Il registratore digitale

Il registratore digitale

Nella pratica di laboratorio capita spesso di dover registrare segnali lentamente variabili per intervalli di tempo lunghi.

Lo scopo dell' esperienza è quello di realizzare un registratore digitale, ovvero uno strumento che campioni un segnale ad intervalli di tempo regolari (e programmabili) e registri le misure eseguite ed i tempi a cui sono state eseguite in un file.

Utilizzeremo il nostro registratore ed una termocopia (o una Pt100) per misurare

- la curva di raffreddamento di un certo volume d'acqua;
- l'andamento della temperatura nell'arco di una giornata in aula informatica.

Due parole sui sensori

Un sensore (o trasduttore) è un dispositivo che genera un segnale elettrico correlato con una grandezza fisica.

Termometro a resistenza metallica

La resistività di un metallo, per temperature non troppo basse, segue un andamento quasi lineare:
$R(T) = R0(1 + a_1 T + a_2 T^2 + ...)$
$R_0 = 100\Omega$
$a_1 = 3.9083 \cdot 10^{-3}\ ^\circ C^{-1}$
$a_2 = -5.775 \cdot 10^{-7}\ ^\circ C^{-1}$

Vantaggi: precisione, prontezza (piccola massa) e discreta linearità su un ampio intervallo di temperature (Pt da 10-800 K).

Termocoppia

La termocoppia sfrutta la dipendenza dalla temperatura della forza elettromotrice ai capi di una giunzione tra metalli diversi (effetto Seebeck).
Questa forza elettromotrice è funzione crescente di T, ed è quasi lineare in prossimità della temperatura ambiente.

Vantaggi: prontezza (piccola massa), facilità di accoppiamento termico (con fili sottili e lunghi), esteso intervallo di lavoro (70-1000 K) e basso costo.

Requisiti del registratore

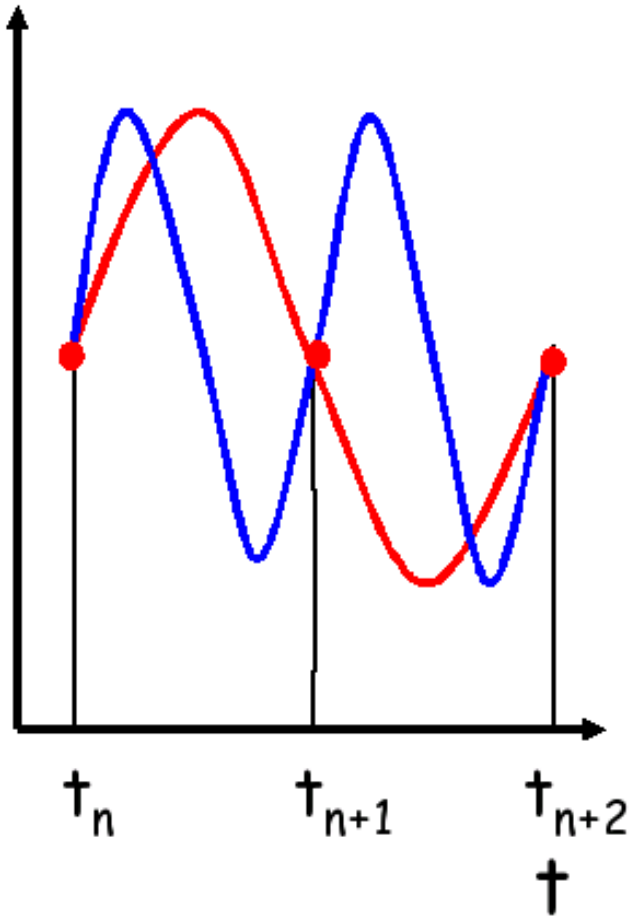
- Possibilità di scegliere la frequenza di campionamento e la durata totale del campionamento (incluso durata illimitata ed arresto manuale).
- Corretta decodifica dei dati del multimetro.
- Calibrazione (almeno approssimativa) della misura (per la termocoppia potete utilizzare quanto avete trovato nel primo semestre a Laboratorio di Fisica, per la Pt100 utilizzate i valori precedentemente riportati).
- Grafico in tempo reale degli ultimi N campionamenti, con N programmabile anche nel corso dell'acquisizione.
- Salvataggio su file dei dati registrati e del tempo di campionamento per una successiva analisi (off-line).
- Fit in tempo reale dei dati campionati (opzionale).

Campionamento

A prima vista il campionamento può sembrare un problema banale.

In realtà non è così , e lo si comprende bene solo quando si affronta l'analisi armonica del segnale in ingresso e di quello ricostruito (ovvero la scomposizione del segnale in sinusoidi di frequenza diversa).

Si comprende allora che se si campiona un segnale ad una data frequenza ν risulta impossibile distinguere, e quindi ricostruire, tutte le componenti del segnale con frequenza maggiore di $\nu/2$.



Misura del tempo in UNIX

Abbiamo a disposizione una serie di routines di libreria per la misura del tempo; in particolare possiamo utilizzare funzioni che forniscono il tempo assoluto, funzioni di pausa e timers (che vedremo in seguito).

Le misure di tempo assoluto sono utili per associare tempi e misure.

Le funzioni di pausa generano intervalli di durata fissata in cui il programma non fa nulla, e il computer è libero di dedicarsi ad altre attività.

Routines di timing

time_t time(time_t *)
tempo trascorso dal 1/1/1970 in secondi. time_t identifica sostanzialmente un intero, se viene passato un puntatore non nullo il valore di ritorno viene scritto anche in memoria (all'indirizzo specificato).

int gettimeofday(struct timeval *tv, struct timezone *tz)
scrive nella struttura tv il tempo trascorso dal 1/1/1970 (struttura *timezone* obsoleta: si passi un puntatore nullo).

```
struct timeval {
  int tv_sec;
  int tv_usec;
};
```

void sleep(int nsec)
pausa in secondi

void sleep(int nsec)
pausa in μsecondi

Come calcolare differenze di tempo

Schema di funzione per il calcolo del tempo:

```
double delta_t(struct timeval t0){
  struct timeval t_tmp;
  gettimeofday(&t_tmp);
  // calcolo della differenza di tempo (dt) tra t_tmp e t0
  return dt;
}
```

La definizione di $t0$ è arbitraria e può essere fatta concidere con la prima misura, con l'inizio del programma o con un istante scelto dall'utente.

Schema del programma


```
int fd_metex; ofstream file_out;
TCanvas *window; TGraph *graph;
void init(){
  // apre la porta seriale e ne definisce i parametri di comunicazione
  // apre il file di output e la finestra grafica
}
void end(){
  // attiva l'editor grafico e poi chiude la finestra grafica
  // chiude il file di output e la seriale
}
void misura(){
  double dt  = delta_t(t0);
  double val = get_metex(...);
  double T   = ....; // eventuale conversione val->T
  // aggiornamento grafico e scrittura t e T su file
}
int main(){
  init();
  ...;
  for (...){misura(); sleep(1);} // N misure (1 al secondo);
  ...
  end();
  ...
}
```

Schema del programma (II)

Può essere utile, soprattutto in acquisizione lunghe, mostrare sul grafico gli ultimi `N` dati; fermo restando che tutti i dati acquisiti vengono comunque salvati su file.

Ecco un esempio di come fare

```
...
TGraph *gr;
...
if (i>N) i=0;
gr->SetPoint(i++,t,x);
...
```

Gestione dei segnali

Poichè stiamo acquisendo segnali variabili nel tempo, verosimilmente per osservare un transiente, è utile avere la possibilità di fermare il programma (salvando il file di dati e visualizzando il grafico) quando la parte "interessante" della misura si è esaurita.

In generale siete abituati a considerare un programma come un singolo flusso di istruzioni che vengono eseguite una dietro l'altra in modo rigidamente sequenziale. Se però si vogliono gestire con un programma eventi che non si verificano ad un tempo (o a tempi) predefiniti (asincroni) questo approccio risulta inadeguato.

Linux fornisce uno strumento per la gestione di eventi asincroni: i segnali. I segnali sono eventi software che possono essere associati a diversi eventi fisici: un interrupt da una periferica, una particolare condizione rilevata da un altro programma, un errore di calcolo, lo scadere di un intervallo temporale. Ai segnali viene associata una function che viene eseguita, in modo asincrono rispetto al programma principale, quando l'evento selezionato si verifica.

Gestione dei segnali (II)

Esempio di programma in grado di reagire ad un segnale

signal

```
#include <signal.h>
void signal_handler(int nsig) {
  // azione
}
int main(){
  ...
  signal(SIGNAME, signal_handler);
  ...
  return 0;
}
```

Quando viene inviato al programma il segnale `NAME`, indipendentemente dal punto a cui è arrivata l'esecuzione, viene chiamata la funzione `signal_handler` (l'intero passato in argomento a tale funzione è il valore di `SIGNAME`).

Gestione dei segnali (III)


```
SIGHUP   Il collegamento con il terminale e' stato interrotto.
SIGINT   Interruzione attraverso un comando dalla tastiera.
SIGQUIT  Conclusione attraverso un comando dalla tastiera.
SIGILL   Istruzione non valida.
SIGABRT  Interruzioni di sistema.
SIGFPE   Eccezione in virgola mobile.
SIGKILL  Conclusione immediata.
SIGSEGV  Riferimento non valido a un segmento di memoria.
SIGPIPE  Condotto interrotto.
SIGALRM  Timer.
SIGTERM  Conclusione.
SIGUSR1  Primo segnale definibile dall'utente.
SIGUSR2  Secondo segnale definibile dall'utente.
SIGCHLD  Eliminazione di un processo figlio.
SIGCONT  Riprende l'esecuzione se in precedenza e' stato fermato.
SIGTSTOP Ferma immediatamente il processo.
SIGTSTP  Stop attraverso un comando della tastiera.
SIGTTIN  Processo sullo sfondo che richiede dell'input.
SIGTTOU  Processo sullo sfondo che deve emettere dell'output.
```

Gestione dei segnali (IV)

Corrispondenza segnale - chiave da tastiera

```
> stty -a
intr  = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = <undef>; eol2 = <undef>; start = ^Q; stop = ^S;
susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O;
```

Come inviare i segnali ad un processo ?

- Si può ad esempio inviare il segnale da tastiera (utilizzando la chiave corrisponde) sul terminale in cui il processo è in esecuzione;
- oppure trovare il numero identificativo del processo (`pid`) (in questo caso il programma in esecuzione) dando, su un'altro terminale, il comando

  ```
  ps ax | grep nome-processo
  ```

  e quindi inviando, sempre dalla stesso terminale, il comando

  ```
  kill -s NAME pid
  ```

  dove NAME è la stringa che segue SIG
- oppure agendo sul nome del processo

  ```
  killall -s NAME nome-processo
  ```

Timers

I timers forniscono dei segnali ad intervalli di tempo regolari: sono utili per eseguire operazioni periodiche durante il normale svolgimento del programma (esecuzione asincrona).

setitimer

```
void setitimer(int type, struct itimerval *value,
                         struct itimerval *ovalue);
struct itimerval {
  struct timeval it_value;   // starting time
  struct timeval it_interval; // time interval
};
```

Se type = `ITIMER_REAL` dopo un tempo `it_value` manda un segnale SIGALRM ogni `it_interval` (verificare con man le altre opzioni)

Timers

Esempio di funzione che utilizza un timer

```
void periodic(int nsig) {
  // Questa function verra' chiamata dopo 1.5 sec, e poi ogni sec
}

int main() {
  struct itimerinterval t;
  t.it_value.tv_sec = 1;
  t.it_value.tv_usec = 500000;
  t.it_interval.tv_sec = 1;
  t.it_interval.tv_usec = 0;
  setitimer(ITIMER_REAL,&t,NULL);
  signal(SIGALRM, periodic);
  ...
  while(1){
    // Modifica dei parametri di acquisizione (frequenza,...)
  }
  ...
}
```

Esercitazione in Laboratorio

1. Impostare un programma che, seguendo lo schema illustrato registri (grafichi e salvi su file) le misure di temperature (termocoppia o Pt100) in funzione del tempo ad intervalli regolari. Provate il programma misurando la curva di raffreddamento di un certo volume d'acqua (termocoppia) o la variazione di temperatura ambientale in aula (Pt100).
2. Si aggiunga al programma la funzionalità di mostrare sul grafico solo gli ultimi N dati acquisiti.
3. Si aggiunga al programma la funzionalità di poter interrompere l'acquisizione con un segnale (as esempio ctrl-C) salvando opportunamente il file e visualizzando il grafico.
4. Si organizzi l'acquisizione con un timer e si utilizzi il programma principale per modificare alcuni parametri di acquisizione (ad esempio la frequenza di campionamento).