UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Tesi di Laurea

Sistema client-server per la gestione della telecamera Omnidome®

Relatore
Ch.mo prof. Emanuele Menegatti

Correlatore
Ing. Stefano Ghidoni

Laureando
Morris Soragni

Corso di Laurea in Ingegneria Informatica (V.O.)
Anno Accademico 2010/2011

24 Aprile 2012

ai miei nonni
...mi mancate

Indice

Elenco delle figure

Elenco dei listati

Capitolo 1

Introduzione

Nell'ambito dei sistemi di videosorveglianza la possibilità di visualizzare e controllare da remoto le telecamere sta rivestendo negli ultimi anni un aspetto sempre più importante nello sviluppo dei relativi software. L'ormai diffusa disponibilità di connettività a banda larga, anche *mobile*, rende possibile a gran parte dell'utenza, inclusa quella domestica, l'accesso a tale tecnologia. Uno degli aspetti negativi della maggior parte delle soluzioni di videosorveglianza in commercio è nella necessità di installare sul computer dal quale si intende controllare il sistema un software specifico, fornito dal produttore del sistema stesso; ciò ne rende difficoltoso l'utilizzo non solo da parte di persone con scarsa cultura informatica, ma anche da chi per varie motivazioni non ha sempre il proprio computer a disposizione.

In questo lavoro si è posta l'attenzione su uno dei prodotti per videosorveglianza di punta, l'**Omnidome®** di IT+Robotics Srl, creando un sistema client-server di visione e controllo remoto che non necessita di installazioni sui computer client.

L'utilizzo di un web-server, a cui gli utenti possono collegarsi come ad un normale sito web, rende disponibile il sistema implementato a chiunque sia in possesso di un browser, non escludendo quindi gli utenti *mobile*. L'implementazione di tecnologie del cosiddetto *web 2.0*, come AJAX, rende la fruizione del servizio il più semplice possibile, pur mantenendo un alto livello di efficacia, guidando l'utente nelle varie fasi di utilizzo di Omnidome®. Una connessione ad Internet e un browser web sono quindi gli unici requisiti per poter visualizzare le immagini e controllare il sistema tramite il *Web Con-*

trol Center, da qualunque postazione anche remota; il web-server consente la visione contemporanea delle immagini a tutti gli utenti collegati, e l'integrazione di un sistema di gestione delle prenotazioni permette all'utente il controllo esclusivo, seppur limitato nel tempo, di Omnidome®.

Questo lavoro è strutturato come segue: nel capitolo 2 descriveremo la telecamera Omnidome®, illustrando le caratteristiche che la rendono un prodotto di punta nel campo della videosorveglianza: l'utilizzo congiunto di una telecamera panoramica, in grado di riprendere contemporaneamente tutto l'ambiente circostante a 360°, e di una telecamera prospettica montata su brandeggio mobile, che permette di inquadrare ad alta risoluzione una zona specifica dell'ambiente.

Nel capitolo 3 discuteremo delle scelte implementative: tratteremo in particolare le motivazioni che ci hanno spinto alla realizzazione di un sistema client-server basato sul web. Descriveremo le modalità con cui si sono utilizzate la tecnologia AJAX e la libreria jQuery per migliorare l'esperienza d'uso da parte degli utenti; descriveremo inoltre come si siano integrate le componenti web e le componenti C++ su un PC GNU/Linux tramite web-server Apache e *Shared Memory*.

Nel capitolo 4 tratteremo della comunicazione tra i client e il server: descriveremo ØMQ, la libreria di cui si è fatto uso in questo lavoro, mostrando come la si sia utilizzata per realizzare due diversi livelli di comunicazione tra le parti.

Passeremo quindi a descrivere il sistema "lato server" nel capitolo 5, mostrando il software C++ specifico per l'*unwarping* e l'*unwrapping* delle immagini panoramiche e le modalità con cui le immagini vengono rese disponibili al *Web Control Center*.

Nel capitolo 6 descriveremo le modalità con cui l'utente ottiene l'accesso ad Omnidome®; in particolare, tratteremo il sistema di prenotazione e la gestione della coda di utenti, sia all'interno del server sia dal punto di vista dei client.

L'interfaccia utente di controllo di Omnidome® verrà mostrata nel capitolo 7, descrivendo come viene comandato il brandeggio della telecamera prospettica direttamente dal *Web Control Center*, sia tramite controlli manuali che tramite un sistema intelligente di puntamento.

Capitolo 2

Omnidome

2.1 L'idea

La *computer vision* è una disciplina che negli ultimi anni ha visto una notevole crescita di interesse: si pensi ad esempio alle applicazioni nel campo della videosorveglianza, o alla ricostruzione tridimensionale di ambienti navigabili tramite immagini. È quindi naturale pensare che, con questa crescita di interesse, ci sia stato un parallelo progredire nello sviluppo di nuovi algoritmi e di nuovi tipi di sensori per ottenere sistemi intelligenti di alta qualità.

I tipi di sensori più comunemente utilizzati comprendono:

- telecamere prospettiche, fisse o motorizzate;
- telecamere panoramiche;
- telecamere all'infrarosso.

Questi sensori vengono utilizzati per riprendere qualsiasi tipo di attività, umana o ambientale; le immagini riprese possono poi essere trasmesse direttamente ad un ricevitore, possono essere immagazzinate o possono essere elaborate al fine di estrapolarne *features* di interesse (ad esempio, ricerca di soggetti estranei alla scena o controlli di qualità sul materiale).

Tuttavia questi sensori non sono in grado di soddisfare completamente le esigenze che si possono manifestare al giorno d'oggi; concentrandoci solo sulle prime due tipologie di telecamere si possono facilmente rilevare i seguenti problemi:

- telecamere prospettiche fisse: per la loro natura offrono un campo visivo limitato, estensibile tramite l'uso di ottiche grandangolari, che ne vanno però a limitare il livello di dettaglio ottenibile;

- telecamere prospettiche motorizzate: offrono un miglioramento rispetto alle telecamere prospettiche fisse, in quanto tramite l'uso di opportuni brandeggi possono compiere movimenti in grado di inquadrare tutto (o almeno una gran parte, a seconda delle caratteristiche meccaniche del brandeggio) l'ambiente circostante; in ogni istante però si può considerare la telecamera come se fosse fissa, incontrando nuovamente quei problemi di limitatezza del campo visivo già visti in precedenza: tutto quello che accade all'esterno del campo visivo istantaneo viene perso;

- telecamere panoramiche omnidirezionali: garantiscono la visione contemporanea dell'ambiente circostante nella sua globalità, ma a scapito della risoluzione: l'elevata dimensione del campo visivo costringe infatti ad un basso livello di dettaglio delle immagini acquisite;

- telecamere panoramiche tramite serie di telecamere: non rientrano propriamente nella categoria delle telecamere panoramiche, poichè si tratta di un gruppo di telecamere prospettiche opportunamente posizionate in modo da coprire l'intero ambiente circostante; rimangono i problemi legati ad eventuali punti ciechi e al coordinamento delle immagini riprese, che per la natura delle telecamere possono avere livelli di dettaglio diversi.

L'idea alla base di Omnidome®, marchio registrato di IT+Robotics Srl, è di integrare due tipi di sensori, ovvero una telecamera panoramica e una telecamera prospettica motorizzata, in un unico dispositivo in grado quindi di avere una visione globale a bassa risoluzione dell'ambiente in cui è inserito, ma con la possibilità di aumentare notevolmente il livello di dettaglio di una particolare scena tramite la telecamera prospettica.

Questo può essere realizzato tramite l'implementazione di un sistema software intelligente che utilizza le *features* estratte dalle immagini catturate dalla telecamera panoramica per controllare direttamente il brandeggio della telecamera prospettica, come ad esempio in [1] per il *tracking* di soggetti in movimento e il riconoscimento facciale.

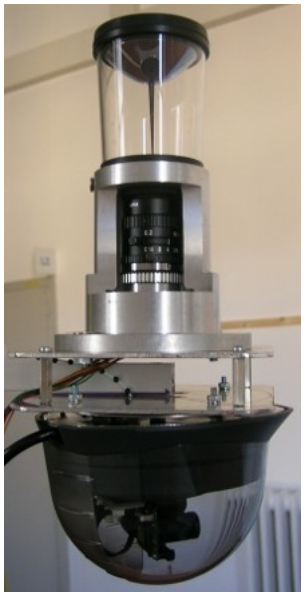

Figura 2.1: Il sensore Omnidome

2.2 L'hardware

Abbiamo già detto come Omnidome® integri due diversi sensori, uno panoramico e uno prospettico. Nel seguito mostreremo i particolari sensori utilizzati nella versione di Omnidome® messa a disposizione per questo lavoro (si veda la fig. 2.1.

2.2.1 Componente panoramica

La componente panoramica di Omnidome® consiste in una normale telecamera prospettica ad ottica fissa (sensore CCD + ottica *C-mount*), sulla quale viene però montato uno specchio iperbolico omnidirezionale che consente la visione a 360° attorno al sensore. Caratteristica peculiare di questi specchi è l'ago centrale, ben visibile nella fig. 2.1, il cui scopo è di eliminare eventuali fenomeni di riflessione della luce dovuti al vetro esterno.

2.2.2 Componente prospettica

La componente prospettica è realizzata tramite l'unità *dome-camera* Mistral, di Videotec Spa, che integra un brandeggio a velocità variabile con una telecamera a colori. Si ha quindi in un'unica unità sia la parte di motorizzazione

che la parte video, con vantaggi in termini di assemblaggio dell'Omnidome®: Mistral è molto compatta, contribuendo pertanto a mantenere contenuti gli ingombri di Omnidome®, e il sistema di cablaggio consente di poter alimentare l'unità tramite un'unica sorgente.

Figura 2.2: Dettaglio del *pigtail cable* di Mistral

Nella figura 2.2 si può notare come, tramite un unico cavo denominato *pigtail cable* per la sua conformazione, sia possibile convogliare l'alimentazione, i segnali di telemetria (controllo motori) e il segnale video. Nella figura 2.3 viene invece mostrato lo schema di collegamento su connettore RJ-45 (alternativo al *pigtail cable*) dei cavi destinati a trasportare i vari segnali:

Pin	Cable	Signal
1	Orange/White	Outgoing RS-422 + / RS-485 A
2	Orange	Outgoing RS-422 – / RS-485 B
3	Green/ White	Incoming RS-422 +
4	Blue	Supply voltage (Positivo alimentazione)
5	Blue / White	Supply Ground (GND)
6	Green	Incoming RS-422 –
7	Brown/ White	Video out+
8	Brown	Video out–

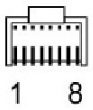

Figura 2.3: Schema di cablaggio su RJ-45 di Mistral

Lo schema di collegamento su RJ-45 permette di sfruttare gli eventuali cablaggi già esistenti nel sito dove va montata l'unità Mistral: il connettore RJ-45 è infatti il connettore standard delle *Local Area Network* (LAN). Ciò consente di collegare l'unità Mistral ad un'unità di controllo senza dover intervenire fisicamente sui cablaggi del sito.

Per entrambi gli schemi di collegamento, il segnale di telemetria può essere convogliato tramite due diversi standard:

- RS-485: utilizza una sola coppia di connettori e consente la sola trasmissione di comandi all'unità;
- RS-422: utilizza due coppie di connettori e consente sia l'invio di comandi che la ricezione di informazioni dall'unità.

Entrambi gli standard sono conosciuti anche come "seriale industriale", per l'analogia del connettore terminatore con quello RS-232 dei normali computer (porta seriale); gli standard non sono però compatibili tra di loro ed è quindi necessario utilizzare un convertitore di segnale RS-232/485 (o RS-232/422) per la connessione dell'unità ad un normale computer, come nel nostro caso.

La meccanica del brandeggio ha le seguenti caratteristiche:

- escursione orizzontale: $\pm 90°$ (per un totale quindi di 180° di escursione in PAN);
- escursione verticale: da $-10°$ a $+90°$ (per un totale di 100° di escursione in TILT);
- *preset*: possbilità di memorizzare 32 diverse configurazioni di posizioni PAN e/o TILT da richiamare tramite comando, più 4 configurazioni preimpostate di fabbrica;
- velocità brandeggio: da 0.5 a 40°/s, su 7 diversi livelli; un ulteriore livello di 200°/s per il richiamo di un *preset*.

Il protocollo utilizzato per la telemetria in quest'unità è il protocollo MACRO, che vedremo in dettaglio in § 2.3.

2.3 Il protocollo MACRO

2.3.1 Descrizione

Il protocollo MACRO è un protocollo di comunicazione su porta seriale; i messaggi sono costituiti da caratteri ASCII stampabili e sono inviati all'interno di appositi delimitatori, che hanno la funzione di denotare l'inizio e la fine della trasmissione. Un algoritmo di checksum inoltre permette di determinare un carattere, da inserire all'interno della stringa di comando, che viene utilizzato

dall'unità per verificare la corretta trasmissione della stringa stessa. Il protocollo è *case-sensitive*, ovvero viene fatta distinzione tra caratteri maiuscoli e minuscoli; all'interno della stringa di comando non sono consentiti spazi.

2.3.2 Parametri porta seriale

Si rende necessario impostare alcuni parametri della porta seriale per ottenere una comunicazione corretta; nel listato seguente viene mostrato il codice C++ che setta questi parametri:

```
CameraPositioning::CameraPositioning(const char *device)
{
  // Apro la porta
  if (device != 0)
    port = open(device, O_RDWR | O_NOCTTY | O_NDELAY);
  else
    // Porta di default: /dev/ttyS0
    port = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY);

  if (port == -1)
    fprintf(stderr, "Unable to open port %s\n\n", device);

  struct termios flags;
  tcgetattr(port, &flags);

  // Impostazione velocità di trasmissione: 9600 baud
  cfsetispeed(&flags, B9600);
  cfsetospeed(&flags, B9600);

  // Disabilita la parità
  flags.c_cflag &= ~PARENB;
  // Abilita il controllo del numero di bit dati
  flags.c_cflag &= ~CSIZE;
  // Imposta 8 bit di dati
  flags.c_cflag |= CS8;
  // Setta "stop bit" pari a 2
  flags.c_cflag |= CSTOPB;

  // Imposta i nuovi parametri quando termina la scrittura
      corrente
  tcsetattr(port, TCSADRAIN, &flags);
}
```

Listato 2.1: Impostazione dei parametri della porta seriale

Una diversa configurazione di queste impostazioni, specialmente per quanto riguarda la parità o gli "stop bit", causa errori nella trasmissione dei comandi

con conseguente impossibilità di controllare l'unità.

2.3.3 Comandi supportati

Movimento nelle direzioni

Il protocollo, essendo destinato all'utilizzo con molteplici tipologie di brandeggi, supporta un grande numero di comandi, tutti identificati univocamente tramite sequenza di caratteri ASCII; se ne è sfruttato solamente un sottoinsieme per realizzare la movimentazione dell'unità Mistral, nello specifico si sono implementati i seguenti comandi di movimento:

Comando	Parametri	Descrizione
Up	v_t	Movimento verso l'alto
Down	v_t	Movimento verso il basso
Right	v_p	Movimento verso destra
Left	v_p	Movimento verso sinistra
Stop		Stop al movimento

I valori v_p e v_t passati come parametri ai comandi indicano rispettivamente il livello di velocità in PAN e in TILT con cui viene eseguito il movimento. Tale livello corrisponde a uno dei 7 valori supportati dall'unità Mistral.

Il protocollo MACRO non supporta i "movimenti in posizione" (si veda il § 7.1 a pag. 50), ma consente solamente di indicare una delle direzioni in cui effettuare il movimento, con la relativa velocità; nel listato 7.1 a pag. 50 si mostrerà come si possano comunque sfruttare questi comandi assieme al comando "*Stop*" per realizzare questa importante tipologia di movimenti.

Movimento al preset

Oltre ai comandi appena visti, va citato il comando "*GoToPresetPosition*", che consente all'unità di muoversi verso uno dei 32 *preset* memorizzabili (a cui se ne aggiungono 4 preimpostati e non modificabili). Questo comando verrà utilizzato per l'inizializzazione del brandeggio ad una posizione nota (si veda § 7.1 a pag. 52).

Capitolo 3

Scelte Implementative

3.1 Introduzione

Uno degli aspetti su cui si focalizza questo lavoro è cercare di rendere semplice all'utente l'utilizzo di Omnidome®. Per questo motivo si sono effettuate delle scelte implementative, in special modo per quanto riguarda la parte di codice più "a stretto contatto" con l'utente stesso:

- l'utilizzo delle tecnologie **AJAX**, che consente lo scambio di dati asincrono con il server, permettendo quindi una visualizzazione dinamica della pagina senza interruzioni;
- il **long polling**, per ottimizzare le tempistiche delle notifiche e ridurre gli sprechi nell'utilizzo della rete;
- **jQuery**, una libreria JavaScript che permette di effettuare modifiche *real-time* sui contenuti della pagina e che realizza l'implementazione dei due punti precedenti;
- l'uso della **Shared Memory**, che consente di utilizzare file mantenuti in memoria come se fossero file "reali" e di condividerli tra programmi diversi.

Nel seguito vedremo in dettaglio ciascuna di queste scelte; il paragrafo finale sarà inoltre dedicato a "mettere assieme i pezzi": mostreremo come si sono integrate tutte le tecnologie all'interno della componente web del software di Omnidome®, consentendo ai vari linguaggi (HTML, JavaScript, PHP e C++) di *dialogare* tra loro.

3.2 Ajax

AJAX, acronimo di *Asynchronous JavaScript And XML*, è un insieme di tecnologie che vengono utilizzate assieme per realizzare applicativi web interattivi. L'origine del termine e la formalizzazione della caratteristiche è rintracciabile in [6].

Ciò che differenzia notevolmente la tecnologia AJAX dai paradigmi standard di programmazione orientata al web e che ne fa la caratteristica fondamentale è la modalità con cui client (il browser web) e server (il web-server) si scambiano i dati, scambio che avviene in background: ciò consente, ad esempio, di poter aggiornare dinamicamente la pagina che si sta visitando, o una porzione di pagina, senza un'esplicita richiesta di *refresh* da parte dell'utente.

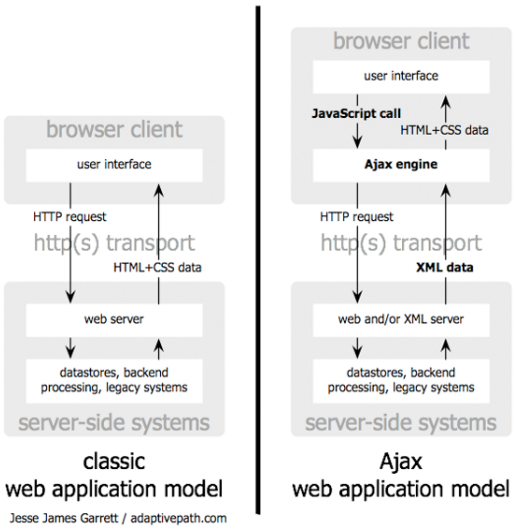

Figura 3.1: Confronto tra i paradigmi tradizionale e AJAX-based

Un esempio

Se, rifacendoci ad una situazione reale, pensiamo a come sono strutturate le *webmail*, ci rendiamo conto che una gran parte della pagina visualizzata si ripete ad ogni azione: se passiamo dalla lista dei messaggi a leggere un messaggio specifico, tutta la parte di intestazione o di controllo delle cartelle rimane invariata, mentre solamente il riquadro destinato ai messaggi viene

aggiornato. Utilizzando un paradigma di programmazione standard, al click sul messaggio desiderato corrisponde un *submit* al server, il quale risponde ricaricando tutta la pagina con il messaggio richiesto al posto della lista.

È immediato verificare che questo paradigma comporta un elevato scambio di dati tra client e server: le pagine vengono ogni volta ricaricate completamente nonostante molte parti rimangano invariate, causando un notevole spreco di banda. L'applicativo risulta inoltre più lento, dato che le richieste al web-server vengono fatte ogni volta che si interagisce con la pagina e che il client deve attendere le risposte prima di poter continuare.

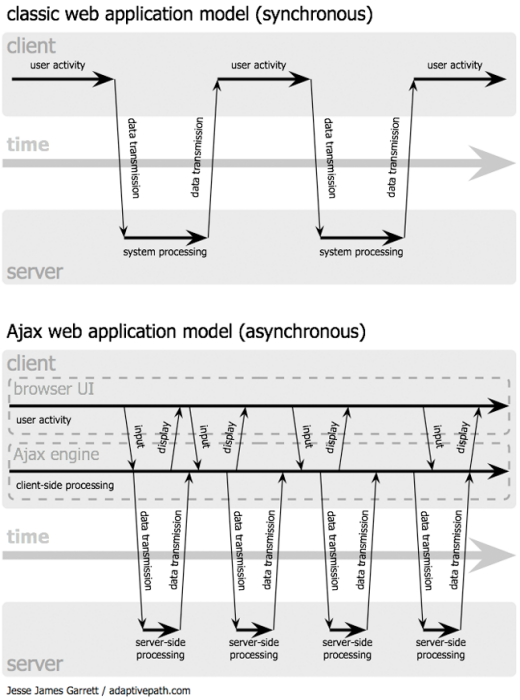

Figura 3.2: Confronto tra i modelli di interazione tradizionale e AJAX-based

Utilizzando invece la tecnologia AJAX, è possibile richiedere al web-server solamente la porzione di pagina che va aggiornata: nell'esempio della webmail, al click sul messaggio desiderato verrà inviata una richiesta (in background) al web-server; utilizzando JavaScript poi l'applicativo mostrerà la risposta del server (il messaggio, nel nostro esempio) nel browser.

Si nota subito la differenza rispetto all'uso del solo HTML: non si va più a ricaricare l'intera pagina, diminuendo di molto il traffico dati col server; la

presentazione dei dati in arrivo è inoltre affidata al client, consentendo così di diminuire il carico di lavoro del web-server. Si ottiene in questo modo un'applicativo più veloce e reattivo. Le richieste asincrone permettono al client, come mostrato in figura 3.2, di poter continuare ad interagire con la pagina, senza dover attendere le risposte dal server.

Componenti

Le componenti principali della tecnologia AJAX sono:

- un insieme di regole di rappresentazione e di stile, ottenuto tramite l'utilizzo di (X)HTML e CSS;
- un sistema di visualizzazione delle informazioni e di interazione, ottenuto tramite manipolazione dell'albero DOM (acronimo di *Document Object Model*) via JavaScript;
- un oggetto per lo scambio asincrono in background di dati, quale ad esempio XMLHttpRequest;
- un formato per lo scambio di dati, che solitamente è XML, ma può includere HTML, JSON o anche testo semplice;
- l'utilizzo di JavaScript come collante tra le varie componenti.

L'insieme di tecnologie AJAX è utilizzabile sui più diffusi sistemi operativi e browser web; per quanto riguarda le implementazioni, esistono numerosi framework e librerie, anche open-source; tra queste vi è **jQuery**, la libreria utilizzata in questo lavoro anche per implementare le tecnologie AJAX, che vedremo in § 3.4 a pag. 15.

3.3 Long polling

Il *long polling* è una tecnica largamente impiegata per simulare il *push* di notifiche dal server ai client, ovvero l'invio di informazioni da parte del server senza che vengano richieste esplicitamente dal client (si pensi, ad esempio, al funzionamento di alcuni *social network*); nel long polling, in modo simile a quanto accade in una normale connessione, il client contatta il server per una richiesta di informazioni (denominata appunto *polling*). Il server, se

ha a disposizione l'informazione richiesta, la invia immediatamente al client; se l'informazione non è disponibile, invece di inviare una risposta (che può essere vuota, o un messaggio sulla non disponibilità dell'informazione), il server mantiene aperta la connessione fino a quando i dati da inviare non diventano disponibili. Una volta ricevuta la risposta, il client solitamente effettua subito una nuova richiesta: in questo modo il server ha sempre una richiesta pendente.

Il vantaggio dell'utilizzo del *long polling* è proprio nel mantenere aperta la connessione tra server e client: portando come esempio il nostro sistema di prenotazione, che vedremo in dettaglio nel § 6.2 a pag. 36, abbiamo la necessità da parte dei client di conoscere le variazioni nello stato della coda sul server; in uno scenario implementativo senza *long polling* avremmo dovuto impostare un *setInterval* JavaScript, dentro cui andare a richiedere l'informazione al server. Questo comporterebbe però la creazione di una nuova connessione ogni volta che la funzione nel *setInterval* viene richiamata, con un notevole spreco di risorse; si deve considerare infatti che il server risponde comunque alla richiesta, anche se non ha informazioni a disposizione, causando quindi la successiva chiusura della connessione da parte del client. Si può essere tentati di rimediare aumentando l'intervallo della funzione di richiesta, a scapito però della precisione temporale nel ricevere l'informazione: avendo impostato, ad esempio, un intervallo di 10 secondi e supponendo che la variazione nella coda si produca dopo 2 secondi dall'ultima richiesta, sarebbe comunque necessario attendere 8 secondi prima di venirne a conoscenza.

Tramite *long polling*, al contrario, l'informazione sulla variazione nella coda viene ricevuta dal client non appena questa si verifica: la connessione è già in attesa di veicolare questa informazione.

3.4 jQuery

jQuery è una libreria JavaScript che si propone di semplificare la selezione e manipolazione degli elementi HTML, la gestione degli eventi, le animazioni di transizione e la comunicazione AJAX; l'essere *free* e open-source ne hanno decretato il rapido successo, tanto che oltre il 55% dei 10.000 siti più visitati utilizza jQuery (fonte: [7]); altre librerie analoghe non superano il 5%, con

un bacino d'utenza molto inferiore ai 24 milioni di siti che invece utilizzano jQuery (si veda [8] per una lista dei principali siti).

Caratteristiche

Alcune delle caratteristiche principali di jQuery sono:

- *Sizzle*, il motore di selezione open-source, che consente di effettuare selezioni di singoli elementi e/o interi gruppi HTML;
- modifica e attraversamento dell'albero DOM (*Document Object Model*), anche sulla base di proprietà CSS come "*id*", "*class*" e "*name*";
- gestione degli eventi, che vanno dal click del mouse o la pressione di un tasto alla gestione del *submit*;
- effetti e animazioni per la transizione degli elementi;
- funzioni specifiche per l'utilizzo delle tecnologie AJAX;
- estensione delle funzionalità tramite l'utilizzo di plugin.

3.5 Shared Memory

In campo software, si indica con **Shared Memory** sia una realizzazione pratica della comunicazione tra processi, sia un metodo per diminuire l'utilizzo della memoria tramite *virtual mappings*, ovvero la mappatura plurima della stessa risorsa in modo tale da renderla disponibile a più processi.

In questo lavoro si è utilizzata la Shared Memory per realizzare la comunicazione tra processi: si utilizza una porzione di memoria RAM per contenere le immagini prodotte dal software C++, immagini che poi vengono mostrate all'utente tramite il *Web Control Center*.

Il vantaggio dell'utilizzo della Shared Memory è nella maggior velocità rispetto ai metodi di inter-comunicazione tradizionali, quali le *named pipe* o gli *Unix domain socket*; si è però limitati per quanto riguarda la distribuzione dei processi, che devono risiedere tutti sulla stessa macchina fisica.

Esistono varie implementazioni della Shared Memory, alcune standardizzate come le API POSIX [9]. In questo lavoro si è tuttavia voluto sfruttare una recente incarnazione di questa tecnologia, offerta dai kernel linux 2.6 e

successivi: viene creato un *device* speciale, **/dev/shm**, sotto forma di RAM-disk; questo device è caratterizzato dall'essere scrivibile da tutti gli utenti del sistema, garantendo quindi l'accesso anche ai software che vengono lanciati tramite utenti non privilegiati.

3.6 Integrazione con il software di Omnidome

Nel nostro lavoro si sono sfruttate entrambe le categorie di funzioni jQuery:

- funzioni $(), specifiche dell'oggetto jQuery, che vengono utilizzate per selezionare gli elementi e operare su di essi;
- funzioni di convenienza, come *$.ajax()*, che non operano sull'oggetto jQuery specifico.

Di seguito verranno elencate alcune funzionalità richieste a questo lavoro che sono state gestite tramite jQuery. L'ultimo paragrafo sarà invece dedicato all'integrazione della *Shared Memory*.

3.6.1 Selezione degli elementi

L'uso di attributi "*id*", "*class*" e "*name*" adeguati consente di effettuare selezioni sugli oggetti in maniera rapida:

```
<div id="ptz_up"><input type="image" src="up.png"
    class="image" name="btn_go_u" /></div>
<div id="ptz_left"><input type="image" src="left.png"
    class="image" name="btn_go_l" /></div>
<div id="ptz_right"><input type="image" src="right.png"
    class="image" name="btn_go_r" /></div>
<div id="ptz_down"><input type="image" src="down.png"
    class="image" name="btn_go_d" /></div>
<div id="ptz_plus"><input type="image" src="zoom_plus.png"
    class="image" name="btn_zoom_p" /></div>
<div id="ptz_minus"><input type="image" src="zoom_minus.png"
    class="image" name="btn_zoom_m" /></div>
```

Listato 3.1: Pulsanti di movimento e zoom, con relativi attributi

Il prefisso comune nell'attributo "*name*" dei pulsanti di movimento, e quello analogo dei pulsanti di zoom, permette di selezionare tutti i pulsanti tramite un unico selettore jQuery:

```
$().ready(function() {
  $('input[name^="btn_go_"], input[name^="btn_zoom_"]').hide();
}
```

Listato 3.2: Uso dei selettori per nascondere i pulsanti al caricamento della pagina

$().ready() è una funzione specifica che viene eseguita automaticamente al caricamento della pagina, largamente utilizzata per manipolare l'albero DOM o aggiungere gestori di eventi a vari elementi.

3.6.2 Ajax

Anche implementare le tecnologie AJAX diventa semplice tramite jQuery; in questo lavoro si è scelto di appoggiarsi, lato server, ad uno script PHP per la gestione della comunicazione via socket ØMQ (si veda 4.1 a pag. 23); a jQuery è affidata la richiesta AJAX allo script e la lettura della risposta.

```
function panoClick(e) {
  var offset = $('.panoImage').offset();
  var point_x = Math.floor(e.pageX - offset.left);
  var point_y = Math.floor(e.pageY - offset.top)
  $.ajax({
    type: "GET",
    url: "socket.php",
    data: "go=p&x="+point_x+"&y="+point_y,
    success: function(server_response) {
      $('#socket').text(server_response);
    }
  });
}
```

Listato 3.3: Funzione per l'invio del comando *GP*

Nel listato 3.3 viene mostrata la funzione che gestisce l'evento *click del mouse sull'immagine panoramica*: si calcolano le coordinate relative all'immagine del punto clickato, indipendenti dalla posizione dell'immagine stessa nella pagina e dalla porzione di pagina correntemente visualizzata nella finestra del browser; la richiesta AJAX, effettuata tramite metodo GET, consiste nell'invio allo script PHP del comando corrispondente all'azione intrapresa (nel caso specifico, *vai alla posizione* $\langle x, y \rangle$), con i relativi argomenti passati come parametri; la funzione definita in "*success*" permette di gestire la risposta del server alla richiesta stessa.

3.6.3 Visibilità degli elementi

Funzioni come *.show* e *.hide* consentono rispettivamente di far comparire o nascondere un elemento, come avviene per i controlli di movimento una volta che l'utente ha ottenuto (rispettivamente: perso) il controllo di Omnidome®. Le funzioni simili *.fadeIn* e *.fadeOut* aggiungono un effetto di comparsa (e scomparsa) graduale, che rende più fluida l'esperienza d'uso del *Web Control Center*.

3.6.4 Conto alla rovescia

Oltre alle funzioni standard di jQuery, si è aggiunto il plugin **countdown**, utilizzato per creare il conto alla rovescia del tempo di utilizzo rimanente (si veda il punto 3 a pag. 39); la struttura modulare di jQuery fa sì che una volta caricato il file js del plugin, le nuove funzioni possano essere utilizzate da jQuery come se fossero funzioni standard.

3.6.5 Aggiornamento dei frames

Il listato 3.4 mostra una funzione che è stato necessario introdurre per eliminare un problema di sfarfallio delle immagini che altrimenti si presenta con alcuni browser, come *Mozilla Firefox*; altri browser non sono invece affetti da questo problema.

```
function updatePano(date) {
  var img = new Image();
  img.onload = function() {
    $('.panoImage').attr("src", img.src);
  }
  img.src = "output/pano" + getNextPanoImg() + ".jpg?refresh="
      + date;
}
```

Listato 3.4: Funzione ausiliaria per l'aggiornamento delle immagini

La funzione *updatePano*, assieme all'analoga *updatePtz*, ha lo scopo di caricare l'immagine successiva, ottenuta tramite il mantenimento di un indice che viene aggiornato dalla funzione *getNextPanoImg* (e dall'analoga *getNextPtzImg*, per le immagini prospettiche); l'aggiunta del parametro "*refresh*" (il cui valore è pari al timestamp del momento in cui la funzione viene chiamata)

all'attributo "*src*" dell'immagine evita che il browser possa mostrare un'immagine in cache. Le immagini infatti sono salvate in una serie limitata di files numerati in sequenza, su cui vengono scritte ciclicamente: se si vanno a caricare normalmente, il browser rileva che il nome del file corrisponde ad un file già caricato e pertanto lo mostra dalla propria cache, invece che andarlo a scaricare nuovamente. Questo comportamento è corretto per il normale funzionamento del web, nell'ottica del riuso delle risorse e di evitare sprechi di banda, ma non è invece corretto per il nostro scenario di utilizzo. L'aggiunta di *refresh* forza il browser a caricare di nuovo il file, anche se il nome base non è cambiato.

Ciò consente di mantenere una sequenza limitata di immagini (ci si potrebbe limitare anche ad una sola), con risparmi in termini di utilizzo della memoria, pur garantendo che l'immagine mostrata sia sempre aggiornata.

Le funzioni *updatePano* e *updatePtz* vengono chiamate ad intervalli temporali regolari tramite un *setInterval* JavaScript: è possibile quindi cambiare la frequenza di aggiornamento delle immagini andando a cambiare questo intervallo, che si può calcolare tramite la seguente formula:

$$t = \frac{1000}{f}$$

dove f è la frequenza di aggiornamento desiderata (in frames al secondo).

3.6.6 Long polling

Il listato 3.5 mostra l'implementazione del *long polling* tramite jQuery:

```
function poll() {
  $.ajax({
    type: "GET",
    url: "socket.php",
    data: "poll=1",
    success: function(server_response) { ... },
    complete: poll,
    timeout: 30030
  });
}
```

Listato 3.5: Implementazione del long polling in jQuery

Rispetto a quanto già visto in precedenza, risaltano due nuovi elementi:

- "*complete*" contiene il nome della funzione da chiamare al completamento della richiesta corrente; il suo valore è posto uguale alla funzione *poll()* stessa, creando un ciclo infinito di chiamate;
- "*timeout*" è il tempo in millisecondi per cui si attende la risposta dal server, prima di chiudere la connessione e iniziare nuovamente il ciclo.

Nel listato 3.5 si è tralasciata l'implementazione della funzione "*success*", che verrà vista in dettaglio in § 6.2 a pag. 42.

3.6.7 Shared Memory

Per sfruttare in maniera semplice la Shared Memory si è creato uno *script bash*, da lanciare all'avvio del sistema operativo, che gestisce alcuni collegamenti simbolici tra il filesystem e il RAM-disk /dev/shm. Un esempio dello script è riportato nel listato 3.6:

```
#!/bin/bash

rm omniserver/build/output
rm public_html/output
mkdir /dev/shm/output
ln -s /dev/shm/output/ omniserver/build/output
ln -s /dev/shm/output/ public_html/output
```

Listato 3.6: Script bash per la gestione dei link simbolici

La directory *omniserver/build/output* è la directory all'interno della quale il software C++ scrive i frame man mano che vengono prodotti, con la caratteristica di ciclicità già vista in precedenza; *public_html/output* è invece la directory in cui il web-server si aspetta di trovare le immagini da mostrare nel *Web Control Center*; le ultime tre righe dello script realizzano la vera e propria inter-comunicazione tra i due software: viene creata una directory all'interno del RAM-disk (per una maggiore organizzazione), che viene successivamente collegata alle due directory già viste, che non sono quindi "reali" ma semplicemente dei collegamenti simbolici alla stessa area di memoria condivisa.

Capitolo 4

ØMQ

4.1 Introduzione

In un mondo sempre più inter-connesso, chi scrive software si trova ad affrontare una nuova sfida: creare un prodotto che da un lato rispecchi tutte le caratteristiche richieste, anche di sicurezza, ma che possa comunicare in un modo il più semplice ed efficace possibile con altri software.

Se pensiamo infatti all'odierna diffusione dei *social network* e al microcosmo di applicativi che fanno uso dei loro dati, o alla sempre maggiore diffusione di servizi *cloud-based*, risulta evidente che gli sviluppatori si trovano a dover gestire lo scambio di dati non solo tra software diversi, ma addirittura tra software scritti in linguaggi di programmazione diversi, che funzionano su sistemi operativi diversi. La comunicazione stessa può avvenire con schemi di volta in volta mutevoli, in cui variano il numero e il ruolo delle parti coinvolte.

Chiunque abbia sviluppato software di questo tipo conosce come ci siano stati diversi tentativi di risolvere la sfida del *software connesso*: HTTP, WebSockets, protocolli proprietari e protocolli peer-to-peer, per citarne alcuni. Ci si ritrova in una situazione vicina al paradosso: hardware sempre più collegati tra di loro tramite Internet, ma software che non riescono a sfruttare questa interconnessione.

ØMQ (scritto anche *ZeroMQ*, *0MQ*, *zmq*), sviluppato da iMatix Corporation, nasce con questi due principi:

- risolvere il problema di connettere i diversi software, in qualsiasi ambito

si stia lavorando;

- mantenere basso il livello generale di complessità, di modo che sia semplice integrare questa tecnologia nel proprio software.

I punti di forza di ØMQ sono l'utilizzo di un modello di I/O (input-output) *asincrono*, che consente di creare applicazioni multi-core altamente scalabili; la possibilità di utilizzare diversi livelli di trasporto, come *in-process*, *inter-process*, *TCP* e *multicast*; la disponibilità di diverse tipologie di socket che consentono di creare schemi di comunicazione anche complessi; API per molti linguaggi e sistemi operativi [10].

Un altro grande vantaggio nell'utilizzare ØMQ è la possibilità di non doversi preoccupare dei dettagli implementativi: la libreria gestisce automaticamente il threading per i propri socket, dei quali basta indicare la tipologia (ad esempio, ZMQ_REP o ZMQ_PUB che vedremo nel seguito) per avere automaticamente impostato il corretto schema di comunicazione.

4.2 Request e Reply

La prima tipologia di comunicazione tra i client e il server implementata è quella denominata **request-reply**, ottenuta utilizzando la coppia di socket REQ-REP.

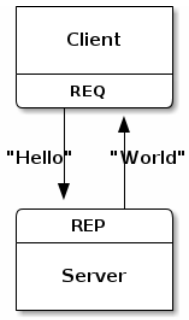

Figura 4.1: Schema di comunicazione Request-Reply

Questa coppia realizza la più comune forma di comunicazione, ovvero l'invio di una richiesta e l'attesa di una risposta. Nel nostro sistema, sfruttando il paradigma *N a 1* di ØMQ, ogni client utilizza un socket di tipo ZMQ_REQ che realizza la parte *request* della comunicazione. Tutti i socket client sono

collegati allo stesso socket ZMQ_REP, creato dal software C++ per gestire le richieste in arrivo.

Il collegamento *N a 1* avviene tramite il binding del socket ZMQ_REP e di tutti i socket ZMQ_REQ alla medesima porta TCP, che in questo lavoro è la 5555; l'accodamento e la distribuzione dei messaggi da e al corretto socket client sono affidati a ØMQ.

4.3 Publisher e Subscriber

L'altra tipologia di comunicazione implementata in questo lavoro è quella denominata **publisher-subscriber**, che rispetto alla precedente utilizza la coppia di socket PUB-SUB.

La modalità di comunicazione è di tipo *1 a N*: è il software C++ a pubblicare messaggi, con cadenza temporale variabile, tramite il proprio socket ZMQ_PUB; i client rimangono in attesa, tramite il proprio socket ZMQ_SUB, dell'arrivo di questi messaggi.

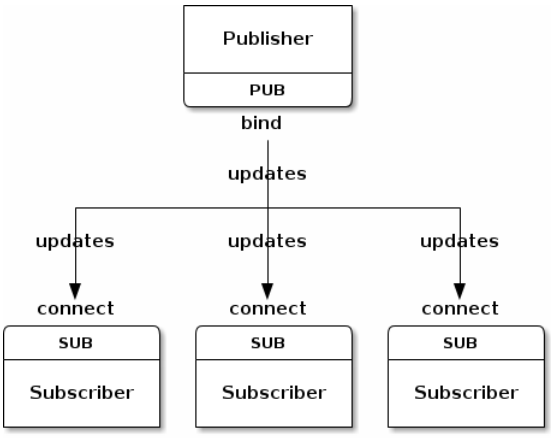

Figura 4.2: Schema di comunicazione Publisher-Subscriber

Non si è invece sfruttata la possibilità dei client subscriber di filtrare i messaggi in arrivo: si è preferito far arrivare tutti i messaggi ai client, per notificare le variazioni nella coda di utenti, come descritto in § 6.2 a pag. 42.

Anche in questa tipologia il collegamento *1 a N* avviene tramite binding del socket ZMQ_PUB e dei socket ZMQ_SUB alla medesima porta TCP; ri-

manendo coerenti con la scelta effettuata per la coppia request-reply si è qui scelta la porta 5556.

4.4 I messaggi

I messaggi scambiati in entrambe le coppie di socket sono stringhe di caratteri, il cui contenuto è di forma diversa a seconda della sorgente (e quindi della tipologia) del messaggio.

Occorre prestare attenzione a come il linguaggio di programmazione che si sta utilizzando tratta le stringhe: in C, ad esempio, le stringhe sono terminate da un byte NULL. Pertanto quando si invia, ad esempio, la stringa "Hello", stiamo in realtà inviando la stringa "Hello\0", che ha lunghezza pari a 6 e non 5; in altri linguaggi, come Python, le stringhe non sono terminate dal byte NULL: questo comporta che l'invio e la ricezione di stringhe tra linguaggi diversi può provocare comportamenti inaspettati e/o seri malfunzionamenti, se le due parti comunicanti non sono a conoscenza del reciproco formato delle stringhe.

In ØMQ si è scelto di fissare la seguente regola: **le stringhe sono specificate dalla loro lunghezza e vengono trasmesse senza il byte terminatore NULL**. Si sono pertanto utilizzate delle funzioni C++ di convenienza per trattare le stringhe che vengono scambiate dalle parti [11]:

```
std::string OmniServer::s_recv(zmq::socket_t &socket)
{
    zmq::message_t message;
    socket.recv(&message);

    return std::string(static_cast<char*>(message.data()),
        message.size());
}
```

Listato 4.1: Ricezione di una stringa ØMQ dal socket e conversione in stringa C++

```
bool OmniServer::s_send(zmq::socket_t &socket, const
    std::string &string)
{
    zmq::message_t message(string.size());
    memcpy(message.data(), string.data(), string.size());

    bool rc = socket.send(message);
    return (rc);
```

```
}
```

Listato 4.2: Conversione di una stringa C++ in stringa ØMQ e invio al socket

Per la parte software scritta in PHP non è stato necessario utilizzare funzioni di convenienza, dato che il linguaggio gestisce automaticamente i terminatori di stringa.

4.4.1 Messaggi REQ-REP

I messaggi inviati tramite i socket client ZMQ_REQ hanno una lunghezza fissa di 42 bytes e sono strutturati come segue:

Byte Iniziale	Byte Finale	Descrizione
00	31	Session ID
32	33	Comando
34	37	Primo argomento
38	41	Secondo argomento

il *Session ID* verrà descritto in dettaglio in § 6.2 a pag. 37. I possibili comandi sono i seguenti:

Comando	Descrizione
QB	Prenota l'utilizzo di Omnidome®
QD	Cancella la prenotazione
GP	Muovi alla posizione (x, y)
GU	Muovi verso l'alto
GD	Muovi verso il basso
GL	Muovi a sinistra
GR	Muovi a destra
ZP	Aumenta lo zoom
ZM	Diminuisci lo zoom

Gli argomenti sono pari a "0000" (quattro volte zero) per tutti i comandi ad esclusione di **GP**, nel quale contengono le coordinate x e y del punto clickato nell'immagine panoramica, con un eventuale padding di uno o più "0" (zero) per portare ogni argomento ad avere lunghezza pari a 4 caratteri.

Il software C++, tramite il socket ZMQ_REP, risponde ai comandi precedenti con una stringa vuota di *acknowledgement*, con la sola esclusione del comando **QB**, al quale risponde con una stringa di testo composta da tre valori separati con uno spazio:

- il primo è il valore di ritorno della chiamata al metodo *book*, descritto in dettaglio in § 6.3.1 a pag. 44;
- il secondo valore corrisponde al numero di elementi attualmente in attesa di poter utilizzare Omnidome®;
- il terzo valore è pari al tempo in secondi per cui viene reso possibile l'utilizzo di Omnidome®.

4.4.2 Messaggi PUB-SUB

I messaggi inviati dal software C++ tramite il socket ZMQ_PUB ai client ZMQ_SUB hanno struttura simile alla risposta generata nel caso precedente: si tratta sempre di tre valori delimitati dallo spazio, ma si caratterizzano come segue:

- il primo valore è un carattere fisso, pari a "*" (asterisco). Lo scopo di questo carattere è di fungere da *header* prestabilito per il filtraggio effettuato lato client ZMQ_SUB.

  L'utilizzo di (almeno) un carattere si è reso necessario poichè l'implementazione del sistema di filtraggio dei messaggi a livello di API non permette l'utilizzo di una stringa vuota come filtro, se lo si crea tramite linguaggio PHP;

- il secondo valore è pari a:
  - Session ID dell'elemento attualmente in testa alla coda, se la coda non è vuota;
  - "0" (zero), se la coda è vuota;
- il terzo valore è pari al tempo in secondi per cui viene reso possibile l'utilizzo di Omnidome®.

Poichè il tipo di comunicazione è unidirezionale, i client ZMQ_SUB non inviano alcuna risposta alla controparte ZMQ_PUB.

Capitolo 5

Image Processing

5.1 Introduzione

Abbiamo visto nel § 2.2.1 a pag. 5 che l'Omnidome® utilizza uno specchio iperbolico per la visione panoramica dell'ambiente circostante. L'immagine ripresa tramite questo tipo di specchio è sostanzialmente circolare, simile alla seguente:

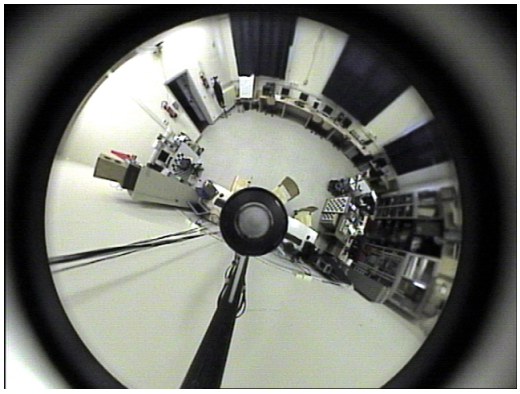

Figura 5.1: Immagine omnidirezionale

Questo tipo di immagine però è scarsamente utilizzabile ai fini di questo lavoro: risulta infatti difficile per un osservatore non abituato a trattare con le immagini panoramiche distinguere i dettagli della scena, specialmente per quanto riguarda la metà inferiore dell'immagine che risulta "capovolta" rispetto alla metà superiore.

Per questo motivo si implementano opportuni algoritmi per effettuare l'*unwarping* e l'*unwrapping* dell'immagine nel corrispondente *cilindro pano-*

ramico:

Figura 5.2: Immagine *unwrapped*

L'immagine omnidirezionale viene proiettata lungo la faccia interna di un cilindro (*unwarping*), che viene successivamente tagliato e "srotolato" (*unwrapping*). Il risultato è simile all'immagine precedente, dove si può riconoscere la stessa scena ripresa nella figura 5.1 in una disposizione sicuramente più agevole da analizzare per l'occhio umano. Ponendo il centro del sistema di riferimento nel centro dell'immagine omnidirezionale, l'*unwrapping* inizia dal raggio parallelo all'asse delle y negative e procede in senso anti-orario, come si può verificare osservando la disposizione degli oggetti nelle due immagini: la lavagna bianca, posta vicina all'asse delle y positive nell'immagine omnidirezionale, si trova circa a metà dell'immagine panoramica *unwrapped*. Bisogna sempre tenere presente che l'immagine è in realtà circolare: la scena che termina nel bordo di sinistra continua a partire dal bordo di destra, quindi i "due" muri visibili alle estremità dell'immagine panoramica *unwrapped* sono in realtà lo stesso muro, come si può verificare anche dal passacavi.

In questo lavoro si utilizzano le immagini panoramiche *unwrapped* per garantire una migliore esperienza di utilizzo da parte degli utenti, che possono navigare più agevolmente all'interno della scena tramite i movimenti da far eseguire a Omnidome® per inquadrare la zona interessata.

5.2 L'algoritmo di unwrapping

In letteratura esistono molti algoritmi di *unwrapping* (si veda ad esempio [12], [13] e [2]); per questo lavoro si è scelto di utilizzare un metodo alquanto efficiente fornito dalle librerie OpenCV che sfrutta le *look-up tables* e già implementato anche in [1].

L'algoritmo ottenuto effettua una conversione da coordinate immagine a coordinate polari, salvando il risultato in due matrici di valori *float*; queste matrici vengono poi utilizzate dalla funzione di OpenCV ***cvRemap*** per effettuare la vera e propria trasformazione geometrica dall'immagine panoramica all'immagine *unwrapped*. I listati che seguono riportano rispettivamente

il doppio ciclo *for* che popola le *look-up tables* in fase di inizializzazione e l'*unwrapping* vero e proprio:

```
// W, H: numero di colonne e righe delle matrici LUT
for (int i = 0; i < H; ++i)
{
  for (int j = 0; j < W; ++j)
  {
    // Calcolo delle coordinate polari corrispondenti al pixel
        di coordinate (i, j)
    theta = 2.0*M_PI*static_cast<float> (j) / W;
    // rMax, rMin: raggio esterno e raggio interno
        dell'immagine panoramica
    rho = rMax - i * (rMax - rMin) / H;

    // *dataX, *dataY: puntatori ai dati delle LUT
    *(dataX + i*W + j) = yCenter + rho * sin(theta);
    *(dataY + i*W + j) = xCenter + rho * cos(theta);
  }
}
```

Listato 5.1: Riempimento delle *look-up tables* dell'algoritmo di *unwrapping*

```
IplImage *Unwrapper::UnwrapImage(IplImage *OriginalImg)
{
  // Check sull'inizializzazione delle LUT. Altrimenti, non
      faccio niente e ritorno il puntatore inizializzato a 0
  if (LUTX)
  {
    cvRemap(OriginalImg, UnwrappedImg, LUTX, LUTY, RemapMode,
        cvScalarAll(0));
    return UnwrappedImg;
  }
  else
    return 0;
}
```

Listato 5.2: Implementazione C++ dell'algoritmo di *unwrapping* tramite OpenCV

RemapMode è un intero, che contiene il valore del *flag* corrispondente al metodo di interpolazione utilizzato per effettuare la trasformazione geometrica. Possibili valori di questo *flag* sono:

CV_INTER_NN: utilizza il metodo *Nearest Neighbour*, che seleziona il valore del pixel più vicino, senza considerare i valori degli altri pixel vicini;

CV_INTER_LINEAR: il metodo utilizzato è la *Bi-linear Interpolation*, che considera un quadrato di 2×2 pixel noti attorno al pixel da valutare, calcolandone la media pesata per ottenere il valore interpolato finale;

CV_INTER_CUBIC: simile al metodo precedente, la *Bi-cubic Interpolation* utilizza un quadrato di 4×4 pixel, che garantisce immagini più lisce e prive di artefatti a scapito però della velocità di calcolo.

Per gli scopi di questo lavoro si è scelto di utilizzare l'interpolazione bilineare (CV_INTER_LINEAR), che offre il miglior compromesso tra qualità dell'immagine risultante e velocità di calcolo.

5.3 Frames I/O

5.3.1 Acquisizione dei frames

L'acquisizione dei frames da parte del software C++ avviene tramite la classe *FrameGrabberDevice*, che utilizza le API **Video4Linux 2** (si veda [3] per la documentazione dettagliata) per inviare dei comandi diretti all'hardware video, mediante l'uso della funzione *ioctl*. Attraverso questa classe i frames vengono acquisiti via *Memory Mapping*, un metodo di streaming input/output in cui vengono scambiati solamente i puntatori ai buffer temporanei tra l'applicazione e il driver, senza andare quindi a copiare il dato in sè. Il *Memory Mapping* è utilizzato principalmente per mettere in corrispondenza i buffer nella memoria del dispositivo (che può essere la memoria di una scheda video, o una scheda di acquizione) con lo spazio di indirizzamento dell'applicazione: il metodo, nel nostro caso, consente al software C++ di avere accesso diretto ai frames acquisiti, frames che vengono memorizzati temporaneamente all'interno della memoria delle schede di acquisizione video e successivamente scartati.

Tramite la classe *FrameGrabberDevice* si acquisiscono le immagini da entrambe le telecamere, lasciando poi all'algoritmo di *unwrapping* del § 5.2 il compito di elaborare l'immagine panoramica.

5.3.2 Salvataggio dei frames

Una volta che le immagini sono state acquisite e che l'immagine panoramica è stata trasformata nella corrispondente immagine *unwrapped*, si rende ne-

cessario trasferire queste immagini al web-server in modo che possano essere visualizzate all'interno del *Web Control Center* di Omnidome®.

Si utilizza allo scopo la *Shared Memory* già vista in § 3.5 a pag. 16, andando a salvare in una directory "output" i frames appena letti tramite le funzioni standard di OpenCV:

```
std::sprintf(fn_pano, "%s%d.jpg", "output/pano", i);
std::sprintf(fn_ptz, "%s%d.jpg", "output/ptz", i);
i = ++i % DUMPED_FRAMES;
cvSaveImage(fn_pano, panoImg, JPEG_PARAMS);
cvSaveImage(fn_ptz, ptzImg, JPEG_PARAMS);
```

Listato 5.3: Salvataggio dei frames su buffer circolare

L'utilizzo del contatore i permette di creare un buffer circolare di immagini: rimangono salvati solamente DUMPED_FRAMES frames totali per tipo; ciò consente sia una migliore gestione della memoria RAM, che non viene riempita da frames "inutili" (poichè troppo vecchi), sia agevola la scrittura delle funzioni jQuery *updatePano* e *updatePtz* già viste nel § 3.6.5 a pag. 19. Si è preferito adottare questa soluzione, rispetto a quella molto più comune di utilizzare un'unica immagine da sovrascrivere ad intervalli regolari, per limitare eventuali sovrapposizioni tra scrittura del frame e sua contemporanea lettura, le quali possono causare interruzioni premature nel download dell'immagine.

Il valore di DUMPED_FRAMES va valutato in base alle caratteristiche delle telecamere: per questo lavoro si è scelto un valore pari a 1,5 volte il *framerate* di acquisizione impostato per le stesse.

L'argomento JPEG_PARAMS della funzione *cvSaveImage* è un array di interi che consente di andare a modificare la qualità delle immagini JPEG salvate. Si è impostato questo livello al 50%, buon compromesso tra qualità e dimensione dei file, che risultano essere di circa 13KB (immagine panoramica *unwrapped*) e 20KB (immagine prospettica). Le dimensioni dei file, assieme alla frequenza di aggiornamento delle immagini nel *Web Control Center* (si veda il già citato § 3.6.5 a pag. 19), sono parametri che vanno valutati attentamente in base alle caratteristiche dell'utenza a cui si intende rivolgere il risultato di questo lavoro: file troppo pesanti o una frequnza di aggiornamento troppo elevata possono causare saturazioni di banda per quegli utenti, specialmente *mobile*, che non hanno a disposizione linee a larghissima banda.

Capitolo 6

Il sistema di prenotazione

6.1 Il problema

Una delle problematiche più importanti che si è cercato di risolvere con questo lavoro è la gestione dell'accesso a Omnidome® da parte degli utenti.

Le caratteristiche del sistema sin qui descritto ne consentono infatti la fruizione contemporanea da parte di più utenti, che possono accedere al *Web Control Center* e visualizzare le immagini trasmesse da Omnidome®; la visualizzazione anche contemporanea delle immagini non crea alcun tipo di problema mentre, anticipando brevemente l'argomento del prossimo capitolo ovvero la possibilità di controllare la telecamera prospettica mobile tramite *Web Control Center*, ci si rende immediatamente conto di come questo accesso contemporaneo sia fonte di possibili problemi: se tutti gli utenti collegati possono far muovere la telecamera nello stesso istante, il risultato che si ottiene è una pessima esperienza d'uso da parte degli utenti stessi.

Si è pertanto stabilito che solo un utente alla volta possa controllare Omnidome®, per un tempo preimpostato; da questo ne è derivata la necessità di creare un sistema di prenotazione e di gestione della coda di utenti che fosse il più possibile integrato con il resto del software visto finora.

Il sistema di prenotazione implementato è composto da due elementi principali:

- l'interfaccia utente, costituente il *Web Control Center* di Omnidome®, tramite la quale gli utenti possono prenotarsi e conoscere lo stato della propria prenotazione;

- il software C++ che gestisce la coda di utenti.

La comunicazione tra le parti è affidata a ØMQ, come visto in § 4.4.1 a pag. 27.

Di seguito si descriveranno in dettaglio le due componenti, mostrando come avvenga l'interazione tra le stesse e descrivendo alcune scelte implementative che influiscono sulla fruizione del *Web Control Center*.

6.2 L'interfaccia utente

Nell'ottica di mantenere ad un basso livello di difficoltà la prenotazione e il successivo controllo di Omnidome®, si sono assunti i seguenti punti:

1. la prenotazione non deve essere automatica: l'utente che arriva sul *Web Control Center* può anche solo voler visualizzare le immagini live di Omnidome®;

2. gli utenti devono essere identificabili tramite un codice univoco per poterli distinguere all'interno della coda;

3. i pulsanti che consentono il movimento della telecamera prospettica devono essere visibili solamente se l'utente ha il controllo di Omnidome®: l'utente altrimenti potrebbe essere indotto a pensare di non doversi prenotare per controllare il brandeggio e quindi potrebbe cercare di clickare sui pulsanti, rimanendo disorientato dalla mancanza di risposta del sistema;

4. l'utente, una volta effettuata la prenotazione, deve ricevere un *feedback* anche visivo riguardo lo stato della prenotazione;

5. la modalità di avanzamento della coda deve essere automatica: non deve pertanto essere l'utente a verificare il proprio stato all'interno della coda, ma deve essere il sistema stesso ad informare gli utenti riguardo l'ottenimento del controllo di Omnidome® o il perdurare dello stato di attesa;

6. deve essere previsto un meccanismo di uscita dalla coda per quegli utenti che seppur prenotati chiudono il *Web Control Center*, in modo da non far attendere inutilmente gli utenti successivi.

La realizzazione pratica del punto 3 è già stata trattata in § 3.6.3 a pag. 19: si utilizzano le funzioni *.fadeIn* e *.fadeOut* di jQuery per far comparire i pulsanti di movimento non appena l'utente ottiene il controllo di Omnidome®. I pulsanti vengono creati al caricamento della pagina, ma subito nascosti tramite la funzione *.hide*: in questo modo si velocizza il successivo apparire degli stessi, che non devono essere creati "al momento" ma semplicemente resi visibili.

L'utente quindi vede i pulsanti di movimento solamente per il periodo di tempo in cui può controllare Omnidome®. Allo stesso modo, viene abilitata o meno la gestione del click sull'immagine panoramica, gestione che avviene tramite il codice già visto nel listato 3.3 a pag. 18.

Per quanto riguarda il punto 2, l'identificazione dell'utente avviene utilizzando le funzioni di sessione di PHP [4]. Oltre a creare un vettore "*superglobal*" (ovvero accessibile a tutti gli script) in cui è possibile registrare informazioni che rimangono persistenti per tutta la sessione di lavoro all'interno delle pagine PHP, viene assegnato ad ogni utente un identificativo alfanumerico univoco. Nel listato seguente viene mostrata l'inizializzazione e la memorizzazione di questo identificativo da parte dello script PHP:

```
//session id
static $SID;

//init session, se non lo è già
if (!session_id()) {
  session_start();
  $SID = session_id();
}

//necessario per non avere richieste pendenti
session_write_close();
```

Listato 6.1: Inizializzazione del *session_id*

L'utilizzo della variabile statica SID assicura il mantenimento del *session_id* anche in chiamate successive dello script.

Si è inoltre reso necessario aggiungere al codice una chiamata alla funzione *session_write_close*, che appena invocata termina l'accesso all'array superglobal di sessione, evitando così che lo script PHP rimanga in esecuzione e consentendo quindi che ritorni immediatamente: si eliminano così eventuali

problemi nella ricezione delle notifiche, che non possono essere consegnate se un'istanza precedente dello script PHP rimane pendente.

A partire dalla versione 5 del PHP, è possibile scegliere l'algoritmo con il quale calcolare il *session_id*. Nel file di configurazione di PHP, si sono impostati i seguenti valori:

```
; Select a hash function for use in generating session ids.
; Possible Values
;   0  (MD5 128 bits)
;   1  (SHA-1 160 bits)
; This option may also be set to the name of any hash function
    supported by
; the hash extension. A list of available hashes is returned
    by the hash_algos()
; function.
; http://php.net/session.hash-function
session.hash_function = 1

; Define how many bits are stored in each character when
    converting
; the binary hash data to something readable.
; Possible values:
;   4  (4 bits: 0-9, a-f)
;   5  (5 bits: 0-9, a-v)
;   6  (6 bits: 0-9, a-z, A-Z, "-", ",")
; Default Value: 4
; Development Value: 5
; Production Value: 5
; http://php.net/session.hash-bits-per-character
session.hash_bits_per_character = 5
```

Listato 6.2: Impostazioni file *php.ini*

Con questa configurazione, il *session_id* generato ha una lunghezza pari a 32 byte. La combinazione dell'algoritmo SHA-1 e dell'utilizzo di 5 bit per carattere garantisce un buon compromesso tra lunghezza della chiave generata e livello di univocità dei *session_id*; non si sono sfruttate le nuove funzionalità introdotte con la versione 5.3.0 di PHP, che consentono di utilizzare funzioni di *hashing* più complesse (ad esempio SHA512), che andrebbero a generare *session_id* di lunghezza molto maggiore.

I punti 1 e 4-6 sono invece gestiti tramite un *event-handler* aggiunto al pulsante di prenotazione e tramite una funzione *poll* che viene richiamata ciclicamente, secondo i dettami del *long polling* visto in § 3.3 a pag. 14. Nello specifico, il *workflow* è il seguente:

1. l'utente preme il pulsante "prenota", sul quale è stato caricato un *handler* del click tramite jQuery;

2. l'handler del click chiama tramite AJAX lo script PHP, con dei parametri appositi che consentono di inviare tramite socket ZMQ_REQ (si veda § 4.2 a pag. 24) il comando di prenotazione;

3. lo script PHP ritorna una delle due possibili risposte:

   (a) l'utente può subito controllare Omnidome®: si mostrano i pulsanti di movimento, si abilita l'handler del click sull'immagine panoramica, si mostra e si fa partire il conto alla rovescia, che offre un'indicazione visiva del tempo ancora a disposizione per il controllo di Omnidome®; si sostituisce al pulsante "prenota" un pulsante (disabilitato) che reca l'indicazione "in uso";

   (b) l'utente deve attendere: un popup informa l'utente di quale sia il tempo massimo di attesa, calcolato sul numero di utenti che già erano in coda; non viene mostrato alcun conto alla rovescia; si sostituisce al pulsante "prenota" un pulsante (disabilitato) che reca l'indicazione "in attesa";

4. viene fatta partire la funzione di *long polling* per ricevere le notifiche sull'avanzamento della coda.

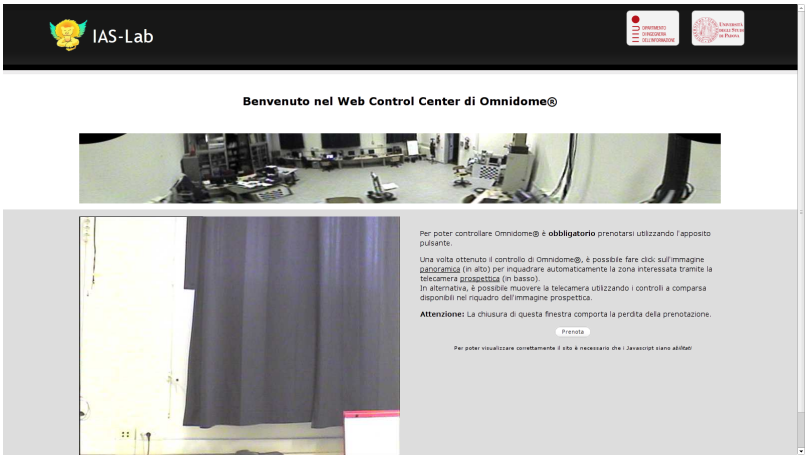

Figura 6.1: Web Control Center: solo visualizzazione

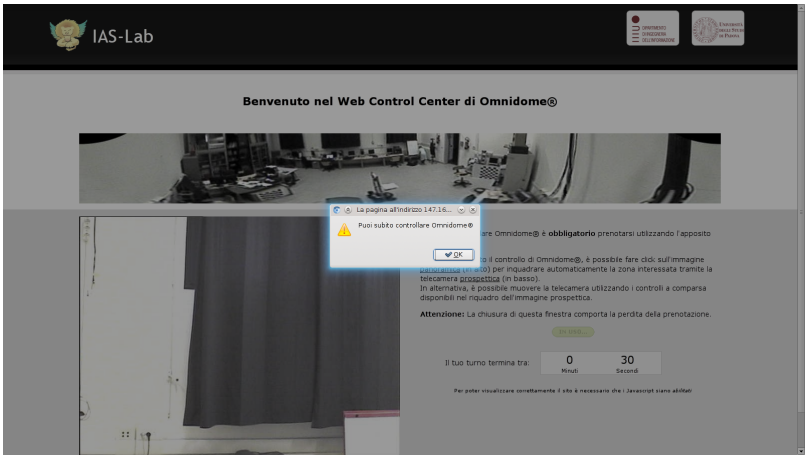

Figura 6.2: Web Control Center: prenotazione e controllo immediato

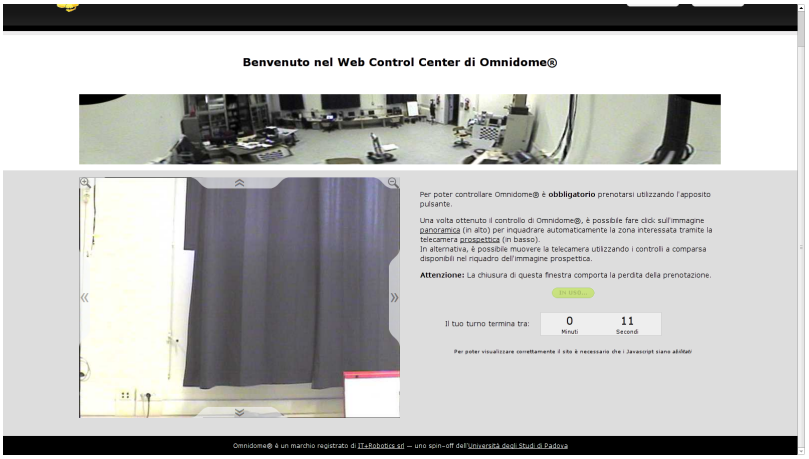

Figura 6.3: Web Control Center: in controllo, pulsanti di movimento abilitati

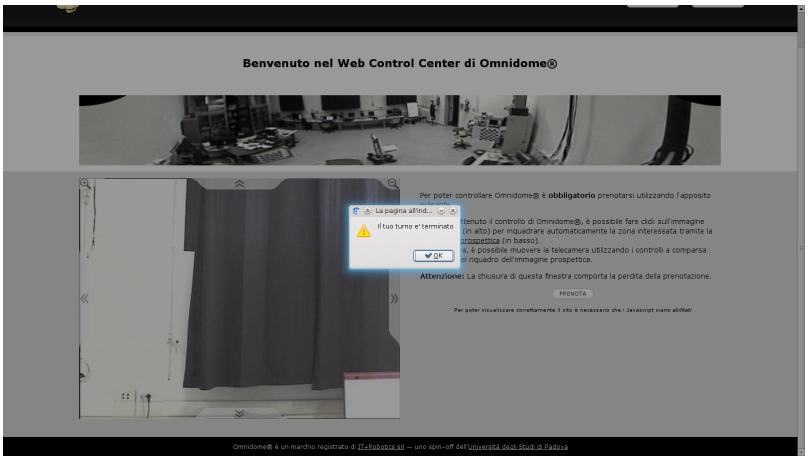

Figura 6.4: Web Control Center: termine del turno di controllo

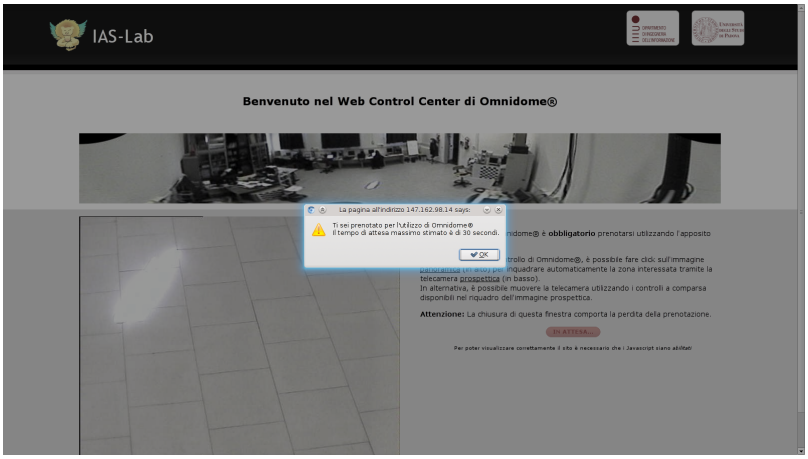

Figura 6.5: Web Control Center: prenotazione e attesa con tempo massimo stimato

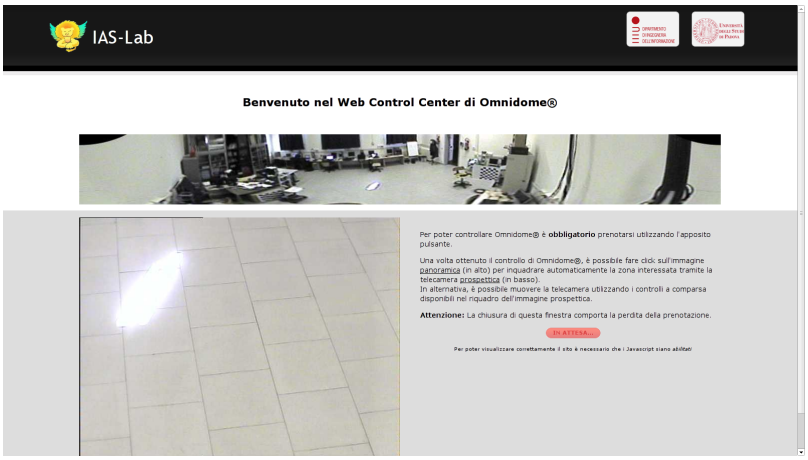

Figura 6.6: Web Control Center: attesa del turno di controllo

Per quanto riguarda il punto 3-b è chiaro perchè si parli di *tempo massimo di attesa* e non venga mostrato il conto alla rovescia: si è implementata una funzione di uscita dalla coda per quegli utenti che "vanno via" dal *Web Control Center*, conseguentemente il tempo di attesa può essere inferiore rispetto a quanto calcolato in origine.

Questo workflow è ben visibile nelle figure 6.1 - 6.6, che raccolgono le varie fasi della prenotazione da parte di due utenti, mostrando quindi sia la visualizzazione della pagina per l'utente che ha il controllo di Omnidome® sia la pagina per l'utente in attesa.

La procedura sin qui descritta è valida per il primo click effettuato sul pulsante "prenota": in seguito la gestione è affidata alla funzione di *long polling* del punto 4. In dettaglio la funzione esegue i punti seguenti:

1. richiama lo script PHP con dei parametri appositi per monitorare, tramite socket ZMQ_SUB (si veda § 4.3 a pag. 25), le notifiche dell'avanzamento della coda; si sfruttano gli argomenti *complete* e *timeout* della funzione *$.ajax* già visti nel listato 3.5 a pag. 20;

2. quando è disponibile un aggiornamento della coda, legge la risposta dallo script PHP; la risposta contiene l'indicazione se l'utente corrente sia l'utente attualmente abilitato ad utilizzare Omnidome®:

   (a) in caso affermativo, si eseguono le stesse operazioni del precedente punto 3-a;

   (b) in caso negativo, occorre distinguere due ulteriori casi:

      i. l'utente ha appena terminato di controllare Omnidome®: tramite popup viene informato del termine del periodo a disposizione, vengono nascosti i controlli e disabilitato l'handler del click sull'immagine panoramica;

      ii. l'utente non ha ancora raggiunto la testa della coda: rimane il messaggio di attesa e tutti i controlli sono disabilitati.

Si è implementata poi una funzione specifica per rimuovere l'utente dalla coda, quando abbandona il *Web Control Center*. Ciò si è reso necessario per evitare inutili attese agli utenti successivi e tempi morti nell'utilizzo di Omnidome®: lo *scheduler* all'interno del software C++ infatti assegnerebbe comunque il tempo di utilizzo al *session_id* dell'utente, anche se si è disconnesso dalla pagina; ne risulterebbe un periodo in cui Omnidome® risulta "in utilizzo", ma nessuno lo sta effettivamente utilizzando.

Per realizzare la funzione di rimozione si possono pensare due diverse strade:

- il software C++ contatta tutti i client in coda (identificati dai loro *session_id*) per vedere se sono ancora connessi o meno;

- l'utente, tramite lo script PHP, informa il software C++ che intende lasciare il *Web Control Center*.

Nel primo caso si sarebbe dovuto implementare un meccanismo per verificare la presenza dei client, tramite *ping* periodici o sfruttando la già vista tecnica del *long polling*, anche se questa volta a parti invertite. Si è però preferito scegliere la seconda modalità, sfruttando l'evento JavaScript *window.onbeforeunload*, che permette di eseguire una funzione quando si esce dalla pagina corrente.

Con "uscita dalla pagina corrente" si intendono sia la chiusura della finestra del browser (o della scheda, per i browser più moderni) sia il caricamento all'interno della stessa finestra (o scheda) di un indirizzo diverso rispetto a quello di partenza, sia tramite l'inserimento diretto nella barra degli indirizzi sia tramite il click su un tag HTML <a>. Poichè nella struttura del *Web Control Center* di Omnidome® alcuni pulsanti sono realizzati tramite tag <a>, si è reso necessario fare in modo che questi venissero ignorati dalla nostra funzione, altrimenti la loro pressione avrebbe comportato la cancellazione della prenotazione. Si è utilizzata pertanto una variabile di tipo *boolean*, che inserita in un costrutto *if* permette di controllare le condizioni in cui la funzione d'uscita va chiamata o meno.

6.3 La gestione della coda

La gestione vera e propria della coda di utenti è affidata al software C++; la struttura dati che si è scelto di utilizzare per memorizzare gli utenti è la **deque** (contrazione di *double-ended queue*), nel caso specifico di stringhe (*std::deque<std::string> q*) [5].

La scelta della deque, rispetto alla più nota struttura dati "vector", è stata fatta in base alla maggior efficienza della prima nell'inserimento e cancellazione degli elementi in testa alla deque stessa; altre caratteristiche peculiari di questa struttura dati (alcune rintracciabili anche in vector) sono:

- accesso ai singoli elementi della deque in base alla loro posizione;
- possibilità di scorrere gli elementi in qualsiasi ordine;
- inserimento e cancellazione degli elementi sia in testa che in coda.

Per quanto riguarda le necessità di questo lavoro, l'inserimento di un elemento nella deque, corrispondente ad un utente che si sta prenotando per

l'utilizzo di Omnidome®, viene effettuato in coda; l'utente in testa è quello che attualmente ha il controllo e l'avanzamento della deque, corrispondente al termine del periodo di utilizzo da parte di un utente e il passaggio al successivo, avviene tramite cancellazione della testa della deque. L'uso di un *iteratore* permette inoltre di verificare la presenza di un elemento all'interno della deque o cancellare un elemento in qualsiasi posizione, realizzando così la fuoriuscita dalla deque di un utente che lascia il *Web Control Center*.

6.3.1 Prenotazione

Il metodo che gestisce la prenotazione è il metodo *book*, che viene chiamato passandogli come unico argomento il *session_id* dell'utente:

```
int OmniServer::book(std::string sessionid)
{
  int retval = 0;
  // coda vuota
  if (q.empty())
  {
    q.push_back(sessionid);
    retval = 1; // coda vuota, prenota e utilizza da subito
  }
  // coda non vuota, sessionid non presente
  else if (find(q.begin(), q.end(), sessionid) == q.end())
  {
    q.push_back(sessionid);
    retval = 2; // coda non vuota, prenota e aspetta
  }
  // coda non vuota, sessionid presente, front
  else if (sessionid.compare(q.front()) == 0)
  {
    currentOwner = true;
    retval = 3; // prenotato, sessionid è già l'utilizzatore
  }
  // coda non vuota, sessionid presente, non front
  else
  {
    currentOwner = false;
    retval = 4; // prenotato, sessionid è già in attesa
  }
  if (!q.empty())
    sem_post(&queue_sem);
  return retval;
}
```

Listato 6.3: Implementazione C++ del metodo *book*

Il metodo *book* consente di distinguere 4 possibili casi:

1. la deque è vuota, il *session_ id* in argomento può quindi iniziare da subito a controllare Omnidome®;
2. la deque non è vuota e il *session_ id* in argomento non è noto: viene aggiunto alla coda della deque e messo in attesa;
3. la deque non è vuota, il *session_ id* in argomento è noto ed è in testa alla deque: l'utente corrispondente sta già controllando Omnidome®;
4. la deque non è vuota, il *session_ id* in argomento è noto ma non è in testa alla deque: l'utente corrispondente è già in attesa.

Nell'implementazione pratica i casi 3 e 4 non sono mai raggiunti: la tecnica del *long polling* consente infatti di evitare chiamate periodiche a questo metodo, ed è evidente come solo chiamate successive alla prima possano raggiungere questi casi; tuttavia si è preferito mantenerli nel codice, in un'ottica di pluralità di controlli che garantiscano maggior robustezza a tutto il software nel complesso.

Si potrebbe infatti pensare di manipolare le funzioni JavaScript del *Web Control Center*, tramite l'uso di particolari *console* utilizzate normalmente dagli sviluppatori in fase di debug, aggirando quindi i controlli effettuati dalla pagina stessa: anche se ciò accadesse, non sarebbe però possibile evitare gli ulteriori controlli effettuati dal software C++, rendendo vano qualsiasi tentativo di attacco.

Il metodo *book* infine fa scattare il semaforo posto sulla deque, il cui utilizzo vedremo successivamente descrivendo lo *scheduler*, e ritorna al ciclo principale lo stato della prenotazione, stato che viene poi comunicato tramite socket ZMQ_REP (si veda il § 4.2 a pag. 24) al client corrispondente.

6.3.2 Avanzamento

Come accennato in precedenza, il compito di far avanzare la deque di utenti man mano che questi utilizzano per il tempo prestabilito Omnidome® è affidato a un apposito thread del software C++ denominato *scheduler* la cui implementazione, molto semplice seppur altamente efficace per gli scopi di questo lavoro, è la seguente:

```
void *scheduler(void *omniserver)
{
  OmniServer *omni = (OmniServer *) omniserver;
  while(1)
  {
    sem_wait(&omni->queue_sem);
    pthread_mutex_lock(&omni->queue_mutex);
    if (!omni->q.empty())
    {
      // effettivo utilizzo dell'Omnidome®
      sleep(OMNI_TIMER);
      omni->q.pop_front();
    }
    pthread_mutex_unlock(&omni->queue_mutex);
    sem_post(&omni->zmq_sem);
  }
}
```

Listato 6.4: Implementazione C++ del thread *scheduler*

Il thread esegue un ciclo infinito, mettendosi in attesa sul semaforo della deque. Quando riceve il segnale che sblocca il semaforo, verifica che la deque non sia vuota e si addormenta per il tempo OMNI_TIMER, corrispondente al tempo concesso all'utente per il controllo di Omnidome®. Data la natura di OMNI_TIMER, che non richiede particolare precisione (in questo lavoro si assume infatti pari a 120 secondi), si è scelto di mantenere l'utilizzo dello *sleep* in secondi, invece di ricorrere a funzioni più precise (come può essere *usleep*).

Una volta terminato il tempo di utilizzo di Omnidome®, lo scheduler effettua la cancellazione della testa della deque e, tramite il *sem_post* al semaforo del thread che gestisce il socket ZMQ_PUB (si veda § 4.3 a pag. 25), segnala a quest'ultimo di inviare le notifiche dell'avvenuto cambio di utilizzatore a tutti i client in ascolto. Le notifiche vengono inviate come messaggi PUB-SUB, già trattati in § 4.4.2 a pag. 28.

6.3.3 Uscita

L'uscita dalla deque, che avviene in corrispondenza della chiusura della finestra (o della scheda) del browser, viene gestita dal software C++ come un normale comando; l'implementazione è immediata, grazie all'uso dell'*iteratore* già descritto in precedenza:

```
std::deque<std::string>::iterator where =
    find(omni->q.begin(), omni->q.end(), sid);
if ((omni->q.size() > 1) && (where != omni->q.end()))
{
  omni->q.erase(where);
}
```

Listato 6.5: Implementazione C++ del comando *QD*

L'*iteratore* viene inizializzato con valore pari all'indice numerico ritornato dall'algoritmo *find*, che ricerca il *session_id* (salvato nella variabile SID) scorrendo la deque dalla testa alla coda; l'implementazione della deque identifica con *.begin()* la testa e con *.end()* il valore successivo alla coda.

In questo modo è possibile verificare la presenza di un elemento all'interno della deque verificando che il valore dell'*iteratore* sia diverso da *.end()*, come si può notare nelle condizioni dell'*if*; l'ulteriore condizione sulla dimensione della deque, ovvero che contenga almeno 2 elementi, evita di far svuotare la deque nel caso particolare in cui vi è un solo utente in coda, il quale si disconnette prima del termine del periodo di tempo assegnato: ciò comporterebbe infatti lo svuotamento della deque con conseguente errore nella chiamata *omni->q.pop_front()* vista nel listato 6.4 a pag. 46.

Capitolo 7

Il sistema di puntamento

In questo capitolo verrà trattato l'altro aspetto cruciale di questo lavoro, il sistema di puntamento di Omnidome®, descrivendo in particolare come avvenga la trasmissione del comando di movimento dal *Web Control Center* all'hardware del brandeggio.

Gli utenti, una volta ottenuto il controllo dell'unità, si trovano davanti ad una modalità di invio dei comandi quantomeno familiare: frecce indicanti le quattro direzioni principali e pulsanti di zoom, senza dimenticare la più moderna modalità "click and go", attraverso la quale è possibile clickare in un punto specifico dell'immagine panoramica e vedere la telecamera prospettica muoversi sino ad inquadrare la zona richiesta.

Come si è già visto nel § 2.3.3 a pag. 9, il brandeggio utilizzato in questo lavoro manca però di alcune caratteristiche fondamentali per la realizzazione immediata dei comandi riportati sopra: si rende perciò necessario implementare dei metodi che consentano il movimento "in posizione" o il movimento "incrementale".

Questo porta ad un doppio risultato: da un lato la possibilità di realizzare dei movimenti che corrispondano ai comandi più naturali, dall'altro l'indipendenza tra le due componenti (*Web Control Center* e brandeggio), poichè con il tipo di implementazione che mostreremo esse andranno a condividere un protocollo comune di comandi (di movimento e di zoom), che però vengono realizzati da ciascuna in modo autonomo. Ciò permette ad esempio di sostituire il brandeggio di Omnidome® dovendo solamente implementare all'interno del software C++ i comandi richiesti dal protocollo, senza dover

riscrivere anche l'intero codice del *Web Control Center*.

7.1 Movimenti in posizione

Per "movimento in posizione" si intende la possibilità di indicare al brandeggio una posizione finale da raggiungere e che questi si muova alla posizione desiderata senza ulteriori indicazioni e/o interventi. Questo comporta, da parte del firmware di controllo del brandeggio, la conoscenza istante per istante della posizione dello stesso, in modo da calcolare gli spostamenti richiesti per raggiungere la nuova configurazione.

Come abbiamo già accennato, il brandeggio a disposizione per questo lavoro non consente di essere comandato "in posizione", né tantomeno lo consente il protocollo MACRO visto in § 2.3.3 a pag. 9; si sono tuttavia sfruttate le soluzioni implementate in precedenti lavori (si veda ad esempio il già citato [1]) che realizzano il movimento "in posizione" tramite calcolo del tempo necessario ad effettuare il movimento stesso: la velocità di rotazione dell'unità è infatti costante, permettendo di calcolare direttamente il tempo necessario a compiere una determinata variazione d'angolo.

Nel listato 7.1 viene mostrato il metodo della classe *CameraMotion* che realizza il movimento in posizione del brandeggio, in questo esempio riferito al solo PAN.

```
void CameraMotion::ChangeHPosition(void)
{
  float Hdelta = nextHPosition - curHPosition;
  useconds_t motionTime = static_cast<useconds_t>
      (fabsf(Hdelta) / PTHeadHSpeeds[SpeedIdx] * 1000000);

  if (motionTime > 0)
  {
    if (Hdelta < 0)
      PTHead.Left(PTHeadHSpeedOrder[SpeedIdx]);
    else
      PTHead.Right(PTHeadHSpeedOrder[SpeedIdx]);
    usleep(motionTime);
    PTHead.Stop();
  }
  float realMotion = static_cast<float> (motionTime) *
      PTHeadHSpeeds[SpeedIdx] / 1000000.0;
  if (Hdelta < 0.0)
    realMotion *= -1.0;
  curHPosition += realMotion;
```

```
  return;
}
```

Listato 7.1: Implementazione C++ del movimento in posizione

Si calcola la quantità di microsecondi necessaria al brandeggio per coprire la differenza di angolo data da *Hdelta* e lo si fa muovere nella direzione corretta per un tempo pari a questa quantità; successivamente si salva in *curHPosition* la posizione corrente del brandeggio, utile per il calcolo di *Hdelta*.

Il valore di *nextHPosition* è invece pari all'angolo (assoluto) di destinazione; vanno tenuti in considerazione eventuali limiti imposti dal brandeggio: in quello qui utilizzato l'escursione orizzontale è pari a 180°, il che comporta l'imposizione di limiti per l'angolo in ingresso, come evidenziato nel seguente listato.

```
 void CameraMotion::HSetPoint(float angle)
{
  pthread_mutex_lock(&mutex);
  if (!nowOperating)
  {
    if (angle <= 90.0 && angle >= -90.0 && fabs(angle -
        curHPosition) >= MIN_ANGLE_DIFFERENCE)
    {
      nextHPosition = angle;
    }
  }
  pthread_mutex_unlock(&mutex);
  return;
}
```

Listato 7.2: Imposizione dei limiti nell'escursione orizzontale

Oltre a valutare se l'angolo sia all'interno dell'escursione consentita, si valuta anche la differenza di movimento: con questo si vogliono evitare movimenti talmente piccoli da risultare visivamente quasi insignificanti. Per questo lavoro si è impostato il valore MIN_ANGLE_DIFFERENCE pari a 1°.

Questo tipo di movimento corrisponde alla modalità che abbiamo precedentemente definito "click and go": l'utente definisce la posizione che il brandeggio deve raggiungere non tanto tramite inserimento diretto dei valori di PAN e TILT, quanto piuttosto tramite il click su un punto dell'immagine panoramica. Nei listati precedenti si sono espressi i concetti di "posizione corrente" e "posizione da raggiungere" tramite angoli; rimane da mostrare co-

me venga effettuata la conversione tra coordinate di un punto dell'immagine panoramica e angolo corrispondente.

Occorre quindi definire i due sistemi di riferimento, rispettivamente dell'immagine panoramica e del brandeggio, e definire una funzione di trasformazione delle coordinate.

Coordinate dell'immagine panoramica

Il sistema di riferimento è quello di OpenCV, che assegna al pixel in alto a sinistra dell'immagine l'origine (0, 0), e al pixel in basso a destra il valore $(w-1, h-1)$, dove si indicano con w e h rispettivamente la larghezza (*width*) e l'altezza (*height*) dell'immagine. Ogni incremento unitario nelle coordinate indica pertanto uno dei pixel adiacenti.

Il centro dell'immagine panoramica risulta così corrispondere alle coordinate $(\lfloor \frac{w-1}{2} \rfloor, \lfloor \frac{h-1}{2} \rfloor)$.

Coordinate del brandeggio

Il sistema di riferimento del brandeggio prende come punto di origine (0°, 0°) la posizione dello stesso dopo l'esecuzione del comando *GoToHighLimit*, ovvero il movimento al *preset* n° 50 (si veda § 2.3.3 a pag. 9): questo *preset* corrisponde ai valori del firmware di PAN e TILT pari a 0° (zero gradi), valori che vengono impostati di fabbrica come iniziali.

La rotazione in PAN verso sinistra (rispetto alla posizione di origine) corrisponde a valori negativi descrescenti della prima coordinata; verso destra a valori positivi crescenti. La rotazione in TILT verso l'alto (rispetto alla posizione di origine) corrisponde a valori negativi decrescenti della seconda coordinata; verso il basso a valori positivi crescenti.

Ogni incremento unitario nelle coordinate indica l'incremento di 1° decimale.

Conversione tra i sistemi di coordinate

Definiti i sistemi di riferimento la conversione dalle coordinate dell'immagine panoramica agli angoli del brandeggio è immediata:

$$x_B = \frac{x_P}{w_P} \times 360 - 180 + \Delta_{hor}, \qquad y_B = \frac{y_P}{h_P} \times 90 - 45 + \Delta_{ver}$$

Nelle formule precedenti si sono indicate con la lettera B le coordinate del brandeggio e con la lettera P le coordinate dell'immagine panoramica; w_P e h_P corrispondono a larghezza e altezza di quest'ultima.

Il senso delle formule è chiaro: si calcola la posizione relativa del punto $\langle x_P, y_P \rangle$ all'interno dell'immagine panoramica, si trasforma questa posizione nella corrispondente in gradi e la si centra.

I valori Δ_{hor} e Δ_{ver} indicano rispettivamente l'*offset* orizzontale e verticale che è necessario introdurre per tener conto di eventuali disallineamenti tra la posizione iniziale del brandeggio e il centro dell'immagine panoramica dovuti al montaggio delle telecamere.

Con i risultati ottenuti l'implementazione di un metodo *GoToPosition* è a questo punto naturale:

```
void OmniServer::GoToPosition(int x, int y)
{
  ptz->HSetPoint(static_cast<float> (x) / panoImg->width *
      360.0 - 180.0 + H_ANGLE_OFFSET);
  ptz->VSetPoint(static_cast<float> (y) / panoImg->height *
      90.0 - 45.0 + V_ANGLE_OFFSET);
  ptz->Move();

  return;
}
```

Listato 7.3: Implementazione del metodo *GoToPosition*

7.2 Movimenti incrementali

Per "movimento incrementale" si intende un movimento concettualmente molto simile al "movimento in posizione" del § 7.1, con la seguente differenza: in questo caso non viene passata una posizione finale da raggiungere, quanto un valore di cui ci si vuole spostare.

Come nel caso precedente, le limitazioni del brandeggio in uso forzano la ricerca di una strada alternativa per implementare questo tipo di movimento. Riutilizzando quanto già visto nel listato 7.1, si è aggiunto un metodo specifico per il "movimento incrementale":

```
void OmniServer::StepMove(float x, float y)
{
  ptz->HSetPoint(ptz->curHPosition + x);
  ptz->VSetPoint(ptz->curVPosition + y);
```

```
  ptz->Move();

  return;
}
```

Listato 7.4: Implementazione del metodo *StepMove*

Gli argomenti di questo metodo sono gli incrementi (in gradi decimali) in PAN e TILT di cui ci si vuole muovere; la posizione finale da raggiungere viene calcolata a partire dalla posizione corrente andandole a sommare l'incremento desiderato. È quindi possibile utilizzare questo metodo per realizzare i movimenti lungo le quattro direzioni principali, ponendo di volta in volta uno degli argomenti pari a 0 (zero):

```
// PANTILT_STEP è pari a 5.0°
case 'U':
  // U - vai su
  if (omni->isOwnedBy(sid))
  {
    omni->StepMove(0.0, -PANTILT_STEP);
  }
case 'D':
  // D - vai giù
  if (omni->isOwnedBy(sid))
  {
    omni->StepMove(0.0, PANTILT_STEP);
  }
case 'L':
  // L - vai a sinistra
  if (omni->isOwnedBy(sid))
  {
    omni->StepMove(-PANTILT_STEP, 0.0);
  }
case 'R':
  // R - vai a destra
  if (omni->isOwnedBy(sid))
  {
    omni->StepMove(PANTILT_STEP, 0.0);
  }
```

Listato 7.5: *Parsing* ed esecuzione dei movimenti incrementali

Il frammento di codice precedente è inserito all'interno di un doppio costrutto **switch-case** innestato che effettua il *parsing* della stringa in arrivo. L'*if* a guardia di ogni comando di movimento è un controllo aggiuntivo analogo a quanto già visto nel § 6.3.1 a pag. 44: il suo scopo è di evitare che utenti

collegati al *Web Control Center* tentino di aggirare i controlli JavaScript della pagina per muovere Omnidome® senza rispettare il proprio turno nella coda.

Per effettuare il controllo, si confronta il *session_id* letto dalla stringa contenente il comando in arrivo con il *session_id* presente in testa alla deque, la quale denota l'utente che attualmente ha il controllo di Omnidome®:

```
bool OmniServer::isOwnedBy(std::string sessionid)
{
  return (sessionid.compare(q.front()) == 0);
}
```

Listato 7.6: Controllo aggiuntivo sul *session_id*

7.3 Zoom

La telecamera prospettica a disposizione è ad ottica fissa, non consentendo quindi di implementare comandi di zoom. Ciò nonostante si sono comunque implementati ove possibile:

- nel *Web Control Center*: i pulsanti di zoom sono presenti e seguono il comportamento degli altri pulsanti di movimento per quanto riguarda la comparsa e scomparsa;
- nel protocollo di comunicazione: si sono codificati i comandi di zoom (si veda § 4.4.1 a pag. 27) che vengono inviati dal *Web Control Center* al software C++ tramite ØMQ;
- nel software C++: il thread che effettua il *parsing* dei comandi in arrivo tramite socket ZMQ_REP è in grado di riconoscere anche i comandi di zoom, ai quali però non è associata alcuna azione.

In questo modo, se si sostituisce l'attuale telecamera prospettica con una in grado di effettuare lo zoom, si dovranno implementare solamente i comandi di zoom nel software C++ e associarli alla stringa ØMQ di comando corrispondente.

Capitolo 8

Conclusioni e sviluppi futuri

L'obiettivo che si è voluto raggiungere con questo lavoro è stato rendere disponibile, con modalità che fossero le più semplici possibili, la fruizione dei contenuti prodotti dalla telecamera Omnidome® agli utenti interessati.

La telecamera Omnidome® è in grado di mostrare contemporaneamente due tipi di immagini: un'immagine *panoramica* e un'immagine *prospettica*. La prima è un'immagine a 360° dell'ambiente circostante, che rende pertanto possibile coprire l'intera area nella quale la telecamera è posizionata; questa caratteristica la rende un prodotto ideale per quelle aree in cui sia necessario effettuare videosorveglianza, anche se abbiamo visto come le telecamere omnidirezionali abbiano degli svantaggi in termini di livello di dettaglio rispetto alle telecamere prospettiche, le quali d'altra parte hanno un campo visivo limitato che non permette di conoscere ciò che accade al di fuori di esso. La seconda immagine fornita da Omnidome® è un'immagine prospettica, grazie alla quale si va ad incrementare il livello di dettaglio di una porzione di scena particolarmente interessante.

Abbiamo mostrato come tramite il controllo di un brandeggio mobile sia possibile pilotare la telecamera prospettica direttamente dal software C++ con l'ausilio del protocollo di comunicazione MACRO. Si sono inoltre descritte delle soluzioni implementative per coordinare le due immagini, mostrando come si sia riusciti a far muovere il brandeggio in base alle coordinate di un punto nell'immagine panoramica, che viene quindi inquadrato ad alto livello di dettaglio dalla telecamera prospettica.

La combinazione delle due telecamere e del brandeggio mobile garantisce

quindi che né si perdano i dettagli di una scena da analizzare, né si perdano informazioni sugli eventi che stanno contemporaneamente avvenendo fuori dal campo visivo prospettico.

La telecamera Omnidome®, assieme al software C++ che gestisce le immagini e il brandeggio, costituisce il nucleo del nostro sistema: attorno a questo nucleo si sono implementate soluzioni per la presentazione delle informazioni, per la gestione degli utenti e per la comunicazione tra le parti.

Per quanto concerne la presentazione delle informazioni, si è configurato un web-server Apache che consente l'accesso sia al controllo del brandeggio che alla visione delle immagini della telecamera Omnidome®. La pagina di gestione che ne è risultata, che abbiamo chiamato *Web Control Center*, sfrutta le tecnologie del *web 2.0* per offrire agli utilizzatori una esperienza d'uso quanto più efficace possibile: abbiamo visto come le tecnologie AJAX e *long polling* permettano di ottimizzare la comunicazione tra il fruitore (che abbiamo definito *client*) e il produttore dei contenuti (il *server*); inoltre, abbiamo visto come l'utilizzo di una libreria JavaScript come jQuery consenta di intervenire sulla visualizzazione dei contenuti, fornendo all'utente una serie di informazioni utili all'utilizzo del *Web Control Center*. Ad esempio si sono implementati dei *countdown* dinamici per tenere informato l'utente su quanto tempo abbia ancora a disposizione per l'utilizzo di Omnidome®, o si disabilita la gestione del movimento (anche nascondendo i pulsanti) quando l'utente non può controllare l'unità.

La gestione degli utenti è stata inserita nel software C++ di Omnidome® e la si è realizzata tramite l'uso di una particolare struttura dati, la *deque*, che garantisce buoni risultati prestazionali per le più comuni operazioni quali inserimento, ricerca e cancellazione di un elemento. Un apposito thread "scheduler" si occupa di assegnare all'utente corretto l'utilizzo di Omnidome®, passando il controllo all'utente successivo dopo un preimpostato intervallo temporale.

Particolare importanza si è data alla comunicazione tra le parti: abbiamo mostrato come, utilizzando la libreria ØMQ, si riescano a creare diversi schemi di comunicazione, ognuno adatto a specifiche esigenze. In questo lavoro, infatti, ci troviamo di fronte a due diverse tipologie di comunicazione: nella prima, che abbiamo definito *N a 1*, ogni client invia comandi al server, comandi che possono riguardare la propria prenotazione o il controllo del

brandeggio di Omnidome®; nella seconda, definita *1 a N*, il server notifica ai client i cambiamenti nello stato della coda, consentendo quindi il subentro nel controllo del brandeggio al nuovo client utilizzatore.

Il risultato finale raggiunto è quello di avere un sistema in grado di mostrare le immagini della telecamera Omnidome® ad un pubblico anche vasto, garantendo la possibilità di controllo esclusivo ad un utente per volta, automatizzando tutto il processo di prenotazione e gestione della stessa.

Ciò che rimane da fare, in possibili lavori futuri, è di terminare l'implementazione dello zoom, che non è stato possibile completare in questo lavoro data la sua mancanza nella telecamera prospettica a disposizione; un'altra possibile area di sviluppo riguarda gli algoritmi di *unwrapping* dell'immagine omnidirezionale: l'implementazione degli algoritmi tramite tecnologia CUDA, sfruttando quindi le capacità di calcolo delle moderne schede video, permetterebbe di abbattere a circa un decimo il tempo necessario a processare un singolo frame, diminuendo per di più il carico di lavoro della CPU. Si potrebbero pertanto ottenere *framerate* più alti mantenendo lo stesso livello prestazionale sul server, o implementare ulteriori algoritmi come quelli già citati di *motion detection* e *tracking*.

Per finire, nell'ottica della commercializzazione di questo lavoro, si può pensare di inserire un meccanismo di autenticazione per limitare l'accesso al *Web Control Center* solo al personale autorizzato.

Ringraziamenti

Eccoci qui, infine. Molta acqua è passata sotto i ponti da quando ho iniziato quest'avventura e molte sono le persone da ringraziare per avermi accompagnato fino a questo traguardo. Anche se è stata dura, difficile e a volte complicata, ora questa parte della mia vita è alle spalle, e se guardando indietro vedo principalmente cose belle, lo è in gran parte grazie a voi.

Vorrei ringraziare per primo il prof. E. Menegatti, senza il quale oggi non ci sarebbe proprio niente da festeggiare, che è riuscito a farmi fare un lavoro che portasse a integrare due mondi che a me piacciono moltissimo, ovvero quello della robotica e quello del web; e i ragazzi dello IAS-Lab, in particolare gli Ing. A. Pretto e S. Ghidoni, che mi hanno seguito prima per il corso di Robotica e poi per questo lavoro di tesi, dandomi indicazioni e consigli sempre preziosi (commenterò di più il codice, promesso!).

Il grazie più sentito va, ovviamente, alla mamma, che mi è sempre stata vicina in questo lungo e tortuoso percorso, sempre pronta a incoraggiarmi e a spronarmi quando ne avevo bisogno, sempre partecipe alla gioia di un esame superato o sostegno dopo uno non dato; senza di lei oggi non sarei qui e non sarei quello che sono... Grazie!

Un'altra persona che mi ha permesso di raggiungere questo obiettivo è mio padre, che mi sempre ha spinto a fare di più, criticandomi quando forse lo meritavo; oggi papà, posso dire di avercela fatta, nonostante tutto.

Un grazie a Michele, con cui ho trascorso gran parte della mia infanzia, più come fratelli che come cugini... Ma quanti calci abbiamo dato a quel pallone? A Sandra e Walter, perchè puoi sempre contarci, basta che non si tratti di rispondere al telefono; a Nicola, Federica, la piccola Sofia, Norma,

Maurizio e Nadir, che ho visto crescere: siete delle persone eccezionali, e io non riuscirò mai a ringraziarvi abbastanza per tutto quello che avete fatto; a Rosella, Stefano e Manuel, che non sono della famiglia ma è come se lo fossero, e forse anche qualcosa di più; a tutte le zie e cugini, che anche se non ci vediamo chiedono sempre di me.

Un ringraziamento speciale va agli ex colleghi di Psicologia: Rosa, Roberta, Anna, Luca, Luigi e Davide. Grazie per i due anni trascorsi assieme, in cui mi sono sentito più a casa che a lavoro... Avete visto che non ho mollato? E un altro grazie speciale va al mio capo, anche se più che un boss è un amico, Simone, che mi ha aiutato in momenti difficili e concesso molta libertà in quest'ultimo periodo... Dai che adesso cominciamo a fare davvero sul serio!

Gli anni trascorsi a Padova mi hanno dato modo di conoscere tante persone fantastiche: primo tra tutti Matteo, compagno d'appartamento per 5 anni, amico prima e vero amico ora... Ma quando facciamo un altro giro al salone del vino novello, eh? Poi tutti gli amici conosciuti tra i banchi del Paolotti prima e della mitica Ke dopo: Alberto e Cristina, il Company, Pablo, il Maz, il Poggia e Sturo ("*Codice, Brisotto, codice!*"), il maratoneta (alcolico, soprattutto) Teo, Darione e il Cesco, Vicky e Jenny, Sandrino e le pause pranzo passate a giocare a biliardo (te lo ricordi Furio?!), Giacomino che non perdeva un *Cordialmente*, Camillo che spuntava dal nulla quando meno te l'aspettavi, Denis e il Fede e l'appuntamento fisso alla San Francisco by night, il Popolo e relativo appartamento in cui non sapevi mai chi potevi incontrare, Ema che ho ritrovato psicologo, l'allegra combriccola di UnipdLeague, col Capo sempre pronto a districare i miei dubbi su questa funzione PHP o quel metodo Java, e guarda un po', ringrazio pure l'addetto Pasqualotto, per non avermi mai sospeso l'account del DEI.

Ho tenuto per ultimi gli amici più stretti, gli aaamiconi, *assolutamente* in ordine sparso: Bubi, che "*una palla al piede al confronto è una gioia*" (auto cit.); l'uomo-nuovo-io-non-sclero-più Alessio, portatore sano di febbre gialla ed esperto di *fluidi*; Flavio, altro esperto di *fluidi*, l'unico in grado di scroccare griglia e braci pronte al vicino; Cristiano ("*guarda quanto è grasso quel bambino!*") ed Elena ("*Cristiano piantala.*"); il particolare Sergio Maria, che ci ha aperto la taverna (e la cantina!) tante volte (a proposito, quando ci inviti di nuovo?); Marchy che non sconfigge la pasta di Checco,

Vale che mette i cioccolatini nelle ciabatte, il little aaamicone Matteo, che speriamo smetta di urlare prima o poi (povera Vale); Mauro, di cui conosco a memoria i nascondigli in Vietcong, unica ragione per cui è nato "*coltello!*"; Igor, tappa fissa per caffè e Simpsons dopo la Forcellini; Tcior che mi ha fatto mangiare gratis per un bel pezzo; Marco l'africano e Alessandra, fenomenale produttrice di dolcetti; il Ciccio, che "*me morosa no voe*", di qualsiasi cosa si tratti; Elisa, amica speciale ma con cui è meglio non bere mai Montenegro (non potete vincere, fidatevi); il Tama, per avermi messo al tavolo "amici della sposa" al matrimonio.

Per ultimissime ho tenuto 4 persone, che in un modo o nell'altro mi sono state vicine in queste settimane di lavoro: Ciarblà, che è *indubbiamente* (dai, non potevo non metterlo, cosa credevi?!) una delle pochissime persone che la mia tesi l'ha letta e persino *usata*, dandomi qualche dritta su LaTeX (varda che bravo) e sorbendosi talvolta le mie paranoie; Federica, che in tutto questo c'è passata da poco e che mi è stata vicina per molte cose che non c'entravano niente con la tesi; Tea, che vedo troppo poco ma che comunque c'è sempre, con cui parlerei ore e ore perchè dai nostri discorsi vengono sempre fuori ottimi consigli; e, beati gli ultimi che saranno i primi, Maria, per i nostri contatti quotidiani in cui si è fatto a gara a chi stava messo peggio con il "mondo università" in generale... Dai che ce la fai anche tu, resisti!

Che cosa rimane da dire? Rimane da dire... *Hora finita est!*

M.

Bibliografia

[1] S. Ghidoni, A. Pretto e E. Menegatti. «Cooperative Tracking of Moving Objects and Face Detection with a Dual Camera Sensor». In: *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010)* (2010).

[2] D. Scaramuzza e R. Siegwart. «Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles». In: *IEEE Transactions on Robotics* 24(5) (ott. 2008), pp. 1015–1026.

[3] *Video for Linux Two API Specification*. 2009. URL: `http://www.linuxtv.org/downloads/legacy/video4linux/API/V4L2_API/spec-single/v4l2.html`.

[4] *PHP: Sessions - Manual*. 2012. URL: `http://www.php.net/manual/en/features.sessions.php`.

[5] *deque - C++ Reference*. 2012. URL: `http://www.cplusplus.com/reference/stl/deque/`.

[6] Jesse James Garrett. *Ajax: A New Approach to Web Applications*. 2005. URL: `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`.

[7] *jQuery Usage Statistics*. 2012. URL: `http://trends.builtwith.com/javascript/JQuery`.

[8] *Sites Using jQuery*. 2012. URL: `http://docs.jquery.com/Sites_Using_jQuery`.

[9] The Open Group. *Documentazione POSIX shm_ open*. 1997. URL: `http://pubs.opengroup.org/onlinepubs/007908799/xsh/shm_open.html`.

[10] Pieter Hintjens. *ØMQ in a Hundred Words*. 2012. URL: `http://zguide.zeromq.org/page:all#-MQ-in-a-Hundred-Words`.

[11] *ZeroMQ helpers for example applications*. 2012. URL: `https://github.com/imatix/zguide/blob/master/examples/C%2B%2B/zhelpers.hpp`.

[12] G. Scotti et al. «A novel dual camera intelligent sensor for high definition 360 degrees surveillance». In: *Intelligent Distributed Surveillance Systems (IDSS-04), IEE* (feb. 2004), pp. 26–30.

[13] G. Scotti et al. «Dual camera intelligent sensor for high definition 360 degrees surveillance». In: *IEE Proceedings, Vision, Image and Signal Processing* 152(2) (apr. 2005), pp. 250–257.