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

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Sistemazione controllo web di SAURON

Elaborato di esame di:

Emanuel Bonusi, Francesco De Rose, Nicola Ferrari

Consegnato il: **23 luglio 2007**

Sommario

Nel Laboratorio di Robotica Avanzata della facoltà è presente il robot Morgul, che fa parte del sistema SAURON. In questa relazione si descrive il progetto e l'implementazione di un sistema di controllo via web del robot Morgul, che fornisce tra le sue funzionalità fondamentali la gestione dell'accesso esclusivo al robot ed il rilevamento e la segnalazione all'utente delle condizioni di stallo.

1. Introduzione

Nella versione attualmente visitabile del sito vi è una sezione dedicata all'interazione e alla comunicazione da remoto con uno dei robot disponibili nel laboratorio di robotica: Morgul (Mobile Observator Robot for Guarding the University Laboratories). In questa pagina si possono impartire al robot, tramite la pressione di appositi pulsanti, dei piccoli compiti e contemporaneamente, grazie a due webcam, si può avere una visione in diretta su come Morgul stia eseguendo i compiti assegnati. Ad ogni compito, disponibile sulla pagina web, è associato uno script che verrà eseguito non appena il tasto relativo viene premuto.

L'elaborato da noi svolto prevede di aggiungere, all'interno della pagina web descritta in precedenza, una sezione relativa allo stato corrente di Morgul. Per esempio: se per qualche motivo durante l'esecuzione del lavoro assegnatogli il robot dovesse entrare in uno stato di stallo esso lo segnalerà tramite un messaggio testuale nella pagina dei comandi. In previsione di sviluppi possibili in futuro, abbiamo aggiunto, a corredo dello stato, degli ulteriori messaggi come per esempio il nome del robot che si sta comandando.

Oltre a ciò si è provveduto alla modifica delle applicazioni di controllo dei movimenti del robot implementati negli anni precedenti al nostro. In questa attività, abbiamo dovuto introdurre per forza di cose una breve procedura per intercettare lo stato di stallo del robot. Accennando brevemente al suo funzionamento, la procedura monitora costantemente la posizione corrente di Morgul e se questa è sempre la stessa per un certo numero di volte, attiverà il messaggio di allarme da visualizzare.

2. Il problema affrontato

Riducendo all'osso le specifiche di progetto abbiamo evidenziato e definito i problemi che influenzano lo sviluppo dell'elaborato. L'obiettivo principale da raggiungere è la gestione tramite un'interfaccia web della comunicazione fra due interlocutori: il primo umano che ha a disposizione un qualsiasi browser internet, mentre il secondo, Morgul.

Il secondo problema affrontato è stato quello di impartire dei comandi a Morgul da una postazione remota rispetto al laboratorio in modo da poterlo interrogare sul proprio stato operativo e sulle proprie condizioni di errore.

Perché sia possibile instaurare un dialogo con il robot abbiamo già accennato al fatto che l'interlocutore umano debba disporre di un browser web ma questa non è l'unica condizione a cui esso deve sottostare. Infatti abbiamo deciso di garantire un minimo grado di protezione e sicurezza sia nei confronti di Morgul che in quelli delle persone che abitualmente frequentano il laboratorio. Si è deciso che un'utente possa comandare il robot a patto che esso sia autorizzato, tramite autenticazione basata su username/password. Queste ultime possono essere richieste tramite e-mail all'amministratore del sito, che deciderà se accettare o meno la proposta di iscrizione.

La nostra applicazione web rende disponibile un'unica risorsa, in questo caso il robot del laboratorio, ad un numero, teoricamente vasto di possibili interlocutori. Detto questo è facile immaginare quali possano essere le conseguenze se due o più utenti, anche se preventivamente autorizzati manovrino

contemporaneamente il robot. Onde evitare questo spiacevole e pericoloso scenario si è implementato un sistema di lock esclusivo in modo tale che solo una persona alla volta possa disporre dell'unica risorsa condivisa.

Trovati i problemi abbiamo diviso il gruppo in due parti in modo da procedere parallelamente nella risoluzione degli stessi. Le strategie risolutive sono descritte nei capitoli successivi.

3. La soluzione adottata

3.1. Architettura generale del sistema

Il sistema di controllo a distanza di Morgul che si è progettato ed implementato è basato su un'architettura a livelli. In questo modo si è anzitutto ottenuta una decomposizione abbastanza naturale del problema, che ha permesso di suddividere agevolmente il lavoro tra i componenti del gruppo; inoltre ciò ha permesso di integrare senza eccessive difficoltà il nuovo sistema con quello preesistente e, in futuro, permetterà la facile espandibilità.

Per fornire una visione generale dell'architettura del sistema è bene analizzare un esempio che permetta di descriverne concretamente i componenti. Supponiamo che un utente remoto e autorizzato decida di impartire un comando a Morgul: per fissare le idee, sia il comando di accensione del microcontrollore del robot.

L'utente quindi utilizzerà il browser web di un qualsiasi calcolatore collegato ad internet per richiedere la pagina http://frost.ing.unibs.it/protected/fdr/turnOn.php (tipicamente arriverà a richiedere la pagina attraverso un collegamento presente nella pagina principale del sistema di controllo, ma per semplicità si supponga ora che la richiesta si diretta).

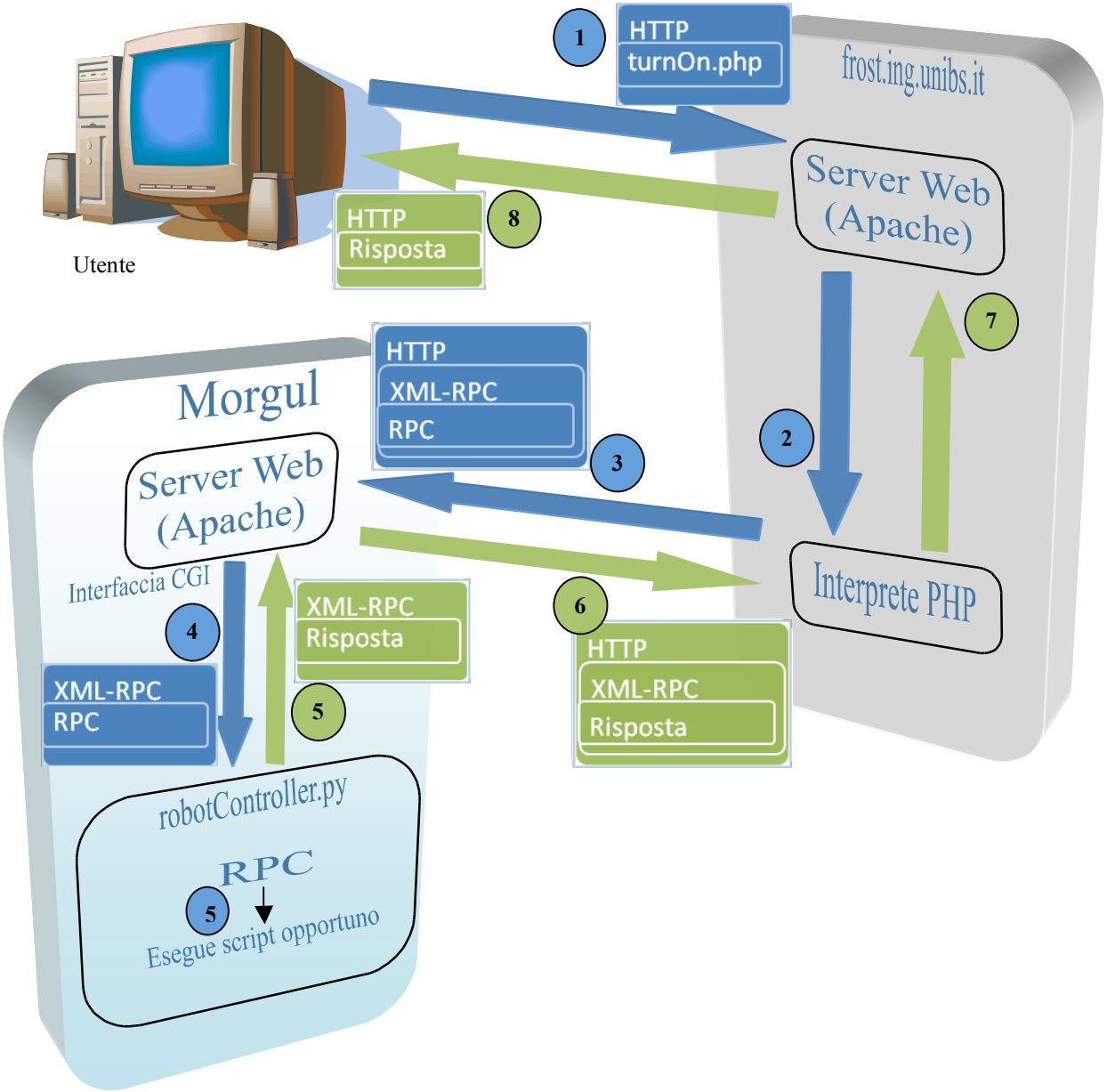
Tale richiesta è servita da Frost, calcolatore che assolve, tra l'altro, alla funzione di server web per il laboratorio. La pagina in questione è scritta utilizzando il linguaggio PHP; il server web di Frost pertanto invocherà, al momento di servire la richiesta, l'interprete PHP affinché esegua il codice contenuto nella pagina.

Il codice PHP che compone la pagina contiene al suo interno una chiamata a procedura remota (RPC – *Remote Procedure Call*). Per la precisione, si tratta di una chiamata a procedura remota che sfrutta il protocollo XML-RPC e che invoca su Morgul la procedura di nome turnOn.

Poiché lo standard XML-RPC prevede che il trasporto sia effettuato su protocollo HTTP, su Morgul è stato installato il server web Apache. Esso riceve la richiesta HTTP, ne estrae il contenuto XML-RPC (che non saprebbe altrimenti interpretare) e lo inoltra, attraverso l'interfaccia CGI, all'applicazione robotController.

L'applicazione robotController esamina il contenuto della richiesta XML-RPC, estrae il nome della procedura (in questo caso turnOn) e la esegue. La procedura turnOn (in realtà è un metodo, poiché si tratta di un'applicazione ad oggetti) invoca in modo non bloccante lo script di shell onrobot.sh. Essa inoltre restituisce un valore (in questo caso specifico una stringa che contiene il messaggio che il robot è in fase di accensione) il quale viene impacchettato in una risposta XML-RPC e restituito al server web di Morgul. Il server web di Morgul può a questo punto rispondere alla richiesta XML-RPC che gli era stata inviata dall'interprete PHP in esecuzione su Frost; quest'ultimo a sua volta può usare la risposta ottenuta (che in questo caso contiene la stringa di testo) per confezionare una pagina HTML che restituisce al server web in esecuzione su Frost, il quale, a sua volta, la inoltra al calcolatore dell'utente.

Nel frattempo, poiché la chiamata è non bloccante, su Morgul viene eseguito lo script onrobot.sh. Esso effettua varie operazioni: dispone la riproduzione di un avviso acustico in laboratorio (tramite un'opportuna richiesta CGI verso Frost), registra che il robot è stato acceso creando, su Morgul, il file onrobot e, infine, invoca il programma di gestione del microcontrollore del robot richiedendo l'accensione.



3.2. Server web su Frost

Su Frost, il calcolatore che fornisce il servizio web per il laboratorio, era già installato il demone Apache, unitamente al modulo per l'interpretazione delle pagine PHP. Si sono creati, nella directory /Library/WebServer/Documents/protected/fdr, i file che vengono descritti nelle sezioni immediatamente seguenti. Tutta la directory è protetta, con il metodo del file htaccess, in modo che soltanto gli utenti autorizzati, i quali conoscono nome utente e password, possono accedere ai servizi. Il punto iniziale di accesso ai servizi di controllo via web del robot è nella pagina framemain.php.

3.2.1. xmlrpc.inc

Questo file contiene le funzioni di libreria PHP che implementano un client XML-RPC: per questo motivo viene incluso, mediante l'opportuna direttiva PHP, da tutte le pagine che necessitano di effettuare richieste XML-RPC. Il file è tratto dal progetto "XML-RPC for PHP" [3].

3.2.2. connect.php

Questo file contiene il codice PHP che inizializza la connessione verso il server XML-RPC su Morgul: per questo motivo viene incluso, mediante l'opportuna direttiva PHP, da tutte le pagine che necessitano di effettuare richieste XML-RPC. La connessione è inizializzata verso la macchina herbie.ing.unibs.it sulla porta 8081 per ragioni di configurazione della rete del laboratorio: questa destinazione è in realtà traslata mediante NAT (Network Address Translation) verso la porta 80 di Morgul.

3.2.3. framemain.php

Si tratta della pagina principale a cui accedono gli utenti del sistema. Essa si limita soltanto a definire due frame orizzontali: in quello superiore viene caricata la pagina video.php, mentre in quello inferiore viene caricata la pagina getStatus.php.

3.2.4. video.php

Questa pagina, che viene tipicamente caricata nel frame supriore definito da framemain.php, contiene i tag necessari per il caricamento delle due applet Java Evocam che mostrano lo stream video trasmesso dalle due webcam installate in laboratorio e che monitorano l'area di lavoro del robot.

3.2.5. redirectHome.php

Questa pagina contiene il tag meta di html che ridirige il browser dell'utente verso getStatus.php dopo un secondo di tempo. Essa è inclusa da tutte le pagine di risposta alle richieste inviate al robot, in modo che all'utente sia riproposta la visualizzazione dello stato del robot.

3.2.6. getstatus.php

Questa pagina contiene l'invocazione XML-RPC della procedura getStatus su Morgul; con le invocazioni ottenute in risposta a questa chiamata costruisce una tabella che riassume all'utente lo stato del robot.

La pagina inoltre contiene i collegamenti ipertestuali a tutte le altre pagine che controllano le altre funzioni del robot.

3.2.7. clearStall.php

Questa pagina invoca tramite XML-RPC la procedura clearStall (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.2.8. flash.php

Questa pagina invoca tramite XML-RPC la procedura flashHeadlight (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.2.9. headlightsOff.php

Questa pagina invoca tramite XML-RPC la procedura turnOffHeadlights (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.2.10. lockRobot.php

Questa pagina invoca tramite XML-RPC la procedura lock (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.2.11. park.php

Questa pagina invoca tramite XML-RPC la procedura park (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.2.12. shortPatrol.php

Questa pagina invoca tramite XML-RPC la procedura shortPatrol (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.2.13. turnOff.php

Questa pagina invoca tramite XML-RPC la procedura turnOffRobot (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.2.14. turnOn.php

Questa pagina invoca tramite XML-RPC la procedura turnOnRobot (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.2.15. unlockRobot.php

Questa pagina invoca tramite XML-RPC la procedura unlock (descritta successivamente nella relativa sezione) su Morgul, mostrando all'utente un breve messaggio che contiene il valore restituito dalla procedura.

3.3. Server XML-RPC su Morgul

Il cuore del sistema è costituito dal server XML-RPC di Morgul. Esso riceve le richieste (che per ora provengono esclusivamente dal server web di Frost, ma in estensioni future potrebbero anche venire da altri calcolatori) ed esegue le azioni opportune per soddisfarle, restituendo informazioni circa lo stato del robot.

La scelta del protocollo XML-RPC rispetto a soluzioni analoghe quali CORBA o SOAP presenta numerosi vantaggi. Anzitutto per il trasporto esso utilizza HTTP, pertanto non presenta particolari problemi nell'interazione con firewall ed altri apparati di rete; le richieste sono espresse in formato XML, garantendo quindi ampia interoperabilità e la possibilità per un umano di leggere e comprendere le richieste in caso di necessità di debug. Inoltre si tratta di uno standard aperto e tutto sommato semplice, ma ciononostante dalle rilevanti potenzialità espressive: permette, ad esempio, il passaggio e la restituzione, oltre che di tipi di dato primitivo, di strutture dati moderatamente complesse quali stringhe, vettori, o array associativi (questi ultimi, chiamati anche dizionari in alcuni linguaggi di programmazione, sono insiemi di coppie chiave-valore).

Non ultimo per importanza, sono disponibili liberamente numerose librerie ed esempi che rendono molto semplice la scrittura tanto di client quanto di server nei più diffusi linguaggi di programmazione, quali C, C++, Python, PHP.

Nel caso specifico, il nostro client è scritto nel linguaggio Python in quanto esso è particolarmente adatto ai compiti di integrazioni e amministrazione di sistemi ed è orientato alla prototipazione rapida. Esso è costituito dal file /home/sauronweb/public/cgi-bin/robotController.py su Morgul; si tratta infatti di un programma che viene invocato tramite interfaccia CGI dal server web di Morgul al momento dell'arrivo di una richiesta. In realtà le librerie Python per la gestione di XML-RPC sono molto potenti e permettono con sforzo minimo di trasformare robotController.py da applicazione CGI a demone autosufficiente, vale a dire che implementa autonomamente il protocollo HTTP per la parte relativa alla gestione di XML-RPC; qualora in futuro dovesse sorgere questa necessità si consiglia vivamente di consultare la documentazione su [2], prestando particolare attenzione agli esempi.

Il sorgente dell'applicazione robotController si compone di quattro parti principali: l'importazione delle librerie e la definizione delle costanti stringa che contengono vari percorsi del filesystem, la definizione della classe robotInterface, la definizione del metodo invoke, l'inizializzazione del server XML-RPC vero e proprio.

La parte fondamentale dell'applicazione è la classe RobotInterface. Tutti i metodi pubblici da essa definiti sono resi disponibili col proprio nome ai client: pertanto, per far sì che Morgul sia in grado a rispondere ad una nuova invocazione XML-RPC, è sufficiente definire in RobotInterface un opportuno metodo con quel nome. Tipicamente i metodi richiameranno degli script di shell (oppure direttamente dei programmi).

Al fine di evitare perniciosi problemi di timeout e di dare la massima reattività all'interfaccia utente, è fortemente suggerito che l'eventuale invocazione di altri programmi o script da parte dei metodi della classe RobotInterface sia non bloccante, vale a dire che il metodo non attenda il termine del processo figlio generato. Per comodità dello sviluppatore si è definita la funzione invoke, che permette appunto di invocare in modo non bloccante un programma esterno dall'interno di RobotController.

Le ultime righe di codice inizializzano il server XML-RPC, registrando la classe RobotInterface, e lo avviano, permettendo l'elaborazione della richiesta CGI.

- **L'istruzione handler.register_instance(RobotInterface()) nella parte di inizializzazione del server XML-RPC registra la classe RobotInterface, in altre parole dice al server XML-RPC di esporre ai client esterni i metodi della stessa. Soltanto una classe può essere registrata col server XML-RPC; nel caso si cerchi di registrare più classi, soltanto l'ultima sarà funzionante.**

Nei paragrafi successivi si esamineranno i metodi della classe RobotInterface, che pertanto sono pubblicati dal server XML-RPC.

3.3.1. getMsg

Questa funzione attualmente restituisce sempre la stringa “Computo ergo sum”. In versioni successive del sistema sarà possibile modificarla in modo che restituisca un breve messaggio da comunicare all'utente circa lo stato del robot.

3.3.2. isLocked

Questa funzione controlla l'esistenza o meno del file /home/sauronweb/status/roboLock, per verificare se il robot è in stato di lock (cioè in uso) oppure no. Essa restituisce 1 nel primo caso, 0 nel secondo.

3.3.3. lock

Questa funzione crea il file /home/sauronweb/status/roboLock, a significare che il robot è in stato di lock (cioè in uso). Essa restituisce la stringa “locking robot”.

3.3.4. unlock

Questa funzione elimina, se esiste, il file /home/sauronweb/status/roboLock, a significare che il robot non è in stato di lock (cioè non è in uso). Essa restituisce la stringa “unlocking robot”.

- **Questa funzione non effettua alcun controllo di sicurezza per verificare che l'utente che richiede lo sblocco del robot sia autorizzato: essa non verifica neppure che l'utente che richiede l'operazione sia il medesimo che aveva acquisito il lock. È responsabilità del client XML-RPC che invoca la procedura assicurarsi che sia sicuro farlo.**

3.3.5. getOdometryPosition

Questa funzione legge il file /home/sauronweb/status/odometryPosition, per ottenere la posizione stimata del robot a partire dai dati di odometria, e la restituisce al chiamante. Si noti pertanto che tale valore è corretto ed aggiornato solo se un altro processo (tipicamente quello che controlla i movimenti del robot) scrive informazioni aggiornate in questo file.

3.3.6. getPhotoStatus

Questa funzione controlla se la webcam del robot è in uso oppure no, restituendo 1 nel primo caso, 0 altrimenti. La webcam è considerata in uso se nel sistema è in esecuzione un processo il cui nome corrisponde all'espressione regolare “fotografa”.

3.3.7. isRobotOn

Questa funzione restituisce il contenuto del file /home/sauronweb/status/onRobot, per verificare se il microcontrollore del robot è acceso oppure no. Tale file contiene il valore 1 se il microcontrollore è acceso, e 0 altrimenti.

3.3.8. isStalled

Questa funzione controlla l'esistenza o meno del file /home/sauronweb/status/stalled, per verificare se il robot ha incontrato uno stallo oppure no, restituendo 1 nel primo caso, 0 nel secondo. Si noti pertanto che tale valore è corretto ed aggiornato solo se un altro processo (tipicamente quello che controlla i movimenti del robot) crea questo file in caso di stallo.

3.3.9. clearStall

Questa funzione elimina, se esiste, il file /home/sauronweb/status/stalled, a significare che l'utente ha preso conoscenza della condizione d'errore. Essa restituisce la stringa “Clearing stall error”.

3.3.10. getStatus

Questa funzione restituisce un insieme di coppie chiave-valore che riassumono lo stato del robot. In particolare, le coppie chiave-valore sono:

chiave	**descrizione del valore**
robotname	il nome del robot come è restuituito dallo stack TCP/IP del sistema operativo
islocked	il valore restituito dalla funzione isLocked
message	il valore restituito dalla funzione getMsg
fotografa	il valore restituito dalla funzione getPhotoStatus
odometryPosition	il valore restituito dalla funzione getOdometryPosition

robotOn	il valore restituito dalla funzione isRobotOn
stalled	il valore restituito dalla funzione isStalled

3.3.11. turnOnRobot

Questa funzione invoca in modo non bloccante lo script /home/sauronweb/scripts/onrobot.sh al fine di accendere il microcontrollore del robot. Essa restituisce la stringa “Turning on robot controller”.

3.3.12. turnOffRobot

Questa funzione invoca in modo non bloccante lo script /home/sauronweb/scripts/offrobot.sh al fine di spegnere il microcontrollore del robot. Essa restituisce la stringa “Turning off robot controller”.

3.3.13. flashHeadlights

Questa funzione invoca in modo non bloccante il programma morgulc con gli opportuni parametri al fine di lampeggiare i fari frontali del robot. Essa restituisce la stringa “Flashing...”.

3.3.14. turnOnHeadlights

Questa funzione invoca in modo non bloccante il programma morgulc con gli opportuni parametri al fine di accendere i fari frontali del robot. Essa restituisce la stringa “Turning on headlights...”.

3.3.15. turnOffHeadlights

Questa funzione invoca in modo non bloccante il programma morgulc con gli opportuni parametri al fine di spegnere i fari frontali del robot. Essa restituisce la stringa “Turning on headlights...”.

3.3.16. shortPatrol

Questa funzione fa compiere al robot un breve giro di pattuglia del laboratorio. Essa anzitutto controlla se il robot è in lock: se questo è il caso restituisce la stringa “Cannot move a locked robot” terminando senza fare null'altro.

Altrimenti essa acquisisce il lock sul robot mediante la chiamata alla funzione lock e invoca in modo non bloccante lo script /home/sauronweb/scripts/patrol.sh restituendo la stringa “Going for patrol”.

Si noti che, poiché la chiamata è non bloccante, è responsabilità dello script invocato quella di rilasciare il lock al termine dei movimenti del robot.

3.3.17. park

Questa funzione fa rientrare il robot nella sua docking station a partire da una posizione in cui sono visibili nella webcam le relative luci di allineamento. Essa anzitutto controlla se il robot è in lock: se questo è il caso restituisce la stringa “Cannot move a locked robot” terminando senza fare null'altro.

Altrimenti essa acquisisce il lock sul robot mediante la chiamata alla funzione lock e invoca in modo non bloccante lo script /home/sauronweb/scripts/goHome.sh restituendo la stringa “Going home”.

- **È responsabilità dell'utente accertarsi, prima di invocare la funzione, che le luci della docking station siano visibili e che l'area di manovra sia sgombra. Si noti che, nella fase di parcheggio, nel robot viene inibito il comportamento di evitare le collisioni.**

3.4. Script su Morgul

Gli script di shell su Morgul sono raccolti nella directory /home/sauronweb/scripts. Essi costituiscono un utile livello di indirettezza tra il server XML-RPC e i programmi di controllo del movimento del robot,

poiché, assolvendo agli aspetti amministrativi (ad esempio il rilascio del lock al termine dei movimenti), permette a questi ultimi di concentrarsi sulla logica di movimento del robot.

Nei paragrafi successivi si esamineranno in dettaglio questi script.

3.4.1. fotografa.sh

Questo script esegue, periodicamente e con un intervallo in secondi specificato da linea di comando, una cattura dell'immagine ripresa dalla webcam, salvandola nella directory /home/sauronweb/patrolPics (dopo che la stessa è stata svuotata) con un numero progressivo.

3.4.2. goHome.sh

Questo script accende il microcontrollore del robot, spegne i fari frontali, uccide tutti i processi che utilizzano la webcam ed avvia i due processi che permettono di portare il robot nella docking station.

3.4.3. inviaFoto.sh

Questo script trasferisce, tramite il protocollo ftp, le immagini memorizzate nella directory /home/sauronweb/patrolPics su Frost per la pubblicazione sul sito web.

3.4.4. onrobot.sh

Questo script esegue il programma morgulc con gli opportuni parametri per accendere il microcontrollore del robot ed i fari frontali; esso inoltre scrive il valore 1 in /home/sauronweb/status/onRobot e comanda la riproduzione di alcuni avvisi audio in laboratorio.

3.4.5. offrobot.sh

Questo script esegue il programma morgulc con gli opportuni parametri per spegnere il microcontrollore del robot ed i fari frontali; esso inoltre scrive il valore 0 in /home/sauronweb/status/onRobot e comanda la riproduzione di un avviso audio in laboratorio.

3.4.6. patrol.sh

Questo script fa compiere al robot un breve giro di pattuglia del laboratorio. Esso anzitutto richiama onrobot.sh, poi avvia in background fotografa.sh, svuota su Frost la directory che contiene le immagini del giro precedente ed invoca il programma /home/sauronweb/programs/passeggiaNew.

Al termine esso trasferisce le fotografie appena scattate invocando inviaFoto.sh, riporta il robot nella docking station tramite goHome.sh, spegne il robot con offrobot.sh ed infine rilascia il lock tramite unlock.sh.

3.4.7. patrollong.sh

Questo script è analogo a patrol.sh, con la differenza che esegue il programma /home/sauronweb/programs/passeggiaNew per fare compiere dei movimenti più lunghi al robot.

3.5. Mantenimento dello stato

Al fine di mantenere lo stato dei processi e permettere loro di comunicare per scambiarsi informazioni, si è deciso di adottare una soluzione “a file”. Ogni processo quindi, per comunicare agli altri il proprio stato (ad esempio processo in uso e quindi robot in lock) o delle informazioni utili (posizione del robot piuttosto che data dell’ultima pattuglia), scrive su un apposito file che verrà successivamente letto dal processo destinatario.

Abbiamo quindi creato un’apposita directory: /home/sauronweb/status/ su Morgul, nella quale verranno creati i file.

L’elenco dei file presenti nella directory (e relativa funzionalità) è il seguente:

- ErrorMessage: file nel quale vengono scritti gli errori riscontrati in modo dettagliato. Attualmente non è implementato, ma è presente per comodità di espansione nelle versioni future.
- lastPatrol: file la cui data di ultima modifica viene mantenuta sincronizzata con la data dell'ultima esecuzione di patrol (script che ha lo scopo principale di richiamare il programma che esegue la pattuglia).
- odometryPosition: file nel quale viene scritta la posizione attuale di Morgul. Quest'ultima è una terna (x,y,θ), che viene aggiornata automaticamente dal programma e viene letta da robotController per essere poi presentata all'utente attraverso l'interfaccia web.
- onRobot: file utilizzato per segnalare all'utente se il robot è acceso oppure spento. Questo perché prima di eseguire qualsiasi compito, il microcontrollore di Morgul deve essere esplicitamente acceso con un comando. Spesso invece, l'utente si dimentica questa banale azione, ed è per questo motivo che si è deciso di riportare in modo chiaro sull'interfaccia web (anche attraverso l'utilizzo dei colori rosso e verde) questa informazione. Nel file viene scritto "1" se il robot è acceso (verde sull'interfaccia web), "0" se spento (rosso sull'interfaccia web).
- stalled: file creato quando il robot entra in una situazione di stallo. Anche in questo caso, l'informazione viene riportata sulla pagina web con l'utilizzo dei colori. Nel file viene scritto "1" se viene rilevata una situazione di stallo (colore rosso sull'interfaccia).
- lock: file creato quando un processo acquisisce il lock sul robot. In questo modo, nessun altro potrà avervi accesso finché non verrà rilasciato (attraverso la cancellazione del file). Anche quest'ultima informazione viene riportata all'utente.

3.6. Modifica del software di pattuglia

Nell'ultima parte dell'elaborato abbiamo proceduto alla modifica del file passeggia.cpp, file che effettivamente contiene le direttive che permettono al robot di eseguire il percorso di pattuglia.

Il file originale infatti, non prevedeva la gestione di eventuali stalli del robot, che, in particolari situazioni, poteva venirsi a trovare impossibilitato nei movimenti ma comunque sempre in azione.

Questo portava il vecchio software a non terminare mai, e solo l'azione manuale dell'utente poteva terminare il programma.

Abbiamo così aggiunto al programma originale la funzione `controlState` che ad ogni ciclo controlla la posizione del robot. Quest'ultima è definita come una variabile `ArPose` della libreria ARIA, cioè una terna (x,y,θ) dove x e y sono le coordinate cartesiane e θ l'angolo di orientamento. Se viene rilevata per cinque volte consecutive la stessa posizione (quindi la stessa terna), il programma ora considera il robot in stallo e quindi impossibilitato a muoversi. Come conseguenza viene creato nella directory /home/sauronweb/status il file stalled e poi terminato il programma.

La scelta di considerare il robot in stallo dopo cinque iterazioni consecutive uguali è stata presa in quanto si è dovuto considerare il caso in cui ci si trovi in prossimità di ostacoli. In questo caso infatti, il robot procede con andatura ridotta, cercando di evitare l'ostacolo e di trovare una strada che lo porti alla posizione goal prefissata nel programma.

Consideriamo ad esempio la situazione in cui il robot stia procedendo con un'andatura lenta, a causa di un ostacolo. Se avessimo considerato come stallo due posizioni uguali consecutive, saremmo potuti incappare nell'errore di considerare il robot in stallo anche se si stava semplicemente muovendo lentamente. Questo perché bisogna considerare che il programma può eseguire un set di istruzioni molto più velocemente di quanto il robot possa eseguire effettivamente un movimento anche breve, e, a maggior ragione, se il robot sta procedendo lentamente.

Abbiamo così deciso di considerare cinque posizioni uguali consecutive come sicurezza per uno stallo.

Si è poi aggiunto al programma di pattuglia una funzione che gli permettesse di scrivere su file per le ragioni esaminate nel Paragrafo 3.5.

4. Modalità operative

Per utilizzare il sistema è sufficiente utilizzare un qualsiasi browser web per richiedere la pagina http://frost.ing.unibs.it/protected/fdr/framemain.php. Comparirà una pagina che permette di vedere il laboratorio attraverso le webcam e permette di controllare il robot.

4.1. Componenti necessari

Per quanto riguarda Frost, è necessario installare il server web Apache abilitando il modulo per l'interpretazione del codice PHP. Le librerie per il client XML-RPC sono incluse nei file del progetto.

Per quanto riguarda Morgul, è necessario installare il server web Apache, abilitando l'esecuzione di programmi CGI in /home/sauronweb/public_html/cgi-bin. Poiché l'applicazione robotController è scritta nel linguaggio di programmazione Python, è necessario che esso sia installato; tuttavia l'interprete Python è generalmente presente di default in qualsiasi distribuzione Linux recente. Le librerie Python per implementare il server XML-RPC fanno parte delle librerie di base, pertanto non è necessario installarle separatamente.

4.2. Modalità di installazione

Le modalità di installazione dei componenti software citati nel paragrafo precedente sono quelle standard dei sistemi operativi utilizzati, segnatamente Mac OS X su Frost e Linux su Morgul. Per istruzioni dettagliate per l'installazione dei file che compongono il progetto si faccia invece riferimento al file INSTALL sul cd.

4.3. Modalità di taratura

Il sistema realizzato non ha bisogno di alcuna taratura. Per la taratura del sistema di parcheggio di Morgul si veda la documentazione apposita.

4.4. Avvertenze

Il server XML-RPC non effettua alcuna autenticazione sull'identità (e quindi l'autorizzazione) di chi effettua le richieste. Occorre pertanto garantire la sicurezza tramite attraverso il controllo di accesso alla rete del laboratorio.

5. Conclusioni e sviluppi futuri

La finalità principale di questo progetto era di stabilire un quadro metodologico per fornire le funzionalità di controllo remoto del robot Morgul. Quanto prodotto costituisce un framework facilmente estensibile in futuro per implementare strutture di controllo più complesse.

La scelta progettuale di utilizzare il protocollo XML-RPC permette ampia libertà implementativi per quanto riguarda le future estensioni del progetto. Ad esempio, non sarebbe eccessivamente complicato scrivere una applet Java che invoca la procedura getOdometryPosition di Morgul per ottenere la posizione stimata del robot e la visualizza graficamente sovrapposta ad una mappa del laboratorio.

La struttura tutto sommato semplice di robotController ed il fatto che il linguaggio in cui esso è scritto, ovvero Python, non è compilato, rende estremamente semplice e veloce l'aggiunta di nuove funzionalità.

Eventuali sviluppi futuri comprendono l'introduzione di ulteriori compiti da far svolgere al robot e l'inserimento di un sistema di controllo manuale, utile specialmente nel caso il robot vada in stallo.

Bibliografia

[1] Kidd, E.: XML-RPC HOWTO, http://www.tldp.org/HOWTO/XML-RPC-HOWTO/index.html

[2] Van Rossum, G. et al.: "Python Library Reference", sezioni 18.24 e 18.25, http://docs.python.org/lib/lib.html

[3] Dumbill, E. et al: "XML-RPC for PHP", http://phpxmlrpc.sourceforge.net/doc-2/

[4] MobileRobots Inc., "MobileRobots Advanced Robotics Interface for Applications (ARIA) Developer's Reference Manual", version 2.5.1

Indice