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)

Documentazione del progetto Robot Swarm

Elaborato di esame di: **Nicola Mauri**

Consegnato il: **13 novembre 2001**

Sommario

Nell'ambito dell'attività di ricerca sul demining umanitario, il gruppo di Robotica ha sviluppato un software per la simulazione di gruppi di robot in un ambiente caratterizzato dalla presenza di mine anti–uomo. Questo lavoro, denominato Robot Swarm, è stato oggetto di studio di una tesi di laurea [1] *e di un successivo elaborato del corso di Robotica, raggiungendo dimensioni e complessità notevoli. Ulteriori sviluppi di questo progetto sono tuttavia messi a repentaglio dalla carenza di documentazione sia sui principi base adottati nell'ideazione del simulatore, sia sui numerosi dettagli implementativi.*

Il presente lavoro cerca di colmare tale vuoto, rendendo disponibile della documentazione il più possibile organica e sistematica sul sistema. Per far questo si attingerà innanzitutto dagli articoli esistenti sull'argomento [2], [3]. *La parte di documentazione più consistente sarà tuttavia ottenuta mediante un processo di* reverse-engineering *su quasi 9000 linee di codice C, allo scopo di estrapolare i modelli usati per descrivere i robot, l'ambiente circostante, le strategie di navigazione, nonché l'architettura interna dei moduli software di cui l'applicativo risulta composto.*

In questo documento, oltre a una breve introduzione, si descrive solo come è stata progettata e condotta l'attività di documentazione. Il risultato vero e proprio del lavoro di documentazione si trova nel documento "Robot Swarm – Documentazione".

1. Introduzione

1.1. Il problema del demining

Il Laboratorio di Robotica Avanzata (ARL) dell'Università di Brescia segue ormai da diversi anni gli sviluppi e le problematiche legate all'impiego di robot nelle attività di sminamento. L'introduzione di macchine automatiche per l'individuazione e il disinnesco delle mine antiuomo, in sostituzione degli attuali operatori umani, renderebbe finalmente meno rischiose per l'uomo queste attività e contribuirebbe a velocizzare la bonifica di vaste aree, liberando decine di paesi nel mondo da un pesante ostacolo allo sviluppo economico e civile.

1.2. Perché un simulatore?

Dal punto di vista tecnico, ciò che contraddistingue l'approccio elaborato presso l'ARL è l'idea di usare, anziché un singolo veicolo multi–accessoriato, un insieme di piccoli e semplici robot cooperanti. I robot possono essere suddivisi in squadre e ognuna di queste può essere specializzata in un compito particolare.

Un primo evidente vantaggio di questo metodo sta nel fatto che i robot sono piccoli ed economici, così che i danni per la perdita di una unità risultano molto più contenuti. In secondo luogo, il guasto o la distruzione di un robot non impedirebbe comunque agli altri robot di continuare il loro lavoro, eventualmente riorganizzando e riadattando i propri task alla nuova situazione. C'è poi da tenere in considerazione che sono disponibili oggi numerosi sensori per la rilevazione delle mine (a infrarossi, termici, elettromagnetici, a rilevazione di gas, ecc.) ma che ne nessuno di questi è da solo in grado di assicurare al 100 per cento la localizzazione degli ordigni; ciò suggerisce l'uso una opportuna combinazione di questi sensori, nella quale l'approccio multi-robot si rivela particolarmente appropriato [3].

Sull'altro lato della medaglia abbiamo il non trascurabile problema di dover coordinare il gruppo di robot mediante una qualche forma di controllo distribuito. I robot non solo dovranno cercare di evitare interferenze con altri robot della flotta, ma dovranno anche suddividersi in modo efficiente il carico di lavoro (ovvero le porzioni di terreno da ispezionare) mediante un'opportuna strategia di navigazione, aiutarsi a vicenda scambiandosi le informazioni provenienti dai diversi sensori, e ridistribuire i compiti in caso di guasto a un elemento della flotta.

La complessità di questi problemi esige che ogni proposta venga vagliata e dimostrata "sul campo". Poiché condurre esperimenti con macchine vere sarebbe lungo, costoso e complesso, ecco farsi strada l'idea del simulatore. Robot Swarm consente di verificare e comparare l'efficacia di diverse strategie di navigazione, al variare dell'ambiente in cui i robot si trovano ad operare.

Single Robot	Multi Robot
Complicato e pesante	Semplici e leggeri
Costoso	Più economici
Perdita elevata in caso di incidente	Perdita contenuta in caso di incidente
Navigazione semplice	Navigazione complessa (necessità di controllo coordinato)

Principali differenze tra l'approccio tradizionale e un sistema Multi Robot.

1.3. Caratteristiche salienti di Robot Swarm

Il simulatore Robot Swarm prevede fino a 4 squadre di robot (tre squadre per rilevare le mine e una quarta squadra per la rimozione delle mine stesse). In ogni squadra possono trovare posto fino a 4 robot. Le squadre si muovono in un ambiente fittizio (mondo) che presenta ostacoli (alberi, pietre, mattoni) e dove sono state posizionate mine di vari tipi. Per aumentare ulteriormente il grado di realismo, è possibile simulare errori di misurazione sui sensori, errori sul movimento dei motori e guasti casuali ai robot.

Una descrizione dettagliata ed esauriente delle caratteristiche del simulatore è stata sviluppata all'interno del lavoro di documentazione.

1.4. Principali risultati ottenuti

In una ricerca condotta tramite Robot Swarm sono state analizzate sei diverse strategie di movimentazione [3]. Ogni strategia è stata provata su mondi differenti e con numeri di robot per squadra diversi. La simulazione ha permesso di mostrare, tra le altre cose, come le strategie ordinate di ricerca forniscano in generale prestazioni migliori rispetto alle strategie in cui i robot si muovono in modo casuale all'interno dell'ambiente. I risultati completi della ricerca si possono reperire in [3].

2. Il problema affrontato

2.1. Le dimensioni del progetto

Per avere un'idea delle dimensioni e della complessità del progetto Robot Swarm basta osservare la tabella seguente, che elenca tutti i file sorgente C che compongono il simulatore.

Nome file	*Linee di codice*
multirobots.c	2995
multirobots.h	75
browser.c	198
browser.h	13
colors.h	53
context.h	28
gen_types.h	10
graphics.c	1798
graphics.h	87
header.h	23
include.h	55
khep_serial.c	176
khep_serial.h	69
robot.c	717
robot.h	62
sim.c *(main file)*	1859
sim.h	138
types.h	22
user_info.h	17
user_info.c	19
world.c	246
world.h	62
user.c	262
user.h	13
Totale 25 file	*8997 linee di codice*

Tutto questo codice arriva non commentato e privo di qualsiasi documentazione di supporto.

2.2. L'evoluzione del progetto

Il simulatore Robot Swarm è stato sviluppato a partire dai sorgenti di un altro simulatore di robot, denominato **Khepera Simulator** (attualmente noto col nome commerciale di

Webots). Questo programma, realizzato presso l'Università di Nizza è pensato come supporto all'utilizzo dei robot della serie K-family [4].

Robot Swarm eredita da Khepera l'architettura generale e molte delle caratteristiche relative ai robot e ai mondi di simulazione (alcune delle quali, peraltro, risultano non più utilizzate). Il nostro simulatore introduce tuttavia importanti aggiunte che riguardano:

1. gestione di oggetti di tipo 'mina' nel mondo di simulazione;
2. simulazione di più robot suddivisi in squadre;
3. implementazione delle sei diverse strategie di movimentazione.
4. riorganizzazione sostanziale dell'interfaccia utente.

L'intervento sui sorgenti ha riguardato in gran parte i seguenti file:

- `multirobots.c`
- `sim.c`
- `world.c`
- `robot.c`
- `graphics.c`.

La prima versione di Robot Swarm è stata rilasciata nel 1998, (Tesi di laurea [1]). Alcuni ritocchi al programma sono stati apportati nel settembre 2000 (Elaborato d'esame di Giulio Urlini). Gli interventi hanno riguardato soprattutto l'interfaccia utente, in particolare le finestre di dialogo per il caricamento e il salvataggio dei mondi di simulazione. Sono stati aggiunti i file:

- `browser.c`
- `browser.h`

e sono stati modificati i file:

- `sim.c`
- `graphics.c`.

2.3. Documentazione e riusabilità

La mancanza di documentazione non solo preclude ulteriori sviluppi di questo sistema ma rende anche difficile **riutilizzare** in altri progetti il codice scritto. È infatti abbastanza logico aspettarsi che i diversi lavori di simulazione di robot abbiano in comune molte caratteristiche, e la disponibilità di un set di funzioni base per la gestione dei robot, dei sensori, della rappresentazione grafica, può rendere più semplice e più rapida la realizzazione di nuovi strumenti.

Il codice rilasciato per Khepera era già in origine non documentato. Nonostante questa parte consistente del programma non sia stata sviluppata dall'ARL, abbiamo provveduto a documentarla ugualmente, con l'intento di fornire al laboratorio dei componenti software riutilizzabili in altri progetti.

2.4. Caratteristiche dei sorgenti

Riportiamo qui di seguito una scheda riassuntiva delle caratteristiche implementative del programma.

Linguaggio	C (Unix)
Sistema Operativo	Linux librerie X11 compilatore GCC
Numero di file sorgente	25
Numero di *struct* definite	9
Numero totale di funzioni utente (sottoprocedure)	142
Totale simboli a visibilità globale (variabili globali, macro, funzioni, ecc..)	478

3. La soluzione adottata

3.1. Documentare il progetto

3.1.1. Documentazione in linea

Lo scopo di questo lavoro, come si è detto, è quello di ricavare della documentazione utile a partire dalle righe di codice che implementano il simulatore Robot Swarm. Un primo problema è: dove sistemare questa documentazione?

Affidarsi semplicemente a dei commenti a margine del codice sorgente ha lo svantaggio che le informazioni sono frammentate e sparse su diversi file sorgente, per cui è difficile avere una visione d'insieme del sistema.

Un documento cartaceo statico può essere di sicura utilità, ma risulta molto difficile aggiornarlo e mantenere la consistenza con il codice. Se ad esempio nei sorgenti viene cambiato il nome di una funzione, occorre modificare manualmente il nome di tale funzione in tutte le occorrenze presenti nel documento (particolare attenzione deve essere posta nel fatto che la documentazione, per essere utile in futuro, deve essere aggiornata ad ogni modifica del codice; se tale operazione è troppo scomoda o macchinosa, è probabile che non verrà svolta dai futuri programmatori, e quindi il progetto si svilupperà nuovamente senza un adeguato apparato di documentazione).

La strada scelta è stata allora quella di dotare il codice sorgente di commenti "speciali". Tali commenti descrivono le componenti del codice (funzioni, strutture, macro, ecc.) con l'ausilio di alcune *label*. Ogni *label* è seguita da un particolare testo (ad esempio, la descrizione di una funzione, dei suoi parametri, del dato ritornato), come nel seguente semplice esempio.

```
/**
* Questa funzione esegue la somma tra due interi.
* \param n1 Primo addendo
* \param n2 Secondo addendo
* \return La somma tra i due addendi
*/

int Somma(int n1, int n2)
{
return(n1+n2);
}
```

Per il compilatore C questi sono come normali commenti (del tipo /* */) e in quanto tali vengono ignorati. Un apposito *parser* invece analizza il codice sorgente alla ricerca di queste *label*, ne estrae il testo associato e da qui costruisce una pagina di testo di documentazione, come riportato di seguito.

int Somma (int *n1*, int *n2*)

Questa funzione esegue la somma tra due interi.

Parametri:

n1 Primo addendo

n2 Secondo addendo

Restituisce:

La somma tra i due addendi

Funzione definita alla riga **25** nel file **prova.c.**

Richiamata dalle funzioni **Prodotto()** e **Sottrazione()**.

In questo modo è più semplice garantire la consistenza tra il codice e la documentazione. Se infatti il nome di una funzione cambia, è sufficiente lanciare nuovamente il parser, il quale automaticamente rigenererà la documentazione.

Altro punto interessante è che il parser può di solito produrre l'output in svariati formati: HTML, LaTex e così via. Se il formato lo supporta, verranno generalmente utilizzati link ipertestuali. Per esempio il parser può generare un elenco alfabetico di tutte le funzioni definite nel programma e associare ad ogni funzione nell'elenco un link alla pagina che contiene la sua descrizione. Questa funzionalità è estremamente utile come riferimento rapido per il programmatore.

Un tool per l'ambiente Linux in grado di svolgere questi compiti è **Doxygen**, del quale parleremo nelle sezioni successive.

Chiaramente, grazie a questo sistema, il nostro lavoro si "riduce" ad esaminare i sorgenti e ad inserire gli opportuni commenti nel codice. Lasceremo poi che Doxygen vada a raccogliere qua e là le informazioni depositate, le organizzi, le impagini in modo ordinato, e generi così il file completo della documentazione.

Note

- Oltre allo sforzo di documentare moduli e funzioni, si è anche deciso di aggiungere commenti nel mezzo del codice, soprattutto laddove il significato delle istruzioni poteva risultare di più difficile interpretazione.
- Per mantenere la leggibilità tutte le modifiche testuali apportate durante questo lavoro sono contenute entro le 160 colonne, in modo da consentire la stampa dei file sorgenti in modalità compressa (80x2=160 colonne), senza fastidiosi ritorni a capo. I caratteri di tabulazione sono stati poi convertiti in spazi.

3.1.2. Documentazione statica

Oltre a ciò, per dare una visione globale del sistema e fornire schemi e rappresentazioni grafiche, si è comunque rivelata necessaria la stesura di un documento aggiuntivo. In questo documento ("Robot Swarm – Documentazione") vengono raccolte e illustrate:

- Le specifiche del sistema
- L'organizzazione dei sorgenti

- I modelli del robot, dei sensori, dei motori e del mondo di simulazione
- Il formato dei file generati dal programma.

3.2. Il tool Doxygen

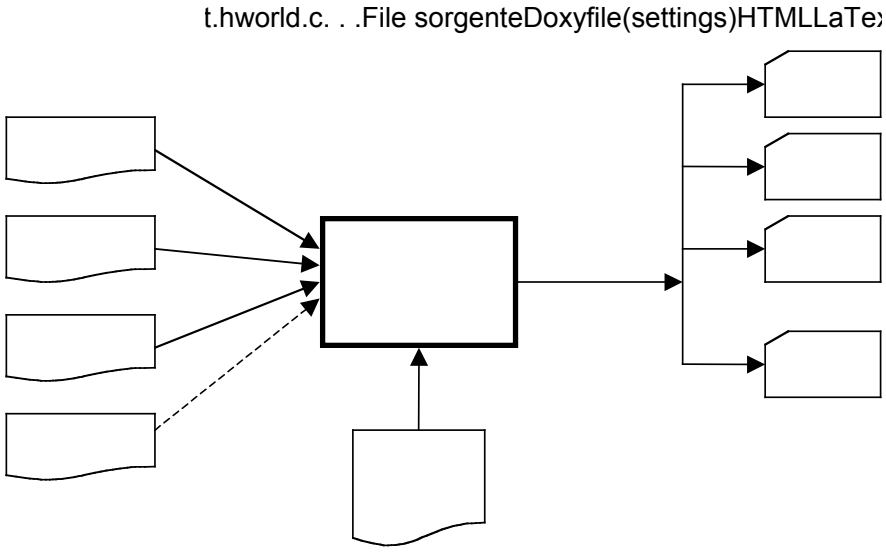
Per la documentazione del codice ci siamo avvalsi di Doxygen, un tool specifico per programmi scritti in C/C++. Doxygen è in grado di analizzare i sorgenti di un programma e generare automaticamente documentazione in svariati formati quali HTML, Compressed HTML, Postscript, RTF (Rich Text Format, MS-Word), XML, PDF, LaTex e pagine di manuale (UNIX man pages).

Doxygen risulta particolarmente adatto nella descrizione delle classi nella programmazione object-oriented, ma risulta efficace anche – ed è proprio il nostro caso – per un programma in C standard.

Doxygen gira sulla maggior parte dei sistemi UNIX-like, Linux incluso, ed è disponibile per il download all'indirizzo:

http://www.stack.nl/~dimitri/doxygen/download.html.

Il funzionamento di Doxygen è schematizzato nella figura seguente.



Doxygen è un tool alquanto sofisticato e dispone di funzionalità molto avanzate. Per non appesantire eccessivamente il lavoro di chi deve (e dovrà) documentare, si è scelto di ridurre al minimo il numero di comandi usati nel documentare il codice. Per semplicità nelle sezioni descriveremo solo questo insieme ristretto di opzioni, che abbiamo ritenuto adatto agli scopi di questo progetto, e supporremo l'utilizzo del file di configurazione `Doxyfile` fornito con questa documentazione. Per ulteriori dettagli sulle altre caratteristiche di Doxygen, consultare l'apposita documentazione all'URL:

http://www.stack.nl/~dimitri/doxygen/index.html.

Il lavoro di documentazione può essere strutturato in tre fasi: documentazione dei sorgenti, configurazione, esecuzione di Doxygen.

3.2.1.1 Passo I: Documentare i file sorgenti

Si tratta di arricchire di "commenti speciali" i file sorgente, aggiungendo una descrizione – concisa o dettagliata – per ogni elemento del codice del programma (funzione, macro, struttura, variabile globale, ...), come illustrato nel paragrafo 3.3.

3.2.1.2 Passo II: Configurare i parametri

Tramite il file di configurazione `Doxyfile` è possibile controllare nei minimi dettagli l'analisi dei sorgenti e la generazione dell'output da parte di Doxygen. Per generare un file di configurazione di default è sufficiente usare il comando:

```
doxygen -g
```

Il file `Doxyfile` creato conterrà le informazioni di supporto necessarie alla corretta impostazione di tutti i parametri di configurazione. Il file `Doxyfile` incluso in questa versione di Robot Swarm è già stato opportunamente configurato ed è consigliabile non modificarlo, eccetto per il caso in cui si voglia abilitare la generazione di ulteriori formati di output.

3.2.1.3 Passo III: Processing dei file sorgente

Dopo aver regolato i parametri del `Doxyfile` è possibile generare effettivamente la documentazione con il comando:

```
doxygen <nome del Doxyfile>
```

Se il file di configurazione si chiama proprio `Doxyfile` e si trova nella directory corrente è possibile lanciare il tool semplicemente con il comando

```
doxygen
```

Se non vi sono errori, a esecuzione terminata troverete la documentazione generata con gli output selezionati nelle rispettive directory.

Dopo qualsiasi modifica del codice sorgente, la documentazione può essere rigenerata semplicemente ripetendo il precedente comando.

3.3. Istruzioni per documentare i sorgenti

Un buon sistema di documentazione deve anche essere sufficientemente semplice ed essenziale da richiedere al programmatore il minimo sforzo aggiuntivo e non appesantire troppo il codice. Per questo motivo abbiamo selezionato un insieme molto ristretto di opzioni tra quelle messe a disposizione da Doxygen, che ora brevemente passiamo in rassegna.

3.3.1. Documentare le funzioni

Ad ogni funzione che si vuole documentare è necessario far precedere un *blocco di documentazione* che, come già sottolineato, non è altro che uno speciale commento in linguaggio C inserito in corrispondenza della definizione della funzione. Ogni blocco di documentazione ha la seguente struttura:

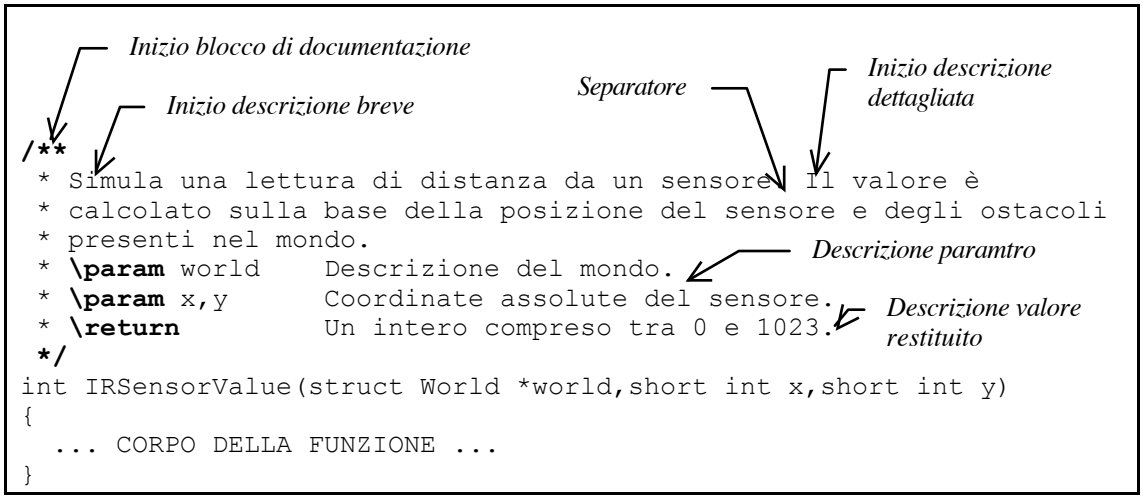
```
/**
 * ... text ...
 */
```

Tutti i commenti che iniziano con la sequenza `/**` vengono trattati da Doxygen come "commenti speciali" ovvero blocchi di documentazione.

Ogni blocco di documentazione contiene una descrizione breve, più – in via opzionale – una descrizione dettagliata, ed altre informazioni. La descrizione breve termina con un punto e uno spazio. Il testo successivo viene trattato come descrizione dettagliata.

```
/**
 * Descrizione breve. Descrizione dettagliata e
 * ulteriori informazioni.
 */
```

Per inserire una descrizione dei parametri della funzione è necessario usare il comando `\param`, mentre per il valore ritornato il comando `\return`, come illustrato nel seguente esempio.




```
/**
 * Simula una lettura di distanza da un sensore. Il valore è
 * calcolato sulla base della posizione del sensore e degli ostacoli
 * presenti nel mondo.
 * \param world    Descrizione del mondo.
 * \param x,y      Coordinate assolute del sensore.
 * \return         Un intero compreso tra 0 e 1023.
 */
int IRSensorValue(struct World *world,short int x,short int y)
{
  ... CORPO DELLA FUNZIONE ...
}
```

La corrispondente documentazione generata da Doxygen si presenterà così:

int IRSensorValue (struct World * *world*, short int *x*, short int *y*)

Simula una lettura di distanza da un sensore.

Il valore è calcolato sulla base della posizione del sensore e degli ostacoli presenti nel mondo.

Parametri:

world Descrizione del mondo.

x,y Coordinate assolute del sensore.

Restituisce:

Un intero compreso tra 0 e 1023.

Definizione alla linea **249** del file **robot.c**.

Referenced by **InitSensors**(), e **MultiInitSensors**().

Link alle funzioni che chiamano questa funzione

3.3.1.1 Parametri di ingresso/uscita

Per meglio chiarire il funzionamento dei parametri passati alle procedure, abbiamo adottato delle etichette nella sezione `\param` che precisano se i parametri sono di *ingresso* (IN), di *uscita* (OUT), di *ingresso/uscita* (IN/OUT). Un parametro è considerato di ingresso se l'oggetto a cui punta viene solo letto dalla procedura ma non modificato. Un parametro è considerato di uscita se l'oggetto a cui punta non viene letto ma viene modificato. Infine, un parametro si dice di ingresso/uscita se l'oggetto puntato viene sia utilizzato sia modificato dalla procedura.

```
\param   *world     (IN) Mondo in cui il robot si muove
\param   *object    (OUT) Oggetto che il robot ha urtato
\param   *robot     (IN/OUT) Il robot, con le coordinate dopo l'urto
```

Note

- Tali etichette non vengono gestite dal parser Doxygen.
- Laddove non specificato, i parametri si intendono di *ingresso*.

3.3.2. Documentare costanti e variabili globali

Per le costanti e le variabili globali è possibile utilizzare blocchi di documentazione simili a quelli già visti per le funzioni, ma risulta in generale più comodo e più leggibile usare blocchi di commento su una sola linea. Di seguito ne è mostrato un esempio:

```
double robot_x;   ///< Coordinata x del robot.
```

oppure, mediante commenti chiusi:

```
double robot_x;   /**< Coordinata x del robot. */
```

Questi commenti dunque *seguono* la definizione dell'oggetto anziché precederla. Non consentono in generale la distinzione tra descrizione breve e descrizione dettagliata.

Note

Attenzione alle macro! Occorre usare sempre e solo commenti di tipo chiuso:

```
#define MAX 100     /**<   Commento     */
```

non quelli di tipo:

```
#define MAX 100     ///<   Commento
```

Infatti il preprocessore C sostituisce ogni costante con tutta la parte di testo che va dalla sua definizione fino alla fine della riga , quindi l'istruzione:

```
x=MAX+1;
```

verrebbe nel primo caso espansa come:

```
x=MAX /**<   Commento */ +1;
```

mentre nel secondo caso risulterebbe:

```
x=MAX ///<   Commento +1;
```

dando origine ad un errore di sintassi (che spesso è di difficile decifrazione per l'utente, visto che il codice non espanso appare sintatticamente "corretto").

3.3.3. Documentare i componenti di una struct

Nel documentare una struttura si è soliti usare un blocco di documentazione per descrivere la struttura nel suo complesso e dei commenti su una sola linea per dettagliare i singoli componenti della struttura, come mostra il seguente esempio.

```
/**
 * Modello del robot.
 * Il robot definito da questa struttura consta di 8 sensori e 2
 * ruote motrici. La struttura comprende anche i quattro vettori di
 * controllo (V1, V2, V3, V4) che regolano la movimentazione del
 * robot.
*/
struct Robot
{
  u_char   State;          ///< Flag di stato del robot.
  double   X;              ///< Posizione x del robot [mm].
  double   Y;              ///< Posizione y del robot [mm].
  double   Alpha;          ///< Orientazione del robot [rad].

  double   Alpha1;         ///< Angolo del vettore V1.
  double   Alpha2;         ///< Angolo del vettore V2.
  double   Alpha3;         ///< Angolo del vettore V3.
  double   Gain1;          ///< Modulo del vettore V1.
  double   Gain2;          ///< Modulo del vettore V2.
  double   Gain3;          ///< Modulo del vettore V3.
  double   Gain4;          ///< Modulo del vettore V4.

  struct Motor     Motor[2];       ///< I due motori del robot.
  struct IRSensor  IRSensor[8];    ///< I sensori di distanza.

};
```

3.3.4. Documentare un file sorgente

Per aggiungere la descrizione di un file occorre inserire (in un punto qualsiasi del file) un blocco di documentazione con il comando `\file <nomefile>`, come riportato nel seguente esempio.

```
/** \file robot.c
 * Questo file contiene procedure per simulare un robot singolo.
 * Vengono simulate le misure dei sensori di distanza e di mine.
 * Viene calcolato lo spostamento del robot sulla base
 * dei vettori di controllo.
 */
```

Anche in questo caso il punto fa da separatore tra la descrizione breve e la descrizione dettagliata.

3.4. Struttura dell'output HTML

Tra i formati attualmente supportati da Doxygen, il formato HTML è probabilmente quello più flessibile e potente, in quanto ad ogni simbolo del programma può essere associato un link ipertestuale alla pagina che contiene la sua descrizione. Gli altri formati, anche se "statici", mantengono in ogni caso un *layout* molto simile a quello HTML.

La lingua usata da Doxygen per generare i titoli e le intestazioni può essere scelta tra diverse possibilità. Abbiamo scelto la lingua italiana (nonostante alcune traduzioni alquanto infelici...) perché il codice è stato da noi commentato in italiano e riteniamo sia più leggibile un documento scritto interamente con la stessa lingua.

3.4.1. I menu e la sezione Pagina principale

La pagina principale (index.html) presenta il seguente menu:

Pagina Principale	Lista dei composti	Lista dei files	Membri dei composti	Membri dei files
Ritorna alla pagina principale	*Elenca tutte le struct*	*Elenca tutti i file sorgente del programma*	*Elenco alfabetico di tutti i campi di tutte le struct*	*Elenco alfabetico di tutte le funzioni, le macro e le variabili globali*

3.4.2. La sezione Lista dei composti (Compound list)

In questa sezione vengono elencate tutte le classi, le union e le struct dichiarate nei sorgenti del programma, con la descrizione breve tra parentesi. Nel nostro caso compaiono solo strutture struct.

Per ogni struttura struct definita, Doxygen genera poi una pagina HTML dettagliata, raggiungibile facendo clic sul nome della struttura stessa.

3.4.3. Le pagine di documentazione delle struct

Nelle pagine che descrivono ciascuna struct vengono riportati: la descrizione breve della struct, le descrizioni brevi dei campi ("attributi pubblici"), la descrizione dettagliata della struct e le descrizioni dettagliate dei campi.

3.4.4. La sezione Lista dei file

In questa sezione vengono elencati tutti i file che compongono il programma e che sono stati analizzati da Doxygen, con la loro breve descrizione. Cliccando sul nome si va alla pagina di documentazione del file, cliccando sul link [codice] viene visualizzato il codice C del file (sempre in versione HTML).

3.4.5. Le pagine di documentazione dei file

Nella pagina di descrizione di ogni file vengono indicate le classi, le struct, le union, le macro, le funzioni e le variabili globali definite all'interno del file. Di tutti questi elementi viene dato prima un elenco con le sole descrizioni brevi, e poi, a seguire, un elenco con le descrizioni dettagliate.

3.4.6. La sezione Membri dei composti (Compound members)

Questa sezione riporta in ordine alfabetico tutti gli identificatori usati come campi all'interno di struct, union o classi. Essi risultano di una qualche utilità solo nel caso si impiegassero costrutti di tipo classe (C++).

3.4.7. La sezione Membri dei file (File members)

Questa è probabilmente la sezione più interessante e utile della documentazione generata da Doxygen. Essa riporta in ordine alfabetico tutte le variabili globali, le funzioni, le macro, le struct, le definizioni di tipo (ovvero tutti i simboli a visibilità globale) che compaiono nei file sorgente.

Accanto a ogni nome viene riportata la lista dei file nei quali il simbolo è definito o dichiarato. Cliccando il nome del file si viene richiamata la pagina che descrive il simbolo all'interno del file.

Questo elenco è di straordinaria utilità ogni qualvolta – ad esempio durante l'analisi dei sorgenti – si incontra una variabile o una funzione di cui si ignora il significato e non si sa dove sia effettivamente definita, oppure – durante la scrittura di nuovo codice – quando si vuole conoscere rapidamente i parametri di una funzione senza dover recuperare il file nel quale è definita.

4. Modalità operative

4.1. Componenti necessari

Per sfruttare i vantaggi del sistema di documentazione è necessario procurarsi il software *Doxygen*. L'applicativo può essere scaricato da:

http://www.stack.nl/~dimitri/doxygen/download.html.

4.2. Modalità di installazione

Doxygen è generalmente disponibile per Linux nei consueti pacchetti RPM (Red Hat Packet Manager), dunque non dovrebbero presentarsi particolari problemi di installazione.

4.3. Modalità di configurazione

Una volta installato il software è possibile configurarlo mediante il comando `doxygen -g` come descritto in dettaglio nel paragrafo 3.2.1.2. Si consiglia tuttavia di utilizzare il file di configurazione `Doxyfile` fornito con la presente documentazione.

5. Conclusioni e sviluppi futuri

5.1. Riepilogo

In questo lavoro sono stati portati a termine i seguenti punti:

- Analisi approfondita dei sorgenti del progetto Robot Swarm.
- Stesura di un apparato di documentazione statica: modelli usati nella simulazione (modelli dei robot, del mondo circostante, strategie di navigazione, ecc.).
- Creazione di un apparato di documentazione in linea mediante il tool software Doxygen, che consente di generare documentazione in formato elettronico (Html, Pdf, Tex, ...) a partire da commenti speciali inseriti nel codice sorgente.
- Aggiunta di opportuni commenti alle parti più significative del codice.
- Stesura di una "to-do list" che raccoglie alcuni problemi rilevati nell'analisi dei sorgenti e propone delle opportune modifiche.

5.2. Sviluppi futuri

Durante questo lavoro di documentazione abbiamo rilevato le seguenti possibilità di sviluppo:

5.2.1. Riorganizzazione del codice

Allo stato attuale la dislocazione delle procedure nei file sorgenti non sembra rispondere a un criterio ben chiaro. Vanno identificati i diversi livelli di astrazione del codice e definita una gerarchia che ordini i file sorgente in base al livello di astrazione che implementa.

5.2.2. Uso di metodologie object-oriented

Un simulatore come Robot Swarm ben si presta ad essere scritto anche con un linguaggio a oggetti (ad esempio il C++). Il numero di linee di codice necessarie per rappresentare e gestire i robot, le squadre di robot, gli oggetti e il mondo di simulazione, si ridurrebbe drasticamente, rendendo il codice più compatto, più leggibile e più facile da mantenere.

5.2.3. Simulazione dello scambio di messaggi

Attualmente lo scambio di informazioni tra i robot simulati è gestito in modo indiretto dal simulatore, senza preoccuparsi di dove poi effettivamente le informazioni vengono acquisite, dove vengono memorizzate e come vengono trasmesse da un robot all'altro.

Sarebbe interessante studiare e simulare nel suo completo il sistema di interfacciamento tra il robot e il mondo esterno. In particolare i punti da sviluppare sono:

1. Come il robot può conoscere la propria posizione?
2. Come il robot può conoscere la posizione degli altri robot?
3. Come il robot effettivamente comunica il rilevamento di una mina?

Gli ultimi due punti richiedono, fra l'altro, lo sviluppo di un opportuno protocollo di comunicazione tra i robot.

5.2.4. Simulazione dei guasti

Attualmente il programma prevede l'interessante possibilità di simulare un guasto tra i robot, dopo un certo numero di passi di simulazione. Quando un robot è guasto, viene escluso dalla squadra, e i robot restanti riorganizzano eventualmente la loro posizione. Le limitazioni sono due:

- La funzione è attivabile solo staticamente togliendo il commento ad alcune righe di codice. Si potrebbe introdurre la possibilità di abilitare, disabilitare e regolare la simulazione dei guasti direttamente a livello di interfaccia utente.
- È possibile simulare un solo robot guasto per volta. Guasti di due o più robot non sono possibili.

5.3. Il futuro di Robot Swarms

Questo lavoro di documentazione ha evidenziato diverse carenze nell'organizzazione del codice o dei moduli di cui il software è composto. Non poteva essere altrimenti, considerato che sul progetto hanno lavorato persone diverse in tempi diversi. È evidente dunque che la struttura dei sorgenti necessiterebbe di un deciso riordino. Ma è altrettanto evidente che tale riordino andrebbe fatto in funzione di ciò che si vuole ottenere dal simulatore nel futuro, ovvero verso quali funzionalità lo si vuole estendere. Attualmente non è ancora emersa una direzione precisa, dunque abbiamo preferito lasciare il codice intatto e fornire la documentazione del sistema così come risulta allo stato attuale. Questa scelta però non ci ha esonerato dal vagliare con attenzione la metodologia di documentazione. Essa, lo ribadiamo, deve essere particolarmente facile da riapplicare per quanti andranno a lavorare e ad aggiornare il simulatore.

In futuro questa metodologia di documentazione potrebbe essere applicata con successo anche agli altri progetti di Robotica. Essa richiede un certo lavoro aggiuntivo durante lo sviluppo, ma nel lungo periodo può far risparmiare tempo prezioso e permettere di condividere più facilmente il progetto tra persone diverse, aumentando, in un certo senso, il valore del progetto stesso.

Bibliografia

[1] A. Cavagnini, P. Ransenigo: "Strategie di cooperazione per squadre di robot autonomi dedicati all'esplorazione di ambienti esterni" – *Tesi di Laurea 1998/99,* Università degli Studi di Brescia.

[2] R. Cassinis: "Multiple single-sensor robots rather than multi-sensor platforms: a reasonable alternative?", *ARL*, Università degli Studi di Brescia.

[3] R. Cassinis, G. Bianco, A. Cavagnini, P. Ransenigo: "Strategies for navigation of robot swarms to be used in landmines detection", *Proc. Eurobot '99*, Zurich, 1999.

[4] O. Michel: "Khepera Simulator version 2.0 User Manual", 1996, http://diwww.epfl.ch/lami/team/michel/khep-sim/index.html

[5] Dimitri van Heesch, "Doxygen v.1.2.1 User Manual", 2000, http://www.stack.nl/~dimitri/doxygen/index.html

Indice