Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Tesi di Laurea

Implementazione di un sistema mobile ed autonomo per la ricerca di oggetti in base al colore

Laureando

Simone Mariotti

Relatore

Prof. Marco Baioletti

Correlatore

Dott. Emanuele Palazzetti

Anno Accademico 2013 - 2014

Alla mia famiglia

Indice

Introduzione

I primi automi eseguivano azioni preimpostate semplici e ripetitive; erano da considerarsi poco più che macchine utensili. Si pensi ai robot delle catene di produzione di un'autovettura, eseguono sempre le stesse azioni, si muovono sempre degli stessi centimetri o millimetri portandosi sempre nella stessa posizione, eseguono la loro azione programmata e passano al successivo compito. Gli algoritmi che governano tali robot, data la loro natura e le specifiche che rispettano, non possono essere considerati intelligenti.
Intelligente, in ambito informatico, ha una definizione completamente differente dall'intelligenza così come la percepiamo in ambito umano. Una macchina non possiede creatività, non impara e non immagina. Se sembra possedere una di queste qualità è perché è stata programmata in tal senso. Una macchina non può andare oltre la sua programmazione, o almeno non ha ancora potuto. Nessuna macchina ad oggi ha superato il Test di Turing[1]. Eugene Goostman, un software appositamente sviluppato per emulare il pensiero di un ragazzo di 13 anni di origine Ucraina e con una limitata conoscenza dell'inglese, è riuscito ad "ingannare" solo il 29% dei giudici evitando le domande in modo credibile per un ragazzo con le carat-

[1] È una condizione proposta da Alan Turing nel 1950 che una macchina deve soddisfare per essere considerata intelligente.

teristiche dichiarate. Joshua Tenenbaum, Professore Ph.D. di Scienze Cognitive Computazionali al MIT, in merito ha dichiarato che il risultato non è impressionante.[5]
In informatica un algoritmo è considerato intelligente se a seguito dell'analisi di un problema sceglie la soluzione più giusta o più efficiente sulla base delle informazioni che possiedono.

In questo lavoro di tesi la nostra intelligenza artificiale realizza una forma semplificata del *bin-picking*, tecnica usata in ambito industriale per identificare e prelevare oggetti disposti alla rinfusa in un ambiente.[3] Nel nostro caso si limiterà a due compiti "classici" per le IA[2], cercare e identificare il suo obiettivo. All'avvio del sistema l'app Android che svolge le funzioni di IA, ci chiederà di selezionare tramite touch screen un colore visibile sul video proveniente dalla camera. Una volta selezionato il colore e premuto il pulsante start, il robot si muoverà autonomamente alla ricerca di tale colore nell'ambiente. Date le caratteristiche dell'algoritmo implementato, l'ambiente di test potrà avere qualsiasi forma ed estensione; per stabilire i suoi confini potremo utilizzare un materiale a bassa riflessività, come carta nera opaca, oppure un oggetto per formare un piano sopraelevato rispetto all'area circostante. Il robot si aggirerà per l'ambiente evitando ostacoli e senza uscire dall'area delimitata finché non troverà un oggetto del colore cercato. A questo punto avvertirà di aver raggiunto l'obiettivo e si metterà in attesa di un nuovo input da parte dell'utente.

[2]Intelligenza Artificiale.

Capitolo 1

Visione Artificiale

La *visione artificiale* ha come compito la creazione di un modello del mondo reale in tre dimensioni a partire da numerose immagini bidimensionali per cercare di emulare il processo biologico della vista. Negli esseri umani, così come in molte altre specie, vedere non è solo scattare una fotografia bidimensionale mentale di un ambiente, ma è interpretare le informazioni ricevute attraverso la retina per analizzare e creare un modello 3D dell'ambiente circostante.

Un sistema di visione artificiale, per provare ad avvicinarsi alla capacità visiva e di interpretazione umana, ha bisogno di numerosi componenti di diversa natura: ottici, elettronici e meccanici per acquisire e memorizzare le immagini da elaborare. Il primo tentativo in tal senso è datato 1883 quando Paul Gottlieb Nipkow inventò il primo sistema in grado di trasformare informazioni visive in un segnale elettrico.[7]

Il sistema si basa su un disco con dei fori praticati lungo una spirale che parte dal centro del disco e procede verso l'esterno. Il disco ruota ad una velocità costante mentre l'immagine inquadrata viene focalizzata da una lente

verso i fori in modo che un sensore, posto sul retro del disco, possa percepire i cambi di luminosità della porzione di scena inquadrata e convertire questa variazione in segnali elettrici.

Il ricevitore emetterà luce in base a dei segnali elettrici e attraverso un disco identico al primo e in rotazione sincrona proietterà un'immagine con tante righe quante sono i fori del disco.[6] Questo sistema porterà alla creazione della TV meccanica nel 1925. Negli anni successivi, con l'introduzione della

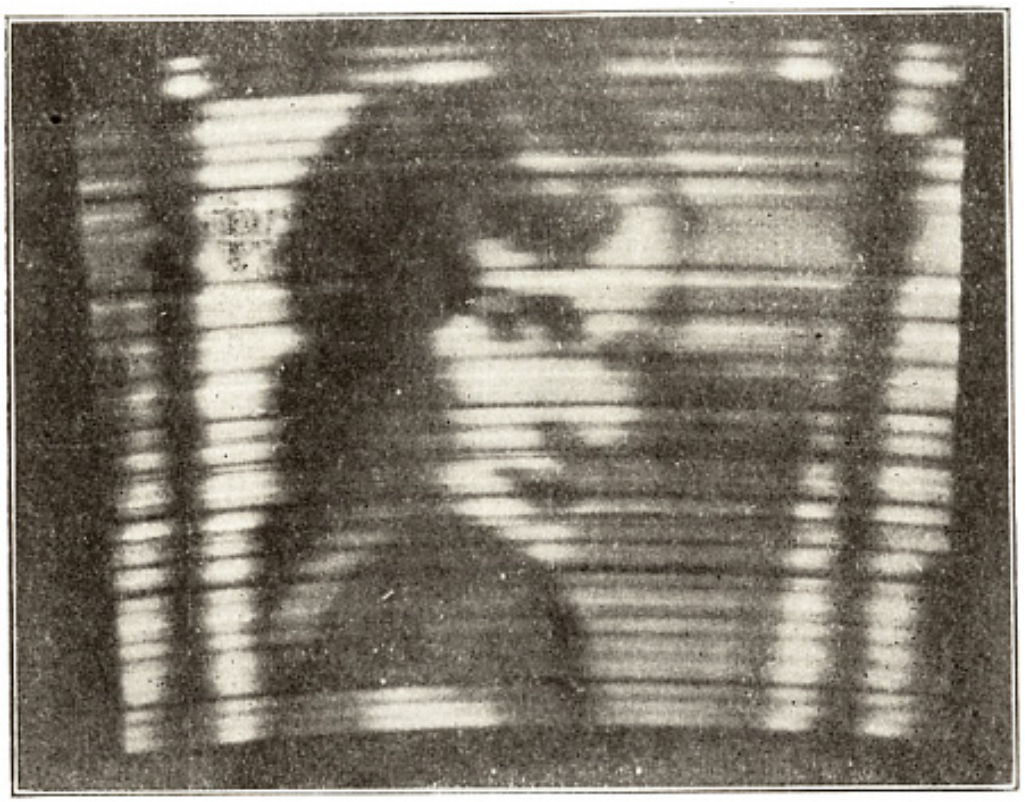

Figura 1.1: Immagine visibile su una TV meccanica del 1925

miniaturizzazione nell'elettronica, sono stati compiuti enormi progressi. Da un punto di vista teorico, gli scienziati cominciarono ad ispirarsi al corpo umano nel tentativo di riprodurre il suo comportamento. Analizzando da

vicino l'occhio e in particolare la retina hanno scoperto che era formata da minuscoli recettori sensibili alla luce collegati al cervello tramite il nervo ottico. Si è così deciso di creare dei micro sensori di luce e formarne un'enorme matrice. Il primo risultato di questi studi si ebbe circa 40 anni dopo e fu il sensore CCD[1], ideato da Willard S. Boyle e George E. Smith nel 1969 presso i Bell Laboratories, mentre dobbiamo aspettare il 1993 per vedere i primi prototipi funzionanti del sensore CMOS[2] sviluppati presso il Jet Propulsion Laboratory, entrambi hanno come elemento base il fotodiodo che equivale ad un pixel. Il CCD è un sensore analogico[3] che ha bisogno di più energia ma offre in genere una qualità superiore un rumore minore ad un costo più elevato.

Il CMOS è un sensore digitale che offre una buona qualità di immagine ad

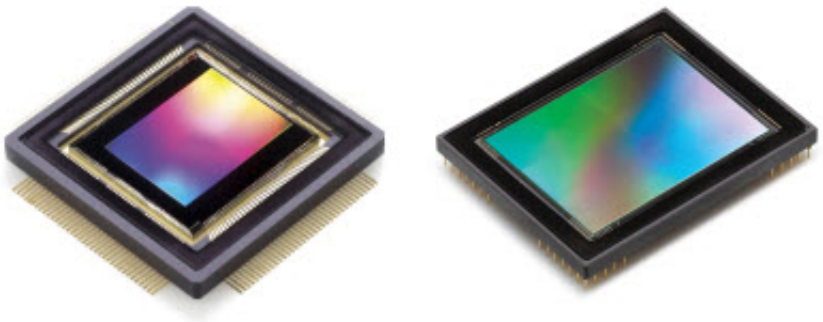

Figura 1.2: Due sensori moderni. A sinistra un sensore CMOS a destra uno CCD

costo minore; per contro l'immagine presenta un forte rumore a causa della

[1]Charge-Coupled Device.

[2]Complementary Metal-Oxide-Semiconductor.

[3]Il sensore più grande esistente è di 1,4 Gigapixel ed è montato sul telescopio Pan-STARRS sviluppato per l'individuazione di meteoriti in rotta di collisione con la Terra.

conversione analogico-digitale.
La visione artificiale ha principalmente tre utilizzi:

- **Ricognizione:** ricercare uno o più oggetti, scelti a priori, e organizzarli in insiemi generici o classi mantenendo informazioni riguardo il loro posizionamento nella scena.
  Esempio: ricercare in un'immagine o in un video tutte le persone, le macchine o gli animali.

- **Identificazione:** identificare una istanza specifica di una classe di oggetti.
  Esempio: ricercare in un'immagine o in un video un volto, una macchina o un animale specifico..

- **Individuazione:** cercare una condizione specifica nell'immagine.
  Esempio: cercare imperfezioni nelle immagini a raggi X di superfici o materiali.

Un campo che ne fa massiccio utilizzo è la modellazione di ambienti 3D a partire da due o più immagini 2D; questo è stato reso possibile grazie all'enorme aumento della capacità di elaborazione grafica delle GPU.
La visione artificiale è stata applicata a molti altri campi nei quali si è verificata una vera e propria rivoluzione.

Uno di questi è la medicina nella quale l'introduzione di questa tecnica ha portato all'analisi di radiografie, angiografie, tomografie e altre immagini mediche. In questo modo è possibile identificare anomalie e problemi, quali i tumori, che non sarebbero visibili all'occhio umano se non in seguito a procedure molto invasive.

Un altro campo è quello del controllo di veicoli autonomi i quali stanno aumentando ad un ritmo vertiginoso, progressivamente al migliorarsi delle tecniche di visione artificiale. Autovetture, droni, robot e carri per il rifornimento sono solo degli esempi a cosa può portare la visione artificiale applicata nei settori civili e di ricerca. La nostra tesi si concentra proprio

Figura 1.3: Drone della società statunitense Amazon. Consegnerà prodotti a domicilio in completa autonomia

sull'applicazione ad un robot autonomo dei concetti di visione artificiale appena descritti.

Capitolo 2

Componenti hardware

2.1 UDOO Quad

UDOO è un progetto tutto italiano di una piattaforma hardware destinata alla generazione dei "maker", i cosiddetti *artigiani digitali* del 21° secolo, cioè persone che inventano e producono apparecchi di ogni genere e condividono i risultati pubblicamente sotto licenza open-source così che possano essere la base di un progetto di un altro maker. La scheda UDOO ha visto la luce dopo una sorprendente campagna di crowdfunding [1] che ha raggiunto la quota richiesta dagli sviluppatori per iniziare la produzione in solo due giorni. Per permettere l'utilizzo di librerie e applicazioni computazionalmente pesanti come openCV, PureData e altre UDOO è dotata di un processore ARM Freescale i.MX6 Cortex-A9 Quad core 1GHz che supporta sia Android che Linux. Il tutto è completato da una GPU Vivante, 1GB di RAM DDR3, connettore SATA, ingresso microfono, uscita audio, Ethernet,

[1] Dall'inglese crowd, folla e funding, finanziamento. In italiano finanziamento collettivo. La piattaforma utilizzata è Kickstarter.

HDMI, USB, connettore per display LVDS con touch screen, connettore CSI per camera esterna e connettività bluetooth e Wi-Fi. La board supporta i sistemi operativi Linux ed Android nelle rispettive versioni compilati per i processori ARM. Il sistema operativo risiede su una memoria microSD e il caricamento viene eseguito grazie ad un lettore di schede microSD integrato. Quello che però ha fatto di questa scheda la nostra scelta per questo progetto di tesi, è la presenza all'interno di un Arduino Due completamente integrato. È presente una CPU Atmel SAM3X8E ARM Cortex-M3 [2] e 76 GPIO[3], di cui 62 digitali e 14 digitali/analogici, disposti per essere perfettamente compatibili con la piedinatura dell'Arduino Due e dell'Arduino Uno Rev.3.

La presenza di un Arduino Due all'interno della board rende UDOO una scheda di prototipazione a tutti gli effetti, aprendo nuovi scenari e possibilità grazie alla versatilità e semplicità di Arduino, la potenza di calcolo del Freescale i.MX6 e le numerose periferiche disponibili per Linux o Android. È possibile accedere alla shell del sistema operativo come root tramite la porta seriale integrata e modificare a piacimento la configurazione del sistema operativo. Arduino è collegato al Freescale i.MX6 tramite un bus interno e quindi viene rilevato come una normale periferica USB da Linux. Su Android la comunicazione tra i due dispositivi avviene sullo stesso bus ma usa lo standard USB OTG[4]. L'interconnessione tra l'accessorio Arduino

[2]La stessa di cui dispone l'Arduino Due.

[3]General Purpose Input/Output.

[4]On-The -Go è una specifica che permettere di agire come host ad un qualsiasi dispositivo (tipicamente smartphone e tablet). A differenza dell'USB classico l'OTG è driver-less, cioè non necessita l'installazione di driver specifici per ogni dispositivo.

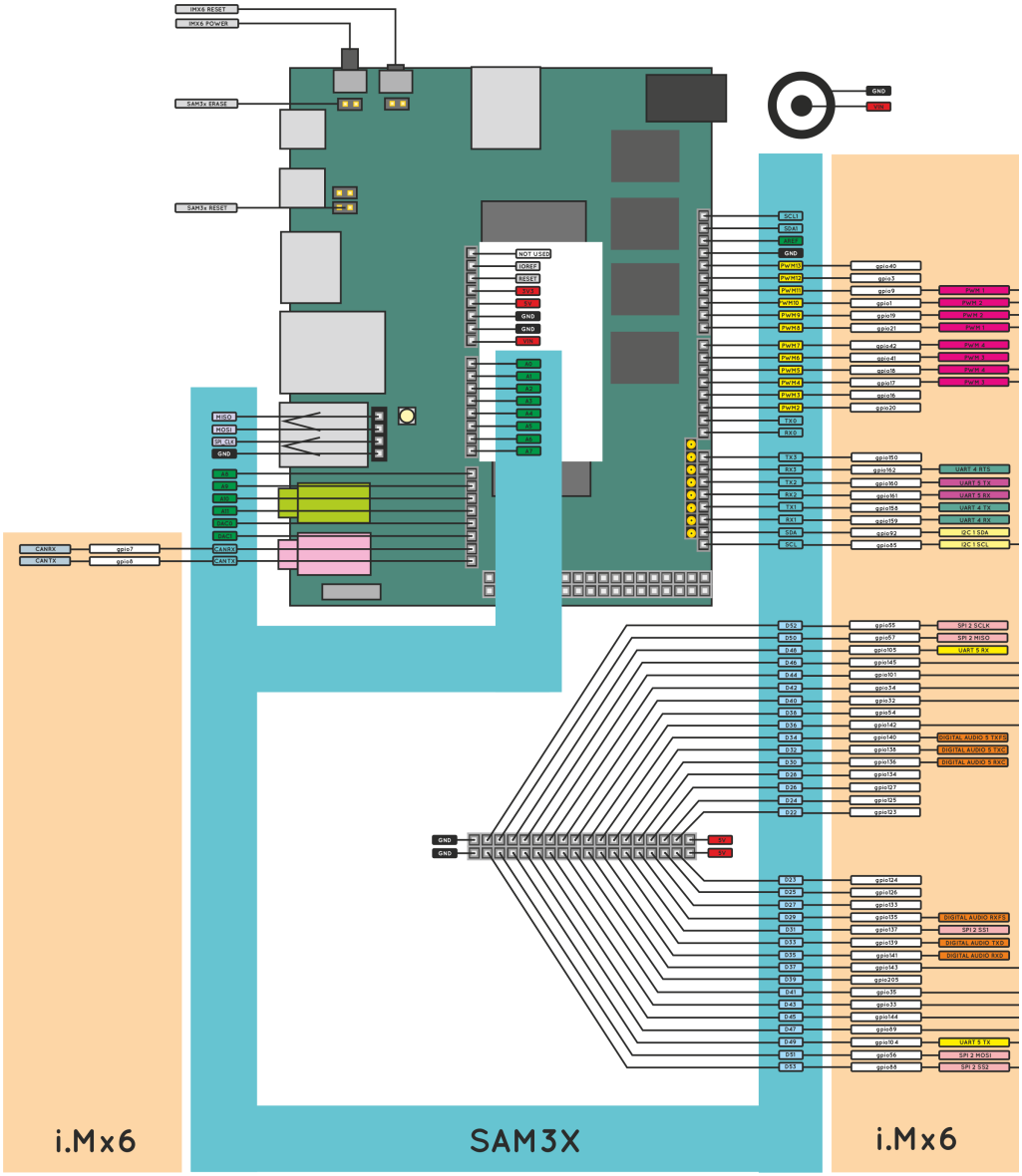

Figura 2.1: Schema piedinatura UDOO

e l'applicazione Android è realizzata tramite l'ADK[5] 2012 che permette l'integrazione del sistema operativo con altre periferiche o dispositivi, tramite una connessione USB o Bluetooth.

[5]Android Development Kit.

2.2 Tank Kit

Per dare la giusta stabilità e manovrabilità al robot si è deciso di usare una locomozione a cingoli che richiede solo due motori e permette di ruotare sul posto o comunque in spazi ristretti: la nostra scelta è stata il "Multi-Chassis - Tank Version". Questa piattaforma, appositamente pensata per la realizzazione di robot multifunzione, si è rivelata la scelta perfetta in quanto possiede due potenti motori DC già forniti di riduttori 48:1 per affrontare terreni impervi e scoscesi, quattro ruote da 52 mm di diametro a cui sono applicati i due cingoli. È presente anche un alloggiamento per un servomotore standard che nella nostra applicazione non è stato usato. L'intelaiatura, di alluminio spesso 2,5 mm, presenta numerosi fori e asole per il montaggio di accessori quali sensori, staffe e motori. Presenta inoltre un "doppio fondo" in cui sono alloggiati i motori DC e i riduttori e in cui è possibile sistemare altri componenti che non debbano essere facilmente accessibili.

2.3 Sensori

2.3.1 Sensore di riflessività - QRD1114

Avendo la necessità di fornire al robot un modo per rilevare eventuali sconfinamenti dall'ambiente di sperimentazione che fosse il più flessibile possibile. Abbiamo optato per il sensore di riflessività QRD1114: questo sensore è costituito da un LED infrarosso e un fototransistor tarato sulla luce infrarossa, con un filtro per la luce solare così da ridurre il rumore di lettura. Il robot era stato pensato per lavorare su superfici con angoli a 90°; in questo scenario,

l'uso di un sensore di distanza puntato verso terra sarebbe stato sufficiente a rilevare l'imminente caduta. L'uso del sensore di riflessività, invece, ha reso possibile l'impiego dell'automa in ambienti generici con l'unico vincolo di renderli delimitati da un recinto, realizzato con materiale a bassa riflettività come del semplice cartoncino nero opaco. Il sensore, non facendo differenza tra il cartoncino nero o un eventuale margine di caduta a lato della superficie, può essere impiegato per rilevare entrambe le condizioni. Il sensore così come fornito dal produttore non è direttamente utilizzabile; per far sì che Arduino potesse acquisire dal sensore valori proporzionali alla riflessività del materiale in esame, abbiamo dovuto realizzare un circuito elettronico di interfaccia.

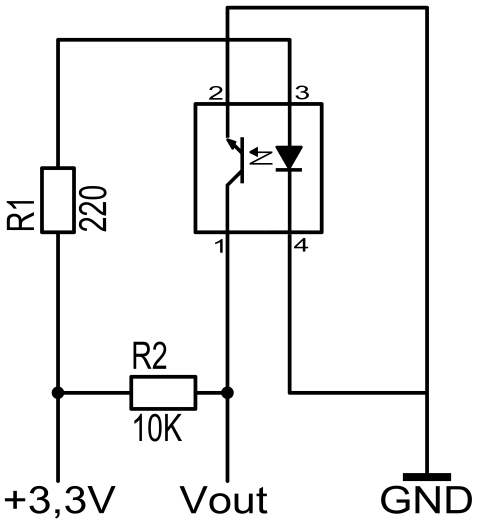

Figura 2.2: Circuito di interfaccia tra Arduino Due e il sensore QRD1114

Il circuito alimenta il LED tramite una resistenza da 220Ω (*R1*) e collega

V_{CC}, pari a 3,3 V nell'Arduino Due, al collettore del fototransistor tramite una resistenza da 10 $k\Omega$ ($R2$) mentre l'emettitore è collegato a terra; il punto da cui prelevare il segnale (V_{out}) è tra $R2$ e il collettore. Il principio alla base del circuito è: il LED è sempre acceso e illumina in modo diffuso parallelamente al fototransistor. Il fototransistor in assenza di luce o, nel nostro caso, in presenza di una bassa riflessività si trova in stato di interdizione; i pin di Arduino impostati come input sono in configurazione "alta impedenza", equivalenti ad un interruttore aperto dal punto di vista circuitale, quindi non c'è passaggio di corrente né tramite il transistor né tramite la resistenza $R2$ il che porta esattamente il valore di V_{CC} in ingresso ad Arduino. Quando il fototransistor è totalmente illuminato, cioè in presenza di alta riflessività, entra in stato di conduzione così che nel punto V_{out} si venga a trovare la massa. Ogni stato intermedio di illuminazione equivale ad una conduzione parziale del fototransistor a cui corrisponde una tensione proporzionale alla riflessività sul pin V_{out}. Il sensore può essere utilizzato in modalità digitale o analogica semplicemente collegando il pin V_{out} ad un pin digitale o analogico e cambiando la configurazione relativa all'interno della programmazione di Arduino. La configurazione digitale si è rivelata inadatta all'applicazione in quanto risulta troppo sensibile a piccole variazioni. In tal caso, anche un minimo movimento sul piano verticale causato da una superficie non perfettamente piana, potrebbe causare un allarme di sconfinamento del robot. La configurazione analogica invece ci permette di avere 1024 valori discreti dal trasduttore, che potranno essere confrontati con un valore soglia oltre al quale il robot rileva un allarme di sconfinamento. La libertà nello scegliere una soglia di allarme ci permette di effettuare una

calibrazione affinata in base al materiale su cui si svolge la sperimentazione per minimizzare la possibilità di falsi positivi.

2.3.2 Sensore di distanza a ultrasuoni - HC-SR04

L'algoritmo di governo del robot, necessita della distanza degli oggetti che si trovano di fronte ad esso; la scelta è ricaduta su un sensore ad ultrasuoni piuttosto che su un sensore ad infrarossi in quanto quest'ultimo è soggetto a rumore dovuto all'illuminazione dell'ambiente di sperimentazione. In particolare abbiamo scelto il sensore HC-SR04 che offre ottime prestazioni, è facile da integrare ed è di dimensioni contenute.
Il sensore può rilevare oggetti distanti da 2 cm a 400 cm con una risoluzione di 0,3 cm entro un cono frontale di 30° per lato rispetto alla perpendicolare dal sensore, complessivamente 60°.

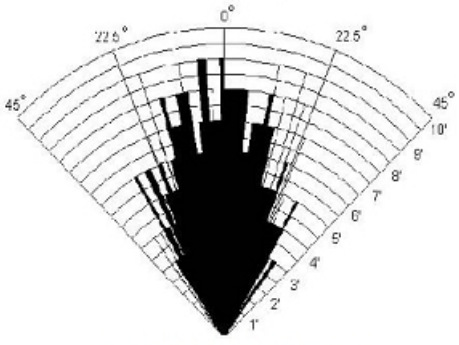

Figura 2.3: Risposta del sensore rispetto alla posizione dell'ostacolo

Il collegamento alla piattaforma di sviluppo è effettuato tramite quattro pin: $+5V$, *trig*, *echo* e *GND*. Una volta alimentato il sensore, Arduino invia sul pin *trig* un impulso di durata 10μs; il sensore emette quindi un impulso ultrasonico e restituisce sul pin *echo* un segnale positivo di durata pari al

tempo di percorrenza, andata e ritorno, del segnale ultrasonico. Una volta ottenuto il tempo di percorrenza si utilizza la formula del moto rettilineo uniforme:

$$v = \frac{\Delta s}{\Delta t}$$

sapendo che la velocità del suono a 20 °C è approssimabile a 344 m/s allora la durata dell'impulso di *echo* sarà compreso tra 116.3 μs e 23,26 ms rispettivamente per distanze percorse di 4 cm (2+2) e di 8 m (4+4).
Per ricavare la distanza (d) in centimetri a partire dalla durata dell'impulso notando che 344 m/s = 29,1 cm/μs e indicando con T_e la durata dell'impulso di echo usiamo:

$$d = \frac{T_e}{2 \cdot 29,1}$$

Arduino Due accetta input solo fino a 3,3 V, valori più elevati di tensione comprometterebbero in modo irreparabile il funzionamento della scheda.
In questo caso il circuito di interfaccia porterà la tensione di 5 V del segnale in uscita dal sensore ad un valore accettato dall'Arduino Due e cioè 3,3 V; serve quindi un partitore di tensione.

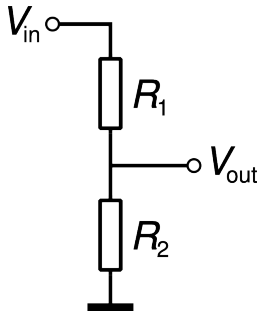

Figura 2.4: Schema generico del partitore di tensione

Il partitore di tensione, a fronte di un noto valore di tensione di ingresso V_{in}, riesce a fornire il valore di tensione desiderato in uscita V_{out} tramite

due resistenze opportunamente dimensionate.

La formula che regola il partitore di tensione è la seguente:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

nel nostro caso $V_{in} = 5$ V e $V_{out} = 3{,}3$ V quindi scegliendo arbitrariamente il valore di R_1 pari a 1 kΩ, ricaviamo il valore di R_2 tramite la formula inversa:

$$R_2 = R_1 \cdot \frac{V_{out}}{V_{in} - V_{out}}$$

che da come risultato 1941,48 Ω; approssimando il valore appena calcolato al più vicino taglio standard di resistenze, cioè $R_2 = 2$ kΩ, si ottiene $V_{out} = 3,\overline{3}$, che è assolutamente accettabile.

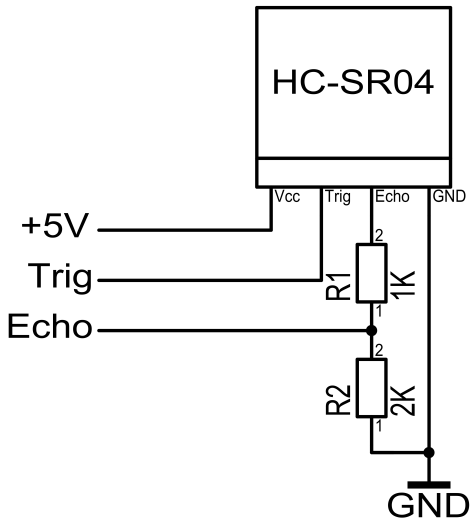

Figura 2.5: Circuito di interfaccia tra Arduino Due e il sensore HC-SR04

Capitolo 3

Componenti software

3.1 OpenCV

Per dare al robot una visione dell'ambiente circostante ci serviva una libreria di visione artificiale. Le più famose disponibili per Java sono OpenCV e JavaCV; la nostra scelta è stata OpenCV per via della maggiore disponibilità di documentazione. OpenCV[1] è stata originariamente sviluppata da Intel per migliorare le prestazioni delle loro CPU con applicazioni computazionalmente pesanti come per esempio gli algoritmi di *ray tracing*. [2] Questa libreria è rilasciata sotto licenza BSD.

La libreria per poter funzionare su un dispositivo Android ha bisogno di un'applicazione di supporto chiamata OpenCV Manager che può essere scaricata dal Play Store o installata direttamente con il pacchetto APK.[3] La

[1] Open Source Computer Vision.

[2] Tecnica di geometrica ottica che analizza il percorso dei raggi di luce. In grafica 3D è un algoritmo di rendering che costruisce la scena facendo partire i raggi luminosi dalla camera (visuale del giocatore) invece che dalla sorgente luminosa.[1]

[3] Android Package, il tipo di file delle app Android simile al formato JAR.

libreria viene caricata in modo asincrono all'avvio dell'app e fornisce ogni frame come una matrice in cui ogni elemento rappresenta un pixel; OpenCV usa il tipo di dato interno Mat per rappresentare le matrici. Tutte le operazioni effettuate su oggetti di tipo Mat usano algoritmi appositamente ottimizzati per le operazioni tra matrici.

3.2 ADK

In Android, a partire dalla versione 2.3.4, è supportato il protocollo Android Open Accessory (AOA) e ogni accessorio sviluppato per Android deve comunicare con il sistema operativo tramite lo stesso protocollo.
A causa della limitata autonomia dei dispositivi su cui Android è maggiormente utilizzato (smarthphone e tablet), l'AOA impone che l'accessorio funga da host e alimenti quindi il bus utilizzato.[4]
A partire dal 2011, Google ha rilasciato open source gli schemi e l'implementazione della libreria ADK per tutti i dispositivi Arduino compatibili, così da fornire un'interfaccia comune per garantire la comunicazione con i dispositivi Android, previa un'opportuna configurazione. Android distingue diversi accessori sulla base delle informazioni da loro forniti durante la connessione preliminare.
Queste informazioni possono essere distinte nei seguenti valori:

- Vendor
- Name
- Longname

- Version
- Url
- Serial

Il dispositivo Android accetterà la richiesta di connessione solo se le stringhe identificative in suo possesso combaciano con quelle fornite dall'accessorio. Per ottenere effettivamente la connessione bisogna indicare ad Android quali sono gli accessori supportati e quale applicazioni è in grado di gestire un particolare accessorio. Questo si ottiene inserendo la successiva direttiva nel Manifest file dell'app

Codice 3.1: Porzione del Manifest file dell'app

```
...
<meta-data
    android:name="android.hardware.usb.action.
                                USB_ACCESSORY_ATTACHED"
    android:resource="@xml/usb_accessory_filter" />
...
```

Alla connessione di qualsiasi accessorio, il sistema operativo sarà incaricato di suggerire ed aprire l'apposita app in grado di gestire l'accessorio connesso. L'associazione con un particolare accessorio viene verificato tramite il file usb_accessory_filter.xml associato.

Nel nostro caso il file usb_accessory_filter.xml si presenta così:

Codice 3.2: usb_accessory_filter.xml

```
<resources>
    <usb-accessory
            version="0.1.0"
```

```
            model="Mobile-Tanker"
            manufacturer="Simone Mariotti"/>
</resources>
```

Le stesse identiche stringhe saranno impostate durante la fase di configurazione di Arduino in modo da permette l'accoppiamento.

3.3 ADK Toolkit

L'ADK Toolkit è una libreria che aggiunge un grado di astrazione all'ADK, semplificando l'inizializzazione della connessione, l'invio e la ricezione dei messaggi.

Il Toolkit si basa su due classi principali: AdkManager e AdkMessage. AdkManager espone metodi per la gestione della connessione e per l'invio e la ricezioni di dati. AdkMessage rappresenta il messaggio ricevuto dall'accessorio; tramite dei metodi ausiliari, *getString()*, *getFloat()*, *getInt()*, è possibile ottenere il typecast del messaggio ricevuto, normalmente rappresentato da un array di byte. Il recupero del messaggio originale può essere ottenuto grazie ai metodi *getBytes()* e *getByte()*.
Grazie a questa libreria l'uso dell'ADK, originariamente poco intuitivo, diventa efficiente ed elegante.

Codice 3.3: Inizializzazione della connessione con l'accessorio

```
private AdkManager mAdkManager;

@Override
protected void onCreate(Bundle savedInstanceState) {
```

```
    ...
    mAdkManager = new AdkManager(this);
}

@Override
protected void onResume() {
    super.onResume();
    mAdkManager.open();
}
```

Codice 3.4: Lettura e scrittura dati

```
// Write
adkManager.write("Ciao da Android!");

// Read
AdkMessage response = adkManager.read();
System.out.println(response.getString());
// Esempio di output: "Ciao da Arduino!"
```

Codice 3.5: Chiusura della connessione con l'accessorio

```
@Override
protected void onDestroy() {
    ...
    mArduino.close();
}
```

Capitolo 4

Implementazione

È definito “agente” una qualunque entità in grado di percepire l’ambiente circostante attraverso dei sensori e di eseguire delle azioni attraverso degli attuatori. Nel campo dell’intelligenza artificiale è definito intelligente quell’agente che “fa la cosa giusta al momento giusto”. Il compito dell’agente è quello di stabilire una sequenza di azioni che portano al raggiungimento di uno stato obiettivo.[2]

Il successivo diagramma descrive in linea di massima, il funzionamento dell’algoritmo che governa l’agente: preso un frame e un messaggio di Arduino restituisce una risposta generata dalla logica dell’app.

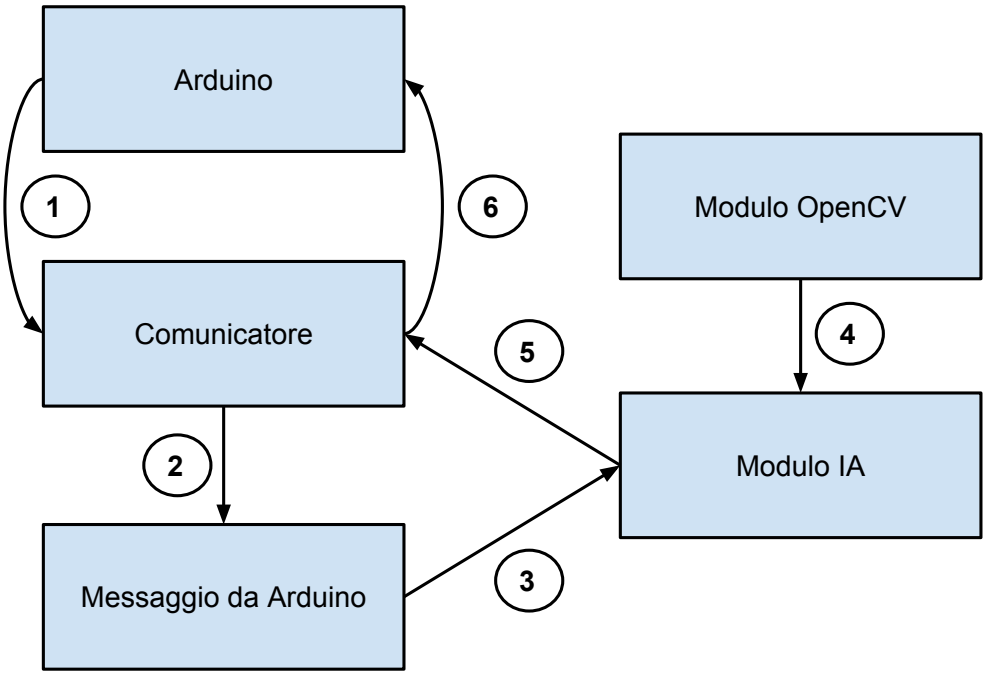

Figura 4.1: Processo decisionale del sistema

Le fasi principali sono:

1. ricezione del messaggio da Arduino.
2. salvataggio e decodifica del messaggio.
3. lettura del messaggio da parte del modulo di IA.
4. acquisizione dell'obiettivo dall'ambiente.
5. preparazione del comando elaborato dall'IA.
6. invio del comando ad Arduino.

4.1 Android

L'app per Android è stata sviluppata avendo come priorità la modularità. Difatti è previsto un meccanismo per alterare il comportamento dell'agente, sostituendo o modificando una singola classe senza coinvolgere il resto del codice.

La modularizzazione a grana grossa viene fatta a livello di pacchetto[1]. Ci sono tre pacchetti, ognuno con un compito preciso, **logic**, **messaging** e **opencv**.

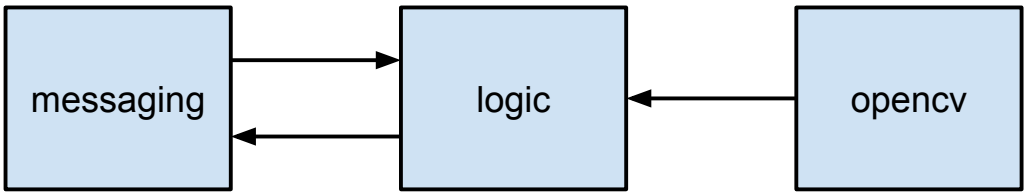

Figura 4.2: Schema di interconnessione dei pacchetti

4.1.1 Il pacchetto *opencv*

Il pacchetto **opencv** si occupa di analizzare i frame proveniente dalla camera alla ricerca dell'obiettivo, comunicando poi la posizione dell'obiettivo o l'assenza dello stesso alla classe RobotLogic del pacchetto **logic**.

La classe *ColorBlobDetector*

ColorBlobDetector è la classe che effettivamente cerca l'obiettivo nei frame proveniente dalla camera. L'obiettivo gli viene fornito da RobotActivity tramite *setTargetHsvColor()* nel momento in cui l'utente lo indica sull'in-

[1]Un contenitore che racchiude classi che svolgono compitini affini.

terfaccia utente. I colori in OpenCV vengono rappresentati con il tipo di dato interno Scalar che si comporta in modo simile ad un array. Il metodo che esegue l'operazione di ricerca è *process()*. Per prima cosa converte lo schema di colori del frame sotto analisi da RGBa[2] a HSV[3] e filtra l'immagine in modo che rimangano solo i pixel che hanno un colore che si trova entro un certo intorno del colore cercato. Viene poi chiamata la funzione di OpenCV *findContours()* che cerca i contorni delle zone con il colore di nostro interesse e li restituisce come una ArrayList di MatOfPoint.
MatOfPoint è un tipo di dato di OpenCV che è usato per memorizzare una quantità variabile di punti che sono in relazione tra di loro, come quelli che delineano un contorno. In seguito *process()* cerca il contorno che ha area maggiore e scarta tutti quelli che sono più piccoli della soglia impostata ad un decimo del maggior valore di area così trovato. I contorni rimasti saranno quelli analizzati da TargetSearch. La scelta del limite è seguita a delle prove empiriche che hanno mostrato come questa soglia permetta di scartare la quasi totalità di falsi positivi dovuti a riflessi o errori di rilevamento del colore.

La classe *TargetSearch*

Il metodo *searchColours()* di questa classe per prima cosa invoca *process()* di ColorBlobDetector passandogli il frame da analizzare e raccoglie i risultati tramite *getContours()* che restituisce i contorni individuati da *process()*.

[2]Ogni colore è rappresentato da quattro canali: R (red), G (green), B (blue) e a (alpha).

[3]Anche in questo schema i colori sono rappresentati da quattro canali ma hanno denominazione e significato differenti: H (hue), S (saturation), V (value) e a (alpha).

Scorre poi uno ad uno i contorni così ricevuti e trova per ognuno un *bounding rectangle*, cioè il rettangolo di area minima che contiene per intero il contorno corrente, scartando tutti i rettangoli minori di un certo valore e mantenendo un riferimento a quello di area maggiore trovato fino a quel momento. Sull'immagine visualizzata i rettangoli saranno tutti colorati di blu tranne quello di area maggiore che sarà colorato di rosso. Sarà proprio questo rettangolo il nostro obiettivo.
Fornisce un feedback all'utente sul colore scelto e la gamma di colori simili che sono stati cercati. Solo tre dati forniti da questa classe servono alla RobotLogic per valutare l'azione da intraprendere: la posizione dell'obiettivo, la larghezza del frame e il *bounding rectangle* che racchiude l'obiettivo.
Le posizioni sono sempre riferite all'asse x in quanto l'agente non ha capacità di movimento sull'asse y e possono essere:

- TARGET_POSITION_LEFT: il centro dell'obiettivo si trova a sinistra del centro del frame.
- TARGET_POSITION_RIGHT: il centro dell'obiettivo si trova a destra del centro del frame.
- TARGET_POSITION_FRONT: il centro dell'obiettivo e il centro del frame coincidono.
- TARGET_POSITION_NONE: l'obiettivo non è visibile.

In base alla posizione dell'obiettivo chiama il metodo adatto della classe UpdateDirection del pacchetto **logic** per mostrare, sul display, l'azione che sta per eseguire l'agente.

4.1.2 Il pacchetto *messaging*

Questo pacchetto è un modulo completamente a se stante, è così possibile cambiare il protocollo di comunicazione senza alterare la logica dell'agente; al suo interno avviene la codifica e la decodifica di tutte le stringhe inerenti i messaggi ricevuti e inviati.
Nel nostro caso i messaggi sono formati da tre byte separati da virgole. I tre valori potranno quindi assumere 256 valori diversi. I messaggi diretti da Arduino ad Android hanno questa forma:

MessageType,Message,Data

dove MessageType può assumere due valori: INFO, che indica che il messaggio trasporta un dato informativo, o STATE, che indica che il messaggio comunica lo stato in cui si trova Arduino.
Se il MessageType è INFO allora il Message può essere di due tipi:

- DISTANCE: sulla porzione Data del messaggio sarà presente la distanza del primo oggetto di fronte all'agente.
- TERMINATE: indica l'intenzione di Arduino di chiudere l'app Android.

Se il MessageType è STATE allora il Message può essere di quattro tipi:

- IDLE: l'agente non sta eseguendo alcuna azione.
- SEARCHING: l'agente si sta muovendo in cerca dell'obiettivo.
- HUNTING: l'agente si sta muovendo verso l'obiettivo.

- EMERGENCY: l'agente è in emergenza, non eseguirà nessun comando fino al termine della situazione di emergenza.

I messaggi diretti da Android ad Arduino hanno invece la forma:

Command,Parameter1,Parameter2

in cui Command identifica l'azione da eseguire, Parameter1 e Parameter2 sono due parametri opzionali che modificano il comportamento standard dell'azione richiesta.

La classe *IncomingMessage*

Questa classe è un *singleton*[4] che estende la classe Observable e insieme alla classe RobotLogic del pacchetto **logic** implementa il design pattern "Observer-Observable".
Accoglie il messaggio ricevuto da Arduino, crea un oggetto di tipo ArduinoMessage per decodificare e custodire il messaggio ricevuto e informa gli Observer della presenza di un nuovo messaggio tramite *triggerObservers()*. Su richiesta fornisce l'ultimo messaggio ricevuto tramite *getIncoming()*.

La classe *ArduinoMessage*

In questa classe sono definite tutte le costanti necessarie per la decodifica delle stringhe ricevute da Arduino. Decodifica la stringa passatagli come parametro in fase di istanziazione ed espone diversi metodi che consentono di fare un controllo veloce sul contenuto del messaggio. Un esempio della versatilità di questi metodi pubblici è:

[4]È un design pattern descritto dalla cosiddetta "Gang of four" nel libro "Design Patterns". Permette la creazione di una sola istanza della classe e ne regola l'accesso.

Codice 4.1: Metodo *update()* di RobotLogic del pacchetto **logic**

```
@Override
public void update(Observable observable, Object data) {
    ArduinoMessage incomingMessage =
        IncomingMessage.getInstance().getIncoming();
    if (!incomingMessage.hasError()) {
        if (incomingMessage.isInfoMessage() &&
            incomingMessage.hasDistance()) {
            mDistance = incomingMessage.getData();
        }
        if (incomingMessage.isTerminateCommand()) {
            mRobotActivity.finish(); }
}
```

La classe *MessageEncoder*

In questa classe sono definite tutte le costanti necessarie per la codifica delle stringhe da inviare Arduino ed espone numerosi metodi statici per la codifica vera e propria:

- *moveForward(int motorLeft, int motorRight)*: comanda di muoversi in avanti e applicare ai motori le potenze indicate da motorLeft e motorRight
- *moveForward()*: comanda di muoversi in avanti e applicare ai motori potenza pari a DEFAULT_VELOCITY ad entrambi i motori.

- *moveBackward(int motorLeft, int motorRight)*: comanda di muoversi in indietro e applicare ai motori le potenze indicate da motorLeft e motorRight
- *moveBackward()*: comanda di muoversi in indietro e applicare ai motori potenza pari a DEFAULT_VELOCITY ad entrambi i motori.
- *stop()*: comanda di fermarsi.
- *turnLeft(int velocity, int mode)*: comanda di ruotare verso sinistra e applicare ai motori potenza pari a *velocity*.
- *turnLeft(int mode)*: comanda di ruotare verso sinistra e applicare ai motori potenza pari a DEFAULT_VELOCITY.
- *turnRight(int velocity, int mode)*: comanda di ruotare verso destra e applicare ai motori potenza pari a *velocity*.
- *turnRight(int mode)*: comanda di ruotare verso destra e applicare ai motori potenza pari a DEFAULT_VELOCITY.
- *search()*: comanda di cercare l'obiettivo.

Nei metodi *turnLeft* e *turnRight*, *mode* può assumere il valore TURN_ON_SPOT o TURN_NORMALLY che fanno girare l'agente rispettivamente utilizzando entrambi i cingoli o solo il cingolo opposto alla direzione di rotazione.
Grazie a questi metodi pubblici la parte di logica dell'app invia messaggi ad Arduino senza avere la minima idea della stringa che effettivamente sarà inviata.
Tutto quello che si limita a fare per comandare, per esempio, ad Arduino di girare sul posto verso destro e con una velocità di 200 è:

Codice 4.2: Esempio di codifica e invio di un comando ad Arduino

```
Communicator.setOutgoing(MessageEncoder.turnRight(200,
    MessageEncoderDecoder.TURN_ON_SPOT));
```

La classe *Communicator*

Questa classe utilizza degli "short-lived thread"[5] per implementare la comunicazione da e verso Arduino.
Il thread lanciato ad intervalli regolati è definito da CommunicatorThread, che implementa l'interfaccia Runnable e definisce il metodo *run()*; nel metodo *run()* avviene la vera e propria comunicazione con Arduino. Ad ogni esecuzione di *run()* invia un messaggio ad Arduino, se non è presente un nuovo messaggio reitera quello precedente.
Successivamente il thread resta in attesa di un messaggio da Arduino e, se quest'ultimo è diverso dal precedente, allora lo passa a IncomingMessage che lo decodificherà e notificherà i propri Observer della presenza di un nuovo messaggio, altrimenti lo ignora.
Il messaggio da inviare è memorizzato nel campo privato mOutgoing che viene di volta in volta impostato da RobotLogic tramite il "setter"[6] *setOutgoing()*. L'uso degli "short-lived thread" in Java è piuttosto semplice:

Codice 4.3: Metodo di inizializzazione degli short-lived thread in Communicator

```
private ScheduledExecutorService mScheduler;
...
```

[5]Threads che, una volta lanciati, terminano rapidamente.
[6]Metodo che fornisce l'accesso in scrittura ad un campo privato di un oggetto.

```
public void start() {
    CommunicationThread thread = new CommunicationThread();
    mScheduler = Executors.newSingleThreadScheduledExecutor();
    mScheduler.scheduleAtFixedRate(thread, 0,
        READING_POLLING_TIME, TimeUnit.MILLISECONDS);
}
```

Ottenuto da Java il riferimento allo ScheduledExecutorService si imposta l'esecuzione ad intervalli regolari di READING_POLLING_TIME millisecondi del nostro CommunicatorThread.
La *read()* fornita dall'ADK è bloccante, quindi finché Arduino non invia un messaggio, e questo è disponibile sul buffer di lettura, l'esecuzione di *run()* resta appesa. Questo comportamento potrebbe portare all'indesiderata situazione in cui un messaggio in attesa di essere inviato ad Arduino venga sovrascritto dal successivo messaggio elaborato da RobotLogic. Per evitare questo scenario Arduino invia ad ogni iterazione del proprio ciclo *loop()* almeno un messaggio ad Android, che, in assenza di situazioni particolari, sarà la lettura proveniente dal sensore di distanza; in questo modo oltre ad evitare il blocco del thread, si invia ad Arduino un'informazione comunque utile per la navigazione.

4.1.3 Il pacchetto *logic*

Si occupa di reperire le informazioni dal mondo reale e analizzarle al fine di eseguire l'azione più adatta. Contiene tre classi: RobotActivity, RobotLogic e UpdateDirections.

La classe *RobotActivity*

Come suggerisce il nome è l'Activity vera e propria, cioè quella classe che il sistema operativo istanzia all'avvio dell'app. Essa stessa istanzia e prepara tutti gli altri oggetti per l'esecuzione. Implementa due interfacce: View.OnTouchListener e CvCameraViewListener. La prima permette di gestire gli input da touch screen, la seconda è un'interfaccia presente nella libreria OpenCV e permette di "intercettare" i frame provenienti dalla camera, prima che vengano renderizzati a schermo, tramite l'override[7] del metodo *OnCameraFrame()*.

All'avvio si occupa di inizializzare OpenCV tramite la callback *BaseLoaderCallback* e ottiene il riferimento all'istanza dell'ADK tramite l'ADKToolkit, inizializza il Communicator e passa il riferimento di quest'ultimo a TargetSearch durante l'esecuzione.

È buona norma interrompere le connessioni e liberare il canale usato nel momento in cui un'app viene chiusa. Per questo nel metodo *onDestroy()* viene chiuso il canale usato per comunicare con Arduino tramite il metodo *close()* fornito dall'ADKToolkit e viene fermato lo scheduler che esegue il Communicator

In aggiunta alle normali funzioni di un'Activity, RobotActivity integra un listener per gli eventi touch localizzati all'interno della porzione di interfaccia utente che mostra il video proveniente dalla camera. In tal caso, viene invocato il metodo *onTouch()* ereditato da View.OnTouchListener che esegue la media dei colori presenti in una regione di 8x8 pixel intorno alle coordinate in cui è avvenuto il tocco da parte dell'utente. Il colore calcola-

[7] Tecnica che permette di ridefinire il comportamento di un metodo ereditato.

to, che rappresenta l'oggetto ricercato dall'agente nella scena, sarà passato alla classe TargetSearch del pacchetto **opencv** tramite il metodo pubblico di quest'ultima *setTargetHsvColor()*.

La classe *UpdateDirections*

Questa classe è un *singleton* e si occupa di mostrare all'utente tramite immagini e testi quello che l'agente sta facendo e quale sarà la sua prossima mossa. Dovendo agire sull'UI[8] deve essere eseguita sul thread che si occupa dell'UI, per questo implementa l'interfaccia *Runnable* e ogni sua esecuzione è lanciata tramite *runOnUiThread()*. Le informazioni visualizzabili sono limitate e ben distinte, ad ognuna corrisponde un metodo da invocare per visualizzare quell'informazione a schermo. I metodi disponibili sono:

- *left()*: indica che l'obiettivo è visibile, è stato individuato e si trova alla sinistra dell'agente.
- *right()*: indica che l'obiettivo è visibile, è stato individuato e si trova alla destra dell'agente.
- *aimed()*: indica che l'obiettivo è visibile, è stato individuato e si trova esattamente di fronte all'agente che si muoverà in quella direzione.
- *search()*: indica che l'obiettivo non è visibile, l'agente si muoverà secondo l'algoritmo di ricerca.
- *found()*: indica che l'obiettivo è visibile e l'agente si trova a meno di 30 centrimetri.

[8] User Interface, interfaccia utente.

- *avoidingLeft()*: indica che è presente un ostacolo sul percorso dell'agente a meno di 30 centimetri. L'agente lo aggirerà verso sinistra.
- *avoidingRight()*: indica che è presente un ostacolo sul percorso dell'agente a meno di 30 centimetri. L'agente lo aggirerà verso destra.
- *chooseColor()*: indica che l'agente è in attesa che venga impostato il colore da cercare.

La classe espone anche altri metodi che mostrano (*show()*) o nascondono (*hide()*) la porzione di interfaccia che visualizza le indicazioni, oppure bloccano (*lock()*) e sbloccano (*unlock()*) la possibilità di cambiare le indicazioni visualizzate.

La classe astratta *BaseAi* e l'interfaccia *IBaseAi* del pacchetto *ai*

Forniscono la base di partenza per la classe RobotLogic.
L'interfaccia ha due metodi: *targetPosition()* e *think()*. BaseAi sfrutta l'interfaccia IBaseAi e implementa il metodo *targetPosition()*. Tramite questo metodo, TargetSearch fornisce all'IA la larghezza del frame, la posizione dell'obiettivo e il *buonding rect* dell'obiettivo; a partire da quest'ultimi saranno calcolati i valori di *mFrameWidth*, *mTargetCenter*, *mTargetInSight*, *mTargetDirection* e *mTargetWidth* che saranno utilizzati in *think()*.
Questa organizzazione fa sì che i comportamenti e i dati indispensabili per una classe di IA siano definiti in BaseAi e quindi ogni nuova classe nata per sostituire o migliorare la logica attuale dovrà estendere BaseAi e implementare *think()*

La classe *RobotLogic*

La classe RobotLogic racchiude l'algoritmo che governa il robot.
Viene chiamata in causa ogni volta che arriva un nuovo messaggio da Arduino. Per far questo si è usato il design pattern "Observer-Observable" in cui l'"Observer" è questa stessa classe e l'"Observable" è la classe IncomingMessage del pacchetto **messaging**.
Alla ricezione di ogni messaggio viene invocato il metodo *getIncoming()* della classe IncomingMessage che fornisce un'istanza di ArduinoMessage che contiene la decodifica del messaggio appena ricevuto. Se il messaggio contiene informazioni sulla distanza del primo oggetto nella direzione dell'agente allora tale distanza viene salvata nell'istanza in modo che successivamente si possa usare per valutare in modo più preciso lo stato dell'ambiente.
Il metodo principale della classe è *think()* ed è invocato ogni volta che dal pacchetto **opencv**, e in particolare dalla classe TargetSearch, giunge un aggiornamento sull'obiettivo, la sua eventuale presenza in scena e la sua posizione. Viene effettivamente eseguito *think()* solo se è stato premuto il pulsante *Start* presente sull'interfaccia utente; questo metodo come prima cosa, stabilisce in che fase l'iterazione precedente ha portato il sistema.
Le possibilità sono:

- "Cheer Phase": indica che l'obiettivo è stato trovato e l'agente sta girando su se stesso per segnalare la fine della ricerca.
- "Avoiding Phase": indica che sulla traiettoria dell'agente si è presentato un ostacolo e lo sta aggirando. Si compone di tre sottofasi:

  - Fase 1: scegliere in modo casuale una direzione tra destra e sinistra e ruota di circa 90° in quella direzione.
  - Fase 2: si muove in avanti per un tempo prestabilito.
  - Fase 3: ruota di 90° nella direzione opposta a quella scelta nella fase 1.

- "Search Phase": indica che l'obiettivo non è stato ancora trovato e l'agente si sta muovendo per trovarlo.

Se si trova nella "Cheer Phase" ignora ogni comunicazione proveniente da Arduino, finisce la rotazione di segnalazione e invoca il metodo *reset()* di RobotActivity per preparare il sistema all'inserimento di un nuovo obiettivo da parte dell'utente.
Se si trova nella "Avoiding Phase" esegue in successione le tre fasi. Si può interrompere solo se durante le manovre di aggiramento dell'ostacolo l'obiettivo compare nella scena.
Se si trova nella "Search Phase" controlla se l'obiettivo è presente in scena e si trova a meno di 30 centimetri, in caso di risposta affermativa ferma l'agente, interrompe l'esecuzione di *think()* e attiva la "Cheer Phase". Se l'obiettivo non è in vista ma c'è un oggetto a meno di 30 centimetri ferma l'agente e attiva la "Avoiding Phase".
Se l'obiettivo non è visibile rimane in "Search Phase" e si muove in avanti, saranno poi i bordi dell'ambiente di sperimentazione ad innescare la modalità di emergenza di Arduino e causare quindi una rotazione dell'agente.
Se l'obiettivo è visibile ma si trova a più di 30 centimetri effettua le correzioni di rotta necessarie per portarlo in direzione di marcia.
Se l'obiettivo è precisamente di fronte all'agente si muoverà in quella dire-

zione.

La rotazione necessaria per portare l'obiettivo esattamente di fronte all'agente implica movimenti lenti e precisi che è difficile ottenere con i motori DC di cui è fornito l'agente. Lo scenario tipico è che i motori, impostati ad una certa potenza, si sforzano per ruotare l'agente senza riuscirci. Aumentare la potenza, anche di un piccolo quantitativo, fa ruotare l'agente velocemente. Purtroppo l'energia richiesta per mettere in rotazione l'agente cambia in funzione di troppe variabili: la direzione di rotazione, l'uso di entrambi i cingoli o solamente uno, la presenza di piccoli ostacoli sotto l'agente. Per ovviare a questo problema che impedisce una corretta movimentazione dell'agente quando sono necessari movimenti precisi è stata usata una soluzione adattiva che permette all'agente di trovare sempre la minima energia necessaria per muoversi. Per far questo l'agente prende come riferimento la posizione dell'obiettivo e tenta di ruotare, se all'iterazione successiva non nota uno spostamento relativamente all'obiettivo, allora aumenta l'energia inviata ai motori. L'incremento di energia viene arrestato non appena l'agente riesce a muoversi; questo produce una rotazione fluida e controllata.

4.2 Arduino

La prima operazione effettuata su Arduino è l'inizializzazione dell'ADK fornendo version, model e manufacturer identici a quelli indicati in usb_accessory_filter.xml nell'app Android.

Nel *setup()* avviene la configurazione dei pin utilizzati come input o output e si inizializza la connessione seriale a 115200 bps.

In Arduino il metodo principale è *loop()*, che come dice il nome stesso viene eseguito in ciclo infinito. Gli stati assegnati agli output sono persistenti attraverso le varie iterazioni del ciclo. La prima operazione fatta è il controllo dei sensori di riflessività: se dovessero essere in stato di "warning" si interrompe l'esecuzione del *loop()* forzandolo a passare all'iterazione successiva e si fa un altro controllo; se i sensori sono ancora in "warning" allora si innesca la modalità "emergency". Durante questa fase la sicurezza dell'agente è prioritaria: vengono ignorati tutti i messaggi proveniente da Android fino alla risoluzione della situazione di emergenza.
Le situazioni di emergenza sono gestite dal metodo *emergency()* e possono essere di tre tipi:

- Allarme sul sensore sinistro: esegue una breve retromarcia e ruota lentamente sul posto verso destra.
- Allarme sul sensore destro: esegue una breve retromarcia e ruota lentamente sul posto verso sinistra.
- Allarme su entrambi i sensori: esegue una breve retromarcia e ruota velocemente sul posto in una direzione scelta casualmente.

Il resto del metodo *loop()* è interamente compreso dentro un *if* che controlla se Arduino è collegato ad una periferica in grado di comunicare secondo il protocollo Android Open Accessory (AOA).
Successivamente Arduino controlla se è disponibile un messaggio inviato da Android, procedendo in tal caso alla decodifica ed all'esecuzione del comando ricevuto. In caso contrario entra in uno stato "conservativo" in cui è connesso ad una periferica ma non sta ricevendo comandi. L'accesso a

questo stato provoca l'immediata fermata dell'agente in quanto, dato che i motori continuerebbero ad eseguire l'ultimo comando inviato, si muoverebbe senza controllo esponendosi a dei pericoli.
La decodifica del comando è effettuata da *decodeCommand()* che divide la stringa nelle sue tre parti fondamentali command, param1, param2. Controlla che command sia un comando conosciuto e, nel caso che il comando preveda dei parametri aggiuntivi, che param1 e param2 rientrino negli intervalli previsti. In seguito tramite un costrutto *switch* applicato a command esegue le azione previste per quel dato comando o comando-parametri ricevuti. Per rafforzare l'approccio conservativo sopra citato, se il comando ricevuto non riesce ad essere decodificato correttamente si fermano i motori finché non arriva un comando legittimo.
Alla fine del ciclo *loop()*, indipendentemente dal fatto che sia stato ricevuto un messaggio corretto, viene attivato il sensore di distanza a ultrasuoni, calcolata la distanza in centimetri con la formula illustrata precedentemente nella sezione 2.3.2 e inviata ad Android come messaggio di tipo INFO.
Il modulo Arduino include, inoltre, le seguenti funzioni con le rispettive responsabilità:

- *setDirection()*: i motori sono collegati rispettando la stessa polarità: dando la stessa potenza ad entrambi ruoteranno nello stesso verso. Questo, considerando che sono montati in modo speculare sui due cingoli, comporterà una rotazione sul posto per l'agente. Per ovviare il problema questo metodo riceve due ha come parametri *side* e *direction*: *side* identifica su quale cingolo si vuole impostare la direzione il cui valore può essere LEFT, RIGHT, BOTH mentre *direction* indica la

direzione che si vuole impostare e può assumere i valori FORWARD e BACKWARD. Con queste informazioni sarà il metodo ad impostare le direzioni corrette ai motori per far sì che l'agente vada nella direzione effettivamente desiderata.

- *brake()* e *releaseBrake()*: rispettivamente attivano e disattivano i freni dei motori. L'Arduino Motor Shield modula il segnale inviato ai motori in modo da farli fermare velocemente.

- *moveForward()* e *moveBackward()*: impostano la direzione di marcia desiderata con *setDirection()* e attivano i motori alle potenze ricevute come parametri.

- *turnLeft()* e *turnRight()*: attivano il cingolo opposto alla direzione di rotazione richiesta alla potenza ricevuta per parametro. Se il parametro mode è impostato su TURN_ON_SPOT significa che è stata richiesta da Android la rotazione sul posto e quindi oltre al cingolo sopracitato attiva anche l'altro con pari potenza ma direzione opposta.

- *stop()*: disattiva i motori. Può effettuare una fermata HARD o una SOFT. Nel primo caso usa *brake()* per diminuire il tempo di frenata mentre nel secondo il robot sarà arrestato dall'attrito con la superficie.

- i metodi *emergency()*, *sensorsOk()*, *FRIsOut()* e *FLIsOut()* lavorano tutti all'unisono per evitare che l'agente esca dall'area di test: leggono costantemente i valori di riflessività rilevati dai sensori QRD1114 e agiscono di conseguenza come illustrato all'inizio della presente sezione.

Conclusioni

Il robot riesce a distinguere colori che siano distanti tra loro 10° nel modello di colori HSV, il che permette di cercare 36 colori differenti.

Riesce a trovare adeguatamente l'obiettivo seppur senza seguire un percorso efficiente data l'assenza di conoscenza dell'ambiente di sperimentazione. Molte situazioni di stallo si risolvono grazie alla randomizzazione di alcune parti del processo decisionale mentre altre, perlopiù avvenute in presenza di due oggetti nelle vicinanze dei bordi dell'ambiente, hanno richiesto un intervento esterno per la soluzione. L'affidabilità dimostrata nella ricerca è comunque tale che riteniamo tutti gli obiettivi prefissati per questo lavoro di tesi raggiunti in modo ottimale.

La grande modularità del software ha di fatto creato un semplice framework per l'implementazione di nuovi algoritmi di ricerca *context-aware*[9], da usare con una qualsiasi piattaforma composta da un'app Android ed un Arduino Due. Come visto nel Capitolo 4, l'estensione della classe BaseAi e l'implementazione del metodo *think()* è tutto ciò che serve per creare un algoritmo di IA funzionante.

[9]Che sono a conoscenza del contesto in cui operano.

Sviluppi futuri

Il software è stato rilasciato con licenza BSD è quindi migliorabile, modificabile e personalizzabile da futuri studenti.
Miglioramenti possibili sono:

- aggiungere sensori per la navigazione come GPS, bussole e giroscopi. Permetterebbero di eliminare la navigazione randomizzata e seguire un efficiente percorso di ricerca dell'obiettivo.
- implementare una base di conoscenza in grado di tenere traccia degli ostacoli e degli oggetti riconosciuti in modo da velocizzare ogni successiva ricerca.
- aggiungere un attuatore in grado di manipolare l'obiettivo una volta individuato.

Bibliografia

[1] Ambroglio M., Ray Tracing GPU-based - http://www.unibg.it/dati/corsi/38001/45691-lezione_06052011_optix.pdf

[2] Dolce F., Soluzione di problemi di direction finding con l'utilizzo di un algoritmo di posizionamento ad ancora singola, Pag. 16, Tesi di Laurea in Informatica, Università di Perugia, 2013

[3] Ghita O., Whelan P. F., A Systems Engineering Approach to Robotic Bin Picking, Dublino, 2008 - http://cdn.intechopen.com/pdfs-wm/5761.pdf

[4] Gupta G., Abraham J., Develop AOA USB Accessories For Android-Based Systems, 2013 - http://electronicdesign.com/embedded/develop-aoa-usb-accessories-android-based-systems

[5] Mann A., That Computer Actually Got an F on the Turing Test, 2014 - http://www.wired.com/2014/06/turing-test-not-so-fast/

[6] IEEE Global History Network, Nipkow Scanning Disk, 2013 - http://www.ieeeghn.org/wiki/index.php/Nipkow_Scanning_Disk

[7] Paci G., Paci R., La nascita della televisione, 2011 - `http://online.scuola.zanichelli.it/fare/files/2008/04/Paci_5985_09_Nascita_televisione.pdf`

Elenco delle immagini