UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Sviluppo di un'applicazione Android per il controllo remoto di robot

Elaborato di esame di:

Piero Dotti, Paolo Guglielmini, Francesco Percassi

Consegnato il: **24 Luglio 2014**

Sommario

Il lavoro svolto consiste nello sviluppare un'applicazione per sistemi operativi Android che permetta di controllare in remoto un robot mobile. È stata realizzata un'interfaccia grafica che, sfruttando il framework Aria, permette sia di comandare manualmente il robot che di inviare comandi più complessi.
Per conseguire questo obiettivo è stato necessario ricompilare la libreria ARIA per dispositivi Android e sfruttare le tecnologie di Java che permettono di integrare nell'applicazione del codice nativo per consentirne la comunicazione della Virtual Machine Java con il suddetto framework.

1. Introduzione

Questo progetto nasce nel contesto del corso di Robotica e tratta lo sviluppo di un'applicazione per dispositivi Android che permetta, mediante un'interfaccia grafica, di instaurare una connessione con un robot mobile dotato di un server *Arnl,* al fine di comandarlo, sfruttando le funzionalità e le primitive offerte da quest'ultimo. Grazie ad essa è possibile per l'operatore controllare remotamente il robot senza necessariamente disporre di un calcolatore con installato un client dedicato, come ad esempio *MobileEyes*.

In particolare l'applicativo permette all'operatore di:

1. Comandare il robot tramite una pulsantiera;
2. Localizzare il robot all'interno di una mappa rappresentate l'ambiente in cui questo opera
3. Indirizzare il robot verso un punto specifico;
4. Impostare una modalità di movimento autonoma;
5. Visualizzare sullo schermo lo stream video acquisto dalla camera eventualmente montata su di esso;
6. Inviare comandi al calcolatore del robot comandi tramite la shell remota.

Il problema affrontato

L'interazione remota tra il dispositivo ed il robot avviene attraverso un client che, inviando dei comandi ad un server *Arnl* installato ed eseguito sull'unità mobile, ne invoca le funzionalità da esso offerte.

Il terminale Android, che svolge il ruolo di client, deve realizzare l'invio di comandi remoti al server *Arnl*, impartiti dall'utente, per mezzo dell'interfaccia grafica. Uno dei problemi principali affrontati nel progetto è l'integrazione tra il codice Java, utilizzato per lo sviluppo di applicazioni Android, ed il framework *Aria,* il quale è scritto in C++.

Dal momento che non era disponibile, in data del progetto, una versione del framework ARIA compatibile con l'architettura Android, se ne è resa necessaria la ricompilazione per quest'ultima.

1.1. Libreria ARIA

La libreria ARIA offre primitive di comunicazione che interfacciano localmente (tramite porte seriale) o remotamente (tramite una infrastruttura di rete) un calcolatore al robot o ad altri agenti come il Server *Arnl*.

Il codice sorgente è distribuito liberamente ma ne esistono distribuzioni ufficiali solo GNU/Linux e Windows. In essa sono contenute numerose classi che realizzano l'astrazione del concetto di robot ed implementano particolari funzionalità, come ad esempio il monitoraggio dello stato o l'interfacciamento ai componenti montati su di esso.

Nel lavoro affrontato la libreria ARIA è stata usata principalmente per la gestione della comunicazione client (l'applicazione eseguita sul dispositivo Android) e server (il robot) utilizzando i metodi presenti nel componente *ArNetworking* della libreria Aria.

ArNetworking è un insieme di primitive del protocolli di Aria, impiegate per implementare servizi di rete finalizzati al controllo remoto di robot mobile.

1.1.1. Architettura ArNetworking

ArNetworking è un protocollo di rete espandibile, utilizzato per aggiungere servizi di rete ad un'applicazione di controllo di un robot integrato con la librerie ARIA e gli altri moduli software *MobileRobots*.

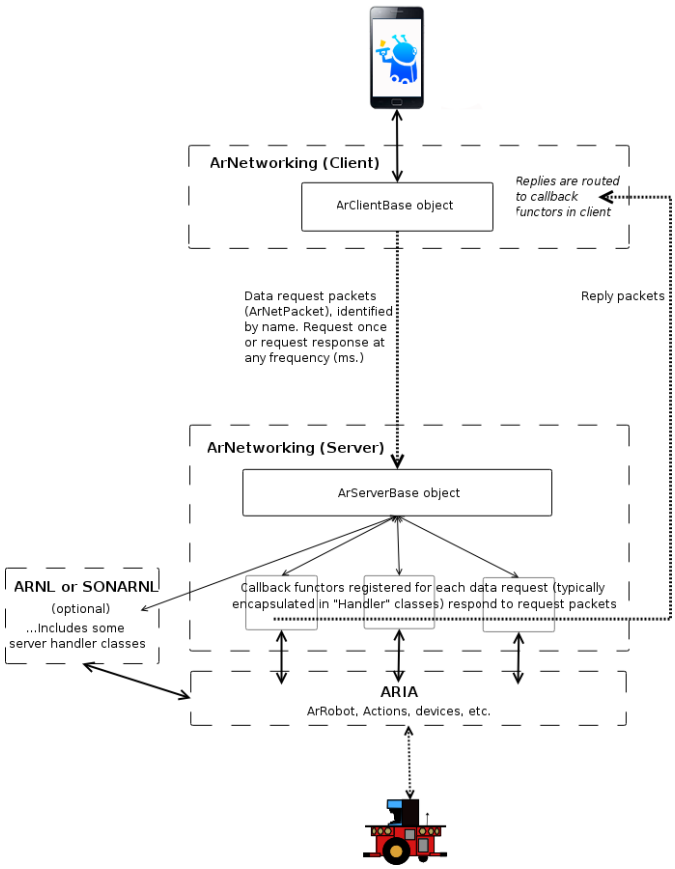
ArNetworking è utilizzato per implementare un architettura client-server, nella quale il client richiede al server di realizzare un singolo comando o di avviare una comunicazione periodica per lo scambio di informazioni di diagnostica.

Il ruolo di Client può essere svolto sia dall'interfaccia operatore grafica standard (*MobileEyes*) che da un componente software ad-hoc. Nel progetto si realizzerà un'interfaccia grafica non standard che offre funzionalità simili a *MobileEyes* su un dispositivo Android.

Nelle applicazioni standard il programma che agisce da Server è eseguito sul calcolatore montato sul robot, il quale utilizza ARIA (e possibilmente ARNL o SONARL) per controllarlo.

L'architettura realizzata è di tipo 3-tier nella quale il robot agisce da *robot server* per il calcolatore montato su di esso. Il computer a sua volta svolge il ruolo di robot client e da *ArNetworking server*.

Questa architettura garantisce una elevata affidabilità dei comandi critici del robot in quanto essi sono trasmessi tramite il canale di stretto contatto tra il robot e il suo calcolatore. In questo modo il client gestisce i comandi ad alto livello in modo trasparente rispetto alle problematiche legata all'inaffidabilità della rete, garantendo una maggiore immediatezza e flessibilità dell'interfaccia utente.



In questo elaborato il client in questione è rappresentato dall'applicazione Android.

1.1.1.1 Lato Server

Un programma server *ArNetworking* comprende un oggetto *ArSeverBase*, al quale viene associato per ogni possibile tipologia di richiesta che si intende gestire un funtore di callback.

La libreria *ArNetworking* rende disponibili una varietà di classi in grado di gestire la richiesta di informazioni sullo stato del robot e l'invio di operazioni di controllo.

Il server delega ai funtori la gestione delle diverse possibili richieste inoltrate dal client. Ad ogni funtore viene associata una callback, la quale viene invocata in corrispondenza dell'opportuna richiesta, ricevendo in ingresso un oggetto *ArNetPacket* contenente il payload del pacchetto e il puntatore ad un oggetto che verrà utilizzato per gestire la risposta.

ArNetworking include la classe *ArServerHandlerCommands* che offre delle primitive per la gestione di comunicazioni che non necessitano di una risposta, come ad esempio l'esecuzione di azioni o il cambiamento dello stato del robot. Inoltre questa classe è in grado di fornire una lista di comandi disponibili per il client.

ArNetworking permette di estendere le funzionalità standard offerte dalla libreria tramite la classe *ArServerSimpleCom*, con la quale è possibile aggiungere nuovi comandi all'oggetto *ArServerCommands*.

In seguito vengono elencati i servizi più significativi offerti da *ArNetworking*:

- *ArServerInfoRobot*: fornisce al client informazioni sullo stato del robot (posizione e velocità corrente, stato del server e voltaggio della batteria);
- *ArServerInfoSensor*: fornisce al client i dati rilevati dalle letture dei sensori;
- *ArServerHandlerMap*: fornisce al client i dati relativi alla mappa utilizzata.

1.1.1.2 Lato Client

Un programma client è composto da un oggetto *ArClientBase* che realizza una connessione con il server, tramite un socket, identificato da un nome e una porta.

La classe *ArClientBase* esegue un thread asincrono per comunicare col server tramite le funzioni *request*() e *requestOnce*(); per gestire le risposte il client deve essere dotato di funtori di callback.

1.2. Breve descrizione degli ambienti di lavoro

1.2.1. I robot mobili

L'ambiente di lavoro in cui questa relazione è stata svolta è il laboratorio di robotica avanzata dell'Università degli Studi di Brescia.

I robot mobili presi in considerazione appartengono alla stessa famiglia, la MobileRobots Inc; in particolare due sono dei Pioneer 1m (*Speedy* e *Tobor*) e uno è un Pioneer 3AT (*Morgul*).

Questi sono controllati mediante programmi eseguibili su un laptop montato su di essi e dotato di sistema operativo Debian 7.

L'applicazione server che permette di stabilire una connessione e di comunicare con il robot mobile è *ArnlServer*. Le richieste che possono essere avanzate da un client, sono per esempio il raggiungimento del goal, ottenere la mappa corrente, lo stato del robot oppure la configurazione di parametri.

1.2.1.1 Lo streaming Video

I robot Morgul e Tobor sono dotati di una camera video, il cui flusso di immagini catturato viene reso disponibile per gli utenti connessi alla rete *Robotica* mediante un'applicazione server eseguita sul robot stesso. Questo servizio viene offerto da un'applicazione a linea di comando chiamata MJPG-streamer che fornisce, tramite protocollo http, lo stream in tempo reale della camera tramite format MJPG. Il robot Speedy al contrario ha una telecamera che non è gestibile da MJPG-streamer.

Per potere accedere alla sorgente video è necessario collegarsi all'indirizzo *IP* del calcolatore montato sul robot alla porta *8081* effettuando una richiesta di *stream*.

1.2.2. Lato Client

L'applicazione che è stata sviluppata dovrà essere installata su un dispositivo dotato di un sistema operativo Android; quest'ultimo è un sistema operativo open source che si pone la finalità di realizzare la portabilità su diversi dispositivi, quali smartphone, tablet e televisori.

I sorgenti delle applicazioni per questo sistema operativo sono scritti utilizzando il linguaggio Java; compilandoli si ottengono dei file in bytecode. Questi non sono eseguiti direttamente da Android ma vengono trasformati in un formato chiamato *dex-code*, il quale sarà poi eseguibile dalla *Dalvik Virtual Machine*, ovvero la macchina virtuale che Android utilizza per eseguire le applicazioni.

Per poter richiamare codice nativo scritto in C o C++ da un'applicazione Java, viene utilizzata l'interfaccia JNI.

Utilizzando la *Java Native Interface* è possibile richiamare all'interno del codice Java delle porzioni di codice nativo, ovvero codice legato ad un sistema operativo e ad un'architettura hardware o più generalmente scritte in altri linguaggi di programmazione, al fine di utilizzare funzionalità intrinsecamente non portabili.

2. La soluzione adottata

2.1. Compilazione di ARIA per dispositivi Android

Nel seguito viene descritta la procedura di compilazione della libreria ARIA per Android in modo tale che questa risulti quanto più comprensibile e conseguentemente possa essere facilmente rieseguita con nuove versioni di ARIA con il minimo sforzo possibile.

Per poter eseguire la compilazione è necessario dopo avere installato la *Native Development Kit* di Android (*NDK*) fornita da google, creare un nuovo file con estensione *.mk*, il quale è il corrispettivo per Android di un makefile per architetture Unix. Questo file configura come il compilatore della libreria nativa di Android dovrà trattare i diversi sorgenti.

In allegato alla relazione è stato fornito un implementazione del file *Android.mk* funzionante.

Alcuni file sorgente non possono essere facilmente compilati secondo questa procedura a causa di alcune differenze nell'implementazione delle libreria standard di Android rispetto a linux. Per sopperire a queste mancanze sono state effettuate delle modifiche.

Al fine di rendere trasparente il processo di compilazione della libreria Aria, i file incompatibili con l'architettura non sono stati modificati, ma ne sono stati creati di nuovi e sostitutivi (che presentano il suffisso "_ANDROID"). In questo modo, nel caso in cui venga rilasciata una nuova versione di Aria, il processo di compilazione risulterebbe semplificato in quanto il file *source_list.mk* contiene l'elenco dei file sorgenti originali compatibili e l'elenco di quelli sostitutivi.

Di seguito sono riportate le incompatibilità emerse in fase di compilazione e le soluzioni adottate:

- il metodo *isspace*[1] non veniva trovato ed è stato risolto con l'inclusione dell'header ctype.h;
- Le funzioni realizzate dai metodi *pthread_cancelstate*[2], *pthread_canceltype*[3] non sono supportate da Android, pertanto sono stati sostituiti da metodi omonimi vuoti;
- Il metodo *pthread_cancel*[4], che permette la cancellazione di un thread, non è fornito nell'implementazione di Android e pertanto è stato sostituito con *pthread_kill*[5] per mezzo di un define;

[1] http://www.cplusplus.com/reference/cctype/isspace/

[2] http://man7.org/linux/man-pages/man3/pthread_setcanceltype.3.html

[3] http://man7.org/linux/man-pages/man3/pthread_cancel.3.html

- In Android non è possibile ottenere l'identificatore di un thread tramite una system call e pertanto è stata utilizzata la funzione *gettid*[6];
- La classe *ArSerialConnection*, deputata alla gestione della comunicazione per mezzo di porte seriali, non è supportata nella sua totalità e, non avendo alcuna utilità pratica in questo contesto applicativo, le funzioni di cui è composta sono state reimplementate vuote;
- Dato che il compilatore Android cerca il file *soundcard.h* all'interno della cartella sys, la quale non è presente, questa è stata creata con all'interno l'header ricercato.

Inoltre sono stati esclusi dalla compilazione i file sorgente che fanno esplicito riferimento a Windows, ovvero quelli che presentano il suffisso WIN.

2.2. L'applicazione Android

L'applicazione sviluppata si compone di due parti distinte: il codice Java a cui è deputata la gestione dell'interfaccia grafica e delle interazioni con l'utente e il codice nativo scritto in C++, che si occupa della comunicazione con il server.

Questi due componenti comunicano tramite la tecnologia JNI, descritta nel capitolo precedente, e sono strutturati in modo tale che il livello inferiore (C++) offra delle primitive al livello superiore (Java) realizzando così un'architettura a due livelli. In particolare il C++ non può mai notificare informazioni alla macchina virtuale Java ma spetta a quest'ultima il compito di richiederle esplicitamente. In questo modo si realizza una proprietà di parziale indipendenza tra i due livelli.

2.2.1. AriaManager

Il codice Java implementato si occupa della ricezione e traduzione di comandi da parte dell'operatore e della conseguente invocazione delle primitive offerte dal codice nativo; queste sono impiegate per comunicare con un server *Arnl* per mezzo del protocollo *ArNetworking*.

La classe principale che si occupa della gestione e della comunicazione con il server è *AriaManager* con la quale è possibile inizializzare la comunicazione, inviare i comandi ricevuti dall'interfaccia grafica e ricevere informazioni di diagnostica.

Segue una descrizione più dettagliata degli attributi che caratterizzano la comunicazione:

- *command*: variabile intera che rappresenta l'ultimo comando ricevuto dall'interfaccia grafica;
- *parameters*: array di interi che rappresenta eventuali parametri del comando;
- *actualMovement*: variabile intera che rappresenta la direzione di movimento attuale del robot;
- *actualMap*: stringa deputata a contenere la mappa scaricata dal server.

Una volta istanziato un oggetto *AriaManager*, questo inizializza una comunicazione utilizzando l'indirizzo IP; dopodiché, nel caso di successo, richiama il ciclo principale.

Segue il codice della funzione *init*:

```
public void init(String ip) throws ArConnectionFailedException
{
        int status = initAria(ip);
        if (status == 0)
                startLoop();
        else
        {
                destroy();
                throw new ArConnectionFailedException(status);
        }
}
```

---

[4] http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_cancel.html

[5] http://man7.org/linux/man-pages/man3/pthread_kill.3.html

[6] http://linux.die.net/man/2/gettid

InitAria è una funzione nativa che demanda al C++ l'inizializzazione di tutti i costrutti necessari per instaurare una comunicazione con il server *Arnl*. Se questa operazione va a buon fine la funzione restituisce un valore intero pari a 0, cosicché venga invocata la funzione *startloop*, rappresentante il ciclo principale di comunicazione, altrimenti viene invocata la funzione di terminazione *destroy* seguita dal lancio di un'eccezione.

Una volta avviato il ciclo principale, caratterizzato da una frequenza di 100 millisecondi, esso si occupa di notificare al robot l'ultimo comando registrato per mezzo dell'interfaccia grafica. La scelta di temporizzare la frequenza dell'invio dei pacchetti contenenti le azioni da eseguire è finalizzata ad evitare un carico di rete eccessivo e di controllare il flusso di pacchetti.

Questo approccio ricalca quello adottato dal client di esempio fornito assieme nella documentazione della libreria ARIA.

Segue il codice della funzione *run*:

```
public void run()
{
        if (command == COMMAND_KILL)
                return;
        double basicInfos[] = getRobotBaseData();
        if (basicInfos[6] == 1 && actualMap == null)
                actualMap = getRobotMap();
        setRobotStatus(actualMovement, command, parameters);
        command = 0;
        parameters = null;
        receiver.receiveRobotInfos(basicInfos, actualMap);
        handler.postDelayed(updateData, 100);
}
```

Il metodo verifica che il comando in ingresso sia diverso da COMMAND_KILL, in quanto questo rappresenta la fase di spegnimento del client.

Il metodo nativo *getRobotBaseData* restituisce i parametri che caratterizzano lo status del robot corrente, formattati come un array di double contenente, nell'ordine, la posizione rispetto agli assi X e Y, l'angolo di rotazione, il voltaggio della batteria, la velocità scalare e angolare e un parametro detto *isMapReady*. Durante il ciclo, se questo assume valore 1 significa che il codice nativo ha terminato di scaricare la mappa dal server.

Se *isMapReady* assume valore 1, e la mappa non è ancora stata istanziata nell'*AriaManager*, viene chiamato il metodo *getRobotMap*, che scriverà la mappa all'interno dell'attributo *actualMap*. Sarà poi compito dell'oggetto *receiver* (di tipo *RobotInfosReceiver*) di aggiornare l'interfaccia grafica. Per la descrizione di quest'ultima si rimanda al paragrafo successivo.

Di seguito viene chiamato il metodo nativo *setRobotStatus*, il quale svolge la funzione di modificare lo stato corrente del robot per mezzo degli attributi *actualMovement*, *command* e *parameters*. Si rimanda alla sezione successiva per l'analisi del suddetto metodo in C++.

Questo metodo non prevede la gestione di una coda nel caso di comandi impartiti con distanza temporale inferiore alla frequenza di aggiornamento, condizione che può verificarsi nel caso di pressione contemporanea di due tasti di movimento.

L'attributo *actualMovement* assume un valore intero in base all'azione da eseguire, come riportato di seguito:

Nome	Codice	Descrizione
MOVEUP	8	Muovi il robot in avanti
MOVEDOWN	2	Muovi il robot in indietro
MOVELEFT	4	Muovi il robot a sinistra

MOVERIGHT	6	Muovi il robot a destra
COMMAND_NULL	0	NoOp

L'attributo *command* assume un valore intero in base all'azione da eseguire, come riportato di seguito:

Nome	Codice	Descrizione
COMMAND_SAFEON	10	Abilito il safe drive
COMMAND_SAFEOFF	11	Disabilita il safe drive
COMMAND_STOP	12	Ferma il robot
COMMAND_GETMAP	20	Richiede la mappa
COMMAND_WANDERER	30	Movimento casuale del robot
COMMAND_GOTOHOME	31	Manda il robot nella posizione iniziale
MODE_RESTORE	40	Ripristina il pilotaggio manuale interrompendo l'azione attuale
COMMAND_KILL	-1	Chiudi il server
COMMAND NULL	0	NoOp

2.2.2. Interfaccia grafica

L'interfaccia grafica è gestita da due classi: *AriaView* e *AriaClient*.

In particolare *AriaClient* comunica con *AriaManager* modificando gli attributi *command*, *parameters* ed *actualMovment* in base ai comandi fisici ricevuti dall'operatore.

Per garantire una maggiore usabilità dell'applicativo, sono stati realizzati in fase di progettazione due layout per descrivere la disposizione degli oggetti all'interno dell'interfaccia grafica, uno dedicato ai dispositivi *smartphone* ed uno per i *tablet*.

Viene nel seguito presentata, a titolo d'esempio, l'interfaccia per *tablet* accompagnata dalla descrizione dettagliata di ogni sua componente.

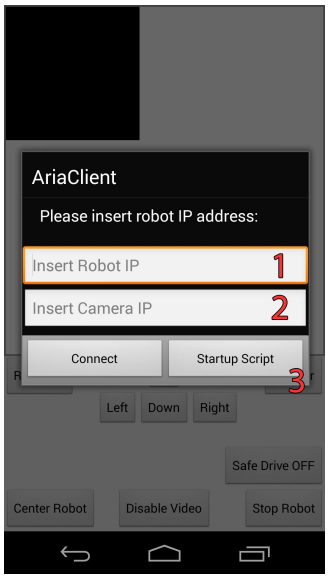
2.2.2.1 Interfaccia di setup

Per potere utilizzare le funzionalità offerte dall'applicazione è necessario eseguire all'avvio una fase di inizializzazione. In particolare questa avviene specificando l'indirizzo IP associato al robot con il quale instaurare una comunicazione e specificando l'indirizzo IP del calcolatore sul quale è eseguita eventualmente l'applicazione che offre lo stream video.

Per poter agevolare la fase di setup, che permette di instaurare una comunicazione tra l'applicativo e il robot, la quale generalmente necessita del fatto che *ArnlServer* sia avviato, è possibile specificare un comando di avvio da eseguire tramite shell remota.

Per esempio sarà possibile richiamare uno script di avvio caricato preventivamente sul calcolatore del robot, il quale includerà al suo interno l'accensione del robot, l'uscita dal dock, l'avvio del servizio di streaming video e l'avvio di *ArnlServer*.

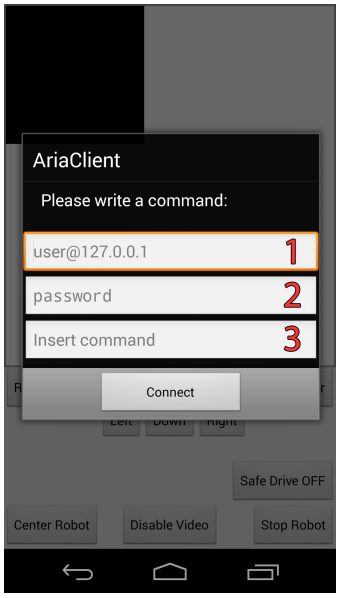
Nelle seguente immagine viene mostrata l'interfaccia che si presenta all'utente all'avvio dell'applicazione:



In riferimento all'immagine se ne elencano le componenti:

1. Indirizzo IP del calcolatore montato sul robot;
2. Indirizzo IP del calcolatore sul quale è eseguita l'applicazione che offre lo stream video. Nel caso in cui questo sia lasciato vuoto, l'applicazione tenterà di connettersi allo stesso indirizzo IP specificato nel campo descritto nel punto 1;
3. Tasto per accedere all'interfaccia per gestire l'invio di comandi di shell al calcolatore montato sul robot, prima che l'applicazione instauri una comunicazione con *ArnlServer*.

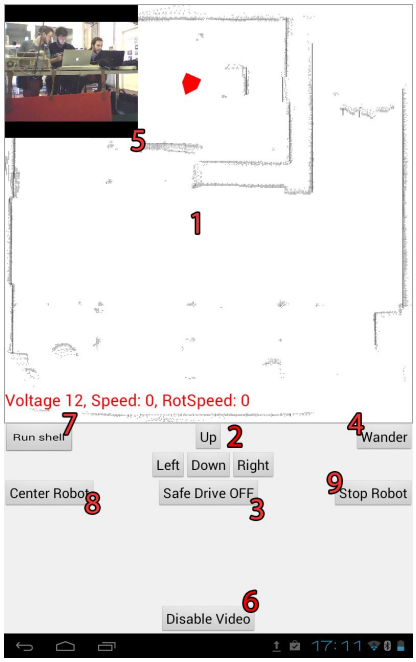
Nelle seguente immagine viene mostrata l'interfaccia che permette di configurare l'invio di comandi di shell:



In riferimento all'immagine se ne elencano le funzionalità:

1. Nome dell'utente del calcolatore montato sul robot con il quale si intende instaurare la comunicazione
2. Password del suddetto utente
3. Comando di shell da eseguire

2.2.2.2 Interfaccia principale



In riferimento all'immagine se ne le elencano le funzionalità:

1. **Mappa**: rappresentazione virtuale della mappa scaricata dal server su cui opera il robot. In corrispondenza della posizione attuale del robot è presente un cursore rosso, il quale si sposterà in corrispondenza dello spostamento del robot. Inoltre, nel caso in cui la mappa ecceda le dimensioni del dispositivo, questa potrà essere navigata utilizzando il tocco delle dita sulla superficie dello schermo. Un'importante funzione della mappa consiste nel poter dirigere il robot in una specifica posizione effettuando un tocco all'interno della stessa nella posizione prescelta;
2. **Pulsantiera**: permette il controllo manuale del robot nelle quattro direzioni;
3. **Attivazione/disattivazione modalità *safedrive***: permette di abilitare (o disabilitare) il controllo del robot in modalità sicura, in modo tale che questo, sfruttando i suoi dispositivi di rilevamento degli ostacoli non li colpisca accidentalmente;
4. **Attivazione modalità *wanderer***: permette di abilitare la modalità *wanderer*, che consiste nel movimento autonomo casuale del robot all'interno della mappa. Una volta abilitata non è possibile eseguire nessun altro comando ad esclusione dell'interruzione della stessa;
5. **Stream video**: finestra nella quale viene mostrato all'operatore lo stream video nel caso in cui questo sia reso disponibile dal robot server;
6. **Mostra/nascondi la finestra dello stream video**;
7. **Run Shell**: tasto che permette l'esecuzione remota di comandi da shell da inviare tramite protocollo ssh, permettendo così l'esecuzione di script presenti sulla macchina;
8. **Center Robot**: tasto che permette, nel caso in cui la dimensione della mappa ecceda le dimensioni dello schermo, di centrarla sul cursore rappresentante il robot;
9. **Stop robot**: permette di arrestare immediatamente il robot a prescindere dall'azione in corso.

2.2.3. C++

Il metodo invocato da Java al fine di inizializzare tutti i costrutti necessari alla comunicazione è *initAria*, del quale se ne riporta il codice sorgente:

```
JNIEXPORT jint JNICALL Java_com_mobilerobots_handler_AriaManager_initAria( JNIEnv* env,
                                        jobject thiz, jstring serverIP )
{
        jboolean iscopy;
        const char* nativeString = (env)->GetStringUTFChars(serverIP, &iscopy);
         int argc = 3;
        char * argv[3] = {(char*)"aria", (char*)"-host", (char *)nativeString};
         Aria::init();
        ArArgumentParser parser(&argc, argv);
        ArClientSimpleConnector clientConnector(&parser);
        parser.loadDefaultArguments();
        if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
         {
                Aria::logOptions();
                return -1;
          }
        client = new ArClientBase();
        if (!clientConnector.connectClient(client))
        {
        if (client->wasRejected())
                return -2;
         else
                return -3;
        }
        client->runAsync();
        inputHandler = new InputHandler(client);
        inputHandler->safeDrive();
        outputHandler = new OutputHandler(client);
        if (iscopy == JNI_TRUE)
                env->ReleaseStringUTFChars(serverIP, nativeString);
        return 0;
}
```

Nel codice viene istanziato un nuovo oggetto di tipo *ArClientBase* che rappresenta l'astrazione software del client. Mediante la funzione *connectClient* dell'oggetto *clientConnector* viene instaurata una connessione con il server specificato negli attributi, solo se questo risulta operativo e disponibile.

In tal caso è possibile stabilire una connessione e, tramite il metodo *runAsync*, creare un thread associato al client.

Dopodiché vengono istanziati due oggetti per gestire l'input e l'output del client chiamati rispettivamente *InputHandler* e *OutputHandler*.

In particolare l'oggetto *inputHandler* viene impiegato per gestire la comunicazione dal client al robot per l'invio di primitive ad alto livello, come ad esempio il movimento casuale, oppure l'invio di comandi di spostamento nelle 4 direzioni. Per impartire quest'ultime al robot la classe fornisce dei metodi per modificare i propri attributi interni ed utilizza la funzione *sendInput* per trasferirli successivamente al server.

Nell'implementazione della classe *inputHandler* questi attributi sono *myTransRatio* e *myRotRatio* che rappresentano rispettivamente lo spostamento e la rotazione differenziale rispetto allo stato corrente del robot.

L'azione di spostamento viene realizzata inviando un pacchetto *ArNetPacket* tramite la funzione *requestOnce*. Questa ha in ingresso due parametri: il comando da eseguire (in questo caso "*ratioDrive*") e il pacchetto formattato con le informazioni necessarie per realizzare lo spostamento del robot.

Di seguito è riportato il codice della funzione *sendInput*:

```
void InputHandler::sendInput(void)
{
	ArNetPacket packet;
	packet.doubleToBuf(myTransRatio);
	packet.doubleToBuf(myRotRatio);
	packet.doubleToBuf(80);
	if (currentRobotNumber<0)
		for (int i=0; i<maxRobotSoFar; i++)
			robotNumberTable[i]->requestOnce("ratioDrive", &packet);
	 else
		myClient->requestOnce("ratioDrive", &packet);
	sinceScan = time(NULL) - lastScan;
	myRotRatio = myTransRatio = 0;
}
```

OutputHandler invece richiede periodicamente informazioni dal server aggiornando i propri attributi interni come per esempio la posizione o la velocità del robot.

Questi due oggetti sono rispettivamente impiegati dalle funzioni *setRobotStatus* e *getRobotBaseData*, delle quali viene riportato il codice:

```
JNIEXPORT jdoubleArray JNICALL
Java_com_mobilerobots_handler_AriaManager_getRobotBaseData(JNIEnv *env, jclass cls)
{
	jdoubleArray result;
	result = env->NewDoubleArray(7);
	jdouble fill[7] = {0, 0, 0, 0, 0, 0, 0};
	fill[0] = outputHandler->getX();
	fill[1] = outputHandler->getY();
	fill[2] = outputHandler->getTh();
	fill[3] = outputHandler->getVolt();
	fill[4] = outputHandler->getVel();
	fill[5] = outputHandler->getRotVel();
	if (outputHandler->isMapReady())
		fill[6] = 1;
	env->SetDoubleArrayRegion(result, 0, 7, fill);
	return result;
}
```

Questa funzione sfrutta i metodi messi a disposizione dall'*OutputHandler* al fine di ottenere informazioni aggiornate sullo stato del robot per renderle disponibili al livello superiore quando richieste:

```
JNIEXPORT jint JNICALL Java_com_mobilerobots_handler_AriaManager_setRobotStatus( JNIEnv*
env, jobject thiz, jint direction, jint command, jintArray parameters )
{
	if (robotState > 0)
	{
	if (command == MODE_RESTORE)
		robotState = 0;
	return -1;
	}
	switch(direction)
	{
```

```
                case 2:
                        inputHandler->down();
                        break;
                case 4:
                        inputHandler->left();
                        break;
                case 6:
                        inputHandler->right();
                        break;
                case 8:
                        inputHandler->up();
                        break;
                default:
                        break;
        }
        switch(command)
        {
                case 10:
                        inputHandler->safeDrive();
                        break;
                case 11:
                        inputHandler->unsafeDrive();
                        break;
                case 12:
                        inputHandler->doStop();
                        break;
                case MODE_HOME:
                        robotState = MODE_HOME;
                        inputHandler->goHome();
                        return 0;
                        break;
                case MODE_WANDERER:
                        robotState = MODE_WANDERER;
                        inputHandler->wanderer();
                        return 0;
                        break;
                case MODE_GOTO:
                        robotState = MODE_GOTO;
                        return 0;
                        break;
                default:
                        break;
        }
        inputHandler->sendInput();
        return 0;
}
```

Questa funzione gestisce due tipi di azioni:

- *direction*: identifica uno spostamento nelle quattro direzioni. Viene gestito per mezzo di metodi dell'*InputHandler* (up(), down(), left() right()) i quali modificano gli attributi interni dello stesso. La funzione *sendInput* farà in modo che tali attributi siano inviati al server, modificando così lo stato del robot.
- *command*: identifica comportamenti complessi del robot richiamando un metodo di *inputHandler* che realizza la comunicazione con il server utilizzando la funzione *requestOnce*, la quale accetta un unico parametro in ingresso che identifica univocamente l'azione da eseguire. Ad esempio *requestOnce("wander")* indica al robot di muoversi casualmente nella mappa.

3. Modalità operative

3.1. Componenti necessari

I componenti necessari per utilizzare il sistema sviluppato sono:

- Un dispositivo dotato di sistema operativo Android 3.1 o successivo su cui deve essere installata l'applicazione *EyeDroid*; si rimanda al paragrafo successivo per ulteriori dettagli concernenti l'installazione;
- Uno dei robot mobili presenti in laboratorio tra *Speedy*, *Tobor* e *Morgul* sul quale sia avviabile *arnlServer*

In alternativa, nel caso in cui non si disponga fisicamente dei robot, è possibile utilizzare come server l'applicazione *MobileSim*[7] che permette di simulare un robot MobileRobots che utilizza *Aria*. L'applicazione deve essere installata su un calcolatore con sistema operativo Windows o Linux (Ubuntu o Debian).

3.2. Modalità di installazione

3.2.1. Compilazione dei sorgenti

L'applicazione è stata sviluppata usando l'IDE Eclipse, l'Android SDK (Software Development Kit[8]) e Apache ANT[9] (per la compilazione a riga di comando) per la parte Java. Per quanto riguarda la compilazione della libreria ARIA è stato utilizzato la NDK (Native Development Kit[10]).

Il risultato del processo di compilazione sarà un pacchetto con estensione *.apk*.

È stato creato, per semplificare l'operazione di compilazione, uno script di shell chiamato *build.sh* che effettua automaticamente sia la compilazione della libreria ARIA che quella dell'applicazione.

3.2.2. Installazione

Per potere installare il pacchetto *apk*, ottenuto direttamente o per mezzo della compilazione, è necessario, una volta copiato all'interno del dispositivo, raggiungerlo con un *file manager* ed eseguirlo. Per permettere questa operazione è necessario abilitare l'installazione di applicazioni provenienti da sorgenti sconosciute modificando le impostazione di sicurezza di *Android*.

In alternativa è possibile installare il pacchetto *apk* utilizzando il seguente comando da terminale *adb install EyeDroid.apk*.

Questa seconda procedura automatica è stata inserita nel file di compilazione *build.sh*.

3.2.3. Scenario applicativo

Di seguito vengono presentate le modalità di installazione dell'applicativo sviluppato con riferimento al robot *Morgul* presente nel laboratorio di robotica avanzata dell'Università degli studi di Brescia:

1. Si avvii l'applicazione sul dispositivo Android sulla quale è installata;
2. Nel form di inserimento dell'*IP* del robot si inserisca l'indirizzo, aggiornato dinamicamente, fornito nella pagina *http://riffraff.ing.unibs.it/~cassinis/Dynamic_Addresses/IP-morgul2013*;
3. Nel form di inserimento dell'*IP* dello stream video si inserisca l'indirizzo, aggiornato dinamicamente, fornito nella pagina *http://riffraff.ing.unibs.it/~cassinis/Dynamic_Addresses/IP-morgul2013a*;

[7] http://robots.mobilerobots.com/wiki/MobileSim

[8] http://developer.android.com/sdk/index.html.

[9] http://ant.apache.org/

[10] https://developer.android.com/tools/sdk/ndk/index.html

4. Con un elaboratore connesso alla rete dell'università ci si connetta a *Morgul* utilizzando il protocollo *SSH*, mediante l'istruzione da terminale *ssh user@IP*, dove con *IP* si intende l'indirizzo del robot fornito con il primo collegamento;
5. Si immetta come password *user*;
6. Nel caso in cui il robot sia spento si esegua il comando *morgulc –R* per accenderlo;
7. Si esegua da terminale lo script *morgulExitFromDock.sh* per permettere a *Morgul* di uscire dal dock e di garantire la corretta localizzazione;
8. Sempre da terminale spostarsi nella cartella */usr/local/Arnl/examples/* ed eseguire *ArnlServer* utilizzando il comando *./arnlServer -map /usr/local/Aria/maps/ARLConGoal.map.*

Per semplificare l'avvio, ed evitare l'utilizzo di un terminale esterno per accendere Morgul ed eseguire gli script di inizializzazione, è possibile sfruttare lo script di avvio, chiamato *setupRobot.sh,* operando nel seguente modo:

1. Si avvii l'applicazione sul dispositivo Android sulla quale è installata;
2. Nel form di inserimento dell'*IP* del robot si inserisca l'indirizzo, aggiornato dinamicamente, fornito nelle pagina *http://riffraff.ing.unibs.it/~cassinis/Dynamic_Addresses/IP-morgul2013*;
3. Nel form di inserimento dell'*IP* dello stream video si inserisca l'indirizzo, aggiornato dinamicamente, fornito nella pagina *http://riffraff.ing.unibs.it/~cassinis/Dynamic_Addresses/IP-morgul2013a*;
4. Aprire l'interfaccia di gestione degli script premendo il bottone *Startup Script*;
5. Inserire nel primo campo la stringa formattata come user@IP, dove IP identifica l'indirizzo ottenuto al punto 2;
6. Inserire nel campo password *user*;
7. Inserire come comando iniziale la stringa *setupRobot.sh* (il quale contiene le direttive per accendere il robot, avviare *ArnlServer* ed eseguire lo script per uscire dal dock)

3.3. Avvertenze

Se l'operatore comanda il robot in remoto, non potendo mantenere un contatto visivo sullo stesso e dovendosi affidare unicamente sulla mappa visualizzata sullo schermo del dispositivo, è consigliabile che l'applicazione sia eseguita in modalità *safe drive*. In tal modo il sistema automatico di rilevamento delle collisioni permette al robot di individuare gli ostacoli, impedendogli così di urtare accidentalmente oggetti e danneggiarsi. Oltretutto è consigliabile sfruttare la funzionalità offerta dall'applicativo per visualizzare sul dispositivo lo stream video qualora fosse reso disponibile dal robot mobile.

Nel caso di emergenza è presente il pulsante *STOP* che blocca immediatamente il robot.

Il sistema mette a disposizione due modalità, il controllo manuale, realizzato con la pulsantiera, e il controllo automatico, che realizza funzioni di movimento autonomo. Quando viene attivata un'azione di questo tipo il sistema ignorerà qualsiasi altro comando ricevuto a meno che non venga inviato un comando di *stop* per mezzo dell'interfaccia grafica, il quale avrà come effetto quello di arrestare l'azione in corso.

4. Conclusioni e sviluppi futuri

Questo progetto ci ha permesso di acquisire conoscenze concernenti lo sviluppo di applicazioni Java per dispositivi Android che, sfruttando l'invocazione di codice nativo, possano interfacciarsi con la libreria ARIA scritta in C++.

L'applicazione presenta dei margini di sviluppo futuri, come ad esempio l'estensione al controllo di multipli robot, da realizzarsi mediante un interfaccia grafica che permetta di spostarsi agilmente tra le differenti viste associate ad ogni singola unità mobile.

Un ulteriore perfezionamento interesserebbe un raffinamento del controllo manuale del robot. Allo stato attuale questi possono essere comandati nelle quattro direzioni secondo una velocità lineare ed angolare costante. Risulterebbe interessante al contrario mettere a disposizione a livello di interfaccia grafica un controllo più evoluto, come ad esempio l'emulazione del funzionamento di un joystick virtuale, in grado di variare dinamicamente questi parametri.

In alternativa si potrebbero sfruttare gli accelerometri integrati nei tablet per permettere all'operatore di controllare manualmente il robot.

Infine un interessante miglioramento sarebbe sfruttare la librerie per l'elaborazione d'immagine, come *OpenCV* per C++, al fine di elaborare le immagini catturate dallo *stream* per realizzare funzionalità avanzate per il rilevamento di persone e oggetti.

Se in un futuro fosse resa disponibile una tecnologia che permetta l'interfacciamento con un dispositivo Android tramite una porta seriale, questo risulterebbe utilizzabile come calcolatore del robot dopo aver opportunamente integrato i sorgenti deputati alla gestione delle porte seriali.

Bibliografia

[1] Documentazione delle API Android: http://developer.android.com/reference/packages.html

[2] Documentazione Thread Android: http://developer.android.com/reference/java/lang/Thread.html

[3] Documentazione Android NDK: http://developer.android.com/tools/sdk/ndk/index.html

[4] Documentazione tecnologia JNI: http://developer.android.com/training/articles/perf-jni.html

[5] Nozioni sulla tecnologia JNI: http://en.wikipedia.org/wiki/Java_Native_Interface

[6] Generalità su Aria: http://robots.mobilerobots.com/Wiki/ARIA

[7] Manuale di Aria: http://robots.mobilerobots.com/Aria/docs/main.html

[8] Documentazione di Aria: http://math.hws.edu/vaughn/cpsc/336/Aria/docs/index.html

[9] Sorgenti per la visualizzazione di MJPG su Android: http://bitbucket.org/neuralassembly/simpemjpegview/src/

[10] Documentazione di ArNetworking: http://math.hws.edu/vaughn/cpsc/336/Aria/ArNetworking/docs/index.html

[11] Forum generale di supporto: http://stackoverflow.com

Sommario