Progetto Controllo Digitale

Prof. Aldo Balestrino

Studio e modifica di un sistema ad anello chiuso in ambiente RTAI-Lab con aggiunta di nuove caratteristiche.

Francesco Serafini e Stefano Zingarini

Indice

Analisi progetto

Sistema di riferimento

Il progetto si basa sullo studio e modifica di un sistema ad anello chiuso simulato attraverso RTAI Lab (Figura 1).
Il sistema di riferimento è caratterizzato da:

- Segnale di ingresso.
- Controllore stadio di ingresso e retroazione.
- Impianto.
- Uscita del sistema in base al segnale di ingresso.

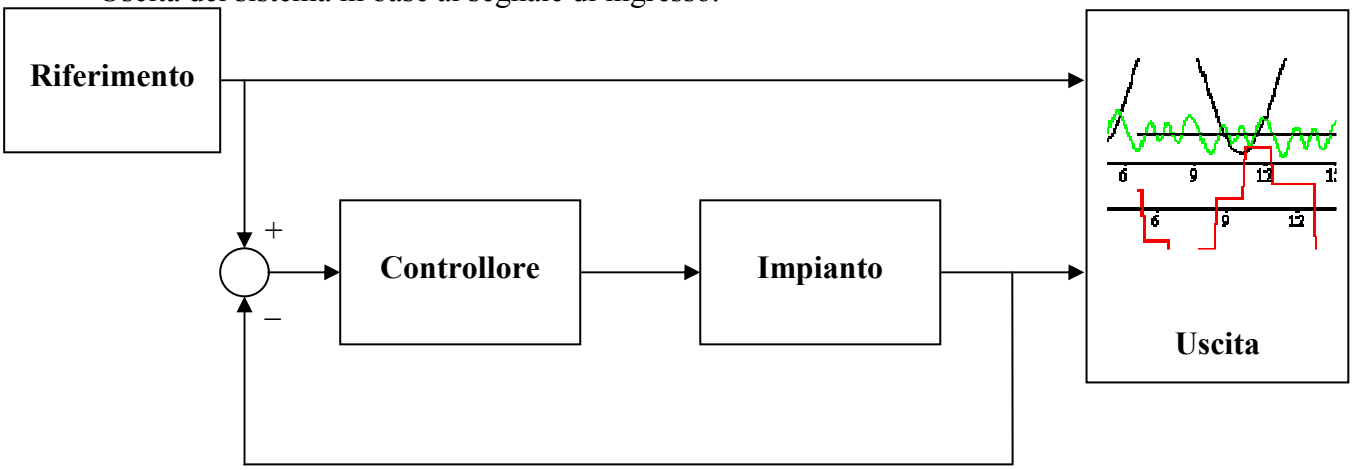


Figura 1 (sistema di riferimento ad anello chiuso)

Sistema da implementare

Il sistema ad anello chiuso presentato ha la retroazione direttamente collegata all'uscita e non introduce disturbi esterni nel ciclo, le modifiche richieste sono (Figura 2):

- Inserire un blocco che permette la variazione in modo proporzionale della retroazione, questo permette anche di ottenere un ciclo aperto.
- Inserire del rumore all'ingresso dell'impianto.
- Inserire del rumore all'uscita senza influenzare il segnale di retroazione.

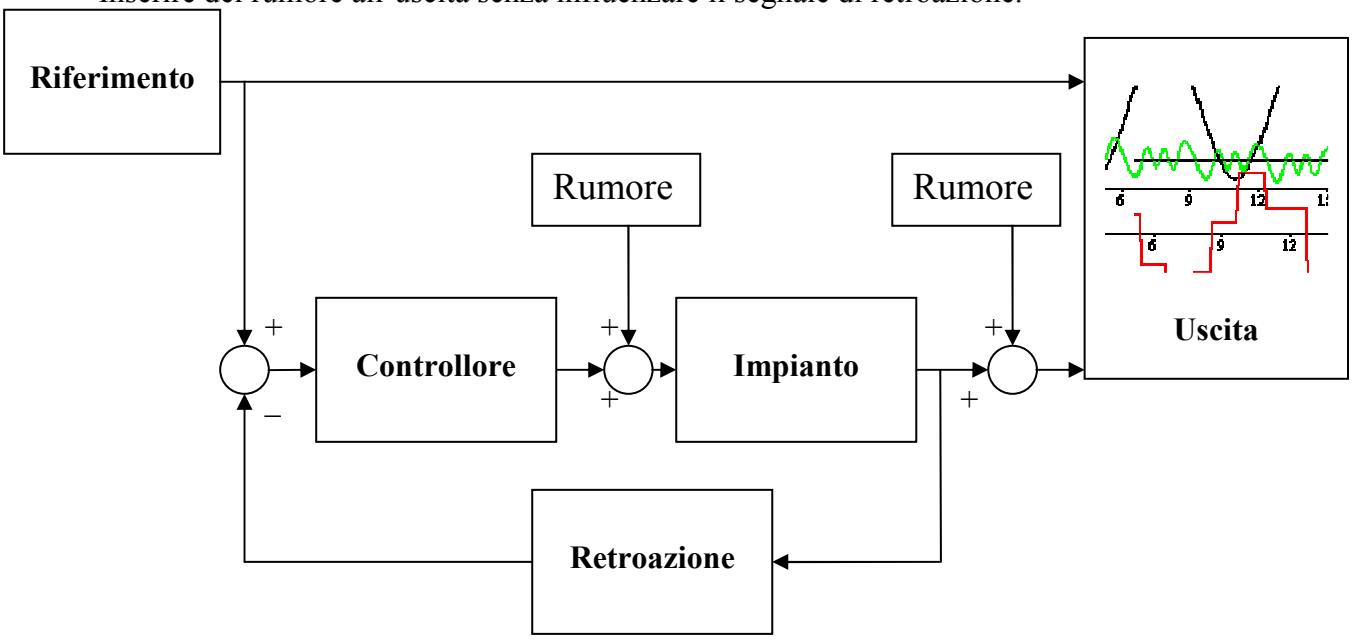


Figura 2 (Sistema finale)

Manuale utente

Guida da allegare alla copertina del DVD

Linux RTAI Live DVD x86 Versione _ _ _ _ _ _ Data _ _ _ _ _ _

Avvio: digitare al bootloader *livecd xdrv=XXX*

XXX rappresenta il driver di Xorg.conf per la vs. sk. Video:
ad esempio: *ati nvidia fbdev (usare fbdev per SVGA)*

Effettuare la login grafica in KDM *root :* _ _ _ _ _ _

Avviare il terminale Konsole e seguire la procedura:

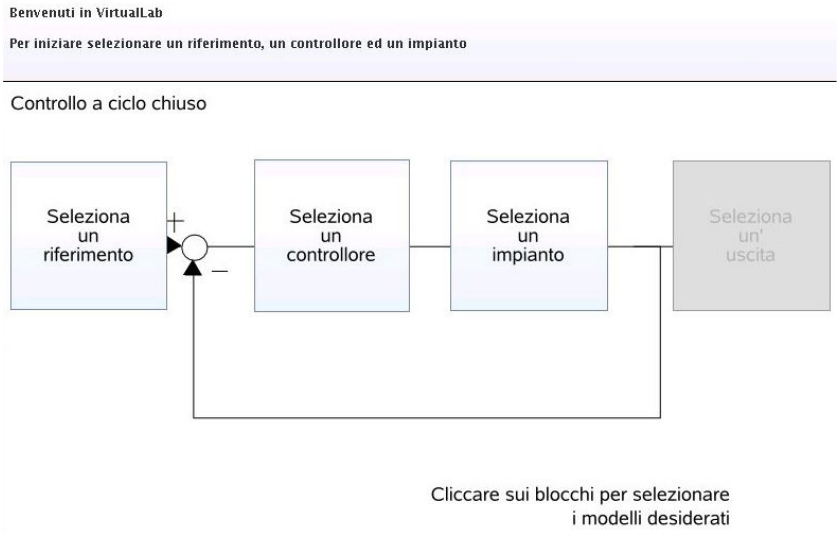
1) Avviare il sistema RealTime: *loadrtai*
2) Installare la patch:
    a. Inserire la penna USB
    b. Attendere la configurazione dalla distribuzione linux
    c. Verificare il punto di montaggio: *mount*
    d. Eseguire i seguenti comandi:

    cd Desktop/Documenti/VirtualLab
    tar –jxf /mnt/YYYY/progetto.tar.bz2
    YYYY indica il punto di montaggio
3) Avviare il servizio RPC: *./virtuallabserver.sh*
4) Avviare il progetto Java da NetBeans cliccando su 

Guida utente

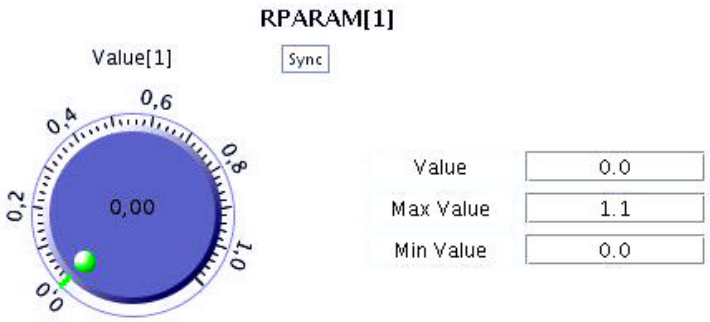
Seguono i passi per effettuare l'avvio del progetto dal dvd live rtai

1) Avviare il computer facendo il boot da DVD
2) al bootloader digitare *livecd xdrv=XXX*
   XXX rappresenta il driver di Xorg.conf per la vs. sk. Video:
   ad esempio: *ati nvidia fbdev (usare fbdev per SVGA)*
3) Effettuare la login grafica in KDM: utilizzare la login *root*
4) Avviare il terminale Konsole e seguire la procedura:
   1) Avviare il sistema RealTime: *loadrtai*
   2) Installare la patch:
      a. Inserire la penna USB
      b. Attendere la configurazione dalla distribuzione linux
      c. Verificare il punto di montaggio: *mount*
      d. Eseguire i seguenti comandi:
      cd Desktop/Documenti/VirtualLab
      tar –jxf /mnt/YYYY/progetto.tar.bz2
      YYYY indica il punto di montaggio
5) Avviare il servizio RPC: *./virtuallabserver.sh*
6) Avviare il progetto Java da NetBeans cliccando su 
7) Compare la seguente interfaccia grafica:



8) Selezionare:
   Ingresso: sinusoide
   Controllore: no controller
   Impianto: plant_noise
9) Cliccare su *SI* per proseguire
10) Configurare la frequenza del seno ad esempio 0.1
11) Cliccare su *Seleziona un uscita* per avviare la simulazione

Configurazione e test nuovo sistema



(Figura Utente1: come si presentato i parametri di configurazione)

Dalla figura Utente1 possiamo notare come possiamo agire attivamente durante la simulazione per modificare la configurazione del sistema:

ID del componente interno al blocco	Paramentro del componente	Valore
R PARAM[1] R indica che i parametri saranno numeri reali, I interi	Value[0] .. Value[n]	0.0 Valore effettivo, *sync* per attivare la modifica

Rumore all'uscita del sistema: PARAM[1]

RPARAM[1]	Value[0] Value[1]	Varianza del rumore Media del rumore
IPARAM[1]	Value[0] Value[1]	Identificatore sequenza casuale algoritmo del rumore Fase iniziale algoritmo del rumore
G.NOISE.Output	Gain	Attenuazione del rumore, utilizzata per evitare di cambiare media e varianza

Rumore all'uscita del controllore: PARAM[2]

RPARAM[2]	Value[0] Value[1]	Varianza del rumore Media del rumore
IPARAM[2]	Value[0] Value[1]	Identificatore sequenza casuale algoritmo del rumore Fase iniziale algoritmo del rumore
G.NOISE.Plant	Gain	Attenuazione del rumore, utilizzata per evitare di cambiare media e varianza

Attenuazione impianto:

RPARAM[4]	Gain	Attenuazione impianto rispetto al segnale del controllore

Attenuazione impianto:

G.FDBK	Gain	Attenuazione retroazione

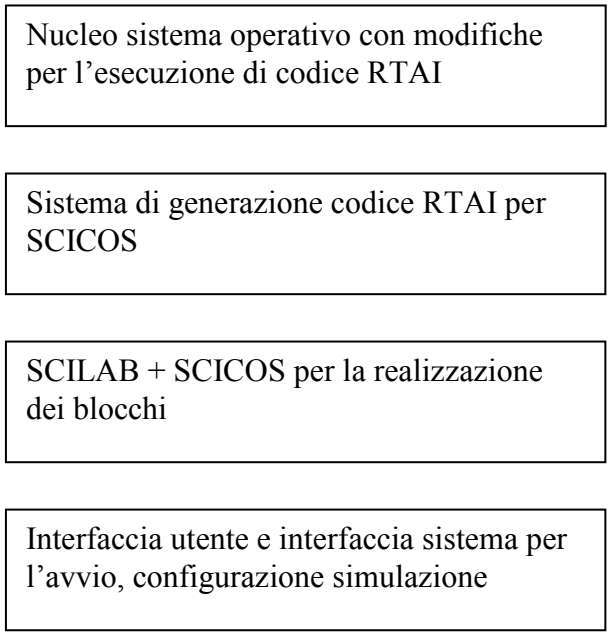
Realizzazione

Il sistema finale comprensivo dei disturbi può essere realizzato modificando il sistema iniziale aggiungendo i componenti richiesti, il tutto utilizzando il sistema di simulazione RTAI-Lab collegato a SCICOS per la realizzazione dei blocchi.
I passi sono stati i seguenti:

- Studio del sistema iniziale
    - Avvio sistema RTAI
    - Simulazione RTAI -Lab
    - Studio componenti sistema iniziale: SCICOS
    - Studio dell'interfaccia VirtualLab in Java
- Realizzazione nuovi blocchi
    - Sistema di test, senza modifica del segnale
    - Algoritmo per creare il rumore gaussiano con media e varianza modificabile in corsa
    - Creazione nuovi blocchi SCICOS
    - Generazione sorgente e binario SCICOS
- Fase di test
    - Test del sistema RTAI-Lab
    - Simulazione con vari input
    - Raccolta dei dati

Studio del sistema iniziale

Il sistema di simulazione è realizzato per la piattaforma RTAI-Lab questa è composta dai seguenti componenti:



Per una visione tecnica e approfondita vi rimandiamo all'appendice.

Avvio sistema RTAI

Avviando il DVD con il sistema RTAI preconfigurato possiamo avviare i componenti sopra descritti:

1. Avvio del DVD ed accesso alla modalità grafica
2. Attivazione dei servizi, moduli, real-time con il comando *loadrtai*
3. Avvio del interfaccia al sistema RTAI con il comando *virtualserver*
4. Utilizzare l'interfaccia grafica utente per la composizione del "sistema" ad anello chiuso ed avvio simulazione

Studio componenti sistema iniziale

Il sistema di riferimento è composto da vari blocchi i quali sono stati realizzati attraverso SCICOS. Importante nella piattaforma RTAI-Lab l'utilizzo delle *mailbox* accessibili attraverso le primitive di SCICOS (oppurtunamente adattato a RTAI-Lab): ogni *mailbox* ha una direzione che può essere in ingresso o in uscita, il sistema a blocchi realizzato utilizza le *mailbox* per far comunicare i blocchi il tutto in maniera trasparente all'utente e allo sviluppatore (Figura 3).

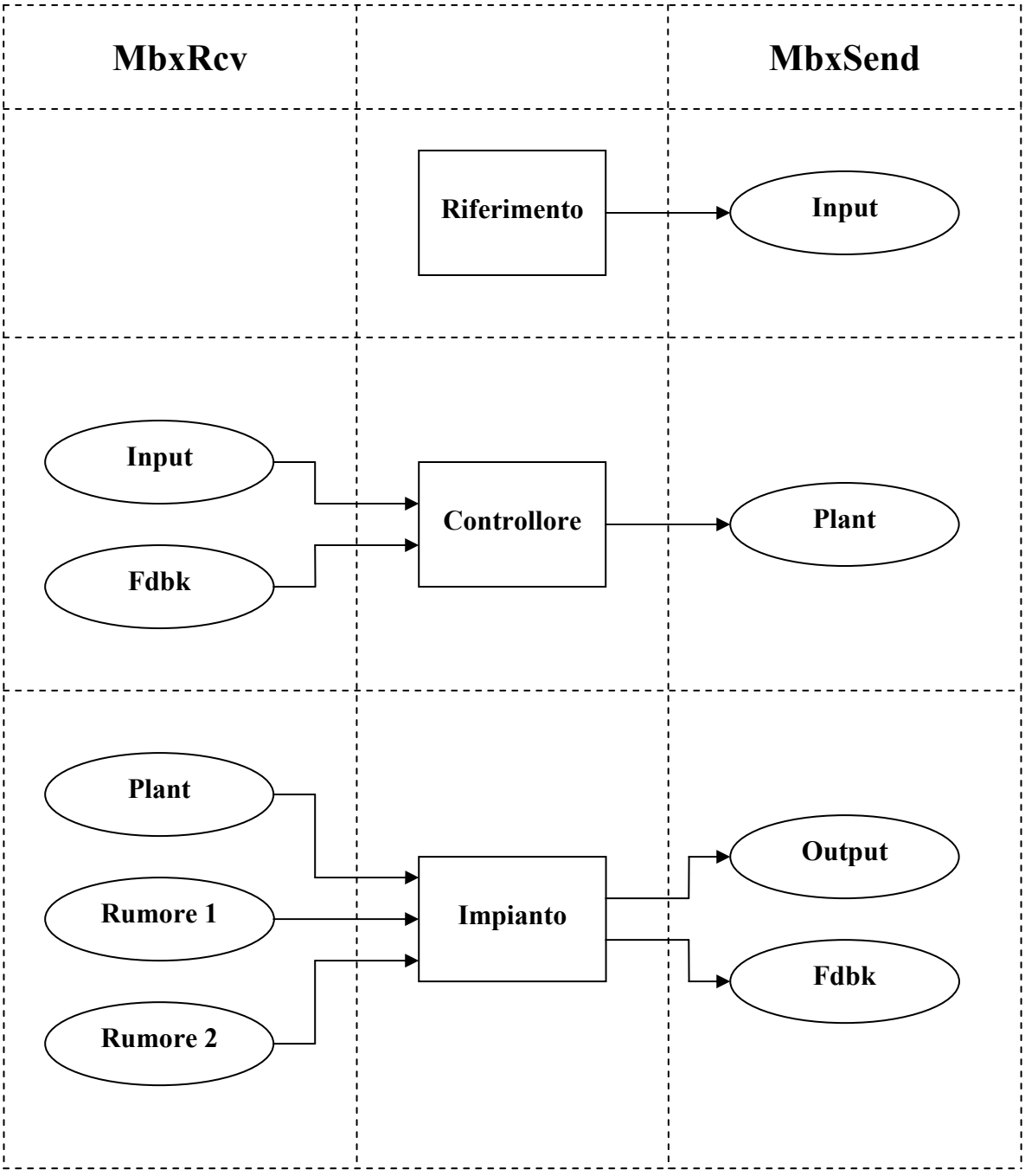


Figura 3 (componenti)

Studio interfaccia utente

L'interfaccia utente è realizzata in linguaggio Java e sfrutta la libreria *swing* e la comunicazione XML RPC per dialogare con l'interfaccia al sistema real-time che carica i moduli binari relativi alla logica.
La versione mostrata in Figura 4 rappresenta una interfaccia ideale per gestire anche la modifica apportata al sistema iniziale.

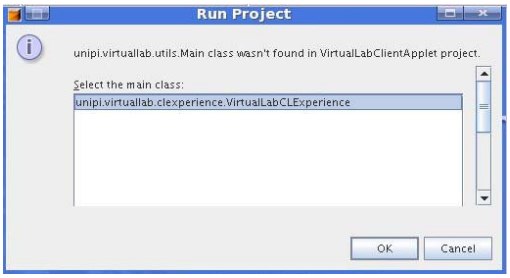


Figura 4.bis (avvio classe Java)

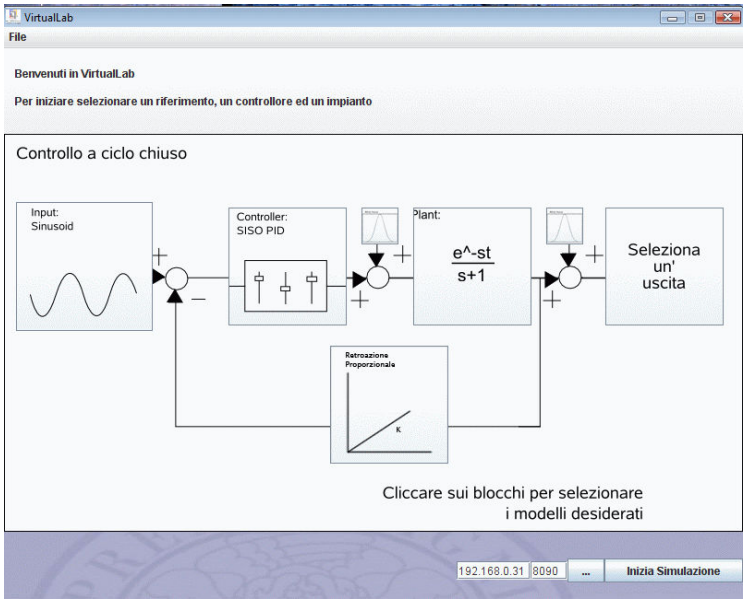


Figura 4 (Interfaccia utente)

Realizzazione sistema di test

Il sistema di test viene realizzato costruendo una catena a ciclo aperto di blocchi che non modicano il segnale di ingresso.
Questo ci permette di isolare la modifica introdotta.
In figura 5 vediamo il sistema di solo test per il rumore generato dal sorgente in C:

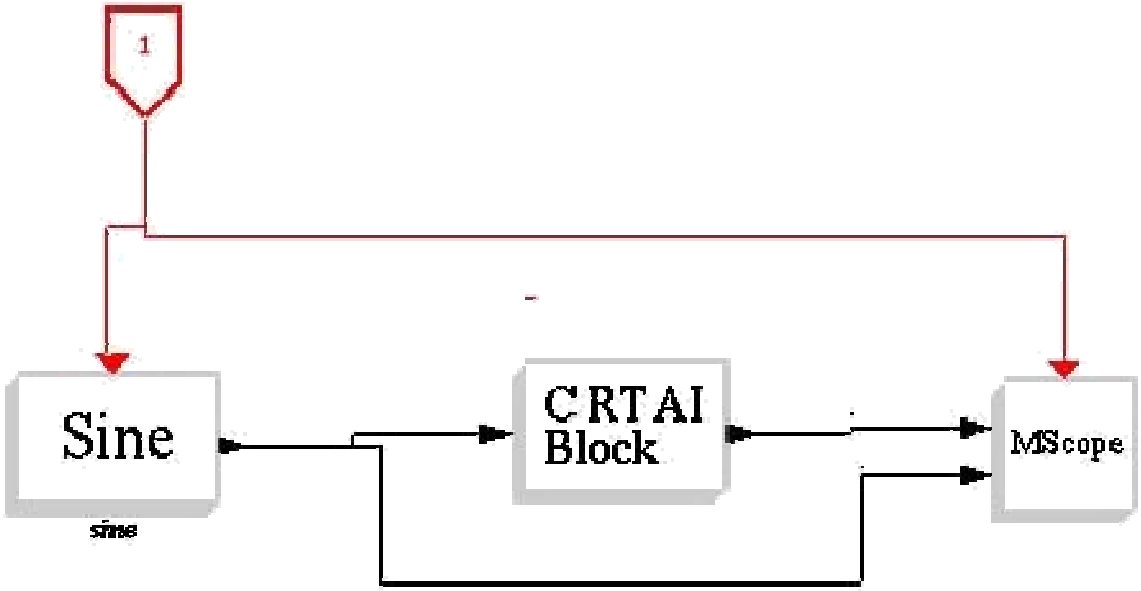


Figura 5 (oscilloscopio diretto)

Caricamento procedura esterna

Scicos ha nella libreria dei componenti un blocco particolare che permette il caricamento di una procedura esterna in C che supporta ingressi ed uscite, questo sarà il nostro componente per la generazione del rumore.
In figura 6 vediamo la base della procedura generata da scicos:

```
Function definition in C
Here is a skeleton of the functions which you should edit

#include <math.h>
#include <stdlib.h>
#ifndef MODEL
#include <scicos/scicos_block.h>
#endif

void noiseblk(scicos_block *block,int flag)
{
  if (flag == 4) { /* initialization */
   nciseblk_bloc_init(block,flag);

  } else if(flag == 1) { /* output computation*/
   set_block_error(noiseblk_bloc_outputs(block,flag));
  } else  if (flag == 5) { /* ending */
    set_block_error(noiseblk_bloc_ending(block,flag));
  }
}

int roiseblk_bloc_init(scicos_block *block,int flag)
{
return 0;}
int roiseblk_bloc_outputs(scicos_block *block,int flag)
{
return 0;}
int roiseblk_bloc_ending(scicos_block *block,int flag)
{
return 0;}
```

Figura 6 (Sorgente creato automaticamente da scicos)

La procedura ha vari parametri per interagire con il sistema, questi fanno parte della struttura:

*scicos_block *block*

All'interno abbiamo rpar, ipar (r = reale ed i = intero) che indicano i valori modificabili in corsa dall'interfaccia utente e permettono quindi una configurazione attiva del blocco creato: nel nostro caso questi indicano la media e la varianza del rumore gaussiano:

rpar[0]	Varianza
rpar[1]	Media

Al termine della stesura del codice scicos informa il corretto caricamento della procedura trasformata in libreria

```
-->// generated by builder.sce: Please do not edit this file

-->// ----------------------------------------------------------

-->noiseblk_path=get_absolute_file_path('loader.sce');

-->link(noiseblk_path+'libnoiseblk.so',['noiseblk'],'c');
shared archive loaded
Link done
```

Figura 7 (caricamente libreria procedura scicos)

Generazione Rumore

```
#define M_PI          3.14159265358979323846
void noiseblkv_c(int *n,int *typ,double *sig,double *mean,double *y_i,doubl
{
int i;
 double rand1, rand2;
 double rand_m;
 double ampl, phase;
rand_m=RAND_MAX;
 for(i=0;i<(*n);i++)
 {
  rand1=rand()/rand_m;
  while((rand1<=0)||(rand1>=1)) rand1=rand()/rand_m;
  rand2=rand()/rand_m;
  while((rand2<=0)||(rand2>=1)) rand2=rand()/rand_m;
  ampl=sig[i]*sqrt(2*-log(rand1));
  phase=2*M_PI*rand2;
  switch(*typ)
  {
   case 0 :
   {
    ampl=sig[i]*sqrt(2*-log(rand1));
    phase=2*M_PI*rand2;
    y_i[i]=mean[i]+ampl*cos(phase);
    break;
   }

   case 1 :
   {
    ampl=sig[i]*sqrt(2*-log(rand1));
    phase=2*M_PI*rand2;
    y_q[i]=mean[i]+ampl*sin(phase);
    break;
   }

   case 2 :
   {
    ampl=sig[i]*sqrt(-log(rand1));
    phase=2*M_PI*rand2;
    y_i[i]=mean[i]+ampl*cos(phase);
    y_q[i]=mean[i]+ampl*sin(phase);
    break;
   }
  }
 }
 return;
}
void noiseblk(scicos_block *block,int flag)
{
  double *y1;
  double *y2=NULL;
  int ny,typ;
  y1=(double *)block->outptr[0];
  ny=block->outsz[0];
  typ=block->ipar[1];
  if(flag==6)
  {
   srand(block->ipar[0]);
  }
  else if(flag==1)
  {
   switch(typ)
   {
    case 0 :
    {
     noiseblkv_c(&ny,&typ,&block->rpar[0],&block->rpar[ny],&y1[0],y2);
     break;
    }
    case 1 :
    {
     noiseblkv_c(&ny,&typ,&block->rpar[0],&block->rpar[ny],y2,&y1[0]);
     break;
    }
    case 2 :
    {
     y2=(double *)block->outptr[1];
     noiseblkv_c(&ny,&typ,&block->rpar[0],&block->rpar[ny],&y1[0],&y2[0]);
     break;
    }
   }
  }
}
```

Realizzazione nuovi blocchi

La produzione di blocchi eseguibili dal sistema RTAI-Lab real-time avviene solo su *superblock* eccone un esempio in Figura 8.

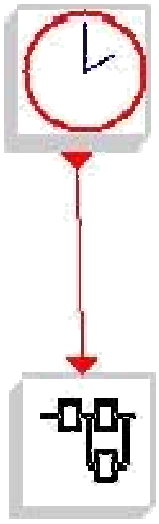


Figura 8(superblock indica un blocco complesso espandibile)

Il sistema di test come già spiegato è stato realizzato senza variare il segnale di ingresso, quindi ora in Figura 9 vediamo il controllore che riporta il nel sommatore i segnali di ingresso e la retroazione e grazie alla *mailbox* invia i dati al blocco dell'impianto.

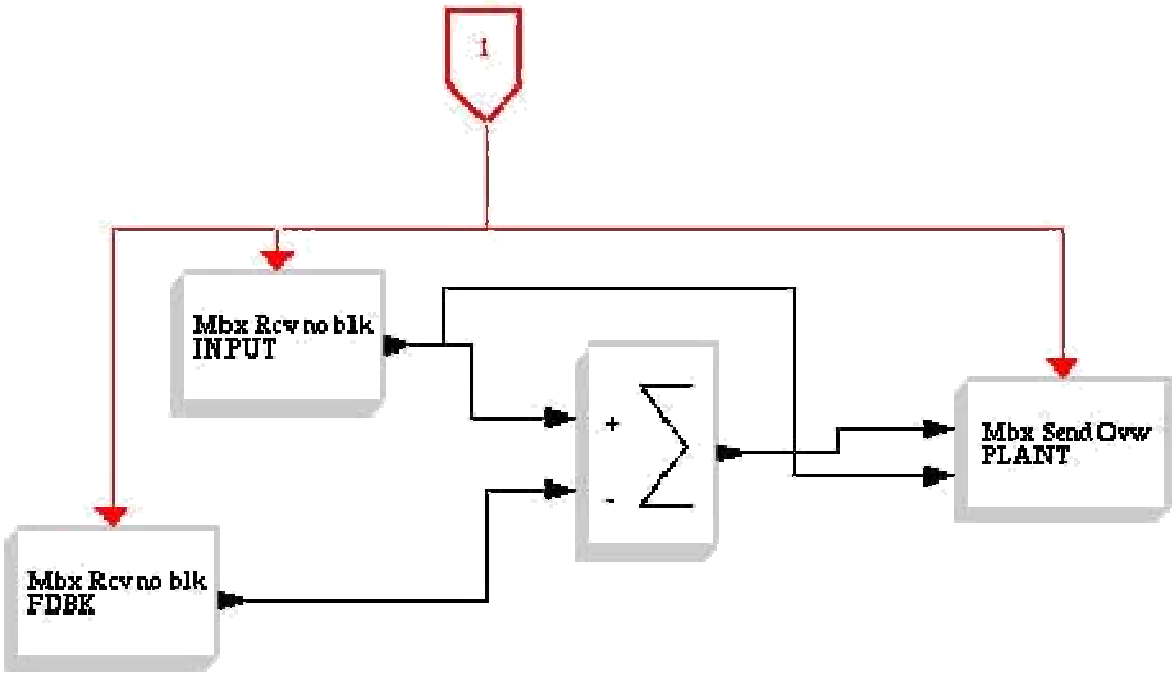


Figura 9(controllore diretto)

Introduzione dei disturbi nel sistema

Il blocco mostrato in Figura 10 introduce i disturbi ed implementa tutte le caratteristiche richieste dalla modifica:

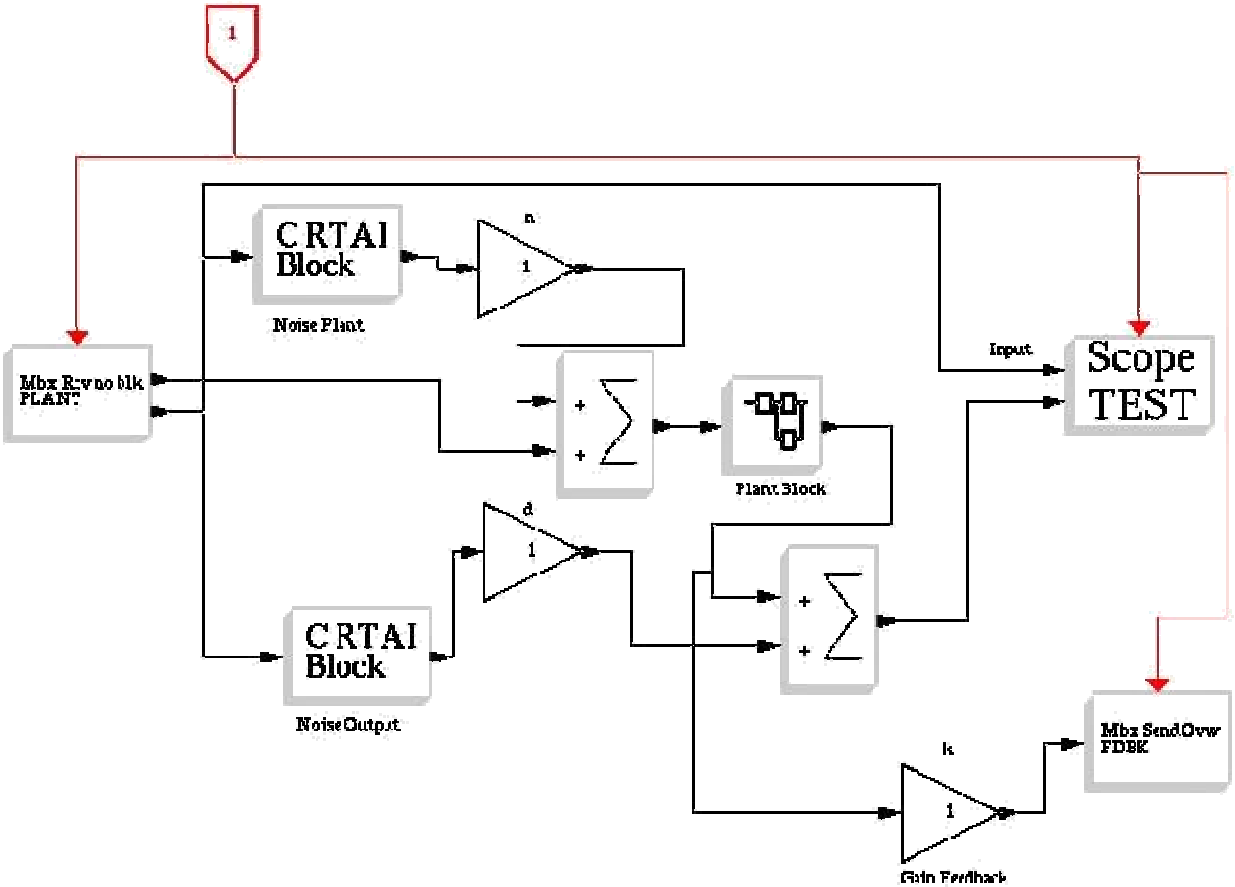


Figura 10 (introduzione disturbi nel sistema)

I sommatori vengono utilizzati con i segni positivi in entrambi gli ingressi:

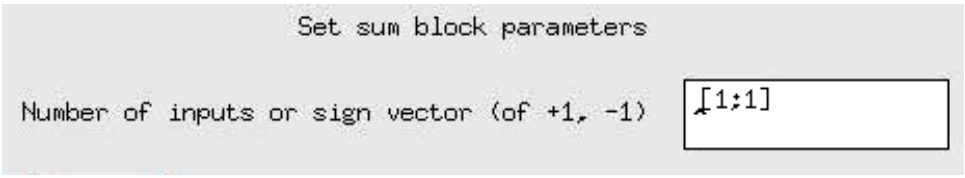


Figura 11(segni per i sommatori)

I parametri di configurazione di tutti i blocchetti sono modificabili in corsa grazie all'interfaccia grafica e possono essere rivisti anche durante l'elaborazione del blocco scicos:

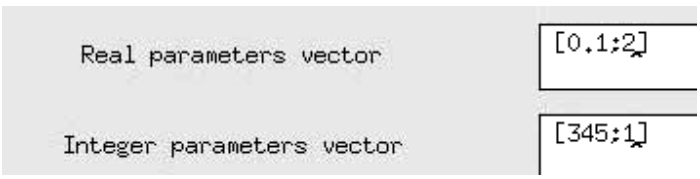


Figura 12 (Configurazione Media e Varianza da scicos)

Fase di test

La fase di test è stata realizzata variando i segnali di ingresso e le impostazioni in corsa dall'interfaccia utente Java.
I risultati proposti graficamente sono stati realizzati con il segnale sinusoidale in ingresso.

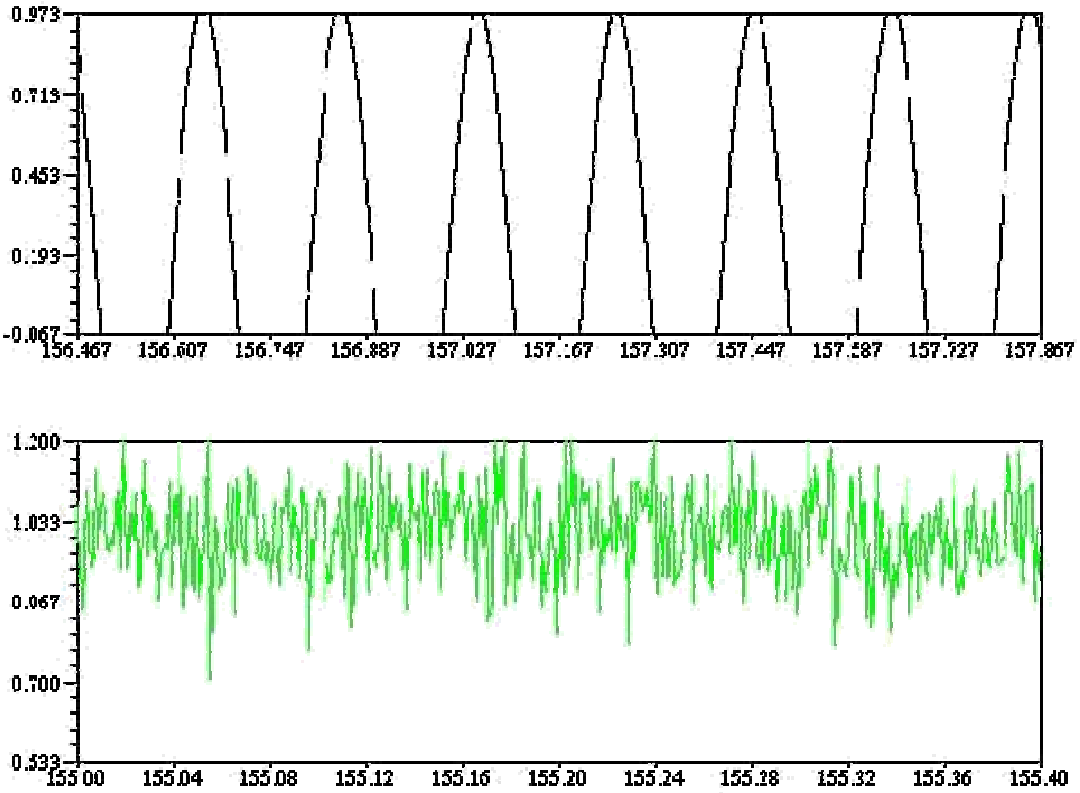


Figura 13(segnale sinusoidale in alto e rumore gaussiano con media 1.0)

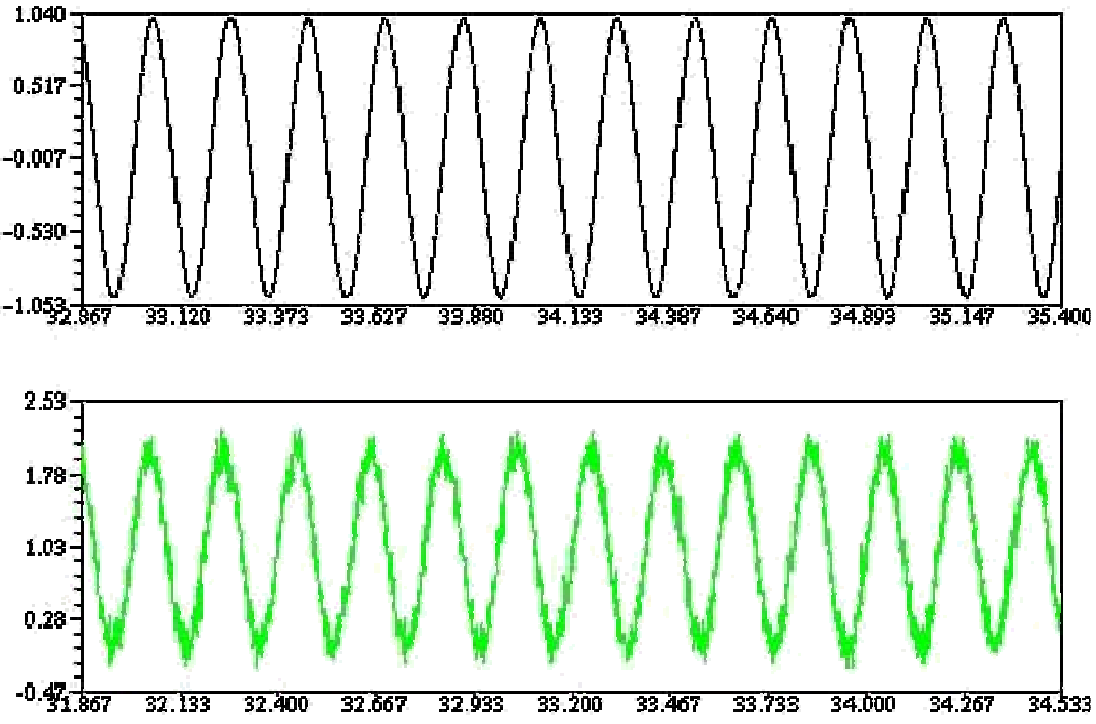


Figura 14 (segnale sinusoidale sommato al rumore gaussiano)

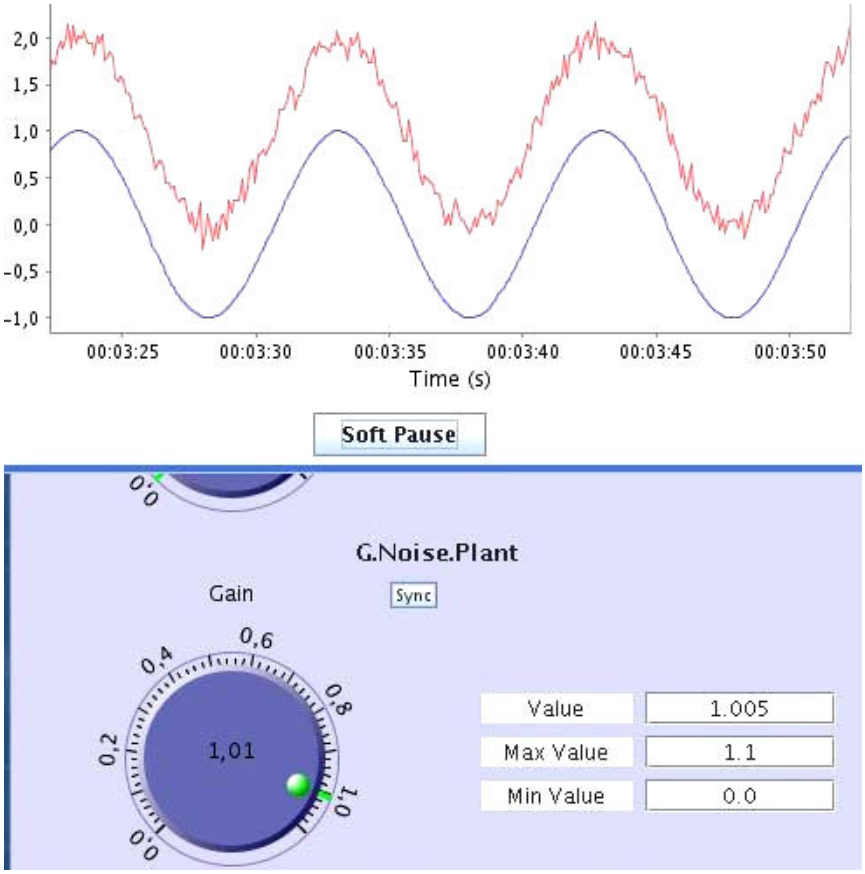


Figura 15 (aggiunta del rumore all'uscita del controllore di media 1)

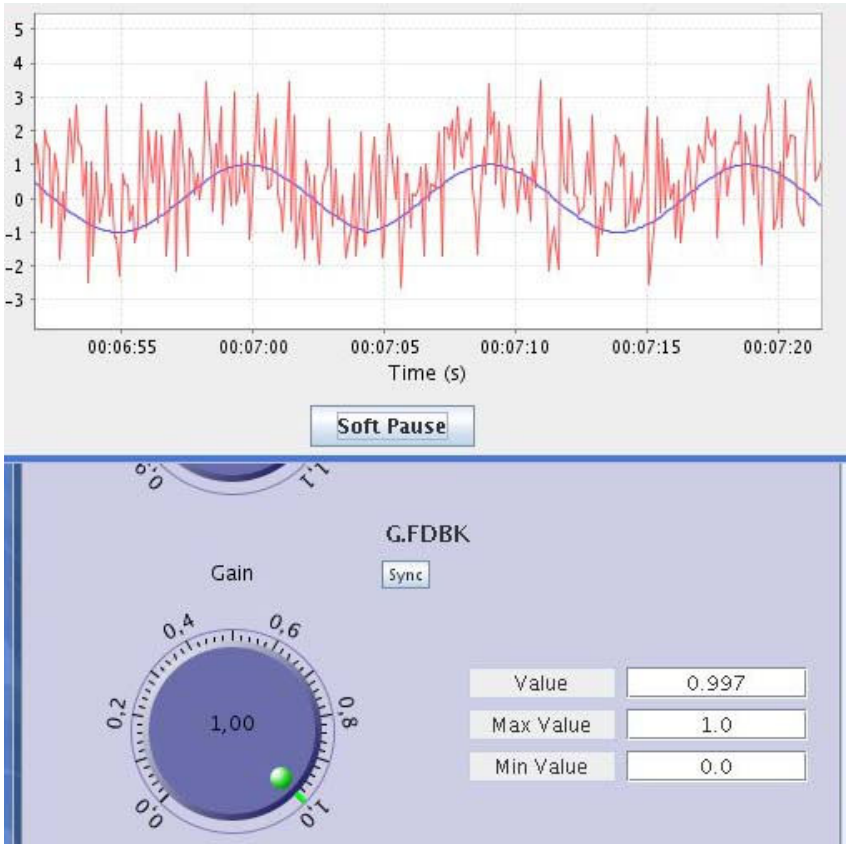


Figura 16 (Attivazione retroazione e rumori)

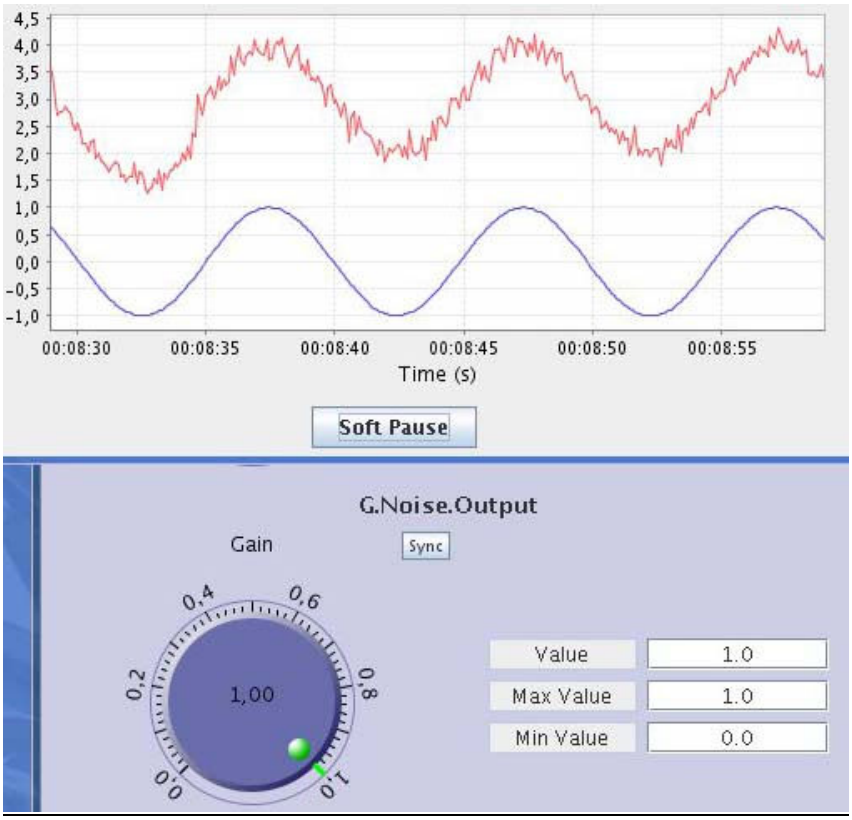


Figura 17(no retroazione, si rumori gaussiani 1 e 2 di media)

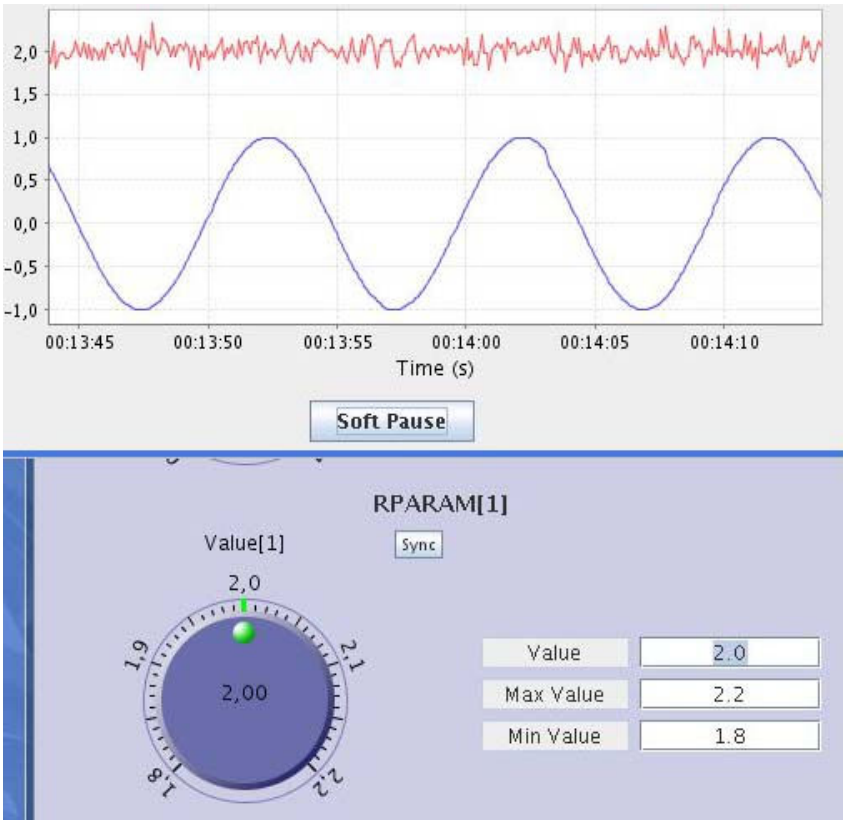


Figura 18 (no retroazione, solo rumore con media 2 all'uscita)

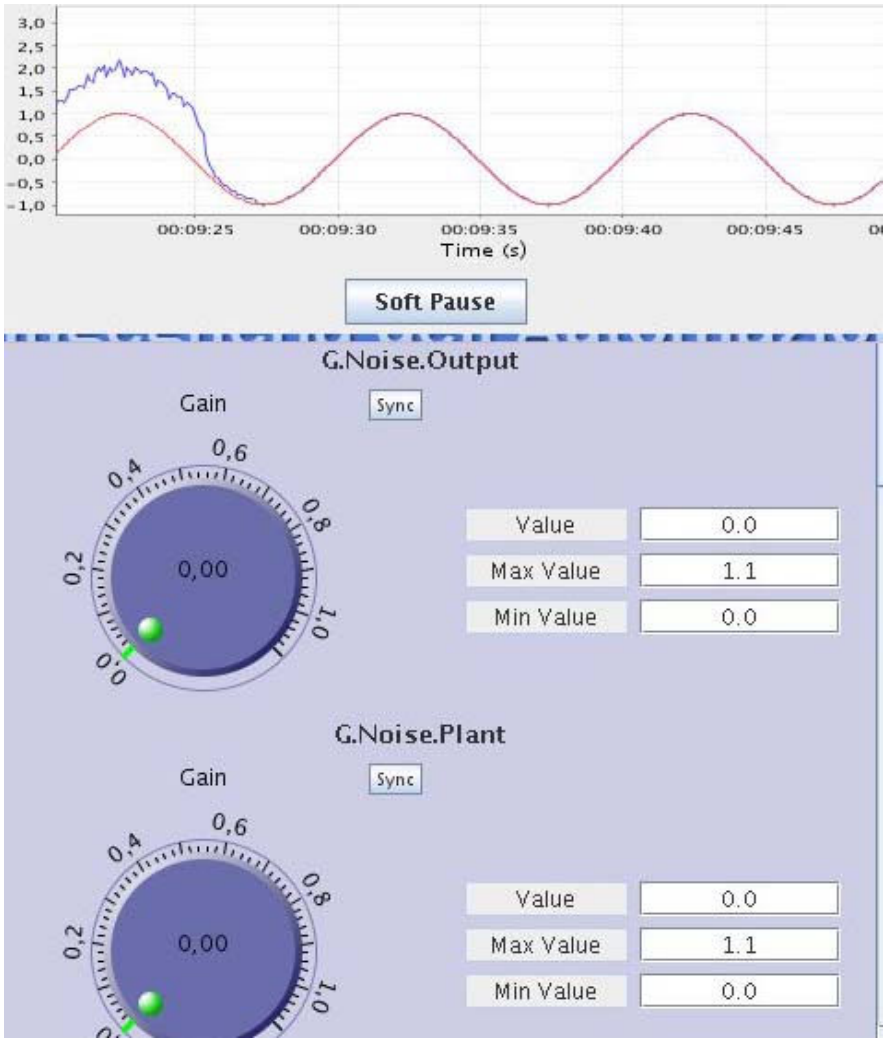


Figura 19 (no retroazione, no rumori)