ROVIS - ROver for VIdeo Streaming

Corso di Fisica dei Sistemi Complessi - Prof. Sandro Rambaldi

Alessandro Cordella
Natale Vadalà
alessandro.cordella@studio.unibo.it
natale.vadala@studio.unibo.it

Novembre 2018

Indice

1 Introduzione

L'obiettivo del presente lavoro è stato quello di costruire un robot capace di muoversi su ruote e trasmettere un canale di *streaming video over HTTP*. Il dispositivo ottenuto è controllabile da un'interfaccia web ed è possibile visionare ciò che è posto di fronte al robot tramite una webcam.
Nelle seguenti sezioni verranno descritte le fasi di progettazione e realizzazione, i componenti hardware e le tecniche software utilizzate, e verranno presentati eventuali sviluppi futuri.

2 Progettazione

La fase di progettazione è stata condotta attraverso l'analisa della specifica e lo studio di progetti open-source precedenti.

2.1 Specifica

Progettare e realizzare un robot, controllabile attraverso un'interfaccia Web, dotato di una webcam in modo da poter trasmettere lo streaming video sulla stessa interfaccia.
Inoltre, si richiede l'uso microcontrollore o di un single-board computer, di componenti low-cost e software open-source.

2.2 Analisi dei requisiti

Hardware Ciò che serve per realizzare ROVIS:

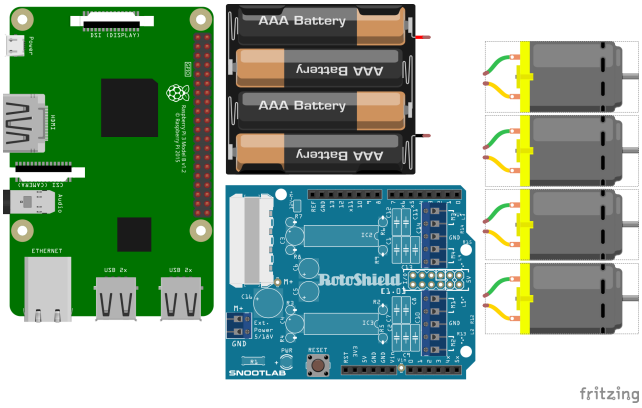
- Microcontrollore / Single-board computer
  Si è deciso di utilizzare un RaspberryPi 3 model B[1], che presenta nativamente una scheda di rete wireless, sebbene vada bene un qualsiasi Raspberry dotato di scheda di rete (interna o esterna) e porta USB per la webcam.

- Una "carrozzeria" con tutti gli alloggi necessari e 4 motori continui
  Per semplicità si è deciso di acquistare un *case*[2] che comprendesse già i motori fissati, evitando di incorrere in ulteriori future complicazioni con l'asseblaggio e messa in asse delle ruote, del moto, ecc...

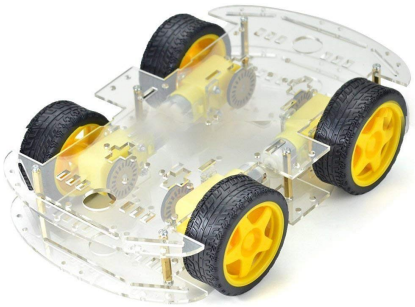
[1] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[2] https://www.diymore.cc/collections/robot-chassis/products/diymore-4-wheel-robot-chassis-smart-car-with-speed-and-tacho-encoder-for-arduino-raspberry-pi-robot-diy-kits-65x26mm-tire

- Una scheda che faccia da driver per i motori
  Acquistata su Amazon.it[3], una shield con dei chip L293D[4] (driver per motori molto comuni), permette di collegare i motori e gestirne la direzione (moto orario/antiorario) e la velocità, ottebibili grazie allo sfruttamento di alcuni pin GPio d'appoggio per il controllo singolo di ogni motore ai e ponti H per permettere i cambi di direzione. Va alimentato con 5V.

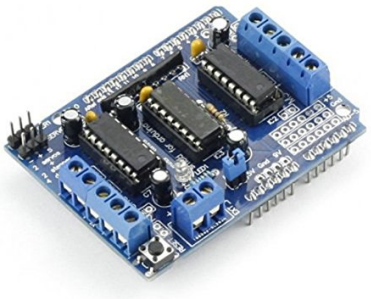
- Un Power-bank e un alloggiamentoper per shield e motori
  Si è deciso di utilizzare un power bank per il RaspberryPi per il semplice motivo che è più comodo, a nostro avviso, servire l'alimentazione da una porta micro USB invece che saldare un ulteriore alloggiamento per batterie sul RaspberryPi (Ma andrebbe comunque un'alimentazione uguale a quella per i motori e la shield, cioè 4* batterie AAA quindi 4.8-5V).



(a) RaspberryPi, case per batterie, 4 motori continui e L293D drive shield



(b) *Case* con motori e ruote



(c) L293D drive shield

[3] http://amzn.eu/d/a9PD82e
[4] https://www.robot-italy.com/it/l293d-motor-driver.html

Software Ciò che serve per realizzare ROVIS:

- Software per acquisizione video e streaming
  Essendo su un sistema operativo Unix, la scelta ricade su ffmpeg[5], che permette di codificare/decodificare gli stream video presi in input e restituirne i video in output (online come streaming o offline come file video).

- Piattaforma per orchestrazione dei server
  Per rendere omogeneo e semplice lo sviluppo si è deciso di utilizzare NodeJS[6], che fa al caso nostro (essendo una piattaforma event-driven) e per il semplice motivo che così backend e frontend sono stati scritti entrambi in JavaScript.

- Stream Server
  Lanciato da NodeJS, di default resta in ascolto sulla porta 8081, è il server che aspetta uno stream MPEG-TS[7] (streaming video) da parte di ffmpeg, verificandone il *secret* passato come parametro.

- Web Socket Server
  Lanciato da NodeJS, di default resta in ascolto sulla porta 8082, è il server che aspetta una richiesta websocket da parte dello Stream Server.

- HTTP Server
  Lanciato da NodeJS, di default resta in ascolto sulla porta 808, è il componente che serve l'interfaccia web per il controllo di ROVIS, e su cui viene proiettato lo streaming video. Il video è ottenuto dal WebSocket Server e riprodotto in un canvas HTML5.

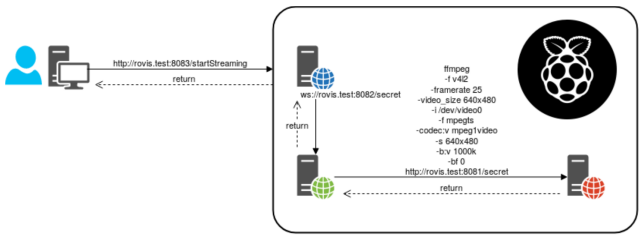
3 Realizzazione

3.1 Hardware

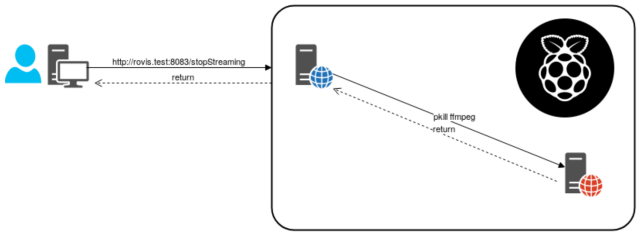
3.2 Software

Per garantire il controllo del dispositivo tramite un'interfaccia web è stato necessario progettare un server al quale un client può collegarsi. È stato utilizzato il framework JavaScript *NodeJs* per gestire sia la parte client che server dell'applicazione.
Inoltre si è dovuto ospitare del Python all'interno del progetto con la libreria

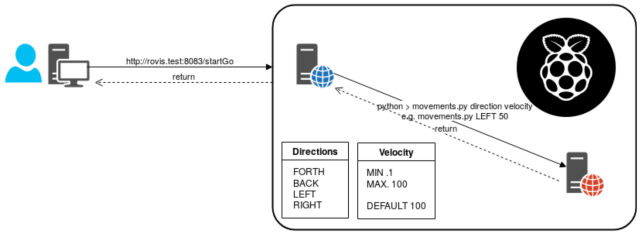
[5] https://www.ffmpeg.org/
[6] https://nodejs.org/it/
[7] https://en.wikipedia.org/wiki/MPEG_transport_stream



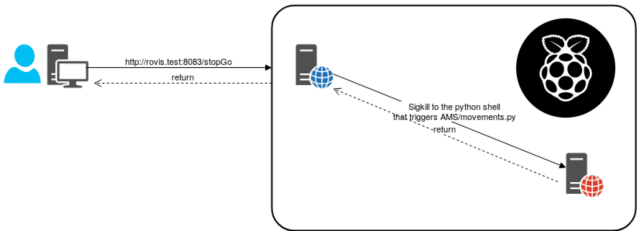
(a) Caso d'uso 1: ***stop streaming***



(b) Caso d'uso 2: ***stop streaming***



(c) Caso d'uso 3: ***start go***



(d) Caso d'uso 4: ***stop go***

AMSpy[8], il quale ci ha permesso di controllare la driver shield L293D (Attraverso la libreria Python RPi.GPIO per Raspberry) e ci ha fornito le primitive per inizializzare e settare la shield e interfacciarsi con i motori, cosa che sarebbe stata ben più complessa se avessimo dovuto reimplementare il tutto da zero in JavaScript. Nelle seguenti sezioni verrà descritta la struttura gerarchica delle cartelle e le funzionalità presenti nei file al loro interno.

Server side I file presenti all'interno della cartella ***server /*** contengono le funzioni per gestire la parte server del sistema, mentre all'interno delle sue sottocartelle sono presenti altri file che verranno descritti in seguito.

.env Questo file nascosto, processato da NodeJS, permette di definire le variabili d'ambiente per tutto il progetto. All'interno sono definite le porte di default per i server, il *secret* utilizzato per costruire le richieste, l'indirizzo IP e l'opzione di registrazione. Ovviamente per cambiare *secret*/porte/IP/opzioni basta modificare tale file e rilanciare il progetto.

```
STREAM_SECRET = "test"
STREAM_PORT = 8081
WEBSOCKET_PORT = 8082
HTML_PORT = 8083
RECORD_STREAM = false
ADDRESS = "0.0.0.0"
```

Listing 1: File .env

main.js Il file *main.js* è il file principale del server. Al suo interno sono presenti le funzioni di inizializzazione del sistema e per la gestione dello streaming video.
Qui vengono istanziati i 3 server discussi nel paragrafo 2.2/Software e, inoltre, è definito il percorso da cui servire i files per il frontend.

```
// Inizializzazione path per frontend
app.use("/public", express.static(__dirname + "/public"));

//Istanziazione Stream Server
var streamServer = http.createServer( function(request, response) {
...
})

//Istanziazione WebSocket Server
var socketServer = new WebSocket.Server({port: process.env.WEBSOCKET_PORT, perMessageDeflate: false});
....

//Istanziazione HTML Server
app.listen(process.env.HTML_PORT, () => {
  console.log('App listening on port '+process.env.HTML_PORT);
});
```

Listing 2: main.js

[8] https://github.com/lipoja/AMSpi

Per avviare il server è necessario digitare il seguente comando:

```
node main.js
```

Listing 3: Avvio server

Una volta lanciato il comando, sarà possibile accedere all'interfaccia web digitando l'indirizzo IP del Raspberry seguito dalla porta su cui gira l'HTML server (default 8083).

```
192.168.1.10:8083
```

Listing 4: Esempio indirizzo server

routes.js Questo file è quello che definisce le API, cioè gestisce la comunicazione client-server, associando ad ogni interazione del client sulla pagina web una specifica azione del server. Gestisce quindi le funzioni che regolano la webcam e il movimento. In particolare viene utilizzata la libreria di NodeJS *python-shell*[9] per eseguire uno script Python prodotto da noi (*movements.py*) per la gestione dei motori.

```
const routes = require('express').Router();
const { exec } = require('child_process');

routes.get('/', (req, res) => {
        res.status(200).json({ message: 'Connected!' });
   res.sendFile('public/index.html',{root: __dirname});
});
routes.get('/startGo', (req, res) => {
   res.status(200).json({ message: 'startGo!' });
   startGo(req.query.direction, req.query.vel);
});

routes.get('/stopGo', (req, res) => {
   res.status(200).json({ message: 'stopGo!' });
   stopGo("stopGo",req.query.vel);
});

routes.get('/startStream', (req, res) => {
   res.status(200).json({ message: 'streaming yes!' });
   streaming(true);
});
routes.get('/stopStream', (req, res) => {
   res.status(200).json({ message: 'streaming no!' });
   streaming(false);
});
routes.get('/getScreen', (req, res) => {
   res.status(200).json({ message: 'screenshot!' });
        screenshot();
});



var python_process;
function stopGo(){
   python_process.kill('SIGINT');
}
function startGo(direction,vel) {
   let options = {
         mode: 'text',
         pythonOptions: ['-u'], // get print results in real-time
         args: [direction,vel]
      };
       var PythonShell = require('python-shell');
```

[9] https://www.npmjs.com/package/python-shell

```
        var pyshell = new PythonShell('AMSpi/movements.py',options);
        ...
        python_process = pyshell.childProcess;
}
function streaming(bool) {
      console.log("ROUTES STREAMING -> start "+bool+" stop "+!bool)
      if(bool){
         url =
              'http://'+process.env.ADDRESS+":"+process.env.STREAM_PORT+"/"+process.env.STREAM_SECRET
         exec('ffmpeg -f v4l2 -framerate 25 -video_size 640x480 -i /dev/video0 -f mpegts -codec:v mpeg1video -s
              640x480 -b:v 1000k -bf 0 '+url,(error, stdout, stderr) => {
            ...
         });
      }
      else {
         exec('pkill ffmpeg', (error, stdout, stderr) => {
            ...
         });

      }
}
function screenshot(bool) {
   ...
}
```

Listing 5: routes.js

AMSpi/ *AMSpi* è la libreria Python utilizzata per controllare i motori.

AMSpi/movements.py Script scritto da noi che sfrutta la libreria. Inizialmente setta i pin GPIO d'appoggio per i ponti H e i 4 pin per controllare ogni motore. In seguito triggera le azioni di *start* e *stop* dei motori, in base alla direzione e alla velocità passata come input.

```
from AMSpi import AMSpi
import time
import sys

if __name__ == '__main__':
    # Calling AMSpi() we will use default pin numbering: BCM (use GPIO numbers)
    # if you want to use BOARD numbering do this: "with AMSpi(True) as amspi:"
    with AMSpi() as amspi:
      try:
      c=sys.argv[1]
      v=int(sys.argv[2])
      go_time=10
      # Set PINs for controlling shift register (GPIO numbering)
      amspi.set_74HC595_pins(21, 20, 16)
      # Set PINs for controlling all 4 motors (GPIO numbering)
      amspi.set_L293D_pins(5, 6, 13, 19)
      if c=="forth":
         amspi.run_dc_motors([amspi.DC_Motor_1, amspi.DC_Motor_2, amspi.DC_Motor_3,
               amspi.DC_Motor_4],speed=v)
      if c=="back":
         amspi.run_dc_motors([amspi.DC_Motor_1, amspi.DC_Motor_2, amspi.DC_Motor_3, amspi.DC_Motor_4],
               clockwise=False,speed=v)
      if c=="left":
         amspi.run_dc_motors([amspi.DC_Motor_1, amspi.DC_Motor_3], clockwise=False,speed=v)
         amspi.run_dc_motors([amspi.DC_Motor_2, amspi.DC_Motor_4],speed=v)
      if c=="right":
         amspi.run_dc_motors([amspi.DC_Motor_1, amspi.DC_Motor_3],speed=v)
         amspi.run_dc_motors([amspi.DC_Motor_2, amspi.DC_Motor_4], clockwise=False,speed=v)
      time.sleep(go_time)
   except KeyboardInterrupt:
      print("KeyboardInterrupt")
```

Listing 6: AMSpy/movements.py

Client side I file presenti all'interno della cartella ***public/*** sono quelli che gestiscono la parte client dell'applicazione.

index.html È il file contenente la struttura HTML del sito web. immagine

css/ All'interno di questa cartella sono presenti i file che regolano lo stile della pagina.

js/ In questa cartella sono contenuti i file che regolano le interazioni dell'utente con la pagina e comunicano al server le azioni intraprese da esso.

js/arrow.js Binding fra eventi sulla pagina web (click con il mouse/ click su tastiera) e azioni (richieste HTTP Get) dal browser dell'utente al server.

```
var first=0;
$(document).ready(function(){

   //Settaggio e deinizione listener per slider con velocita'
   var velocity=$('#velocity')[0].value
   $('#velocity').on("input change", function(){
      $('#velocityValue')[0].value=$(this)[0].value
      velocity=$(this)[0].value
   })

   //Listener sui bottoni della pagina web, al click del mouse
   $('button').on('mousedown', function(e) {
         switch($(this).attr("id")) {
            case "keyboard_key_left":
               $("#keyboard_key_left").css("background","green");
               //console.log("left hw");
               startGo("left", velocity);
               break;

            case "keyboard_key_up":
               ...

            case "keyboard_key_right":
               ...

            case "keyboard_key_down":
               ...

            case "start":
               startStreaming();
               break;

            case "stop":
               stopStreaming();
               break;

            case "screenshot":
               getScreenshot();
               break;

            default:
               return;
         }
         //e.preventDefault();
   })

   //Listener sui bottoni della pagina web, al rilascio del mouse
   $('button').on('mouseup', function(e) {
         switch($(this).attr("id")) {
            case "keyboard_key_left":
               $("#keyboard_key_left").css("background","white");
               stopGo();
               break;
            case "keyboard_key_up":
```

```
                ...
            case "keyboard_key_right":
                ...
            case "keyboard_key_down":
                ...
        }
        e.preventDefault();
    });

    //Listener sugli eventi della tastiera, al click di un tasto
    $(document).keydown(function(e) {
        switch(e.which) {
            case 37: // left
             $("#keyboard_key_left").css("background","green");
             //console.log("left");
             startGo("left", velocity);
             break;


            case 38: // up
           ...

            case 39: // right
           ...

            case 40: // down
                ...

        }
        e.preventDefault(); // prevent the default action (scroll / move caret)
    });

    //Listener sugli eventi della tastiera, al rilascio di un tasto
    $(document).keyup(function(e) {
        switch(e.which) {
            case 37: // left
             $("#keyboard_key_left").css("background","white");
             stopGo();
             break;

            case 38: // up
           ...

            case 39: // right
           ...

            case 40: // down
                ...

        }
        e.preventDefault(); // prevent the default action (scroll / move caret)
    });

});

}


function startGo(direction, vel) {
    if(first==0) {
        first++;

       $.get("/startGo", {direction, vel}, function(data, status){
           console.log("FRONTEND STARTGO -> direction "+direction + " vel "+vel);
       });

    }
}

//Chiamate alle API del server
function stopGo() {
    vel=0;
    $.get("/stopGo", {vel},function(data, status){
         first=0;
        console.log("FRONTEND STOPGO -> vel "+vel);
       });
}
function startStreaming() {
    $.get("/startStream", function(data, status){
        console.log("FRONTEND STARTSTREAM");
       });
}
function stopStreaming() {
    $.get("/stopStream", function(data, status){
```

```
        console.log("FRONTEND STOPSTREAM");
      });
}
function getScreenshot() {
   $.get("/getScreen", function(data, status){
        console.log("FRONTEND SCREENSHOT");
      });
}
```

Listing 7: js/arrow.js

js/canvas.js Fetching dell'elemento *video-canvas* sull'HTML, e dell'indirizzo del web socket server, per istanziare un oggetto di tipo JSMPEG Player (Il componente che riproduce il video su browser).

js/jsmpeg.min.js Libreria JavaScript importata staticamente[10], offre la possibilità di istanziare e utilizzare Video player, con relativi decoder (MPEG-TS, MPEG-1) e renderers (WebGL, Canvas2D).

4 Ulteriori sviluppi e Conclusioni

[10] https://github.com/phoboslab/jsmpeg