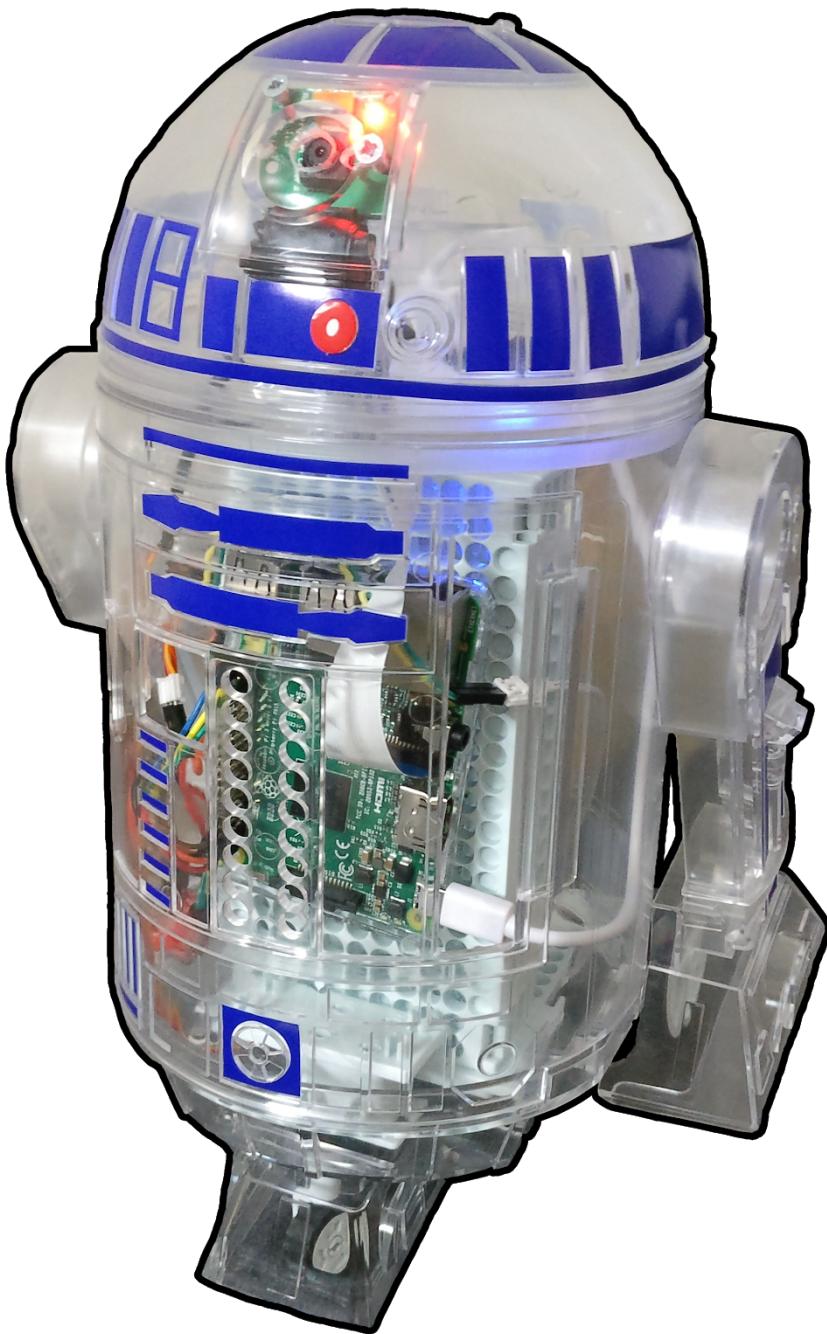


R2-D2 STAR WARS DROID



Studente: **Pindaro Margherita**
Classe: *5C Informatica*
Anno Scolastico: *2017/2018*
Oggetto: *Esame di Stato*
Scuola: *IIS E. Alessandrini*

Sommario

1. Introduzione	2
1.1 Motivazioni dietro la scelta	3
2. Sistema Hardware	4
2.1 Scelta del controllore	4
2.2 Struttura e Meccanica	5
2.3 Componenti	6
2.4 Schema elettrico	7
3. Architettura del Software	8
3.1 Descrizione dell'architettura	8
3.2 Raspberry Pi Network	9
3.3 Script Python Camera	9
3.3.1 Firme dei Metodi riguardanti la Camera	10
3.3.2 Firme dei Metodi riguardanti il Web Socket	11
3.4 Pagina web	12
3.4.1 Metodi Javascript	12
3.5 Pagina PHP	13
3.5.1 Shell_exec	14
3.6 Script Python per il movimento	15
3.6.1 Script che comandano la direzione	15
3.6.2 Script che controllano il motore DC	16
3.7 L'applicazione	18
3.8 Ottimizzazione	18
4. Design	19
4.1 Stile grafico	19
4.2 User Experience	20
5. Conclusioni	21
5.1 Funzionamento	21
5.2 Possibili improvement	21
6. Attachments	22
6.1 Log	22
6.2 Parts	25

1.Introduzione

R2-D2 Star Wars Droid, come dice il nome, è un robottino, in plastica dura, dall'apparenza simile al celebre robot del franchise Star Wars.

Tale droide è comandato tramite un'**applicazione**, fruibile mediante l'installazione su qualsiasi dispositivo Android oppure connettendosi via browser, ed è in grado di muoversi in tutte le direzioni.

L'applicazione, oltre ad essere un controller virtuale, permette di collegarsi alla vista del robot: esso è infatti dotato di una **camera**, dalla inclinazione e risoluzione regolabile, che **registra tutto ciò che vede**.

L'applicazione è connessa al robot tramite hotspot Wi-Fi, configurato sullo stesso. Quest'ultimo rende l'utilizzo del droide estremamente libero: è possibile usarlo in **qualsiasi luogo**, a patto che la superficie sia piatta e che la batteria sia carica.

Con tale premessa si presenta quindi questo giocattolo *homemade*, costruito appositamente con l'obiettivo di essere simpatico, stimolante e soprattutto semplice da utilizzare. Infatti la semplicità è il punto forte del robottino: esso è facilmente modificabile, espandibile e riparabile.

La realizzazione ha compreso tutti gli ambiti di insegnamento dell'ITIS Alessandrini. Dal punto di vista software, sono state utilizzate competenze acquisite dalla didattica triennale del programma di Sistemi e Reti, Informatica e Tecnologia e Progettazione di Sistemi Informatici, mentre dal punto di vista hardware, grazie a competenze in elettronica, è stato possibile creare il circuito da zero.

Nella documentazione verrà descritto il processo di costruzione, le scelte di design, e l'architettura e struttura del robot in sè, sia lato software che lato hardware, offrendo sia un'ampia panoramica che un'analisi nei minimi dettagli.

1.1 Motivazioni dietro la scelta

La scelta di fare R2-D2 come progetto di maturità deriva da una serie di ragioni.

Il mio obiettivo primario è stato quello di fare qualcosa che sublimasse i programmi didattici di Informatica, Sistemi e TPS di questi tre anni con l'elettronica, campo che, personalmente, trovo molto interessante e divertente. A quest'ultima mi sono avvicinata tramite uno stage presso il Politecnico di Milano, effettuato la scorsa estate, nell'ambito dell'alternanza scuola lavoro, che mi ha permesso di acquisire dimestichezza con le basi di elettronica e robotica.

Inoltre ho voluto unirmi all'onda del recente trend tecnologico *IoT, Internet of Things*, ovvero l'insieme di quelle tecnologie che consentono di trasformare un qualunque oggetto – sia esso un termostato, un orologio, un frigo o un'automobile – in un dispositivo connesso a Internet e per questo dotato di tutte le funzionalità e caratteristiche di device nati e progettati per essere utilizzati sul web. Questo concetto si è riflettuto nel concept del robot, un “drone di terra”, comandato via Internet da qualsiasi device.

Da un lato invece più prettamente ludico ed estetico ho voluto che il mio progetto avesse la mia firma, qualcosa che lo rendesse riconoscibile e a cui avrei potuto affezionarmi.

A tal proposito ho quindi deciso di inserire al suo interno la mia passione verso il fantasy e la fantascienza, ed è ciò che ha inoltre contraddistinto tutte le mie bozze di progetto. Come prima idea avevo quella di realizzare un *Dalek* di *Doctor Who*, in grado di sparare proiettili *Nerf Gun* (e anche accantonato per questo), come seconda quella di creare gli *Scacchi del Mago* di *Harry Potter*, che è però risultato essere un progetto troppo ambizioso, fino ad arrivare al simpatico *R2-D2*.

2. Sistema Hardware

2.1 Scelta del controllore

La parte hardware del droide è formata dalla camera e dai componenti necessari per il movimento.

Anche se a primo impatto può sembrare una parte banale, essa ha determinato la natura del progetto, a partire dalla scelta del “cervello” del robot. Con cervello si intende tale componente, programmabile, che, tramite comandi, può controllare tutto ciò a cui è collegato.

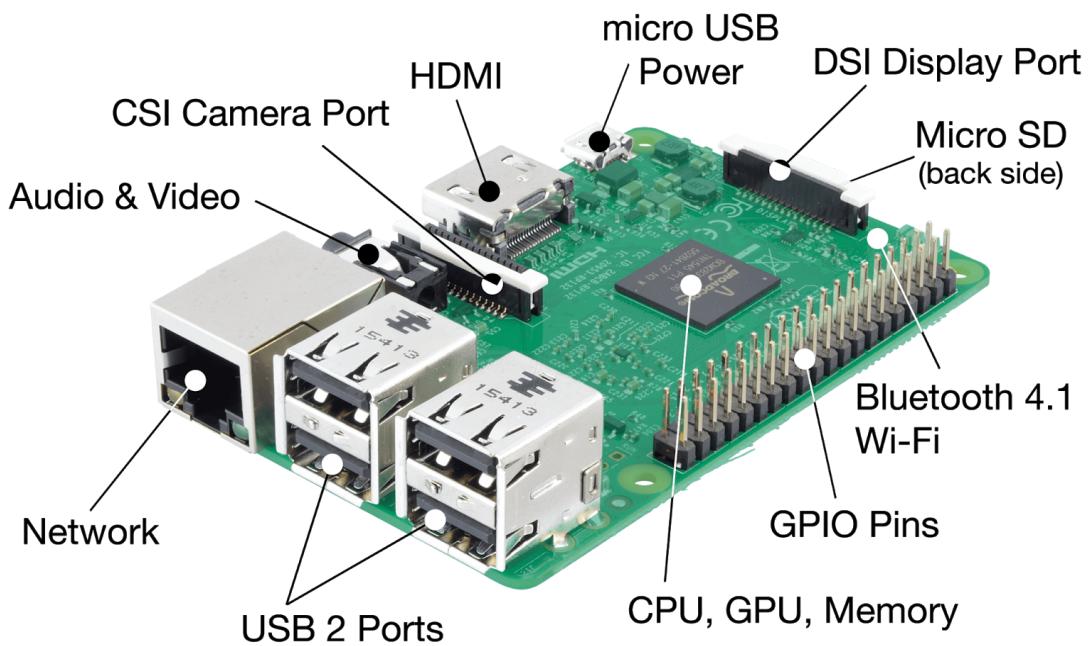
Questa decisione sarebbe potuta ricadere o sul microcontrollore Arduino oppure sulla board **Raspberry Pi**, ovvero un mini computer, ognuno coi propri vantaggi.

Se avessi guardato solamente al problema del movimento avrei indubbiamente scelto Arduino, più reattivo, con codice nativo e con maggiori risorse online. Tuttavia esso non avrebbe potuto svolgere il compito per cui il robot è pensato: la registrazione di un video, che invece, Raspberry Pi, grazie ai bus CSI per la camera incorporati, può gestire senza alcun problema.

Per questi motivi la scelta è ricaduta su Raspberry Pi, modello 3 B+ (in figura). Questa board, è un **computer** piccolo ed economico, realizzato su una singola scheda, della grandezza di una carta di credito, il cui sistema operativo e memoria risiedono in una microSD. È dotato di GPIO, ovvero pin destinati a input ed output utilizzati per controllare l'elettronica.

Specifiche del Raspberry Pi 3 B+	
CPU	1.2GHz 64-bit quad-core ARMv8
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Connettività	802.11n, Bluetooth 4.1, Ethernet 10/100
USB	4 Porte USB 2.0
Uscita Video	HDMI
Altre interfacce	GPIO 40 Pin, Jack audio 3.5mm, Camera Serial Interface (CSI), Display Serial Interface (DSI),

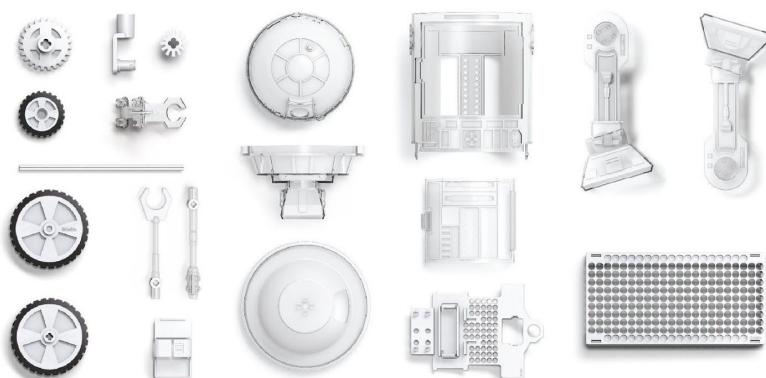
Foto di un Raspberry Pi 3 B+ con descrizione dei suoi componenti



2.2 Struttura e Meccanica

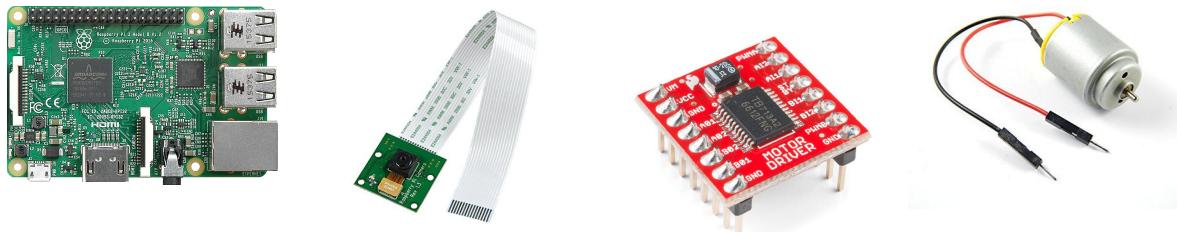
Per poter costruire un robot è necessario avere chiara consapevolezza di come sia strutturato. La presenza di diversi componenti in diverse quantità può cambiare fattori quali il numero di motori o l'energia richiesta per potersi muovere.

Inizialmente avevo previsto di stampare la struttura in 3D, tramite le stampanti presenti a scuola. Tuttavia stampare la scocca sarebbe stato ostico e controproducente. Al tempo stesso la scuola era interessata a provare l'elettronica per bambini *Little Bits*, che produce un kit con R2-D2. Per questo motivo l'istituto ha deciso di finanziare il progetto, acquistando il kit, di cui **ho solo tenuto la struttura** (il resto è andato al Laboratorio di Informatica 1).



Struttura del Robot

2.3 Componenti



Raspberry Pi 3 B+

Controllore che è stato già definito nel paragrafo 2.1

Camera CSI

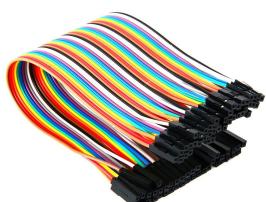
Camera con cavo CSI che, rispetto al cavo USB, si connette direttamente alla GPU del Raspberry, impattando pochissimo sulla CPU.

Tb6612fng driver

Modulo di controllo del motore ed è in grado di pilotare uno o due motori a corrente continua con una corrente costante di 1.2A (3.2A di picco).

DC Motor

Motore elettrico in corrente continua (DC motor, CC motor) è quel motore che, sostanzialmente, permette alle ruote di girare.



Servomotore

I servomotori sono attuatori speciali muniti di un sistema di feedback che permette di controllarne la posizione angolare. Non ruotano in modo continuo.

Cavi Breadboard

Cavi utilizzati per i collegamenti tra raspberry-driver, driver-motore DC, raspberry-servomotore

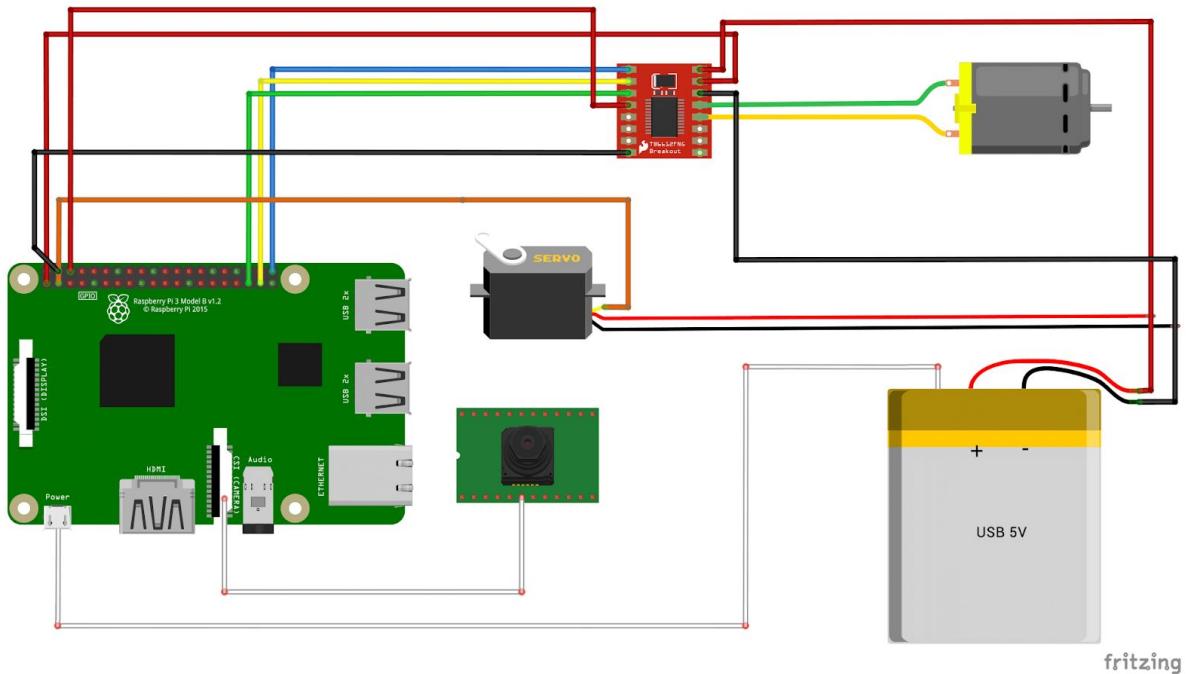
Power Bank

Batteria da 5V e 15000mAh, con due uscite usb. È stato scelto di usare questa al posto di una dispendiosa e scomoda batteria LIPO per la sua semplicità

Cavi USB

Cavi USB 2.0 usati per alimentare il Raspberry e i motori.

2.4 Schema elettrico



Schema elettrico, semplificato, del circuito del robot.

Lo schema riporta il cablaggio tra i diversi componenti, come i collegamenti tra i GPIO del Raspberry Pi al driver e al servomotore, il collegamento diretto con la camera, e l'intricato collegamento con la batteria.

Infatti, precedentemente, lo schema comprendeva due batterie distinte: la prima , un power bank, che alimentava il Raspberry, il quale a suo volta alimentava il servomotore, e una seconda, da 9V, che alimentava solamente il driver. Tuttavia, in seguito a un disguido, ho deciso di rendere più sicuro il circuito abbassando il voltaggio e alimentare separatamente i motori e il Raspberry Pi.

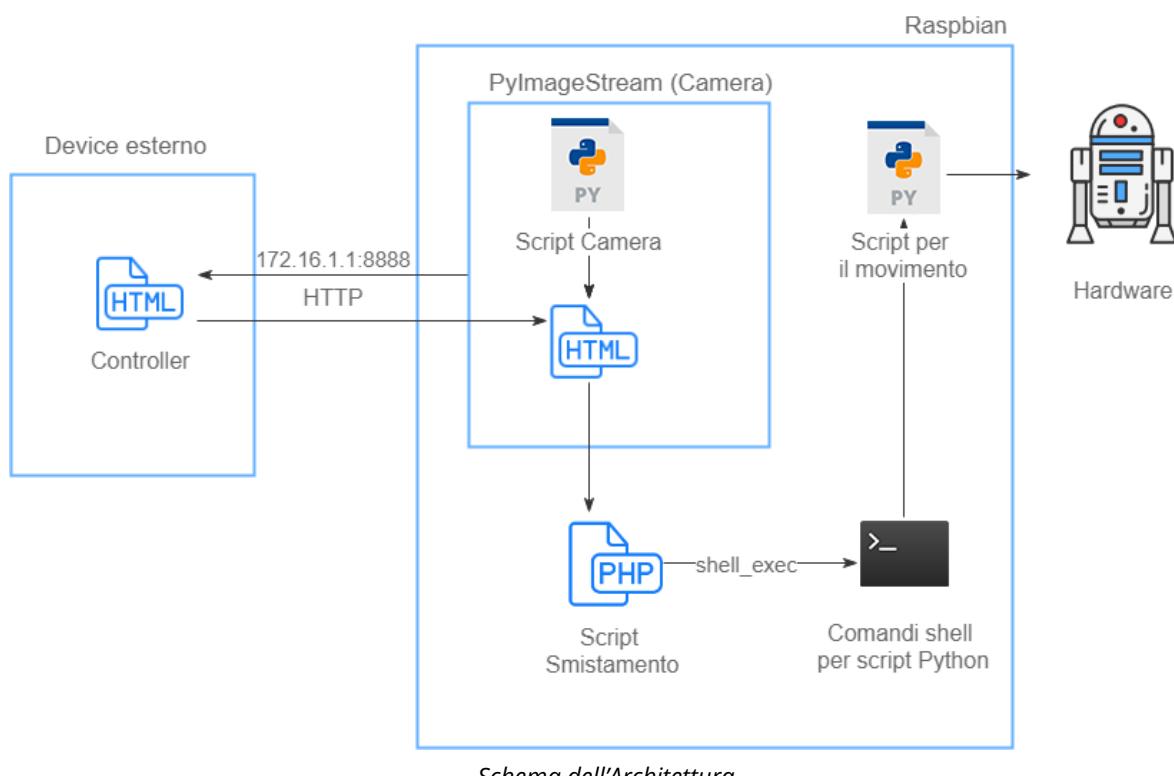
Così, dalle due porte USB del power bank escono due cavi separati, uno destinato alla sola alimentazione del Raspberry, l'altro, adeguatamente tagliato e saldato, si sdoppia e si collega direttamente con il servomotore e con il driver del motore DC.

3. Architettura del Software

3.1 Descrizione dell'architettura

Il software si appoggia sul Raspberry Pi grazie al Sistema Operativo *open source Raspbian*, basato sul SO Linux *Debian*. Il core del software è uno **script Python**, il quale avvia la camera ed occupa la porta 8888 dell'indirizzo del Raspberry stesso, dove verrà mostrata la **pagina web** che corrisponderà all'applicazione. In essa è presente il *display* del video della camera e i bottoni corrispondenti ai comandi, che, se premuti, realizzeranno una chiamata Ajax ad una **pagina PHP**, la quale provvederà a lanciare i **file shell**, che contengono i comandi che permettono di eseguire i diversi script Python che si occupano del movimento.

L'applicazione che visualizza tale pagina web è stata scritta con MIT App Inventor, e si collega con il Raspberry tramite Wi-Fi.



3.2 Raspberry Pi Network

Per collegare il robot al device che lo comanda ho deciso di utilizzare un **Hotspot Wi-Fi**, configurato sul Raspberry stesso, rendendolo così un **Access Point**.

Un Access Point non è altro che un dispositivo internet che mette a disposizione una rete Wi-Fi alla quale è possibile collegarsi per poter accedere alla rete locale ed a internet.

Questa via non è l'unica scelta che avevo a disposizione: avrei potuto utilizzare la tecnologia Bluetooth. Tuttavia sarebbe risultata troppo lenta per le prestazioni che il robot richiede, oltre che accorciare il range di utilizzo. Un'altra opzione avrebbe potuto essere l'utilizzo di un router esterno, che avrebbe sì alleggerito il carico di lavoro sul Raspberry Pi, ma ne avrebbe ridimensionato drasticamente la portabilità.

Per poter configurare il Raspberry come un Access Point ho utilizzato *dnsmasq* e *hostapd*, che consentono rispettivamente di utilizzare il Wi-Fi del Raspberry come punto di accesso e di installare un server DHCP e DNS.

Dunque ho configurato l'interfaccia wlan0, identificata dall'indirizzo ip 172.16.1.0 e dalla subnet mask 255.255.0.0.

In tal modo il droide crea un suo network, che, non essendo comunque connesso alla rete internet, non permetterà la navigazione.

L'IP, ovviamente statico, di R2-D2 è 172.16.1.1/16.

L'HTTP Server utilizzato è Apache 2.4.33, esso è necessario affinchè la pagina PHP possa funzionare.

3.3 Script Python Camera

Lo script della camera, lanciato in modo automatico all'accensione del Raspberry Pi, è il *core* di tutto il software, che d'ora in poi chiameremo *PyImageStream*, che è anche il nome della cartella e del progetto che contiene tutto il codice.

PyImageStream rende il Raspberry un Server che "streamma" nel formato MJPEG dalla WebCam CSI via **WebSocket**.

I vantaggi di questo sistema sono:

- Il supporto dell'alta risoluzione delle immagini (del video) e, nei casi in cui la connessione è lenta, l'adattamento automatico del frame rate per mantenerla.
- Il Client richiede al Server una nuova immagine solo quando quella precedente si è completamente caricata. In tal modo si evita di avere lag oppure immagini corrotte.

- Funziona su **tutti** i Browser, compresi quelli mobili, che supportano i WebSocket.
- È utilizzabile anche per camere USB, e non solo CSI.
- Spegne automaticamente la camera se non ci sono Client connessi, in modo tale da salvare energia, non stressare troppo la WebCam e ridurre l'uso della CPU.
- Non scrive su disco alcun file, preservando la memoria del Raspberry.

PyImageStream è stato scritto in Python, linguaggio perfetto per il Raspberry, e necessita dei prerequisiti, quali:

- Python 3, per poter scrivere il codice
- PIP, per installare i pacchetti Python
- **Tornado Web Server**
- Python Imaging Library, per gestire le immagini
- Pygame, usato per catturare le immagini dalla WebCam

Per poter lanciare il programma è necessario eseguire `main.py`, il quale può essere lanciato con diversi parametri:

```
comando: main.py [-h] [--port PORT] [--camera CAMERA] [--width WIDTH]
                  [--height HEIGHT] [--quality QUALITY] [--stopdelay STOPDELAY]
Lancia il Server PyImageStream.
Parametri opzionali:
-h, --help            mostra questo messaggio
--port PORT          Web server port (default: 8888)
--camera CAMERA      Lista delle camere, la prima camera è 0 (default: 0)
--width WIDTH         Width (default: 640)
--height HEIGHT       Height (default: 480)
--quality QUALITY    JPEG Qualità da 1 (peggiore) a 100 (migliore) (default: 70)
--stopdelay STOPDELAY Delay in secondi prima che la camera si fermi dopo che
                        tutti i client si sono disconnessi (default: 7)
```

3.3.1 Firme dei Metodi riguardanti la Camera

`__init__`: Costruttore della classe Camera, inizializza la camera.

```
def __init__(self, index, width, height, quality, stopdelay):
```

`self`: Oggetto *camera*, su cui viene applicato il metodo.

`index`: *Intero*, camera nell'indice che verrà utilizzata.

`width`: *Intero*, larghezza dell'immagine registrata.

`height`: *Intero*, altezza dell'immagine registrata.

`qualità`: *Intero*, qualità dell'immagine registrata.

stopdelay: *Intero*, secondi passati i quali la camera si ferma che la camera si fermi dopo che tutti i client si sono disconnessi.

request_start: Dopo un controllo, avvia la Camera

```
def request_start(self):
```

self: Oggetto camera su cui viene applicato il metodo

request_stop: Dopo un controllo, avvia la procedura di stop della Camera

```
def request_stop(self):
```

self: Oggetto camera su cui viene applicato il metodo

_start: Inizia la registrazione

```
def _start(self):
```

self: Oggetto camera su cui viene applicato il metodo

_stop: Dopo un controllo ferma la registrazione

```
def _stop(self):
```

self: Oggetto camera su cui viene applicato il metodo

get_jpeg_image_bytes: Ottiene l'immagine dalla Camera

```
def get_jpeg_image_bytes(self):
```

self: Oggetto camera su cui viene applicato il metodo

3.3.2 Firme dei Metodi riguardanti il Web Socket

check_origin: Controllo l'origine da cui arriva la richiesta

```
def check_origin(self, origin):
```

self: Oggetto WebSocket su cui viene applicato il metodo

origin: Stringa, header origin contenuto all'interno della richiesta http proveniente dal client

open: Apre la connessione con un client

```
def open(self):
```

self: Oggetto WebSocket su cui viene applicato il metodo

on_message: Comunica con il Client

```
def on_message(self, message):
```

self: Oggetto WebSocket su cui viene applicato il metodo

message: Stringa contenente il messaggio

on_close: Chiude la connessione con il Client e richiama request_stop

```
def on_close(self):  
    self: Oggetto WebSocket su cui viene applicato il metodo
```

3.4 Pagina web

PyImageStream, con *main.py*, apre quindi il server e genera un source. Tale source viene utilizzato da una pagina HTML, che sarà quella vista dal Client quando si connette all'indirizzo del Raspberry Pi e alla porta impostata (**172.16.1.1:8888**).

Se *main.py* è il core del back-end, la pagina, chiamata *index.html*, è il core del front-end. La pagina mostra la registrazione video e al tempo stesso rimane in ascolto dei comandi. Si compone di un tag ** centrale, grande, il quale source è quello generato da *main.py*, e di bottoni attorno (*<a>*) corrispondenti ai comandi.

```
<a id="stop" name="stop" onClick="stop()" style="color:#f72222"  
class="square_btn"> STOP</a>
```

Ogni bottone ha associata una funzione javascript, diversa da quelle degli altri bottoni, ma con in comune la stessa base. Infatti tutte le funzioni, che si attivano al click, fanno una chiamata Ajax a una pagina PHP comune, tramite URL, al quale viene aggiunto in GET il comando che andrà eseguito.

Come framework è stato utilizzato materializecss.com, di cui è stato tenuto solo parte del CSS, non minimizzato, riguardante i Grid Layouts, sui quali si basa la struttura responsive della pagina HTML

3.4.1 Metodi Javascript

```
var xhttp = new XMLHttpRequest();  
var pagina="http://172.16.1.1/esegui.php";  
  
/*************Funzione Stop*****/  
function stop() {  
    var url= pagina + "?comando=stop";  
    xhttp.onreadystatechange = function() {  
        var testo = xhttp.responseText;  
        if(testo=="ok"){  
            //do nothing  
        }  
    };  
  
    xhttp.open("GET", url, true);  
    xhttp.send();  
}
```

Per poter fare le chiamate alla pagine PHP ho deciso di utilizzare Ajax (Asynchronous Javascript), senza servirmi della libreria JQuery per evitare di appesantire ulteriormente l'applicazione. All'URL della pagina PHP viene aggiunto il comando, che in questo caso è *stop*, ma può essere anche *go*, *go back*, *left*, *center* e *right*.

3.5 Pagina PHP

```
<?php
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL | E_STRICT);

header('Access-Control-Allow-Origin: *');

$comando=$_GET['comando'];

if($comando == 'forward'){
    //Lancio L'sh del forward
    $out = shell_exec("sshpass -p raspberry ssh -o
StrictHostKeyChecking=no pi@raspberrypi sudo
/home/pi/Desktop/forward.sh 2>&1");

}else if($comando == 'reverse'){
    //Lancio L'sh di reverse
    $out = shell_exec("sshpass -p raspberry ssh -o
StrictHostKeyChecking=no pi@raspberrypi sudo
/home/pi/Desktop/reverse.sh 2>&1");

}else if($comando == 'stop'){
    //Lancio L'sh dello stop
    $out = shell_exec("sshpass -p raspberry ssh -o
StrictHostKeyChecking=no pi@raspberrypi sudo /home/pi/Desktop/stop.sh
2>&1");

}else if($comando == 'left'){
    $out = shell_exec("sshpass -p raspberry ssh -o
StrictHostKeyChecking=no pi@raspberrypi sudo python3
/home/pi/Desktop/left.py 2>&1");

}else if($comando == 'right'){
    $out = shell_exec("sshpass -p raspberry ssh -o
StrictHostKeyChecking=no pi@raspberrypi sudo python3
/home/pi/Desktop/right.py 2>&1");
```

```
else{
    $out = shell_exec("sshpass -p raspberry ssh -o
        StrictHostKeyChecking=no pi@raspberrypi sudo python3
        /home/pi/Desktop/center.py 2>&1");
}

echo $out;

?>
```

Il codice riportato sopra è la pagina PHP nella sua interezza.

Essa, dopo aver permesso l'accesso da qualsiasi tipo di Client, settato, per motivi di debug, la visualizzazione di eventuali errori, e appreso quale comando è richiesto, effettua un processo di smistamento tramite *if*. La pagina PHP sostanzialmente si occupa di usare il comando **shell_exec**, il quale, come da nome, esegue nella shell linux ciò che viene richiesto, come se si stesse scrivendo su terminale.

3.5.1 Shell_exec

Shell_exec è un comando PHP che, come detto precedentemente, esegue un comando via shell e restituisce l'output come stringa.

```
string shell_exec ( string $cmd )
```

Nella sua forma più base si presenta come sopra, tuttavia, a seconda di ciò che deve fare, potrebbe richiedere altri parametri, come nel nostro caso.

```
$out = shell_exec("sshpass -p raspberry ssh -o StrictHostKeyChecking=no
pi@raspberrypi sudo python3 /home/pi/Desktop/left.py 2>&1");
```

È possibile notare come, fuori da un esempio, la scrittura si complichi notevolmente e di come il comando base, ovvero `sudo python3 /home/pi/Desktop/left.py` si adorni di numerosi parametri.

- `sshpass -p` Serve per poter accedere alla shell tramite script senza che venga richiesta una password
- `raspberry` Nome dell'host
- `ssh -o` Specifica che l'accesso da terminale remoto deve essere sicuro
- `StrictHostKeyChecking=no` Abilita l'accesso anche se l'host remoto ha chiave sconosciuta
- `pi@raspberrypi` Specifica nome utente e password con cui lanciare il comando
- `2` Si riferisce al secondo file descriptor del processo
- `>` Specifica la direzione
- `&1` Specifica che il ridirezionamento deve essere nello stesso percorso del primo file descriptor

3.6 Script Python per il movimento

Vi sono due tipi di comandi: quelli che riguardano il movimento delle ruote e quelli che riguardano invece la direzione del robot. Essi necessitano di due diverse procedure.

3.6.1 Script che comandano la direzione

Gli script che gestiscono la direzioni sono i più **semplici**. Sono tre, uguali, la cui unica differenza è l'inclinazione (in gradi) del servomotore sul quale andranno ad agire: per il centro 79, per andare a sinistra 100 mentre per andare a destra 60.

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD)
GPIO.setup(3, GPIO.OUT)
pwm=GPIO.PWM(3, 50)
pwm.start(0)

def SetAngle(angle):
    duty = angle / 18 + 2
    GPIO.output(3, True)
    pwm.ChangeDutyCycle(duty)
    sleep(1)
    GPIO.output(3, False)
    pwm.ChangeDutyCycle(0)

SetAngle(60)
pwm.stop()
GPIO.cleanup()
```

Lo script agisce direttamente sui GPIO del Raspberry Pi, importando la loro libreria, e facendo un setup del pin che verrà utilizzato.

Il valore dell'angolo scelto (60 in questo caso) tuttavia non corrisponde a quello che andrà passato al servomotore per avere la giusta inclinazione, dato che vi sono fattori di elettrotecnica, come il duty cycle, che vanno tenuti in considerazione. A tal proposito ho realizzato il metodo `SetAngle(angle)` il quale, avendo un numero da 0 a 180 in ingresso, lo elabora tramite una formula e lo "invia" al servomotore.

È importante eseguire a fine programma una **clean** dei GPIO, con la quale vengono liberati i GPIO utilizzati, lasciandoli disponibili ad altri programmi che li utilizzeranno.

3.6.2 Script che controllano il motore DC

Gli script che invece gestiscono il motore DC, ovvero il componente che permette alle ruote di girare, risultano essere più complessi.

A discapito di quanto si possa pensare, la complessità non deriva dalla difficoltà dell'algoritmo in sé, bensì dal fatto che questi script, a differenza di quelli che controllano la direzione, non hanno una fine definita. Se negli altri ci si limitava a dire al servomotore l'angolo a cui andare, in questi il programma, potenzialmente infinito, termina solo quando il robot riceve un altro comando dello stesso tipo.

A tal scopo inizialmente avevo pensato di eliminare il processo del programma con il comando shell:

```
pkill -f forward.py
```

Effettivamente il processo veniva eliminato, e dal punto di vista software, il programma fermato. Tuttavia rimaneva “energia” ai GPIO interessati, i quali, da un altro programma simile, ne ricevevano altra, discordante, causando così malfunzionamenti.

Il trucco a cui ho pensato per ovviare a ciò è stato quello di utilizzare, oltre al comando shell, uno script fittizio (che è successivamente diventato quello per lo *stop*), il quale occupa i GPIO interessati e subito dopo effettua una clean (approfondita nel paragrafo 3.6.1).

Quindi la pagina PHP, quando richiama i comandi di *stop*, *forward* e *reverse*, non chiama direttamente i programmi python, bensì dei file .sh, che, quando lanciati, eseguono i comandi shell presenti in essi. A seguire *forward.sh*:

```
sudo python3 /home/pi/Desktop/stop.py
sudo pkill -f /home/pi/Desktop/reverse.py
sudo python3 /home/pi/Desktop/forward.py
```

In esso pulisco i GPIO coinvolti con il comando di stop, elimino il processo del programma “opposto” a quello che sta venendo eseguito, in questo caso *reverse*, e infine lancio lo script che effettivamente fa ciò che il comando richiede.

```
from gpiozero import PWMOutputDevice
from gpiozero import DigitalOutputDevice
from time import sleep

#GPIO CONSTANTS
PWM_DRIVE_ = 21 # ENA - H-Bridge enable pin
FORWARD_PIN = 26 # IN1 - Forward
REVERSE_PIN = 19 # IN2 - Reverse
```

```
drive = PWMOutputDevice(PWM_DRIVE_, True, 0, 1000)
forward = DigitalOutputDevice(FORWARD_PIN)
reverse = DigitalOutputDevice(REVERSE_PIN)

a=1

def allStop():
    forward.value = False
    reverse.value = False
    drive.value = 0

def forwardDrive():
    forward.value = True
    reverse.value = False
    drive.value = 1.0

def reverseDrive():
    forward.value = False
    reverse.value = True
    drive.value = 1.0

def main():
    allStop()
    while a>0:
        forwardDrive()
        sleep(5)
```

Come per gli script del servomotore, *forward.py* e *reverse.py* sono pressochè identici: l'unica differenza è nella funzione chiamata nel main. In esso, come detto prima, vi è loop infinito, forzato tramite una condizione che risulta essere sempre vera ($1 > 0$).

Il motore DC ha due entrate, le quali, se sono entrambe su *false* fanno sì che non si azioni, mentre se una delle due è *true* il motore gira, o in senso orario o antiorario. Su questo si basano le tre funzioni *allStop()*, *forwardDrive()*, *reverseDrive()*.

3.7 L'applicazione

Nell'era degli smartphone, sarebbe stato improponibile realizzare il controller virtuale del robot su browser. Per questo motivo ho scelto di realizzare un'app **nativa Android**.

Con app native ci si riferisce ad applicazioni mobile sviluppate interamente nel linguaggio del dispositivo sul quale vengono lanciate.

Nello specifico, *Droid Controller*, è stata sviluppato utilizzando MIT App Inventor, tramite l'utilizzo di un piccolo trucco: un **web viewer**. Quindi si tratta sì di una app nativa, ma sostanzialmente è una Web-App.

Il web viewer si connette all'indirizzo del Raspberry Pi alla porta 8888 (172.16.1.1:8888) e l'app è quindi pronta all'uso.

3.8 Ottimizzazione

Durante il testing si è presentato un grave problema: la ricezione dei comandi discussi nel paragrafo 3.6.2 aveva un ritardo di circa 5 secondi.

A tal proposito ho dovuto effettuare un lavoro di ottimizzazione che ha portato a diverse modifiche del codice.

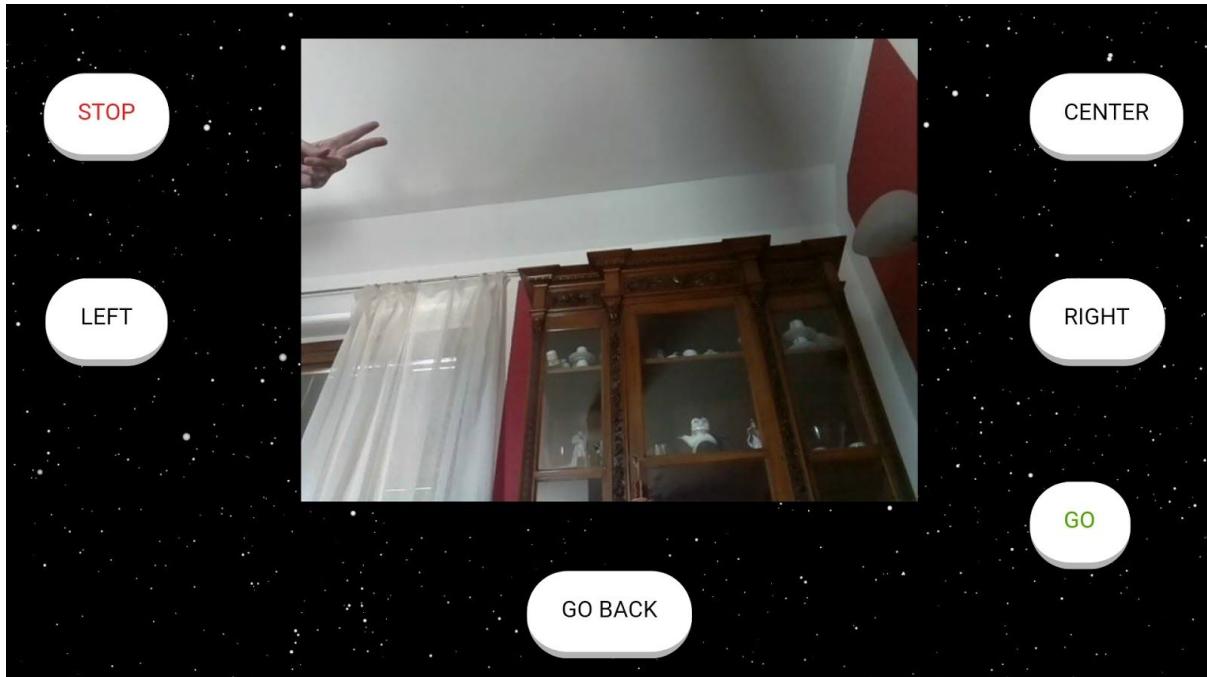
- La sessione Ajax in *index.html* non viene mai chiusa, per evitare che si perda tempo aprendone ogni volta una nuova
- L'utilizzo dei file .sh al posto dell'uso di comandi shell_exec singoli, i quali, ogni volta, devono effettuare un "login" al terminale
- Riduzione al minimo del Framework Materialize

In seguito a tali modifiche il ritardo si è abbassato a circa 2 secondi.

4. Design

4.1 Stile grafico

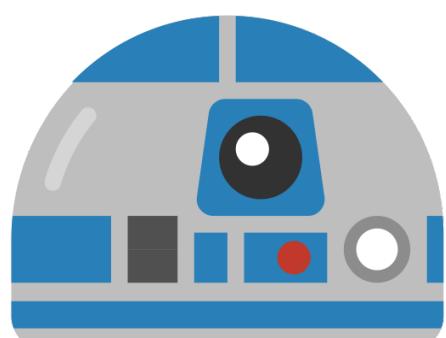
Screenshot dell'Applicazione



Siccome un prodotto non è solo progettazione, ma anche aspetto estetico, design, ho voluto dedicargli una sezione apposita. Questo lato, ovvero quello di essere accattivante e interessante, è alla base del progetto, e si ritrova già nella scelta di realizzare **R2-D2** in modo specifico, e non un generico robot in cartonato.

Avendo come soggetto un robot a tema *Star Wars*, la scelta di stile era quasi obbligata: qualcosa che richiamasse tale saga. Tuttavia, per evitare di apparire pacchiana e scontata, ho deciso comunque di dare all'applicazione dei tratti **minimali**. Per questi motivi lo sfondo è sì lo spazio, ma i pulsanti sono semplici rettangoli bianchi dagli angoli smussati.

Anche l'icona dell'applicazione si basa sullo stesso concetto: ispirazione *Star Wars* (in questo caso si tratta proprio di R2-D2) ma *minimal*.



Icona dell'Applicazione

4.2 User Experience

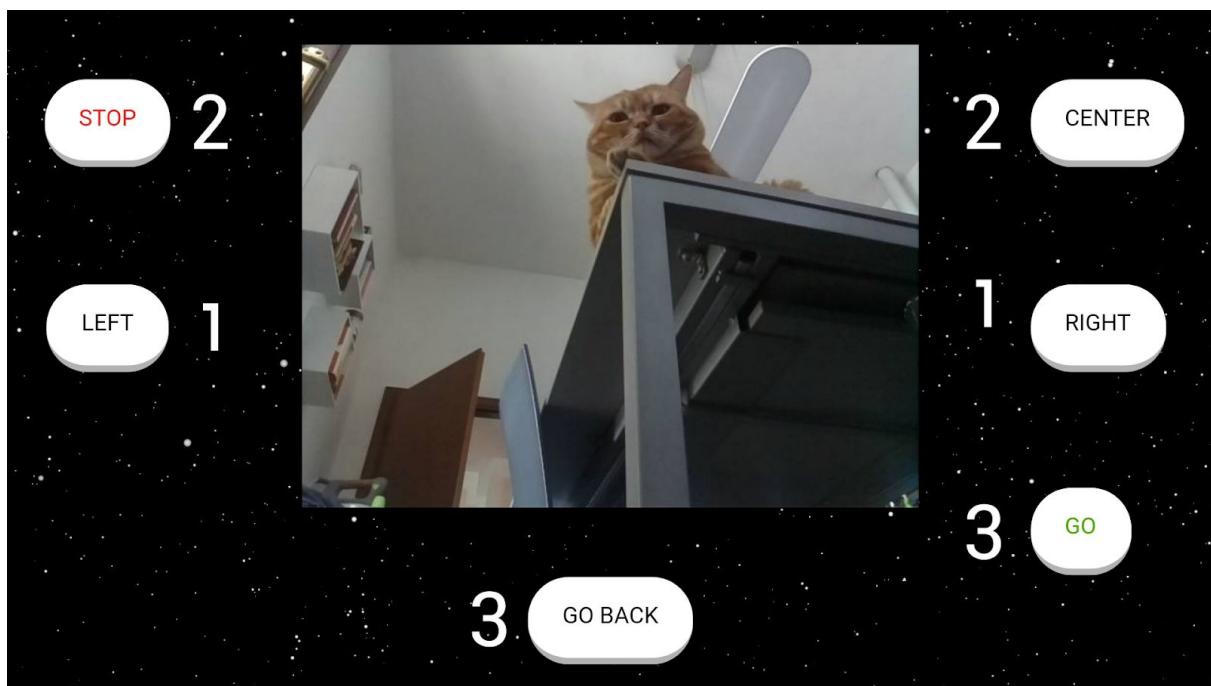
Altro lato non indifferente da tenere conto è l'*User Experience* o *UX*, specie se il front-end, come in questo caso, è un controller.

La User Experience è quindi una dimensione della progettazione che mette al centro le caratteristiche e i **bisogni** degli utenti, focalizzandosi sul loro contesto d'uso.

Si traduce in questo contesto come il bisogno della pagina web, e di conseguenza dell'applicazione, di essere **responsive** (si adatta a qualsiasi schermo), orientata orizzontalmente, e l'avere una posizione dei tasti ben studiata.

Infatti la loro disposizione non è affatto casuale: i tasti che verranno utilizzati di più, destra e sinistra (*left* e *right*) sono posizionati rispettivamente a destra e a sinistra, e in centro, dove sono di "default" i pollici dell'utilizzatore. Secondi a loro ci sono i pulsanti di *stop* e di centro (*center*), che dopo diversi test, sono risultati essere usati meno di *left* e *right*, ma di più rispetto ai comandi di accelerazione (*go*) e retromarcia (*go back*).

Inoltre, per evidenziare al meglio i tasti di "accensione" e "spegnimento" tra i diversi bottoni, ho deciso di colorare il testo rispettivamente di verde e di rosso.



Gerarchia dell'utilizzo

5. Conclusioni

5.1 Funzionamento

In questa ultima sezione illustrerò le “*istruzioni per l’uso*” di *R2-D2 Star Wars Droid*.

1. Prima cosa da fare è fornirgli alimentazione. Per fare ciò si solleva la testa e si inseriscono i due cavi USB nelle porte del power bank.
2. Fatto ciò bisogna aspettare circa 30 secondi affinchè il Raspberry Pi possa accendersi e la camera inizializzarsi.
3. Connettersi alla rete Wi-Fi *MPindaroRasp* da un qualsiasi dispositivo dotato di scheda Wi-Fi.
4. Aprire l’applicazione se la si ha precedentemente installata, nel caso contrario si può scaricarla da <https://we.tl/TmbBhgjte2> (chiedo perdono per il cartaceo) oppure ci si può connettere all’indirizzo 172.16.1.1:8888 tramite browser. Se si è da mobile è consigliabile attivare la rotazione automatica e porre il device orizzontalmente.
5. Dal controller è possibile comandare il robot premendo i pulsanti, con un solo tap, senza pressione continua.
6. Alla fine dell’utilizzo premere “Stop” e staccare i cavi USB precedentemente attaccati.

5.2 Possibili *improvement*

- Aggiunta dei suoni durante il movimento.
- Aggiunta di un bottone per lo spegnimento sicuro del Raspberry Pi.
- Implementazione della registrazione dell’audio.
- Realizzazione della modalità “*Guida Autonoma*”, tramite l’utilizzo di sensori infrared.
- Realizzazione della modalità “*Forza Motrice*”, tramite l’utilizzo di sensori di prossimità.
- Leila.

6. Attachments

6.1 Log

Da 18/04/2018 a 08/06/2018

Data	Descrizione	Note	Link Utili	Ore
18/04/18	Creazione sito e streaming raspberry	Motion non andava	https://www.techradar.com/how-to/computing/use-a-raspberry-pi-to-remotely-watch-your-home-1314466	3
19/04/18	Streaming raspberry	Abbandonato Motion in favore di IR_Webcam_Interface, capire come mandarlo a una propria pagina	https://elinux.org/RPi-Cam-Web-Interface	2
22/04/18	Streaming raspberry	Abbandonato PIR_Webcam_Interface in favore di Motion, ma con camera USB. Camera CSI si con script python. USB pronta all'uso CSI capire come mandare al src di 	https://picamera.readthedocs.io/en/release-1.13/recipes2.html#web-streaming	2,5
26/04/18	Streaming Raspberry, Motori	Provato nuovo Script, probabilmente quello definitivo. Cambiamento tipo di app: nativa realizzata tramite MIT App Inventor dove c'è un Web Viewer che visualizza la pagina restituita da iprasp:8888. Raspberry diventa un server. Provato i motori: servo nessun problema, DC mancante.	https://github.com/Bronkognor/PylImageStream	2
27/04/18	Streaming Raspberry, Motori	Raspberry si arresta in modo inaspettato, da risolvere. Finito la pagina base restituita dallo script. Testato il servo proprietario, funziona allo stesso modo. Test DC saltato: necessita del driver, gentilmente offerto poi da Gioiusa. Assegnato un IP statico al raspberry		2,5

03/04/18	Dc Motor, Design App	Test DC motor: nessun problema eccetto un ipotetico bisogno di una potenza maggiore. Revisione design app per le chiamate e visione del codice per lanciare script python da php. Vedere se si può fare lo stesso da js	2,5
04/05/18	Design App	Rifacimento Design: durato molto colpa di due apici del cazzo. Ricerca fallimentare di un code editor	1,5
10/05/18	Comunicazione App-Raspberry	Prova del comando shell_exec, senza risultato. Installamento di Apache e php5, scrittura del codice ajax sull'app e del file php	2
16/05/18	Comunicazione App-Raspberry, Autorun	Installato l'autorun del programma e configurato un hotspot all'interno del raspberry: non c'è più bisogno di un access point	http://www.stefanoteodorani.it/2017/05/20/hotspot-wifi-con-raspberry-pi-3/ 3
17/05/18	Comunicazione App-Raspberry	Risoluzione di errori temporanei. Risoluzione dell'errore del shell_exec.	2,5
27/05/18	Algoritmo Generale	Ritestato i motori: Dc ha dato problemi ma risolto. Problema programmi python: reverse.py e acc.py usano gli stessi pin e facendo pkill del processo i pin gpio rimangono attivi. Test vari	2,5
28/05/18	Interruzione programma Python	Provato diversi script o comando del terminale: nessuno funzionante	1
28/05/18	Interruzione programma Python	Illuminazione durante il sonno: pkillare il processo, lanciare un altro programma python che occupa i pin utilizzati e che poi fa una clean dei gpio. Da testare	0,1
30/05/18	Interruzione programma Python	Testata l'illuminazione del giorno precedente: funziona. Ribuildata l'applicazione App Inventor con aggiunta dell'icona, nome serio (Droid controller) e migliorie grafiche	0,5
31/05/18	Algoritmo Generale e Design App	Rivisto il concept dell'app, apportate ingenti migliorie grafiche. Riscritto la pagina php, dato inizialmente problemi con header e errori di sintassi. Una volta sistemati l'app risponde lentamente	2,5

31/05/18	Ottimizzazione Algoritmo	Per migliorare i tempi e risolvere il problema del ritardo (circa 4 secondi) codice shellexe non lancia più i singoli programmi python, i singole kill, ma file sh che contengono tali comandi. Ritardo dimezzato e ritenuto accettabile	1,5
01/06/18	Assemblaggio	Assemblato il robot	2
02/06/18	Assemblaggio	Sistemata la direzione del servo	0,4
04/06/18	Installazione Camera	Bucata la testa del robot e avvitata la camera con il Gianni: il cavo sembra instabile	2
06/06/18	Riparazione Camera	Il cavo si è rotto il giorno 5, sostituita la camera a scuola. Testato e si è bruciato il driver TB6612FNG e forse pin Raspberry	2
07/06/18	Riparazione	Rivisto il circuito generale: abbassato il voltaggio da 9V a 5V con l'utilizzo di una sola batteria, e alimentazione ai motori indipendente da quella del raspberry. Quindi smontato il robot, sistemato e rimontato. Cambiato il raspberry: GPIO Pin bruciati, bisogna vedere se tutti o solo alcuni. Fissato il raspberry alla struttura e rifinito il lavoro di montaggio.	4
07/06/18	Test e Photoshoot	Test del robottino per i corridoi della scuola, scattata qualche foto e filmato qualche video	1
08/06/18	Test	Test durante la giornata delle eccellenze	2

45

6.2 Parts

Descrizione	Link Amazon	Quantità	Prezzo unitario	Prezzo totale
Littlebits Robot Project Star, 680-0011-EU	http://amzn.eu/5HJFL7y	1	107,99 €	107,99 €
AZDelivery camera per Raspberry Pi	http://amzn.eu/qZ9JMT b	1	11,99 €	11,99 €
Cavi Breadboard	http://amzn.eu/7PT82Zi	1	7,99 €	7,99 €
Raspberry Pi 3 Model B+	http://amzn.eu/7A0HjuY	1	41,00 €	41,00 €
Power Bank 15.000 mAh	http://amzn.eu/4cFEVjv	1	13,99 €	13,99 €
Subtotale				182,96 €