Università degli Studi di Padova

Corso di Laurea Triennale in Ingegneria Informatica

Tracking in Android per NXT

30 settembre 2013

Relatore: Michele Moro Studente : Alberto Gatto

Anno Accademico 2012/2013

Alla mia famiglia,
per aver saputo convivere
con la mia incompiutezza.

Sommario

Questa tesi è dedicata allo sviluppo di un applicativo per smartphone Android che esegua il tracking di un oggetto e governi un brick NXT con lo scopo di inseguirlo. Il problema del video tracking è stato e continua tuttora ad essere di primaria importanza nel panorama informatico a causa delle sue innumerevoli applicazioni. Uno sguardo verso il futuro ci lascia inoltre presuppore che l'utilizzo di metodi per individuare un oggetto (o più di uno) grazie all'uso di una telecamera sia destinato ad aumentare notevolmente e termini come "realtà aumentata" cominciano ad essere familiari tra l'opinione pubblica. La diffusione capillare di potenti computer "mobili" quali sono gli smartphone apre, in relazione al problema del video tracking, un numero infinito di possibilità; sono già presenti librerie specializzate nella computer vision per i sistemi operativi mobili più famosi. Un ambito fortemente interessato all'object tracking è sicuramente la robotica; il kit robotico programmabile rilasciato dalla Lego e denominato LEGO Mindstorms NXT risulta probabilmente l'esempio più fortunato del tentativo di sdoganare dai laboratori scientifici l'utilizzo e la sperimentazione coi robot. In questo contesto si inserisce il presente lavoro di tesi che si prefigge di utilizzare le librerie grafiche OpenCV e in particolare l'algoritmo Camshift per l'individuazione di un oggetto ripreso dalla telecamera di uno smartphone Android; quest'ultimo avrà il ruolo di sensore di posizione con il compito di istruire il robot, sfruttando la connettività Bluetooth, sugli spostamenti da compiere per inseguire l'obiettivo. Si vedrà come questa configurazione, sebbene funzionante, dimostri sensibili lacune dovute alla relativamente limita capacità di calcolo dello smartphone e dal metodo di video tracking scelto.

Indice

1 Introduzione

Si consideri il comparto hardware di cui sono dotati i moderni smartphone:

- Processori multi-core: i dual core rappresentano ormai lo standard mentre i modelli di fascia alta montano processori quad o addirittura octa core;
- RAM: dal gigabyte fino ai 2/3 ipotizzando per i prossimi modelli 4 GB o più;
- Memoria fisica: dai pochi GB alle decine ed in continua espansione.

A fronte di queste caratteristiche si possono ragionevolmente considerare alla stregua di veri e propri computer, ciò ha destato l'interesse di una grande quantità di sviluppatori software e di ricercatori che hanno dato vita a numerosi progetti in cui queste nuove potenzialità sono state impiegate in svariati campi applicativi. Uno di questi non poteva che essere la robotica infatti uno smartphone offre un sostanzioso comparto di connettività con interfacciamenti WiFi, Bluetooth, GSM/UMTS/HSPA+, NFC, USB oltre ad un buon numero di sensori come accelerometri, GPS, giroscopio, sensore di prossimità e addirittura barometro. Oltre all'impressionante elenco appena citato merita una menzione particolare dovuta alla natura di questa tesi, la presenza di fotocamere (anteriore e posteriore) di alta qualità e risoluzione (dell'ordine del megapixel e della decina di megapixels rispettivamente). Inoltre i sistemi operativi mobili con cui sono equipaggiati questi dispositivi offrono la possibilità di realizzare applicazioni sfruttando linguaggi di programmazione di alto livello come possono essere Objective-C, C#, Java o C++. Tutte queste considerazioni mostrano l'eccellente propensione di uno smartphone a diventare l'unità centrale di elaborazione di un robot o eventualmente ad interfacciarsi con un'unità già presente e sollevarla da carichi di lavoro o compiti eccessivamente onerosi. La pubblicazione intitolata *Using the Android Platform to control Robots* [6] è un ottimo esempio di quanto appena affermato ed è stata lo stimolo iniziale che ha dato il via al progetto qui esposto. In essa il comparto robot utilizzato è il LEGO Mindstorms NXT, presente anche nei laboratori del nostro dipartimento, a cui viene interfacciato uno smartphone Android; il risultato prova come questo sia un connubio vincente infatti il dispositivo Android viene utilizzato per svolgere programmi Java complessi che forniscono il sistema nel suo insieme di un comportamento sofisticato impossibile da ottenere con la potenza di calcolo messa a disposizione dal solo blocco NXT mentre dall'altro lato il robot LEGO completa il quadro sopperendo alla mancata possibilità di controllare

attuatori, come ad esempio dei servomotori, da parte dello smartphone. Tale pubblicazione offre inoltre una disamina dei possibili metodi di connessione e propone una libreria realizzata appositamente per rendere trasparente lo scambio di dati tra lo smartphone e l'NXT esponendo una API che nasconda le operazioni sottostanti; la tecnologia di rete scelta è Bluetooth e la motivazione risiede principalmente nel fatto che è un'interfaccia presente in pressochè la totalità degli smartphone e non necessita di cavi, come per esempio richiede l'USB. Partendo perciò dall'esperienza positiva che il lavoro appena citato porta con sè si è pensato di sfruttare la comprovata capacità di integrazione tra smartphone e brick NXT per ampliare le potenzialità del robot LEGO in un settore particolarmente importante: la computer vision (visione computazionale). La computer vision è quel campo di studio che si dedica ai metodi di acquisizione, elaborazione, analisi e comprensione di immagini provenienti dal mondo reale con lo scopo di estrarne informazione; essendo perciò un flusso video prodotto da una fotocamera una sequenza di immagini che si succedono con l'andare del tempo, esso rientra nell'ambito operativo della visione computazionale ed anzi ricopre un ruolo fondamentale in un sottodominio della stessa denominato video tracking. Quest'ultimo si occupa della localizzazione di un oggetto (o più di un oggetto) che si sposta nella scena con l'andare del tempo venendo ripreso da una fotocamera; processare il flusso video così prodotto è un compito esoso di risorse data l'ingente quantità di informazione contenuta nel video quindi inadatto al Mindstorms che offre potenza di calcolo limitata, inoltre per la maggior parte dei sistemi operativi mobili attualmente diffusi sono presenti porting di librerie estremamente performanti e specializzate nella computer vision quali sono le OpenCV. Un'altra eccellente pubblicazione intitolata *A Computer Vision Application to Accurately Estimate Object Distance* [7] mostra come eseguire object tracking, seppure non in ambito mobile, in modo efficace e discute una parte delle problematiche che andremo ad affrontare; prendendo spunto dalla suddetta lettura abbiamo individuato l'algoritmo migliore per tenere traccia dell'oggetto in relazione alle nostre esigenze: Camshift [1]. A valle di quanto finora detto troviamo perciò un robot LEGO Mindstorms NXT che supportato dal collegamento con uno smartphone Android è ora in grado di localizzare nella scena l'oggetto scelto come obiettivo; la considerazione immediatamente successiva riguarda le modalità con cui determinare gli spostamenti che il robot deve effettuare per inseguire il target in questione. Ci troviamo di fronte ad un problema di Visual servoing (Asservimento visivo o VS) che, considerando l'intrinseca difficoltà di tale compito e preferendo centrare questo lavoro di tesi principalmente sulla problematica del tracking, ci ha portati a sviluppare un algoritmo in cui vengono impartiti al robot gli spostamenti da compiere in base ad una semplice legge di controllo.

Nella Sezione 2 si fornisce una panoramica generale delle tecnologie usate partendo dal sistema operativo mobile Android ed in particolare dal suo Native Development Kit che ci ha permesso di utilizzare Java Native Interface, proseguendo per una descrizione del LEGO Mindstorms NXT e della Java Virtual Machine, denominata leJOS, sviluppata per il robot in seguito alla sua diffusione; si conclude introducendo le librerie OpenCV. La Sezione 3 esplicita il problema dell'Object Tracking dopo aver illustrato pochi rudimenti relativi alla computer vision, successivamente viene esaminata una prima soluzione per l'individuazione di un oggetto nella scena per poi giungere a Camshift [1], l'algoritmo che si è scelto di utilizzare; si conclude discutendo le modalità con cui istruire il robot sugli spostamenti necessari per inseguire l'oggetto bersaglio. Nella Sezione 4 si esamina passo passo, partendo dalla struttura generale per arrivare ai singoli componenti, l'applicativo sviluppato. Infine troviamo lo spazio dedicato alle conclusioni con un'ottica di riguardo verso possibili sviluppi futuri. Il codice sorgente dell'applicazione è fornito nell'appendice.

Obiettivi

L'obiettivo di questo lavoro di tesi è lo sviluppo di un applicativo per la piattaforma mobile Android che riesca ad eseguire il tracking di un oggetto e successivamente invii al blocco NXT le coordinate degli spostamenti da compiere per mantenere l'obiettivo centrato sulla scena. Per fare questo è necessario disporre di un metodo per determinare la posizione dell'oggetto nell'immagine inquadrata dallo smartphone. Il problema immediatamente successivo consiste nel mantenere traccia dell'oggetto lungo il susseguirsi dei frame e calcolarne lo spostamento relativo alla fotocamera. Si è deciso di risolvere il problema della localizzazione dell'obiettivo da inseguire fornendo la UI di un metodo di selezione manuale dell'area di interesse nella scena; il passaggio successivo consiste nell'utilizzo dell'algoritmo Camshift che esegue tracking basato sul colore ovvero insegue obiettivi basandosi su un campione iniziale di colori dei pixel. Infine si è sviluppato un algoritmo euristico per il calcolo degli spostamenti necessari a mantenere la distanza e la posizione iniziale dell'oggetto rispetto alla fotocamera.

2 Tecnologie utilizzate

2.1 Android

Android è un sistema operativo open source sviluppato principalmente per dispositivi touchscreen come smartphone e tablet, si basa attualmente sul kernel Linux 3.x e l'ultima versione stabile è la 4.3, nome in codice Jelly Bean, che è la piattaforma per la quale si è realizzata l'applicazione. Android è sviluppato da Google ed è stato svelato per la prima volta nel 2007, in concomitanza si è visto il rilascio del primo Software Development Kit (SDK) per gli sviluppatori (il quale includeva: strumenti di sviluppo, librerie, un emulatore e la documentazione).

Figura 1: Logo Android

La struttura del sistema illustrata, in Figura [2], vede alla base il kernel Linux già citato, a monte di tutto gli applicativi ed immediatamente sotto l'application framework che gli sviluppatori incorporano nei loro programmi. In aggiunta Android fornisce uno strato intermedio chiamato middleware che include librerie che forniscono servizi come salvataggio dei dati in memoria, accesso allo schermo, multimedia e navigazione web. Siccome le librerie dello strato middleware sono compilate in linguaggio macchina i servizi messi a disposizione sono estremamente veloci inoltre esse implementano funzioni specifiche del dispositivo rendendo trasparente all'application framework il fatto che possa essere in esecuzione su terminali diversi tra loro.
Lo strato middleware contiene inoltre la virtual machine Dalvik e le sue librerie Java [3]. Android usa la macchina virtuale Dalvik con compilazione just-in-time per eseguire Dalvik 'dex-code' (Dalvik Executable) che solitamente è tradotto da Java bytecode. La piattaforma hardware principale è ARM tuttavia è attualmente in sviluppo il progetto Android-x86 che ha come fine l'allargamento del supporto all'architettura x86.

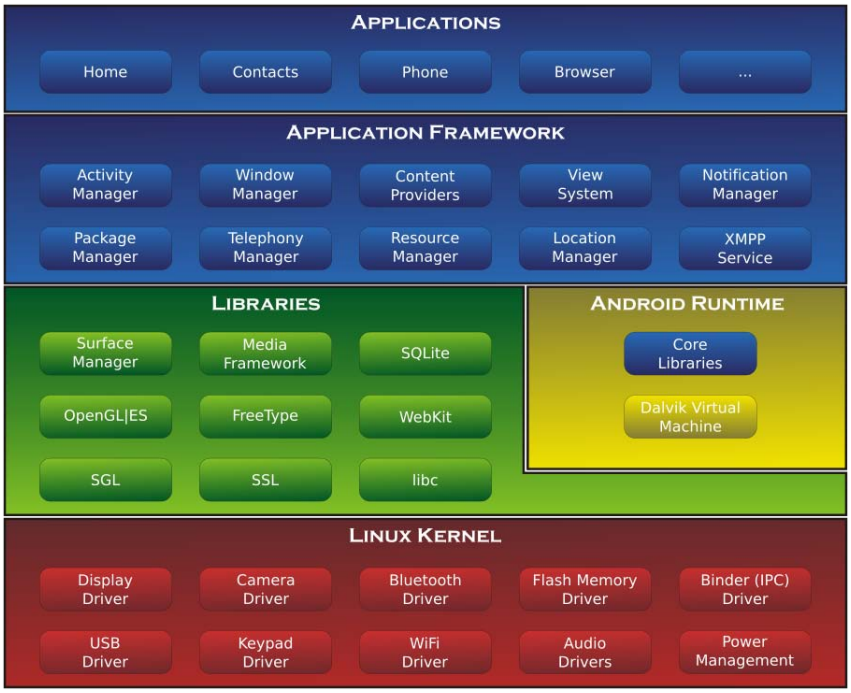


Figura 2: Architettura del sistema operativo Android

Le applicazioni Android sono scritte utilizzando il linguaggio di programmazione Java. Vari strumenti dell'Android SDK compilano il codice e producono un pacchetto Android ovvero un archivio con estensione `.apk`. Tutti i file sorgente contenuti in un `.apk` sono considerati come un'unica applicazione e l'archivio viene utilizzato per installarla sui vari dispositivi Android. Il codice di ogni singolo programma viene eseguito in isolamento rispetto alle altre applicazioni infatti ogni programma gode di una virtual machine Dalvik dedicata, questa scelta evita, nel caso di errori e malfunzionamenti locali, che essi si propaghino portando all'instabilità il sistema.

Le applicazioni in Android possono essere formate da quattro tipologie di componenti:

- Activities: una *activity* è costituita da una singola schermata dotata di un'interfaccia utente (UI);
- Services: un *service* è un componente che viene eseguito in background e svolge operazioni lunghe e/o non interattive per questo non è dotato di UI;
- Content providers: un *content provider* è dedicato alla gestione dei dati condivisi perciò agisce da supporto alla comunicazione tra applicazioni, non è dotato di UI;

- Broadcast receveirs: un *broadcast recevier* risponde ad eventi di sistema che riguardano l'intero dispositivo, non è dotato di UI.

Ogni applicazione prevede un file di manifesto (manifest.xml), dove vengono dichiarati i componenti della stessa, i requisiti software e hardware per il suo utilizzo e la lista dei permessi che si richiedono per eseguire il codice. In seguito si approfondisce la prima componente sopra introdotta in quanto strettamente collegata, a differenza delle rimanenti, con questo lavoro di tesi.

2.1.1 Activities

Un'activity è un componente dell'applicazione che fornisce una schermata con cui l'utente può interagire, per questo motivo gli viene data una finestra che tipicamente riempie lo schermo (anche se non è strettamente necessario) nella quale disegnare la sua interfaccia utente. Un applicativo in generale è composto da più activity accompagnate da altri componenti tra quelli che sono stati in precedenza elencati. Solitamente un'activity viene designata come principale comportandone la visualizzazione a schermo appena viene lanciata l'applicazione, ciò viene specificato nel manifesto precedentemente introdotto (manifest.xml) assieme alla lista delle activity che compongono l'applicativo. L'interfaccia utente è composta da una gerarchia di view, oggetti derivati dalla classe *View*; ogni view controlla un determinato rettangolo di spazio dentro la finestra dell'activity ed è predisposta all'interazione con l'utente (per esempio una view può essere un bottone che una volta premuto dà il via ad un'azione). Android fornisce un gran numero di view già pronte per l'uso che possono essere impiegate per costruire la UI; la disposizione delle view nell'interfaccia utente solitamente avviene appoggiandosi ad un apposito file XML il cui contenuto viene caricato in runtime oppure è possibile creare le necessarie view nel codice dell'activity. Quest'ultimo approccio è sconsigliato in quanto non permette di separare il design della UI dal codice dell'applicativo. Il ciclo vitale di una activity è caratterizzato da un percorso ben definito perciò essa può trovarsi solamente in uno dei seguenti stati:

- Resumed: l'activity è in primo piano sullo schermo ed ha il focus dell'utente (questo stato è spesso conosciuto come "running");

- Paused: parzialmente visibile a causa di un'altra activity che è adesso in primo piano, non può ricevere input dall'utente perché non ha il focus. In questo stato l'activity è ancora in esecuzione tuttavia può essere terminata in gravi casi di mancanza di risorse del sistema;

- Stopped: l'activity è totalmente oscurata da un'altra activity (ci si riferisce a questa situazione dicendo che la prima citata è in "back-

ground"). In questo stato l'activity è salvata in memoria, scollegata dal window manager e può essere terminata quando vi è necessità di memoria altrove.

Quando un'activity passa da uno stato all'altro a causa dell'interazione con l'utente o di un intervento del sistema operativo, viene notificata attraverso dei metodi callback; si può effettuare un ovverride di tali metodi in modo da svolgere le operazioni richieste dal cambio di stato. Il codice seguente mostro lo scheletro di un'activity ed in particolare i metodi callback sui quali effettuare un ovverride.

```
1  public class ExampleActivity extends Activity {
2      @Override
3      public void onCreate(Bundle savedInstanceState) {
4          super.onCreate(savedInstanceState);
5          // The activity is being created.
6      }
7      @Override
8      protected void onStart() {
9          super.onStart();
10         // The activity is about to become visible.
11     }
12     @Override
13     protected void onResume() {
14         super.onResume();
15         // The activity has become visible (it is now "resumed").
16     }
17     @Override
18     protected void onPause() {
19         super.onPause();
20         // Another activity is taking focus (this activity is about to be "paused
       ").
21     }
22     @Override
23     protected void onStop() {
24         super.onStop();
25         // The activity is no longer visible (it is now "stopped")
26     }
27     @Override
28     protected void onDestroy() {
29         super.onDestroy();
30         // The activity is about to be destroyed.
31     }
32 }
```

Listing 1: Scheletro di un'activity

Nel loro insieme questi metodi definiscono l'intero ciclo di vita di un'activity; è possibile riconoscere tre cicli annidati nel ciclo di vita principale:

- l'**intero ciclo di vita** è compreso tra le chiamate a *onCreate()* (dove avviene il setup dello stato globale come per esempio la definizione del layout della UI) e *onDestroy()* (dove vengono rilasciate tutte le risorse),

- il **tempo di vita visibile** è compreso tra le chiamate a *onStart()* e *onStop()*, in questo lasso di tempo l'attività è visibile a schermo e l'utente può interagirvi,

- il **tempo in foreground** è compreso tra le chiamate a *onResume()* e *onPause()*, in questo frangente l'attività è visualizzata sullo schermo al di sopra di tutte le altre activity quindi ha il focus e riceve direttamente input dall'utente. Un'activity può effettuare molte transizioni da e verso lo stato di foreground perciò il codice presente nei precedenti due metodi non dovrà impegnare il sistema operativo con grossi carichi di lavoro.

La Figura [3] illustra questi tre cicli annidati e le strade che un'activity può percorrere nella sua transizione tra stati (i rettangoli rappresentano i metodi di callback). Quando da un'activity se ne richiama un'altra entrambe sperimentano transizioni nei loro cicli di vita. La prima activity si mette in pausa per poi fermarsi (non si ferma finché è visibile sullo schermo) mentre la seconda viene creata, le due operazioni avvengono in contemporanea. L'ordine delle chiamate dei metodi callback è ben definito quando le due activity fanno parte dello stesso processo (quindi della stessa applicazione) e una sta richiamando l'altra. Di seguito vediamo elenchiamo i passi che vengono eseguiti quando l'activity A avvia l'activity B:

1. Il metodo *onPause()* dell'activity A viene eseguito.

2. I metodi dell'activity B *onCreate()*, *onStart()* e *onResume()* sono eseguiti in questa sequenza (B ha ora il focus).

3. Se l'activity A non è più visibile sullo schermo avviene la chiamata a *onStop()*.

Conoscendo perciò l'ordine delle chiamate possiamo per esempio gestire il passaggio di informazioni tra le activity. È possibile avviare un'activity grazie al metodo *startActivity()*, passandogli un `Intent` che specifica quale activity deve essere iniziata. L'intent può puntualizzare una determinata activity oppure descrivere l'azione che si desidera compiere con l'activity da richiamare (in quest'ultimo caso sarà il sistema che selezionerà l'activity appropriata); un intent può inoltre trasportare una ridotta quantità di dati che l'activity può necessitare al suo avvio. Di seguito mostriamo come avviare un'activity specificandone il nome della classe nell'intent:

```
1 Intent intent = new Intent(this, SignInActivity.class);
2 startActivity(intent);
```

Possiamo, come detto in precedenza, voler compiere un'azione per la quale la nostra applicazione non possiede un'activity dedicata perciò creiamo un intent contenente l'azione voluta e gli eventuali dati richiesti per portarla a termine; ecco un esempio:

```
1 Intent intent = new Intent(Intent.ACTION_SEND);
2 intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
3 startActivity(intent);
```

In questo caso EXTRA_EMAIL può essere una lista di indirizzi a cui spedire un'email, l'activity che verrà avviata tramite questo intent leggerà la lista con cui compilerà il campo destinatari.

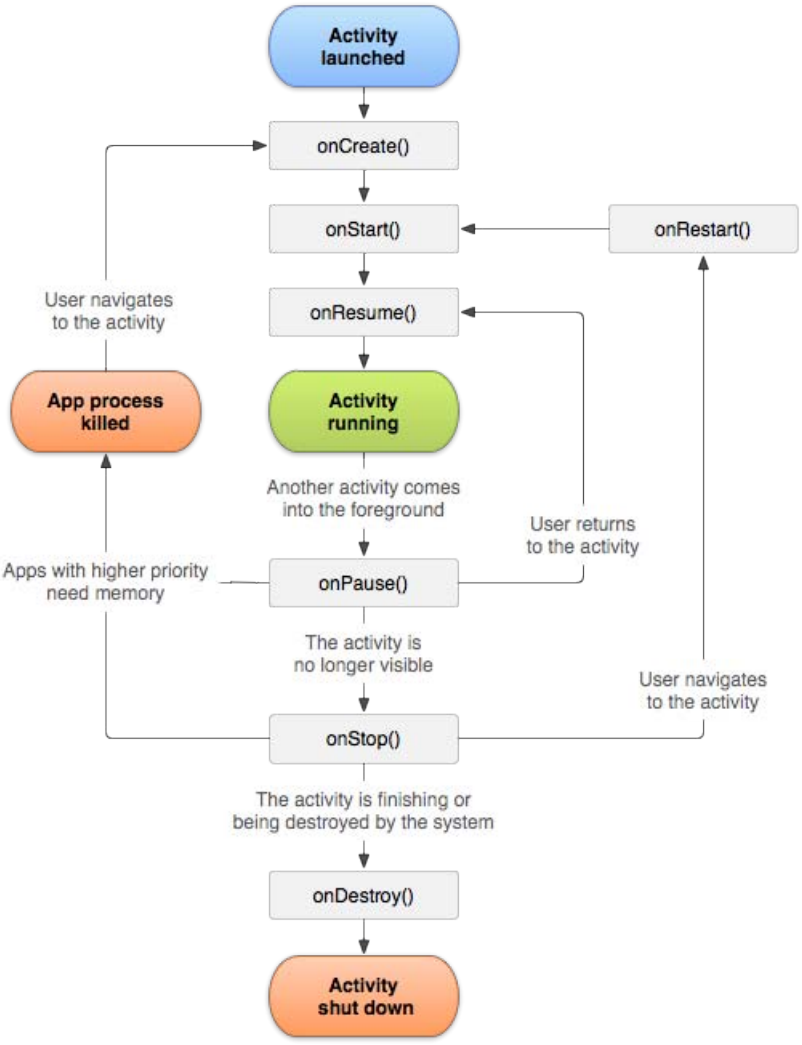


Figura 3: Ciclo di vita di un'activity

2.1.2 Native Development Kit e JNI

Il Native Development Kit è un insieme di strumenti che permettono di implementare parti dell'applicativo usando codice nativo come C o C++. Per certi tipi di applicazioni ciò è molto utile in quanto è possibile utilizzare o riutilizzare librerie sviluppate con tali linguaggi ma la maggioranza degli sviluppatori non necessita di questa funzionalità; inoltre l'uso dell'NDK comporta oltre ai benefici sopra citati anche delle ripercussioni sopratutto sulle performance destinate nel caso migliore a rimanere costanti a fronte di una maggiore complessità del codice. Buoni candidati all'uso dell'NDK sono programmi che fanno un uso intensivo della cpu, nel nostro caso specifico l'elaborazione di un flusso video comporta un notevole carico di lavoro sul processore. Il framework Android fornisce due modalità per utilizzare codice nativo:

- Scrivere l'applicazione utilizzando il framework Android e usare JNI per accedere alle API fornite dall'Android NKD. Questa modalità permette di poter scrivere codice nativo all'occorrenza e rimanere nella comodità dell'Android framework per il resto del tempo (è il metodo utilizzato in questo lavoro di tesi).

- Scrivere un'activity nativa che permetta di implementare i metodi di callback del suo ciclo di vita in codice nativo. L'Android SDK fornisce la classe `NativeClass`, una classe di supporto che notifica il codice nativo di ogni chiamata ai metodi del ciclo di vita dell'activity.

In seguito forniamo una bozza su come utilizzare gli strumenti dell'NDK:

1. Posizionare il proprio codice native in `<project>/jni/...`

2. Creare il file `<project>/jni/Android.mk` che ha lo scopo di descrivere i sorgenti nativi all'NDK build system

3. Compilare il codice nativo chiamando l'"ndk-build" script dalla cartella del proprio progetto:
   ```
   cd <project>
   <ndk>/ndk-build
   ```

   Gli strumenti di compilazione copieranno le librerie condivise necessarie all'applicazione nella suddetta cartella.

4. Infine compilare il tutto usando l'SDK come fosse un qualunque applicativo, l'SDK includerà le librerie condivise nel pacchetto `.apk`.

Java Native Interface (JNI) è un framework di programmazione che consente a codice Java in esecuzione su una Java Virtual Machine (JVM) di chiamare, ed essere chiamato da, applicazioni native (programmi specifici per un hardware e un sistema operativo) e librerie scritte in altri linguaggi come C, C++ e assembly. L'idea alla base di JNI è la volontà di fornire una strada alternativa per accedere a funzionalità che possono non essere fornite dalle librerie standard Java o nel nostro caso da Android. Il framework JNI permette a metodi nativi di utilizzare oggetti Java nello stesso modo in cui verrebbero usati da codice Java, codice nativo può inoltre creare oggetti Java, ispezionarli e usarli per compiere qualsiasi operazione. Queste enormi potenzialità portano però con se alcune implicazioni di cui è necessario tenere conto tra le quali:

- Errori difficili da individuare nell'uso di JNI possono destabilizzare l'intera JVM in modi che sono molto ardui da riprodurre e scovare.
- Ogni applicazione che usa JNI perde la portabilità.
- Il framework JNI non fornisce un garbage collector per risorse allocate al di fuori della JVM perciò il codice nativo deve prendersi la responsabilità di rilasciare le risorse utilizzate.

In JNI le funzioni native vengono implementate in file `.c` o `.cpp` separati. Quando la JVM invoca la funzione passa un puntatore `JNIEnv`, un puntatore `jobject` e tutti gli argomenti dichiarati dal metodo Java, ecco un esempio:

```
1 JNIEXPORT void JNICALL Java_ClassName_MethodName
2   (JNIEnv *env, jobject obj)
3 {
4    /*Implement Native Method Here*/
5 }
```

Il puntatore *env* è una struttura che contiene l'interfacciamento alla JVM infatti sono presenti tutte le funzioni necessarie per interagire con la JVM e per manipolare oggetti Java. Un esempio di quanto detto può essere una funzione che converta da/a array nativi partendo da/a array Java, stesso discorso per le stringhe native e le stringhe Java, oppure funzioni che instanziano oggetti, lanciano eccezioni ecc. Praticamente tutto quello che può essere fatto col codice Java deve passare per `JNIEnv`. Riportiamo una funzione che converte una stringa Java in una nativa:

```
1  //C++ code
2 extern "C"
3 JNIEXPORT void JNICALL Java_ClassName_MethodName
4   (JNIEnv *env, jobject obj, jstring javaString)
5 {
6    //Get the native string from javaString
7    const char *nativeString = env->GetStringUTFChars(javaString, 0);
```

```
8
9     //Do something with the nativeString
10
11    //DON'T FORGET THIS LINE!!!
12    env->ReleaseStringUTFChars(javaString, nativeString);
13 }
```

Come abbiamo visto i tipi di dato come stringhe e array devono essere esplicitamente convertiti tramite chiamate ad appositi metodi sfruttando `JNIEnv` mentre per i rimanenti tipi di dato nativi esiste una mappatura Java $\Longleftrightarrow$ codice nativo.

Native Type	Java Language Type	Description	Type signature
unsigned char	jboolean	unsigned 8 bits	Z
signed char	jbyte	signed 8 bits	B
unsigned short	jchar	unsigned 16 bits	C
short	jshort	signed 16 bits	S
long	jint	signed 32 bits	I
long long __int64	jlong	signed 64 bits	J
float	jfloat	32 bits	F
double	jdouble	64 bits	D
void			V

Figura 4: Mappatura tipi di dato

Lato Java è necessario dichiarare un metodo nativo utilizzando la keyword `native` in modo da notificare il compilatore che quel metodo appartiene ad una libreria esterna perciò non è necessario che ne cerchi l'implementazione nel codice Java:

```
1 public native void MethodName();
```

Si deve poi esplicitare il caricamento della libreria esterna; l'esempio seguente riassume quanto detto:

```
1 //File: ClassName.java
2 class ClassName {
3     public native void MethodName();
4
5     static {
6         System.loadLibrary("ClassName");
7     }
8
9     public static void main(String[] args) {
```

```
10          ClassName c = new ClassName();
11          c.MethodName();
12      }
13  }
```

2.2 LEGO Mindstorms NXT

Figura 5: Lego Mindstorms NXT

Il LEGO Mindstorms NXT è un kit robotico programmabile rilasciato nella seconda metà dell'anno 2006. La sua grande diffusione nelle scuole e nelle università è dovuta al prezzo ragionevolmente modesto, in confronto ad altre piattaforme per la robotica, ed alla sua enorme adattabilità perché dotato di un vasto corredo di parti meccaniche e sensori che gli permettono di assumere svariate configurazioni.

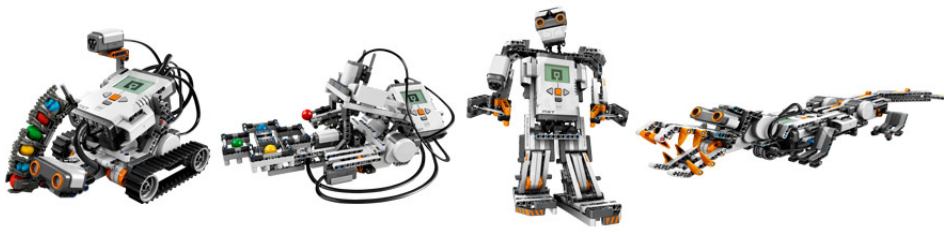


Figura 6: Esempio di configurazioni

La lista dei sensori comprende:

- Tre identici servomotori potenti a sufficienza per muovere il robot ma allo stesso tempo dotati di encoder di rotazione ottici interni che misurano la rotazione con un alto grado di accuratezza.

- Un sensore ad ultrasuoni in grado di misurare la distanza tra il sensore ed un ostacolo oltre che di rilevare movimento.

- Un sensore sonoro che tramite un microfono rileva il volume ritornando un valore in una scala da 0 (assenza di suoni) a 100 (suoni molto forti).

- Un sensore di luce che rileva l'intensità di luce in una direzione; comprende anche un LED per l'illuminazione di un oggetto.

- Un sensore touch che rileva se è premuto, urtato, o rilasciato.

Ci sono inoltre un sensore di temperatura ed altri sensori forniti da compagnie terze (giroscopio, compasso, RFID reader ecc).

Figura 7: Nxt brick

Il componente principale del kit è un'unita di elaborazione a forma di mattone chiamato "NXT brick", contiene un processore a 32-Bit-AMR7, un display LCD monocromatico di 100x64 pixel, quattro bottoni che possono essere usati per navigare l'interfaccia utente a menù gerarchici, un altoparlante ed una batteria ricaricabile Li-Ion. Può essere collegato con al massimo quattro sensori e 3 motori elettrici grazie ad una variante dei cavi RJ12. Il brick può comunicare con l'esterno tramite una porta USB oppure grazie alla tecnologia wireless Bluetooth. In questo lavoro di tesi il robot, riportato in Figura [8], è composto dalla sola unità centrale collegata a due servomotori che gli garantiscono la mobilità; è presente inoltre una struttura per la collocazione dello smartphone.

Figura 8: Il robot visto di lato

LEGO fornisce un ambiente di programmazione chiamato NXT-G per sviluppare semplici programmi per il brick con un linguaggio di programmazione visuale. La buona accoglienza che gli utenti hanno mostrato nei confronti di questo fortunato kit di robotica ha fatto sì che LEGO rendesse pubbliche le specifiche tecniche del brick NXT, questo ha generato una lunga lista di firmware, ambienti di sviluppo e librerie per svariati linguaggi di programmazione (C, asm, Java, Matlab...) alternativi a quello fornito di default. Noi useremo un firmware Java chiamato leJOS.

2.2.1 leJOS NXJ

Figura 9: Logo di leJOS

leJOS NXJ è un ambiente di programmazione per LEGO Mindstorms NXT, la principale caratteristica è che permette di programmare il robot usando codice Java. È composto da:

- Un firmware per sostituire quello presente originariamente nell'NXT che contiene una Java Virtual Machine.
- Una libreria di classi Java (`classes.jar`) che implementano la leJOS NXJ Application Programming Interface (API) e fornisce un Java Runtime (ovvero i pacchetti *java.**) ottimizzato per l'NXT.

- Un linker che effettua il collegamento tra le classi Java dell'utente e `classes.jar` per formare un file binario che può essere caricato ed eseguito sul brick.
- Un tools per sostituire il firmware, caricare i programmi, eseguirne il debug ed altre funzionalità.
- Una API per PC in modo da poter scrivere programmi su computer che comunicano con codice su leJOS NXJ usando stream Java tramite Bluetooth o USB, o utilizzando il LEGO Communications Protocol (LCP).

leJOS NXJ fornisce supporto per l'accesso alle porte I^2C del robot quindi per comunicare con i sensori ed i servomotori. Sfruttando la natura object oriented di Java, gli sviluppatori di leJOS NXT sono riusciti a nascondere i dettagli implementativi dell'accesso agli attuatori o dell'interazione coi sensori. Questo ha reso possibile agli sviluppatori di robot poter programmare in un alto livello di astrazione senza doversi preoccupare di dettagli come indirizzi esadecimali o componenti hardware. Concludiamo riportando un programma leJOS d'esempio:

```
1  import lejos.nxt.Motor;
2  import lejos.nxt.Button;
3  public class Example{
4    public static void main(String[] args){
5      Motor.A.forward();
6      Button.waitForPress();
7      Motor.A.backward();
8      Button.waitForPress();
9      System.exit(1);
10   }
11 }
```

2.3 OpenCV: Open source Computer Vision

Figura 10: Logo di OpenCV

OpenCV è una libreria open source dedicata alla computer vision il cui sviluppo iniziale risale al 1999, è scritta in C/C++ e supporta i principali sistemi operativi (Linux, Windows e Mac OS X). Implementata con l'idea di essere

computazionalmente efficiente essa mira principalmente alle applicazioni real-time perciò è scritta in linguaggio C ottimizzato e sfrutta hardware dotato di più core. Uno degli obiettivi della libreria è quello di fornire una struttura per la computer vision semplice che possa aiutare gli sviluppatori a scrivere applicativi relativamente complicati in modo veloce. OpenCV contiene moltissime funzioni che spaziano da quelle per uso medicale alla robotica, inoltre è presente una completa libreria per l'apprendimento automatico (Machine Learning Library - MLL) [2].
La sua struttura è modulare e di seguito sono elencate alcune tra le parti principali:

- **core** - contiene le strutture dati principali come l'array multidimensionale `Mat` e le funzioni base utilizzate da tutti gli altri moduli.
- **imgproc** - un modulo per processare immagini che include filtraggio lineare e non, trasformazioni geometriche dell'immagine, conversioni tra spazi di colore, istogrammi e via dicendo.
- **video** - gestisce l'analisi video e fornisce funzionalità come la sottrazione dello sfondo o algoritmi per l'object tracking.
- **highgui** - un'interfaccia semplice da usare per la ripresa di video che offre anche funzionalità per l'interazione con l'utente.
- ...

Dal 2010 è stato eseguito il porting all'ambiente Android permettendo l'utilizzo di tutto il suo potenziale anche nello sviluppo di applicazioni per il sistema operativo mobile. Successivamente è stata introdotta un'ulteriore semplificazione nell'uso di OpenCV per Android con l'introduzione dell'OpenCV Manager, quest'ultimo è un servizio che mira a gestire i binari della libreria nei dispositivi utente rendendo il tutto trasparente; permette inoltre che la libreria sia condivisa dalle applicazioni che la richiedono nello stesso apparecchio. L'OpenCV Manager porta quindi i seguenti vantaggi:

- Minore utilizzo di memoria; tutte le applicazioni usano i binari forniti dal Manager e non devono tenerli al loro interno.
- Ottimizzazioni hardware per tutte le piattaforme supportate.
- Sorgenti `.apk` sicuri poiché pubblicati da OpenCV ed ospitati nel Google Play Store.
- Correzioni di bug ed aggiornamenti automatici e con cadenza regolare.

In Figura [11] è riportato un diagramma che mostra le operazioni eseguite al lancio del primo applicativo installato che utilizza le librerie OpenCV.

1. Una qualsiasi applicazione che utilizza le librerie OpenCV viene installata da Google Play oppure manualmente.
2. Al primo lancio viene suggerita l'installazione dell'OpenCV Manager.
3. OpenCV Manager viene scaricato ed installato.
4. All'avvio dell'OpenCV Manager viene suggerita l'installazione delle librerie OpenCV specifiche per l'architettura del dispositivo in uso.
5. Quando l'installazione è completa l'applicativo può essere lanciato.

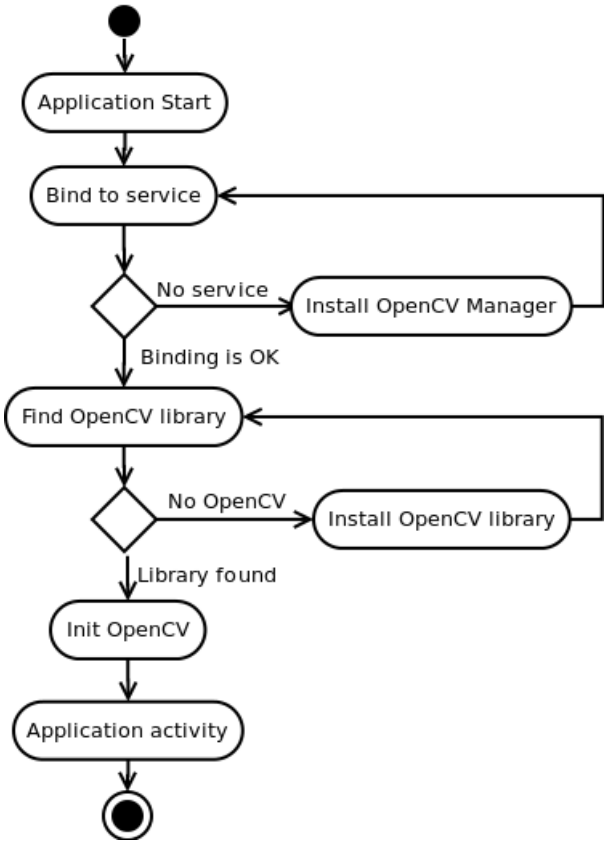


Figura 11: Modello di utilizzo per l'utente

Quando vengono lanciate applicazioni che usano OpenCV in un momento successivo avremo:

1. Una qualsiasi applicazione che utilizza le librerie OpenCV viene installata da Google Play oppure manualmente.
2. Al primo lancio l'applicazione parte normalmente.
3. Se la versione OpenCV richiesta non è installata ne viene suggerita l'installazione da Google Play.
4. Quando l'installazione è completa l'applicativo può essere lanciato.

Dal punto del vista del codice Java che deve essere scritto per poter utilizzare OpenCV Manager troviamo la classe di aiuto `OpenCVLoader` che contiene il metodo `static boolean initAsync(String Version, Context AppContext, LoaderCallbackInterface Callback)`. Quest'ultimo si preoccupa di caricare ed inizializzare le librerie utilizzando OpenCV Manager, terminato il compito avvisa il programma con una chiamata al metodo `onManagerConnected()` di cui deve essere stato fatto un override. Di seguito riportiamo uno snippet che mostra quanto detto a scopo puramente illustrativo:

```
1  public class MyActivity extends Activity implements HelperCallbackInterface
2  {
3  private BaseLoaderCallback mOpenCVCallBack = new BaseLoaderCallback(this) {
4     @Override
5     public void onManagerConnected(int status) {
6       switch (status) {
7         case LoaderCallbackInterface.SUCCESS:
8         {
9            Log.i(TAG, "OpenCV loaded successfully");
10           // Create and set View
11           mView = new puzzle15View(mAppContext);
12           setContentView(mView);
13        } break;
14        default:
15        {
16           super.onManagerConnected(status);
17        } break;
18      }
19    }
20 };
21
22 /** Call on every application resume **/
23 @Override
24 protected void onResume()
25 {
26     Log.i(TAG, "Called onResume");
27     super.onResume();
28
29     Log.i(TAG, "Trying to load OpenCV library");
30     if (!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_4, this,
       mOpenCVCallBack))
31     {
32         Log.e(TAG, "Cannot connect to OpenCV Manager");
33     }
34 }
```

3 Computer Vision

Secondo quanto affermato da Bradski e Kaehler [2] computer vision è la trasformazione di un'immagine di dati bidimensionale in una decisione o in una nuova rappresentazione. Tali trasformazioni sono fatte per raggiungere uno scopo, per esempio potremmo voler sapere se nella scena ripresa è presente una persona. Una nuova rappresentazione invece può essere la trasformazione di un'immagine a colori in una a scala di grigi. Per noi esseri umani che siamo creature visuali è facile sottovalutare la difficoltà intrinseca nella computer vision e trovare una persona in una foto può sembrare un compito al limite del risibile tuttavia il nostro cervello divide il segnale video in più canali che gli forniscono differenti informazioni; una parte del cervello si occupa di identificare nella scena le zone più importanti lasciando in secondo piano il resto mentre anni e anni di esperienza associativa permettono per esempio di predire il percorso di un oggetto. Tutto questo invece non è disponibile ad un computer che può contare solo sull'immagine proveniente da disco o da una fotocamera ovvero su una *griglia di numeri*! La Figura [12] mostra appunto quanto detto.

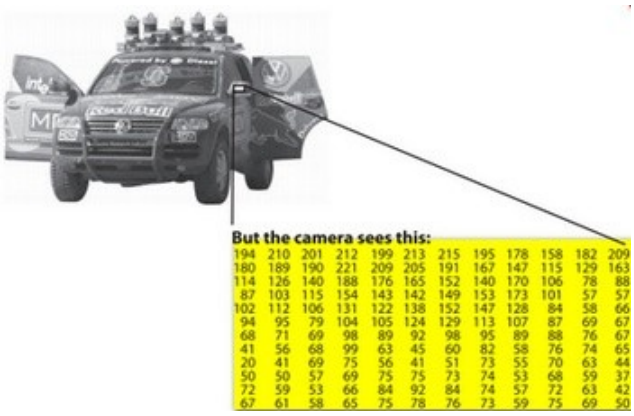


Figura 12: Immagine vista da un computer

3.1 Rappresentazione di un'immagine e modelli di colore

Un pixel può essere definito come l'elemento più piccolo di un'immagine ed è caratterizzato, oltre che dalla sua posizione, anche da valori come colore e intensità, variabili in funzione del sistema di rappresentazione adottato. In un'immagine monocroma ad un pixel è associata una luminosità dove un valore pari a 0 rappresenta il nero mentre il massimo rappresenta il bianco. In

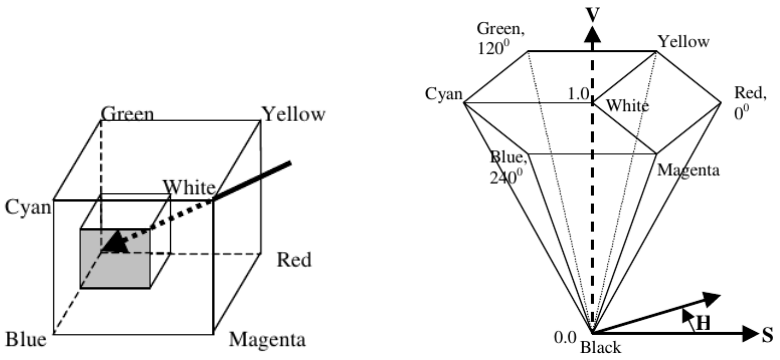
computer vision quando si parla di un'immagine a 8 bit ci si riferisce al fatto che viene utilizzato un byte per rappresentare il valore, considerando il caso precedente, della luminosità che perciò può variare da 0 a 255. Un'immagine a scala di grigi è quindi un array bidimensionale di pixel dove ogni pixel è rappresentato da un byte. Immagini binarie sono invece caratterizzate dal fatto che un pixel può assumere solo i valori 0 o 1; l'immagine sarà bianca dove non ci sono pixel e nera dove sono presenti. Le immagini a colori contengono molta più informazione della sola intensità luminosa o della presenza o meno di un pixel perché devono codificarne il colore. Ci sono diversi modelli di colore per fare ciò, noi vedremo in particolare il modello di colore standard RGB ed il modello HSV; quest'ultimo in particolare tornerà utile per gli scopi che ci siamo prefissi in questo lavoro di tesi.

(a) Scala di grigi (b) Binaria (c) RGB

Figura 13: Immagine esempio nei tre diversi modelli di colore

Lo standard per i colori è il modello Red-Green-Blue (RGB); diversamente dalle immagini a scala di grigi tale modello di colori è di tipo additivo e si basa sui tre colori rosso (*Red*), verde (*Green*) e blu (*Blue*), da cui appunto il nome RGB. Interessante è notare che tale modello tenta di descrivere che tipo di luce debba essere emessa per produrre un determinato colore infatti il campo in cui ha trovato maggior successo è la visualizzazione di immagini in dispositivi elettronici come computer o televisori (prima dell'era digitale era sostenuto da una solida teoria sulla percezione umana dei colori). Un colore nel modello RGB è espresso con una tripletta *(r,g,b)* che indica quanto di ogni singolo colore è incluso, ogni componente può variare da 0 ad un valore massimo predefinito; *(0,0,0)* rappresenta il nero e *(max,max,max)* rappresenta il bianco (dove *max* è il massimo precedentemente citato). Nell'informatica una componente è associata ad un valore binario digitale perciò variando il numero di bit che vengono dedicati per esprimerla troviamo conseguentemente il valore *max*; il numero totale di bit usati per un colore RGB prende il nome di **profondità del colore**. Nel caso RGB abbia una profondità del colore di 24 bit possono essere codificati 16 milioni di colori, un numero di

gran lunga maggiore di quanto sia discernibile dall'occhio umano, e vengono dedicati 3 byte per la codifica, uno per il rosso, uno per il verde e uno per il blu. Il grafico di questo modello di colore è rappresentato in Figura [14a].



(a) Cubo RGB (b) Piramide esagonale HSV

Figura 14: Modelli di colore

Il modello di colore Hue-Saturation-Value (HSV) si ottiene proiettando il cubo RGB lungo la sua diagonale principale [1]; se notiamo la freccia diagonale in Fig. [14a] corrisponde all'asse y verticale della piramide esagonale HSV in Fig. [14b], cambia solamente il verso. Il modello HSV è particolarmente orientato alla prospettiva umana, essendo basato sulla percezione che si ha di un colore in termini di tonalità (detta anche tinta - hue), saturazione (saturation) e luminosità (value). In modo più grossolano è possibile vedere la componente *hue* come il "tipo" di colore mentre le componenti *saturation* e *value* forniscono rispettivamente la quantità di nero e bianco con cui quel relativo colore è mischiato. Il fatto che le componenti saturation e value siano separate dalla tinta rende questo modello meno suscettibile alle variazioni di luce rispetto ad RGB, questo perché il colore percepito di un oggetto nel mondo reale varia enormemente con condizioni di luce diverse tuttativa la tinta sottostante cambia molto meno di quanto farebbero le componenti rosso, verde e blu. Per questo esatto motivo nel nostro applicativo useremo HSV come modello di colore e le immagini RGB (o meglio BGR) che gestiremo dovranno essere convertite prima di realizzare il tracking.

3.2 Object tracking

Il problema dell'object tracking è un tema centrale della computer vision che si occupa di riconoscere e tenere traccia di un obiettivo (o più obiettivi) in una scena. Questi obiettivi possono essere conosciuti a priori, per esempio

avendone a disposizione un modello, oppure possono essere riconosciuti attraverso caratteristiche pregnanti come forma, colore o trama. Tenere traccia di un oggetto non pone solamente il problema di individuarlo nella scena ma successivamente anche quello di seguirlo nelle scene successive se per esempio stiamo prendendo in considerazione un video. Ci sono differenti tecniche per realizzare l'object tracking, noi useremo Camshift e Mean Shift, entrambi sono implementati nelle librerie OpenCV. Camshift - Continuosly Adpative Mean Shift - è un algoritmo di tracking per colore sviluppato per la prima volta da Bradski [1] ed originariamente concepito come un metodo per tenere traccia del viso di una persona. Mean Shift è un robusto algoritmo che ha lo scopo di trovare il picco di una distribuzione di probabilità scalando gradienti di densità; la stima del gradiente avviene grazie all'utilizzo dei momenti di un'immagine calcolati a partire da una finestra di ricerca. La differenza principale tra Camshift e Mean Shift si trova nella finestra di ricerca infatti quest'ultimo opera su una finestra di dimensione fissa mentre Camshift ridimensiona la finestra ad ogni iterazione. Per applicare Mean Shift ad un immagine dobbiamo convertirla in una funzione di densità di probabilità, questo richiede una sorta di corrispondenza tra valore del pixel e probabilità. Per fare ciò useremo la componente *hue* del modello di colore HSV. Come già detto in precedenza HSV è poco suscettibile a variazioni luminose dettate dall'ambiente se paragonato a RGB, tuttavia pixel con valori di bassa saturazione o luminosità verranno posti a zero tanto quanto pixel con valori di luminosità o saturazione troppo elevati perché causano "rumore" nella tinta. Infine per creare la funzione di densità di probabilità costruiamo un istogramma da una selezione di pixel generatrice.

3.2.1 Back projection

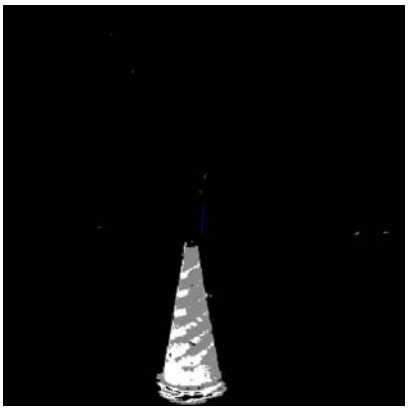
L'algoritmo Mean Shift e di conseguenza Camshift funzionano scalando i gradienti di una distribuzione di probabilità perciò per applicare questo metodo alle immagini dobbiamo convertire un'immagine in una distribuzione di probabilità. Otteniamo ciò grazie all'utilizzo di un istogramma di colori per l'oggetto obiettivo. Un istogramma può essere visto come un conteggio di dati organizzato in una collezione di contenitori (*bins*). Nel nostro caso i dati conteggiati sono i valori della tinta dei pixel (la componente *hue*) provenienti da una selezione in un'immmagine di una sequenza video. Quello che avviene è che ogni pixel viene aggiunto in uno dei contenitori (*bins*) in base al valore della sua tinta; alla fine il conteggio in ogni bins viene normalizzato o convertito in percentuale rispetto al numero di pixel del campione. Il risultato è che il valore percentile di ogni contenitori rappresenta la probabilità che

un pixel di quella tinta faccia parte dell'immagine obiettivo. In Figura [15] riportiamo un istogramma d'esempio:

Figura 15: Istogramma creato da un campione di pixel

Ora che siamo in possesso di un istogramma possiamo calcolare la probabilità di ogni pixel di un nuovo frame della sequenza video e così creare una funzione di probabilità bidimensionale che prende il nome di back projection. La back projection è salvata come un'immagine a scala di grigi con le stesse dimensioni dell'immagine da cui nasce. Detto in modo grossolano la back projection è un modo di registrare quanto bene un pixel di una nuova immagine si adatta alla distribuzione dei pixel nell'istogramma modello; al lato pratico si sceglie come selezione di pixel generatrice una porzione dell'immagine che contenga esclusivamente l'oggetto obiettivo, se ne calcola l'istogramma, si trova la back projection del frame successivo. Di seguito riportiamo un'immagine e la sua relativa back projection calcolata usando l'istogramma di Fig. [15].

(a) Cono arancione



(b) Back projection del cono

Figura 16: Un cono e la sua back projection

Cerchiamo di formalizzare le operazioni eseguite per trovare nell'immagine di Fig. [16a] aree che abbiano la maggiore probabilità di contenere il cono dal quale è stato appunto calcolato l'istogramma (Fig. [15]) usato per ottenere la back projection di Fig. [16b].

a. Per ogni pixel di Fig.[16a] (p.es. $p(i,j)$) troviamo il valore della componente *hue* e cerchiamo nell'istogramma il contenitore (*bin*) corrispondente (p.es. $(h_{i,j}, s_{i,j})$)

b. Cerchiamo nell'istogramma il corrispondente bin - $(h_{i,j}, s_{i,j})$ - e leggiamo il valore.

c. Salviamo il valore letto in una nuova immagine detta back projection (è consigliabile normalizzare l'istogramma in precedenza).

d. Terminata l'operazione per tutti i pixel otteniamo la Fig. [16b].

e. Dal punto di vista statistico i valori salvati nella back projection rappresentano la probabilità che un pixel nell'immagine di partenza appartenga al cono sul quale è stato calcolato l'istogramma usato. Dal punto di vista visivo le aree più chiare sono quelle che hanno la maggiore probabilità.

3.2.2 Momenti di un'immagine

I momenti di un'immagine vengono usati dall'algoritmo Mean Shift, per stimare il gradiente di densità, oltre che da Camshift per ridimensionare la finestra di ricerca. Queste funzioni trovano impiego in statistica, meccanica e nell'analisi di un'immagine. In statistica i momenti sono calcolati su una funzione di densità di probabilità e ne rappresentano alcune proprietà; il momento di ordine zero ci dice la probabilità totale, quello di ordine uno il valore atteso e quello di secondo ordine la varianza. Nell'analisi di immagini possiamo considerare un'immagine una distribuzione bidimensionale di intensità perciò possiamo calcolare i momenti di questa distribuzione che ci danno proprietà come l'area totale, le coordinate del baricentro e l'orientazione. Per una funzione continua bi-dimensionale $f(x,y)$ il momento di ordine $(p+q)$ è definito come:

$$\Phi_{pq} = \int\int \Psi_{pq}(x,y) f(x,y) dxdy \tag{3.1}$$

dove Φ_{pq} è il momento di ordine (p+q), Ψ_{pq} è la funzione nucleo e f è la densità di distribuzione. Ci sono diverse funzioni nucleo che producono momenti per svariati scopi tuttavia noi siamo interessati ai momenti geometrici

la cui funzione nucleo è $\Psi_{pq}(x, y) = x^p y^q$ e sono indicati con M_{pq} al posto di Φ_{pq}. D'ora in poi verranno usati in modo interscambiabile i termini momenti e momenti geometrici inoltre lavorando con distribuzioni di probabilità discrete quali sono le back projection useremo sommatorie per rappresentare il processo di integrazione. La funzione generatrice dell'Eq. 3.1 diventa perciò:

$$M_{pq} = \sum_x \sum_y x^p y^q f(x, y) \tag{3.2}$$

Adattando questa definizione ad un'immagine digitale i cui pixel sono caratterizzati da un'intensità (nel caso di immagini binarie o a scala di grigi) $I(x, y)$ si ottiene:

$$M_{pq} = \sum_x \sum_y x^p y^q I(x, y) \tag{3.3}$$

In alcuni casi è conveniente normalizzare l'intensità in analogia ad una funzione di densità di probabilità, questo si ottiene dividendo $M_{p,q}$ per la doppia sommatoria $\sum_x \sum_y I(x, y)$. Semplici proprietà delle immagini derivate attraverso i momenti includono:

- l'**area** (per immagini binarie) o somma dei livelli di grigio (per immagini a scala di grigio): M_{00}
- il centroide (o "baricentro") dell'immagine, dove l'intensità dei pixel può essere paragonata alla massa in fisica:

$$(x_c, y_c) = (\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}) \tag{3.4}$$

  Una proprietà del centroide di un'immagine (o di una sua regione) che viene usata dall'algoritmo Mean Shift consiste nel fatto che il vettore che parte dal centro della regione ed arriva al centroide della regione ha la stessa direzione del vettore gradiente. Il vettore gradiente per una funzione di densità punta verso la direzione con la maggiore pendenza perciò il vettore gradiente per una back projection punterà verso una zona di maggiore probabilità. Mean Shift sfrutta questa proprietà e scala il gradiente per trovare l'oggetto nella scena.

3.2.3 Mean Shift e Camshift

Come è già stato accennato l'algoritmo che useremo per l'object tracking è Camshift che si basa a sua volta sull'algoritmo Mean Shift del quale abbiamo anticipato il principio fondante, in questa sezione li vedremo più da vicino.

Mean Shift

Mean Shift agisce come uno stimatore del gradiente di una funzione di densità; il gradiente è il vettore tangente alla funzione di densità diretto verso la maggiore pendenza. Il processo iterativo sviluppato da Fukunaga e Hoestler [5] viene utilizzato per cercare massimi locali nella funzione di densità; questo processo, detto anche algoritmo Mean Shift, vede lo spostamento in modo iterativo di una finestra di ricerca di dimensioni fisse centrandola ripetutamente sul centroide del set di punti contenuti nella finestra stessa [4]. In questo modo Mean Shift può scalare fino a trovare una norma locale, in parole semplici trova il valore che compare più frequentemente in una distribuzione di frequenza; perciò usando Mean Shift sulla back projection troviamo una norma locale, o un picco, nella funzione di densità di probabilità che tradotto equivale a trovare il punto che localmente ha la probabilità più alta di essere l'oggetto cercato. Tale algoritmo può tenere traccia di oggetti che si muovono con due gradi di libertà ovvero traslazioni lungo gli assi x ed y [2]. La procedura iterativa tipica di Mean Shift è riportata in Fig. [17].

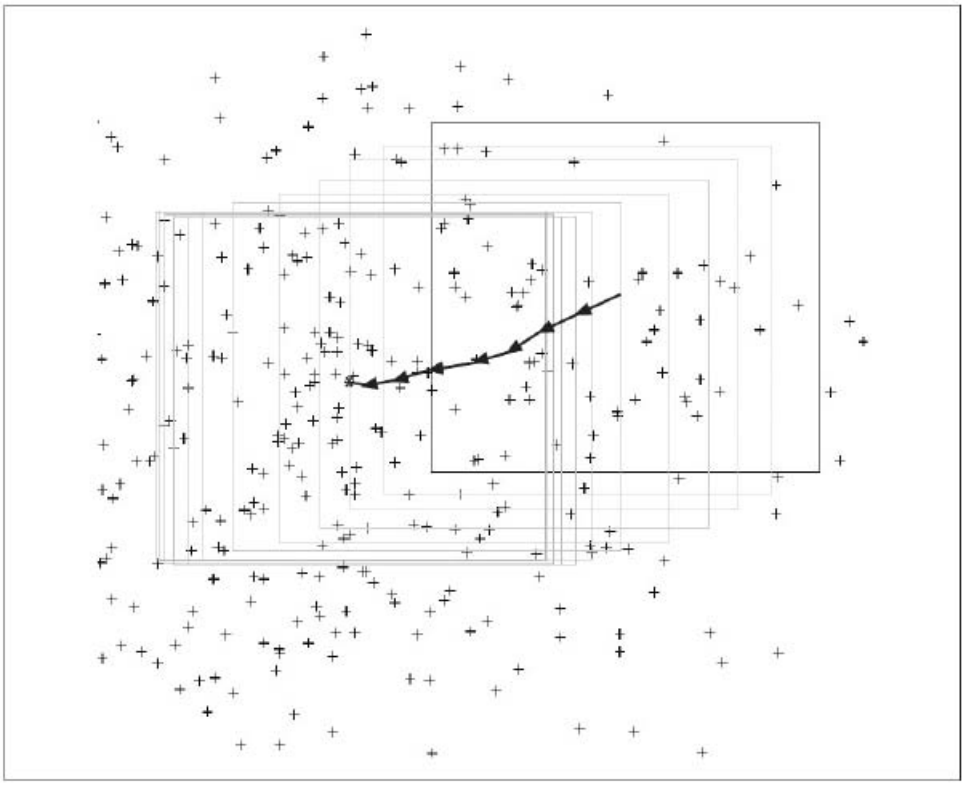


Figura 17: Spostamento della finestra di ricerca applicando l'algoritmo Mean Shift ad un set di dati.

Dicendo che Mean Shift stima un grandiente di densità ci si riferisce al fatto che stima il gradiente, o la direzione di massima pendenza, di una funzione di densità di probabilità, nel nostro caso la back projection. Per ottenere quindi il gradiente è necessario calcolare il centroide della finestra di ricerca che avrà coordinate:

$$(x_c, y_c) = (\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}) \tag{3.5}$$

Sottraendo le coordinate del centro della corrente finestra di ricerca da quelle del centroide si ottiene un vettore Mean Shift; quest'ultimo ha la stessa direzione e verso del gradiente della funzione di densità al centro della finestra di ricerca. Questa procedura viene ripetuta iterativamente e vede il calcolo del vettore Mean Shift seguito dalla traslazione della finestra lungo tale vettore fino alla convergenza. Ciò significa che la finestra si fermerà sulla norma locale ovvero sul massimo; figurativamente questo approccio scala il gradiente fino alla cima quando non può fare altro che scendere.

Algorithm 1: MeanShift

input : posizione e grandezza della finestra di ricerca, soglia
nuovafinestra←finestra di ricerca;
repeat
  vecchiafinestra←nuovafinestra;
  Calcola M_{00}, M_{10}, M_{01};
  $x_c \leftarrow \frac{M_{10}}{M_{00}}$;
  $y_c \leftarrow \frac{M_{01}}{M_{00}}$;
  $dx \leftarrow x_c$−nuovafinestra.centro.x;
  $dy \leftarrow y_c$−nuovafinestra.centro.y;
  nuovafinestra.centro.x ←nuovafinestra.$x + dx$;
  nuovafinestra.centro.y ←nuovafinestra.$y + dy$;
until *differenza tra vecchiafinestra e nuovafinestra* $\leq$ *soglia*;

Se per esempio una fotocamera sta catturando un pallone in movimento è evidente che durante la ripresa la sua posizione cambierà da un frame al successivo; se il pallone tuttavia durante il passaggio da un frame all'altro è ancora parzialmente contenuto nella finestra di ricerca la procedura scalerà il gradiente per ricentrare la finestra di ricerca sull'obiettivo. In questo modo è possibile tenere traccia dell'oggetto da un frame al successivo. Vale la pena osservare che una velocità eccessiva del pallone o una velocità di cattura dei frame troppo bassa possono risultare in un pallone posizionato da un frame all'altro fuori dalla finestra di ricerca, in questo caso l'algoritmo perde traccia dell'oggetto.

Camshift

Camshift, versione contratta di Continuosly Adaptive Mean Shift, fu inizialmente sviluppato per essere impiegato in interfacce percettive mirando a fornire il tracking del viso dell'utente. Il nome dell'algoritmo deriva dal fatto che continua a ridimensionare in modo autonomo la finestra di ricerca infatti Mean Shift non era mai nato come algoritmo di tracking perciò non prende in considerazione che lo spostamento dell'oggetto possa avvenire anche lungo l'asse z rispetto al piano dell'immagine. Camshift, partendo dal risultato raggiunto da Mean Shift, è stato studiato per gestire dinamicamente i cambiamenti tra immagini succcessive perché ridimensiona con continuità la finestra di ricerca. Il risultato è un algoritmo che tiene traccia dell'oggetto che si sposta lungo i tre assi rispetto al piano dell'immagine fornendo inoltre la sua area e l'angolo di rotazione rispetto all'asse z. Elenchiamo di seguito l'algoritmo in forma di pseudo codice per poi andare a commentarlo:

Algorithm 2: Camshift

input : posizione e grandezza della finestra di ricerca, soglia
nuovafinestra← finestra di ricerca;
repeat
  vecchiafinestra← nuovafinestra;
  $M_{00} \leftarrow$ MeanShift(nuovafinestra);
  $s \leftarrow 2\sqrt{\frac{M_{00}}{256}}$;
  nuovafinestra.larghezza← s;
  nuovafinestra.altezza← $1.2s$;
until *differenza tra vecchiafinestra e nuovafinestra* $\leq$ *soglia*;

Come possiamo vedere la novità sta nel fatto che dopo la chiamata a Mean Shift, Camshift ridimensiona la finestra utilizzando l'area ovvero il momento di ordine zero; l'area dipende dalla coordinata dell'asse z ed aumenta col diminuire della distanza dalla fotocamera. Notiamo inoltre che l'algoritmo converte M_{00} in unità di pixel piuttosto che intensità, questo deriva dal fatto che M_{00} viene calcolato usando il valore dell'intensità dei pixel nell'immmagine a scala di grigi perciò per effettuare la conversione che vogliamo dobbiamo dividere per 256. Siccome M_{00} corrisponde all'area, Camshift può approssimare la lunghezza del lato con una semplice operazione di radice; poiché l'algoritmo è stato ideato per connettere le aree della distribuzione attigue si raddoppia la lunghezza ottenuta:

$$s \leftarrow 2\sqrt{\frac{M_{00}}{256}} \tag{3.6}$$

Infine dato che Camshift nasce con l'obiettivo di inseguire il viso di una persona, la finestra di ricerca viene ridimensionata con una forma molto affine

a quella umana che è approssimativamente ellittica; questo viene ottenuto aggiustando la larghezza della finestra a s e l'altezza a $1.2s$. Terminiamo con alcune considerazioni riguardanti i vantaggi di utilizzare Camshift come algoritmo di tracking:

- **Camshift è robusto rispetto al moto irregolare dovuto alla distanza dalla telecamera**: molti algoritmi hanno problemi legati al fatto che un oggetto vicino al punto dal quale è inquadrato si muove molto più velocemente rispetto ad un oggetto lontano tuttavia Camshift grazie al ridimensionamento dinamico della finestra si comporta molto bene in entrambe le situazioni; quando lontano la sua finestra ridotta ed il suo moto molto lento aiutano l'algoritmo a rintracciare l'oggetto che quando invece è vicino ha una finestra molto grande che riesce a seguire gli spostamenti veloci.

- **Camshift tende ad ignorare oggetti estranei**: dato che l'algoritmo usa una finestra di ricerca che scala i gradienti delle distribuzioni locali tenderà ad ignorare tutto ciò che è estraneo alla distribuzione perciò finché lo spostamento dell'oggetto tra due fotogrammi consecutivi non è sufficientemente grande Camshift sarà ancorato all'oggetto obiettivo nonostante l'ambiente di contorno.

- **Camshift è robusto rispetto a parziale occlusione**: siccome la finestra viene centrata sulla norma della distribuzione anche se una parte dell'oggetto viene nascosta la finestra rimarrà concentrata sulla zona ancora esposta alla fotocamera.

- **Camshift è veloce**: dato che l'algoritmo esegue calcoli dispendiosi ma in misura ridotta ai pixel che compongono la finestra di ricerca, elimina la necessità di elaborare l'immagine nella sua interezza perciò può raggiungere prestazioni sufficienti per essere impiegato in applicativi real-time.

3.3 Calcolo degli spostamenti

L'obiettivo della Visual servoing consiste nell'usare informazioni acquisite tramite sensori di visione (nel nostro caso la fotocamera di uno smartphone) per controllare la posa/moto di un robot (o di una parte di esso). In generale le tecniche di Visual Servoing sono classificate secondo le seguenti tipologie:

- **Position Based** (PBVS), basato sulla posizione: le informazioni estratte dalle immagini (features) vengono usate per ricostruire la posa 3D corrente dell'oggetto (posizione/orientamento); combinandola con

la conoscenza della posa 3D desiderata si genera un segnale di errore "cartesiano" che guida il robot.

- **Image Based** (IBVS), basato sull'immagine: la legge di controllo è basata sull'errore calcolato direttamente sui valori numerici delle features estratte, senza passare per la ricostruzione 3D, questa tecnica non coinvolge perciò in nessun modo la stima della posizione dell'oggetto nello spazio o in un qualche sistema di riferimento che non sia l'immagine stessa; il robot si muove in modo da portare le features correnti (quello che "vede" nella telecamera) verso i loro valori desiderati.

Noi utilizzeremo l'ultimo approccio citato raffigurato in Figura [18]:

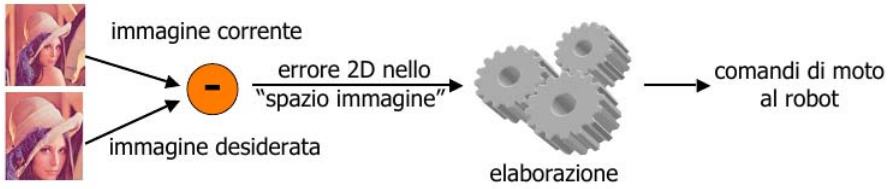


Figura 18: Image-based visual servoing

È importante notare le differenze tra le tecniche precedenti. Mentre quest'ultima utilizzava solo l'errore sul piano immagine e non veniva effettuato nessun tipo di calcolo di stima della posizione, nella prima elencata si ha una ricostruzione della posizione dell'oggetto nello spazio 3D (è una tecnica basata su un modello). Le caratteristiche (o features) in esame possono essere linee, punti, momenti geometrici, ecc; nel nostro caso utilizzeremo il centro (C) del BoundingRect che conterrà l'oggetto durante il tracking da un frame all'altro, il valore desiderato in cui portare la feature sarà invece il centro della scena. Per fare questo si è diviso il frame ricevuto in 9 aree, come riportato in Figura [19], delle quali solo a quattro è stato dato un significato in termini operativi:

0. la zona centrale è il punto in cui desideriamo portare la nostra feature, non vengono impartiti altri ordini al robot una volta che il centro dell'oggetto (C) si trova in questa area.

1. l'oggetto si sta allontanando e spostando verso sx rispetto alla fotocamera

2. l'oggetto si sta allontanando

3. l'oggetto si sta allontanando e spostando verso dx rispetto alla fotocamera

Implementando un metodo che riconosce in quale area si trova momentaneamente C è possibile perciò istruire il robot sugli spostamenti che deve compiere. Questa tecnica, sebbene molto basilare offre l'interessante proprietà di non dipendere da una conoscenza del modello dell'oggetto (IBVS) e di essere quasi insensibile ai parametri intrinseci/calibrazione della telecamera.

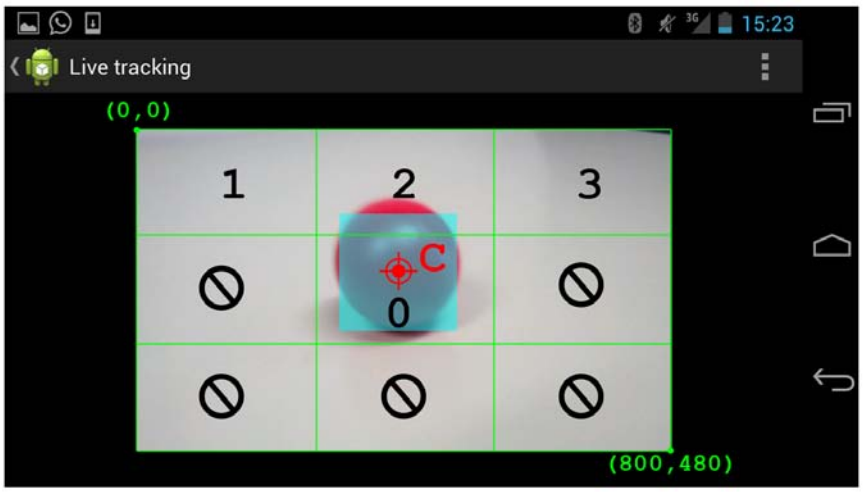


Figura 19: Grafica che illustra la legge di controllo utilizzata

4 Implementazione

L'implementazione dell'applicativo è stata realizzata in Java ed è composta da quattro classi ed una libreria nativa C++ dal lato Android, più una classe per il brick NXT. L'applicazione eseguita sullo smartphone è fondamentalmente multithreading e questa scelta implementativa è pressoché obbligata dalla necessità di mantenerla reattiva ed evitare l'infausta finestra di notifica ANR (Application Not Responsive). In generale il sistema mostra questa finestra quando l'applicazione non risponde all'input dell'utente ovvero quando il codice in esecuzione sul thread UI (o thread principale) è per esempio in attesa che termini un'operazione di rete o se sta svolgendo calcoli estremamente complessi che impegnano il processore per un tempo troppo lungo. Il thread UI è il principale thread di esecuzione di un'applicazione ed è dove normalmente la totalità del codice viene eseguito. Tutti i componenti di un'applicativo (Activity, Service, ContentProvider, BroadcastReceiver) vengono creati in questo thread inoltre, per esempio, in un'applicazione costituita da una singola activity tutti i metodi del suo ciclo di vita vengono eseguiti nel thread. In Android qualsiasi riga di codice che vada a modificare la UI **deve** accadere nel thread UI. Quanto detto suggerisce che in qualsiasi situazione un'applicazione debba eseguire una potenziale lunga operazione questa non dovrebbe avvenire nel thread UI, dovrebbe tuttavia essere creato un apposito thread operaio (worker thread) che svolga tale lavoro. Così facendo si è sicuri che il thread UI (che gestisce gli eventi dell'interfaccia grafica) continui ad essere eseguito prevenendo il sistema dal concludere che il nostro codice si è bloccato (ANR). La creazione di un worker thread implica che tutto il codice eseguito su quel thread non sarà a carico del thread UI, tuttavia abbiamo sottolineato in precedenza che qualsiasi istruzione che miri a modificare la UI deve essere eseguita nel thread principale; come possiamo agire nel caso ci fosse la necessità di avvisare l'utente modificando la UI al termine di un'operazione avvenuta in un worker thread? La risposta a questa domanda non è univoca poiché il framework Android prevede più soluzioni in merito tuttavia quella da noi adottata e certamente la più generale consiste nell'utilizzo di un `Handler`. Un handler permette di inviare e processare messaggi ed oggetti Runnable (contenitori di codice che può essere eseguito) associati con la coda di attesa dedicata ai messaggi di un thread. Ogni istanza della classe Handler è associata ad un singolo thread e alla sua coda di messaggi, questa associazione avviene al momento della creazione dell'handler nel thread, da quel momento in poi l'Handler invierà ogni messaggio o runnable a quella coda di messaggi. Quando un processo viene creato per l'applicativo che andiamo a lanciare, il suo main thread è associato ad una coda messaggi dedicata alla gestione dei componenti principali che lo

compongono (activity, broadcast receiver, etc) oltre che ad ogni finestra che verrà creata. E' possibile perciò creare dei thread secondari e comunicare col thread principale grazie ad un handler; ciò avviene grazie alle chiamate ai metodi `post()` e `sendMessage()` effettuate dal thread secondario. Il risultato è che l'eventuale messaggio o runnable verrà messo nella coda dei messaggi del thread UI e schedulato per l'esecuzione. Il problema immediatamente successivo consiste nel come far pervenire l'handler di un thread agli eventuali utilizzatori poiché non è permesso passarlo tramite un intent [1], la soluzione consiste nell'utilizzo della classe `Messenger`. Messenger è una classe che agisce da riferimento per un handler e che può essere perciò usata per inviargli messaggi; questo permette di instaurare una comunicazione fra processi/activity basata sullo scambio di messaggi semplicemente creando un messenger che punta ad un determinato handler e poi passarlo all'activity che lo necessita.

```
1 Intent intent = new Intent(this, CameraActivity.class);
2 intent.putExtra("connThreadHandler", new Messenger(connManagerThread.getHandler()
      ));
3 startActivity(intent);
```

L'esempio sopra elencato è tratto dal nostro applicativo e mostra come sia possibile lanciare l'activity contenuta in `CameraActivity.class` passandole, grazie a Messenger, l'handler del thread che gestisce la connessione Bluetooth col brick NXT.

4.1 Struttura generale dell'applicativo

Elenchiamo di seguito le classi che fanno parte della struttura dell'applicativo accompagnandole con una breve spiegazione del compito che intendono svolgere:

- `MainActivity`: è la classe che contiene l'activity che per prima viene caricata una volta lanciata l'applicazione, il suo compito è di inizializzare le librerie OpenCV, caricare la libreria nativa, avviare il thread che gestisce la connessione al brick NXT ed aspettare che questa sia stata stabilita; una volta che tutti i passaggi sono stati completati viene abilitato il tasto per avviare l'activity di tracking.

- `CameraActivity`: è la classe che contiene l'activity in cui avviene il tracking perciò è composta principalmente dalla `JavaCameraView`, tale View è lo strumento fornito dal porting OpenCV4Android per

[1] Handler non implementa nè l'interfaccia Serializable nè Parcelable quindi non può essere aggiunto all'intent come un Extra

mostrare sulla schermata dello smartphone la sequenza video ripresa dalla fotocamera; in aggiunta è presente anche una view che si è appositamente sviluppato per permettere la selezione dell'oggetto di cui eseguire il tracking.

- `NXTConnectionManager`: è la classe che si occupa di stabilire una connessione Bluetooth con il brick NXT. Utilizza le classi `Handler` e `Looper` per realizzare un ciclo infinito di attesa non impegnata nel quale è possibile comunicare messaggi al brick; ospita inoltre una classe interna che è un thread in ascolto di eventuali messaggi diretti all'applicativo.

- `RectSelection`: è la classe che implementa una View personalizzata il cui scopo è visualizzare una selezione quadrata ridimensionabile a schermo con la quale andare ad indicare l'oggetto di cui fare il tracking.

- `native_opencv`: è il file sorgente cpp che contiene la libreria di chiamate native; esegue quasi la totalità delle operazioni sui frame di ingresso e prepara quanto necessario per poi poter richiamare l'algoritmo di tracking vero e proprio.

- `NXTNavigator`: è la classe che funge da navigatore per quel che riguarda il brick NXT; costruita attorno ad un loop principale, il suo obiettivo è quello di ricevere le coordinate del punto dove portare il robot e fare le operazioni necessarie perché ciò avvenga.

L'**entry point** è la MainActivity che carica entrambe le librerie utilizzate: OpenCV e nativa. Successivamente istanzia un oggetto appartenente alla classe `NXTConnectionManager` e così facendo viene creato un thread in background che esegue tutti i passi necessari ad instaurare una connessione Bluetooth col brick NXT, si riferirà in seguito a tale thread anche come "thread di connettività". Al termine delle operazioni il thread dedicato alla connettività **avvisa** MainActivity comunicando l'esito finale in base al quale viene o meno abilitato il pulsante che lancia CameraActivity. Quest'ultima è l'activity che mostra la sequenza video ripresa dalla fotocamera e permette dapprima di selezionare l'oggetto obiettivo grazie a `RectSelection` e successivamente di eseguirne il tracking. Nel metodo di tracking, al ritorno della chiamata a Camshift, disponiamo dell'area oltre che delle coordinate del centro del rettangolo che contiene l'oggetto, questo permette di calcolare di quanto far avanzare il robottino ed in che direzione; non resta che inviare le istruzioni tramite il thread di connettività e rimanere in attesa che l'applicativo sul robot, contenuto nella classe `NXTNavigator`, comunichi allo

smartphone che lo spostamento è stato eseguito. Sottolineiamo che finché le operazioni sul robot non sono concluse lo smartphone e quindi il processo di tracking resta attivo ma non vengono inviate ulteriori coordinate, il cellulare diventa *cieco* nonostante i software che sono in esecuzione permettano l'accodamento di ulteriori spostamenti; questa scelta deriva dal fatto che durante il moto del robot le coordinate percepite dovrebbero tenere in considerazione in che stadio di avanzamento si trova, possiamo scegliere di evitare questa ulteriore complicazione del problema obbligando all'attesa lo smartphone.

4.2 Connettività Android-NXT:Bluetooth

La connettività tra lo smartphone Android ed il brick NXT è realizzata nella classe `NXTConnectionManager` che estende la classe Thread, a tutti gli effetti essa svolge il compito di worker thread di cui abbiamo già discusso. In questa classe troviamo utilizzato un componente derivante da leJOS: `NXTComm`. NXTComm è un'interfaccia che tutte le classi di leJOS che vogliono comunicare a basso livello col brick NXT devono implementare; il suo scopo è quello di instaurare una comunicazione col brick del robot, le seguenti linee di codice mostrano come:

```
1 nxtComm = NXTCommFactory.createNXTComm(NXTCommFactory.BLUETOOTH);
2 nxtComm.open(nxtInfo,NXTComm.PACKET)
```

`NXTCommFactory` crea una versione di NXTComm adatta al sistema operativo (Android) [2] in uso ed al protocollo che è stato richiesto (Bluetooth o USB). A questo punto si può invocare il metodo `open()` dove `nxtInfo` è un contenitore di tutte le informazioni (p.es. l'indirizzo MAC) riguardanti uno specifico brick NXT e le connessioni che può instaurare. A questo punto se le operazioni vanno a buon fine è possibile ottenere i canali di comunicazione aperti:

```
1 DataOutputStream dataOut = new DataOutputStream(nxtComm.getOutputStream());
2 DataInputStream dataIn = new DataInputStream(nxtComm.getInputStream());
```

Il metodo `run` contiene il loop principale nel quale l'handler della classe riceve le coordinate e le invia tramite il metodo `sendData()` al brick. La peculiarità della classe `NXTConnectionManager` risiede nella presenza di una classe interna il cui nome esplicita già la funzione per la quale è stata definita: `innerListener`. Lo scopo di quest'ultima è chiaramente quello di rimanere in ascolto sul canale di ingresso della connessione Bluetooth in attesa del via libera da parte del brick NXT ogni volta che è giunto a destinazione dopo aver ricevuto l'ordine di avanzare. Notiamo infine come

[2] È necessario includere la classe NXTCommAndroid nei sorgenti dell'applicativo perché per problemi di dipendenze non è contenuta in pccomm.jar

è stato possibile realizzare nel metodo `run` della classe principale un loop infito di attesa non impegnata grazie all'uso di `Looper` in congiunzione con `Handler`. La classe Looper è infatti utilizzata per implementare una coda di messaggi; un thread di norma non ha una coda di messaggi a lui associata quindi si ovvia a tale mancanza richiamando i metodi `prepare()` e `loop()`, solitamente si usa in congiunzione con un handler e di seguito mostriamo uno script che riunisce quanto detto:

```
1  class LooperThread extends Thread {
2      public Handler mHandler;
3
4      public void run() {
5          Looper.prepare();
6
7          mHandler = new Handler() {
8              public void handleMessage(Message msg) {
9                  // process incoming messages here
10             }
11         };
12
13         Looper.loop();
14     }
15 }
```

4.3 Selezione ROI

Uno dei passi fondamentali per eseguire il tracking dell'oggetto consiste nel permettere all'utente di specificarlo tramite l'interfaccia dell'applicativo, putroppo il framework Android non fornisce nessun componente standard che riesca a fare questo. Nonostante la grande collezione di `View` a cui un programmatore che voglia sviluppare in Android può attingere a piene mani, esistono situazioni in cui diviene necessario creare una view apposita; la classe `RectSelection` nasce da questa urgenza. Per definire una view personalizzata è sufficiente creare una nuova classe che estenda `View`[3], inoltre la minima richiesta che viene fatta è l'implementazione di un costruttore che prenda come parametri un `Context` e un `AttributeSet`.

```
1 class RectSelection extends View {
2     public RectSelection(Context context, AttributeSet attrs) {
3         super(context, attrs);
4     }
5 }
```

Sorvolando velocemente su Context, un'interfaccia per accedere ad informazioni generali riguardanti l'ambiente dell'applicativo, ci soffermiamo su

[3] È possibile definire attributi custom per la proprio view usando il costrutto `<declare-styleable>`; è di norma fare ciò creando un apposito file `res/values/attrs.xml`

AttributeSet; quando una View viene creata da un layout XML tutti gli attributi che la descrivono presenti nel file vengono letti e passati al costruttore sopra citato sotto forma di un AttributeSet. La chiamata `super(context, attrs)` permette perciò al layout editor di creare e modificare un'istanza della view. Il passo più importante durante la realizzazione di una view personalizzata è l'override del metodo `onDraw()` il cui parametro è un `Canvas` ovvero un oggetto che la view può utilizzare per disegnarsi. Il framework Android che gestisce la grafica divide la fase di disegno in due principali momenti:

- **Che cosa disegnare** gestito dalla classe `Canvas`
- **Come disegnarlo** gestito dalla classe `Paint`

In parole povere `Canvas` definisce le forme che possono essere disegnate a schermo mentre `Paint` definisce il colore, lo stile, il font e così via di ogni forma disegnata. Prima di poter disegnare dobbiamo perciò creare un oggetto Paint e questo viene fatto nel costruttore perché creare oggetti dentro il metodo `onDraw()` riduce significativamente le prestazioni. A questo punto possiamo disegnare il rettangolo di selezione sfruttando la classe `Rect` il cui costruttore è `public Rect (int left, int top, int right, int bottom)`:

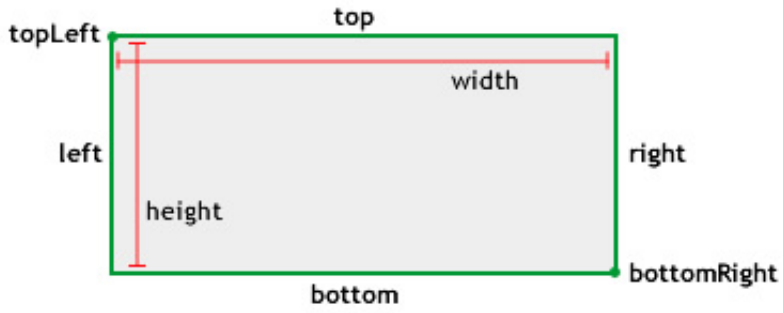


Figura 20: Schema rettangolo di selezione

Quello che siamo riusciti ad ottenere fino a questo punto è un rettangolo statico disegnato a schermo, manca la componente dinamica ovvero la capacità di essere spostato sopra all'oggetto obiettivo e di essere ridimensionato in modo da poterlo contenere nel modo più stringente possibile. Come molti framework per la realizzazione di UI anche Android supporta un modello di eventi in ingresso. Le azioni dell'utente vengono trasformate in eventi che a loro volta innescano chiamate a funzioni di callback di cui si

può eseguire l'override per personalizzare il comportamento del proprio applicativo. L'evento di ingresso più comune in Android è il tocco che innesca il metodo `onTouchEvent(android.view.MotionEvent)`, l'override di tale metodo è perciò la strada maestra per gestire l'interazione con l'utente; in prima approssimazione è possibile scrivere del codice apposito che determini se il tocco è parte di un "gesto più complesso" come per esempio può essere un *pinch to zoom* tuttavia è disponibile una classe, `ScaleGestureDetector`, che riconosce questo genere di input ed aiuta a gestirlo. Nel nostro applicativo il trascinamento è stato realizzato intercettando gli eventi direttamente nell'override del metodo `onTouchEvent(MotionEvent ev)` mentre per permettere di scalare il rettangolo si è definita una classe interna che estende `ScaleGestureDetector.SimpleOnScaleGestureListener`, una classe di comodità concepita per essere usata solo quando si necessità di captare una minima parte tra tutti gli eventi legati alla riscalatura.

4.4 Librerie native

Abbiamo già visto come sia possibile scrivere un'applicativo utilizzando il framework Android in combinazione con JNI per accedere ad eventuali librerie native. Nel nostro caso il file sorgente `native_opencv.cpp` ospita il codice C++ che andremo ad esaminare ma prima di inoltrarci in questa direzione motiviamo la scelta di utilizzare codice nativo. L'adozione di JNI è infatti molto circostanziale perché nel migliore degli scenari si riescono a pareggiare le prestazioni ottenute usando codice ottimizzato per Dalvik che svolge la funzione richiesta, tuttavia per questo lavoro di tesi risulta preferibile per due motivi principalmente:

- l'elaborazione del flusso video che dobbiamo svolgere richiede un **intenso utilizzo della cpu**,
- richiamare funzioni OpenCV tramite le API Java **comporta in ogni caso una chiamata JNI**.

OpenCV4Android è il nome ufficiale del porting di OpenCV sulla piattaforma Android, sono così rese disponibili tramite una API una buona parte (ma non tutte) delle funzioni disponibili in OpenCV. Con la Java API ogni funzione disponibile è racchiusa in un'intefaccia Java, questo significa che la funzione vera e propria resta scritta in C++ e viene successivamente compilata. Ecco quindi che per ogni chiamata è ragionevole associare una penalità dovuta all'overhead insito nell'utilizzo di JNI, tale penalità deve essere contata due volte: una all'atto della chiamata iniziale ed una al momento del ritorno. La somma delle penalità che si accumulano ad ogni utilizzo di una

funzione OpenCV durante l'elaborazione di un frame rende le prestazioni progressivamente peggiori. Di seguito viene mostrato in modo grafico quanto detto:

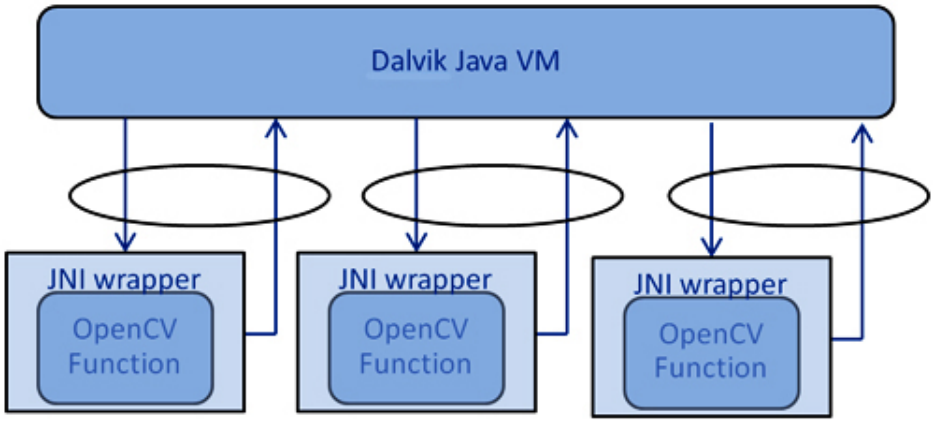


Figura 21: Tre chiamate a funzioni OpenCV usando le Java API

Una soluzione leggermente più complessa che però fornisce prestazioni migliori consiste nell'utilizzare il Native Development Kit; con questo approccio il codice che elabora il frame è interamente scritto in C++ perciò le chiamate alla libreria OpenCV avvengono tutte da codice nativo. Non resta che incapsulare il tutto in un'unica funzione che verrà chiamata da codice Java una volta per frame.

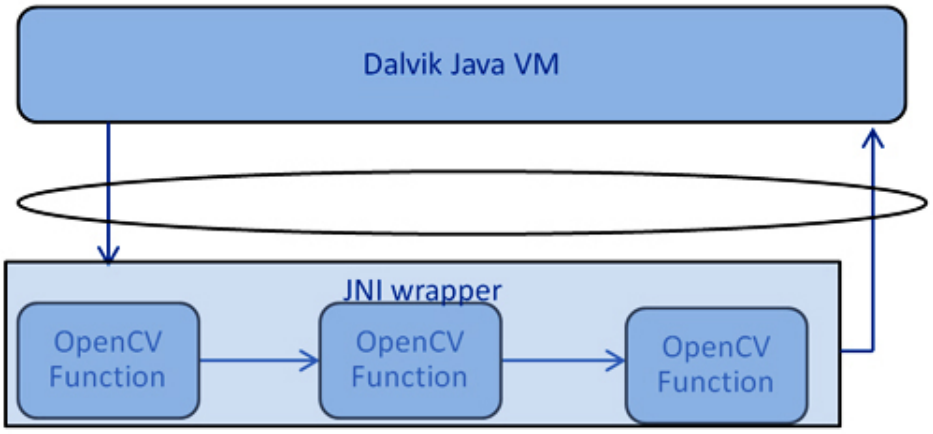


Figura 22: Utilizzando c++ nativo per unificare il tutto in un'unica chiamata

Nel nostro applicativo in realtà questa ottimizzazione avviene parzialmente in quanto eseguiamo due chiamate native delle quali riportiamo i prototipi:

```
1 JNIEXPORT void JNICALL Java_android2nxt_CameraActivity_calcHist(JNIEnv*, jobject,
      jlong addrFrame, jlong addrHist, int x0, int y0, int x1, int y1)
2
3 JNIEXPORT void JNICALL Java_android2nxt_CameraActivity_calcBack(JNIEnv*, jobject,
      jlong addrFrame,jlong addrHist, jlong addrBack)
```

Come lascia intendere il nome del primo metodo il suo scopo principale è quello di calcolare l'istogramma a partire dalla selezione dell'oggetto, i parametri di ingresso sono: il frame catturato dalla fotocamera, una matrice delle stesse dimensioni del frame, le coordinate della selezione; al ritorno della chiamata la matrice `addrHist` conterrà l'istogramma. La seconda funzione invece vede come parametri in ingresso sempre un frame catturato dalla fotocamera, l'istogramma ritornato dalla chiamata precedente e una matrice che conterrà, una volta eseguito il codice, la back projection del frame. Ora abbiamo tutto quello che serve per invocare Camshift che verrà trattato nella prossima sotto sezione. Chiudiamo con un ultimo dettaglio riguardante il debug del codice nativo che nonostante sia un argomento complesso, può essere inizialmente indirizzato grazie alla chiamata:

```
1 __android_log_print(ANDROID_LOG_DEBUG, "JNI DEBUG","cols is %i and row is %i",
      pMatFrame->cols,pMatFrame->rows);
```

Tale funzione ci permette di stampare qualsiasi genere di informazione proveniente dal codice nativo direttamente sull'output dedicato al log di Android per poi essere visionata grazie a logcat (il sistema di log di Android fornisce questo strumento per visionare l'output di debug eventualmente filtrandolo).

4.5 Tracking e trasmissione coordinate

Una volta eseguiti compiti preparatori come caricare le librerie e stabilire una connessione col brick NXT è possibile lanciare l'activity principale CameraActivity. Quest'ultima presenta a schermo la sequenza video ripresa dalla fotocamera grazie alla view `JavaCameraView`, inoltre è dotata di un *menù opzioni* dal quale è possibile eseguire le seguenti azioni:

- **selezionare l'oggetto** del quale fare il tracking grazie alla view appositamente sviluppata e contenuta nella classe `RectSelection`, un esempio è in Figura [23].

- **avviare il tracking** vero e proprio cambiando il valore di una variabile booleana sentinella presente nel principale metodo della classe, eseguito alla ricezione di ogni nuovo frame proveniente dalla fotocamera.

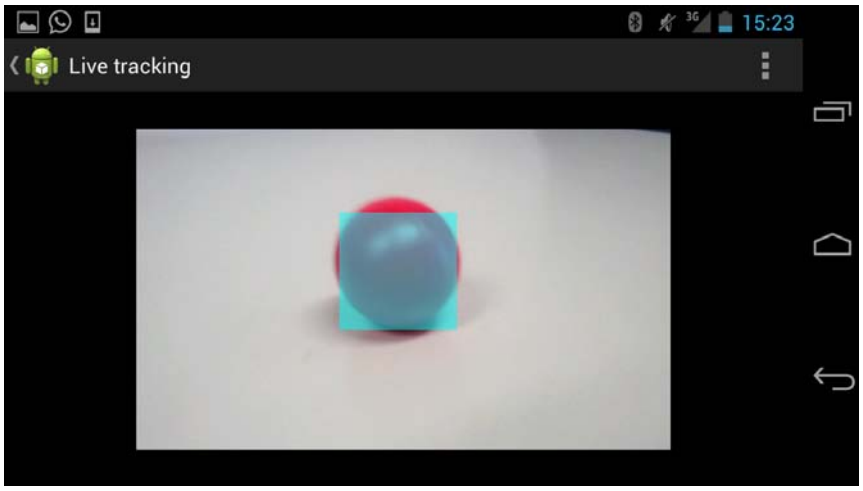


Figura 23: Rettangolo di selezione

Ritorniamo al momento del lancio di CameraActivity scatenato dall'invio di un intent proveniente da `MainActivity`, in quel frangente la prima operazione eseguita a destinazione è quella di estrarre l'handler per il thread che gestisce la connettività e salvarlo in una variabile locale [4]; il passo successivo consiste nel notificare JavaCameraView che la classe `CameraActivity` implementa l'interfaccia statica `CvCameraViewListener` contenuta nella classe `CameraBridgeViewBase`. Quest'ultima è una classe basilare che realizza l'interazione tra la fotocamera e le librerie OpenCV, il suo compito principale è quello di controllare quando la fotocamera può essere abilitata, processare ogni frame ricevuto e richiamare eventuali funzioni di callback registrate che solitamente manipolano il frame prima che sia disegnato a schermo. Il metodo `onCameraFrame(Mat inputFrame)` è appunto una di tali funzioni ed in essa si andranno ad eseguire le operazioni sul frame ricevuto. A questo punto la situazione è la seguente: abbiamo una selezione quadrata che contiene l'oggetto da inseguire, abbiamo istruito JavaCameraView di richiamare il metodo onCameraFrame() per ogni frame ricevuto e ci apprestiamo a invocare le funzioni necessarie per tenere traccia dell'oggetto tra frame consecutivi. Prima di far ciò è tuttavia necessario invocare il metodo `computeSelection()` che "trasforma" le coordinate della view dedicata alla selezione in coordinate di una ROI (region of interest) interna al frame; la Figura [24] mostra in verde le coordinate nel sistema di riferimento della JavaCameraView mentre in rosso quelle relative a CameraActivity ovvero all'activity visualizzata a schermo. Il metodo computeSelection() tenendo in

[4] In realtà viene salvato l'oggetto Messenger che permette di comunicare con l'handler del thread connettività, come verrà spiegato in seguito

considerazione gli offset evidenziati dalle frecce bidirezionali bianche riesce a tradurre le coordinate della selezione e salvarle nell'oggetto Rect.

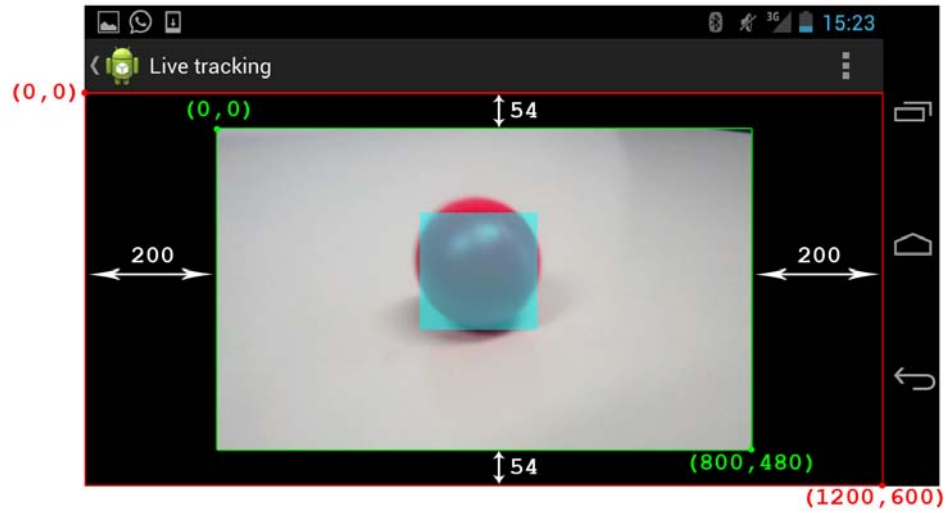


Figura 24: Coordinate delle varie componenti nel sistema di riferimento di Android

Una volta che le coordinate della ROI sono state calcolate finalmente possiamo richiamare il metodo nativo per il calcolo dell'istogramma e a partire dal frame successivo anche il metodo che calcola la back projection, infine troviamo la chiamata al metodo java che implementa l'algoritmo Camshift. Al ritorno da tutte queste chiamate avremo, salvato in una variabile locale, un BoundingRect che tiene traccia dell'oggetto obiettivo e del quale possiamo conoscere le coordiante del centro oltre che l'area. Per i nostri scopi useremo solo le prime e nel metodo `computeDistance()` grazie ad una serie di costrutti if annidati applichiamo la strategia di visual servoing che abbiamo discusso precedentemente per preparare in un'array locale le coordinate del punto che il robot dovrà raggiungere e che verranno spedite successivamente. Notiamo infine che per comunicare tali coordinate viene utilizzato l'oggetto Messenger, ricevuto dalla MainActivity, che incapsula l'handler del thread di connettività:

```
1 Message mex = Message.obtain();
2 mex.what = 1;
3 mex.arg1 = obj_ctr[0];
4 mex.arg2 = obj_ctr[1];
5 connManagerMessenger.send(mex);
```

4.6 Il brick NXT: la classe NXTNavigator

La classe `NXTNavigator` contiene il programma che governa il robot e si basa principalmente sulle classi `Navigator` e `DifferentialPilot` di leJOS. Quest'ultima è un'astrazione del meccanismo di pilotaggio di un NXT e contiene metodi per controllarne gli spostamenti: avanzare, retrocedere, ruotare ecc. Il suo funzionamento è legato alla presenza di due motori indipendentemente controllabili e che il robot sia quindi dotato di sterzo differenziale, lo schema è mostrato in Figura [25]:

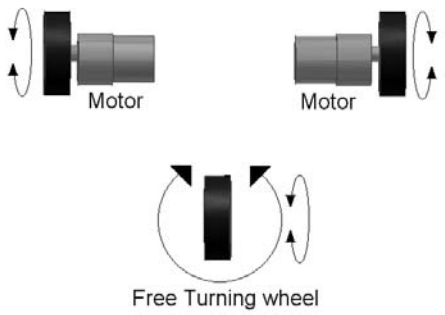


Figura 25: Schema di uno sterzo differenziale

La classe `Navigator` invece si assicura che il robot segua il percorso impartitogli che è formato da una sequenza di punti; il comportamento standard consiste nell'attraversare tutti i punti del percorso senza fermarsi per i punti intermedi tuttavia questa è un'impostazione modificabile. Internamente è dotata di un worker thread che invia i comandi al `MoveController` della classe, questo può essere o un DifferentialPilot o uno SteeringPilot. Infine Navigator usa un `PoseProvider` o meglio una classe che ne implementa l'interfaccia, il suo compito è quello di tenere traccia della posizione del robot durante la navigazione. All'avvio dell'applicazione, dopo l'inizializzazione delle variabili, comincia la fase dedicata allo stabilire la connessione con lo smartphone che consiste nell'attesa in entrata di una connessione Bluetooth; una volta che si verifica ciò avviene una breve fase di handshake ed il brick NXT viene messo in attesa di coppie di coordinate. Ricevuta una coppia di coordinate vengono immediatamente aggiunte al percorso da seguire tramite Navigator per poi rimanere in attesa che lo spostamento sia stato portato a termine; una volta che il robot è arrivato a destinazione invia un messaggio allo smartphone avvisandolo che ora è disponibile a ricevere ulteriori coordinate. Da notare che ad ogni punto raggiunto viene azzerata la posizione:

```
1 nav.getPoseProvider().setPose(new Pose());
```

questo perché ricevendo la coppia di coordinate successive, se non fosse avvenuto l'azzeramento, il robot avrebbe interpretato la destinazione come posizione rispetto al piano di riferimento iniziale.

Conclusioni e sviluppi futuri

In questo lavoro di tesi ci si era prefissi di sviluppare un applicativo per la piattaforma mobile Android che riuscisse ad eseguire il tracking di un oggetto obiettivo, per realizzare tale funzionalità sono state impiegate le librerie OpenCV ed in particolare l'implementazione dell'algoritmo Camshift in esse contenuto. L'applicativo inoltre avrebbe dovuto instaurare una connessione Bluetooth col robot LEGO Mindstorms NXT al quale comunicare l'entità degli spostamenti necessari a mantenere traccia dell'oggetto nella scena. L'applicazione prodotta, il cui codice sorgente è riportato in appendice, ha raggiunto il traguardo che ci eravamo dati in modo soddisfacente; di seguito è possibile osservare alcune fasi di tracking di una pallina:

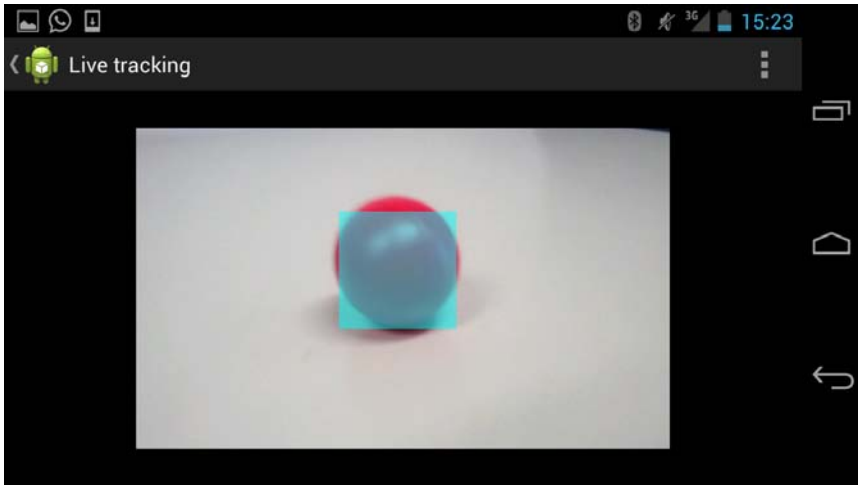


Figura 26: Viene scelto l'oggetto da inseguire

in primo luogo si è selezionata la pallina nella scena grazie all'apposito quadrato di selezione, infine si è avviato il processo di tracking.

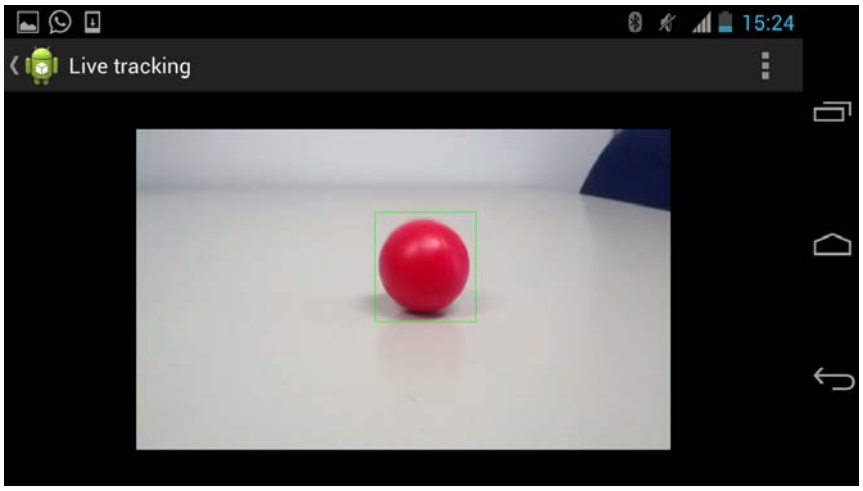


Figura 27: Tracking di una pallina

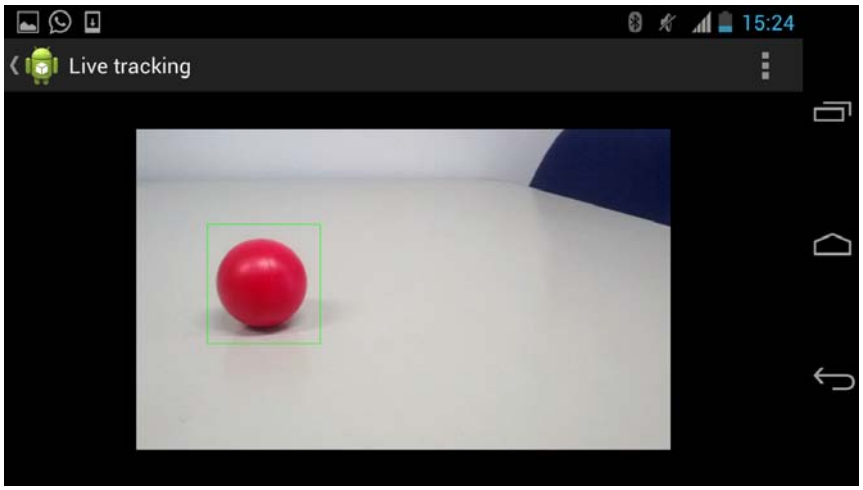


Figura 28: Tracking di una pallina dopo uno spostamento

Com'è possibile notare nelle Figure [27] e [28] il rettangolo verde, ovvero il BoundingRect, tiene traccia della pallina durante gli spostamenti. La capacità di inseguire l'oggetto si è potuto verificare che dipende dalla velocità con cui esso si muove nella scena (un oggetto troppo veloce verrà "perso" dall'algoritmo di tracking) e dalla sua lontananza rispetto alla fotocamera; in particolare quest'ultimo fattore si nota in un visibile e significativo calo nel numero di frame visualizzati per secondo ed è motivato dalla maggiore difficoltà di ricerca dell'obiettivo nella scena in quanto la sua area diminuisce progressivamente. Osserviamo che un calo nel numero di frame visualizzati equivale ad applicare l'algoritmo Camshift su fotogrammi in cui l'oggetto

ha compiuto un grande spostamento tra l'uno e l'altro, si può perciò ricondurre il problema alla difficoltà di tenere traccia di un obiettivo troppo veloce ovvero che si sposta troppo tra un fotogramma ed il successivo. La capacità di calcolo dello smartphone risulta quindi adeguata a realizzare object tracking su mobile e nonostante sia quantificabile in poco più di 1,5m la soglia di lontananza in cui p.es. diventa difficile seguire gli spostamenti della pallina di Figura [28], il fatto che le nuove generazioni di smartphone offrano processori significativamente più potenti di quello utilizzato in questo lavoro di tesi lascia ipotizzare che questi limiti tenderanno a ridursi. Dal punto di vista del robot NXT invece la tecnica di Image Based VS impiegata si è rivelata efficace anche se basilare, in particolare si è potuto notare che la complessità del problema proprio dell'asservimento visivo meriterebbe uno sforzo e un dispendio di energie di gran lunga superiore ma incompatibile con le necessità di questa tesi. Si è infatti palesata la facilità con cui il più piccolo movimento impartito al robot possa generare una reazione di eventi che portano all'instabilità del sistema dovuta principalmente al fatto che la legge di controllo deve tenere conto della dinamica del robot. Questa constazione può suggerire l'idea che possibili lavori futuri indirizzino i problemi sopra citati e forniscano una soluzione ottimale, andando ad estendere la parte robotica di questo progetto. Inoltre l'algoritmo Camshift ci permette di conoscere oltre alle coordinate del centro dell'oggetto anche la sua area (o meglio l'area del BoundingRect), questa informazione nella nostra implementazione non viene utilizzata e potrebbe venire impiegata per stimare la distanza dell'oggetto (magari imponendo l'obbligo di posizionare l'oggetto obiettivo ad una determinata distanza al momento della sua selezione iniziale).

A Codice sorgente dell'applicativo

Listing 2: **MainActivity.java**

```
1  /**
2   * Classe che ospita l'activity principale, il suo compito consiste nell'
       inizializzare
3   * le librerie OpenCV oltre a quella nativa, avviare il thread che stabilisce la
       connessione
4   * Bluetooth col brick NXT e, in caso tutto vada a buon fine, abilitare il lancio
5   * di CameraActivity.
6   *
7   * @author Alberto Gatto
8   *
9   */
10
11 package it.agatto.android2nxt;
12
13 import org.opencv.android.BaseLoaderCallback;
14 import org.opencv.android.LoaderCallbackInterface;
15 import org.opencv.android.OpenCVLoader;
16
17 import android.os.Bundle;
18 import android.os.Handler;
19 import android.os.Looper;
20 import android.os.Message;
21 import android.os.Messenger;
22 import android.annotation.SuppressLint;
23 import android.app.Activity;
24 import android.content.Intent;
25 import android.graphics.Color;
26 import android.util.Log;
27 import android.view.Menu;
28 import android.view.View;
29 import android.widget.Button;
30 import android.widget.TextView;
31
32 public class MainActivity extends Activity {
33
34     private static final String TAG="Android2NXT::MainActivity";
35     private NXTConnectionManager connManagerThread;
36     private Handler mainThreadHandler;
37
38     /**
39      * BaseLoaderCallback è la classe che gestisce il ritorno all'applicazione
40      * una volta terminato il compito dell'OpenCV manager: le casistiche si
41      * trattano in base al valore di ritorno.
42      */
43     private BaseLoaderCallback managerOpenCV = new BaseLoaderCallback(this) {
44
45
46         @Override
47         public void onManagerConnected(int status) {
48             TextView text;
49
50             switch(status) {
51             case LoaderCallbackInterface.SUCCESS:
52             {
53                 /**
54                  * Le librerie OpenCV sono state caricate con successo, possiamo
55                  * continuare con l'esecuzione del programma e caricare le altre
```

```
56                  * librerie necessarie.
57                  */
58                 Log.i(TAG, "Librerie OpenCV caricate con successo");
59                 text = (TextView) findViewById(R.id.text1);
60                 text.setText(text.getText()+" fatto.");
61
62                 // Libreria contenente codice nativo esterno
63                 System.loadLibrary("native_opencv");
64
65                 // Avvia la creazione del Thread che gestisce le comunicazione di
66                 // rete via Bluetooth con il brick NXT
67
68                 text = (TextView) findViewById(R.id.text2);
69                 text.setVisibility(View.VISIBLE);
70                 connManagerThread = new NXTConnectionManager(mainThreadHandler);
71                 connManagerThread.start();
72             } break;
73             case LoaderCallbackInterface.INIT_FAILED:
74             {
75                 // Il caricamento delle librerie OpenCV non è andato a buon fine
76                 Log.i(TAG, "Librerie OpenCV impossibili da caricare");
77                 text = (TextView) findViewById(R.id.text1);
78                 text.setTextColor(Color.RED);
79                 text.setText(text.getText()+" fallito.");
80
81             }
82             default:
83             {
84                 super.onManagerConnected(status);
85             }
86             }
87         }
88     };
89
90     @SuppressLint("HandlerLeak")
91     @Override
92     protected void onCreate(Bundle savedInstanceState) {
93         super.onCreate(savedInstanceState);
94
95         /**
96          * Viene definito un handler per la classe MainActivity; da notare che
97          * viene passato al costruttore dell'Handler il looper dell'activity.
98          * Ne viene in seguito definito il comportamento ovvero le azioni che
99          * deve compiere all'arrivo di messaggi diretti a MainActivity.
100         */
101        mainThreadHandler = new Handler(Looper.getMainLooper()) {
102            @Override
103            public void handleMessage(Message inputMessage) {
104                TextView text;
105                switch (inputMessage.what) {
106                case 1:
107                    /**
108                     *  inizializzazione del thread di rete andata a buon fine,
109                     *  possiamo abilitare il lancio della Camera Activity
110                     */
111                    text = (TextView) findViewById(R.id.text2);
112                    text.setText(text.getText()+" fatto.");
113                    Button button = (Button) findViewById(R.id.send_button);
114                    button.setEnabled(true);
115                    break;
116
117                case 0:
```

```
118                     // inizializzazione del thread di rete fallita,
119                     // notifica l'utente
120                     text = (TextView) findViewById(R.id.text2);
121                     text.setTextColor(Color.RED);
122                     text.setText(text.getText()+" fallito.");
123                     break;
124                 default:
125                     /*
126                      * Passa gli altri eventi alla UI, importante!!
127                      */
128                     super.handleMessage(inputMessage);
129             }
130             }
131         };
132
133         // Crea la view principale dell'àattivit principale
134         setContentView(R.layout.activity_main);
135
136         /**
137          * Inizializza le librerie OpenCV utilizzando OpenCV Manager
138          */
139
140         Log.i(TAG, "Caricamento delle librerie OpenCV in corso");
141         if (!OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_3, this,
    managerOpenCV))
142         {
143             Log.e(TAG, "Impossibile connettersi all'OpenCV Manager");
144         }
145     }
146
147     @Override
148     public boolean onCreateOptionsMenu(Menu menu) {
149         // Crea il menu e gli aggiunge gli eventuali elementi se presenti.
150         getMenuInflater().inflate(R.menu.activity_main, menu);
151         return true;
152     }
153
154     /**
155      * Il metodo viene chiamato quando l'utente clicca il bottone per
156      * l'avvio della CameraActivity.
157      * @param view
158      */
159     public void startTracking(View view) {
160         Log.i(TAG,"Metodo startTracking in esecuzione, avvio dell'àattivit di
    tracking in corso");
161         Intent intent = new Intent(this, CameraActivity.class);
162         intent.putExtra("connThreadHandler", new Messenger(connManagerThread.
    getHandler()));
163         startActivity(intent);
164     }
165
166 }
```

Listing 3: **CameraActivity.java**

```
1 /**
2  * Classe che contiene l'activity dedicata alla visualizzazione sullo schermo
3  * dei fotogrammi provenienti dalla fotocamera ed alla loro elaborazione.
4  * Utilizza la view sviluppata ad hoc e contenuta in RectSelection.java per
5  * permettere all'utente di selezionare una parte del fotogramma contenente
6  * l'oggetto di cui fare il tracking. Esegue il tracking ed invia le coordinate
```

```
7  * al thread adibito alla àconnettivit.
8  *
9  * @author Alberto Gatto
10 */
11 package it.agatto.android2nxt;
12
13 import org.opencv.android.CameraBridgeViewBase;
14 import org.opencv.android.CameraBridgeViewBase.CvCameraViewListener;
15 import org.opencv.core.Core;
16 import org.opencv.core.CvType;
17 import org.opencv.core.Mat;
18 import org.opencv.core.Point;
19 import org.opencv.core.Rect;
20 import org.opencv.core.RotatedRect;
21 import org.opencv.core.Scalar;
22 import org.opencv.core.TermCriteria;
23 import org.opencv.video.Video;
24
25 import android.os.Bundle;
26 import android.os.Message;
27 import android.os.Messenger;
28 import android.os.RemoteException;
29 import android.app.Activity;
30 import android.content.Intent;
31 import android.util.Log;
32 import android.view.Menu;
33 import android.view.MenuItem;
34 import android.view.View;
35 import android.view.WindowManager;
36
37 public class CameraActivity extends Activity implements CvCameraViewListener {
38
39     private static final String TAG="Android2NXT::CameraActivity";
40
41     // oggetto che rappresenta il rettangolo di selezione
42     private Rect prev_selection;
43
44     // è l'oggetto di ritorno del metodo Camshift
45     private RotatedRect curr_selection;
46
47     // array in cui verranno salvati i valori degli
48     // spostamenti da comunicare al robot
49     private int[] obj_ctr = new int[2];
50
51
52     private MenuItem               mSelectTarget;
53     private MenuItem               mTrackTarget;
54
55     private boolean              firstFrame = true;
56     private Mat                    hist;
57     private Mat                    backProjection;
58     private Mat                    mRgba;
59     private Mat                    hsv;
60     private Mat                    mask;
61
62     private CameraBridgeViewBase   mOpenCvCameraView;
63     private boolean rectVisible = false, isTracking = false;
64     private static boolean blind=false;
65
66     // Messenger che permette di inviare messaggi
67     // all'handler del thread di àconnettivit
68     private Messenger connManagerMessenger;
```

```
69
70      @Override
71      protected void onCreate(Bundle savedInstanceState) {
72          Log.i(TAG, "called onCreate");
73          super.onCreate(savedInstanceState);
74          Intent intent = getIntent();
75          connManagerMessenger = (Messenger)intent.getParcelableExtra("
        connThreadHandler");
76
77          getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
78
79          setContentView(R.layout.activity_camera);
80
81          /**
82           * Seleziona la view che mostra a schermo i fotogrammi provenienti
83           * dalla fotocamera ed aggiunge ai metodi di callback da chiamare
84           * all'arrivo di un nuovo frame quello implementato nella classe
85           *  CameraActivity.
86           */
87          mOpenCvCameraView = (CameraBridgeViewBase) findViewById(R.id.
        camera_activity_surface_view);
88          mOpenCvCameraView.setCvCameraViewListener(this);
89
90      }
91
92      @Override
93      public boolean onCreateOptionsMenu(Menu menu) {
94          Log.i(TAG, "called onCreateOptionsMenu");
95          mSelectTarget = menu.add("Select target");
96          mTrackTarget = menu.add("Track!").setEnabled(false);
97          // Inflate the menu; this adds items to the action bar if it is present.
98          getMenuInflater().inflate(R.menu.activity_camera, menu);
99          return true;
100     }
101
102     @Override
103     public void onPause()
104     {
105         if (mOpenCvCameraView != null)
106             mOpenCvCameraView.disableView();
107         super.onPause();
108     }
109
110     @Override
111     public void onResume()
112     {
113         super.onResume();
114         if (mOpenCvCameraView != null)
115             mOpenCvCameraView.enableView();
116     }
117
118     public void onDestroy() {
119         super.onDestroy();
120         if (mOpenCvCameraView != null)
121             mOpenCvCameraView.disableView();
122     }
123
124     // Inizializza tuttle le matrici necessarie
125     public void onCameraViewStarted(int width, int height) {
126         mRgba = new Mat(height, width, CvType.CV_8UC4);
127         hsv = new Mat(height, width, CvType.CV_8UC3);
128         hist = new Mat(height, width, CvType.CV_8UC1);
```

```
129         backProjection = new Mat(height, width, CvType.CV_8UC1);
130         mask = new Mat(height, width, CvType.CV_8UC1);
131         prev_selection = new Rect();
132         curr_selection = new RotatedRect();
133         obj_ctr[0]=0;
134         obj_ctr[1]=0;
135     }
136 
137     // Rilascia quanto precedentemente inizializzato
138     public void onCameraViewStopped() {
139         mRgba.release();
140         hsv.release();
141         hist.release();
142         backProjection.release();
143         mask.release();
144     }
145 
146     public Mat onCameraFrame(Mat inputFrame) {
147         // il frame inputFrame è nel formato RGBA
148         inputFrame.copyTo(mRgba);
149 
150         // se siamo abilitati al tracking, ovvero è stato selezionata una ROI
151         // (region of interest), procediamo col tracking dell'oggetto.
152         if (this.isTracking) {
153             // all'arrivo del primo frame è necessario calcolare l'istogramma
    partendo dalla ROI
154             if (this.firstFrame) {
155                 computeSelection();
156                 calcHist(mRgba.getNativeObjAddr(),  hist.getNativeObjAddr(),
    prev_selection.x, prev_selection.y, prev_selection.width, prev_selection.
    height);
157                 // istogramma calcolato
158                 this.firstFrame = false;
159             } else {
160                 // ora col frame successivo al primo dobbiamo estrarre
161                 // dall'immagine il solo canale hue dopo averla ovviamente
162                 // convertita in HSV; calcoliamo poi la backprojection ed
163                 // infine invochiamo camshift
164                 calcBack(mRgba.getNativeObjAddr(),hist.getNativeObjAddr(),
    backProjection.getNativeObjAddr());
165                 // TermCriteria sono i parametri che verranno passati a Mean
166                 // Shift e che decideranno come àdovr comportarsi
167                 this.curr_selection = Video.CamShift(backProjection, this.
    prev_selection, new TermCriteria(TermCriteria.EPS,10,1));
168                 if (curr_selection == null) Log.i(TAG, "curr_selection returned
    null");
169                 this.prev_selection = this.curr_selection.boundingRect();
170                 // disegna il rettangolo che contiene l'oggetto
171                 Core.rectangle(mRgba, new Point(prev_selection.x, prev_selection.
    y), new Point(prev_selection.x+prev_selection.width, prev_selection.y+
    prev_selection.height), new Scalar(0, 255, 0, 255));
172                 Log.i(TAG, "Coordinate del centro della selezione corrente: ("+
    curr_selection.center.x+", "+curr_selection.center.y+"), ha un'area di:"+
    curr_selection.size.area());
173                 if (!blind) {
174                     // Calcola variazione distanza/spostamento dell'oggetto
175                     // ed invia le coordinate al robot
176                     computeDistance();
177                     // SE è NECESSARIO SPOSTARSI poni blind=true ed invia
178                     // il comando, altrimenti semplicemente non fare niente
179                     // ed aspetta il frame successivo per decidere
180                     if (obj_ctr[0] != 0 || obj_ctr[1] !=0) {
```

```
181                         try {
182                             blind = true;
183                             Message mex = Message.obtain();
184                             mex.what = 1;
185                             mex.arg1 = obj_ctr[0];
186                             mex.arg2 = obj_ctr[1];
187                             connManagerMessenger.send(mex);
188                         } catch (RemoteException e) {
189                             // TODO Auto-generated catch block
190                             e.printStackTrace();
191                         }
192                     }
193                 }
194             }
195         }
196 
197         return mRgba;
198 
199     }
200 
201     /**
202      * Inizialmente il ùmen mostra come attivo solo l'elemento che se
203      * toccato fa comparire la selezione per permettere di specificare
204      * l'oggetto da inseguire.
205      * Una volta che si è posizionata la selezione il ùmen abilita anche
206      * l'elemento che avvia il tracking.
207      */
208     public boolean onOptionsItemSelected(MenuItem item) {
209         Log.i(TAG, "called onOptionsItemSelected; selected item: " + item);
210 
211         if (item == mSelectTarget) {
212             View selection = findViewById(R.id.rect_selection);
213             if (this.rectVisible) {
214                 selection.setVisibility(View.INVISIBLE);
215                 this.mTrackTarget.setEnabled(false);
216                 this.rectVisible = false;
217             } else {
218                 selection.setVisibility(View.VISIBLE);
219                 this.mTrackTarget.setEnabled(true);
220                 this.rectVisible = true;
221             }
222         } else if (item == mTrackTarget) {
223             View selection = findViewById(R.id.rect_selection);
224             if (this.isTracking) {
225                 mSelectTarget.setEnabled(true);
226                 mTrackTarget.setTitle("Track!");
227                 selection.setVisibility(View.VISIBLE);
228                 this.isTracking = false;
229             } else {
230                 mSelectTarget.setEnabled(false);
231                 mTrackTarget.setTitle("Stop");
232                 selection.setVisibility(View.INVISIBLE);
233                 this.firstFrame = true;
234                 this.isTracking = true;
235             }
236 
237 
238         }
239 
240         return true;
241     }
242 
```

```
243     /**
244      * Metodo che traduce le coordinate della view di selezione
245      * in coordinate del fotogramma.
246      */
247     private void computeSelection() {
248         View rect = findViewById(R.id.rect_selection);
249         int rectSize = RectSelection.getRectSize();
250 
251         int _x0, _y0, _x1, _y1;
252 
253         _x0 = (int) (rect.getX()) - 198;
254         _y0 = (int) (rect.getY()) - 54;
255         _x1 = rectSize+_x0;
256         _y1 = rectSize+_y0;
257 
258         // TODO: controlli molto basilari, necessario implementare
259         // qualche soluzione ùpi sicura.
260         if (_x0 > 0 && _y0 > 0 && _x1 > 0 && _y1 > 0) {
261             this.prev_selection.x = _x0;
262             this.prev_selection.y = _y0;
263             this.prev_selection.width = rectSize;
264             this.prev_selection.height = rectSize ;
265         }
266 
267         Log.i(TAG, "Coordinate della selezione sono: ("+_x0+", "+_y0+") e ("+_x1+
    ", "+_y1+");");
268 
269     }
270 
271     /**
272      * Metodo in cui viene deciso in quale area del fotogramma è
273      * contenuto il centro del boundingRect. Attua la strategia
274      * di visual servoing per guidare il robot nell'inseguimento
275      * dell'obiettivo.
276      */
277     private void computeDistance() {
278         int x = (int)curr_selection.center.x;
279         int y = (int)curr_selection.center.y;
280         if (x>267 && x<534 && y<160) {
281             // l'oggetto si è allontanato, avanza
282             obj_ctr[0]=10;
283         } else if (x<267 && y<160){
284             // l'oggetto si è allontanato e spostato a sx
285             obj_ctr[0]=10;
286             obj_ctr[1]=1;
287         } else if (x>534 && y<160) {
288             // l'oggetto si è allontanato e spostato a dx
289             obj_ctr[0]=10;
290             obj_ctr[1]=-1;
291         } else if (x>267 && x<534 && y>160) {
292             // l'oggetto è al centro
293             obj_ctr[0]=0;
294             obj_ctr[1]=0;
295         } else {
296             // valore di default
297             obj_ctr[0]=0;
298             obj_ctr[1]=0;
299         }
300     }
301 
302     public static synchronized void setBlindness(boolean blindness) {
303         blind = blindness;
```

```
304     }
305
306
307     public native void calcHist(long matAddrRgba, long matHist, int x0, int y0,
        int x1, int y1);
308     public native void calcBack(long matAddrRgba, long matHist, long matBckp);
309 }
```

Listing 4: **NXTConnectionManager.java**

```
1  /**
2   * Classe incaricata di gestire la connessione Bluetooth col brick NXT a partire
3   * dall'instaurare la comunicazione fino alla trasmissione delle coordinate.
4   *
5   * @author Alberto Gatto
6   */
7
8  package it.agatto.android2nxt;
9
10 import java.io.DataInputStream;
11 import java.io.DataOutputStream;
12 import java.io.IOException;
13
14 import android.annotation.SuppressLint;
15 import android.os.Looper;
16 import android.os.Handler;
17 import android.os.Message;
18 import android.util.Log;
19 import lejos.pc.comm.NXTComm;
20 import lejos.pc.comm.NXTCommException;
21 import lejos.pc.comm.NXTCommFactory;
22 import lejos.pc.comm.NXTInfo;
23
24
25 @SuppressLint("HandlerLeak")
26 public class NXTConnectionManager extends Thread {
27
28     private String TAG;
29     private String addrNXT;
30     private DataOutputStream dataOut;
31     private DataInputStream dataIn;
32     private NXTComm nxtComm;
33     private Handler conHandler, mainThreadHandler;
34     private Thread innerThread;
35     private boolean connected = false;
36
37     /*
38      * Nel costruttore andiamo ad inizializzare le variabili principali
39      * con i valori del brick utilizzato in laboratorio. Notiamo inoltre
40      * che il parametro ricevuto è l'handler del main thread associato
41      * a MainActivity.
42      */
43     public NXTConnectionManager(Handler handle) {
44         this.TAG="Android2NXT::NXTConnectionManager";
45         this.addrNXT="00:16:53:01:30:21";
46         this.mainThreadHandler = handle;
47     }
48
49
50     /*
51      * Metodo che inizialmente si occupa di aprire la connessione chiamando
```

```
52      * l'apposito metodo openConnection() e di avvisare MainActivity dell'esito
53      * di tale operazione.
54      * Successivamente avvia il thread della classe interna in modo che resti
55      * in ascolto sul canale d'ingresso per eventuali messaggi provenienti
56      * dall'NXT. Infine attiva il ciclo loop che resta in attesa di istruzioni
57      * per inviare le coordinate al brick.
58      *
59      * (non-Javadoc)
60      * @see java.lang.Thread#run()
61      */
62     @Override
63     public void run() {
64         Log.i(TAG, "Entrati nel metodo run() dell'NXTConnectionManager");
65         // Moves the current Thread into the background
66         android.os.Process.setThreadPriority(android.os.Process.
       THREAD_PRIORITY_BACKGROUND);
67         connected = this.openConnection();
68         if (connected) {
69             // è andata buon fine la connessione, aggiorna la UI ed
70             // aspetta le coordinate da inviare
71             mainThreadHandler.sendEmptyMessage(1);
72 
73             // attiva il thread in ascolto dell'NXT brick
74             innerThread = new innerListener();
75             innerThread.start();
76 
77             // qui viene realizzato il loop
78             Looper.prepare();
79             conHandler = new Handler() {
80 
81                 @Override
82                 public void handleMessage(Message inputMessage) {
83                     switch (inputMessage.what) {
84                     case 1:
85                         // è arrivata una nuova coppia di coordinate contenute
86                         // in arg1 e arg2.
87                         Log.i(TAG, "Comando per far avanzare il robot ricevuto");
88                         sendData(inputMessage.arg1,inputMessage.arg2);
89                     default:
90                         super.handleMessage(inputMessage);
91                     }
92                 }
93             };
94             Looper.loop();
95 
96         } else {
97             // connessione non andata a buon fine, aggiorna la UI
98             mainThreadHandler.sendEmptyMessage(0);
99         }
100    }
101 
102 
103    public boolean openConnection() {
104 
105        // Inizializza l'oggetto NXTInfo con le informazioni riguardo al brick
       usato
106        NXTInfo nxtInfo = new NXTInfo(NXTCommFactory.BLUETOOTH, "NXT", addrNXT);
107 
108        Log.i(TAG, "Tentativo di connessione a "+nxtInfo.name+" in corso");
109 
110        try {
111            nxtComm = NXTCommFactory.createNXTComm(NXTCommFactory.BLUETOOTH);
```

```
112
113             // Tenta di stabilire la connessione
114             if (!nxtComm.open(nxtInfo,NXTComm.PACKET)) {
115                 Log.i(TAG, "Fallito il tentativo di connessione.");
116                 return false;
117             }
118
119             Log.i(TAG, "Connessione stabilita, inizializzazione degli stream I/O
    in corso");
120             dataOut = new DataOutputStream(nxtComm.getOutputStream());
121             dataIn = new DataInputStream(nxtComm.getInputStream());
122
123             // Semplice handshake per verificare che il programma in esecuzione
124             // sul robot sia quello che ci aspettiamo.
125             dataOut.writeUTF(new String("NNP-H"));
126             dataOut.flush();
127             if (dataIn.readUTF().compareTo("NNP-H") != 0) {
128                 Log.i(TAG, "Handshake fallito, assicurarsi che il blocco NXT stia
     eseguendo il programma adeguato.");
129                 return false;
130             }
131             System.out.println("Handshake eseguito, la connessione è operativa");
132         } catch (IOException e) {
133             Log.e(TAG, "Impossibile attivare gli stream di I/O");
134             return false;
135         } catch (NXTCommException e) {
136             Log.e(TAG, "Impossibile stabilire una connessione Bluetooth");
137             return false;
138         }
139
140         return true;
141     }
142
143
144     public boolean closeConnection() {
145         try {
146             dataOut.close();
147             dataIn.close();
148             nxtComm.close();
149             conHandler.getLooper().quit();
150         } catch (IOException e) {
151             Log.e(TAG, "Errore durante la chiusura della connessione");
152             return false;
153         }
154         Log.i(TAG, "Connessione chiusa con successo");
155         return true;
156     }
157
158
159     /*
160      * Invia la coppia di coordinate contenuta nei parametri di ingresso.
161      */
162     private boolean sendData(float x,float y) {
163         try {
164             dataOut.writeFloat(x);
165             dataOut.writeFloat(y);
166             dataOut.flush();
167         } catch (IOException e) {
168             Log.e(TAG, "Errore durante l'invio dei dati");
169             return false;
170         }
171         return true;
```

```
172     }
173
174     public Handler getHandler() {
175         return this.conHandler;
176     }
177
178     /**
179      * Classe interna che si mette in ascolto sul canale d'ingresso
180      * della connessione Bluetooth.
181      * @author Alberto Gatto
182      *
183      */
184     public class innerListener extends Thread {
185
186         @Override
187         public void run() {
188             // Moves the current Thread into the background
189             android.os.Process.setThreadPriority(android.os.Process.
    THREAD_PRIORITY_BACKGROUND);
190             while (connected) {
191                 try {
192                     // aspetta il "via libera" da parte del blocco NXT e
193                     // cambia il valore di blind nella CameraActivity.
194                     if (dataIn.readUTF().compareTo("Next") == 0) {
195                         Log.i(TAG, "GoNEXT!");
196
197                         // ora CameraActivity òpu ricalcolare le prossime
198                         // coordinate da spedire.
199                         CameraActivity.setBlindness(false);
200                     }
201                 } catch (IOException e) {
202                     // TODO Auto-generated catch block
203                     e.printStackTrace();
204                 }
205             }
206         }
207
208
209     }
210
211
212 }
```

Listing 5: **RectSelection.java**

```
1  /**
2   * RectSelection è la classe che si occupa di realizzare la view di selezione
3   * dell'oggetto da inseguire. Disegna un quadrato sullo schermo e ne gestisce
4   * lo spostamento e la scalatura.
5   *
6   * @author Alberto Gatto
7   */
8  package it.agatto.android2nxt;
9
10 import android.content.Context;
11 import android.graphics.Canvas;
12 import android.graphics.Paint;
13 import android.graphics.Paint.Style;
14 import android.graphics.Rect;
15 import android.support.v4.view.MotionEventCompat;
16 import android.util.AttributeSet;
```

```
17 import android.util.Log;
18 import android.view.MotionEvent;
19 import android.view.ScaleGestureDetector;
20 import android.view.View;
21
22 public class RectSelection extends View {
23
24     private Paint mRectPaint;
25     private Rect rectangle;
26     private float mLastTouchX,mLastTouchY;
27     // The `active 'pointer is the one currently moving our object.
28     private int mActivePointerId = -1;
29     private ScaleGestureDetector mScaleDetector;
30
31     // fattore di scalatura, viene aggiornato ad ogni variazione
32     static private float mScaleFactor = 1.f;
33
34     // dimensione iniziale del quadrato
35     private final static int rect_size = 300;
36
37
38     /**
39      * Inizializza lo ScaleGestureDetector passandogli un'istanza della classe
40      * interna ScaleListener realizzata per gestire l'evento di scalatura della
41      * view. Inzializza inoltre anche l'oggetto Paint ed il Rect da disegnare
42      * sullo schermo.
43      * @param context
44      * @param attrs
45      */
46     public RectSelection(Context context,AttributeSet attrs) {
47         super(context,attrs);
48         mScaleDetector = new ScaleGestureDetector(context, new ScaleListener());
49         mRectPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
50         mRectPaint.setColor(android.graphics.Color.CYAN);
51         mRectPaint.setStyle(Style.FILL);
52         rectangle = new Rect(0,0,rect_size,rect_size);
53     }
54
55     private class ScaleListener extends ScaleGestureDetector.
       SimpleOnScaleGestureListener {
56
57         /**
58          * Quando viene riconosciuto un evento di scalatura recupera
59          * di quanto è stata scalata la view e salva il valore
60          * nella variabile della classe apposita; non permettere
61          * inoltre che si possa ingrandire o rimpicciolire troppo.
62          */
63         @Override
64         public boolean onScale(ScaleGestureDetector detector) {
65             mScaleFactor *= detector.getScaleFactor();
66             Log.d("LOG","Entrato in onScale con scaleFactor "+mScaleFactor);
67
68             // Don't let the object get too small or too large.
69             mScaleFactor = Math.max(0.1f, Math.min(mScaleFactor, 5.0f));
70
71             /** Chiamata necessaria quando si cambiano àpropriet della
72              *  view che ne modificano l'aspetto; comunica al sistema
73              *  una richiesta per essere ridisegnata.
74              */
75             invalidate();
76             return true;
77         }
```

```
78      }
79
80      /**
81       * Chiamata che svolge la funzione di disegno vera e propria,
82       * da notare che le chiamate a save() e restore() servono a
83       * disegnare sempre su un Canvas "fresco" nel senso di non
84       * alterato da precedenti chiamate.
85       */
86      @Override
87      protected void onDraw(Canvas canvas) {
88
89          super.onDraw(canvas);
90          canvas.save();
91          canvas.scale(mScaleFactor, mScaleFactor);
92          canvas.drawRect(rectangle, mRectPaint);
93          canvas.restore();
94      }
95
96      /**
97       * Metodo che gestisce i tocchi in ingresso; per ogni tocco
98       * recupera il suo indice (in un sistema multi touch possono
99       * esserci infatti ùpi tocchi contemporaneamente) e le coordinate.
100      * L'indice del primo tocco è l'unico che àavr il permesso
101      * di spostare la view almeno fino a quando àrimarr a contatto
102      * con lo schermo.
103      */
104     @Override
105     public boolean onTouchEvent(MotionEvent ev) {
106
107         mScaleDetector.onTouchEvent(ev);
108
109         switch (MotionEventCompat.getActionMasked(ev)) {
110         case MotionEvent.ACTION_DOWN: {
111
112             final int pointerIndex = MotionEventCompat.getActionIndex(ev);
113             final float x = MotionEventCompat.getX(ev, pointerIndex);
114             final float y = MotionEventCompat.getY(ev, pointerIndex);
115
116             // Remember where we started (for dragging)
117             mLastTouchX = x;
118             mLastTouchY = y;
119             // Save the ID of this pointer (for dragging)
120             mActivePointerId = MotionEventCompat.getPointerId(ev, pointerIndex);
121             break;
122
123         }
124
125         case MotionEvent.ACTION_MOVE: {
126             // Find the index of the active pointer and fetch its position
127             final int pointerIndex = MotionEventCompat.findPointerIndex(ev,
    mActivePointerId);
128
129             final float x = MotionEventCompat.getX(ev, pointerIndex);
130             final float y = MotionEventCompat.getY(ev, pointerIndex);
131
132             // Calculate the distance moved
133             final float dx = x - mLastTouchX;
134             final float dy = y - mLastTouchY;
135
136             this.setX(this.getX()+dx);
137             this.setY(this.getY()+dy);
138             invalidate();
```

```
139
140             // Remember this touch position for the next move event
141             mLastTouchX = x;
142             mLastTouchY = y;
143             break;
144
145         }
146
147         case MotionEvent.ACTION_UP: {
148             mActivePointerId = -1;
149             break;
150         }
151
152         case MotionEvent.ACTION_CANCEL: {
153             mActivePointerId = -1;
154             break;
155         }
156
157         case MotionEvent.ACTION_POINTER_UP: {
158
159             final int pointerIndex = MotionEventCompat.getActionIndex(ev);
160             final int pointerId = MotionEventCompat.getPointerId(ev, pointerIndex
    );
161             if (pointerId == mActivePointerId) {
162                 final int newPointerIndex = pointerIndex == 0 ? 1 : 0;
163                 mLastTouchX = MotionEventCompat.getX(ev, newPointerIndex);
164                 mLastTouchY = MotionEventCompat.getY(ev, newPointerIndex);
165                 mActivePointerId = MotionEventCompat.getPointerId(ev,
    newPointerIndex);
166             }
167             break;
168         }
169         }
170         return true;
171     }
172
173     // ritorna la misura del rettangolo inziale moltiplicata
174     // per il fattore di scalatura.
175     static public int getRectSize() {
176         return (int)(rect_size*mScaleFactor);
177     }
178
179 }
```

Listing 6: **nativa_opencv.cpp**

```
1  /**
2   * Libreria di funzioni native che si occupano della maggior parte
3   * delle operazioni svolte sulle immagini provenienti dalla fotocamera.
4   *
5   * @author: Alberto Gatto
6   */
7
8  #include <jni.h>
9  #include <opencv2/core/core.hpp>
10 #include <opencv2/highgui/highgui.hpp>
11 #include <opencv2/imgproc/imgproc.hpp>
12 #include <android/log.h>
13
14 using namespace std;
15 using namespace cv;
```

```
16
17 extern "C" {
18
19     /**
20      * Parametri usati per ridurre il rumore nell'immagine.
21      * vmin imposta un limite per valori troppo vicini al nero.
22      * smin imposta un limite per valori troppo vicini al grigio.
23      * vmax invece imposta un limite per valori di àluminosit
24      * troppo alti.
25      */
26     int vmin=65,vmax=256,smin=55;
27     //int vmin = 10, vmax = 256, smin = 30;
28
29 JNIEXPORT void JNICALL Java_it_agatto_android2nxt_CameraActivity_calcHist(JNIEnv
       *, jobject, jlong addrFrame, jlong addrHist, int x0, int y0, int x1, int y1)
       {
30
31     Mat* pMatFrame = (Mat*)addrFrame;
32     Mat* pMatHist = (Mat*)addrHist;
33     Mat frameHSV;
34     Mat mask;
35     Mat hue;
36     Rect selection;
37
38     float hranges[] = {0, 180};
39     const float* phranges = hranges;
40     // numero di contenitori/colonne dell'istogramma
41     int bins = 30;
42
43     selection.x = x0;
44     selection.y = y0;
45     selection.width = x1;
46     selection.height = y1;
47
48     // chiamate di debug, l'output èvisibile grazie a logcat
49     __android_log_print(ANDROID_LOG_DEBUG, "JNI DEBUG","cols is %i and row is %i",
           pMatFrame->cols,pMatFrame->rows);
50     __android_log_print(ANDROID_LOG_DEBUG, "JNI DEBUG", "selection x is %i, y is %
           i, width is %i, height is %i",x0,y0,x1,y1);
51
52     selection &= Rect(0, 0, pMatFrame->cols, pMatFrame->rows);
53
54     // ARGB stored in java as int array becomes BGRA at native level
55     cvtColor(*pMatFrame, frameHSV, CV_BGR2HSV, 0);
56     // esegue l'esclusione dei valori che rendono l'immagine rumorosa
57     inRange(frameHSV, Scalar(0, smin, MIN(vmin, vmax)), Scalar(180, 256, MAX(vmin,
           vmax)), mask);
58     int ch[]={0,0};
59     hue.create(frameHSV.size(),frameHSV.depth());
60     // estra il solo canale hue
61     mixChannels(&frameHSV,1,&hue,1,ch,1);
62
63     Mat roi(hue,selection), maskroi(mask,selection);
64     // calcola l'istogramma e lo normalizza
65     calcHist(&roi, 1, 0, maskroi, *pMatHist, 1, &bins, &phranges);
66     normalize(*pMatHist,*pMatHist,0,255,NORM_MINMAX);
67 }
68
69 JNIEXPORT void JNICALL Java_it_agatto_android2nxt_CameraActivity_calcBack(JNIEnv
       *, jobject, jlong addrFrame,jlong addrHist, jlong addrBack) {
70
71     Mat* pMatFrame = (Mat*)addrFrame;
```

```
72    Mat* pMatHist = (Mat*)addrHist;
73    Mat* pMatBack = (Mat*)addrBack;
74    Mat frameHSV;
75    Mat mask;
76    Mat hue;
77
78    float hranges[] = {0, 180};
79    const float* phranges = hranges;
80
81    cvtColor(*pMatFrame, frameHSV, CV_BGR2HSV, 0);
82    inRange(frameHSV, Scalar(0, smin, MIN(vmin, vmax)), Scalar(180, 256, MAX(vmin,
          vmax)), mask);
83    int ch[]={0,0};
84    hue.create(frameHSV.size(),frameHSV.depth());
85    mixChannels(&frameHSV,1,&hue,1,ch,1);
86
87    // calcola la back projection
88    calcBackProject(&hue, 1, 0, *pMatHist, *pMatBack, &phranges);
89    *pMatBack &= mask;
90 }
91
92 }
```

Listing 7: **NXTNavigator.java**

```
1  /**
2   * Classe che controlla gli spostamenti del robot NXT.
3   * Attende che venga stabilta una connessione tramite Bluetooth con un device
       esterno, stabilisce con un
4   * handshake se il tipo di protocollo di comunicazione è quello prescelto.
5   * Quest'ultimo prevede l'invio di una coppia di coordinate che il robot deve
       interpretare come coordinate
6   * di un target da seguire.
7   * nota: avanzare di 100f significa compiere 10cm in avanti
8   * @author Alberto Gatto
9   *
10  */
11
12 package it.agatto.nxtclient;
13
14 import java.io.DataInputStream;
15 import java.io.DataOutputStream;
16 import java.io.IOException;
17 import lejos.nxt.Button;
18 import lejos.nxt.ButtonListener;
19 import lejos.nxt.Motor;
20 import lejos.nxt.Sound;
21 import lejos.nxt.comm.Bluetooth;
22 import lejos.nxt.comm.NXTConnection;
23 import lejos.robotics.navigation.DifferentialPilot;
24 import lejos.robotics.navigation.Navigator;
25 import lejos.robotics.navigation.Pose;
26
27 public class NXTNavigator {
28     private String name;
29     private boolean connected;
30     private DataInputStream is;
31     private DataOutputStream os;
32     private boolean go;
33     private Float[] position;
34     private DifferentialPilot pilot;
```

```
35      private Navigator nav;
36      private NXTConnection btconn;
37
38      public NXTNavigator(String name) {
39          this.name = name;
40          this.connected = false;
41          this.go = true;
42          this.btconn = null;
43          this.position = new Float[2];
44          this.position[0]= (float) 0;
45          this.position[1]= (float) 0;
46          this.pilot = new DifferentialPilot(56, 116, Motor.A, Motor.B, false);
47          this.nav = new Navigator(pilot);
48          Button.ESCAPE.addButtonListener(new ButtonListener() {
49
50              public void buttonPressed(Button b) {
51                  go = false;
52
53              }
54
55              @Override
56              public void buttonReleased(Button b) {
57                  // TODO Auto-generated method stub
58
59              }
60          });
61      }
62
63      /**
64       * Tenta di stabilire la connessione aspettando per un determinato
65       * periodo che un client si connetta.
66       *
67       * @return true se la connessione è andata a buon fine, false altrimenti
68       * @throws IOException
69       */
70      private boolean waitConnection() throws IOException {
71
72          System.out.print(this.name + ": waiting for a bluetooth incoming
        connection..");
73          btconn = Bluetooth.waitForConnection(20000, NXTConnection.PACKET);
74
75          if (btconn == null) {
76              System.out.println(" timeout reached.");
77              return this.connected;
78          } else {
79              os = btconn.openDataOutputStream();
80              is = btconn.openDataInputStream();
81              System.out.println(" established.");
82              System.out.print(this.name + ": NNP protocol handshake in progress.."
        );
83              try {
84                  String s = is.readUTF();
85                  if ((s.compareTo("NNP-H")) == 0) {
86                      os.writeUTF(s);
87                      os.flush();
88                      System.out.println(" passed.");
89                      this.connected = true;
90                      Sound.beepSequence();
91                  } else {
92                      System.out.println(" failed.");
93                      is.close();
94                      os.close();
```

```
95                  btconn.close();
96                  return this.connected;
97              }
98          } catch (IOException e) {
99              btconn.close();
100             System.out.println("I/O problem!");
101             throw new IOException();
102         }
103     }
104     return this.connected;
105 }
106
107 /**
108  * Metodo che àd il via al lavoro della classe; inizialmente si mette in
109  * ascolto di una connessione in ingresso e se òci va a buon fine entra
110  * nel loop infinito di attesa coordinate e spostamento del robot verso
111  * le stesse una volta ricevute.
112  */
113 public void startNav() {
114     float x,y;
115     try {
116         if (this.waitConnection()) {    // connessione attiva
117             System.out.println(this.name +": receiving coordinates..");
118             while (this.go) {
119                 try {
120                     x = this.is.readFloat();
121                     y = this.is.readFloat();
122                     System.out.println("x -> "+x+" , y -> "+y);
123                     nav.addWaypoint(x, y);
124
125                     /*
126                      * teoricamente aspettiamo che finisca di raggiungere la
127                      * destinazione per avvertire il cellulare che lo
128                      * spostamento è stato compiuto e òpu calcolare il
129                      * prossimo passo da compiere.
130                      * */
131                     nav.followPath();
132
133                     // TODO: controllare il valore di ritorno... torna false
134                     // se si è fermato prima di raggiunere la destinazione
135                     // per altre causee
136                     if (nav.waitForStop()) {
137                         // destinazione raggiunta senza intoppi
138                         os.writeUTF("Next");
139                         os.flush();
140                     } else {
141                         // è sorto qualche problema ed il robot si è fermato
142                     }
143
144                     // riassetta il sistema di coordinate rispetto al quale
145                     // interpreta i comandi ricevuti
146                     nav.getPoseProvider().setPose(new Pose());
147                 } catch (IOException e) {
148                     // TODO Auto-generated catch block
149                     e.printStackTrace();
150                 }
151             }
152             System.out.println(this.name+": exit the main loop.");
153         }
154     } catch (IOException e) {
155         // TODO Auto-generated catch block
156         System.out.println("Connessione interrotta lato pc");
```

```
157         }
158     }
159 
160     public void stopNav() {
161         try {
162             this.is.close();
163             this.os.close();
164         } catch (IOException e) {
165             // TODO Auto-generated catch block
166             e.printStackTrace();
167         }
168         this.btconn.close();
169     }
170 
171 }
```

Riferimenti bibliografici

[1] Gary Bradski, *Computer vision face tracking for use in a perceptual user interface*, Intel Technology Journal (1998), Online at `http://www.dis.uniroma1.it/~nardi/Didattica/SAI/matdid/tracking/camshift.pdf`.

[2] Gary Bradski and Adrian Kaehler, *Learning opencv: Computer vision with the opencv library*, 1st ed., O'Reilly Media, Sep. 2008.

[3] Charlie Collins, Michael Galpin, and Matthias Kaeppler, *Android in practice*, Manning Publications, 2011.

[4] Dorin Comaniciu and Peter Meer, *Mean shift analysis and applications*, Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, vol. 2, 1999, p. 1197–1203.

[5] Keinosuke Fukunaga and Larry D. Hostetler, *The estimation of the gradient of a density function, with applications in pattern recognition*, IEEE Transactions on Information Theory, vol. 21, 1975, p. 32–40.

[6] Stephan Göbel, Ruben Jubeh, Simon-Lennert Raesch, and Albert Zündorf, *Using the android platform to control robots*, Research paper, Kassel University, 2011.

[7] Kayton B. Parekh, *A computer vision application to accurately estimate object distance*, Honors projects, Macalester College, 2010.