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)

Reimplementazione ARNL su Morgul

Elaborato di esame di:

Andrea Bonisoli, Giovanni Ducci, Marco Molinari, Cristian Paganessi, Diego Rossini.

Consegnato il: **22 Luglio 2011**

Sommario

Il lavoro svolto ha avuto come scopo il testing e l'implementazione della libreria ARIA, in grado di supportare il protocollo SCIP 2.0 per la comunicazione con il laser range scanner UHG-08LX della Hokuyo, installato a bordo del robot Morgul, un Pioneer 3-AT della Adept MobileRobots. Dopo aver caratterizzato il laser e individuato la sua posizione a bordo del robot, sono state documentate le modifiche necessarie da apportare ai programmi esistenti al fine di renderli adeguati al loro utilizzo. Al termine del progetto il robot Morgul è in grado di muoversi nell'ambiente del laboratorio evitando gli ostacoli rilevati dal sensore laser.

1. Introduzione

Il presente elaborato tratta l'attività di progetto svolta nell'ambito del corso di Robotica Mobile presso il Laboratorio di Robotica Avanzata (*ARL*) della Facoltà di Ingegneria.

L'intera attività è stata svolta operando su Morgul, un *Pioneer* 3-AT prodotto dalla Adept MobileRobots, facente parte della scuderia di robot mobili del laboratorio.

Il problema affrontato è stato quello di testare il funzionamento del laser range scanner UHG-08LX, installato a bordo del robot Morgul, utilizzando i pacchetti software ARIA e ARNL al fine di ottenere una corretta gestione delle misure effettuate dal sensore.

Lo svolgimento del progetto comporta il raggiungimento di due principali obiettivi:

- verificare il corretto funzionamento del sensore laser e della sua gestione tramite la libreria ARIA;
- testare alcuni programmi di prova per gestire il comportamento del robot utilizzando le misure rilevate dal laser range scanner.

L'obiettivo consiste quindi nel descrivere le operazioni da eseguire al fine di ottenere un corretto funzionamento del laser UHG-08LX, attraverso l'utilizzo dei metodi delle librerie ARIA e ARNL.

2. Componenti necessari

2.1. Laser range scanner Hokuyo UHG-08LX

Il laser scanner della Hokuyo della serie UHG-08LX ha le seguenti caratteristiche:

- intervallo di rilevamento: 30÷11000 mm (Accuratezza non garantita oltre gli 8000 mm);
- accuratezza: ±30 mm nel range [100÷1000] mm, 3% della misura nel range [1000÷8000] mm;
- angolo di rilevamento: 270°;
- angolo di risoluzione: 0.36°;
- tempo di scansione: 67 msec/scansione;
- protocollo: SCIP 2.0;
- interfaccia: USB 2.0;
- alimentazione: 12 V DC tramite alimentatore.

Per poter far funzionare il sensore laser è necessario installare il driver USB/seriale CDC ACM, che permette la connessione con il laser scanner e le librerie urg version 0.8.12, che consentono di interagire con il dispositivo laser. Inoltre è necessario installare le librerie Boost e SDL, che sono delle dipendenze, per potere installare le librerie urg.

L'interfacciamento con il laser non è stato eseguito direttamente con le librerie sopra citate, ma con la libreria ARIA. Essa però non è in grado di comunicare con il protocollo SCIP 2.0, ma solamente con la versione 1.1. È stato quindi necessario renderla compatibile, come illustrato nel capitolo 4.

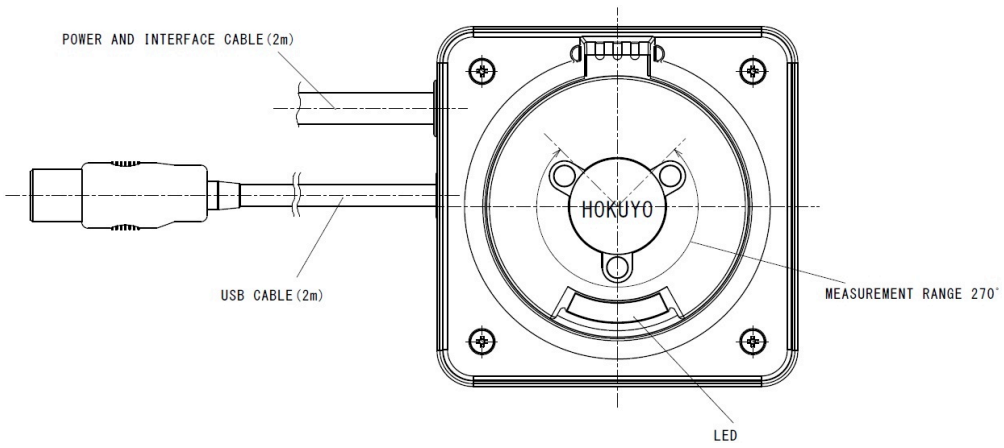


Figura 1: Vista dall'alto del laser range scanner UHG-08LX.

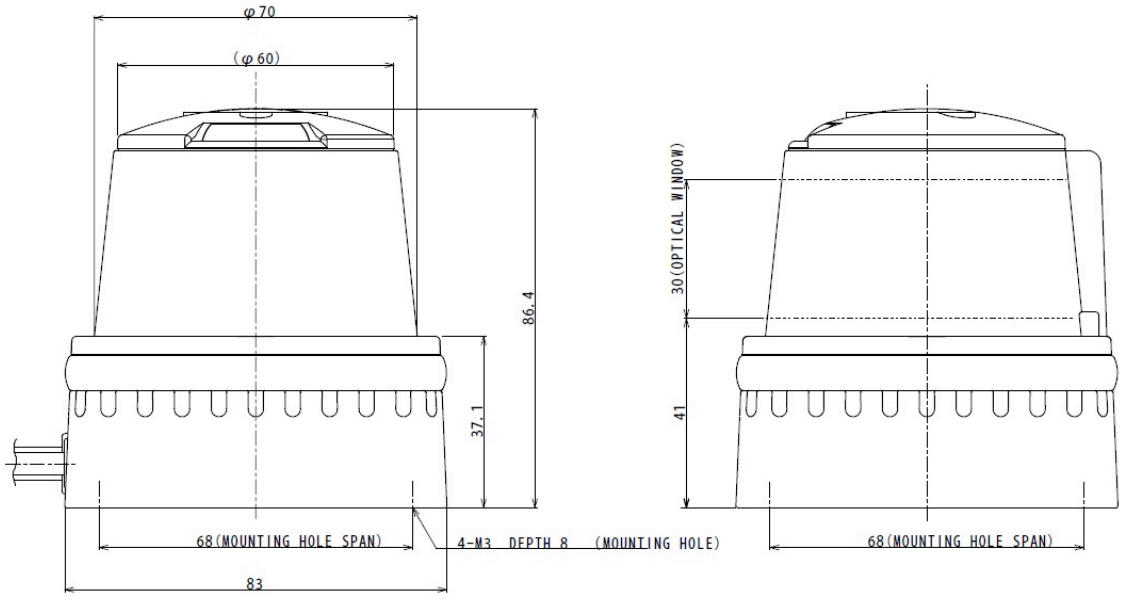
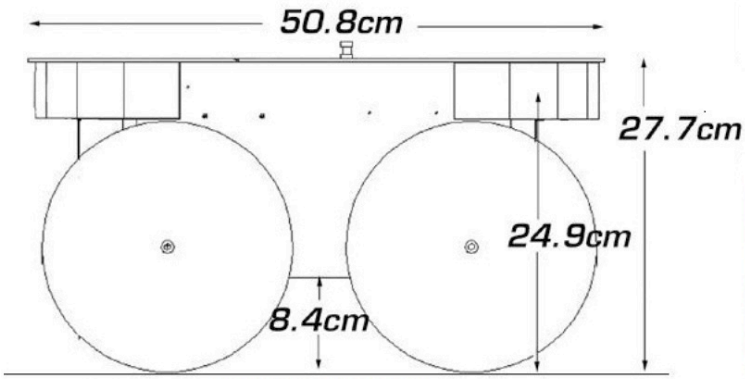


Figura 2: Vista laterale del laser range scanner UHG-08LX**.**

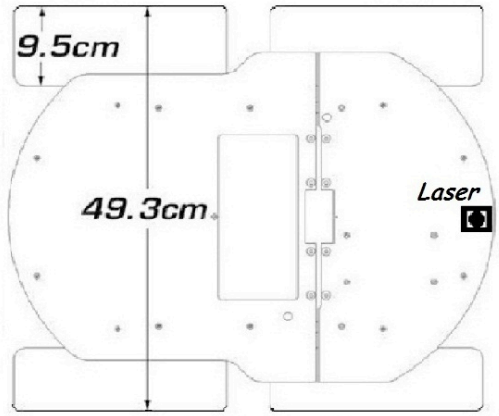
2.2. Robot Pioneer 3-AT

Le principali caratteristiche del robot Morgul sono:

- movimento: 4 ruote skid-steer comandate da 4 motori DC ognuno con encoder;
- sonar anteriori e posteriori;
- bumper anteriori e posteriori per rilevare eventuali collisioni.



Vista laterale del robot



Vista dall'alto del robot

Posizione del laser sul robot Morgul

3. Attività svolte

3.1. Documentazione utilizzata

Il paradigma di programmazione utilizzato per la realizzazione di tutto il codice è orientato agli oggetti. Il linguaggio ad alto livello è il C++; l'ambiente di lavoro standard definito per i robot *Pioneer 3*-AT è ARIA. Essa è una libreria di classi già implementate su una distribuzione *Debian* di Linux installata su un calcolatore che risiede sul robot Morgul.

La figura 3 mostra chiaramente il concetto di ereditarietà delle classi in cui ogni classe sottostante eredita i metodi della classe gerarchicamente superiore. In particolare la classe *ArRangeDevice* contiene metodi relativi al robot che possono essere tranquillamente usati da tutte le classi sottostanti. Va da sé che questo modo di programmare è intuitivamente quello più adatto per applicazioni di questo tipo. I dispositivi fisici

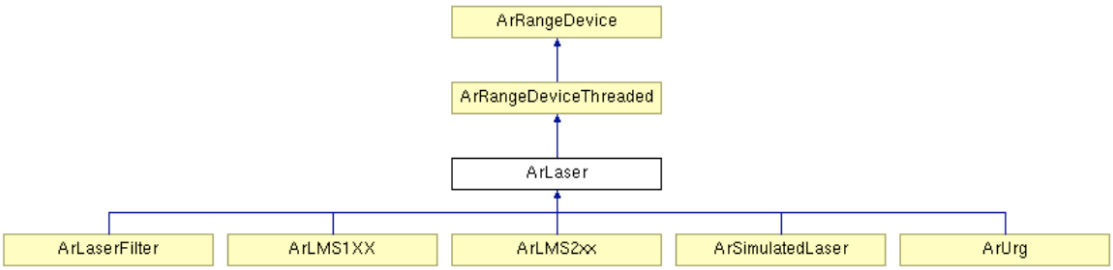


Figura 3: Diagramma delle classi per ArLaser.

sono visti sì come oggetti fisici ma anche e soprattutto come oggetti software, fatto questo che rende semplice e accessibile tutto il codice prodotto. Le classi di maggior rilevanza nell'ambito del lavoro affrontato sono ovviamente la *ArLaser* e la *ArUrg*. Queste classi sono già state modificate durante elaborati precedenti [1], in modo da ottimizzare la comunicazione tra robot e sensore laser, tramite protocollo SCIP 2.0. In seguito verranno trattati nel dettaglio sia il codice prodotto che le modifiche apportate ai file già esistenti.

3.2. Misure utili

La prima cosa da fare è definire le "*misure utili*" relativamente al problema affrontato. Come già detto il sensore laser non è in grado di rilevare la scansione su tutto l'angolo giro ma solo su un angolo di 270°. Quest'area di lavoro va ulteriormente diminuita a causa della struttura montata sul robot che ostacola il laser, rendendo prive di significato le distanze associate ad alcuni angoli della scansione. In figura 4 vengono meglio indicate le posizioni degli elementi di disturbo; ciò ha permesso di determinare gli angoli di rilevamento utili durante il movimento del robot.

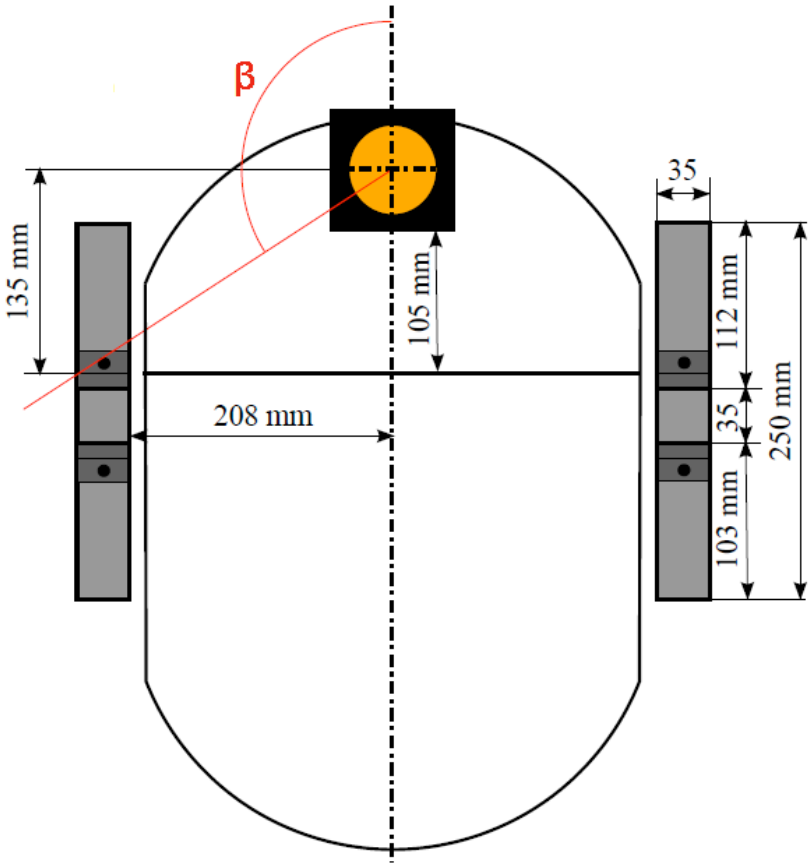


Figura 4: Misure degli elementi di disturbo.

Dalle misure riportate in figura 4 si è quindi determinato l'angolo massimo utile alla scansione:

$$\beta = arctg\left(\frac{135}{208}\right) + 90° = 123°$$

Si deduce dunque che l'intervallo di angoli utili è [-123°÷123°]. Grazie ad alcune prove di misura fatte in ambiente aperto si è poi trovato l'effettivo intervallo angolare, ovvero [-124°÷126°].

3.2.1. Note sul posizionamento del laser range scanner

Al momento dell'installazione fisica del sensore laser su Morgul, deve essere opportunamente modificato il file di configurazione in modo da evitare malfunzionamenti nel rilevamento delle misure effettivamente utili per il movimento del robot.

- **Se non si dovesse specificare tale posizione si ha che le misure reali nell'intervallo di misura, sarebbero "falsate" da un offset che le renderebbe errate.**

Il parametro di *default*, se non diversamente specificato, coincide con l'asse passante per il baricentro fisico del generico robot *Pioneer* 3-AT; il centro del laser invece è stato assunto come il centro del cilindro su base quadrata che contiene lo specchio rotante il quale permette di effettuare la scansione circolare. Ovviamente quest'ultima misura può solamente essere indicativa vista l'impossibilità di centrare perfettamente il riferimento del fascio laser all'interno del dispositivo.

3.3. Comportamento del sensore laser in presenza di un ostacolo

Al fine di chiarire meglio il funzionamento del laser è stato sviluppato il programma *MisureLaser.cpp* che stampa a video le misure "*rawreadings*" rilevate. Esso è pensato per eseguire misure con il laser e scartare quelle che presentano una distanza superiore ad un valore impostabile. Si riporta in figura 5 uno *screenshot* del risultato dell'esecuzione del programma suddetto; da notare la quantità di informazioni utili che si possono dedurre da questa schermata oltre alle misure rilevate. Per esempio è possibile sapere:

- modello del robot con relativo file di configurazione p3at.p;
- invocazione della classe *ArLaserConnector* per connettere via hardware l'oggetto laser all'oggetto robot, che sono oggetti software;
- *Starting Step* e *Ending Step* oltre a *Start deg* e *End deg* che rappresentano rispettivamente lo *step* di inizio e fine e l'angolo di inizio e fine della rilevazione delle misure;
- informazioni su produttore, venditore, *firmware* e *serial number* del laser;
- protocollo di comunicazione tra il laser e la libreria ARIA: SCIP 2.0;
- nome e numero dei sensori laser presenti sul robot;
- dimensione e indice delle letture riportate.

In questa prova si è impostato come intervallo di rilevamento la distanza di 1 metro e si è sfruttato il range massimo possibile dell'angolo di misura, ovvero [-124°÷126°]. È stato posizionato un ostacolo costituito da un palo verticale di legno del diametro di 5 cm ad una distanza di 50 cm dal laser.

Ogni record di misura è così composto:

- *indice all'interno del vettore delle misure;*
- *distanza dalla superficie o dall'oggetto rilevato in mm*;
- *angolo corrispondente.*

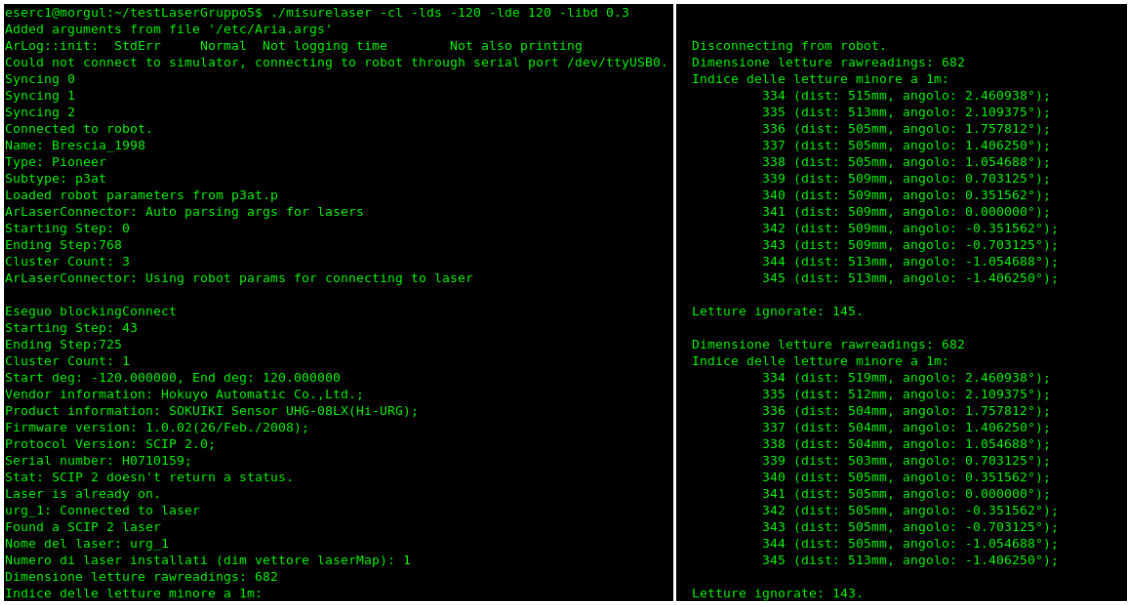


Figura 5: *Screenshot* del risultato dell'esecuzione del programma *MisureLaser.cpp.*

Si può notare che le misure non indicano esattamente la distanza a cui è stato posizionato l'ostacolo, ma presentano un errore massimo di 19 mm rispetto al valore vero. Questo risultato è compatibile con le specifiche fornite dal *datasheet*.

La figura 6 riporta invece uno *screenshot* della schermata di prova del programma *ProvaLaser.cpp*. In questo caso le misure riportate sono solo alcune tra quelle effettuate e potrebbero bastare per il corretto movimento del robot. Ad esempio queste misure sarebbero in grado di sostituire l'informazione fornita dai sonar.

In questa dimostrazione vengono visualizzate le distanze lette in corrispondenza di sette particolari angoli, corrispondenti ai gradi nell'intervallo [-90°÷90°] con uno step di 30°. A questo punto non rimane che effettuare i test di ripetibilità di misura, per verificare l'accuratezza e più in generale l'efficienza del dispositivo montato su questo tipo di robot.

Figura 6: Misure utili rilevate dal laser.

Non si deve perdere di vista l'obiettivo finale di tutto il lavoro: Morgul dovrà potersi muovere con il laser unitamente agli altri sistemi complementari di navigazione, come i sonar. Nelle prossime sezioni e nei prossimi capitoli verranno inoltre affrontati in maniera più dettagliata tutti questi aspetti insieme a tutto il codice utilizzato.

3.3.1. Test sulla ripetibilità del laser

Il laser effettua una scansione su 270° ruotando a velocità costante in senso antiorario. Esso fornisce una serie di coppie di valori, ognuna delle quali contiene la distanza dallo strumento e l'angolo associato a tale distanza. Più precisamente il laser è in grado di fornire al massimo 1024 misure sull'angolo giro; dato che il suo raggio d'azione è di 270°, il massimo numero di scansioni è:

$$\frac{1024 \cdot 270°}{360°} = 768 \text{ coppie distanza-angolo}$$

Un'eventuale incertezza sul valore delle misure di distanza ed angolo associato porta ad un possibile errore nel calcolo della posizione degli ostacoli. Questo si ripercuote in un'errata movimentazione dei motori che a sua volta porta ad un probabile impatto non voluto. Si possono avere:

- *Distanze non precise*: errore già descritto nel *datasheet* del laser, in cui è indicata l'incertezza di misura (0.1 %FS). Quest'incertezza non dà però problemi significativi per le applicazioni in esame;
- *Angoli errati*: sul *datasheet* del dispositivo non è precisata questa voce. Per angoli errati si intende una possibile incertezza sul valore della posizione angolare a seguito di scansioni ripetute. Ciò significa che un oggetto fisso può essere rilevato in una posizione angolare differente da quella indicata dalle scansioni precedenti. Questo problema potrebbe derivare dalla velocità di rotazione non costante dello specchio.

Il test sulla ripetibilità di misura è stato effettuato al fine di quantificare l'eventuale errore sulla misura angolare dopo una serie di scansioni ripetute consecutivamente. Il principio seguito durante il test è molto semplice. A partire dall'indice del vettore delle misure, si può calcolare l'angolo con:

$$\alpha_n = \frac{(n-1) \cdot 270°}{N}$$

dove N è il numero totale di misure effettuate. La formula deriva dal fatto che il vettore contiene gli N valori nel seguente modo:

INDICE	DISTANZA	ANGOLO
0	l_1	α_1
1	l_2	α_2
...	...	...
767	l_{768}	α_{768}

Tabella 1: Per come sono stati settati i parametri nei programmi utilizzati, N=768.

Ne risulta dunque che ad ogni indice del vettore corrispondono rispettivamente una distanza ed un angolo. Se il laser fosse posizionato in un ambiente privo di ostacoli e l'unico oggetto fosse un palo in una posizione arbitraria si otterrebbero:

- N-*m* misure di distanza pari a 5000 mm; valore di lunghezza tornato dal laser nel caso in cui non incontra ostacoli;
- *m* distanze pari a d_2, ovvero la distanza tra il laser e l'oggetto in questione.

Dove *m* sono le misure relative alle scansioni che incontrano l'oggetto. La figura 7 illustra quanto detto.

Figura 7:

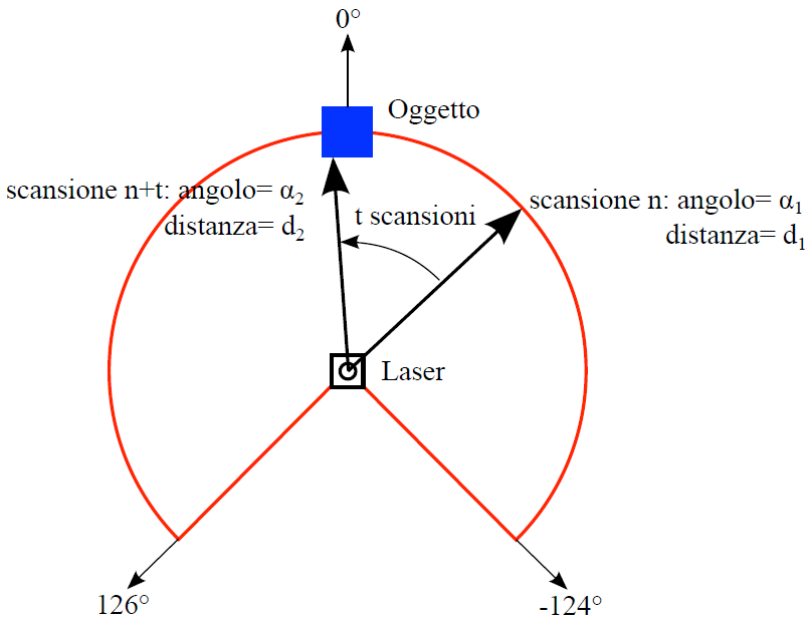
Test di ripetibilità:

d_1=5000 mm

$\alpha_1 \epsilon$[-124°÷0°]

d_2=distanza oggetto

$\alpha_2 \epsilon$[0°÷126°]



Le misure diverse da 5000 mm dipendono dalla posizione reciproca palo-laser e dalle dimensioni del palo stesso. È irrilevante per il test calcolare a priori quanti *m* valori si avranno. Per verificare l'assenza di errori angolari ci si basa sul fatto che ad ogni elemento del vettore corrisponde un indice. Basta dunque controllare che dopo un certo numero di scansioni gli indici corrispondenti a distanze diverse da 5000 mm siano sempre gli stessi. Se ciò accade, dato che il vettore è riempito ogni volta da *N* valori, significa che il laser rileva l'oggetto sempre nella stessa posizione, altrimenti si avrà un errore angolare pari a:

$$errore\ angolare = \left| indice\_scansione(t_0) - indice\_scansione(t_1) \right| \cdot \left(\frac{270°}{N} \right) \qquad (1)$$

Calcolando l'eventuale differenza tra gli indici di due scansioni, fatte in istanti diversi ma relative alla stessa misura, l'errore commesso sarà proporzionale alla distanza tra i due indici. Per comprendere meglio il test effettuato si può vedere un esempio:

	SCANSIONE_0	SCANSIONE_1	SCANSIONE_2	SCANSIONE_3	SCANSIONE_4
INDICE	234	234	237	237	238
DISTANZA [mm]	1456	1456	1456	1456	1456
ANGOLO [°]	35	35	35	35	35

Nelle prime due scansioni la scrittura avviene nella stessa locazione, quella con indice 234, quindi l'errore è pari a 0°; tra la seconda e la terza invece si hanno indici diversi di una stessa misura. Applicando l'equazione (1) può essere calcolato l'errore angolare commesso, di valore:

$$errore\ angolare = |234 - 237| \cdot \left(\frac{270°}{768} \right) = 1.055°$$

Nell'esempio si è supposto che l'oggetto da osservare fosse molto sottile in modo da valutare una sola misura utile con il suo relativo errore. In realtà le misure utili sono più di una.

Se un errore di 1.055° può sembrare irrilevante in realtà si deve tener conto che il laser effettua una scansione ogni 67 ms. Volendo effettuare più scansioni, dato che l'errore commesso sul valore dell'angolo rilevato è cumulativo, il valore angolare finale è affetto da un errore che comincia ad essere consistente. Supponendo ad esempio di fare venti scansioni, ognuna con lo stesso errore descritto nella tabella precedente, applicando l'equazione (1) si avrà un errore finale pari a:

$$errore = 20 \cdot \left(\frac{270°}{768} \right) = 7.03°$$

Questo errore può rappresentare una notevole complicazione ai fini del movimento del robot visto che 7° rappresentano un errore del 2.6% circa sull'area di misura.

Nella figura 8 vengono riportati alcuni risultati del test effettuato. In particolare sono stati eliminati, direttamente dal software utilizzato, i valori che non indicavano la distanza laser-palo (ovvero 5000 mm). Come si può notare ogni valore angolare rilevato è associato allo stesso indice tra una scansione e la successiva. Dunque l'errore commesso è pari a zero, si può quindi affermare che il sensore laser non commette errori nella determinazione dell'angolo di misura.

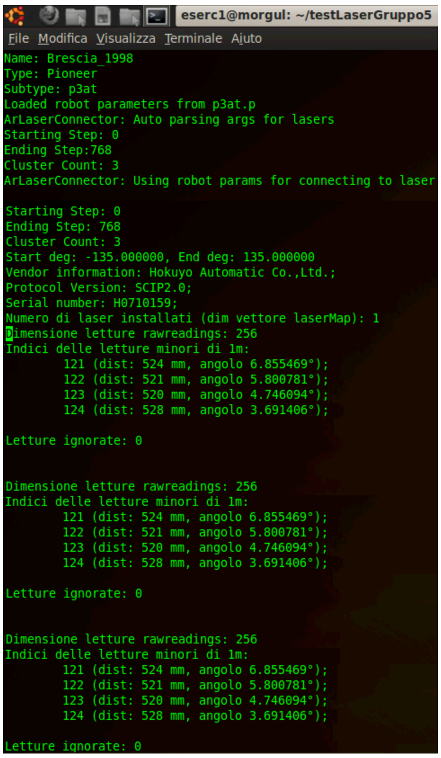


Figura 8: Risultati sperimentali ottenuti al termine del test.

3.4. Test del programma *Wander*

Il programma *Wander.cpp* è stato concepito per permettere al robot di muoversi in maniera casuale in un ambiente sconosciuto senza che esso urti contro alcun ostacolo. Il robot sfrutta l'informazione proveniente dai sonar e, nei casi estremi, dai bumper. Uno degli obiettivi finali di questo elaborato è quello di poter utilizzare il software *Wander.cpp* unicamente con le informazioni provenienti dal laser. Si é reso dunque necessario disattivare tutti i sonar installati prima di far funzionare il programma.

Non essendoci nessun comando diretto per escludere il funzionamento dei sonar sul robot *Morgul,* la soluzione risiede ancora una volta nel paradigma della programmazione ad oggetti. Si sono commentate semplicemente le righe di codice in cui viene definito e aggiunto al robot l'oggetto *sonar*. Dopo una necessaria ricompilazione del codice si è notato che i dispositivi corrispondenti continuano ad emettere onde sonore. Ciò è dovuto al fatto che, pur non utilizzando l'oggetto *sonar*, l'alimentazione dei trasduttori è comunque attiva. Per avere la certezza assoluta che il calcolatore non utilizzasse le informazioni provenienti dai sonar è stata effettuata una prova sperimentale.

In figura 9 si può vedere che i sonar sono stati coperti con delle barre opportunamente sagomate di *teflon* spesse circa 1 cm. Il *teflon* funziona esattamente come uno schermo e riflette le onde sonore emesse dai sonar in modo da simulare un ostacolo di fronte ad ogni trasduttore. Nei due casi possibili di funzionamento il comportamento del robot dovrebbe essere:

- *sonar attivi*: sono stati attivati solo i sonar e, per sicurezza, i bumper. In queste condizioni, eseguendo wander, il robot dovrebbe girare su se stesso dal momento che ogni *sonar* rileva un ostacolo;
- *sonar disattivati*: i sonar vengono disattivati ma restano utilizzabili le informazioni provenienti dai bumper. Eseguendo nuovamente il programma il robot dovrebbe muoversi in modo corretto sfruttando solo le informazioni provenienti dai bumper.

Durante la prova il robot ha eseguito esattamente i comportamenti aspettati. Si può concludere quindi che il codice sviluppato permette una semplice ma efficace gestione dei vari oggetti fisici presenti su Morgul che corrispondono agli oggetti software definiti nel codice prodotto.

A questo punto sono state sostituite le informazioni provenienti dai sonar con quelle ottenute dal laser

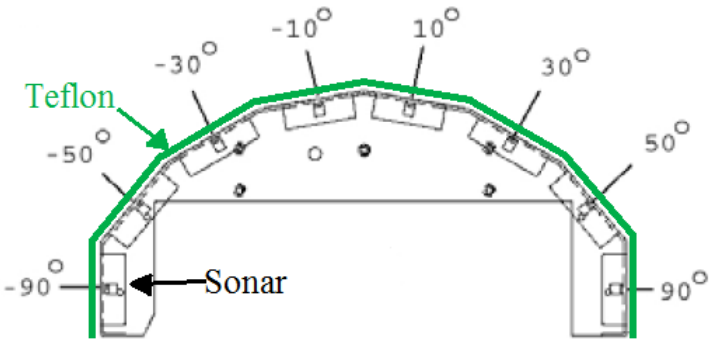


Figura 9: Vista dall'alto della parte di Morgul su cui sono montati i sonar.

range scanner. Per far ciò è stata utilizzata la classe *ArActionAvoidFrontLaser* preparata da [1]: questa implementa una *action* che si occupa di rilevare gli ostacoli presenti nell'ambiente tramite il laser e fa ruotare il robot al fine di evitarli. Tra i parametri che questa classe accetta vi è la minima distanza a cui rilevare gli ostacoli, l'angolo entro cui considerarli, la velocità di rotazione e i gradi di rotazione da effettuare.

Il programma quindi ottenuto è stato utilizzato per far muovere il robot all'interno del laboratorio. Dalle prove effettuate si è notato che il robot è in grado di muoversi evitando gli ostacoli rilevati dal sensore, mentre naturalmente sussistono delle difficoltà con le strutture non rilevabili perché poste ad una altezza inferiore o superiore al sensore, come ad esempio oggetti sul pavimento o mensole e bordi dei tavoli.

3.4.1. Ulteriori prove

Poiché la *action* sfruttata nel programma *Wander* richiede molti parametri tra loro correlati, sono state effettuate ulteriori prove sperimentali al fine di stimarne alcuni. Sono di particolare interesse i valori minimi di distanza e angolo della rilevazione del laser e dell'angolo di sterzata del robot.

A tale scopo è stata ricreata una situazione ritenuta difficoltosa: è stato posto un ostacolo sottile (di spessore 2 cm) esattamente di fronte al robot, mentre è stata settata la velocità massima di spostamento (1.2 m/sec). La figura 10 rappresenta meglio la situazione descritta.

Considerando la velocità di scansione del laser e quella di spostamento del robot, si può stimare la

Figura 10: Posizione dell'ostacolo di prova.

distanza minima a cui rilevare l'ostacolo:

$$d_{\min} = 1200 \cdot 67 \cdot 10^{-3} = 80.4 \ \ mm$$

Questa misura non è quella minima realmente utilizzabile in quanto è necessario tenere in considerazione alcuni fattori, il principale è lo spazio di frenata del robot, che non è nullo.

Sperimentalmente è stato misurato, in dieci prove, lo spazio di frenata procedendo come segue:

- sono stati modificati i parametri della *action* del programma *Wander* al fine di fermare il robot, qualora il sensore avesse rilevato un ostacolo ad una distanza inferiore o uguale a 900mm (questa distanza è stata scelta perché sperimentalmente si è valutato che garantiva un buon margine di sicurezza);
- si è fatto avanzare il robot in linea retta verso un ostacolo non rigido, per evitare danni in caso di collisione, e per una distanza sufficiente a fargli raggiungere la velocità massima;
- dunque, una volta fermato, è stata misurata l'effettiva distanza tra l'ostacolo e il laser.

La tabella 2 mostra i risultati ottenuti di cui successivamente è stata calcolata la media aritmetica al fine di stimare l'angolo minimo di sterzata del robot.

Il valore della media è pari a 596.5 mm, c olo di sterzata pari a:

prova	Distanza rilevata dall'ostacolo
1	570 mm
2	610 mm
3	595 mm
4	615 mm
5	585 mm
6	640 mm
7	600 mm
8	550 mm

Tabella 2: Misure sperimentali spazio di frenata.

$$\alpha = \arctan\left(\frac{260mm}{596.5mm}\right) = 23.6°$$

dove 260 mm rappresenta la distanza tra l'asse del robot ed il suo punto più esterno.

A seguito di prove successive, l'angolo minimo di sterzata che si può impostare, ottenendo un buon comportamento del robot quando esegue il programma *Wander,* risulta essere 30°.

4. Modifiche effettuate

4.1. Modifiche alla libreria ARIA

In seguito al lavoro svolto da Agarossi & al. [1] sono state apportate delle modifiche alla libreria ARIA per renderla compatibile con il protocollo SCIP 2.0 supportato dal laser range scanner UHG-08LX. Queste hanno coinvolto principalmente la classe *ArUrg*, ma anche la classe *ArRobot*. Nello specifico è stato aggiunto un metodo (ed il relativo prototipo) alla classe *ArRobot*, mentre è stata modificata l'implementazione dei metodi della classe *ArUrg* per renderli compatibili con entrambe le versioni dei protocolli. Pertanto, rispetto alla libreria non modificata di ARIA o in caso di aggiornamenti, è necessario che i seguenti file vengano sovrascritti con quelli presenti sul cd allegato alla relazione:

- *ArRobot.cpp*
- *ArRobot.h*
- *ArUrg.cpp*

Nel calcolatore presente a bordo di Morgul, queste classi si trovano nella directory:

/usr/local/Aria/src (per i file .cpp)

/usr/local/Aria/include (per i file .h)

➢ **Ovviamente è necessario ricompilare la libreria per rendere effettive le modifiche.**

Queste modifiche sul calcolatore montato a bordo del robot erano già state effettuate in precedenza rispetto all'inizio del lavoro e quindi non è stato necessario sostituirli. Se invece si dovesse installare nuovamente la libreria ARIA sarebbe necessario sovrascrivere i file indicati. Si noti tuttavia, che in caso di aggiornamenti della libreria, queste modifiche potrebbero essere incompatibili o non correttamente funzionanti.

I problemi potrebbero derivare dal fatto che una nuova versione di ARIA potrebbe presentare ulteriori modifiche su questi stessi file. La loro sovrascrittura porterebbe ad avere file di versioni precedenti, ma modificati, assieme a nuove versioni degli stessi file. Di conseguenza sovrascrivere semplicemente questi file non è una modifica funzionante; si rende quindi necessario operare manualmente un *merge* tra le diverse modifiche effettuate da Agarossi & al. [1] e quelle della nuova versione della libreria ARIA.

Nello specifico è quindi necessario aggiungere oppure integrare le modifiche nella nuova versione della libreria aggiungendo alla classe *ArRobot* il metodo *checkRangeDevicesLaserCurrentPolar* (sia il prototipo nel file .h che l'implementazione nel file .cpp). La classe *ArUrg* invece è stata pesantemente modificata e le implementazioni di quasi tutti i metodi sono state alterate per poter supportare entrambe le versioni dei protocolli SCIP e non è quindi possibile conoscere a priori come operare. Si suggerisce quindi di confrontare singolarmente ogni metodo al fine di valutare l'entità delle differenze e delle modifiche necessarie. Inoltre anche le nuove versioni delle classi *ArLaser* e *ArLaserConnector* potrebbero avere conseguenze sulle funzionalità e andrebbero quindi verificate.

Oltre a tutto ciò è stato anche modificato il file contenente i parametri del robot. Questa modifica, pur non essendo strettamente indispensabile, consente di ridurre la quantità di parametri che è necessario specificare da linea di comando, rendendo quindi più agevole l'esecuzione dei programmi. Si veda il

paragrafo 4.3 per i dettagli. Nello specifico la modifica ha coinvolto i parametri relativi al primo sensore laser dei dieci che è possibile specificare, impostando il tipo e la porta corretti per il laser montato. Pertanto, se viene sovrascritto anche il seguente file:

- *p3at.p*

non è più richiesto impostare questi parametri da linea di comando al momento dell'esecuzione dei programmi. Nel calcolatore presente a bordo di Morgul, questo file si trova nella directory:

/usr/local/Aria/params

In dettaglio le modifiche apportate hanno coinvolto le seguenti righe così sostituite:

```
Section Laser parameters
;SectionFlags for Laser parameters:
; -- start of changed parameters for Morgul robot
LaserType urg            ; type of laser
LaserPortType serial     ; type of port the laser is on
LaserPort /dev/ttyACM0   ; port the laser is on
LaserAutoConnect true    ; if the laser connector should autoconnect this laser or not
; -- end of changed parameters for Morgul robot
```

Va tuttavia notato che, mentre le modifiche alla libreria sono generali e possono essere utilizzate anche con altri robot o altri modelli, queste modifiche al file sono specifiche per il robot Morgul e non possono essere trasferite su altre macchine.

4.2. Funzioni disponibili

Dopo aver apportato le modifiche illustrate alla libreria, è possibile utilizzare alcune funzioni all'interno dei programmi per ottenere i dati forniti dal sensore laser. Come prima cosa deve essere creata una istanza della classe *ArLaserConnector*, il cui costruttore richiede come parametri obbligatori i puntatori alle istanze di *ArArgumentParser*, *ArRobot* e *ArRobotConnector* su cui si vuole operare. Questa istanza si occuperà di ricevere tutti i parametri forniti sia da linea di comando che tramite il file di configurazione e predisporrà i collegamenti utili al funzionamento del laser range scanner. Per connettere effettivamente il sensore laser è necessario utilizzare il metodo *connectLasers* della classe *ArLaserConnector* che fornisce come valore di ritorno un booleano: un valore di return "false" è conseguenza del fatto che si è verificato un errore ed è consigliabile interrompere il programma. Se poi fosse richiesto di accedere direttamente alle misure fornite dal laser, si deve richiedere l'istanza della classe *ArLaser* al robot tramite il metodo *findLaser*. Poiché il robot supporta fino ad un massimo di dieci sensori ed a priori non è noto quale numero venga assegnato, potrebbe essere necessario cercare, tramite un ciclo for, l'istanza corretta. In tal caso si può utilizzare come esempio il seguente codice:

```
ArLaser *laser;
for (int i = 1; i <= 10; i++) {
        if (robot.findLaser(i) != NULL) {
                laser=robot.findLaser(i);
                break; }
}
```

La classe *ArLaser* quindi mette a disposizione alcuni metodi: i più utili sono *isConnected*, che verifica se il sensore è ancora connesso al robot, *lockDevice* e *unlockDevice* che settano e tolgono rispettivamente un lock sul sensore per evitare accessi concorrenti, ed infine il metodo *getRawReadings*. Quest'ultimo consente di accedere ad una lista di oggetti di tipo *ArSensorReading* i cui metodi *getRange*

e *getSensorTh* permettono rispettivamente di ottenere la distanza rilevata in millimetri e l'angolo di tale misura in gradi. Per ulteriori dettagli si rimanda alla documentazione delle classi di ARIA [3].

Allegato a questa relazione (si veda l'appendice A) c'è un template che contiene tutti i principali comandi qui illustrati e può essere utilizzato come base di partenza per ogni programma che debba utilizzare il sensore. In alternativa si può utilizzare la classe *ArActionAvoidFrontLaser* come esempio per lo sviluppo di *action* che richiedono i dati forniti dal range scanner.

4.3. Parametri da linea di comando

La classe *ArLaserConnector* comprende alcuni parametri che possono essere impostati al momento dell'esecuzione da linea di comando. L'elenco completo dei parametri si può trovare nella documentazione di ARIA, qui di seguito sono illustrati i parametri utilizzati nel corso di questo progetto con il loro significato: alcuni di questi sono necessari solo se non è stato modificato il file dei parametri, alcuni sono invece indispensabili, mentre alcuni servono solamente a specificare particolari comportamenti.

4.3.1. Parametri compresi nel file di configurazione

- *laserType (o -lt)*: questo parametro deve essere impostato con l'opzione "urg" e, non con l'opzione "uhg" come si potrebbe pensare, poiché serve per distinguere la famiglia di laser e la classe da utilizzare per la comunicazione.
- *laserPortType (o -lpt)*: questo parametro deve essere impostato come "serial" nel caso in cui il laser sia direttamente connesso al robot.
- *laserPort (o -lp)*: questo parametro specifica la porta a cui è connesso il sensore; nel caso del progetto svolto era necessario utilizzare l'opzione "/dev/ttyACM0".

4.3.2. Parametri necessari

- *connectLaser (o -cl)*: questo parametro richiede la connessione al laser e non necessita di opzioni.

4.3.3. Parametri opzionali

- *laserDegreesStart (o -lds)*: questo parametro può essere specificato per modificare l'angolo iniziale di lettura del sensore.
- *laserDegreesEnd (o -lde)*: questo parametro invece può essere specificato per modificare l'angolo finale di lettura del sensore.
- *laserIncrementByDegrees (o -libd)*: questo parametro può essere specificato per modificare l'intervallo in gradi tra una misura e la consecutiva.

5. Conclusioni e sviluppi futuri

Grazie al lavoro svolto è ora possibile sfruttare le funzionalità della libreria ARIA al fine di utilizzare il laser range scanner Hokuyo UHG-08LX, montato su Morgul. I test svolti mostrano come sia possibile interfacciarsi con il laser al fine di utilizzarne le misure, anche per la realizzazione di action che debbano sfruttarle. Ne è un esempio lo scopo di questo lavoro: eseguire il programma *wander.cpp* utilizzando le sole informazioni fornite dal trasduttore stesso.

Questi test possono quindi essere considerati una guida per futuri sviluppi e per altri progetti che debbano interagire con tale sensore. Infatti il laser può essere sfruttato per costruire piante di ambienti sconosciuti. L'unico difetto è che la scansione avviene su un solo piano di lavoro, dunque si ottengono solo mappe in 2D; per una scansione 3D è necessario far ruotare il laser. Grazie ad una trattazione trigonometrica delle misure ottenute si può ricavare una pianta 3D.

Bibliografia

[1] Agarossi A., Finardi S., Pachera M., Rigo M.: “Adattamento della Libreria ARIA al Protocollo SCIP 2.0 per il Laser Hokuyo UHG-08LX”, sito web (http://www.ing.unibs.it/~arl/docs/projects/Sen_07).

[2] Facchetti F., Fausti S., Marini L.: “Interfacciamento Laser Scanner Hokuyo UHG-08LX”, sito web (http://www.ing.unibs.it/~arl/docs/projects/Sen_06).

[3] ARIA Developer’s Reference Manual: http://www.ing.unibs.it/~arl/docs/documentation/Aria%20documentation/Current/Aria-Reference/

Appendice A: Template per l'interfacciamento con il laser

A.1 TemplateLaser.cpp

```
#include <string>
#include "templatelaser.h"

/* main function */
int main(int argc, char **argv)
{
// robot and devices
        ArRobot robot;

/* Inizializza ARIA con i parametric di default(destinazione e livello del log della libreria)*/
        Aria::init();
        ArArgumentParser parser(&argc, argv);
        parser.loadDefaultArguments();
        ArLog::init(ArLog::StdErr, ArLog::Normal);

// connector ArRobotConnector utilizzato al posto di ArSimpleConnector
   ArRobotConnector robotConnector(&parser, &robot);

if (!robotConnector.connectRobot())
  {
    // Error connecting:
    // if the user gave the -help argumentp, then just print out what happened,
    // and continue so options can be displayed later.
    if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
    {
      ArLog::log(ArLog::Terse, "Could not connect to robot, will not have parameter file so options displayed later may not include everything");
    }
    // otherwise abort
    else
    {
      ArLog::log(ArLog::Terse, "Error, could not connect to robot.");
      Aria::logOptions();
      Aria::exit(1);
        }
  }

robot.runAsync(true);
```

```
/*il laserConnectror provvede a tutte le operazioni necessarie per il laser riceve i parametri da linea di comando, si veda la documentazione di ARIA*/
ArLaserConnector laserConnector(&parser, &robot, &robotConnector,true,ArLog::Verbose);


  if (!laserConnector.connectLasers(false, false, true))
  {
    printf("Could not connect to lasers... exiting\n");
    Aria::exit(2);
  }


/*cerco il laser sul robot se non è necessario accedere direttamente al laser questa parte non è necessaria.*/
        ArLaser *laser;
        for (int i = 1; i <= 10; i++)
        {
           if(robot.findLaser(i) != NULL) {
                 laser=robot.findLaser(i);
                 break; }
        }



        Aria::shutdown();
        return 0;
}
```

A.2 TemplateLaser.h

```
#include "/usr/local/Aria/include/Aria.h"
```

Indice