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)

Riorganizzazione database Sauron

(Trasferire tutti i file di stato del sistema su un database MySQL)

Elaborato di esame di:

Oscar Venturini, Stefano Bennati, Giacomo Negretti, Simone Frassanito

Consegnato in data: **11 luglio 2011**

Sommario

Nel presente lavoro, il team ha rimodernato tutta l'infrastruttura di comunicazione dei dati di stato dei robot. Le informazioni ora sono memorizzate in un server centralizzato che le comunica a chi ne fa richiesta tramite un tunnel SSH.

1. Introduzione

Per il normale funzionamento dei robot e del servizio web Sauron è necessario disporre di numerosi dati, generati sia dai robot che da alcune applicazioni in esecuzione su di essi.

Il sistema implementato per salvare e leggere questi dati si basava sulla scrittura e lettura di file contenuti nel file system dei singoli robot. Questo sistema presenta evidenti svantaggi in fatto di dispersione, sicurezza e disponibilità dei dati.

2. Il problema affrontato

L'obiettivo del progetto è quello di implementare nuovamente questo sistema in modo che operi su un database centralizzato remoto. Questo database sarà installato su un calcolatore apposito, conterrà i dati di tutti i robot, dovrà essere scalabile per contenere nuovi robot, e dovrà essere raggiungibile dai robot sia dall'interno che dall'esterno della rete del laboratorio.

3. La soluzione adottata

È stato scelto il DBMS MySQL per la realizzazione del database. Lo schema ER[1] prevede tre tabelle chiamate: `status`, `log` e `images`.

La tabella status contiene un record per ciascun robot, e tanti campi quanti sono i dati significativi che si vogliono memorizzare.

La tabella log viene utilizzata per registrare il log dell'utilizzo dei programmi, ciascun record contiene un *timestamp*, il nome del robot che lo ha generato, e il messaggio di log.

La tabella `images` contiene i nomi delle immagini salvate durante le pattuglie, associate ad un *timestamp*, al nome del robot e alla periferica che le ha scattate.

Per rendere possibile la connessione al database, in modo trasparente rispetto alla rete a cui il robot è collegato, è stato scelto di usare un *port forwarding*[2] tramite il servizio criptato SSH.

La porta locale `3306`, sulla quale il database ascolta, è stata reindirizzata alla porta sul server remoto, in questo modo il robot è in grado di fare richieste al database remoto esattamente come se fosse in locale.

Questo servizio va avviato e mantenuto attivo per tutto il tempo necessario. A far questo ci pensano degli script chiamati `rs_sauron_daemon` e `rs_sauron_check`, che saranno trattati in dettaglio più avanti.

---

1 Schema Entità-Relazioni.

2 Un inoltro automatico della porta, si legga il seguito per conoscere l'implementazione scelta.

4. Modalità operative

4.1. Componenti necessari

Sul server si è installato ex-novo Debian Squeezy con kernel 2.6.32-5-686. Il nome di host scelto è "Rosie"

Oltre ai servizi inclusi in Debian è stato necessario installare manualmente i servizi:

- `ntp`
- `mysql_admin`
- `mysql_server`

  Risponde solamente alle richieste che provengono da `localhost`, per questioni di sicurezza. Per questo motivo l'accesso da altre macchine è consentito solamente tramite un tunnel SSL
- `phpmyadmin`

  Per potervi accedere dall'esterno è stato necessario configurare il server Apache in modo che ascolti sia sulla porta 80 che sulla porta 3307, la quale è stata resa accessibile dall'esterno tramite un'opportuna configurazione del NAT.

È stato necessario modificare i file di configurazione di:

- Demone ssh: /etc/ssh/sshd-config, in modo che risponda alle porte 22 e 8025
- Apache: in modo che risponda alle porta 80 e 3307 (HTTP) e 3308 (HTTPS)

Sono state inserite configurazioni personalizzate anche nelle tabelle di instradamento[3]:

- Custom Service Table:
  - SSH_Rosie - port 8025
  - SSH_Morgul - port 8023
- Inbound Services:
  - MYSQL_Rosie - ALLOW Always - 192.0.2.5 - Any
  - SSH_Rosie - ALLOW Always - 192.0.2.5 - Any
  - SSH_Morgul - ALLOW Always - 192.0.2.10 - Any

4.2. Modalità di installazione

Tutti i file di libreria necessari al funzionamento dei programmi sono contenuti nella cartella `rs_lib`. Sarà necessario creare una cartella nella home dell'utente chiamata `rs_lib` tramite:

```
mkdir ~/rs_lib
```

e copiarvi il contenuto.

4.2.1. Installazione Database

Si ipotizza di essere nella directory padre di `rs_lib`.

Creare la struttura del database

Per installare il database, creare la struttura usando la query presente in:

[3] Le *tabelle di routing*

```
rs_lib/db/db-rs_lib.sql
```

caricandola tramite *phpMyAdmin*[4], oppure con un qualsiasi altro applicativo di gestione del database.

Una volta creato il database, per sicurezza eliminare il file relativo alla struttura tramite:

```
rm -R rs_lib/db
```

Aggiunta di un robot

Per aggiungere un robot basta inserire un nuovo record all'interno della tabella "status".

L'unico campo che va inizializzato è la chiave primaria, ovvero il campo `name`, nel quale va inserito il nome del robot

4.2.2. Installazione tunnel SSH

Prima di continuare con l'installazione è necessario modificare i seguenti file:

- `rs_lib/ssh/rs_sauron-daemon.sh`
- `rs_lib/ssh/rs_sauron-check.sh`

correggendo i percorsi assoluti in modo da indirizzare alla cartella home dell'utente corretto.

Impostazione del certificato

Per l'autenticazione automatica, bisogna creare la chiave pubblica e privata del robot e copiare quella pubblica all'interno del file `~/.ssh/authorized_keys` sul server Rosie, in questo modo, al collegamento, non viene chiesta la password. Per ogni robot vanno create nuove chiavi.

Avvio automatico con il sistema operativo

Gli script che aprono la connessone SSH si trovano in `~/rs_lib/ssh/`. Bisogna ora impostare l'apertura automatica del *port forwarding* all'avvio del sistema operativo. Per fare questo copiare il demone da utilizzare tramite:

```
sudo cp ~/rs_lib/ssh/rs_sauron-daemon.sh /etc/init.d/
sudo chmod +x /etc/init.d/rs_sauron-daemon.sh
sudo chgrp root /etc/init.d/rs_sauron-daemon.sh
sudo chown root /etc/init.d/rs_sauron-daemon.sh
```

e attivarlo tramite:

```
sudo update-rc.d rs_sauron-daemon.sh defaults
```

A questo punto è possibile eliminare `~/rs_lib/ssh/rs_sauron-daemon.sh`.

N.B. Il demone, in esecuzione, crea il file `/var/run/rs_sauron-daemon.pid` per controllare il servizio. All'interno del file vi è scritto l'ID del processo del demone.

Controllo automatico di esistenza della connessione

Per attivare il controllo automatico di corretto funzionamento del port forwarding, impostare il servizio `cron` aggiungendo il contenuto di `~/rs_lib/ssh/crontab-entry` a `/etc/crontab` tramite:

```
sudo cp /etc/crontab /etc/crontab.old
sudo cat ~/rs_lib/ssh/crontab-entry >> /etc/crontab
```

a questo punto è possibile eliminare `~/rs_lib/ssh/crontab-entry`.

---

[4] www.phpmyadmin.net

Note

Tutte le operazioni di avvio e controllo automatico possono essere eseguite posizionandosi in `~/rs_lib/ssh` e invocando `install-daemon`. Si noti che lo script di installazione non cancella `rs_sauron-daemon.sh` e `crontab-entry` automaticamente.

4.2.3. Installazione C++

Per compilare un programma C++ con le funzioni di `rs_dbinterface` sono necessari, oltre ai normali strumenti di sviluppo per C++, anche alcune librerie particolari: `libmysqlcppconn4` e `libmysqlclient15-dev`.

Queste possono essere facilmente installate tramite il gestore di pacchetti preferito.

4.2.4. Installazione PHP

Non sono necessari componenti aggiuntivi in quanto tutto il necessario è già contenuto nell'installazione standard di PHP.

4.2.5. Installazione Python

Per usare la libreria Python, è richiesta l'installazione del modulo `python-mysqldb`, installabile tramite il gestore dei pacchetti, oppure è possibile usare il programma `Setup Tools` (se installato) per installare il pacchetto equivalente tramite il comando: `easy_install MySQL_python`.

4.2.6. Installazione Shell

Non sono necessari componenti aggiuntivi per utilizzare le funzioni Shell, in quanto esse chiamano direttamente il programma `mysql`.

4.3. Modalità d'uso

La directory radice è `rs_lib`; essa contiene delle sottocartelle nelle quali si trovano i file da includere nei progetti.

4.3.1. Con C/C++

Per poterne utilizzare le funzioni di comunicazione con il database basta includere nel codice dei propri programmi il file `rs_dbinterface.h`, nel quale sono anche definiti i parametri per la connessione al database.

Sarà poi necessario modificare il makefile in modo da compilare anche il file `rs_dbinterface.cpp`. Si noti che esempi di makefile già aggiornati per compilare anche questo file si possono trovare fra i sorgenti C++ distribuiti con questo documento, nella cartella Programmi.

Per semplicità di realizzazione, questi *makefile* cercano i file `rs_dbinterface.cpp` e `rs_dbinterface.h` all'interno della cartella in cui si trovano, per compilare con successo i programmi è quindi necessario creare in quella cartella una copia o un collegamento simbolico ai due file.

Si veda il capitolo relativo alle modifiche ai programmi per sapere quali file sostituire o modificare.

4.3.2. Con PHP

Per utilizzare le funzioni di interfacciamento al database dei robot, è sufficiente modificare il file `config.php` che contiene i parametri di connessione a MySQL: IP, nome utente, password e nome del database. A questo punto andrà incluso il file `rs_dbinterface.php` in tutti gli script PHP che richiedono l'interfacciamento alle tabelle dei robot.

4.3.3. Con Python

La libreria è costituita dal file `~/rs_lib/python/rs_dbinterface.py`, nella quale sono anche definiti i parametri per la connessione al database. Per utilizzarla bisogna importarla nello script Python. Il metodo consigliato è posizionarsi nella directory nella quale è presente lo script Python e creare un collegamento simbolico al file di libreria tramite:

```
ln -s ~/rs_lib/python/rs_dbinterface.py
```

All'interno dello script aggiungere la riga:

```
from rs_dbinterface import *
```

A questo punto è possibile chiamare le funzioni all'interno dello script Python.

Note

Si può effettuare un test del corretto funzionamento della libreria di Python posizionandosi in `~/rs_lib/python/` e invocando:

```
python rs_dbinterface_test.py
```

Il test effettua un controllo di corretto funzionamento per le diverse classi di funzioni disponibili (in modo che siano usati tutti i moduli richiesti da Python). Il test termina correttamente quando la connessione al database viene chiusa e non vengono riportati messaggi di errore.

4.3.4. Con la Shell

La libreria in Shell è costituita dal file `~/rs_lib/shell/rs_dbinterface.sh`. Per poter utilizzare le funzioni contenute al suo interno bisogna semplicemente aggiungere allo script Shell che ne usufruirà la seguente stringa:

```
. ~/rs_lib/shell/rs_dbinterface.sh
```

Si suppone che i file siano installati in `~/rs_lib/`, dove `~` equivale a `/home/morgulweb/` ed è consigliato utilizzare il percorso assoluto onde evitare che script eseguiti da altri utenti cerchino di importarlo dalla propria home directory.

I parametri di connessione sono contenuti direttamente nel file `rs_dbinterface.sh` .

4.4. Descrizione delle funzioni

Note

Tutte le funzioni booleane, per convenzione, ritornano `0` se terminano correttamente, `1` se si è verificato un errore.

Sulla Shell

Tutte le funzioni hanno un valore di uscita che può essere:

- `0` se è possibile connettersi al database e la query è stata eseguita correttamente;
- `1` se non è possibile connettersi al database.

Per verificare il valore di uscita di una funzione bisogna testare il parametro `$?` dopo aver eseguito la funzione. I parametri vengono passati alla funzione tutti come stringhe, e quindi con i doppi apici:

```
rs_function "param1" "param2" .. "paramN"
```

I valori di ritorno delle funzioni sono tutti di tipo stringa.

Su Python

Si noti che Python, avendo una tipizzazione dinamica forte, non consente di specificare il tipo di ritorno delle funzioni. Per tutte le funzioni di accesso a variabili di stato (quelle che iniziano con `rs_set` o `rs_get`) è ritornato direttamente l'oggetto del database.

rs_get_property

Ottiene il valore di un campo dal database, questa funzione è da intendersi privata ossia usata dai metodi pubblici che accedono ai singoli parametri.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
  - `param_name`: contiene una stringa che identifica in nome del campo di cui si vuole conoscere il valore
- Valore di ritorno: una stringa contenente il valore del parametro richiesto
- Implementazione:
  - **C++: `char * rs_get_property(char *robot_name, char *param_name)`**

    Ritorna `NULL` se si è verificato un errore
  - **PHP: `function rs_get_property(string $robot_name, string $param_name)`**

    Se si verifica un errore nella query viene invocato il `die()` e lo script termina
  - **Python:** non implementata
  - **Shell: `rs_get_property "$robot_name" "$param_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_set_property

Imposta il valore di un campo dal database, questa funzione è da intendersi privata ossia usata dai metodi pubblici che accedono ai singoli parametri.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
  - `param_name`: contiene una stringa che identifica in nome del campo di cui si vuole conoscere il valore
  - `value`: contiene una stringa che definisce il nuovo valore da assegnare al parametro, se il parametro è di tipo numerico deve essere comunque convertito in stringa
  - `timestamp_update`: contiene un valore booleano che stabilisce se aggiornare oppure no il timestamp relativo a quel parametro. Non a tutti i parametri è associato un campo timestamp
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++**: **`bool rs_set_property(char *robot_name, char *param_name, char *value, bool timestamp_update)`**
  - **PHP:** **`function rs_set_property(string $robot_name, string $param_name, string $value, bool $timestamp_update)`**
  - **Python:** non implementata
  - **Shell:** **`rs_set_property "$robot_name" "$param_name" "$value" "$timestamp_update"`**

rs_connect

- Tenta la connessione al database specificato tramite costanti.
- Parametri: nessuno
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++**: **`bool rs_connect()`**
  - **PHP: `function rs_connect()`**

    Si connette a MySQL e seleziona il database `robot`, in caso di errore termina con un `die()` segnalando l'errore
  - **Python**: **`rs_connect()`**

    Restituisce il valore di `MySQLdb.connect()`
  - **Shell**: non implementata.

    Nel caso di script Shell non viene instaurata una connessione e disconnessione col database. Le funzioni si connettono al database tutte le volte che effettuano una query attraverso l'opzione `-e` di MySQL. Questo è necessario perché lo script deve instaurare una sessione non interattiva col server MySQL

rs_disconnect

Tenta la disconnessione dal database

- Parametri: nessuno
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++: `bool rs_connect()`**

    Ritorna sempre il valore `0`
  - **PHP: `function rs_disconnect()`**

    Può anche non essere richiamata dato che in PHP al termine di uno script vengono chiuse tutte le connessioni ai database
  - **Python: rs_disconnect()**

    Restituisce il valore di `MySQLdb.connect().close()`
  - **Shell:** funzione non implementata.

    Nel caso di script Shell non viene instaurata una connessione e disconnessione col database. Le funzioni si connettono al database tutte le volte che effettuano una query attraverso l'opzione *-e* di mysql. Questo è necessario perchè lo script deve instaurare una sessione non interattiva col server mysql

rs_get_locked

Restituisce il valore del campo `locked`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto
- Implementazione:
  - **C++: `char* rs_get_locked(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore
  - **PHP: `function rs_get_locked(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_get_locked(robot_name)`**
  - **Shell: `rs_get_locked "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_set_locked

Imposta il valore del parametro `locked`, che non ha un campo timestamp.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
  - `ip`: contiene una stringa che rappresenta l'indirizzo IP del calcolatore che ha l'accesso esclusivo al robot

- Valore di ritorno: codice d'uscita della funzione
- Implementazione:
  - **C++**: **`bool rs_set_locked(char* robot_name, char* ip)`**
  - **PHP: `function rs_set_locked(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_set_locked(robot_name, ip)`**
  - **Shell: `rs_set_locked "$robot_name" "$ip"`**

rs_get_odo_x

Restituisce il valore del campo `odo_x`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: un *float* contenente il valore del parametro richiesto
- Implementazione:
  - **C++**: **`float rs_get_odo_x(char* robot_name)`**

    All'interno della procedura viene utilizzata la funzione `atoi` quindi il valore in uscita sarà un *float* ma con precisione intera. Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_odo_x(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_get_odo_x(robot_name)`**
  - **Shell: `rs_get_odo_x "$robot_name"`**

rs_get_odo_y

Restituisce il valore del campo `odo_y`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: un *float* contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`float rs_get_odo_y(char* robot_name)`**

    All'interno della procedura viene utilizzata la funzione `atoi` quindi il valore in uscita sarà un *float* ma con precisione intera. Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_odo_y(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_get_odo_y(robot_name)`**
  - **Shell: `rs_get_odo_y "$robot_name"`**

rs_get_odo_theta

Restituisce il valore del campo `odo_theta`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: un *float* contenente il valore del parametro richiesto
- Implementazione:
  - C++: **`float rs_get_odo_theta(char* robot_name)`**

    All'interno della procedura viene utilizzata la funzione `atoi` quindi il valore in uscita sarà un *float* ma con precisione intera. Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_odo_theta(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_get_odo_theta(robot_name)`**
  - **Shell: `rs_get_odo_theta "$robot_name"`**

rs_get_odo

Restituisce il valore dell'odometria.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: un vettore di tre *float* contenente il valore dell'odometria (x,y e theta)
- Implementazione:
  - **C++**: **`float rs_get_odo(char* robot_name)`**

    All'interno della procedura viene utilizzata la funzione `atoi` quindi il valore in uscita sarà un *float* ma con precisione intera. Ritorna `NULL` se si è verificato un errore
  - **PHP**: non implementata
  - **Python: `rs_get_odo(robot_name)`**
  - **Shell:** non implementata

rs_set_odo_x

Imposta il valore del campo `odo_x`, aggiorna il *timestamp* dell'odometria.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
  - `value`: contiene un *float* che definisce i valori dell'odometria nella componente x
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++: `bool rs_set_odo_x(char* robot_name, float value)`**

il valore viene letto come float ma scritto come stringa nel database

- **PHP:** non implementata, vedi `set_odo()`
- **Python:** `rs_set_odo_x(robot_name, odo_x)`
- **Shell:** `rs_set_odo_x "$robot_name" "$value"`

rs_set_odo_y

Imposta il valore del campo `odo_y`, aggiorna il timestamp dell'odometria.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
  - `value`**:** contiene un *float* che definisce i valori dell'odometria nella componente y
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++:** `bool rs_set_odo_y(char* robot_name, float value)`

    Il valore viene letto come *float* ma scritto come stringa nel database
  - **PHP:** non implementata, vedi `set_odo()`
  - **Python:** `rs_set_odo_y(robot_name, odo_y)`
  - **Shell:** `rs_set_odo_y "$robot_name" "$value"`

rs_set_odo_theta

Imposta il valore del campo "odo_theta", aggiorna il timestamp dell'odometria.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
  - `value`: contiene un *float* che definisce i valori dell'odometria nella componente theta
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++:** `bool rs_set_odo_theta(char* robot_name, float value)`

    Il valore viene letto come *float* ma scritto come stringa nel database
  - **PHP:** non implementata, vedi `set_odo()`
  - **Python:** `rs_set_odo_theta(robot_name, odo_theta)`
  - **Shell:** `rs_set_odo_theta "$robot_name" "$value"`

rs_set_odo

Imposta il valore dell'odometria, aggiorna il *timestamp* dell'odometria.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
  - `value`: contiene un vettore di *float* che definisce i valori dell'odometria nelle componenti x,y e theta
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++: `bool rs_set_odo(char* robot_name, float *value)`**

    I valori vengono letti come *float* ma scritti come stringa nel database
  - **PHP: `function rs_set_odo($robot, $odo_x, $odo_y, $odo_theta)`**
  - **Python: `rs_set_odo(robot_name, odo_x, odo_y, odo_theta)`**
  - **Shell: `rs_set_odo "robot_name" "$odo_x" "$odo_y" "$odo_theta"`**

    In questo caso i parametri sono `odo_x`, `odo_y` e `odo_theta` e non un vettore

rs_get_odo_timestamp

Restituisce il valore del timestamp `odo_timestamp` relativo all'ultima modifica di una delle componenti dell'odometria.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`char* rs_get_odo_timestamp(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_odo_timestamp(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_get_odo_timestamp(robot_name)`**
  - **Shell: `rs_get_odo_timestamp "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_get_battery

Restituisce il valore del campo `battery`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`char* rs_get_battery(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_battery(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_get_battery(robot_name)`**
  - **Shell: `rs_get_battery "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_set_battery

Imposta il valore del campo `battery`, aggiorna il timestamp relativo

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
  - `value`: contiene una stringa che definisce il valore della batteria
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++: `bool rs_set_battery(char* robot_name, char* string)`**
  - **PHP: `function rs_set_battery(string $robot_name, string $value)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_set_battery(robot_name, value)`**
  - **Shell: `rs_set_battery "$robot_name" "$value"`**

rs_get_battery_timestamp

Restituisce il valore del timestamp `battery_timestamp` relativo all'ultima modifica del campo `battery`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto
- Implementazione:
  - **C++**: **`char* rs_get_battery_timestamp(char* robot_name)`**

Ritorna `NULL` se si è verificato un errore

- **PHP**: **`function rs_get_battery_timestamp(string $robot_name)`**

  In caso di errore invoca il `die()` e segnala il tipo di errore

- **Python: `rs_get_battery_timestamp(robot_name)`**
- **Shell: `rs_get_battery_timestamp "$robot_name"`**

  Se si verifica un errore ritorna una stringa vuota

rs_get_last_patrol

Restituisce il valore del campo "last_patrol" che è un timestamp.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`char* rs_get_olast_patrol(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore

  - **PHP**: **`function rs_get_last_patrol(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore

  - **Python: `rs_get_last_patrol(robot_name)`**
  - **Shell: `rs_get_last_patrol "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_set_last_patrol

Imposta il valore del campo `last_patrol` al timestamp attuale.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
- Valore di ritorno: il codice booleano di ritorno della funzione
- Implementazione:
  - **C++**: **`bool rs_set_last_patrol(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore

  - **PHP**: **`function rs_set_last_patrol(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore

  - **Python: `rs_set_last_patrol(robot_name)`**
  - **Shell: `rs_set_last_patrol "$robot_name"`**

rs_get_powered

Restituisce il valore del campo `powered`.

- Parametri:
  - `robot_name`**:** contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: un booleano contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`bool rs_get_powered(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_powered(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_get_powered(robot_name)`**
  - **Shell: `rs_get_powered "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_set_powered

Imposta il valore del campo `powered`, aggiorna il *timestamp* relativo

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
  - `value`: contiene un booleano che comunica se il robot è acceso o spento
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++: `bool rs_set_powered(char* robot_name, bool value)`**
  - **PHP: `function rs_set_powered(string $robot_name, string $value)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python: `rs_set_powered(robot_name, value)`**
  - **Shell: `rs_set_powered "$robot_name" "$value"`**

rs_get_powered_timestamp

Restituisce il valore del timestamp `powered_timestamp` relativo all'ultima modifica del campo `powered`

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`char* rs_get_powered_timestamp(char* robot_name)`**

Ritorna `NULL` se si è verificato un errore

- **PHP**: **`function rs_get_powered_timestamp(string $robot_name)`**

  In caso di errore invoca il `die()` e segnala il tipo di errore

- **Python:** **`rs_get_powered_timestamp(robot_name)`**
- **Shell:** **`rs_get_powered_timestamp "$robot_name"`**

  Se si verifica un errore ritorna una stringa vuota

rs_get_stalled

Restituisce il valore del campo `stalled`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: un booleano contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`bool rs_get_stalled(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore

  - **PHP**: **`function rs_get_stalled(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore

  - **Python**: **`rs_get_stalled(robot_name)`**
  - **Shell**: **`rs_get_stalled "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_set_stalled

Imposta il valore del campo `stalled`, aggiorna il timestamp relativo

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
  - `value`: contiene un booleano che comunica se il robot è in stallo
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++**: **`bool rs_set_stalled(char* robot_name, bool value)`**
  - **PHP**: **`function rs_set_stalled(string $robot_name, string $value)`**

    In caso di errore invoca il die() e segnala il tipo di errore

  - **Python**: **`rs_set_stalled(robot_name, value)`**
  - **Shell**: **`rs_set_stalled "$robot_name" "$value"`**

rs_get_stalled_timestamp

Restituisce il valore del timestamp "stalled_timestamp" relativo all'ultima modifica del campo "stalled".

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`char* rs_get_stalled_timestamp(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_stalled_timestamp(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python**: **`rs_get_stalled_timestamp(robot_name)`**
  - **Shell**: **`rs_get_stalled_timestamp "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_get_emergency

Restituisce il valore del campo `emergency`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: un booleano contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`bool rs_get_emergency(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_emergency(string $robot_name)`**

    In caso di errore invoca il die() e segnala il tipo di errore
  - **Python**: **`rs_get_emergency(robot_name)`**
  - **Shell**: **`rs_get_emergency "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_set_emergency

Imposta il valore del campo `emergency`, aggiorna il *timestamp* relativo

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo "name" del database
  - `value`: contiene una stringa contenente il valore booleano che indica se il robot è in arresto di emergenza
- Valore di ritorno: codice booleano d'uscita della funzione.
- Implementazione:
  - **C++**: **`bool rs_set_emergency(char* robot_name, bool value)`**
  - **PHP**: **`function rs_set_emergency($robot, $value)`**
  - **Python**: **`rs_set_emergency(robot_name, value)`**
  - **Shell**: **`rs_set_emergency "$robot_name" "$value"`**

rs_get_emergency_timestamp

Restituisce il valore del timestamp `emergency_timestamp` relativo all'ultima modifica del campo `emergency`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`char* rs_get_emergency_timestamp(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore
  - **PHP**: **`function rs_get_emergency_timestamp(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore
  - **Python**: **`rs_get_emergency_timestamp(robot_name)`**
  - **Shell**: **`rs_get_emergency_timestamp "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_get_docked

Restituisce il valore del campo `docked`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: un booleano contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`bool rs_get_docked(char* robot_name)`**

Ritorna `NULL` se si è verificato un errore

- **PHP**: **`function rs_get_docked(string $robot_name)`**

  In caso di errore invoca il `die()` e segnala il tipo di errore

- **Python**: **`rs_get_docked(robot_name)`**
- **Shell**: **`rs_get_docked "$robot_name"`**

  Se si verifica un errore ritorna una stringa vuota

rs_set_docked

Imposta il valore del campo `docked`, aggiorna il *timestamp* relativo

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
  - `value`: contiene un booleano che comunica se il robot è collegato alla *dockstation*
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++**: **`bool rs_set_docked(char* robot_name, bool value)`**
  - **PHP**: **`function rs_set_docked($robot, $value)`**
  - **Python**: **`rs_set_docked(robot_name, value)`**
  - **Shell**: **`rs_set_docked "$robot_name" "$value"`**

    Se si verifica un errore ritorna una stringa vuota

rs_get_docked_timestamp

Restituisce il valore del timestamp `docked_timestamp` relativo all'ultima modifica del campo `docked`.

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
- Valore di ritorno: una stringa contenente il valore del parametro richiesto.
- Implementazione:
  - **C++**: **`char* rs_get_docked_timestamp(char* robot_name)`**

    Ritorna `NULL` se si è verificato un errore

  - **PHP**: **`function rs_get_docked_timestamp(string $robot_name)`**

    In caso di errore invoca il `die()` e segnala il tipo di errore

  - **Python**: rs_get_docked_timestamp(robot_name)
  - **Shell**: **`rs_get_docked_timestamp "$robot_name"`**

    Se si verifica un errore ritorna una stringa vuota

rs_log

Inserisce un nuovo record nella tabella di log

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo `name` del database
  - `description`: contiene la stringa di log che si vuole salvare nel database
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++**: `bool rs_log(char* robot_name, char* description)`
  - **PHP**: `function rs_log($robot_name, $description)`
  - **Python**: `rs_log(robot_name, description)`
  - **Shell**: `rs_log "$robot_name" "$description"`

rs_save_image

Inserisce un nuovo record nella tabella delle immagini

- Parametri:
  - `robot_name`: contiene una stringa che identifica il nome del robot, come descritto dal campo name del database
  - `camera_name`: contiene una stringa che identifica il nome del dispositivo che ha scattato le fotografie
  - `file_name`: contiene una stringa che identifica il nome del file immagine
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **C++**: `bool rs_save_image(char* robot_name, char *camera_name, char* filename)`
  - **PHP**: `function rs_save_image($robot_name, $camera_name, $file_name)`
  - **Python**: `rs_set_save_image(robot_name, camera_name, file_name)`
  - **Shell**: `rs_save_image "$robot_name" "$camera_name" "$file_name"`

rs_writeDebug

Inserisce un nuovo record nel file di debug locale. Questa funzione viene usata per il debug interno e quindi è da considerarsi privata

- Parametri:
  - `str`: contiene la stringa di debug da salvare nel file di debug
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
- **C++**: `bool rs_writeDebug(char *str)`

rs_get_logs

Stampa 100 righe di log nel formato: `data-ora: testo log <br/>`.

- Parametri:
  - `robot`**:** nome del robot
  - `start`**:** numero da cui partire a stampare le 100 righe di *log*
- Valore di ritorno: codice booleano d'uscita della funzione
- Implementazione:
  - **PHP: `function rs_get_logs(string $nome_robot)`**

5. Lista dei file modificati

5.1. Su Morgul

In questa sezione sono riportati i file modificati, raggruppati per percorso.

5.1.1. ~/Programmi/

~/Programmi/checkdock/

- tipo: script Bash
- uso: controlla se il robot è collegato alla base di ricarica; viene richiamato da molti programmi ed anche dall'interfaccia web di Sauron
- file modificati: **`checkdock`**
- note: copiare in `/usr/bin`

~/Programmi/exitFromDock/

- tipo: programma C++
- uso: fa uscire in modo automatico il robot dalla base di ricarica
- file modificati: **`exitFromDock.cpp`**
- note: copiare l'eseguibile in `/usr/bin`; modificato il comportamento originale aggiungendo una chiamata a `checkdock` alla fine dell'esecuzione

~/Programmi/morguldock/

- tipo: programma C++
- uso: fa rientrare il robot nella base di ricarica utilizzando la visione
- file modificati:
  - **`main.cpp`**
  - **`main.h`**
  - **`dock_centeraction.cpp`**
  - **`Dock_centerAction.h`**
- note: copiare l'eseguibile in `/usr/bin`; modificato il comportamento originale aggiungendo una chiamata a checkdock all'inizio dell'esecuzione

~/Programmi/photoreporter/

- tipo: script bash
- uso: scatta le fotografie durante la pattuglia
- file modificati: photoreporter.sh
- note: copiare in `/usr/bin`

~/Programmi/photoreporter-panorama/

- tipo: script bash
- uso: scatta le fotografie durante la pattuglia
- file modificati: **`photoreporter-panorama.sh`**
- note: copiare in `/usr/bin`

~/Programmi/patrol_map/

- tipo: programma C++
- uso: esegue una pattuglia
- file:modificati:
  - **defines.h**
  - **patrolmap.cpp**
  - **dump.cpp**
  - **dump.h**
  - **Debug/objects.mk**
  - **Debug/subdir.mk**
  - **Release/objects.mk**
  - **Release/subdir.mk**
- note: copiare l'eseguibile in `/usr/bin`

~/Programmi/exitFromDock/

- tipo: programma C++
- uso: riporta il robot in una posizione corretta dopo aver fallito un tentativo di attracco
- file modificati:
  - **main.cpp**
  - **main.h**
- note: copiare l'eseguibile in `/usr/bin`

~/Programmi/RemoteControlServer/

- tipo: programma C++
- uso: permette di guidare il robot da remoto
- file modificati:
  - **main.cpp**
  - **main.h**
  - **VarSpeedAction.cpp**
  - **VarSpeedAction.h**
- note: copiare l'eseguibile in `/usr/bin`; l'applet ritorna spesso un errore di connessione non causato da questo programma.

5.1.2. ~/scripts/

All'interno di questa cartella sono stati modificati i seguenti script:

- **autopatrolmap:** avvia una pattuglia
- **cambiastato:** ritorna un valore e segnala l'operazione nella tabella di *log*
- **checksetlock:** controlla se il robot è stato riservato[5] da qualcuno
- **emergencyStop:** interrompe i movimenti del robot e lo spegne
- **exitDock:** accende il robot e lo fa uscire dalla base
- **goHome:** fa rientrare il robot nella base e se fallisce chiama `recoverdock`
- **offrobot:** spegne il robot
- **onrobot:** accende il robot
- **patrolmap:** avvia una pattuglia
- **robotparameters:** definisce tutte le costanti usate negli script
- **unlock:** elimina un eventuale blocco sul robot

[5] Ossia se è in stato di "lock".

5.1.3. ~/public_html/cgi_bin/

- **`robotController.py`**: gestore RPC[6] per le azioni richieste tramite il sito web

5.1.4. ~/public_html/

- **`log.php`**: visualizza i messaggi di *log* estratti dalla tabella *log* del database

5.2. Su Frost

5.2.1. ~/Library/WebServer/Documents/protected/

- **`morgulinteract.html`**: pannello web per il controllo di Morgul
- **`morgulstatus.php`**: visualizza un sommario del valore delle variabili di stato di Morgul
- **`speedyinteract.html`**: pannello web per il controllo di Speedy

6. Conclusioni e sviluppi futuri

Allo stato attuale il sistema permette al robot Morgul di eseguire tutte le funzioni per il quale è stato programmato utilizzando il database centrale.

Tuttavia gli algoritmi sono rimasti gli stessi, non sono quindi ottimizzati per sfruttare le caratteristiche di un database. Alcuni programmi sono addirittura non funzionanti, come ad esempio la gestione dell'accesso esclusivo[7] tramite indirizzo IP. Il primo passo per migliorare il progetto è quindi quello di eseguire una reingegnerizzazione dei programmi che permettono il funzionamento del robot.

Un altro passo importante è portare queste modifiche anche sugli altri robot a disposizione del laboratorio ed adeguare di conseguenza l'interfaccia di controllo web del progetto Sauron.

Un ulteriore miglioramento attuabile sarebbe quello di modificare la gestione delle immagini dei pattugliamenti, creando un deposito centralizzato sincronizzato con la tabella images del database.

Bibliografia

[1] MySQL Reference Manual, MySQL C API

[2] Specifiche Python 2.7

[3] Specifiche MySQL-python

[4] http://kimmo.suominen.com/docs/ssh/ per la procedura di creazione tunnel con SSH

[6] Remote Procedure Call

[7] Effettuata tramite il *lock*, una condizione durante la quale il robot mette a disposizione le sue risorse a un particolare utente. Le risorse ritornano disponibili quando l'utente che controlla il robot rilascia il *lock*.

Indice

SOMMARIO **1**

1. INTRODUZIONE **1**

2. IL PROBLEMA AFFRONTATO **1**

3. LA SOLUZIONE ADOTTATA **1**

4. MODALITÀ OPERATIVE **2**
4.1. Componenti necessari **2**
4.2. Modalità di installazione **2**
4.2.1. Installazione Database 2
Creare la struttura del database 2
Aggiunta di un robot 3
4.2.2. Installazione tunnel SSH 3
Impostazione del certificato 3
Avvio automatico con il sistema operativo 3
Controllo automatico di esistenza della connessione 3
Note 4
4.2.3. Installazione C++ 4
4.2.4. Installazione PHP 4
4.2.5. Installazione Python 4
4.2.6. Installazione Shell 4
4.3. Modalità d'uso **4**
4.3.1. Con C/C++ 4
4.3.2. Con PHP 4
4.3.3. Con Python 5
Note 5
4.3.4. Con la Shell 5
4.4. Descrizione delle funzioni **6**
Note 6
Sulla Shell 6
Su Python 6
rs_get_property 6
rs_set_property 6
rs_connect 7
rs_disconnect 8
rs_get_locked 8
rs_set_locked 8
rs_get_odo_x 9
rs_get_odo_y 9
rs_get_odo_theta 10
rs_get_odo 10
rs_set_odo_x 10
rs_set_odo_y 11
rs_set_odo_theta 11
rs_set_odo 12
rs_get_odo_timestamp 12
rs_get_battery 13
rs_set_battery 13

rs_get_battery_timestamp 13
rs_get_last_patrol 14
rs_set_last_patrol 14
rs_get_powered 15
rs_set_powered 15
rs_get_powered_timestamp 15
rs_get_stalled 16
rs_set_stalled 16
rs_get_stalled_timestamp 17
rs_get_emergency 17
rs_set_emergency 18
rs_get_emergency_timestamp 18
rs_get_docked 18
rs_set_docked 19
rs_get_docked_timestamp 19
rs_log 20
rs_save_image 20
rs_writeDebug 20
rs_get_logs 21

5. LISTA DEI FILE MODIFICATI 22
5.1. Su Morgul 22
5.1.1. ~/Programmi/ 22
~/Programmi/checkdock/ 22
~/Programmi/exitFromDock/ 22
~/Programmi/morguldock/ 22
~/Programmi/photoreporter/ 22
~/Programmi/photoreporter-panorama/ 22
~/Programmi/patrol_map/ 23
~/Programmi/exitFromDock/ 23
~/Programmi/RemoteControlServer/ 23
5.1.2. ~/scripts/ 23
5.1.3. ~/public_html/cgi_bin/ 24
5.1.4. ~/public_html/ 24
5.2. Su Frost 24
5.2.1. ~/Library/WebServer/Documents/protected/ 24

6. CONCLUSIONI E SVILUPPI FUTURI 24

BIBLIOGRAFIA 24

INDICE 25