

Spider-chase

Marco Mecchia
Luigi Giugliano
Simone Romano

Indice

1	Introduzione	3
1.1	Requisiti funzionali	3
2	Architettura e componenti	3
2.1	Ragni meccanici	4
2.2	STM32F401RE Nucleo	6
2.3	Driver motore	6
2.4	Modulo wireless	7
3	Dettagli implementativi	9
3.1	Controller	9
3.1.1	Android Controller	9
3.1.2	RoboWarsApp	9
3.2	Driver motore	9
3.3	Modulo wireless	16
3.4	Visione Artificiale	24
4	Conclusioni	33
4.1	Sviluppi futuri	33

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.

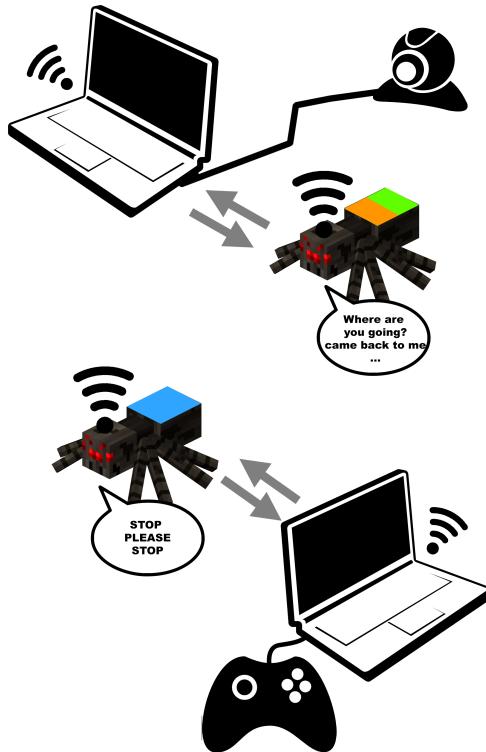
2 Architettura e componenti

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.

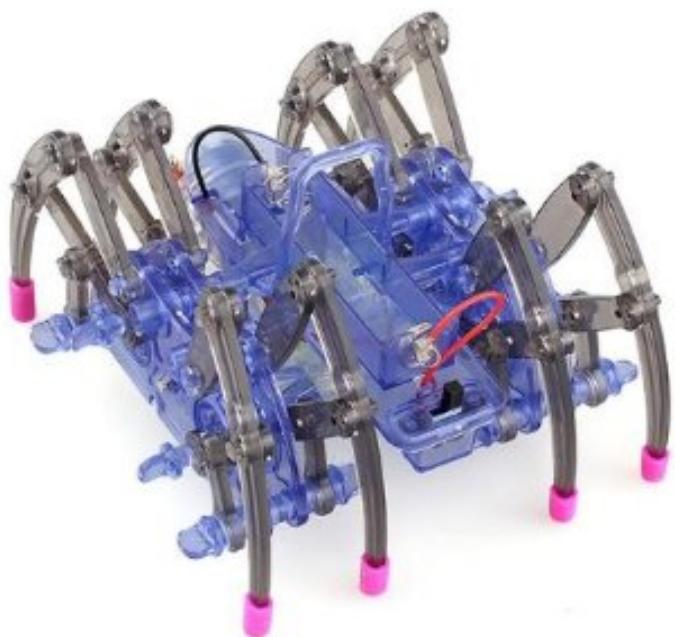
L'architettura totale pianificata é mostrata in figura.



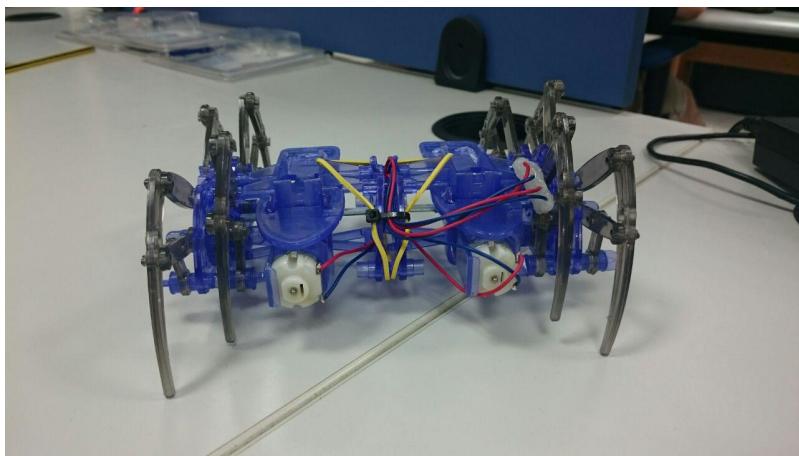
Tutti e tre i ragni possono ricevere messaggi wireless che impostano le velocitá dei motori. Ricevuto il messaggio, la board regola le velocitá tramite il driver. Il requisito di localizzazione é soddisfatto da un modulo di visione artificiale: una webcam posta in alto riconosce i ragni, ed il pc alla 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.

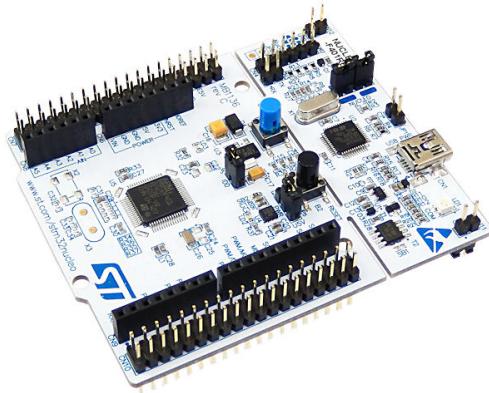


Benché questa semplice soluzione sia sufficiente a far muovere il ragno avanti e indietro a velocità prefisse, essa non andava incontro al nostro requisito di potersi muovere in qualsiasi direzione dello spazio. Per questo motivo, abbiamo rimosso le zampe a destra di tre ragni, e quelle di sinistra agli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



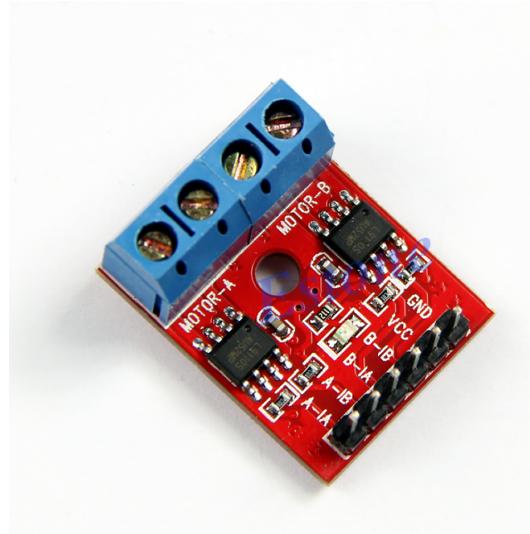
2.2 STM32F401RE Nucleo

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.



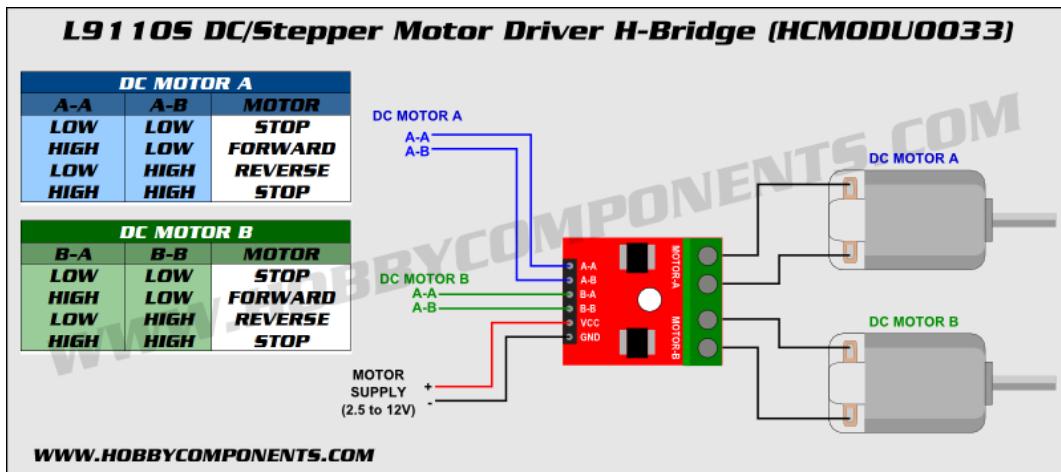
2.3 Driver motore

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 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 1.

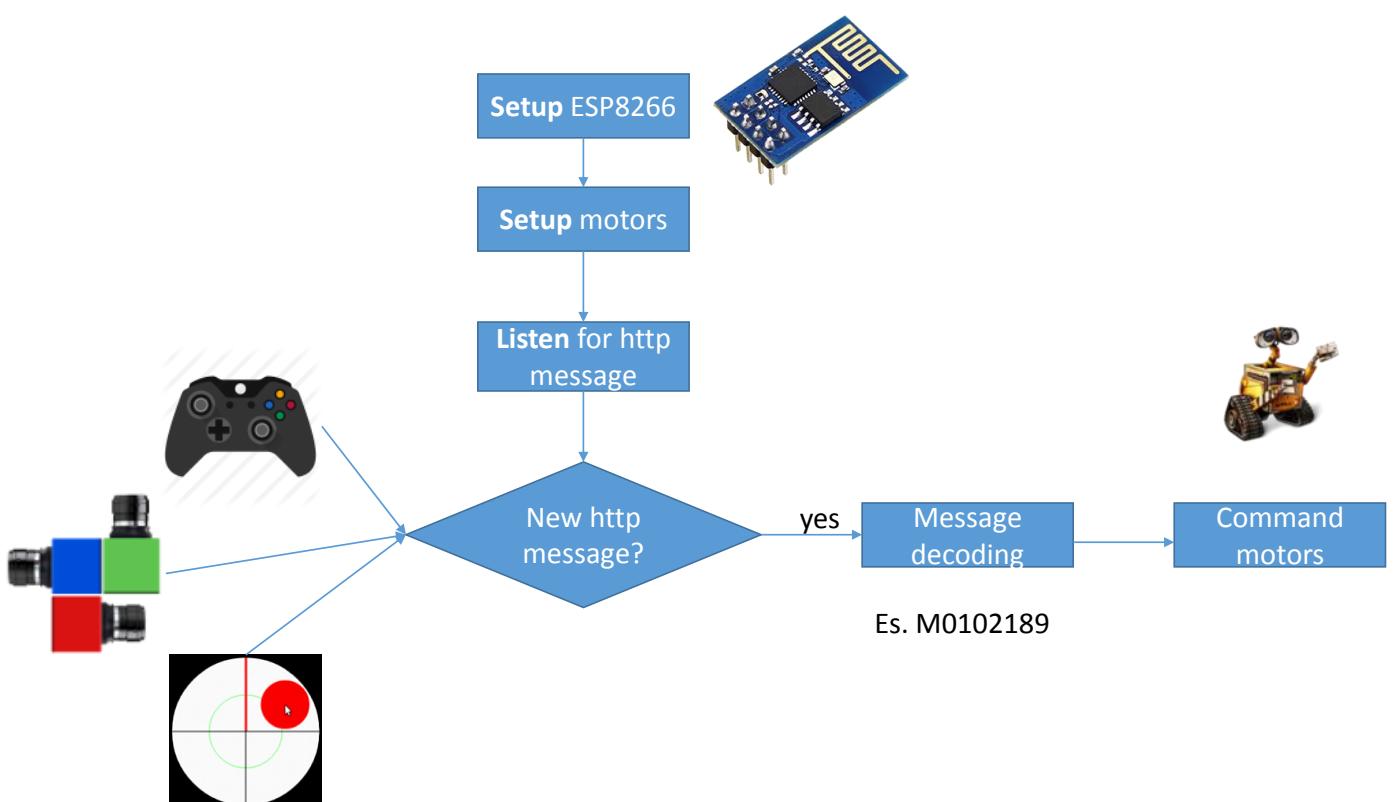


Figura 1: Il diagramma mostra lo schema di funzionamento della board che controlla il robot.

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 2 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.

```
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;
```

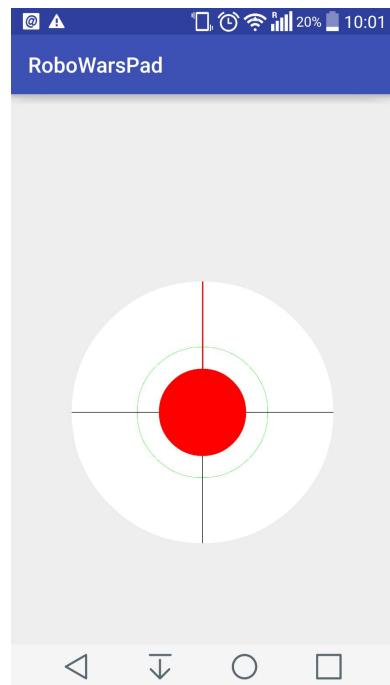


Figura 2: L'immagine mostra l'interfaccia grafica del controller Android realizzato per comandare il robot.

```

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     */
96     pwmccb}, {
97     }, {
98     NULL} },
99     0, 0};

100 //configuration for right engine
101 static PWMConfig pwm2cfg = {200000, /* 10kHz PWM clock frequency
102     . . */
103     */
104     pwm2ccb}, {
105     }, {
106     NULL} },
107     0, 0};

108 void parse_string(char *command, int *velocity) {
109     char left [3];
110     char right[3];
111     int toret [2];
112     char type = command[0];
113
114     left [0] = command[2];
115     left [1] = command[3];
116     left [2] = command[4];
117
118     right [0] = command[5];
119     right [1] = command[6];
120     right [2] = command[7];
121
122     int le, ri;
123
124     le = atoi(left);
125     ri = atoi(right);
126
127     velocity [0] = le;
128     velocity [1] = ri;
129
130 }
131
132 void init_motor() {
133     mapping.type1 = GPIOA;
134     mapping.port1 = GPIOA_PIN8;
135
136
137

```

```

138 mapping.type2 = GPIOB;
139 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     int velocity[2];
186     parse_string(command, velocity);
187     int pwm1 = 1, pwm2 = 1;
188
189     /*****DEMO*****
190     //PIN D7-D6

```

```

186 /*int vel = 0;
187 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     pwmEnableChannel(&PWMD3, 0, pwm2);
295 }
296 }
297 }
```

.../RoboWars/MotorController.h

3.3 Modulo wireless

Per l'utilizzo della board con il sistema ChibiOS è stata realizzata una apposita libreria che racchiude diverse funzionalità. Il modulo ESP8266 può essere configurato tramite comandi seriali (vedi 3). 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.

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' 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.

```

1 /**
2  * @author Simone Romano - s.romano1992@gmail.com
3 */
4 #include "ch.h"
5 #include "hal.h"
6 #include "test.h"
```

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

```

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
13#define TIME_IMMEDIATE ((systime_t)0)
14#define MSG_TIMEOUT (msg_t)-1
15#define Q_TIMEOUT MSG_TIMEOUT
16#define COMMAND_SLEEP 500
17#define COMMAND_LONG_SLEEP 20000
18#define DEBUG 1
19
20 char * readResponse(void);
21 void printWebPage(void);
22 int mystrcontains(char* text, char* toFind);
23 void sendToESP8266(char* command, int delay);
24 void readAndPrintResponse(void);
25 int mystrlen(char* text);
26 static void println(char *p);
27 void blinkBoardLed(void);
28 char* StrStr(const char *str, const char *target);
29 char *strcat(char *dest, const char *src);
30 char *strcpy(char *dest, const char *src);
31 int 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 =
    "AT+CWJAP=\\"Romano Wi-Fi\\",\\"160462160867\\\"\r\n";
42 static char* ESP8266_CHECK_IP = "AT+CIFSR\r\n";
43 static char* ESP8266_GET_IP_ADD = "AT+CIFSR\r\n"; // get ip
    address
44 static char* ESP8266_CHECK_VERSION = "AT+GMR\r\n";
45 static char* ESP8266_MULTIPLE_CONNECTION = "AT+CIPMUX=1\r\n";
46 static char* ESP8266_START_SERVER = "AT+CIPSERVER=1,80\r\n";
47 static char* ESP8266_SET_AS_ACCESS_POINT = "AT+CWMODE=2\r\n"; // configura come access point
48 static char* ESP8266_SET_AS_CLIENT = "AT+CWMODE=1\r\n";
49 static char* ESP8266_SEND_TCP_DATA = "AT+CIPSEND=";
50 static char* ESP8266_CLOSE_CONN = "AT+CIPCLOSE=";
51 char clientID[2];
52 char command[9]; //could be "M023120" for: motor left to 23
    and motor right to 120
53                                         //else could be "PXXYY___" for position is
    xx yy
54                                         //char request;
55

```

```

56 static THD_WORKING_AREA(waThread1, 2048);
57 /**
58 * Asynchronous serial SD1
59 */
60 static msg_t Uart1EVT_Thread(void *p) {
61     int letterAfterPlus = 0;
62     int spaceAfterD = 0;
63     int x_charRead = 0, y_charRead = 0;
64     int BUFF_SIZE = 1024;
65     char received [BUFF_SIZE];
66     int pos = 0;
67     event_listener_t ell;
68     eventflags_t flags;
69
70     chEvtRegisterMask(chnGetEventSource(WIFI_SERIAL), &ell, 1);
71     while (TRUE) {
72         chEvtWaitOne(1);
73
74         chSysLock();
75         flags = chEvtGetAndClearFlagsI(&ell);
76         chSysUnlock(); //wait for events;
77
78         if (flags & CHN_INPUT_AVAILABLE) { //events received
79             msg_t charbuf;
80             do {
81                 charbuf = chnGetTimeout(WIFI_SERIAL, TIME_IMMEDIATE);
82                 chThdSleepMicroseconds(100);
83                 if (charbuf != Q_TIMEOUT) {
84                     if (DEBUG)
85                         chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%c
86 ", (char)charbuf);
87                     if (pos < BUFF_SIZE) {
88                         received [pos] = (char)charbuf;
89                         pos++;
90                     }
91                 }
92             } while (charbuf != Q_TIMEOUT );
93             received [pos] = '\0';
94             /******DO SOMETHING WITH RECEIVED MESSAGE******/
95             char* clearRequest = StrStr(received, "+IPD"); //HTTP
96             REQUEST
97             if (StrStr(received, "+IPD") != NULL){
98                 char* subStringCommand = StrStr(received, " c=");
99                 if (subStringCommand[0] == 'c'){
100                     if (DEBUG)
101                         chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s
102 ", "Received http request");
103                     clientID [0] = clearRequest [5];
104                     clientID [1] = '\0';
105                     request = subStringCommand [0]; //it is c for command (
106                     c=M0... )
107                     command [0] = subStringCommand [2];
108                     command [1] = subStringCommand [3];

```

```

105         command[2] = subStringCommand[4];
106         command[3] = subStringCommand[5];
107         command[4] = subStringCommand[6];
108         command[5] = subStringCommand[7];
109         command[6] = subStringCommand[8];
110         command[7] = subStringCommand[9];
111         command[8] = '\0';
112         if (DEBUG){
113             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s
114             ", "Client id=");
115             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s\
n", clientID);
116             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s
117             ", "Command=");
118             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s\
n", command);
119         }
120         control_motor(command);
121         printWebPage();
122     }
123     /*****END*****/
124     pos = 0;
125 }
126 }
127
128 void blinkBoardLed() {
129     palSetPad(GPIOA, GPIOA_LED_GREEN);
130     chThdSleepMilliseconds(500);
131     palClearPad(GPIOA, GPIOA_LED_GREEN);
132 }
133
134 /**
135 * This function set the ESP8266 (connected to serial SD1)
136 * as access point and start a server on port 80.
137 * It will be ready to accept http request.
138 * NOTE: first to call this function , be sure that:
139 * -you started serial SD2 to read response and debug
140 * -you started serial SD1 to send request to ESP8266
141 * The code:
142 * sdStart(MONITOR_SERIAL, &uartCfgMonitor);
143 * palSetPadMode(GPIOA, 9, PAL_MODE_ALTERNATE(7));
144 * palSetPadMode(GPIOA, 10, PAL_MODE_ALTERNATE(7));
145 * sdStart(WIFI_SERIAL, &uartCfgWiFi);
146 * ESP8266_setAsAP();
147 */
148 void ESP8266_setAsAP(void) {
149     chThdCreateStatic(waThread1, sizeof(waThread1), NORMALPRI,
150                         Uart1EVT_Thread,
151                         NULL);
152     sendToESP8266(ESP8266_RESET, COMMAND_SLEEP);
153     sendToESP8266(ESP8266_SET_AS_ACCESS_POINT, COMMAND_SLEEP);
154     sendToESP8266(ESP8266_GET_IP_ADD, COMMAND_SLEEP);

```

```

154     sendToESP8266(ESP8266_MULTIPLE_CONNECTION, COMMAND_SLEEP) ;
155     sendToESP8266(ESP8266_START_SERVER, COMMAND_SLEEP) ;
156 }
157 /**
158 * This function set the ESP8266 (connected to serial SD1)
159 * as a client to the wifi connection defined in command
160 * ESP8266_CONNECT_TO_WIFI and start a server on port 80.
161 * It will be ready to accept http request.
162 * NOTE: first to call this function, be sure that:
163 * -you started serial SD2 to read response and debug
164 * -you started serial SD1 to send request to ESP8266
165 * The code:
166 * sdStart(MONITOR_SERIAL, &uartCfgMonitor) ;
167 * palSetPadMode(GPIOA, 9, PALMODE_ALTERNATE(7)) ;
168 * palSetPadMode(GPIOA, 10, PALMODE_ALTERNATE(7)) ;
169 * sdStart(WIFI_SERIAL, &uartCfgWiFi) ;
170 * ESP8266_setAsClient() ;
171 */
172 void ESP8266_setAsClient(void) {
173     chThdCreateStatic(waThread1, sizeof(waThread1), NORMALPRIO,
174         Uart1EVT_Thread,
175         NULL) ;
176     sendToESP8266(ESP8266_RESET, COMMAND_SLEEP) ;
177     sendToESP8266(ESP8266_SET_AS_CLIENT, COMMAND_LONG_SLEEP) ;
178     sendToESP8266(ESP8266_MULTIPLE_CONNECTION, COMMAND_LONG_SLEEP)
179         ;
180     sendToESP8266(ESP8266_LIST_WIFI, COMMAND_LONG_SLEEP) ;
181     sendToESP8266(ESP8266_CONNECT_TO_WIFI, COMMAND_LONG_SLEEP) ;
182     sendToESP8266(ESP8266_START_SERVER, COMMAND_LONG_SLEEP) ;
183     sendToESP8266(ESP8266_CHECK_IP, COMMAND_LONG_SLEEP) ;
184 }
185 int mystrlen(char* text) {
186     int length = 0;
187     while (true) {
188         if (text[length] == '\0')
189             return length;
190         length++;
191     }
192 }
193 /**
194 * This function send a command on serial port SD1 to
195 * ESP8266. You can listen for the response by calling
196 * readAndPrintResponse() function .
197 */
198 void sendToESP8266(char* command, int delay) {
199     chprintf((BaseChannel *)WIFI_SERIAL, command) ;
200     chThdSleepMilliseconds(delay) ;
201 }
202 /**
203 * This method reads input from SD1 and print on Serial usb

```

```

206 * monitor (SD2).
207 */
208 void readAndPrintResponse() {
209     char buff[1];
210     int pos = 0;
211     char charbuf;
212     while (true) {
213         charbuf = chnGetTimeout(WIFI_SERIAL, TIME_IMMEDIATE);
214         if (charbuf == EOF) {
215             break;
216         }
217         buff[0] = charbuf;
218         chprintf((BaseChannel *)MONITOR_SERIAL, buff, 1);
219     }
220     buff[0] = '\0';
221     chprintf((BaseChannel *)MONITOR_SERIAL, '\0', 1);
222 }
223
224 void printWebPage() {
225     char cipSend[100] = {"AT+CIPSEND="};
226     char webView[600] = {"<html>-"};
227     char webView1[20] = {"-</html>"};
228     if (request == 'c')
229         strcat(webPage, command);
230     strcat(webPage, webView1);
231     strcat(cipSend, clientID);
232     strcat(cipSend, ",");
233     int pageLength = strlen(command);
234     char pageLengthAsString[100];
235     itoa(pageLength, pageLengthAsString);
236     strcat(cipSend, pageLengthAsString);
237     strcat(cipSend, "\r\n");
238     sendToESP8266(cipSend, COMMAND_SLEEP);
239     sendToESP8266(command, COMMAND_SLEEP);
240 }
241
242 /**
243 * Return length of str as char (es. '47')
244 */
245 int strlen(const char * str){
246     int len;
247     for (len = 0; str[len]; len++);
248     return len;
249 }
250
251 char *strcpy(char *dest, const char *src){
252     unsigned i;
253     for (i=0; src[i] != '\0'; ++i)
254         dest[i] = src[i];
255     dest[i] = '\0';
256     return dest;
257 }
258 char* StrStr(const char *str, const char *target) {
259     if (!*target) return str;

```

```

260 char *p1 = (char*)str , *p2 = (char*)target ;
261 char *p1Adv = (char*)str ;
262 while (*++p2)
263     p1Adv++;
264 while (*p1Adv) {
265     char *p1Begin = p1 ;
266     p2 = (char*)target ;
267     while (*p1 && *p2 && *p1 == *p2) {
268         p1++;
269         p2++;
270     }
271     if (!*p2)
272         return p1Begin ;
273     p1 = p1Begin + 1;
274     p1Adv++;
275 }
276 return NULL;
277 }
278 char *strcat(char *dest , const char *src){
279     size_t i,j;
280     for (i = 0; dest[i] != '\0'; i++)
281         ;
282     for (j = 0; src[j] != '\0'; j++)
283         dest[i+j] = src[j];
284     dest[i+j] = '\0';
285     return dest ;
286 }
287 static void println(char *p) {
288
289     while (*p) {
290         chSequentialStreamPut(MONITOR_SERIAL, *p++);
291     }
292     chSequentialStreamWrite(MONITOR_SERIAL, (uint8_t *) "\r\n" , 2)
293     ;
294 }
295 /* itoa: convert n to characters in s */
296 void itoa(int n, char s[]){
297     int i, sign;
298     if ((sign = n) < 0) /* record sign */
299         n = -n;           /* make n positive */
300     i = 0;
301     do {                /* generate digits in reverse order */
302         s[i++] = n % 10 + '0'; /* get next digit */
303     } while ((n /= 10) > 0); /* delete it */
304     if (sign < 0)
305         s[i++] = '-';
306     s[i] = '\0';
307     reverse(s);
308 }
309 /* reverse: reverse string s in place */
310 void reverse(char s[]){
311     int i, j;

```

```

313     char c;
314
315     for ( i = 0, j = strlen(s)-1; i<j; i++, j--) {
316         c = s[ i ];
317         s[ i ] = s[ j ];
318         s[ j ] = c;
319     }
320 }
```

..../RoboWars/ESP8266.h

3.4 Visione Artificiale

```

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)
```

```

38     self.s_s_c1Lower = np.array((s_s_c1HSV[0][0][0]-10,100,100))
39     self.s_s_c1Upper = np.array((s_s_c1HSV[0][0][0]+10,255,255))
40
41     s_s_c2HSV = cv2.cvtColor(self.second_spider_color2, cv2.
42 COLOR_BGR2HSV)
42     self.s_s_c2Lower = np.array((s_s_c2HSV[0][0][0]-10,100,100))
43     self.s_s_c2Upper = np.array((s_s_c2HSV[0][0][0]+10,255,255))
44
45     self.camera = cv2.VideoCapture(0)
46 # video recorder
47
48
49     grabbed = False
50     while not grabbed:
51         (grabbed, frame) = self.camera.read()
52         #print("No frame D:")
53
54     frame = imutils.resize(frame, width=900)
55     h = frame.shape[0]
56     #print h
57     # resize the frame, blur it, and convert it to the HSV
58     # color space
59     fourcc = cv2.cv.CV_FOURCC('m', 'p', '4', 'v')
60     self.video_writer = cv2.VideoWriter("output.avi", fourcc,
61     12, (900, h))
62     if not self.video_writer :
63         print "!!! Failed VideoWriter: invalid parameters"
64         sys.exit(1)
65
66     def close_all(self):
67         # cleanup the camera and close any open windows
68         self.camera.release()
69         self.video_writer.release()
70         cv2.destroyAllWindows()
71
72     def get_Spider(self, color1Lower, color1Upper, color2Lower,
73     color2Upper):
74
75         (grabbed, frame) = self.camera.read()
76         #frame = cv2.flip(frame,1)
77         s_x = None
78         s_y = None
79         f_x = None
80         f_y = None
81         # if we are viewing a video and we did not grab a frame,
82         # then we have reached the end of the video
83         if not grabbed:
84             print("No frame D:")
85
86         # resize the frame, blur it, and convert it to the HSV
87         # color space
88         frame = imutils.resize(frame, width=900)
89         blurred = cv2.GaussianBlur(frame, (11, 11), 0)
90         #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
107    # only proceed if at least one contour was found
108    if len(cnts) > 0:
109        # find the largest contour in the mask, then use
110        # it to compute the minimum enclosing circle and
111        # centroid
112        c = max(cnts, key=cv2.contourArea)
113        ((fx, fy), radius) = cv2.minEnclosingCircle(c)
114        M = cv2.moments(c)
115        center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"])
116                ))
117
118        # only proceed if the radius meets a minimum size
119        if radius > 5:
120            # draw the circle and centroid on the frame,
121            # then update the list of tracked points
122            cv2.circle(frame, (int(fx), int(fy)), int(radius),
123                      (0, 255, 255), 2)
124            cv2.circle(frame, center, 5, (0, 0, 255), -1)
125
126    mask = cv2.inRange(hsv, color2Lower, color2Upper)
127    mask = cv2.erode(mask, None, iterations=3)
128    mask = cv2.dilate(mask, None, iterations=2)
129
130    # cv2.imshow("mask2",cv2.bitwise_and(hsv,hsv,mask=mask));
131
132    # find contours in the mask and initialize the current
133    # (x, y) center of the ball
134    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
135                           cv2.CHAIN_APPROX_SIMPLE)[-2]
136    center = None
137
138    # only proceed if at least one contour was found
139    if len(cnts) > 0:
140        # find the largest contour in the mask, then use
141        # it to compute the minimum enclosing circle and
142        # centroid
143        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
216     # only proceed if the radius meets a minimum size
217     if radius > 5:
218         # draw the circle and centroid on the frame,
219         # then update the list of tracked points
220         cv2.circle(frame, (int(f_x), int(f_y)), int(radius),
221                    (0, 255, 255), 2)
222         cv2.circle(frame, center, 5, (0, 0, 255), -1)
223
224
225     cv2.imshow("Frame", frame)
226     self.video_writer.write(frame)
227
228     if f_x and f_y:
229         p1 = np.array([f_x, f_y])
230         return p1
231
232     #dati due punti restituisce la matrice omogenea di
233     #rototraslazione
234     def calculate_matrix(self, p0, p1, versor=[1,0]):
235
236         vet_diff = p1 - p0
237         x_axis = np.array(versor)
238         dot_product = np.dot(vet_diff, x_axis)
239         module = np.linalg.norm(vet_diff)
240         cos_arg = dot_product/module
241         angle = np.arccos(cos_arg)
242
243         #print(vet_diff, cos_arg, np.degrees(angle))
244
245         rot_matrix = [cos_arg, -np.sin(angle), np.sin(angle),
246                      cos_arg]
247         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])
247     matrix = np.reshape(matrix, (3,3))
248     #print(matrix)
249     return matrix
250
251 def launch_curl(string):
252     subprocess.call("curl -m 1 http://192.168.4.1/?cm=0"+string,
253 shell=True)
254 if __name__ == '__main__':
255     g = Vision()
256     stop = True
257     old_command = "stop"
258
259     while True:
260         returns = g.get_Spider(g.f_s_c1Lower, g.f_s_c1Upper, g.
261 f_s_c2Lower, g.f_s_c2Upper)
262         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]
272 is not None and returns[1] is not None:
273             #print(first_matrix, second_matrix)
274
275             #print(first_matrix[0][0])
276
277             print(old_command)
278             points = returns[0]
279             first_matrix = returns[1]
280
281             p0 = [first_matrix[0][2], first_matrix[1][2]]
282             p1 = point
283
284             new_matrix_x= g.calculate_matrix(np.array(p0), p1)
285             new_matrix_y= g.calculate_matrix(np.array(p0), p1, [0,1])
286
287             cos_x = new_matrix_x[0][0]
288             cos_y = new_matrix_y[0][0]
289
290             #print("Coseno rispetto X: "+ str(cos_x))
291             #print("Coseno rispetto Y: "+ str(cos_y))
292
293             #epsilon di rotazione del robot rispetto alla telecamera
294             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 = Process(target=launch(curl , args=( '000000' , )))
341                 p.start()
342                 continue
343             elif cos_y < -epsilon_fron:
344                 print("Vai a Avanti_Verticale_Arancione")
345                 if old_command != "front":
346                     old_command = "front"
347                     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         print("Missing one of the components")
448         if old_command != "stop":
449             old_command = "stop"
450             p = Process(target=launch_curl , args=( '128128' , ))
451             p.start()
452             continue

```

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.