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 14
3.4 Visione Artificiale 21

4 Conclusioni **30**
4.1 Sviluppi futuri 30

1 Introduzione

Lo scopo del progetto "Spider-chase" é 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.

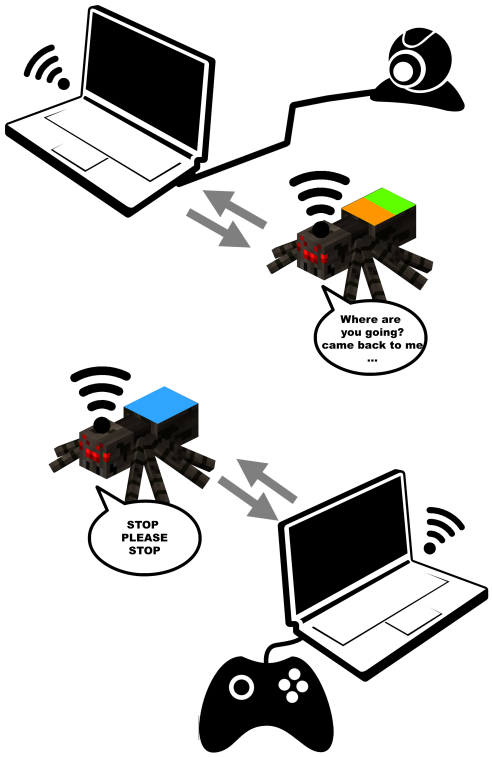
2 Architettura e componenti

Nel progetto abbiamo utilizzato i seguenti componenti:

- 6 ragni meccanici DIY, ciascuno con un motore.
- 3 schede STM Nucleo.
- 3 moduli wireless (modello).
- 3 driver per motori (modello).
- 1 webcam.

- 1 pc.
- 1 controller Xbox.
- 1 cellulare Android.

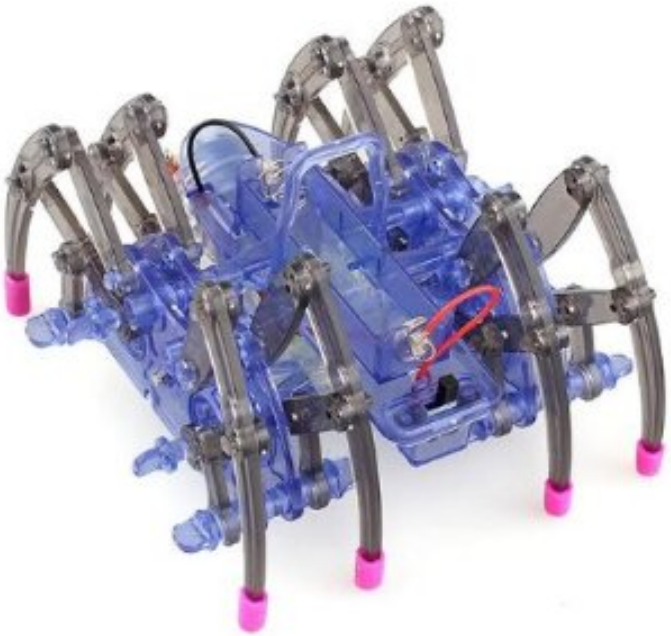
L'architettura totale pianificata é mostrata in figura.



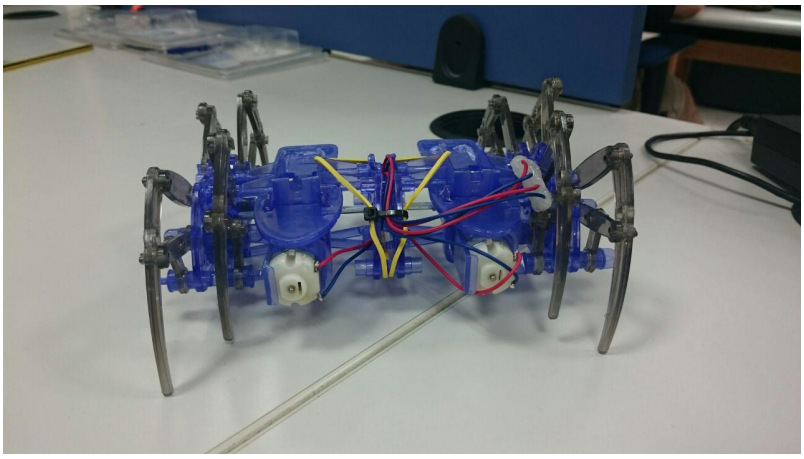
Tutti e tre i ragni possono ricevere messaggi wireless che impostano le velocitá dei motori. Ricevuto il messaggio, la board regola le velocitá tramite il driver. Il requisito di localizzazione é soddisfatto da un modulo di visione artificiale: una webcam posta in alto riconosce i ragni, ed il pc alla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

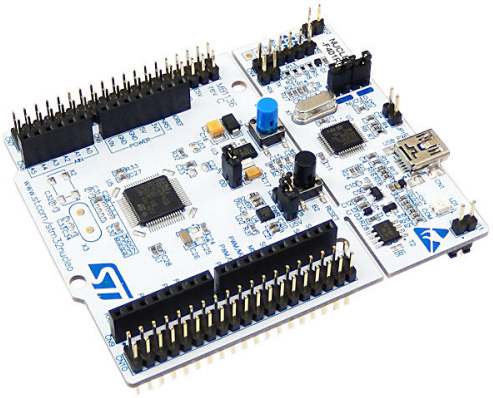


Benché questa semplice soluzione sia sufficiente a far muovere il ragno avanti e indietro a velocitá prefisse, essa non andava incontro al nostro requisito di potersi muovere in qualsiasi direzione dello spazio. Per questo motivo, abbiamo rimosso le zampe a destra di tre ragni, e quelle di sinistra agli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



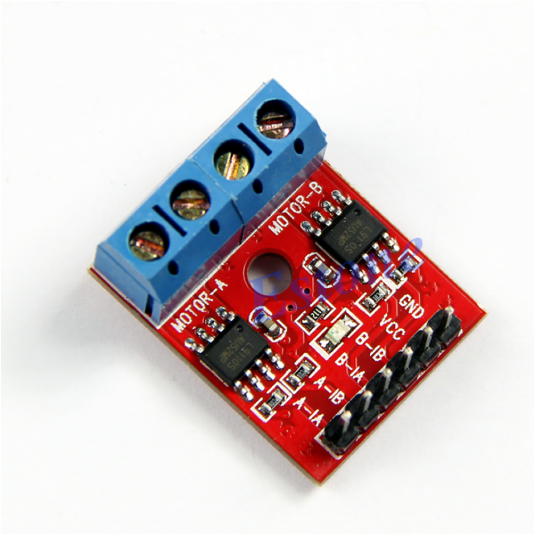
2.2 STM32F401RE Nucleo

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 connettivitá Arduino e ST Morpho, é possibile espandere le funzionalitá del microcontrollore scegliendo da una vasta gamma di shield. Inoltre, la STM32 nucleo non richiede due ingressi separati per alimentazione e debugging.



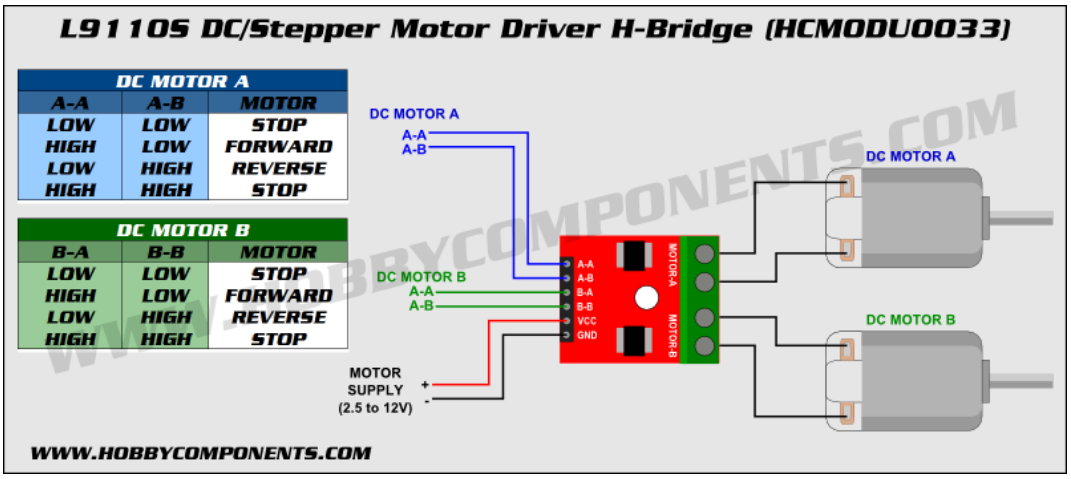
2.3 Driver motore

Il driver del motore utilizzato é 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 velocitá del motore é 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 é maggiore di quella che arriva all'ingresso 2, allora il ragno va in avanti, altrimenti va all'indietro. Maggiore é la differenza di velocitá, piú il ragno va velocemente. Di seguito é 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.

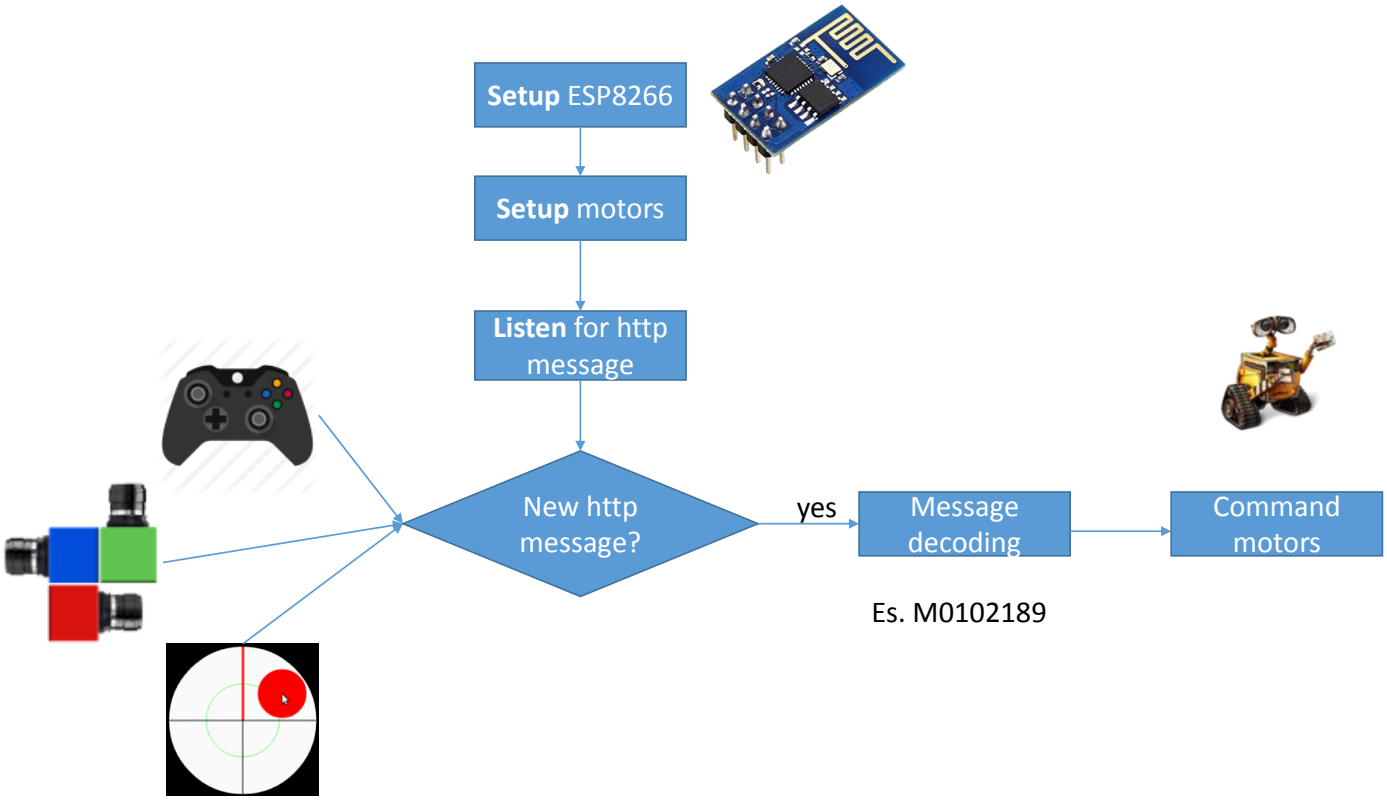


Figura 1: Il diagramma mostra lo schema di funzionamento della board che controlla il robot.

3 Dettagli implementativi

3.1 Controller

Per la comunicazione con i robot sono state utilizzate delle richieste http, con il formato descritto della sezione **??**. 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 **??**. L'immagine in figura 3.1.1 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 velocitá, ed una funzione di utilitá che estrae le due velocitá a partire da una stringa.

```
1  void (*functioPtrLeftUP)();
2  void (*functioPtrLeftDOWN)();
3  void (*functioPtrRightUP)();
4  void (*functioPtrRightDOWN)();
5
6  static struct Mapping_GPIO {
7    stm32_gpio_t * type1;
8    unsigned int port1;
9
10   stm32_gpio_t * type2;
11   unsigned int port2;
12
13   stm32_gpio_t * type3;
14   unsigned int port3;
```

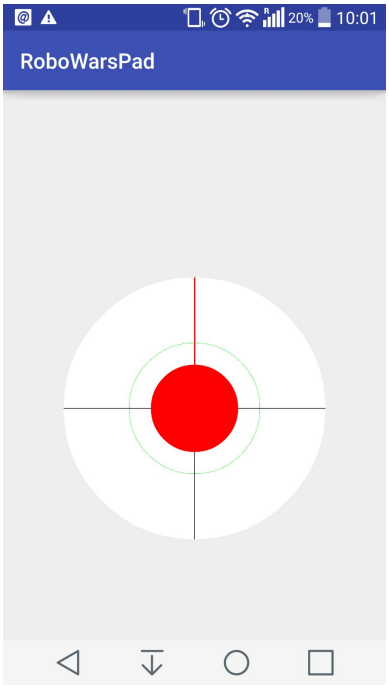


Figura 2: L'immagine mostra l'intercaccia grafica del controller Android realizzato per comandare il robot.

```
15
16    stm32_gpio_t * type4;
17    unsigned int port4;
18  } mapping;
19
20  static void Sinistra_Avanti_up() {
21    palSetPad(mapping.type1, mapping.port1);
22  }
23
24  static void Sinistra_Avanti_Down() {
25    palClearPad(mapping.type1, mapping.port1);
26  }
27
28  static void Sinistra_Dietro_up() {
29    palSetPad(mapping.type2, mapping.port2);
30  }
31
32  static void Sinistra_Dietro_Down() {
33    palClearPad(mapping.type2, mapping.port2);
34  }
35
36  static void Destra_Avanti_up() {
37    palSetPad(mapping.type3, mapping.port3);
38  }
39
```

```
40 static void Destra_Avanti_Down() {
41   palClearPad(mapping.type3, mapping.port3);
42 }
43
44 static void Destra_Dietro_up() {
45   palSetPad(mapping.type4, mapping.port4);
46 }
47
48 static void Destra_Dietro_Down() {
49   palClearPad(mapping.type4, mapping.port4);
50 }
51
52 static void pwmpcb(PWMDriver * pwmp) {
53   (void)pwmp;
54   (*functioPtrLeftDOWN)();
55 }
56
57 static void pwmc1cb(PWMDriver * pwmp) {
58   (void)pwmp;
59   (*functioPtrLeftUP)();
60 }
61
62 static void pwm2pcb(PWMDriver * pwmp) {
63   (void)pwmp;
64   (*functioPtrRightDOWN)();
65 }
66
67 static void pwm2c1cb(PWMDriver * pwmp) {
68   (void)pwmp;
69   (*functioPtrRightUP)();
70 }
71
72 static void clearAllPads(){
73     palClearPad(mapping.type1, mapping.port1);
74     palClearPad(mapping.type2, mapping.port2);
75     palClearPad(mapping.type3, mapping.port3);
76     palClearPad(mapping.type4, mapping.port4);
77 }
78
79 //configuration for left engine
80 static PWMConfig pwm1cfg =  {10000,
81                              500,
82                              pwmpcb,
83                 {{PWM_OUTPUT_ACTIVE_HIGH, pwmc1cb},
84                 {PWM_OUTPUT_DISABLED, NULL},
85                              {PWM_OUTPUT_DISABLED, NULL},
86                 {PWM_OUTPUT_DISABLED, NULL}},
87                              0,
88                 0};
89
90 //configuration for right engine
91 static PWMConfig pwm2cfg = {10000,
92                              500,
93                              pwm2pcb,
```

```
94                  {{PWM_OUTPUT_ACTIVE_HIGH, pwm2c1cb},
95                  {PWM_OUTPUT_DISABLED, NULL},
96                  {PWM_OUTPUT_DISABLED, NULL},
97                  {PWM_OUTPUT_DISABLED, NULL}},
98                                  0,
99                  0};
100
101 void parse_string(char * command, int * velocity) {
102   char left[3];
103   char right[3];
104   int toret[2];
105   char type = command[0];
106
107   left[0] = command[2];
108   left[1] = command[3];
109   left[2] = command[4];
110
111   right[0] = command[5];
112   right[1] = command[6];
113   right[2] = command[7];
114
115   int le, ri;
116
117   le = atoi(left);
118   ri = atoi(right);
119
120   velocity[0] = le;
121   velocity[1] = ri;
122 }
123
124 void init_motor() {
125   mapping.type1 = GPIOA;
126   mapping.port1 = GPIOA_PIN8;
127
128   mapping.type2 = GPIOB;
129   mapping.port2 = GPIOB_PIN10;
130
131   mapping.type3 = GPIOB;
132   mapping.port3 = GPIOB_PIN4;
133
134   mapping.type4 = GPIOB;
135   mapping.port4 = GPIOB_PIN5;
136
137   palSetPadMode(mapping.type1, mapping.port1,
138                 PAL_MODE_OUTPUT_PUSHPULL |
      PAL_STM32_OSPEED_HIGHEST);
139   palClearPad(mapping.type1, mapping.port1);
140   palSetPadMode(mapping.type2, mapping.port2,
141                 PAL_MODE_OUTPUT_PUSHPULL |
      PAL_STM32_OSPEED_HIGHEST);
142   palClearPad(mapping.type2, mapping.port2);
143   palSetPadMode(mapping.type3, mapping.port3,
144                 PAL_MODE_OUTPUT_PUSHPULL |
      PAL_STM32_OSPEED_HIGHEST);
```

```
145     palClearPad(mapping.type3, mapping.port3);
146     palSetPadMode(mapping.type4, mapping.port4,
147                   PAL_MODE_OUTPUT_PUSHPULL |
        PAL_STM32_OSPEED_HIGHEST);
148     palClearPad(mapping.type4, mapping.port4);
149     functioPtrLeftUP = &Sinistra_Avanti_up;
150     functioPtrLeftDOWN = &Sinistra_Avanti_Down;
151
152     functioPtrRightUP = &Destra_Avanti_up;
153     functioPtrRightDOWN = &Destra_Avanti_Down;
154
155     pwmStart(&PWMD1, &pwm1cfg);
156     pwmEnablePeriodicNotification(&PWMD1);
157     pwmEnableChannel(&PWMD1, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD1,
        10000));
158     pwmEnableChannelNotification(&PWMD1, 0);
159
160     pwmStart(&PWMD3, &pwm2cfg);
161     pwmEnablePeriodicNotification(&PWMD3);
162     pwmEnableChannel(&PWMD3, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD3,
        10000));
163     pwmEnableChannelNotification(&PWMD3, 0);
164 }
165
166 void control_motor(char* command) {
167
168     int velocity[2];
169     parse_string(command, velocity);
170     int pwm1 = 1, pwm2 = 1;
171
172     if (velocity[0] >= 128) {
173       velocity[0] = velocity[0] - 128;
174       clearAllPads();
175       functioPtrLeftUP = &Sinistra_Avanti_up;
176       functioPtrLeftDOWN = &Sinistra_Avanti_Down;
177       pwm1 =  10000 - 77.95 * velocity[0] + 100;
178       pwmEnableChannel(&PWMD1, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD1,
        pwm1));
179     }
180     else {
181     clearAllPads();
182       functioPtrLeftUP = &Sinistra_Dietro_up;
183       functioPtrLeftDOWN = &Sinistra_Dietro_Down;
184       pwm1 = 77.95 * velocity[0] + 100;
185       pwmEnableChannel(&PWMD1, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD1,
        pwm1));
186     }
187
188     if (velocity[1] >= 128) {
189       velocity[1] = velocity[1] - 128;
190       clearAllPads();
191       functioPtrRightUP = &Destra_Avanti_up;
192       functioPtrRightDOWN = &Destra_Avanti_Down;
193       pwm2 =  10000 - 77.95 * velocity[1] + 100;
```

```
194         pwmEnableChannel(&PWMD3, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD3,
            pwm2));
195     }
196     else {
197     clearAllPads();
198         functioPtrRightUP = &Destra_Dietro_up;
199         functioPtrRightDOWN = &Destra_Dietro_Down;
200         pwm2 = 77.95 * velocity[1] + 100;
201         pwmEnableChannel(&PWMD3, 0, PWM_PERCENTAGE_TO_WIDTH(&PWMD3,
            pwm2));
202     }
203
204 }
```

../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 3.3). 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 3.3.

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.

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.

```
#include "ch.h"
#include "hal.h"
#include "test.h"
#include "chprintf.h"

#define EOF '\377'
#define WIFI_SERIAL &SD1
#define MONITOR_SERIAL &SD2
#define MAX_LENGTH  100
#define TIME_IMMEDIATE  ((systime_t)0)
#define MSG_TIMEOUT (msg_t)-1
#define Q_TIMEOUT   MSG_TIMEOUT
#define COMMAND_SLEEP 500
#define COMMAND_LONG_SLEEP 20000
#define DEBUG 0

char * readResponse(void);
void printWebPage(void);
int mystrcontains(char* text, char* toFind);
void sendToESP8266(char* command, int delay);
void readAndPrintResponse(void);
int mystrlen(char* text);
static void println(char *p);
void blinkBoardLed(void);
char* StrStr(const char *str, const char *target);
char *strcat(char *dest, const char *src);
char *strcpy(char *dest, const char *src);
int strlen(const char * str);
void itoa(int n, char s[]);
void reverse(char s[]);

static SerialConfig uartCfgWiFi = {115200,
    };

static char* ESP8266_HELLO = "AT\r\n";
static char* ESP8266_RESET = "AT+RST\r\n";
static char* ESP8266_LIST_WIFI = "AT+CWLAP\r\n";
static char* ESP8266_CONNECT_TO_WIFI =
    "AT+CWJAP=\"Romano Wi-Fi\",\"160462160867\"\r\n";
static char* ESP8266_CHECK_IP = "AT+CIFSR\r\n";
static char* ESP8266_GET_IP_ADD = "AT+CIFSR\r\n";
static char* ESP8266_CHECK_VERSION = "AT+GMR\r\n";
static char* ESP8266_MULTIPLE_CONNECTION = "AT+CIPMUX=1\r\n";
static char* ESP8266_START_SERVER = "AT+CIPSERVER=1,80\r\n";
static char* ESP8266_SET_AS_ACCESS_POINT = "AT+CWMODE=2\r\n";
static char* ESP8266_SET_AS_CLIENT = "AT+CWMODE=1\r\n";
static char* ESP8266_SEND_TCP_DATA = "AT+CIPSEND=";
static char* ESP8266_CLOSE_CONN = "AT+CIPCLOSE=";
char clientID[2];
char command[9];
char request;
static THD_WORKING_AREA(waThread1, 2048);

```

```
54
55 static msg_t Uart1EVT_Thread(void *p) {
56   int letterAfterPlus = 0;
57   int spaceAfterD = 0;
58   int x_charRead = 0, y_charRead = 0;
59   int BUFF_SIZE = 1024;
60   char received[BUFF_SIZE];
61   int pos = 0;
62   event_listener_t el1;
63   eventflags_t flags;
64
65   chEvtRegisterMask(chnGetEventSource(WIFI_SERIAL), &el1, 1);
66   while (TRUE) {
67     chEvtWaitOne(1);
68
69     chSysLock();
70     flags = chEvtGetAndClearFlagsI(&el1);
71     chSysUnlock();  //wait for events;
72
73     if (flags & CHN_INPUT_AVAILABLE) {  //events received
74       msg_t charbuf;
75       do {
76         charbuf = chnGetTimeout(WIFI_SERIAL, TIME_IMMEDIATE);
77         chThdSleepMicroseconds(100);
78         if (charbuf != Q_TIMEOUT) {
79           if (DEBUG)
80             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%c"
    , (char)charbuf);
81           if (pos < BUFF_SIZE) {
82             received[pos] = (char)charbuf;
83             pos++;
84           }
85         }
86       } while (charbuf != Q_TIMEOUT );
87       received[pos] = '\0';
88
89       char* clearRequest = StrStr(received, "+IPD");
90       if (StrStr(received, "+IPD") != NULL){
91         if (DEBUG)
92           chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s",
    "Received http request");
93         clientID[0] = clearRequest[5];
94         clientID[1] = '\0';
95         request = clearRequest[16];
96         command[0] = clearRequest[18];
97         command[1] = clearRequest[19];
98         command[2] = clearRequest[20];
99         command[3] = clearRequest[21];
100        command[4] = clearRequest[22];
101        command[5] = clearRequest[23];
102        command[6] = clearRequest[24];
103        command[7] = clearRequest[25];
104        command[8] = '\0';
105        if (DEBUG){
```

```
106             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s",
    "Client id=");
107             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s\n"
    , clientID);
108             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s",
    "Command=");
109             chprintf((BaseSequentialStream*)MONITOR_SERIAL, "%s\n"
    , command);
110           }
111           if (request == 'c')
112             control_motor(command);
113           //printWebPage();
114         }
115
116         pos = 0;
117       }
118     }
119 }
120
121 void blinkBoardLed() {
122   palSetPad(GPIOA, GPIOA_LED_GREEN);
123   chThdSleepMilliseconds(500);
124   palClearPad(GPIOA, GPIOA_LED_GREEN);
125 }
126
127 void ESP8266_setAsAP(void) {
128   chThdCreateStatic(waThread1, sizeof(waThread1), NORMALPRIO,
      Uart1EVT_Thread,
129                     NULL);
130   sendToESP8266(ESP8266_RESET, COMMAND_SLEEP);
131   sendToESP8266(ESP8266_SET_AS_ACCESS_POINT, COMMAND_SLEEP);
132   sendToESP8266(ESP8266_GET_IP_ADD, COMMAND_SLEEP);
133   sendToESP8266(ESP8266_MULTIPLE_CONNECTION, COMMAND_SLEEP);
134   sendToESP8266(ESP8266_START_SERVER, COMMAND_SLEEP);
135 }
136
137 void ESP8266_setAsClient(void) {
138   chThdCreateStatic(waThread1, sizeof(waThread1), NORMALPRIO,
      Uart1EVT_Thread,
139                     NULL);
140   sendToESP8266(ESP8266_RESET, COMMAND_SLEEP);
141   sendToESP8266(ESP8266_SET_AS_CLIENT, COMMAND_LONG_SLEEP);
142   sendToESP8266(ESP8266_MULTIPLE_CONNECTION, COMMAND_LONG_SLEEP)
      ;
143   sendToESP8266(ESP8266_LIST_WIFI, COMMAND_LONG_SLEEP);
144   sendToESP8266(ESP8266_CONNECT_TO_WIFI, COMMAND_LONG_SLEEP);
145   sendToESP8266(ESP8266_START_SERVER, COMMAND_LONG_SLEEP);
146   sendToESP8266(ESP8266_CHECK_IP, COMMAND_LONG_SLEEP);
147 }
148
149 int mystrlen(char* text) {
150   int length = 0;
151   while (true) {
152     if (text[length] == '\0')
```

```
153         return length;
154       length++;
155     }
156 }
157
158
159 void sendToESP8266(char* command, int delay) {
160   chprintf((BaseChannel *)WIFI_SERIAL, command);
161   chThdSleepMilliseconds(delay);
162 }
163
164 void readAndPrintResponse() {
165   char buff[1];
166   int pos = 0;
167   char charbuf;
168   while (true) {
169     charbuf = chnGetTimeout(WIFI_SERIAL, TIME_IMMEDIATE);
170     if (charbuf == EOF) {
171       break;
172     }
173     buff[0] = charbuf;
174     chprintf((BaseChannel *)MONITOR_SERIAL, buff, 1);
175   }
176   buff[0] = '\0';
177   chprintf((BaseChannel *)MONITOR_SERIAL, '\0', 1);
178 }
179
180 void printWebPage() {
181   char cipSend[100] = {"AT+CIPSEND="};
182   char webPage[600] = {"<html>"};
183   char webPage1[20] = {"</html>"};
184   if (request == 'c')
185     strcat(webPage,command);
186   strcat(webPage,webPage1);
187   strcat(cipSend,clientID);
188   strcat(cipSend,",");
189   int pageLength = strlen(command);
190   char pageLengthAsString[100];
191   itoa(pageLength,pageLengthAsString);
192   strcat(cipSend,pageLengthAsString);
193   strcat(cipSend,"\r\n");
194   sendToESP8266(cipSend, COMMAND_SLEEP);
195   sendToESP8266(command, COMMAND_SLEEP);
196 }
197
198 int strlen(const char * str){
199     int len;
200     for (len = 0; str[len]; len++);
201     return len;
202 }
203
204 char *strcpy(char *dest, const char *src){
205   unsigned i;
206   for (i=0; src[i] != '\0'; ++i)
```

```
207     dest[i] = src[i];
208   dest[i] = '\0';
209   return dest;
210 }
211
212 char* StrStr(const char *str, const char *target) {
213   if (!*target)
214     return str;
215   char *p1 = (char*)str, *p2 = (char*)target;
216   char *p1Adv = (char*)str;
217   while (*++p2)
218     p1Adv++;
219   while (*p1Adv) {
220     char *p1Begin = p1;
221     p2 = (char*)target;
222     while (*p1 && *p2 && *p1 == *p2) {
223       p1++;
224       p2++;
225     }
226     if (!*p2)
227       return p1Begin;
228     p1 = p1Begin + 1;
229     p1Adv++;
230   }
231   return NULL;
232 }
233 char *strcat(char *dest, const char *src){
234     size_t i,j;
235     for (i = 0; dest[i] != '\0'; i++)
236         ;
237     for (j = 0; src[j] != '\0'; j++)
238         dest[i+j] = src[j];
239     dest[i+j] = '\0';
240     return dest;
241 }
242 static void println(char *p) {
243
244   while (*p) {
245     chSequentialStreamPut(MONITOR_SERIAL, *p++);
246   }
247   chSequentialStreamWrite(MONITOR_SERIAL, (uint8_t * )"\r\n", 2)
      ;
248 }
249
250  void itoa(int n, char s[]){
251      int i, sign;
252      if ((sign = n) < 0)
253          n = -n;
254      i = 0;
255      do {
256          s[i++] = n % 10 + '0';
257      } while ((n /= 10) > 0);
258      if (sign < 0)
259          s[i++] = '-';
```

```
260       s[i] = '\0';
261       reverse(s);
262   }
263
264 void reverse(char s[]){
265     int i, j;
266     char c;
267
268     for (i = 0, j = strlen(s)-1; i<j; i++, j--) {
269         c = s[i];
270         s[i] = s[j];
271         s[j] = c;
272     }
273 }
```

../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
12   def __init__ (self):
13
14     #arancione
15     self.first_spider_color1 = np.uint8([[[57,114,255]]])
16     #verde
17     self.first_spider_color2 = np.uint8([[[102,151,49]]])
18     #fucsia
19     #self.first_spider_color2 = np.uint8([[[180,95,245]]])
20
21     #azzurro
22     self.second_spider_color1 = np.uint8([[[216,145,0]]])
23     #rosso
24     #self.second_spider_color1 = np.uint8([[[78,63,223]]])
25     #giallo
26     self.second_spider_color2 = np.uint8([[[0,255,255]]])
27
28
29     f_s_c1HSV = cv2.cvtColor(self.first_spider_color1,cv2.
       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.
        COLOR_BGR2HSV)
34      self.f_s_c2Lower = np.array((f_s_c2HSV[0][0][0]-10,100,100))
35      self.f_s_c2Upper = np.array((f_s_c2HSV[0][0][0]+10,255,255))
36
37      s_s_c1HSV = cv2.cvtColor(self.second_spider_color1,cv2.
        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.
        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,
        16, (900, h))
61      if not self.video_writer :
62        print "!!! Failed VideoWriter: invalid parameters"
63        sys.exit(1)
64
65    def close_all(self):
66      # cleanup the camera and close any open windows
67      self.camera.release()
68      self.video_writer.release()
69      cv2.destroyAllWindows()
70
71    def get_Spider(self, color1Lower, color1Upper, color2Lower,
        color2Upper):
72
73      (grabbed, frame) = self.camera.read()
74      #frame = cv2.flip(frame,1)
75      s_x = None
76      s_y = None
77      f_x = None
78      f_y = None
79      # if we are viewing a video and we did not grab a frame,
80      # 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           # only proceed if the radius meets a minimum size
147           if radius > 5:
148             # draw the circle and centroid on the frame,
149             # then update the list of tracked points
150             cv2.circle(frame, (int(s_x), int(s_y)), int(radius),
151             (255, 255, 0), 2)
152             cv2.circle(frame, center, 5, (0, 0, 255), -1)
153
154         cv2.imshow("Frame", frame)
155         self.video_writer.write(frame)
156
157         if f_x and f_y and s_x and s_y:
158           p0 = np.array([f_x,f_y])
159           p1 = np.array([s_x,s_y])
160
161           points = dict()
162
163           points["p0"] = p0
164           points["p1"] = p1
165
166           return [points, self.calculate_matrix(p0,p1)]
167
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     # only proceed if at least one contour was found
205     if len(cnts) > 0:
206       # find the largest contour in the mask, then use
207       # it to compute the minimum enclosing circle and
208       # centroid
209       c = max(cnts, key=cv2.contourArea)
210       ((f_x, f_y), radius) = cv2.minEnclosingCircle(c)
211       M = cv2.moments(c)
212       center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"
    ]))
213
214       # only proceed if the radius meets a minimum size
215       if radius > 5:
216         # draw the circle and centroid on the frame,
217         # then update the list of tracked points
218         cv2.circle(frame, (int(f_x), int(f_y)), int(radius),
219         (0, 255, 255), 2)
220         cv2.circle(frame, center, 5, (0, 0, 255), -1)
221
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
      rototraslazione
232   def calculate_matrix(self, p0, p1, versor=[1,0]):
233
234     vet_diff = p1 - p0
235     x_axis = np.array(versor)
236     dot_product = np.dot(vet_diff,x_axis)
237     module = np.linalg.norm(vet_diff)
238     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),
        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
        [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,
      shell=True)
253
254 if __name__ == '__main__':
255
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.
      f_s_c2Lower, g.f_s_c2Upper)
263     point = g.get_Spider_Inseguitore(g.s_s_c1Lower, g.
      s_s_c1Upper)
264
265     key = cv2.waitKey(1) & 0xFF
266     if key == ord("q"):
267       g.close_all()
268       print("BYEEEEE")
269       break
270
271     if returns is not None and point is not None and returns[0]
      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
      rispetto all'asse y
297       epsilon_fron = 0.5
298       #epsilon di stop
299       epsilon_stop = 230
300
301       diff = np.linalg.norm(p1-p0, ord = 2)
302
303       #print(diff)
304       if diff < epsilon_stop:
305         if old_command != "stop":
306           old_command = "stop"
307           print("FERMATIIIIIIIII")
308           p = Process(target=launch_curl,args=('128128', ))
309           p.start()
310
311         continue
312
313       if abs(first_matrix[0][0]) <= epsilon_rot_robot:
314
315         diff = points["p0"] - points["p1"]
316
317         if diff[1] < 0:
318           #print("Arancione Davanti")
319
320           if cos_x > epsilon_rot_enemy:
321             print("Vai a Destra__Verticale__Arancione")
322             if old_command != "right":
323               old_command = "right"
324               p = Process(target=launch_curl,args=('000255', ))
325               p.start()
326               continue
327           elif cos_x < -epsilon_rot_enemy:
328             print("Vai a Sinistra__Verticale__Arancione")
329             if old_command != "left":
330               old_command = "left"
331               p = Process(target=launch_curl,args=('255000', ))
332               p.start()
333               continue
334
335           if cos_y > epsilon_fron:
336             print("Vai a Indietro__Verticale__Arancione")
337             if old_command != "back":
338               old_command = "back"
339               p = Process(target=launch_curl,args=('000000', ))
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 
447       else:
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

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 realtá aumentata. Ció migliorerá il riconoscimento dei ragni ed eliminerá la necessitá del doppio colore per stabilire l'orientamento del ragno inseguitore.
- L'algoritmo di inseguimento potrá essere migliorato pianificando traiettorie di curvatura e aggiungendo a bordo del ragno un sensore di prossimitá.
- Si potrebbe eliminare la necessitá di un sistema di orientamento globale montando una webcam direttamente sui ragni.