UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Estensione della libreria Aria per l'utilizzo del dispositivo Kinect come sensore di distanza

Elaborato di esame di:

Paolo Laffranchini, Stefania Tomagra

Consegnato il: **17 Luglio 2013**

Sommario

Il lavoro svolto consiste nell'adattare la libreria Aria al fine di utilizzare il sensore Kinect in ambiente Linux per permettere ad un robot di sfruttare la mappa di profondità ottenibile da questo innovativo dispositivo come sorgente di informazioni in merito all'ambiente in cui si trova.
L'obiettivo è di utilizzare questi dati per localizzare ostacoli durante il suo movimento e permettergli di reagire in maniera adeguata in situazioni dove la presenza di oggetti possa interferire con la sua navigazione.
Per conseguire questo obiettivo, sarà necessario estendere la libreria Aria in modo tale che possa interagire con il dispositivo Kinect per l'acquisizione della matrice di profondità.
La definizione delle nuove classi dovrà seguire lo stesso meccanismo di funzionamento delineato da quelle già esistenti in modo tale da garantire che le nuove funzionalità siano compatibili con ciò che è già stato implementato all'interno della libreria.

1. Introduzione

1.1. Il Kinect

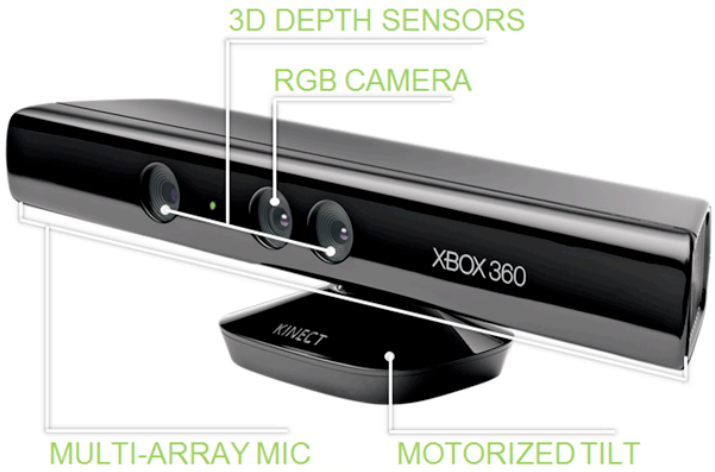

Figura 1: Il dispositivo Kinect

Il Kinect è un accessorio che è stato originariamente pensato come dispositivo di controllo per la console Microsoft Xbox 360. Il 31 maggio 2010 è stato pubblicato negli Stati Uniti da PrimeSense un brevetto dal titolo "Depth Mapping using Project Patterns" che descrive la tecnologia implementata dal Kinect.

L'apertura del Kinect mostra la presenza di [2]:

Figura 2: Componenti interni del Kinect

- un array di microfoni orientati verso il basso (3 sul lato destro e uno sul lato sinistro);
- tre apparati ottici utilizzati per il riconoscimento visuale del corpo in movimento: due videocamere ed un sensore infrarossi aggiuntivo;
- una ventola per la dissipazione del calore;
- 64MB di memoria flash DDR2;
- un accelerometro Kionix KXSD9 a tre assi;
- Prime Sense PS1080--A2, il chip che rappresenta il cuore della tecnologia di Kinect.

Più in dettaglio gli apparati ottici si compongono da una telecamera RGB e da un doppio sensore di profondità a raggi infrarossi. Il doppio sensore di profondità è costituito da un proiettore strutturato a infrarossi e da una telecamera sensibile alla stessa banda e che serve per leggere quanto rilevato di raggi infrarossi.

Visti i componenti hardware, uno schema di funzionamento del dispositivo potrebbe essere il seguente

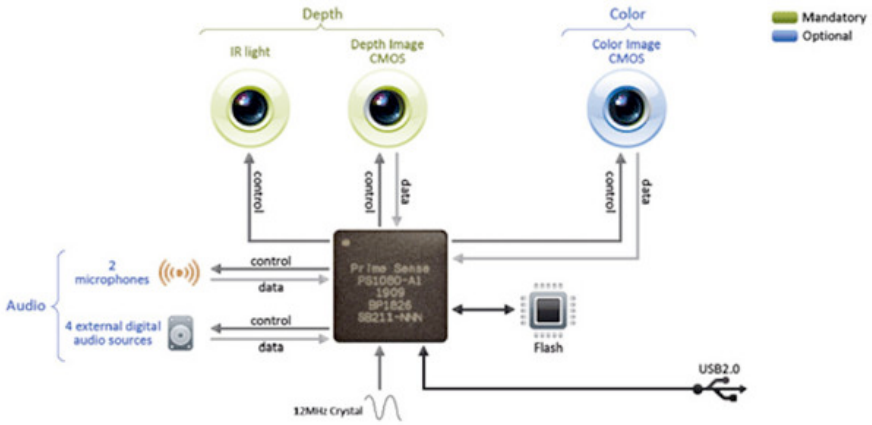

Figura 3: Schema di funzionamento del Kinect

nel quale viene evidenziato come il chip PS1080-A2 di PrimeSense sovraintenda tutta la procedura di analisi della scena controllando adeguatamente gli apparati ottici e audio al fine di raccogliere le informazioni di cui necessita. A puro scopo informativo di seguito un elenco dei vari chip presenti

all'interno di Kinect:

- Wolfson Microelectronics WM8737G – Stereo ADC con microfono preamp
- Fairchild Semiconductor FDS8984 – N-Channel PowerTrench MOSFET
- NEC uPD720114 – USB 2.0 hub controller
- H1026567 XBOX1001 X851716-005 GEPP
- Marvell AP102 – SoC con interfaccia Camera controller
- Hynix H5PS5162FF 512 megabit DDR2 SDRAM
- Analog Devices AD8694 ; Quad, Low Cost, Low Noise, CMOS Rail-to-Rail Output Operational Amplifier
- TI ADS7830I – 8-Bit, 8-Channel Sampling A/D Converter with I2C Interface
- Allegro Microsystems A3906 – Low Voltage Stepper and Single/Dual DC Motor Driver
- ST Microelectronics M29W800DB – 8 Mbit (1Mb x8 or 512Kb x16) NV Flash Memory
- PrimeSense PS1080-A2 – SoC image sensor processor
- TI TAS1020B USB audio controller front and center

Completa la lista dell'hardware del dispositivo un accelerometro Kionix MEMS KXSD9, utilizzato per controllare l'inclinazione e stabilizzare l'immagine.

1.1.1. Il sensore Kinect

Attraverso studi sperimentali [3], è stato determinato che il sensore di Kinect processa il flusso video ad un frame rate di 30 Hz. Lo stream RGB usa una risoluzione VGA ad 8 bit di profondità con dimensione di 640 x 480 pixel con un filtro di Bayer per il colore, mentre il sensore monocromatico di profondità genera un flusso dati a risoluzione VGA (640 x 480 pixel) con 11 bit di profondità, quindi con 2048 livelli di sensitività. Il sensore di Kinect ha un raggio di utilizzo da un minimo di 0.4 ad un massimo di 8.0 metri. Esso ha un campo di vista angolare di 57° orizzontalmente e 43° verticalmente, mentre il fulcro motorizzato è capace di uno spostamento di 27° gradi verso l'alto o verso il basso, controllabile via software. Il campo orizzontale ad una distanza di circa 0.8 m è all'incirca 87 cm, mentre quello verticale circa 63 cm, risultando quindi in una risoluzione di 1.3 mm per pixel. La risoluzione spaziale x/y a 2 metri di distanza dal sensore è di 3 mm, mentre la risoluzione di profondità z sempre a due metri di distanza è di 1 cm.

2. Il problema affrontato

Lo scopo di questo lavoro è di integrare le potenzialità del dispositivo Kinect in ambiente Linux al fine di ampliare i modi di funzionamento di un ambiente dedicato alla programmazione di robot mobili. Nel nostro caso specifico la libreria presa in considerazione nel nostro lavoro è Aria, sviluppata da Mobile Robots™.

L'obiettivo del progetto consisterà quindi nell'ampliamento di Aria in modo tale da fornire la possibilità al programmatore di sfruttare tutte le potenzialità di questo dispositivo nell'ambito della robotica. Infatti, un'opportunità consiste nella sua possibile sostituzione ad un altro dispositivo, come ad esempio il laser, dal momento che è possibile utilizzare la matrice di profondità per implementare un comportamento di rilevazione di distanza fra il Kinect stesso e ciò che lo circonda.

A tal proposito i dati di profondità acquisiti possono essere sfruttati per valutare la distanza degli oggetti che circondano il robot, in modo tale da rilevare l'eventuale presenza di ostacoli che potrebbero causare interferenze con il suo sistema di navigazione. Queste ultime, infatti, potrebbero introdurre alterazioni del corretto svolgimento delle sue operazioni o addirittura nei casi peggiori danni fisici al robot.

Lo scopo di questo lavoro quindi è quello di utilizzare il Kinect come dispositivo di distanza per il rilevamento degli ostacoli, grazie alle informazioni messe a disposizione dalla matrice di profondità. Infatti, in ciascun "pixel" della matrice è presente la stima della distanza dell'oggetto che può essere localizzato spazialmente a partire dall'indice dello stesso all'interno di questa struttura dati.

3. La libreria Aria

La libreria Aria è un framework dedicato allo sviluppo di applicazioni software per la programmazione di robot mobili.

Essa fornisce numerose classi che astraggono il concetto di robot e ne permettono la sua gestione, che comprende il monitoraggio dello stato del robot e l'interfacciamento ai componenti montati su di esso. Questi ultimi possono essere di varia natura e definiscono le possibilità comportamentali del robot in termini di come esso può interagire con l'ambiente circostante. Aria definisce per questo scopo differenti classi che raggruppano i vari tipi di dispositivi, che possono essere utilizzati a seconda della loro natura.

Il lavoro affrontato si concentra sui dispositivi di distanza dai quali il robot potrà ricevere informazioni riguardo ciò che lo circonda, che saranno poi elaborate in modo tale da variare il suo comportamento in diverse situazioni.

Per questa particolare tipologia di dispositivi Aria definisce la classe ArRangeDevice che ne rappresenta la loro generalizzazione. Questa incapsula e definisce i metodi comuni ai diversi dispositivi affinché la libreria stessa possa sfruttare questa interfaccia per ottenere i loro dati in maniera indipendente dall'implementazione specifica di ciascuno di essi.

Ad esempio attraverso il metodo getCurrentReadingPolar il robot può ottenere, senza preoccuparsi dei dettagli di funzionamento dei dispositivi, la distanza dell'oggetto a lui più vicino presente nella regione polare specificata come parametro d'ingresso al metodo stesso.

Un'altra classe che Aria mette a disposizione al programmatore è ArDeviceConnection, la quale, basandosi sullo stesso principio precedentemente enunciato, permette la comunicazione a basso livello con i dispositivi. Ad esempio, a seconda del tipo di connessione che un dispositivo richiede, può fornire una trasmissione attraverso una porta seriale o attraverso la rete.

4. La soluzione adottata

4.1. ArKinect

La classe ArKinect è stata creata come sottoclasse di ArRangeDeviceThreaded, che deriva a sua volta da ArRangeDevice e che permette di eseguire un thread separato che si occuperà specificatamente di ottenere le informazioni necessarie dal dispositivo.

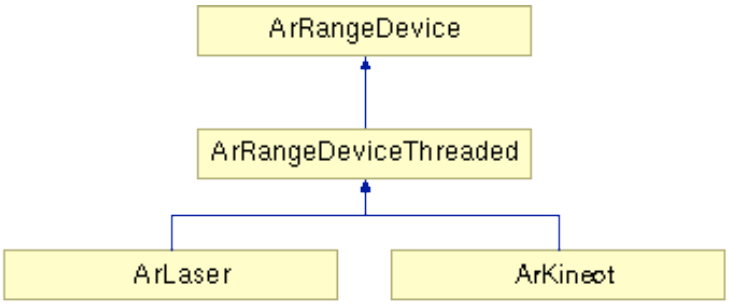

Figura 4: Schema di ereditarietà della classe ArKinect

La classe segue il modello di funzionamento di ArLaser. I metodi principali che fornisce consistono in:

- setParams: inizializza alcuni parametri che dipendono in maniera diretta dalle caratteristiche del dispositivo.
- runThread: acquisisce i dati direttamente dal dispositivo in maniera asincrona rispetto al resto del programma.
- sensorInterp: esegue il parsing delle informazioni ottenute dalla precedente funzione in maniera specifica a seconda del dispositivo, provvedendo a effettuare la trasformazione di queste ultime in dati significativi e memorizzando i risultati in opportune strutture dati.
- raw_depth_to_millimeters: trasforma i valori presenti nella matrice di profondità in distanze in millimetri in modo tale da poter essere utilizzate adeguatamente all'interno della libreria.
- get_depth_matrix: ritorna al programma utente la più recente matrice profondità che è stata ottenuta dal dispositivo Kinect.
- set_kinect_angle, get_kinect_angle, get_whole_tilt_state: alcuni metodi di utilità per il controllo del motore del dispositivo.

Di seguito saranno descritti i metodi più rilevanti delle classi ArKinect.

4.1.1. SetParams

È un metodo protetto della classe ArKinect, tra i parametri in ingresso ci sono startingDegrees e endingDegrees che delimitano la regione polare da scandire. Per quanto riguarda i parametri di misura il campo di vista angolare orizzontale del Kinect è di 57° e la riga della matrice di profondità in cui sono salvati i dati è lunga 640 pixel. La risoluzione angolare quindi è pari a 57/640=0.01. In altre parole le misurazioni hanno fra loro una distanza angolare di 0.01° o, equivalentemente, per ogni grado di scostamento angolare il sensore esegue 10 letture (1°/0.1). Ogni punto di misurazione è detto step.

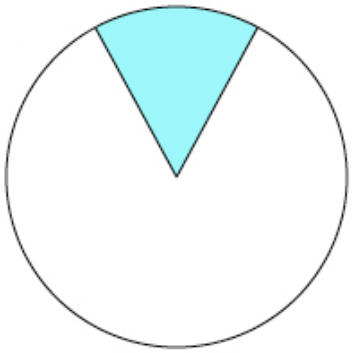

Figura 5: Campo visivo del Kinect

Questo metodo dovrà quindi operare la seguente trasformazione:

double startingStepRaw;
double endingStepRaw;
startingStepRaw = -(startingDegrees - 28.5) / .1;
endingStepRaw = -(endingDegrees - 28.5) / .1;

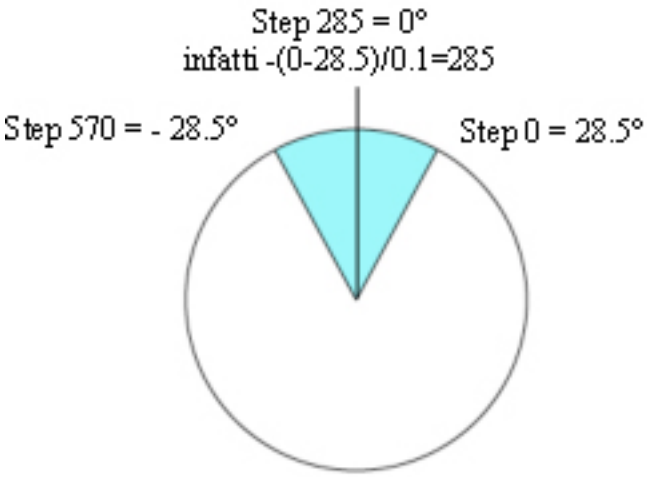

Figura 6: Schema per il calcolo degli step che scandiscono il campo visivo

Il metodo setParams richiama a sua volta setParamsByStep utilizzando i nuovi valori calcolati per gli step.

4.1.2. SetParamsBySteps

È un metodo protetto della classe ArKinect; tra i parametri in ingresso ci sono startingStep e endingStep (calcolati dal metodo SetParams) che delimitano la regione polare da scandire, quest'ultima deve essere compresa tra 0 e 570. Se i valori sono corretti, verranno inizializzate le strutture dati in cui saranno memorizzate le letture dal sensore. Il numero di queste ultime, dipenderà direttamente dai valori startingStep, endingStep e clusterCount. Le strutture dati appena create, istanze della classe ArSensorReadings, verranno quindi aggiunte al vettore di letture myRawReadings.

Un particolare importante di questo metodo consiste inoltre nel calcolo dell'angolo che indicherà, per ciascuna lettura, la direzione con cui questa è stata presa. Infatti, dal momento che vengono utilizzati più pixel adiacenti della matrice di profondità per stimare la distanza di un oggetto, l'angolo della lettura reale che verrà memorizzata deve essere normalizzato per rispecchiare questo aspetto.

Di seguito è riportata la parte più significativa del codice.

```
ArSensorReading *reading;
int onStep;
double angle;

for (onStep = myStartingStep;
    onStep < myEndingStep;
    onStep += myClusterCount)
{
  /// FLIPPED
  if (!myFlipped)
    angle = ArMath::subAngle(ArMath::subAngle(28.5, onStep * 0.1),
                             myClusterMiddleAngle);
  else
    angle = ArMath::addAngle(ArMath::addAngle(-28.5, onStep * 0.1),
                             myClusterMiddleAngle);
  reading = new ArSensorReading;
  reading->resetSensorPosition(ArMath::roundInt(mySensorPose.getX()),
                               ArMath::roundInt(mySensorPose.getY()),
                               ArMath::addAngle(angle,
                                                mySensorPose.getTh()));
  myRawReadings->push_back(reading);
}
```

4.1.3. RunThread

Questo metodo della classe ArKinect ereditato da ArRangeDeviceThreaded, è eseguito, in un thread a se stante, per tutta la durata del programma.

È costituito da un ciclo while, al cui interno è controllato se è stata instaurata la connessione tramite la variabile booleana myIsConnected. Questa, è impostata a true dal metodo blockingConnect nel caso in cui sia stato possibile instaurare un collegamento valido con il dispositivo Kinect. Nel caso in cui si verifichino errori, come ad esempio l'assenza del dispositivo o un errore durante la sua inizializzazione, il valore di myIsConnected rimarrà a false, indicando di conseguenza l'anomalia al resto della libreria.

In seguito sono acquisiti in maniera asincrona i dati dal dispositivo nel seguente modo:

```
readingRequested.setToNow();
myReadingMutex.lock();
if(readLine((char*)tmpDepthReading, DEPTH_MATRIX_SIZE_BYTE, 0)){
reading_available = true;
  }
  else reading_available = false;
  myReadingMutex.unlock();
```

Come prima cosa, viene memorizzato il momento in cui è stata fatta la richiesta di lettura di nuovi dati dal dispositivo. In seguito, viene richiamato il metodo readLine, che sfrutterà a sua volta il l'oggetto ArKinectConnection per aggiornare la mappa di profondità tramite i dati acquisiti dal kinect; il risultato è salvato in tmpDepthReading.

4.1.4. SensorInterp

È un metodo protetto della classe ArKinect che è richiamato per fare il parsing dei dati acquisiti dal metodo runThread, in modo da ottenere le stime indicanti le distanze degli oggetti dal robot.

Nell'attuale implementazione, da questa struttura dati è estratto il solo vettore centrale, contenente quindi 640 valori di profondità. Questi dati sono mediati, quindi, dieci per volta, in modo da ottenere una risoluzione angolare di 1°. Queste misurazioni vengono convertite in millimetri tramite la funzione raw_depth_to_millimeters ed il risultato è poi inserito in un vettore di oggetti di tipo ArSensorReading.

Per prima cosa è controllato che ci siano letture disponibili ed è controllato che non siano troppo vecchie per essere processate; successivamente viene eseguito il parsing delle letture ottenute e i dati contenuti nella matrice di profondità sono trasformati in distanze calcolate in millimetri; di seguito è riportata la parte del codice che si occupa di questa seconda parte.

```
 lockDevice();
 myDataMutex.lock();
 int i = 0;
 int len = DEPTH_MATRIX_WIDTH_PX;
 std::list<ArSensorReading *>::reverse_iterator it;
 ArSensorReading *sReading;

 /****  Parse the reading from the device and calculate range ****/
 bool ignore = false;
 int j = 0;
 int k=0;
 for (it = myRawReadings->rbegin(), i = 0;
     it != myRawReadings->rend() && i < len;
     it++, i += 10)
 {
   int tmp=0;
   int real_depth_value = 0;
   float median_real_depth = 0;
```

```
  for(j=i;j<i+10;j++){
        real_depth_value = raw_depth_to_millimeters(currentDepthReading[j+(DEPTH_MATRIX_WIDTH_PX*DEPTH_MATRIX_HEIGHT_PX/2)]);
        tmp+=real_depth_value;
  }
   median_real_depth = tmp/10;
  sReading = (*it);
  sReading->newData(median_real_depth, pose, encoderPose, transform, counter,
                    time, ignore, 0);
  //ArLog::log(ArLog::Normal, "median_real_depth: %f\n", median_real_depth);
 k++;
}
myDataMutex.unlock();
kinectProcessReadings();
unlockDevice();
```

4.1.5. ReadLine

È un metodo protetto della classe ArKinect ed è utilizzato al suo interno per leggere, tramite un oggetto di ArKinectConnection, i dati provenienti dal sensore di profondità del kinect.

Memorizza la lettura ottenuta direttamente nel buffer che viene passato come parametro al metodo stesso.

```
myConnMutex.lock();
ret = myConn->read((const char*)buf, size, 0);
if(ret==-1){
ArLog::log(ArLog::Terse, "ArKinectConnection::read: Connection invalid: kinect device not opened.");
}
myConnMutex.unlock();
```

4.1.6. raw_depth_to_millimeters

Questo metodo viene richiamato da sensorInterp per trasformare i valori ottenuti direttamente dal Kinect in distanze in millimetri che rispecchino la reale locazione degli oggetti rilevati.

Dal momento che la matrice di profondità può essere acquisita utilizzando diversi formati di rappresentazione delle informazioni, il metodo si comporta in maniera differente a seconda del tipo di formato che è stato impostato.

Nel caso in cui sia stato scelto il formato FREENECT_DEPTH_MM o FREENECT_DEPTH_REGISTERED, i valori presenti nella matrice di profondità sono già espressi in millimetri ed il metodo non effettua alcuna trasformazione.

Nel caso in cui, invece, sia stato scelto il formato FREENECT_DEPTH_11BIT, i valori "grezzi" presenti nella matrice verranno convertiti secondo la seguente trasformazione:

$$real\_depth = 123.6 * \tan(raw\_depth / 2842.5 + 1.1863);$$ [5]

4.1.7. Get_depth_matrix

Questo metodo può essere utilizzato per ritornare al programma utente l'intera matrice di profondità.

Ad esempio, come accade nel programma di esempio fornito, questa funzionalità può essere utilizzata nel caso si fosse interessati a visualizzare in una finestra ciò che il dispositivo sta inquadrando in un determinato momento.

La funzione copia in un buffer fornito dall'utente la più recente matrice di profondità che è stata letta dal dispositivo.

4.1.8. Metodi per il controllo del motore del tilt del Kinect

La classe ArKinect presenta inoltre alcuni metodi di utilità per il controllo dell'inclinazione del Kinect.

Questi metodi sono stati forniti per permettere al dispositivo di cambiare l'inclinazione a cui si trova in modo tale da fornire ulteriori possibilità di utilizzo dello stesso. Infatti, potrebbe essere possibile sfruttare questa funzionalità per ampliare il raggio d'azione della IR camera ed ottenere quindi un maggiore campo visivo da poter sfruttare per il monitoraggio dell'ambiente.

Per esempio, questa funzionalità potrebbe risultare interessante per verificare la presenza di eventuali gradini sul percorso del robot o per permettergli di controllare se determinati ostacoli siano ad una altezza tale da potergli consentire il passaggio al di sotto di essi.

I metodi che sono stati implementati sono:

- bool set_kinect_angle(double angle): consente di inclinare il dispositivo ad una angolazione scelta dall'utente. Il range di angoli possibili è compreso tra [-30, 30].
- double get_kinect_angle(): ritorna il valore in cui è attualmente inclinato il dispositivo.
- freenect_raw_tilt_state* get_whole_tilt_state(): ritorna lo stato globale del motore del Kinect; si rimanda alla documentazione di libfreenect per i dettagli su questa struttura dati.

4.2. ArKinectConnection

ArDeviceConnection è la classe di riferimento della libreria Aria per gestire la connessione con i dispositivi, quindi ArKinectConnection, utilizzata per connettersi al Kinect è stata creata come sua sottoclasse.

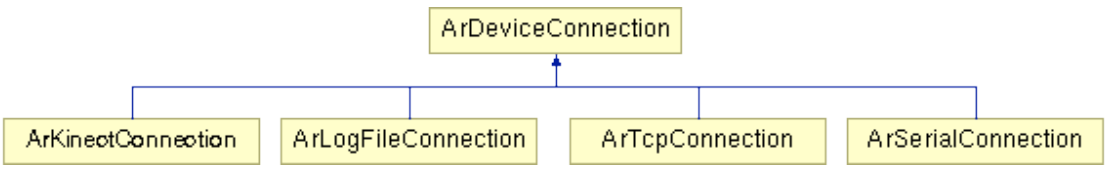

Figura 7: Schema di ereditarietà della classe ArKinectConnection

Il costruttore di questa classe riceve in ingresso due parametri: il primo indica il livello di log che si desidera utilizzare per l'interfacciamento con il dispositivo; il secondo indica il modo di funzionamento della IR camera: questo può essere settato al valore FREENECT_DEPTH_MM oppure al valore FREENECT_DEPTH_REGISTERED nel caso si voglia che la matrice della profondità ritornata rispecchi già la distanza reale, in millimetri, degli oggetti nel campo visivo del Kinect; in alternativa, è possibile utilizzare il valore FREENECT_DEPTH_11BIT in modo tale che i valori di profondità siano quelli “grezzi” del dispositivo (range: 0-2048).

I metodi principali che fornisce consistono in:

- Read: si occupa di leggere i dati rilevati dal kinect.
- Open: instaura la connessione con il dispositivo e inizializza le variabili principali.
- Close: chiude la connessione con il kinect e libera il contesto.

La classe presenta inoltre i metodi get_kinect_angle, set_kinect_angle, get_whole_tilt_state; questi ultimi sono accessibili dai programmi utente tanto quanto gli omonimi metodi della classe ArKinect precedentemente descritti (questi ultimi chiamano i primi). Per questo motivo è possibile utilizzare alla stessa maniera gli uni o gli altri. Ciononostante è preferibile servirsi di quelli della classe ArKinect per mantenere la coerenza implementativa che impone che sia il dispositivo ad essere rappresentato da tali metodi e non la connessione con cui si comunica con esso.

4.2.1. Read

Questo metodo è utilizzato per copiare i dati di profondità rilevati dal kinect in un buffer di dimensione 640x480x2.

```
if(myConnStatus == STATUS_OPEN){
   kinect_mutex.lock();
   if(!freenect_process_events(kinect_ctx)){
      memcpy((void*)data,(void*)depth_data,640*480*sizeof(uint16_t));
   }
   kinect_mutex.unlock();
}
```

4.2.2. Open

È il metodo utilizzato per instaurare la connessione con il kinect.

Dopo aver inizializzato il contesto tramite la funzione freenect_init, è controllato che sia presente il dispositivo e che possa essere effettuata una connessione. Sono poi settati i seguenti parametri: il buffer in cui salvare i dati di profondità, la callback per gestire l'evento di lettura dei dati e la risoluzione per la telecamera a infrarossi che si occupa dell'acquisizione dei dati. Infine il kinect è posto a 0° rispetto al suo asse di movimento verticale e la variabile myConnStatus è impostata a STATUS_OPEN.

```
kinect_mutex.lock();
if (freenect_init(&kinect_ctx, NULL) < 0) {
  ArLog::log(ArLog::Terse, "ArKinectConnection::init_device: Error: Kinect device could not be initialized.");
    return -1;
}
freenect_set_log_level(kinect_ctx, log_level);
freenect_select_subdevices(kinect_ctx,(freenect_device_flags)
                                (FREENECT_DEVICE_MOTOR|FREENECT_DEVICE_CAMERA));
int nr_devices = freenect_num_devices (kinect_ctx);
if (nr_devices < 1) {
   ArLog::log(ArLog::Terse, "ArKinectConnection::find_device: Error: no kinect devices found.");
   close();
   myConnStatus = STATUS_OPEN_FAILED;
```

```
  return -1;
}
if (freenect_open_device(kinect_ctx, &kinect_dev, 0) < 0) {
   ArLog::log(ArLog::Terse, "ArKinectConnection::open: Error: Kinect device could not be opened.");
   close();
   myConnStatus = STATUS_OPEN_FAILED;
   return -1;
}
if(freenect_set_depth_buffer(kinect_dev,(void*)depth_map)<0){
   ArLog::log(ArLog::Terse, "ArKinectConnection::set_depth_buffer: Error: could not be set depth buffer on Kinect device.");
   close();
   myConnStatus = STATUS_OPEN_FAILED;
   return -1;
}
freenect_set_depth_callback(kinect_dev, ArKinectConnection::get_depth_matrix);
if(freenect_set_depth_mode(kinect_dev,freenect_find_depth_mode(FREENECT_RESOLUTION_MEDI
        UM, kinect_depth_format)) < 0){
  ArLog::log(ArLog::Terse, "ArKinectConnection::open: Error: cannot set depth mode on Kinect device.");
    close();
    myConnStatus = STATUS_OPEN_FAILED;
   return -1;
}
int start_depth = freenect_start_depth(kinect_dev);
if(start_depth){
   ArLog::log(ArLog::Terse, "ArKinectConnection::start_depth: Error: depth data stream could not be started.");
   myConnStatus=STATUS_OPEN_FAILED;
   return -1;
}
myConnStatus = STATUS_OPEN;
kinect_mutex.unlock();
```

4.2.3. Close

Questa funzione setta a STATUS_CLOSED_NORMALLY la variabile myConnStatus e poi richiama il metodo freenect_shutdown(kinect_ctx) della libreria libfreenect che si occupa di chiudere la connessione.

4.3. Kinect_example

Per verificare il funzionamento delle modifiche apportate alla libreria Aria è stato scritto il programma kinect_example, basato sui programmi originali ActiveMedia Robots™ e ha l'obiettivo di testare l'utilizzo del dispositivo Kinect come sensore di distanza.

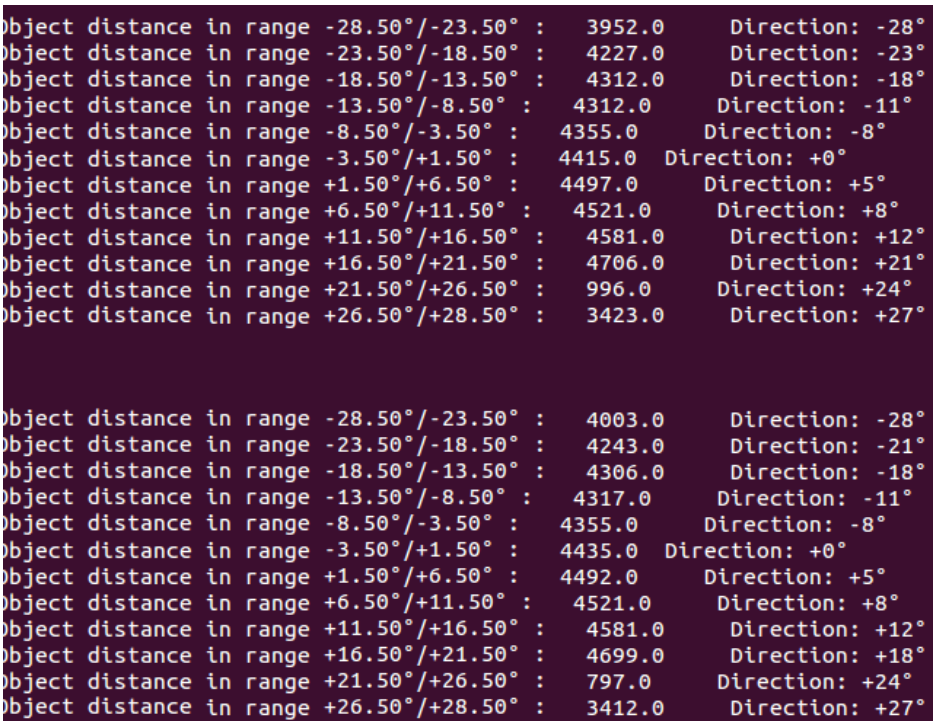

Figura 8: Screenshot dimostrativo del programma

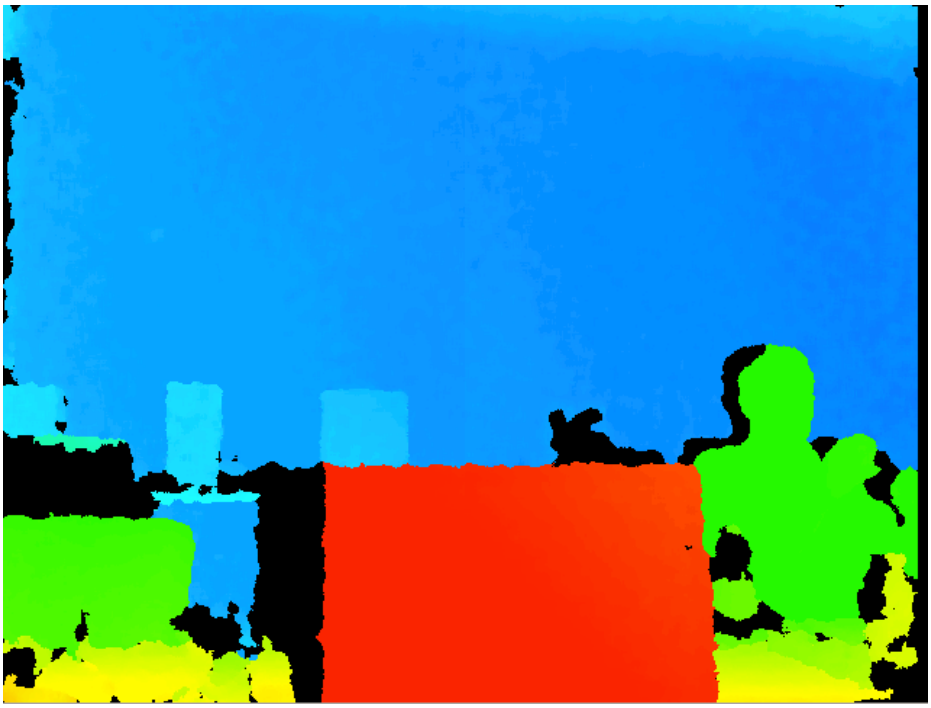

Figura 9: Visualizzazione della matrice di profondità

Il programma di esempio, ad ogni iterazione, stampa sullo schermo una serie di letture chiamando il metodo getCurrentReadingPolar della classe ArRangeDevice. Questo metodo è stato chiamato per dieci volte consecutive scandendo l'intera regione polare visibile dal dispositivo Kinect con un incremento di 5° fra ciascuna lettura. Lo screenshot seguente mostra due iterazioni in cui vengono mostrate le dieci letture specificando la sezione angolare in cui ciascuna è stata effettuata, riportando la distanza in millimetri dell'oggetto più vicino nella suddetta regione ed indicando la direzione in cui questo si trova.

Inoltre, durante l'esecuzione del programma è anche mostrato all'utente un'immagine che rende visibile in tempo reale cosa sta inquadrando il Kinect. I vari oggetti sono rappresentati con colori diversi in relazione alla distanza dal dispositivo, ad esempio quelli più vicini sono mostrati con sfumature di rosso e quelli più lontani con gradazioni di blu.

È possibile inoltre cambiare l'angolazione in cui è orientata verticalmente il Kinect: cliccando il tasto "q" si può inclinare il dispositivo verso l'alto, tramite il pulsante "z" verso il basso e con "a" torna invece alla posizione di partenza (parallela al piano di appoggio del Kinect).

5. Modalità operative

5.1. Componenti necessari

5.1.1. Kinect

I dati sono acquisiti tramite il sensore di profondità del Kinect. Ha le seguenti caratteristiche:

- intervallo di rilevamento: da 0.4m a 4m;
- campo di vista: 57° orizzontalmente, 43° verticalmente
- spostamento fulcro motorizzato: 27° in verticale

5.1.2. Libreria Aria

La libreria Aria utilizzata è la 2.7.2 ed è scaricabile da www.ing.unibs.it/~arl/.

5.1.3. Libreria Libfreenect

La libreria cross-platform open source libfreenect fornisce le funzionalità base per connettersi alla telecamera, configurare i valori, scaricare immagini e offre funzioni per interagire con i led e il motore. L'obiettivo è di fornire un riferimento per interfacciarsi con il Kinect da un hardware diverso dalla Xbox.

5.1.4. Linux

Il lavoro svolto è stato sviluppato per essere utilizzato in ambiente Linux.

5.1.5. Il codice sviluppato

Sono necessari i file sorgenti ArKinectConnection.cpp e ArKinectDevice.cpp con i relativi header ArKinectConnection.h e ArKinectDevice.h.

Gli header devono essere inseriti nella cartella di Aria include/ mentre i sorgenti devono essere inseriti nella cartella src/

Il makefile deve essere modificato per includere i suddetti nell'elenco dei file sorgenti: per fare ciò è necessario aggiungere i nomi dei due .cpp alla variabile CFILES definita all'interno del Makefile stesso.

Inoltre, è necessario specificare nella opzione di linking CXXLINK la libreria libfreenect.

1. CFILES := \
   …
   ArKinect.cpp \
   ArKinectConnection.cpp \
   …

2. CXXLINK := … -lfreenect

5.2. Modalità di installazione

5.2.1. Installazione e compilazione Libfreenect su Linux

Si è scelto di installare le librerie libfreenect compilandole manualmente come indicato dalla sezione Getting Started nella wiki di OpenKinect (http://openkinect.org/wiki/Getting_Started/it#Compilazione_manuale_sotto_Linux).
Per la compilazione è necessario avere i seguenti programmi/librerie installati:

- cmake
- libusb-1.0-0
- libusb-1.0-0-dev
- pkg-config
- libglut3
- libglut3-dev

Questo comando dovrebbe installare tutti i pacchetti necessari per recuperare, compilare ed eseguire la demo glview:

$ sudo apt-get install cmake libglut3-dev pkg-config build-essential libxmu-dev libxi-dev libusb-1.0-0-dev
$ git clone https://github.com/OpenKinect/libfreenect.git
$ cd libfreenect
Per eseguire cmake e compilare:
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
$ sudo ldconfig /usr/local/lib64/
Per testare che vada il Kinect
$ sudo bin/glview

5.2.2. Installazione libreria Aria

Per l'installazione della libreria Aria si rimanda al lavoro svolto da Ceriani, Gatti, Caffi, Tomson [1], in particolare al paragrafo 3.1 che spiega passo per passo come installare e compilare tale libreria. Si noti, infine, che tale lavoro è aggiornato alla versione 2.7.0 di Aria, mentre in questa sede si è utilizzata la 2.7.2.

5.3. Avvertenze

- Il metodo get_depth_matrix della classe ArKinectDevice, utilizzato come callback del dispositivo e che permette il salvataggio della matrice di profondità, è stato dichiarato come metodo static della classe ArKinectConnection. Il motivo risiede nel fatto che la funzione

freenect_set_depth_callback della libreria libfreenect non permette che venga utilizzato un metodo non static membro di una classe come callback.
Questo implica che anche la variabile depth_data dove si salva temporaneamente la matrice di profondità ottenuta dal dispositivo è stata dichiarata static per poter essere accessibile dal suddetto.
A causa di questi motivi nell'attuale versione della classe, non è possibile utilizzare più di un dispositivo Kinect allo stesso tempo.

- Nota sul funzionamento: si è notato un comportamento anomalo del dispositivo, il quale ogni volta che viene riconnesso al pc tramite cavo usb non permette l'acquisizione della matrice di profondità in maniera corretta. Pertanto, per questo motivo, la prima esecuzione del programma mostrerà sempre il massimo valore di distanza che è stato impostato per il dispositivo all'interno dell'applicazione dimostrativa.

6. Conclusioni e sviluppi futuri

Grazie al lavoro svolto è ora possibile sfruttare le funzionalità della libreria Aria al fine di utilizzare il Kinect in modo da ottenere dati relativi alle distanze degli oggetti, così da permettere a un robot di evitare eventuali ostacoli presenti.

Per quanto riguarda alcune possibili rifiniture al lavoro effettuato, la prima estensione da applicare potrebbe essere quella di implementare il settaggio dei parametri del dispositivo tramite i file di configurazione del robot. Allo stato corrente, infatti, questi ultimi sono stati cablati all'interno del codice della classe, ma potrebbe risultare utile una loro ridefinizione a tempo d'esecuzione in base a ciò che preferisce l'utente. A questo proposito sarebbe necessario aggiungere alla classe ArKinect un metodo pullUnsetParamsFromRobot per gestire durante l'inizializzazione delle strutture dati i casi in cui l'utente richieda esplicitamente l'utilizzo di attributi personalizzati che differiscano da quelli di default. Per implementare tale metodo è possibile usare come linea guida il metodo implementato dalla classe ArLaser.

Una seconda modifica potrebbe essere quello di valutare la possibilità di utilizzo di Kinect multipli per ottenere in maniera parallela maggiori informazioni sull'ambiente circostante.

In merito invece ad eventuali sviluppi futuri, le alternative sono molteplici:

- Modificare il calcolo delle distanze degli oggetti utilizzando un metodo più sofisticato rispetto al semplice utilizzo della riga centrale della matrice di profondità. Ad esempio utilizzare un insieme di righe adiacenti.
- Aggiungere un metodo per l'acquisizione dello streaming video della telecamera rgb.
- Sfruttare il video rgb per implementare, tramite classici algoritmi di computer vision, la localizzazione degli oggetti nell'ambiente per poi effettuare un mappaggio di questi ultimi sulla matrice di profondità in modo tale da ottenere la loro distanza. Questa funzionalità potrebbe ad esempio essere sfruttata da un robot dotato di bracci meccanici per il recupero di oggetti depositati nell'ambiente.
- Implementare un programma per la ricostruzione in 3D dell'ambiente circonstante.
- Implementare delle ArAction ad-hoc che sfruttino la matrice di profondità e lo streaming video per poter consentire al robot di seguire oggetti o persone in movimento.
  Sempre a questo proposito potrebbe essere anche possibile implementare dei comportamenti in cui il robot possa essere comandato tramite gestualità.

Altri lavori

Un altro lavoro in cui è stato preso in considerazione il dispositivo Kinect è stato quello di Panni e Musatti [6]. È possibile consultarlo sul sito http://www.ing.unibs.it/~arl/ nel caso si cerchino ulteriori informazioni sul dispositivo che non sono state trattate in questa sede.

Bibliografia

[1] Ceriani L., Caffi A., Gatti S., Tomson N. M. .: "Guida all'installazione di Aria versione 2.7.0 su Acer Aspire One e collaudo su robot Tobor", giugno 2009. http://www.ing.unibs.it/~arl/

[2] Maggio L., Ricupero S. C., Viviani A., Repici G. .: "Il dispositivo Microsoft Kinect", 2012. http://www.luigimaggio.altervista.org/documenti/tesinaRealtaVirtualeKinect.pdf

[3] Frati V. .: "Hand Tracking con Kinect e rendering per dispositivi aptici indossabili", 2011. http://www.ing.unisi.it/biblio/tesi/frativalentino_1011.pdf

[4] Sito ufficiale libreria libfreenect: openkinect.org

[5] http://openkinect.org/wiki/Imaging_Information

[6] Musatti, Panni: "Introduzione all'utilizzo del sensore Kinect in ambiente UNIX", 2013. http://www.ing.unibs.it/~arl/

Indice

SOMMARIO **1**

1. INTRODUZIONE **1**
1.1. Il Kinect **1**
1.1.1. Il sensore Kinect 3

2. IL PROBLEMA AFFRONTATO **4**

3. LA LIBRERIA ARIA **4**

4. LA SOLUZIONE ADOTTATA **5**
4.1. ArKinect **5**
4.1.1. SetParams 5
4.1.2. SetParamsBySteps 6
4.1.3. RunThread 7
4.1.4. SensorInterp 8
4.1.5. ReadLine 9
4.1.6. raw_depth_to_millimeters 9
4.1.7. Get_depth_matrix 10
4.1.8. Metodi per il controllo del motore del tilt del Kinect 10
4.2. ArKinectConnection **10**
4.2.1. Read 11
4.2.2. Open 11
4.2.3. Close 12
4.3. Kinect_example **13**

5. MODALITÀ OPERATIVE **14**
5.1. Componenti necessari **14**
5.1.1. Kinect 14
5.1.2. Libreria Aria 14
5.1.3. Libreria Libfreenect 14
5.1.4. Linux 14
5.1.5. Il codice sviluppato 14
5.2. Modalità di installazione **15**
5.2.1. Installazione e compilazione Libfreenect su Linux 15
5.2.2. Installazione libreria Aria 15
5.3. Avvertenze **15**

6. CONCLUSIONI E SVILUPPI FUTURI **16**

ALTRI LAVORI **17**

BIBLIOGRAFIA **17**

INDICE **18**