# Spider-chase

Marco Mecchia
Luigi Giugliano
Simone Romano

## Indice

# 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.

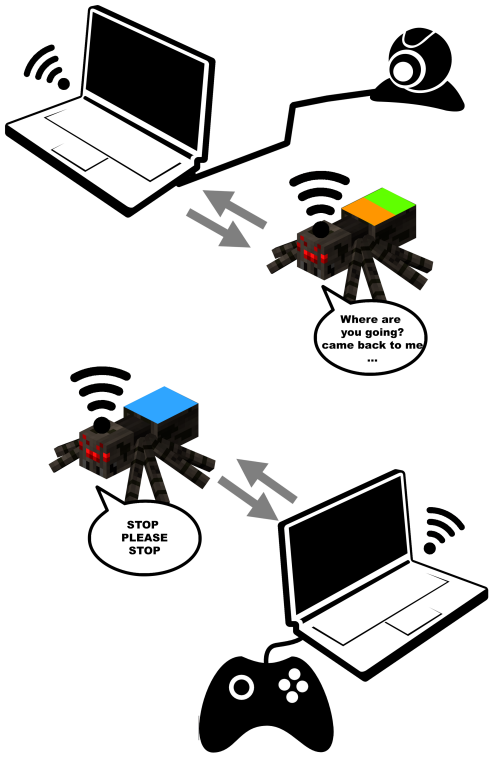
# 2 Architettura e componenti

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.

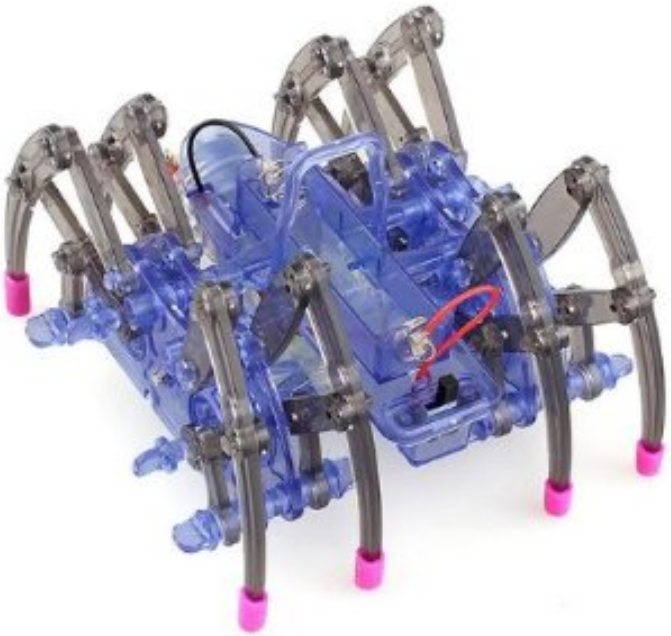
L'architettura totale pianificata é mostrata in figura.



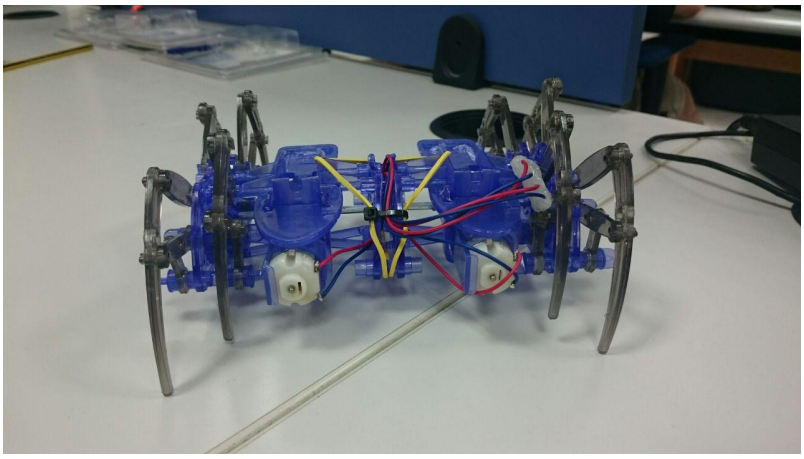

Tutti e tre i ragni possono ricevere messaggi wireless che impostano le velocitá dei motori. Ricevuto il messaggio, la board regola le velocitá tramite il driver. Il requisito di localizzazione é soddisfatto da un modulo di visione artificiale: una webcam posta in alto riconosce i ragni, ed il pc alla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

Figura Driver

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, piú il ragno va velocemente.

## 2.4 Modulo wireless

Il modulo wireless utilizzato é il modello mostrato in figura.

Figura Modulo

I suoi pin si collegano, oltre che all'alimentazione ed alla massa, agli ingressi seriali della board. La comunicazione seriale ci ha consentito di comunicare in maniera molto semplice con il modulo, grazie anche agli esempi forniti dalle funzioni di base di ChiBiOs. Il modulo lavora principalmente nello strato di rete, in quanto puó funzionare sia come client che come access point. Inoltre, se impostato come access point, lavora anche nello strato delle applicazioni, in quanto riconosce richieste HTTP di tipo sia GET che POST. Nel nostro caso, abbiamo fatto creare al modulo una rete WiFi, e per mandare messaggi al ragno é sufficiente mandare richieste GET tramite browser oppure curl una volta connessi alla rete creata dal modulo.

## 2.5 Visione Artificiale e logiche di inseguimento

Per il modulo di visione artificiale, abbiamo utilizzato lo standard *de facto* nel campo, **opencv**. Il modulo é stato progettato per essere utilizzato con una webcam o una fonte video piazzata in alto rispetto ai ragni. In

questo modo, é stata semplificata notevolmente la logica di inseguimento, in quanto dal punto di vista del modulo i ragni si muovono su una griglia bidimensionale. Esso riconosce i ragni tramite cartoncini colorati che sono stati messi al di sopra di loro. Una volta riconosciuti entrambi i ragni, il modulo invia un messaggio al ragno inseguitore tramite una richiesta HTTP GET. La logica di inseguimento seguita dal modulo e' molto semplice: Il ragno inseguitore aggiusta il proprio orientamento, cioé ruota su se stesso, fin quando gli "occhi" non puntano il ragno inseguito, dopodiche' il ragno prosegue dritto.

# 3 Codice

## 3.1 Controller

## 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;
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},
```

```
                {PWM_OUTPUT_DISABLED, NULL}},
                                0,
                0};

//configuration for right engine
static PWMConfig pwm2cfg = {10000,
                                500,
                                pwm2pcb,
                {{PWM_OUTPUT_ACTIVE_HIGH, pwm2c1cb},
                {PWM_OUTPUT_DISABLED, NULL},
                {PWM_OUTPUT_DISABLED, NULL},
                {PWM_OUTPUT_DISABLED, NULL}},
                                0,
                0};

void parse_string(char * command, int * velocity) {
  char left[3];
  char right[3];
  int toret[2];
  char type = command[0];

  left[0] = command[2];
  left[1] = command[3];
  left[2] = command[4];

  right[0] = command[5];
  right[1] = command[6];
  right[2] = command[7];

  int le, ri;

  le = atoi(left);
  ri = atoi(right);

  velocity[0] = le;
  velocity[1] = ri;
}

void init_motor() {
  mapping.type1 = GPIOA;
  mapping.port1 = GPIOA_PIN8;

  mapping.type2 = GPIOB;
  mapping.port2 = GPIOB_PIN10;

  mapping.type3 = GPIOB;
  mapping.port3 = GPIOB_PIN4;

  mapping.type4 = GPIOB;
  mapping.port4 = GPIOB_PIN5;

  palSetPadMode(mapping.type1, mapping.port1,
                  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

```
1  #include "ch.h"
2  #include "hal.h"
3  #include "test.h"
4  #include "chprintf.h"
5
6  #define EOF '\377'
7  #define WIFI_SERIAL &SD1
8  #define MONITOR_SERIAL &SD2
9  #define MAX_LENGTH  100
10 #define TIME_IMMEDIATE  ((systime_t)0)
11 #define MSG_TIMEOUT (msg_t)-1
12 #define Q_TIMEOUT   MSG_TIMEOUT
13 #define COMMAND_SLEEP 500
14 #define COMMAND_LONG_SLEEP 20000
15 #define DEBUG 0
16
17 char * readResponse(void);
18 void printWebPage(void);
19 int mystrcontains(char* text, char* toFind);
20 void sendToESP8266(char* command, int delay);
21 void readAndPrintResponse(void);
22 int mystrlen(char* text);
23 static void println(char *p);
24 void blinkBoardLed(void);
25 char* StrStr(const char *str, const char *target);
26 char *strcat(char *dest, const char *src);
```

```
27 char *strcpy(char *dest, const char *src);
28 int strlen(const char * str);
29 void itoa(int n, char s[]);
30 void reverse(char s[]);
31
32 static SerialConfig uartCfgWiFi = {115200,
33     };
34
35 static char* ESP8266_HELLO = "AT\r\n";
36 static char* ESP8266_RESET = "AT+RST\r\n";
37 static char* ESP8266_LIST_WIFI = "AT+CWLAP\r\n";
38 static char* ESP8266_CONNECT_TO_WIFI =
39     "AT+CWJAP=\"Romano Wi-Fi\",\"160462160867\"\r\n";
40 static char* ESP8266_CHECK_IP = "AT+CIFSR\r\n";
41 static char* ESP8266_GET_IP_ADD = "AT+CIFSR\r\n";
42 static char* ESP8266_CHECK_VERSION = "AT+GMR\r\n";
43 static char* ESP8266_MULTIPLE_CONNECTION = "AT+CIPMUX=1\r\n";
44 static char* ESP8266_START_SERVER = "AT+CIPSERVER=1,80\r\n";
45 static char* ESP8266_SET_AS_ACCESS_POINT = "AT+CWMODE=2\r\n";
46 static char* ESP8266_SET_AS_CLIENT = "AT+CWMODE=1\r\n";
47 static char* ESP8266_SEND_TCP_DATA = "AT+CIPSEND=";
48 static char* ESP8266_CLOSE_CONN = "AT+CIPCLOSE=";
49 char clientID[2];
50 char command[9];
51 char request;
52 static THD_WORKING_AREA(waThread1, 2048);
53
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
```

```
import time
from multiprocessing import Process

class Vision:


  def __init__ (self):

    #arancione
    self.first_spider_color1 = np.uint8([[[57,114,255]]])
    #verde
    self.first_spider_color2 = np.uint8([[[102,151,49]]])
    #fucsia
    #self.first_spider_color2 = np.uint8([[[180,95,245]]])

    #azzurro
    self.second_spider_color1 = np.uint8([[[216,145,0]]])
    #rosso
    #self.second_spider_color1 = np.uint8([[[78,63,223]]])
    #giallo
    self.second_spider_color2 = np.uint8([[[0,255,255]]])


    f_s_c1HSV = cv2.cvtColor(self.first_spider_color1,cv2.
    COLOR_BGR2HSV)
    self.f_s_c1Lower = np.array((f_s_c1HSV[0][0][0]-10,100,100))
    self.f_s_c1Upper = np.array((f_s_c1HSV[0][0][0]+10,255,255))

    f_s_c2HSV = cv2.cvtColor(self.first_spider_color2,cv2.
    COLOR_BGR2HSV)
    self.f_s_c2Lower = np.array((f_s_c2HSV[0][0][0]-10,100,100))
    self.f_s_c2Upper = np.array((f_s_c2HSV[0][0][0]+10,255,255))

    s_s_c1HSV = cv2.cvtColor(self.second_spider_color1,cv2.
    COLOR_BGR2HSV)
    self.s_s_c1Lower = np.array((s_s_c1HSV[0][0][0]-10,100,100))
    self.s_s_c1Upper = np.array((s_s_c1HSV[0][0][0]+10,255,255))

    s_s_c2HSV = cv2.cvtColor(self.second_spider_color2,cv2.
    COLOR_BGR2HSV)
    self.s_s_c2Lower = np.array((s_s_c2HSV[0][0][0]-10,100,100))
    self.s_s_c2Upper = np.array((s_s_c2HSV[0][0][0]+10,255,255))

    self.camera = cv2.VideoCapture(1)

  def get_Spider(self, color1Lower, color1Upper, color2Lower,
    color2Upper):

    (grabbed, frame) = self.camera.read()
    #frame = cv2.flip(frame,1)
    s_x = None
    s_y = None
    f_x = None
    f_y = None
```

```
55      # if we are viewing a video and we did not grab a frame,
56      # then we have reached the end of the video
57      if not grabbed:
58        print("No frame D:")
59
60      # resize the frame, blur it, and convert it to the HSV
61      # color space
62      frame = imutils.resize(frame, width=900)
63      blurred = cv2.GaussianBlur(frame, (11, 11), 0)
64      #cv2.imshow("Blurred", blurred)
65      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
66      cv2.imshow("hsv", hsv)
67
68      # construct a mask for the color "green", then perform
69      # a series of dilations and erosions to remove any small
70      # blobs left in the mask
71      mask = cv2.inRange(hsv, color1Lower, color1Upper)
72      mask = cv2.erode(mask, None, iterations=3)
73      mask = cv2.dilate(mask, None, iterations=2)
74
75      cv2.imshow("mask1",cv2.bitwise_and(hsv,hsv,mask=mask));
76
77      # find contours in the mask and initialize the current
78      # (x, y) center of the ball
79      cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
80      cv2.CHAIN_APPROX_SIMPLE)[-2]
81      center = None
82      # only proceed if at least one contour was found
83      if len(cnts) > 0:
84        # find the largest contour in the mask, then use
85        # it to compute the minimum enclosing circle and
86        # centroid
87        c = max(cnts, key=cv2.contourArea)
88        ((f_x, f_y), radius) = cv2.minEnclosingCircle(c)
89        M = cv2.moments(c)
90        center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"
        ]))
91
92        # only proceed if the radius meets a minimum size
93        if radius > 5:
94          # draw the circle and centroid on the frame,
95          # then update the list of tracked points
96          cv2.circle(frame, (int(f_x), int(f_y)), int(radius),
97          (0, 255, 255), 2)
98          cv2.circle(frame, center, 5, (0, 0, 255), -1)
99
100
101     mask = cv2.inRange(hsv, color2Lower, color2Upper)
102     mask = cv2.erode(mask, None, iterations=3)
103     mask = cv2.dilate(mask, None, iterations=2)
104
105     cv2.imshow("mask2",cv2.bitwise_and(hsv,hsv,mask=mask));
106
107     # find contours in the mask and initialize the current
```

```
108     # (x, y) center of the ball
109     cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
110     cv2.CHAIN_APPROX_SIMPLE)[-2]
111     center = None
112     # only proceed if at least one contour was found
113     if len(cnts) > 0:
114       # find the largest contour in the mask, then use
115       # it to compute the minimum enclosing circle and
116       # centroid
117       c = max(cnts, key=cv2.contourArea)
118       ((s_x, s_y), radius) = cv2.minEnclosingCircle(c)
119       M = cv2.moments(c)
120       center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"
      ]))
121
122       # only proceed if the radius meets a minimum size
123       if radius > 5:
124         # draw the circle and centroid on the frame,
125         # then update the list of tracked points
126         cv2.circle(frame, (int(s_x), int(s_y)), int(radius),
127         (255, 255, 0), 2)
128         cv2.circle(frame, center, 5, (0, 0, 255), -1)
129
130     cv2.imshow("Frame", frame)
131     key = cv2.waitKey(1) & 0xFF
132     if key == ord("q"):
133       close_all()
134
135     if f_x and f_y and s_x and s_y:
136       p0 = np.array([f_x,f_y])
137       p1 = np.array([s_x,s_y])
138
139       points = dict()
140
141       points["p0"] = p0
142       points["p1"] = p1
143
144       return [points, self.calculate_matrix(p0,p1)]
145
146
147   def get_Spider_Inseguitore(self, color1Lower, color1Upper):
148
149     (grabbed, frame) = self.camera.read()
150     #frame = cv2.flip(frame,1)
151     s_x = None
152     s_y = None
153     f_x = None
154     f_y = None
155     # if we are viewing a video and we did not grab a frame,
156     # then we have reached the end of the video
157     if not grabbed:
158       print("No frame D:")
159
160     # resize the frame, blur it, and convert it to the HSV
```

```
161     # color space
162     frame = imutils.resize(frame, width=900)
163     blurred = cv2.GaussianBlur(frame, (11, 11), 0)
164     #cv2.imshow("Blurred", blurred)
165     hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
166     cv2.imshow("hsv", hsv)
167
168     # construct a mask for the color "green", then perform
169     # a series of dilations and erosions to remove any small
170     # blobs left in the mask
171     mask = cv2.inRange(hsv, color1Lower, color1Upper)
172     mask = cv2.erode(mask, None, iterations=3)
173     mask = cv2.dilate(mask, None, iterations=2)
174
175     cv2.imshow("mask1",cv2.bitwise_and(hsv,hsv,mask=mask));
176
177     # find contours in the mask and initialize the current
178     # (x, y) center of the ball
179     cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
180     cv2.CHAIN_APPROX_SIMPLE)[-2]
181     center = None
182     # only proceed if at least one contour was found
183     if len(cnts) > 0:
184       # find the largest contour in the mask, then use
185       # it to compute the minimum enclosing circle and
186       # centroid
187       c = max(cnts, key=cv2.contourArea)
188       ((f_x, f_y), radius) = cv2.minEnclosingCircle(c)
189       M = cv2.moments(c)
190       center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"
      ]))
191
192       # only proceed if the radius meets a minimum size
193       if radius > 5:
194         # draw the circle and centroid on the frame,
195         # then update the list of tracked points
196         cv2.circle(frame, (int(f_x), int(f_y)), int(radius),
197         (0, 255, 255), 2)
198         cv2.circle(frame, center, 5, (0, 0, 255), -1)
199
200
201
202     cv2.imshow("Frame", frame)
203     key = cv2.waitKey(1) & 0xFF
204     if key == ord("q"):
205       close_all()
206
207     if f_x and f_y:
208       p1 = np.array([f_x, f_y])
209       return p1
210
211   #dati due punti restituisce la matrice omogenea di
        rototraslazione
212   def calculate_matrix(self, p0, p1, versor=[1,0]):
```

```
213
214         vet_diff = p1 - p0
215         x_axis = np.array(versor)
216         dot_product = np.dot(vet_diff,x_axis)
217         module = np.linalg.norm(vet_diff)
218         cos_arg = dot_product/module
219         angle = np.arccos(cos_arg)
220
221         #print(vet_diff, cos_arg, np.degrees(angle))
222
223         rot_matrix = [cos_arg, -np.sin(angle), np.sin(angle),
            cos_arg]
224         centr_vet = [(p0[0]+p1[0])/2, (p0[1]+p1[1])/2]
225         matrix = np.array([rot_matrix[0], rot_matrix[1], centr_vet
            [0], rot_matrix[2], rot_matrix[3], centr_vet[1], 0 ,0 ,1])
226         matrix = np.reshape(matrix, (3,3))
227         #print(matrix)
228         return matrix
229
230     def close_all():
231         # cleanup the camera and close any open windows
232         self.camera.release()
233         cv2.destroyAllWindows()
234
235 def launch_curl(string):
236     subprocess.call("curl -m 1 http://192.168.4.1/?c=m0"+string,
            shell=True)
237
238 if __name__ == '__main__':
239
240     g = Vision()
241
242     stop = True
243     old_command = "stop"
244
245     while True:
246         returns = g.get_Spider(g.f_s_c1Lower, g.f_s_c1Upper, g.
            f_s_c2Lower, g.f_s_c2Upper)
247         point = g.get_Spider_Inseguitore(g.s_s_c1Lower, g.
            s_s_c1Upper)
248
249         if returns is not None and point is not None and returns[0]
            is not None and returns[1] is not None:
250             #print(first_matrix, second_matrix)
251
252             #print(first_matrix[0][0])
253
254             print (old_command)
255             points = returns[0]
256             first_matrix = returns[1]
257
258             p0 = [first_matrix[0][2], first_matrix[1][2]]
259             p1 = point
260
```

```
261         new_matrix_x= g.calculate_matrix(np.array(p0), p1)
262         new_matrix_y= g.calculate_matrix(np.array(p0), p1, [0,1])
263 
264         cos_x = new_matrix_x[0][0]
265         cos_y = new_matrix_y[0][0]
266 
267         #print("Coseno rispetto X: "+ str(cos_x))
268         #print("Coseno rispetto Y: "+ str(cos_y))
269 
270         #epsilon di rotazione del robot rispetto alla telecamera
271         epsilon_rot_robot = 0.8
272         #epsilon di rotazione dell'avversario rispetto al robot
273         epsilon_rot_enemy = 0.5
274         #epsilon di rotazione dell'avversario rispetto al robot
        rispetto all'asse y
275         epsilon_fron = 0.5
276         #epsilon di stop
277         epsilon_stop = 230
278 
279         diff = np.linalg.norm(p1-p0, ord = 2)
280 
281         #print(diff)
282         if diff < epsilon_stop:
283           if old_command != "stop":
284             old_command = "stop"
285             print("FERMATIIIIIIII")
286             p = Process(target=launch_curl,args=('128128', ))
287             p.start()
288 
289           continue
290 
291         if abs(first_matrix[0][0]) <= epsilon_rot_robot:
292 
293           diff = points["p0"] - points["p1"]
294 
295           if diff[1] < 0:
296             #print("Arancione Davanti")
297 
298             if cos_x > epsilon_rot_enemy:
299               print("Vai a Destra__Verticale__Arancione")
300               if old_command != "right":
301                 old_command = "right"
302                 p = Process(target=launch_curl,args=('000255', ))
303                 p.start()
304                 continue
305             elif cos_x < -epsilon_rot_enemy:
306               print("Vai a Sinistra__Verticale__Arancione")
307               if old_command != "left":
308                 old_command = "left"
309                 p = Process(target=launch_curl,args=('255000', ))
310                 p.start()
311                 continue
312 
313             if cos_y > epsilon_fron:
```

```
314             print("Vai a Indietro__Verticale__Arancione")
315             if old_command != "back":
316               old_command = "back"
317               p = Process(target=launch_curl,args=('000000', ))
318               p.start()
319               continue
320           elif cos_y < -epsilon_fron:
321             print("Vai a Avanti__Verticale__Arancione")
322             if old_command != "front":
323               old_command = "front"
324               p = Process(target=launch_curl,args=('255255', ))
325               p.start()
326               continue
327         else:
328           #print("Verde Davanti")
329           if cos_x > epsilon_rot_enemy:
330             if old_command != "left":
331               old_command = "left"
332               print("Vai a Sinistra__Verticale__Arancione")
333               p = Process(target=launch_curl,args=('255000', ))
334               p.start()
335               continue
336           elif cos_x < -epsilon_rot_enemy:
337             if old_command != "right":
338               old_command = "right"
339               print("Vai a Destra__Verticale__Arancione")
340               p = Process(target=launch_curl,args=('000255', ))
341               p.start()
342               continue
343
344           if cos_y > epsilon_fron:
345             if old_command != "front":
346               old_command = "front"
347               p = Process(target=launch_curl,args=('255255', ))
348               p.start()
349               print("Vai a Avanti__Verticale__Arancione")
350               continue
351           elif cos_y < -epsilon_fron:
352             if old_command != "back":
353               old_command = "back"
354               p = Process(target=launch_curl,args=('000000', ))
355               p.start()
356               print("Vai a Indietro__Verticale__Arancione")
357               continue
358       else:
359         diff = points["p0"] - points["p1"]
360
361         if diff[0] < 0:
362           #print("Arancione Sinistra")
363           if cos_x > epsilon_rot_enemy:
364             if old_command != "back":
365               old_command = "back"
366               print("Vai a Indietro__Orizzontale__Arancione")
367               p = Process(target=launch_curl,args=('000000', ))
```

```
                p.start()
                continue
            elif cos_x < -epsilon_rot_enemy:
              if old_command != "front":
                old_command = "front"
                print("Vai a Avanti__Orizzontale__Arancione")
                p = Process(target=launch_curl,args=('255255', ))
                p.start()
                continue

            if cos_y > epsilon_fron:
              if old_command != "left":
                old_command = "left"
                p = Process(target=launch_curl,args=('255000', ))
                p.start()
                print("Vai a Sinistra__Orizzontale__Arancione")
                continue
            elif cos_y < -epsilon_fron:
              if old_command != "right":
                old_command = "right"
                p = Process(target=launch_curl,args=('000255', ))
                p.start()
                print("Vai a Destra__Orizzontale__Arancione")
                continue
          else:
            #print("Verde Sinistra")
            if cos_x > epsilon_rot_enemy:
              if old_command != "front":
                old_command = "front"
                print("Vai a Avanti__Orizzontale__Verde")
                p = Process(target=launch_curl,args=('255255', ))
                p.start()
                continue
            elif cos_x < -epsilon_rot_enemy:
              if old_command != "back":
                old_command = "back"
                print("Vai a Indietro__Orizzontale__Arancione")
                p = Process(target=launch_curl,args=('000000', ))
                p.start()
                continue

            if cos_y > epsilon_fron:
              if old_command != "right":
                old_command = "right"
                p = Process(target=launch_curl,args=('000255', ))
                p.start()
                print("Vai a Destra__Orizzontale__Arancione")
                continue
            elif cos_y < -epsilon_fron:
              if old_command != "left":
                old_command = "left"
                p = Process(target=launch_curl,args=('255000', ))
                p.start()
                print("Vai a Sinistra__Orizzontale__Arancione")
```

```
422                 continue
423
424
425         else:
426           print("Missing one of the components")
427           if old_command != "stop":
428             old_command = "stop"
429             p = Process(target=launch_curl,args=('128128', ))
430             p.start()
431           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.