Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Quinquennale in Ingeneria Informatica

tesi di laurea

Interfacciamento telecamera per tracking

Relatore: prof. Emanuele Menegatti
Correlatore: ing. Stefano Ghidoni

Laureando: Fabio Rossignoli

26 Ottobre 2010

Alla mia famiglia,
nel suo significato più ampio.

Sommario

Questa tesi tratta dell'integrazione del brandeggio Ulisse Compact 2, prodotto dall'azienda Videotec S.p.A., nel sistema di tracking per videosorveglianza implementato grazie ad un middleware (NMM 4). Un brandeggio è un supporto per telecamera che può ruotare contemporaneamente in senso orizzontale e verticale, lungo assi denominati pan e tilt, mentre un middleware, in breve, è un software di interfaccia tra applicazioni e risorse di una rete.

Il lavoro, svolto nello IAS Lab (Intelligent Autonomous Systems Laboratory) dell'Università di Padova, è partito dall'installazione della connessione seriale tra la telecamera ed il PC, così da permetterne la reciproca comunicazione. Si sono fatti i dovuti test di funzionamento attraverso l'interfaccia seriale CuteCom e si è poi implementato il protocollo di comunicazione dell'Ulisse Compact, denominato "Protocollo MACRO", in linguaggio di alto livello. Una volta ottenuti questi strumenti base di comunicazione, si è proceduto alla scrittura di un'interfaccia che permettesse ad un normale utente il totale controllo dell'apparecchiatura. A questo punto si è iniziato il lavoro di integrazione nel software di tracking che ne ha richiesto inizialmente lo studio teorico ed in seguito la completa comprensione del codice che lo implementa. Si sono quindi incapsulati i metodi di controllo precedentemente preparati per l'Ulisse nel codice sorgente del pacchetto di videosorveglianza e fall detection. Nel contempo si avviava lo studio e l'installazione del middleware NMM (Network-Integrated Multimedia Middleware) fino a renderlo operativo all'interno del laboratorio, sulla stessa macchina già in comunicazione con la telecamera e con installato il tracking di cui si è parlato. L'ultimo passo è stata l'implementazione dell'Ulisse in NMM, per quanto riguarda l'inseguimento dei soggetti trackati e l'inquadramento delle cadute rilevate dallo stesso sistema. Infine si sono fatti diversi test che hanno dato risultati soddisfacenti, in particolare nella fase di inseguimento del soggetto trackato, grazie anche alla grande versatilità, soprattutto in ambito di controllo, del brandeggio (capacità di posizionamento secondo coordinate assolute).

Tutti i software usati (GNU/Linux, Eclipse, NMM, CuteCom, LaTeX[1] [2]) sono opensource.

Indice

Capitolo 1

Introduzione

Usando una frase fatta, tanto in voga nell'ambiente dell'economia di largo consumo, si può tranquillamente affermare che la videosorveglianza è uno di quei settori che non conosce crisi.

Nata come concetto di "controllo del territorio" all'alba della civiltà, trascende oggi il semplice ambito dell'autoconservazione e diventa indispensabile strumento di sicurezza e controllo sociale.

Arricchita delle nuove tecniche e tecnologie offre ampi ambiti di ricerca e sviluppo tanto sul piano dell'invenzione di nuovi strumenti, quanto nella creazione di un'intelligenza artificiale in grado di sostituire, seppur limitatamente, quella umana.

Dall'unione dei moderni apparecchi di ripresa e dallo studio e implementazione degli algoritmi di elaborazione delle immagini, ecco nascere, appunto, la videosorveglianza intelligente.

1.1 La videosorveglianza

L'atto di videosorvegliare consiste nel vigilare, generalmente un luogo o comunque un bene, a distanza, tramite l'utilizzo di telecamere strategicamente posizionate.

La videosorveglianza nasce in ambito analogico e la sua storia è molto più breve di quella dei sistemi anti-intrusione, esistenti sin dall'antichità.

Il primo esperimento di TVCC (TeleVisione a Circuito Chiuso) viene attribuita all'ingegnere tedesco Walter Bruch nel 1942. L'impianto venne realizzato dalla Siemens a Peenemunde sulla rampa di lancio Prüfstand VII e serviva a monitorare i lanci dei razzi V-2 , i più sofisticati missili militari della Seconda Guerra Mondiale.

Per le prime applicazioni ad uso non militare bisogna attendere il 1965, quando gli Stati Uniti, intuirono il prezioso contributo che un sistema di videosorveglianza avrebbe potuto offrire per il controllo del territorio.

Nel Regno Unito nei primi anni '70 fu l'IRA (Irish Republican Army), a contribuire ad una diffusione massiccia degli impianti di videosorveglianza, seguita a metà del decennio alla prima rivoluzione elettronica nella videosorveglianza, con l'introduzione del CCD, un piccolo circuito integrato in grado di acquisire le immagini e di sostituire il più ingombrante e costoso tubo Vidicon.

Per quanto riguarda la registrazione video, il prototipo di registratore video su nastro magnetico fu sviluppato nel 1952 dalla Ampex con un apparecchio che utilizzava una testina rotante ed un nastro che si muoveva relativamente lento. All'inizio del 1956 la Ampex fece esordire il VR-1000 (vedi Figura 1.1), il primo della serie dei registratori su nastro video da 2 pollici Quadruplex (4 tracce). La prima trasmissione televisiva registrata magneticamente e trasmessa in differita usando il nuovo sistema di registrazione Ampex fu "Douglas Edwards and the News" della CBS il 30 novembre del 1956.

Figura 1.1: L'Ampex VR-1000

L'assemblaggio Quad Head aveva 4 testine che ruotavano a 14400 rpm. Scrivevano il segnale video verticalmente lungo la larghezza di un nastro largo 2 pollici (circa 5 cm) che si muoveva alla velocità di 15 pollici (38 cm) al secondo. Questo permetteva a programmi lunghi ore di essere registrati su una singola bobina (nel 1956 una bobina di nastro costava 300 dollari, equivalenti a circa duemila dollari nel 2000, e i registratori costavano da settantacinquemila a centomila dollari, circa mezzo milione di dollari di oggi).

Nel 1967 la Ampex introdusse il VR-3000, un registratore video portatile, che rivoluzionò la registrazione televisiva di alta qualità sul campo, senza necessità di lunghi cavi o veicoli di supporto. Immagini televisive potevano essere raccolte ovunque, anche da aerei, elicotteri o barche. Il primo sistema che sostituiva le

ingombranti e scomode bobine con le prime cassette compatte contenti un nastro magnetico, fu introdotto dalla Sony verso la fine degli anni '60, ma il sistema era ancora poco pratico. Nel 1972 la Philips mise sul mercato l'N1500, probabilmente l'antenato dei sistemi di videoregistrazione domestica, di dimensioni molto più contenute, perciò adatto per le applicazioni di uso quotidiano. Tuttavia il sistema era ancora molto costoso e la durata delle registrazioni era piuttosto limitata, infatti su una cassetta era possibile registrare non oltre trenta minuti di video. Il 1° Giugno 1975 la Sony lanciò sul mercato giapponese i primi due modelli di Betamax, il lettore SL-6300 e la console LV-1801. Dopo un anno e trentamila prodotti venduti però, arrivò il VHS della JVC, apparentemente simile, ma in realtà molto diverso dal Betamax. Le cassette erano di maggiori dimensioni, la qualità era decisamente più scadente ma la capacità di immagazzinamento per ogni singolo nastro arrivò a toccare le quattro ore. A differenza di Sony, JVC cercò altri alleati, diffondendo e vendendo il brevetto VHS anche ad altre aziende, sia tra i produttori che tra le case cinematografiche e questo contribuì ad una solida affermazione del prodotto, ma soprattutto a mantenere i prezzi del VHS più bassi rispetto al concorrente. Pochi anni più tardi comparvero i primi registratori VHS "time laps", con i quali divenne possibile registrare in maniera economica, fino a 960 ore di filmati su una stessa videocassetta. Nel 1999 comparvero sul mercato i primi DVR, apparati in grado di acquisire un video e archiviarlo su hard-disk.

Iniziò così l'era della videosorveglianza come tramissione e storage di dati digitali (non più segnali analogici).

Il sistema collegamento più diffuso per applicazioni CCTV analogiche è di fatto il cavo Coassiale RG59, con crimpatura di terminali a connettore BNC .

La trasmissione digitale di dati video CCTV si sviluppa invece sullo standard Ethernet, che utilizza cavi in rame multicoppia twistati UTP e STP per la trasmissione dati su protocollo TCP/IP e UDP.

Grazie all'utilizzo del protocollo universalmente riconosciuto come standard per la tramissione dati, l'evoluzione della videosorveglianza digitale (chiamata in breve IP) ha permesso la completa integrazione di questi sistemi nelle reti di calcolatori moderne, favorendone la diffusione (vedi Figura 1.2).

Queste ultime evoluzioni hanno favorito il passaggio dalla videosorveglianza, rimasta per anni relegata alla "semplice" videoregistrazione di flussi video, alla videosorveglianza intelligente, che forte della possibilità di elaborazione delle immagini riprese, apre la strada a nuove tecnologie, ad oggi ancora quasi assenti nelle soluzioni commerciali, di videocontrollo.

1.2 Contesto applicativo

Il lavoro di tesi va ad inserirsi in un più ampio progetto, teso a realizzare un completo sistema di tracking mediante videocamere. L'obiettivo complessivo è quello di ottenere un software commerciale che implementi una videosorveglianza

Figura 1.2: Esempi di strumentazione per TVCC collegati tra loro.

intelligente, sempre più libera dal controllo umano e più autonoma. Seppur lontani dall'avere un sistema autosufficiente, si tende a sostituire l'operatore per le situazioni nelle quali gli errori o i limiti fisici umani comportano dei vincoli se non addirittura dei fallimenti.

Un primo esempio è quello del controllo di vaste aree o numerosi locali. È ovvio che in tale circostanza, un solo operatore risulta insufficiente, in quanto gli sarebbe richiesta una continua concentrazione (fisicamente impossibile) e il controllo di numerosi monitor che inevitabilmente sarebbero guardati uno per volta; inoltre l'aumento del personale comporterebbe certamente un costo non trascurabile. In questi casi l'autonomia del software permetterebbe un controllo in real-time, anche parallelo, di tutti gli ambienti.

Un secondo esempio è quello del riconoscimento di determinate situazioni. Essendo la maggior parte dei sistemi di sorveglianza basati su registrazioni a circuito chiuso senza la presenza di operatori, succede che la verifica di eventi magari importanti per il controllo di un territorio, avviene a posteriori dell'accadimento degli stessi. Sostanzialmente quel che si fa oggi è di prendere la regitrazione e riguardarla, cercando quel che interessa. Un sistema autonomo sarebbe in grado di rilevare prontamente questi eventi e di procedere in modo adeguato, avvisando le autorità se riconosciuto ad esempio uno sparo o chiamado i soccorsi se riconosciuta una pesona che cade a terra.

All'interno del laboratorio IAS Lab dell'Università di Padova sono installati tre apparecchi di ripresa:

1. una telecamera panoramica;

2. una telecamera PTZ solidale con la panoramica (Figura 1.3);

3. una telecamera Ulisse Compact (cfr. cap. 2).

Figura 1.3: La telecamera panoramica OmniDome e la PTZ solidali sullo stesso palo.

Lavori già svolti, hanno permesso l'integrazione delle prime due apparecchiature in un software di tracking basato su sottrazione dello sfondo, mentre l'ultima, acquistata di recente, è oggetto di questa tesi. Prodotta dall'azienda Videotec S.p.A., è un brandeggio ad alte prestazioni per l'utilizzo in ogni tipo di ambiente.

Il calcolatore su cui si opera è equipaggiato con un processore Intel Core 2 6400, 3 GB di RAM, scheda video NVIDIA GeForce 7600 GS e HD da 230 GB su cui è installato il sistema operativo GNU/Linux (distribuzione Ubuntu 9.10 Karmic Koala), versione del kernel 2.6.31.

Esso è dotato di due framegrabber PCI Brooktree Corporation (Bt878), utilizzati per la cattura del flusso video e di porte USB standard per la comunicazione seriale (Figura 1.5).

1.3 Obiettivi

Gli obiettivi di questa tesi sono:

1. implementazione di un'interfaccia di comunicazione con la telecamera Ulisse Compact;
2. implementazione del protocollo di comunicazione della telecamera Ulisse Compact;
3. integrazione con software di tracking esistente;
4. installazione e configurazione del middleware NMM per la trasmissione video su generica rete di comunicazione;
5. interfacciamento della telecamera Ulisse Compact nel middleware, nella parte di "fall detection" esistente;
6. interfacciamento della telecamera Ulisse Compact nel middleware, ai fini dell'ottenimento di un "following" continuo del soggetto trackato;
7. implementazione di un metodo "point and click" che permette, clickando sull'immagine visualizzata, di far spostare il brandeggio in modo che visualizzi quella zona.

1.4 Lavoro di tesi

Il lavoro di tesi si concentra su tre punti principali:

- il controllo del brandeggio e dello zoom della videocamera;
- l'integrazione della videocamera nel software di tracking;
- l'integrazione della videocamera nel middleware NMM.

1.5 Ipotesi di progetto

In questo progetto si suppone di avere un sistema di tracking basato su telecamere già funzionante, che fornisce una visione del mondo attraverso una mappa 2D all'interno della quale sono identificati i target. Questa è realizzata a partire dai frame della telecamera omnidirezionale OmniDome, che genera una visione a 360° dell'ambiente circostante (Figura 1.4), elaborata al fine di ottenere un immagine rettangolare da quella tonda originale.

Pur non essendo la rilevazione del movimento sempre affidabile, si ipotizza che i target cui far convergere l'Ulisse, siano sempre "buoni" e non ci si porrà mai il problema della correttezza del matching.

Si suppone inoltre di conoscere la geometria e della mappa e dell'ambiente, ricavando così, all'occorrenza, le posizioni assolute di oggetti e telecamere.

Infine l'area ristretta della sperimentazione, il laboratorio, permette di tollerare errori in posizione piuttosto ampi, essendo la porzione di mondo da catturare piccola rispetto l'ampia finestra della telecamera.

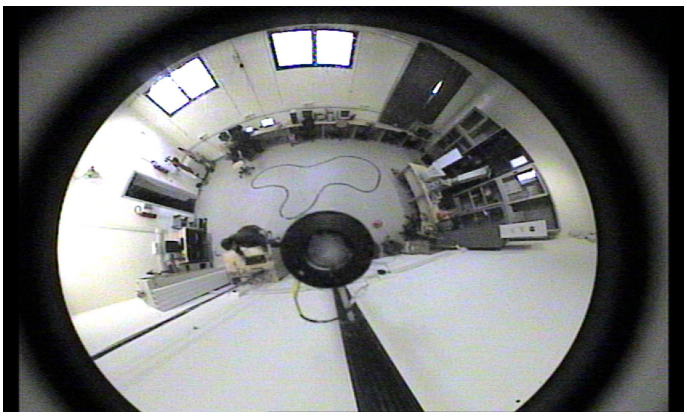


Figura 1.4: L'immagine generata dall'OmniDome.

1.6 Materiale già presente

Come già citato, si sfrutterà un software di tracking preesistente, all'interno del quale inserire i nuovi moduli. Le videocamere OmniDome (panoramica) e PTZ sono già installate e funzionanti, mentre l'Ulisse Compact, seppur già fissato al soffitto del laboratorio, necessita della costruzione sia fisica che software, del collegamento seriale.

Si provvederà anche all'installazione e configurazione del middleware NMM sul calcolatore descritto inizialmente, in cui da terzi verrà incluso il tracking e col presente lavoro sarà invece implementato l'interfacciamento con la nuova telecamera.

Figura 1.5: La postazione di lavoro con PC al quale sono collegate tutte le telecamere.

Capitolo 2

La telecamera Ulisse Compact

Introduzione

Figura 2.1: Il brandeggio Ulisse Compact della ditta Videotec S.p.A. installato sul soffitto del laboratorio.

Il brandeggio usato è prodotto dalla ditta Videotec S.p.A. ed è denominato Ulisse Compact (Figura 2.1).

È un brandeggio ad alte prestazioni con possibilità di movimenti pan e tilt associati ad una precisa telemetria. In grado di offrire una continua rotazione, anche ad elevata velocità, possiede un'alta qualità video ed un menù di configurazione relativamente semplice.

Velocità e precisione sono le caratteristiche fondamentali di questa apparec-

chiatura, in grado di raggiungere una velocità orizzontale di 200°/s in rotazione continua ed ha un range verticale da -90° a +90°.

Inoltre vi è la possibilità di gestire posizioni predeterminate in maniera assoluta e, in generale, avere un'accuratezza nel movimento dell'ordine del decimo di grado. Una logica di controllo interna imposta una curva di accelerazione adeguata per non danneggiare i mototi passo-passo.

L'unità è in grado di verificare continuamente in modo autonomo la propria posizione, per prevenire o correggere spostamenti accidentali, quali ad esempio quelli dovuti agli agenti atmosferici.

La telecamera Sony è a colori, ha risoluzione 720x576 e lo zoom arriva a 36x con una lunghezza focale variabile da 3.4 mm a 122.4 mm, consentendo un angolo di campo massimo di 57.8° e minino di 1.7°, apertura massima variabile tra f/1.6 e f/4.5 e fornendo un'eccezionale precisione sia con gli oggetti vicini che quelli lontani. È possibile dare un valore predefinito di ingrandimento e questo verrà raggiunto nel minor tempo possibile.

Il sensore CCD Super HAD (versione da zoom ottico 10x) ha dimensione 1/3 di pollice e garantisce una grande sensibilità anche in ambienti scarsamente illuminati.

Si ha la possibilità di creare maschere personalizzate per la ripresa.

L'Ulisse è equipaggiato di OSD (On Screen Display) per la configurazione, un'interfaccia RS485/RS422 per un totale controllo del sistema e la possibilità di update remoti del firmware.

Le impostazioni permettono la modifica del bilanciamento del bianco, dell'esposizione e molto altro.

Da notare che, sia per quanto riguarda pan e tilt, sia per quanto riguarda lo zoom, è permesso lo spostamento continuo, o l'ingrandimento continuo fino all'immissione di un esplicito comando di stop. Nel caso quest'ultimo non fosse inviato, il valore della grandezza in questione arriverà a fine scala dove automaticamente si fermerà.

In definitiva l'Ulisse Compact è una telecamera ideale per la videosorveglianza, per il controllo anche di grandi aree e possiede una grande flessibilità che permette anche la sperimentazione di nuove tecniche e tecnologie che con altri hardware meno versatili non sarebbe possibile fare [3].

2.1 Collegamento

La connessione fisica dell'Ulisse ha comportato un discreto impiego di tempo.

Questo perché in laboratorio, pur essendo il brandeggio già fissato al soffitto e predisposto con un cavo di rete Ethernet come mezzo per la comunicazione seriale, mancava di documentazione circa le modalità di collegamento. Si è dunque resa necessaria una verifica empirica, effettuata stabilendo un set di prove con l'uso di un multimetro, per riuscire a stabilire la giusta configurazione dei

fili per il successivo interfacciamento con una porta seriale utile alla connessione al PC.

Sono quindi stati identificati i seguenti fili in tensione (circa 495 mV):

- verde;
- bianco-verde;
- arancione;
- bianco-arancione.

Tra gli altri 4 fili, si sono potute identificare le 2 masse, poiché i restanti 2 fili erano evidentemente flottanti:

- azzurro;
- bianco-azzurro.

Per inciso, restano quindi inutilizzati il marrone e il bianco-marrone (Figura 2.2).

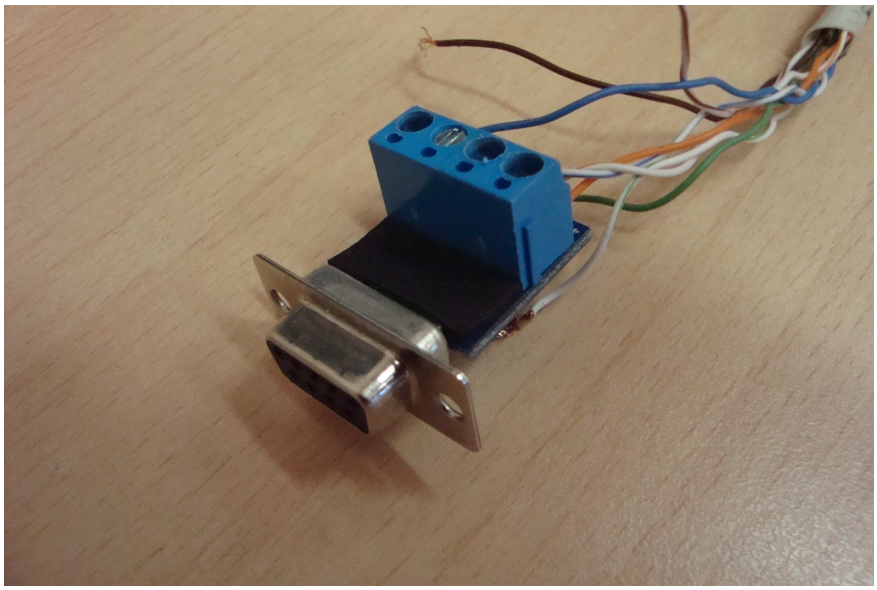


Figura 2.2: Il collegamento seriale dell'Ulisse Compact.

Dalle specifiche si sa che i collegamenti seriali possono essere 2; si sono così stabilite le griglie per le prove, al fine di stabilire a quali combinazioni di fili afferissero le due porte:

+	-	GND	funziona (sì/no)
verde	arancione	azzurro	no
arancione	verde	azzurro	no
bianco-verde	bianco-arancione	bianco-azzurro	no
bianco-arancione	bianco-verde	bianco-azzurro	no

Tabella 2.1: Prima griglia di prove.

+	-	GND	funziona (sì/no)
verde	bianco-verde	azzurro	no
verde	bianco-verde	bianco-azzurro	no
bianco-verde	verde	azzurro	no
bianco-verde	verde	bianco-azzurro	no
arancione	bianco-arancione	azzurro	no
arancione	bianco-arancione	bianco-azzurro	sì
bianco-arancione	arancione	azzurro	non testato
bianco-arancione	arancione	bianco-azzurro	non testato

Tabella 2.2: Seconda griglia di prove.

Alla fine si è proceduto al collegamento fisico dei fili secondo questo schema (Figura 2.3):

- 495+ arancione;
- 495- bianco-arancione;
- GND bianco-azzurro.

Grazie all'utilizzo di un'interfaccia ci comunicazione seriale chiamata CuteCom si sono effettuate delle prove di trasmissione, inviando stringhe appartenenti al Protocollo MACRO di comunicazione seriale dell'Ulisse Compact al fine di stabilire il corretto funzionamento di tutto il sistema.

Si è quindi collegata la seriale ad un crosscable SERIALE->USB per l'utilizzo sotto GNU/Linux (Figura 2.4).

2.2 Configurazione

Preliminarmente all'utilizzo, l'Ullisse Compact prevede una fase di configurazione eseguibile attraverso degli switch posti sul brandeggio stesso e accessibili mediante un piccolo portellino su di un lato.

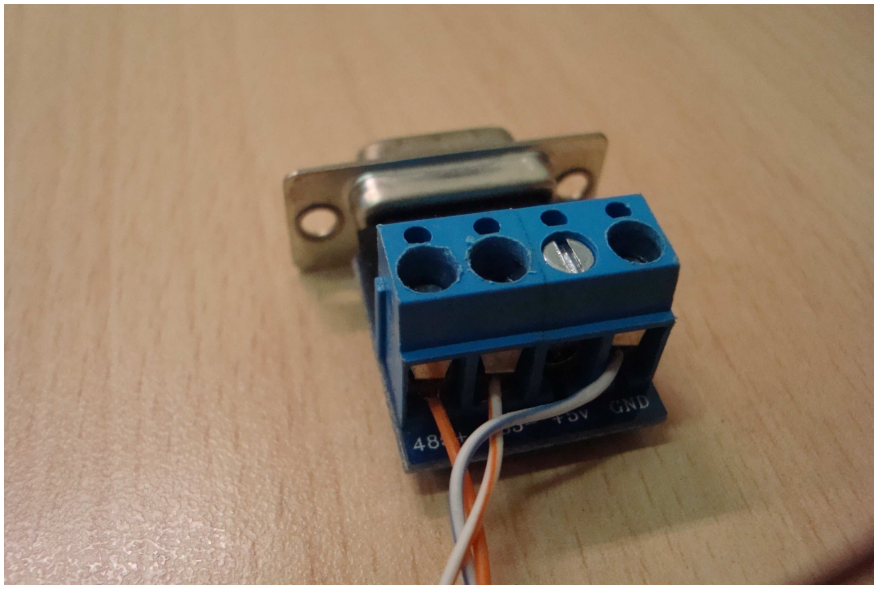


Figura 2.3: Il collegamento dei fili alla porta seriale.

Configurazioni scelte per i dip-switch:

DIP1

OFF	ON	ON	ON	ON	ON	ON	ON
1	2	3	4	5	6	7	8

1 = ON –> VISUALIZZAZIONE CONFIGURAZIONE ABILITATA (non si può muovere la telecamera).

1 = OFF –> VISUALIZZAZIONE DISABILITATA (posso comandare la telecamera).

2 = ON + 3 = ON + 4 = ON –> VELOCITÀ IMPOSTATA A 38400 baud.

5 = ON + 6 = ON –> MODO RS485 BIDIREZIONALE HALF DUPLEX (quindi seriale 1 “RS485-1” abilitata in full duplex e seriale 2 “RS485-2” disabilitata).

7 = ON –> TERMINAZIONE RS485-1 ABILITATA.

8 = ON –> TERMINAZIONE RS485-2 ABILITATA.

DIP2

OFF	ON	ON	ON	ON	ON	ON	ON	ON	ON
1	2	3	4	5	6	7	8	9	10

1 = ON + 2 = OFF + 3 = OFF + 4 = OFF + 5 = OFF + 6 = OFF + 7 = OFF + 8 = OFF + 9 = OFF + 10 = OFF –> INDIRIZZO TELECAMERA È “1” (cioè uso l’identificatore “TM1” per comandarla).

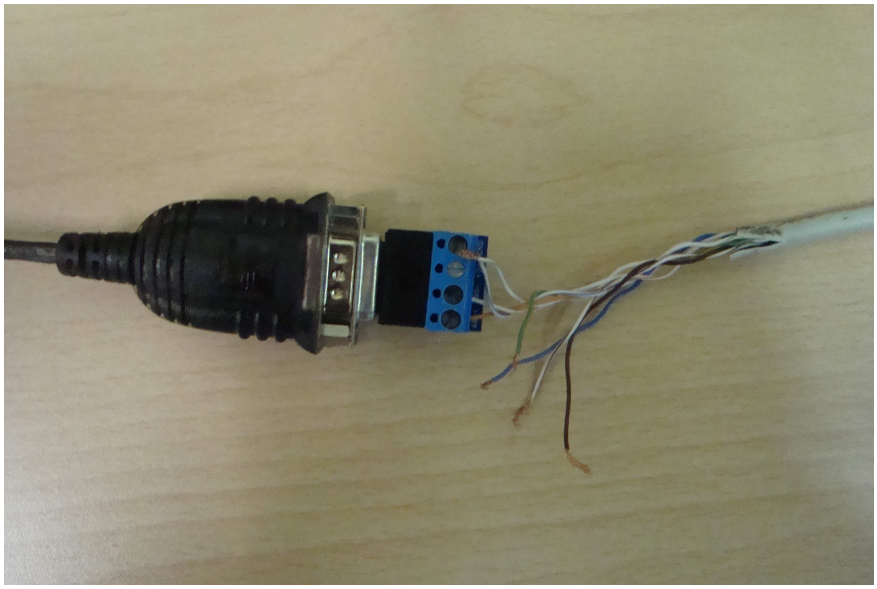
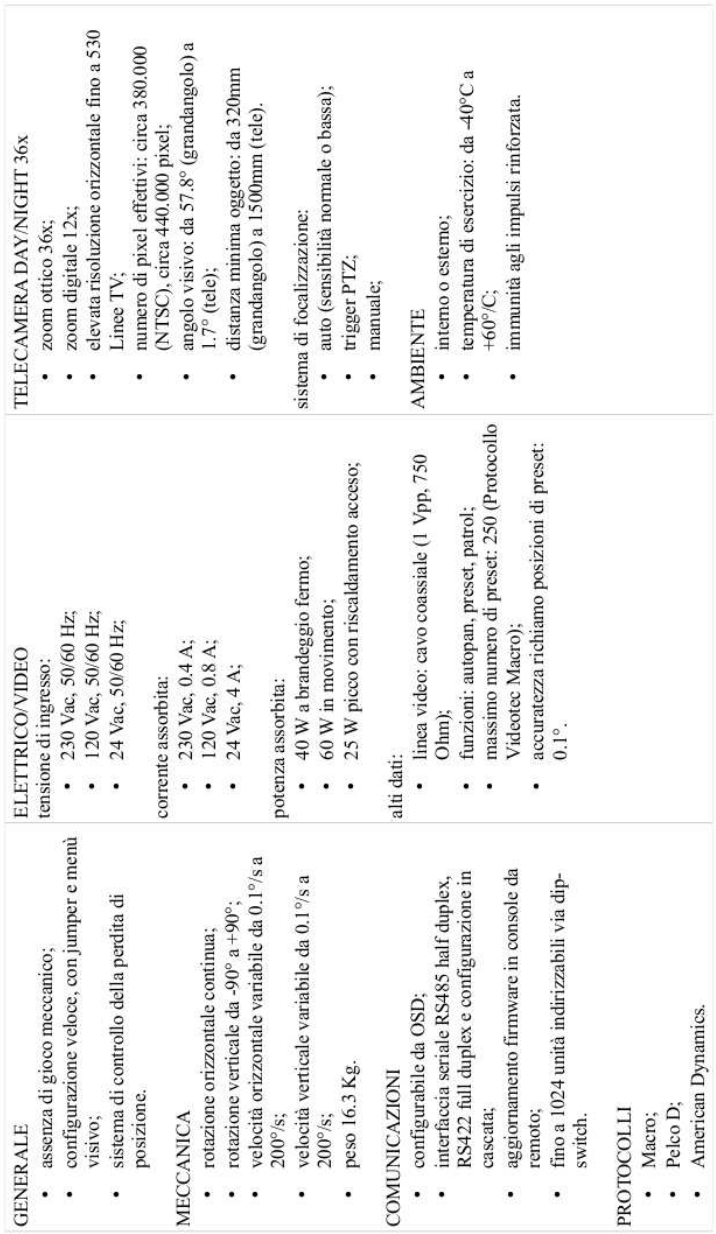


Figura 2.4: Il collegamento della seriale RS485 con la RS422 per la connessione al PC.

DIP3			
OFF	OFF	OFF	OFF
1	2	3	4

1 = OFF + 2 = OFF + 3 = OFF + 4 = OFF –> USO DEL PROTOCOLLO MACRO

2.3 Datasheet



DATA SHEET

GENERALE
- assenza di gioco meccanico;
- configurazione veloce, con jumper e menù visivo;
- sistema di controllo della perdita di posizione.

MECCANICA
- rotazione orizzontale continua;
- rotazione verticale da -90° a +90°;
- velocità orizzontale variabile da 0.1°/s a 200°/s;
- velocità verticale variabile da 0.1°/s a 200°/s;
- peso 16.3 Kg.

COMUNICAZIONI
- configurabile da OSD;
- interfaccia seriale RS485 half duplex, RS422 full duplex e configurazione in cascata;
- aggiornamento firmware in console da remoto;
- fino a 1024 unità indirizzabili via dip-switch.

PROTOCOLLI
- Macro;
- Pelco D;
- American Dynamics.

ELETTRICO/VIDEO
tensione di ingresso:
- 230 Vac, 50/60 Hz;
- 120 Vac, 50/60 Hz;
- 24 Vac, 50/60 Hz;

corrente assorbita:
- 230 Vac, 0.4 A;
- 120 Vac, 0.8 A;
- 24 Vac, 4 A;

potenza assorbita:
- 40 W a brandeggio fermo;
- 60 W in movimento;
- 25 W picco con riscaldamento acceso;

alti dati:
- linea video: cavo coassiale (1 Vpp, 750 Ohm);
- funzioni: autopan, preset, patrol;
- massimo numero di preset: 250 (Protocollo Videotec Macro);
- accuratezza richiamo posizioni di preset: 0.1°.

TELECAMERA DAY/NIGHT 36x
- zoom ottico 36x;
- zoom digitale 12x;
- elevata risoluzione orizzontale fino a 530 Linee TV;
- numero di pixel effettivi: circa 380.000 (NTSC), circa 440.000 pixel;
- angolo visivo: da 57.8° (grandangolo) a 1.7° (tele);
- distanza minima oggetto: da 320mm (grandangolo) a 1500mm (tele).

sistema di focalizzazione:
- auto (sensibilità normale o bassa);
- trigger PTZ;
- manuale;

AMBIENTE
- interno o esterno;
- temperatura di esercizio: da -40°C a +60°/C;
- immunità agli impulsi rinforzata.

Figura 2.5: Datasheet Ulisse Compact.

Capitolo 3

Implementazione del protocollo

Introduzione

Il "MACRO Protocol" [4] è il protocollo fornito dall'azienda Videotec, per il controllo da remoto del brandeggio Ulisse Compact di produzione della ditta medesima.

Esso si compone di una serie di caratteri ASCII, raggruppati tra parentesi quadre "[ ]" che segnano convenzionalmente l'inizio e la fine del messaggio. Quest'ultimo sarà poi inoltrato via seriale.

Il sistema è case sensitive e nessuno spazio bianco è ammesso.

I parametri sono solitamente numerici e separati da virgole.

Non è possibile scendere ulteriormente in particolari, poiché il protocollo MACRO è protetto da copyright e la sua diffusione non è permessa. Si riporteranno quindi le linee guida generiche e si sostituiranno le stringhe di comando reali con alcune fittizie, ma che ne richiamino comunque il significato.

3.1 Interfaccia C++ di controllo

Di seguito la struttura generale dei metodi di controllo del brandeggio. [5]

```
#define MAX_BUF 50

class UlissePositioning {

  public:
    UlissePositioning(const char *device = 0);
    ~UlissePositioning();

    // spostamenti orizzontali e verticali
    void Up(int speed) {MoveCommand("SU", speed);}
    void Down(int speed) {MoveCommand("GIÙ", speed);}
```

```
    void Left(int speed) {MoveCommand("SINISTRA", speed);}
    void Right(int speed) {MoveCommand("DESTRA", speed);}

    void UpLeft(int speedX, int speedY) {MoveCommand("SU-SINISTRA", speedX, speedY);}
    void UpRight(int speedX, int speedY) {MoveCommand("SU-DESTRA", speedX, speedY);}
    void DownLeft(int speedX, int speedY) {MoveCommand("GIÙ-SINISTRA", speedX, speedY);}
    void DownRight(int speedX, int speedY) {MoveCommand("GIÙ-DESTRA", speedX, speedY);}

    void Stop(void) {SendCommandToDevice("STOP");}

    // spostamento a una posizione ben definita
    void GoToSpecificPosition(char *pan, char *tilt, char *run, char *zoom);

    // spostamento a una posizione di preset
    void GoToPresetPosition(int presetPosition);

    void SendCommandToDevice(const char command[], const char par1[] = "", \
                    const char par2[] = "");

    void SendCommandToDevice2(const char command[], const char par1[] = "", \
                    const char par2[] = "", const char par3[] = "", const char par4[]="");

    char EvaluateChecksum(const char Str[]);

  private:

    void MoveCommand(const char *command, int speed1, int speed2 = 0);

    // buffer di memorizzazione
    char buf[MAX_BUF];

    // porta utilizzata
    int port;

};
```

Le istruzioni sono certamente intuitive e non hanno bisogno di lunghe spiegazioni. Si fanno solo alcune sottolineature per quelli di maggiore interesse.

In particolare il metodo “GoToSpecificPosition” accetta quali parametri, le posizioni assolute di pan e tilt, la velocità di spostamento e il fattore di zoom. Esso permette di spostare in una posizione specifica l’Ulisse, secondo il sistema di riferimento ad esso solidale.

Ci sono poi due metodi, di poco diversi, chiamati “SendCommandToDevice” e “SendCommandToDevice2”. Essi gestiscono la trasmissione del messaggio ASCII via seriale e si occupano di dargli senso secondo il protocollo di funzionamento.

Differiscono per il numero di parametri, a loro volta dipendenti dal tipo di operazione che si vuole imporre alla telecamera. Per quel che riguarda l’utilizzo

fatto in questa tesi, le tipologie sono così suddivise: metodo a due parametri per i movimenti "semplici" (vai a destra, vai a sinistra, stop, eccetera), metodo a quattro parametri per i movimenti secondo coordinate assolute (vai al punto [x,y]).

Infine il metodo "EvaluateChecksum" serve a calcolare una somma, dipendente dal tipo e dal numero dei caratteri componenti la stringa che si vuole inviare, che associata a quest'ultima dà la necessaria validità all'istruzione che altrimenti verrebbe scartata.

3.2 Interfaccia C++ di comando

Viene sotto riportato il codice col quale è possibile implementare un'interfaccia di comando "manuale" del brandeggio.

```
#include <iostream>
#include "ulissepositioning.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <termios.h>
#include <stdint.h> // per poter utilizzare interi a 16 bit
#include <string.h>
#include <boost/lexical_cast.hpp>

#define BUF 10

using namespace std;

string convert(int number);
string p1;
string p2;
string p3;
string p4;

int main() {
UlissePositioning ulisse;
bool control = true;
while (control) {
cout << "inserire un comando per l'ulisse" << endl;
cout << "1=destra|2=sinistra|3=su|4=giu|5=stop|
          6=mandastringa|7=checksum|8=posizione|9=uscita" << endl;
int scelta;
cin >> scelta;
cin.ignore(2,'\n');
```

```
cout << "la tua scelta è stata: " << scelta << endl;

switch (scelta) {
case 1:
ulisse.Right(1);
break;
case 2:
ulisse.Left(1);
break;
case 3:
ulisse.Down(1);
break;
case 4:
ulisse.Up(1);
break;
case 5:
ulisse.Stop();
break;
case 6:
cout << "inserire comando" << endl;
char stringa[6];
cin.getline(stringa,6);
cout << "inserire parametro 1" << endl;
char parametro1[3];
cin.getline(parametro1,3);
cout << "inserire parametro 2" << endl;
char parametro2[3];
cin.getline(parametro2,3);
ulisse.SendCommandToDevice(stringa,parametro1,parametro2);
break;
case 7:
cout << "calcola il checksum" << endl;
char comando[30];
cout << "inserire comando" << endl;
cin.getline(comando,30);
char checksum;
checksum = ulisse.EvaluateChecksum(comando);
cout << checksum << endl;
break;
case 8:
cout << "inserire parametro 1 (pan)" << endl;
int pan;
cin >> pan;
// ulisse accetta solo valori compresi tra -180 e 180 per il pan
pan = pan - round((double)pan/360)*360;
cout << pan << endl;
cout << (int)pan/360 << endl;
if (pan>0)
pan = pan*100;
```

```
else
pan = 65536 + pan*100;
p1 = convert(pan);
const char *para1;
para1 = p1.c_str();
cout << "inserire parametro 2 (tilt)" << endl;
int tilt;
cin >> tilt;
// ulisse accetta solo valori compresi tra -90 e 90 per il tilt
tilt = tilt - round((double)tilt/180)*180;
tilt = -tilt;
if (tilt>0)
tilt = tilt*100;
else
tilt = 65536 + tilt*100;
p2 = convert(tilt);
const char* para2;
para2 = p2.c_str();
cout << "inserire parametro 3 (velocita)" << endl;
int vel;
cin >> vel;
p3 = convert(vel*100);
const char *para3;
para3 = p3.c_str();
cout << "inserire parametro 4 (zoom)" << endl;
int zoom;
cin >> zoom;
p4 = convert(zoom*1000);
const char *para4;
para4 = p4.c_str();
ulisse.GoToSpecificPosition((char*)para1,(char*)para2,(char*)para3,(char*)para4);
break;
case 9:
control = false;
}
}
//cout << ulisse.EvaluateChecksum("TM1PC1Men+") << endl;
return 0;
}

string convert(int number) {
std::ostringstream sin;
sin << number;
std::string val = sin.str();
return val;
}
```

Si pone l'attenzione sulle cose di maggior interesse.

Il "case 6" è stato implementato per poter permettere all'utente di utilizzare comandi non gestiti direttamente dal codice scritto. Qui infatti è richiesto l'inserimento manuale di tutta la stringa da trasmettere all'Ulisse.

Il "case 7" calcola semplicemente il checksum, di cui si è già parlato, di una determinata stringa. È tornato molto utile nel lavoro di tesi, per i test attraverso interfaccia seriale CuteCom, perché come si è detto occorre inserire questo valore assieme al comando perché abbia validità.

Il "case 8" che risulta essere più corposo degli altri, serve per portare il brandeggio in una posizione assoluta appartenente al range previsto, cioè tra -180° e 180° sull'asse pan e tra -90° e +90° sull'asse tilt.

3.3 Schema dell'ambiente

Lo schema dell'ambiente è stato studiato in relazione alla tipologia di modellizzazione dello stesso, derivante dal software di tracking.

La conoscenza che abbiamo di forma e dimensione della stanza reale e della sua immagine prodotta ed elaborata artificialmente dal programma, ci permette di mappare lo spazio 3D in un piano 2D, sfruttando l'ipotesi, ai fini del lavoro non restrittiva, che la persona da trackare si troverà, in ogni istante, necessariamente solidale al pavimento.

Si fissano ora alcuni punti fissi sull'immagine artificiale:

1. posizione del brandeggio;
2. proiezione del brandeggio sul pavimento;
3. angoli del pavimento della stanza;
4. stipiti e centro della porta di ingresso.

Di questi si sono salvate le posizioni in unità di misura "pixel".

Si sono poi fissati i seguenti punti fissi reali:

1. posizione del brandeggio;
2. proiezione del brandeggio sul pavimento;
3. proiezione del centro della porta di ingresso sul pavimento.

Di questi si sono salvate le posizioni in unità di misura "metri".

A questo punto si è cercata una relazione di proporzionalità tra i due sistemi di riferimento e la si è verificata.

Grazie a questa conversione si trasforma la parte di mondo di interesse generata dal tracking in quella della telecamera, riuscendo così a comandarla con sufficiente precisione.

Infine occorre precisare che i punti generati dalla trasformazione tengono conto dello zero dell'Ulisse, fissato secondo le proprie coordinate assolute in -21 su asse pan e -21 su asse tilt.

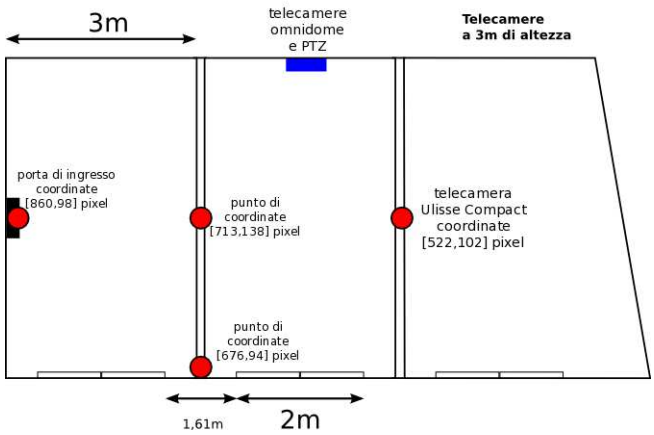


Figura 3.1: Schema della pianta del laboratorio con riportate le coordinate (in pixel) dei punti di maggior interesse tra quelli usati e i riferimenti reali in metri.

Capitolo 4

NMM (Network-Integrated Multimedia Middleware)

Introduzione

Il Network-Integrated Multimedia Middleware (NMM) [6] è il risultato di sei anni di ricerca presso il Laboratorio di Computer Graphics at Saarland University di Saarbrücken, in Germania.

Esso implementa applicazioni di rete mobile e multimediali permettendone la comunicazione reciproca. Sono supportate varie tecnologie e sistemi operativi, come Windows e GNU/Linux, cosa che permette lo sviluppo cross-platform. Usando questa tecnologia unica, tutti i dispositivi disponibili, distribuiti in una rete, possono essere collegati e controllati in maniera trasparente con la creazione di nuove periferiche virtuali. Ad esempio, un telefono cellulare può diventare un ricevitore radio o una registrazione video può essere visualizzata su tre televisori contemporaneamente.

Il software è rilasciato sotto licenze che permettono di utilizzarlo all'interno di prodotti commerciali oppure Open Source e in progetti di ricerca. È prodotto da Motama, azienda specializzata nella progettazione e sviluppo di sistemi di rete distribuiti e multimediali che spaziano dai sistemi embedded e mobili, ai PC, ai cluster computing su larga scala.

Essa è uno spin-off dell'Università del Saarland. Fondata nel 2005 è diretta dal dottor Marco Lohse, Michael Repplinger e il prof. Philipp Slusallek. Offre lo sviluppo dei prodotti, le licenze della tecnologia NMM e supporto nello sviluppo di moduli personalizzati per le applicazioni basate su NMM.

Attualmente questo software è utilizzato nella creazione di prodotti nei settori di:

- home entertainment;
- creazione, elaborazione e distribuzione di contenuti multimediali;

- IPTV;
- rendering server based;
- building technologies;
- ricerca applicata e sviluppo;
- e-learning.

4.1 Tecnologia alla base di NMM

Oltre ai PC, un numero crescente di dispositivi multimediali, quali set-top box, PDA e telefoni cellulari, vengono forniti con funzionalità di rete. Tuttavia le infrastrutture multimediali, ad oggi, adottano un approccio centralizzato, in cui tutte le elaborazioni multimediali avvengono all'interno di un unico sistema e la rete è usata per lo streaming di contenuti predefiniti da un server ai client.

Concettualmente, questi approcci sono costituiti da due applicazioni isolate, un server e un client. La realizzazione di scenari complessi è dunque complicata e soggetta a errori, soprattutto perché il client ha in genere un controllo limititato, se non assente, del server.

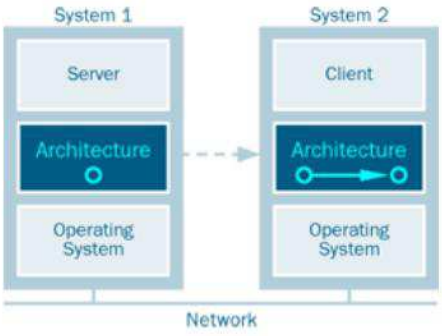


Figura 4.1: Streaming tradizionale client-server composto di due applicazioni isolate che non forniscono un controllo versatile o estendibile.

Il Network-Integrated Multimedia Middleware (NMM) supera queste limitazioni, permettendo l'accesso a tutte le risorse all'interno della rete: dispositivi multimediali distribuiti e componenti software possono essere controllati in modo trasparente e integrato in un'applicazione.

A differenza di tutte le altre architetture multimediali disponibili, NMM è un middleware, cioè uno strato di software in esecuzione tra i sistemi e le applicazioni.

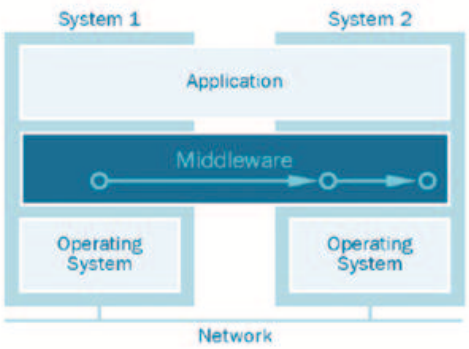


Figura 4.2: NMM come middleware multimediale fornisce uno strato software che facilita lo sviluppo di applicazioni fornendo un accesso trasparente a tutte le risorse all'interno della rete come i dispositivi o i componenti software.

Per fare un esempio, questo consente lo sviluppo rapido e facile di un'applicazione che riceve un segnale TV da un dispositivo remoto, compresa l'implementazione di un controllo trasparente del ricevitore TV sorgente. Anche un PDA con potere limitato di calcolo può servirsi di questa applicazione: le conversioni necessarie per adeguare il contenuto audio e video alle risorse fornite dal PDA possono essere distribuite all'interno della rete. La distribuzione è trasparente per gli sviluppatori e nessun carico è aggiunto alle applicazioni in locale.

Infine NMM fornisce un framework multimediale standard per tutti i tipi di applicazioni desktop.

4.2 Architettura generale di NMM

L'architettura del micro-core del middleware è opensource e strutturata perché sia possibile l'integrazione di arbitrarie risorse, che potrebbero essere presenti in una data rete (flussi audio, flussi video, eccetera). È dotato di un sistema messaggistica di sistema, sia per poter richiedere questi dati, sia per tutte le altre comunicazioni "da" e "verso" i componenti che risultano connessi tra loro tramite un grafo, nel quale però siano gestiti, sia il flusso dei dati, sia il tipo di dato stesso.

NMM supporta tutti i formati di tramissione standard, quali TCP, UDP e RTP. Infine è compatibile con formati standard come XML.

Grazie ai molti plugin, NMM fornisce un alto livello di astrazione e di trasparenza attraverso un'organizzazione a grafi distribuiti usati per specificare i processi multimediali di streaming distribuito. Possiede poi un'architettura ad elevata sincronizzazione che offre alta qualità in termini appunto di sincronia e per la cattura e per il rendering dei flussi multimediali presenti in rete.

Le API di NMM e le sue librerie offrono i seguenti vantaggi:

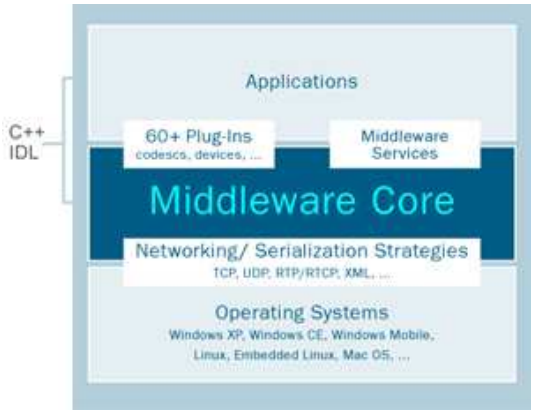


Figura 4.3: NMM fornisce un'architettura micro-core opensource che permette l'intergrazione delle risorse presenti in rete.

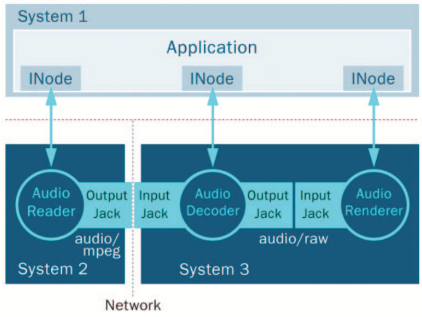


Figura 4.4: Grafi di flusso distribuiti, estesi su più sistemi consentono la facile creazione di applicazioni multimediali distribuite, ad esempio, per la riproduzione di MP3.

- API intuitive object-oriented C++ sommate ad interfacce di controllo object-oriented definite nell'Interface Definition Language (IDL) per la specifica sia out-of-band che per l'interazione instream;

- un nucleo di librerie cross-platform eseguite nativamente su Windows XP, Windows Vista, Windows 7, Windows CE, Windows Mobile, GNU/Linux, Embedded GNU/Linux, Mac OS;

- un grande numero di plugin per dispositivi I/O, codec, elaborazione dati, visualizzazione su schermo e molto altro.

4.3 Servizi middleware aggiuntivi

NMM ha un sistema peer-to-peer di distribuzione del Registro dei Servizi (Registry Service 4.8) che gestisce i dispositivi, i componenti software, rende disponibile la potenza di elaborazione e la larghezza di banda.

Ha un servizio di condivisione (Session Sharing) che permette di collegare le applicazioni in esecuzione, al fine di condividere un singolo dispositivo multimediale o riutilizzare parti comuni di un grafo di flusso (flow graph 4.6), ad esempio, per la transcodifica. Sincronizzazione e controllo vengono così stabiliti tra i vari host.

Per il passaggio della riproduzione di contenuti multimediali, da un sistema mobile ad un sistema fisso, il middleware è dotato di un servizio di consegna (Handover Service) per il passaggio continuo e sincronizzato dei grafi di flusso.

È prevista la configurazione automatica delle applicazioni che permette di specificare un preciso task, come la riproduzione di media, semplicemente specificando, magari a distanza, le fonti dei media e dei dispositivi di rendering. Il corrispondente grafo viene automaticamente istanziato e distribuito.

4.4 L'organizzazione a nodi, jack e flow graph

L'approccio progettuale generale dell'architettura di NMM è simile ad altre architetture multimediali. All'interno di NMM, tutte le periferiche hardware (ad esempio una scheda TV) e componenti software (ad esempio, decoder) sono rappresentati da entità chiamate "nodi". Un nodo ha delle proprietà che includono porte di input e output, chiamate "jack", che ne racchiudono anche il relativo formato multimediale supportato. Un formato definisce con precisione il flusso multimediale fornito, specificandone il tipo (ad esempio audio/raw per i flussi audio non compressi), oltre ai parametri aggiuntivi, quali la frequenza di campionamento. Poiché un nodo può fornire più ingressi e uscite, i suoi jack sono etichettati con tag. A seconda del tipo di nodo, vengono prodotti determinati dati, eseguite determinate operazioni su di essi o infine "consumati".

- Source: produce i dati ed ha un jack di uscita.
- Sink: consuma i dati che riceve dal suo jack di input.
- Filter : ha un jack di input e uno di output. Modifica solo i dati del flusso e non cambia il formato o i parametri specifici del formato.
- Converter: ha un jack di input e uno di output. Può modificare sia i dati del flusso che il formato o i parametri (ad esempio da video normale a compresso oppure modificarne la risoluzione).
- Multipexer: ha più jack di input, ma un unico jack di output.

- Demultiplexer: ha un unico jack di input, ma diversi jack di output.

Questi nodi possono essere collegati per creare un diagramma di flusso, purché i due jack tra loro connessi gestiscano lo stesso formato dati. Inoltre i parametri legati al formato dei due jack devono essere vicendevolmente disponibili. Nella figura si vede lo scherma di decodifica e riproduzione di un file MP3.

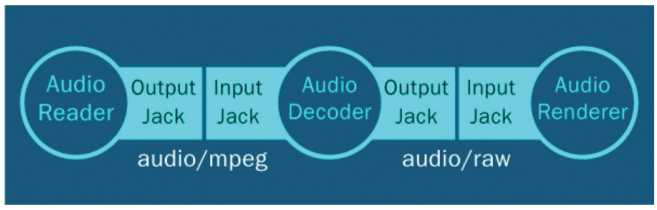


Figura 4.5: Grafo di flusso per la riproduzione di file MP3.

4.5 Sistema di messaging

L'architettura NMM utilizza un sistema di messaggistica standard per tutte le comunicazioni. Ci sono due tipi di messaggi:

1. a buffer: memorizzano i dati multimediali;

2. a eventi: sono identificati da un nome e possono includere qualsiasi tipo di parametro. Sono usati per le informazioni di controllo.

Ci sono due diversi tipi di paradigmi di interazione utilizzati all'interno di NMM. In primo luogo, i messaggi sono trasmessi su jack connessi tra loro. Questo tipo di interazione è detta instream ed è più spesso eseguita in direzione downstream, cioè dai Source ai Sink; NMM comunque consente anche l'invio di messaggi in upstream.

Si noti che sia i buffer che gli eventi possono essere inviati instream. Per questa comunicazione, sono utilizzati i cosiddetti "composite events" che contengono una serie di eventi da gestire all'interno di una singola fase di esecuzione. Gli eventi instream sono molto importanti per i grafici di flusso multimediale. Per esempio, la fine di un flusso (ad esempio la fine di un file) possono essere segnalate con l'inserimento di uno specifico evento, alla fine di una serie di buffer. Oggetti "listener" esterni possono essere registrati per essere avvisati quando si verificano determinati eventi in un nodo (ad esempio per l'aggiornamento della GUI alla fine di un file o per la selezione di un nuovo file).

Gli eventi sono utilizzati anche per il secondo tipo di interazione chiamato out-of-band, cioè l'interazione tra l'applicazione e gli oggetti NMM, come ad

esempio i nodi o i jack. Gli eventi sono utilizzati per controllare gli oggetti o per l'invio di notifiche dagli oggetti ai listener registrati.

4.6 Grafi di flusso distribuiti (Distributed Flow Graphs)

La particolarità di NMM è il fatto che i suoi grafi di flusso possono essere distribuiti attraverso la rete: dispositivi multimediali locali o remoti oppure lo stesso codice incapsulato nei nodi, possono essere integrati all'interno di un comune flow graph distribuito per l'elaborazione multimediale. Tutto questo è trasparente agli sviluppatori e non appesantisce le parti del grafo che operano a livello locale.

Nella Figura 4.5 viene mostrato un esempio di un diagramma di flusso distribuito per la riproduzione di file codificati (compressi), ad esempio, file MP3. Un nodo Source per la lettura dei dati dal file system locale (AudioReader) è collegato ad un nodo per la decodifica di flussi audio (AudioDecoder). Questo decoder è collegato ad un nodo Sink per il rendering audio non compresso utilizzando una scheda audio (AudioRenderer). Una volta che il grafo è stato avviato, il nodo Source legge una certa quantità di dati da un file, incapsula in buffer, ad esempio, di dimensione 1024 byte e trasmette questi buffer al suo successore. Dopo la decodifica in audio non compresso, il nodo Converter manda i buffer di dati al nodo Sink.

L'applicazione può gestire tutte le parti di questo flow graph utilizzando le interfacce (ad esempio INode) per controllare gli aspetti generici di tutti i nodi istanziati. Si noti che tre diversi host sono presenti nell'esempio (si veda Figura Figura 4.4). L'applicazione gira sull'host1, il nodo Source sull'host2, il decoder e il nodo Sink sull'host3. NMM crea automaticamente le connessioni di rete tra l'applicazione e i tre nodi distribuiti (interazione out-of-band), ma anche tra il Source e il nodo di decoder (interazione instream). Pertanto, i dati audio compressi MPEG vengono trasmessi attraverso la rete.

Si noti che un flow graph così semplice fornisce già molti benefici. In primo luogo, consente a un'applicazione di accedere ai file memorizzati su sistemi distribuiti, senza la necessità di un file system distribuito, come NFS. In secondo luogo, i dati di flusso tra i nodi collegati viene gestito automaticamente dal NMM. In terzo luogo, l'applicazione agisce come "telecomando" per tutti i nodi distribuiti. Per fare un esempio, questo consente in modo trasparente di cambiare il volume di uscita della scheda audio a distanza attraverso una semplice chiamata di metodo su una specifica interfaccia, ad esempio IAudioDevice.

4.7 Sincronizzazione distribuita

Poiché i grafi di flusso in NMM possono essere distribuiti, è possibile il rendering audio e video su sistemi diversi. Per esempio, il flusso video di un file MPEG2 può essere presentato su un grande schermo collegato ad un PC, mentre il corrispondente audio viene riprodotto su un dispositivo mobile. Per realizzare la riproduzione sincronizzata di nodi distribuiti su tutta la rete, NMM fornisce un'architettura generica di sincronizzazione distribuita. Inoltre, le presentazioni dei media possono essere eseguite anche su sistemi diversi contemporaneamente. Una domanda comune è la riproduzione del flusso audio utilizzando sistemi diversi situati in diverse stanze di una casa, ma la teoria è la stessa su rete geografica.

La base per eseguire la sincronizzazione distribuita è una comune fonte di temporizzazione: un orologio statico all'interno di ogni spazio di indirizzi. Questo orologio rappresenta l'orologio di sistema che globalmente è sincronizzato con il Network Time Protocol (NTP) e può quindi essere assunto come riferimento per la rete. Con l'impostazione corrente, il tempo di offset tra sistemi diversi spazia da 1 a 5 ms, il che è sufficiente.

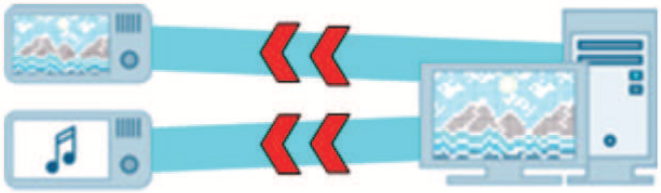


Figura 4.6: Esempio di sincornizzazione multi-room e multi-device.

4.8 Registry Service

Il servizio "Registro di Sistema" (Server Registry) in NMM permette la rilevazione, la prenotazione e l'istanziamento di nodi disponibili su host locali e remoti. Su ogni host c'è un registry server unico che amministra tutti i nodi di quel particolare sistema. Per ogni nodo, il Registro ne memorizza una descrizione che include il tipo (Sink, ad esempio), il nome (PlaybackNode, ad esempio), le interfacce fornite (IAudioDevice, ad esempio, per aumentare o diminuire la produzione volume) e gli input e output supportati (ad esempio, un formato di input audio/raw con i suoi parametri aggiuntivi, come la frequenza di campionamento).

L'applicazione utilizza un "Registro Cliente' (Client Registry) per inviare richieste ai registry server degli host siano essi locali o remoti e vengono contattati

per essere collegati ad una porta nota. Dopo aver elaborato la query con successo il Registro di Sistema interpellato riserva i nodi richiesti che vengono quindi creati. Per i nodi istanziati sullo stesso host che esegue anche le richieste, il Registro di Sistema assegnerà questi oggetti all'interno dello spazio degli indirizzi della richiesta stessa per evitare il che il loro sormontarsi generi overhead.

Per configurare e creare complessi diagrammi di flusso distribuito, un'applicazione può richiedere ogni nodo separatamente o utilizzare una "graph description" come query. Tale descrizione comprende una serie di "nodi descrizione" collegati tra loro.

Perché un'applicazione possa creare un diagramma di flusso distribuito, il Server Registry deve essere in esecuzione su ciascun host partecipante. Per le applicazioni che operano esclusivamente a livello locale questo non è richiesto. Un server registry è sempre in esecuzione all'interno dell'applicazione stessa e non accessibile da host remoti.

Prima che possa essere utilizzato, deve stabilire quali dispositivi e componenti software sono disponibili su un determinato host. Pertanto, il deve essere inizializzato almeno una volta usando il comando seguente.

```
user@linux:~/nmm/bin> ./serverregistry -s

Create config file with plugin information ...
Loading plugins...
AC3DecodeNode                                    available
AVDemuxNode                                      available
AVIReadNode                                      available
... and many further nodes

Config file successfully written.
```

4.9 Utilizzo di NMM

4.9.1 Installazione e test NMM

Non si riporteranno in questo elaborato tutti i passi necessari all'installazione e configurazione del software NMM, ma si ricorda solamente che in ambiente GNU/Linux esistono due strade principali. La prima è l'installazione da sorgenti e la seconda è attraverso i pacchetti precompilati, se esistono, per la propria distribuzione.

Si raccomanda, in caso di installazione da sorgenti, di tener presenti tutte le dipendenze necessarie, in particolare, essendo NMM dotato di molti plugin, si dovrà prestare attenzione alle richieste di ciascuno di essi, se li si volesse usare.

Una volta installato, il middleware è a disposizione del sistema. Lo si può quindi testare col comando:

```
$ serverregistry -s
```

che restituisce la lista dei nodi disponibili nel sistema. Nel caso avessimo fatto particolari scelte circa la presenza di determinati plugins, nella lista sottostante dovremmo essere in grado di identificarne il nodo corrispondente. Ovviamente in caso di fallimento del comando o in assenza dei nodi previsti, sarà il caso di ripetere l'installazione non essendo andata a buon fine.

```
serverregistry and Network-Integrated
Multimedia Middleware (NMM) Version 2.2.0

Copyright (C) 2005-2010
Motama GmbH, Saarbruecken, Germany
http://www.motama.com

See licence for terms and conditions of usage

No plugin information available! If you start NMM for
the first time this information is created automatically.
Note: If you ever change your hardware configuration or
update the NMM version you must delete the file
/home/alberto/.nmm/plugins.2.2.0.xps._usr_local
or run 'serverregistry -s' again.

Create config file with plugin information ...
Loading plugins...
AC3DecodeNode                                   available
AC3ParseNode                                    available
ALSAPlaybackNode                                available
ALSARecordNode                                  available
AVDemuxNode                                     available
AVMuxNode                                       available
AnalyseDataIdNode                               available
AudioMuxNode                                    available
BrightnessNode                                  available
BufferDropNode                                  available
BufferShapingNode                               available
BufferTimestampControlNode                      available
BufferTimestampNode                             available
CopyNode                                        available
DVBS2ReadNode                                   not available
DVBSimulatorNode                                available
DevNullNode                                     available
DummyAudioSinkNode                              available
DummyVideoSinkNode                              available
FFMPEGDeinterlaceNode                           available
FFMpegAVIReadNode                               available
FFMpegAVIWriteNode                              available
```

```
FFMpegAudioDecodeNode                   available
FFMpegAudioEncodeNode                   not available
FFMpegDecodeNode                        available
FFMpegEncodeNode                        available
FLACReadNode                            available
FramerateConverterNode                  available
GenericReadNode                         available
GenericWriteNode                        available
H264DecodeNode                          available
IVTVReadNode                            not available
IcecastNode                             available
IdNode                                  available
JPEGDecodeNode                          available
JPEGEncodeNode                          available
LogoNode                                available
M4AReadNode                             available
MP3ReadNode                             available
MPEGAudioDecodeNode                     available
MPEGAudioEncodeNode                     available
MPEGDemuxNode                           available
MPEGReadNode                            available
MPEGTSDemuxNode                         available
MPEGTSReadNode                          available
MPEGTimeShiftingNode                    available
MPEGTimeShiftingNode2                   available
MPEGVSHDetectionNode                    available
MPEGVideoDecodeNode                     available
MagickManipulationNode                  available
MagickReadNode                          available
MagickWriteNode                         available
MessageSeekNode                         not available
NetSourceNode                           not available
OGMDemuxNode                            available
OSDManagerNode                          available
OggVorbisDecodeNode                     available
OverlayNode                             available
PCMDecodeNode                           available
PCMEncodeNode                           available
PNGReadNode                             available
PNGWriteNode                            available
PlaybackNode                            available
RGBtoRGBConverterNode                   available
RGBtoYV12ConverterNode                  available
RawNode                                 available
RecordNode                              available
ScopeNode                               available
TSDemuxNode                             available
TVCardReadNode                          available
TVCardReadNode2                         available
```

```
TimeDisplayNode                                 available
TimedBufferDropNode                             available
URLNode                                         available
VISCACameraReadNode                             not available
VideoCropNode                                   available
VideoGrabNode                                   available
VideoMuxNode                                    available
VideoScalerNode                                 available
VoiceDetectorNode                               not available
WavReadNode                                     available
WavWriteNode                                    available
WhiteNoiseNode                                  available
XDisplayNode                                    available
YUVDeInterlaceNode                              available
YUVtoRGBConverterNode                           available
YUVtoYUVConverterNode                           available
Finished loading plugins ...

Config file successfully written.
```

Proviamo ora ad istanziare ed usare un grafo.

NMM installa di default alcuni esempi, molto utili soprattutto quando successivamente andremo a creare flow graph personalizzati con nodi creati autonomamente. Spostarsi quindi nella directory audio degli esempi (nota, per il percorso si fa riferimento alla gerarchia di directory del pacchetto TAR scompattato; ognuno nel suo sistema, in caso di installazione globale, verificherà dove i file di cui parleremo sono collocati):

```
$ cd nmm-2.2.0/apps/clic/gd/linux/playback/audio/
```

All'interno troveremo file di estensione ".gd": sono descrizioni di grafi già pronte. Essi rappresentano con semplice sintassi i nodi che andranno a formare il grafo. Si consiglia di aprirne qualcuno per farsi un'idea di quel che si sta dicendo.

Si testi ora il funzionamento del middleware, provando ad eseguire la riproduzione di un file audio con l'ausilio del software "clic" (installato di default) ed il grafo "wavplay.gd':

```
$ clic wavplay.gd -i <file audio .wav>
```

se si riesce a sentire il file audio, allora si può procedere anche con un test video:

```
$ cd nmm-2.2.0/apps/clic/gd/linux/playback/video/
```

successivamente:

```
$ clic noise.gd
```

nel caso si apra una finestra, mostrandoci il video di un "disturbo", allora si può con tranquillità concludere che l'installazione è andata a buon fine.

4.9.2 Metodologie di utilizzo

I metodi di utilizzo di NMM sono due:

1. attraverso l'utility "clic", usata anche negli esempi precedenti;
2. attraverso la scrittura di codice C/C++ che si interfacci con le librerie di NMM, utilizzandone il namespace e quindi le librerie.

4.9.3 Clic (Command Line Interaction and Configuration)

L'applicazione Clic viene utilizzata per costruire in modo automatico grafi di NMM a partire da una loro descrizione testuale, la "graph description". Se ne vede poi brevemente la sintassi.

Inoltre, Clic permette di utilizzare il generatore di grafo di NMM che crea automaticamente un flow graph distribuito da un URL dato: la sorgente del flusso multimediale e l'uscita che possono essere audio o video, vengono rese disponibili in rete.

Si spiegherà ora la descrizione del "WavReadNode" usato precedentemente:

```
% This graph description plays a WAV file using the
% ALSAPlaybackNode
% It can only be run on Linux.
% Command:
% clic wavplay.gd -i <local WAV file>
WavReadNode
! ALSAPlaybackNode
```

Esso consiste di due parti principali: la prima parte è un commento (opzionale) e lo si fa sempre precedere da "%", mentre la seconda specifica il flaw graph e come sono connessi i suoi nodi. Il carattere "!" serve appunto a connettere i nodi tra loro; man mano che se ne aggiungono è sufficiente scriverli in cascata.

Se si copia e incolla il testo precedente in un file e lo si rinomina "wavplay.gd" è possibile utilizzarlo con Clic, come fatto sopra nella fase di test. Per richiamare il comando:

```
$ clic wavplay.gd -i file.wav
```

Per avere informazioni più dettagliate sull'utilizzo di Clic, si rimanda alla documentazione in linea [6]. Una panoramica delle opzioni la si può anche avere col comando:

```
$ clic -h
```

4.9.4 Creazione di un flow graph

Ecco come collegare i nodi. Un nodo è identificato dal suo nome, che è normalmente quello della classe C++. Il punto esclamativo (!) serve per indicare un collegamento tra due nodi come si è visto nella descrizione del grafico per la riproduzione di file WAV. In questo esempio l'istruzione

```
WavReadNode ! ALSAPlaybackNode
```

richiede i nodi WavReadNode e ALSAPlaybackNode. Se le richieste avranno successo, il WavReadNode sarà collegato all'ALSAPlaybackNode. Infine Clic imposta i parametri specificati da riga di comando, per esempio un file di input e costruisce l'intero flow graph.

Per specificare grafi di flusso più complessi che includono demultiplexer, abbiamo bisogno di una sintassi più lunga che descriva i diversi jack di input e output di cui un (de)multiplexer è dotato. Una stringa univoca che viene chiamata "jack tag" è associata ad ogni jack per identificarlo. Per esempio sono utilizzate per distinguere tra audio e video. Se un nodo fornisce solo un ingresso o un'uscita la stringa default viene usata come jack tag ed è usato nelle descrizioni dei grafi se nessun altro jack tag è specificato. Per poter utilizzare un jack di input specifico che deve essere utilizzato per una connessione, l'etichetta relativa deve essere scritta prima del nome del nodo se questo è di input oppure dopo se è di output. Il seguente esempio mostra la descrizione del player WAV con jack tag "default":

```
            WavReadNode         "default"
! "default" ALSAPlaybackNode
```

In questo esempio non è necessario specificare i tag, perché Clic utilizza il default jack per collegare i nodi se nessun altro è specificato.

Ecco invece una sintassi più complessa per descrivere i rami di un demultiplexer che legge un file MPEG. Per le diramazioni, i nodi corrispondenti sono incapsulati tra parentesi graffe { } come si vede nell'esempio seguente.

```
% This graph description realizes a simple mpeg video player with ac3 audio.
% Use the -i option of clic to specify the mpeg video file.

  GenericReadNode
! MPEGDemuxNode
{
  { ["mpeg_video0"] ! MPEGVideoDecodeNode
                    ! XDisplayNode }
  { ["ac3_audio0"]  ! AC3DecodeNode
                    ! ALSAPlaybackNode }
}
```

L'MPEGDemuxNode demultiplexa i messaggi in entrata e spedisce il buffer audio al jack di uscita audio "ac3_audio0 e il buffer video al jack di uscita video "mpeg_video0. Dopo MPEGDemuxNode, le parentesi racchiudono tutte le branche collegate a questo nodo. Ogni ramo è a sua volta racchiuso tra parentesi graffe e inizia con il jack tag di uscita del MPEGDemuxNode, incapsulato in parentesi quadre. Questo jack di uscita è collegato al nodo principale del ramo. Quindi in questo esempio abbiamo due rami. Il primo consiste nell'MPEGVideoDecodeNode, collegato all'XDisplayNode. L'MPEGVideoDecodeNode è il primo nodo di questo ramo ed è collegato alla presa di uscita del MPEGDemuxNode con jack tag "mpeg_video0. Il secondo ramo l'AC3DecodeNode è collegato all'ALSAPlaybackNode. Il primo nodo di questo ramo è l'AC3DecodeNode che è collegato al jack di uscita dell'MPEGDemuxNode con jack tag "mpeg_audio0.

L'esempio seguente estende il file sopra con l'aggiunta di un altro XDisplayNode. Si vede che sono supportati i rami nidificati.

```
% This graph description plays an MPEG file with AC3 audio
% using the ALSAPlaybackNode and two XDisplayNodes.
% It can only be run on Linux.

% Command:
%   clic mpegplay_ac3_two_displays.gd -i file

  GenericReadNode
! MPEGDemuxNode
{
  { ["mpeg_video0"] ! MPEGVideoDecodeNode
                    {
                      { ["default"] ! XDisplayNode }
                      { ["default"] ! XDisplayNode }
                    }
  }
  { ["ac3_audio0"]  ! AC3DecodeNode
                    ! ALSAPlaybackNode
```

```
  }
}
```

Aggiungere un nodo multiplexer non differisce molto da quanto visto per il demultiplexer:

```
% This graph description converts an MPEG video file with AC3
% audio into an avi file.
% Unfortunately the flow graph does not work because the FFMpegEncodeNode
% requires a bitrate.

  GenericReadNode
! MPEGDemuxNode
{
  { ["mpeg_video0"] ! MPEGVideoDecodeNode
                    ! FFMpegEncodeNode
                    !
    ["video"]
  }
  {
    ["ac3_audio0"] !
    ["audio"]
  }
} AVMuxNode
! AVIWriteNode
```

Questo esempio può essere utilizzato per convertire un file MPEG in un file AVI con DivX-video e audio AC3. La prima parte consiste in un MPEGVideoDecodeNode che è collegato al FFMpegEncodeNode. L'ingresso del primo nodo di questo ramo, l'MPEGVideoDecodeNode, è collegata al jack di uscita dell'MPEGDemuxNode con jack tag mpeg_video0. L'uscita dell'ultimo nodo invece, l'FFMpegVideoEncodeNode, è collegato al jack di ingresso del jack AVMuxNode con tag video. Il secondo ramo non ha nodi. In questo ramo l'uscita del jack MPEGDemuxNode con jack tag ac3_audio0 è collegata al jack di ingresso di AVmuxNode con jack tag Audio. Pertanto, i buffer audio sono direttamente passati dal MPEGDemuxNode al AVMuxNode senza alcuna conversione.

La sintassi di questo esempio è corretta, ma il flow graph non può essere eseguito perché l'FFMpegEncodeNode accetta solo dati con bitrate di codifica specificato nel formato di connessione. Quindi, abbiamo bisogno di poter specificare un formato di connessione da utilizzare tra due nodi. Lo statement "format" preceduto dal carattere "@" ha questo scopo. Esso si aspetta un elenco di formati separato da virgole, definiti nel "Format.hpp". L'unica eccezione è il formato che viene scritto come una singola stringa.

Nota: il formato di connessione deve essere specificato sulla riga del simbolo di connessione (!). Nell'esempio riportato di seguito viene illustrato il concetto.

```
% This graph description converts an MPEG video file with AC3 audio
% into an AVI file. Furthermore the desired bitrate of the video is
% specified using the connection format.
% Use the -i option of clic to specify the mpeg video file and -o option
% to specify the name of the output file.

  GenericReadNode
! MPEGDemuxNode
{
  { ["mpeg_video0"] ! MPEGVideoDecodeNode
                    ! FFMpegEncodeNode
                      @ Format ("video/mpeg4", bitrate = 1200000)
                    !
    ["video"]
  }
  {
    ["ac3_audio0"] !
    ["audio"]
  }
} AVMuxNode
! AVIWriteNode
```

Per eseguire questo diagramma di flusso, copiare la descrizione in un file, ad esempio, mpeg_to_avi.gd ed eseguire:

```
$ ./clic mpeg_to_avi.gd -i /home/bob/video/movie.mpeg -o /home/bob/video/movie.avi
```

con l'opzione "-i" che specifica il file di input e "-o" il file di output.

Inoltre, è possibile specificare due ulteriori parametri per impostare il protocollo di trasporto utilizzato per trasmettere dati tra due nodi, che è particolarmente utile se sono in esecuzione su host differenti. Entrambi i parametri sono preceduti dal carattere "@". Per configurare uno specifico protocollo di trasporto utilizzato per la comunicazione instream, possono essere utilizzati il "setBufferStrategy" e il "setEventStrategy", come si vede nella descrizione di grafo seguente:

```
% This graph description describes a simple MP3 player which specifies the
% used transport protocols between the nodes.
% Use the -i option of clic to specify the MP3 file
```

```
GenericReadNode     @ setEventStrategy("TCPStrategy")
                    @ setBufferStrategy("RTPStrategy") !
MPEGAudioDecodeNode @ setBufferStrategy("RTPStrategy") !
PlaybackNode
```

Il parametro "setBufferStrategy" definisce il protocollo di trasporto utilizzato per trasmettere il multimedia buffer e il parametro "setEventStrategy" come trasportare gli eventi instream. In questo esempio il protocollo RTP (Realtime Transport Protocol) è utilizzato per trasmettere i buffer MP3 e il protocollo TCP risulta invece affidabile per la trasmissione di eventi instream. Per il trasporto del multimedia buffer tra l'MPEGAudioDecodeNode e il PlaybackNode, il viene utilizzato il protocollo RTP, ma nessun protocollo per gli eventi instream è specificato. Se uno di questi parametri o entrambi non sono settati, la strategia di trasporto sarà quella di default di NMM, che è la "LocalStrategy", in grado di trasportare buffer ed eventi tra i nodi nello stesso spazio di indirizzo in modo efficiente, attraverso l'inoltro di puntatori oppure c'è il trasporto "TCPStrategy", che esegue le stesse operazioni dell'altro, ma tra nodi che girano su host diversi o in diversi spazi di indirizzi.

4.9.5 SDK (Software Development Kit)

Il Software Developer Kit (NMM-SDK) permette di iniziare velocemente con lo sviluppo di software basato su NMM. Esso fornisce una semplice interfaccia NMM-IDL, un plugin che implementa questa interfaccia e un'applicazione che utilizza questi due componenti in un diagramma di flusso.

L'uso di questo software prevede ovviamente la precedente intallazione del pacchetto NMM-2.2.0. Senza lunghe spiegazioni, come sempre in ambiente GNU/Linux è sufficiente scompattare il pacchetto ed eseguire i soliti

```
$ ./configure
$ make
$ make install
```

Se tutto è andato a buon fine, col comando

```
$ serverregistry -s
```

dovremmo vedere i nomi dei nodi che abbiamo aggiunto:

```
MioNuovoNodo        available
```

Capitolo 5

Tracking e Fall Detector

Introduzione

Lo scopo principale di questa tesi è l'integrazione, del brandeggio Ulisse Compact in un software di tracking preesistente.

A tal fine si sono implementati alcuni metodi, scritti in linguaggio di alto livello (C++), che ne permettono il controllo. Da essi sono state sviluppate le interfacce per il middleware NMM che consentono l'utilizzo della telecamera [7] ai fini della fall detection e del tracking continuato del soggetto in movimento.

Tracking mediante sottrazione dello sfondo

Questa è una tecnica che si basa essenzialmente sui seguenti punti [8]:

1. modellazione del background;
2. estrazione del foreground;
3. segmentazione dell'immagine;
4. filtraggio e correzione;
5. tracciamento.

MODELLAZIONE DEL BACKGROUND

A partire da una serie di immagini riprese da una telecamera fissa, è possibile individuare un'immagine artificiale che viene definita "immagine di background". Essa rappresenta la parte immobile della scena, cioè quella parte di mondo ripresa che tende a cambiare lentamente nel tempo. Nel caso contingente si tratta della stanza vuota.

Per ottenere questa immagine si costruisce, a partire da un insieme di immagini osservate in condizioni standard (stanza vuota), un set di dati che verranno

memorizzati per un certo tempo e che conterranno appunto "l'immagine di partenza". Ora i frame successivi a quelli utilizzati per la modellizzazione dello sfondo, saranno considerati "di input".

ESTRAZIONE DEL FOREGROUND

Il "foreground" contiene la porzione dell'immagine ripresa che tende a variare velocemente nel tempo. In questo caso, sono le persone in movimento nella stanza. Per ottenerlo dall'immagine originale che, com'è intuibile, è la somma di background e foreground, si esegue una sottrazione. La cosiddetta "background substraction".

Si esegue quindi un confronto tra il modello dello sfondo (dato dalle prime immagini memorizzate) e i vari frame di input. Un oggetto che dovesse apparire nel campo visivo della telecamera, sarebbe notevolmente diverso dalla situazione base e quindi attraverso quella che abbiamo definito sottrazione otterremo l'oggetto in questione.

SEGMENTAZIONE

Una volta ottenuto il foreground, si tratta di riconoscere ed etichettare i blob contenenti il soggetto o i soggetti da trackare. Questa è una fase molto delicata che comporta problemi non indifferenti di under-segmentation o over-segmentation. Il primo è costiuito da quegli errori causati dal fatto che il programma genera un numero di blob inferiore a quello effettivo. Capita ad esempio se due oggetti sono molto vicini e quindi vengono "fusi" assieme. Il secondo, in maniera opposta, si ha quando il programma rileva un numero di blob superiore a quello effettivo. Questo può capitare (ed è successo in laboratorio) a causa delle ombre, oppure di particolari situazioni di luce che riflessa dagli oggetti inganna il software.

FILTRAGGIO E CORREZIONE

A questo punto è opportuno eseguire operazioni di filtraggio e correzione per ovviare ai molti problemi di matching errato che possono esser dovuti a fattori quali sovrapposizioni di oggetti, scarsa o eccessiva luminosità e via dicendo.

TRACCIAMENTO

Per ottenere l'inseguimento degli oggetti in movimento, è opportuno predisporre strutture dati adeguate. Nel codice utilizzato sono state implementate delle liste di oggetti "trackedcontourn" in cui se ne memorizza il contorno (posizione, dimensioni, eccetera). In particolare vengono usate due liste, una "passata" e una "presente". In questo modo attraverso il confronto reciproco dei dati, secondo un criterio di prossimità (l'idea è che tra due frame consecutivi, lo spostamento sia minimo e quindi se in queste due immagini riconosco due oggetti che hanno un certo livello, alto, di congruenza, stabilisco che si tratta dello stesso oggetto che si è distanziato), è possibile stabilire dove un oggetto si sia spostato e tenerne, appunto, traccia. Inoltre dal contorno ricavo il cosiddetto bounding box, un rettangolo che inquadra il soggetto identificato.

Nel lavoro di tesi, si vuole far seguire alla telecamera una singola persona che cammina ed eventualmente cade nella stanza. Essa verrà rilevata dall'al-

goritmo tramite un'identificazione basata sulle dimensioni. In breve l'ipotesi è che nella stanza, vi sia un solo soggetto in movimento e che di tutte le cose che possono, per qualche motivo, muoversi, esso è certamente quello di dimensioni maggiori (ipotesi verosimile poiché in una stanza difficilmente avremo qualcosa in movimento più grande di un uomo).

Rilevazione di cadute

La rilevazione delle cadute, parte dall'assunto che il tracking riesca ad identificare il bounding box del soggetto in movimento.

Data questa ipotesi, i passi principali della fall detection sono:

1. controllo della geometria del "bounding box candidato";
2. creazione di strumenti di tuning e controllo delle prestazioni.

1. CONTROLLO DEL BOUNDING BOX

In breve, vengono valutate la geometria e l'affidabilità dei dati riguardanti i vari bounding box identificati dal tracking. L'ipotesi di fondo è che se una persona cade, il suo rettangolo di contorno passa dall'essere maggiormente sviluppato in verticale, all'esserlo in orizzontale. Inoltre col passare del tempo esso mantiene le stesse coordinate, cioè è fermo. Sempre su base temporale, si scartano quei bounding box che pur verificando le regole geometriche, durano per pochi frame. Vengono considerati errori perché se una persona cade è realistico che lì rimanga per un tempo abbastanza lungo.

2. TUNING E PRESTAZIONI

Sono strumenti implementati per gestire situazioni di falsi positivi o per cercare di ovviare alle difficoltà del tracking in quanto può capitare, per le promblematiche precedentemente descritte, che fallisca e generi dati inconsistenti o errati.

5.1 Creazione del controllo

La prima necessità per l'integrazione del brandeggio Ulisse Compact nei software sopra descritti, è quella della creazione di metodi che ne permettano il totale controllo.

Per fare questo si è usato il linguaggio di alto livello C/C++ [5], in ambiente di lavoro GNU/Linux [9]. Questo perché il contesto lo richiedeva implicitamente ed anche perché, il kernel Linux ha un alto grado di efficienza. Un linguaggio compilato e non interpretato, quale appunto quello usato, unito ad un kernel in grado di sfruttare molto bene il tempo che gli è dato, danno ampia garanzia di risultato.

Dalla documentazione fornita con la telecamera e dal protocollo da essa usato, sono state selezionate le funzioni di interesse:

- spostamento in verticale;
- spostamento in orizzontale;
- spostamento verso una posizione predeterminata;
- aumento zoom;
- diminuzione zoom;

che hanno richiesto la creazione dei seguenti metodi:

MOVIMENTI BASE: UP, DOWN, LEFT, RIGHT, STOP

```
void Up(int speed) {MoveCommand("MXUp", speed);}
void Down(int speed) {MoveCommand("MXDn", speed);}
void Left(int speed) {MoveCommand("MXLf", speed);}
void Right(int speed) {MoveCommand("MXRg", speed);}
void Stop(void) {SendCommandToDevice("MXSt");}
```

MOVIMENTI OBLIQUI: UP-LEFT, UP-RIGHT, DOWN-LEFT, DOWN-RIGHT

```
void UpLeft(int speedX, int speedY) {MoveCommand("MXUL", speedX, speedY);}
void UpRight(int speedX, int speedY) {MoveCommand("MXUR", speedX, speedY);}
void DownLeft(int speedX, int speedY) {MoveCommand("MXDL", speedX, speedY);}
void DownRight(int speedX, int speedY) {MoveCommand("MXDR", speedX, speedY);}
```

SPOSTAMENTO IN UNA POSIZIONE DI PRESET

```
void GoToPresetPosition(int presetPosition);
```

SPOSTAMENTO VERSO UNA POSIZIONE SPECIFICA

```
void GoToSpecificPosition(char *pan, char *tilt, char *run, char *zoom);
```

Infine sono stati creati dei metodi “di contorno”, necessari a:

- spedire il comando (stringa ASCII) attraverso la porta seriale [10];
- calcolare le strutture di controllo necessarie al corretto utilizzo delle istruzioni del brandeggio, quale il calcolo di un checksum;

- ricevere e interpretare le istruzioni di movimento.

SPEDIZIONE MESSAGGIO CON 2 PARAMETRI

```
void SendCommandToDevice(const char command[],
          const char par1[] = "",
          const char par2[] = "");
```

SPEDIZIONE MESSAGGIO CON 4 PARAMETRI

```
void SendCommandToDevice2(const char command[],
          const char par1[] = "",
          const char par2[] = "",
          const char par3[] = "",
          const char par4[]="");
```

CALCOLO DEL CHECKSUM

```
char EvaluateChecksum(const char Str[]);
```

RICEZIONE E INTERPRETAZIONE COMANDI DI MOVIMENTO

```
void MoveCommand(const char *command, int speed1, int speed2 = 0);
```

Infine si è proceduto alla creazione di un'interfaccia utente per il controllo del brandeggio, slegato da tracking o altro, al semplice scopo di test o di utilizzo "manuale" della videocamera.

Esso consiste in un "main" creato per stampare a video un menù a scelta multipla, dal quale l'utente può selezionare tutti i comandi (metodi) sopra elencati ed ha in più una funzione che gli permette un controllo totale del messaggio da inviare via seriale, per l'uso di comandi non previsti dal codice implementato, nel caso fosse necessario.

Infine si è predisposta la possibilità di interrogazione dell'Ulisse, per ricevere alcune informazioni quale ad esempio la sua posizione assoluta.

5.2 Creazione dei nodi NMM

Per l'utilizzo del brandeggio Ulisse nel middleware, occorre integrarne il software precedentemente prodotto, nei nodi NMM del tracking [3] [11] e della fall detection [12].

Ecco una breve descrizione dei nodi coinvolti nel software di tracking in cui si sta operando:

- DedistNode: nodo che riceve uno stream video da videocamera omnidirezionale e lo rende disponibile dedistorto in rete. In particolare l'immagine originale ha dimensioni 640x480, mentre quella elaborata 1280x156. Sarà molto importante ricordare questa geometria, poiché su di essa si baseranno gli spostamenti dell'Ulisse.
- FallDetectionNode: questo nodo gestisce insieme sia il tracking che la fall detection.
- VideoEventReceiverNode: questo nodo riceve il video.

Figura 5.1: Immagine dedistorta della stanza senza oggetti in movimento.

5.2.1 Integrazione al FallDetectionNode

Il nodo è di tipo "GenericFilterNode" e la sua parte centrale è il metodo "processBuffer", in cui vanno specificate le operazioni che il nodo deve effettuare.

Poiché il tracking lavora "frame per frame", anche questo nodo lavorerà allo stesso modo. Risulta probabilmente utile riportare questa parte di codice, affinché sia immediatamente individuabile leggendo il sorgente fisicamente locato sulla macchina.

```
[...]

Buffer* FallDetectionNode::processBuffer(Buffer *in_buffer)
{

if (!inputImg)
inputImg = cvCreateImage( cvSize(width, height), IPL_DEPTH_8U, 3  );

if(!outputImg)
```

```
outputImg = cvCreateImage( cvSize(dedistWidth, dedistHeight), IPL_DEPTH_8U, 3  );


if(countFrames == numFramesDiscarded + 1){
countFrames = 0;

//acquisisco il frame e lo metto in un'immagine openCV
inputImg->imageData = in_buffer->getData();

list<TrackedContour> &inputFall = detector.GetTrackedList();
outputImg = detector.DoFrameAnalysis(inputImg, &TimeStamp);
outputImg = fallDetector.DoFallAnalysis(outputImg, inputFall );
list<TrackedContour> &targetFall = fallDetector.getTargetFall();

//gestione evento mouseClick----------------------------------
if(mouseClick){

CEvent* ce = new CEvent();

if(ULISSE){

TInValue<int>* posiX = new TInValue<int>(posX);
TInValue<int>* posiY = new TInValue<int>(posY);
Event* e1 = new Event("MOUSE_CLICK_ULISSE", 2);
e1->setValue(0,posiX);
e1->setValue(1,posiY);
ce->insertEvent(e1);
}
sendEventDownstr(ce);

mouseClick = false;
mouseCount = 1;
}
//plot del rettangolo sulla finestra openCV per il pointAndClick
if(mouseCount > 0 && mouseCount < 7){
cvRectangle(outputImg,cvPoint(posX-30, posY-30), cvPoint(posX+30, posY+30), cvScalar(255,0,0),
mouseCount++;
}else
mouseCount = 0;
//------------------------------------------------------------

//visualizzazione finestra di output. Viene fatto con le openCV e non con l'XDisplayNode
//per poter utilizzare il pointAndClick.
cvShowImage("Output", outputImg);
cvSetMouseCallback("Output", my_mouse_callback, (void*) outputImg);
cvWaitKey(10);

[...]
//parte di generazione dell'evento
```

```
if(fallDetector.existsTarget() && targetFall.front().sendEvent){

//caduta rilevata Ulisse
fallDetector.getAbsolutePosition(&hPos, &vPos);
TInValue<int>* id2 = new TInValue<int>((inputFall.back().ID));
TInValue<int>* posH = new TInValue<int>(hPos);
TInValue<int>* posW = new TInValue<int>(vPos);
TInValue<Time>* eTimestamp2 = new TInValue<Time>(clock.getTime());

// creo  downstream Event per Ulisse
Event* e1 = new Event("ULISSE_DETECTED_EVENT", 4);
e1->setValue(0,id2);
e1->setValue(1,posH);
e1->setValue(2,posW);
e1->setValue(3, eTimestamp2);
//CEvent* ce = new CEvent();
ce->insertEvent(e1);
sendEventDownstr(ce);

}
}
countFrames++;

//metodo per seguire il main object
//CvRect mainObj = detector.getMainObject();
int mainObjX = detector.getMainObjectX();
int mainObjY = detector.getMainObjectY();
int mainObjH = detector.getMainObjectH();
int mainObjW = detector.getMainObjectW();
TInValue<int>* posizX = new TInValue<int>(mainObjX+mainObjW/2);
TInValue<int>* posizY = new TInValue<int>(mainObjY+mainObjH/2);

// creo  downstream Event
Event* e2 = new Event("ULISSE_TRACK", 2);
e2->setValue(0,posizX);
e2->setValue(1,posizY);
CEvent* ce2 = new CEvent();
ce2->insertEvent(e2);
sendEventDownstr(ce2);

//creazione del buffer di output
out_buffer_size = outputImg->imageSize;
Buffer* out_buffer = getNewBuffer( out_buffer_size );
out_buffer->setUsed(out_buffer_size);
out_buffer->setData( outputImg->imageSize, outputImg->imageData );

in_buffer->release();// rilascio del buffer

return out_buffer;
```

```
} //processBuffer

[...]
```

Passi salienti:

1. "getData()" ottiene il frame contenuto nel buffer di input;
2. "mouseClick" gestisce il puntamento manuale del brandeggio (usate le OpenCV [13]);
3. "parte di generazione dell'evento" è il codice che si occupa di generare l'occorrenza di caduta rilevata e mette a disposizione con essa le coordinate assolute dell'oggetto identificato;
4. "metodo per seguire il main object" indica che i metodi sottostanti sono quelli implementati per far fare al brandeggio il following continuato del soggetto identificato;
5. "out_buffer" rende il frame (processato) disponibile sulla rete;

Si crei ora il file di descrizione del grafo (i file .gd di cui si è parlato nel capitolo di creazione dei flowgraph). Per far ciò è sufficiente inserire in un file di estensione ".gd" le seguenti righe:

```
TVReadCardNode2
! YUVtoRGBConverterNode
! DedistNode
! FallDetectionNode
! RGBtoYV12ConverterNode
! XDisplayNode
```

dove:

- TVReadCardNode2 è un nodo già presente in NMM che permette l'acquisizione dello streaming video mediante framegrabber;
- YUVtoRGBConverterNode è un nodo, già presente in NMM che permette la conversione dalla codifica YUV (di default per il framegrabber in uso) ad RGB che è il tipo di codifica utilizzata dal software di tracking;
- FallDetectionNode è il nodo abbiamo testé analizzato. Gestisce per NMM il tracking e l'identificazione delle cadute;
- RGBtoYV12ConverterNode è un nodo, già presente in NMM che permette la conversione dalla codifica RGB (che abbiamo usato per l'elaborazione dell'immagine) a YUV nuovamente;

- XDisplayNode è un nodo già presente in NMM che permette la visualizzazione di un flusso video (in codifica YUV). Si occupa autonomamente della generazione di una finestra di riproduzione.

5.2.2 Integrazione al VideoEventReceiverNode

Il nodo è di tipo *FilterNode* e la sua parte centrale è il metodo "VideoEventReceiverNode", in cui vanno specificate le operazioni che il nodo deve effettuare.

Questo nodo lo si potrebbe definire "gestore di eventi".

In pratica quel che fa è ricevere degli output (eventi), generati dagli altri nodi della rete, nel momento in cui accade qualcosa di cui si vuole il sistema si accorga.

Nello specifico della rilevazione delle cadute, riceverà (nel senso, in qualche modo, di "si accorge che esiste") un evento ogni qual volta il software di fall detection prima rileverà e poi genererà l'"evento caduta", favorendo così il passaggio delle relative informazioni (nel nostro caso delle coordinate) al codice che si occuperà di spostare la videocamera (PTZ o Ulisse Compact) cosicché inquadri il soggetto rilevato.

Si riporta il codice:

```
#include "VideoEventReceiverNode.hpp"
#include "nmm/plugins/NMMRegisterPlugin.hpp"
#include "nmm/nmm_prefix.hpp"

NMM_REGISTER_PLUGIN( VideoEventReceiverNode, NMM, "VideoEventReceiverNode" )

#define PI 3.14159265


/*parametri ulisse*/
#define centerX 522
#define centerY 100

#define panZero 21
#define tiltZero 21

#define MULTPANTILT 100
#define VELSTART "700"
#define VEL "700"

#define ALTEZZAULISSE 144 //pixel corrispondenti a 4 metri con 48 pixel al metro
/*parametri ulisse*/

namespace NMM{

//Queste funzioni sono quelle che vengono chiamate in caso di ricezione dell'evento relativo.
```

```
Result EventListener::fallDetectedUlisse(const int id, const int posH, const int posW, const T

LocalTime localtime = LocalTime(eTimestamp);
cout << "EventUlisse: caduta rilevata alle ore " << localtime.getHour()<<":"<<localtime.getMin

int iPan = panUlisse(posH,posW);
int iTilt = tiltUlisse(posH);

string pan = int2str(iPan);
string tilt = int2str(iTilt);

const char* cpan, *ctilt;
cpan = pan.c_str();
ctilt = tilt.c_str();

ulisse->GoToSpecificPosition((char*)cpan, (char*)ctilt, VEL, "0");

return SUCCESS;
}

Result EventListener::ulisseMove(const int posH, const int posW){
int iPan = panUlisse(posH,posW);
int iTilt = tiltUlisse(posH);

string pan = int2str(iPan);
string tilt = int2str(iTilt);

const char* cpan, *ctilt;
cpan = pan.c_str();
ctilt = tilt.c_str();

ulisse->GoToSpecificPosition((char*)cpan, (char*)ctilt, VEL, "0");
}

Result EventListener::mouseEventUlisse(const int x, const int y){

cout << "mouseEventUlisse" << endl;

int iPan = panUlisse(x,y);
int iTilt = tiltUlisse(x);

string pan = int2str(iPan);
string tilt = int2str(iTilt);

const char* cpan, *ctilt;
cpan = pan.c_str();
ctilt = tilt.c_str();

ulisse->GoToSpecificPosition((char*)cpan, (char*)ctilt, VEL, "0");
```

```
return SUCCESS;
}

//VIDEOEVENTRECEIVERNODE CLASS
VideoEventReceiverNode::VideoEventReceiverNode(const char* name )
: GenericFilterNode(name){

//registrazione evento in ascolto
registerEventListener("ULISSE_DETECTED_EVENT",
new TEDObject4<EventListener,
int, int, int, Time,
Result(EventListener:: * )(const int, const int, const int, const Time),Result>
(&m_event_listener, &EventListener::fallDetectedUlisse));

registerEventListener("MOUSE_CLICK_ULISSE",
new TEDObject2<EventListener,
int, int,
Result(EventListener:: * )(const int, const int),Result>
(&m_event_listener, &EventListener::mouseEventUlisse));

registerEventListener("ULISSE_TRACK",
new TEDObject2<EventListener,
int, int,
Result(EventListener:: * )(const int, const int),Result>
(&m_event_listener, &EventListener::ulisseMove));

camera = new CameraMotion;
ulisse = new UlissePositioning;

//set Ulisse to Zero position
int iPanZero = convertPan(panZero);
int iTiltZero = convertTilt(tiltZero);

string sPanZero = (itoa((int)iPanZero));
string sTiltZero = (itoa((int)iTiltZero));

ulisse->GoToSpecificPosition((char*)sPanZero.c_str(), (char*)sTiltZero.c_str(), VELSTART, "0");

pthread_create(&cameraThread, NULL, startCamera, (void *) camera);
}

[...]

//Il processBuffer non deve fare niente, soltanto inoltrare lo stream in input.
Buffer* VideoEventReceiverNode::processBuffer(Buffer * in_buffer){
return in_buffer;
}
```

```
}//nmm namespace

void *startCamera(void *cam){

if (cam){
CameraMotion *camera = (CameraMotion *) cam;
camera->StartCamera();
}

return 0;
}

//solo per Ulisse

//sx, dx
int panUlisse(int x, int y){

int diffX = x - centerX;
int diffY = y - centerY;
int angle = round(atan2(diffY,diffX)*180/PI);

cout << "diffX " << diffX << " diffY " << diffY << " angle " << angle << endl;
//angle += panZero;
cout << " angle-panZero " << angle << endl;
return convertPan(angle);

}

//up, down
int tiltUlisse(int x){

int diffX = x - centerX;
int y = ALTEZZAULISSE;
cout << "diffX " << diffX << " ALTEZZAULISSE " << y << endl;
int angle = round(atan2(diffX, y)*180/PI);

cout << "diffX " << diffX << " angle " << angle << endl;
if (angle > 0)
angle = -angle;
//angle = angle + 90 + tiltZero;
angle = angle + 90;

cout << "diffX " << diffX << " -angle " << angle << endl;

return convertTilt(angle);

}

int convertPan(int pan){
```

```
pan = - pan;
pan = pan - round((double)pan/360)*360;
if(pan > 0)
pan = pan*MULTPANTILT;
else
pan = 65536 + pan*MULTPANTILT;
return pan;
}

int convertTilt(int tilt){
tilt = - tilt;
//tilt = tilt - round((double)tilt/180)*180;
if(tilt > 0)
tilt = tilt*MULTPANTILT;
else
tilt = 65536 + tilt*MULTPANTILT;
return tilt;
}

string int2str(int number){
std::ostringstream sin;
sin << number;
std::string val = sin.str();
return val;
}

#include "nmm/nmm_suffix.hpp"
```

Passi salienti:

1. "parametri ulisse" sono variabili che tengono conto dello studio della geometria dell'ambiente di lavoro (laboratorio) e che servono a convertire le coordinate spaziali dell'oggetto rilevato in quelle assolute del brandeggio;

2. "fallDetectedUlisse" è il metodo che gestisce la telcamera nel momento in cui è rilevata una caduta. Si occupa quindi di assegnare un timestamp, di richiedere e memorizzare le coordinate ed infine di passare i parametri ai metodi che muoveranno l'Ulisse;

3. "ulisseMove" è il metodo che registra gli eventi legati al following continuo del soggetto e permette quindi di seguirlo durante il suo spostamento;

4. "mouseEventUlisse" è il metodo che permette l'utilizzo manuale del brandeggio con una metodologia del tipo "punta e clicca". In sostanza, cliccando su un punto dell'immagine di input (sempre visibile all'utente) è possibile inquadrare esattamente quella zona;

5. "VideoEventReceiverNode" questo è il cuore del nodo. Qui i metodi "in ascolto" registrano gli eventi rilevati e mettono a disposizione i dati conseguenti;

6. "GoToSpecificPosition" è il metodo, specifico per l'Ulisse Compact, che comunica a quali coordinate spostarsi;

7. tutti i metodi dopo il commento "solo per Ulisse", sono stati studiati per implementare il cambio di coordinate e le geometrie già accenntate.

Questo nuovo nodo andrà ovviamente aggiunto al file di descrizione precedentemente creato:

```
TVReadCardNode2
! YUVtoRGBConverterNode
! DedistNode
! FallDetectionNode
! RGBtoYV12ConverterNode
! VideoEventReceiverNode
! XDisplayNode
```

Alla fine il flow graph che implementeremo in rete, con i parametri settati risulterà:

```
TVReadCardNode2 $setDevice(‘‘/dev/video0’’) INITIALIZED
! YUVtoRGBConverterNode
! DedistNode
! FallDetectionNode
! RGBtoYV12ConverterNode
! VideoEventReceiverNode
! XDisplayNode $setPosition(0,0) INITIALIZED

TVReadCardNode2 $setDevice(‘‘/dev/video1’’) INITIALIZED
! XDisplayNode $setPosition(0,540) INITIALIZED
```

Per completezza si riporta brevemente il significato di quel codice non precedentemente spiegato:

- "$setDevice("/dev/video0") INITIALIZED" è un comando interno di NMM che serve a specificare la periferica utilizzata (in questo caso il device video). Nel caso di input diversi, basta modificare la stringa tra parentesi tonde;

- "$setPosition(0,0) INITIALIZED" passa invece dei parametri utili al settaggio del brandeggio. Questi vanno determinati di volta in volta, a seconda del posizionamento delle proprie telecamere.

Capitolo 6

Conclusioni

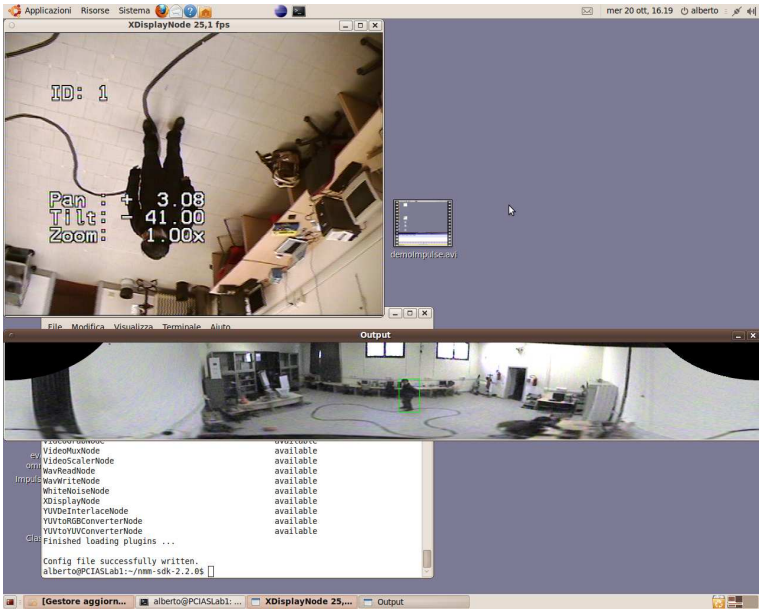


Figura 6.1: Il brandeggio Ulisse Compact che durante il tracking inquadra esattamente il soggetto in movimento.

Il lavoro svolto in questa tesi si inserisce in quello molto più ampio dello sviluppo di un software di tracking completo e performante, ai fini della videosorveglianza.

Sia essa rivolta al controllo del territorio, piuttosto che all'utilizzo in ambienti di particolare sensibilità, data l'enorme importanza ricoperta, è sicuramente opportuno che efficienza ed efficacia dei sistemi implementati vadano di pari passo.

Quel che emerge dagli studi effettuati per arrivare ad integrare il brandeggio

oggetto dell'elaborato, nel preesistente ambiente di lavoro, è quello di una situazione che ottiene i risultati cercati, ma che può certamente essere migliorata sotto diversi aspetti.

Innanzitutto l'utilizzo dei software opensource è certamente una scelta ottimale per quanto concerne la potenza e versatilità di GNU/Linux e la sua predisposizione ad un utilizzo multiutente ricco di strumenti rivolti alla programmazione. Lo è meno quando si utilizzano programmi poco conosciuti e quindi dove, in caso di problemi, si è un po' lasciati a sè stessi. È il caso del middleware NMM che ha richiesto pazienti settimane solo per l'ottenimento di un'installazione funzionante ed altrettanto tempo si è dovuto dedicare alla sua configurazione.

Quanto poi concerne l'utilizzo vero e proprio, la scarsità di documentazione ha rallentato pesantemente i tempi e solo un lavoro a più mani ha permesso di raggiungere dei risultati. Soprattutto quando è stato il momento di cimentarsi con il trasferimento su rete di nodi per flussi multimediali che non fossero audio, bensì video, ci si è resi conto di come poco ancora la stessa azienda produttrice, Motama, abbia collaudato il suo prodotto in quel settore.

Nel percorso di questo svolgimento si è arrivati persino a dare un contributo allo sviluppo di NMM, grazie all'identificazione di bug, poi comunicati attraverso il forum ufficiale, cosa che ha determinato addirittura il rilascio di una nuova relase. Tutto questo non mette comunque in ombra le potenzialità di un pacchetto di strumenti davvero potenti e promettenti.

L'Ulisse, al termine del lavoro, risulta comandabile in qualsiasi delle sue funzioni da remoto, è interfacciato col software di rilevamento delle cadute (il brandeggio in caso di caduta rilevata, inquadra il soggetto) ed anche in realtime col tracking. Si è implemetata infatti una funzione che potremmo definire di "following", con la quale man mano che vengono identificati gli oggetti da trackare, li faccia seguire alla telecamera in ogni loro spostamento, in generale, senza perdere la giusta inquadratura.

Ci sono poi dei limiti legati a vari fattori.

Il primo è certamente esterno e riguarda il software di motion detection. Esso non è sempre impeccabile e capita quindi che faccia spostare il brandeggio sugli oggetti sbagliati o capita addirittura che li perda per qualche istante e questi intanto escano dall'inquadratura.

Inoltre ogni qual volta l'oggetto trackato non verifica le ipotesi fatte, le cose non funzionano più. È il caso ad esempio in cui per qualche motivo, a volte per la prospettiva, altre per veri e propri errori di "identificazione del soggetto", il bounding box generato per il contornamento di quel che si sta muovendo, non risulta "il più grande" tra quelli generati. Poiché l'Ulisse segue rigorosamente quest'ultimo, è ovvio che vada a puntare nella direzione errata.

Il secondo è dovuto al tipo di modellizzazione dell'ambiente. Certamente semplificata, non permette un'ottimale posizionamento, in senso assoluto, degli oggetti in movimento nella stanza. Soprattutto se questi si trovano in setto-

ri particolari, quali gli angoli o nelle zone limitrofe alle pareti. Per ovviare a questo è comunque possibile agire sul tuning del sistema, migliorandolo anche con l'ausilio di correttori appropriati che tengano conto delle limitazioni e del brandeggio e del motion detector.

Il terzo ed ultimo è dovuto alla gestione del movimento a 360°, col soggetto quindi in grado di raggiungere ogni punto della stanza, in particolare il punto sotto la telecamera e di poterla aggirare sia da nord a sud, sia da est a ovest. Per esemplificare, questo comporta che quando una persona è esattamente sotto ad essa, l'asse tilt raggiunge il fondo scala a +90°. Se ora lo spostamento prosegue in linea retta, il brandeggio è costretto a far risalire il tilt, ruotare di 180° sul pan e poi scendere nuovamente a circa +90°. La complessità del movimento comporta qualche volta degli errori nell'inquadratura.

Infine è possibile vedere uno screenshot del programma in esecuzione (Figura 6.1). Il soggetto al centro della stanza si sta muovendo e il controllo dell'Ulisse lo mantiene ben inquadrato al centro dell'immagine che si nota essere però sottosopra. Questo dipende dal fatto che la telecamera è stata fissata al soffitto, ma normalmente il suo utilizzo è sulla sommità di pali o colonne e quindi di default è ruotata nell'altro senso. Si è speso parecchio tempo nel cercare la funzione di ribaltamento dell'immagine, ma pur seguendo le indicazioni dell'azienda produttrice non si è avuto successo.

Il sitema risulta in definitiva adatto allo scopo per cui lo si è studiato, ma ha certamente ampi margini di miglioramento. Sarebbe auspicabile un'adozione lunga nel tempo del middleware, per averne una vera e più completa conoscenza e viste le sue potenzialità potrà certamente rivelarsi una scelta vincente.

Il tracking, pur funzionando, ha dei limiti causati da situazioni si videoripresa particolari, alcune delle quali accennate precedentemente (vedi generazione di blob in caso di ombre). Una sua revisione e integrazione con altre tecniche, quale ad esempio il meanshift tracking basato su istogrammi di colore, potrebbe portare ad un efficacia nettamente superiore, con benefici che si tradurrebbero immediatamente a cascata sulla fall detection e sulla gestione dei brandeggi.

In futuro resta anche aperta la possibilità di passare da un utilizzo delle telecamere indoor ad uno outdoor, dove certamente uno strumento come l'Ulisse Compact può dare il meglio di sè, con le sue tecnologie a grandi zoom ottici, resistenza alle condizioni atmosferisce e movimenti a 360°.

Bibliografia

[1] L. Lamport, *LaTeX: A Document Preparation System*. Addison Wesley Professional, 2nd ed., June 1994. ISBN: 0-201-52983-1.

[2] A. Baudoin, "Impara LaTeX!." documentazione online di Latex, 1998.

[3] B. Lain, "Tracking ptz in tempo reale per videosorveglianza." Tesi di laurea magistrale.

[4] D. Zattara, "Ulisse macro protocol." Azienda Videotec S.p.A.

[5] P. V. Massimo Romagnoli, *Linguaggio C e C++*. Petrini Editore, 1st ed., Gennaio 2001.

[6] Motama, "Nmm site - http://www.motama.com/nmm.html."

[7] D. Forsyth and J. Ponce, *Computer Vision - A modern approach*. Prentice Hall Professional Technical Reference, 1st ed., 2002. ISBN:013-0-85198-1.

[8] R. L. D. Bloisi, L. Iocchi, "Modellazione del background segmentazione tracking e proiezione sul sistema di riferimento gis globale," Giugno 2007.

[9] S. GNU/Linux, "Slackware site - http://www.slackware.com."

[10] A. Iannella, "Serial port programming - http://slackware.osuosl.org/slackware-3.3/docs/mini/serial-port-programming."

[11] E. M. Stefano Ghidoni, Alberto Pretto, "Cooperative tracking of moving objects and face detection with a dual camera sensor,"

[12] A. Salamone, "Sistemi di telecamere intelligenti per la videosorveglianza." Tesi di laurea magistrale.

[13] G. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly, 1st ed., September 2008. ISBN: 978-0-59651613-0.

Elenco delle tabelle

Elenco delle figure