UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Marker

Elaborato di esame di: **Roberto Gentile, Buntia Burè Girelli, Massimo Zanuzzi**

Consegnato il: **26 settembre 2005**

Sommario

Il progetto svolto si integra in un più ampio progetto che ha come finalità quella di sviluppare un sistema di localizzazione a basso costo per il robot ActivMedia Pioneer 3 "Morgul", robot sviluppato presso la facoltà di Ingegneria di Brescia. Il nostro compito è stato quello di completare il lavoro iniziato da un tesista, Roberto Fedrigotti, il quale ha sviluppato un sistema di localizzazione che si basa sul riconoscimento di un marker attivo applicato al robot. Abbiamo quindi sviluppato, utilizzando le librerie Aria, un programma che si integra con quello elaborato dal tesista, e che funge da interfaccia verso la rete.

1. Introduzione

Il nostro progetto trova collocazione all'interno di un progetto più importante e di più ampio respiro: tale progetto si pone come obiettivo la realizzazione di un robot in grado di svolgere operazioni di video-sorveglianza all'interno di un ambiente prestabilito. Per poter far questo il robot deve essere in grado di muoversi *"autonomamente"* all'interno di un ambiente che egli stesso conosce (dispone, infatti, della mappa dell'ambiente in cui deve operare), evitando opportunamente eventuali ostacoli contro cui può venire a contatto, siano essi previsti o imprevisti, statici o in movimento. A tal fine deve essere dotato di sensori di rilevamento presenza, quali i sonar, e di sistemi che gli permettono di ottenere delle informazioni sulla sua posizione, quali ad esempio l'odometria.

Il robot che si è scelto di utilizzare è un robot interamente progettato e realizzato presso la facoltà di Brescia, e prende il nome di MORGUL (Mobile Observation Robot for Guarding the University Laboratories).

Figura 1.0: robot Morgul.

Il robot utilizzato è un modello Pioneer III AT della Activmedia. È dotato, come si può notare dalla figura "*Figura 1.0*", di 4 ruote che conducono scivolando (ovvero skid-steer), ad ognuna delle quali è collegato un motore D.C. dotato di encoder (per un totale di 4 motori), che permette al robot di eseguire controlli odometrici. Lo scopo per il quale è stato progettato è appunto quello di eseguire un controllo di sorveglianza automatico basato su connessione internet. Da un punto di vista sensoristico, il robot è dotato di sonar e bumper situati nella parte frontale del robot. Tale peculiarità può essere vista nella figura "*Figura 1.1*".

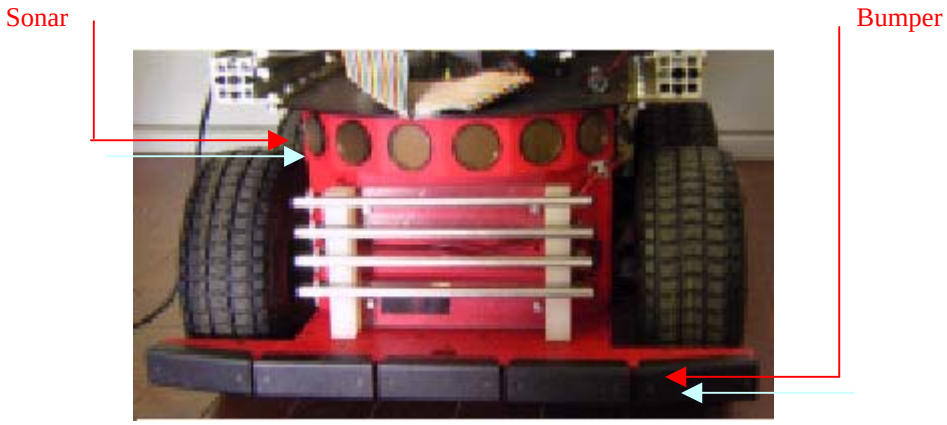


Figura 1.1: parte frontale del robot Morgul.

Il robot *Morgul*, così com'è realizzato, è in grado di muoversi all'interno di un ambiente prestabilito, ma non è in grado di svolgere operazioni di video-sorveglianza *"autonomamente"*; infatti, mediante programmi quali Saphira, è possibile far muovere il robot all'interno di un ambiente del quale si dispone una mappa (in formato .map), facendolo muovere:

- ⇨ Casualmente, radendo le pareti ed evitando ostacoli utilizzando programmi quali "*wander*";
- ⇨ Facendogli seguire un percorso prestabilito definendo sulla mappa dei goal, ovvero dei punti sulla mappa per i quali passare.

Tramite il programma "*Mobile eyes*" è inoltre possibile far acquisire al robot le immagini catturate dalla telecamera di cui il robot stesso è dotato (nella figura "*Figura 1.0*" si può notare appesa nella parte alta del robot, sotto si può apprezzare un suo ingrandimento).

A.  B. 

Figura 1.2: A: telecamera applicata al robot Morgul B: telecamera Philips ToUCam 2 Pro.

Quello che non permette tuttavia al robot di muoversi *"autonomamente"*, è l'impossibilità dello stesso di essere in grado, in ogni istante, di conoscere la sua posizione con esattezza. Questo problema si riscontra in tutti i robot, di qualsiasi natura essi siano, industriali o mobili. Per farsi un'idea, basti pensare al momento in cui un robot viene attivato: in quell'istante tutte le variabili associate alla sua posizione sono azzerate o hanno assunto un valore casuale, ed esso non può in nessun modo conoscere *"autonomamente"* la sua posizione. Nei robot industriali è previsto solitamente un sistema/meccanismo di inizializzazione, che, portando il robot in una posizione prestabilita (detta origine), permette di inizializzare correttamente le sue coordinate. Un sistema di questo tipo non può essere utilizzato per robot quali Morgul, a meno che non si preveda ogni volta di farlo tornare, prima di disattivarlo, in una posizione prestabilita definita ad esempio "casa": in quella maniera il robot al momento dell'accensione saprebbe esattamente dove si trova sulla mappa (questa è la soluzione adottata per molti robot commerciali di uso domestico, quale ad esempio il robot da giardino che pubblicizzano molto in tv).

La necessità del robot di conoscere con esattezza la sua posizione non si limita tuttavia al solo momento dell'attivazione (infatti, visto che per il robot *Morgul* è stata creata una stazione di ricarica, si potrebbe usare quella come "casa"): sul robot viene eseguito un programma che, sulla base delle librerie Aria e

della mappa dell'ambiente, è in grado di far seguire al robot un determinato percorso. Questo programma fa riferimento unicamente all'odometria, quindi vi è la possibilità che possano esserci degli errori nel calcolo delle posizioni. Pertanto i dati di posizione (x,y) devono essere periodicamente aggiornati, in modo da correggere eventuali scarti sulla posizione accumulatisi.

È necessario quindi creare un sistema in grado di

- Localizzare nell'ambiente circoscritto il robot.
- Comunicargli la sua posizione

1.1. Localizzazione

Per localizzare il robot, si è pensato di utilizzare una telecamera (a basso costo) il cui campo visivo riesca ad inquadrare tutto l'ambiente in cui il robot deve muoversi; l'idea è che, nel caso in cui il robot ne abbia bisogno, deve poter conoscere la propria posizione semplicemente facendo richiesta ad un calcolatore esterno (detto *"Server Fedro"*) connesso ad una telecamera fissa in grado di acquisire tutta la scena in cui si muove il robot stesso. Una volta ottenuti i dati richiesti è in grado di aggiornare il proprio data-base interno. Nella realtà la richiesta fatta dal robot viene inviata non direttamente al Server Fedro, ma ad un altro calcolatore che si interpone tra loro, sul quale gira il programma *"Black Box"*: questo perché il Server Fedro fornisce le coordinate della posizione del robot sull'immagine ottenuta dalla telecamera, mentre al robot servono le coordinate nel modo reale (occorre perciò fare una conversione delle coordinate). Tale calcolatore si occuperà di inoltrare la richiesta al calcolatore Server Fedro, e di effettuare la conversione dei dati ottenuti.

Per avere un'idea più chiara del sistema adottato si osservi la figura *"Figura 1.3"*.

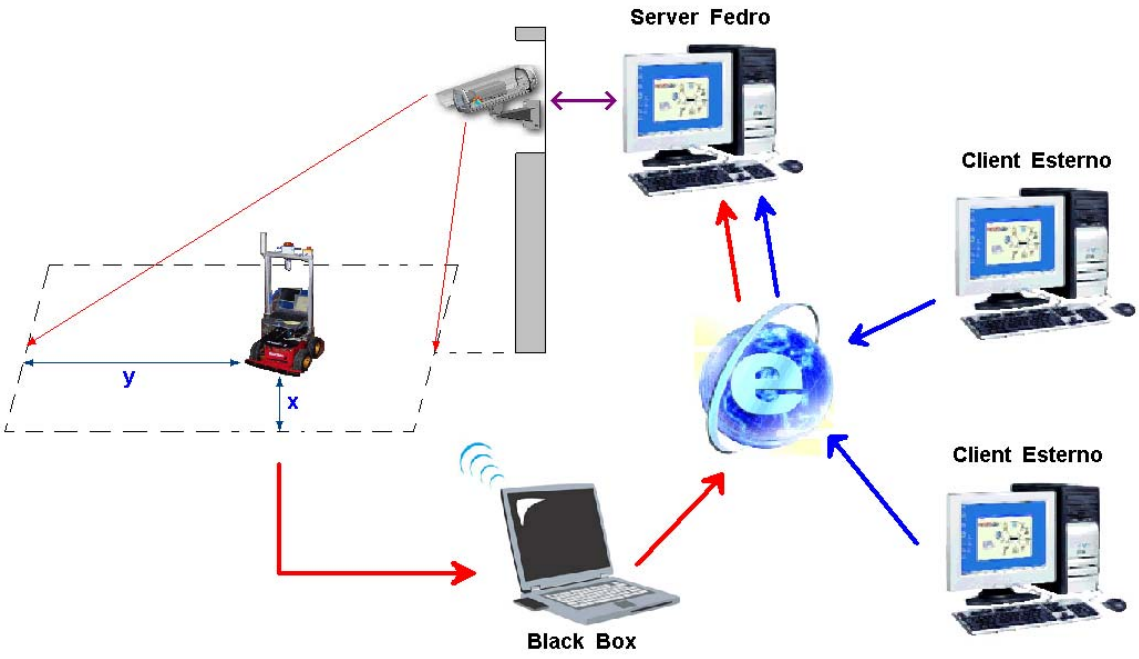


Figura 1.3: sistema adottato.

Per testare il sistema si è scelto di far muover *Morgul* all'interno del giardino del campus universitario, mentre come telecamera si è utilizzata una telecamera a basso costo (Philips ToUCam 2 Pro, vedi in*"Figura 1.2B"*) posta nell'ufficio del prof. Cassinis situato al secondo piano dell'edificio universitario. Un'immagine di quanto inquadrato dalla telecamera si può apprezzare nella figura *"Figura 1.4"* (o collegandosi all'indirizzo http://www.ing.unibs.it/~cassinis/ARL/ per guardare le immagini catturate dalla telecamera in diretta).

La localizzazione del robot nell'immagine, anche in un'ideale situazione di assoluta visibilità (ovvero cielo limpido senza nessuna nuvola, con una luminosità ideale), non è immediata come può sembrare: per come abbiamo presentato finora il robot *Morgul*, non sempre sarà possibile riconoscerlo all'interno dell'immagine; infatti, se il robot si trova nella parte bassa dell'immagine, ovvero vicino alla telecamera, la sua localizzazione risulta essere semplice, si può individuare il robot con un certo grado di affidabilità; ma proviamo a supporre che il robot venga a trovarsi in mezzo agli alberi posti sulla parte sinistra/alta dell'immagine, o più semplicemente appena sotto la gradinata posta nella parte centrale/alta dell'immagine, in questo caso la sua localizzazione *esatta* non è più così scontata.

Figura 1.4: immagine catturata dalla telecamera nell'ufficio del prof. Cassinis.

Per poter localizzare più facilmente, in maniera più affidabile, il robot mediante l'utilizzo della telecamera, è stato quindi necessario applicare al robot un marker attivo, costituito da led ad alta intensità che vengono fatti lampeggiare ad una frequenza ben definita, e ben distinguibile da ogni comportamento di qualsiasi oggetto o fenomeno esistente in natura. Nelle immagini *"Figura 1.5"* e *"Figura 1.6"* si può apprezzare rispettivamente, i led ad alta intensità utilizzati, e il sistema completo del marker applicato nella parte superiore del robot *Morgul*.

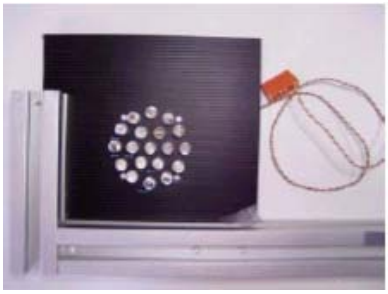


Figura 1.5: blocco led ad alta intensità.

Figura 1.6: sistema marker attivo posto sull'estremità del robot Morgul.

In particolare, la frequenza alla quale lampeggiano i led ad alta intensità, è tale da creare dei battimenti con la frequenza della telecamera d'acquisizione della scena, in modo tale che la telecamera stessa è in grado di riconoscere il robot in qualunque posizione si trovi all'interno della scena inquadrata, anche alla luce del giorno. La telecamera quindi, una volta attivato il marker sul robot, acquisisce una serie di 75 immagini successive prese all'apposita frequenza; tale immagini vengono "passate" ad un calcolatore collegato serialmente alla telecamera il quale, attraverso un algoritmo sviluppato dal tesista Roberto Fedrigotti, per elaborazione e differenza di queste immagini, riesce ad individuare nell'immagina un *blob* che attesta, con una certa percentuale di attendibilità, la presunta posizione del robot. Facciamo notare che attraverso questo algoritmo vengono ricavati una serie di parametri indicanti:

- Coordinate della posizione del robot nell'immagine: < x > < y >;
- Massimo valore del blob individuato: < maxvalue >;
- Numero totale di punti appartenenti al blob individuato: < numtot >;
- Affidabilità del blob individuato come quello associato l robot: < average >;
- Un parametro per indicare se l'elaborazione è affetta da errori: < errore >;
- Un parametro indicante il tempo in cui è terminata l'ultima elaborazione: < timestamp >.

Una volta che si è riuscito ad identificare la posizione del robot all'interno dell'immagine (ottenendo le coordinate <x> e <y>), si utilizza l'algoritmo di conversione del programma *Black Box* (sviluppato dagli ormai laureati Panteghini-Cagno), per ottenere le coordinate reali del robot nell'ambiente in cui esso si trova, riferite ad un'origine identificata nel mondo reale.

1.2. Comunicazione

Il sistema di localizzazione realizzato si basa principalmente sulla possibilità di far individuare il robot all'interno dell'ambiente in cui lui stesso opera (nel momento in cui esso ne faccia richiesta), mediante l'utilizzo di una telecamera e di un algoritmo di elaborazione delle immagini provenienti dalla stessa. Per poter far eseguire l'algoritmo è necessario utilizzare un calcolatore, che viene collegato alla telecamera attraverso una connessione seriale (sia essa RS-232 o USB): su tale calcolatore quindi verrà fatto eseguire l'algoritmo "*Fedro*", il quale si occuperà di

- Colloquiare con la telecamera, ovvero di catturare le immagini provenienti da essa;
- Elaborare in successione tali immagini al fine di individuare il blob che più rassomiglia al comportamento del marker, e che quindi identifica sull'immagine la posizione del robot;

Ne segue che nel momento in cui il robot avrà bisogno di conoscere la propria posizione, dovrà riuscire in qualche modo a collegarsi a tale calcolatore per ottenere i dati richiesti. Questo è appunto l'obiettivo del nostro progetto, ovvero creare un sistema di comunicazione che permetta di rendere disponibili i parametri elaborati dall'algoritmo *"Fedro"* in primo luogo al robot, ma anche al mondo esterno. Il calcolatore che è collegato alla telecamera di localizzazione deve quindi occuparsi di mettere in esecuzione un software il cui compito è:

- Mettere a disposizione tali dati al robot e a chiunque ne faccia richiesta dall'esterno, segnalando eventuali anomalie.

Tale programma, che noi abbiamo chiamato *"Server TCP/IP"*, è un software che dovrà obbligatoriamente andare ad integrarsi con l'algoritmo *"Fedro"*, in maniera tale da poter condividere facilmente delle strutture dati (quelle che occorre rendere visibili al resto del mondo) senza quindi dover prevedere meccanismi di salvataggio dati su file, che rallenterebbero eccessivamente l'esecuzione del programma, con conseguente rischio di perdita del sincronismo con la telecamera; una perdita di sincronismo con la telecamera risulterebbe infatti molto deleteria, in quanto l'elaborazione in tal caso darebbe dei risultati totalmente sbagliati. La fusione tra i due pacchetti software ha dato perciò origine ad un unico programma, chiamato di comune accordo con Fedrigotti *"**Server Fedro**"*: tale programma dovrà essere in esecuzione (perennemente) sul calcolatore che sarà collegato alla telecamera che effettuerà la localizzazione, e dovrà essere in grado di gestire:

- La comunicazione con la telecamera;
- La comunicazione basata su protocollo TCP/IP con il robot Morgul/Black Box ed il resto del mondo.

- **Facciamo notare che la scelta del protocollo di comunicazione TCP/IP è supportata dal fatto che il 96% delle LAN esistenti supportano a basso livello il protocollo Ethernet, e, sopra di esso, il protocollo IP;**

- **Finora abbiamo parlato di *"resto del mondo"*: questo perché nel progetto è stata prevista la passibilità che non solo il robot Morgul faccia una richiesta di localizzazione, ma anche un qualsiasi Client esterno; il Server da noi realizzato, infatti, è in grado di gestire contemporaneamente fino a 10 connessioni simultanee, lasciando ovviamente un canale preferenziale al robot, l'unico ad avere priorità rispetto alle altre 9 possibili connessioni;**

- **All'interno del giardino del campus universitario è in funzione una rete privata di tipo wireless.**

Fondamentalmente il programma da noi realizzato ha il compito di realizzare un server: una volta attivato, il Server TCP/IP attiva un Socket e si mette in ascolto su una porta prestabilita, aprendo una comunicazione di tipo TCP/IP con qualsiasi Client che si colleghi a tale porta. Nel momento in cui un Client esterno esegue una richiesta di localizzazione (per esempio *"where am I"*), il nostro Server deve:

a. Collegarsi al programma *"Fedro"* facendo una richiesta di elaborazione immagini;
b. Attendere la risposta, gestendo opportunamente Time-out e/o situazioni di malfunzionamento;
c. Mandare una risposta, sulla base del protocollo TCP/IP, ai Client che ne hanno fatto richiesta.

In particolare, dal punto di vista del robot *Morgul*, la richiesta di localizzazione viene fatta nella seguente modalità:

a. Attraverso il programma realizzato del prof. Tampalini, il robot accende il marker;
b. Il programma del prof. Tampalini attiva una connessione TCP/IP sul calcolatore sul quale è in esecuzione il programma *"Black Box"*;
c. Viene inoltrata la richiesta di localizzazione al programma *"Server Fedro"*, che elabora le immagini provenienti dalla telecamera e inoltra al programma *"Black Box"* i dati ricavati;
d. Il programma *"Black Box"* converte le coordinate dell'immagine in coordinate reali ed invia la risposta al robot *Morgul*;

e. In caso di risposta negativa, a seconda del tipo di messaggio d'errore, il robot *Morgul*, attraverso un apposito programma realizzato dallo studente Oberti, ruota lo specchio del marker , e riprende la procedura dal passo *a*, fino ad ottenere una risposta positiva dal programma *"Server Fedro"*.

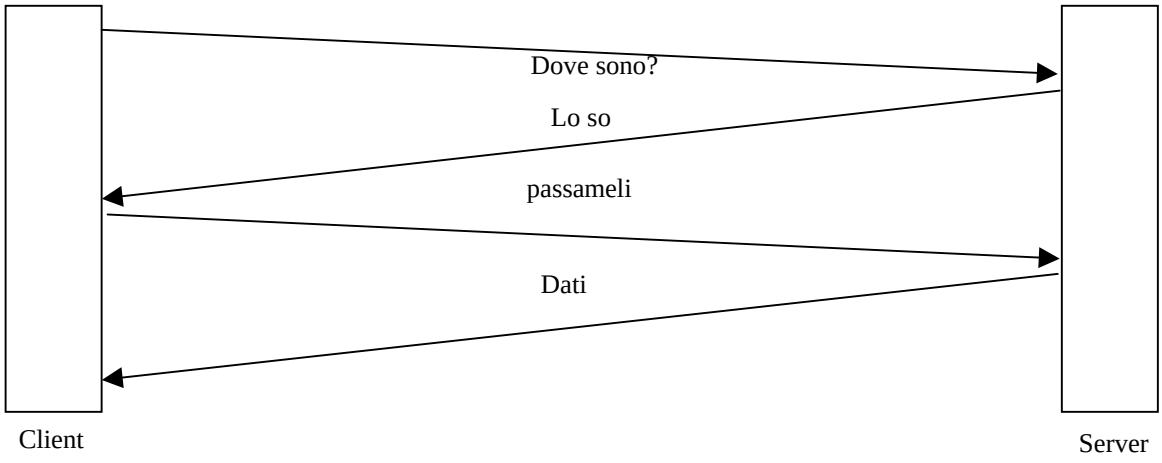
1.3. Protocollo adottato per la comunicazione Server/Client

Per uniformare la comunicazione tra Server e Client, si è deciso di adottare un protocollo di comunicazione comune, basato sull'utilizzo di messaggi predefiniti, da utilizzare secondo una sequenza ben definita.

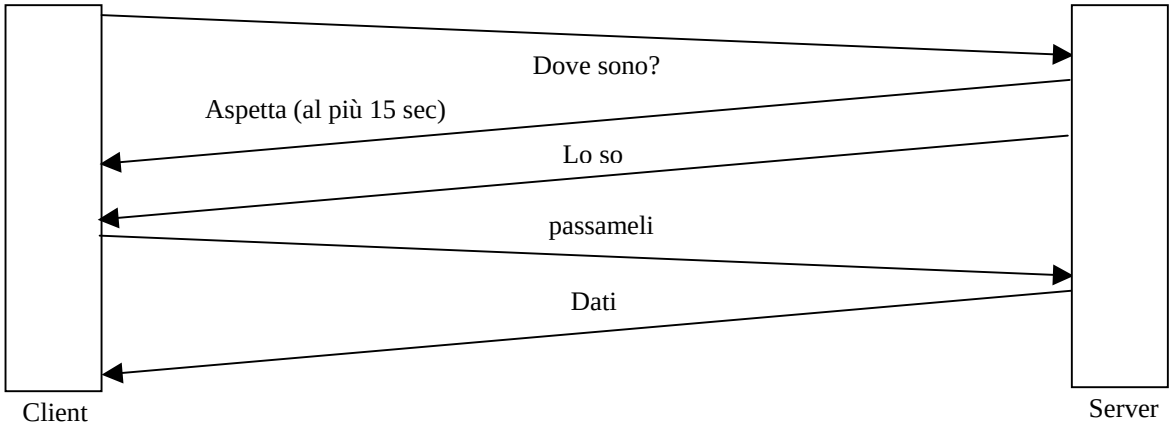
- **Tale protocollo viene adottato per la richiesta e comunicazione di dati tra Server e Client, qualunque essi siano (Server Fedro / BB, Server Fedro / Client, BB / Robot);**
- **Il protocollo è stato sviluppato di comune accordo tra le parti (Fedrigotti, Gruppo ServerFedro, Gruppo BB) e professori (Cassinis e Tampalini).**

Di seguito diamo una spiegazione succinta della struttura e delle situazioni previste dal protocollo di comunicazione:

Caso A: Risposta positiva, Server Fedro riceve la richiesta dal Client, ottiene la posizione dalla Web Cam e invia i dati correttamente al Client

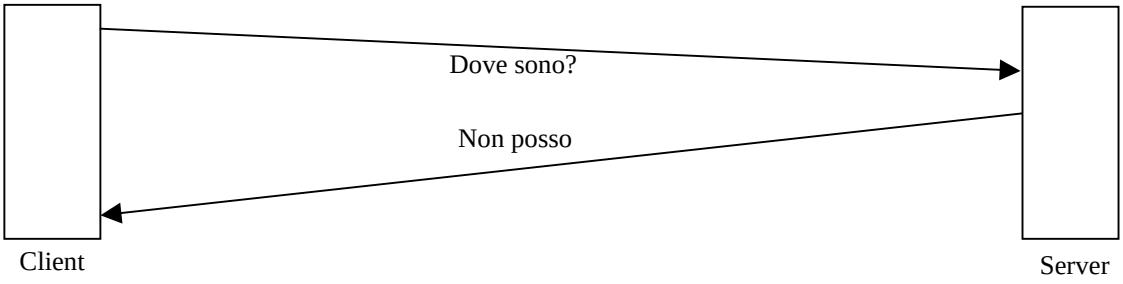


Caso B: Risposta positiva, Server Fedro riceve la richiesta dal Client, ottiene la posizione dalla Web Cam e invia i dati correttamente al Client dopo un Time Out al più 15 secondi.



Caso C: Risposta negativa, Server Fedro riceve la richiesta dal Client, non può ottenere la posizione dalla Web Cam e segnala al Client l'impossibilità nell'ottenerli. In realtà nel programma sviluppato, questa situazione può avvenire in 3 circostanze diverse:

- ⇨ L'elaborazione di Fedro è portata a termine con successo ma non riesce ad individuare nell'immagine elaborata (per qualche motivo) il blob associato al robot.
- ⇨ L'elaborazione non è portata a termine a causa della perdita del collegamento con la Webcam.
- ⇨ L'elaborazione non è portata a termine a causa della perdita del sincronismo con la Webcam.



Sempre di comune accordo si è pensato di associare ad ogni messaggio da inviare tra Client e Server, un messaggio in formato XML, messaggio che è salvato all'interno dell'omonimo file XML (tutti raccolti nella cartella ./marker/ServerFedro/xml). Nel momento in cui in futuro si vorrà cambiare il contenuto di questi messaggi, o aggiungere nuovi tag, basta semplicemente modificare questi file, senza dover mettere mano al codice. Ogni controllo è stato fatto appunto all'interno del codice in maniera più flessibile possibile, al punto di facilitare queste possibili modifiche future.

2. Il problema affrontato

L'elaborato consiste quindi, nel creare un modulo software da integrare con l'applicativo FEDRO, in modo tale da poter rendere pubblici i dati acquisiti ed elaborati dallo stesso comunicandoli su reti TCP/IP.

Il modulo deve essere in grado di poter accettare più connessioni contemporanee, per adesso senza alcuna forma di protezione/sicurezza (autenticazione dei partecipanti, comunicazioni cifrate, etc.).

3. La soluzione adottata

Per meglio integrarsi con l'applicativo FEDRO, cioè senza richiederne particolari modifiche, abbiamo deciso di utilizzare la tecnologia multithread, in questo modo il modulo diventa un sottoprocesso, totalmente indipendente, del processo padre FEDRO. La comunicazione tra i due processi avviene grazie alla condivisione di parte della memoria regolata da un semaforo: questo per evitare tutti quei problemi che la tecnica del mutithreading ha con le memorie condivise.

3.1. Il server

Deciso come relazionarsi a FEDRO, abbiamo impostato la struttura del nostro applicativo. Abbiamo deciso di implementare un server TCP/IP capace di gestire un numero predefinito (scelto dall'utente) di connessioni contemporanee; il server è un thread che deve:

- Verificare la presenza di nuove richieste di connessione.
- Gestire le connessioni "on-line".

- Chiudere ed eliminare dalla memoria connessioni ormai sconnesse.
- Gestire la memoria condivisa con l'applicativo FEDRO.

3.2. La connessione

Data l'impostazione al server TCP/IP ci siamo focalizzati sulla singola comunicazione con l'host remoto. Per rendere indipendenti tra loro le varie comunicazioni contemporanee abbiamo deciso di gestire ogni singola connessione accettata dal server come un thread in continuo ascolto sul canale di comunicazione dati. Questo ci ha permesso di:

- Gestire separatamente le comunicazioni.
- Separare il protocollo di comunicazione dalla gestione della connessione.

Di fatto ogni singola connessione deve verificare:

- La presenza dell'insorgere di errori di comunicazione.
- La presenza d'informazioni sul canale dati (aperto dal server) con l'host remoto: la logica di comunicazione non è di competenza della singola connessione.

3.3. Il protocollo

Per semplificare la leggibilità e la modifica del modulo software, abbiamo deciso di separare la parte "a basso livello" della gestione della connessione e quindi del socket TCP/IP da quella "ad alto livello" della logica di comunicazione e della formattazione dei messaggi da scambiare: il protocollo è stato quindi implementato separatamente. Nel momento in cui la connessione si accorge della presenza di dati sul socket, richiama il protocollo passandogli tutti i dati; è compito del protocollo verificarli, prendere le giuste decisioni ed eventualmente mandare una risposta al client remoto.

Nella figura *"Figura 3.1"*, posta sotto, è mostrata la struttura riassuntiva del modulo software implementato.

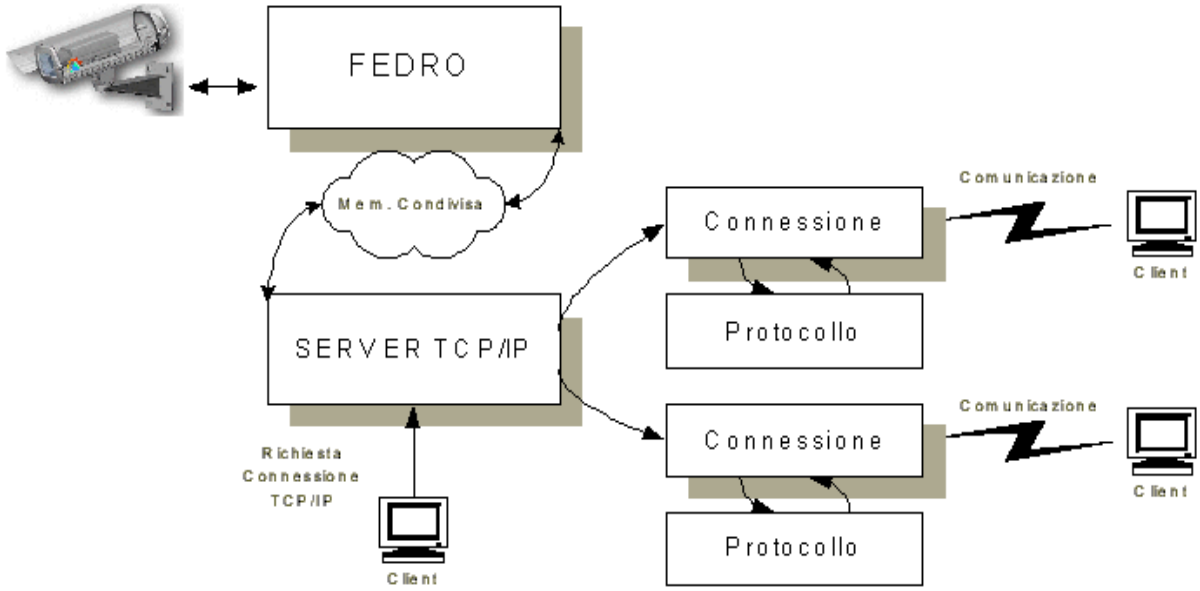


Figura 3.1: struttura riassuntiva del modulo software implementato.

4. Le classi utilizzate

L'implementazione di tutto il lavoro è stata fatta in C++ sfruttando le librerie Aria (ver. 2.3 release 3) per la creazione e il mantenimento dei thread e dei canali socket TCP/IP.

Utilizzando una programmazione ad oggetti è stata creata una classe per ogni entità precedentemente individuata: Server, Connessione e Protocollo; una quarta classe (ScambioDati) contiene tutte quelle informazioni che devono essere scambiate tra il server TCP/IP e l'applicativo FEDRO: di fatto un'istanza di ScambioDati rappresenta la memoria condivisa dai due processi.

Seguirà ora una breve spiegazione del funzionamento di ogni singola classe usata: per meglio comprendere quanto esposto l'immagine sottostante rappresenta il diagramma delle classi con tutti gli attributi e metodi.

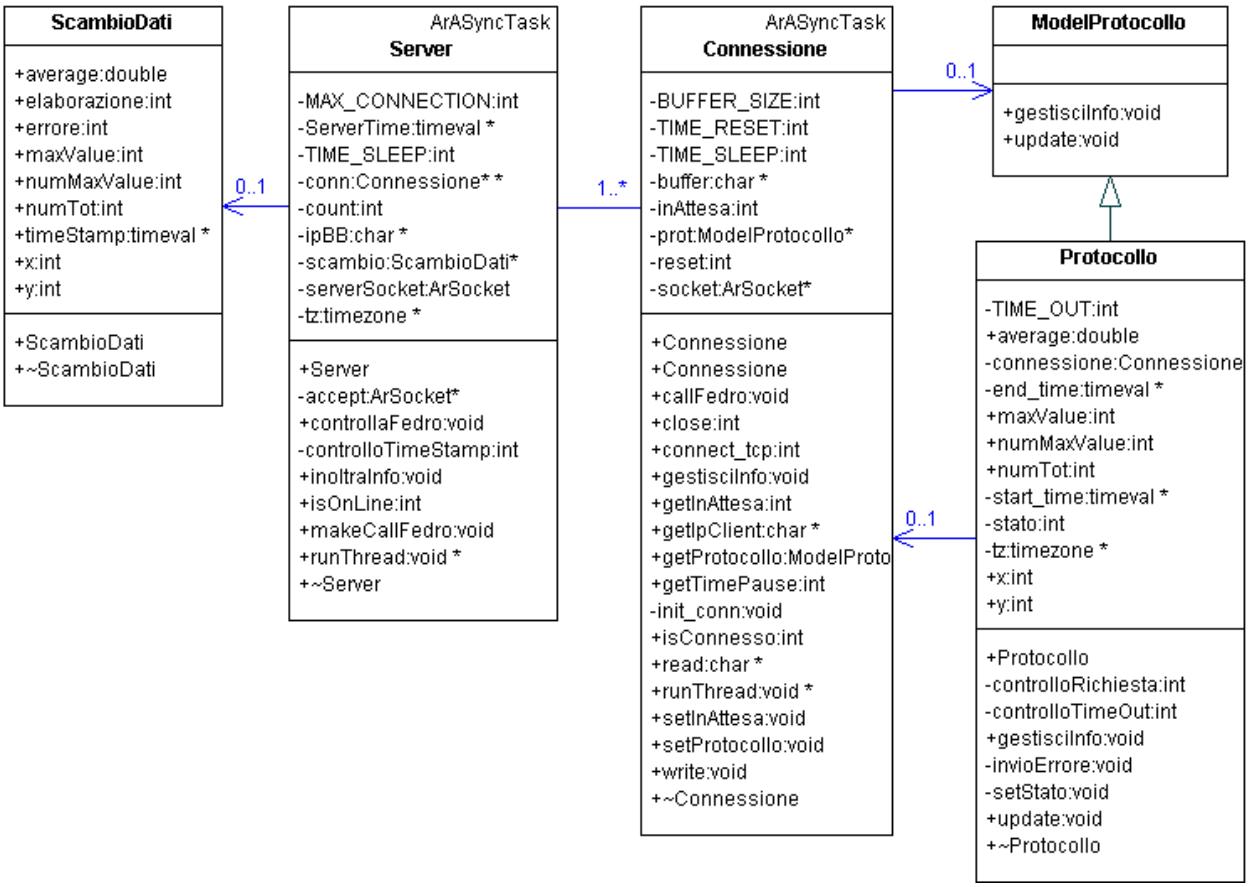


Figura 4.1: diagramma delle classi utilizzate.

4.1. Server.h

La classe Server implementa un server TCP/IP che rimane in ascolto di connessioni su una determinata porta. Questa classe estende la classe ArASyncTask della libreria Aria: è un thread asincrono. Ad ogni esecuzione del thread vengono innanzitutto monitorate le connessioni presenti per verificare che non ci siano connessioni chiuse, in questo caso le stesse vengono eliminate dalla memoria: infatti viene gestito un array di connessioni, con una dimensione massima preimpostata. In seguito il server verifica la presenza di richieste di connessione: se c'è una nuova richiesta si provvede a creare e memorizzare nell'array un nuovo oggetto Connessione col relativo canale socket; essendo però le connessioni gestite in un buffer di dimensione fissa e prestabilita, quando il buffer è pieno le nuove richieste di connessione

verranno semplicemente scartate. Per evitare problemi di funzionamento del sistema FEDRO/BLACKBOX abbiamo deciso di dare maggiore priorità ad un certo IP remoto (configurabile dall'utente), in modo tale che se anche il buffer connessioni fosse pieno, una connessione dall'IP specificato non viene comunque negata (quell'IP sarà quello assegnato al robot). Altro ruolo fondamentale dell'oggetto Server è la gestione della memoria condivisa con FEDRO. Di fatto il Server svolge la funzione di arbitro per quanto riguarda le richieste di elaborazione immagini da parte di FEDRO (ricordo che FEDRO elabora le immagini on-demand), in modo che se più connessioni richiedono l'avvio della procedura di calcolo il Server invia un'unica richiesta e se un'elaborazione è già in atto, il Server non richiederà mai una nuova elaborazione ma semplicemente ne attende la conclusione per poi informare le connessioni della presenza di nuovi dati disponibili. I parametri di configurazione del Server sono:

- La porta su cui è accessibile il servizio.
- Il numero massimo di connessioni memorizzabili che corrisponde alla dimensione del buffer Connessioni.
- L'indirizzo IP remoto con maggiore priorità rispetto agli altri.
- Il tempo di pausa del thread (in millisecondi)
- Il tempo di validità dei dati prodotti da FEDRO (in millisecondi): se i dati sono ancora ritenuti validi, allora ad una richiesta (da parte di almeno una Connessione) di elaborazione il Server non informerà FEDRO ma invierà alla/e Connessione/i i dati.

Tutti questi parametri di configurazione, ed altri che verranno specificati in seguito, sono elencati nel file "./marker/ServerFedro/xml/config.xml" che viene caricato dal Server al suo avvio per configurare l'applicazione. Nella figura *"Figura 4.2"*, posta di seguito, è riassunta la logica di funzionamento del thread del Server.

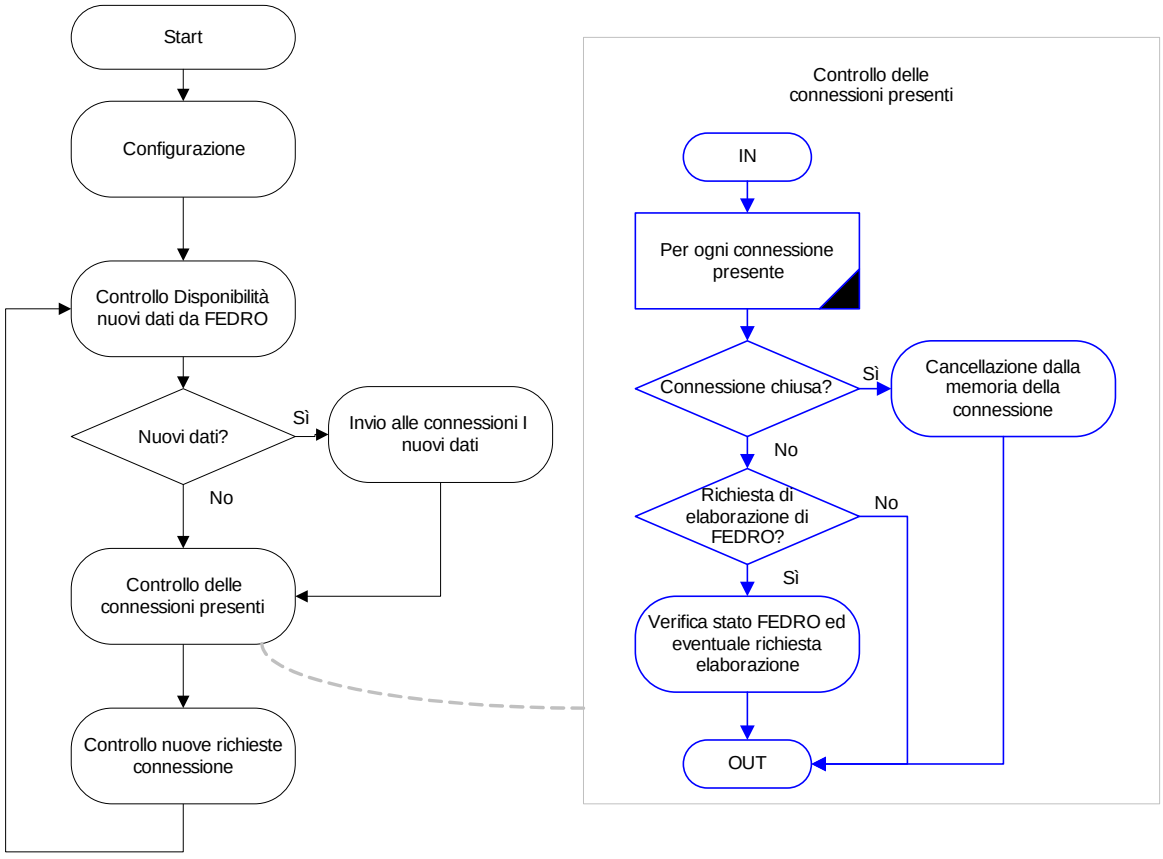


Figura 4.2: *logica di funzionamento del thread del Server.*

4.2. Connessione.h

La classe Connessione gestisce direttamente il socket TCP/IP per la comunicazione dati implementando i metodi per leggere e scrivere stringhe di caratteri, per connettere e disconnettere lo stesso, per gestire gli errori di comunicazione dovuti al canale. Come la classe Server, anche la Connessione estende la classe ArASyncTask della libreria Aria: è anch'esso un thread asincrono in continua attesa di dati dal socket aperto.

In particolare l'oggetto Connessione è in grado di gestire un time-out dovuto all'inutilizzo del socket dati: se per un determinato lasso di tempo (scelto dall'utente) non si riceve e non si invia nessuna informazione, l'oggetto connessione chiude il socket disconnettendo il client remoto (sarà il server che successivamente si accorgerà di questa inattività ed eliminerà l'oggetto Connessione dalla memoria).

Quando la Connessione deve invocare l'elaborazione delle immagini da parte di FEDRO, entra in uno stato d'attesa; il Server nel controllare le Connessioni presenti prenderà atto di questo suo stato e deciderà se inizializzare oppure no l'elaborazione di FEDRO. Viceversa quando il Server propaga i nuovi dati ottenuti da FEDRO alle singole Connessioni, queste decidono se processarli oppure no a seconda se ne erano in attesa o no.

La gestione dei dati ottenuti dal Server o di quelli arrivati via socket dal client remoto non è implementata nella classe Connessione ma in quella Protocollo: ogni oggetto Connessione, quando viene creato, attiva anche un oggetto di tipo Protocollo.

I parametri di configurazione delle Connessioni sono:

- Il tempo di pausa del thread (in millisecondi).
- Il tempo di time-out della connessione (in secondi), se negativo il controllo del time-out è disabilitato.
- La dimensione massima del buffer dati, ovvero il numero massimo di caratteri alfanumerici che possono essere letti in una sola passata dal socket TCP/IP (valore consigliato è 512).

4.3. Protocollo.h

La classe Protocollo implementa il protocollo di comunicazione descritto in precedenza nell'introduzione. Di fatto un oggetto Protocollo riceve come input i dati pervenuti via socket e/o quelli ottenuti dall'elaborazione di FEDRO; in base tutte queste informazioni decide:

Quali messaggi spedire al client remoto.

- Se chiudere o no la Connessione a cui è legato.
- Se attivare o no l'elaborazione di FEDRO.

Lo stato d'avanzamento del protocollo è mantenuto attraverso una variabile numerica che indica lo stato corrente dell'oggetto; gli stati possibili sono:

- Stato zero: in attesa del messaggio *"where am I"* dal client remoto.
- Stato uno: in attesa dei risultati dell'elaborazione di FEDRO.
- Stato due: in attesa del messaggio *"give me"* dal client remoto.

Nella figura *"Figura 4.3"* viene mostrata la logica utilizzata per l'implementazione del protocollo.

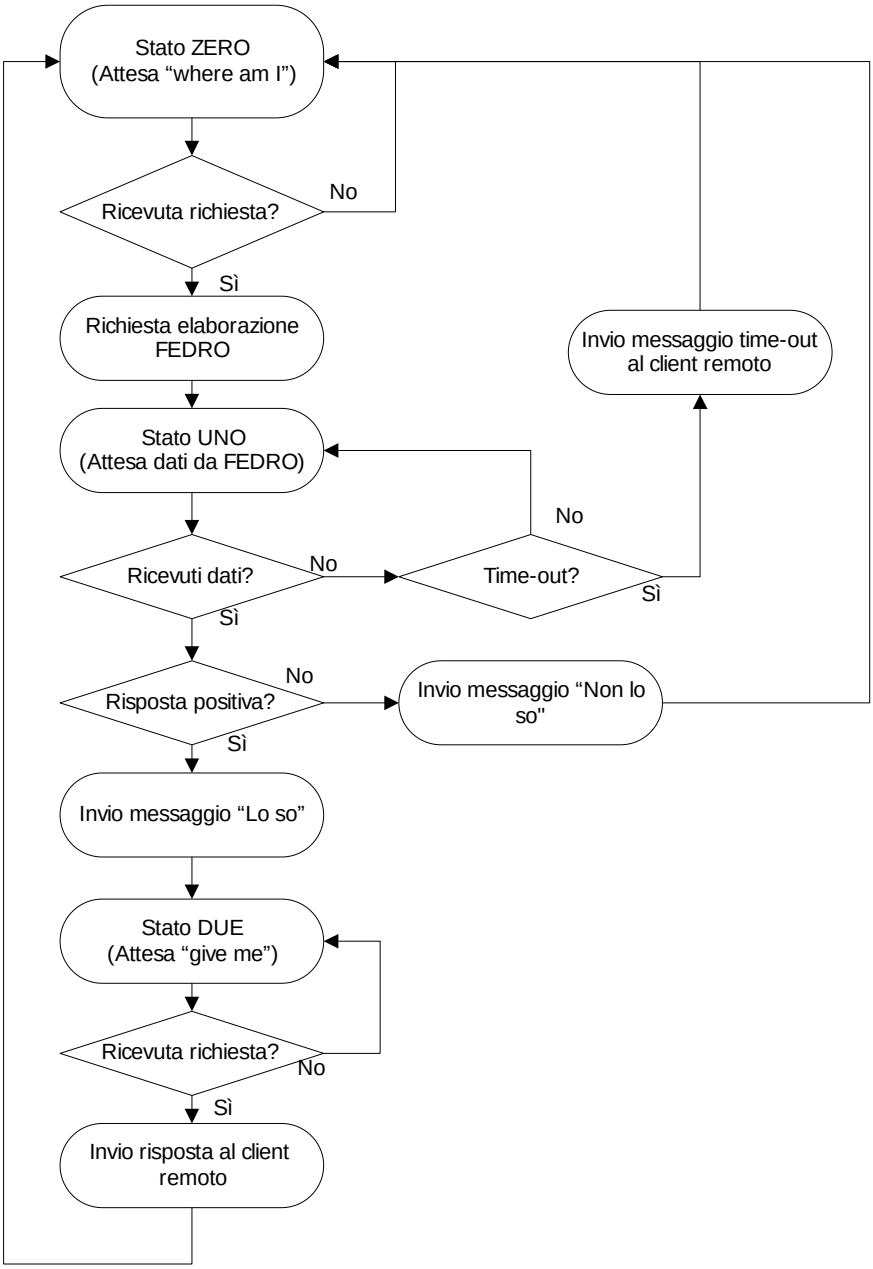


Figura 4.3: logica utilizzata per l’implementazione del protocollo.

L’oggetto Protocollo implementa anche un controllo sul time-out dovuto all’eccessiva attesa dei risultati elaborati da FEDRO, in pratica se dal momento della richiesta di elaborazione (fatta direttamente alla Connessione, ed indirettamente al Server e quindi a FEDRO) passa un lasso di tempo maggiore di quello consentito (valore impostato dall’utente) allora il Protocollo cambia stato (dall’uno allo zero) segnalando al client remoto l’avvenuto time-out.

Il controllo sui messaggi ricevuti e la formattazione dei messaggi da spedire è implementato attraverso l’uso di file xml. Ogni messaggio ricevuto dal client remoto viene confrontato con un template salvato su file xml (il template corretto, e quindi il file da caricare, è deciso a seconda dello stato corrente del protocollo); ogni messaggio da spedire viene caricato da file, eventualmente modificato inserendovi le parti variabili e poi spedito. L’eventuale modifica del file xml da spedire avviene ricercando nello stesso file la posizione corretta nella quale inserire i dati richiesti: di fatto viene ricercata una particolare parola racchiusa tra parentesi quadre, a questa parola verrà sostituito il corretto valore da spedire.

Per esempio il file “/xml/Dati(SF).xml” contiene:

./xml/Dati(SF).xml

```
<?xml version="1.0" encoding="ISO-8859-1"?><server_data><blob><coord><x>[x]</x><y>[y]</y></coord><max_value>[max_value]</max_value><num_max_value>[num_max_value]</num_max_value><average_value>[average_value]</average_value><num_tot>[num_tot]</num_tot></blob></server_data>
```

Come si può vedere i valori contenuti in certi tag sono racchiusi tra parentesi quadre, il sistema li ricerca e li sostituisce con gli opportuni valori calcolati. Una possibile trasformazione può essere per esempio:

```
<?xml version="1.0" encoding="ISO-8859-1"?><server_data><blob><coord><x>256</x><y>310</y></coord><max_value>25</max_value><num_max_value>18</num_max_value><average_value>0.9536</average_value><num_tot>13</num_tot></blob></server_data>
```

Il messaggio così rielaborato verrà poi spedito al client remoto.

Il parametro di configurazione del protocollo è:

- il numero di secondi dopo i quali generare il timeout per un’eccessiva attesa dei risultati da FEDRO.

4.4. ScambioDati.h

La classe ScambioDati è stata pensata per permettere il passaggio di dati dall’applicativo FEDRO al nostro progetto senza richiedere una riscrittura pesante del codice di FEDRO; questa è l’unica classe condivisa dai due applicativi che di fatto ne utilizzavano la medesima istanza assegnata nel main del programma. Quanto detto si può apprezzare nello spaccato di file sorgente mostrato di seguito:

Start.cpp

```
…
int main()
{
	…
	/*Creata l’istanza di ScambioDati: scambio*/
	ScambioDati *scambio = new ScambioDati();

	/*scambio è passato al costruttore di Server*/
	Server server(…,scambio);

	/*scambio è passato al costruttore di FEDRO*/
```

Questa classe è molto semplice, contiene unicamente (come variabili pubbliche) tutte quelle informazioni elaborate da FEDRO che devono essere scambiate via TCP/IP col protocollo precedentemente descritto:

- Coordinata x nell'immagine.
- Coordinata y nell'immagine.
- Il valore massimo calcolato.
- Il numero di pixel con valore massimo.
- Il numero totale di pixel del blob.
- Il valore medio dei pixel.
- Codice di un eventuale errore rilevato.

Oltre a queste informazioni, nella classe ScambioDati sono presenti altre due variabili necessarie per il corretto scambio dei dati tra i due applicativi. Infatti essendo i due processi asincroni è stato necessario implementare un meccanismo capace di regolare l'utilizzo dei dati condivisi e la richiesta di elaborazione di FEDRO delle immagini (una richiesta remota di localizzazione ricevuta dal server TCP/IP non prevede la chiamata di una funzione del programma di FEDRO, questo avrebbe implicato una dipendenza tra i due processi, ma semplicemente prevede la valorizzazione di una certa variabile condivisa).

La prima variabile è 'elaborazione': può assumere solo i valori 0 (false) ed 1 (true) ed indica a FEDRO se iniziare oppure no la gestione delle immagini catturate dalla webcam; 'elaborazione' viene impostata ad 1 dall'oggetto Server dopo aver verificato la corretta richiesta remota di localizzazione secondo il protocollo stabilito; ad ogni nuova richiesta successiva se la variabile è già ad 1 non viene toccata (questo permette di implementare solo un'unica attivazione di FEDRO anche a fronte di più richieste). Questa variabile viene posta a 0 solo da FEDRO quando ha già concluso la sua elaborazione.

Di seguito è mostrata la logica adottata dal server TCP/IP per invocare FEDRO ad ogni nuova richiesta remota di localizzazione.

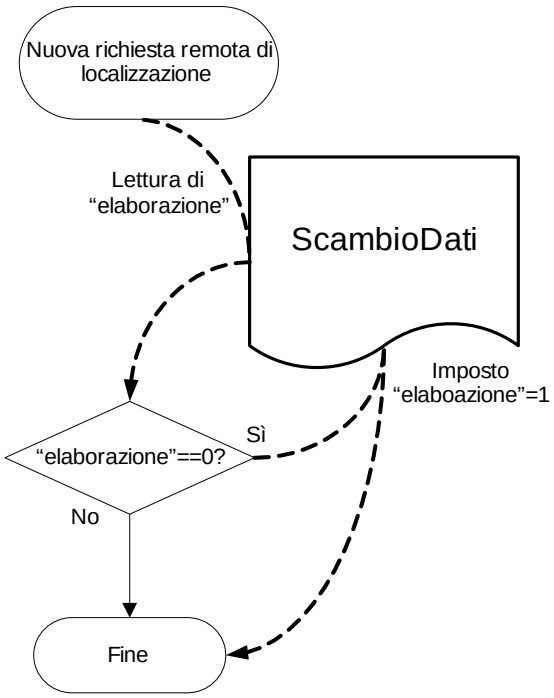


Figura 4.4: logica adottata dal server TCP/IP per invocare FEDRO.

L'altra variabile è 'timeStamp': indica l'istante di tempo in cui FEDRO ha terminato la sua elaborazione delle immagini ed ha memorizzato in ScambioDati i risultati ottenuti. Questa variabile viene sempre e

solo impostata da FEDRO; il server TCP/IP semplicemente vi accede per controllare la validità dei dati (ricordo che i dati elaborati sono ritenuti validi per un certo periodo di tempo impostato dall'utente: vedi par. 4.1 Server.h).

Per concludere la classe ScambioDati contiene anche la definizione di alcuni tipi di errori d'elaborazione riscontrati da FEDRO:

- ERRNO : nessun errore rilevato.
- ERRSI : è avvenuto un errore generico non identificabile.
- ERRNOTFOUND: elaborazione terminata, ma non è stato ottenuto un risultato positivo.
- ERRWEBCAM: elaborazione non terminata perché si è persa la connessione con la webcam.
- ERRSINCWEBCAM: elaborazione non terminata perché si è persa il sincronismo con la webcam.

5. Modalità operative

Questo paragrafo descrivere, in maniera completamente esauriente, le modalità necessarie per utilizzare il software realizzato. Permette cioè ad un utente, anche relativamente inesperto, di essere in grado di verificare in modo autonomo che quanto è stato realizzato funzioni.

5.1. Componenti necessari

Per far funzionare correttamente il software ServerFedro è necessario disporre di:

- Un robot mobile dotato di sonar, odometria, telecamera. Il software è stato sviluppato per robot di tipo Activmedia Pioneer 3, ed in particolare per il robot *Morgul* realizzato presso la facoltà di Ingegneria di Brescia. Il robot deve essere dotato di marker attivo costituito da led ad alta intensità.
- Librerie Aria 2.3 release 3, scaricabili dal sito www.activmedia.com.
- GCC 3.4.1 per la compilazione del software.
- S.O. Linux.

5.2. Modalità di installazione

Per l'installazione delle librerie Aria si rimanda il lettore all'apposita guida; per l'installazione del software *"ServerFedro"* è semplicemente necessario estrarre il file formato zip fornito all'utente nella cartella desiderata. Per far funzionare il programma è necessario la sua compilazione, per la quale è possibile sfruttare uno script incluso nel pacchetto ServerFedro; i passi da eseguire sono:

a. Per estrarre l'archivio, digitare nella shell ($ indica il prompt):
   $ unzip marker.zip –d

a. Entrare nella cartella "ServerFedro"; a tal fine digitare:
   cd ./marker/ServerFedro

b. Compilare i file sorgente: poiché è necessario compilare insieme sia i file di ServerTCP/IP e Fedro, è già predisposto all'interno di questa cartella un Makefile che permette di compilare correttamente i file desiderati; a tal fine digitare:
   ./make

c. Il file eseguibile prodotto dalla compilazione si chiama "server_fedro", quindi per mettere in esecuzione il programma basta semplicemente scrivere:
   ./server_fedro

5.3. Esecuzione

Per far eseguire correttamente il programma una volta che è stato compilato, basta semplicemente scrivere nella shell lo script:

- **./marker/ServerFedro/server_fedro** , se ci si trova nella directory in cui è stato estratto l'intero pacchetto software,
- Oppure digitare **./server_fedro** se ci si trova già nella cartella ServerFedro.

Questo, ovviamente, vale solamente per quanto riguarda l'esecuzione del programma ServerFedro, il quale, ricordiamo:

1. Esegue il collegamento con la telecamera;
2. Si mette in ascolto sulla porta prestabilita;

Per far funzionare correttamente tutto il sistema, invece, occorre ricordare che il sistema completo è costituito da molteplici programmi, i quali devono essere eseguiti in un ordine ben stabilito:

1. Per prima cosa occorre far partire il programma *"ServerFedro"*; a questo scopo occorre portarsi nella cartella "ServerFedro" e scrivere lo script **./server_fedro**
2. In seguito occorre far partire il programma *"Black Box"*
3. A questo punto è possibile far partire l'esecuzione dei Client, siano essi esterni o il robot stesso.

- **IMPORTANTE: l'ordine di avvio dei software NON è casuale, ma deve essere quello indicato, altrimenti si generano dei problemi con l'esecuzione dei programmi connessi;**
- **Prima di mettere in esecuzione il programma, assicurarsi di disporre di una connessione Ethernet sul calcolatore che si sta utilizzando.**

5.4. Test

All'interno del pacchetto fornito all'utente sono inclusi anche delle cartelle che permettono di simulare il funzionamento dell'intero sistema di rilevazione; tali cartelle si trovano all'interno della cartella *"marker"* ottenuta estraendo l'archivio dal file *"marker.zip"* fornito all'utente, e sono:

- Cartella ./ServerFedro contenente appunto il programma ServerFedro;
- Cartella ./ServerFedro/client_test_SF che contiene una versione semplificata del programma che fa da Client al Server Fedro, ovvero che simula la richiesta da parte di un client esterno;
- Cartella ./TEST/BB che contiene una versione semplificata del programma Black Box;
- Cartella ./TEST/BB/client che contiene una versione semplificata del programma che fa da Client al server Black Box, ovvero che simula la richiesta da parte del robot;
- Cartella ./TEST/FIFO_terminal, che contiene un programma che simula la presenza ed il comportamento della telecamera.

Prima di far eseguire i programmi, è necessario farli compilare; quindi, collocandosi nella directory ove si è estratto il pacchetto software, digitare il seguente script:

./marker/ServerFedro/make

./marker/ServerFedro/client_test_SF/make

./marker/TEST/BB/make

./marker/TEST/BB/client/make

./marker/TEST/FIFO_terminal/make

A questo punto è possibile mettere in esecuzione i diversi programmi; poiché ogni programma scrive nella shell delle informazioni relative alle proprie operazioni, è necessario aprire per ogni programma una shell diversa:

./marker/ServerFedro/server_fedro

./marker/TEST/FIFO_terminal/ fifo_srv

./marker/TEST/BB/BlackBox

Con questi script si sono fatti partire in successione il programma che gestisce il ServerFedro, la connessione e l'invio dati con la telecamera ed il server Black Box; a questo punto, a piacimento, si possono far partire i diversi Client;

./marker/TEST/BB/client/clientSocketTCP

./marker/ServerFedro/client_test_SF/clientSocketTCP

- **IMPORTANTE: l'ordine di avvio dei software NON è casuale, ma deve essere quello indicato, altrimenti si generano dei problemi con le FIFO (tra programma di prova "fifo_srv"e "ServerFedro") e con l'esecuzione dei programmi connessi;**

- **IMPORTANTE: mentre il programma *"Black Box"*, e i diversi client possono essere eseguiti indipendentemente sullo stesso calcolatore o su calcolatori differenti, i programmi *"ServerFedro"* e *"fifo_srv"* devono necessariamente essere eseguiti sullo stesso calcolatore;**

- **Utilizzare il programma *"fifo_srv"* solamente nel caso in cui non si disponga di una telecamera collegata al calcolatore sul quale si vuole far eseguire il programma *"ServerFedro"*: in tal caso modificare il programma *"connessioneWebcam.cpp"* secondo le indicazioni reperite nella tesi di Fedrigotti.**

- **Prima di mettere in esecuzione i programmi, assicurarsi di disporre di una connessione Ethernet sui calcolatori che si stanno utilizzando.**

5.5. Avvertenze

- Nel momento in cui si decide quante possibili connessioni contemporanee far gestire al server, sarebbe bene valutare che un numero eccessivo di connessioni potrebbe rallentare eccessivamente l'esecuzione del programma, o causare quel minimo rallentamento che causerebbe la perdita del sincronismo con la telecamera. Occorre anche tenere conto della velocità attestata alla rete W-I utilizzata per la connessione tra robot *Morgul* e il calcolatore sul quale è in esecuzione il programma *"Black Box"*.
- E' importante sincerarsi della presenza delle corrette versioni delle librerie Aria e del compilatore GCC: questi componenti potrebbero infatti portare a complicazioni qualora non fossero della versione indicata
- Per il resto si consiglia di seguire con attenzione le avvertenze poste come commenti all'interno dei paragrafi precedenti.

6. Conclusioni e sviluppi futuri

Allo stato attuale delle cose il progetto Server Fedro risulta essere completato e funzionante. Tale progetto ha portato alla realizzazione di un software in grado di acquisire mediante telecamera la posizione di un robot dotato di marker attivo all'interno di un ambiente prestabilito, e di comunicare tale posizione a chiunque ne faccia richiesta, robot compreso. Il programma sviluppato, in particolare, ha le seguenti peculiarità:

- Realizza un server in grado di sostenere fino a 10 connessioni contemporanee; il numero massimo di configurazioni disponibili accettate può essere modificato a piacimento dall'utente

andando a modificare il parametro contenuto nei tag ***<max_connection>10</max_connection>*** contenuti nel file XML *"config.xml"* situato nella cartella ./ServerFedro/xml/;

- Gestisce l'interrogazione dei thread associati alle connessioni in maniera intelligente: per non appesantire troppo il ciclo di sistema, è predisposta all'interno del programma una variabile indicante quanto tempo deve passare prima che uno stesso thread venga acceduto nuovamente: in questo modo il thread viene eseguito qualche volta al secondo. Tale parametro può essere modificato a piacimento dall'utente andando a modificare il parametro contenuto nei tag ***<time_sleep>200</time_sleep>*** contenuti nel file XML *"config.xml"* situato nella cartella ./ServerFedro/xml/;
- Prevede un meccanismo secondo il quale, nel momento in cui un client fa la richiesta di localizzare il robot, l'elaborazione delle immagini (che porta via complessivamente un tempo stimabile attorno i 7/8 secondi) viene eseguita solamente se è passato un certo limite di tempo dall'ultima volta che le coordinate del robot sono state calcolate. Tale limite di tempo si può cambiare andando a modificare il parametro contenuto nei tag ***<time_val>1500</time_val>*** contenuti nel file XML *"config.xml"* situato nella cartella ./ServerFedro/xml/;
- All'interno del programma sono definite delle costanti: tali costanti sono memorizzate all'interno del file XML *"config.xml"* situato nella cartella ./ServerFedro/xml/, e vengono caricati nel programma nel momento in cui lo stesso viene avviato. Questa soluzione permette all'utente di gestire comodamente tali parametri di configurazione, modificandoli a piacimento con comodità, permettendogli di non dover ogni volta ricompilare tutto il programma;
- Permette di gestire il protocollo di scambio dati tra client e server in maniera completamente flessibile: è possibile infatti in un futuro andare a modificare i messaggi che i client e il server si scambiano senza dover andare a mettere mano al codice; è sufficiente modificare i file contenuti nella cartella ./ServerFedro/xml/
- Gestisce correttamente ogni tipologia d'errore che può verificarsi durante il funzionamento del programma stesso: ogni volta che viene fatta una richiesta di localizzazione e si verifica un intoppo, il server risponde, come da protocollo, con un messaggio <Non lo so> indicando, sia nel messaggio mandato al Client che tramite una stampa a video, la tipologia d'errore verificatosi.

Dal punto di vista di uno sviluppo futuro, si potrebbe sviluppare un sistema che si preoccupi di fornire al pacchetto una sorta di protezione/sicurezza; si potrebbe prevedere, per esempio, un sistema di:

- Autenticazione dei partecipanti (di facile realizzazione);
- Comunicazioni cifrate;
- ecc..

Glossario

Black Box: programma che viene messo in esecuzione sul calcolatore che si interpone tra il robot *Morgul* ed il calcolatore sul quale è in esecuzione il programma *"Server Fedro"*. Il suo compito è di

a. Mettersi in ascolto sulla porta TCP/IP sul quale si collegherà esclusivamente il robot *Morgul*;
b. Nel momento in cui arriva una richiesta di localizzazione da parte del robot, attivare una connessione con il programma *"Server Fedro"*;
c. Inviare la richiesta di localizzazione al server secondo le modalità previste dal protocollo di gestione dati;
d. Una volta ottenute le coordinate del robot nell'immagine, applicare l'algoritmo di conversione realizzato da Panteghini-Cagno per ricavare le coordinate reali della posizione del robot nell'ambiente in cui si trova;
e. Comunicare al robot le coordinate ottenute, od eventuali messaggi d'errore.

Fifo_srv: programma che permette di "ingannare" il programma *"ServerFedro"*, simulando la presenza della telecamera; il programma che gestisce l'acquisizione delle immagini dalla telecamera, infatti, non fa altro che creare una struttura dati FIFO e memorizzarci dentro, di volta in volta, le immagini provenienti dalla telecamera. Il programma *"fifo_srv"* non fa altro che andare a riempire la struttura FIFO sulla quale andrà poi ad operare l'algoritmo di elaborazione delle immagini (algoritmo che fa parte del programma ServerFedro).

Mappa .map: una mappa .map non è altro che la pianta di un ambiente reale, all'interno del quale si vuole far muovere il robot stesso. Le librerie Aria le utilizzano all'interno dei programmi quali Saphira, per poter conoscere dove il robot si muove e, in base agli spostamenti che il robot effettua, conoscere la sua posizione all'interno della mappa, previo il sapere DA DOVE inizia a muoversi il robot. L'estensione è dovuta al programma utilizzato per creare queste mappe.

Mobile eyes: è un programma che sfrutta le librerie Aria. Tramite questo programma, previo aver stabilito un contatto radio con il robot, permette di visualizzare in una finestra del desktop le immagini provenienti dalla telecamera collegata al robot in questione

Odometria: meccanismo tramite il quale è possibile rilevare e tener traccia dello spostamento effettuato da un robot. Data la posizione di partenza, il tipo di movimento e la mappa dell'ambiente, il robot tramite l'odometria è in grado di calcolare quale è la sua nuova posizione.

Più approfonditamente: Il problema della localizzazione di robot mobili, in ambienti non strutturati, rappresenta uno dei filoni di ricerca tra i più attivi ed importanti. Poter conoscere con sufficiente precisione il posizionamento del robot nel mondo reale costituisce un'esigenza imprescindibile per gli sviluppi futuri della robotica mobile. Il modo più semplice per rispondere alla domanda fondamentale nel problema della localizzazione, ovvero *"where am I?"***,** consiste nello sfruttamento di informazioni odometriche: mediante encoder calettati sui motori di trazione è possibile calcolare il numero di giri compiuti dalle ruote del robot in un lasso di tempo unitario e stimare quindi lo spazio percorso. Si parla in questo caso di localizzazione *dead-reckoning* basata su sensori propriocettivi (o idiotetici) in quanto la stima sulla posizione è ottenuta dalla sola analisi di informazioni sensoriali proprie del robot. L'odometria è tuttavia affetta da errori che possono facilmente causare nel tempo una deriva della posizione stimata del robot rispetto a quella reale con conseguente perdita totale dell'orientamento. Le principali componenti d'errore che determinano tali derive sono raggruppabili in due categorie distinte: 1) errori sistematici (anche detti deterministici): rientrano in questa categoria gli errori causati da disallineamento fra le ruote motrici, differenze fra diametro reale delle ruote e diametro utilizzato per la stima, limitata risoluzione dell'encoder utilizzato per la rilevazione di posizione. 2) errori non sistematici (anche detti non deterministici): rientrano in questa categoria gli errori causati ad esempio da movimento del robot su superfici non perfettamente piane, profili di velocità tali da determinare slittamento delle ruote, ecc..

Saphira: programma tramite il quale, una volta conosciuto il robot che si vuole utilizzare (o simulare) e l'ambiente in cui esso si muove (tramite il caricamento dell'apposita mappa), è possibile far muovere il robot o fargli eseguire delle operazioni (realmente o tramite simulazione).

Wander: è un programma d'esempio scritto utilizzando le librerie Aria, attraverso il quale è possibile far muovere il robot che si sta utilizzando all'interno di un ambiente del quale si dispone la mappa: il movimento che il robot attua è un movimento di tipo casuale, e fa in modo che ogni volta i sensori di presenza del robot captano un ostacolo, esso assuma il comportamento desiderato (la selezione del tipo di comportamento, del tipo di sensori da attivare, della velocità con cui si muove il robot, della distanza a cui si rileva un oggetto come ostacolo, ecc.. possono essere gestiti all'interno del programma wander mediante il settaggio di opportune variabili e la scrittura di opportune linee di codice).

Bibliografia

[1] ActivMedia Robotics, LLC: “ARIA (Activmedia Robotics Interface for Application) Reference Manual”, 2002.

[2] Archivio della mailing list ARIA-users (http://robots.activmedia.com/archives/ariausers/threads.html).

[3] ActivMedia Incorporated, “Pioneer 1 Gripper & Experimenter’s Module Manual”, 1997.

[4] Rossi, Viola: "Morguldoc 2: elaborazione di immagini per il riconoscimento di marker attivi", 2004 (http://www.ing.unibs.it/~cassinis/docs/projects/Mor_02.pdf)

[5] Manzini: "Controllore per marker attivo di robot mobile", 2005 (http://www.ing.unibs.it/~cassinis/docs/projects/Mor_04.pdf)

[6] Roberto Fedrigotti: "Utilizzo di marker ottici interattivi in un sistema distribuito su rete locale per la localizzazione di robot mobili", 2005

Indice