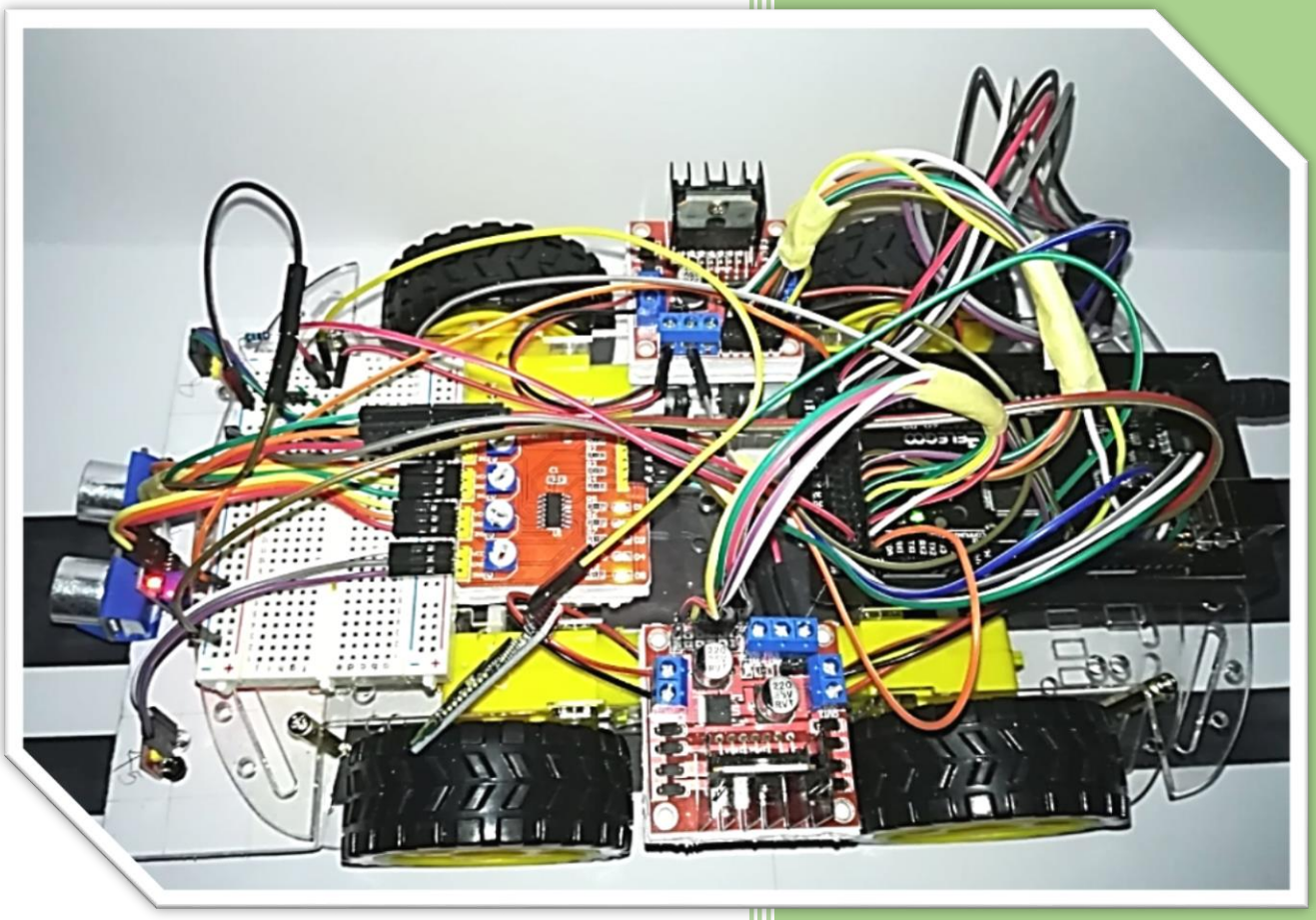
Anno accademico
2017/2018

Progetto Sistemi Embedded



Magni Andrea & Mercanti Davide
Università degli Studi di Milano
Anno accademico 2017/2018

BREVE DESCRIZIONE

Il progetto che abbiamo deciso di realizzare prevede la costruzione di una "macchina intelligente".

La macchina andrà a svolgere diverse funzioni, principalmente di comunicazione o di valutazione e risoluzione di problematiche, tra le quali:

- riuscire a muoversi all'interno di una strada composta da due corsie, rimanendo all'interno di una corsia
- riuscire ad effettuare una manovra di cambio corsia
- valutare la presenza di un semaforo
  - nel caso in cui venga rilevato un semaforo valuti la sua condizione e regoli la velocità di conseguenza
- valutare la presenza di possibili ostacoli durante il percorso
  - nel caso in cui venga rilevato un ostacolo valuti il tipo di ostacolo (mobile o fisso) e decida se continuare il percorso o effettuare una manovra di cambio corsia
- comunicare con un telefono per permettere lo scambio di informazioni o ricezione di comandi tra la macchina e l'utente in diretta

ELENCO COMPONENTI

Quantità	Componente	Link Amazon
1	Board Arduino Mega	https://www.amazon.it/dp/B06XRJNB27/
1	Board generica (non sono necessari molti GPIO)	https://www.amazon.it/dp/B01MRJR8UF/
1	Kit telaio ruote motori	https://www.amazon.it/dp/B073169GBF/
2	Modulo gestione motori - **L298N**	https://www.amazon.it/dp/B013QTC18K/
2	Modulo radio - **nRF24L01**	https://www.amazon.it/dp/B016BAPC0K/
1	Modulo Bluetooth - **HC-05**	https://www.amazon.it/dp/B01G9KSAF6/
4	Sensore ad infrarossi - **TE174** o **FC-35**	https://www.amazon.it/dp/B00XXEJD58/
3	Sensore ad ultrasuoni - **HC-SR04**	https://www.amazon.it/dp/B00R2U8HK6/
1	Resistenza 8.2 ohm	
3	Resistenza 220 ohm	
1	Resistenza 1 Kohm	https://www.amazon.it/dp/B00J02WSFE/
1	Resistenza 2.2 Kohm	
1	Resistenza 10 Kohm	
2	Breadboard	https://www.amazon.it/dp/B06XRDRT2R/
1	Cavetteria	https://www.amazon.it/dp/B01N40EK6M/
	Polistirolo cartonato	

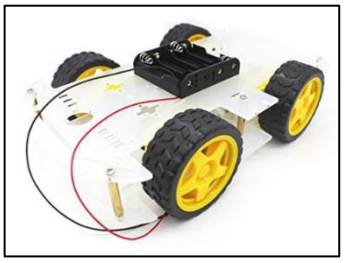
DESCRIZIONE COMPONENTI UTILIZZATI

Per la creazione di questo progetto sono stati utilizzati diversi componenti, aggiungendo sempre più funzionalità e potendo così testare il funzionamento di diversi sensori e moduli con relative interazioni all'interno di un singolo progetto.

Andremo ora ad analizzare i principali componenti utilizzati all'interno del progetto:

Telaio / Motori / Ruote

Per la realizzazione di questo progetto o simili è sicuramente fondamentale una scocca che permetta alla macchina di sostenere i vari componenti e successivamente, una volta realizzato il codice e inseriti i vari sensori e moduli, potersi muovere liberamente o all'interno di un tracciato.

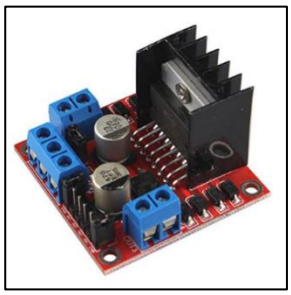


Sicuramente la scocca nello specifico è un componente il cui acquisto non risulta strettamente necessario (sarebbe stato possibile crearlo attraverso del cartone), avendo però la necessità di comprare delle ruote e relativi motori a corrente continua la differenza di costo tra il comprare un kit telaio/motori/ruote oppure delle ruote e motori singolarmente era irrisoria se non addirittura più conveniente. Abbiamo quindi optato per questo kit con telaio realizzato in plexiglass. I motori presenti nel kit sono dei comunissimi motori a corrente continua ai quali è possibile fornire una tensione massima di 6 Volt. Le ruote sono un mix di plastica e gomma.

Per quanto questo kit abbia comportato grandi comodità, bisogna anche dire che i motori non sono particolarmente precisi. Questo lato negativo è evidente durante la guida automatica della macchina, la quale presenta piccole imprecisioni che possono essere notate durante il suo percorso e sono dovute alla non perfetta taratura dei motori. La comodità portata dal kit unita al costo irrisorio ne fanno un acquisto decisamente sensato per il progetto.

Modulo L298N

Questo modulo è sicuramente un acquisto necessario quando si vuole realizzare un progetto di questo genere.

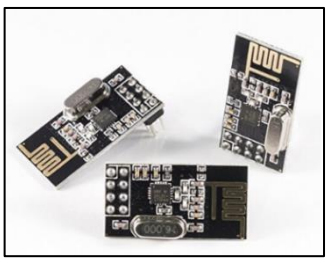


Il modulo L298N è utilizzato per comandare fino ad un massimo di due motori a corrente continua (o di altro genere), può funzionare con tensioni che variano dai 5 ai 35 Volt e fornire sino a 2 Ampere di corrente per ogni uscita. Particolarmente utile la presenza di un'uscita stabile a 5 Volt, la quale può essere utilizzata per alimentare Arduino, dei componenti oppure per far passare 5 Volt sulla breadboard.

Per il progetto sono stati utilizzati due di questi componenti, potendo così gestire singolarmente ogni motore. Questa soluzione non è necessaria in quanto si potrebbe utilizzare un solo modulo collegando in parallelo a coppie i motori.

Modulo nRF24L01

Questo modulo viene utilizzato per creare un canale radio attraverso il quale comunicare. Esso non è necessario nel momento in cui si voglia costruire un semplice line follower, può invece risultare molto utile nel momento in cui si ha la necessità di comunicare con altri componenti (ad esempio un radiocomando per controllare la macchina).



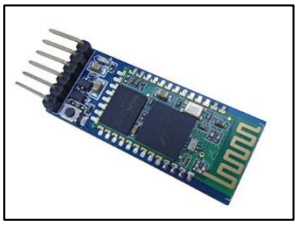
All'interno del progetto è stato inserito per permettere la comunicazione con un secondo Arduino, all'interno del quale è caricato un programma che simula un semaforo.

Il semaforo e la macchina potranno quindi comunicare tra loro, nello specifico sarà il semaforo che invierà alla macchina il suo stato, la macchina andrà ad analizzare lo stato corrente e valuterà come comportarsi.

Modulo HC-05

Questo modulo permette di gestire una comunicazione tramite Bluetooth e non è sicuramente un acquisto necessario per il progetto dato che un'eventuale connessione tra Arduino e un secondo Arduino è già permessa dal modulo radio.

La scelta di aggiungere questo modulo è stata fatta per due motivi, il principale è quello di poter comunicare con un telefono tramite una delle tante applicazioni presenti sugli store e poter così effettuare un debug in live, molto utile per determinare problemi nel comportamento della macchina in fase di progettazione. In precedenza al posto del Bluetooth era presente un display LCD sul quale erano visualizzate le informazioni. Il secondo motivo per cui ci siamo interessati a questo componente è la possibilità di avere una connessione senza fili con un PC con cui scambiare dati nel caso di futuri sviluppi del progetto, come ad esempio delle applicazioni con Processing per visualizzare il movimento della macchina in live.

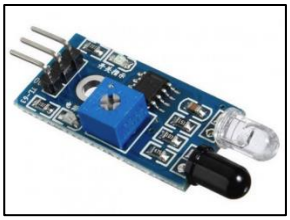


Sensore HC-SR04

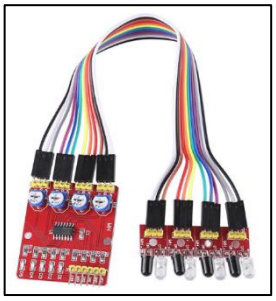
Questo sensore permette di valutare la distanza tra esso ed un oggetto posto di fronte a lui tramite l'utilizzo degli ultrasuoni. Componente sicuramente interessante per i progetti in cui si interagisce con l'ambiente circostante.

Trova un doppio utilizzo all'interno del progetto poiché risulta utile sia per la macchina che per il semaforo. Per la macchina la sua funzione è quella di permettere la valutazione della presenza di eventuali ostacoli di fronte al suo percorso, per quanto riguarda il semaforo invece il suo utilizzo è quello di creare una zona di apertura e una di chiusura di invio del segnale radio, evitando così l'invio continuo di dati tra il semaforo e la macchina che potrebbero portare a situazioni indesiderate (la macchina che si ferma lontano dal semaforo).

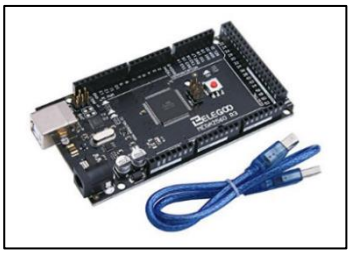
Sensore FC-35 / FC-51



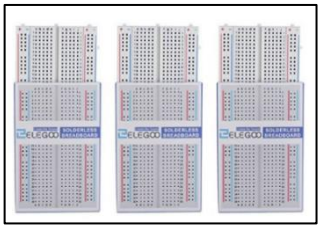
Questo modulo ad infrarossi viene utilizzato per creare permettere alla macchina di seguire un percorso. Il funzionamento del modulo è molto semplice, è presente un led la cui funzione è quella di trasmettitore ed un fotodiodo o fototransistor che funziona come ricevitore. Il trasmettitore IR invia un segnale infrarosso che, nel caso di una superficie riflettente (ad esempio di colore bianco), rimbalza in alcune direzioni compresa quella del ricevitore IR il quale cattura il segnale rilevando l'oggetto. Nel caso di una superficie assorbente (ad esempio di colore nero) il segnale IR non è riflesso e l'oggetto non può essere rilevato dal sensore. Questo risultato si verificherebbe anche se l'oggetto è assente.



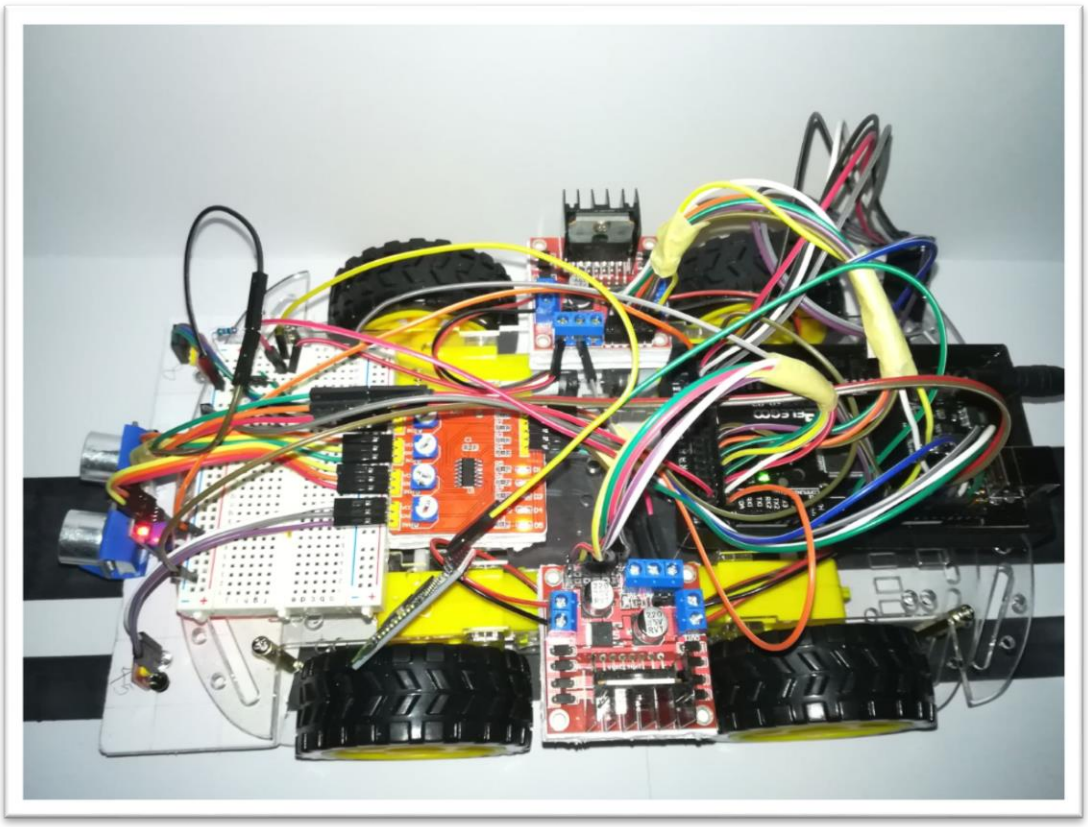
Componentistica varia

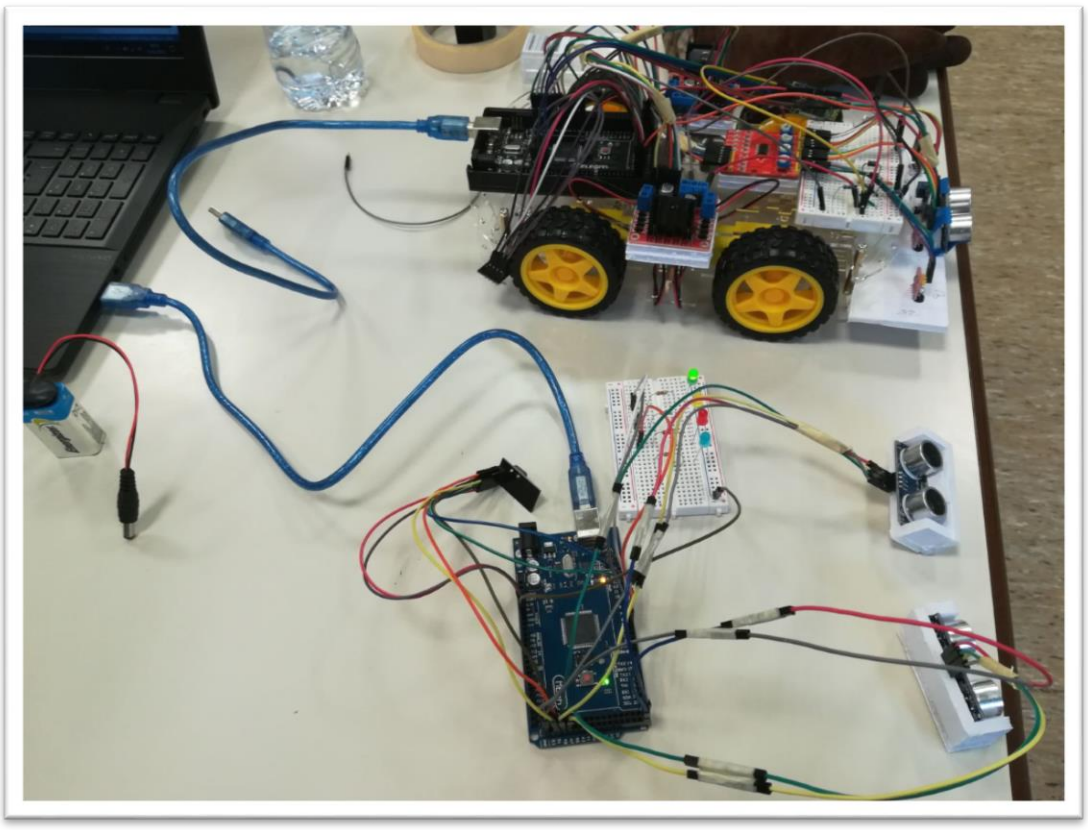


Per concludere, gli altri componenti utilizzati per il progetto sono una piattaforma Arduino Mega e relativa breadboard per comandare i vari moduli e sensori della macchina, un secondo Arduino Mega con breadboard per creare il semaforo. Inoltre è stata usata molta cavetteria, delle resistenze, dei led per mostrare lo stato del semaforo, due pile 18650 da 3.7 Volt e due pile da 9 Volt per alimentare i due Arduino.

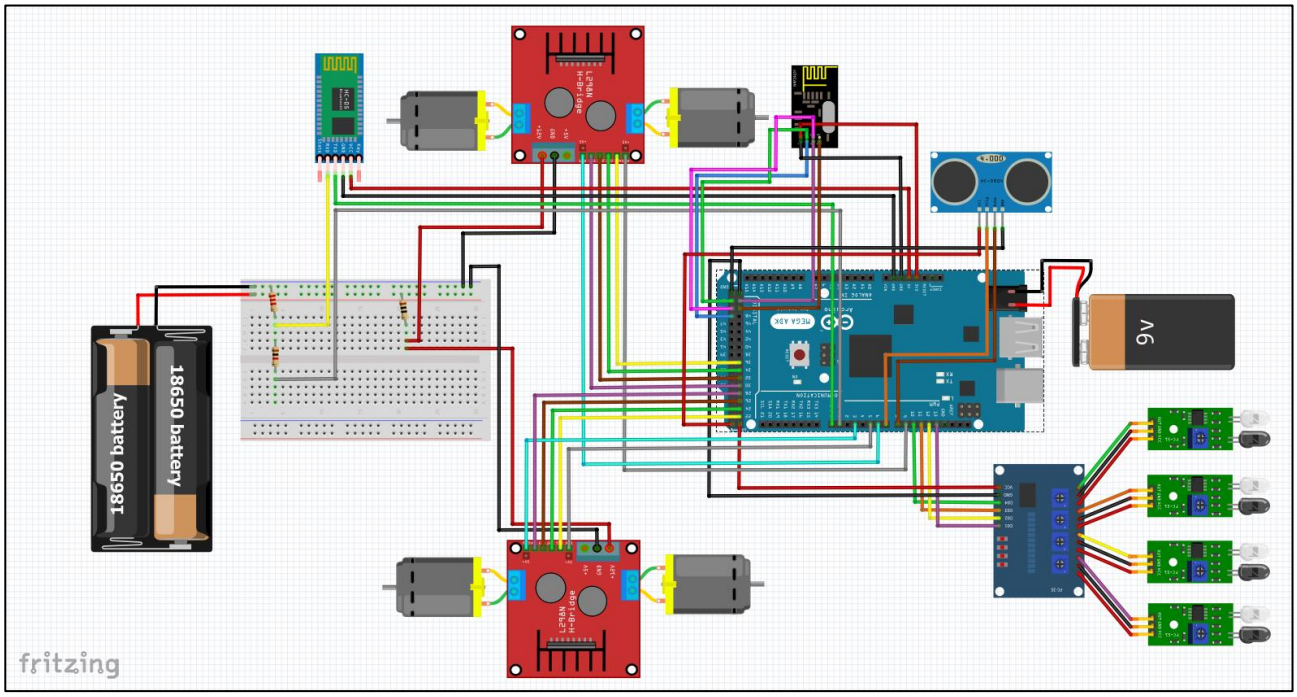


FOTO

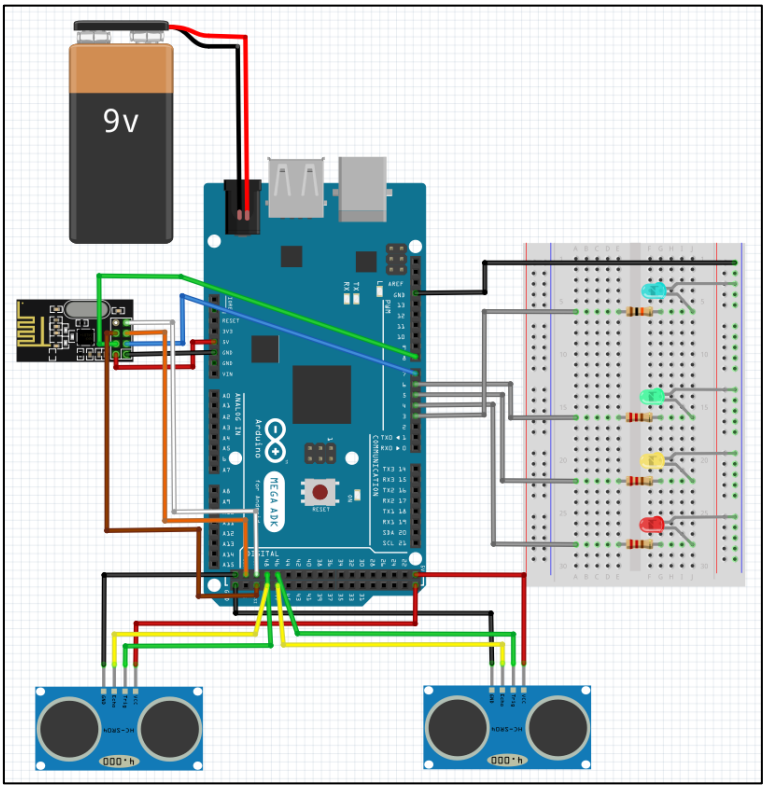




SCHEMA COLLEGAMENTI MACCHINA



SCHEMA COLLEGAMENTI SEMAFORO



POSSIBILI UPGRADE

- Realizzazione di un sistema di semafori, ad esempio un incrocio, il quale valuti la presenza di cose di macchine nelle varie strade e decida come impostare i semafori in modo intelligente.
- Realizzazione di un sistema GPS per tenere traccia degli spostamenti della macchina, probabilmente sulla piccola scala dei movimenti della macchina potrebbe essere difficile o inutile da sviluppare ma potrebbe portare ad idee alternative collegate.
- Realizzazione di un collegamento tra la macchina e Processing scrivendo del codice per ricreare il tracciato percorso dalla macchina durante ogni sessione di utilizzo.
- Realizzazione di un sistema di sterzo per permettere alla macchina di curvare.
- Realizzazione di un sonar mobile, creato unendo un sensore ad ultrasuoni ad un servomotore. In questo modo sarebbe possibile analizzare le distanze degli oggetti non solo se frontali rispetto al modulo ad ultrasuoni ma anche attorno ad esso.

COLLEGAMENTO MACCHINA - SMARTPHONE

Il progetto ha la possibilità di interfacciarsi ad una periferica che supporta il collegamento Bluetooth, personalmente abbiamo testato le funzionalità del progetto grazie a questa applicazione presente sullo store Android:
https://play.google.com/store/apps/details?id=com.giumig.apps.bluetoothserialmonitor&hl=it

Non sono garantite compatibilità con altre applicazioni dato che per il momento non abbiamo avuto modo di effettuare test multipli.

Nel momento in cui si voglia stabilire una connessione con la macchina, dopo aver installato ed aperto l'applicazione dovrebbe comparare il dispositivo **HC-05**. Selezionarlo ed inserire come key **1234** nel caso in cui sia richiesta. A questo punto avrete una connessione attiva tra macchina e smartphone.

Il telefono può inviare determinati comandi premendo i seguenti numeri:

1 *Visualizzazione comandi*
2 *Attivazione debug*
3 *Disattivazione debug*
4 *Aumento potenza motori*
5 *Riduzione potenza motori*
6 *Attivazione controllo manuale*
7 *Disattivazione controllo manuale*
8 *Attivazione modalità standby*
9 *Disattivazione modalità standby*

Per utilizzare i comandi manuali uscire interfacciarsi al modulo Bluetooth come Joystick invece che come terminale e usare i seguenti tasti:

W *Per andare in avanti*
A *Per andare a sinistra*
D *Per andare a destra*
S *Per andare in retro*
Z *Per fermare la macchina*
Q *Per uscire dalla modalità manuale impostando lo standby*

CODICE REALIZZATO

Per la realizzazione di questo progetto sono stati creati principalmente quattro file:

- **macchina.ino** per la gestione della macchina
- **semaforo.ino** per la gestione del semaforo
- **motor.h** e **motor.cpp** una libreria creata appositamente per il progetto

Inoltre sono in fase di testing ulteriori file / funzioni che non siamo ancora riusciti ad implementare al 100% al momento della stesura della relazione ma verranno comunque integrati successivamente.

```
/*
 * Nome file: macchina.ino
 *
 * Questo programma permette alla macchina, attraverso diverse funzioni, sensori e moduli di poter valutare
 * ed interagire con l'esterno. Nello specifico la macchina dovrà riuscire a muoversi all'interno di una strada
 * appositamente creata e comunicare attraverso segnali Bluetooth e Radio con altri oggetti.
 *
 * Informazioni dettagliate sul progetto al seguente link: https://github.com/AndreaMagni/SistemiEmbedded/
 *
 * Copyright © 2018, Magni Andrea & Mercanti Davide, 23 Febbraio 2018
 *
 * This program is free software: you can redistribute it and/or modify it under the terms of the GNU General
 * Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the
 * implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General Public License with this program.
 * If not, see <http://www.gnu.org/licenses/>.
 */

/*
 * Inclusione librerie necessarie per il funzionamento del programma
 *
 * NewPing.h utilizzata per la gestione del sensore ad ultrasuoni
 * SPI.h utilizzata per la comunicazione con dispositivi SPI (Serial Peripheral Interface)
 * RF24.h utilizzata per la gestione del modulo radio
 * Motor.h creata per semplificare l'utilizzo dei motori
 */

#include <NewPing.h>
#include <SPI.h>
#include <RF24.h>
#include "Motor.h"
```

```
/*
 *  Definizione etichette per i sensori ad infrarossi
 *  nessun ostacolo rilevato [1] - ostacolo rilevato [0]
 *
 *  DETECT_ED - Sensore esterno destro
 *  DETECT_CD - Sensore centrale destro
 *  DETECT_CS - Sensore centrale sinistro
 *  DETECT_ES - Sensore esterno sinistro
 */

#define DETECT_ED 10
#define DETECT_CD 11
#define DETECT_CS 12
#define DETECT_ES 13

/*
 *  Dichiarazione e assegnazione di diverse variabili
 *
 *  turn_left: stato della macchina "cambio corsia verso sinistra"
 *  turn_right: stato della macchina "cambio corsia verso destra"
 *  debug: stato debug della macchina (attivo / disattivo)
 *  standby: stato standby della macchina (attivo / disattivo)
 *  manual: stato comandi manuali della macchina (attivo / disattivo)
 *  command: determina il comando da eseguire tramite Bluetooth
 *  received_loop: determina l'ultimo loop ricevuto dal semaforo
 *  max_acceptable_loop: determina il massimo loop accettabile oltre il quale la
 *                                macchina valuta il semaforo giallo come se fosse rosso
 *  motor_power_plus: determina la quantità di potenza aggiunta via Bluetooth ai motori
 *  motor_power_minus: determina la quantità di potenza tolta via Bluetooth ai motori
 *  motor_power_difference: differenza tra motor_power_plus e motor_power_minus
 *  set_delay: imposta il delay dopo l'invio di un messaggio del menù Bluetooth
 *  obstacle_distance: imposta la distanza sotto la quale viene rilevato un ostacolo
 *  pipe: definisce l’indirizzo di comunicazione radio
 *  channel: definisce il channel nell’indirizzo di comunicazione radio
 */

bool turn_left = false;
bool turn_right = false;
bool debug = false;
bool standby = false;
bool manual = false
int command = 0;
int received_loop;
int max_acceptable_loop = 75;
int motor_power_plus = 0;
int motor_power_minus = 0;
int motor_power_difference = 0;
int set_delay = 1000;
int obstacle_distance = 15;
```

```
const uint64_t pipe = 0xABCDEF;     // unsigned long long
const uint8_t channel = 0x1;        // unsigned char

/*
 *  Creazione oggetti
 *
 *  sonar(trigger_pin, echo_pin, max_cm_distance)
 *  Sensore ad ultrasuoni utilizzato per le rilevazioni delle distanza
 *
 *  mt(speed_[m1, m2, m3, m4], dir_a_[m1, m2, m3, m4], dir_b_[m1, m2, m3, m4])
 *  Gestione dei quattro motori, velocità e verso, in un singolo oggetto
 *
 *  radio(ce, csn)
 *  Modulo radio per la comunicazione con il semaforo
 */

NewPing sonar(7, 8, 200);
Motor mt(5, 3, 9, 6, 22, 26, 30, 34, 24, 28, 32, 36);
RF24 radio(49, 53);

/*
 *  Funzione secureDigitalRead(int pin)
 *
 *  Questa funzione è stata creata per avere più sicurezza sui dati letti
 *  da parte dei sensori ad infrarossi. Essi a volte risultano imprecisi e
 *  quindi è stato inserito un controllo.
 */

bool secureDigitalRead(int pin) {
  bool register tmp1, tmp2, tmp3;
  tmp1 = digitalRead(pin);
  delay(1);
  tmp2 = digitalRead(pin);
  delay(1);
  tmp3 = digitalRead(pin);
  return (tmp1&&tmp2) || (tmp2&&tmp3) || (tmp1&&tmp3);
}

/*
 *  Funzione obstacleDetection()
 *
 *  La funzione permette alla macchina di determinare se è presente un
 *  ostacolo entro una quantità di centimetri specificata in precedenza
 *  nella variabile obstacle_distance
 */

bool obstacleDetection() {
  int pingTime = sonar.ping_median(3);
  int distance = sonar.convert_cm(pingTime);
```

```
  if (distance <= obstacle_distance) {
    if (debug) {
      Serial.print(" Ostacolo rilevato entro ");
      Serial.print(distance);
      Serial.println(" cm");
    }
    return true;
  }
  else return false;
}

/*  Funzione light_status()
 *
 *  La funzione permette alla macchina di determinare lo stato del semaforo
 *  tramite la ricezione di dati via segnale radio. Una volta ricevuti i
 *  dati vengono elaborati, nel caso in cui il semaforo sia rosso oppure
 *  sia giallo verso la fine la macchina si ferma. Nel resto dei casi
 *  la macchina continua il tragitto
 */

bool light_status() {

  if (radio.available()) {
    char received_status[7] = "";
    radio.read(&received_status, sizeof(received_status));
    if (strncmp(received_status, "001",3) == 0) {
      if (debug) Serial.println(" Semaforo rosso - Macchina ferma");
      return false;
    } else if (strncmp(received_status, "010",3) == 0) {
      received_loop = received_status[4];
      if (received_loop >= max_acceptable_loop) {
        if (debug) {Serial.println(" Semaforo giallo - Macchina ferma loop "); Serial.println(received_loop);}
        return false;
      } else {
        return true;
      }
    } else {
      return true;
    }
  }

}

/* Setup */

void setup() {

  Serial.begin(9600);
  pinMode(DETECT_ED,INPUT);
```

```
  pinMode(DETECT_CD,INPUT);
  pinMode(DETECT_CS,INPUT);
  pinMode(DETECT_ES,INPUT);
  radio.begin();
  radio.setChannel(channel);
  radio.openReadingPipe(0, pipe);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();

}

/* Loop */

void loop() {

  if (!manual) {

    while (Serial.available() > 0){

      command = Serial.read();

      switch (command) {

        case '1':
          Serial.println(" Elenco comandi disponibili:");
          Serial.println("");
          Serial.println(" 1 - Visualizzazione comandi");
          Serial.println(" 2 - Attivazione debug");
          Serial.println(" 3 - Disattivazione debug");
          Serial.println(" 4 - Aumento potenza motori");
          Serial.println(" 5 - Riduzione potenza motori");
          Serial.println(" 6 - Attivazione controllo manuale");
          Serial.println(" 7 - Disattivazione controllo manuale");
          Serial.println(" 8 - Attivazione modalità standby");
          Serial.println(" 9 - Disattivazione modalità standby");
          Serial.println("");
          for (int i = 30; i >= 1; i--) {
            if (i%10 == 0) {
              Serial.print(" Disconnessione menù in ");
              Serial.print(i);
              Serial.println(" secondi");
              delay(1000);
            } else if (i == 1) {
              Serial.println(" Disconnessione riuscita");
              delay(1000);
            }
          }
          break;
```

```
case '2':
  Serial.println(" 2 | Attivazione debug");
  debug = true;
  delay(set_delay);
  break;

case '3':
  Serial.println(" 3 | Disattivazione debug");
  debug = false;
  delay(set_delay);
  break;

case '4':
  Serial.println(" 4 | Aumento potenza motori");
  Serial.println("");
  Serial.println(" Attualmente la potenza dei motori");
  Serial.print(" differisce dai valori originali per ");
  motor_power_plus += 5;
  motor_power_difference = motor_power_plus - motor_power_minus;
  Serial.println(motor_power_difference);
  delay(set_delay);
  break;

case '5':
  Serial.println(" 5 | Riduzione potenza motori");
  Serial.println("");
  Serial.println(" Attualmente la potenza dei motori");
  Serial.print(" differisce dai valori originali per ");
  motor_power_minus += 5;
  motor_power_difference = motor_power_plus - motor_power_minus;
  Serial.println(motor_power_difference);
  delay(set_delay);
  break;

case '6':
  Serial.println(" 6 | Attivazione controllo manuale");
  manual = true;
  delay(set_delay);
  break;

case '7':
  Serial.println(" 7 | Disattivazione controllo manuale");
  manual = false;
  delay(set_delay);
  break;

case '8':
  Serial.println(" 8 | Attivazione modalità standby");
  standby = true;
```

```
        delay(set_delay);
        break;

      case '9':
        Serial.println(" 9 | Disattivazione modalità standby");
        standby = false;
        delay(set_delay);
        break;

      default:
        Serial.println(" Comando non riconosciuto");
        Serial.println("");
        Serial.println(" Premere 1 per visualizzare i comandi");
        delay(set_delay);
        break;

    }

  }

}

if (standby) {

  mt.car_stop();
  Serial.println(" STANDBY ATTIVO");
  Serial.println("");
  Serial.println(" Premere 9 per disattivare");
  Serial.println("");
  delay(3000);

} else {

  if (manual) {

    while (Serial.available() > 0){

      command = Serial.read();

      switch (command) {

        case 'W':
          mt.car_forward(80 + motor_power_difference);
          //delay(300);
          break;

        case 'A':
          mt.car_left(125 + motor_power_difference);
          //delay(300);
```

```
        break;

      case 'S':
        mt.car_backward(80 + motor_power_difference);
        //delay(300);
        break;

      case 'D':
        mt.car_right(125 + motor_power_difference);
        //delay(300);
        break;

      case 'Q':
        manual = false;
        standby = true;
        command = 0;
        break;

      default:
        mt.car_stop();
        break;

     }

    }

  } else {

    if (!light_status()) {

      mt.car_stop();
      if (debug) {Serial.println(" ATTENZIONE: stop per il semaforo");}

    } else {

      if (obstacleDetection() && !turn_left && secureDigitalRead(DETECT_ED) && secureDigitalRead(DETECT_CD) &&
secureDigitalRead(DETECT_CS)) {
        mt.car_stop();
        if (debug) {Serial.println(" Valutazione ostacolo in corso. Attendere 5 secondi");}
        delay(5000);
        if (obstacleDetection()) {
          if (debug) {Serial.println(" Ostacolo fisso. Manovra di cambio corsia in corso"); delay(1000);}
          turn_left = true;
          if (debug) {Serial.println(" Manovra: c.c. retro D -> S");}
          mt.car_backward(80);
          delay(500);
          if (debug) {Serial.println(" Manovra: c.c. sinistra D -> S");}
          mt.car_left(220);
          delay(800);
        } else {
```

```
      if (debug) {Serial.println(" Ostacolo mobile. Rimanere nella stessa corsia"); delay(1000);}
     }

    } else if (obstacleDetection() && !turn_right && secureDigitalRead(DETECT_ES) && secureDigitalRead(DETECT_CD)
&& secureDigitalRead(DETECT_CS)) {

     mt.car_stop();
     if (debug) {Serial.println(" Valutazione ostacolo in corso. Attendere 5 secondi");}
     delay(5000);
     if (obstacleDetection()) {
      if (debug) {Serial.println(" Ostacolo fisso. Manovra di cambio corsia in corso"); delay(1000);}
      turn_right = true;
      if (debug) {Serial.println(" Manovra: c.c. retro S -> D");}
      mt.car_backward(80);
      delay(500);
      if (debug) {Serial.println(" Manovra: c.c. destra S -> D");}
      mt.car_right(220);
      delay(800);
     } else {
      if (debug) {Serial.println(" Ostacolo mobile. Rimanere nella stessa corsia"); delay(1000);}
     }

    }

    if (turn_left) {
     if (secureDigitalRead(DETECT_CD)) {
      turn_left = false;
      while (secureDigitalRead(DETECT_CD) && !(secureDigitalRead(DETECT_CS))) {
       mt.car_right(140);
       if (debug) {Serial.println(" Manovra: c.c. allineamento");}
      }
     } else {
      mt.car_forward(75);
      if (debug) {Serial.println(" Manovra: c.c. avanti D -> S");}
     }
    } else if (turn_right) {
     if (secureDigitalRead(DETECT_CS)) {
      turn_right = false;
      while (secureDigitalRead(DETECT_CS) && !(secureDigitalRead(DETECT_CD))) {
       mt.car_left(140);
       if (debug) {Serial.println(" Manovra: c.c. allineamento");}
      }
     } else {
      mt.car_forward(75);
      if (debug) {Serial.println(" Manovra: c.c. avanti S -> D");}
     }
    } else {
     if (digitalRead(DETECT_CD) && digitalRead(DETECT_CS)) {
      if (debug) {Serial.println(" Manovra: avanti");}
      mt.car_forward(70 + motor_power_difference);
```

```
        } else if (!(digitalRead(DETECT_CD)) && digitalRead(DETECT_CS)) {
          if (debug) {Serial.println(" Manovra: sinistra");}
          mt.car_left(110 + motor_power_difference);
        } else if (digitalRead(DETECT_CD) && !(digitalRead(DETECT_CS))) {
          if (debug) {Serial.println(" Manovra: destra");}
          mt.car_right(110 + motor_power_difference);
        } else if (!(digitalRead(DETECT_CD)) && !(digitalRead(DETECT_CS))) {
          if (debug) {Serial.println(" Manovra: stop");}
          mt.car_stop();
        }
       }
      }

     if (debug) {Serial.println("------------------------------------------------------------");}

    }

  }

}
```

```
/*
 *  Nome file: semaforo.ino
 *
 *  Questo programma permette al semaforo di valutare la presenza di una macchina all'interno di un determinato range
 *  stabilito dalla presenza di due sensori ad ultrasuoni. Quando la macchina supera il primo sensore viene attivato
 *  il canale radio e lo stato del semaforo viene comunicato alla macchina. Il programma è stato creato per permettere
 *  una distinzione del semaforo giallo tramite la variabile loopTime, la quale permette di valutare se il giallo è
 *  appena iniziato e quindi la macchina può proseguire oppure è quasi finito e la macchina deve fermarsi.
 *
 *  Informazioni dettagliate sul progetto al seguente link: https://github.com/AndreaMagni/SistemiEmbedded/
 *
 *  Copyright © 2018, Magni Andrea & Mercanti Davide, 23 Febbraio 2018
 *
 *  This program is free software: you can redistribute it and/or modify it under the terms of the GNU General
 *  Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option)
 *  any later version.
 *
 *  This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the
 *  implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
 *  License for more details.
 *
 *  You should have received a copy of the GNU General Public License with this program.
 *  If not, see <http://www.gnu.org/licenses/>.
 */

/* Inclusione librerie */

#include <NewPing.h>
#include <SPI.h>
#include <RF24.h>

/* Definizione etichette per i LED */

#define RED_LED 4
#define YELLOW_RED 5
#define GREEN_LED 6

/* Creazione oggetti */

RF24 radio(7, 8);
NewPing sonarA(46, 47, 500);
NewPing sonarB(49, 48, 500);

/* Dichiarazione e assegnazione variabili */
/* multiplier 50 corrisponde più o meno a 10 secondi */

bool debug = 1;
bool car_presence = false;
int loop_time;
int loop_time_multiplier = 50;
int max_distance = 25;
const uint64_t pipe = 0xABCDEF;
const uint8_t channel = 0x1;

/* Setup */

void setup() {

  Serial.begin(9600);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(YELLOW_RED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  radio.begin();
```

```
  radio.setChannel(channel);
  radio.openWritingPipe(pipe);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();

}

/* Loop */

void loop() {

  loop_time = 0;

  for (loop_time; loop_time <= loop_time_multiplier*1; loop_time++) {
    green_light();
    if (debug) {Serial.print("Loop time = "); Serial.println(loop_time);}
    if (car_presence) light_status();
  }

  for (loop_time; loop_time <= loop_time_multiplier*2; loop_time++) {
    yellow_light();
    if (debug) {Serial.print("Loop time = "); Serial.println(loop_time);}
    if (car_presence) light_status();
  }

  for (loop_time; loop_time <= loop_time_multiplier*3; loop_time++) {
    red_light();
    if (debug) {Serial.print("Loop time = "); Serial.println(loop_time);}
    if (car_presence) light_status();
  }

}

/* Funzione obstacle_detection_A */

bool obstacle_detection_A() {

  int ping_time = sonarA.ping_median(3);
  int distance = sonarA.convert_cm(ping_time);
  if (debug) {Serial.print("Distanza ostacolo A - "); Serial.println(distance);}
  if (distance <= max_distance) return true;
  else return false;

}

/* Funzione detected_A */

void detected_A() {

  if (debug) Serial.println("Trovato ostacolo A - Settaggio car a true");
  car_presence = true;

}

/* Funzione obstacle_detection_B */

bool obstacle_detection_B() {

  int ping_time = sonarB.ping_median(3);
  int distance = sonarB.convert_cm(ping_time);
  if (debug) {Serial.print("Distanza ostacolo B - "); Serial.println(distance);}
  if (distance <= max_distance) return true;
  else return false;
```

```
}

/* Funzione detected_B */

void detected_B() {

  if (debug) Serial.println("Trovato ostacolo B - Settaggio car a false");
  car_presence = false;

}

/* Funzione light_status */

void light_status() {

  int green_status = digitalRead(GREEN_LED);
  int yellow_status = digitalRead(YELLOW_RED);
  int red_status = digitalRead(RED_LED);
  char actual_status[7];
  sprintf(actual_status, "%d%d%d,%c", green_status, yellow_status, red_status, (char)loop_time);
  radio.write(&actual_status, sizeof(actual_status));
  if (debug) {Serial.print("Dati inviati: "); Serial.println(actual_status); Serial.println("------------------------------");}

}

/* Funzione green_light */

void green_light() {

  digitalWrite(GREEN_LED, HIGH);
  digitalWrite(YELLOW_RED, LOW);
  digitalWrite(RED_LED, LOW);
  if (obstacle_detection_A()) detected_A();
  if (obstacle_detection_B()) detected_B();
  if (debug) {Serial.println("Stato semaforo attuale: VERDE"); Serial.print("Trasmissione: "); Serial.println(car_presence);}

}

/* Funzione yellow_light */

void yellow_light() {

  digitalWrite(GREEN_LED, LOW);
  digitalWrite(YELLOW_RED, HIGH);
  digitalWrite(RED_LED, LOW);
  if (obstacle_detection_A()) detected_A();
  if (obstacle_detection_B()) detected_B();
  if (debug) {Serial.println("Stato semaforo attuale: GIALLO"); Serial.print("Trasmissione: "); Serial.println(car_presence);}

}

/* Funzione red_light */

void red_light() {

  digitalWrite(GREEN_LED, LOW);
  digitalWrite(YELLOW_RED, LOW);
  digitalWrite(RED_LED, HIGH);
  if (obstacle_detection_A()) detected_A();
  if (obstacle_detection_B()) detected_B();
  if (debug) {Serial.println("Stato semaforo attuale: ROSSO"); Serial.print("Trasmissione: "); Serial.println(car_presence);}

}
```

```
/*
 *  Nome file: Motor.h
 *
 *  Libreria creata per gestire i motori
 *
 *  Informazioni dettagliate al seguente link: https://github.com/AndreaMagni/SistemiEmbedded/
 *
 *  Copyright © 2018, Magni Andrea & Mercanti Davide, 23 Febbraio 2018
 *
 *  This program is free software: you can redistribute it and/or modify it under the terms of the GNU General
 *  Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option)
 *  any later version.
 *
 *  This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the
 *  implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
 *  License for more details.
 *
 *  You should have received a copy of the GNU General Public License with this program.
 *  If not, see <http://www.gnu.org/licenses/>.
 */

#ifndef Motor_h
#define Motor_h

#include "Arduino.h"

class Motor {
  public:
    Motor(int PIN_SPEED_M1, int PIN_SPEED_M2, int PIN_SPEED_M3, int PIN_SPEED_M4, int PIN_DIR_A_M1, int PIN_DIR_A_M2, int
PIN_DIR_A_M3, int PIN_DIR_A_M4, int PIN_DIR_B_M1, int PIN_DIR_B_M2, int PIN_DIR_B_M3, int PIN_DIR_B_M4);
    void car_forward(int SPEED);
    void car_backward(int SPEED);
    void car_left(int SPEED);
    void car_right(int SPEED);
    void car_forward(int SPEED_1, int SPEED_2, int SPEED_3, int SPEED_4);
    void car_backward(int SPEED_1, int SPEED_2, int SPEED_3, int SPEED_4);
    void car_left(int SPEED_1, int SPEED_2, int SPEED_3, int SPEED_4);
    void car_right(int SPEED_1, int SPEED_2, int SPEED_3, int SPEED_4);
    void car_stop();
  private:
    int _PIN_SPEED_M1;
    int _PIN_SPEED_M2;
    int _PIN_SPEED_M3;
    int _PIN_SPEED_M4;
    int _PIN_DIR_A_M1;
    int _PIN_DIR_A_M2;
    int _PIN_DIR_A_M3;
    int _PIN_DIR_A_M4;
    int _PIN_DIR_B_M1;
    int _PIN_DIR_B_M2;
    int _PIN_DIR_B_M3;
    int _PIN_DIR_B_M4;
};

#endif
```

```
/*
 *  Nome file: Motor.cpp
 *
 *  Libreria creata per gestire i motori
 *
 *  Informazioni dettagliate al seguente link: https://github.com/AndreaMagni/SistemiEmbedded/
 *
 *  Copyright © 2018, Magni Andrea & Mercanti Davide, 23 Febbraio 2018
 *
 *  This program is free software: you can redistribute it and/or modify it under the terms of the GNU General
 *  Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option)
 *  any later version.
 *
 *  This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the
 *  implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
 *  License for more details.
 *
 *  You should have received a copy of the GNU General Public License with this program.
 *  If not, see <http://www.gnu.org/licenses/>.
 */

#include "Arduino.h"
#include "Motor.h"

Motor::Motor(int PIN_SPEED_M1, int PIN_SPEED_M2, int PIN_SPEED_M3, int PIN_SPEED_M4, int PIN_DIR_A_M1, int
PIN_DIR_A_M2, int PIN_DIR_A_M3, int PIN_DIR_A_M4, int PIN_DIR_B_M1, int PIN_DIR_B_M2, int PIN_DIR_B_M3, int
PIN_DIR_B_M4) {
  pinMode(PIN_SPEED_M1, OUTPUT);
  pinMode(PIN_DIR_A_M1, OUTPUT);
  pinMode(PIN_DIR_B_M1, OUTPUT);
  pinMode(PIN_SPEED_M2, OUTPUT);
  pinMode(PIN_DIR_A_M2, OUTPUT);
  pinMode(PIN_DIR_B_M2, OUTPUT);
  pinMode(PIN_SPEED_M3, OUTPUT);
  pinMode(PIN_DIR_A_M3, OUTPUT);
  pinMode(PIN_DIR_B_M3, OUTPUT);
  pinMode(PIN_SPEED_M4, OUTPUT);
  pinMode(PIN_DIR_A_M4, OUTPUT);
  pinMode(PIN_DIR_B_M4, OUTPUT);
  _PIN_SPEED_M1 = PIN_SPEED_M1;
  _PIN_DIR_A_M1 = PIN_DIR_A_M1;
  _PIN_DIR_B_M1 = PIN_DIR_B_M1;
  _PIN_SPEED_M2 = PIN_SPEED_M2;
  _PIN_DIR_A_M2 = PIN_DIR_A_M2;
  _PIN_DIR_B_M2 = PIN_DIR_B_M2;
  _PIN_SPEED_M3 = PIN_SPEED_M3;
  _PIN_DIR_A_M3 = PIN_DIR_A_M3;
  _PIN_DIR_B_M3 = PIN_DIR_B_M3;
  _PIN_SPEED_M4 = PIN_SPEED_M4;
  _PIN_DIR_A_M4 = PIN_DIR_A_M4;
  _PIN_DIR_B_M4 = PIN_DIR_B_M4;
}

void Motor::car_forward(int SPEED) {
  analogWrite(_PIN_SPEED_M1, SPEED);
  analogWrite(_PIN_SPEED_M2, SPEED);
  analogWrite(_PIN_SPEED_M3, SPEED);
  analogWrite(_PIN_SPEED_M4, SPEED);
  digitalWrite(_PIN_DIR_A_M1, LOW);
  digitalWrite(_PIN_DIR_B_M1, HIGH);
  digitalWrite(_PIN_DIR_A_M2, LOW);
  digitalWrite(_PIN_DIR_B_M2, HIGH);
  digitalWrite(_PIN_DIR_A_M3, LOW);
```

```
  digitalWrite(_PIN_DIR_B_M3, HIGH);
  digitalWrite(_PIN_DIR_A_M4, LOW);
  digitalWrite(_PIN_DIR_B_M4, HIGH);
}

void Motor::car_forward(int SPEED_1, int SPEED_2, int SPEED_3, int SPEED_4) {
  analogWrite(_PIN_SPEED_M1, SPEED_1);
  analogWrite(_PIN_SPEED_M2, SPEED_2);
  analogWrite(_PIN_SPEED_M3, SPEED_3);
  analogWrite(_PIN_SPEED_M4, SPEED_4);
  digitalWrite(_PIN_DIR_A_M1, LOW);
  digitalWrite(_PIN_DIR_B_M1, HIGH);
  digitalWrite(_PIN_DIR_A_M2, LOW);
  digitalWrite(_PIN_DIR_B_M2, HIGH);
  digitalWrite(_PIN_DIR_A_M3, LOW);
  digitalWrite(_PIN_DIR_B_M3, HIGH);
  digitalWrite(_PIN_DIR_A_M4, LOW);
  digitalWrite(_PIN_DIR_B_M4, HIGH);
}

void Motor::car_backward(int SPEED) {
  analogWrite(_PIN_SPEED_M1, SPEED);
  analogWrite(_PIN_SPEED_M2, SPEED);
  analogWrite(_PIN_SPEED_M3, SPEED);
  analogWrite(_PIN_SPEED_M4, SPEED);
  digitalWrite(_PIN_DIR_A_M1, HIGH);
  digitalWrite(_PIN_DIR_B_M1, LOW);
  digitalWrite(_PIN_DIR_A_M2, HIGH);
  digitalWrite(_PIN_DIR_B_M2, LOW);
  digitalWrite(_PIN_DIR_A_M3, HIGH);
  digitalWrite(_PIN_DIR_B_M3, LOW);
  digitalWrite(_PIN_DIR_A_M4, HIGH);
  digitalWrite(_PIN_DIR_B_M4, LOW);
}

void Motor::car_backward(int SPEED_1, int SPEED_2, int SPEED_3, int SPEED_4) {
  analogWrite(_PIN_SPEED_M1, SPEED_1);
  analogWrite(_PIN_SPEED_M2, SPEED_2);
  analogWrite(_PIN_SPEED_M3, SPEED_3);
  analogWrite(_PIN_SPEED_M4, SPEED_4);
  digitalWrite(_PIN_DIR_A_M1, HIGH);
  digitalWrite(_PIN_DIR_B_M1, LOW);
  digitalWrite(_PIN_DIR_A_M2, HIGH);
  digitalWrite(_PIN_DIR_B_M2, LOW);
  digitalWrite(_PIN_DIR_A_M3, HIGH);
  digitalWrite(_PIN_DIR_B_M3, LOW);
  digitalWrite(_PIN_DIR_A_M4, HIGH);
  digitalWrite(_PIN_DIR_B_M4, LOW);
}

void Motor::car_left(int SPEED) {
  analogWrite(_PIN_SPEED_M1, SPEED);
  analogWrite(_PIN_SPEED_M2, SPEED);
  analogWrite(_PIN_SPEED_M3, SPEED);
  analogWrite(_PIN_SPEED_M4, SPEED);
  digitalWrite(_PIN_DIR_A_M1, HIGH);
  digitalWrite(_PIN_DIR_B_M1, LOW);
  digitalWrite(_PIN_DIR_A_M2, HIGH);
  digitalWrite(_PIN_DIR_B_M2, LOW);
  digitalWrite(_PIN_DIR_A_M3, LOW);
  digitalWrite(_PIN_DIR_B_M3, HIGH);
  digitalWrite(_PIN_DIR_A_M4, LOW);
  digitalWrite(_PIN_DIR_B_M4, HIGH);
}
```

```
void Motor::car_left(int SPEED_1, int SPEED_2, int SPEED_3, int SPEED_4) {
  analogWrite(_PIN_SPEED_M1, SPEED_1);
  analogWrite(_PIN_SPEED_M2, SPEED_2);
  analogWrite(_PIN_SPEED_M3, SPEED_3);
  analogWrite(_PIN_SPEED_M4, SPEED_4);
  digitalWrite(_PIN_DIR_A_M1, HIGH);
  digitalWrite(_PIN_DIR_B_M1, LOW);
  digitalWrite(_PIN_DIR_A_M2, HIGH);
  digitalWrite(_PIN_DIR_B_M2, LOW);
  digitalWrite(_PIN_DIR_A_M3, LOW);
  digitalWrite(_PIN_DIR_B_M3, HIGH);
  digitalWrite(_PIN_DIR_A_M4, LOW);
  digitalWrite(_PIN_DIR_B_M4, HIGH);
}

void Motor::car_right(int SPEED) {
  analogWrite(_PIN_SPEED_M1, SPEED);
  analogWrite(_PIN_SPEED_M2, SPEED);
  analogWrite(_PIN_SPEED_M3, SPEED);
  analogWrite(_PIN_SPEED_M4, SPEED);
  digitalWrite(_PIN_DIR_A_M1, LOW);
  digitalWrite(_PIN_DIR_B_M1, HIGH);
  digitalWrite(_PIN_DIR_A_M2, LOW);
  digitalWrite(_PIN_DIR_B_M2, HIGH);
  digitalWrite(_PIN_DIR_A_M3, HIGH);
  digitalWrite(_PIN_DIR_B_M3, LOW);
  digitalWrite(_PIN_DIR_A_M4, HIGH);
  digitalWrite(_PIN_DIR_B_M4, LOW);
}

void Motor::car_right(int SPEED_1, int SPEED_2, int SPEED_3, int SPEED_4) {
  analogWrite(_PIN_SPEED_M1, SPEED_1);
  analogWrite(_PIN_SPEED_M2, SPEED_2);
  analogWrite(_PIN_SPEED_M3, SPEED_3);
  analogWrite(_PIN_SPEED_M4, SPEED_4);
  digitalWrite(_PIN_DIR_A_M1, LOW);
  digitalWrite(_PIN_DIR_B_M1, HIGH);
  digitalWrite(_PIN_DIR_A_M2, LOW);
  digitalWrite(_PIN_DIR_B_M2, HIGH);
  digitalWrite(_PIN_DIR_A_M3, HIGH);
  digitalWrite(_PIN_DIR_B_M3, LOW);
  digitalWrite(_PIN_DIR_A_M4, HIGH);
  digitalWrite(_PIN_DIR_B_M4, LOW);
}

void Motor::car_stop() {
  analogWrite(_PIN_SPEED_M1, 0);
  analogWrite(_PIN_SPEED_M2, 0);
  analogWrite(_PIN_SPEED_M3, 0);
  analogWrite(_PIN_SPEED_M4, 0);
  digitalWrite(_PIN_DIR_A_M1, LOW);
  digitalWrite(_PIN_DIR_B_M1, LOW);
  digitalWrite(_PIN_DIR_A_M2, LOW);
  digitalWrite(_PIN_DIR_B_M2, LOW);
  digitalWrite(_PIN_DIR_A_M3, LOW);
  digitalWrite(_PIN_DIR_B_M3, LOW);
  digitalWrite(_PIN_DIR_A_M4, LOW);
  digitalWrite(_PIN_DIR_B_M4, LOW);
}
```

Questo codice non è ancora stato inserito all'interno del codice di Arduino nel momento della stampa di questo documento, è in fase di testing e abbiamo preferito inserirlo perché è stato fatto del lavoro per realizzarlo. La sua funzione è quella di migliorare la gestione dell'input via Bluetooth per poter realizzare chiamate complesse rispetto a quelle semplici presenti attualmente.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_CMD_BUFFER 256
#define N_COMANDI 5

// Globale:
char buffer[MAX_CMD_BUFFER];
char * comandi[] = {"ava","sx","ind","dx","standby"};


// stop è il n. di lettere da confrontare
// Restituisce:
// 0  (attesa) fin quando il comando nel buffer (fino alla posizione stop) è parte di un c. conosciuto
// 1  (comando riconosciuto)
// -1 (comando inesistente)
int rec(int stop) {
	int compar;
	for(int i=0; i<N_COMANDI; i++) {
		if (strlen(comandi[i])<stop) continue;
		compar = strncmp(comandi[i],buffer,stop);
		if (compar==0)
			return strlen(comandi[i])==stop;
		else continue;
	}
	return -1;
}

void buff_init() {
	for (int i=0;i<MAX_CMD_BUFFER; i++) buffer[i]='\0';
}


// Restituisce l'indice del vettore dei comandi a cui si trova il comando attualmente presente nel buffer
int get_cmd_number() {
	int i;
	for(i=0; i<N_COMANDI; i++)
		if (strncmp(comandi[i],buffer,strlen(comandi[i]))==0) return i;
	return -1;
}

int get_param(){
	char param_txt[3];
	param_txt[0]=getchar();
	param_txt[1]=getchar();
	param_txt[2]=getchar();
	return atoi(param_txt);
}

int main() {
	int  buffer_pos, cmd_temp;
	while (1/*Serial.available() > 0*/){

		cmd_temp = -1;
		buffer_pos = 0;
		buff_init();

		do buffer[buffer_pos] = getchar()/*Serial.read()*/;
		while (!(cmd_temp=rec(buffer_pos++)) && buffer_pos<MAX_CMD_BUFFER) ;
```

```
                switch (cmd_temp) {
                        case 1:
                                printf("Eseguo: %s\n",comandi[get_cmd_number()]);
                                switch (get_cmd_number()) {
                                        case 0: ; // codice per "ava"
                                        printf("parametro: %i",get_param());
                                        break;
                                        case 1: ; // codice per "sx"
                                        break;
                                        // ET CETERA
                                }; break;
                        case -1: printf("Comando: %s non riconosciuto\n",buffer); break;
                        default: printf("\n\nFATAL ERROR\n\n");
                }
        }

        return 0;
}
```