

Spider-chase

Marco Mecchia
Luigi Giugliano
Simone Romano

Indice

1	Introduzione	3
1.1	Requisiti funzionali	3
2	Architettura e componenti	4
2.1	Ragni meccanici	5
2.2	STM32F401RE Nucleo	7
2.3	Driver motore	7
2.4	Modulo wireless	8
3	Dettagli implementativi	10
3.1	Controller	10
3.1.1	Android Controller	10
3.1.2	RoboWarsApp	10
3.2	Driver motore	10
3.3	Modulo wireless	11
3.4	Visione Artificiale	12
4	Conclusioni	14
4.1	Sviluppi futuri	14
Appendice A	Librerie ChibiOS	15
A.1	Libreria ESP8266	15
A.2	Codice motori	22
Appendice B	Visione artificiale	29

Capitolo 1

Introduzione

Lo scopo del progetto "Spider-chase" è quello di mettere a frutto le conoscenze acquisite nel corso di robotica, sperimentando varie soluzioni riguardanti robot mobili. In particolare, nel nostro progetto abbiamo deciso di utilizzare dei ragni robot DIY (Do It Yourself) dal basso costo, controllati dal microcontrollore STM32 Nucleo. Questa scelta ci ha consentito di:

- Sperimentare con componenti economici.
- Montare i robot in maniera autonoma, senza fare affidamento su prodotti precostruiti, in modo da poter fare modifiche strutturali anche in corso d'opera.
- Utilizzare ChiBiOs, un sistema operativo embedded fornito da STM, in modo da poter programmare i robot in maniera astratta ed evitare la programmazione *bare metal*.

1.1 Requisiti funzionali

Ad inizio progetto ci siamo proposti i seguenti requisiti funzionali:

- Ogni ragno deve essere in grado di muoversi liberamente nello spazio, in qualunque direzione.
- Ogni ragno deve essere in grado di ricevere istruzioni via wireless.
- Ogni ragno deve essere alimentato in maniera autonoma e non deve essere vincolato a sorgenti di alimentazione fisse.
- Un ragno deve essere in grado di stabilire la posizione di un altro ragno ed eventualmente inseguirlo.

Tali requisiti sono stati tutti soddisfatti dal risultato finale.

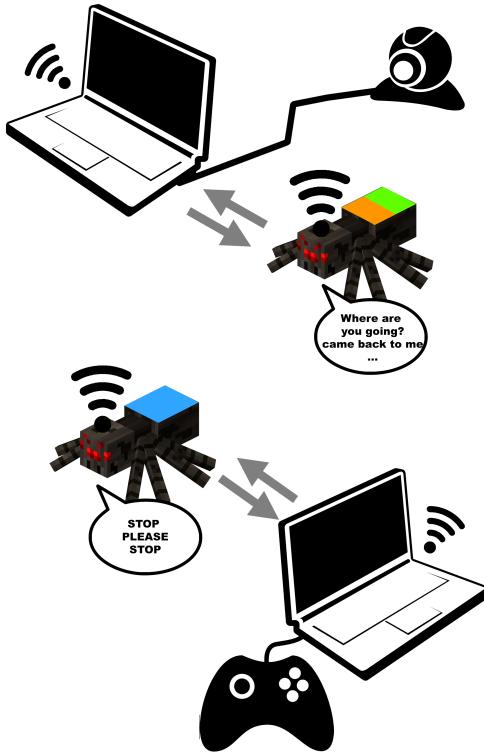
Capitolo 2

Architettura e componenti

Nel progetto abbiamo utilizzato i seguenti componenti:

- 6 ragni meccanici DIY, ciascuno con un motore.
- 3 schede STM Nucleo.
- 3 moduli wireless ESP8266.
- 3 driver per motori L9110S.
- 1 webcam.
- 1 pc.
- 1 controller Xbox.
- 1 cellulare Android.

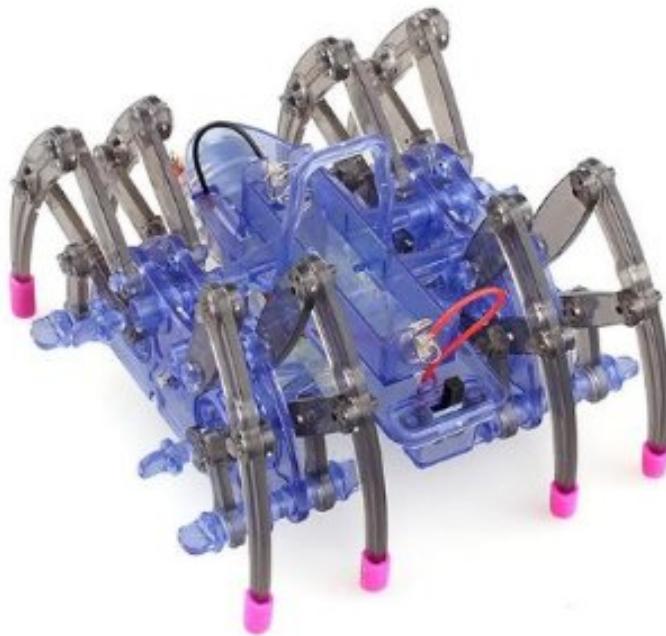
L'architettura totale pianificata é mostrata in figura.



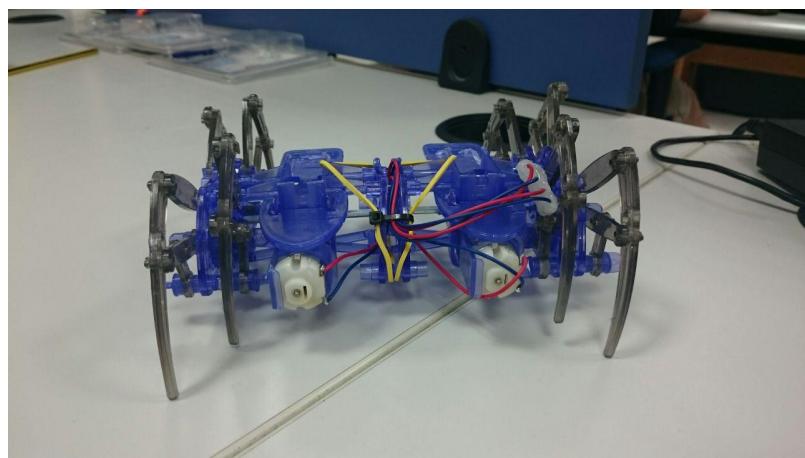
Tutti e tre i ragni possono ricevere messaggi wireless che impostano le velocitá dei motori. Ricevuto il messaggio, la board regola le velocitá tramite il driver. Il requisito di localizzazione é soddisfatto da un modulo di visione artificiale: una webcam posta in alto riconosce i ragni, ed il pc alla quale é collegata invia messaggi direttamente ai ragni tramite WiFi. Inoltre, é possibile comandare i ragni tramite un cellulare Android od un pc, selezionando il ragno al quale si vogliono impartire i comandi. Ogni ragno é alimentato in maniera indipendente da 4 pile stilo poste in un portapile al di sotto del ragno stesso.

2.1 Ragni meccanici

I ragni meccanici che abbiamo comprato, mostrati in figura, utilizzano un solo motore per muovere sia le zampe a sinistra che quelle a destra.

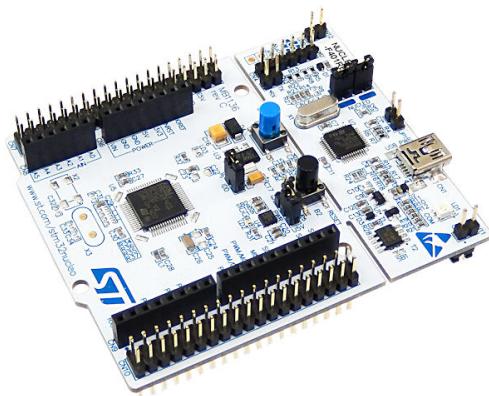


Benché questa semplice soluzione sia sufficiente a far muovere il ragno avanti e indietro a velocità prefisse, essa non andava incontro al nostro requisito di potersi muovere in qualsiasi direzione dello spazio. Per questo motivo, abbiamo rimosso le zampe a destra di tre ragni, e quelle di sinistra agli altri tre. Unendo i ragni così divisi, abbiamo ottenuto tre ragni totali, con il pregio di avere motori separati per zampa.



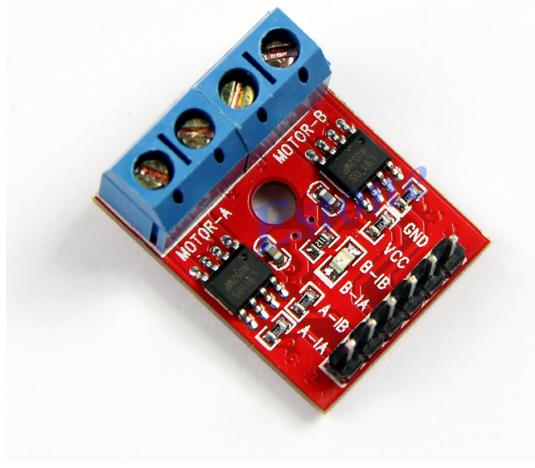
2.2 STM32F401RE Nucleo

La board Nucleo STM32 fornisce un'infrastruttura affidabile e flessibile per gli utenti che vogliono sperimentare nuove idee e prototipi che funzionino con tutte la linea di microcontrollori STM32. Grazie al supporto per la connettività Arduino e ST Morpho, è possibile espandere le funzionalità del microcontrollore scegliendo da una vasta gamma di shield. Inoltre, la STM32 nucleo non richiede due ingressi separati per alimentazione e debugging.



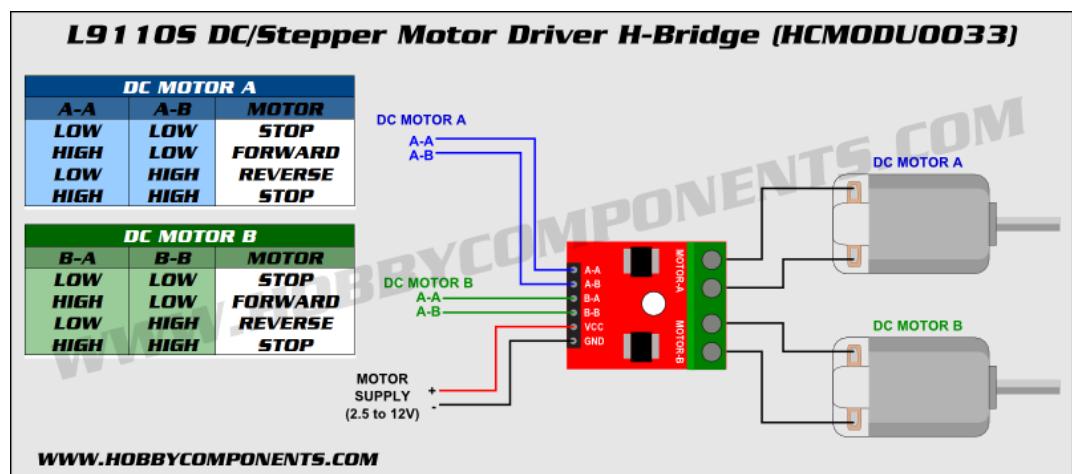
2.3 Driver motore

Il driver del motore utilizzato è il modello L9110s mostrato in figura.



Dei suoi 10 pin totali, 4 sono di input e 4 di output, 1 di alimentazione e 1 di massa. Quelli in entrata vanno collegati agli output digitali della board, mentre di quelli in uscita 2 sono per il motore destro e 2 per quello sinistro, di cui 1 va all'alimentazione del motore ed uno alla massa. La

regolazione della velocità del motore è molto semplice e si basa sui PWM che sono collegati agli output digitali della board: se la frequenza del PWM che arriva all'ingresso 1 è maggiore di quella che arriva all'ingresso 2, allora il ragno va in avanti, altrimenti va all'indietro. Maggiore è la differenza di velocità, più il ragno va velocemente. Di seguito è riportato un diagramma riepilogativo.



2.4 Modulo wireless

Il movimento dei robot è controllato da remoto tramite connessione wireless. La comunicazione wireless è garantita grazie ai moduli ESP8266 che, connessi alla board nucleo stm32f4, sono in grado di:

1. connettersi ad una rete locale, acquisendo un indirizzo IP
2. creare una connessione wireless locale

In entrambi i casi, connettendosi alla rete del modulo è possibile scambiare messaggi tramite richieste http. Un semplice protocollo è stato realizzato per comandare i robot. L'idea è quella di codificare delle istruzioni di movimento in delle stringhe composte da 8 byte con il seguente significato:

- il primo byte indica la tipologia di comando (attualmente è 'M' che sta per 'MOVE')
- il secondo byte è l'ID del robot; è stato pensato per robot connessi alla stessa rete
- 2 triple di 3 byte che codificano l'informazione di movimento relativamente per il motore destro e sinistro; una tripla può assumere un valore intero compreso tra 0 e 255: tra 0 e 127 si regola la velocità di

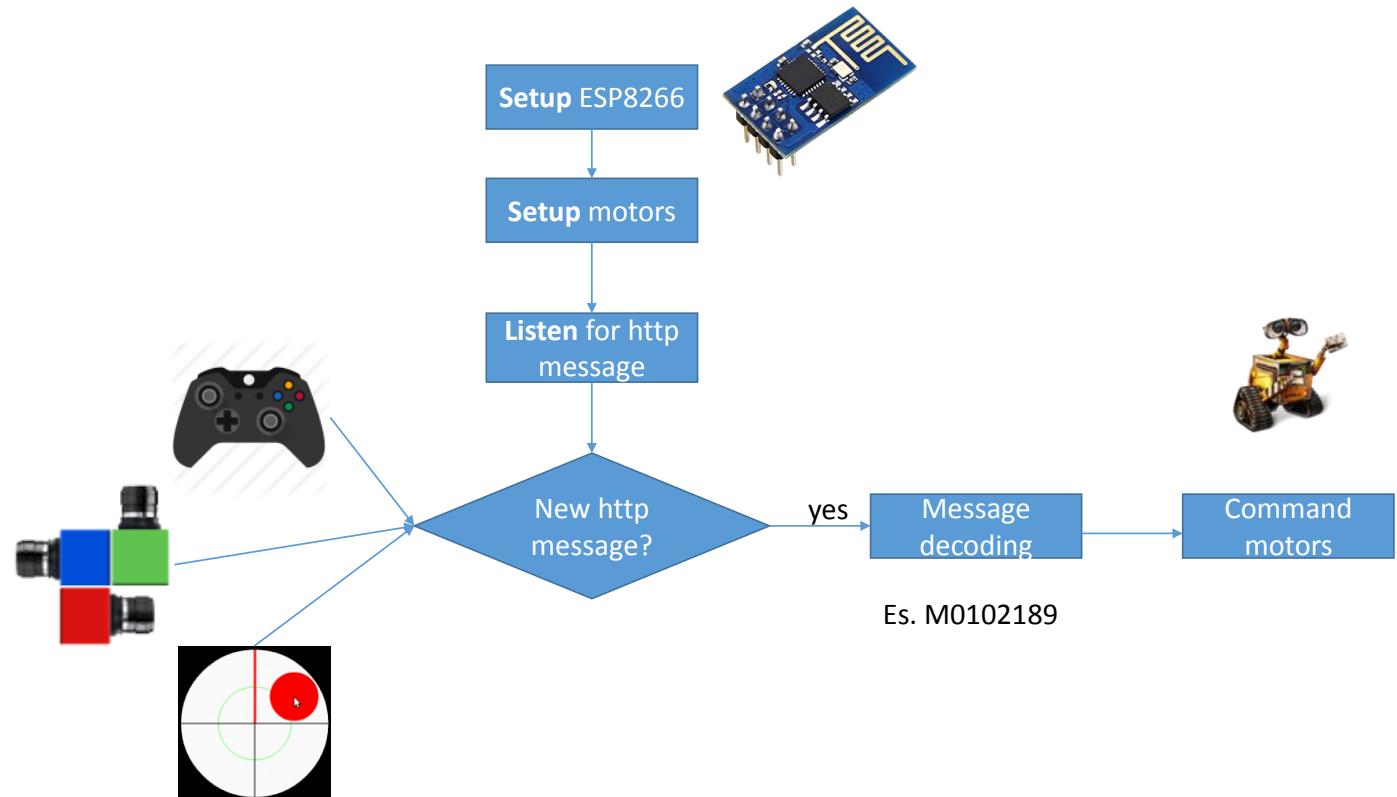


Figura 2.1: Il diagramma mostra lo schema di funzionamento della board che controlla il robot.

un motore in un senso; tra 128 e 255 si regola la velocità del motore nell'altro senso

Il funzionamento del robot è descritto dal diagramma in figura 2.1.

Capitolo 3

Dettagli implementativi

3.1 Controller

Per la comunicazione con i robot sono state utilizzate delle richieste http, con il formato descritto della sezione 2.4. I due robot sono comandati rispettivamente da:

- Soft-Joystick realizzato su Android (app Android) per il robot inseguito.
- RoboWarsApp, una web app realizzata con nodeJS che consente di comandare il robot inseguito tramite tastiera o joystick xbox360.
- Richieste http tramite python per il robot inseguitore.

3.1.1 Android Controller

L'app android utilizza una componente grafica che realizza un Joystick (<https://github.com/zerokol/JoystickView>) e, a seconda della direzione del Joystick, invia richieste http al modulo ESP8266 con il formato descritto in 2.4. L'immagine in figura 3.1 mostra la semplice interfaccia grafica dell'app Android per il controllo del robot.

3.1.2 RoboWarsApp

La web app sviluppata consiste nella semplice interfaccia mostrata in figura.

3.2 Driver motore

Il driver del motore contiene la funzione di inizializzazione, la funzione di controllo del motore date le velocità, ed una funzione di utilità che estrae le due velocità a partire da una stringa.

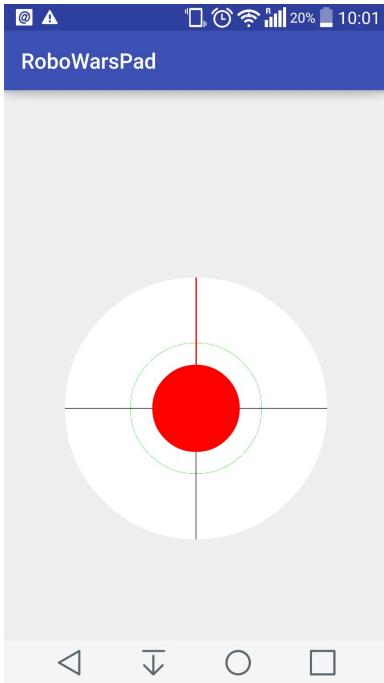


Figura 3.1: L’immagine mostra l’intercaccia grafica del controller Android realizzato per comandare il robot.

3.3 Modulo wireless

Per l’utilizzo della board con il sistema ChibiOS è stata realizzata una apposita libreria che racchiude diverse funzionalità (vedi A.1). Il modulo ESP8266 può essere configurato tramite comandi seriali (vedi 3.2). Per ogni comando, il modulo genera una risposta che può essere un ACK o un messaggio di errore. La libreria consente la configurazione del modulo come Access Point o come client (fornendo le credenziali di accesso ad una rete Wi-Fi) tramite due semplici funzioni. Chiamando qualsiasi di queste due funzioni, viene creato un Thread asincrono che rimane in ascolto di eventi sulla seriale a cui è collegato il modulo ESP8266. Nel caso specifico i moduli sono stati utilizzati come Access Point. I comandi fondamentali inviati in fase di setup sono:

- comando di reset: ”AT+RST”
- comando di setup della modalità di funzionamento: ”AT+CWMODE=2” (per modalità access point)
- comando di start del server: ”AT+CIPSERVER=1,80”;

Maggiori dettagli sui comandi disponibili sono mostrati nella tabella 3.2.

Quando viene inviata una richiesta http al modulo, viene generato un evento sulla seriale e viene costruita la stringa con il messaggio ricevuto. Tale messaggio è composto da una intestazione e da una cosa. Un parser del messaggio è stato realizzato per estrarre la stringa di interesse (es. M0123200); decodificato il messaggio viene invocato il metodo 'control_motor' della libreria 'MotorController.h' (vedi A.2) che prende in carico la richiesta e comanda i motori.

Oltre alla seriale utilizzata per comunicare con l'ESP8266, la libreria utilizza anche la seriale USB per stampare a video i messaggi ricevuti dal modulo in tempo reale. Tale meccanismo è utile in fase di debug per conoscere lo stato del modulo ed intercettare eventuali errori.

3.4 Visione Artificiale

ESP8266 AT Command Set

Function	AT Command	Response
Working	AT	OK
Restart	AT+RST	OK [System Ready, Vendor:www.ai-thinker.com]
Firmware version	AT+GMR	AT+GMR 0018000902 OK
List Access Points	AT+CWLAP	AT+CWLAP +CWLAP:(4,"RochefortSurLac",-38,"70:62:b8:6f:6d:58",1) +CWLAP:(4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1) OK
Join Access Point	AT+CWJAP? AT+CWJAP="SSID","Password"	Query AT+CWJAP? +CWJAP:"RochefortSurLac" OK
Quit Access Point	AT+CWQAP=? AT+CWQAP	Query OK
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105 OK
Set Parameters of Access Point	AT+ CWSAP? AT+ CWSAP= <ssid>,<pwd>,<chl>, <ecn>	Query ssid, pwd chl = channel, ecn = encryption
WiFi Mode	AT+CWMODE? AT+CWMODE=1 AT+CWMODE=2 AT+CWMODE=3	Query STA AP BOTH
Set up TCP or UDP connection	AT+CIPSTART=? (CIPMUX=0) AT+CIPSTART = <type>,<addr>,<port> (CIPMUX=1) AT+CIPSTART= <id><type>,<addr>,<port>	Query id = 0-4, type = TCP/UDP, addr = IP address, port= port
TCP/UDP Connections	AT+ CIPMUX? AT+ CIPMUX=0 AT+ CIPMUX=1	Query Single Multiple
Check join devices' IP	AT+CWLIF	
TCP/IP Connection Status	AT+CIPSTATUS	AT+CIPSTATUS? no this fun
Send TCP/IP data	(CIPMUX=0) AT+CIPSEND=<length>; (CIPMUX=1) AT+CIPSEND= <id>,<length>	
Close TCP / UDP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	
Set as server	AT+ CIPSERVER= <mode>[,<port>]	mode 0 to close server mode; mode 1 to open; port = port
Set the server timeout	AT+CIPSTO? AT+CIPSTO=<time>	Query <time>0~28800 in seconds
Baud Rate*	AT+CIOBAUD? Supported: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600	Query AT+CIOBAUD? +CIOBAUD:9600 OK
Check IP address	AT+CIFSR	AT+CIFSR 192.168.0.106 OK
Firmware Upgrade (from Cloud)	AT+CIUPDATE	1. +CIPUPDATE:1 found server 2. +CIPUPDATE:2 connect server 3. +CIPUPDATE:3 got edition 4. +CIPUPDATE:4 start update
Received data	+IPD	(CIPMUX=0): + IPD, <len>; (CIPMUX=1): + IPD, <id>, <len>; <data>
Watchdog Enable*	AT+CSYSWDTENABLE	Watchdog, auto restart when program errors occur: enable
Watchdog Disable*	AT+CSYSWDTDISABLE	Watchdog, auto restart when program errors occur: disable

* New in V0.9.2.2 (from <http://www.electrodragon.com/w/Wi07c>)

Figura 3.2: La tabella contiene i comandi riconosciuti dal modulo ESP8266 con relativa risposta.

Capitolo 4

Conclusioni

4.1 Sviluppi futuri

Il progetto sviluppato ha soddisfatto appieno i requisiti funzionali che ci siamo preposti. Tuttavia, alcuni aspetti potranno essere sicuramente migliorati in futuro:

- I cartoncini colorati potranno essere sostituiti da markers per la realtà aumentata. Ciò migliorerà il riconoscimento dei ragni ed eliminerà la necessità del doppio colore per stabilire l'orientamento del ragno inseguitore.
- L'algoritmo di inseguimento potrà essere migliorato pianificando traiettorie di curvatura e aggiungendo a bordo del ragno un sensore di prossimità.
- Si potrebbe eliminare la necessità di un sistema di orientamento globale montando una webcam direttamente sui ragni.

Appendice A

Librerie ChibiOS

A.1 Libreria ESP8266

In questo capitolo è riportata la libreria realizzata per il controllo del modulo ESP8266.h. La libreria è stata utilizzata nel codice caricato sulla board nucleo stm32f4.

```
1 /**
2  * @author Simone Romano - s.romano1992@gmail.com
3  */
4 #include "ch.h"
5 #include "hal.h"
6 #include "test.h"
7 #include "chprintf.h"
8
9 #define EOF '\377'
10#define WIFI_SERIAL &SD1
11#define MONITOR_SERIAL &SD2
12#define MAXLENGTH 100 //lunghezza max messaggi trasmessi dal
//modulo WiFi, si potrebbe ridurre
14#define TIME_IMMEDIATE ((systime_t)0)
15#define MSG_TIMEOUT (msg_t)-1
16#define Q_TIMEOUT MSG_TIMEOUT
17#define COMMAND_SLEEP 500
18#define COMMAND_LONG_SLEEP 20000
19
20char * readResponse(void);
21void printWebPage(void);
22int mystrcontains(char* text, char* toFind);
23void sendToESP8266(char* command, int delay);
24void readAndPrintResponse(void);
25int mystrlen(char* text);
26static void println(char *p);
27void blinkBoardLed(void);
28char* StrStr(const char *str, const char *target);
29char *strcat(char *dest, const char *src);
30char *strcpy(char *dest, const char *src);
31int strlen(const char * str);
```

```

32 void itoa(int n, char s[]) ;
33 void reverse(char s[]) ;
34
35 static SerialConfig uartCfgWiFi = {115200, // bit rate
36     } ;
37
38 static char* ESP8266_HELLO = "AT\r\n";
39 static char* ESP8266_RESET = "AT+RST\r\n";
40 static char* ESP8266_LIST_WIFI = "AT+CWLAP\r\n";
41 static char* ESP8266.CONNECT_TO_WIFI =
42     "AT+CWJAP=\"Romano Wi-Fi\",\"160462160867\"\r\n";
43 static char* ESP8266_CHECK_IP = "AT+CIFSR\r\n";
44 static char* ESP8266_GET_IP_ADD = "AT+CIFSR\r\n"; // get ip
45     address
46 static char* ESP8266_CHECK_VERSION = "AT+GMR\r\n";
47 static char* ESP8266_MULTIPLE_CONNECTION = "AT+CIPMUX=1\r\n";
48 static char* ESP8266_START_SERVER = "AT+CIPSERVER=1,80\r\n";
49 static char* ESP8266_SET_AS_ACCESS.POINT = "AT+CWMODE=2\r\n"; // configura
50                                         //come access point
51 static char* ESP8266_SET_AS_CLIENT = "AT+CWMODE=1\r\n";
52 static char* ESP8266_SEND_TCP_DATA = "AT+CIPSEND=";
53 static char* ESP8266_CLOSE_CONN = "AT+CIPCLOSE=";
54 char clientID[2];
55 char command[9]; //could be "M023120" for: motor left
56                         //to 23 and motor right to 120
57                         //else could be "PXXYY__" for position is
58     xx yy
59 char request;
60 static THD_WORKING_AREA(waThread1, 2048);
61 /**
62 * Asynchronous serial SD1
63 */
64 static msg_t Uart1EVT_Thread(void *p) {
65     int letterAfterPlus = 0;
66     int spaceAfterD = 0;
67     int x_charRead = 0, y_charRead = 0;
68     int BUFF_SIZE = 1024;
69     char received [BUFF_SIZE];
70     int pos = 0;
71     event_listener_t el1;
72     eventflags_t flags;
73
74     chEvtRegisterMask(chnGetEventSource(WIFI_SERIAL), &el1, 1);
75     while (TRUE) {
76         chEvtWaitOne(1);
77
78         chSysLock();
79         flags = chEvtGetAndClearFlagsI(&el1);
80         chSysUnlock(); //wait for events;
81
82         if (flags & CHN_INPUT_AVAILABLE) { //events received
83             msg_t charbuf;
84             do {

```

```

83     charbuf = chnGetTimeout(WIFI_SERIAL, TIME_IMMEDIATE) ;
84     chThdSleepMilliseconds(1) ;
85     if (charbuf != Q_TIMEOUT) {
86         chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%c",
87             (char)charbuf);
88         if (pos < BUFF_SIZE) {
89             received[pos] = (char)charbuf;
90             pos++;
91         }
92     } while (charbuf != Q_TIMEOUT) ;
93     received[pos] = '\0';
94     /******DO SOMETHING WITH RECEIVED MESSAGE***** */
95     char* clearRequest = StrStr(received, "+IPD"); //HTTP
REQUEST
96     if (StrStr(received, "+IPD") != NULL){
97         chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s",
98             " Received http request");
99         clientID[0] = clearRequest[5];
100        clientID[1] = '\0';
101        request = clearRequest[17]; //it is c for command (c=M0
... )
102        command[0] = clearRequest[19];
103        command[1] = clearRequest[20];
104        command[2] = clearRequest[21];
105        command[3] = clearRequest[22];
106        command[4] = clearRequest[23];
107        command[5] = clearRequest[24];
108        command[6] = clearRequest[25];
109        command[7] = clearRequest[26];
110        command[8] = '\0';
111        chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s", "
Client id=");
112        chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s\n",
clientID);
113        chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s", "
Command=");
114        chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s\n",
command);
115        printWebPage();
116    }
117    /******END***** */
118    pos = 0;
119 }
120 }
121 }
122 void blinkBoardLed() {
123     palSetPad(GPIOA, GPIOA_LED_GREEN);
124     chThdSleepMilliseconds(500);
125     palClearPad(GPIOA, GPIOA_LED_GREEN);
126 }
127 }
128

```

```

129 /**
130 * This function set the ESP8266 (connected to serial SD1)
131 * as access point and start a server on port 80.
132 * It will be ready to accept http request.
133 * NOTE: first to call this function , be sure that:
134 * -you started serial SD2 to read response and debug
135 * -you started serial SD1 to send request to ESP8266
136 * The code:
137 * sdStart(MONITOR_SERIAL, &uartCfgMonitor);
138 * palSetPadMode(GPIOA, 9, PAL_MODE_ALTERNATE(7));
139 * palSetPadMode(GPIOA, 10, PAL_MODE_ALTERNATE(7));
140 * sdStart(WIFI_SERIAL, &uartCfgWiFi);
141 * ESP8266_setAsAP();
142 */
143 void ESP8266_setAsAP(void) {
144     chThdCreateStatic(waThread1, sizeof(waThread1), NORMALPRI,
145                       Uart1EVT_Thread,
146                       NULL);
147     sendToESP8266(ESP8266_RESET, COMMAND_SLEEP);
148     sendToESP8266(ESP8266_SET_AS_ACCESS_POINT, COMMAND_SLEEP);
149     sendToESP8266(ESP8266_GET_IP_ADD, COMMAND_SLEEP);
150     sendToESP8266(ESP8266_MULTIPLE_CONNECTION, COMMAND_SLEEP);
151     sendToESP8266(ESP8266_START_SERVER, COMMAND_SLEEP);
152 }
153 /**
154 * This function set the ESP8266 (connected to serial SD1)
155 * as a client to the wifi connection defined in command
156 * ESP8266_CONNECT_TO_WIFI and start a server on port 80.
157 * It will be ready to accept http request.
158 * NOTE: first to call this function , be sure that:
159 * -you started serial SD2 to read response and debug
160 * -you started serial SD1 to send request to ESP8266
161 * The code:
162 * sdStart(MONITOR_SERIAL, &uartCfgMonitor);
163 * palSetPadMode(GPIOA, 9, PAL_MODE_ALTERNATE(7));
164 * palSetPadMode(GPIOA, 10, PAL_MODE_ALTERNATE(7));
165 * sdStart(WIFI_SERIAL, &uartCfgWiFi);
166 * ESP8266_setAsClient();
167 */
168 void ESP8266_setAsClient(void) {
169     chThdCreateStatic(waThread1, sizeof(waThread1), NORMALPRI,
170                       Uart1EVT_Thread,
171                       NULL);
172     sendToESP8266(ESP8266_RESET, COMMAND_SLEEP);
173     sendToESP8266(ESP8266_SET_AS_CLIENT, COMMAND_LONG_SLEEP);
174     sendToESP8266(ESP8266_MULTIPLE_CONNECTION, COMMAND_LONG_SLEEP)
175     ;
176     sendToESP8266(ESP8266_LIST_WIFI, COMMAND_LONG_SLEEP);
177     sendToESP8266(ESP8266_CONNECT_TO_WIFI, COMMAND_LONG_SLEEP);
178     sendToESP8266(ESP8266_START_SERVER, COMMAND_LONG_SLEEP);
179     sendToESP8266(ESP8266_CHECK_IP, COMMAND_LONG_SLEEP);
}

```

```

180 int mystrlen(char* text) {
181     int length = 0;
182     while (true) {
183         if (text[length] == '\0')
184             return length;
185         length++;
186     }
187 }
188
189 /**
190 * This function send a command on serial port SD1 to
191 * ESP8266. You can listen for the response by calling
192 * readAndPrintResponse() function.
193 */
194 void sendToESP8266(char* command, int delay) {
195     chprintf((BaseChannel *)WIFI_SERIAL, command);
196     chThdSleepMilliseconds(delay);
197 }
198
199 /**
200 * This method reads input from SD1 and print on Serial usb
201 * monitor (SD2).
202 */
203 void readAndPrintResponse() {
204     char buff[1];
205     int pos = 0;
206     char charbuf;
207     while (true) {
208         charbuf = chnGetTimeout(WIFI_SERIAL, TIME_IMMEDIATE);
209         if (charbuf == EOF) {
210             break;
211         }
212         buff[0] = charbuf;
213         chprintf((BaseChannel *)MONITOR_SERIAL, buff, 1);
214     }
215     buff[0] = '\0';
216     chprintf((BaseChannel *)MONITOR_SERIAL, '\0', 1);
217 }
218
219 void printWebPage() {
220     char cipSend[100] = {"AT+CIPSEND="};
221     /*char webView[512] = {"<html><head></head><body>"}
222      "<form>"*/
223     /*<input type='text' name='c'></input>"*/
224     /*<input type='submit' value='Send'>"""
225     "</form>"};*/
226     char webView[600] = {"<html><head></head><body><table border
227     =0>"}
228     "<tr><td></td><td><form><input type='hidden' name='c' value='
229     M0127127'>
228     </input><input type='submit' value='^'></form></td><td></td></tr
229     >"}
229     "<tr><td><form><input type='hidden' name='c' value='M0255255'><
229     input>
```

```

230 <input type='submit' value='<'></form></td><td></td><td><form>
231 <input type='hidden' name='c' value='M0127000'></input><input
232     type='submit'
233     value='>'></form></td></tr>""
234 "<tr><td></td><td><form><input type='hidden' name='c' value='
235     M0000127'>
236 </input><input type='submit' value='-'></form></td><td></td></tr>
237 ></table>";
238 };
239 char webPage1[20] = {"</body></html>"};
240 if (request == 'c')
241     strcat(webPage, command);
242 strcat(webPage, webPage1);
243 strcat(cipSend, clientID);
244 strcat(cipSend, ", ");
245 int pageLength = strlen(webPage);
246 char pageLengthAsString[100];
247 itoa(pageLength, pageLengthAsString);
248 strcat(cipSend, pageLengthAsString);
249 strcat(cipSend, "\r\n");
250 sendToESP8266(cipSend, COMMAND_SLEEP);
251 sendToESP8266(webPage, COMMAND_SLEEP);
252 char closeCommand[40];
253 strcpy(closeCommand, ESP8266_CLOSE_CONN);
254 strcat(closeCommand, clientID);
255 strcat(closeCommand, "\r\n");
256 sendToESP8266(closeCommand, COMMAND_SLEEP);
257 }
258 /**
259 * Return length of str as char (es. '47')
260 */
261 int strlen(const char * str){
262     int len;
263     for (len = 0; str[len]; len++);
264     return len;
265 }
266 char *strcpy(char *dest, const char *src){
267     unsigned i;
268     for (i=0; src[i] != '\0'; ++i)
269         dest[i] = src[i];
270     dest[i] = '\0';
271     return dest;
272 }
273 char* StrStr(const char *str, const char *target) {
274     if (!*target) return str;
275     char *p1 = (char*)str, *p2 = (char*)target;
276     char *p1Adv = (char*)str;
277     while (*++p2)
278         p1Adv++;
279     while (*p1Adv) {
280         char *p1Begin = p1;
281         p2 = (char*)target;

```

```

281     while (*p1 && *p2 && *p1 == *p2) {
282         p1++;
283         p2++;
284     }
285     if (!*p2)
286         return p1Begin;
287     p1 = p1Begin + 1;
288     p1Adv++;
289 }
290 return NULL;
291 }
292 char *strcat(char *dest, const char *src){
293     size_t i,j;
294     for (i = 0; dest[i] != '\0'; i++)
295         ;
296     for (j = 0; src[j] != '\0'; j++)
297         dest[i+j] = src[j];
298     dest[i+j] = '\0';
299     return dest;
300 }
301 static void println(char *p) {
302
303     while (*p) {
304         chSequentialStreamPut(MONITOR_SERIAL, *p++);
305     }
306     chSequentialStreamWrite(MONITOR_SERIAL, (uint8_t *)"\r\n", 2)
307     ;
308 }
309 /* itoa: convert n to characters in s */
310 void itoa(int n, char s[]){
311     int i, sign;
312     if ((sign = n) < 0) /* record sign */
313         n = -n;           /* make n positive */
314     i = 0;
315     do {                /* generate digits in reverse order */
316         s[i++] = n % 10 + '0'; /* get next digit */
317     } while ((n /= 10) > 0); /* delete it */
318     if (sign < 0)
319         s[i++] = '-';
320     s[i] = '\0';
321     reverse(s);
322 }
323 /* reverse: reverse string s in place */
324 void reverse(char s[]){
325     int i, j;
326     char c;
327
328     for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
329         c = s[i];
330         s[i] = s[j];
331         s[j] = c;
332     }
333 }
```

334 }

images/ESP8266.h

A.2 Codice motori

In questo capitolo è riportata la libreria realizzata per il controllo dei motori. La libreria ESP8266.h riceve e decodifica il messaggio e invoca la funzione di controllo dei motori che traduce il comando in segnali pwm per il driver dei motori.

```

1 void (*functioPtrLeftUP)() ;
2 void (*functioPtrLeftDOWN)() ;
3 void (*functioPtrRightUP)() ;
4 void (*functioPtrRightDOWN)() ;

5
6 //1 positivo motore sinistro
7 //2 negativo motore sinistro
8
9 //3 positivo motore destro
10 //4 negativo motore destro
11
12 static struct Mapping_GPIO {
13     stm32_gpio_t * type1;
14     unsigned int port1;
15
16     stm32_gpio_t * type2;
17     unsigned int port2;
18
19     stm32_gpio_t * type3;
20     unsigned int port3;
21
22     stm32_gpio_t * type4;
23     unsigned int port4;
24 } mapping;
25
26 static void Sinistra_Avanti_up() {
27     palClearPad(mapping.type2, mapping.port2);
28     palSetPad(mapping.type1, mapping.port1);
29 }
30
31 static void Sinistra_Avanti_Down() {
32     palClearPad(mapping.type2, mapping.port2);
33     palClearPad(mapping.type1, mapping.port1);
34 }
35
36 static void Sinistra_Dietro_up() {
37     palClearPad(mapping.type1, mapping.port1);
38     palSetPad(mapping.type2, mapping.port2);
39 }
40
41 static void Sinistra_Dietro_Down() {
```

```

42    palClearPad(mapping.type1, mapping.port1);
43    palClearPad(mapping.type2, mapping.port2);
44}
45
46 static void Destra_Avanti_up() {
47    palSetPad(mapping.type3, mapping.port3);
48    palClearPad(mapping.type4, mapping.port4);
49}
50
51 static void Destra_Avanti_Down() {
52    palClearPad(mapping.type3, mapping.port3);
53    palClearPad(mapping.type4, mapping.port4);
54}
55
56 static void Destra_Dietro_up() {
57    palClearPad(mapping.type3, mapping.port3);
58    palSetPad(mapping.type4, mapping.port4);
59}
60
61 static void Destra_Dietro_Down() {
62    palClearPad(mapping.type3, mapping.port3);
63    palClearPad(mapping.type4, mapping.port4);
64}
65
66 //pwm callbacks for left engine
67 static void pwmpcb(PWMDDriver *pwmp) {
68
69    (void)pwmp;
70    (*functioPtrLeftDOWN)();
71}
72
73 static void pwmclcb(PWMDDriver *pwmp) {
74
75    (void)pwmp;
76    (*functioPtrLeftUP)();
77}
78
79 //pwm callbacks for right engine
80 static void pwm2pcb(PWMDDriver *pwmp) {
81
82    (void)pwmp;
83    (*functioPtrRightDOWN)();
84}
85
86 static void pwm2c1cb(PWMDDriver *pwmp) {
87
88    (void)pwmp;
89    (*functioPtrRightUP)();
90}
91
92 //configuration for left engine
93 static PWMConfig pwm1cfg = {200000, /* 10kHz PWM clock frequency
94 . */
95 . 1024, /* Initial PWM period 1S.

```

```

      */
95      pwm1cb}, {
96          pwmpcb, { {PWM_OUTPUT_ACTIVE_HIGH,
97                      PWM_OUTPUT_DISABLED, NULL},
98                      {PWM_OUTPUT_DISABLED, NULL
99          },
100                     PWM_OUTPUT_DISABLED,
101                     NULL} },
102                     0, 0};

103 // configuration for right engine
104 static PWMConfig pwm2cfg = {200000, /* 10kHz PWM clock frequency
105 . */
106 . */
107 . */
108     pwm2cb}, {
109     pwmpcb, { {PWM_OUTPUT_ACTIVE_HIGH,
110                     PWM_OUTPUT_DISABLED, NULL},
111                     {PWM_OUTPUT_DISABLED, NULL
112     },
113                     PWM_OUTPUT_DISABLED,
114                     NULL} },
115                     0, 0};

116 void parse_string(char *command, int *velocity) {
117     char left [3];
118     char right [3];
119     int toret [2];
120     char type = command [0];
121
122     left [0] = command [2];
123     left [1] = command [3];
124     left [2] = command [4];
125
126     right [0] = command [5];
127     right [1] = command [6];
128     right [2] = command [7];
129
130     int le , ri;
131
132     le = atoi(left);
133     ri = atoi(right);
134
135     velocity [0] = le ;
136     velocity [1] = ri ;
137
138 }
139
140 void init_motor() {
141     mapping.type1 = GPIOA;
142     mapping.port1 = GPIOA_PIN8;
143
144     mapping.type2 = GPIOB;
145     mapping.port2 = GPIOB_PIN10;

```

```

140
141 mapping.type3 = GPIOB;
142 mapping.port3 = GPIOB_PIN4;
143
144 mapping.type4 = GPIOB;
145 mapping.port4 = GPIOB_PIN5;
146
147 palSetPadMode(mapping.type1, mapping.port1,
148                 PAL_MODE_OUTPUT_PUSHPULL |
149                 PAL_STM32_OSPEED_HIGHEST);
150 palClearPad(mapping.type1, mapping.port1);
151 palSetPadMode(mapping.type2, mapping.port2,
152                 PAL_MODE_OUTPUT_PUSHPULL |
153                 PAL_STM32_OSPEED_HIGHEST);
154 palClearPad(mapping.type2, mapping.port2);
155 palSetPadMode(mapping.type3, mapping.port3,
156                 PAL_MODE_OUTPUT_PUSHPULL |
157                 PAL_STM32_OSPEED_HIGHEST);
158 palClearPad(mapping.type3, mapping.port3);
159 palSetPadMode(mapping.type4, mapping.port4,
160                 PAL_MODE_OUTPUT_PUSHPULL |
161                 PAL_STM32_OSPEED_HIGHEST);
162 palClearPad(mapping.type4, mapping.port4);
163 functioPtrLeftUP = &Sinistra_Avanti_up;
164 functioPtrLeftDOWN = &Sinistra_Avanti_Down;
165
166 functioPtrRightUP = &Destra_Avanti_up;
167 functioPtrRightDOWN = &Destra_Avanti_Down;
168
169 // start pwm1 (left engine)
170 pwmStart(&PWMD1, &pwm1cfg);
171 pwmEnablePeriodicNotification(&PWMD1);
172 pwmEnableChannel(&PWMD1, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD1, 0)
173     );
174 pwmEnableChannelNotification(&PWMD1, 0);
175
176 // start pwm2 (right engine)
177 pwmStart(&PWMD3, &pwm2cfg);
178 pwmEnablePeriodicNotification(&PWMD3);
179 pwmEnableChannel(&PWMD3, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD3, 0)
180     );
181 pwmEnableChannelNotification(&PWMD3, 0);
182 }
183
184 void control_motor(char* command) {
185
186     int velocity[2];
187     parse_string(command, velocity);
188     int pwml = 1, pwm2 = 1;
189
190     /*****DEMO*****/
191     //PIN D7-D6
192     /*int vel = 0;
193     functioPtrLeftUP = &Sinistra_Avanti_up;
```

```

188 functioPtrLeftDOWN = &Sinistra_Avanti_Down ;
189 while ( vel < 127) {
190     pwm1 = 8 * vel + 1;
191     chprintf(MONITOR_SERIAL, "%d - ", pwm1);
192     pwmEnableChannel(&PWMD1, 0 , pwm1);
193     vel++;
194     chThdSleepMilliseconds(10);
195 }
196 while ( vel > 0) {
197     pwm1 = 8 * vel + 1;
198     chprintf(MONITOR_SERIAL, "%d - ", pwm1);
199     pwmEnableChannel(&PWMD1, 0 , pwm1);
200     vel--;
201     chThdSleepMilliseconds(10);
202 }
203
204 functioPtrLeftUP = &Sinistra_Dietro_up ;
205 functioPtrLeftDOWN = &Sinistra_Dietro_Down ;
206 while ( vel < 127) {
207     pwm1 = 8 * vel + 1;
208     chprintf(MONITOR_SERIAL, "%d - ", pwm1);
209     pwmEnableChannel(&PWMD1, 0 , pwm1);
210     vel++;
211     chThdSleepMilliseconds(10);
212 }
213 while ( vel > 0) {
214     pwm1 = 8 * vel + 1;
215     chprintf(MONITOR_SERIAL, "%d - ", pwm1);
216     pwmEnableChannel(&PWMD1, 0 , pwm1);
217     vel--;
218     chThdSleepMilliseconds(10);
219 }
220
221 //PIN D5-D4
222 vel = 0;
223 functioPtrRightUP = &Destra_Avanti_up ;
224 functioPtrRightDOWN = &Destra_Avanti_Down ;
225 while ( vel < 127) {
226     pwm1 = 8 * vel + 1;
227     chprintf(MONITOR_SERIAL, "%d - ", pwm1);
228     pwmEnableChannel(&PWMD3, 0 , pwm1);
229     vel++;
230     chThdSleepMilliseconds(10);
231 }
232 while ( vel > 0) {
233     pwm1 = 8 * vel + 1;
234     chprintf(MONITOR_SERIAL, "%d - ", pwm1);
235     pwmEnableChannel(&PWMD3, 0 , pwm1);
236     vel--;
237     chThdSleepMilliseconds(10);
238 }
239
240 functioPtrRightUP = &Destra_Dietro_up ;
241 functioPtrRightDOWN = &Destra_Dietro_Down ;

```

```

242 while (vel < 127) {
243     pwm1 = 8 * vel + 1;
244     chprintf(MONITOR SERIAL, "%d - ", pwm1);
245     pwmEnableChannel(&PWMD3, 0, pwm1);
246     vel++;
247     chThdSleepMilliseconds(10);
248 }
249 while (vel > 0) {
250     pwm1 = 8 * vel + 1;
251     chprintf(MONITOR SERIAL, "%d - ", pwm1);
252     pwmEnableChannel(&PWMD3, 0, pwm1);
253     vel--;
254     chThdSleepMilliseconds(10);
255 }
256 /******END DEMO******/
257
258 if (velocity [0] >= 128) {
259     velocity [0] = velocity [0] - 128;      // [0-127]
260     functioPtrLeftUP = &Sinistra_Avanti_up;
261     functioPtrLeftDOWN = &Sinistra_Avanti_Down;
262
263     pwm1 = 8 * velocity [0] + 1;
264     chprintf(MONITOR SERIAL, "%d - ", pwm1);
265
266     pwmEnableChannel(&PWMD1, 0, pwm1);
267 }
268 else { // <128
269     functioPtrLeftUP = &Sinistra_Dietro_up;
270     functioPtrLeftDOWN = &Sinistra_Dietro_Down;
271
272     pwm1 = 8 * velocity [0] + 1;
273     chprintf(MONITOR SERIAL, "%d - ", pwm1);
274
275     pwmEnableChannel(&PWMD1, 0, pwm1);
276 }
277
278 if (velocity [1] >= 128) {
279     velocity [1] = velocity [1] - 128;      // [0-127]
280     functioPtrRightUP = &Destra_Avanti_up;
281     functioPtrRightDOWN = &Destra_Avanti_Down;
282
283     pwm2 = 8 * velocity [1] + 1;
284     chprintf(MONITOR SERIAL, "%d - ", pwm1);
285
286     pwmEnableChannel(&PWMD3, 0, pwm2);
287 }
288 else {
289     functioPtrRightUP = &Destra_Dietro_up;
290     functioPtrRightDOWN = &Destra_Dietro_Down;
291
292     pwm2 = 8 * velocity [1] + 1;
293     chprintf(MONITOR SERIAL, "%d - ", pwm1);
294
295     pwmEnableChannel(&PWMD3, 0, pwm2);

```

```
296 }  
297 }  
298 }
```

..../RoboWars/MotorController.h

Appendice B

Visione artificiale

Il codice seguente è stato realizzato per il robot 'inseguitore'. Degli algoritmi di visione artificiale consentono di localizzare il robot da inseguire; a questo punto opportuni comandi di movimento vengono inviati al robot 'inseguitore' per raggiunger il robot 'inseguito'.

```
1 import numpy as np
2 import argparse
3 import imutils
4 import cv2
5 import subprocess
6 import time
7 from multiprocessing import Process
8
9 class Vision:
10
11     def __init__(self):
12
13         #arancione
14         self.first_spider_color1 = np.uint8([[57,114,255]])
15         #verde
16         self.first_spider_color2 = np.uint8([[102,151,49]])
17         #fucsia
18         #self.first_spider_color2 = np.uint8([[180,95,245]])
19
20         #azzurro
21         self.second_spider_color1 = np.uint8([[216,145,0]])
22         #rosso
23         #self.second_spider_color1 = np.uint8([[78,63,223]])
24         #giallo
25         self.second_spider_color2 = np.uint8([[0,255,255]])
26
27
28         f_s_c1HSV = cv2.cvtColor(self.first_spider_color1, cv2.
29 COLOR_BGR2HSV)
30         self.f_s_c1Lower = np.array((f_s_c1HSV[0][0][0]-10,100,100))
31         self.f_s_c1Upper = np.array((f_s_c1HSV[0][0][0]+10,255,255))
```

```

32
33     f_s_c2HSV = cv2.cvtColor(self.first_spider_color2, cv2.
34                               COLOR_BGR2HSV)
35     self.f_s_c2Lower = np.array((f_s_c2HSV[0][0][0] - 10, 100, 100))
36     self.f_s_c2Upper = np.array((f_s_c2HSV[0][0][0] + 10, 255, 255))
37
38     s_s_c1HSV = cv2.cvtColor(self.second_spider_color1, cv2.
39                               COLOR_BGR2HSV)
40     self.s_s_c1Lower = np.array((s_s_c1HSV[0][0][0] - 10, 100, 100))
41     self.s_s_c1Upper = np.array((s_s_c1HSV[0][0][0] + 10, 255, 255))
42
43     s_s_c2HSV = cv2.cvtColor(self.second_spider_color2, cv2.
44                               COLOR_BGR2HSV)
45     self.s_s_c2Lower = np.array((s_s_c2HSV[0][0][0] - 10, 100, 100))
46     self.s_s_c2Upper = np.array((s_s_c2HSV[0][0][0] + 10, 255, 255))
47
48     self.camera = cv2.VideoCapture(0)
49     # video recorder
50
51     grabbed = False
52     while not grabbed:
53         (grabbed, frame) = self.camera.read()
54         #print("No frame D:")
55
56         frame = imutils.resize(frame, width=900)
57         h = frame.shape[0]
58         #print h
59         # resize the frame, blur it, and convert it to the HSV
60         # color space
61         fourcc = cv2.cv.CV_FOURCC('m', 'p', '4', 'v')
62         self.video_writer = cv2.VideoWriter("output.avi", fourcc,
63                                             12, (900, h))
64         if not self.video_writer:
65             print "!!! Failed VideoWriter: invalid parameters"
66             sys.exit(1)
67
68     def close_all(self):
69         # cleanup the camera and close any open windows
70         self.camera.release()
71         self.video_writer.release()
72         cv2.destroyAllWindows()
73
74     def get_Spider(self, color1Lower, color1Upper, color2Lower,
75                   color2Upper):
76
77         (grabbed, frame) = self.camera.read()
78         #frame = cv2.flip(frame,1)
79         s_x = None
80         s_y = None
81         f_x = None
82         f_y = None
83         # if we are viewing a video and we did not grab a frame,
84         # then we have reached the end of the video

```

```

81 if not grabbed:
82     print("No frame D:")
83
84 # resize the frame, blur it, and convert it to the HSV
85 # color space
86 frame = imutils.resize(frame, width=900)
87 blurred = cv2.GaussianBlur(frame, (11, 11), 0)
88 #cv2.imshow(" Blurred", blurred)
89 hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
90 cv2.imshow("hsv", hsv)
91
92 # construct a mask for the color "green", then perform
93 # a series of dilations and erosions to remove any small
94 # blobs left in the mask
95 mask = cv2.inRange(hsv, color1Lower, color1Upper)
96 mask = cv2.erode(mask, None, iterations=3)
97 mask = cv2.dilate(mask, None, iterations=2)
98
99 # cv2.imshow(" mask1",cv2.bitwise_and(hsv,hsv,mask=mask));
100
101 # find contours in the mask and initialize the current
102 # (x, y) center of the ball
103 cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
104 cv2.CHAIN_APPROX_SIMPLE)[-2]
105 center = None
106 # only proceed if at least one contour was found
107 if len(cnts) > 0:
108     # find the largest contour in the mask, then use
109     # it to compute the minimum enclosing circle and
110     # centroid
111     c = max(cnts, key=cv2.contourArea)
112     ((f_x, f_y), radius) = cv2.minEnclosingCircle(c)
113     M = cv2.moments(c)
114     center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
115
116     # only proceed if the radius meets a minimum size
117     if radius > 5:
118         # draw the circle and centroid on the frame,
119         # then update the list of tracked points
120         cv2.circle(frame, (int(f_x), int(f_y)), int(radius),
121         (0, 255, 255), 2)
122         cv2.circle(frame, center, 5, (0, 0, 255), -1)
123
124
125 mask = cv2.inRange(hsv, color2Lower, color2Upper)
126 mask = cv2.erode(mask, None, iterations=3)
127 mask = cv2.dilate(mask, None, iterations=2)
128
129 # cv2.imshow(" mask2",cv2.bitwise_and(hsv,hsv,mask=mask));
130
131 # find contours in the mask and initialize the current
132 # (x, y) center of the ball
133 cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,

```

```

134     cv2.CHAIN_APPROX_SIMPLE)[-2]
135     center = None
136     # only proceed if at least one contour was found
137     if len(cnts) > 0:
138         # find the largest contour in the mask, then use
139         # it to compute the minimum enclosing circle and
140         # centroid
141         c = max(cnts, key=cv2.contourArea)
142         ((s_x, s_y), radius) = cv2.minEnclosingCircle(c)
143         M = cv2.moments(c)
144         center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"])
145             ))
146
147         # only proceed if the radius meets a minimum size
148         if radius > 5:
149             # draw the circle and centroid on the frame,
150             # then update the list of tracked points
151             cv2.circle(frame, (int(s_x), int(s_y)), int(radius),
152                 (255, 255, 0), 2)
153             cv2.circle(frame, center, 5, (0, 0, 255), -1)
154
155             cv2.imshow("Frame", frame)
156             self.video_writer.write(frame)
157
158         if f_x and f_y and s_x and s_y:
159             p0 = np.array([f_x, f_y])
160             p1 = np.array([s_x, s_y])
161
162             points = dict()
163
164             points["p0"] = p0
165             points["p1"] = p1
166
167             return [points, self.calculate_matrix(p0, p1)]
168
169     def get_Spider_Inseguitore(self, color1Lower, color1Upper):
170
171         (grabbed, frame) = self.camera.read()
172         #frame = cv2.flip(frame, 1)
173         s_x = None
174         s_y = None
175         f_x = None
176         f_y = None
177         # if we are viewing a video and we did not grab a frame,
178         # then we have reached the end of the video
179         if not grabbed:
180             print("No frame D:")
181
182         # resize the frame, blur it, and convert it to the HSV
183         # color space
184         frame = imutils.resize(frame, width=900)
185         blurred = cv2.GaussianBlur(frame, (11, 11), 0)
186         #cv2.imshow(" Blurred", blurred)

```

```

187     hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
188     # cv2.imshow("hsv", hsv)
189
190     # construct a mask for the color "green", then perform
191     # a series of dilations and erosions to remove any small
192     # blobs left in the mask
193     mask = cv2.inRange(hsv, color1Lower, color1Upper)
194     mask = cv2.erode(mask, None, iterations=3)
195     mask = cv2.dilate(mask, None, iterations=2)
196
197     # cv2.imshow("mask1",cv2.bitwise_and(hsv,hsv,mask=mask));
198
199     # find contours in the mask and initialize the current
200     # (x, y) center of the ball
201     cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
202     cv2.CHAIN_APPROX_SIMPLE)[-2]
203     center = None
204
205     # only proceed if at least one contour was found
206     if len(cnts) > 0:
207         # find the largest contour in the mask, then use
208         # it to compute the minimum enclosing circle and
209         # centroid
210         c = max(cnts, key=cv2.contourArea)
211         ((f_x, f_y), radius) = cv2.minEnclosingCircle(c)
212         M = cv2.moments(c)
213         center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
214
215         # only proceed if the radius meets a minimum size
216         if radius > 5:
217             # draw the circle and centroid on the frame,
218             # then update the list of tracked points
219             cv2.circle(frame, (int(f_x), int(f_y)), int(radius),
220             (0, 255, 255), 2)
221             cv2.circle(frame, center, 5, (0, 0, 255), -1)
222
223
224     cv2.imshow("Frame", frame)
225     self.video_writer.write(frame)
226
227     if f_x and f_y:
228         p1 = np.array([f_x, f_y])
229         return p1
230
231     #dati due punti restituisce la matrice omogenea di
232     #rototraslazione
233     def calculate_matrix(self, p0, p1, versor=[1,0]):
234
235         vet_diff = p1 - p0
236         x_axis = np.array(versor)
237         dot_product = np.dot(vet_diff, x_axis)
238         module = np.linalg.norm(vet_diff)
         cos_arg = dot_product/module

```

```

239     angle = np.arccos(cos_arg)
240
241     #print(vet_diff, cos_arg, np.degrees(angle))
242
243     rot_matrix = [cos_arg, -np.sin(angle), np.sin(angle),
244     cos_arg]
244     centr_vet = [(p0[0]+p1[0])/2, (p0[1]+p1[1])/2]
245     matrix = np.array([rot_matrix[0], rot_matrix[1], centr_vet
246     [0], rot_matrix[2], rot_matrix[3], centr_vet[1], 0, 0, 1])
246     matrix = np.reshape(matrix, (3,3))
247     #print(matrix)
248     return matrix
249
250
251 def launch_curl(string):
252     subprocess.call("curl -m 1 http://192.168.4.1/?c=m0"+string,
253     shell=True)
254
255 if __name__ == '__main__':
256
256     g = Vision()
257
258     stop = True
259     old_command = "stop"
260
261     while True:
262         returns = g.get_Spider(g.f_s_c1Lower, g.f_s_c1Upper, g.
262         f_s_c2Lower, g.f_s_c2Upper)
263         point = g.get_Spider_Inseguitore(g.s_s_c1Lower, g.
263         s_s_c1Upper)
264
265         key = cv2.waitKey(1) & 0xFF
266         if key == ord("q"):
267             g.close_all()
268             print("BYEEEEEE")
269             break
270
271         if returns is not None and point is not None and returns[0]
271         is not None and returns[1] is not None:
272             #print(first_matrix, second_matrix)
273
274             #print(first_matrix[0][0])
275
276             print(old_command)
277             points = returns[0]
278             first_matrix = returns[1]
279
280             p0 = [first_matrix[0][2], first_matrix[1][2]]
281             p1 = point
282
283             new_matrix_x= g.calculate_matrix(np.array(p0), p1)
284             new_matrix_y= g.calculate_matrix(np.array(p0), p1, [0,1])
285
286             cos_x = new_matrix_x[0][0]

```

```

287 cos_y = new_matrix_y[0][0]
288
289 #print("Coseno rispetto X: "+ str(cos_x))
290 #print("Coseno rispetto Y: "+ str(cos_y))
291
292 #epsilon di rotazione del robot rispetto alla telecamera
293 epsilon_rot_robot = 0.8
294 #epsilon di rotazione dell'avversario rispetto al robot
295 epsilon_rot_enemy = 0.5
296 #epsilon di rotazione dell'avversario rispetto al robot
297 #rispetto all'asse y
298 epsilon_fron = 0.5
299 #epsilon di stop
300 epsilon_stop = 230
301
302 diff = np.linalg.norm(p1-p0, ord = 2)
303
304 #print(diff)
305 if diff < epsilon_stop:
306     if old_command != "stop":
307         old_command = "stop"
308         print("FERMATIIIIII")
309         p = Process(target=launch_curl, args=(128128, ))
310         p.start()
311
312     continue
313
314 if abs(first_matrix[0][0]) <= epsilon_rot_robot:
315
316     diff = points["p0"] - points["p1"]
317
318     if diff[1] < 0:
319         #print("Arancione Davanti")
320
321         if cos_x > epsilon_rot_enemy:
322             print("Vai a Destra__Verticale__Arancione")
323             if old_command != "right":
324                 old_command = "right"
325                 p = Process(target=launch_curl, args=(000255, ))
326                 p.start()
327             continue
328         elif cos_x < -epsilon_rot_enemy:
329             print("Vai a Sinistra__Verticale__Arancione")
330             if old_command != "left":
331                 old_command = "left"
332                 p = Process(target=launch_curl, args=(255000, ))
333                 p.start()
334             continue
335
336         if cos_y > epsilon_fron:
337             print("Vai a Indietro__Verticale__Arancione")
338             if old_command != "back":
339                 old_command = "back"

```

```

340     p.start()
341     continue
342 elif cos_y < -epsilon_fron:
343     print("Vai a Avanti_Verticale_Arancione")
344     if old_command != "front":
345         old_command = "front"
346         p = Process(target=launch_curl, args=(255255, ))
347         p.start()
348         continue
349 else:
350     #print("Verde Davanti")
351     if cos_x > epsilon_rot_enemy:
352         if old_command != "left":
353             old_command = "left"
354             print("Vai a Sinistra_Verticale_Arancione")
355             p = Process(target=launch_curl, args=(255000, ))
356             p.start()
357             continue
358 elif cos_x < -epsilon_rot_enemy:
359     if old_command != "right":
360         old_command = "right"
361         print("Vai a Destra_Verticale_Arancione")
362         p = Process(target=launch_curl, args=(000255, ))
363         p.start()
364         continue
365
366 if cos_y > epsilon_fron:
367     if old_command != "front":
368         old_command = "front"
369         p = Process(target=launch_curl, args=(255255, ))
370         p.start()
371         print("Vai a Avanti_Verticale_Arancione")
372         continue
373 elif cos_y < -epsilon_fron:
374     if old_command != "back":
375         old_command = "back"
376         p = Process(target=launch_curl, args=(000000, ))
377         p.start()
378         print("Vai a Indietro_Verticale_Arancione")
379         continue
380 else:
381     diff = points["p0"] - points["p1"]
382
383 if diff[0] < 0:
384     #print("Arancione Sinistra")
385     if cos_x > epsilon_rot_enemy:
386         if old_command != "back":
387             old_command = "back"
388             print("Vai a Indietro_Orizzontale_Arancione")
389             p = Process(target=launch_curl, args=(000000, ))
390             p.start()
391             continue
392 elif cos_x < -epsilon_rot_enemy:
393     if old_command != "front":

```

```

394     old_command = "front"
395     print("Vai a Avanti__Orizzontale__Arancione")
396     p = Process(target=launch_curl ,args=( '255255' , ))
397     p.start()
398     continue
399
400     if cos_y > epsilon_fron:
401         if old_command != "left":
402             old_command = "left"
403             p = Process(target=launch_curl ,args=( '255000' , ))
404             p.start()
405             print("Vai a Sinistra__Orizzontale__Arancione")
406             continue
407         elif cos_y < -epsilon_fron:
408             if old_command != "right":
409                 old_command = "right"
410                 p = Process(target=launch_curl ,args=( '000255' , ))
411                 p.start()
412                 print("Vai a Destra__Orizzontale__Arancione")
413                 continue
414     else:
415         #print("Verde Sinistra")
416         if cos_x > epsilon_rot_enemy:
417             if old_command != "front":
418                 old_command = "front"
419                 print("Vai a Avanti__Orizzontale__Verde")
420                 p = Process(target=launch_curl ,args=( '255255' , ))
421                 p.start()
422                 continue
423             elif cos_x < -epsilon_rot_enemy:
424                 if old_command != "back":
425                     old_command = "back"
426                     print("Vai a Indietro__Orizzontale__Arancione")
427                     p = Process(target=launch_curl ,args=( '000000' , ))
428                     p.start()
429                     continue
430
431             if cos_y > epsilon_fron:
432                 if old_command != "right":
433                     old_command = "right"
434                     p = Process(target=launch_curl ,args=( '000255' , ))
435                     p.start()
436                     print("Vai a Destra__Orizzontale__Arancione")
437                     continue
438             elif cos_y < -epsilon_fron:
439                 if old_command != "left":
440                     old_command = "left"
441                     p = Process(target=launch_curl ,args=( '255000' , ))
442                     p.start()
443                     print("Vai a Sinistra__Orizzontale__Arancione")
444                     continue
445
446     else:
447

```

```
448     print("Missing one of the components")
449     if old_command != "stop":
450         old_command = "stop"
451         p = Process(target=launch_curl, args=( '128128' , ))
452         p.start()
453     continue
```

..../Vision/Vision.py