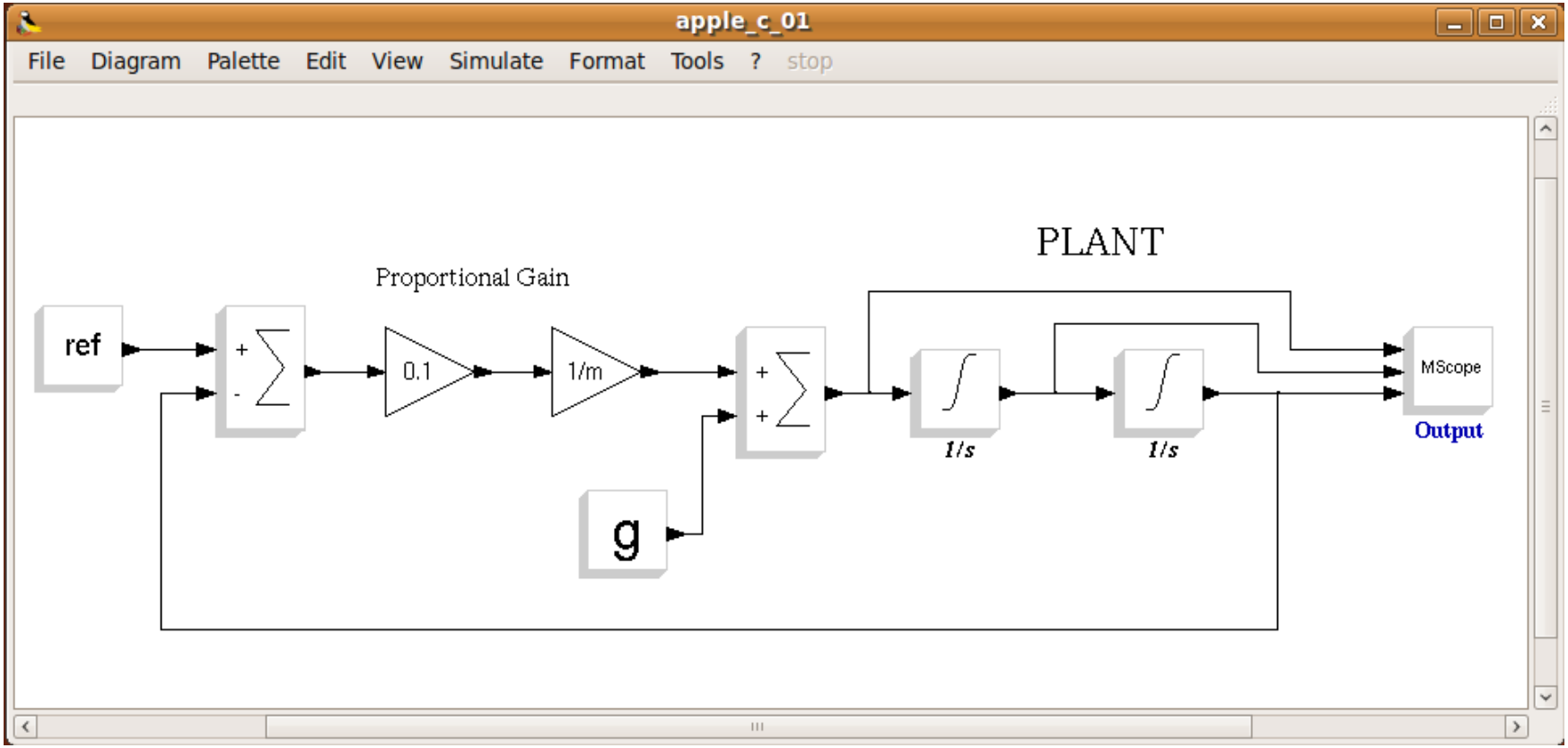
Open loop simulation of the free falling apple.

"apple_ff_01.cos"



Example: the floating apple

First “instinctive” trial-and-error controller (P only)

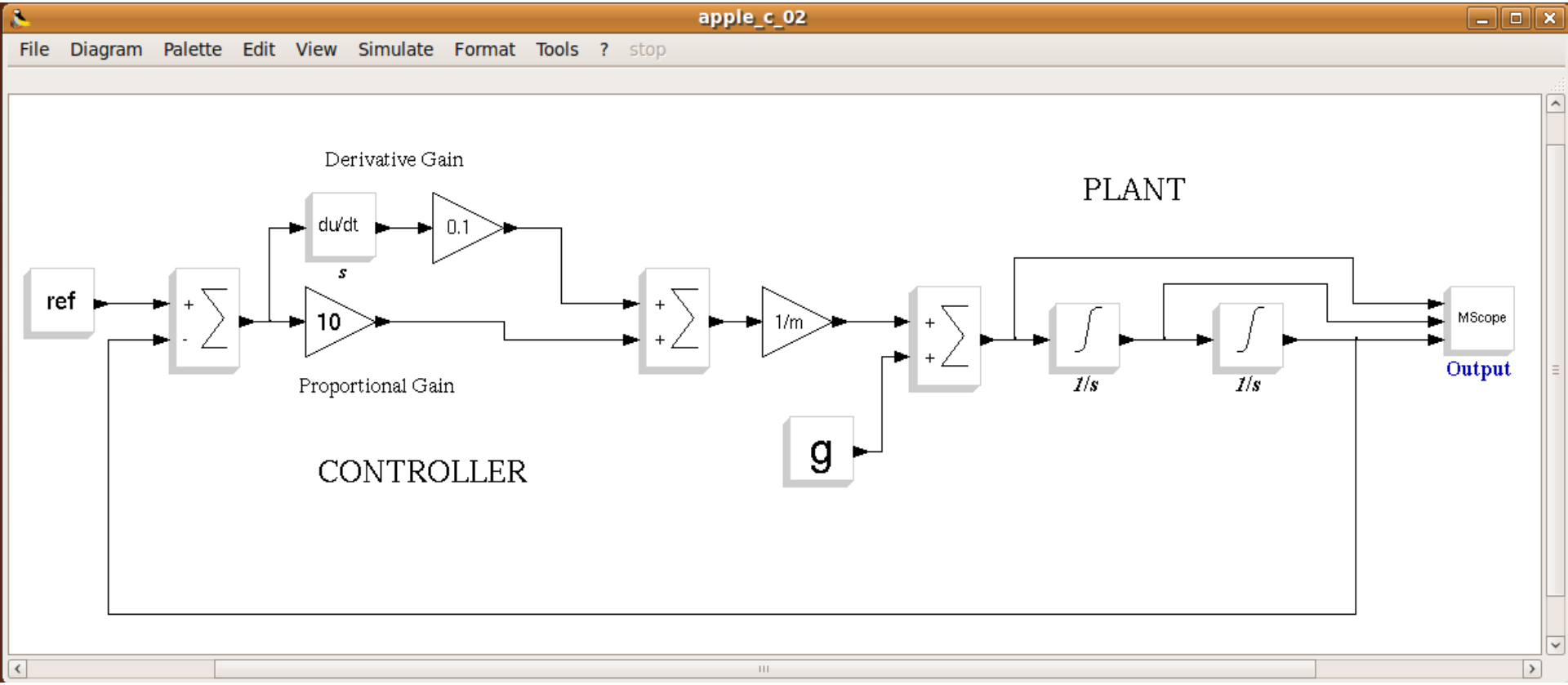


This controller does not work.

“apple_c_01.cos”

Example: the floating apple

Hint: you need to introduce an additional term producing “artificial friction”.



It works !

“apple_c_02.cos”

Example: the floating apple

Conclusions and observations:

“intuitive” design (no theory) may produce catastrophic results

The ubiquitous PID controller may be able to solve most of the common situations.

Not every PLANT can be stabilized using a simple PID

In order to design a suitable controller we need a specific theory

Basic tool for continuous system: the Laplace transform

Solving with “paper and pencil” differential equations is too difficult: we need a better tool.

Laplace transform a differential equation problem in a polynomial problem

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^{\infty} e^{-st} f(t)\, dt.$$

$$s = \sigma + i\omega,$$

F(s) is a complex function of complex variable

Laplace transform for dummies

Write a differential equation

Substitute “d/dt” (derivative) with “s”

Substitute “” (integral) with (1/s)

Compute the transfer function H(s) = N(s) / D(s)

$$H(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0}$$

Examples:

- Electrical circuit (simple RC)
- Mechanical system (mass + friction)

Laplace transform for dummies

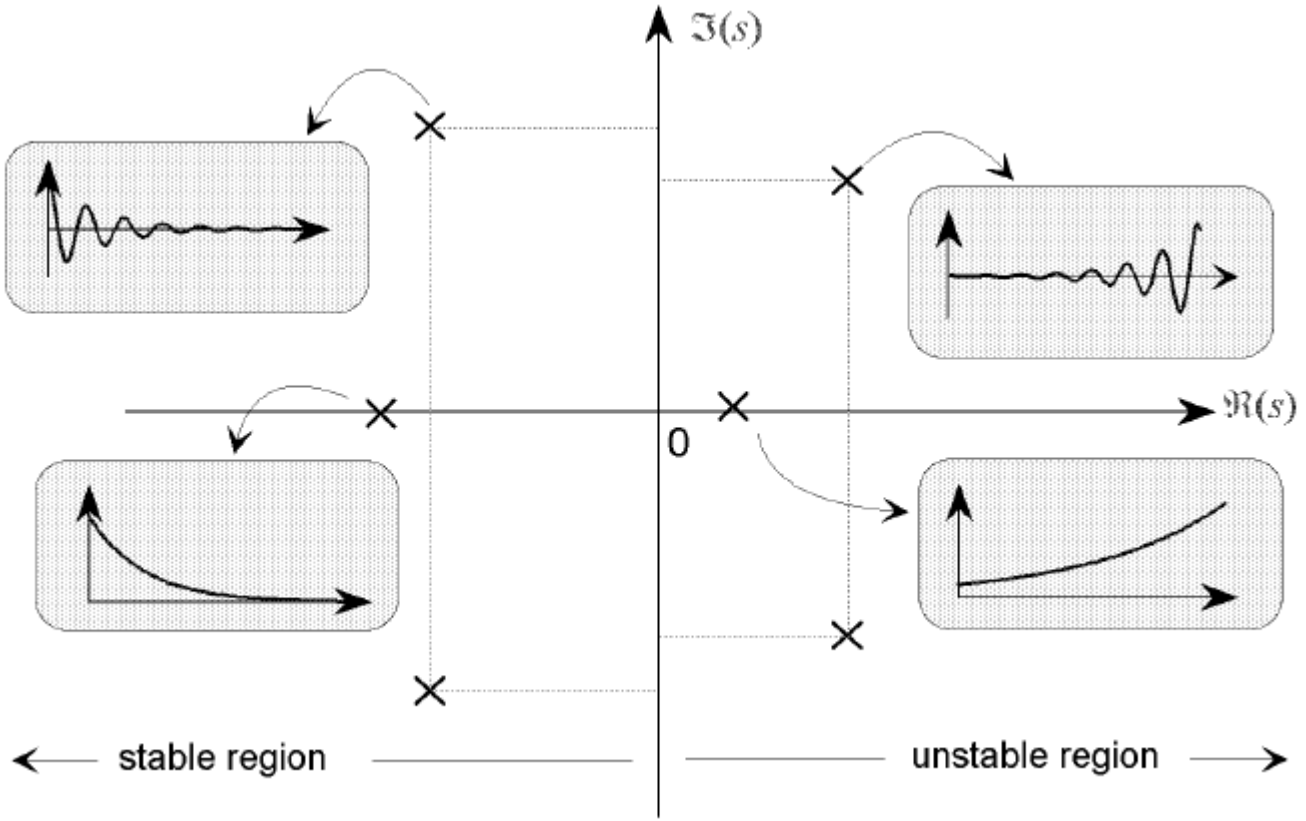
Compute poles “p_i” (D(s) = 0) and zeros “z_i” N(s)=0 and factorize

$$H(s) = \frac{N(s)}{D(s)} = K\frac{(s-z_1)(s-z_2)\dots(s-z_{m-1})(s-z_m)}{(s-p_1)(s-p_2)\dots(s-p_{n-1})(s-p_n)},$$

Forget the zeros for a moment and focus your attention on the poles.

Why are the poles so important?

Poles on “s” plane



The position of the poles is strictly linked with the system response

Poles on “s” plane: the root locus method

Feedback “K” moves the poles on the “s” plane. If you understand the relation between “K” and the position of the poles, you can change the response of the closed loop system.

In our initial example “K” moves the poles, but the poles remain on the (vertical) imaginary axis.

Adding a “zero” (the derivative factor) moves the asymptotic axe of the poles on the stable region.

Fourier transform and frequency response

Fourier transform is a simplified but very useful form of Laplace transform. If you already have the Laplace transfer function H(s), the Fourier transfer function is the same but with $s = j\omega$

Fourier transfer function is used when the input of the system is a single sinusoid because, in this case, the output is another sinusoid with different amplitude and phase, easily calculated from the $F(j\omega)$

Fourier transfer function is very useful because you can consider any periodic input signals as superposition of sinusoids with harmonic frequencies (0, 1, 2 ...n) and with different amplitudes and phases (Fourier transform, FFT, FFTW).

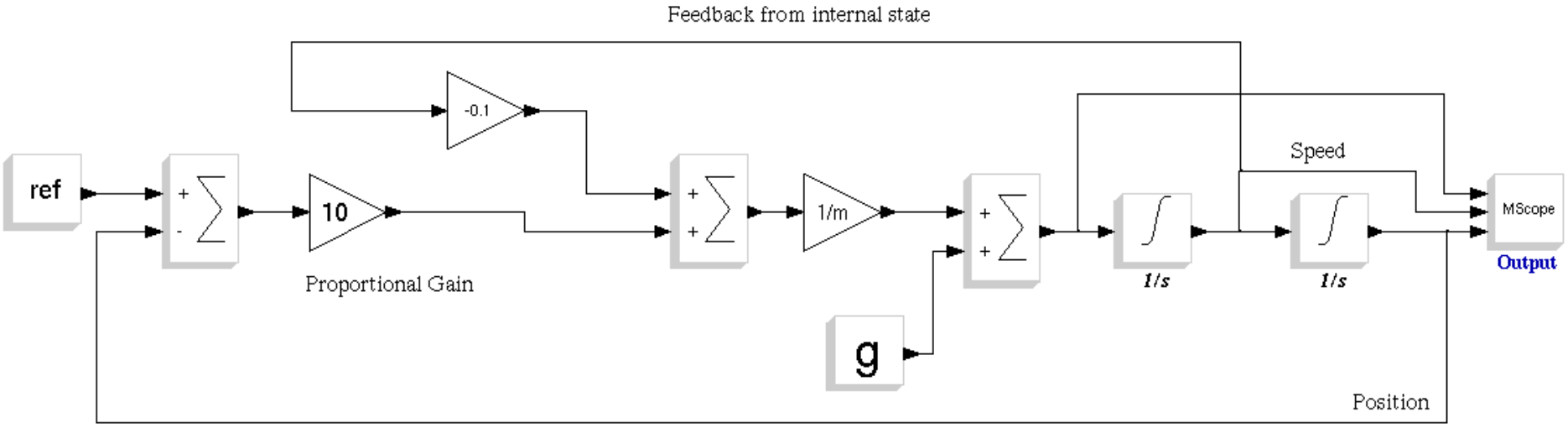
Fourier transform is used also for control system design using Bode, Nyquist and Nichols (Black) diagrams (charts).

Fourier transform could be used for system's identification (parameter measurement).

Gentle introduction to state space control

Do you remember how we have stabilized the free falling apple?

Hint: instead of deriving the position error, suppose you have a sensor measuring the apple speed; use this signal to stabilize the loop.



It works. The response does not change!

"apple_c_ss_01.cos"

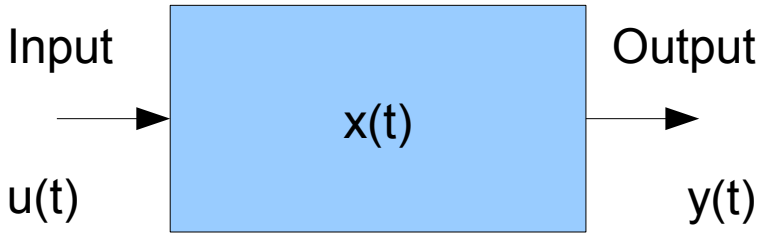
Introduction to state space control

State variable: a function of time associated to an energy stored inside the system; “x” is the “state” (of the system): a vector that includes all the “energy related” variables.

Typical examples of state variables:
Electrical systems: voltage of capacitors and current in the inductor.
Mechanical systems: speed of mass, position of springs

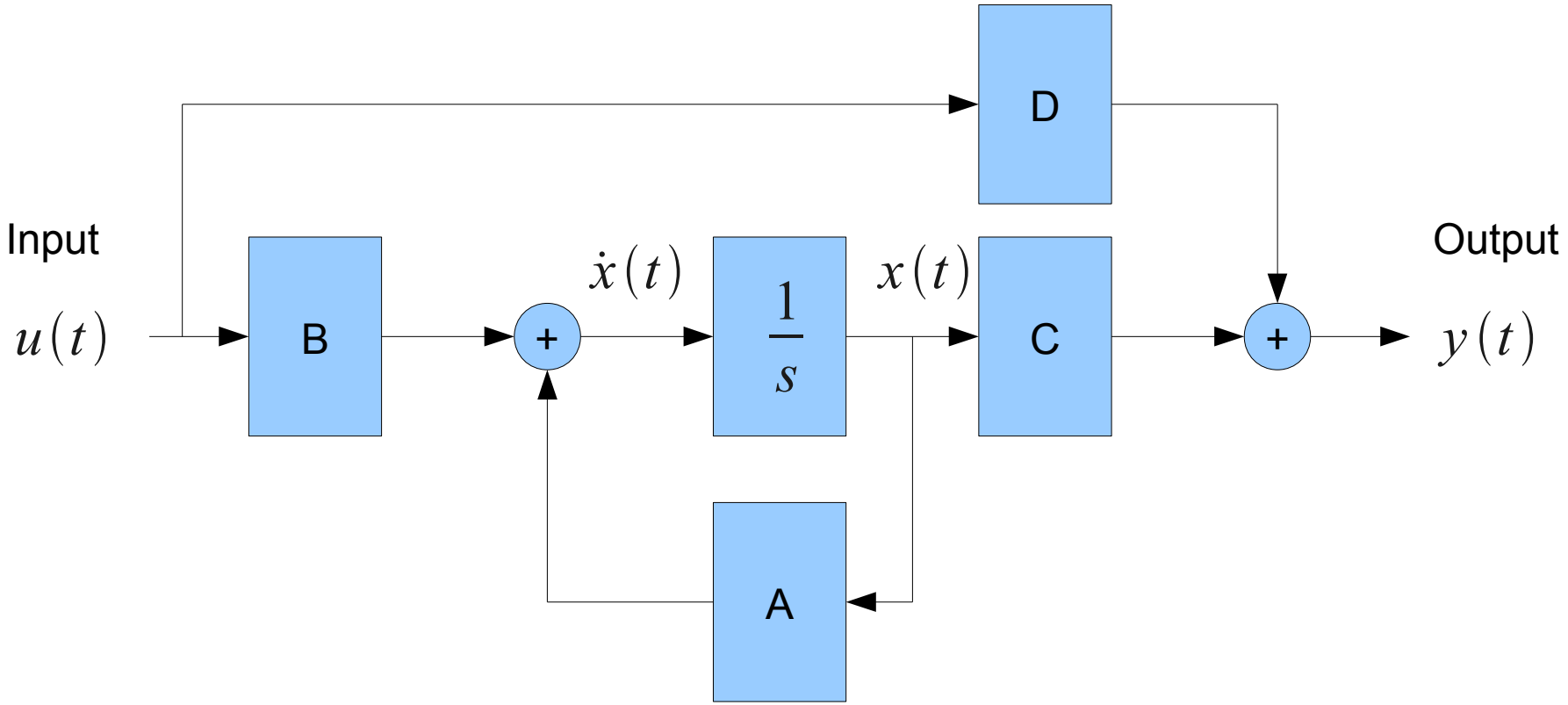
$$\dot{x}(t) = A\,x(t) + B\,u(t)\,;\, internal\ state\ equation$$
$$y(t) = C\,x(t) + D\,u(t)\,;\, output\ equation$$

$$A\,B\,C\,D\ are\ matrixes$$

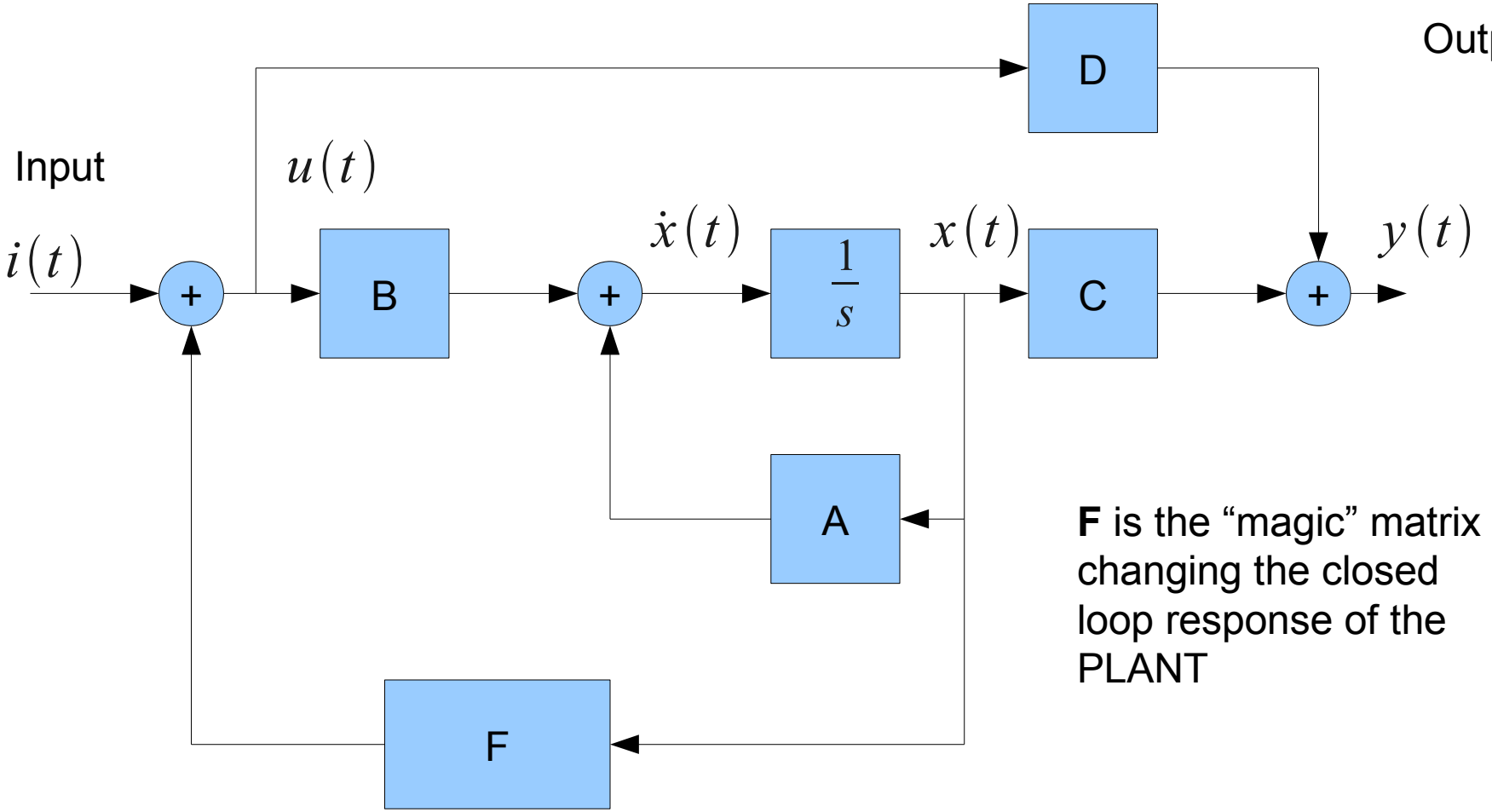


State space representation: the guts

$$\dot{x}(t) = A\,x(t) + B\,u(t)\,;\, internal\ state\ equation$$
$$y(t) = C\,x(t) + D\,u(t)\,;\, output\ equation$$



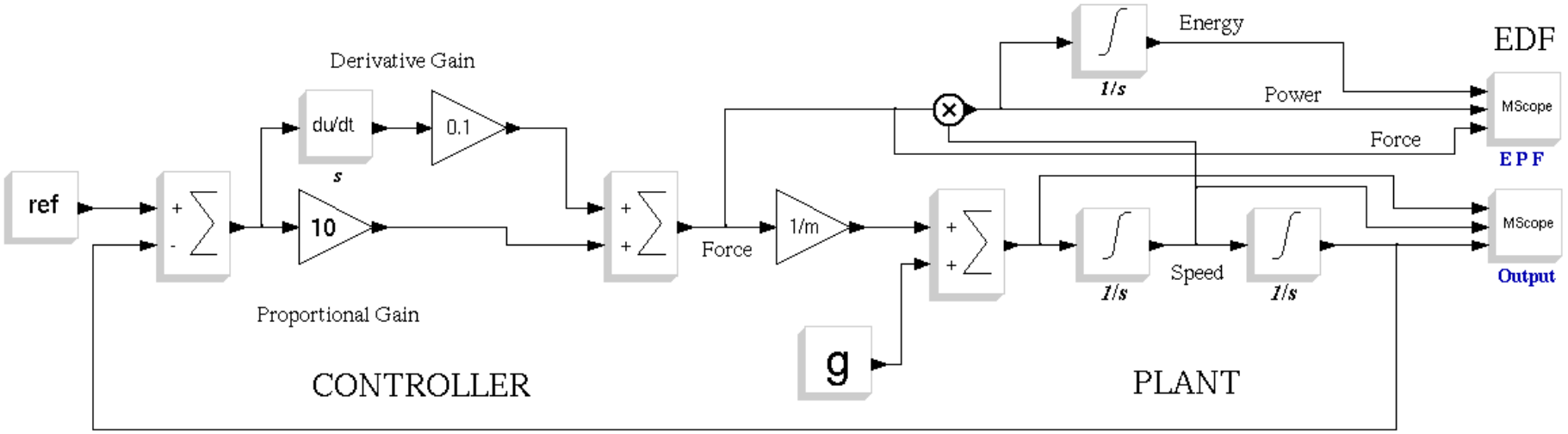
State space feedback control



F is the “magic” matrix changing the closed loop response of the PLANT

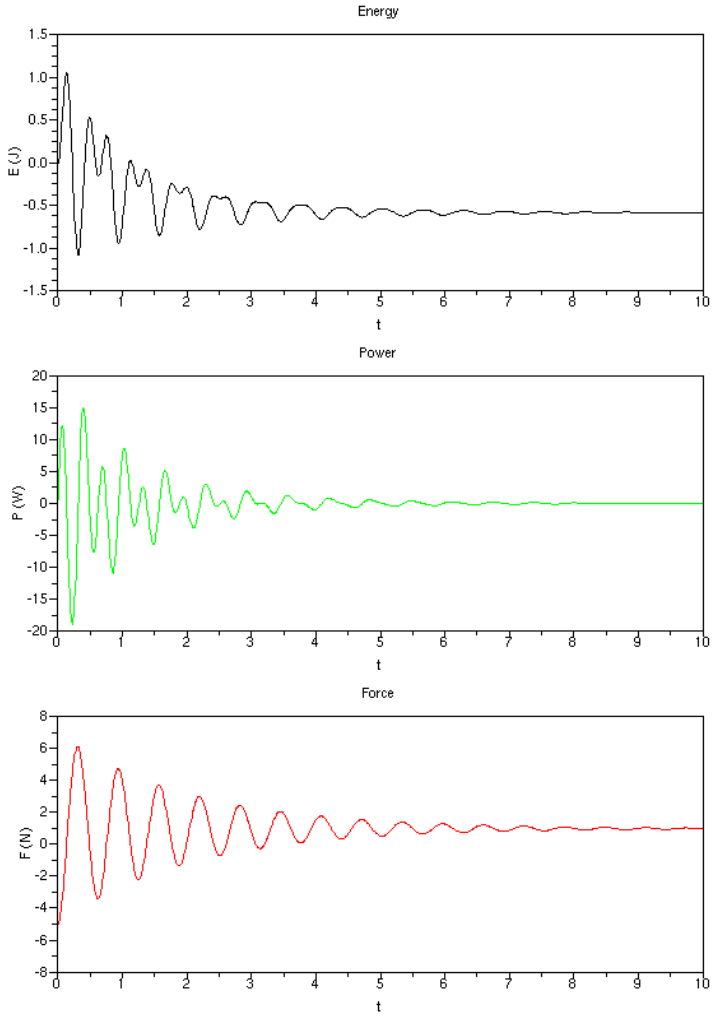
Power and Energy are not Free (ask ENEL, EDF, etc.)

Consider the previous case and measure the power and energy requirement of the actuator (the external “magic” force).



“apple_c_03.cos”

Power and Energy are not Free (ask ENEL)



Observations:

Final energy cost is NEGATIVE: the actuator has “recovered” some energy from the apple. EDF should pay you :-).

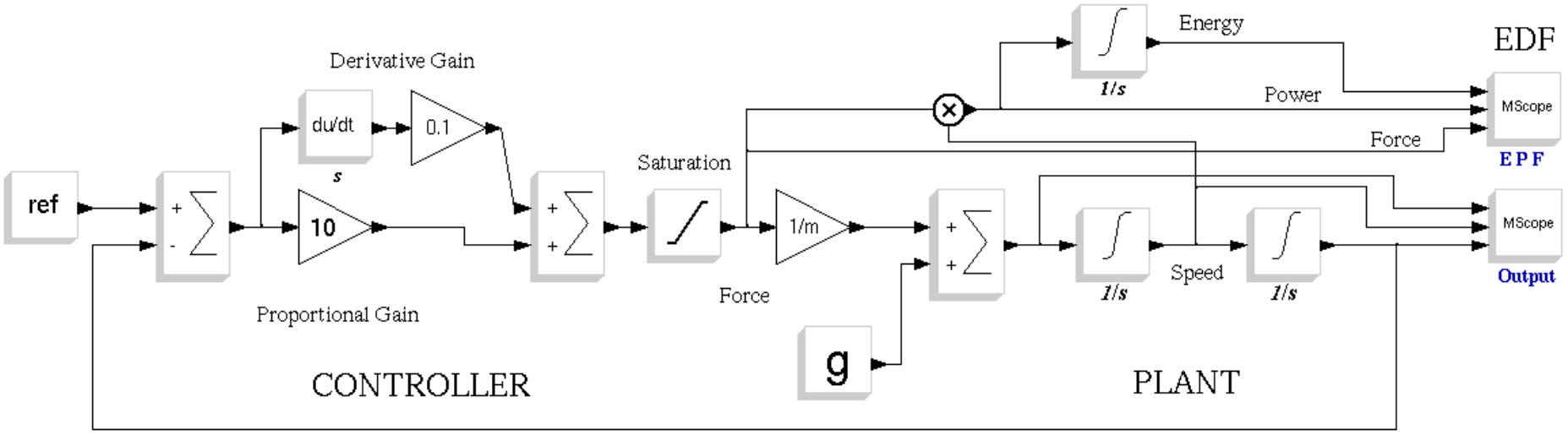
Instantaneous power requirement is around +/- 20 Watt.

Maximum force is around +/- 6 N.

What happens if my actuator is limited to +/- 4 N ?

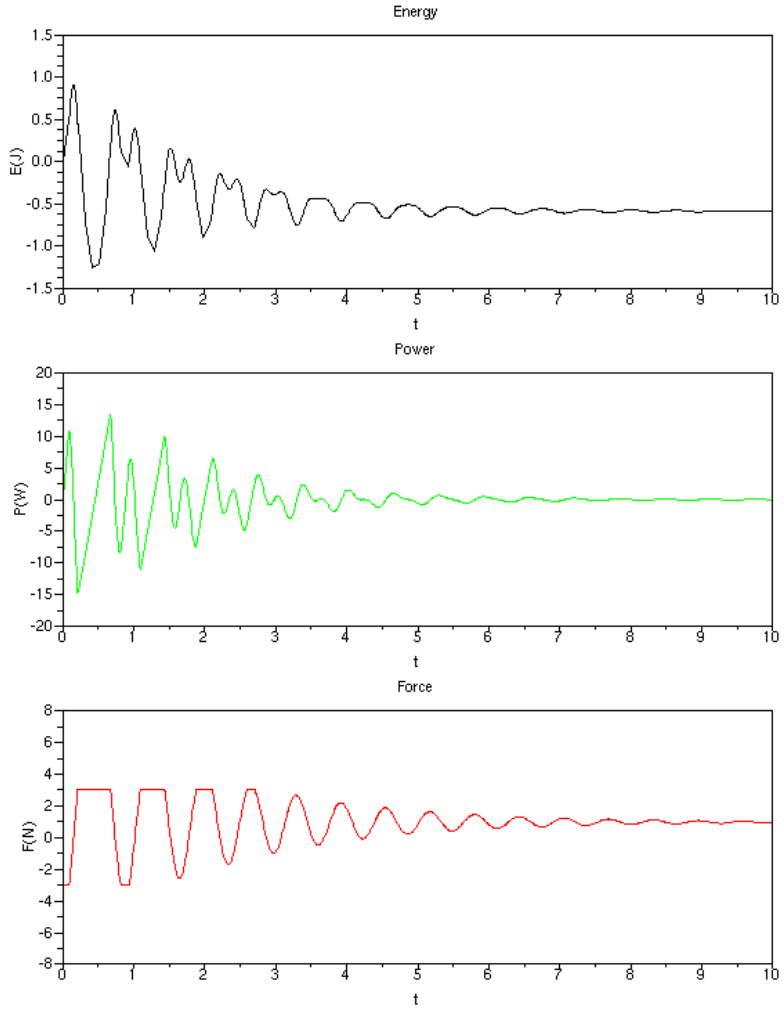
Power and Energy are not Free (ask ENEL)

The same controller but with a force limiter (saturation).



"apple_c_04.cos"

Power and Energy are not Free (ask EDF)



Final energy cost has not changed.

Instantaneous power requirement is a bit lower (+/- 15 against +/- 20 Watt) .

Maximum force is limited to +/- 3N

In practice, every actuator has some physical limits. Powerful actuators are more expensive and have more energy, space and weight requirements.

Loosing the “linear” hypothesis – in this case – is not “catastrophic”: the – linearly – designed controller keeps on working.

Introduction to state space OPTIMAL control

Taking into consideration the previous example, can I design the controller in such a way that the actuator is never pushed above its limits ?

Yes: you need to design a state space optimal controller.

What should you do if you have no access to the complete internal state of the PLANT or you can't (in order to limit costs or weight) install a full set of sensors?

You can re-create the full internal state of the PLANT with an OBSERVER if :
(a) you have a “good enough” model of the plant
(b) the PLANT is not “bastard” (the PLANT must be observable and controllable)
(c) you have sufficient computing power.

Discrete systems

Linear, discrete and time invariant systems

Discrete: all the internal and external signals are sampled in time. They are unequivocally defined only for “t = k Ts”.

Linear: if u1, u2 are two inputs and y1 and y2 are the two corresponding outputs, the input u = u1 + u2, produces the output y = y1+y2

Time invariant: the response of the systems does not vary in time (system parameters are time invariant, no “ageing effect”).

LTI discrete systems: the Purgatory

Why do you need discrete (sampled) systems?

The time sampling signal technology was born (1930, Bell) to improve the communication capability of telephone networks (more conversations on the same wired or radio connection).

No one suggested – at the time - sampled (discrete) feedback systems. Sampled systems were considered an “aberration”.

Why sampling a signal when:

- the sensors produce continuous signals
- the actuators need continuous signals
- all the available computing blocks were based on analog (mechanical, hydraulical or electrical) continuous time technologies.

- 1939 -

Q. Why do you need discrete (sampled) systems?
A. Because there was a war.

If you need to point a cannon to an aicraft you can use a RADAR as sensor BUT all the RADAR signals (distance, bearing, etc.) are intrinsically "sampled" (discrete in time).

- 1945 -

Q. Why do you need discrete (sampled) systems?
A. Because there is the digital computer

Modern computers use digital technology, in which all the signals are sampled in time (discrete in time, sampling) and in amplitude (discrete in value, quantization).

Q. Why do you need discrete (sampled) systems?
A. Because there is a PC with ScicosLab

All the software simulators - running on a PC - model continuous systems using equivalent discrete systems.

Three questions:

How is it possible to build a discrete equivalent of a continuous system?

It is (the discrete equivalent) just an approximation of the continuous one or a completely different “thing”?

What are the hypothesis and the limits of this “emulation”?

Free falling apple discrete simulation

Many years ago somebody (Euler) has shown how it is possible to transform a differential equation into a finite difference equation (sampling in time, from continuous to discrete equivalent).

$v(t) = v_0 + \int a(t)\,dt$; integral (differential) equation

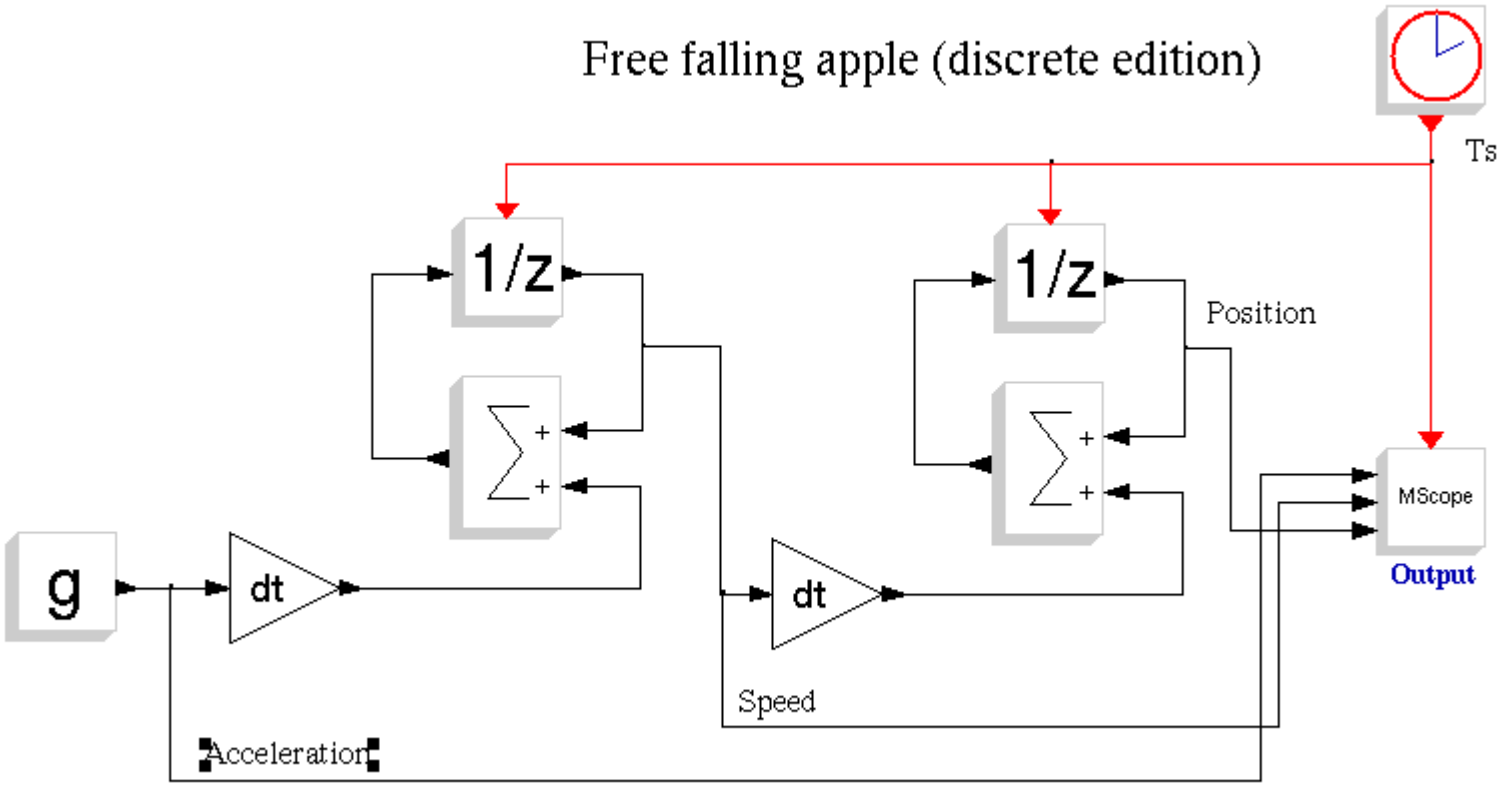
$v_{k+1} = v_k + a_k\, delta$; difference equation

$v_k = v(kT_s)$; time sampling

$delta = T_s$; discrete time = sampling time

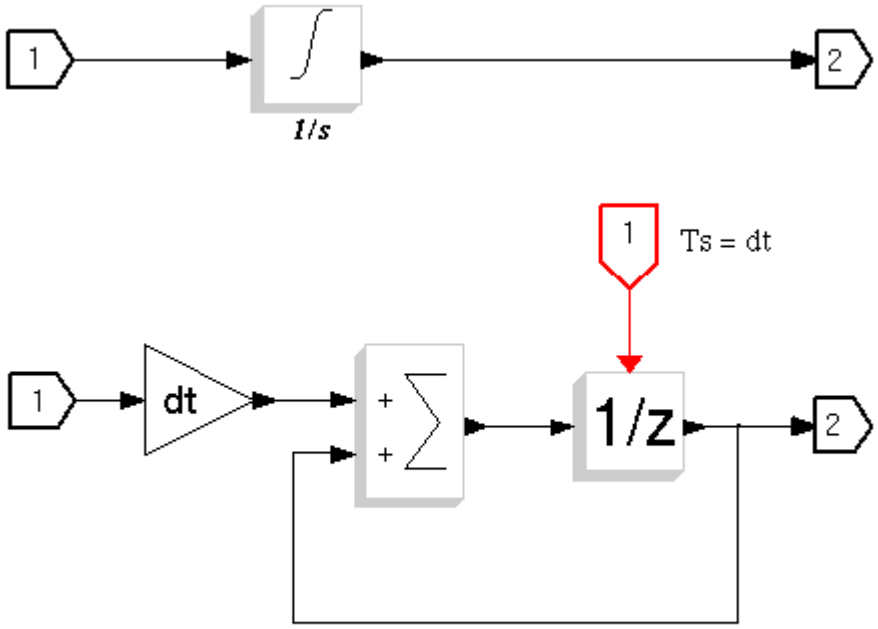
Free falling apple discrete Scicos simulation

Open discussion: “How have you built this nice diagram?”



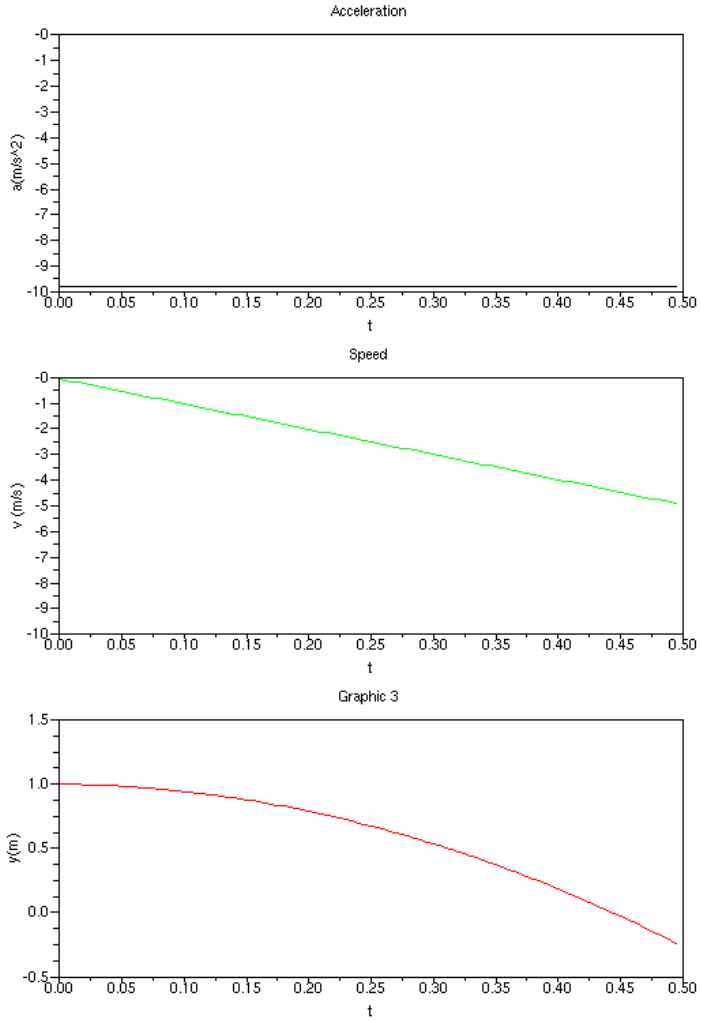
“apple_ffde_02.cos”

The discrete integrator (according to Euler)



1957 stamp of the former Soviet Union commemorating the 250th birthday of Euler. The text says: 250 years from the birth of the great mathematician, academician Leonhard Euler.

Free falling apple discrete Scicos simulation



Observations and open discussion:

The same results.

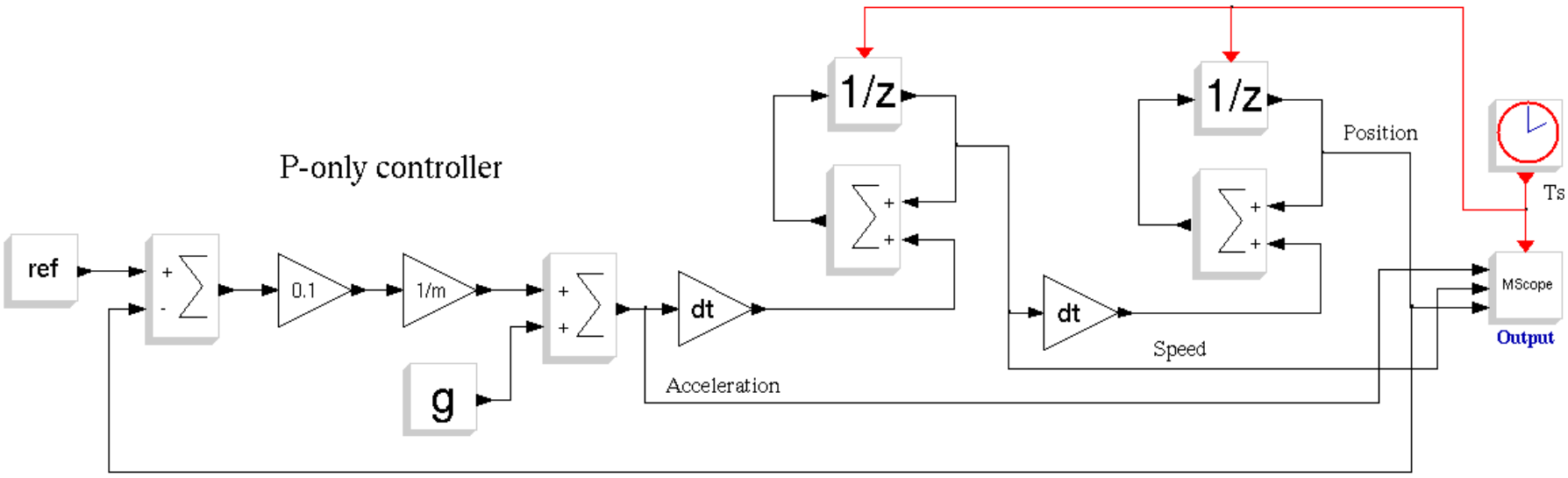
How have you calculated Ts ?

What happens if you change Ts ?

Can you show the discrete nature of the system?

Does Scicos work using this technique ?

First “instinctive” P-only controller

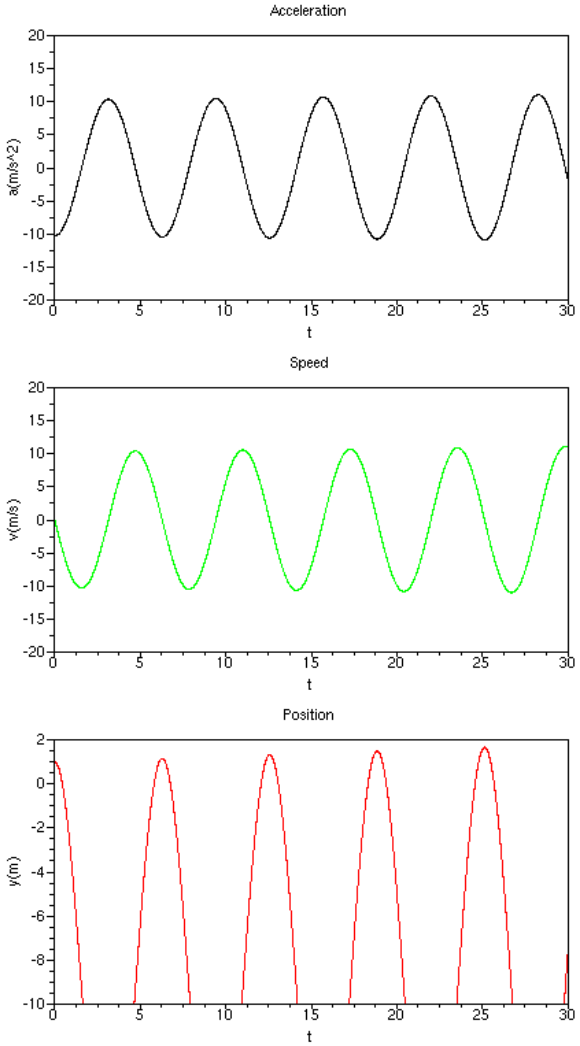


“apple_cde_02.cos”

Stability is not achieved !

“apple_cde_01.cos” is “algrebraic loop”

P-only controller



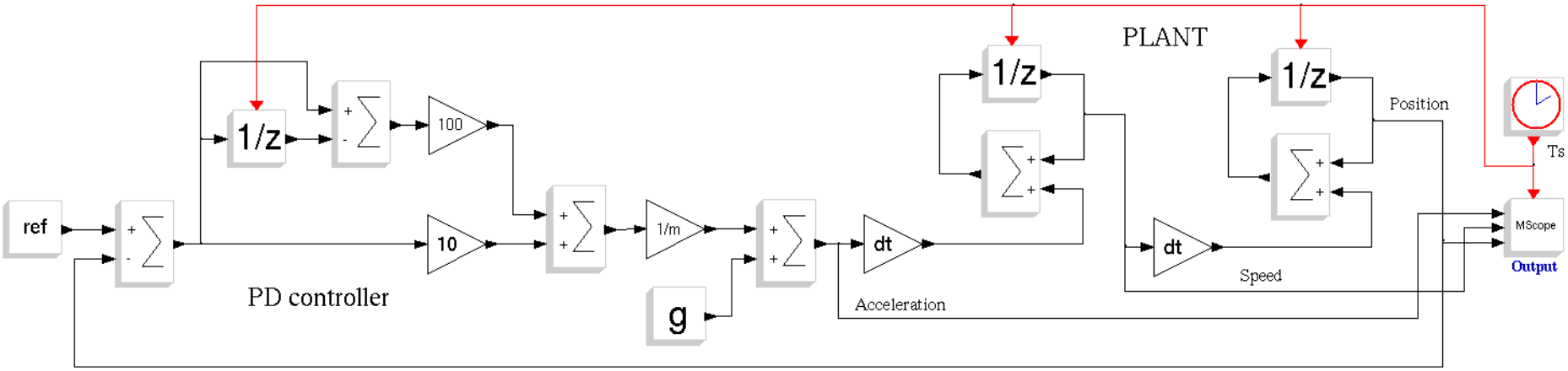
Observations and open discussion:

The same results.

What happens if I change Ts ?

Can you show the discrete nature of the system?

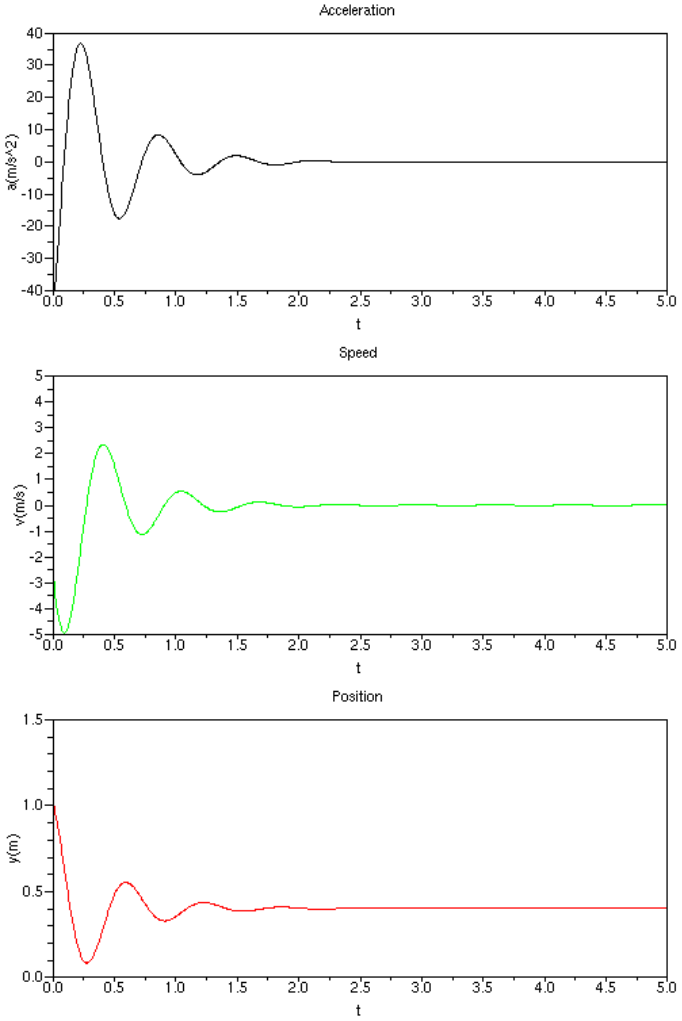
PD controller



"apple_cde_03.cos"

It works !

PD controller



Observations and open discussion:

The same results.

What happens if I change Ts ?

Can you show the discrete nature of the system?

Basic tool for discrete systems: Z transform

Solving by “pencil and paper” discrete equations is too difficult: you need a better tool.

Z transforms a difference equation problem in a polynomial problem

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=0}^{\infty} x[n] z^{-n}.$$

X(z) is a complex function of complex variable

$$z = Ae^{j\varphi} = A(\cos\varphi + j\sin\varphi)$$

Z transform for dummies

Z transform produces N(z)/D(z) polynomial transfer functions.

The dynamic of the system is determined by the position of the poles

$$D(z_i)=0$$

on the “z” complex plane.

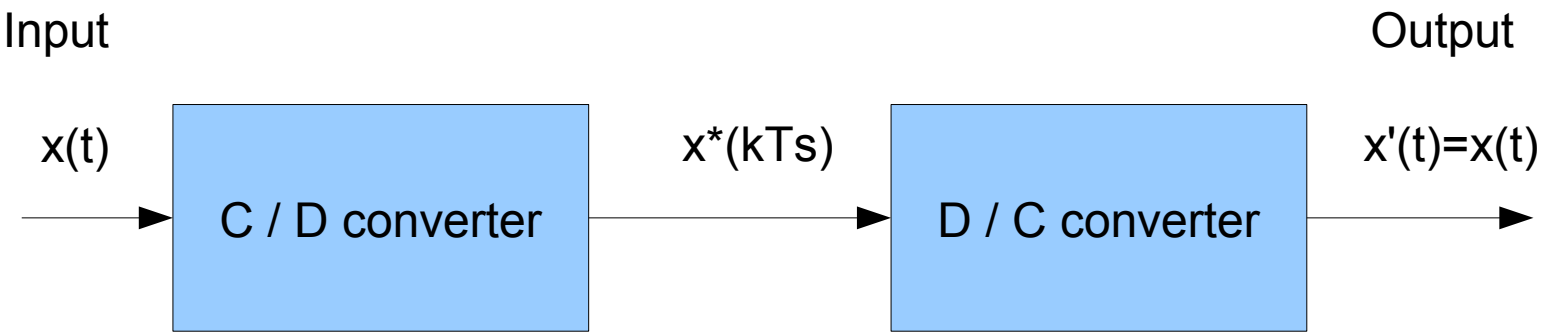
Z transform does not show an explicit dependency by “Ts”, but the poles position is strongly “Ts” dependent.

Before considering the link between “Z” and “L” transforms and the considerations about the system stability, you need to ask yourself the question: “How can I choose Ts?”

Nyquist-Shannon sampling theorem

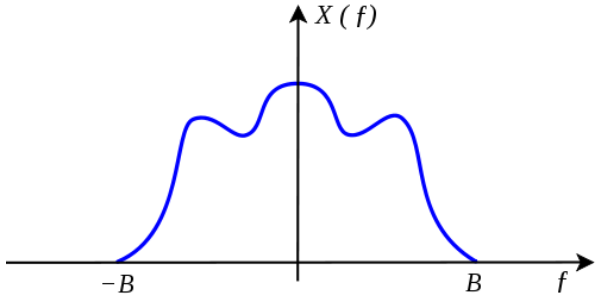
I want to realize an apparently trivial application:

I have a continuous signal “x(t)” and I want to sample it, creating a new, discrete in time, signal “x*(kTs)”. Then, I want to use this sampled signal to recreate EXACTLY (zero error, no approximations) the original continuous signal “x(t)”.

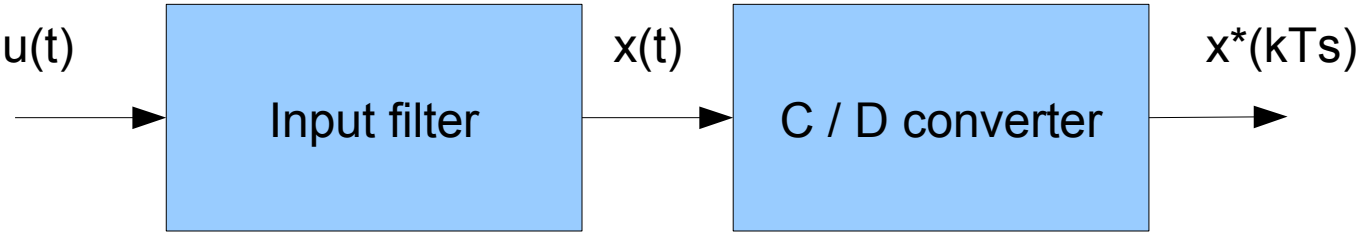


Nyquist-Shannon sampling theorem

First step: the input signal “x(t)” must be strictly band limited.
In other words, it does not have any “energy” beyond a specific frequency B (input signal bandwidth).



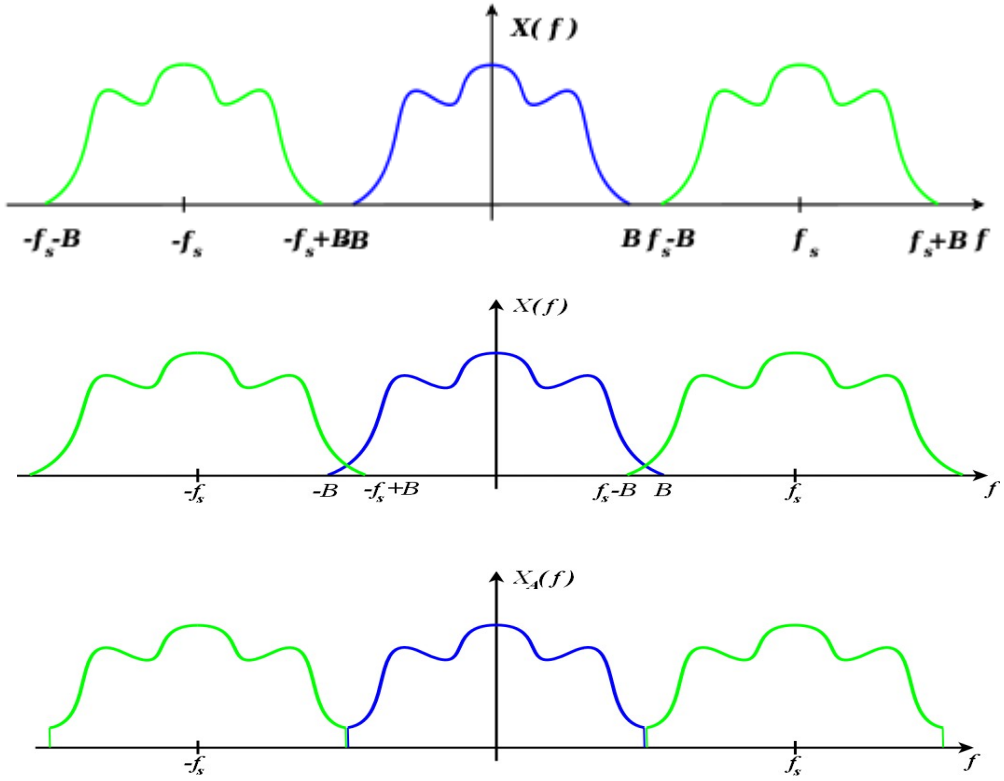
In practice, all the real physical signals have a limited bandwidth.
In case of doubt, you can add an input filter, “good enough” to suppress any undesired input frequency component beyond B.



The input filter (also called “anti aliasing filter”) is optional. In some applications (e.g. sampling scopes) is not used.

Nyquist-Shannon sampling theorem

Second step: the input signal “x(t)” must be sampled with a stable, periodic signal “s” with a frequency Fs = 1/Ts where Fs must be at least 2B. In other words, the sampling frequency must be – at least – twice the input signal bandwidth.



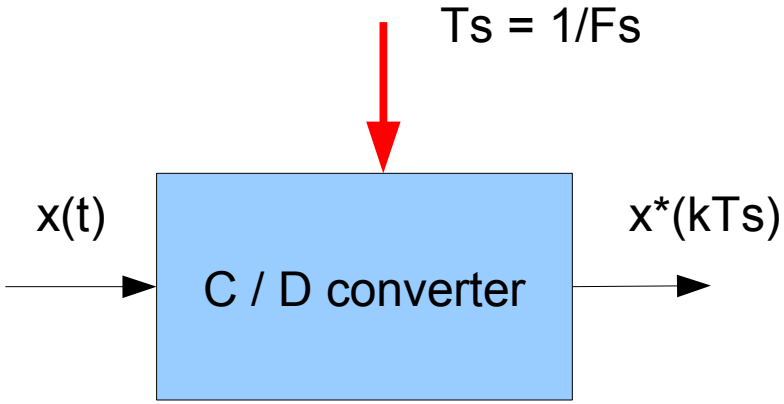
Good.
Sufficient sampling frequency. No spectrum overlapping, no aliasing.

Bad.
Insufficient sampling frequency. The spectrum of the sampled data signal is altered (aliasing).

Nyquist-Shannon sampling theorem

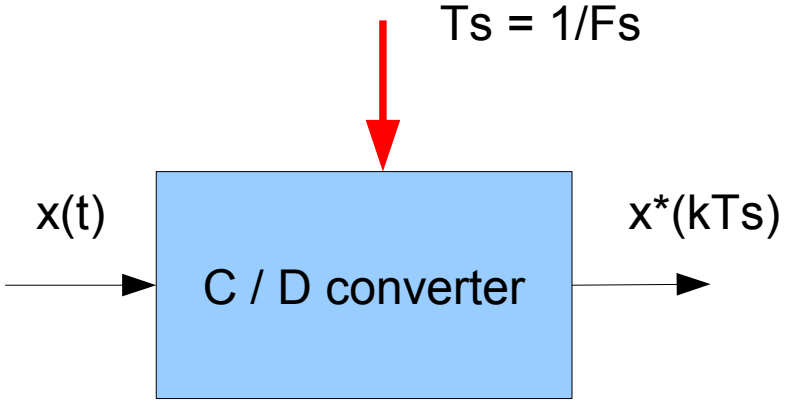
Second step (practical realization): building a good C/D converter - today - is relatively easy (it is just a very fast switch). The critical issue is the Fs stability.

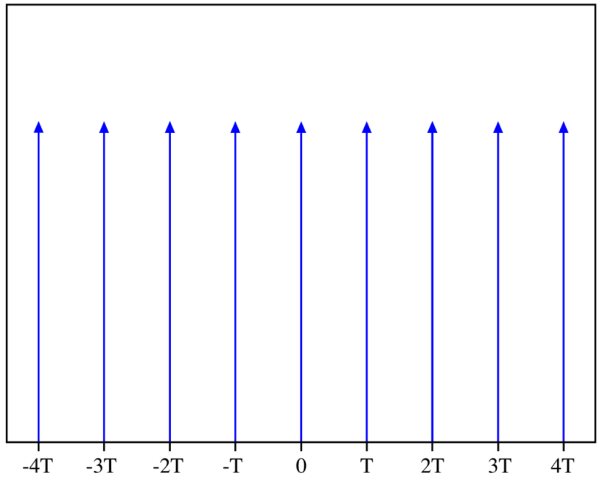
Real C/D converters (called A/D Analog to Digital converters) make also a quantization in amplitude.



Nyquist-Shannon sampling theorem

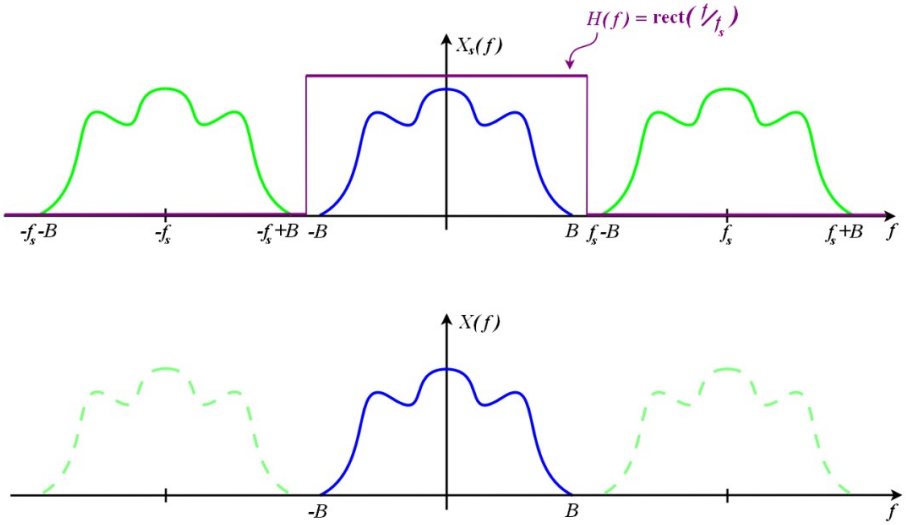
Third step: the sampled signal “x*(kTs)” must be considered as a sequence of Dirac impulse of variable amplitude. In practice, this is not a big issue because we need only to store a sequence of values.



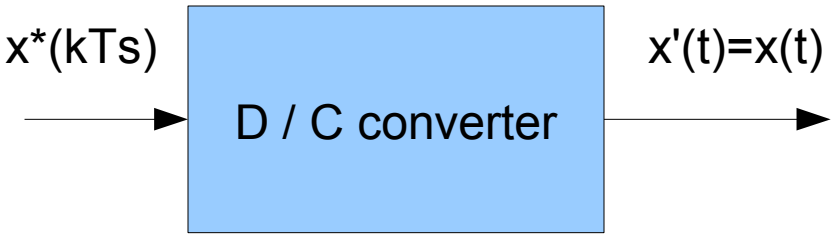


Nyquist-Shannon sampling theorem

Fourth step (the reconstruction): in theory, if we apply the sequence of delta function to the input of a “brick-wall” filter, the output will be equal to the original signal. The “brick-wall” filter removes the images (alias).



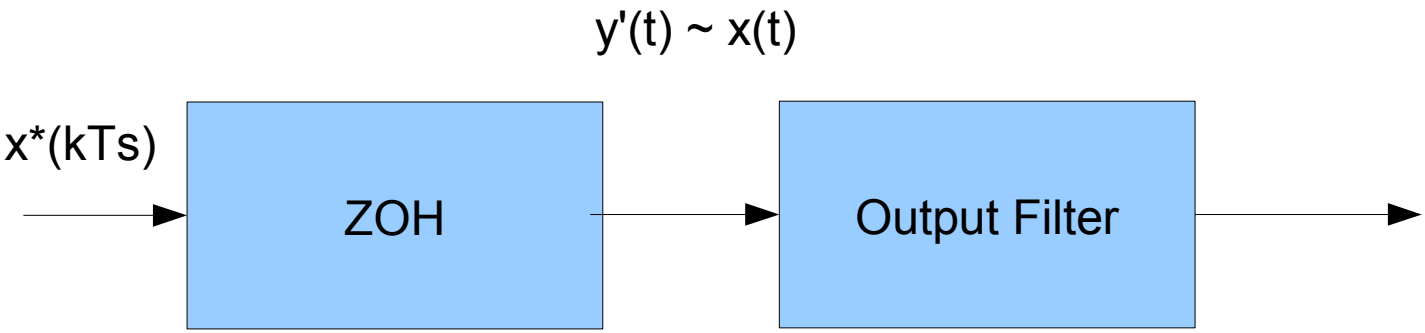
Ideal “brick-wall” filter H(f)



Nyquist-Shannon sampling theorem

Fourth step (practical realization): Dirac impulses are impossible to realize because they have infinite amplitude and bandwidth. Ideal “brick-wall” filters are impossible to realize, too.
In practice, the Dirac impulses are substituted by a zero-order-hold approximation (the familiar staircase reconstruction) and the output filter is designed to approximate the brick-wall and compensate the ZHO distortion.

In some applications (e.g. control system) the output filter is not used.



The sampling theorem for control systems

Regarding “Ts” choice, we can take for granted that we need to sample with a frequency – at least – twice the input signal bandwidth.

For closed loop control systems, the game should be played in a different way, because it is up to us decide the bandwidth of the closed loop system in function of the specifications. We need also to consider the approximations of the practical realization, otherwise we would risk to produce an unstable or marginally stable closed loop system.

A good “rule of thumb” is to choose a sampling frequency from 5 to 20 times the closed loop system bandwidth.

This rule is also known as “at least ten samples during system's closed-loop transient response”.

Obviously, this design rule makes Shannon and Nyquist happy :-).

Do not push too hard on Ts !

Beware! You must avoid the temptation to push too much the sampling frequency.

If you double the sampling frequency, you double the required computational requirement, so the power consumption, so the weight, so the cost....

If you push too much the sampling frequency, you aggravate the problems deriving from finite mathematical precision. You risk to spend time to accumulate “zero” quantity or differentiate equal values with very small denominators.

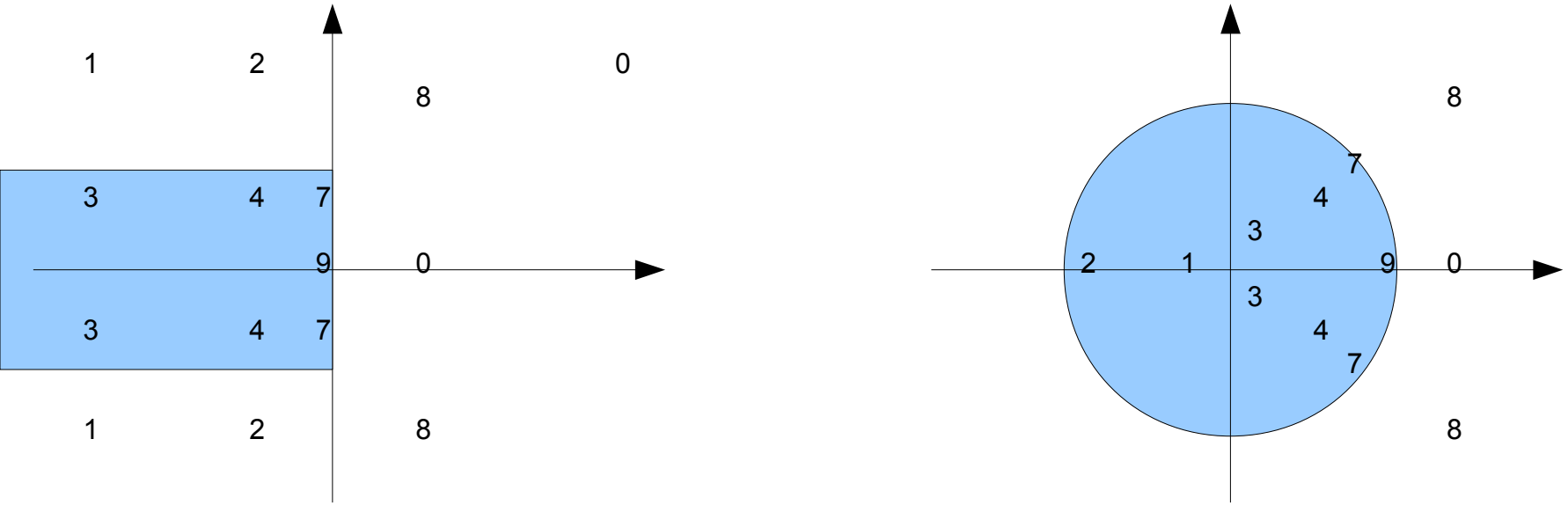
Finally, you become too much sensitive to numerical (finite precision) and signal (real signal) noises.

When it is enough it is enough.

L and Z transforms intimate relationship

Think back to our original problem to create a discrete system that “emulates” a continuous one. Let us suppose that we have already designed our complete system, so we know where the poles are in the “s” (Laplace) plane.

$$s=\sigma+j\omega \quad ; \quad z=e^{sT_s}$$



From L to Z transform

The previous equation is perfect from a theoretical standpoint but does not resolve our problem: transform an H(s) function in a H'(z) equivalent function.

The Tustin transformation can transform the continuous compensator to the respective digital compensator. The digital compensator will achieve an output which approaches the output of its respective analog controller as the sampling interval is decreased.

$$s = \frac{2(z-1)}{T_s(z+1)}$$

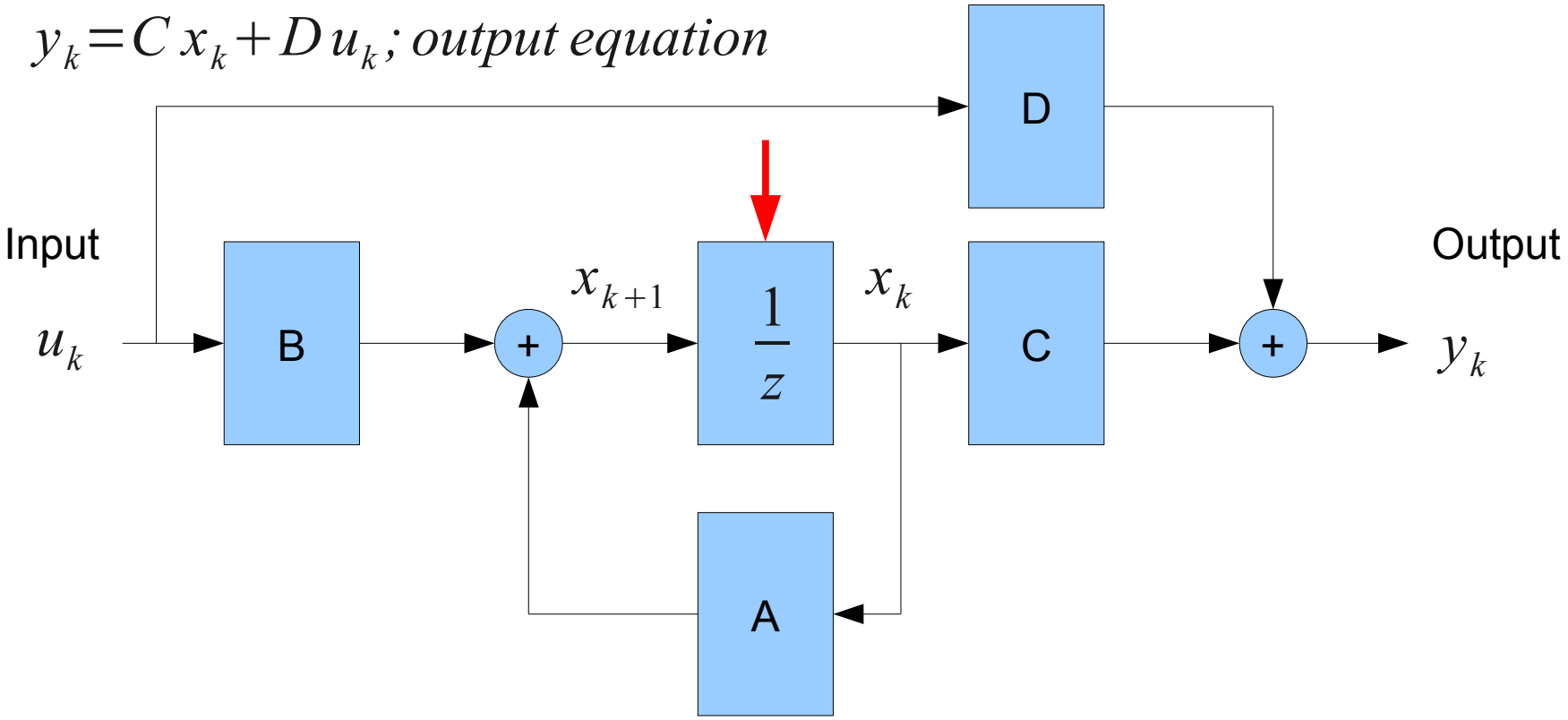
Tustin is just one of the many “approximate” transformation. Notice the direct dependency from Ts.

Discrete state space control

The state space representation is valid in the discrete as in the continuous

$$x_{k+1} = A\,x_k + B\,u_k\,;\ internal\ state\ update\ equation$$

$$y_k = C\,x_k + D\,u_k\,;\ output\ equation$$



Hybrid systems

Linear, hybrid and time invariant systems

Hybrid: internal and external signals are a mix of sampled and continuous time signals.

Linear: if u1, u2 are two inputs and y1 and y2 are the two corresponding outputs, the input u = u1 + u2, produces the output y = y1+y2

Time invariant: the response of the systems does not vary in time (system parameters are time invariant, no “ageing effect”).

LTI hybrid systems: welcome to Inferno

Why do you have fallen into hybrid systems?

Because - most of the time - the PLANT is a continuous time system and the CONTROLLER is realized with digital technology (discrete time system).

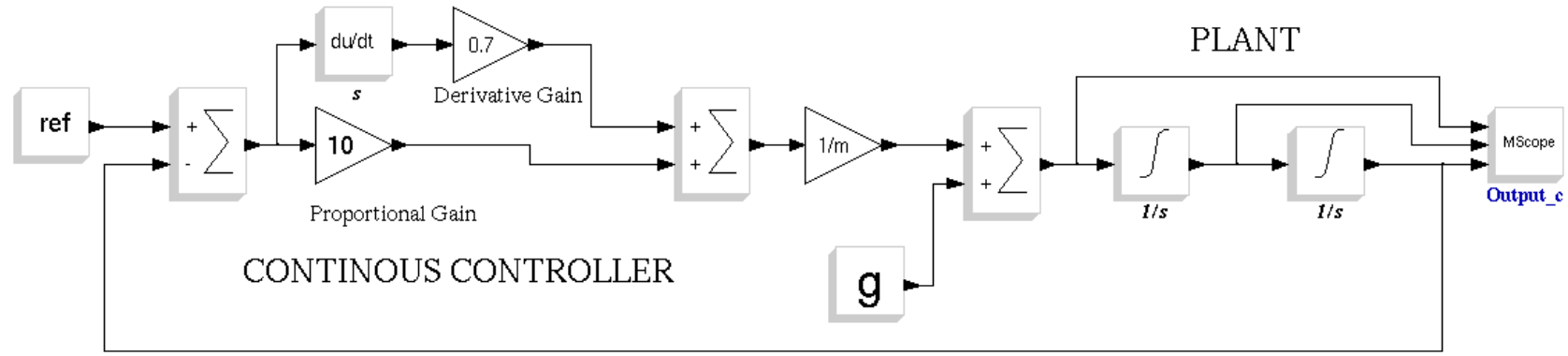
Why digital technology? Because 1000 low quality transistors are cheaper than 1 high quality device. Digital controllers are becoming less expensive than their analog equivalents. Moreover, digital control offers capability well beyond the usual analog PID(networking, supervision, diagnostic, etc.).

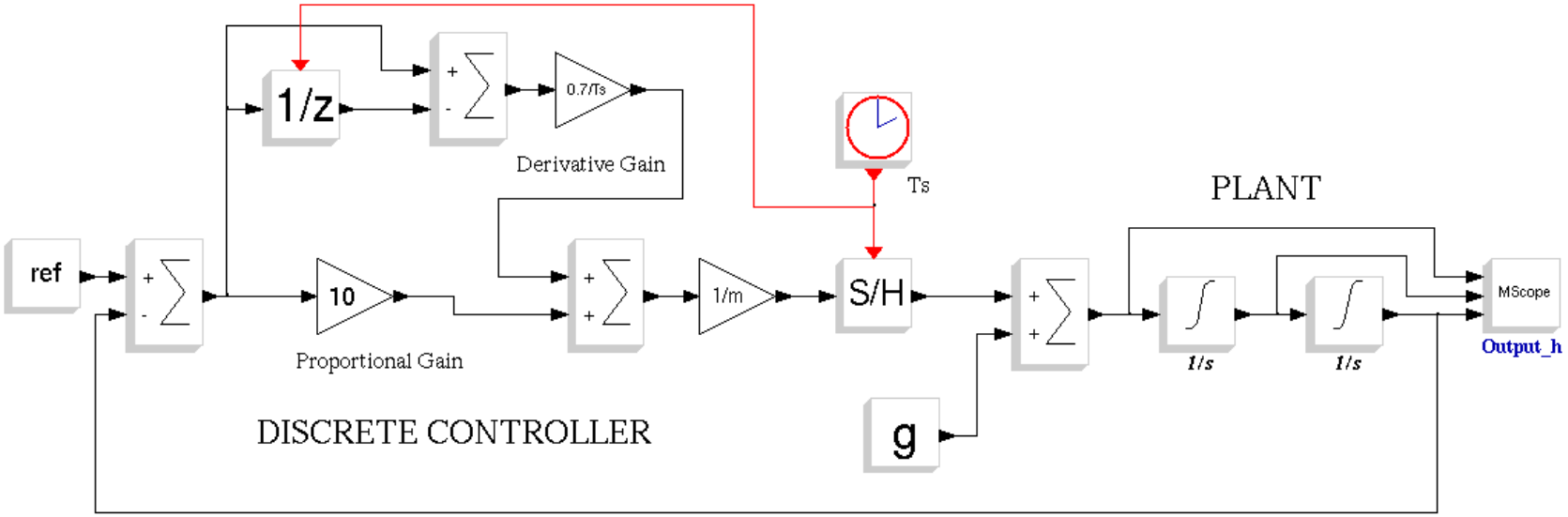
Digital control is a perfect solution for an Internet connected world.

Where is the weak point?

Software

Continuous vs Discrete controller





Continuous vs Discrete controller

Observations and open discussion:

It works but the response is a bit different …

Where is the input sampler ?

Where are the input and the output filters ?

Why is the D gain different and the P gain the same ?

What happens if you change Ts ?

Is digital control cost effective also for very simple systems?

Is digital control “safe” ?

How to design discrete controllers for continuous plant

Old way (design in “s” space):

- You build a model for the PLANT
- You design a controller using continuous time tools (Laplace, root locus, Bode)
- You choose “Ts”
- You transform the continuous controller in a discrete one (Tustin, etc.)

New way (design in “z” space):

- You build a model for the PLANT
- You choose “Ts”
- You transform the continuous plant in a discrete equivalent
- You design a controller using discrete time methods (deadbeat, pole placement, etc.)

Plus and minus of the two design methods

Old way advantages:
- You can continue to use familiar tools and methods (e.g. Bode plots)
- You can postpone the decision about “Ts” (critical decision)

Disadvantages:
- No better performances compared to the “analog” design.

New way advantages:
- You can use high quality transformations to create a discrete equivalent of the PLANT. This improve the design of the controller without additional costs.
- You can use control topology too expensive for analog (c.t.) implementation

Disadvantages:
- You need to choose Ts early (critical decision)
- You need to master discrete design techniques

Do not push too hard on Ts !

Beware! You must avoid the temptation to push too much the sampling frequency.

If you double the sampling frequency, you double the required computational requirement, so the power consumption, so the weight, so the cost....

If you push too much the sampling frequency, you aggravate the problems that derive from the finite mathematical precision. You risk to spend time to accumulate “zero” quantity or differentiate equal values with very small denominators.

Finally, you become too much sensitive to numerical (finite precision) and signal (real signal) noises, in other words, avoid “zipping” poles in (1,0)

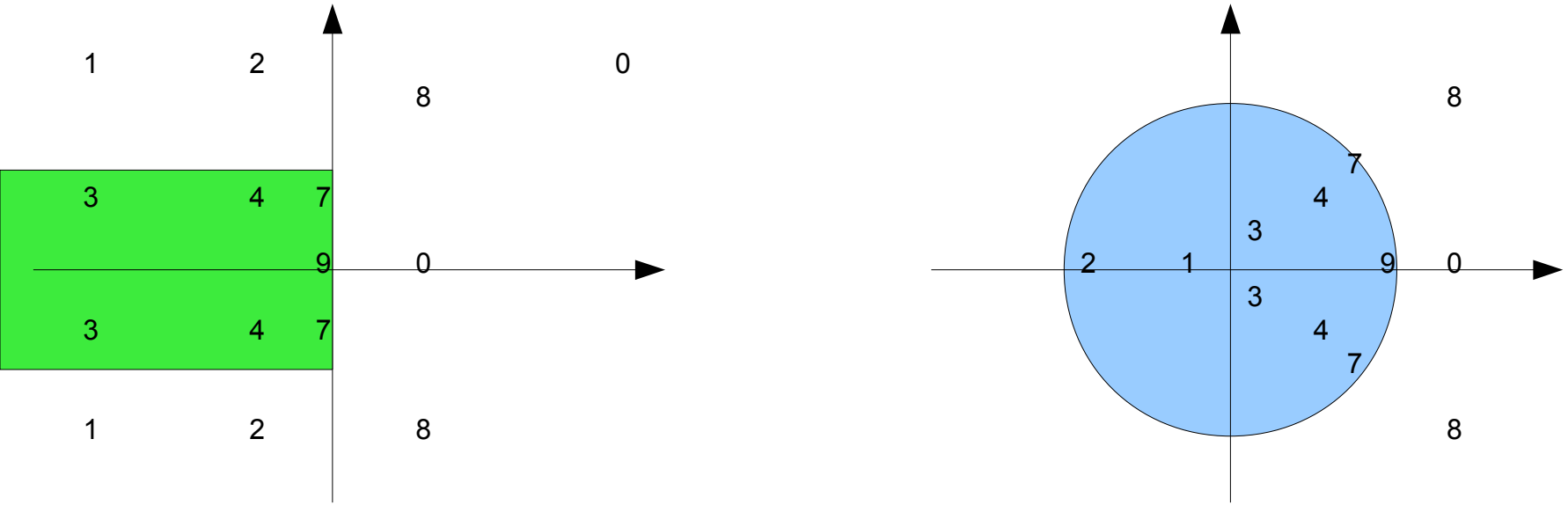
$$z = e^{sT_s}$$

Small Ts means $z \rightarrow (1,0)$ not matter the value of “s”.

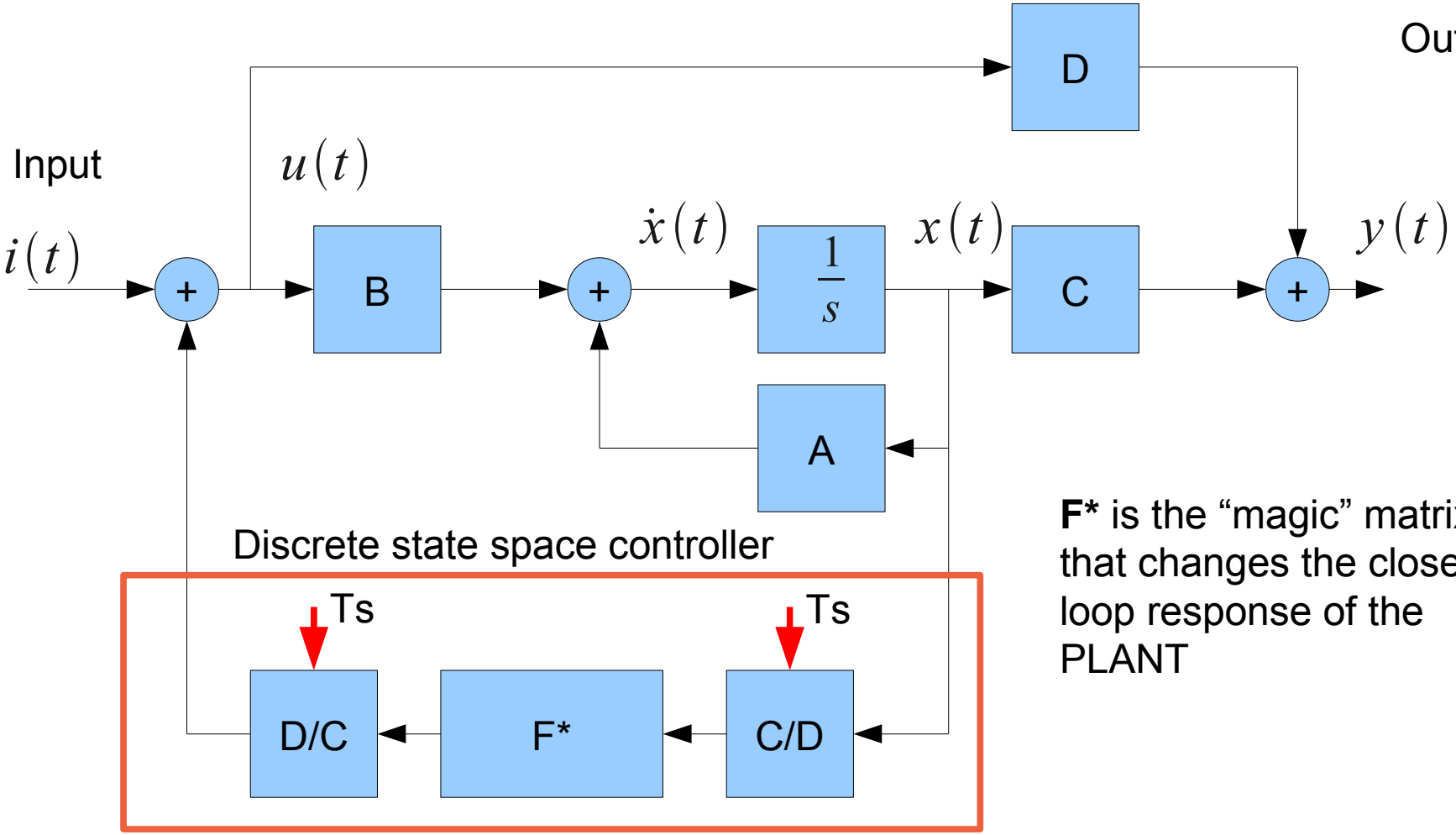
L and Z transforms intimate relationship

Think back to our original problem to create a discrete system that “emulates” a continuous one. Let us suppose that we have already designed our complete system, so we know where the poles are in the “s” (Laplace) plane.

$$s=\sigma+j\omega \quad ; \quad z=e^{sT_s}$$



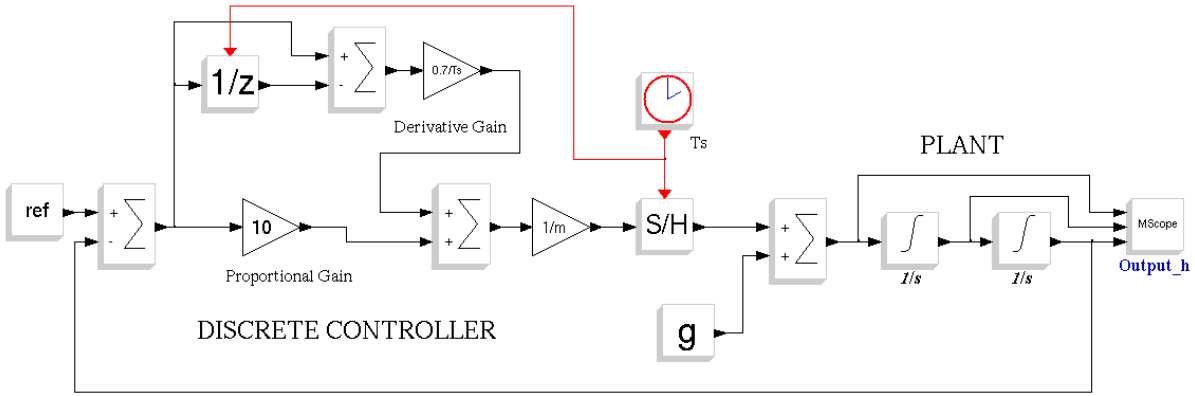
Hybrid state space control

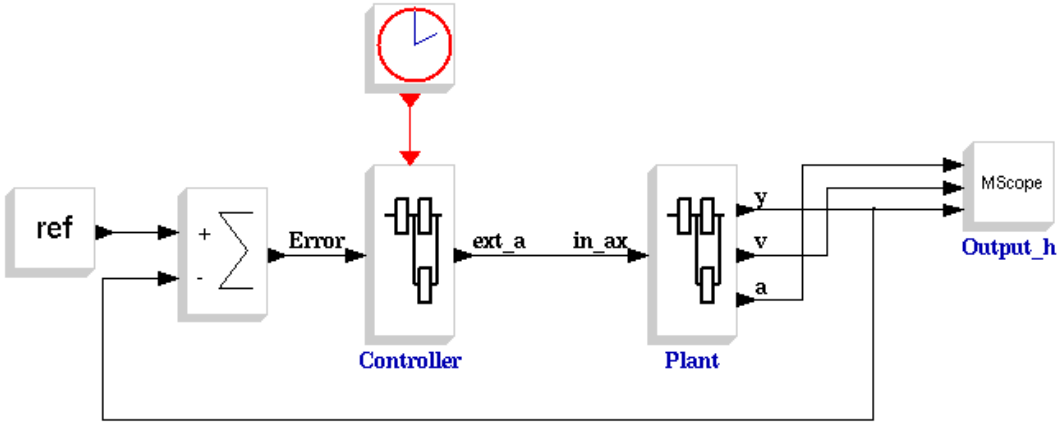


F* is the “magic” matrix that changes the closed loop response of the PLANT

SuperBlocks and Code Generation

Scicos code generation: PD controller example

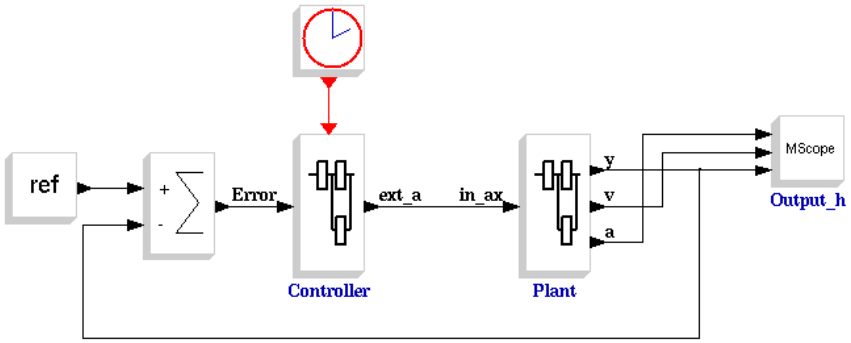


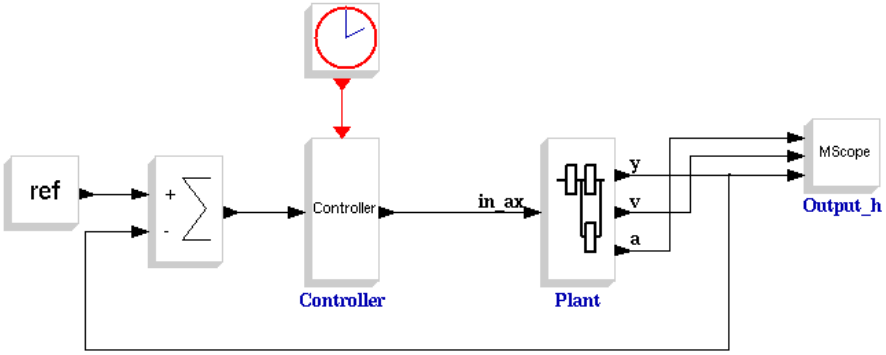


Super blocks help a lot to simplify and organize complex diagrams

(apple_hc_03.cos, apple_hc_05.cos)

Scicos code generation: PD controller example

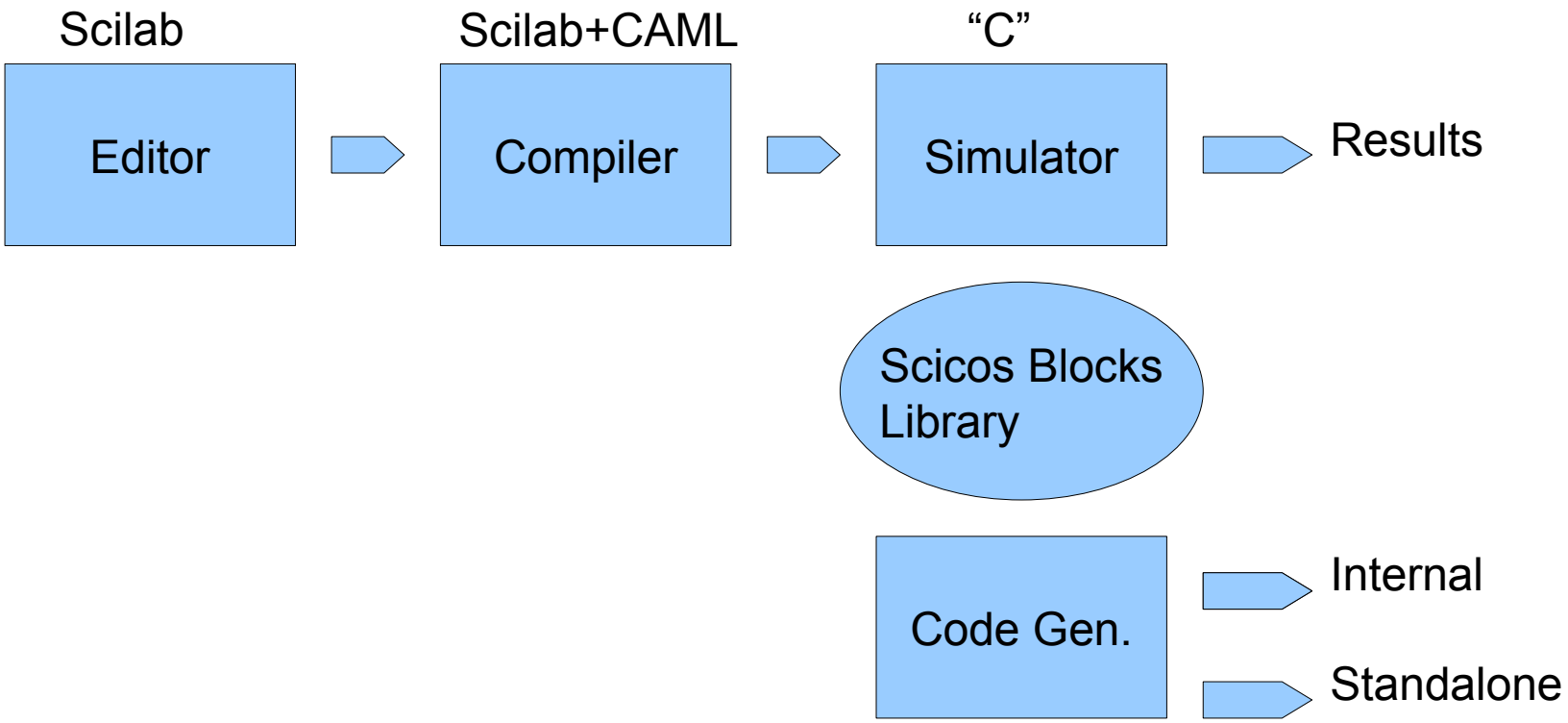




Before and after code generation

(apple_hc_05.cos, apple_hc_06.cos)

Scicos architecture



Scicos block : how does it work ?

Interfacing function: the Scicos block “user's interface”.
A Scilab script that is launched when you “double click” over a Scicos block.

Computational function: the Scicos block simulation function.
The code (typically a C function compiled as shared library) called during the simulation.

Scicos block computational function

```
#include <windows.h>          /* Compiler's include files's */
#include "scicos_block4.h"    /* Specific for Scicos block development */
#include "machine.h"

void custom_bock(scicos_block *block, int flag)
{
  //** scicos_block is a “C” complex data structure that contains in/out ports parameters and values, block's parameters and states

switch(flag) {

     case Init: //** It is called just ONE TIME before simulation start. Put your initialization code here
     break;

     case StateUpdate: //** It is called EACH CYCLE. Read the input ports and update the internal state of the block
                       //** Use this section for OUTPUT blocks (e.g. D/A converter, digital output, etc.)
     break;

     case OutputUpdate://** It is called EACH CYCLE. Read the internal state and update the output
                         //** Use this section for INPUT block (e.g. A/D converter, digital input, etc.)
     break;

     case Ending://** It is called just ONE TIME at simulation end. Put your “shut down” code here.
     break;

  } // close the switch

} // close the computational function
```