

Aspetti Implementativi del Software dei Sistemi di Controllo

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory

Dipartimento di Matematica e Informatica - Università di Catania, Italy

santoro@dmf.unict.it



Programmazione Sistemi Robotici

Implementazione di un Sistema di Controllo

L'implementazione software di un sistema di controllo si basa sempre su un algoritmo caratterizzato dall'**esecuzione di un loop temporizzato di un insieme attività**:

- Avvio delle attività sulla base di un **tempo di campionamento**, ΔT
- Lettura dell'ingresso del sistema di controllo
- Esecuzione dell'algoritmo di controllo specifico
- Invio dei risultati all'output

Algoritmo

```
while True do
```

```
    On each  $\Delta T$ ;
```

```
    in  $\leftarrow$  read_input();
```

```
    out  $\leftarrow$  compute_control();
```

```
    write_output(out);
```

```
end
```

Implementazione di un Sistema di Controllo

Qualora il sistema completo sia composto da diversi sotto-sistemi, in teoria, ognuno di essi dovrà essere implementato con un **loop (computazionale) differente** e i vari loop dovranno essere **eseguiti concorrentemente**



Sistema S1

while *True* **do**

```
On each  $\Delta T_1$ ;  
in1  $\leftarrow$  read_input1();  
out1  $\leftarrow$  compute_S1();  
write_output1(out1);
```

end

Sistema S2

while *True* **do**

```
On each  $\Delta T_2$ ;  
in2  $\leftarrow$  read_input2();  
out2  $\leftarrow$  compute_S2();  
write_output2(out2);
```

end

Poichè i sotto-sistemi sono interconnessi, occorrerà altresì implementare le opportune **forme di comunicazione** tra i “loop”: nel caso specifico la variabile *out1* di S1 è anche l’input *in2* di S2.

Implementazione di un Sistema di Controllo

Tuttavia se i tempi di campionamento coincidono $\Delta T_1 = \Delta T_2 = \Delta T$, è possibile “fondere” i due loop:

Sistema S1+S2

```
while True do  
  On each  $\Delta T$ ;  
  in1  $\leftarrow$  read_input1();  
  out1  $\leftarrow$  compute_S1();  
  in2  $\leftarrow$  out1;  
  out2  $\leftarrow$  compute_S2();  
  write_output2(out2);  
end
```



Implementazione di un Sistema di Controllo

Analogamente, qualora uno dei due intervalli sia **multiplo intero** dell'altro $\Delta T_1 = n\Delta T_2$, è possibile implementare il sistema come:

Sistema S1+S2

```
while True do
```

```
  On each  $\Delta T_2$ ;
```

```
   $count \leftarrow count + 1$ ;
```

```
  if  $count = n$  then
```

```
     $count \leftarrow 0$ ;
```

```
     $in1 \leftarrow read\_input1()$ ;
```

```
     $out1 \leftarrow compute\_S1()$ ;
```

```
  end
```

```
   $in2 \leftarrow out1$ ;
```

```
   $out2 \leftarrow compute\_S2()$ ;
```

```
   $write\_output2(out2)$ ;
```

```
end
```



Scelta del Tempo di Campionamento

Serie di Fourier

Un qualunque segnale (periodico) $s(t)$ può essere rappresentato come una **combinazione lineare** di sinusoidi e cosinusoidi di coefficienti a_i e b_i :

$$s(t) = \frac{a_0}{2} + \sum_{n=1}^N \left[a_n \cos\left(\frac{2\pi}{T} nt\right) + b_n \sin\left(\frac{2\pi}{T} nt\right) \right]$$

In altri termini, il segnale originario può essere ricostruito usando una **combinazione lineare di sinusoidi a frequenze diverse**.

Un segnale proveniente da un sensore può essere assimilato ad un segnale del tipo indicato.

Teorema del Campionamento

Un qualunque segnale (periodico) $s(t)$ può essere **ricostruito fedelmente** qualora venga **campionato** ad una frequenza f_{sample} e che sia **almeno due volte** la frequenza della sinusoide di frequenza massima

Regole pratiche

- **Teorema del campionamento:** principalmente utile in caso di segnali audio
- **Più piccolo è, meglio è:** soprattutto quando sono presenti **approssimazioni** legate a calcolo differenziale (derivate e integrali)
- **Ma non troppo piccolo:** all'interno del periodo di campionamento, la CPU esegue una computazione; la durata della computazione **nel caso peggiore** non deve essere maggiore del tempo di campionamento
- **Ma neanche troppo grande,** altrimenti il sistema diventa incontrollabile
- **Usate i multipli:** nel caso sia necessario (o utile) avere, in un sistema, **differenti periodi di campionamento**, sceglierli in modo che siano tutti **multipli interi** di un periodo comune semplifica l'implementazione

Suddivisione in Task

Organizzazione del software di un sistema di controllo

- Un insieme di **task** ognuno dei quali implementa un sotto-sistema
- Un ambiente di **scheduling** in grado di eseguire tali task con la loro **periodicità** (basa sul tempo di campionamento)
- Un ambiente di **comunicazione** tra task, al fine di consentire lo scambio dati e la sincronizzazione dei task stessi

Esempio: Task di Sistema di Controllo



Consideriamo il sistema in Figura e supponiamo che **S2** sia campionato a ΔT , e **S1** campionato a $4\Delta T$.

Task_S1()

```
in1 ← read_input1();  
out1 ← compute_S1();  
write_output1(out1);
```

Task_S2()

```
in2 ← read_input2();  
out2 ← compute_S2();  
write_output2(out2);
```

Esempio: Scheduling e Comunicazione di Sistema di Controllo



Consideriamo il sistema in Figura e supponiamo che **S2** sia campionato a ΔT , e **S1** campionato a $4\Delta T$.

Sistema S1+S2

```
while True do
  On each  $\Delta T$ ;
  count  $\leftarrow$  count + 1;
  if count = 4 then
    count  $\leftarrow$  0;
    Task_S1();
    in2  $\leftarrow$  out1;
  end
  Task_S2();
end
```

Componenti

- **Rosso:** Task
- **Verde:** Scheduler
- **Blu:** Comunicazioni

Modello Strutturato

- Una **funzione** (corpo del task) che implementa il comportamento del sistema
- Una **struttura dati** che ingloba le informazioni di stato del task

Modello a Oggetti

- Una **classe** che rappresenta il sotto-sistema con...
- un **metodo principale** (es. `run()`) che implementa il comportamento del task
- Un **insieme di attributi** che inglobano le informazioni di stato del task

Uso di un **timer hardware**, programmato con il periodo ΔT , alla cui scadenza viene attivata una procedura:

Soluzioni adottate

- Uso di una variabile condivisa (globale) in polling
- Procedura di *callback* associata al timer
- Invocazione di una procedura bloccante di *attesa* dell'evento di timer

In tutti i casi, sono necessarie un insieme di funzioni di libreria, o un sistema operativo, che supportino la gestione del timer nel modo specifico

Interrupt Service Routine + Polling di variabile condivisa

```
bool timer_elapsed ← false;
```

```
TimerISR () begin
```

```
| timer_elapsed ← true;
```

```
end
```

```
Main () begin
```

```
| while True do  
| | if timer_elapsed then  
| | | timer_elapsed ← false;  
| | | // Do the control tasks  
| | end  
| end
```

```
end
```

Procedura di Callback associata al Timer

```
TimerCallback () begin
```

```
| // Do the control tasks  
end
```

```
Main () begin
```

```
| SetTimerCallback( $\Delta T$ , TimerCallback);  
| // ... do other things  
end
```

Chiamata bloccante di attesa evento di Timer

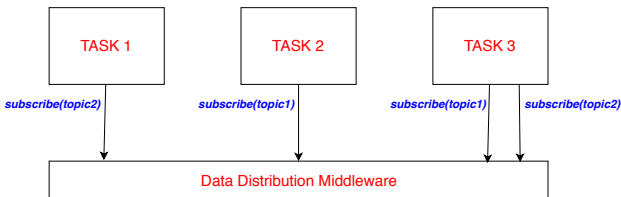
```
Main () begin
|   SetupTimer( $\Delta T$ );
|   while True do
|       |   WaitTimerEvent();
|       |   // Do the control tasks
|   end
end
```

Soluzioni adottate

- **Uso di variabili globali, con sezioni critiche qualora l'accesso sia effettuato da task concorrenti**
- Riferimenti tra oggetti (se il sistema è modellato a oggetti), con sezioni critiche qualora sia necessario
- **Uso di un middleware di comunicazione**

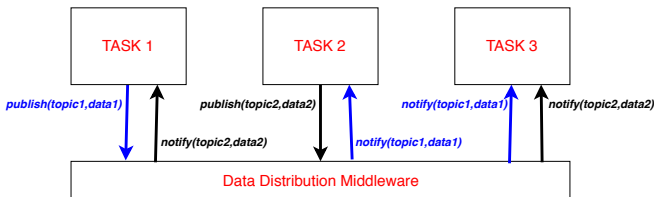
Modello Publisher/Subscriber (Data Distribution Model)

- I dati da scambiare sono modellati attraverso **tipi strutturati** e identificati da un **topic**
- I task interessati a un certo *topic* effettuano una chiamata di **subscribe** su quel topic specifico



Modello Publisher/Subscriber (Data Distribution Model)

- I task che producono un dato con un certo *topic* effettuano una chiamata di **publish**, specificando *topic* e *dati*
- I task che si erano sottoscritti su quel topic riceveranno una **notify** con i dati che sono stati pubblicati



- **uORB:** middleware di comunicazione per sistemi embedded (centralizzato)
- **CORBA-DDS (Data Distribution Service):** middleware di comunicazione basato sullo standard dell'Object Management Group (centralizzato e distribuito)
- **ROS (Robot Operating System):** middleware di comunicazione progettato appositamente per sistemi robotici (centralizzato e distribuito)

Sistema di Controllo con Modello Publisher/Subscriber



Task S1

while *True* **do**

 On each ΔT_1 ;

$in1 \leftarrow read_input1();$

$out1 \leftarrow compute_S1();$

$publish("mydata", out1);$

end

Task S2

$subscribe("mydata");$

while *True* **do**

$in2 \leftarrow wait_data("mydata");$

$out2 \leftarrow compute_S2();$

$write_output2(out2);$

end

Sistemi Operativi Real-Time

- L'esecuzione di task di controllo richiede **tempi certi**, altrimenti le conseguenze possono essere catastrofiche (soprattutto se il sistema è safety-critical)
- Se ΔT_c è il tempo di esecuzione "worst-case" di un task e ΔT il suo periodo, allora deve accadere che $\Delta T_c < \Delta T$
- Tuttavia il tempo rimanente $\Delta T - \Delta T_c$ deve essere tale da consentire al sistema di svolgere eventuali ulteriori attività

Vincolo di “Fattibilità”

- Vincolo di schedulazione possibile di N task periodici:

$$\sum_{i=1}^N \frac{\Delta T_{c_i}}{\Delta T_i} < 1$$

dove ΔT_{c_i} è il **worst-case execution time** del task i e ΔT_i è il **periodo** del task i

Caratteristiche degli Schedulatori nei Sistemi Real-Time Multi-task

- La schedulazione multi-task è effettuata sulla base della **priorità** dei task
- La priorità **non cambia** sulla base dell'uso recente del tempo di CPU, così come accade invece nei sistemi operativi "general-purpose"
- La **priorità** è invece assegnata (staticamente o dinamicamente) sulla base delle **caratteristiche temporali** del task stesso:
 - **RM** Rate Monotonic
 - **DM** Deadline Monotonic
 - **EDF** Earliest Deadline First

Politiche di Schedulazione nei Sistemi Real-Time

- **Round-Robin (RR) con Priorità:** Lo schedulatore sceglie il task con priorità più alta e lo manda in esecuzione, prelazionandolo al successivo “scheduling tick” (o se il task va autonomamente in “sleep” a seguito di una chiamata di sistema bloccante)
- **FIFO (con priorità):** Lo schedulatore sceglie il task con priorità più alta e lo manda in esecuzione; il task viene prelazionato solo se esso effettua esplicitamente di una chiamata di sistema bloccante

Alcuni SO Real-Time

- **FreeRTOS.** Kernel real-time per sistemi embedded (open-source)
- **NuttX.** Kernel real-time per sistemi embedded (open-source, usato negli autopiloti dei droni)
- **RTAI.** Kernel Linux real-time (open-source)
- **QNX.** Kernel real-time Unix-like (proprietario)
- **VxWorks.** Kernel real-time Unix-like (proprietario)

Aspetti Implementativi del Software dei Sistemi di Controllo

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory

Dipartimento di Matematica e Informatica - Università di Catania, Italy

santoro@dmf.unict.it



Programmazione Sistemi Robotici