Università degli studi di Trieste
Facoltà di Ingegneria
Dipartimento di Elettrotecnica, Elettronica ed Informatica

Sviluppo di un Driver per il Controllo di un Robot Mobile in Ambiente Multipiattaforma

Tesi di Laurea in
PROGRAMMAZIONE DEI CALCOLATORI

Laureando:
Damiano Vittor

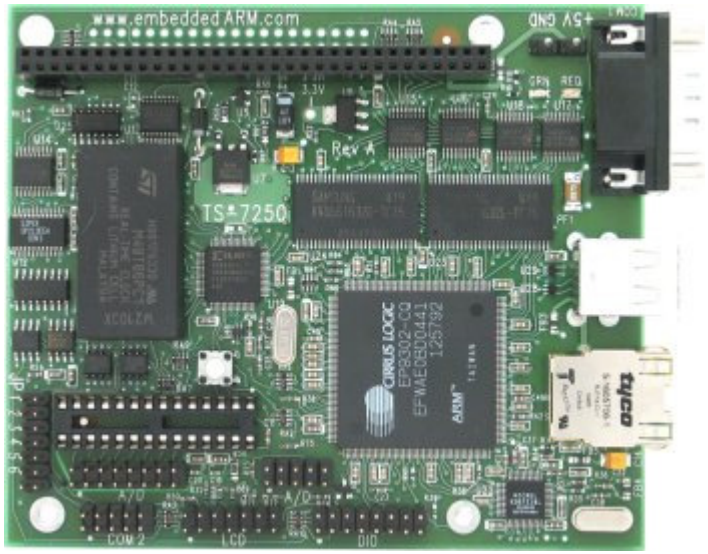
Relatore:
Dott. Ing. Massimiliano Nolich

Anno Accademico 2007-2008

Introduzione

- Sviluppo di Controlli Per Robot Mobili
- Portabilità
- Riutilizzabilità del codice
- Elaborazione distribuita
- Hardware Abstraction Layer

Hardware



Piattaforma di Sviluppo

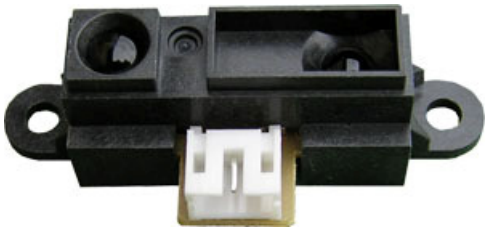
- Piattaforma Dedicata TS-7250 (ARM9)
- Linux 2.4 + Estensione RealTime

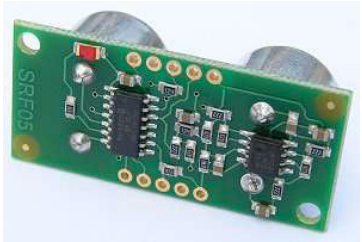
Periferiche

- Porte Digital I/O
- SPI
- ADC 5 Canali
- Ethernet
- 2 Seriali
- Controller USB con 2 porte
- Bus PC104
- 32Mb Memoria Flash

Sensori

- Infrarosso
  - Accesso tramite ADC
- Ultrasuoni
  - Logica integrata nel modulo
  - Accesso tramite Digital I/O
- Odometrici
  - Accesso tramite Digital I/O





Attuatori

- Circuito Wirz#203 modula tensione motori
  - Accesso tramite porte Digital I/O
  - Comando con segnale PWM

Il problema

Obiettivi:

Sviluppare una logica di controllo robotico

Requisiti:

- GESTIONE DELL'HARDWARE IN TEMPO REALE
- FACILMENTE PORTABILE
- INDIPENDENTE DALLA PIATTAFORMA E DALL'HARDWARE
- RIUTILIZZABILE

La soluzione adottata

- Portabilità
- Indipendente dalla piattaforma e dall'hardware
- Riutilizzabilità del codice

Adozione di Player

- Gestione dell'hardware in tempo reale

Patch RTAI per il Kernel Linux

RTAI

Sviluppato dal Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano.

Comprende

- Una patch per kernel LINUX.
- Una serie di Moduli del Kernel e di librerie per lo sviluppo.

Caratteristiche

- Introduce un nuovo schedulatore.
- Programma i processi (task) in maniera precisa e deterministica.

RTAI Utilizzo

Kernel ARM cross-compilato con estensioni RTAI.

Sono stati sviluppati 4 task per:

- il controllo dei attuatori
- l'utilizzo dei sensori

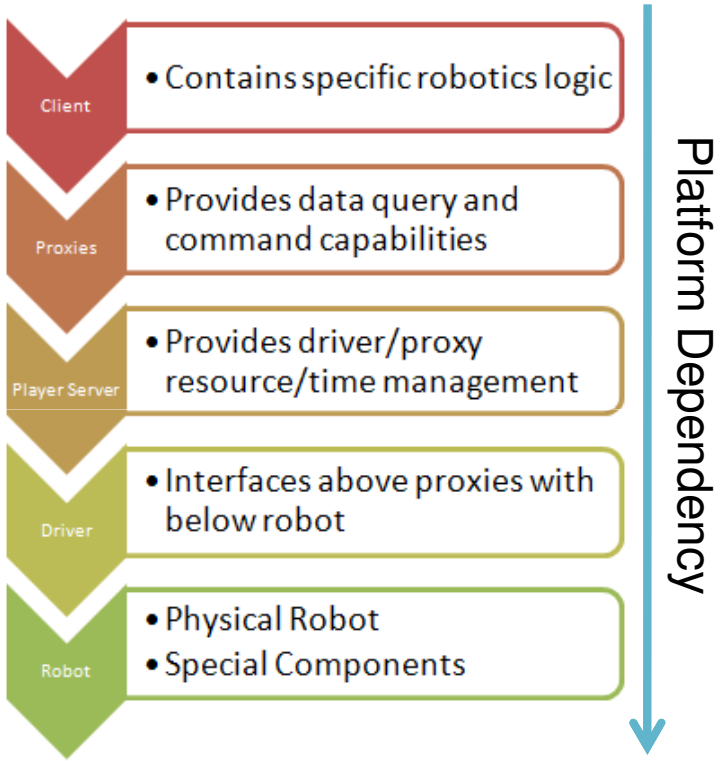
Si presentano come dei moduli per il kernel.

Task RTAI - Esempio

```
int init_module(void){
   rt_set_periodic_mode();
   rt_task_init(&rt_task, sonar, 0, STACK_SIZE, TASK_PRIORITY, 0,
   0);
   rtf_create(FIFO, 10);
   rt_set_oneshot_mode();
   start_rt_timer(0);
   rt_task_make_periodic_relative_ns(&rt_task, 1000000LL,
   TICK_PERIOD);
   return 0;
}
static void sonar(int t){
   //prendi le informazione dal hardware e mettile nella FIFO
   rtf_put(FIFO,&distance, sizeof(distance));
}
void cleanup_module(void){
   stop_rt_timer();
   rtf_destroy(FIFO);
   rt_task_delete(&rt_task);
}
```

Player

- Interfaccia per il controllo di dispositivi robotici
- Utilizzato per il controllo di robot sia per la simulazione
- Inizialmente sviluppato alla USC. Progetto ad ampia diffusione nel settore.



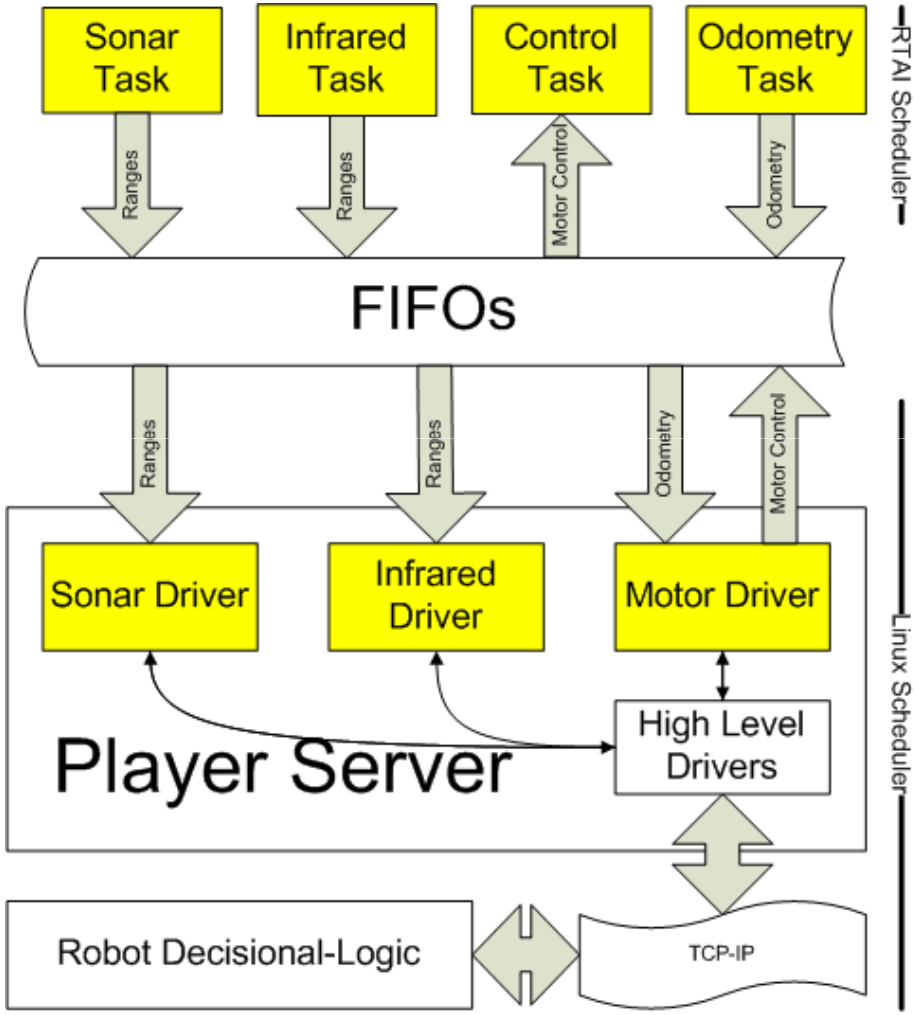
E' stato cross-compilato per funzionare sulla Piattaforma ARM insieme alle librerie da cui dipende.

Player Caratteristiche

- Architettura modulare
- Definisce interfacce per le più comuni periferiche robotiche. (HAL)
- Elementi atomici chiamati driver che si collegano tra loro tramite le interfacce.
- Sistema di messaggistica comune.
- Libreria di Driver di alto livello già sviluppati.
- Serie di Tool per lo sviluppo di applicazioni robotiche
- Progetto Open Source con Licenza GNU

Architettura del Driver Sviluppato

- Task real-time per la gestione dell'hardware
- FIFO come IPC
- Player eseguito come processo di Linux



Moduli RTAI

- Inizializzazione dispositivi
- Schedulare letture dati sensori
- Generare il segnale PWM per i circuito di controllo motori
- Comunicazione tramite FIFO
- Nessuna elaborazione dati

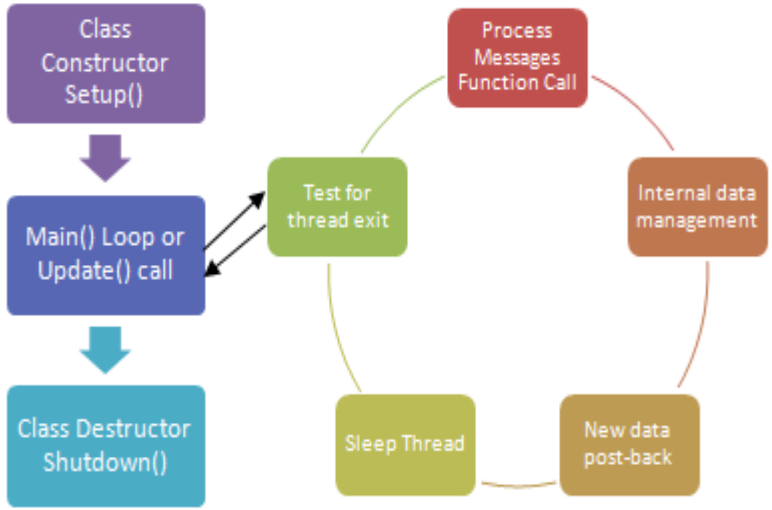
Driver Player

- Inserimento e rimozione moduli RTAI
- Lettura dati sensori (tramite FIFO)
- Elaborazione dati sensori
- Controllo motori (non sviluppato) (tramite FIFO)

Esempio - Driver Player

```
int Ir2d120x::Setup() {
  //calculating angular coefficients
for the voltage/distance function
  buildM();
  //starting task
  system("insmod rt_irda.o");
  //opening fifo...
  if ((fifo_irda = open(device,
O_RDONLY))< 0){ …  }
 StartThread();
}

int Ir2d120x::Shutdown() {
  StopThread();
  system("rmmod rt_irda.o");
  if (close(fifo_irda)< 0){ … }
  return 0;
}
 //questa è la parte che viene eseguita quando il modulo è caricato
 extern "C" {
 Int player_driver_init(DriverTable* table)
   {
     //registra il driver dentro il framework Player informandolo su come
          instanziare un oggetto di questo driver.
     Ir2d120x::Ir2d120x_Register(table);
   }
 }
```



Esempio - Corpo del Thread

```
//Controlliamo se dobbiamo interrompere il thread
pthread_testcancel();

// Elaboriamo i messaggi in attesa: ExampleDriver::ProcessMessage() è
   chiamato per ogni messaggio.
ProcessMessages();

//Leggo da fifo e interagisco con l'hardware
  read(fifo_irda, &bufValue, sizeof(bufValue));
  <qui elaboro i dati e calcolo la distanza>

// Pubblico il risultato con Driver::Publish()
Publish(device_addr, PLAYER_MSGTYPE_DATA, PLAYER_IR_DATA_RANGES,
   (unsigned char*)&irData, sizeof(player_ir_data_t),NULL);

  // Sleep (interrompo il thread per un certo tempo)
usleep(100000);
```

Strumenti Utilizzati

- C/C++
- Kdevelop e Make come IDE e BuildTool
- Cross-Toolchain per l'ARM fornita.
- Linux Ubuntu (8.04) Virtualizzato in Windows Vista
- VirtualBox
- file-system su NFS per lo sviluppo

Risultati Ottenuti

- Player cross-compilato su piattaforma embedded ARM.
- Ricompilato Kernel con Estensioni RTAI
- Driver Sonar e Infrarosso e relativi moduli RTAI funzionanti.
- Moduli real-time per odometri e motori non testati per mancanza hardware.
- File-System esteso Memoria FLASH

Conclusioni

- In questa tesi sono stati sviluppati:
  - Moduli RTAI per la gestione di ultrasuoni, infrarosso, odometri e controllo motori.
  - Driver Player per i sensori utilizzati.
- E' ora possibile sviluppare tramite Player programmi di logica robotica indipendenti dall'hardware che funzionino su quest'hardware.
- Si può simulare questi algoritmi facilmente.