UNIVERSITÀ DEGLI STUDI DI PALERMO

SCUOLA POLITECNICA

Corso di Laurea Ingegneria Cibernetica

TANK

Digital Manufacturing

Studenti:
Alessio Tumminello
Massimiliano Naimi

Professore:
Ernesto Lo Valvo

Sommario

Introduzione

Nella seguente relazione viene esposta la gestione di un Tank tramite controllo remoto via joypad collegato tramite Bluetooth.

Raspberry PI

È una single-board di prototipazione usata in diversi ambiti progettata dalla Raspberry Pi Foundation. La potenzialità di tale dispositivo risiede nel basso costo e nella quantità di dispositivi facilmente interfacciabili (sensori, attuatori, ecc..). I sistemi operativi supportati dal Raspberry sono molteplici, il più utilizzato è il Raspbian, una derivazione della distribuzione Debian di Linux.

In particolare la Raspberry Pi 3b+ ha le seguenti caratteristiche:

- SoC**:** Broadcom BCM2837B0 Quad Core Cortex-A53 a 1.4 GHz
- Gpu: Broadcom VideoCore IV Dual Core a 400 MHz
- Ram:1GB LPDDR2 900 MHz
- Wireless LAN dual-band a 2,4 GHz e 5 GHz
- Bluetooth 4.2 / BLE
- Ethernet e PoE
- Porta HDMI
- CSI Camera Connector, DSI Display Connector, GPIO header 40-pin

HALRUN

Il linguaggio HAl (**H**ardware **A**bstration **L**ayer), è un metodo di programmazione integrato in LinuxCNC ad alto livello, grazie al quale si riesce a controllare diversi sistemi collegando molteplici blocchi funzionali. È possibile utilizzare tale metodo di programmazione al fine di pilotare attuatori e dispositivi di input come controller in modo intuitivo ed immediato.
Una funzione molto utile integrata è l'*Halmeter*, tale strumento permette di visionare lo stato e la classificazione dei dispositvi collegati al raspberry.

Progettazione Hardware

1.1Componenti utilizzati

Di seguito saranno elencati tutti i componenti utilizzati per la realizzazione del Tank:

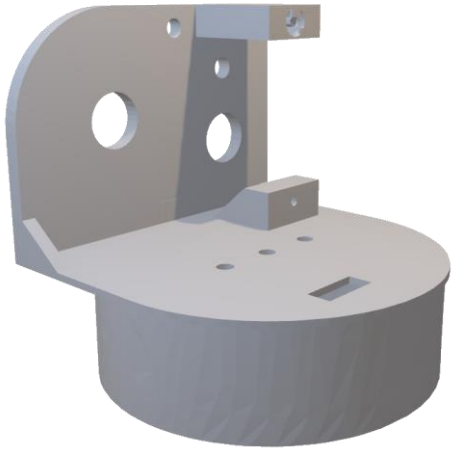
- Raspberry Pi 3 B+
- NanoDLP
- Ponte ad H
- Motori C.C. a 5V
- Servomotore sg90
- Raspberry Pi Camera
- Cannone gatling stampato in 3D
- Breadboard
- Cavi Jumper
- Base di legno
- Base in alluminio con cingoli
- Ingranaggi di trasmissione

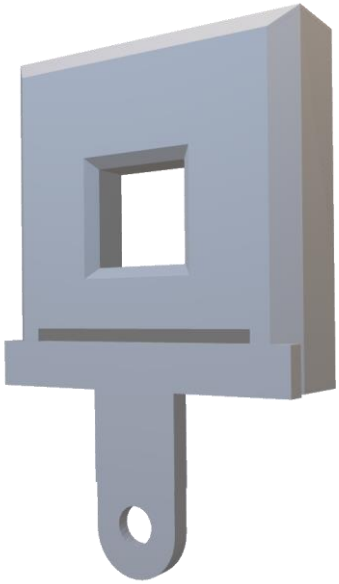
1.2 Struttura del tank

Per la realizzazione del progetto è stato sfruttato un telaio in alluminio con dei cingoli ed un rapporto di trasmissione ad ingranaggi, sulla base di quest'ultimi abbiamo modellato una base in legno su cui abbiamo posizionato il Raspberry Pi, il ponte ad H e un cannone gatling.
Il progetto relativo al cannone è stato scaricato da *Thingverse* in formato ***stl*** ed opportunamente modificato tramite *Tinkercad* per alloggiare su di esso la camera, inoltre la base è stata ulteriormente modificata per posizionare il servomotore e permettere la rotazione.
Sulla parte posteriore del Tank è stato posizionato il ponte ad H seguito dal Raspberry Pi su cui è stata installata la shield NanoDLP. Anteriormente è stato allocato il cannone gatling con la relativa base, e la Raspberry Pi Camera. Inferiormente è stata fissato il pacco batterie per l'alimentazione.
Di seguito i link con i file gcode dei vari pezzi disegnati e l'intero progetto di Thingverse:
https://drive.google.com/open?id=1lFGUznLMYtzXPeRaF1sJTVf3kbXHz6FK
https://drive.google.com/open?id=1QPz0jsupJjOWk5QEjmx2qa8ioqQIqz81
https://www.thingiverse.com/thing:3420083

File STL

Un formato STL rappresenta la geometria della superficie di un oggetto nelle tre dimensioni. Questo formato rappresenta la costruzione numerica costituita da un insieme di triangoli, ciascuno dei quali ha una posizione nota dei suoi tre vertici. La risoluzione di stampa 3D è strettamente legata alla quantità delle facce dei triangoli.





Di seguito la prima parte del gcode relativo al supporto della camera, da cui si evincono molti dei comandi già trattati a lezione:

G90

M82

M106 S255

M140 S65

M190 S65

M104 S225 T0

M109 S225 T0

G21 ;metric values

G90 ;absolute positioning

M82 ;set extruder to absolute mode

M107 ;start with the fan off

G28 X0 Y0 ;move X/Y to min endstops

G28 Z0 ;move Z to min endstops

M106 VENTOLA ON - S255 velocità ventola

M140 Imposta la temperatura del piano di stampa a 65

M190 Aspetta finché la temperatura del piano di stampa non raggiunge quella impostata cioe 65

M104 S65 imposta la temperatura dell'estrusore a 65C

M109Imposta la temperatura di estrusione ed attende

M82 Seleziona i codici assoluti dell'estrusore E (default)
M107 spegne ventola

G1 Z15.0 F9000 ;move the platform down 15mm

G92 E0 ;zero the extruded length

G1 F200 E5 ;extrude 5mm of feed stock

G92 E0 ;zero the extruded length again

M117 Printing

G92 E0

G1 E-3.0000 F6000

G1 Z0.300 F1002

; layer 1, Z = 0.200 *Risoluzione di stampa*

T0

; tool H0.200 W0.675 *Selezione offset di T0* **H0.200 risoluzione ugello**

; outer perimeter

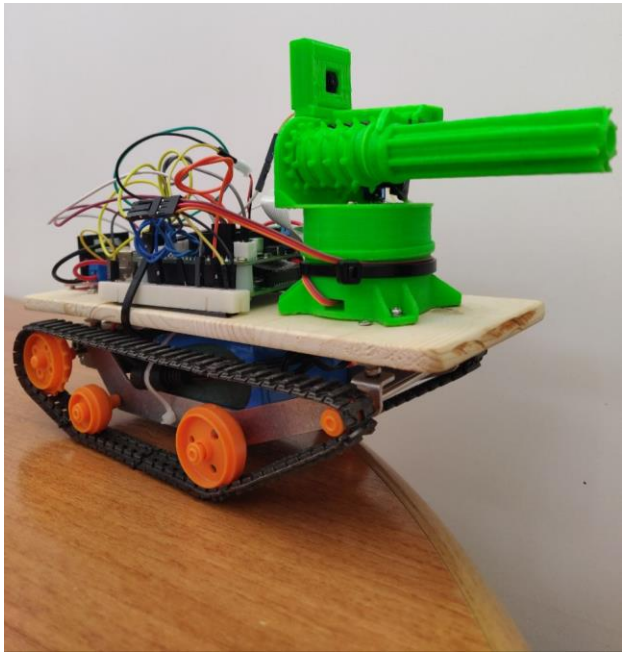
G1 X103.205 Y96.045 F4800

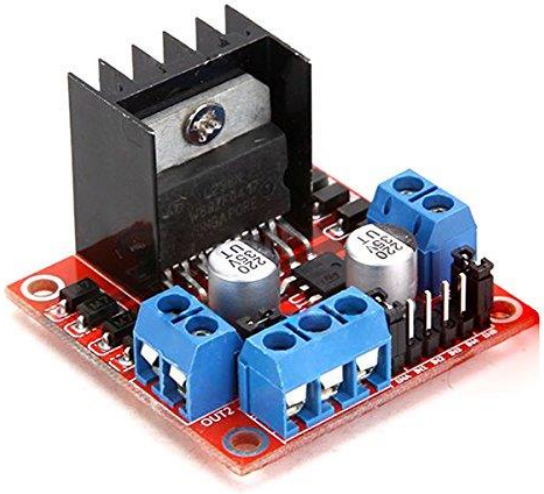
G1 E0.0000 F1800

G92 E0

G1 X106.751 Y96.045 E0.1990 F900

G1 X106.751 Y124.869 E1.8168





1.3 Motori C.C. e driver

I motori utilizzati per l'avanzamento e lo sterzo del tank sono dei motori in corrente continua

con tensione di funzionamento a 5v ed un assorbimento in corrente di 0.5A con una velocità di rotazione pari a 200 rpm. In figura sono presenti i motori installati e il relativo rapporto di trasmissione che è configurato al fine di rendere indipendente il movimento di un cingolo rispetto all'altro.

Questi motori sono pilotati tramite il *ponte ad H* alimentato a 5-12V esso permette di gestire i motori tramite due segnali pwm generati nell'ambiente HAL. Dal momento che l'uscita 12V della NanoDLP è stata impiegata per l'azionamento del cannone, l'unica altra porta disponibile a 12V che potesse alimentare il ponte ad H era quella relativa alla ventola di raffreddamento della NanoDLP, di conseguenza è stata utilizzata quest'ultima, mentre, la ventola è stata alimentata a 5V attraverso la breadboard su cui abbiamo distribuito GND e 5V.

Per quanto riguarda la rotazione del cannone, essendo necessaria una rotazione continua, è stato modificato un *servomotore sg90* in particolare rimuovendo l'elettronica di controllo, è stato possibile utilizzare il motore in c.c. al suo interno. La gestione di quest'ultimo è stata possibile tramite la porta GPIO 29, dato che essa ci permette di pilotare l'uscita a 12V della NanoDLP si è riusciti a controllare l'accensione e lo spegnimento di questo piccolo motore c.c. e quindi di azionare il cannone. Il relativo UV LED presente nella porta ci permette di visionare il corretto funzionamento di quest'ultima.

1.4 Servomotori

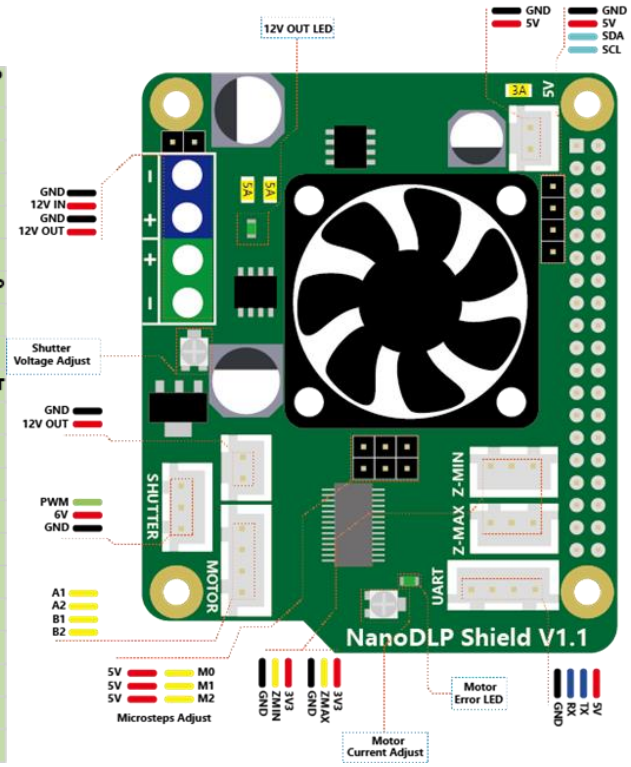
I servomotori sono dei motori c.c. controllati da una piccola scheda logica che ne limita il movimento da 0° a 180°, possono essere alimentati a 5v e comandati tramite un segnale pwm. È stata impostata la frequenza a 50 Hz mentre il duty-cycle di default è pari a 2.5% che viene incrementato o decrementato dalla funzione updown.

1.5 Alimentazione

In funzione del consumo energetico calcolato approssimativamente del tank, si è scelto di utilizzare un pacco batterie formato da 6 celle ***18650*** con una tensione nominale d'uscita pari a 12V e una corrente di 4400mAh. Il ponte ad H è stato alimentato da 5V a 12V, a 5V tramite la breadboard, mentre a 12V dal pin utilizzato per l'alimentazione della ventola di raffreddamento della NanoDLP.

1.6Collegamenti NanoDLP

NanoDLP Shield	BCM 编码	功能名	物理引脚 BOARD编码		功能名	BCM 编码	NanoDLP Shield
		3.3V	1	2	5V		
SDA	2	SDA.1	3	4	5V		
SCL	3	SCL.1	5	6	GND		
	4	GPIO.7	7	8	TXD	14	RXD
GND		GND	9	10	RXD	15	TXD
PWM	17	GPIO.0	11	12	GPIO.1	18	LIMIT TOP
	27	GPIO.2	13	14	GND		GND
EN	22	GPIO.3	15	16	GPIO.4	23	DIR
		3.3V	17	18	GPIO.5	24	LIMIT BOT
	10	MOSI	19	20	GND		GND
	9	MISO	21	22	GPIO.6	25	STEP
	11	SCLK	23	24	CE0	8	
GND		GND	25	26	CE1	7	
	0	SDA.0	27	28	SCL.0	1	
UV LED	5	GPIO.21	29	30	GND		GND
Fault	6	GPIO.22	31	32	GPIO.26	12	
	13	GPIO.23	33	34	GND		GND
	19	GPIO.24	35	36	GPIO.27	16	
	26	GPIO.25	37	38	GPIO.28	20	
GND		GND	39	40	GPIO.29	21	



Per i segnali di comando sono stati usati i seguenti pin Gpio del raspberry:

- Pin 3: ingresso SDA.1 relativo al segnale PWM 1 UP
- Pin 5: ingresso SCL.1 relativo al segnale PWM1 DOWN
- Pin 8: ingresso RXD relativo al PWM 0 UP
- Pin 10: ingresso TXD relativo PWM 0 DOWN
- Pin 18: Per il PWM 2
- Pin 29: UV LED e uscita 12V

Progettazione Software

2.1 *Halrun*

Dopo aver completato la parte hardware si è passati alla stesura del codice HAl.
Inizialmente sono stati caricati tutti i thread che ci servivano tramite comandi loadrt e loadusr, questi sono:

- thread
- halgpio
- toggle
- hal_input
- pwmgen
- updown
- conv_s32_float

Tramite il comando *addf* abbiamo collegato le varie funzione ai vari thread.
Con il comando *setp* sono stati impostati i parametri del pwm dell'updown.
Infine attraverso i *net* sono stati effettuati i collegamenti tra i vari segnali ingresso/uscita.

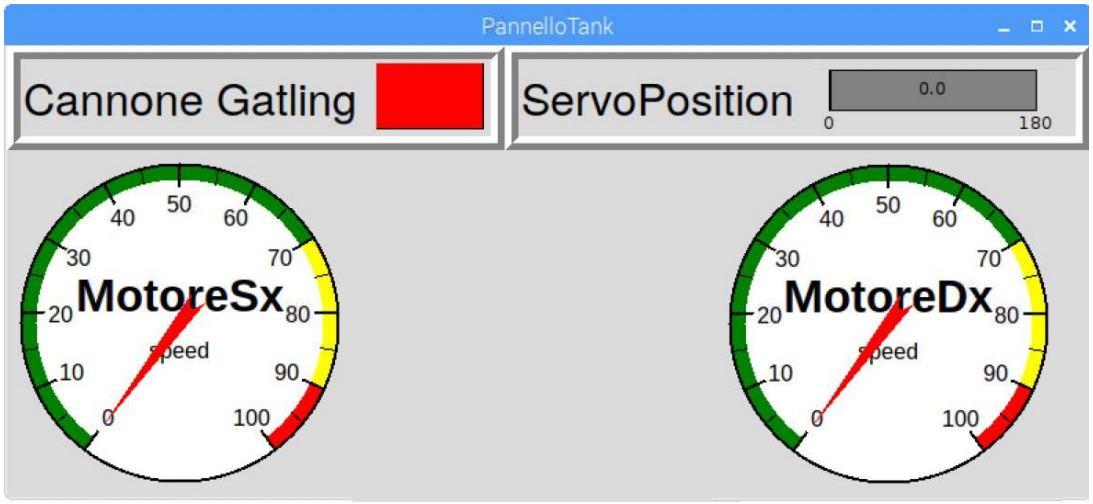
2.2 *Installazione Camera*

La Raspberry Pi Camera, collegata al CSI Camera Connector, è stata abilitata per la connessione tramite una serie di *shell command*:

- ***sudo raspi-config*** (permette di accedere alla schermata di configurazione avanzata abilitando la cam)
- ***ifconfig*** (consente di visualizzare l'indirizzo ip del raspberry che ci serivrà successivamente per la connessione in streaming)
- ***nano rpi_camera_surveillance_system.py*** (crea un file python dove verrà inserito il programma)
- ***python3 rpi_camera_surveillance_system.py*** (Avvia il programma sopracitato)

2.3 *XML*

All'interno della piattaforma LinuxCNC è presente un ***Virtual Control Panel***, tale piattaforma permette la visione grafica degli elementi input/output collegati, è possibile programmare tali panel tramite linguaggio ***xml***. Nel nostro caso è stato costruito tale pannello per la visione della velocità dei motori, la posizione del servomotore e lo stato di accensione del cannone.



2.4 *Connessione Joypad*

Per collegare il controller al raspberry è stato necessario modificare alcuni file di configurazione bluetooth, in particolare bisogna disabilitare la funzione ERTM (Enhanced Re-Transmission Mode) attiva di default, tale funzione impedisce il collegamento continuo del controller con il raspberry causandone la disconnessione. Di seguito l'elenco dei comandi utilizzati per la disattivazione di tale funzione ed il collegamento del controller:

1. ***sudo apt-get install xboxdrv*** (installazione dei driver di funzionamento)
2. ***sudo –I echo 'options bluetooth disable_ertm=Y' | sudo tee -a /etc/modprobe.d/bluetooth.conf*** (disattivazione dell'ERTM tramite permessi da SuperUser)
3. ***sudo bluetoothctl*** (avvia lo strumento di gestione bluetooth del raspberry)
4. ***Agent on*** (crea un'utenza attiva per la gestione)
5. ***Default agent*** (Imposta l'utenza agent come default)
6. ***scan on*** (Ricerca eventuali dispositivi visibili nell'area)
7. ***connect*** (Seguito dal MAC ADDRESS permette il collegamento)

Funzionamento del sistema

3.1 Avanzamento e sterzo del Tank

Per il sistema di comando si è utilizzato un controller wireless XBOX in grado di interfacciarsi con il Raspberry tramite Bluetooth permettendoci così il controllo da remoto.
L'avanzamento del veicolo avviene tramite il movimento contemporaneo delle due levette analogiche, mentre lo sterzo avviene sempre tramite quest'ultime ma essendo i due motori indipendenti l'uno dall'altro, azionando il motore sinistro e quindi controllando l'analogico sinistro è possibile sterzare a sinistra, viceversa a destra.

3.2 Funzionamento del cannone gatling

La struttura del cannone è stata progettata in modo che esso possa ruotare su se stesso di un angolo di 180° tramite un servomotore, tramite le frecce direzionali su/giù.
Un ulteriore servomotore, opportunamente modificato in un motore a c.c. che permette la rotazione continua, è stato impiegato per il sistema di sparo del cannone.
L'attivazione del cannone avviene tramite il tasto X del joypad.

3.3 Funzionamento streaming camera

La Raspberry Pi Camera, programmata in python tramite uno script, permette il collegamento in live-streaming della stessa purché ci si connetta sotto la stessa rete.
La visione del video in streaming è facilmente raggiungibile da qualsiasi dispositivo inserendo l'indirizzo ip locale del raspberry, il tutto attraverso la porta 8000.

Appendice

File .hal

loadrt threads name1=fast fp1=0 period1=100000 name2=slow period2=1000000

Per l'esecuzione delle funzioni sono stati caricati dei threads, veloce e lento, a cui collegare le varie funzioni utili al nostro scopo.

loadrt hal_gpio dir=5353483 exclude=61755380

NanoDLP Shield	BCM	Board	Nome	Exclude 1=si 0=no	Dir 1=out 0=in
	27	13	GPIO.2	1	0
	26	37	GPIO.25	1	0
STEP	25	22	GPIO.6	0	1
BOT	24	18	GPIO.5	0	0
DIR	23	16	GPIO.4	0	1
ENA	22	15	GPIO.3	0/1	1/0
	21	40	GPIO.29	1/0	0/1
	20	38	GPIO.28	1	0
	19	35	GPIO.24	1	0
TOP	18	12	GPIO.1	0	0
PWM	17	11	GPIO.0	0	1
	16	36	GPIO.27	1	0
TxD	15	10	RXD	0	1
RxD	14	8	TXD	0	1
	13	33	GPIO.23	1	0
	12	32	GPIO.26	1	0
	11	23	SCLK	1	0
	10	19	MOSI	1	0
	9	21	MISO	1	0
	8	24	CE0	1	0
	7	26	CE1	1	0
Fault	6	31	GPIO.22	0	0
UV LED	5	29	GPIO.21	0	1
	4	7	GPIO.7	1	0
SCL	3	5	SCL.1	0	1
SDA	2	3	SDA.1	0	1

*Dato che non necessitiamo di tutte le porte GPIO attraverso i parametri **dir** ed **exclude** si è utilizzato una maschera binaria in modo tale da selezionare solamente i pin utilizzati ed inoltre specificare se sono di Input o di Output.*
*Per il **dir** la maschera utilizzata è stata:*
10100011011000000001011
*Mentre per l'**exclude:***
10111100011101100110010
*Che convertiti in decimale risultano **dir=5353483 exclude=61755380***

loadusr -W hal_input -KRAL :0

*Con la funzione hal_input seguita dal KRAL identifichiamo il dispositivo di input che stiamo utilizzando in questo caso al joypad dell'xbox è riferito un **KRAL:0***

loadusr -Wn PannelloTank pyvcp -c PannelloTank PannelloTank.xml

Caricamento del pannello xml

loadrt pwmgen output_type=2,2,0

Sono stati utilizzati 2 PWM di tipo 2 per i motori in c.c., poiché con questo tipo siamo in grado di eseguire un up/down dei pin. Mentre per il servomotore è stato utilizzato un PWM di tipo 0, che accetta solo comandi positivi.

loadrt updown count=1

La funzione updown è stata utilizzata per impostare il range di funzionamento del servomotore.

loadrt conv_s32_float count=1

*Poiché il segnale in uscita dall'updown è un segnale di tipo **s32** mentre il duty cycle del pwm è di tipo **float** si è utilizzato questa funzione per convertire tale segnale.*

loadrt toggle count=1

La funzione toggle è stata utilizzata per l'attivazione/disattivazione del sistema di sparo del cannone

addf hal_gpio.read fast 1

addf hal_gpio.write fast -1

Dato che queste due funzioni si occupano di leggere/scrivere i valori del pwm sono state assegnate al thread fast.

addf pwmgen.update slow

Questa funzione richiede il floating point dunque deve essere collegata al thread slow

addf pwmgen.make-pulses fast

Genera gli impulsi quindi non richiede il floating point e quindi la colleghiamo al thread fast.

addf updown.0 slow

addf conv-s32-float.0 slow

Queste due funzioni richiedono il floating point dunque devono essere collegate al thread slow

addf toggle.0 fast

Si è collegata la funzione toggle al thread fast poiché non richiede il floating point

setp pwmgen.0.scale 1

setp pwmgen.0.pwm-freq 50

setp pwmgen.0.enable 1

setp pwmgen.1.scale 1

setp pwmgen.1.pwm-freq 50

setp pwmgen.1.enable 1

setp pwmgen.2.enable 1

setp pwmgen.2.pwm-freq 50

setp pwmgen.2.scale 100

*Si è impostato una frequenza di 50Hz per tutti e 3 i PWM utilizzati, abilitati tramite la pwmgen enable. Mentre per quanto riguarda lo **scale** ossia il range di variazione del*

segnale per i motori è stato impostato a 1 mentre per il servomotore è stato impostato pari a 100, per ottenere la massima velocità.

setp updown.0.min 0x00000003

setp updown.0.max 0x0000000C

Dato che il range di valori del duty cycle per il servomotore va da 2.5 a 15 per garantire un funzionamento ottimale si è settato un valore minimo pari a 3 (in s32=0x00000003) ed un valore massimo pari a 12 (in s32=0x0000000C).

setp updown.0.clamp true

Per far si che non venga superato questo range di valori di duty cycle si è impostato il clamp a true.

#MOTORE SINISTRO

net up1 pwmgen.0.up => hal_gpio.pin-08-out

net down1 pwmgen.0.down => hal_gpio.pin-10-out

net assex input.0.abs-y-position => pwmgen.0.value PannelloTank.mymeter

Attraverso il net si è collegato il fronte di salita del pwm al pin8, mentre il fronte di discesa al pin10. Il range di valori del pwm che ci ha permesso di pilotare il motore è regolato attraverso il segnale di input proveniente dall'analogico sinistro del joypad.

#MOTORE DESTRO

net up2 pwmgen.1.up => hal_gpio.pin-03-out

net down2 pwmgen.1.down => hal_gpio.pin-05-out

net assey input.0.abs-rz-position => pwmgen.1.value PannelloTank.mymeter1

Attraverso il net si è collegato il fronte di salita del pwm al pin3, mentre il fronte di discesa al pin5. Il range di valori del pwm che ci ha permesso di pilotare il motore è regolato attraverso il segnale di input proveniente dall'analogico destro del joypad.

#SERVO ROTAZIONE CANNONE

net ALFA1 <= input.0.abs-hat0y-is-neg => updown.0.countdown

net BETA1 <= input.0.abs-hat0y-is-pos => updown.0.countup

Tramite le freccette direzionali (su/giù) del joypad si è incrementato/decrementato il contatore della funzione updown, regolando così, l'orientamento del cannone.

net GAMMA1 updown.0.count => conv-s32-float.0.in

net DELTA1 conv-s32-float.0.out => pwmgen.2.value PannelloTank.my-bar

*Il valore corrente del contatore è stato prima convertito da **s32-> float** ed infine mandato come duty cycle al pwm 2.*

net YPS1 <= pwmgen.2.pwm => hal_gpio.pin-12-out

Infine il segnale PWM generato è stato indirizzato alla porta GPIO 12.

```
#MOTORE GATLING
net SPARO1 input.0.btn-x =>toggle.0.in
net SPARO2 toggle.0.out=>hal_gpio.pin-29-out PannelloTank.my-led
```

Si è collegato il motore in c.c. alla porta 29 pilotata attraverso il pulsante X del joypad per l'attivazione/disattivazione del motore tramite toggle.

```
start
```

File phyton

```
import io
import picamera
import logging
import socketserver
from threading import Condition
from http import server
```

Inclusione librerie

```
PAGE="""\
<html>
<head>
<title>TANK CAMERA</title>
</head>
<body>
<center><h1>SISTEMA DI PUNTAMENTO</h1></center>
<center><img src="stream.mjpg" width="640" height="480"></center>
</body>
</html>
"""
```

Pannello XML

```
class StreamingOutput(object):
    def __init__(self):
        self.frame = None
        self.buffer = io.BytesIO()
        self.condition = Condition()
    def write(self, buf):
        if buf.startswith(b'\xff\xd8'):
            # New frame, copy the existing buffer's content and notify all
            # clients it's available
            self.buffer.truncate()
            with self.condition:
                self.frame = self.buffer.getvalue()
                self.condition.notify_all()
            self.buffer.seek(0)
        return self.buffer.write(buf)
class StreamingHandler(server.BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
```

```
            self.send_response(301)
            self.send_header('Location', '/index.html')
            self.end_headers()
        elif self.path == '/index.html':
            content = PAGE.encode('utf-8')
            self.send_response(200)
            self.send_header('Content-Type', 'text/html')
            self.send_header('Content-Length', len(content))
            self.end_headers()
            self.wfile.write(content)
        elif self.path == '/stream.mjpg':
            self.send_response(200)
            self.send_header('Age', 0)
            self.send_header('Cache-Control', 'no-cache, private')
            self.send_header('Pragma', 'no-cache')
            self.send_header('Content-Type', 'multipart/x-mixed-replace; boundary=FRAME')
            self.end_headers()
            try:
                while True:
                    with output.condition:
                        output.condition.wait()
                        frame = output.frame
                    self.wfile.write(b'--FRAME\r\n')
                    self.send_header('Content-Type', 'image/jpeg')
                    self.send_header('Content-Length', len(frame))
                    self.end_headers()
                    self.wfile.write(frame)
                    self.wfile.write(b'\r\n')
            except Exception as e:
                logging.warning(
                    'Removed streaming client %s: %s',
                    self.client_address, str(e))
        else:
            self.send_error(404)
            self.end_headers()
class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):
    allow_reuse_address = True
    daemon_threads = True
with picamera.PiCamera(resolution='640x480', framerate=24) as camera:
    output = StreamingOutput()
    #Uncomment the next line to change your Pi's Camera rotation (in degrees)
    #camera.rotation = 90
    camera.start_recording(output, format='mjpeg')
    try:
        address = ('192.168.43.88', 8000)
            Indirizzo ip relativo al raspberry per la connessione in streaming
        server = StreamingServer(address, StreamingHandler)
        server.serve_forever()
    finally:
        camera.stop_recording()
```

File .XML

```
<pyvcp>
<vbox>
<hbox>
<hbox>
<label>
        <text>"Cannone-Gatling "</text>
        <font>("Helvetica", 24)</font>
</label>
        <relief>RIDGE</relief>
        <bd>10</bd>
<rectled>
        <halpin>"my-led"</halpin>
        <height>"50"</height>
        <width>"80"</width>
        <on_color>"green"</on_color>
        <off_color>"red"</off_color>
</rectled>
</hbox>
```

Led rettangolare che verifica il funzionamento del cannone

```
<hbox>
<label>
        <text>"ServoPosition" </text>
        <font>("Helvetica",24)</font>
</label>
        <relief>RIDGE</relief>
        <bd>10</bd>
<bar>
        <halpin>"my-bar"</halpin>
        <min_>3</min_>
        <max_>12</max_>
        <bgcolor>"gray"</bgcolor>
        <fillcolor>"yellow"</fillcolor>
        <range1>3,6, "green"</range1>
        <range2>6,12, "orange"</range2>
</bar>
</hbox>
</hbox>
</vbox>
```

Bar che mostra la posizione del servomotore

```
<hbox>
<hbox>
<meter>
        <halpin>"mymeter"</halpin>
        <text>"MotoreSx"</text>
        <subtext>"speed"</subtext>
        <size>250</size>
        <min_>0</min_>
        <max_>1</max_>
```

```
        <majorscale>10</majorscale>
        <minorscale>5</minorscale>
        <region1>(0.66,1, "red")</region1>
        <region2>(0.33,0.66, "yellow")</region2>
        <region3>(0.33, "green")</region3>
        <halpin>"mymeter"</halpin>
</meter>
</hbox>
```
Meter che raffigura in scala la velocità del motore sinistro
```
<hbox>
<meter>
        <halpin>"mymeter1"</halpin>
        <text>"MotoreDx"</text>
        <subtext>"speed"</subtext>
        <size>250</size>
        <min_>0</min_>
        <max_>1</max_>
        <majorscale>10</majorscale>
        <minorscale>5</minorscale>
        <region1>(0.66,1, "red")</region1>
        <region2>(0.33,0.66, "yellow")</region2>
        <region3>(0.33, "green")</region3>
        <halpin>"mymeter1"</halpin>
</meter>
</hbox>
</hbox>
```
Meter che raffigura in scala la velocità del motore destro
```
</pyvcp>
```