DOCUMENTAZIONE PY_ROBOT

Stefano Decina

stefano.decina@student.univaq.it

mat. 236180

Alessandro Mattei

alessandro.mattei1@student.univaq.it

mat. 235918

Mattia Caputo

mattia.caputo1@student.univaq.it

mat. 243467

Sara Rebecca Di Naccio

sararebecca.dinaccio@student.univaq.it

mat. 244306

Per maggiori informazioni è possibile consultare la Repository di Git Lab

https://gitlab.com/Alesmatte/Py_Robot.git

Ottobre 2018

SOMMARIO

CAPITOLO 1

INTRODUZIONE

Questo Progetto è l'esame finale del corso "Intelligent System and Robotics Laboratory" del Professore Giovanni De Gasperis.

Lo scopo di questo progetto è quello di aggiornare e aggiungere nuove caratteristiche al rover già sviluppato del precedente anno dal team "SmartExploreX".

In questa nuova versione è stato ridefinito il problema "Z":

- Navigare liberamente in un ambiente circoscritto, senza urtare oggetti circostanti a lui, in totale sicurezza.
- Cercare e riconoscere un QR code all'interno di un ambiente circoscritto.
- Una volta trovato tale QR Code il Rover emetterà un suono tramite un Buzzer.

RUOLI DEI MEMBRI DEL TEAM:

Stefano Decina:

- Sviluppo Arduino
- Sviluppo Ros
- Modellazione 3D per stampe 3D
- Animazione 3D
- Sviluppo V-rep
- Sviluppo I.A.
- Beta tester

Alessandro Mattei:

- Specialista Hardware
- Implementazione Raspberry
- Sviluppo Arduino
- Sviluppo Ros
- Sviluppo V-rep
- Sviluppo OpenMV
- Sviluppo OpenCV
- Sviluppo I.A
- Beta Tester

Mattia Caputo:

- Sviluppo V-rep
- Sviluppo Ros
- Sviluppo OpenCV
- Sviluppo I.A.
- Beta tester
- Presentazione

Sara Rebecca Di Naccio:

- Sviluppo Arduino
- Sviluppo Ros
- Sviluppo I.A
- Documentazione
- Presentazione
- Beta tester

REQUISITI FUNZIONALI

Il rover deve:

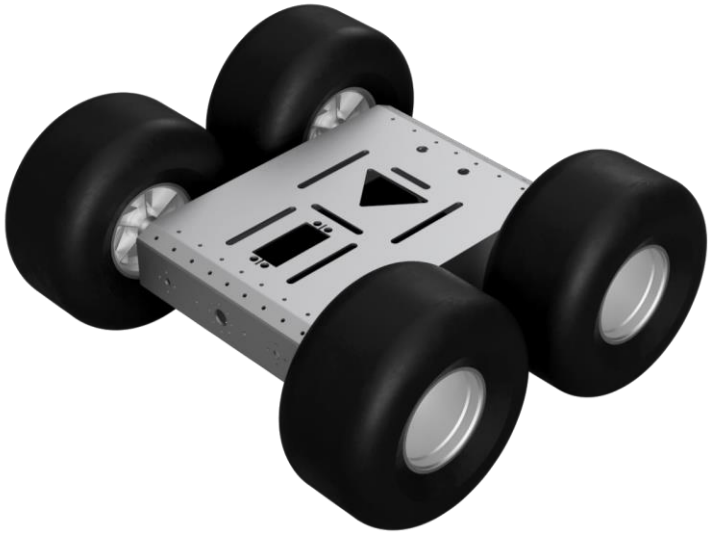
- Navigare liberamente e in sicurezza in un ambiente sconosciuto.
- Riconoscere gli ostacoli tramite pi-camera e sempre tramite essa riconoscere il percorso più sicuro.
- Riconoscere gli ostacoli tramite i sensori a ultrasuoni.
- Riconoscere gli ostacoli tramite il LIDAR.
- Riconoscere un urto con un oggetto tramite i micro-switch
- Fermarsi e cercare una via sicura in caso di urto.
- Riconoscere un QR code tramite OpenVM e fermarsi.
- Evitare gli ostacoli presenti nell'ambiente.
- Aggiustare la direzione tramite la bussola.

COMPONENTI DEL ROVER

Il rover è composto dai seguenti componenti:

- 1X Telaio del rover, in alluminio.
- 4X Motori spazzolati da 12V 0,37A.
- 4X Ruote da offroad.
- 2X Motor Driver L298N.
- 1X Batteria LiHV 3S 5200mah 10C con uscita 52A (in sostituzione alla batteria li-ion 12v 4500mah con uscita 2A, non in grado di alimentare tutto il sistema).
- 3X Sensori ad ultrasuoni HC-SR04 (due in più rispetto alla versione precedente del rover).
- 1X Micro servo SG90 (non presente nella versione precedente del rover).
- 1X Analog voltage divider V2 (Non presente nella versione precedente del rover)
- 1X Compass CMP11 Tilt Compensated Magnetic Compass.
- 1X OpenMv (non presente nella versione precedente del rover).
- 1X Pi-Camera (non presente nella versione precedente del rover).
- 1X Raspberry Pi 3 model B
- 1X Arduino Uno R3
- 2X Arduino Nano V3.2
- 10X Micro Switch a levetta (non presenti nella versione precedente del rover).
- 1X Plexiglass di dimensioni:
- Vari Componenti disegnati e stampati in 3D

TELAIO DEL ROVER



Il telaio del rover è formato da 6 parti, tutte in alluminio. Presenta delle elaborazioni create dal costruttore.

MOTORI

Il rover è motorizzato da quattro motori spazzolati. I motori presentato un sistema di rapporti per la riduzione dei giri/minuto, che aumentano la coppia erogata. Tutti e quattro i motori sono pilotati in modo indipendente, ciò garantisce un controllo eccezionale al rover.

Specifiche tecniche:

Tipologia di motore:	Spazzolato con riduzione
Tipologia di corrente di funzionamento:	DC
Voltaggio Nominale:	12V
Amperaggio Nominale:	0.37A
Rapporto di riduzione:	ND

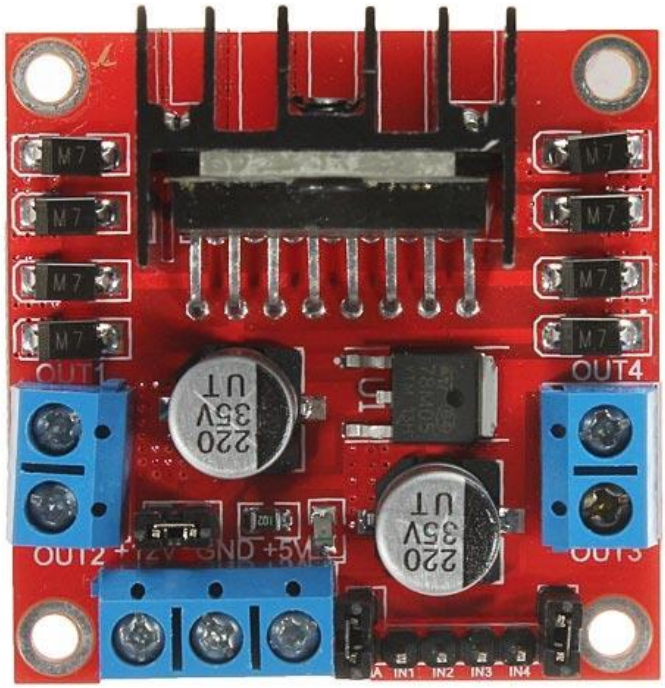
RUOTE

Il rover è dotato di quattro ruote di tipo off road.

Specifiche tecniche:

Dimensioni:	ND

MOTOR DRIVER L298N



Il rover è dotato di 2 motor driver L298N.

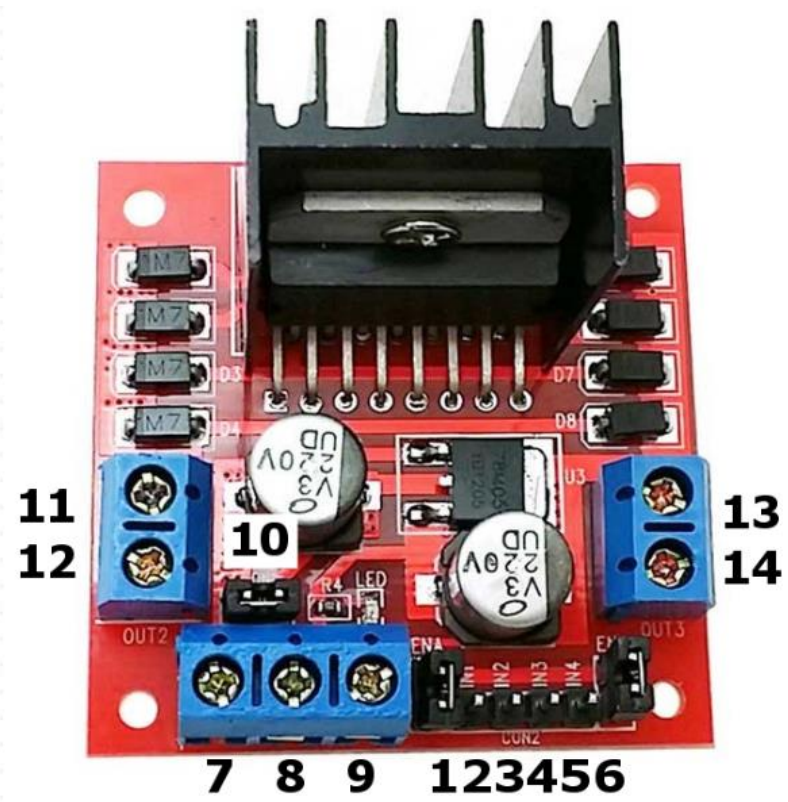
La funzione dei motor driver è quella di prendere un segnale di controllo a bassa corrente e trasformarlo in un segnale a più alta potenza in grado di pilotare un motore. In questo progetto ne useremo 2 perché ciascun driver può pilotare autonomamente 2 motori.

Un H-Bridge è un circuito in grado di pilotare una corrente in entrambe le polarità ed essere controllato dalla Pulse Width Modulation (PWM). Una modulazione a larghezza di impulso (PWM) è una tecnica di modulazione utilizzata per codificare un messaggio in un segnale pulsante. Sebbene questa tecnica di modulazione possa essere utilizzata per codificare le informazioni per la trasmissione, il suo uso principale è quello di consentire il controllo della potenza fornita ai dispositivi elettrici, in particolare ai carichi inerziali come i motori. Più gli impulsi sono lunghi più la ruota si gira più velocemente, più gli impulsi sono brevi, più lentamente ruota la ruota idraulica. I motori dureranno molto più a lungo e saranno più affidabili se controllati tramite PWM

Specifiche tecniche:

- Tensione di alimentazione: VM = 35V (max.)
- Corrente di uscita Iout: 2 A (picco)
- Modalità di funzionamento: CW / CCW / freno corto / arresto

PINS L298N MOTOR DRIVER



N°	DESCRIZIONE
1	ENA - ponticello di abilitazione motore a corrente continua A Non rimuovere nel caso si utilizzi un motore passo-passo. Connettersi a un'uscita PWM per il controllo della velocità del motore DC.
2	IN 1
3	IN 2
4	IN 3
5	IN 4
6	ENB - ponticello di abilitazione motore a corrente continua B Non rimuovere nel caso si utilizzi un motore passo-passo. Connettersi a un'uscita PWM per il controllo della velocità del motore DC.

7	Collegare la tensione di alimentazione del motore, massima di 35V DC. Rimuovere il ponticello [10] se la tensione è > 12V DC
8	GND
9	uscita 5V se 12V ponticello in luogo, ideale per alimentare il vostro Arduino (etc)
10	jumper 12V - rimuovere questo se si utilizza una tensione di alimentazione superiore a 12V DC. Ciò consente l'alimentazione tramite il regolatore 5V di bordo
11	DC motor 1 "+" o motore passo-passo A +
12	motore DC 1 "-" o motore passo-passo A-
13	motore a corrente continua 2 "+" o motore passo-passo B +
14	motore DC 2 "-" o motore passo-passo B-

BATTERIA

Il rover nella precedente versione era dotato di una batteria Li-ion con uscita di 1A che non era in grado di alimentare tutto il sistema, abbiamo quindi dovuto sostituire la batteria con una più potente una Lipoly LiHV.

Le batterie Lipoly ad alta tensione (LiHV) offrono una maggiore densità di potenza rispetto alle tradizionali batterie lipo che consentono di aumentare i tempi di utilizzo senza la penalità di peso aggiuntivo.

LiHV sta per "High Voltage Lithium Polymer". I vantaggi delle batterie LiHV sono la maggiore potenza e tempi di funzionamento più lunghi. Le batterie LiHV possono essere caricate a 4,35 V per cella, superiori alle normali batterie LiPo che si trovano a 4,20 V. Ad esempio, un pacco LiHV 3S caricato a 4,35 V darebbe 13,05 V. Considerando che una batteria Lipo con la stessa carica otterrebbe solo 12,6V. Le batterie LiHV hanno anche una resistenza interna inferiore.

Il grosso svantaggio risiede nella pericolosità delle batterie stesse.

La batteria può esplodere se corto-circuitata, a causa della bassissima resistenza interna e della conseguente tremenda corrente impulsiva che attraversa la cella. Inoltre, una cella Li-Poly può incendiarsi facilmente se forata, per cui le batterie sono, in varie applicazioni, ricoperte da un involucro plastico che dovrebbe prevenire le forature.

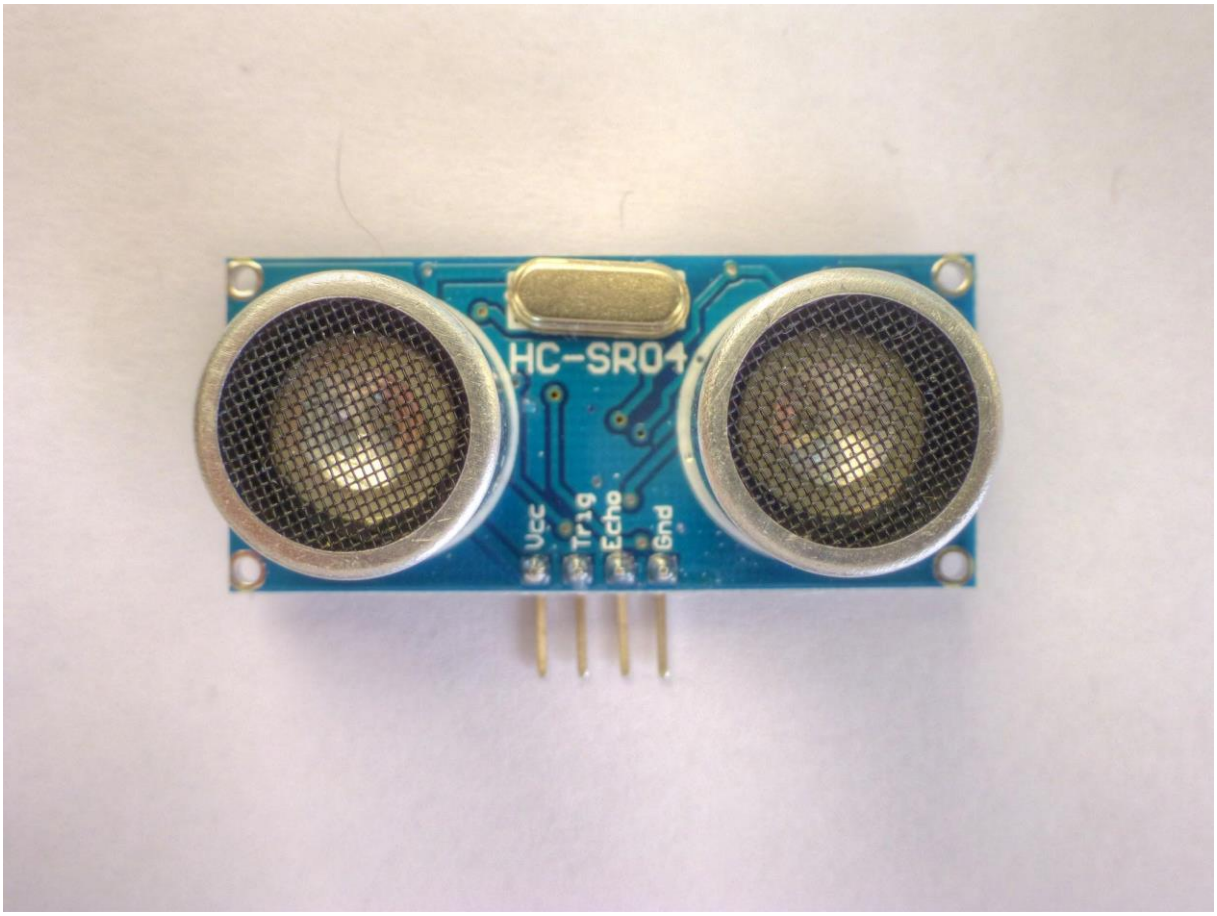
Queste batterie necessitano di usare caricabatterie specifici (vedi foto), per evitare, appunto, incendi ed esplosioni.

Specifiche tecniche Batteria:

Capacità Minima:	5200mAh
Configurazione:	3S / 11.4v / 3Cell

Costante di Scaricamento Nominale:	10C
Constante di Scaricamento Picco (10sec):	20C
Peso Batteria:	318g
Dimensioni:	108 x 36 x 47 mm
Charge Plug:	JST-XH
Discharge Plug:	XT60

SENSORE SONAR



Il rover è provvisto di 3 sensori sonar di tipo HC-SR04, questo sensore ha un trasmettitore, che invia onde ultrasoniche e un ricevitore, che prende il segnale. Si basa sulla tecnologia Sonar ***so**und **na**vigation and ranging*, è una tecnica che utilizza la propagazione del suono per la navigazione e per rilevare la presenza e la posizione di oggetti nello spazio.

Specifiche tecniche:

- Tensione di alimentazione: 5V DC
- Corrente di alimentazione: 15mA
- Frequenza di funzionamento: 40Hz
- Segnale input Trigger: 10uS TTL per impulso
- Angolo di funzionamento: 15 gradi
- Raggio misurato: 2cm - 400cm

PIN HC-SR04

- Vcc: 5VDC

- Trig: Trigger pin
- Echo: Echo pin
- GND: ground

PRINCIPIO BASE DI FUNZIONAMENTO:

Utilizzo dell'IO trigger per almeno 10 microsecondi di segnale in HIGH.

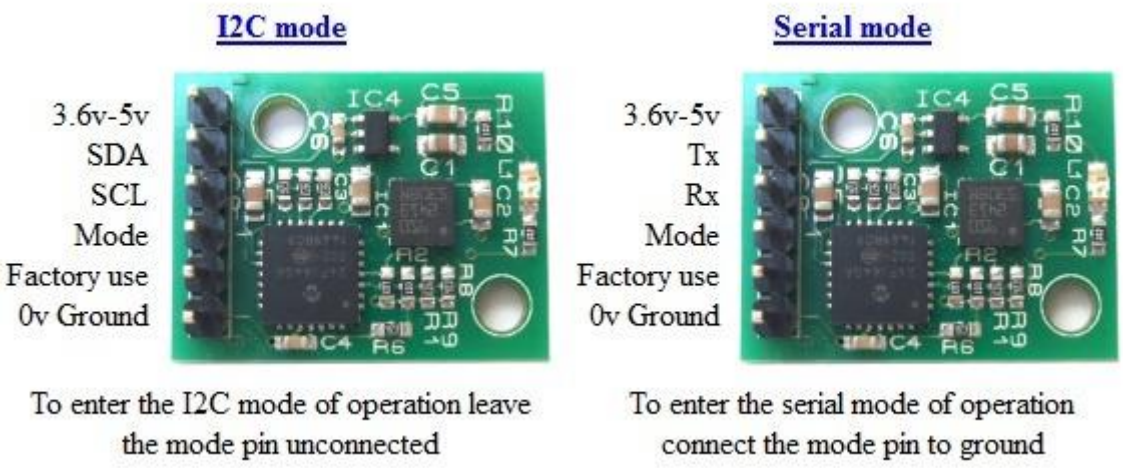
Il modulo invierà automaticamente 8 cicli di impulsi a 40 khz per aumentare eco e rileverà se è presente un segnale di impulso.

Per calcolare la distanza tra il sensore e l'oggetto si misura l'intervallo di tempo tra il trigger e eco.

Le formule per calcolare la distanza è "intervallo di tempo / 58" per le misure in centimetri oppure "intervallo di tempo / 148" per i pollici. In alternativa a queste due formule abbiamo: "Intervallo = tempo di livel HIGH * velocità del suono (340 m/s) /2".

Nota bene: se l'angolo di impatto del segnale con l'ostacolo è superiore a 15 gradi, nessun segnale sarà restituito. Inoltre, se c'è un ostacolo vicino al sensore (2 cm o meno), ci possono essere alcuni problemi nelle letture.

COMPASS CMP11



Il modulo CMPS11 è una vera bussola con tilt, compensazione dell'inclinazione. A differenza dei moduli simili che forniscono solo i dati grezzi, questa piccola scheda incorpora un potente processore a 16 bit che esegue tutti i calcoli complessi per fornire un'uscita da 0 a 359,9 gradi.

Il CMP11 implementa un magnetometro a 3 assi, un giroscopio a 3 assi e un accelerometro a 3 assi. Un filtro Kalman viene applicato al giroscopio e l'accelerometro per rimuovere gli errori causati dall'inclinazione del PCB.

Il CMPS11 produce un risultato di 0-3599 che rappresenta 0-359,9 o da 0 a 255 gradi. L'uscita dei tre sensori che misurano le componenti x, y e z del campo magnetico, insieme con il beccheggio e il rotolo, vengono utilizzati per calcolare il bearing, ciascuno di questi componenti sono resi disponibili anche in forma grezza. Il modulo CMPS11 richiede un'alimentazione a 3,6-5 V e assorbe una corrente nominale di 25 mA. Viene fornita una scelta di interfacce Seriale o I2C.

Questa bussola ritorna le seguenti misure:

- Angolo 8-Bit
- Angolo 16-Bit
- Accellerometro
- Temperatura
- Giroscopio
- Angolo Beccheggio
- Angolo Imbardata
- Angolo Rollio

Tecniche:

• Uscita Compens Compensated Tilt:	da 0 a 359.0 gradi
• Tensione di alimentazione:	da 3,6 V a 5 V
• Corrente:	35 mA tip.
• Risoluzione:	0,1 gradi
• Precisione:	migliore del 2% dopo la calibrazione
• Livelli di segnale:	3,3 V, 5 V tollerante
• Modalità I2C:	fino a 100 kHz
• Modalità seriale:	9600, 19200, 38400 baud

FREQUENZA DI AGGIORNAMENTO DI DATI

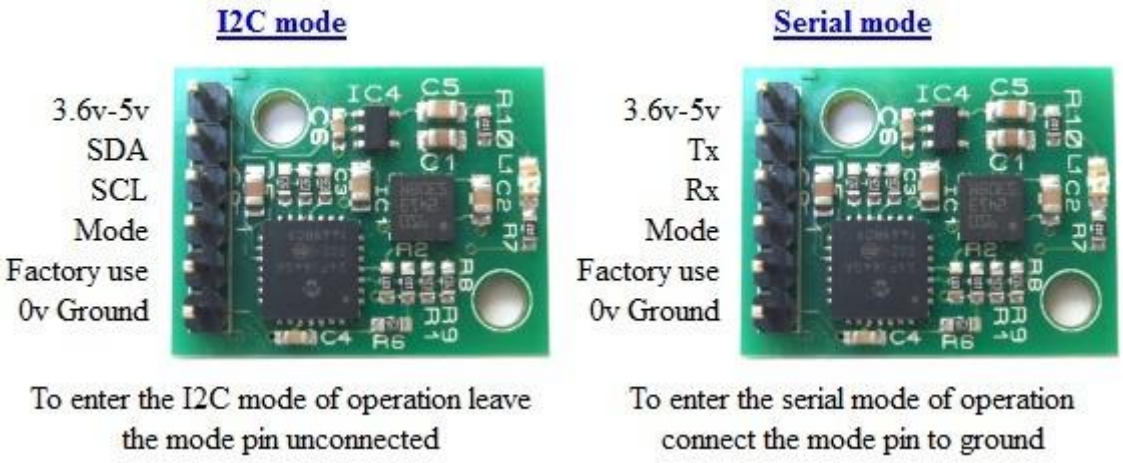
Gli aggiornamenti dell'intestazione compensata di inclinazione si verificano a 75hz con i dati filtrati per mezzo di un buffer di 45 campioni, ciò significa che viene eseguito un aggiornamento completo del buffer ogni 640ms. I dati grezzi del magnetometro e dell'accelerometro sono disponibili ogni 13,3ms.

SELEZIONE DELLA MODALITÀ

CMPS11 esamina solo i pin di selezione della modalità all'accensione.

Per entrare nella modalità operativa I2C lasciare il pin della modalità non connesso

Per accedere alla modalità operativa seriale collegare il pin della modalità a terra



Comunicazione seriale:

Per il rover abbiamo optato per la comunicazione seriale, perché è più compatibile con l'utilizzo che ne facciamo. La modalità seriale opera su un collegamento con una velocità di trasmissione predefinita di 9600 bps (senza parità, 2 bit di stop) e livelli di segnale 3.3v-5v.

COMANDI SERIALI

Di seguito una tabella che descrive i comandi che possono essere inviati a CMPS11 e i dati con cui risponderà.

Comando	Nome	I byte restituiti	Descrizione dei dati restituiti
0x11	OTTIENI LA VERSIONE	1	Versione software
0x12	OTTENERE UN ANGOLO DI 8 BIT	1	Angolo come singolo byte 0-255
0x13	OTTENERE UN ANGOLO DI 16 BIT	2	Angolo come due byte, byte alto prima 0-3599
0x14	Ottieni PITCH	1	Angolo di inclinazione +/- 0-85 ° Kalman filtrato
0x15	RICEVI	1	Angolo di rollio +/- 0-85 ° Kalman filtrato

0x16	OTTIENI PITCH NO KAL	1	Angolo di inclinazione +/- 0-85 ° senza filtro Kalman
0x17	ROLL NO NO KAL	1	Angolo di rollio +/- 0-85 ° senza filtro Kalman
0x19	OTTIENI MAG RAW	6	Dati magnetici grezzi, firmati a 16 bit: X alto, X basso, Y alto, Y basso, Z alto, Z basso
0x20	OTTIENI ACCEL RAW	6	Dati di accelerometro grezzi, firmati a 16 bit: X alto, X basso, Y alto, Y basso, Z alto, Z basso
0x21	OTTIENI GYRO RAW	6	Dati gyro grezzi, firmati a 16 bit: X alto, X basso, Y alto, Y basso, Z alto, Z basso
0x22	OTTIENI TEMP	2	Temperatura come due byte, byte alto prima 0. Risoluzione tipica 8 LSB / ° C
0x23	PRENDI TUTTO	4	angolo alto, angolo basso (0-3599), altezza (+/- 0-85), rotazione (+/- 0-85)
0xF0	CALIBRAZIONE BYTE 1	1	restituisce ok (0x55)
0xF5	CALIBRAZIONE BYTE 2	1	restituisce ok (0x55)
0xF6	CALIBRAZIONE BYTE 3 PIENO	1	restituisce ok (0x55)
0xF7	CALIBRAZIONE BYTE 3 FLAT	1	restituisce ok (0x55)
0xF8	USCITA CALIBRAZIONE	1	restituisce ok (0x55)
0x6A	RIPRISTINO 1	1	restituisce ok (0x55)
0x7C	RIPRISTINO 2	1	restituisce ok (0x55)
0x81	RIPRISTINO 3	1	restituisce ok (0x55)

0xA0	BAUD 19200	1	restituisce ok (0x55)
0xA1	BAUD 38400	1	restituisce ok (0x55)

CALIBRAZIONE DEL CMPS11

Si prega di non farlo finché la comunicazione seriale non funziona completamente. Consiglierei di valutare le prestazioni del CMPS11 prima di implementare questa funzione.

Il suo scopo è quello di rimuovere il guadagno del sensore e l'offset sia del magnetometro che dell'accelerometro e ottiene ciò cercando le massime uscite del sensore. Prima di tutto è necessario accedere alla modalità di calibrazione inviando una sequenza di 3 byte di 0xF0,0xF5 e quindi 0xF6 (leggendo il byte di conferma dopo ognuno di essi).

Il LED si spegnerà e il CMPS11 dovrebbe ora essere ruotato in tutte le direzioni in 3 dimensioni, se viene rilevato un nuovo massimo per uno qualsiasi dei sensori, il LED lampeggerà, quando non è possibile ottenere altri LED lampeggianti in qualsiasi direzione, quindi uscire dalla modalità di calibrazione con un comando di 0xF8.

Accertarsi che CMPS11 non si trovi vicino a oggetti ferrosi in quanto ciò potrebbe distorcere il campo magnetico e causare errori nella lettura. Durante la calibrazione ruotare lentamente la bussola.

Ricorda che l'asse del campo magnetico è improbabile che sia orizzontale, si immerge nella terra con un angolo che varia a seconda della posizione. Nei nostri uffici nel Regno Unito si immerge nella terra a 67 gradi e questo è l'orientamento che ogni asse della bussola deve essere per trovare i massimi.

È necessario trovare sia i massimi positivi che quelli negativi per ciascun asse, quindi ci sono 6 punti da calibrare. Anche l'accelerometro è calibrato allo stesso tempo, pertanto il modulo deve essere posizionato orizzontalmente, invertito e su tutti e 4 i lati per calibrare i 6 punti dell'accelerometro.

Ogni punto dell'accelerometro deve essere stabile per 200 mS per la sua lettura da utilizzare per la calibrazione.

Questo ritardo è intenzionale in modo che i leggeri colpetti al modulo non producano letture accelerometriche che compromettono gli angoli di beccheggio e rollio.

Non c'è ritardo per i punti magnetici. Le prestazioni del modulo sono direttamente correlate a quanto bene si esegue la calibrazione, quindi farlo lentamente e con attenzione.

CALIBRAZIONE DI CMPS11 PER FUNZIONAMENTO SOLO ORIZZONTALE

Se la bussola non richiede la compensazione dell'inclinazione, può essere utilizzata una semplice calibrazione che può essere implementata solo con una rotazione sul piano orizzontale.

Prima di tutto è necessario accedere alla modalità di calibrazione inviando una sequenza di 3 byte di 0xF0,0xF5 e quindi 0xF7 (leggendo il byte di conferma dopo ognuno di essi).

Il LED si spegnerà e il CMPS11 dovrebbe ora essere ruotato in tutte le direzioni su un piano orizzontale, se viene rilevato un nuovo massimo per uno qualsiasi dei sensori, il LED lampeggerà, quando non è possibile ottenere altri LED lampeggianti in qualsiasi direzione, quindi uscire la modalità di calibrazione con un comando di 0xF8.

Accertarsi che CMPS11 non si trovi vicino a oggetti ferrosi in quanto ciò potrebbe distorcere il campo magnetico e causare errori nella lettura. Durante la calibrazione ruotare lentamente la bussola.

RIPRISTONO DELLA CALIBRAZIONE DI FABBRICA DEL CMPS11

Per eseguire un ripristino della calibrazione di fabbrica, scrivere una sequenza di 3 comandi nell'ordine corretto. La sequenza è 0x6A, 0x7C, 0x81 (leggendo il byte di conferma dopo ognuno di essi).

MODIFICA DELLA VELOCITÀ DI TRASMISSIONE

La velocità di trasmissione seriale predefinita di 9600 può essere modificata. Ci sono altre due velocità di trasmissione che possono essere utilizzate, per il 19200 basta inviare 0xA0 o in alternativa per 38400 inviare 0xA1.

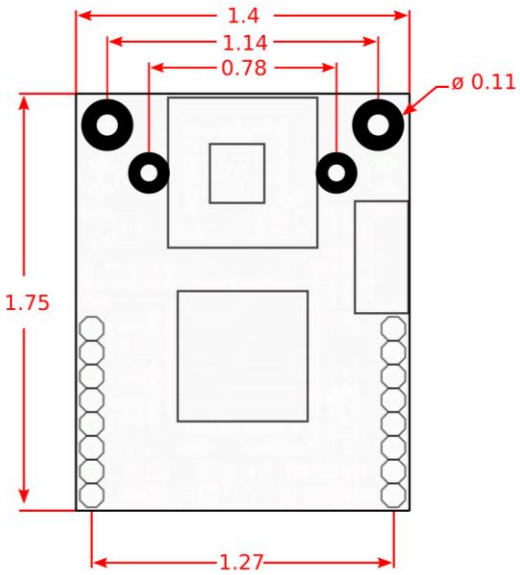
Si noti che CMPS11 sarà sempre impostato sulla frequenza 9600 bps dopo il ciclo di accensione e dopo aver impostato una nuova velocità di trasmissione, la risposta ok (0x55) verrà inviata alla velocità appena selezionata.

OPENMV CAM M4 V2

OpenMV Cam è una piccola scheda a microcontrollore a bassa potenza che consente di implementare facilmente le applicazioni utilizzando la visione artificiale nel mondo reale.

La OpenMV Cam si programma con script Python di alto livello, ciò rende più facile gestire i complessi output degli algoritmi di visione artificiale e lavorare con strutture di dati di alto livello.

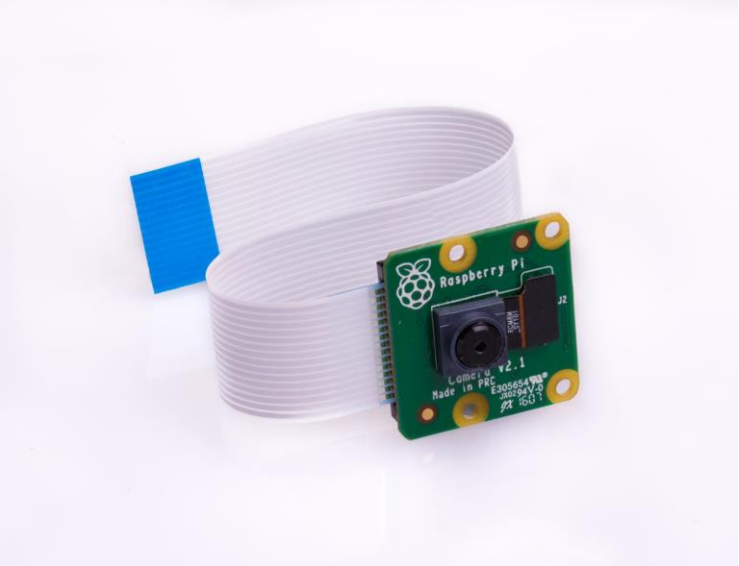
DIMENSIONI



Specifiche tecniche:

- **Processore**: CPU ARM® a 32 bit Cortex®-M4 con FPU a singola precisione 180 MHz (225 DMIPS). Punteggio Mark Mark: 608 (confronta con Raspberry Pi Zero: 2060)
- **Memoria RAM**: 64 KB .DATA /. BSS / Heap / Stack 192KB Frame Buffer / Stack (256 KB totali)
- **Memoria Flash**: Firmware da 9KBK Firmware da 9KB per bootloader 48KB 16KB (1MB totale)
- **Formati Immagine**: JPEG in scala di grigi RGB565
- **Risoluzioni supportate**: Scala di grigi: 320x240 e minore RGB565: 320x240 e inferiore. Scala di grigi JPEG: 320x240 e minore RGB565 JPEG: 320x240 e minore
- **Lente**: Lunghezza focale: 2,8 mm Apertura: F2.0 Formato: 1/3 " HFOV = 70,8 °, VFOV = 55,6 °Mount: M12 * 0,5 Filtro IR tagliato: 650nm (rimovibile)
- **Informazioni elettriche**: Tutti i pin hanno un output di 3,3V re tollerano fino a 5V. Tutti i pin possono erogare fino a 25 mA. Ilpin P6 non ha tolleranza 5V in modalità ADC o DAC. In totale tutti i pin erogano fino a 120 mA .
- **Peso**: 16g
- **Dimensioni**: 45mm x 36mm x 30mm
- **Consumo:** Idle 100mA @ 3.3V Attivo 150mA @ 3.3

PI CAMERA



Il rover è dotato del modulo ottico Pi-Camera per Raspberry-Pi 3 B, verrà usato per identificare visivamente gli ostacoli che si presentano dinanzi al rover.

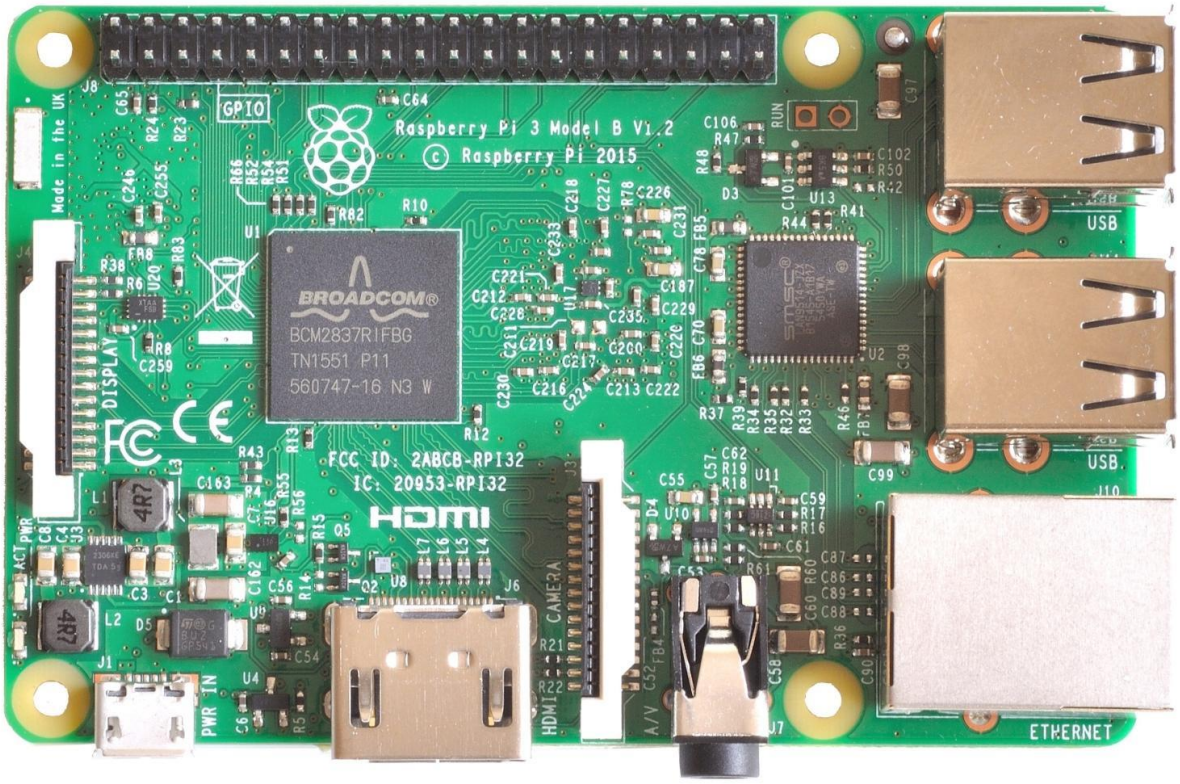
VOLTOMETER

L'analog voltage divider V2 può rilevare la tensione di alimentazione da 0,0245 V a 25 V. Questo modulo è basato sul principio del divisore del resistore. Il modulo di rilevamento della tensione consente di ridurre la tensione di ingresso 5 volte. Poiché la tensione di ingresso analogico di Arduino è normalmente massima di 5 V, la tensione di ingresso del modulo di rilevamento della tensione analogica non può superare 5 Vx5, ovvero 25 V.

SPECIFICHE

- Interface: Analog
- Input Voltage (DC): Massimo 25V, Minimo 0.0245V
- Rivela la tensione di alimentazione fino a 25 V
- Size: 22x30mm (0.87 x 1.18 in)

RASPBERRY PI 3 MODEL B



Il rover utilizza come Main Logic Board la Raspberry Pi model 3 B. È un Singol-Broad Computer di cui è dotato di un processore ARM e una GPU VideoCore. È stato scelto di installare il Sistema operativo Ubuntu MATE in quanto compatibile con la tecnologia ROS Kinetic.

ACCESSO ALLA RASPBERRY PI 3

Possiamo entrare nella raspberry attraverso la connessione wi fi o la connessione ethernet. Entrambi usano il protocollo SSH.

Connessione Ethernet:

- **Windows -** collegare il cavo Ethernet nei dispositivi. Dopodiché condividere la connessione con il laptop. Ora, utilizzare un programma come Putty per stabilire una connessione SSH. Se non si conosce ip del raspberry pi basterà digitare a posto del ip il seguente dominio raspberrypi.local.

- **Linux e Mac -** Devi conoscere l'indirizzo IP di Raspberry Pi, come sopra se non conosci l'indirizzo puoi accedere tramite il dominio Raspberrypi.local.
  Per accedere al tuo Raspberry Pi, apri un Terminale e digita il seguente comando:
  ssh -XC pi@<indirizzo ip>. Ricorda che pi è il nome utente del Raspberry Pi. Con l'opzione -XC si mette a disposizione X11 Forwording, e sarà possibile riprodurre intero Desktop Enviroment del Raspberry Pi.

Connessione WIFI:

Per stabilire una connessione WiFi, basterà accedere alla rete wi-fi creata dal raspberry pi con la seguente Password: raspberrypi

Specifiche tecniche:

SoC	Broadcom DCM2837
CPU	1.2 GHz 64-bit quad-core ARM Cortex A
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 H.264 high-profile decoder ed encoder 250 MHz per BCM2835 e BMC2836. 400 MHz dual core per BMC2837 (1080p60)
MEMORIA (SDRAM)	1 GB (condivisa con la GPU) LPDDR2
PORTE USB	2.0 X4 (attraverso un hub USB integrato)
INPUT	Video Connettore 15-pin MIPI Camera Interface (CSI), utilizzabile con fotocamere Raspberry Pi o Raspberry Pi NoIR
Output Video	Un connettore RCA per il video ed una porta HDMI
Input audio	Attraverso l'interfaccia bus seriale I^2S
Output audio	Jack da 3,5 mm, HDMI, attraverso attraverso l'interfaccia bus seriale I^2S
Memoria	MicroSD
Collegamenti di rete	Ethernet 10/100 mb/s, wireless LAN 802.11n, Bluetooth 4.1
Connettori di I/O	X40 GPIO
Orologio	Nessun orologio RTC a bordo
Corrente	800 mA (4,0 W) assorbita
Alimentazione	5 V via MicroUSB o GPIO
Dimensioni	PCB: [(85,60 mm x 53,98 mm) (3,370079-inch x 2,1251969 inch)] Tot: [(85,60 mm x 56 mm) (3,370079 inch x 2,20472 inch)]

ARDUINO UNO R3

Arduino Uno è un dispositivo basato su microcontrollore che permette di realizzare diversi tipi di circuiti elettronici. Possiede 14 pin digitali programmabili come ingressi o uscite (i quali hanno anche la capacità di essere utilizzati per funzioni dedicate come la generazione di segnale PWM o la comunicazione UART) e 6 ingressi per l'acquisizione ed elaborazione di segnali analogici. Il microcontrollore è l'ATmega328 prodotto da Atmel, ha una velocità di 16MHz, una memoria flash da 32KB, una ram da 2KB e una memoria EEPROM da 1KB. L'alimentazione della scheda avviene tramite porta usb o tramite l'apposito connettore. In caso siano collegati sia il cavo usb sia il connettore di alimentazione, la scheda è capace di scegliere automaticamente la fonte di alimentazione esterna.

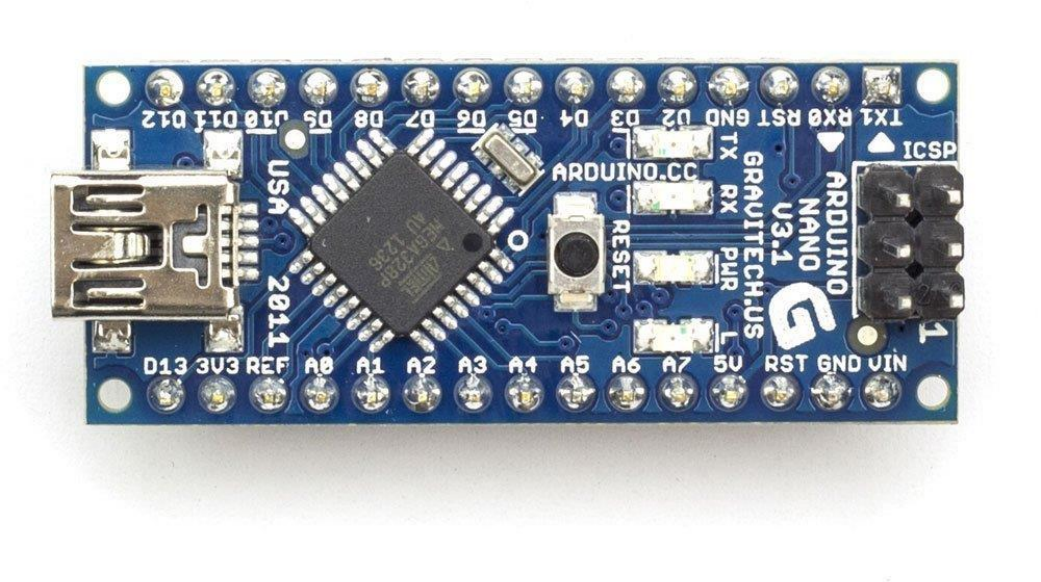
La connessione USB ha 2 funzioni:

- Fornire il circuito
- Permettere lo scambio di dati tra Arduino e altri dispositivi.

Specifiche tecniche:

Tipo Microcontrollore	Atmel ATmega328
Tensione di lavoro	5Vdc
Tensione di alimentazione consigliata	da 7Vdc a 12Vdc
Pin digitali	14 configurabili come ingressi o uscite
Pin analogici	6 ingressi
Massima corrente per pin digitale	40mA massima
Memoria Flash	32KB
Memoria Sram	2KB
Memoria EEPROM	1KB
Velocità di clock del microcontrollore	16MHz

ARDUINO NANO V3.2



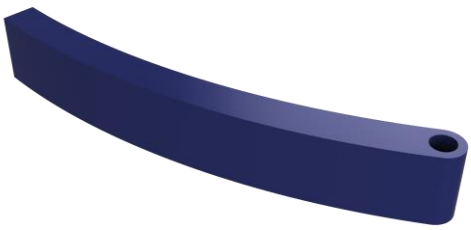
Arduino Uno è un dispositivo basato su microcontrollore che permette di realizzare diversi tipi di circuiti elettronici. Possiede 14 pin digitali programmabili come ingressi o uscite (i quali hanno anche la capacità di essere utilizzati per funzioni dedicate come la generazione di segnale PWM) e 8 ingressi per l'acquisizione ed elaborazione di segnali analogici e un oscillatore a 16 MHz. Utilizza un processore ATMEGA328 che ha 32 kB di memoria per il programma. Manca di un jack di alimentazione e viene programmata attraverso un cavo USB mini.

Specifiche Tecniche:

- Microcontrollore: ATmega328
- Tensione operativa: 5 V
- Tensione di ingresso: da 7 a 12 V
- I/O digitali: 14 (di cui 6 utilizzate come uscite PWM)
- Ingressi analogici: 8
- Corrente DC per pin I/O: 40 mA
- Memoria Flash: 32 kB (di cui 2 kB utilizzati dal bootloader)
- SRAM: 2 kB
- EEPROM: 1 kB
- Clock: 16 MHz
- Dimensioni (mm): 45x18x18

DISEGNI E STAMPE 3D

BAFFI

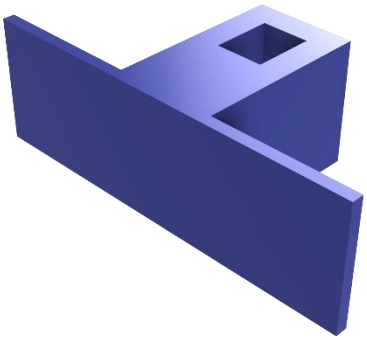


BAFFO DESTRO

BAFFO SINISTRO

Questi due oggetti vengono posti come due baffi nella zona anteriore del Rover. Hanno la capacità di attivare i microswitch quando il Robot colpirà un oggetto X lateralmente (ossia o a destra o a sinistra, ma sempre davanti). In seguito, i microswitch saranno in grado di far tornare il baffo al suo posto e così che la simulazione può procedere.

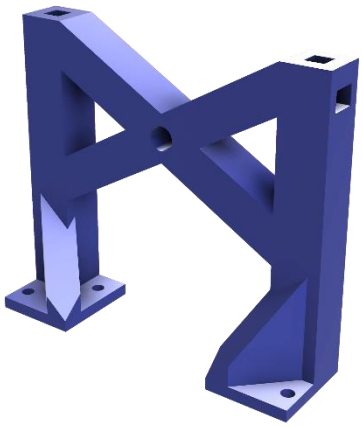
CORPO FRONTALE



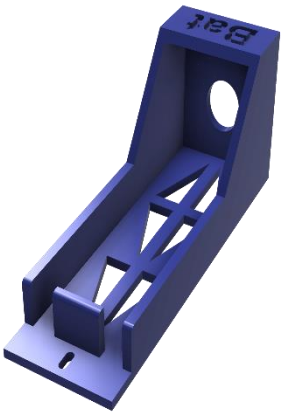
Ha la stessa funzionalità dei “Baffi”, quello di far attivare i microswitch nel momento dell’impatto ma con l’unica differenza che il Rover rileverà l’ostacolo esattamente nella parte centrale della zona anteriore affinché la sua funzione abbia effetto.

TORRI

Le due torri hanno la funzione di sorreggere il Plexiglass sull'ultimo piano, su cui verranno applicati vari dispositivi che contribuiscono al funzionamento del Rover. Ce ne saranno due: una per la zona anteriore del Rover e una per la zona posteriore.



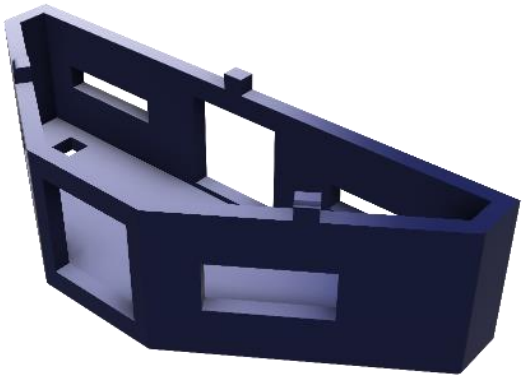
CASE BATTERIA



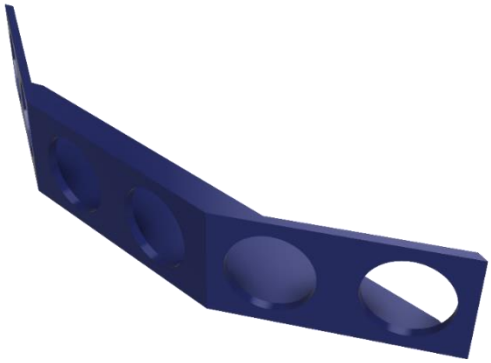
La batteria del Rover sarà posta dentro questo involucro per isolarlo da qualsiasi contatto e stabilizzarlo sul plexiglass.

SCATOLA PER ALIMENTAZIONE

Posto nella zona anteriore del Rover, questo contenitore avrà dentro di sé il pulsante di alimentazione per la sua accensione.



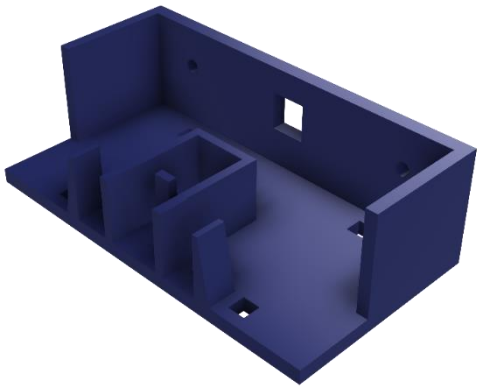
SUPPORTO SONAR



Nella parte anteriore del Rover si trova il supporto dei sonar. Posta leggermente avanti in modo che il rilevamento ultrasonico sia più preciso.

SUPPORTO PRINCIPALE

Nel supporto principale vengono messi i microswitch, il corpo frontale e un modello di applicazione dei baffi affinché possano funzionare in modo ottimale.



CAPITOLO 2

CONNESSIONI

In questo capitolo descriveremo le interconnessioni tra i vari componenti.

ARDUINO UNO – SONAR E VOLTMETER

LEFT SONAR

HC-SR04	Arduino Nano
Echo	6
Trig	7
GND	GND
5V	5V

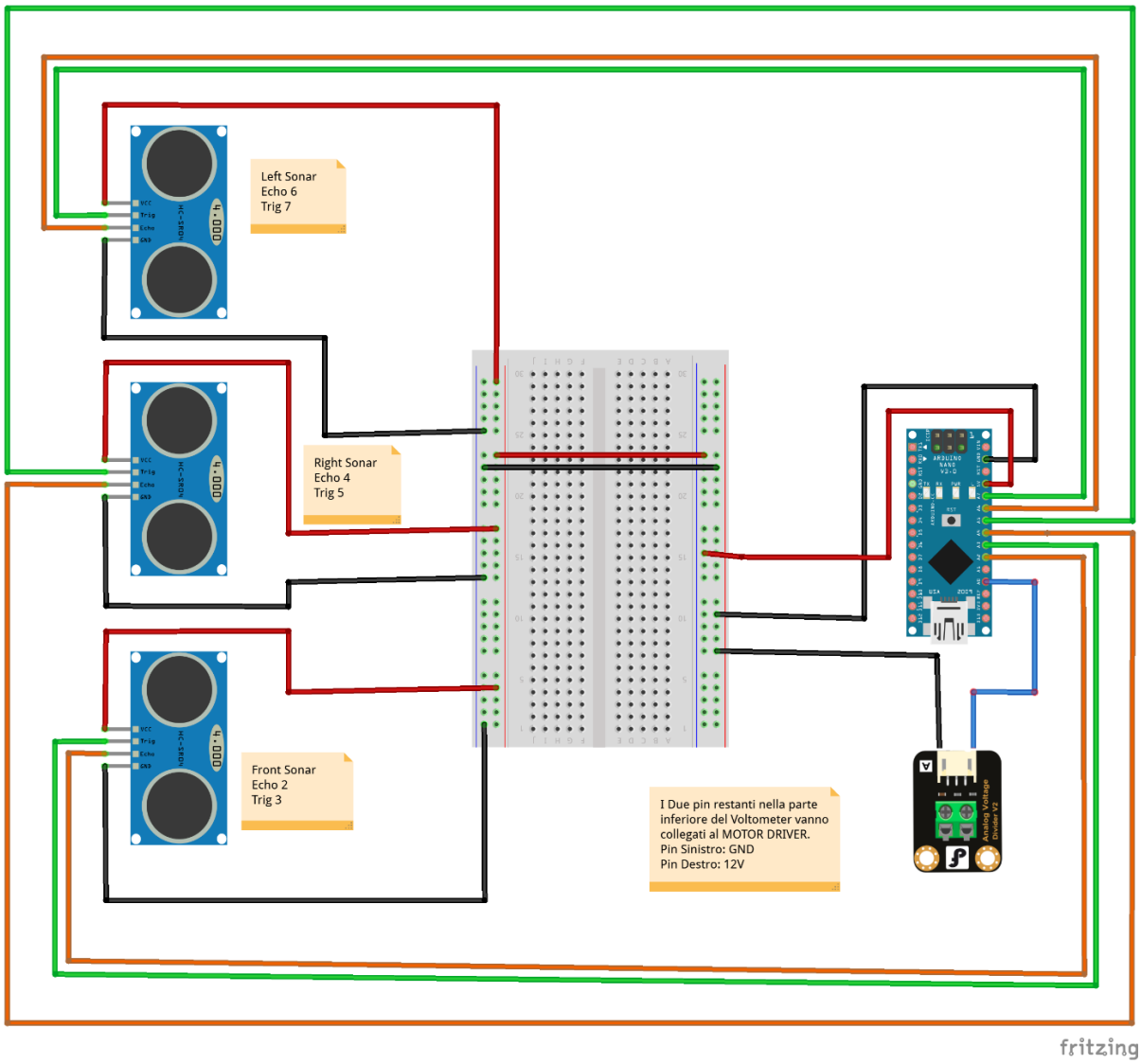
RIGHT SONAR

HC-SR04	Arduino Nano
Echo	4
Trig	5
GND	GND
5V	5V

FRONT SONAR

HC-SR04	Arduino Nano
Echo	2
Trig	3
GND	GND
5V	5V

SCHEMA DI CONNESSIONE



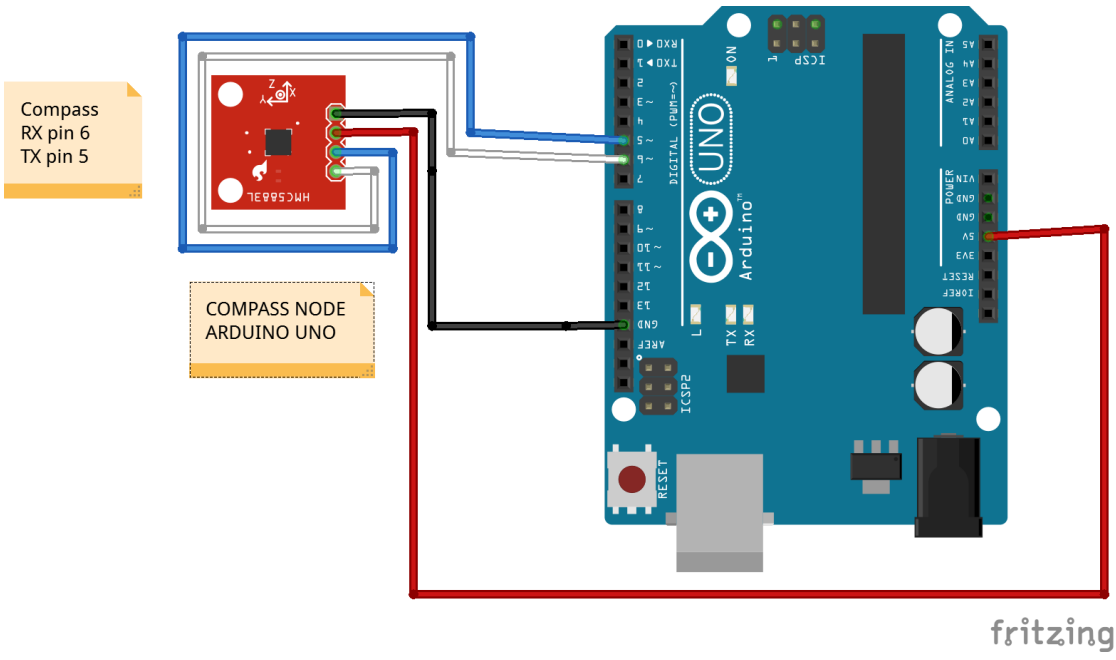
ARDUINO UNO - COMPASS

COMPASS

CMPS11 Pin	Arduino Pin	PMW Pin
GND	GND	No
5V	5V	No
RX	6	Si
TX	5	Si
Factory	Non Connesso	No

Il pin di modalità è collegato a GND perché vogliamo utilizzare la modalità di comunicazione seriale. Se vogliamo usare la mossa I2C, lascia semplicemente che questo pin sia scollegato. Il perno di fabbrica viene utilizzato per ripristinare la bussola ai dati di fabbrica.

SCHEMA DI CONNESSIONE:



ARDUINO NANO – MOTOR DRIVER E SWITCH

MOTOR DRIVER SINISTRO

H-L298N	Arduino Pin
GND	Batteria
VCC	Batteria
AIN1	2
AIN2	4
PWMA	3

MOTOR DRIVER DESTRO

H-L298N	Arduino Pin
GND	Batteria
VCC	Batteria
BIN1	5
BIN2	7
PWMB	6

SWITCH 1

D2F-FL	Arduino Pin	PWM Pin
VCC	5V	No
No	8	Si
Nc	Non Connesso	No

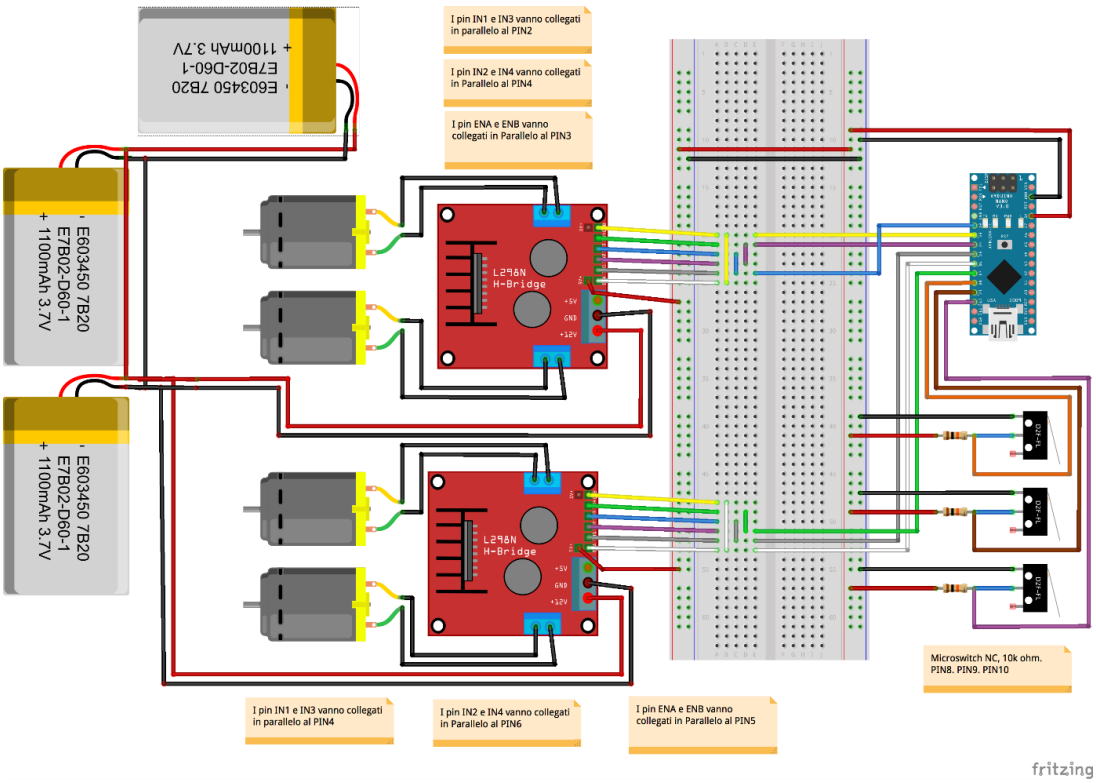
SWITCH 2

D2F-FL	Arduino Pin	PWM Pin
VCC	5V	No
No	9	Si
Nc	Non Connesso	No

SWITCH 3

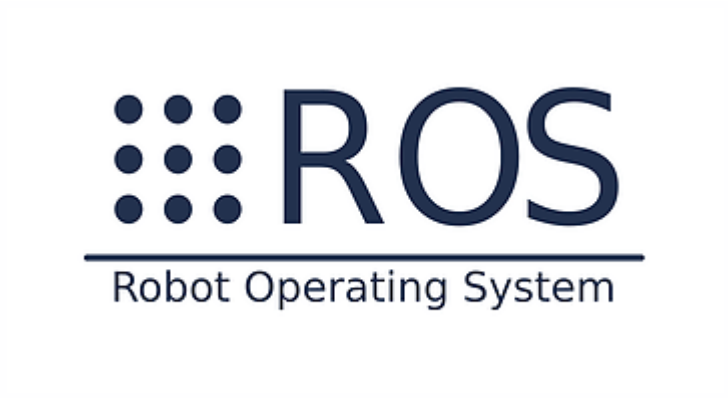
D2F-FL	Arduino Pin	PWM Pin
VCC	5V	No
No	10	Si
Nc	Non Connesso	No

SCHEMA DI CONNESSIONE:



CAPITOLO 3

INTRODUZIONE A ROS



In questo progetto abbiamo deciso di utilizzare ROS come piattaforma di tutto il sistema.

ROS (Robot Operating System) è un sistema open source. Fornisce i servizi che ci si aspetta da un sistema operativo, tra cui l'astrazione dell'hardware, il controllo dei dispositivi di basso livello, l'implementazione delle funzionalità di uso comune, il passaggio dei messaggi tra i processi e la gestione dei pacchetti. Fornisce inoltre strumenti e librerie per ottenere, creare, scrivere ed eseguire codice su più computer.

Il " **Graph** " runtime di ROS è una rete di processi peer-to-peer (potenzialmente distribuiti su macchine) che sono accoppiati liberamente usando l'infrastruttura di comunicazione ROS. ROS implementa diversi stili di comunicazione, tra cui la comunicazione sincrona in stile RPC sui servizi, lo streaming asincrono di dati su argomenti e la memorizzazione di dati su un Parameter Server.

ROS non è un framework in real time, sebbene sia possibile integrare ROS con codice in real time.

SISTEMI OPERATIVI SUPPORTATI DA ROS

Attualmente ROS funziona solo su piattaforme basate su Unix. Il software per ROS è principalmente testato su sistemi Ubuntu e Mac OS X, sebbene la comunità ROS abbia contribuito al supporto di Fedora, Gentoo, Arch Linux e altre piattaforme Linux.

Mentre una porta a Microsoft Windows per ROS è possibile, non è stata ancora completamente esplorata.

La versione attuale della distribuzione ROS è **Kinetic Kame,** installato sul sistema operativo **Ubuntu Mate.**

ROS LIVELLO FILE SYSTEM

I concetti che Ros ricopre principalmente al livello di filesystem sono le risorse che si incontrano normalmente su disco, come ad esempio:

- **Package**: i packege sono l'unità principale per l'organizzazione del software in ROS. Un package può contenere processi di runtime ROS (NODI), una libreria dipendente da ROS, set di dati, file di configurazione o qualsiasi altra cosa che sia organizzata in modo utile insieme.

- **Metapackages** : i Metapackage sono package specializzati che servono solo a rappresentare un gruppo di altri pacchetti correlati. Solitamente i metapackage vengono usati come supporto di località compatibili con le versioni precedenti per gli stack di rosbuild convertiti.

- **Packages Manifests**: Manifests (package.xml) fornisce metadati relativi a un pakage, inclusi nome, versione, descrizione, informazioni sulla licenza, dipendenze e altre meta informazioni come pacchetti esportati. Il manifest del package package.xml è definito in REP-0127.

- **Repository**: sono una raccolta di package che condividono un sistema VCS comune. I pacchetti che condividono un VCS condividono la stessa versione e possono essere rilasciati insieme utilizzando lo strumento di automazione del rilascio di catkin bloom. Spesso questi repository si associano agli stack di Rosbuild convertiti. I repository possono contenere anche un solo pacchetto.

- **Tipi di Messaggi (msg)**: descrizioni dei messaggi, memorizzati in my_package / msg / MyMessageType.msg, definiscono le strutture di dati per i messaggi inviati in ROS.
- **Tipi di Servizio (srv)**: descrizioni del servizio, memorizzate in my_package / srv / MyServiceType.srv, definiscono le strutture dei dati di richiesta e risposta per i servizi in ROS.

ROS COMPUTATION GRAPH LEVEL

Il COMPUTATION GRAPH è la rete peer-to-peer dei processi ROS che stanno elaborando i dati insieme. I concetti di Computation Graph di base di ROS sono NODI, MASTER, PARAMETER SERVER, MESSAGGI, SERVIZI, ARGOMENTI e BORSE, che forniscono tutti i dati al Grafico in diversi modi.

Questi concetti sono implementati nel repository ros_comm .

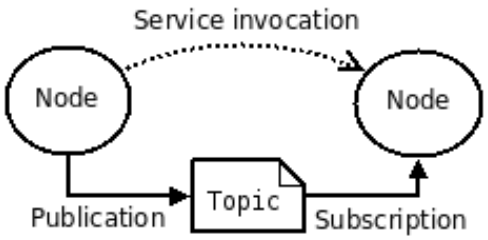
- **Nodi**: i nodi sono processi che eseguono il calcolo. ROS è progettato per essere modulare su una scala a grana fine; un sistema di controllo robot di solito comprende molti nodi. Ad esempio, un nodo controlla i motori delle ruote, un nodo esegue la localizzazione, un nodo esegue la pianificazione del percorso, un nodo fornisce una vista grafica del sistema e così via. Un nodo ROS viene scritto con l'uso di una libreria clientROS, come roscpp o rospy.
- **Master**: Il master ROS fornisce la registrazione del nome e la ricerca del resto del Graph di calcolo. Senza il Master, i nodi non sarebbero in grado di trovarsi, scambiarsi messaggi o invocare servizi.
- **Parameter Server**: il Parameter Server consente di memorizzare i dati con la chiave in una posizione centrale. Attualmente fa parte del Master.

- **Messaggi**: i nodi comunicano tra loro passando messaggi. Un messaggio è semplicemente una struttura dati, comprendente campi tipizzati. Sono supportati i tipi primitivi standard (intero, virgola mobile, booleano, ecc.), Così come le matrici di tipi primitivi. I messaggi possono includere strutture e array arbitrariamente nidificati (molto simili alle strutture C).
- **Topic**: i messaggi vengono instradati tramite un sistema di trasporto con semantica di *pubblicazione* / *sottoscrizione*. Un nodo invia un messaggio PUBBLICANDOLO su un determinato topic. Il topic è un nome che viene utilizzato per identificare il contenuto del messaggio. Un nodo interessato a un determinato tipo di dati sarà SOTTOSCRITTO al topic appropriato. Potrebbero esserci più editori e abbonati concorrenti per un singolotopic, e un singolo nodo può pubblicare e / o iscriversi a piùtopic. In generale, gli editori e gli abbonati non sono consapevoli dell'esistenza reciproca. L'idea è di disaccoppiare la produzione di informazioni dal suo consumo. Logicamente, si può pensare ad un argomento come un *message bus* fortemente tipizzato. Ogni *bus* ha un nome e chiunque può connettersi al bus per inviare o ricevere messaggi purché siano del tipo giusto.
- **Services**: Il modello di pubblicazione / sottoscrizione è un paradigma di comunicazione molto flessibile, ma il suo trasporto molti-a-molti, a senso unico, non è appropriato per le interazioni richiesta / risposta, che sono spesso richieste in un sistema distribuito. La richiesta / risposta viene effettuata tramite i services , definiti da una coppia di strutture di messaggi: uno per la richiesta e uno per la risposta. Un nodo di fornitura offre un servizio sotto un nome e un client utilizza il servizio inviando il messaggio di richiesta e in attesa della risposta.
- **Bags**: le bags sono un formato per salvare e riprodurre i dati dei messaggi ROS. I Bags sono un meccanismo importante per la memorizzazione dei dati, come i dati dei sensori.

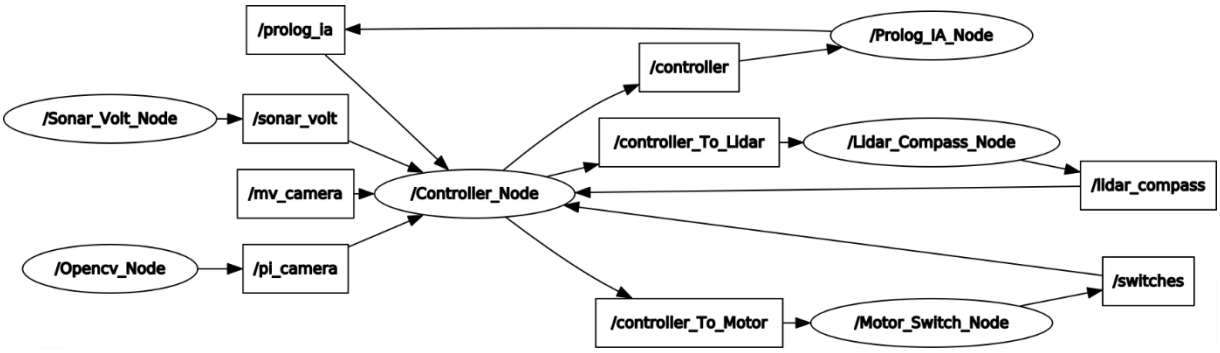
Il ROS Master funge da nameservice nel COMPUTATION GRAPH di ROS. Memorizza gli topic e le informazioni di registrazione dei servizi per i nodi ROS. I nodi comunicano con il Master per segnalare le proprie informazioni di registrazione. Poiché questi nodi comunicano con il Master, possono ricevere informazioni su altri nodi registrati e stabilire connessioni appropriate. Il Master effettuerà anche le callback a questi nodi quando queste informazioni di registrazione cambiano, consentendo ai nodi di creare dinamicamente le connessioni man mano che i nuovi nodi vengono eseguiti.

I nodi si connettono direttamente ad altri nodi; il Master fornisce solo informazioni di ricerca, proprio come un server DNS. I nodi che sottoscrivono un argomento richiedono le connessioni dai nodi che pubblicano quell'argomento e stabiliscono tale connessione tramite un protocollo di connessione concordato. Il protocollo più comune utilizzato in un ROS è chiamato TCPROS, che utilizza socket TCP / IP standard.

Funzionamento della comunicazione tra nodi:



Il Nostro Ros Graph è il seguente



Il nodo Controller è l'unico a comunciare con tutti gli altri nodi. Ogni nodo (nella figura è un OVALE), per comunicare, pubblica un topic (nella fidura è il RETTANGOLO) al quale si sottoscrive e permette al nodo che riceve il messaggio di leggerlo. Ogni freccetta uscente è un output del messaggio, mentre ogni freccetta entratante è un input di messaggio.

CODICE

In questo capitolo riporteremo tutti i codici che abbiamo usato, commentati.

NODI

ARDUINO UNO: COMPASS NODE

```
1.  // libreria ros
2.  #include <ros.h>
3.  // custom message Compass_Node
4.  #include <py_robot/Compass_Node.h>
5.  // libreria Servo Motore
6.  #include <Servo.h>
7.  // libreria seriale di comunicazione con il Compass
8.  #include <SoftwareSerial.h>
9.
10. // handle nodon ROS
11. ros::NodeHandle nh;
12.
13. // assegnamento pin Compass
14. #define compassRX 6
15. #define compassTX 5
16.
17. // assegnamento variabili per Compass
18. #define CMPS_GET_ANGLE8 0x12
19. #define CMPS_GET_ANGLE16 0x13
20. #define CMPS_GET_PITCH 0x14
21. #define CMPS_GET_ROLL 0x15
22. #define CMPS_GET_MAG_RAW 0x19
23. #define CMPS_GET_ACCEL_RAW 0x20
24. #define CMPS_GET_GYRO_RAW 0x21
```

```
#define CMPS_GET_TEMP_RAW 0x22

// Servo
Servo myservo;

// Compass
SoftwareSerial cmps11 = SoftwareSerial(compassTX, compassRX);

// ROS custom message Compass
py_robot::Compass_Node compass_msg;

// ROS inizializzazione Pubblisher Compass
ros::Publisher pub_compass("compass", &compass_msg);

void setup() {
  // ROS inizializzazione nodo
  nh.initNode();
  // ROS creazione Publisher
  nh.advertise(pub_compass);
  // inizializzazione seriale per il Compass
  cmps11.begin(9600);
}

// funzione lettura Compass
// PARAM: CMPS_GET_ANGLE16, CMPS_GET_ANGLE8, CMPS_GET_PITCH, CMPS_GET_ROLL, CMPS_GET_MAG_RAW,  CMPS_GET_ACCEL_RAW, CMPS_GET_TEMP_RAW
// RETURN

void readcompass() {
  // variabili d'appoggio usate per calcolare gli angolazione a 8 bit e a 16 bit
  unsigned char high_byte, low_byte, angle8 ;
  unsigned int angle16;

  // richiesta e lettura dell'angolo a 16 bit
  cmps11.write(CMPS_GET_ANGLE16);
  while (cmps11.available() < 2);
  high_byte = cmps11.read();
  low_byte = cmps11.read();
  // calcolo dell'angolo a 16 bit
  angle16 = high_byte;
  angle16 <<= 8;
  angle16 += low_byte;
  //assegnamento dell'angolo calcolato nella variabile angle16 del custom message compass
  compass_msg.angle16 = (int)angle16;

  // richiesta e lettura dell'angolo a 8 bit
  cmps11.write(CMPS_GET_ANGLE8);
  while (cmps11.available() < 1);
  //assegnamento del valore nella variabile angle8 del custom message compass
  compass_msg.angle8 = (int)cmps11.read();

  // richiesta e lettura del valore di beccheggio
  cmps11.write(CMPS_GET_PITCH);
  while (cmps11.available() < 1);
  //assegnamento del valore nella variabile pitch del custom message compass
  compass_msg.pitch = (int)cmps11.read();

  //richiesta e lettura del valore di rollio
  cmps11.write(CMPS_GET_ROLL);
  while (cmps11.available() < 1);
  //assegnamento del valore nella variabile roll del custom message compass
  compass_msg.roll = (int)cmps11.read();

  //richiesta e lettura dei valori del magnetometro
  cmps11.write(CMPS_GET_MAG_RAW);
```

```
  while (cmps11.available() < 6);
  //assegnamento dei valori 6 nella variabile array mag del custom message compass
  compass_msg.mag[0] = (int)cmps11.read();
  compass_msg.mag[1] = (int)cmps11.read();
  compass_msg.mag[2] = (int)cmps11.read();
  compass_msg.mag[3] = (int)cmps11.read();
  compass_msg.mag[4] = (int)cmps11.read();
  compass_msg.mag[5] = (int)cmps11.read();

  //richiesta e lettura dei valori dell'accelerometro
  cmps11.write(CMPS_GET_ACCEL_RAW);
        while (cmps11.available() < 6);
        //assegnamento dei 6 valori nella variabile array acc del custom message co
    mpass
        compass_msg.acc[0] = (int)cmps11.read();
        compass_msg.acc[1] = (int)cmps11.read();
        compass_msg.acc[2] = (int)cmps11.read();
        compass_msg.acc[3] = (int)cmps11.read();
        compass_msg.acc[4] = (int)cmps11.read();
        compass_msg.acc[5] = (int)cmps11.read();

        //richiesta e lettura dei valori del giroscopio
        cmps11.write(CMPS_GET_GYRO_RAW);
        while (cmps11.available() < 6)
          //assegnamento dei 6 valori nella variabile array gyro del custom message
     compass;
        compass_msg.gyro[0] = (int)cmps11.read();
        compass_msg.gyro[1] = (int)cmps11.read();
        compass_msg.gyro[2] = (int)cmps11.read();
        compass_msg.gyro[3] = (int)cmps11.read();
        compass_msg.gyro[4] = (int)cmps11.read();
        compass_msg.gyro[5] = (int)cmps11.read();

        //richiesta e lettura dei valori del termometro
        cmps11.write(CMPS_GET_TEMP_RAW);
        while (cmps11.available() < 2);
        //assegnamento dei 2 valori nella variabile array temp del custom message c
    ompass;
        compass_msg.temp = (int)cmps11.read();
        compass_msg.temp = (int)cmps11.read();

        delay(100);
      }

      void loop() {
        readcompass();
        // funzione Publish ROS
        pub_compass.publish(&compass_msg);
        // attesa eventi ROS
        nh.spinOnce();
        delay(10);
      }
```

ARDUINO NANO: MOTOR DRIVER & SWITCH NODE

```
// libreria ros
#include <ros.h>
// custom message Motor_Switch_Node
#include <py_robot/Motor_Switch_Node.h>
// custom message Controller_Node
```

```
6.  #include <py_robot/Controller_To_Motor_Node.h>
7.
8.  // handle nodon ROS
9.  ros::NodeHandle nh;
10.
11. // assegnamento pin motori 1 e 2
12. #define SXPWM 3
13. #define SXIN1 4
14. #define SXIN2 5
15.
16. // assegnamento pin motori 3 e 4
17. #define DXPWM 6
18. #define DXIN1 7
19. #define DXIN2 8
20.
21.
22. // assegnamento pin switch
23. #define switchFront 11
24. #define switchLeft 10
25. #define switchRight 9
26.
27. // variabili globali
28. String comando;
29. int vel;
30.
31. //funzione callback
32. // PARAM: messaggio ros velo
33.
34. void callback( const py_robot::Controller_To_Motor_Node& velo) {
35.   // assegnamento del comando impartito dal Nodo Controller
36.   comando = velo.velo;
37.
38. }
39. // funzione sottoscrizione di ros
40. ros::Subscriber<py_robot::Controller_To_Motor_Node> motor_sub("controller_To_Motor",
     callback);
41.
42. // ros custom message sonar_volt
43. py_robot::Motor_Switch_Node switches;
44.
45. // ros inizializzazione Pubblisher sonar_volt
46. ros::Publisher switch_pub("switches", &switches);
47.
48. // funzione per switch
49. // PARAM: switchpin
50. // RETURN 0 se il lo switch non è attivato, 1 se è attivato
51.
52. void setup() {
53.   // inizializzazione nodo
54.   nh.initNode();
55.   nh.advertise(switch_pub);
56.   nh.subscribe(motor_sub);
57.
58.   // pin switch
59.   pinMode(switchFront, INPUT);
60.   pinMode(switchRight, INPUT);
61.   pinMode(switchLeft, INPUT);
62.   // pin motori 1 e 2 di sinistra
63.   pinMode(SXPWM, OUTPUT);
64.   pinMode(SXIN1, OUTPUT);
65.   pinMode(SXIN2, OUTPUT);
66.   analogWrite(SXPWM, 0);
67.
68.   // pin motori 3 e 4 di destra
69.   pinMode(DXPWM, OUTPUT);
70.   pinMode(DXIN1, OUTPUT);
```

```
  pinMode(DXIN2, OUTPUT);
  analogWrite(DXPWM, 0);

  delay(1000);
}

// funzione per switch
// PARAM: switchpin
// RETURN 0 se il lo switch non è attivato, 1 se è attivato

long switchsensor(int switchpin) {
  // controllo se lo switch è stato premuto
  if (digitalRead(switchpin) == HIGH) {
    delay(20);
    // dopo 20 millisecondi controllo di nuovo se è ancora premuto per evitare error
i
    if (digitalRead(switchpin) == HIGH) {
      // ritorna 0 se lo switch non è stato attivato
      return 0;
    }
  }
  // ritorna 1 se lo switch è stato attivato
  return 1;
}

// funzioni relative ai comandi da impartire ai motori

// funzione Avanti
// PARAM: intero vel: parametro velocità
// PARAM: intero tempo: parametro tempo di esecuzione
      void vaiAvanti(int vel, int tempo) {
        // relativo al motor drive collegato ai motori del lato sinistro
        digitalWrite(SXIN1, 1);
        digitalWrite(SXIN2, 0);
        // velocità lato sinistro
        analogWrite(SXPWM, vel);
        // relativo al motor drive collegato ai motori del lato destro
        digitalWrite(DXIN1, 1);
        digitalWrite(DXIN2, 0);
        // velocità lato destro
        analogWrite(DXPWM, vel);

        delay(tempo);
      }

      // funzione Indietro
      // PARAM: intero vel: parametro velocità
      // PARAM: intero tempo: parametro tempo di esecuzione
      void vaiIndietro(int vel, int tempo) {
        // relativo al motor drive collegato ai motori del lato sinistro
        fermo(0);
        digitalWrite(SXIN1, 0);
        digitalWrite(SXIN2, 1);
        // velocità lato sinistro
        analogWrite(SXPWM, vel);
        // relativo al motor drive collegato ai motori del lato destro
        digitalWrite(DXIN1, 0);
        digitalWrite(DXIN2, 1);
        // velocità lato destro
        analogWrite(DXPWM, vel);

        delay(tempo);
        fermo(0);
      }

      // funzione Sinistra
```

```
136.        // PARAM: intero vel: parametro velocità
137.        // PARAM: intero tempo: parametro tempo di esecuzione
138.        void vaiDestra(int vel, int tempo) {
139.          // relativo al motor drive collegato ai motori del lato sinistro
140.          digitalWrite(SXIN1, 1);
141.          digitalWrite(SXIN2, 0);
142.          // velocità lato sinistro
143.          analogWrite(SXPWM, vel);
144.          // relativo al motor drive collegato ai motori del lato destro
145.          digitalWrite(DXIN1, 0);
146.          digitalWrite(DXIN2, 1);
147.          // velocità lato destro
148.          analogWrite(DXPWM, vel);
149.
150.          delay(tempo);
151.          fermo(0);
152.          comando = "s";
153.        }
154.
155.        // funzione Destra
156.        // PARAM: intero vel: parametro velocità
157.        // PARAM: intero tempo: parametro tempo di esecuzione
158.        void vaiSinistra(int vel, int tempo) {
159.          // relativo al motor drive collegato ai motori del lato sinistro
160.          digitalWrite(SXIN1, 0);
161.          digitalWrite(SXIN2, 1);
162.          // velocità lato sinistro
163.          analogWrite(SXPWM, vel);
164.          // relativo al motor drive collegato ai motori del lato destro
165.          digitalWrite(DXIN1, 1);
166.          digitalWrite(DXIN2, 0);
167.          // velocità lato destro
168.          analogWrite(DXPWM, vel);
169.
170.          delay(tempo);
171.          fermo(0);
172.          comando = "s";
173.        }
174.
175.        // funzione Stop
176.        // PARAM: intero vel: parametro velocità
177.        // PARAM: intero tempo: parametro tempo di esecuzione
178.        void fermo(int tempo) {
179.          // relativo al motor drive collegato ai motori del lato sinistro
180.          digitalWrite(SXIN1, 0);
181.          digitalWrite(SXIN2, 0);
182.          // velocità lato sinistro
183.          analogWrite(SXPWM, 0);
184.          // relativo al motor drive collegato ai motori del lato destro
185.          digitalWrite(DXIN1, 0);
186.          digitalWrite(DXIN2, 0);
187.          // velocità lato destro
188.          analogWrite(DXPWM, 0);
189.
190.          delay(tempo);
191.        }
192.
193.
194.        void loop() {
195.
196.          //switch case con i varin casi possibili dei comandi ricevuti dal Nodo Cont
    roller
197.          switch (comando[0]) {
198.
199.            case 'a': vaiAvanti(100, 50);
200.              break;
```

```
201.
202.           case 'b': vaiAvanti(150, 50);
203.             break;
204.
205.           case 'c': vaiAvanti(200, 50);
206.             break;
207.
208.           case 'd': vaiAvanti(255, 50);
209.             break;
210.
211.           case 'e': vaiIndietro(150, 50);
212.             break;
213.
214.           case 'l': vaiSinistra(255, 200);
215.             break;
216.
217.           case 'r': vaiDestra(255, 200);
218.             break;
219.
220.           case 'f': vaiSinistra(255, 50);   // correzione sinistra
221.             break;
222.
223.           case 'g': vaiDestra(255, 50);    // correzione destra
224.             break;
225.
226.           case 'h': vaiSinistra(255, 2100);  // 360 antiorario
227.             break;
228.
229.           case 'i': vaiDestra(255, 2100);   //360 orario
230.             break;
231.
232.           case 'm': vaiSinistra(255, 800); // 90° a sinistra
233.             break;
234.
235.           case 'n': vaiDestra(255, 800);   // 90° a destra
236.             break;
237.
238.           case 'o': vaiSinistra(255, 1200); // 180° a sinistra
239.             break;
240.
241.           case 'p': vaiDestra(255, 1200);   // 180° a destra
242.             break;
243.
244.
245.
246.           case 's': fermo(0);
247.             break;
248.
249.         }
250.
251.         // assegnamento valore, 0 o 1, dello switch centrale nella prima posizione
    dell'array switches,
252.         // prima variabile del custom message Motor_Switch_Node
253.         switches.switches[0] = switchsensor(switchFront);
254.         // assegnamento valore, 0 o 1, dello switch Sinistra nella seconda posizion
    e dell'array switches,
255.         // prima variabile del custom message Motor_Switch_Node
256.         switches.switches[1] = switchsensor(switchLeft);
257.         // assegnamento valore, 0 o 1, dello switch Destro nella terza posizione de
    ll'array switches,
258.         // prima variabile del custom message Motor_Switch_Node
259.         switches.switches[2] = switchsensor(switchRight);
260.         // funzione Publish ROS
261.         switch_pub.publish(&switches);
262.         // procedura da attuare dopo che uno degli switch viene attivato
```

```
263.         if (switches.switches[0] == 0 || switches.switches[1] == 0 || switches.swit
    ches[2] == 0) {
264.           // stop
265.           fermo(0);
266.           // indietro per ___ secondi
267.           vaiIndietro(150, 2000);
268.         }
269.         // attesa eventi ROS
270.         nh.spinOnce();
271.         delay(10);
272.       }
```

ARDUINO NANO: SONAR & VOLTOMETER NODE

```
1.  // libreria ros
2.  #include <ros.h>
3.  // custom message Sonar_Volt_Node
4.  #include <py_robot/Sonar_Volt_Node.h>
5.
6.  // ros
7.  ros::NodeHandle nh;
8.
9.  // assegnamento dei pin sonar
10. #define echoPinFront 2
11. #define trigPinFront 3
12. #define echoPinRight 4
13. #define trigPinRight 5
14. #define echoPinLeft 6
15. #define trigPinLeft 7
16.
17. // assegnamento pin voltometer
18. #define analogInput 0
19.
20. // variabili voltometer
21. float val;
22. float temp;
23.
24. // ros custom message sonar_volt
25. py_robot::Sonar_Volt_Node sonar_volt_msg;
26.
27. // ros inizializzazione Pubblisher sonar_volt
28. ros::Publisher sonar_volt_pub("sonar_volt", &sonar_volt_msg);
29.
30.
31. void setup() {
32.   // inizializzazione nodo
33.   nh.initNode();
34.   nh.advertise(sonar_volt_pub);
35.
36.   // pin sonar
37.   pinMode(echoPinFront, INPUT );
38.   pinMode(trigPinFront, OUTPUT );
39.   pinMode(echoPinRight, INPUT );
40.   pinMode(trigPinRight, OUTPUT );
41.   pinMode(echoPinLeft, INPUT );
42.   pinMode(trigPinLeft, OUTPUT );
43.
44.   // pin voltometer
45.   pinMode(analogInput, INPUT);
```

```
46.
47. }
48.
49. // funzione per il sonar
50. // PARAM: triggerPin, echoPin
51. // RETURN: distance di tipo long, ritorna la lettura della distanza
52.
53. long sonarsensor(int triggerPin, int echoPin) {
54.   long distance = 0;
55.   // spengo il trigger
56.   digitalWrite(triggerPin, LOW);
57.   // accendo il trigger
58.   digitalWrite(triggerPin, HIGH);
59.   // aspetto 10 millisecondi
60.   delayMicroseconds(10);
61.   // spengo il trigger
62.   digitalWrite(triggerPin, LOW);
63.   // tempo ritorno dell'impulso
64.   long times = pulseIn(echoPin, HIGH);
65.   // se il tempo non supera 38000 abbiamo perso l'impulso e ritornerà 0
66.   if (times < 38000) {
67.     // calcolo della distanza
68.     distance = 0.034 * times / 2;
69.     // ritorno la distanza
70.     return distance;
71.   }
72.   // ritorno la distanza = 0
73.   return distance;
74. }
75.
76. // funzione voltometer
77. // RETURN: val di tipo float, ritorna voltaggio in imput dalla batteria
78.
79. float voltometer() {
80.   // lettura segnale
81.     val=analogRead(0);
82.     // operazione di conversione segnale imput in volt
83.     temp=val/40.92;
84.     // sottraggo margine di errore
85.     val=temp - 0.7;
86.     // ritorno valore volt
87.     return val;
88. }
89.
90. void loop() {
91.   // assegnamento valore del sonar centrale nella prima posizione dell'array sonar,

92.   // prima variabile del custom message Sonar_Volt
93.   sonar_volt_msg.sonar[0] = sonarsensor(trigPinFront, echoPinFront);
94.   // assegnamento valore del sonar destro nella seconda posizione dell'array sonar,

95.   // prima variabile del custom message Sonar_Volt
96.   sonar_volt_msg.sonar[1] = sonarsensor(trigPinRight, echoPinRight);
97.   // assegnamento valore del sonar sinistro nella terza posizione dell'array sonar,

98.   // prima variabile del custom message Sonar_Volt
99.   sonar_volt_msg.sonar[2] = sonarsensor(trigPinLeft, echoPinLeft);
100.         // funzione del voltometro e assegnamento nella seconda variabile del custo
    m message
101.         sonar_volt_msg.volt = voltometer();
102.         // funzione Publish ros
103.         sonar_volt_pub.publish(&sonar_volt_msg);
104.         // attesa eventi ROS
105.         nh.spinOnce();
106.         delay(10);
107.       }
```

CONTROLLER NODE

```
#! /usr/bin/python
import rospy
import roslaunch.rlutil
import roslaunch.parent
import roslaunch
import py_robot.msg as PyRobot
import time

controller_msg = PyRobot.Controller_Node()
controller_to_motor_msg = PyRobot.Controller_To_Motor_Node()

angle16 = None
angle8 = None
pitch = None
roll = None
mag = [None for x in range(0, 6)]
acc = [None for x in range(0, 6)]
gyro = [None for x in range(0, 6)]
temp = None
comando = None
switch = [None for x in range(0, 3)]
sonar = [None for x in range(0, 3)]
volt = None
eureca = None
visione = None
launch = None
risposta_prolog = None


def resetvar():
    """
    Funzione che setta le variabili a nullo per evitare l'invio di messaggi non completi.
    Lavora sulle variabili globali angle16, angle8, pitch, roll, mag, acc, gyro,
    temp,  comando, switch, sonar, volt, eureca e visione
    :return: nulla
    """
    global angle16, angle8, pitch, roll, mag, acc, gyro, temp, switch, sonar, volt, eureca, visione, \
        comando, risposta_prolog

    angle16 = None
    angle8 = None
    pitch = None
    roll = None
    mag = [None for x in range(0, 6)]
    acc = [None for x in range(0, 6)]
    gyro = [None for x in range(0, 6)]
    temp = None
    comando = None
    switch = [None for x in range(0, 3)]
    sonar = [None for x in range(0, 3)]
    volt = None
    eureca = None
    visione = None
    risposta_prolog = None


def callback_prolog(msg):
```

```
    """
    funzione di callback per i messaggi provenienti dal nodo Prolog_IA,
    modifica la variabile globale comando
    :param msg: messagio ricevuto
    :return: nulla
    """
    global comando, risposta_prolog

    risposta_prolog = msg.risposta

    comando = ''
    risposta_prolog = msg.risposta
    if risposta_prolog == "dritto":
        comando = "e"
    elif risposta_prolog == "destra":
        comando = "r"
    elif risposta_prolog == "sinistra":
        comando = "q"
    elif risposta_prolog == "indietro":
        comando = "i"
    elif risposta_prolog == "stop":
        comando = "s"
    elif risposta_prolog == "correggi_a_destra":
        comando = "t"
    elif risposta_prolog == "correggi_a_sinistra":
        comando = "u"
    elif risposta_prolog == "fine":
        endfunction()
    else:
        comando = 's'


def startfunction():
    """
    Funzione di inizio programma. Si occupa di far partire i nodi
    :return: nulla
    """
    uuid = roslaunch.rlutil.get_or_generate_uuid(None, False)
    roslaunch.configure_logging(uuid)
    launch = roslaunch.parent.ROSLaunchParent(uuid, [
        "../launch/node_launcher.launch"])  # qui da aggiornare con il path del file launch
    launch.start()
            rospy.loginfo("started")


       def endfunction():
            """
            Funzione di fine programma. Si occupa di far terminare il Nodo Controller

            :return: nulla
            """
            global comando
            launch.shutdown()
            time.sleep(1)
            rospy.signal_shutdown('Mission Complete')


       def callback_switch(msg):
            """
            Funzione Callback che processa i dati inviati dai Nodo Motor_Switch
            :param msg: messaggio ROS dal Nodo Motor_Switch
            :return: nulla
            """
            global switch
            switch = msg.switches
```

```
122.            rospy.loginfo(switch)
123.
124.
125.        def callback_sonar_volt(msg):
126.            """
127.            Funzione Callback che processa i dati inviati dai Nodo Sonar_Volt
128.            :param msg: messaggio ROS dal Nodo Sonar_Volt
129.            :return: nulla
130.            """
131.            global sonar, volt
132.            sonar = msg.sonar
133.            volt = msg.volt
134.
135.
136.        def callback_compass(msg):
137.            """
138.            Funzione Callback che processa i dati inviati dai Nodo Compass
139.            :param msg: messaggio ROS dal Nodo Compass
140.            """
141.            global angle16, angle8, pitch, roll, mag, acc, gyro, temp
142.            angle16 = msg.angle16
143.            angle8 = msg.angle8
144.            pitch = msg.pitch
145.            roll = msg.roll
146.            mag = msg.mag
147.            acc = msg.acc
148.            gyro = msg.gyro
149.            temp = msg.temp
150.
151.
152.        def callback_mvcamera(msg):
153.            """
154.            Funzione Callback che processa i dati inviati dai Nodo MV_Camera
155.            :param msg: messaggio ROS dal Nodo MV_Camera
156.            :return: nulla
157.            """
158.            global eureca
159.            eureca = msg.eureca
160.
161.
162.        def callback_pi_camera(msg):
163.            """
164.            Funzione Callback che processa i dati inviati dai Nodo PI_Camera
165.            :param msg: messaggio ROS dal Nodo PI_Camera
166.            :return: nulla
167.            """
168.            global visione
169.            visione = msg.visione
170.
171.
172.        def ifNotNone(angle16, angle8, pitch, roll, mag, acc, gyro, temp, volt, sonar
    , visione, risposta_prolog, switch):
173.            """
174.            funzione di controlo delle variabili globali,
175.            controlla se sono state modificate tutte
176.            :param risposta_prolog:
177.            :param visione:
178.            :param angle16: misura del angolo a 16
179.            :param angle8: misura del angolo a 8
180.            :param pitch: misura del pitch
181.            :param roll: misura del roll
182.            :param mag: misura del mag
183.            :param acc: misura del acc
184.            :param gyro: misura del gyro
185.            :param temp: misura della temperatura
186.            :param comando: variabile del comando da inviare
```

```
187.            :param volt: misura della batteria
188.            :param sonar: misura dei sonar
189.            :return: True se sono modificate, False se sono ancora None
190.            """
191.
192.            return angle16 is not None and angle8 is not None and pitch is not None a
    nd roll is not None \
193.                    and mag is not None and acc is not None and gyro is not None and t
    emp is not None \
194.                    and volt is not None and sonar is not None and visione is not None
     \
195.                    and risposta_prolog is not None and not switch == [None, None, Non
    e]
196.
197.
198.        def main():
199.            """
200.            Funzione principale che si occupa di inizializzare il Nodo, i Pubblisher
    e i Subscriber. Si occupa anche di salvare
201.            i valori ricevuti dai vari Nodi nei mesaggi Controller_Node.msg
202.            :return: nulla
203.            """
204.            global angle16, angle8, pitch, roll, mag, acc, gyro, temp, switch, sonar,
     volt, eureca, visione, \
205.                comando, risposta_prolog
206.            # inizializzazione nodo Controller
207.            rospy.init_node("Controller_Node", disable_signals=True)
208.            startfunction()
209.            # inizializzazioni Publisher e Subsciber
210.            controller_pub = rospy.Publisher("controller", PyRobot.Controller_Node, q
    ueue_size=1)
211.            controller_to_motor_pub = rospy.Publisher("controller_To_Motor", PyRobot.
    Controller_To_Motor_Node, queue_size=1)
212.            rospy.Subscriber("prolog", PyRobot.Prolog_IA_Node, callback_prolog)
213.            rospy.Subscriber("switches", PyRobot.Motor_Switch_Node, callback_switch)

214.            rospy.Subscriber("sonar_volt", PyRobot.Sonar_Volt_Node, callback_sonar_vo
    lt)
215.            rospy.Subscriber("mv_camera", PyRobot.MV_Camera_Node, callback_mvcamera)

216.            rospy.Subscriber("compass", PyRobot.Compass_Node, callback_compass)
217.            rospy.Subscriber("pi_camera", PyRobot.Pi_Camera_Node, callback_pi_camera)

218.            r = rospy.Rate(1)
219.            while not rospy.is_shutdown():
220.                if ifNotNone(angle16, angle8, pitch, roll, mag, acc, gyro, temp, volt
    , sonar, visione, risposta_prolog, switch):
221.                    controller_msg.angle16 = angle16  # messaggio per il nodo Prolog
    per angle16
222.                    controller_msg.angle8 = angle8  # messaggio per il nodo Prolog pe
    r angle8
223.                    controller_msg.pitch = pitch  # messaggio per il nodo Prolog per
    pitch
224.                    controller_msg.roll = roll  # messaggio per il nodo Prolog per ro
    ll
225.                    controller_msg.mag = mag  # messaggio per il nodo Prolog per mag

226.                    controller_msg.acc = acc  # messaggio per il nodo Prolog per acc

227.                    controller_msg.gyro = gyro  # messaggio per il nodo Prolog per gy
    ro
228.                    controller_msg.temp = temp  # messaggio per il nodo Prolog per te
    mp
229.                    controller_msg.switches = switch  # messaggio per il nodo Prolog
    per switch
```

```
                    controller_msg.volt = volt  # messaggio per il nodo Prolog per volt
                    controller_msg.sonar = sonar  # messaggio per il nodo Prolog per sonar
                    controller_msg.visione = visione  # messaggio per il nodo Prolog per Py Camera

                    if eureca == 'trovato':
                        controller_msg.qrcode = 1  # messaggio per il nodo Prolog per qrcode

                    if comando is not None:
                        controller_to_motor_msg.velo = comando  # messaggio per il nodo Motor_Switch per Motor

                    # funzioni Publish ROS
                    controller_pub.publish(controller_msg)
                    controller_to_motor_pub.publish(controller_to_motor_msg)

                    resetvar()
                    r.sleep()


        if __name__ == '__main__':
            try:
                main()
            except rospy.ROSInterruptException:
                pass
```

OPEN_MV NODE

```
    global sonar, volt
    sonar = msg.sonar
    volt = msg.volt


def callback_compass(msg):
    """
    Funzione Callback che processa i dati inviati dai Nodo Compass
    :param msg: messaggio ROS dal Nodo Compass
    """
    global angle16, angle8, pitch, roll, mag, acc, gyro, temp
    angle16 = msg.angle16
    angle8 = msg.angle8
    pitch = msg.pitch
    roll = msg.roll
    mag = msg.mag
    acc = msg.acc
    gyro = msg.gyro
    temp = msg.temp


def callback_mvcamera(msg):
    """
    Funzione Callback che processa i dati inviati dai Nodo MV_Camera
    :param msg: messaggio ROS dal Nodo MV_Camera
    :return: nulla
    """
    global eureca
    eureca = msg.eureca


```

```
def callback_pi_camera(msg):
    """
    Funzione Callback che processa i dati inviati dai Nodo PI_Camera
    :param msg: messaggio ROS dal Nodo PI_Camera
    :return: nulla
    """
    global visione
    visione = msg.visione


def ifNotNone(angle16, angle8, pitch, roll, mag, acc, gyro, temp, volt, sonar, visione, risposta_prolog, switch):
    """
    funzione di controlo delle variabili globali,
    controlla se sono state modificate tutte
    :param risposta_prolog:
    :param visione:
    :param angle16: misura del angolo a 16
    :param angle8: misura del angolo a 8
    :param pitch: misura del pitch
    :param roll: misura del roll
    :param mag: misura del mag
    :param acc: misura del acc
    :param gyro: misura del gyro
    :param temp: misura della temperatura
    :param comando: variabile del comando da inviare
    :param volt: misura della batteria
    :param sonar: misura dei sonar
    :return: True se sono modificate, False se sono ancora None
    """

    return angle16 is not None and angle8 is not None and pitch is not None and roll is not None \
           and mag is not None and acc is not None and gyro is not None and temp is not None \
           and volt is not None and sonar is not None and visione is not None \
           and risposta_prolog is not None and not switch == [None, None, None]


def main():
    """
    Funzione principale che si occupa di inizializzare il Nodo, i Pubblisher e i Subscriber. Si occupa anche di salvare
    i valori ricevuti dai vari Nodi nei mesaggi Controller_Node.msg
    :return: nulla
    """
    global angle16, angle8, pitch, roll, mag, acc, gyro, temp, switch, sonar, volt, eureca, visione, \
        comando, risposta_prolog
    # inizializzazione nodo Controller
    rospy.init_node("Controller_Node", disable_signals=True)
    startfunction()
    # inizializzazioni Publisher e Subsciber
    controller_pub = rospy.Publisher("controller", PyRobot.Controller_Node, queue_size=1)
    controller_to_motor_pub = rospy.Publisher("controller_To_Motor", PyRobot.Controller_To_Motor_Node, queue_size=1)
    rospy.Subscriber("prolog", PyRobot.Prolog_IA_Node, callback_prolog)
    rospy.Subscriber("switches", PyRobot.Motor_Switch_Node, callback_switch)
    rospy.Subscriber("sonar_volt", PyRobot.Sonar_Volt_Node, callback_sonar_volt)
    rospy.Subscriber("mv_camera", PyRobot.MV_Camera_Node, callback_mvcamera)
    rospy.Subscriber("compass", PyRobot.Compass_Node, callback_compass)
    rospy.Subscriber("pi_camera", PyRobot.Pi_Camera_Node, callback_pi_camera)
    r = rospy.Rate(1)
    while not rospy.is_shutdown():
```

```
        if ifNotNone(angle16, angle8, pitch, roll, mag, acc, gyro, temp, volt, sonar, visione, risposta_prolog, switch):
            controller_msg.angle16 = angle16  # messaggio per il nodo Prolog per angle16
            controller_msg.angle8 = angle8  # messaggio per il nodo Prolog per angle8
            controller_msg.pitch = pitch  # messaggio per il nodo Prolog per pitch
            controller_msg.roll = roll  # messaggio per il nodo Prolog per roll
            controller_msg.mag = mag  # messaggio per il nodo Prolog per mag
            controller_msg.acc = acc  # messaggio per il nodo Prolog per acc
            controller_msg.gyro = gyro  # messaggio per il nodo Prolog per gyro
            controller_msg.temp = temp  # messaggio per il nodo Prolog per temp
            controller_msg.switches = switch  # messaggio per il nodo Prolog per switch
                    controller_msg.volt = volt  # messaggio per il nodo Prolog per volt
                    controller_msg.sonar = sonar  # messaggio per il nodo Prolog per sonar
                    controller_msg.visione = visione  # messaggio per il nodo Prolog per Py Camera

                    if eureca == 'trovato':
                        controller_msg.qrcode = 1  # messaggio per il nodo Prolog per qrcode

                    if comando is not None:
                        controller_to_motor_msg.velo = comando  # messaggio per il nodo Motor_Switch per Motor

                    # funzioni Publish ROS
                    controller_pub.publish(controller_msg)
                    controller_to_motor_pub.publish(controller_to_motor_msg)

                    resetvar()
                    r.sleep()


        if __name__ == '__main__':
            try:
                main()
            except rospy.ROSInterruptException:
                pass
```

OPEN_CV NODE

```
#! /usr/bin/python

from picamera.array import PiRGBArray
from picamera import PiCamera
import cv2
import numpy as np
import time
import py_robot.msg as PyRobot
import rospy

pi_camera_msg = PyRobot.Pi_Camera_Node()

camera = PiCamera()
camera.resolution = (640, 480)
rawCapture = PiRGBArray(camera)
```

```
time.sleep(0.1)


def imgs():
    global camera, rawCapture
    camera.capture(rawCapture, format="bgr")
    image = rawCapture.array
    ris = cv2.Canny(image, 100, 200)
    misura = np.array([])
    height, width = ris.shape
    ris1 = ris.copy()
    for x in range(width - 1):
        for h in range(height - 1):
            hei = height - h - 1
            if ris1[hei, x] == 255:
                misura = np.append(misura, int(hei))
                break
            else:
                ris1.itemset((hei, x), 50)
    misura = np.split(misura, [90, 220])
    media_sinistra = np.median(misura[0])
    media_centrale = np.median(misura[1])
    media_destra = np.median(misura[2])
    print(media_sinistra, media_centrale, media_destra)
    rawCapture.truncate(0)
    if media_sinistra < media_centrale:
        if (media_sinistra < media_destra):
            print("vai a sinistra: ", media_sinistra)
            return "sinistra"
        else:
            print("vai a destra: ", media_destra)
            return "destra"
    elif media_centrale < media_destra:
        print("continua dritto: ", media_centrale)
        return "centro"
    else:
        print("vai a destra: ", media_destra)
        return "destra"


def main():
    rospy.init_node("Pi_Camera_Node", disable_signals=True)
    pi_camera_pub = rospy.Publisher("pi_camera_pub", PyRobot.Pi_Camera_Node, queue_s
ize=1)
    r = rospy.Rate(1)
    while not rospy.is_shutdown():
        pi_camera_msg.visione = imgs()
        pi_camera_pub.publish(pi_camera_msg)
        r.sleep()


if __name__ == '__main__':
    try:
        main()
    except rospy.ROSInterruptException:
        pass
```

PROLOG_IA_NODE

```
#! /usr/bin/python
```

```
2.
3.  import pyswip.prolog
4.  import py_robot.msg as PyRobot
5.  import rospy
6.  import time
7.  import numpy as np
8.
9.  prolog_msg = PyRobot.Prolog_IA_Node()
10.
11. prolog = None
12. commandolder = "avanti"
13. oldangle = None
14.
15.
16. # var global
17. qrcode = None
18. fotocamera = None
19. switch = None
20. sonar = None
21. volt = None
22. lidar = None
23. angle16 = None
24. angle8 = None
25. pitch = None
26. roll = None
27. mag = None
28. acc = None
29. gyro = None
30. temp = None
31. lidarm = None
32.
33.
34. def resetNone():
35.     """
36.     Funzione per il reset delle variabili ambientali
37.     :return:
38.     """
39.     global qrcode, fotocamera, switch, sonar, volt, lidar, angle16, angle8, pitch, r
    oll, mag, acc, gyro, temp
40.     qrcode = None
41.     fotocamera = None
42.     switch = None
43.     sonar = None
44.     volt = None
45.     lidar = None
46.     angle16 = None
47.     angle8 = None
48.     pitch = None
49.     roll = None
50.     mag = None
51.     acc = None
52.     gyro = None
53.     temp = None
54.
55.
56. def prologinit_and_rules():
57.     """
58.     Funzione per l'inizializzazione di prolog e la creazione delle regole
59.     :return: nulla
60.     """
61.     global prolog
62.
63.     ############################
64.     # INIZIALIZZAZIONE PROLOG #
65.     ############################
66.
```

```
67.     prolog = pyswip.Prolog()
68.
69.     ############################
70.     # CREAZIONE COMANDI ROVER #
71.     ############################
72.
73.     # comandi disponibili
74.     prolog.assertz("command(avanti, 1)")
75.     prolog.assertz("command(sinistra, 2)")
76.     prolog.assertz("command(destra, 3)")
77.     prolog.assertz("command(indietro,4)")
78.     prolog.assertz("command(stop, 5)")
79.     prolog.assertz("command(fine, 6)")
80.     prolog.assertz("command(batteria, 7)")
81.     prolog.assertz("command(correggi_a_destra, 8)")
82.     prolog.assertz("command( correggi_a_sinistra, 9)")
83.
84.     ##############################
85.     # CREAZIONE REGOLE GENERALI #
86.     ##############################
87.
88.     # regola per gli switch false se sbatte true se non sbatte
89.     prolog.assertz("switch(_):- switch(sinistra,X), switch(centro,Y), switch(destra,
    Z), X == 0, Y == 0, Z == 0, !")
90.
91.     # regola per il cambio direzione
92.     prolog.assertz("distancetrue(_):- sonar(destra, A), sonar(centro, B), sonar(sini
    stra, C),  A > 50, B > 50, C > 50, !")
93.
94.     # regola per il cambio direzione (indietro) a una certa distanza da un possibile
    ostacolo troppo vicino
95.     prolog.assertz("sonartrue(_):- sonar(destra, A), sonar(centro, B), sonar(sinistr
    a, C),  A > 30, B > 30, C > 30, !")
96.
97.     # regola per capire quale e' il sonar con la distanza maggiore
98.     prolog.assertz("sonar(Y) :- sonar(centro, A), sonar(destra, B), sonar(sinistra,
    C), D is max(A, B),"
99.                    " X is max(D, C), sonar(Y, X),!")
100.
101.          # regola per capire quale e' il sonar tra destra e sinistra maggiore
102.          prolog.assertz("sonardxsx(Y):- sonar(destra, A), sonar(sinistra, B), X is
    max(A, B), sonar(Y, X), Y \== centro, !")
103.
104.          # regola per le condizioni necessarie per il cambio di qualasiasi direzio
    ne
105.          prolog.assertz("condictrue(_):- switch(_), volt(Y), Y>11, qrcode(Q), Q ==
    0, !")
106.
107.          # commando batteria
108.          prolog.assertz("command(X):- volt(Y), Y < 11, C is 7, command(X, C),!")
109.
110.          # comando fine
111.          prolog.assertz("command(X):- qrcode(1), C is 6, command(X,C),!")
112.
113.          # comando correggi a destra
114.          prolog.assertz("command(X):-
    commandolder(O), O == avanti, condictrue(_), distancetrue(_), angle8(A), oldangle8(B
    ),"
115.                         " S is A-B, S > 10, command(X, 8),!")
116.
117.          # comando correggi a sinistra
118.          prolog.assertz("command(X):-
    commandolder(O), O == avanti, condictrue(_), distancetrue(_), angle8(A), oldangle8(B
    ),"
119.                         " S is B-A, S > 10, command(X, 9),!")
120.
```

```
121.            # comando indietro
122.            prolog.assertz("command(X):- commandolder(O), O \== indietro, \+ sonartru
    e(_), volt(Y), Y > 11,"
123.                           " qrcode(Q), Q == 0, command(X,4),!")
124.            prolog.assertz("command(X):- commandolder(O), O \== indietro, switch(_,1)
    , volt(Y), Y > 11,"
125.                           " qrcode(Q), Q == 0, command(X,4),!")
126.
127.            # comando avanti
128.            prolog.assertz("command(X):- commandolder(O), O == avanti, condictrue(_),
     distancetrue(_), command(X,1), !")
129.            prolog.assertz("command(X):- commandolder(O), O \== indietro, condictrue(
    _), distancetrue(_), sonar(U), "
130.                           "U == centro, fotocamera(F), F == centro, command(X,1), !"
    )
131.            prolog.assertz("command(X):- commandolder(O), O \== indietro, condictrue(
    _), distancetrue(_),"
132.                           " sonar(U), U == centro, command(X,1), !")
133.            prolog.assertz("command(X):- commandolder(O), O \== indietro, condictrue(
    _), distancetrue(_), command(X,1), !")
134.
135.            # comando sinistra
136.            prolog.assertz("command(X):- condictrue(_), sonartrue(_), sonardxsx(U), U
     == sinistra, fotocamera(F),"
137.                           " F == sinistra, command(X,2),!")
138.            prolog.assertz("command(X):- condictrue(_), sonartrue(_), sonardxsx(U), U
     == sinistra, command(X,2),!")
139.
140.            # comando destra
141.            prolog.assertz("command(X):- condictrue(_), sonartrue(_), sonardxsx(U), U
     == destra, fotocamera(F),"
142.                           " F == destra, command(X,3), !")
143.            prolog.assertz("command(X):- condictrue(_), sonartrue(_), sonardxsx(U), U
     == destra, command(X,3) ,!")
144.
145.            # comando indietro che viene eseguito quando non c'e' spazio per girare a
     destra o a sinista
146.            prolog.assertz("command(X):- \+ sonartrue(_), volt(Y), Y > 11, qrcode(Q),
     Q == 0, command(X,4),!")
147.
148.            # comando stop (errore)
149.            prolog.assertz("command(X):- command(X,5),!")
150.
151.
152.        def prologIA(commandolder, qrcode, fotocamera, switch, sonar, volt, lidar,
153.                     angle16, angle8, pitch, roll, mag, acc, gyro, temp, oldangle, li
    darm):
154.            """
155.            Funzione per il caricamento dei fatti, per esecuzione delle query e della
     demolizione dei fatti caricati
156.            :param commandolder:  vecchio comando di default e' avanti
157.            :param qrcode: 0 per non trovato 1 per trovato
158.            :param fotocamera: vale sinistra o destra o centro e identifica lo spazio
     libero
159.            :param switch: 0 se non attivo 1 per attivo e' un arry di 3 elementi
160.            :param sonar: vale la misurazione effettuata e' un array di 3 elementi
161.            :param volt: vale la misurazione effettuata
162.            :param lidar: vale le misurazioni effettuate e' un array di 19 elementi
163.            :param angle16: vale la misurazione effettuata
164.            :param angle8: vale la misurazione effettuata
165.            :param pitch: vale la misurazione effettuata
166.            :param roll: vale la misurazione effettuata
167.            :param mag: vale le misurazioni effettuate e' un array di 6 elementi
168.            :param acc: vale le misurazioni effettuate e' un array di 6 elementi
169.            :param gyro: vale le misurazioni effettuate e' un array di 6 elementi
170.            :param temp: vale la misurazione effettuata
```

```
171.            :param oldangle: vale un angolo iniziale
172.            :return: resultato della query
173.            """
174.            global prolog
175.
176.            ####################
177.            # CREAZIONE FATTI #
178.            ####################
179.
180.            # comando vecchio
181.            prolog.assertz("commandolder(" + str(commandolder) + ")")
182.
183.            # qrcode:  0 per false 1 per true
184.            prolog.assertz("qrcode(" + str(qrcode) + ")")
185.
186.            # fotocomera
187.            prolog.assertz("fotocamera(" + str(fotocamera) + ")")
188.
189.            # switch
190.            prolog.assertz("switch(centro, " + str(switch[1]) + ")")
191.            prolog.assertz("switch(sinistra, " + str(switch[0]) + ")")
192.            prolog.assertz("switch(destra, " + str(switch[2]) + ")")
193.
194.            # sonar: mettere prima il centro cosi se sono 3 sonar uguali il sistema p
    rendera il primo e quindi va avanti
195.            prolog.assertz("sonar(centro, " + str(sonar[1]) + ")")
196.            prolog.assertz("sonar(sinistra, " + str(sonar[0]) + ")")
197.            prolog.assertz("sonar(destra, " + str(sonar[2]) + ")")
198.
199.            # volt
200.            prolog.assertz("volt(" + str(volt) + ")")
201.
202.            # compass
203.            prolog.assertz("angle16(" + str(angle16) + ")")
204.            prolog.assertz("angle8(" + str(angle8) + ")")
205.            prolog.assertz("pitch(" + str(pitch) + ")")
206.            prolog.assertz("roll(" + str(roll) + ")")
207.            prolog.assertz("mag(xhigh, " + str(mag[0]) + ")")
208.            prolog.assertz("mag(xlow, " + str(mag[1]) + ")")
209.            prolog.assertz("mag(yhigh, " + str(mag[2]) + ")")
210.            prolog.assertz("mag(ylow, " + str(mag[3]) + ")")
211.            prolog.assertz("mag(zhigh, " + str(mag[4]) + ")")
212.            prolog.assertz("mag(zlow, " + str(mag[5]) + ")")
213.            prolog.assertz("acc(xhigh, " + str(acc[0]) + ")")
214.            prolog.assertz("acc(xlow, " + str(acc[1]) + ")")
215.            prolog.assertz("acc(yhigh, " + str(acc[2]) + ")")
216.            prolog.assertz("acc(ylow, " + str(acc[3]) + ")")
217.            prolog.assertz("acc(zhigh, " + str(acc[4]) + ")")
218.            prolog.assertz("acc(zlow, " + str(acc[5]) + ")")
219.            prolog.assertz("gyro(xhigh, " + str(gyro[0]) + ")")
220.            prolog.assertz("gyro(xlow, " + str(gyro[1]) + ")")
221.            prolog.assertz("gyro(yhigh, " + str(gyro[2]) + ")")
222.            prolog.assertz("gyro(ylow, " + str(gyro[3]) + ")")
223.            prolog.assertz("gyro(zhigh, " + str(gyro[4]) + ")")
224.            prolog.assertz("gyro(zlow, " + str(gyro[5]) + ")")
225.            prolog.assertz("temp(" + str(temp) + ")")
226.
227.            # angolo vecchio
228.            prolog.assertz("oldangle8(" + str(oldangle) + ")")
229.
230.            ##############################################################################
    ##########################################
231.
232.            ############################
233.            # ESEQUZIONE DELLA QUERY #
234.            ############################
```

```
235.
236.             result = list(prolog.query("command(Result)"))
237.
238.             ##########################################################################
    ########################################
239.
240.             ################################
241.             # DEMOLIZIONE DEI FATTI CARICATI#
242.             ################################
243.
244.             # comando vecchio
245.             prolog.retract("commandolder(" + str(commandolder) + ")")
246.
247.             # qrcode:  0 per false 1 per true
248.             prolog.retract("qrcode(" + str(qrcode) + ")")
249.
250.             # fotocomera
251.             prolog.retract("fotocamera(" + str(fotocamera) + ")")
252.
253.             # switch
254.             prolog.retract("switch(centro, " + str(switch[1]) + ")")
255.             prolog.retract("switch(sinistra, " + str(switch[0]) + ")")
256.             prolog.retract("switch(destra, " + str(switch[2]) + ")")
257.
258.             # sonar: mettere prima il centro cosi se sono 3 sonar uguali il sistema p
    rendera il primo e quindi va avanti
259.             prolog.retract("sonar(centro, " + str(sonar[1]) + ")")
260.             prolog.retract("sonar(sinistra, " + str(sonar[0]) + ")")
261.             prolog.retract("sonar(destra, " + str(sonar[2]) + ")")
262.
263.             # volt
264.             prolog.retract("volt(" + str(volt) + ")")
265.
266.             # compass
267.             prolog.retract("angle16(" + str(angle16) + ")")
268.             prolog.retract("angle8(" + str(angle8) + ")")
269.             prolog.retract("pitch(" + str(pitch) + ")")
270.             prolog.retract("roll(" + str(roll) + ")")
271.             prolog.retract("mag(xhigh, " + str(mag[0]) + ")")
272.             prolog.retract("mag(xlow, " + str(mag[1]) + ")")
273.             prolog.retract("mag(yhigh, " + str(mag[2]) + ")")
274.             prolog.retract("mag(ylow, " + str(mag[3]) + ")")
275.             prolog.retract("mag(zhigh, " + str(mag[4]) + ")")
276.             prolog.retract("mag(zlow, " + str(mag[5]) + ")")
277.             prolog.retract("acc(xhigh, " + str(acc[0]) + ")")
278.             prolog.retract("acc(xlow, " + str(acc[1]) + ")")
279.             prolog.retract("acc(yhigh, " + str(acc[2]) + ")")
280.             prolog.retract("acc(ylow, " + str(acc[3]) + ")")
281.             prolog.retract("acc(zhigh, " + str(acc[4]) + ")")
282.             prolog.retract("acc(zlow, " + str(acc[5]) + ")")
283.             prolog.retract("gyro(xhigh, " + str(gyro[0]) + ")")
284.             prolog.retract("gyro(xlow, " + str(gyro[1]) + ")")
285.             prolog.retract("gyro(yhigh, " + str(gyro[2]) + ")")
286.             prolog.retract("gyro(ylow, " + str(gyro[3]) + ")")
287.             prolog.retract("gyro(zhigh, " + str(gyro[4]) + ")")
288.             prolog.retract("gyro(zlow, " + str(gyro[5]) + ")")
289.             prolog.retract("temp(" + str(temp) + ")")
290.
291.             # angolo vecchio
292.             prolog.retract("oldangle8(" + str(oldangle) + ")")
293.
294.             return result
295.
296.
297.         def callback(msg):
298.             """
```

```
299.            funzione di callback di Ros, effetua le valutazioni del mondo circostante
    e fa uuna scelta
300.            :param msg: messaggio ros dal controller
301.            :return: nulla
302.            """
303.            global commandolder, oldangle, commandIA, qrcode, fotocamera, switch, son
    ar, volt,\
304.                lidar, angle8, angle16, pitch, roll, mag, acc, gyro, temp
305.            qrcode = msg.qrcode
306.            fotocamera = msg.visione
307.            switch = msg.switches
308.            switch[0] = int(switch[0])
309.            switch[1] = int(switch[1])
310.            switch[2] = int(switch[2])
311.            sonare = msg.sonar
312.            sonar = (int(sonare[0]), int(sonare[1]), int(sonare[2]))
313.            volt = msg.volt
314.            lidar = msg.lidar
315.            angle16 = msg.angle16
316.            angle8 = msg.angle8
317.            pitch = msg.pitch
318.            roll = msg.roll
319.            mag = msg.mag
320.            acc = msg.acc
321.            gyro = msg.gyro
322.            temp = msg.temp
323.
324.
325.        def ifNotNone(qrcode, fotocamera, switch, sonar, volt, lidar, angle16, angle8
    , pitch, roll, mag, acc, gyro, temp):
326.            """
327.            Funzione di controlo delle variabili globali,
328.            controlla se sono state modificate tutte
329.            :param qrcode:
330.            :param fotocamera:
331.            :param switch:
332.            :param sonar:
333.            :param volt:
334.            :param lidar:
335.            :param angle16:
336.            :param angle8:
337.            :param pitch:
338.            :param roll:
339.            :param mag:
340.            :param acc:
341.            :param gyro:
342.            :param temp:
343.            :return: True se non sono None e False se sono None
344.            """
345.            return qrcode is not None and fotocamera is not None and switch is not No
    ne and sonar is not None and\
346.                   volt is not None and lidar is not None and angle16 is not None and
     angle8 is not None and\
347.                   pitch is not None and roll is not None and mag is not None and acc
     is not None and\
348.                   gyro is not None and temp is not None
349.
350.
351.        def main():
352.            global commandolder, oldangle, commandIA, qrcode, fotocamera, switch, son
    ar,\
353.                volt, lidar, angle8, angle16, pitch, roll, mag, acc, gyro, temp, lida
    rm
354.            commandIA = ''
355.            prologinit_and_rules()
356.            print("prolog rules")
```

```
357.           rospy.init_node("Prologo_IA_Node", disable_signals=True)
358.           prolog_pub = rospy.Publisher("prolog_ia", PyRobot.Prolog_IA_Node, queue_s
    ize=0)
359.           rospy.Subscriber("controller", PyRobot.Controller_Node, callback)
360.           r = rospy.Rate(1)
361.           while not rospy.is_shutdown():
362.               if commandIA == '':
363.                   if ifNotNone(qrcode, fotocamera, switch, sonar, volt, lidar,
364.                                angle16, angle8, pitch, roll, mag, acc, gyro, temp):

365.
366.                       if commandolder == "correggi_a_destra" or\
367.                               commandolder == "correggi_a_sinistra" or oldangle is
    None:
368.                           oldangle = angle8
369.
370.                       commandIA = prologIA(commandolder, qrcode, fotocamera, switch
    , sonar, volt, lidar,
371.                                            angle16, angle8, pitch, roll, mag, acc,
    gyro, temp, oldangle, lidarm)
372.
373.                       if commandIA == "sinistra" or commandIA == "destra" or comman
    dIA == "indietro":
374.                           oldangle = None
375.
376.                       commandolder = None
377.                       commandolder = commandIA[0]['Result']
378.                       prolog_msg.risposta = commandIA[0]['Result']
379.                       prolog_pub.publish(prolog_msg)
380.                       rospy.loginfo(prolog_msg)
381.                       commandIA = ''
382.                       resetNone()
383.                       r.sleep()
384.
385.
386.       if __name__ == '__main__':
387.           try:
388.               main()
389.           except rospy.ROSInterruptException:
390.               pass
```

NODE LAUNCHER

Programma di avvio dei Nodi

```
1.  <launch>
2.      <node pkg="py_robot" type="Prolog_IA_Node.py" name="Prolog_IA_Node" output="scre
    en">
3.      </node>
4.      <node pkg="py_robot" type="Opencv_Node.py" name="Opencv_Node" output="screen">
5.      </node>
6.  </launch>
```

MESSAGGI

COMPASS NODE MESSAGE

```
int16 angle16
int8 angle8
int16 pitch
int16 roll
int16[6] mag
int16[6] acc
int16[6] gyro
int16 temp
```

CONTROLLER NODE MESSAGE

```
int8 qrcode
string fotocamera
bool[3] switches
float64[3] sonar
float32 volt
int16 angle16
int8 angle8
int16 pitch
int16 roll
int16[6] mag
int16[6] acc
int16[6] gyro
int16 temp
string visione
```

CONTROLLER TO MOTOR NODE MESSAGE

```
string velo
```

MV CAMERA NODE MESSAGE

```
string eureca
```

MOTOR_SWITCH NODE MESSAGE

```
bool[3] switches
```

PI CAMERA NODE MESSAGE

```
string visione
```

PROLOG IA NODE MESSAGE

```
string risposta
```

SONAR_VOLT NODE MESSAGE

```
float64[3] sonar
float32 volt
```

CODICI VREP

NODI

CONTROLLER NODE

```
#! /usr/bin/python
import rospy
import py_robot.msg as PyRobot
import roslaunch.rlutil
import roslaunch.parent
import roslaunch
import time
import os

controller_msg = PyRobot.Controller_Node()
controller_to_lidar_msg = PyRobot.Controller_To_Lidar_Node()
controller_to_motor_msg = PyRobot.Controller_To_Motor_Node()

angle16 = None
angle8 = None
pitch = None
roll = None
mag = [None for x in range(0, 6)]
acc = [None for x in range(0, 6)]
gyro = [None for x in range(0, 6)]
temp = None
lidar18 = [None for x in range(0, 180)]
comando = None
switch = [None for x in range(0, 3)]
sonar = [None for x in range(0, 3)]
volt = None
eureca = None
visione = None
launch = None
risposta_prolog = None
fine = False
backflag = True


def resetvar():
    """
```

```
    Funzione che setta le variabili a nullo per evitare l'invio di messaggi non completi.
    Lavora sulle variabili globali angle16, angle8, pitch, roll, mag, acc, gyro,
    temp, lidar18, comando, switch, sonar, volt, eureca e visione
    """
    global angle16, angle8, pitch, roll, mag, acc, gyro, temp, lidar18, switch, sonar, volt, eureca, visione, comando, risposta_prolog

    angle16 = None
    angle8 = None
    pitch = None
    roll = None
    mag = [None for x in range(0, 6)]
    acc = [None for x in range(0, 6)]
    gyro = [None for x in range(0, 6)]
    temp = None
    lidar18 = [None for x in range(0, 180)]
    comando = None
    switch = [None for x in range(0, 3)]
    sonar = [None for x in range(0, 3)]
    volt = None
    eureca = None
    visione = None
    risposta_prolog = None


def callback_prolog(msg):
    """
    funzione di callback per i messaggi provenienti dal nodo Prolog_IA,
    modifica la variabile globale comando
    :param msg: messagio ricevuto
    :return: nulla
    """
    global comando, fine, risposta_prolog
    comando = ""
    risposta_prolog = msg.risposta
    print("prolog")
    if risposta_prolog == 'avanti':
        comando = "e"
    elif risposta_prolog == 'destra':
        comando = "r"
    elif risposta_prolog == 'sinistra':
        comando = "q"
    elif risposta_prolog == 'indietro':
        comando = "i"
    elif risposta_prolog == 'stop':
        comando = "s"
    elif risposta_prolog == 'correggi_a_destra':
        comando = "t"
    elif risposta_prolog == 'correggi_a_sinistra':
        comando = "u"
    elif risposta_prolog == 'attiva_lidar':
        comando = "v"
    elif risposta_prolog == 'fine':
        comando = 's'
        fine = True
    else:
        comando = 's'


def startfunction():
    """
    Funzione di inizio programma. Si occupa di far partire i nodi
    :return: nulla
    """
            global launch
```

```
            path = os.getcwd()
            uuid = roslaunch.rlutil.get_or_generate_uuid(None, False)
            roslaunch.configure_logging(uuid)
            launch = roslaunch.parent.ROSLaunchParent(uuid, [
                 path + "/catkin_ws/src/py_robot_v_rep/launch/node_launcher.launch"])
    # qui da aggiornare con il path del file launch
            launch.start()
            rospy.loginfo("started")


        def endfunction():
            """
            Funzione di fine programma. Si occupa di far terminare il Nodo Controller

            :return: nulla
            """
            global comando, launch
            launch.shutdown()
            time.sleep(1)
            rospy.signal_shutdown('Mission Complete')


        def callback_switch(msg):
            """
            Funzione Callback che processa i dati inviati dai Nodo Motor_Switch
            :param msg: messaggio ROS dal Nodo Motor_Switch
            """
            global switch
            switch = msg.switches
            print("switch")


        def callback_sonar_volt(msg):
            """
            Funzione Callback che processa i dati inviati dai Nodo Sonar_Volt
            :param msg: messaggio ROS dal Nodo Sonar_Volt
            """
            global sonar, volt
            sonar = msg.sonar
            volt = msg.volt
            print("sonar_volt")


        def callback_lidar_compass(msg):
            """
            Funzione Callback che processa i dati inviati dai Nodo Lidar_Compass
            :param msg: messaggio ROS dal Nodo Lidar_Compass
            """
            global angle16, angle8, pitch, roll, mag, acc, gyro, temp, lidar18
            lidar18 = msg.lidar
            angle16 = msg.angle16
            angle8 = msg.angle8
            pitch = msg.pitch
            roll = msg.roll
            mag = msg.mag
            acc = msg.acc
            gyro = msg.gyro
            temp = msg.temp
            print("lidar_compass")


        def callback_mvcamera(msg):
            """
            Funzione Callback che processa i dati inviati dai Nodo MV_Camera
            :param msg: messaggio ROS dal Nodo MV_Camera
            """
```

```
165.            global eureca
166.            eureca = msg.eureca
167.            print("mv_camera")
168.
169.
170.        def callback_pi_camera(msg):
171.            """
172.            Funzione Callback che processa i dati inviati dai Nodo PI_Camera
173.            :param msg: messaggio ROS dal Nodo PI_Camera
174.            """
175.            global visione
176.            visione = msg.visione
177.            print ("pi_camera")
178.
179.
180.        def ifNotNone(angle16, angle8, pitch, roll, mag, acc, gyro, temp, volt, sonar
    , visione):
181.            """
182.            funzione di controlo delle variabili globali,
183.            controlla se sono state modificate tutte
184.            :param risposta_prolog:
185.            :param visione:
186.            :param angle16: misura del angolo a 16
187.            :param angle8: misura del angolo a 8
188.            :param pitch: misura del pitch
189.            :param roll: misura del roll
190.            :param mag: misura del mag
191.            :param acc: misura del acc
192.            :param gyro: misura del gyro
193.            :param temp: misura della temperatura
194.            :param comando: variabile del comando da inviare
195.            :param volt: misura della batteria
196.            :param sonar: misura dei sonar
197.            :return: True se sono modificate, False se sono ancora None
198.            """
199.            return angle16 is not None and angle8 is not None and pitch is not None a
    nd roll is not None and mag is not None and acc is not None and gyro is not None and
     temp is not None and volt is not None and sonar is not None and visione is not None

200.
201.
202.        def main():
203.            """
204.            Funzione principale che si occupa di inizializzare il Nodo, i Pubblisher
    e i Subscriber. Si occupa anche di salvare
205.            i valori ricevuti dai vari Nodi nei mesaggi Controller_Node.msg e Control
    ler_To_Lidar.msg.
206.            """
207.            global angle16, angle8, pitch, roll, mag, acc, gyro, temp, lidar18, switc
    h, sonar, volt, eureca, visione, comando
208.            global fine, risposta_prolog, backflag
209.            resetvar()
210.            # inizializzazione nodo Controller
211.            rospy.init_node("Controller_Node", disable_signals=True)
212.            startfunction()
213.            # inizializzazioni Publisher e Subsciber
214.            controller_pub = rospy.Publisher("controller", PyRobot.Controller_Node, q
    ueue_size=1)
215.            controller_to_lidar_pub = rospy.Publisher("controller_To_Lidar", PyRobot.
    Controller_To_Lidar_Node, queue_size=1)
216.            controller_to_motor_pub = rospy.Publisher("controller_To_Motor", PyRobot.
    Controller_To_Motor_Node, queue_size=1)
217.            rospy.Subscriber("prolog_ia", PyRobot.Prolog_IA_Node, callback_prolog)
218.            rospy.Subscriber("switches", PyRobot.Motor_Switch_Node, callback_switch)
```

```
219.           rospy.Subscriber("sonar_volt", PyRobot.Sonar_Volt_Node, callback_sonar_vo
    lt)
220.           rospy.Subscriber("mv_camera", PyRobot.MV_Camera_Node, callback_mvcamera)

221.           rospy.Subscriber("lidar_compass", PyRobot.Lidar_Compass_Node, callback_li
    dar_compass)
222.           rospy.Subscriber("pi_camera", PyRobot.Pi_Camera_Node, callback_pi_camera)

223.           r = rospy.Rate(0.5)
224.           while not rospy.is_shutdown():
225.               if ifNotNone(angle16, angle8, pitch, roll, mag, acc, gyro, temp, volt
    , sonar, visione):
226.                   print (switch)
227.                   controller_msg.lidar = lidar18  # messaggio per il nodo Prolog pe
    r distanze lidar
228.                   controller_msg.angle16 = angle16  # messaggio per il nodo Prolog
    per angle16
229.                   controller_msg.angle8 = angle8  # messaggio per il nodo Prolog pe
    r angle8
230.                   controller_msg.pitch = pitch  # messaggio per il nodo Prolog per
    pitch
231.                   controller_msg.roll = roll  # messaggio per il nodo Prolog per ro
    ll
232.                   controller_msg.mag = mag  # messaggio per il nodo Prolog per mag

233.                   controller_msg.acc = acc  # messaggio per il nodo Prolog per acc

234.                   controller_msg.gyro = gyro  # messaggio per il nodo Prolog per gy
    ro
235.                   controller_msg.temp = temp  # messaggio per il nodo Prolog per te
    mp
236.                   controller_msg.switches = switch  # messaggio per il nodo Prolog
    per switch
237.                   controller_msg.volt = volt  # messaggio per il nodo Prolog per vo
    lt
238.                   controller_msg.sonar = sonar  # messaggio per il nodo Prolog per
    sonar
239.                   controller_msg.visione = visione  # messaggio per il nodo Prolog
    per Py Camera
240.
241.                   if eureca == 'trovato':
242.                       controller_msg.qrcode = 1  # messaggio per il nodo Prolog per
     qrcode
243.
244.                   if risposta_prolog != 'attiva_lidar':
245.                       controller_to_lidar_msg.on_off_lidar = False
246.                       backflag = True
247.
248.                   if risposta_prolog == 'attiva_lidar' and backflag:  # messaggio p
    er il Nodo  Compass_Servo_Lidar
249.                       controller_to_lidar_msg.on_off_lidar = True
250.                       backflag = False
251.
252.                   if comando is not None:
253.                       controller_to_motor_msg.velo = comando  # messaggio per il no
    do Motor_Switch per Motor
254.
255.                   # funzioni Publish ROS
256.                   controller_pub.publish(controller_msg)
257.                   controller_to_lidar_pub.publish(controller_to_lidar_msg)
258.                   controller_to_motor_pub.publish(controller_to_motor_msg)
259.                   rospy.loginfo(controller_msg)
260.                   rospy.loginfo(controller_to_lidar_msg)
261.                   rospy.loginfo(controller_to_motor_msg)
262.                   if fine:
263.                       time.sleep(2)
```

```
                        endfunction()
                    resetvar()
                    r.sleep()


        if __name__ == '__main__':
            try:
                main()
            except rospy.ROSInterruptException:
                pass
```

LIDAR_COMPASS NODE

```
#! /usr/bin/python

from librerie import *
import numpy as np
import time
import rospy
import py_robot_v_rep.msg as PyRobot

oggetto = []
lidar_compass_msg = PyRobot.Lidar_Compass_Node()
lidarvar = [0.] * 18
clientID = None
handle = None
flagtrue = False


def connessione():
    """
    funzione per connessione
    :return: clientID
    """
    print('Program started')
    simxFinish(-1)  # just in case, close all opened connections
    clientID = simxStart('127.0.0.1', 19998, True, True, 5000, 5)  # Connect to V-
REP
    if clientID == -1:
        print("Failed to connect to Remote API Server")
    else:
        print('Connected to Remote API Server')
        return clientID


def oggetti(clientID):
    """
    funzione che crea l'oggetto lidar nel modo v-rep
    :var globale oggetto: rappresenta l'oggetto lidar, e' di tipo lista
    :param clientID: connesione v-rep
    :return: oggetto
    """
    global oggetto
    nomioggetti = ['Lidar']
```

```
    res, objecthandle = simxGetObjectHandle(clientID, nomioggetti[0], simx_opmode_blocking)
    simxReadProximitySensor(clientID, objecthandle, simx_opmode_streaming)
    simxGetObjectOrientation(clientID, objecthandle, sim_handle_parent, simx_opmode_streaming)
    if not res == 0:
        print ("Creation Error")
        return
    else:
        print ("Creation " + nomioggetti[0])
        oggetto.append(objecthandle)
    return oggetto


def readproximity(clientID, handle):
    """
    funzione che ritorna una lettura, la lettura in questo caso  e' del lidar
    :param clientID: connessione v-rep
    :param handle: oggetto dal quale provine la lettura
    :return ditsanza: distanza letta
    """
    ris, stato, coordinate, handleoggettoris, vettorenormalizzato = simxReadProximitySensor(clientID, handle,

        simx_opmode_buffer)
    if ris > 0:
        print ("Error read proximity")
    distanza = np.linalg.norm(coordinate)
    return distanza


def angololidar(clientID, lidarhandle):
    """
    funzione che ci dice l' orientamento del oggetto passato rispetto al padre nel mondo v-rep,
    in quessto caso  e' l'orientamento del lidar rispetto il padre (corpo del rover)

    :param clientID: connessione v-rep
    :param handle: oggetto a qui misurare l' orientamento rispetto al padre
    :return number: lista che contiene la rotazione rispetto ai 3 assi (misura in gradi deg)
    """
    returnCode, number = simxGetObjectOrientation(clientID, lidarhandle, sim_handle_parent, simx_opmode_buffer)
    if returnCode > 0:
        print ("Error angololidar orientation")
    number = [number[0] * 180 / np.pi, number[1] * 180 / np.pi, number[2] * 180 / np.pi]
    return number


def lidar(clientID, lidarhandle, corangolo):
    """
    funzione che esegue la lettura e lo spostamento del lidar
    :param clientID: connesione v-rep
    :param lidar: oggetto lidar
    :param corangolo: angolo di spostamento/correzione del lidar per la prossima posizione
    :return: distanza misuarata dal lidar
    """
    angolo = angololidar(clientID, lidarhandle)
    angoloeleur = [(angolo[0] + corangolo) / 180 * np.pi, (angolo[1]) / 180 * np.pi, angolo[2] / 180 * np.pi]
    ris = simxSetObjectOrientation(clientID, lidarhandle, sim_handle_parent, angoloeleur, simx_opmode_oneshot_wait)
    if ris > 0:
```

```
        print ("Error lidar orientation")
    distanza = readproximity(clientID, lidarhandle)
    if distanza < 0.1:
        distanza = 0
    time.sleep(0.1)
            return distanza


       def callback(msg):
            """
            funzione chiamata dalla sottoscrizione del nodo di controllo, effettua le letture del lidar,
            e ripristina la posizione del lidar
            :param msg: messaggio ros ricevuto
            :return: nulla
            """
            global lidarvar, clientID, handle, flagtrue
            flagtrue = True
            time.sleep(1)
            if msg.on_off_lidar:
                angolo_original = angololidar(clientID, handle[0])
                angoloeleur_original = [(angolo_original[0]) / 180 * np.pi, (angolo_original[1]) / 180 * np.pi,
                                        angolo_original[2] / 180 * np.pi]
                lidarvar[0] = float(lidar(clientID, handle[0], +89) * 100)
                for i in range(1, 17):
                    lidarvar[i] = float(lidar(clientID, handle[0], -10.0) * 100)
                simxSetObjectOrientation(clientID, handle[0], sim_handle_parent, angoloeleur_original,
                                         simx_opmode_oneshot_wait)


       def main():
            global oggetto, clientID, handle, lidarvar, flagtrue
            clientID = connessione()
            handle = oggetti(clientID)
            time.sleep(1)
            rospy.init_node("Lidar_Compass_Node")
            rospy.Subscriber("controller_To_Lidar", PyRobot.Controller_To_Lidar_Node, callback)
            lidar_pub = rospy.Publisher("lidar_compass", PyRobot.Lidar_Compass_Node, queue_size=0)
            r = rospy.Rate(1)
            while not rospy.is_shutdown():
                if flagtrue:
                    lidar_compass_msg.lidar = lidarvar
                    lidarvar = [0.] * 18
                    flagtrue = False
                lidar_compass_msg.angle16 = int(0)
                lidar_compass_msg.angle8 = int(0)
                lidar_compass_msg.pitch = int(0)
                lidar_compass_msg.roll = int(0)
                lidar_compass_msg.mag = [int(0), int(0), int(0), int(0), int(0), int(0)]
                lidar_compass_msg.acc = [int(0), int(0), int(0), int(0), int(0), int(0)]
                lidar_compass_msg.gyro = [int(0), int(0), int(0), int(0), int(0), int(0)]
                lidar_compass_msg.temp = int(24)
                lidar_pub.publish(lidar_compass_msg)
                rospy.loginfo(lidar_compass_msg)
                r.sleep()


       if __name__ == '__main__':
            try:
```

```
153.             main()
154.         except rospy.ROSInterruptException:
155.             pass
```

MOTOR_SWITCH NODE

```
1.  #! /usr/bin/python
2.
3.  from librerie import *
4.  import numpy as np
5.  import py_robot_v_rep.msg as PyRobot
6.  import time
7.  import rospy
8.
9.
10. switch_msg = PyRobot.Motor_Switch_Node()
11. clientID = None
12. oggetto = []
13. micros = []
14. motors = []
15. stati = []
16.
17.
18. def connessione():
19.     """
20.     funzione per connessione
21.     :return: clientID
22.     """
23.     print('Program started')
24.     simxFinish(-1)  # just in case, close all opened connections
25.     clientID = simxStart('127.0.0.1', 20000, True, True, 5000, 5)  # Connect to V-
    REP
26.     if clientID == -1:
27.         print("Failed to connect to Remote API Server at Port 20000")
28.     else:
29.         print('Connected to Remote API Server at Port 20000')
30.         return clientID
31.
32.
33. def oggetti(clientID):
34.     """
35.     funzione che crea gli oggetti nel modo v-rep
36.     :var globale oggetto: rappresenta gli oggetti creati, e' di tipo lista
37.     :param clientID: connesione v-rep
38.     :return: oggetto
39.     """
40.     global oggetto
41.     nomioggetti = ['Micro_SX', 'Micro_DX', 'Micro_CE', 'Motore_AD', 'Motore_AS', 'Mo
    tore_PD', 'Motore_PS']
42.     for i in range(0, 7):
43.         res, objecthandle = simxGetObjectHandle(clientID, nomioggetti[i], simx_opmod
    e_blocking)
44.         if i < 4:
45.             simxReadProximitySensor(clientID, objecthandle, simx_opmode_streaming)
46.         if not res == 0:
47.             print ("Creation Error")
48.             return
49.         else:
50.             print ("Creation " + nomioggetti[i])
51.             oggetto.append(objecthandle)
52.     return oggetto
53.
```

```
54.
55. def forward(clientID, motors, velocita):
56.     """
57.     funzione che setta i motori per andare avanti
58.     :param clientID: connesione v-rep
59.     :param motors: lista degli oggetti motori del mondo v-rep
60.     :param velocita: velocita' da settare ai motori
61.     :return: nulla
62.     """
63.     ris1 = simxSetJointTargetVelocity(clientID, motors[3], velocita, simx_opmode_one
    shot_wait)
64.     ris2 = simxSetJointTargetVelocity(clientID, motors[1], velocita, simx_opmode_one
    shot_wait)
65.     ris3 = simxSetJointTargetVelocity(clientID, motors[2], velocita, simx_opmode_one
    shot_wait)
66.     ris4 = simxSetJointTargetVelocity(clientID, motors[0], velocita, simx_opmode_one
    shot_wait)
67.     if ris1 > 0 or ris2 > 0 or ris3 > 0 or ris4 > 0:
68.         print ("Error forward")
69.
70.
71. def backward(clientID, motors, velocita):
72.     """
73.     funzione che setta i motori per andare indietro
74.     :param clientID: connesione v-rep
75.     :param motors: lista degli oggetti motori del mondo v-rep
76.     :param velocita: velocita' da settare ai motori
77.     :return: nulla
78.     """
79.     ris1 = simxSetJointTargetVelocity(clientID, motors[3], -
    velocita, simx_opmode_oneshot_wait)
80.     ris2 = simxSetJointTargetVelocity(clientID, motors[1], -
    velocita, simx_opmode_oneshot_wait)
81.     ris3 = simxSetJointTargetVelocity(clientID, motors[2], -
    velocita, simx_opmode_oneshot_wait)
82.     ris4 = simxSetJointTargetVelocity(clientID, motors[0], -
    velocita, simx_opmode_oneshot_wait)
83.     if ris1 > 0 or ris2 > 0 or ris3 > 0 or ris4 > 0:
84.         print ("Error backward")
85.
86.
87. def left(clientID, motors):
88.     """
89.     funzione che setta i motori per andare a sinistra
90.     :param clientID: connesione v-rep
91.     :param motors: lista degli oggetti motori del mondo v-rep
92.     :return: nulla
93.     """
94.     ris1 = simxSetJointTargetVelocity(clientID, motors[0], +0.5, simx_opmode_oneshot
    _wait)
95.     ris2 = simxSetJointTargetVelocity(clientID, motors[1], -
    0.5, simx_opmode_oneshot_wait)
96.     ris3 = simxSetJointTargetVelocity(clientID, motors[2], +0.5, simx_opmode_oneshot
    _wait)
97.     ris4 = simxSetJointTargetVelocity(clientID, motors[3], -
    0.5, simx_opmode_oneshot_wait)
98.     if ris1 > 0 or ris2 > 0 or ris3 > 0 or ris4 > 0:
99.         print ("Error left")
100.
101.        def left_correction(clientID, motors):
102.            """
103.            funzione che setta i motori per eseguire una correzione a sinistra
104.            :param clientID: connesione v-rep
105.            :param motors: lista degli oggetti motori del mondo v-rep
106.            :return: nulla
107.            """
```

```
108.            ris1 = simxSetJointTargetVelocity(clientID, motors[0], +2, simx_opmode_on
    eshot_wait)
109.            ris2 = simxSetJointTargetVelocity(clientID, motors[1], -
    2, simx_opmode_oneshot_wait)
110.            ris3 = simxSetJointTargetVelocity(clientID, motors[2], +2, simx_opmode_on
    eshot_wait)
111.            ris4 = simxSetJointTargetVelocity(clientID, motors[3], -
    2, simx_opmode_oneshot_wait)
112.            if ris1 > 0 or ris2 > 0 or ris3 > 0 or ris4 > 0:
113.                print ("Error left_correction")
114.
115.
116.        def right(clientID, motors):
117.            """
118.            funzione che setta i motori per andare a destra
119.            :param clientID: connesione v-rep
120.            :param motors: lista degli oggetti motori del mondo v-rep
121.            :return: nulla
122.            """
123.            ris1 = simxSetJointTargetVelocity(clientID, motors[0], -
    0.5, simx_opmode_oneshot_wait)
124.            ris2 = simxSetJointTargetVelocity(clientID, motors[1], +0.5, simx_opmode_
    oneshot_wait)
125.            ris3 = simxSetJointTargetVelocity(clientID, motors[2], -
    0.5, simx_opmode_oneshot_wait)
126.            ris4 = simxSetJointTargetVelocity(clientID, motors[3], +0.5, simx_opmode_
    oneshot_wait)
127.            if ris1 > 0 or ris2 > 0 or ris3 > 0 or ris4 > 0:
128.                print ("Error right")
129.
130.        def right_correction(clientID, motors):
131.            """
132.            funzione che setta i motori per eseguire una correzione a destra
133.            :param clientID: connesione v-rep
134.            :param motors: lista degli oggetti motori del mondo v-rep
135.            :return: nulla
136.            """
137.            ris1 = simxSetJointTargetVelocity(clientID, motors[0], -
    2, simx_opmode_oneshot_wait)
138.            ris2 = simxSetJointTargetVelocity(clientID, motors[1], +2, simx_opmode_on
    eshot_wait)
139.            ris3 = simxSetJointTargetVelocity(clientID, motors[2], -
    2, simx_opmode_oneshot_wait)
140.            ris4 = simxSetJointTargetVelocity(clientID, motors[3], +2, simx_opmode_on
    eshot_wait)
141.            if ris1 > 0 or ris2 > 0 or ris3 > 0 or ris4 > 0:
142.                print ("Error right_correction")
143.
144.        def micro(clientID):
145.            """
146.            funzione che rileva lo stato dei microswitch nel modo r-rep
147.            :param clientID: connesione v-rep
148.            :return stati: ritorna gli stati dei microswitch e' di tipo lista
149.            """
150.            global micros, stati
151.            for Micro in micros:
152.                ris, stato, coordinate, handleoggettoris, vettorenormalizzato = simxR
    eadProximitySensor(clientID, Micro, simx_opmode_buffer)
153.                stati.append(stato)
154.            return stati
155.
156.
157.        def callback(msg):
158.            """
159.            funzione che esegue i comandi che arrivano dal controller_node
160.            :param msg: messaggio ros ricevuto
```

```
161.            :return: nulla
162.            """
163.            global clientID, motors
164.            print msg.velo
165.            if msg.velo == "a":
166.                forward(clientID, motors, 1)
167.            elif msg.velo == "b":
168.                forward(clientID, motors, 1.5)
169.            elif msg.velo == "c":
170.                forward(clientID, motors, 2)
171.            elif msg.velo == "d":
172.                forward(clientID, motors, 2.5)
173.
174.            # dritto
175.            elif msg.velo == "e":
176.                forward(clientID, motors, 1.5)
177.
178.            elif msg.velo == "f":
179.                forward(clientID, motors, 3.5)
180.            elif msg.velo == "g":
181.                forward(clientID, motors, 4)
182.            elif msg.velo == "h":
183.                forward(clientID, motors, 5)
184.
185.            # indietro
186.            elif msg.velo == "i":
187.                backward(clientID, motors, 1)
188.
189.            elif msg.velo == "l":
190.                backward(clientID, motors, 1.5)
191.            elif msg.velo == "m":
192.                backward(clientID, motors, 2)
193.            elif msg.velo == "n":
194.                backward(clientID, motors, 2.5)
195.            elif msg.velo == "o":
196.                backward(clientID, motors, 3)
197.            elif msg.velo == "p":
198.                left(clientID, motors)
199.                time.sleep(2)
200.                forward(clientID, motors, 0)
201.
202.            # sinistra
203.            elif msg.velo == "q":
204.                left(clientID, motors)
205.                time.sleep(2)
206.                forward(clientID, motors, 0)
207.
208.            # destra
209.            elif msg.velo == "r":
210.                right(clientID, motors)
211.                time.sleep(2)
212.                forward(clientID, motors, 0)
213.
214.            # stop
215.            elif msg.velo == "s":
216.                forward(clientID, motors, 0)
217.
218.            # correggi a destra
219.            elif msg.velo == "t":
220.                left_correction(clientID, motors)
221.
222.            # correggi a sinistra
223.            elif msg.velo == "u":
224.                right_correction(clientID, motors)
225.
226.            # attiva lidar
```

```
            elif msg.velo == "v":
                backward(clientID, motors, 2)
                time.sleep(0.5)
                backward(clientID, motors, 0)

            else:
                forward(clientID, motors, 0)


        def main():
            global clientID, oggetto, micros, motors, stati
            clientID = connessione()
            newoggetti = oggetti(clientID)
            micros = [newoggetti[0], newoggetti[1], newoggetti[2]]
            motors = [newoggetti[3], newoggetti[4], newoggetti[5], newoggetti[6]]
            rospy.init_node("Motor_Switch_Node")
            rospy.Subscriber("controller_To_Motor", PyRobot.Controller_To_Motor_Node, callback)
            switch_pub = rospy.Publisher("switches", PyRobot.Motor_Switch_Node, queue_size=0)
            r = rospy.Rate(3)

            while not rospy.is_shutdown():
                microswitch = micro(clientID)
                if microswitch[0] or microswitch[1] or microswitch[2]:
                    forward(clientID, motors, 0)
                switch_msg.switches = microswitch
                switch_pub.publish(switch_msg)
                rospy.loginfo(switch_msg)
                stati = []
                r.sleep()


        if __name__ == '__main__':
            try:
                main()
            except rospy.ROSInterruptException:
                pass
```

OPENCV NODE

```
#! /usr/bin/python

from librerie import *
import cv2
import numpy as np
import time
import py_robot.msg as PyRobot
import rospy

clientID = None
oggetto = []
img = None
ris1 = None
ris = None
pi_camera_msg = PyRobot.Pi_Camera_Node()


def connessione():
    """
    funzione per connessione
    :return: clientID
```

```
    """
    print('Program started')
    simxFinish(-1)  # just in case, close all opened connections
    clientID = simxStart('127.0.0.1', 20001, True, True, 5000, 5)  # Connect to V-REP
    if clientID == -1:
        print("Failed to connect to Remote API Server")
    else:
        print('Connected to Remote API Server')
        return clientID


def oggetti(clientID):
    """
    funzione che crea l'oggetto Pi_Camera nel modo v-rep
    :var globale oggetto: rappresenta l'oggetto Pi_Camera, e' di tipo lista
    :param clientID: connesione v-rep
    :return: oggetto
    """
    global oggetto
    nomioggetti = ['Pi_Camera']
    res, objecthandle = simxGetObjectHandle(clientID, 'Pi_Camera', simx_opmode_oneshot_wait)
    err, resolution, image = simxGetVisionSensorImage(clientID, objecthandle, 0, simx_opmode_streaming)
    time.sleep(1)
    if not res == 0:
        print ("Creation Error")
        return
    else:
        print ("Creation " + nomioggetti[0])
        oggetto.append(objecthandle)
    return oggetto


def immagine(clientID):
    """
    funzione che serve per prendere l'immagine da vrep
    :param clientID:
    :return: img immagine
    """
    global oggetto, img
    err, resolution, image = simxGetVisionSensorImage(clientID, oggetto[0], 0, simx_opmode_buffer)
    if not err == 0:
        print ("Image Received")
    else:
        print ("ERROR IMAGE")
    img = np.array(image, dtype=np.uint8)
    img.resize([resolution[0], resolution[1], 3])
    img = np.rot90(img, 2)
    img = np.fliplr(img)
    return img


def imgs(image):
    """
    funzione che elabora l'immagine ricevuta, ritorna comando
    :param image: immagine del mondo v-rep
    :return: string command
    """
    global ris1, ris
    ris = cv2.Canny(image, 100, 200)
    misura = np.array([])
    height, width = ris.shape
    print (height, width)
```

```
    ris1 = ris.copy()
    for x in range(width - 1):
        for h in range(height - 1):
            hei = height - h - 1
            if ris1[hei, x] == 255:
                misura = np.append(misura, int(hei))
                break
            else:
                ris1.itemset((hei, x), 50)
    misura = np.split(misura, [150, 362])
    media_sinistra = np.median(misura[0])
    media_centrale = np.median(misura[1])
    media_destra = np.median(misura[2])
    print ("MEDIANS ")
    print(media_sinistra, media_centrale, media_destra)
    print ("COMMAND ")
            if media_sinistra < media_centrale:
                if (media_sinistra < media_destra):
                    print("vai a sinistra: ", media_sinistra)
                    return "sinistra"
                else:
                    print("vai a destra: ", media_destra)
                    return "destra"
            elif media_centrale < media_destra:
                print("continua dritto: ", media_centrale)
                return "dritto"
            else:
                print("vai a destra: ", media_destra)
                return "destra"


        def main():
            global clientID, oggetto, img, ris1, ris
            clientID = connessione()
            oggetto = oggetti(clientID)
            rospy.init_node("Pi_Camera_Node", disable_signals=True)
            pi_camera_pub = rospy.Publisher("pi_camera", PyRobot.Pi_Camera_Node, queue_size=0)
            r = rospy.Rate(0.5)
            while not rospy.is_shutdown():
                pi_camera_msg.visione = imgs(immagine(clientID))
                pi_camera_pub.publish(pi_camera_msg)
                rospy.loginfo(pi_camera_msg)
                r.sleep()
            # while (1):
            #     cv2.imshow('img', img)
            #     cv2.imshow('ris', ris)
            #     cv2.imshow('ris1', ris1)
            #     k = cv2.waitKey(5) & 0xFF
            #     if k == 27:
            #         break


        if __name__ == '__main__':
            try:
                main()
            except rospy.ROSInterruptException:
                pass
```

PROLOG_IA NODE

```
#! /usr/bin/python
```

```
2.
3.  import pyswip.prolog
4.  import py_robot.msg as PyRobot
5.  import rospy
6.  import time
7.  import numpy as np
8.
9.  prolog_msg = PyRobot.Prolog_IA_Node()
10.
11. prolog = None
12. commandolder = "avanti"
13. oldangle = None
14.
15.
16. # var global
17. qrcode = None
18. fotocamera = None
19. switch = None
20. sonar = None
21. volt = None
22. lidar = None
23. angle16 = None
24. angle8 = None
25. pitch = None
26. roll = None
27. mag = None
28. acc = None
29. gyro = None
30. temp = None
31. lidarm = None
32.
33.
34. def resetNone():
35.     """
36.     Funzione per il reset delle variabili ambientali
37.     :return:
38.     """
39.     global qrcode, fotocamera, switch, sonar, volt, lidar, angle16, angle8, pitch, r
    oll, mag, acc, gyro, temp
40.     qrcode = None
41.     fotocamera = None
42.     switch = None
43.     sonar = None
44.     volt = None
45.     lidar = None
46.     angle16 = None
47.     angle8 = None
48.     pitch = None
49.     roll = None
50.     mag = None
51.     acc = None
52.     gyro = None
53.     temp = None
54.
55.
56. def prologinit_and_rules():
57.     """
58.     Funzione per l'inizializzazione di prolog e la creazione delle regole
59.     :return: nulla
60.     """
61.     global prolog
62.
63.     ############################
64.     # INIZIALIZZAZIONE PROLOG #
65.     ############################
66.
```

```
    prolog = pyswip.Prolog()

    ############################
    # CREAZIONE COMANDI ROVER #
    ############################

    # comandi disponibili
    prolog.assertz("command(avanti, 1)")
    prolog.assertz("command(sinistra, 2)")
    prolog.assertz("command(destra, 3)")
    prolog.assertz("command(indietro,4)")
    prolog.assertz("command(stop, 5)")
    prolog.assertz("command(fine, 6)")
    prolog.assertz("command(batteria, 7)")
    prolog.assertz("command(correggi_a_destra, 8)")
    prolog.assertz("command( correggi_a_sinistra, 9)")
    prolog.assertz("command(attiva_lidar, 10)")

    ############################
    # CREAZIONE REGOLE GENERALI #
    ############################

    # regola per il lidar ritorna l'angolo con la distanza maggiore
    prolog.assertz("lidarmax(X):- lidar(0,A),lidar(10,B),lidar(20,C),lidar(30,D),lidar(40,E),lidar(50,F),lidar(60,G),"
                   "lidar(70,H),lidar(80,I),lidar(90,L),lidar(100,M),lidar(110,N),lidar(120,O),lidar(130,P),"
                   "lidar(140,Q),lidar(150,R),lidar(160,S),lidar(170,T), V is max(A,B), V1 is max(V,C),"
                   "V2 is max(V1,D),V3 is max(V2,E),V4 is max(V3,F),V5 is max(V4,G), V6 is max(V5,H),"
                   " V7 is max(V6,I), V8 is max(V7,L), V9 is max(V8,M), V10 is max(V9,N), V11 is max(V10,O),"
                   " V12 is max(V11,P), V13  is max(V12,Q), V14 is max(V13,R), V15 is max(V14,S), V16 is max(V15,T),"
                   " lidar(X,V16),!")

    prolog.assertz("lidar(P):- lidarmedia(sinistra,X), lidarmedia(destra,Y), R is max(X,Y), lidarmedia(P,R),!")

            # regola per gli switch false se sbatte true se non sbatte
            prolog.assertz("switch(_):- switch(sinistra,X), switch(centro,Y), switch(destra, Z), X == 0, Y == 0, Z == 0, !")

            # regola per il cambio direzione
            prolog.assertz("distancetrue(_):- sonar(destra, A), sonar(centro, B), sonar(sinistra, C),  A > 50, B > 50, C > 50, !")

            # regola per il cambio direzione (indietro) a una certa distanza da un possibile ostacolo troppo vicino
            prolog.assertz("sonartrue(_):- sonar(destra, A), sonar(centro, B), sonar(sinistra, C),  A > 30, B > 30, C > 30, !")

            # regola per capire quale e' il sonar con la distanza maggiore
            prolog.assertz("sonar(Y) :- sonar(centro, A), sonar(destra, B), sonar(sinistra, C), D is max(A, B),"
                           " X is max(D, C), sonar(Y, X),!")

            # regola per capire quale e' il sonar tra destra e sinistra maggiore
            prolog.assertz("sonardxsx(Y):- sonar(destra, A), sonar(sinistra, B), X is max(A, B), sonar(Y, X), Y \== centro, !")

            # regola per le condizioni necessarie per il cambio di qualasiasi direzione
            prolog.assertz("condictrue(_):- switch(_), volt(Y), Y>11, qrcode(Q), Q == 0, !")
```

```
118.
119.           # commando batteria
120.           prolog.assertz("command(X):- volt(Y), Y < 11, C is 7, command(X, C),!")
121.
122.           # comando fine
123.           prolog.assertz("command(X):- qrcode(1), C is 6, command(X,C),!")
124.
125.           # comando attiva lidar
126.           prolog.assertz("command(X):- commandolder(O), O \== indietro, switch(_,1)
    , volt(Y), Y > 11,"
127.                          " qrcode(Q), Q == 0, command(X,10),!")
128.
129.           # comando coregi a destra
130.           prolog.assertz("command(X):-
    commandolder(O), O == avanti, condictrue(_), distancetrue(_), angle8(A), oldangle8(B
    ),"
131.                          " S is A-B, S > 10, command(X, 8),!")
132.
133.           # comando coregi a sinistra
134.           prolog.assertz("command(X):-
    commandolder(O), O == avanti, condictrue(_), distancetrue(_), angle8(A), oldangle8(B
    ),"
135.                          " S is B-A, S > 10, command(X, 9),!")
136.
137.           # comando indietro
138.           prolog.assertz("command(X):- commandolder(O), O \== indietro, \+ sonartru
    e(_), volt(Y), Y > 11,"
139.                          " qrcode(Q), Q == 0, command(X,4),!")
140.
141.           # comando avanti
142.           prolog.assertz("command(X):- commandolder(O), O == avanti, condictrue(_),
     distancetrue(_), command(X,1), !")
143.           prolog.assertz("command(X):- commandolder(O), O \== indietro, condictrue(
    _), distancetrue(_), sonar(U), "
144.                          "U == centro, fotocamera(F), F == centro, command(X,1), !"
    )
145.           prolog.assertz("command(X):- commandolder(O), O \== indietro, condictrue(
    _), distancetrue(_),"
146.                          " sonar(U), U == centro, command(X,1), !")
147.           prolog.assertz("command(X):- commandolder(O), O \== indietro, condictrue(
    _), distancetrue(_), command(X,1), !")
148.
149.           # comando sinistra
150.           prolog.assertz("command(X):- condictrue(_), sonartrue(_), sonardxsx(U), U
     == sinistra, fotocamera(F),"
151.                          " F == sinistra, command(X,2),!")
152.           prolog.assertz("command(X):- condictrue(_), sonartrue(_), sonardxsx(U), U
     == sinistra, command(X,2),!")
153.
154.           # comando destra
155.           prolog.assertz("command(X):- condictrue(_), sonartrue(_), sonardxsx(U), U
     == destra, fotocamera(F),"
156.                          " F == destra, command(X,3), !")
157.           prolog.assertz("command(X):- condictrue(_), sonartrue(_), sonardxsx(U), U
     == destra, command(X,3) ,!")
158.
159.           # comando indietro che viene eseguito quando non c'e' spazio per girare a
     destra o a sinista
160.           prolog.assertz("command(X):- \+ sonartrue(_), volt(Y), Y > 11, qrcode(Q),
     Q == 0, command(X,4),!")
161.
162.           # comando stop (errore)
163.           prolog.assertz("command(X):- command(X,5),!")
164.
165.           ################################################
166.           # CREAZIONE REGOLE PER POST ATTIVAZIONE LIDAR #
```

```
            ##############################################

            # commando batteria
            prolog.assertz("commandlidar(X):- volt(Y), Y < 11, C is 7, command(X, C),
    !")

            # comando fine
            prolog.assertz("commandlidar(X):- qrcode(1), C is 6, command(X,C),!")

            # comando destra
            prolog.assertz("commandlidar(X):-
    lidar(B), B == destra, volt(Y), Y>11, qrcode(Q), Q == 0, command(X,3),!")

            # comando sinistra
            prolog.assertz("commandlidar(X):-
    lidar(B), B == sinistra, volt(Y), Y>11, qrcode(Q), Q == 0, command(X,2),!")

            # comando attiva_lidar in caso in qui gli switch siano ancora attivi
            prolog.assertz("commandlidar(X):-
    switch(_,1), volt(Y), Y > 11, qrcode(Q), Q == 0, command(X,10),!")

            # comando stop (errore)
            prolog.assertz("commandlidar(X):- command(X,5),!")


        def prologIA(commandolder, qrcode, fotocamera, switch, sonar, volt, lidar,
                     angle16, angle8, pitch, roll, mag, acc, gyro, temp, oldangle, li
    darm):
            """
            Funzione per il caricamento dei fatti, per esecuzione delle query e della
     demolizione dei fatti caricati
            :param commandolder:  vecchio comando di default e' avanti
            :param qrcode: 0 per non trovato 1 per trovato
            :param fotocamera: vale sinistra o destra o centro e identifica lo spazio
     libero
            :param switch: 0 se non attivo 1 per attivo e' un arry di 3 elementi
            :param sonar: vale la misurazione effettuata e' un array di 3 elementi
            :param volt: vale la misurazione effettuata
            :param lidar: vale le misurazioni effettuate e' un array di 19 elementi
            :param angle16: vale la misurazione effettuata
            :param angle8: vale la misurazione effettuata
            :param pitch: vale la misurazione effettuata
            :param roll: vale la misurazione effettuata
            :param mag: vale le misurazioni effettuate e' un array di 6 elementi
            :param acc: vale le misurazioni effettuate e' un array di 6 elementi
            :param gyro: vale le misurazioni effettuate e' un array di 6 elementi
            :param temp: vale la misurazione effettuata
            :param oldangle: vale un angolo iniziale
            :return: resultato della query
            """
            global prolog

            ####################
            # CREAZIONE FATTI #
            ####################

            # comando vecchio
            prolog.assertz("commandolder(" + str(commandolder) + ")")

            # qrcode:  0 per false 1 per true
            prolog.assertz("qrcode(" + str(qrcode) + ")")

            # fotocomera
            prolog.assertz("fotocamera(" + str(fotocamera) + ")")

            # switch
```

```
226.            prolog.assertz("switch(centro, " + str(switch[1]) + ")")
227.            prolog.assertz("switch(sinistra, " + str(switch[0]) + ")")
228.            prolog.assertz("switch(destra, " + str(switch[2]) + ")")
229.
230.            # sonar: mettere prima il centro cosi se sono 3 sonar uguali il sistema p
    rendera il primo e quindi va avanti
231.            prolog.assertz("sonar(centro, " + str(sonar[1]) + ")")
232.            prolog.assertz("sonar(sinistra, " + str(sonar[0]) + ")")
233.            prolog.assertz("sonar(destra, " + str(sonar[2]) + ")")
234.
235.            # volt
236.            prolog.assertz("volt(" + str(volt) + ")")
237.
238.            ##############################################
239.            # CREAZIONE FATTI PER L'UTILIZZO DEL LIDAR #
240.            ##############################################
241.
242.            if commandolder == 'attiva_lidar':
243.
244.                # lidar ogni 10 gradi
245.                prolog.assertz("lidar(0, " + str(lidar[0]) + ")")
246.                prolog.assertz("lidar(10, " + str(lidar[1]) + ")")
247.                prolog.assertz("lidar(20, " + str(lidar[2]) + ")")
248.                prolog.assertz("lidar(30, " + str(lidar[3]) + ")")
249.                prolog.assertz("lidar(40, " + str(lidar[4]) + ")")
250.                prolog.assertz("lidar(50, " + str(lidar[5]) + ")")
251.                prolog.assertz("lidar(60, " + str(lidar[6]) + ")")
252.                prolog.assertz("lidar(70, " + str(lidar[7]) + ")")
253.                prolog.assertz("lidar(80, " + str(lidar[8]) + ")")
254.                prolog.assertz("lidar(90, " + str(lidar[9]) + ")")
255.                prolog.assertz("lidar(100, " + str(lidar[10]) + ")")
256.                prolog.assertz("lidar(110, " + str(lidar[11]) + ")")
257.                prolog.assertz("lidar(120, " + str(lidar[12]) + ")")
258.                prolog.assertz("lidar(130, " + str(lidar[13]) + ")")
259.                prolog.assertz("lidar(140, " + str(lidar[14]) + ")")
260.                prolog.assertz("lidar(150, " + str(lidar[15]) + ")")
261.                prolog.assertz("lidar(160, " + str(lidar[16]) + ")")
262.                prolog.assertz("lidar(170, " + str(lidar[17]) + ")")
263.
264.                prolog.assertz("lidarmedia(sinistra, " + str(lidarm[0]) + ")")
265.                prolog.assertz("lidarmedia(destra, " + str(lidarm[1]) + ")")
266.
267.            # compass
268.            prolog.assertz("angle16(" + str(angle16) + ")")
269.            prolog.assertz("angle8(" + str(angle8) + ")")
270.            prolog.assertz("pitch(" + str(pitch) + ")")
271.            prolog.assertz("roll(" + str(roll) + ")")
272.            prolog.assertz("mag(xhigh, " + str(mag[0]) + ")")
273.            prolog.assertz("mag(xlow, " + str(mag[1]) + ")")
274.            prolog.assertz("mag(yhigh, " + str(mag[2]) + ")")
275.            prolog.assertz("mag(ylow, " + str(mag[3]) + ")")
276.            prolog.assertz("mag(zhigh, " + str(mag[4]) + ")")
277.            prolog.assertz("mag(zlow, " + str(mag[5]) + ")")
278.            prolog.assertz("acc(xhigh, " + str(acc[0]) + ")")
279.            prolog.assertz("acc(xlow, " + str(acc[1]) + ")")
280.            prolog.assertz("acc(yhigh, " + str(acc[2]) + ")")
281.            prolog.assertz("acc(ylow, " + str(acc[3]) + ")")
282.            prolog.assertz("acc(zhigh, " + str(acc[4]) + ")")
283.            prolog.assertz("acc(zlow, " + str(acc[5]) + ")")
284.            prolog.assertz("gyro(xhigh, " + str(gyro[0]) + ")")
285.            prolog.assertz("gyro(xlow, " + str(gyro[1]) + ")")
286.            prolog.assertz("gyro(yhigh, " + str(gyro[2]) + ")")
287.            prolog.assertz("gyro(ylow, " + str(gyro[3]) + ")")
288.            prolog.assertz("gyro(zhigh, " + str(gyro[4]) + ")")
289.            prolog.assertz("gyro(zlow, " + str(gyro[5]) + ")")
290.            prolog.assertz("temp(" + str(temp) + ")")
```

```
291.
292.           # angolo vecchio
293.           prolog.assertz("oldangle8(" + str(oldangle) + ")")
294.
295.           ##########################################################################
    ########################################
296.
297.           ##########################
298.           # ESEQUZIONE DELLE QUERY #
299.           ##########################
300.
301.           if commandolder == 'attiva_lidar':
302.               result = list(prolog.query("commandlidar(Result)"))
303.           else:
304.               result = list(prolog.query("command(Result)"))
305.
306.           ##########################################################################
    ########################################
307.
308.           ###################################
309.           # DEMOLIZIONE DEI FATTI CARICATI#
310.           ###################################
311.
312.           # comando vecchio
313.           prolog.retract("commandolder(" + str(commandolder) + ")")
314.
315.           # qrcode:  0 per false 1 per true
316.           prolog.retract("qrcode(" + str(qrcode) + ")")
317.
318.           # fotocomera
319.           prolog.retract("fotocamera(" + str(fotocamera) + ")")
320.
321.           # switch
322.           prolog.retract("switch(centro, " + str(switch[1]) + ")")
323.           prolog.retract("switch(sinistra, " + str(switch[0]) + ")")
324.           prolog.retract("switch(destra, " + str(switch[2]) + ")")
325.
326.           # sonar: mettere prima il centro cosi se sono 3 sonar uguali il sistema p
    rendera il primo e quindi va avanti
327.           prolog.retract("sonar(centro, " + str(sonar[1]) + ")")
328.           prolog.retract("sonar(sinistra, " + str(sonar[0]) + ")")
329.           prolog.retract("sonar(destra, " + str(sonar[2]) + ")")
330.
331.           # volt
332.           prolog.retract("volt(" + str(volt) + ")")
333.
334.           ##############################################################
335.           # DEMOLIZIONE DEI FATTI CARICATI PER L'UTILIZZO DEL LIDAR #
336.           ##############################################################
337.
338.           if commandolder == 'attiva_lidar':
339.
340.               # lidar ogni 10 gradi
341.               prolog.retract("lidar(0, " + str(lidar[0]) + ")")
342.               prolog.retract("lidar(10, " + str(lidar[1]) + ")")
343.               prolog.retract("lidar(20, " + str(lidar[2]) + ")")
344.               prolog.retract("lidar(30, " + str(lidar[3]) + ")")
345.               prolog.retract("lidar(40, " + str(lidar[4]) + ")")
346.               prolog.retract("lidar(50, " + str(lidar[5]) + ")")
347.               prolog.retract("lidar(60, " + str(lidar[6]) + ")")
348.               prolog.retract("lidar(70, " + str(lidar[7]) + ")")
349.               prolog.retract("lidar(80, " + str(lidar[8]) + ")")
350.               prolog.retract("lidar(90, " + str(lidar[9]) + ")")
351.               prolog.retract("lidar(100, " + str(lidar[10]) + ")")
352.               prolog.retract("lidar(110, " + str(lidar[11]) + ")")
353.               prolog.retract("lidar(120, " + str(lidar[12]) + ")")
```

```
354.                 prolog.retract("lidar(130, " + str(lidar[13]) + ")")
355.                 prolog.retract("lidar(140, " + str(lidar[14]) + ")")
356.                 prolog.retract("lidar(150, " + str(lidar[15]) + ")")
357.                 prolog.retract("lidar(160, " + str(lidar[16]) + ")")
358.                 prolog.retract("lidar(170, " + str(lidar[17]) + ")")
359.
360.                 prolog.retract("lidarmedia(sinistra, " + str(lidarm[0]) + ")")
361.                 prolog.retract("lidarmedia(destra, " + str(lidarm[1]) + ")")
362.
363.             # compass
364.             prolog.retract("angle16(" + str(angle16) + ")")
365.             prolog.retract("angle8(" + str(angle8) + ")")
366.             prolog.retract("pitch(" + str(pitch) + ")")
367.             prolog.retract("roll(" + str(roll) + ")")
368.             prolog.retract("mag(xhigh, " + str(mag[0]) + ")")
369.             prolog.retract("mag(xlow, " + str(mag[1]) + ")")
370.             prolog.retract("mag(yhigh, " + str(mag[2]) + ")")
371.             prolog.retract("mag(ylow, " + str(mag[3]) + ")")
372.             prolog.retract("mag(zhigh, " + str(mag[4]) + ")")
373.             prolog.retract("mag(zlow, " + str(mag[5]) + ")")
374.             prolog.retract("acc(xhigh, " + str(acc[0]) + ")")
375.             prolog.retract("acc(xlow, " + str(acc[1]) + ")")
376.             prolog.retract("acc(yhigh, " + str(acc[2]) + ")")
377.             prolog.retract("acc(ylow, " + str(acc[3]) + ")")
378.             prolog.retract("acc(zhigh, " + str(acc[4]) + ")")
379.             prolog.retract("acc(zlow, " + str(acc[5]) + ")")
380.             prolog.retract("gyro(xhigh, " + str(gyro[0]) + ")")
381.             prolog.retract("gyro(xlow, " + str(gyro[1]) + ")")
382.             prolog.retract("gyro(yhigh, " + str(gyro[2]) + ")")
383.             prolog.retract("gyro(ylow, " + str(gyro[3]) + ")")
384.             prolog.retract("gyro(zhigh, " + str(gyro[4]) + ")")
385.             prolog.retract("gyro(zlow, " + str(gyro[5]) + ")")
386.             prolog.retract("temp(" + str(temp) + ")")
387.
388.             # angolo vecchio
389.             prolog.retract("oldangle8(" + str(oldangle) + ")")
390.
391.             return result
392.
393.
394.         def callback(msg):
395.             """
396.             funzione di callback di Ros, effetua le valutazioni del mondo circostante
    e fa uuna scelta
397.             :param msg: messaggio ros dal controller
398.             :return: nulla
399.             """
400.             global commandolder, oldangle, commandIA, qrcode, fotocamera, switch, son
    ar, volt,\
401.                 lidar, angle8, angle16, pitch, roll, mag, acc, gyro, temp
402.             qrcode = msg.qrcode
403.             fotocamera = msg.visione
404.             switch = msg.switches
405.             switch[0] = int(switch[0])
406.             switch[1] = int(switch[1])
407.             switch[2] = int(switch[2])
408.             sonare = msg.sonar
409.             sonar = (int(sonare[0]), int(sonare[1]), int(sonare[2]))
410.             volt = msg.volt
411.             lidar = msg.lidar
412.             angle16 = msg.angle16
413.             angle8 = msg.angle8
414.             pitch = msg.pitch
415.             roll = msg.roll
416.             mag = msg.mag
417.             acc = msg.acc
```

```
418.            gyro = msg.gyro
419.            temp = msg.temp
420.
421.
422.        def ifNotNone(qrcode, fotocamera, switch, sonar, volt, lidar, angle16, angle8
    , pitch, roll, mag, acc, gyro, temp):
423.            """
424.            Funzione di controlo delle variabili globali,
425.            controlla se sono state modificate tutte
426.            :param qrcode:
427.            :param fotocamera:
428.            :param switch:
429.            :param sonar:
430.            :param volt:
431.            :param lidar:
432.            :param angle16:
433.            :param angle8:
434.            :param pitch:
435.            :param roll:
436.            :param mag:
437.            :param acc:
438.            :param gyro:
439.            :param temp:
440.            :return: True se non sono None e False se sono None
441.            """
442.            return qrcode is not None and fotocamera is not None and switch is not No
    ne and sonar is not None and\
443.                   volt is not None and lidar is not None and angle16 is not None and
     angle8 is not None and\
444.                   pitch is not None and roll is not None and mag is not None and acc
     is not None and\
445.                   gyro is not None and temp is not None
446.
447.
448.        def main():
449.            global commandolder, oldangle, commandIA, qrcode, fotocamera, switch, son
    ar,\
450.                volt, lidar, angle8, angle16, pitch, roll, mag, acc, gyro, temp, lida
    rm
451.            commandIA = ''
452.            prologinit_and_rules()
453.            print("prolog rules")
454.            rospy.init_node("Prologo_IA_Node", disable_signals=True)
455.            prolog_pub = rospy.Publisher("prolog_ia", PyRobot.Prolog_IA_Node, queue_s
    ize=0)
456.            rospy.Subscriber("controller", PyRobot.Controller_Node, callback)
457.            r = rospy.Rate(1)
458.            while not rospy.is_shutdown():
459.                if commandIA == '':
460.                    if ifNotNone(qrcode, fotocamera, switch, sonar, volt, lidar,
461.                                 angle16, angle8, pitch, roll, mag, acc, gyro, temp):

462.                        if commandolder == "attiva_lidar":
463.                            lidarslit = np.split(lidar, [8])
464.                            lidarm = [np.median(lidarslit[1]), np.median(lidarslit[0]
    )]
465.
466.                        if commandolder == "correggi_a_destra" or\
467.                                commandolder == "correggi_a_sinistra" or oldangle is
    None:
468.                            oldangle = angle8
469.
470.                        commandIA = prologIA(commandolder, qrcode, fotocamera, switch
    , sonar, volt, lidar,
471.                                             angle16, angle8, pitch, roll, mag, acc,
    gyro, temp, oldangle, lidarm)
```

```
472.
473.                         if commandIA == "sinistra" or commandIA == "destra" or comman
    dIA == "indietro":
474.                             oldangle = None
475.
476.                         commandolder = None
477.                         commandolder = commandIA[0]['Result']
478.                         prolog_msg.risposta = commandIA[0]['Result']
479.                         prolog_pub.publish(prolog_msg)
480.                         rospy.loginfo(prolog_msg)
481.                         if commandolder =="attiva_lidar":
482.                             time.sleep(5)
483.                         commandIA = ''
484.                         resetNone()
485.                         r.sleep()
486.
487.
488.         if __name__ == '__main__':
489.             try:
490.                 main()
491.             except rospy.ROSInterruptException:
492.                 pass
```

SONAR_VOLT NODE

```
1.  #! /usr/bin/python
2.
3.  from librerie import *
4.  import numpy as np
5.  import rospy
6.  import py_robot_v_rep.msg as PyRobot
7.
8.  sonar_volt_msg = PyRobot.Sonar_Volt_Node()
9.  oggetto = []
10. distanza = []
11. clientID = None
12.
13.
14. def connessione():
15.     """
16.     funzione per connessione
17.     :return: clientID
18.     """
19.     print('Program started')
20.     simxFinish(-1)  # just in case, close all opened connections
21.     clientID = simxStart('127.0.0.1', 19999, True, True, 5000, 5)  # Connect to V-
    REP
22.     if clientID == -1:
23.         print("Failed to connect to Remote API Server")
24.     else:
25.         print('Connected to Remote API Server')
26.         return clientID
27.
28.
29. def oggetti(clientID):
30.     """
31.     funzione che crea gli oggetti nel modo v-rep
32.     :var globale oggetto: rappresenta gli oggetti creati, e' di tipo lista
33.     :param clientID: connesione v-rep
34.     :return: oggetto
35.     """
36.     global oggetto
```

```
37.     nomioggetti = ['Proximity_sensorSX', 'Proximity_sensorCE', 'Proximity_sensorDX']
38.     for i in range(0, 3):
39.         res, objecthandle = simxGetObjectHandle(clientID, nomioggetti[i], simx_opmod
    e_blocking)
40.         simxReadProximitySensor(clientID, objecthandle, simx_opmode_streaming)
41.         if not res == 0:
42.             print ("Creation Error")
43.             return
44.         else:
45.             print ("Creation " + nomioggetti[i])
46.             oggetto.append(objecthandle)
47.     return oggetto
48.
49.
50. def readproximity(clientID, handle):
51.     """
52.     funzione che ritorna una lettura, la lettura in questo caso  e' dei sonar
53.     :param clientID: connessione v-rep
54.     :param handle: oggetto dal quale provine la lettura
55.     :return ditsanza: distanza letta
56.     """
57.     global distanza
58.     ris, stato, coordinate, handleoggettoris, vettorenormalizzato = simxReadProximit
    ySensor(clientID, handle[0], simx_opmode_buffer)
59.     distanza.append(np.linalg.norm(coordinate))
60.     ris, stato, coordinate, handleoggettoris, vettorenormalizzato = simxReadProximit
    ySensor(clientID, handle[1],
61.
            simx_opmode_buffer)
62.     distanza.append(np.linalg.norm(coordinate))
63.     ris, stato, coordinate, handleoggettoris, vettorenormalizzato = simxReadProximit
    ySensor(clientID, handle[2],
64.
            simx_opmode_buffer)
65.     distanza.append(np.linalg.norm(coordinate))
66.     return distanza
67.
68.
69. def readvolt(volt):
70.     """
71.     funzione che simula le letture dello statto della batteria
72.     :param volt: numero che corrisponde ai volt della batteria
73.     :return volt: il numero dei volt della batteria
74.     """
75.     return volt
76.
77.
78. def main():
79.     global oggetto, clientID, distanza
80.     clientID = connessione()
81.     newoggetti = oggetti(clientID)
82.     rospy.init_node("Sonar_Volt_Node")
83.     sonar_volt_pub = rospy.Publisher("sonar_volt", PyRobot.Sonar_Volt_Node, queue_si
    ze=0)
84.     r = rospy.Rate(0.5)
85.
86.     while not rospy.is_shutdown():
87.         dist = readproximity(clientID, newoggetti)
88.         dist[0] = dist[0] * 100
89.         dist[1] = dist[1] * 100
90.         dist[2] = dist[2] * 100
91.         sonar_volt_msg.sonar =  dist
92.         distanza = []
93.         sonar_volt_msg.volt = readvolt(float(13))
94.         sonar_volt_pub.publish(sonar_volt_msg)
```

```
        rospy.loginfo(sonar_volt_msg)
        r.sleep()


if __name__ == '__main__':
            try:
                main()
            except rospy.ROSInterruptException:
                pass
```

LAUNCHER

```
<launch>
    <node pkg="py_robot_v_rep" type="Lidar_Compass_Node.py" name="Lidar_Compass_Node" output="screen">
    </node>
    <node pkg="py_robot_v_rep" type="Sonar_Volt_Node.py" name="Sonar_Volt_Node" output="screen">
    </node>
    <node pkg="py_robot_v_rep" type="Motor_Switch_Node.py" name="Motor_Switch_Node" output="screen">
    </node>
    <node pkg="py_robot_v_rep" type="Opencv_Node.py" name="Opencv_Node" output="screen">
    </node>
    <!--
    <node pkg="py_robot_v_rep" type="Controller_Node.py" name="Controller_Node" output="screen">-->
    <!--</node>-->
</launch>
```

MESSAGGI

CONTROLLER NODE MESSAGE

```
bool on_off_lidar
int8 qrcode
string fotocamera
bool[3] switches
float64[3] sonar
float32 volt
float32[18] lidar
int16 angle16
int8 angle8
int16 pitch
int16 roll
int16[6] mag
int16[6] acc
int16[6] gyro
int16 temp
string visione
```

CONTROLLER TO LIDAR NODE MESSAGE

```
bool on_off_lidar
```

CONTROLLER TO MOTOR NODE MESSAGE

```
string velo
```

LIDAR COMPASS NODE MESSAGE

```
float32[18] lidar
int16 angle16
int8 angle8
int16 pitch
int16 roll
int16[6] mag
int16[6] acc
int16[6] gyro
int16 temp
```

MV CAMERA NODE MESSAGE

```
string eureca
```

MOTOR SWITCH NODE MESSAGE

```
bool[3] switches
```

PI CAMERA NODE MESSAGE

```
string visione
```

PROLOG IA NODE MESSAGE

```
string risposta
```

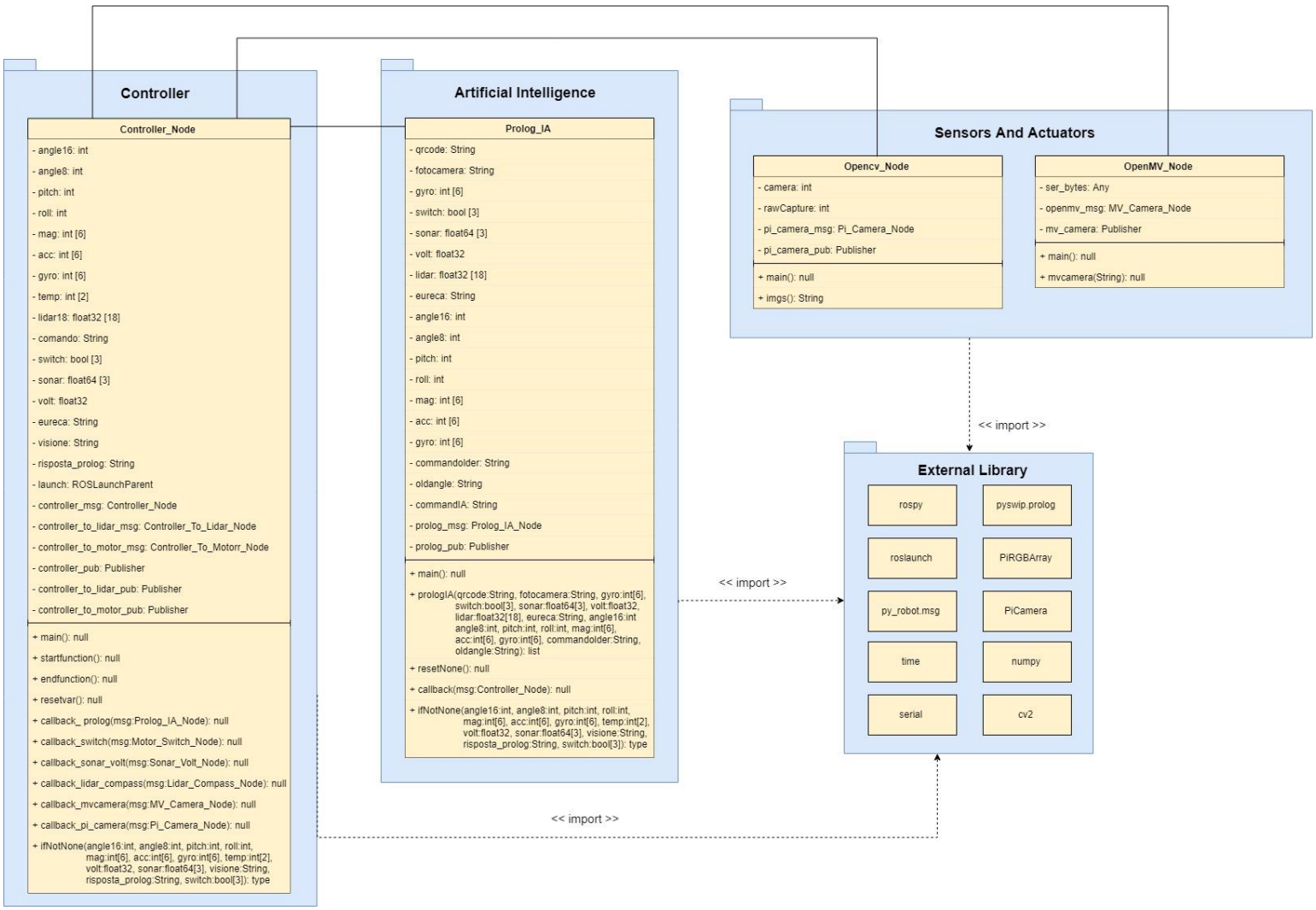
SONAR VOLT NODE MESSAGE

```
1. float64[3] sonar
2. float32 volt
```

DIAGRAMMI

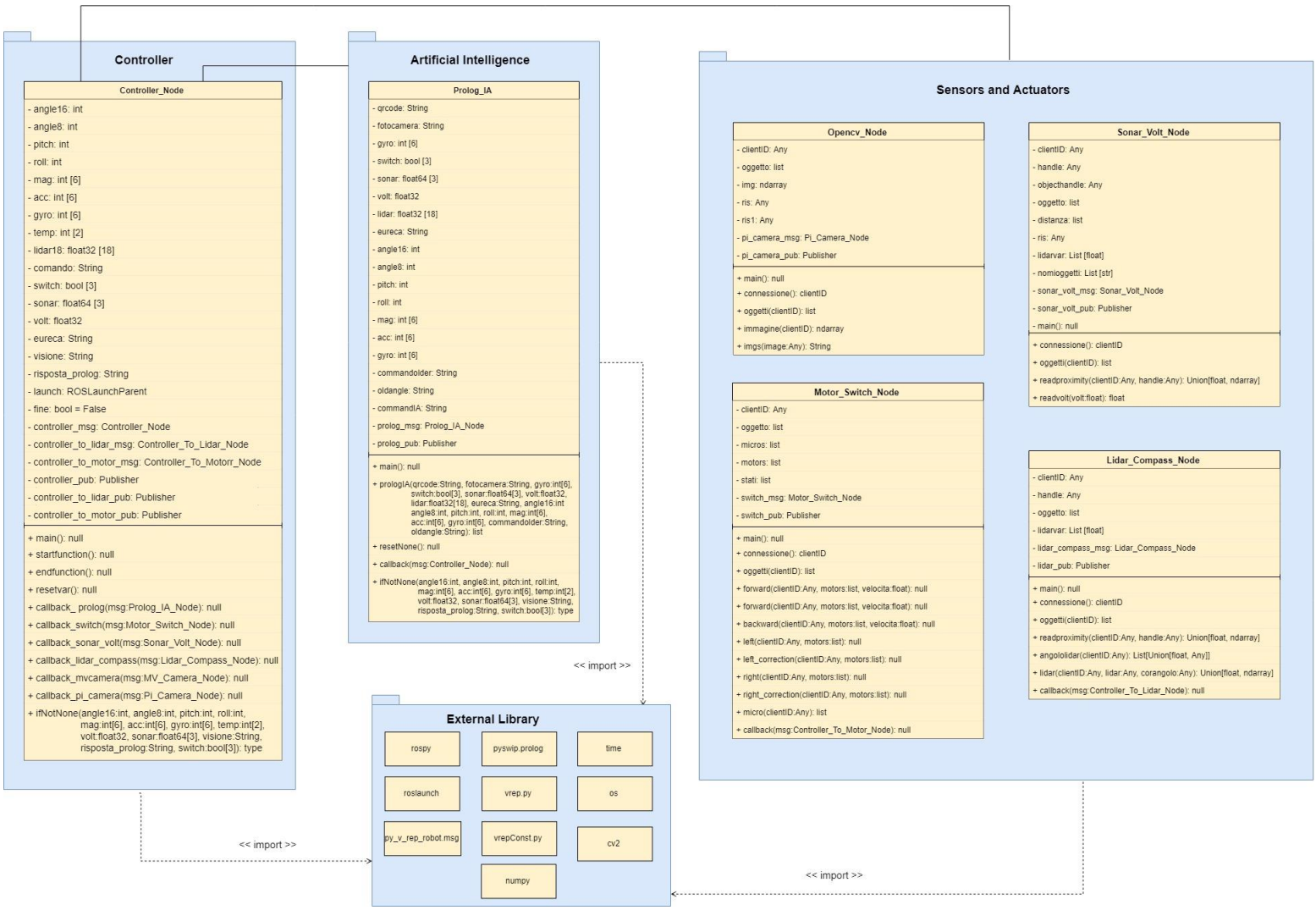
CLASS DIAGRAM PARTE FISICA

Questo diagramma è un tipo di diagramma di struttura statico che descrive la struttura di un sistema mostrando le classi, i loro attributi, le operazioni (oi metodi) e le relazioni tra gli oggetti.



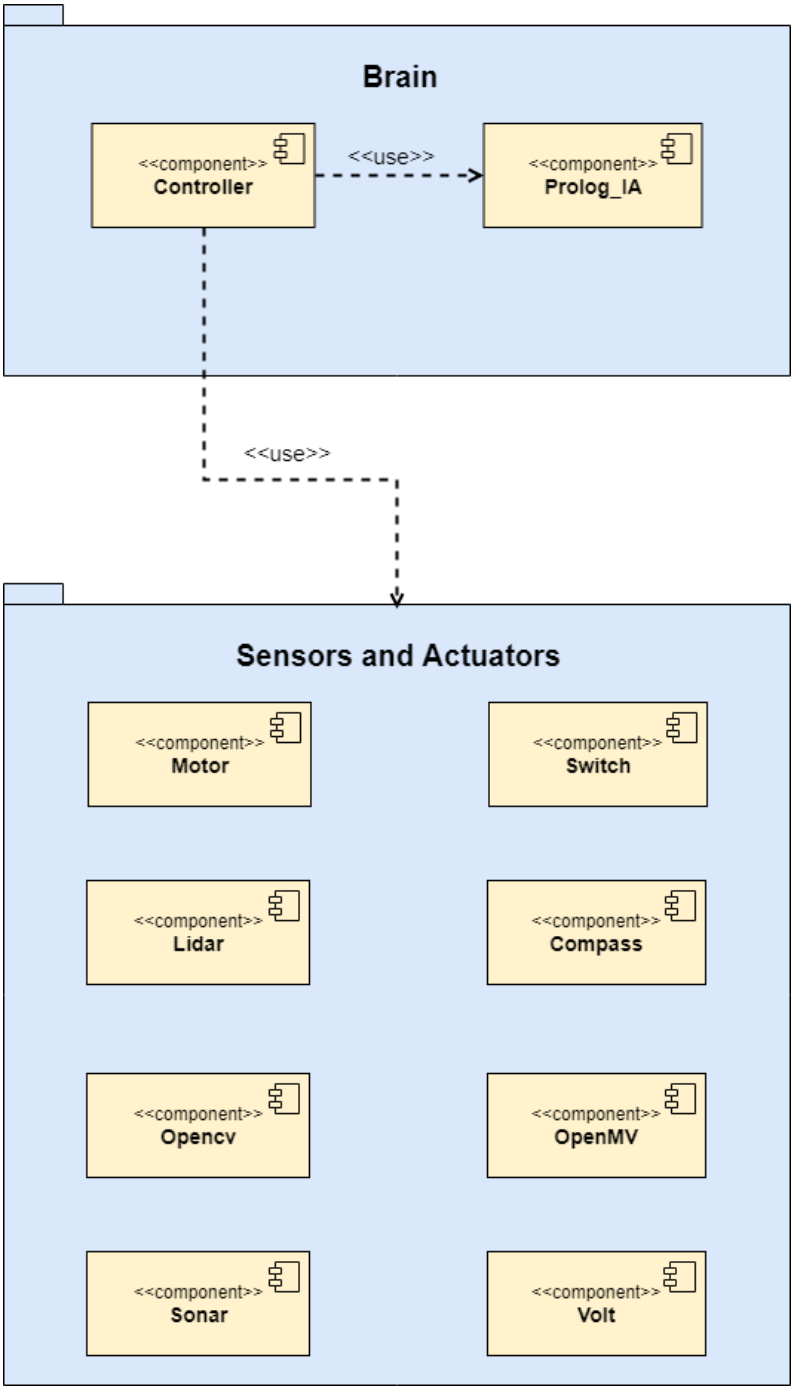
CLASS DIAGRAM PARTE VIRTUALE

Questo diagramma è un tipo di diagramma di struttura statico che descrive la struttura di un sistema mostrando le classi, i loro attributi, le operazioni



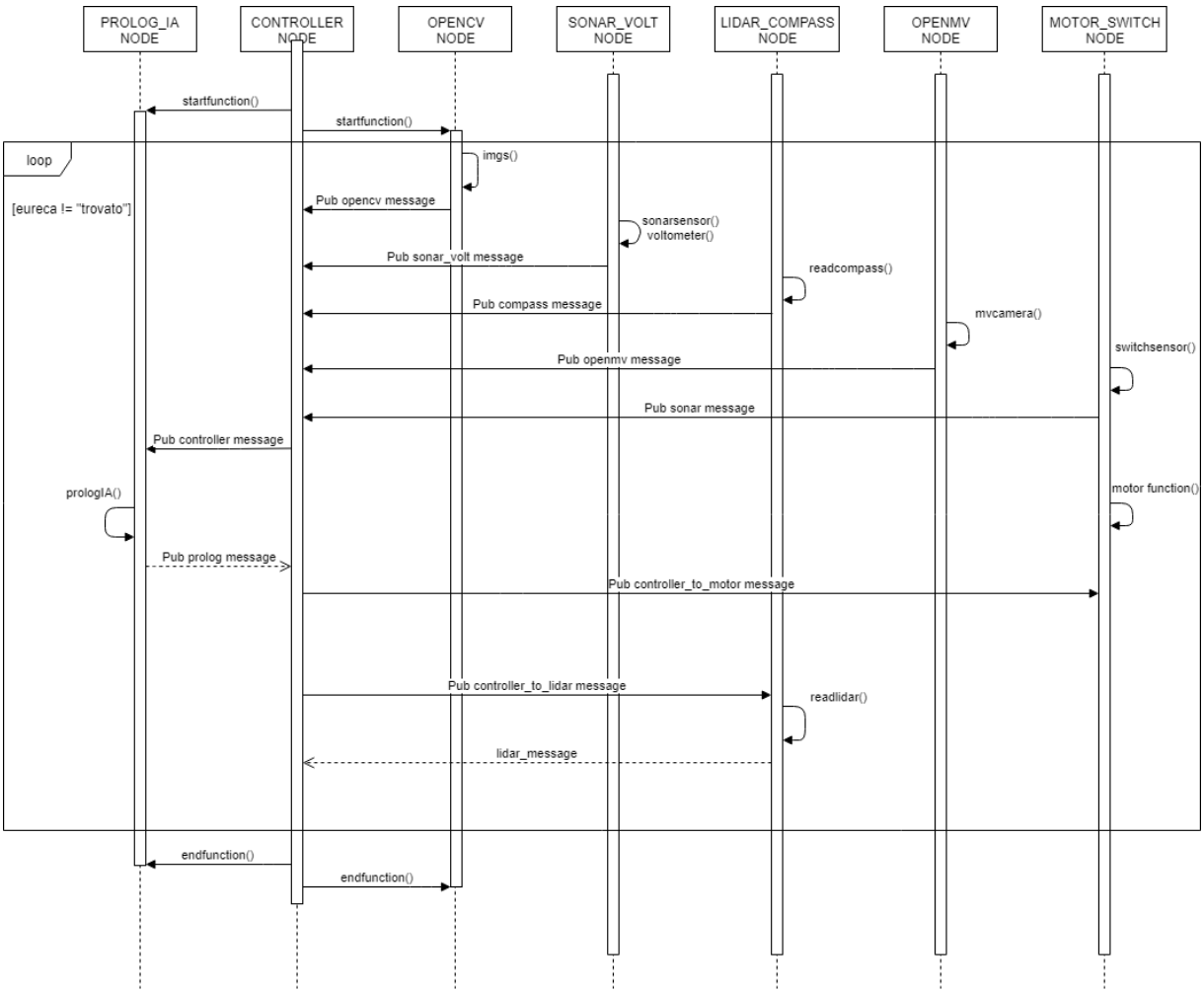
COMPONENT DIAGRAM

I Component Diagram sono utilizzati per visualizzare i componenti fisici in un sistema. Questi componenti sono librerie, pacchetti, file ecc.



SEQUENCE DIAGRAM

Il Sequence Diagram è uno speciale diagramma di interazione che cattura la sequenza temporale del flusso di messaggi da un oggetto a un altro.

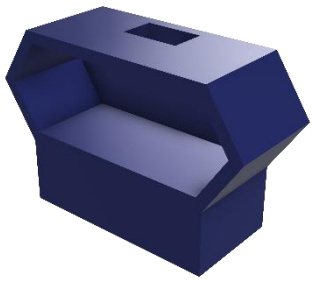


NOTE

Inizialmente la nostra intenzione era di introdurre nel rover anche un lidar che ci permetteva di vedere la distanza attraverso un led ottico per poi toglierlo. Abbiamo stampato, infatti, dei componenti che potrebbero servire in futuro.

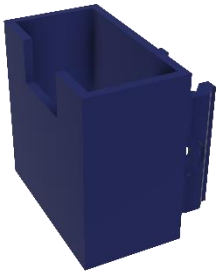
Sono i seguenti:

- Case Lidar



Case per contenere il sensore lidar

- Case MV Camera



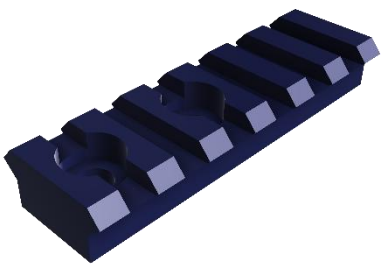
Case per contenere la MV Camera

- Torre

Torre sulla quale viene posto il Lidar e attaccato la slitta.

- Slitta

Slitta che serve per far scorrere sopra e sotto il case MV Camera



LISTINO PREZZI

Tutti i costi relativi alla creazione del Rover PyRobot sono nella reporitory di GitLab