



Università di Pisa

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ROBOTICA E
DELL'AUTOMAZIONE

PROGETTO DEL CORSO DI SISTEMI DI GUIDA E NAVIGAZIONE

SIMULAZIONE DEL SONAR PING 360

Modellazione del sensore e analisi dei dati

Studenti:

Sebastiano Bianco
Alberto Regoli

Professori:

Riccardo Costanzi
Lorenzo Pollini

Anno accademico 2023 - 2024
22 marzo 2024

Indice

Introduzione	1
1 Modello fisico del sensore	2
1.1 Datasheet Ping360	4
1.2 Modellazione Unity	5
2 Acquisizione dei dati e comunicazione con ambiente virtuale	7
2.1 Acquisizione dei dati	7
2.2 Comunicazione con ambiente virtuale	8
3 Elaborazione dati	9
3.1 Modellazione tramite Distribuzione Gaussiana	10
3.2 Generazione dell'immagine post-processing	11
4 Interfaccia Ping Viewer	15
4.1 Comunicazione tra MATLAB e Ping Viewer	15
4.2 Tipologie di Messaggi	16
4.2.1 Messaggio 2300 device_data	16
4.2.2 Messaggio di registrazione	19
4.2.3 Messaggio in tempo reale	20
5 Comunicazione UDP	22
5.1 Comunicazione Unity - Simulink	22
5.2 Comunicazione Simulink - Ping Viewer	23
5.2.1 Blocco <i>udp_1block</i>	24
6 Analisi dei dati	25
6.1 Validazione del modello	25
6.2 Insonificazione di materiali con diversa riflettività	28
6.3 Dettagli delle immagini acustiche	29
6.4 Limiti del modello	30
Conclusioni	32
Bibliografia	35

Introduzione

In questo progetto viene presentata la modellazione di un sonar a scansione meccanica per la navigazione, ponendo attenzione all'aspetto fisico, la comunicazione con il suo software per la rappresentazione dei dati e l'analisi di questi.

Il sensore in questione è il Ping360, prodotto della *BlueRobotics*, il quale consente di ottenere immagini a 360° utilizzando onde acustiche emesse da un trasduttore. Dalla riflessione di queste ultime è possibile ricavare la distanza relativa tra il ROV, su cui viene montato il rilevatore, e gli oggetti nel suo campo visivo.

Il progetto viene sviluppato su tre software:

- *Unity*, per la creazione del modello 3D del sensore e la scansione dell'ambiente simulativo;
- *MATLAB - SIMULINK*, per l'acquisizione e manipolazione dei dati;
- *Ping Viewer*, per osservare le informazioni acquisite.

L'obiettivo finale consiste nella creazione di immagini polari che permettono di mostrare la zona esplorata dal robot e di identificare gli oggetti incontrati. Le simulazioni possono essere svolte sia off-line, dopo avere raccolto i dati da Unity, sia in real-time utilizzando una comunicazione UDP tra tutti i software.

Capitolo 1

Modello fisico del sensore

Il Ping360 è un **sonar** per immagini a scansione meccanica. È progettato principalmente per essere utilizzato su *BlueROV2* per la navigazione in condizioni di scarsa visibilità dell'acqua, ma è adatto anche per applicazioni quali ispezione (sia durante il moto del veicolo sia fermo), raggiamento di ostacoli, localizzazione e tracciamento di bersagli.



Figura 1.1 – Ping360.

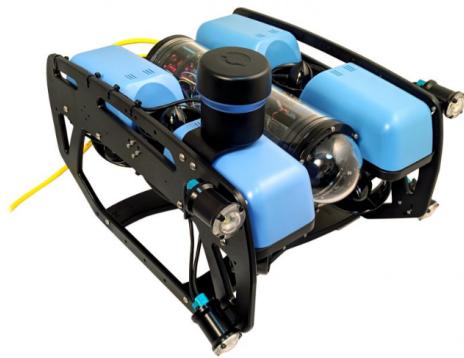


Figura 1.2 – Ping360 e BlueROV2.

Il **sonar** a scansione (SOund Navigation And Ranging) è un sensore attivo che funziona trasmettendo impulsi sonori nell'acqua e registrando gli echi restituiti quando vengono riflessi dagli oggetti di fronte ad esso [4]. Questo è definito attraverso un raggio acustico a forma di "ventaglio" caratterizzato da un ampio angolo verticale (*elevation*) e uno orizzontale stretto (*azimuth*) per insonificare una parte ben delineata dell'ambiente e mostrare successivamente su un visualizzatore.

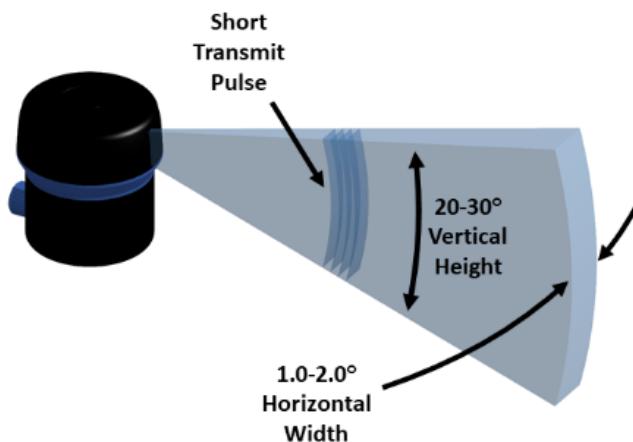


Figura 1.3 – Impulso sonoro generato dal sonar.

Combinando la velocità del suono nell'acqua (v_{sound}) con il tempo in cui sono stati ricevuti gli echi (t_{echo}), è possibile calcolare la distanza percorsa dal suono. L'equazione da utilizzare è

$$distance = v_{sound} \frac{t_{echo}}{2}, \quad (1.1)$$

tenendo conto che la velocità del suono nell'acqua salata è di circa 1500 m/s, ma può variare a seconda di temperatura, di salinità e di profondità.

A livello di funzionamento il sensore può essere paragonato ad una torcia che illumina solo in un'unica direzione, nascondendo tutto ciò che non entra nel suo campo visivo come in figura 1.4 Per ovviare questo problema, il trasduttore è montato su un motore passo-passo, che lo ruota con incrementi di un grado, permettendo l'osservazione dell'ambiente circostante.

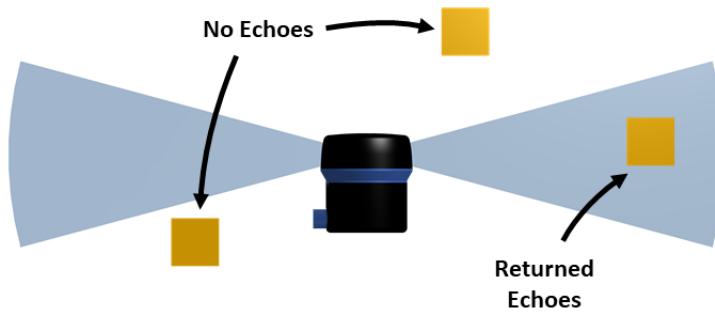


Figura 1.4 – Rappresentazione di zone insonificate e non.

L'output finale del sonar è quindi una *scansione polare* data dall'accostamento dei singoli fasci sonori; questo permette di capire cosa può esserci di lato o dietro un ROV. Inoltre è possibile scansionare un singolo settore per tenere traccia di un target durante lo spostamento.

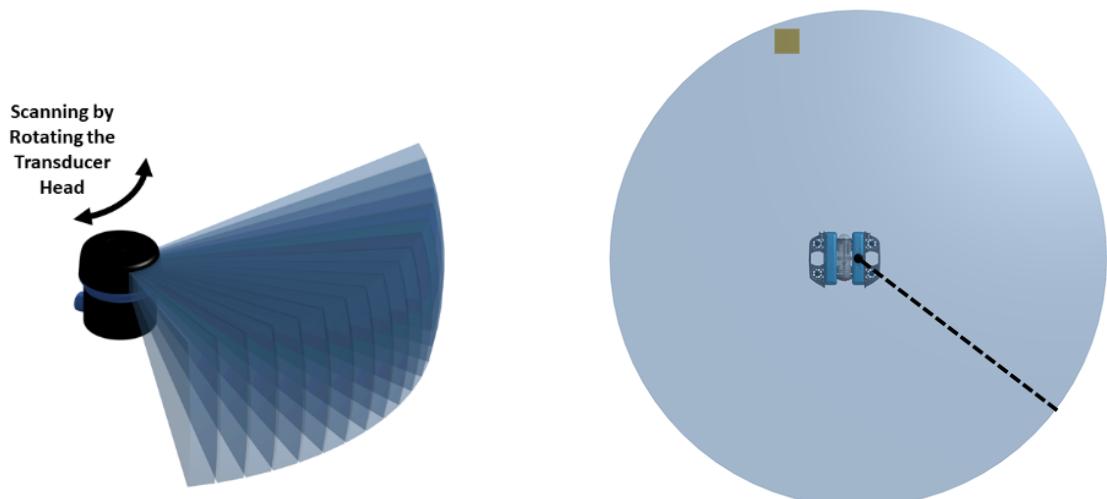


Figura 1.5 – Rotazione del fascio acustico.

Figura 1.6 – Esempio di scansione polare.

Capitolo 1 Modello fisico del sensore

I sonar a scansione non sono in grado di distinguere oggetti colpiti da raggi con lo stesso angolo di *azimuth*, poiché le immagini ottenute sono una proiezione dall'alto dell'ambiente scandagliato. La corretta posizione e l'orientazione del sensore rispetto al robot aiutano a distinguere e insonificare meglio i corpi posti sulla stessa verticale. Nel caso in esempio, il Ping360 viene fissato sopra il ROV con inclinazione nulla rispetto agli assi corpo del veicolo, come in figura 1.2.

Altro elemento da valutare per comprendere meglio geometria e dimensioni del fondale marino è l'ombra acustica, infatti il trasduttore invia energia sonora e colpisce solamente le facce dei corpi in vista generando zone di buio dietro ad essi, in analogia alla luce. La forma delle ombre dipende dalla profondità del robot e dall'inclinazione del raggio sonoro, ad esempio con angolo di discesa elevato è ridotta la zona di ombra, ma è minore la zona scandagliata. Quindi è necessario porre attenzione su come osservarle per evitare di nascondere oggetti a causa di esse.

1.1 Datasheet Ping360

Nelle tabelle sottostanti, riprese dal sito ufficiale della *BlueRobotics* [1], vengono riportate le caratteristiche acustiche, elettriche e fisiche del sensore. Inoltre vengono accennati i tipi di comunicazione che si possono avere con la Raspberry Pi, dove è montato il sistema *BlueOS*, e con il PC su cui viene installato *Ping Viewer* [2].

Caratteristiche elettriche	
Tensione di alimentazione	11 - 25 V
Consumo energetico massimo	5 W

Tabella 1.1 – Elettrico Ping360.

Comunicazione	
Protocollo segnale	USB, UDP, RS485
Protocollo messaggio	Protocollo ping
Messaggi	common, ping360

Tabella 1.2 – Comunicazione Ping360.

Caratteristiche acustiche	
Frequenza	750 kHz
Azimuth	2°
Elevation	30°
Range minimo	0,75 m
Range massimo	50 m
Risoluzione range	0,08% del range
Settore scansionato	2 - 360°
Velocità di scansione a 2 m	0,698 rad/s
Velocità di scansione a 2 m	0,179 rad/s

Tabella 1.3 – Acustica Ping360.

Caratteristiche meccaniche	
Profondità massima	300 m
Intervallo temperatura	0 - 30° C
Peso (con cavo)	510 g

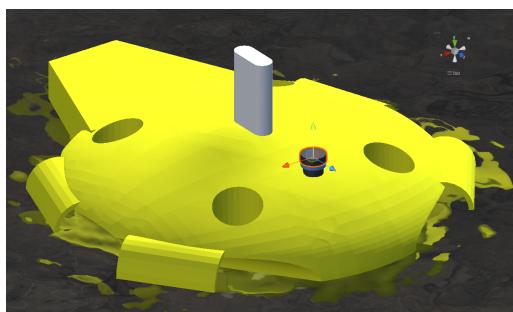
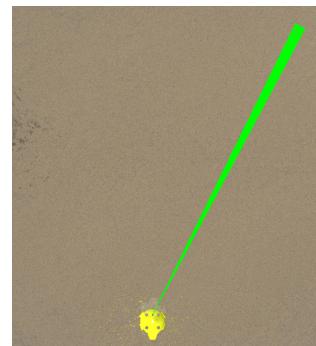
Tabella 1.4 – Meccanica Ping360.

1.2 Modellazione Unity

L’ambiente simulativo per l’acquisizione dei dati è stato sviluppato su Unity. Esso consiste in un piccolo porto che si affaccia su uno specchio d’acqua nel quale sono stati inseriti vari oggetti da scandagliare. Il sistema di riferimento è una terna sinistrorsa con asse Y normale alla superficie marina.

**Figura 1.7** – Ambiente simulativo Unity.

Per poter rappresentare il sensore è stato utilizzato un insieme di cilindri coassiali di tipo *gameObject* all’interno dell’interfaccia grafica. Esso ha un sistema di riferimento solidale, in cui l’asse longitudinale è parallelo all’asse Y del sistema fisso e ruota intorno ad esso. Ping360 è poi fissato sopra il ROV con la stessa posizione e orientazione del caso reale, come in figura 1.2. Nel caso in esame verrà posto sopra l’AUV *Zeno*.

**Figura 1.8** – Modello Ping360.**Figura 1.9** – Fascio acustico in ambiente simulativo.

Capitolo 1 Modello fisico del sensore

Nell’ambiente grafico usato non è possibile ricreare dei segnali acustici e quindi è stato utilizzato come modello un insieme discreto di raggi luminosi in sostituzione. Questi ultimi sono generabili con la funzione *Ray*, che ha come ingresso posizione di partenza del raggio e direzione di propagazione. Attraverso due cicli *for* annidati è possibile creare l’intero fascio spazzando gli angoli di azimuth ed elevation con certo passo. Il valore dell’apertura orizzontale e verticale e la lunghezza dell’impulso sono stati fissati pari ai valori reali riportati in tabella 1.3, ma è possibile modificarli tramite l’*Inspector* del software per adattarli al proprio obiettivo.

Per il controllo della rotazione del sensore viene utilizzata la funzione *transform.Rotate* che varia l’orientazione del corpo ogni qualvolta che viene aggiornato l’ambiente. Il parametro che si occupa della singola rotazione è la velocità angolare *omega* moltiplicata per il tempo di aggiornamento della scena.

In tabella 1.5 vengono riportati tutti i valori fisici modificabili del sensore e i parametri necessari per la comunicazione UDP con gli altri software. A livello di discretizzazione del fascio acustico, sono stati scelti due valori diversi della risoluzione degli angoli per avere il miglior compromesso fra quantità di dati raccolti e rappresentazione grafica.

Inspector	
Azimuth	2°
Elevation	30°
Risoluzione azimuth	0,1°
Risoluzione elevation	1°
Angolo ROV	comando da tastiera
Range raggio	50 m
IP remoto	192.168.2.2
Porta	5005
SettoreAngolare	360°

Tabella 1.5 – Parametri modificabili per la propria missione.

Il parametro *SettoreAngolare* serve a selezionare il tipo di scansione da fare: se assume un valore pari a 360° il sensore acquisirà informazioni su tutto l’ambiente intorno a lui, se invece viene impostato un valore diverso il sonar scansiona solamente un settore angolare di ampiezza pari a quella indicata. Questa doppia natura permette di adattare il sensore alle proprie esigenze, da una maggior raccolta di informazioni (con il primo caso) o di focalizzarsi su una singola parte dell’ambiente (con il secondo). A questo punto, l’impulso sonoro si propaga e viene verificata la presenza di collisioni con i corpi tramite la funzione *Physics.Raycast*. Nel caso ve ne fossero, vengono raccolti i dati derivanti dall’insonificazione.

Dal punto di vista del tempo di aggiornamento della scena e dei dati, è stato scelto affinché ci sia la sincronizzazione di tutto quanto, ovvero ad ogni singola rotazione del motore viene acquisita una nuova misura. La frequenza scelta è pari a 11.44 Hz, poiché garantisce di ruotare alla velocità indicata in tabella 1.3 e di acquisire 400 misurazioni, numero necessario per la sincronizzazione con la rappresentazione su Ping Viewer.

Capitolo 2

Acquisizione dei dati e comunicazione con ambiente virtuale

2.1 Acquisizione dei dati

I dati acquisiti dall’ambiente virtuale si basano su due tipi di informazioni: la distanza dell’oggetto rilevato rispetto alla posizione del sensore e la tipologia di materiale che lo caratterizza, in quanto, al variare di esso si ottiene una riflessione dell’onda acustica differente e quindi una colorazione differente con cui viene rappresentato. In ambiente Unity è possibile osservare tali informazioni tramite le funzioni *hit.distance* e *hit.collider.gameObject.GetComponent<Renderer>().material.name* presenti all’interno della struttura *Raycast*. L’informazione sulla distanza è salvata nella variabile temporale *distances_matrix*, in seguito convertita in bytes nella variabile *tempb* e impilata in un array di bytes. La descrizione del materiale degli oggetti in scena è contenuta all’interno di etichette rappresentate da un numero intero che corrisponde a un valore fittizio di riflettività. Infine, queste informazioni vengono raccolte e inserite all’interno di un array di bytes, *distances_array_bin*, di dimensione pari al numero di raggi emessi nelle due direzioni moltiplicato per 4, in quanto sono necessari 4 byte per rappresentare la distanza in centimetri e il relativo materiale.

```
1 int[,] distances_matrix;    // inizializzazione matrice distanze +  
2     materiale  
3 byte[] distances_array_bin; // inizializzazione array byte per  
4     comunicazione UDP  
5 byte[] tempb;           // variabile per la conversione in byte  
6 distances_matrix = new int[row, column];  
7 distances_array_bin = new byte[column * row * 4];  
8 tempb = new byte[4];  
9  
10 // Costruzione della variabile distances_matrix che contiene le  
11 // informazioni da convertire in Bytes  
12 distances_matrix[row_index, column_index] = (int)(hit.distance * 100) +  
13     getMaterial(hit);  
14 // Conversione della variabile temp in Bytes  
15 tempb = BitConverter.GetBytes(distances_matrix[row_index, column_index]);  
16 distances_array_bin[((row_index*column+(column_index))*4)+0] = tempb[0];  
17 distances_array_bin[((row_index*column+(column_index))*4)+1] = tempb[1];  
18 distances_array_bin[((row_index*column+(column_index))*4)+2] = tempb[2];  
19 distances_array_bin[((row_index*column+(column_index))*4)+3] = tempb[3];
```

Listing 2.1 – Creazione dell’array *distances_array_bin*.

Il valore memorizzato è composto come in figura 2.1: la prima cifra corrisponde al materiale dell’oggetto e le successive caratterizzano la distanza di esso dal robot. Per comporre tale valore si moltiplica per 100 l’informazione delle distanza dell’oggetto, trasformando i metri in centimetri, a cui si moltiplica l’informazione sul materiale che è caratterizzata da un

unica cifra, quindi essa è moltiplicata per 10000 per osservarla in testa al numero finale, come si osserva nella figura sottostante.



Figura 2.1 – Elemento generico contenuto nella variabile *distances_matrix*.

Ulteriori dati fondamentali sono: l’angolo di imbardata del Ping360, l’angolo di imbardata del Rov e l’ampiezza del range massimo dell’onda acustica. I dati sugli angoli sono raccolti sfruttando il comando *transform.eulerAngles.y* e moltiplicati per 100, in modo da acquisirli con due cifre decimali, in seguito convertiti in bytes con maggiore precisione. Il range massimo dell’onda acustica non è altro che la distanza massima di rilevamento. Nel caso in esame, corrisponde alla lunghezza del singolo raggio laser della modellazione in Unity, il cui valore è gestito tramite la variabile *rayMaxDistance*. Nell’utilizzo di Ping Viewer per la produzione dell’immagine, il valore del massimo range è variabile manualmente dalle proprie impostazioni.

Come accennato precedentemente il sensore reale permette di variare l’ampiezza del settore angolare in cui si effettuano le acquisizioni delle misure e allo stesso modo viene gestito tramite la variabile *settoreAngolare*, espresso in gradi. Nel caso si stesse usando Ping Viewer per la rappresentazione grafica è importante imporre manualmente il settore angolare pari al valore scelto in Unity, utilizzando le impostazioni del software. Le informazioni inerenti agli angoli, alla range massimo e al settore angolare sono messi in coda al messaggio da trasmettere all’ambiente Simulink via UDP, sfruttando la funzione definita *AddByteToArray()*.

2.2 Comunicazione con ambiente virtuale

L’invio dei dati dall’ambiente Unity a MATLAB sfrutta la connessione UDP implementata tramite la classe *UDPClient*. Essa necessita di stabilire una porta di comunicazione dell’host, impostata su 5005 e dell’indirizzo IP del destinatario dei dati, cioè l’IP del dispositivo in cui si utilizza l’ambiente Simulink. Stabilita la connessione è possibile inviare i dati passando l’array di bytes come argomento al comando *udpClient.Send()*. In ambiente Simulink, si ricevono i bytes utilizzando un blocco *UDPReceive*, convertiti in tipo *int32* e modellati in una matrice di dimensioni corrispondenti al numero dei raggi, tenendo conto che in fondo al pacchetto di bytes ricevuti ci siano i due angoli di imbardata, del sensore e del Rov, e l’informazione sul range massimo di rilevazione.

Capitolo 3

Elaborazione dati

Ricevuto il pacchetto di bytes, l'informazione dall'ambiente simulativo viene diviso nei seguenti dati: la matrice di distanze e materiale, l'angolo di imbardata del sensore, l'angolo di imbardata del Rov, la dimensione del range massimo e l'ampiezza del settore angolare scansionato. Le variabili necessarie per l'elaborazione dei dati sono ottenute scindendo le informazioni contenute all'interno dei bytes come si descrive nel codice 3.1. L'array *information_ray* contiene le informazioni riguardanti il materiale dell'oggetto individuato in ambiente simulativo e della sua posizione rispetto al robot. Essa è espressa come matrice, tramite la variabile *Ping360*, in una dimensione che rispecchia il numero di raggi luminosi presenti nelle due dimensioni, in Unity. Le due informazioni riguardo l'oggetto rilevato sono raccolte in due matrici differenti: *material_matrix* e *distances_matrix*.

```
1 % Divisione delle informazioni ricevute dall'ambiente simulato
2 information_ray = msg(1:620); % dati ricevuti da
    ambiente simulativo inerenti alla rilevazione degli oggetti
3 angolo_heading = double(msg(621))/100; % angolo rotazione del
    sensore
4 angolo_Rov = double(msg(622))/100; % angolo imbardata ROV
5 rayMaxDistance = double(msg(623)); % lunghezza raggi
6 settoreAngolare = double(msg(624)) % settore scansionato
7
8 % Creazione della matrice delle distanze + materiali
9 Ping360 = double(reshape(information_ray ,20 ,31));
10
11 % Vengono isolate la matrice dei materiali e la matrice dei dati
12 material_matrix = fix(Ping360/10000);
13 distances_matrix = Ping360-(material_matrix*10000);
```

Listing 3.1 – Divisione delle informazioni ricevute dall'ambiente simulato.

L'informazione del materiale è divisa per 10000 ottenendo un'unica cifra, come nel caso di partenza esposto nel capitolo 2.1. Le distanze sono riportate in metri, ecco perché sono divise per 100. Queste due caratterizzazioni sono evidenziate nel codice sopra.

Lo svolgimento di questi calcoli viene fatto attraverso lo schema Simulink rappresentato nella figura sottostante 3.1.

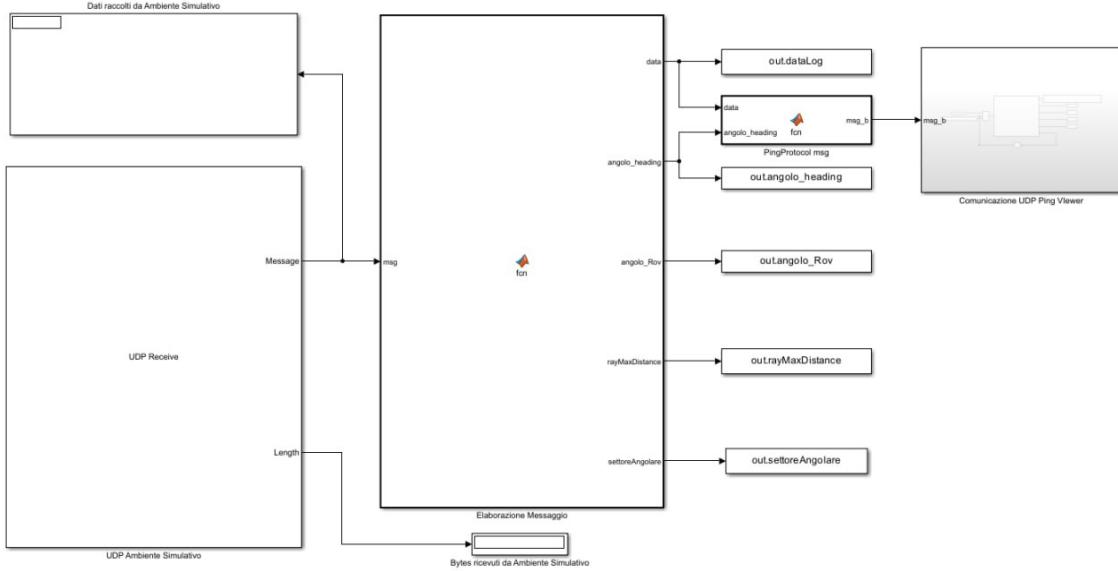


Figura 3.1 – Schema Simulink per l'elaborazione e invio dei dati.

3.1 Modellazione tramite Distribuzione Gaussiana

In ambiente Unity non è possibile emulare a pieno le funzionalità acustiche, infatti viene usato il modello laser tramite la struttura *RayCast*, lavorando nel dominio del tempo. Per tale motivo si incorre al modello Gaussiano che viene adattato al caso in esame, basandosi sulle basi teoriche proposte dall'articolo [5]. L'elemento della Gaussiana è utile per evitare che la rappresentazione grafica sia eccessivamente "rigida", creando una zona intorno alla rilevazione dell'oggetto che possa generare un effetto sfumato, sicuramente più realistico. Infatti, esse sono utilizzate sui campioni che descrivono le distanze.

Ogni elemento della *distances_matrix* è caratterizzata da una funzione Gaussiana con media pari all'elemento stesso e come deviazione standard un valore che dipende dalla distanza, dalla tipologia di materiale dell'oggetto rilevato e dal range massimo di rilevazione impostato. Il secondo motivo per cui si usano le distribuzioni gaussiane è legato al fatto che i raggi luminosi in ambiente Unity sono raggi puntiformi, in cui l'area colpita dal singolo raggio è puntiforme se paragonata alla dimensione dell'intervallo di osservazione. Per risolvere tale problema si utilizzano queste distribuzioni per interpolare l'area colpita dal singolo raggio, ampliando la zona di copertura. Si ottiene così una matrice, *gaussian_matrix*, di dimensione $1200 \times num_ray_azimuth$, dove il numero di colonne è il rapporto tra l'angolo di azimuth e la sua risoluzione. La deviazione standard è definita come:

$$\sigma = 0.002 \text{ rayMaxDistance} \frac{\text{distances_matrix}(i,j)}{100} \frac{1}{\text{material_matrix}(i,j)}; \quad (3.1)$$

In questa maniera l'immagine che rappresenta il risultato finale, non solo avrà una dispersione che dipende dalla distanza, ma essa dipenderà anche dalla superficie dell'oggetto e dalla sua riflettività, espressi in prima approssimazione dal singolo elemento della *material_matrix*. Il valore 0.002 è un guadagno scelto in maniera iterativa in modo tale che la rappresentazione grafica sia più veritiera possibile. È importante sottolineare la presenza della variabile *rayMaxDistance*, infatti, la dispersione che caratterizza le singole gaussiane dipende anche dalla distanza massima di rilevazione, in quanto minore è il suo valore, maggiore è l'accuratezza della misurazione. Questa elaborazione è descritta secondo le righe di codice sottostanti:

3.2 Generazione dell'immagine post-processing

```

1 % Costruzione delle matrici basate sulla densita di probabilita normale;
2 for j = 1:size(distances_matrix,2)
3     for i = 1:size(distances_matrix,1)
4
5         % Costruzione della deviazione standard
6         if material_matrix(i,j) ~= 0
7             sigma = 0.002*rayMaxDistance*distances_matrix(i,j)/100/
material_matrix(i,j);
8         else
9             sigma = 0.002*rayMaxDistance*distances_matrix(i,j)/100;
10        end
11
12        % Matrice densita di probabilita oggetti
13        gaussian_distribution = (material_matrix(i,j) * normpdf(x_gaussian,
distances_matrix(i,j)./100, sigma))';
14        gaussian_matrix(:,j) = gaussian_matrix(:,j) + gaussian_distribution
15    ;
16    end
17 end

```

Listing 3.2 – Utilizzo del modello Gaussiano.

3.2 Generazione dell'immagine post-processing

La rappresentazione grafica in post-processing si sviluppa su due software differenti: MATLAB e PingViewer, in questo capitolo verrà approfondito il primo.

L’obiettivo principale è quello di rappresentare una grafica più vicina possibile a quella originale del Ping Viewer, costruendo circonferenze e assi che fanno da riferimenti per la lettura del grafico stesso rispetto alla posizione centrale che il robot occupa sull’immagine prodotta, come è possibile osservare in figura 3.2. Per ottimizzare la rappresentazione del plot si è scelto di costruire una matrice di dimensione 200 x 200, in modo da andare a colorare solo la cella corrispondente alla posizione dell’oggetto rilevato, riducendo i costi computazionali rispetto al caso in cui debba colorare anche le celle in cui non è stato rilevato alcun oggetto.

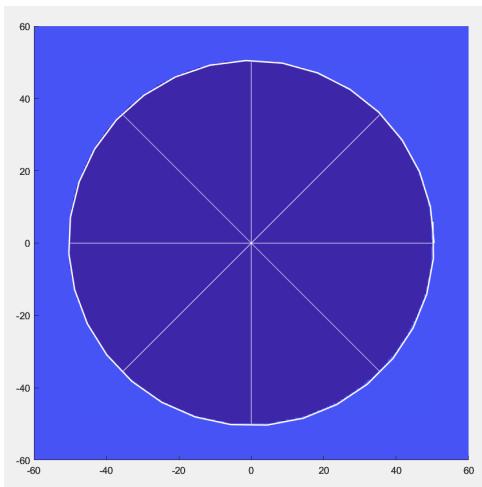


Figura 3.2 – Rappresentazione grafica in MATLAB.

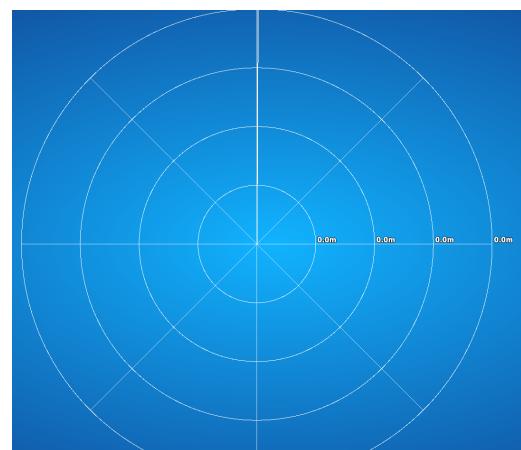


Figura 3.3 – Rappresentazione grafica in PingViewer.

Capitolo 3 Elaborazione dati

La prima informazione utile per produrre l’immagine è l’ampiezza del settore angolare, estratta dall’ambiente Simulink in quanto contenuta all’interno dell’array di bytes proveniente da Unity. In uscita al blocco *Elaborazione Messaggio* presente all’interno dell’ambiente Simulink in figura 3.1, viene salvato l’array *out.dataLog*, contenente la variabile *data*. All’interno di tale blocco si elabora la matrice *gaussian_matrix* per ottenere un unico vettore di dimensione 1200x1. Nel capitolo 4 si osserverà il perché del numero 1200. L’elaborazione consiste nell’effettuare la media sulle righe di tale matrice ottenendo un vettore colonna chiamato *mean_gaussian_matrix*. Esso racchiude le informazioni provenienti dalle densità di probabilità, assumendo così valori compresi tra 0 e infinito; perciò si è pensato di identificare il valor massimo del vettore come *max_data_view* ed effettuare una normalizzazione dei valori del vettore colonna, dividendo i suoi elementi per il *max_data_view*, ottenendo campioni compresi tra 0 e 1. Tale array viene chiamato *SCALE* ed è moltiplicato per 255, valore massimo che rappresenta il range di colori definiti per la costruzione dell’immagine. Il vettore finale viene chiamato *data* ed utilizzato sia per la rappresentazione grafica in MATLAB che con l’utilizzo di Ping Viewer. Il codice sottostante mostra l’elaborazione appena descritta:

```
1 % Costruzione del payload da inserire nel messaggio da inviare a Ping
2 % Viewer
3
4 mean_gaussian_matrix = zeros(1200,1); % inizializzazione del payload
5
6 % Media delle componenti della matrice densità di probabilità oggetti su
7 % riga per ottenere un unico fascio di apertura 1 gradiante
8 for k = 1:size(gaussian_matrix,1)
9 mean_gaussian_matrix(k,1) = mean(gaussian_matrix(k,1:size(gaussian_matrix
10 ,2)));
11 end
12
13 % Normalizzazione dei valori trovati per avere valori tra 0 e 255 al fine
14 % da scriverli in 8 bit o in una scala di colori tra 0 e 255
15
16 max_data_view = max(mean_gaussian_matrix); % max valore della media
17 lungo un fascio
18
19 if max_data_view == 0
20 SCALE = zeros(1200,1);
21 else
22 SCALE = (mean_gaussian_matrix)./max_data_view;
23 end
24
25 data = fix(SCALE*255);
26 % Payload da inserire nel messaggio da inviare a Ping Viewer, composto da
27 % array di 1200 dati
28 data = reshape(data, [1200 1]);
```

Listing 3.3 – Costruzione della variabile *data*.

La manipolazione dei dati sopra descritta si basa sulla teoria di funzionamento di un sensore single beam e viene colorata un unica fila di celle della matrice *ColorMap* alla volta. La funzione *mat2gray*, prende in oggetto il vettore colonna proveniente dall’array *out.dataLog*, in cui è stata salvata la variabile *data* e portata in MATLAB da Simulink, producendo una matrice in scala di grigi caratterizzata da un rumore sulle singole misurazioni di tipo *speckle*. Successivamente, si effettua la conversione in coordinate polari delle posizioni delle per colorare solo le quelle interessate dal settore angolare il cui valore è definito dal sensore, aggiornando ad ogni iterazione la matrice che produce l’immagine, la *ColorMap*. In MATLAB, la funzione *imagesc()* è utilizzata per visualizzare un’immagine

3.2 Generazione dell'immagine post-processing

come una matrice di colori. In figura 3.4 si descrive un esempio di colorazione delle celle della matrice *ColorMap* in seguito a un'analisi dei dati come descritto sopra.

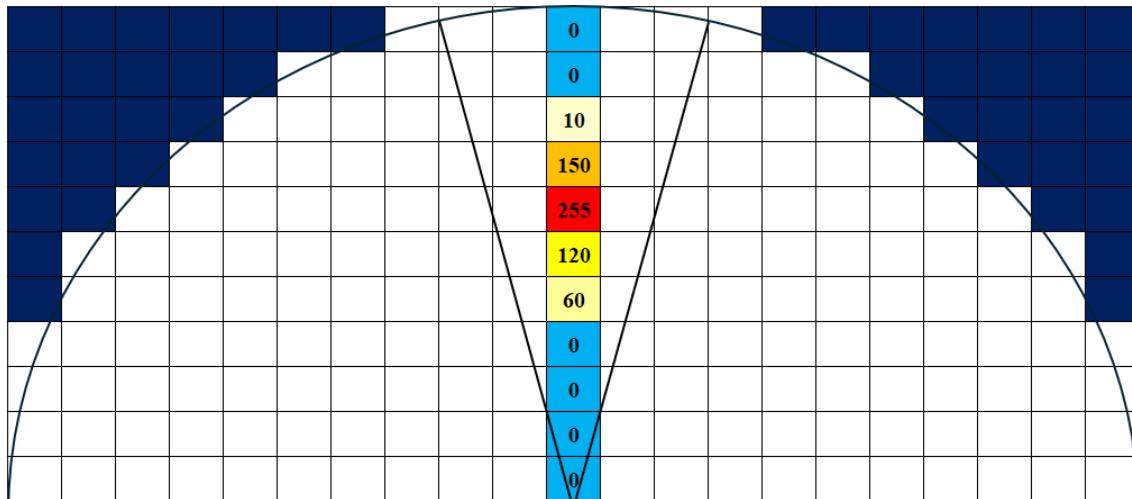


Figura 3.4 – Esempio di colorazione parziale della matrice *ColorMap*.

Le figure sottostanti descrivono un esempio di acquisizione dei dati. La figura 3.5 mostra la scansione di alcuni barili dall'ambiente virtuale; invece le figure 3.6 e 3.7 mettono a confronto le immagini ottenute nei due approcci di rappresentazione grafica in MATLAB e con l'utilizzo del software Ping Viewer.

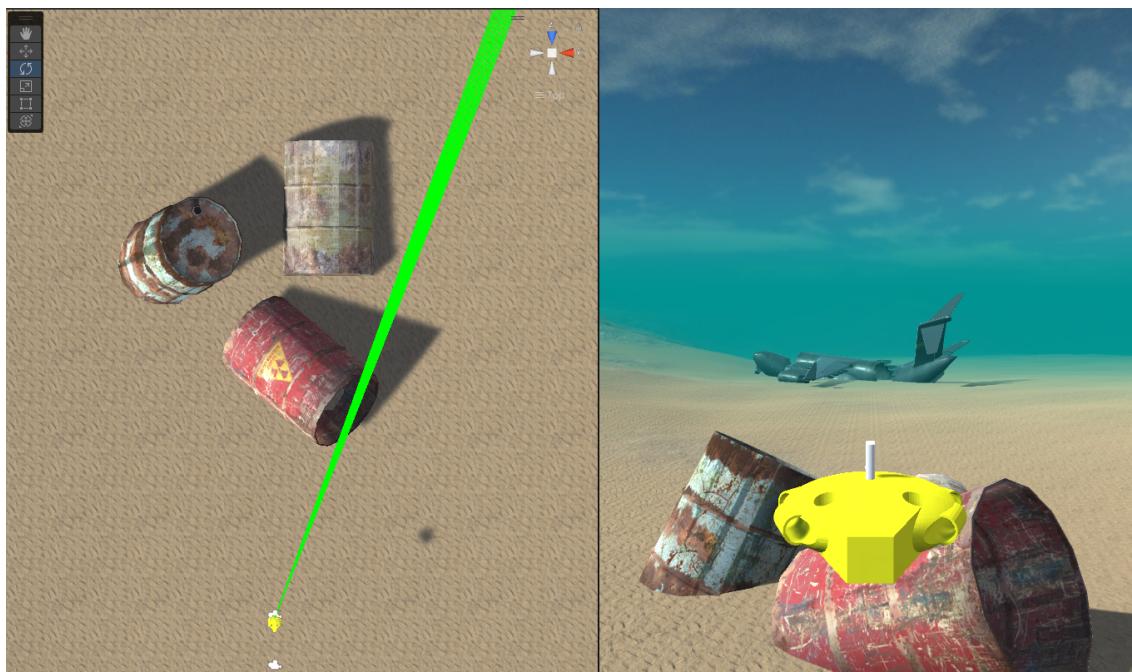


Figura 3.5 – Porzione di acquisizione dell'ambiente virtuale.

Capitolo 3 Elaborazione dati

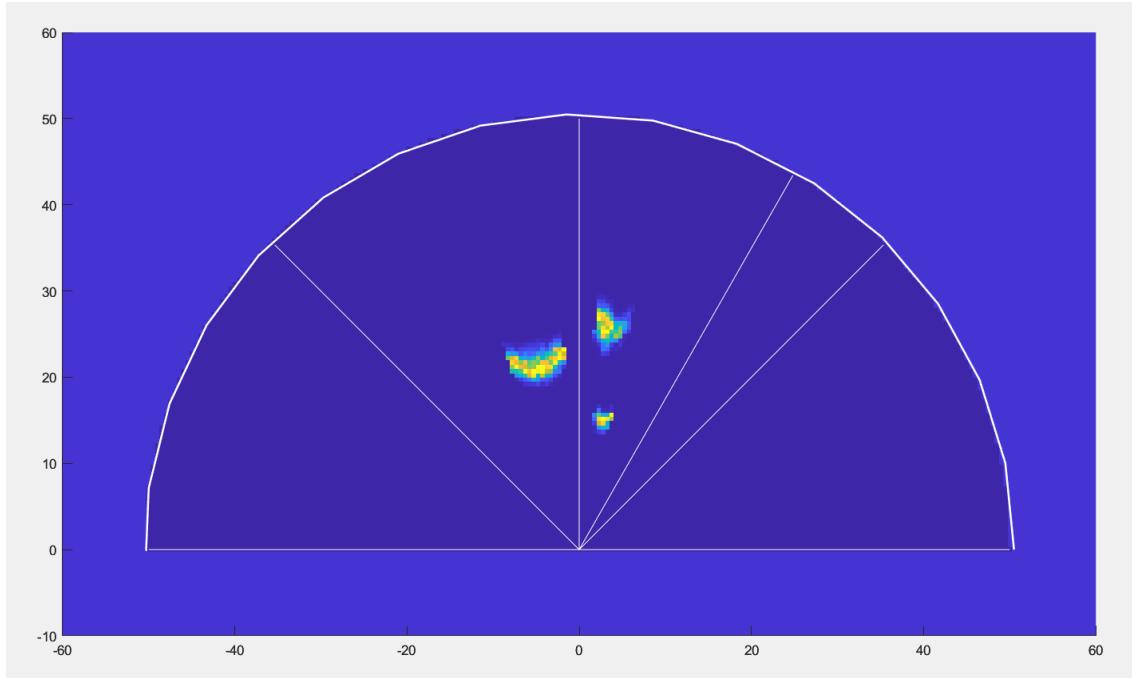


Figura 3.6 – Rappresentazione degli oggetti presenti in figura 3.5 utilizzando l'ambiente MATLAB.

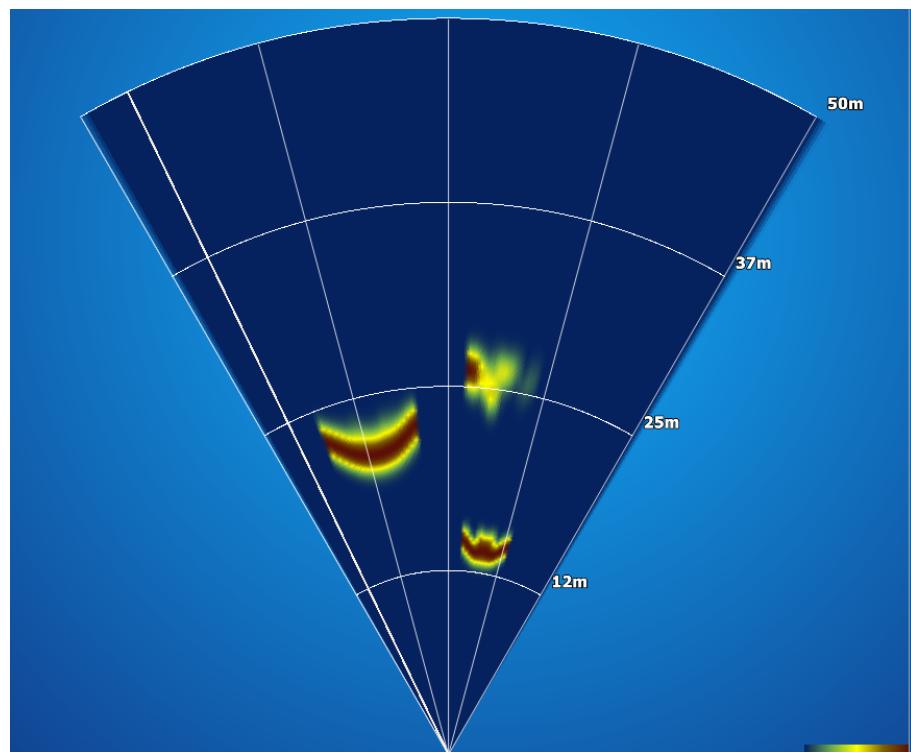


Figura 3.7 – Rappresentazione degli oggetti presenti in figura 3.5 utilizzando il software Ping Viewer.

Capitolo 4

Interfaccia Ping Viewer

Ping Viewer è l’interfaccia grafica per l’altimetro Ping1D e il sonar a scansione per immagini Ping360. L’applicazione consente di connettersi, configurare, visualizzare e registrare dati da un dispositivo Ping. L’analisi del progetto si sofferma prettamente sul sonar.

La Raspberry Pi presente sul robot *BlueRov2* supporta *BlueOS*, un sistema operativo versatile e flessibile per controllare e gestire una varietà di dispositivi subacquei, come veicoli autonomi, sensori e altri strumenti utilizzati nelle attività subacquee. Questo sistema fornisce un’interfaccia utente intuitiva, offrendo funzionalità avanzate per facilitare le operazioni subacquee e la raccolta dati.

Ping Viewer esegue automaticamente la scansione dei dispositivi disponibili sulle porte seriali (COM) e sulle connessioni UDP. Esso si connette utilizzando le porte 9090 (Ping1D) e 9092 (Ping360) sull’host all’indirizzo IP 192.168.2.2. Quest’ultimo è l’indirizzo IP predefinito del computer companion che fa funzionare *BlueOS*. Il computer complementare, il cui indirizzo IP deve necessariamente essere 192.168.2.1, configura automaticamente un bridge di comunicazione su queste porte con un sensore collegato.

È importante ricordare che l’obiettivo di questo progetto è sostituire il sensore fisico con quello modellato in ambiente Unity, quindi bisogna ripercorrere tutte le operazioni che il sensore vero farebbe per comunicare con l’interfaccia grafica.

4.1 Comunicazione tra MATLAB e Ping Viewer

L’analisi dei dati in ambiente MATLAB permette di riprodurre esattamente i pacchetti di messaggi che il sensore invierebbe all’interfaccia per la rappresentazione grafica. Sono rilevanti le impostazioni da imporre per la corretta comunicazione tra il computer su cui è in funzione MATLAB e il dispositivo su cui è attiva l’interfaccia grafica Ping Viewer. Il pc su cui è avviata tale interfaccia permette di definire manualmente i dati di comunicazione tramite il tasto *Manual Connection*; raggiungendo la finestra definita in figura 4.1 è possibile imporre i parametri di configurazione della comunicazione. Il sensore è di tipo Ping360, la cui porta di ricezione è la 9092, utilizzando la comunicazione UDP con l’ambiente Simulink il cui IP statico è 192.168.2.2.

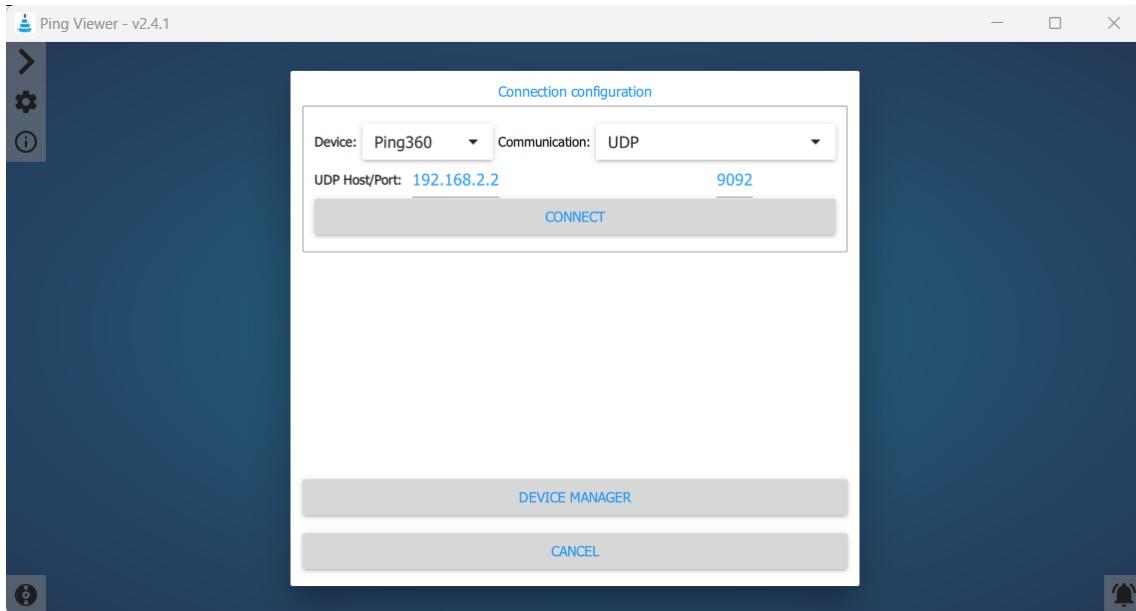


Figura 4.1 – Finestra di Ping Viewer in cui inserire i parametri di configurazione della comunicazione col sensore.

4.2 Tipologie di Messaggi

La comunicazione tra sensore e Ping Viewer avviene per via UDP, trasmettendo un pacchetto di dati in formato di bytes che descrivono alcune caratteristiche di invio e ricezione, di utilizzo del sensore e del payload che dipende dalle acquisizioni del sonar. Il messaggio utilizzato in questo contesto per trasmettere queste informazioni è il *2300 device_data*, definito all'interno della documentazione *Ping Protocol* presente sul sito ufficiale della BlueRobotics.

4.2.1 Messaggio 2300 device_data

Il messaggio *2300 device_data* viene utilizzato per comunicare lo stato attuale del sonar. Dalla documentazione ufficiale presente sul sito online della *BlueRobotics* è evidenziato che tale pacchetto di bytes è costituito da un intestazione, un payload e un checksum, come definito in figura 4.2.

Byte	Tipo	Nome	Descrizione
0	u8	inizio1	Identificatore frame iniziale, ASCII 'B'
1	u8	inizio2	Identificatore frame iniziale, ASCII 'R'
2-3	u16	lunghezza_carico_utile	Numero di byte nel payload.
4-5	u16	messaggio_id	L'ID del messaggio.
6	u8	src_device_id	L'ID del dispositivo che invia il messaggio.
7	u8	dst_device_id	L'ID dispositivo del destinatario previsto del messaggio.
8-n	u8[]	carico utile	Il carico utile del messaggio.
(n+1)-(n+2)	u16	somma di controllo	Il checksum del messaggio. Il checksum viene calcolato come la somma di tutti i byte non checksum nel messaggio.

Figura 4.2 – Formato del messaggio: intestazione, payload e checksum.

In tabella 4.1 sono evidenziati i valori dei parametri scelti adeguatamente per la costruzione dell'intestazione del messaggio 2300.

Nome	valore decimale
Numero di byte del payload	1214
ID del messaggio	2300
ID del dispositivo che invia il messaggio	2
ID del dispositivo che riceve il messaggio	0

Tabella 4.1 – Parametri scelti per caratterizzare l'intestazione.

La costruzione del payload si basa sulla documentazione raccolta in *PingProtocol* [3] definita in figura 4.3.

Capitolo 4 Interfaccia Ping Viewer

Tipo	Nome	Descrizione	Unità
u8	modalità	Modalità operativa (1 per Ping360)	
u8	guadagno_impostazione	Impostazione del guadagno analogico (0 = basso, 1 = normale, 2 = alto)	
u16	angolo	Angolo della testa	gradianti
u16	durata_trasmissione	Durata della trasmissione acustica (1~1000 us)	microsecondi
u16	periodo_campione	Intervallo di tempo tra i singoli campioni di intensità del segnale in incrementi di 25 ns (da 80 a 40.000 == da 2 a 1.000 us)	eicosapenta-nanosecondi
u16	frequenza_di_trasmissione	Frequenza operativa acustica (500~1000 kHz). A causa della larghezza di banda ridotta del ricevitore acustico è pratico utilizzare, ad esempio, solo tra 650 e 850 kHz.	kilohertz
u16	numero_di_campioni	Numero di campioni per segnale riflesso (valori supportati: 200~1200)	campioni
u16	lunghezza_dati	La lunghezza del campo vettoriale precedente	
u8[]	dati	Una serie di misurazioni della forza di ritorno effettuate a intervalli regolari nella regione di scansione. Il primo elemento è la misurazione più vicina al sensore e l'ultimo elemento è la misurazione più lontana nell'intervallo scansionato.	

Figura 4.3 – Costruzione del payload.

In tabella 4.2 sono evidenziati i valori dei parametri scelti adeguatamente per la costruzione del messaggio tipo 2300.

Nome	valore decimale
modalità	1
guadagno_impostazione	0
durata di trasmissione	11
periodo_campione	88
frequenzatrasmissione	750
numero_campioni	1200
lunghezza_dati	1200

Tabella 4.2 – Parametri scelti per caratterizzare il payload.

Il campo *dati* è riempito con i 1200 elementi raccolti nella variabile salvata *out.dataLog* dall'elaborazione dei dati in Simulink. Tale oggetto viene ottenuto come già descritto nel capitolo 3.2. Il coefficiente moltiplicativo 255 in questo caso rappresenta il valore massimo codificabile in 8 bit, in modo da avere un adeguata rappresentazione numerica, distribuita e codificabile in tale formato. Il vettore finale viene chiamato *data* e rappresenta il payload di 1200 bytes che verrà caricati nel messaggio 2300. L'ultimo parametro da descrivere è *heading angle*, cioè l'informazione proveniente dall'ambiente Unity sulla direzione di propagazione dell'onda acustica che in ambiente virtuale corrisponde alla direzione centrale dei raggi laser. Essa viene estratta dal blocco *Elaborazione Messaggio* e codificato in tipo *unit16* all'interno del blocco *PingProtocol msg* e infine aggiunta al vettore *msg*.

4.2.2 Messaggio di registrazione

In fase di sperimentazione e analisi dei dati, il Ping Viewer è in grado di creare e salvare automaticamente dei file di registrazione, file di *Log*, utili per riesaminare le acquisizioni effettuate dal sonar.

Lo script MATLAB *Costruzione_messaggio* è in grado di creare tale messaggio di Log senza la comunicazione UDP con l'interfaccia grafica, avviando il file Simulink e salvando da esso la variabile tridimensionale *out.payloadLog*. Il messaggio di replay presenta una intestazione iniziale con struttura precisa in cui vengono inserite le seguenti informazioni: "4 PingViewer sensor log file b129811", "data dell'elaborazione del file", "ora in cui è iniziata la rilevazione", "versione del PingViewer", dove *b129811* rappresenta un hash commit. A meno di data e ora, i cui valori sono al quanto trascurabili, il restante della stringa rimane costante per ogni sperimentazione. Quindi, tale stringa viene salvata in un array di bytes, che restano costanti per qualsiasi analisi, salvato in un file MATLAB, e viene richiamato nello script *Costruzione_messaggio* con il comando *load('intestazione_decimale.mat')*.

Si costruisce successivamente un vettore *alpha* in cui si memorizzano gli angoli che descrivono la direzione di propagazione dell'onda acustica trasmessa dal sensore.

Inoltre, osservando un messaggio di Log di una sperimentazione passata, si osserva che il tempo della rilevazione è posto in testa all'array di bytes che contiene le varie informazioni, citate sopra, ottenute dalle acquisizioni del sensore. Perciò, tramite un ciclo *for* si costruisce una matrice *tempo* le cui righe contengono l'informazione sulla temporizzazione associata alla rilevazione dei dati dal sonar. In coda all'informazione sul tempo, ogni singolo pacchetto prevede un valore che identifica il numero di bytes contenuto all'interno del messaggio, pari a 1224 per il *2300 device_data*.

00 00 00 34 00 50 00 69	00 6E 00 67 00 56 00 69	...4.P.i.n.g.V.i
00 65 00 77 00 65 00 72	00 20 00 73 00 65 00 6E	.e.w.e.r. .s.e.n
00 73 00 6F 00 72 00 20	00 6C 00 6F 00 67 00 20	.s.o.r. .l.o.g.
00 66 00 69 00 6C 00 65	00 00 00 01 00 00 00 0E	.f.i.l.e.
00 62 00 31 00 32 00 39	00 38 00 31 00 31 00 00	.b.1.2.9.8.1.1..
00 32 00 32 00 30 00 32	00 33 00 2D 00 30 00 36	.2.2.0.2.3.-.0.6
00 2D 00 32 00 39 00 54	00 31 00 38 00 3A 00 32	.-.2.9.T.1.8.:.2
00 39 00 3A 00 33 00 38	00 2D 00 30 00 33 00 3A	.9.:.3.8.-.0.3.:
00 30 00 30 00 00 00 0C	00 76 00 32 00 2E 00 34	.0.0.....v.2...4
00 2E 00 31 00 00 00 2E	00 57 00 69 00 6E 00 64	...1.....W.i.n.d
00 6F 00 77 00 73 00 20	00 31 00 30 00 20 00 56	.o.w.s. .1.0. .V
00 65 00 72 00 73 00 69	00 6F 00 6E 00 20 00 32	.e.r.s.i.o.n. .2
00 30 00 30 00 39 00 00	00 04 00 31 00 30 00 00	.0.0.9.....1.0..
00 01 00 00 00 02 50 52	49 4D 4F 00 4D 45 53 53PRIMO.MESS
41 47 47 49 4F 00 00 00	18 00 30 00 30 00 3A 00	AGGIO.....0.0.:
30 00 30 00 3A 00 30 00	30 00 2E 00 30 00 38 00	0.0.:.0.0....0.8.
38 00 00 04 C8 42 52 BE	04 FC 08 02 00 01 00 0E	8....LBRJ
00 0B 00 58 00 EE 02 B0	04 B0 04 50 41 59 4C 4F	...X.ε.\\.\PAYLO
41 44 00 31 32 30 30 00	42 59 54 45 53 B0 0A 53	AD.1200.BYTES\\S
45 43 4F 4E 44 4F 00 4D	45 53 53 41 47 47 49 4F	ECONDO.MESSAGGIO
00 00 00 18 00 30 00 30	00 3A 00 30 00 30 00 3A0.0.:.0.0.:
00 30 00 30 00 2E 00 31	00 37 00 36 00 00 04 C8	.0.0....1.7.6....L
42 52 BE 04 FC 08 02 00	01 00 0F 00 0B 00 58 00	BRJ
EE 02 B0 04 B0 04 50 41	59 4C 4F 41 44 00 31 32	ε.\\.\PAYLOAD.12
30 30 00 42 59 54 45 53	2D 0A +	00.BYTES-.

Figura 4.4 – Esempio messaggio di Log.

I bytes evidenziati in giallo in figura 4.4 non sono presenti nel messaggio reale, sono stati utilizzati in questo esempio solo per motivi esplicativi e descrittivi.

4.2.3 Messaggio in tempo reale

Impostati i parametri di comunicazione UDP dell’interfaccia Ping Viewer, è possibile inviare un messaggio di handshake all’ambiente Simulink, destrutto in figura 4.5 e 4.6, i cui dati sono trasmessi in formato uint8. Il sensore risponde con un pacchetto di bytes di dati di tipo uint8, definiti in figura 4.7, e solo dopo la sua ricezione Ping Viewer accetta il pacchetto *2300 device_data* di dimensione pari a 1224 bytes. La documentazione a riguardo dei messaggi di handshake non è molto arricchita, infatti per rilevarli e identificare il valore dei singoli byte è stato utilizzato Wireshark, software utile per l’analisi del traffico di rete che consente agli utenti di catturare e analizzare i pacchetti di dati che viaggiano attraverso una rete di computer.

4.2 Tipologie di Messaggi

Byte	Valore (esadecimale)	Valore (decimale)	Tipo	Nome	Descrizione
0	0x42 ("B")	66	u8	inizio1	Questo è un identificatore di inizio messaggio, una lettera ASCII 'B'
1	0x52 ("R")	82	u8	inizio2	Questo è un identificatore di inizio messaggio, una lettera ASCII 'R'
2-3	0x0002	2	u16	lunghezza_carico_utile	Il numero di byte nel <code>general_request</code> payload
4-5	0x0006	6	u16	messaggio_id	Questo messaggio è un <code>general_request</code> messaggio (id pingmessage-common#6).
6	0x00	0	u8	src_device_id	L'ID del dispositivo che invia il messaggio. Questo campo non è attualmente implementato e deve essere popolato con un valore pari a zero
7	0x00	0	u8	dst_device_id	L'ID dispositivo del destinatario previsto del messaggio. Questa parte non è attualmente implementata e deve essere popolata con un valore pari a zero
8-9	0x0005	5	u16	ID_richiesto	Questo è l'id del messaggio che vorremmo che il dispositivo ci trasmettesse. Gli ID validi sono quelli nella <code>get_categoria</code> dei messaggi
10-11	0x00a1	161	u16	somma di controllo	Il checksum del messaggio. Il checksum viene calcolato come la somma di tutti i byte non checksum nel messaggio: $66 + 82 + 2 + 6 + 0 + 0 + 5 = 161$

Figura 4.5 – Documentazione definita in *PingProtocol* per il messaggio di handshake di bytes che il Ping Viewer invia al sensore.

66	82	2	0	6	0	0	0	4	0	160	0
----	----	---	---	---	---	---	---	---	---	-----	---

Figura 4.6 – Array di bytes del messaggio di handshake che invia l'interfaccia Ping Viewer.

66	82	6	0	4	0	2	0	2
67	3	1	1	0	234	0	0	0

Figura 4.7 – Array di bytes del messaggio di handshake che invia il sensore Ping360.

Capitolo 5

Comunicazione UDP

Lo *User Datagram Protocol (UDP)* è uno dei principali protocolli di rete per lo scambio di dati su Internet. È basato sul trasporto di pacchetti e solitamente è usato in combinazione con il protocollo di livello di rete IP. L'UDP è una procedura di tipo *connectionless*, ovvero non viene creata una comunicazione tra mittente e ricevente; inoltre non gestisce il riordinamento dei pacchetti né la ritrasmissione di quelli persi, ed è perciò che generalmente è considerato di minore affidabilità. In compenso è molto rapido ed efficiente per le applicazioni "leggere" in cui la latenza è critica; per questo si usa per le applicazioni per le quali un pacchetto in ritardo o perso non influisce sul passaggio di informazioni.

I messaggi inviati e ricevuti con questo protocollo vengono trasmessi in pacchetti, detti *datagrammi*. Questi sono composti da:

- un'intestazione che contiene informazioni come l'indirizzo IP del mittente e del destinatario, il numero di porta e la lunghezza del pacchetti;
- un campo di *checksum* opzionale nell'intestazione del datagramma per rilevare eventuali errori di trasmissione;
- le informazioni da trasmettere (*payload*).

In questo progetto viene utilizzata comunicazione UDP per collegare i vari software.

5.1 Comunicazione Unity - Simulink

La gestione della comunicazione tra Unity e Simulink avviene semplicemente con il blocco *UDP Receive*, come mostrato in figura 5.1.

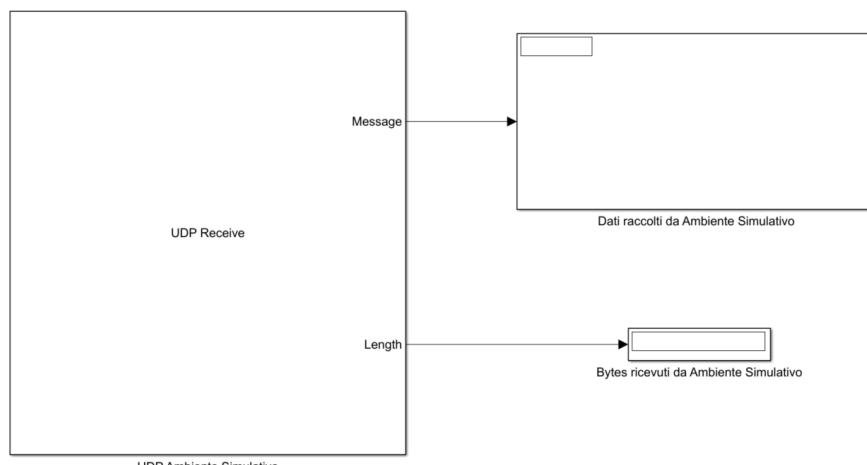


Figura 5.1 – Blocco Simulink UDP Receive per comunicazione con ambiente simulativo.

I parametri da inserire nel blocco per la trasmissione dei dati sono i seguenti:

- indirizzo IP locale pari a '192.168.2.2';
- numero della porta locale pari a 9092 (che identifica il Ping360);
- dimensione buffer di ricezione pari a 500000;
- lunghezza massima messaggio pari a 624;
- tipo di dato pari a *int32*;
- tempo di campionamento pari a 0.0874 secondi.

5.2 Comunicazione Simulink - Ping Viewer

A differenza della precedente trasmissione, l'interazione con il software Ping Viewer per inviare e ricevere datagrammi è leggermente più complessa ed è necessario costruire un blocco Simulink personalizzato (*udp_1block*) con una *S-function*. Questo elemento è il cuore centrale del sottosistema *Comunicazione UDP con Ping Viewer*, poiché tramite un file sorgente in linguaggio C crea il socket per la ricezione e invio dei messaggi con caratteristiche precise. Infatti per lo scambio di dati, Ping Viewer va ad usare 2 indirizzi prefissati e 2 porte per lui stesso e il sensore (nel nostro caso per il modello). I valori di IP sono ottenibili dalla documentazione di *BlueRobotics* e anche la porta per il Ping360, ma quella del software è randomica, poiché è scelta dal sistema operativo su cui è montato il visualizzatore grafico.

	Ping Viewer	Ping360
IP	192.168.2.1	192.168.2.2
Porta	randomica	9092

Tabella 5.1 – Indirizzi IP e porte.

In figura 5.2 vengono mostrati l'input del blocco *udp_1block*, ovvero i messaggi da inviare a Ping Viewer che sono il primo di *handshake* e il *2300_device_data*, e gli output, ovvero i datagrammi ricevuti e il loro corrispondente valore di bytes, l'intero correlato al socket, la porta di Ping Viewer e il flag per fare il cambio di messaggio da inviare.

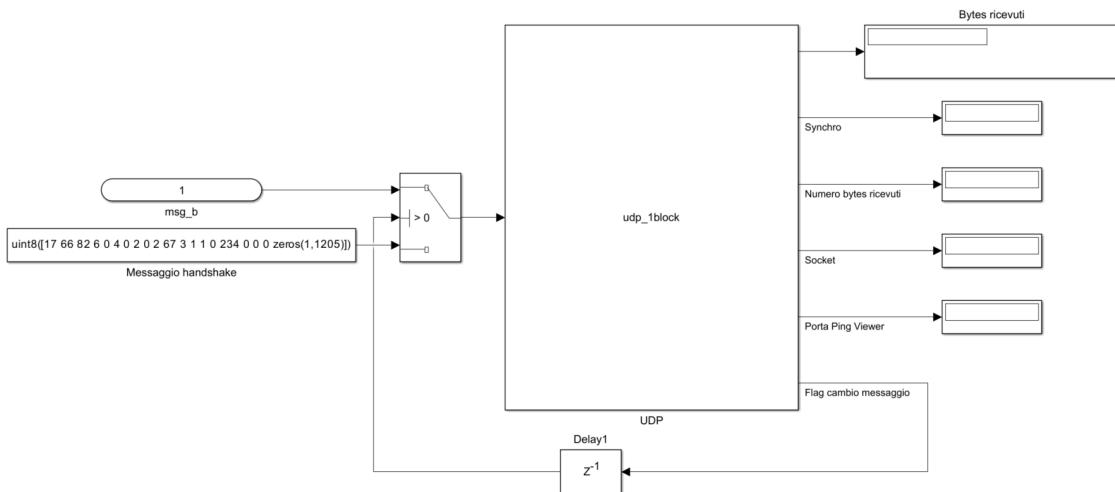


Figura 5.2 – Sottosistema Simulink UDP per comunicazione con Ping Viewer.

5.2.1 Blocco *udp_1block*

Il blocco *udp_1block* è una *S-function* che può essere analizzata su due livelli strettamente collegati, ovvero come file sorgente in linguaggio C per la creazione della comunicazione e come semplice blocchetto in ambiente Simulink. Per il secondo è possibile impostare solo i dati da fornire per lo scambio dei dati, invece per il primo è possibile gestire la costruzione del *socket* e modificare la struttura del blocco stesso. I parametri che possono essere impostati, dopo aver associato alla S-function l'omonimo file .c, sono:

- indirizzo IP locale pari a '192.168.2.2';
- indirizzo IP Ping Viewer pari a '192.168.2.1';
- porta locale pari a 9092;
- tempo di campionamento pari a 0.0874 secondi;
- numero di byte del messaggio ricevuto pari a 12.

Il file sorgente sopra indicato, oltre a svolgere la funzione principale di generare un socket, si occupa della comunicazione tramite le funzioni *recvfrom* e *sendto*. Il codice può essere suddiviso in quattro sezioni: la prima che si occupa di indicare i parametri che inserisce l'utente a mano (quelli appena citati) e gli input e output della S-function (figura 5.2); la seconda che gestisce l'inizializzazione di variabili e strutture con la funzione *mdlInitializeSizes*; la terza che racchiude tutti i comandi per la creazione del socket sotto la funzione *mdlStart*, che viene lanciata solo all'inizio della simulazione; infine la *mdlOutputs* che ingloba tutte le istruzioni per ricevere, inviare messaggi e gestire le variabili di ingresso e di uscita. È importante porre attenzione al fatto che il socket deve essere lo stesso in ricezione ed invio poiché Ping Viewer richiede specifici indirizzi e porte. Proprio per questo vengono usate le funzioni *recvfrom* e *sendto* al posto delle standard *recv* e *send* perché riescono a sfruttare queste informazioni.

Ultimo aspetto da considerare è la gestione dei pacchetti da inviare. Infatti dopo l'arrivo del primo datagrammo da parte del visualizzatore grafico, dal quale otteniamo l'informazione sulla porta, è necessario rispondere in primis con un messaggio di *handshake* di lunghezza 18 bytes e successivamente con l'informazione per disegnare. Le variabili che si occupano del controllo dei dati da inviare sono le seguenti: *B* che verifica che il primo byte dell'input sia 18 o 66 a seconda di cosa inviare e *flag_output* che abilita la trasmissione del payload dopo la comunicazione iniziale.

Capitolo 6

Analisi dei dati

In questa sezione vengono analizzati i dati raccolti nel corso delle simulazioni, ponendo attenzione all'effetto dato dall'insonificazione di materiali diversi e dalla variazione dei parametri dell'acustica o del tipo di scansione richiesta. Inoltre verranno messi a confronto i grafici prodotti da Simulink e Ping Viewer.

6.1 Validazione del modello

La validazione si occupa di garantire un risultato grafico adeguato e rappresentativo degli oggetti rilevati dall'ambiente virtuale in Unity. In figura 6.1 si osserva l'oggetto della rilevazione, un aereo militare naufragato. Le figure 6.2 e 6.3 sono il risultato grafico con il software Ping Viewer e in MATLAB. È possibile vedere come l'elaborazione dei dati fatta in precedenza permette un'ottima rappresentazione congruente con la visualizzazione aspettata e che gli elementi costituenti lo scenario assumono la posizione corretta e non vi è una distorsione dell'immagine data da moti relativi dei punti insonificati.

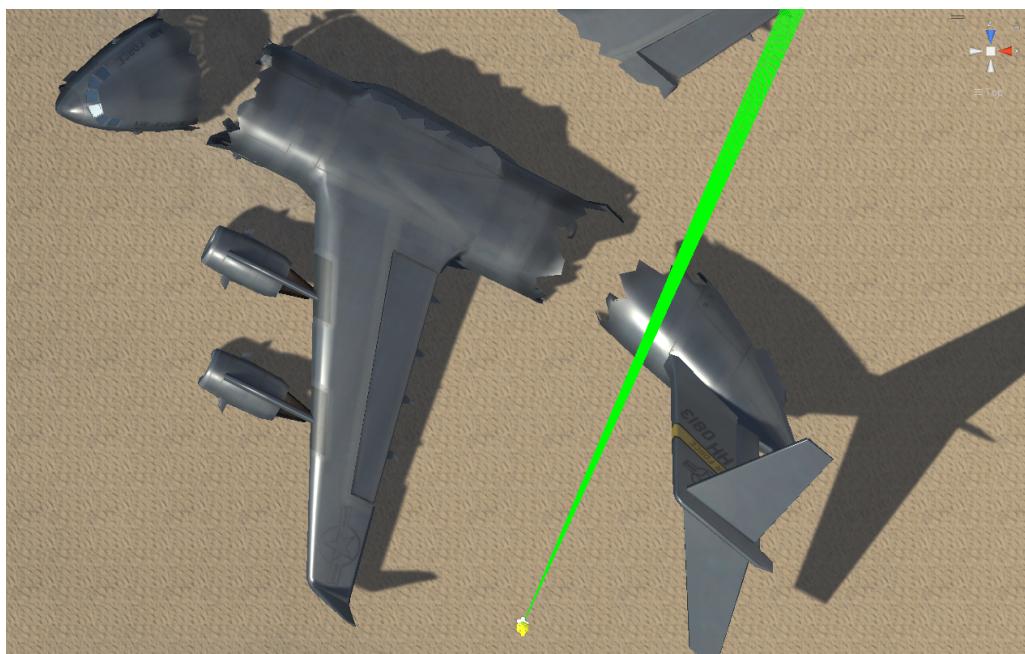


Figura 6.1 – Aereo militare naufragato presente in ambiente virtuale.

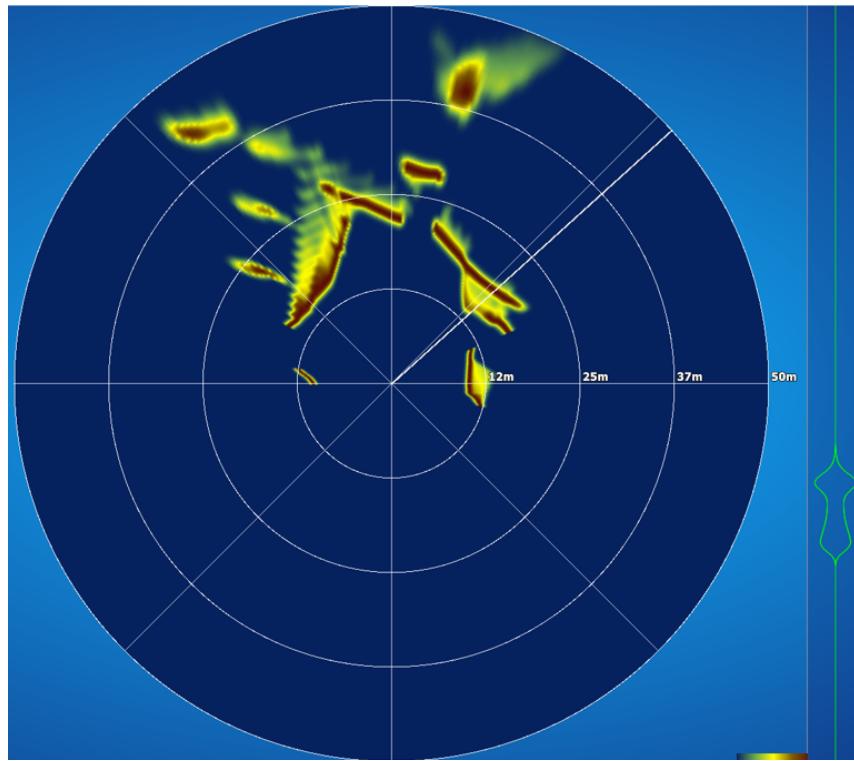


Figura 6.2 – Rappresentazione dell'aereo militare naufragato con Ping Viewer.

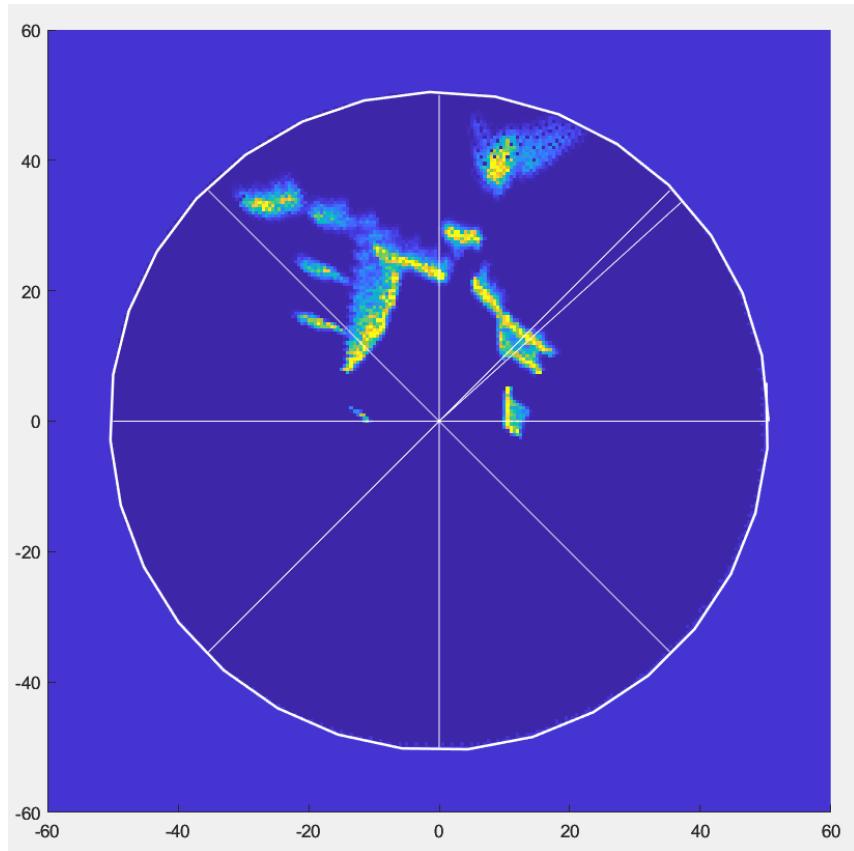


Figura 6.3 – Rappresentazione dell'aereo militare naufragato con MATLAB.

Un risultato altrettanto interessante si osserva riducendo l'ampiezza del range acustico, variando il valore della variabile *rayMaxDistance* e rimodulando il settore angolare. Questo tipo di approccio può essere usato per acquisire informazioni da oggetti di piccole dimensioni con una risoluzione più elevata. Un esempio può essere la rilevazione della statua del *discobolo* raffigurato in 6.4, il cui risultato grafico è descritto nelle figure 6.5 e 6.6. Si può osservare come dettagli più ridotti come il disco, la testa e la gamba destra vengono mostrati in modo chiaro.

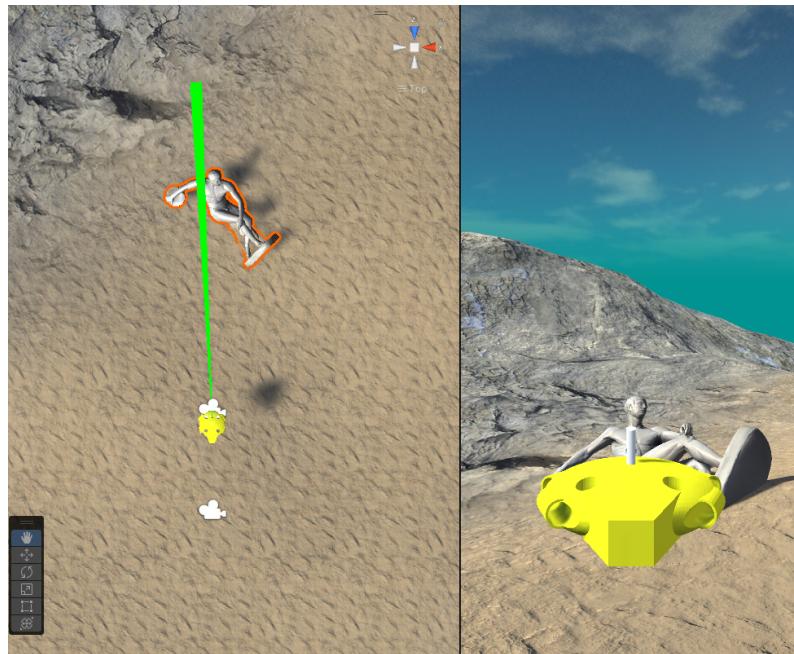


Figura 6.4 – Statua del discobolo presente in ambiente virtuale.

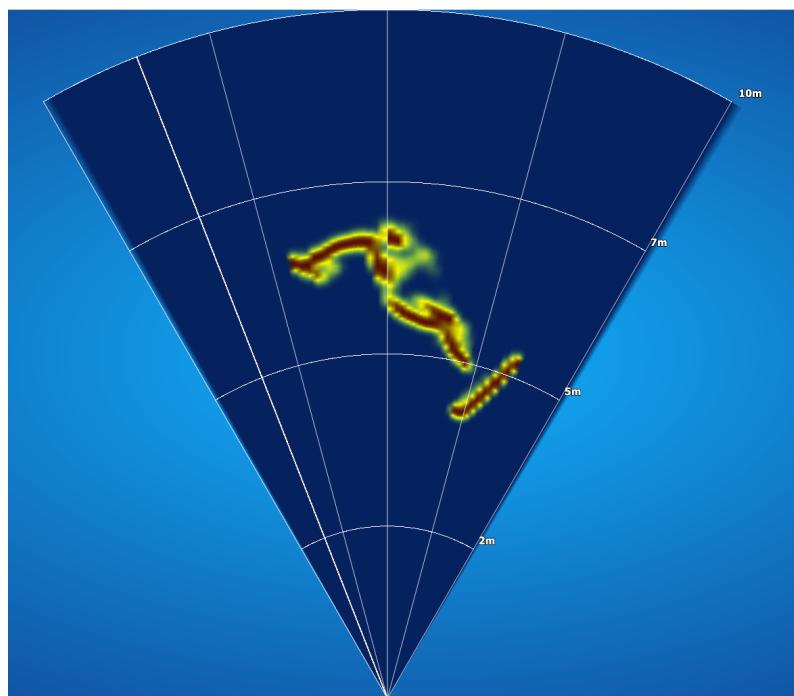


Figura 6.5 – Rappresentazione del discobolo con Ping Viewer.

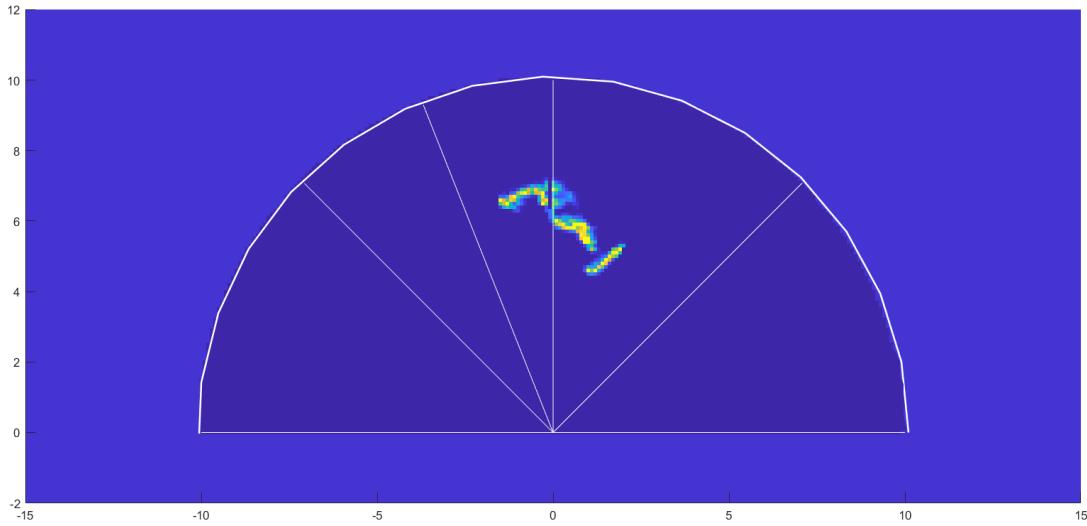


Figura 6.6 – Rappresentazione del discobolo con MATLAB.

I grafici ottenuti attraverso MATLAB a differenza di quelli del software di *BlueRobotics* hanno qualità peggiori dovuto al compromesso grafico-computazionale imposto. Infatti, richiedere una resa visiva maggiore aumenta notevolmente il tempo di rappresentazione. Inoltre, colorando un pixel alla volta, non vi è un gradiente di colori continuo con quelli vicini, cosa che sembra essere gestita da Ping Viewer vista la maggiore uniformità a parità di costruzione dei dati del payload.

6.2 Insonificazione di materiali con diversa riflettività

In questa sezione viene valutato l'effetto del materiale sulla riflessione delle onde acustiche a parità di scansione del settore e di distanza. Il parametro che gestisce l'effetto del materiale è la prima cifra della matrice *distance_matrix* ed è strettamente legato alla riflettività e geometria dell'oggetto.

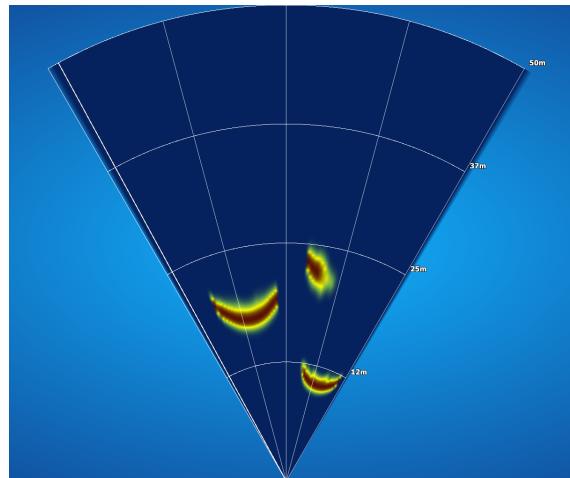


Figura 6.7 – Oggetto a basso valore di riflettività.

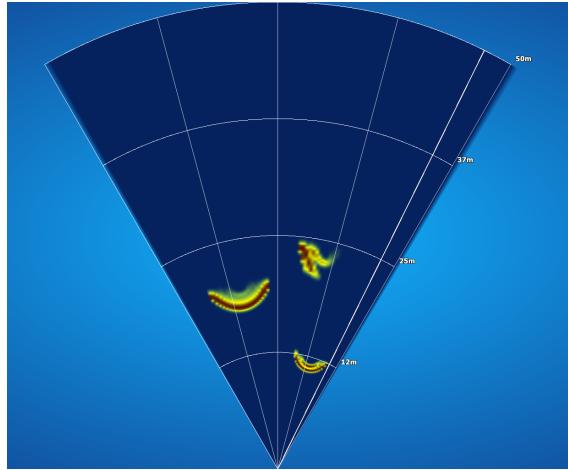


Figura 6.8 – Oggetto ad alto valore di riflettività.

Si può osservare dalle immagini 6.7 e 6.8 che un elevato valore garantisce un’immagine più precisa e nitida poiché si riduce l’incertezza data dalle gaussiane ed è analogo rispetto a quello che succede nella realtà.

6.3 Dettagli delle immagini acustiche

Il modello basato sulle gaussiane permette di osservare i dettagli tipici dell’insonificazione di ambienti. In figura 6.9 viene mostrato un aereo che viene osservato con una certa orientazione per vedere come l’impulso si infrange sulle superficie del veicolo. Con colore rossiccio vengono rappresentati i lati in vista e in blu quelli oscurati perché posizionati in secondo piano (zone di ombra). Invece, con i colori intermedi si vedono quelle zone che non sono colpite direttamente dal fascio, ma nella realtà ricevono il segnale riflesso da quelle direttamente intercettate. A livello geometrico le zone gialle sono colpite solitamente da raggi che hanno angolo di *elevation* grande o che si posizionano a distanze maggiori rispetto a quelle in evidenza. Ad esempio in figura 6.10 la parte sotto l’ala, tra fusoliera e primo motore, ha una colorazione giallastra.

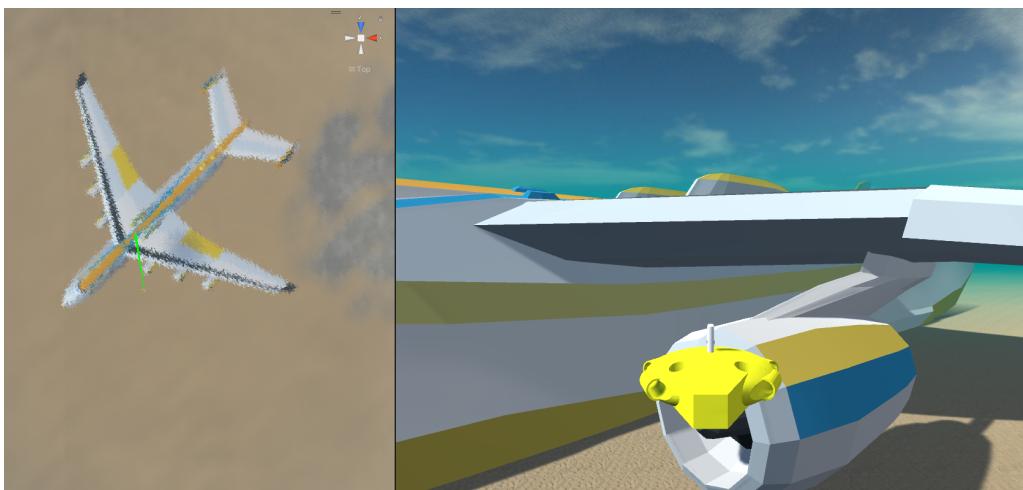


Figura 6.9 – Aeroplano da insonificare su Unity.

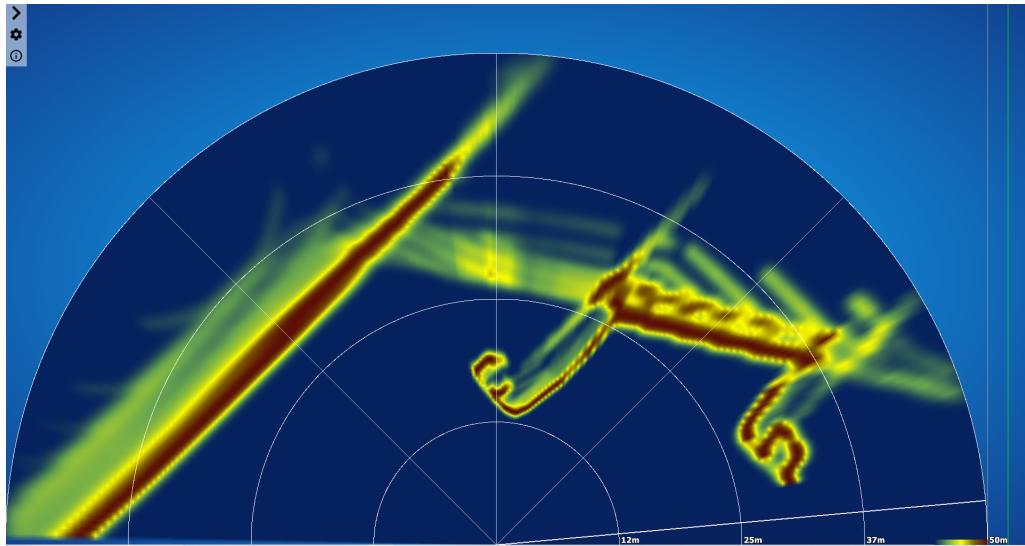


Figura 6.10 – Rappresentazione di zone di ombra acustica.

6.4 Limiti del modello

Nonostante il modello di sensore creato permetta di ottenere immagini molto veritieri esso presenta qualche limite in determinate situazioni a causa della fisica dei raggi e dell'ambiente di lavoro.

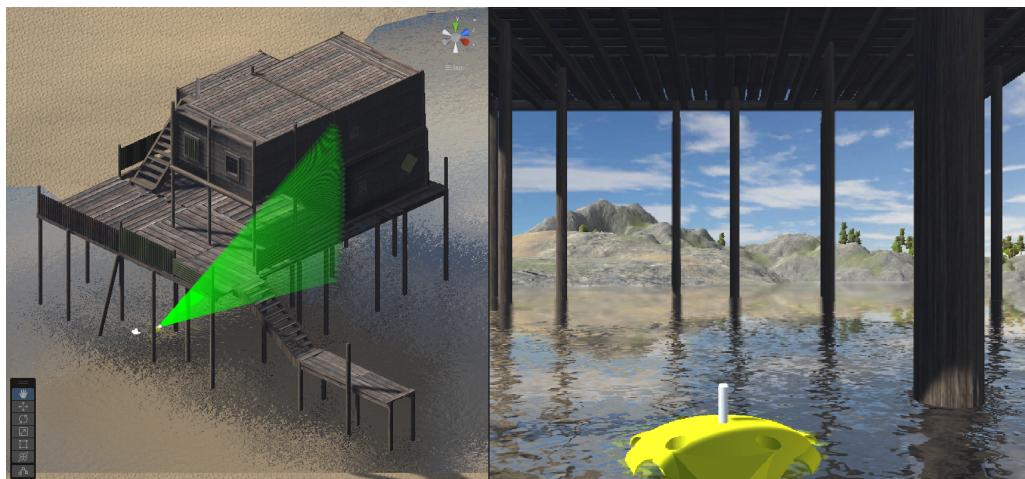


Figura 6.11 – Palafitte in Unity.

In figura 6.11, viene rappresentata la scansione di un porticciolo al fine di definire la posizione delle varie palafitte. Il risultato aspettato dovrebbe essere un'immagine con forte eco da parte dei pali e un altrettanto rumore dato dalla presenza di molti elementi intorno al ROV. In realtà, la visualizzazione è molto "pulita" ed è possibile distinguere ogni elemento, dalla banchina di attracco ai singoli pilastri. Questo è dovuto al fatto che i raggi non hanno la fisica dell'acustica e per il controllo della collisione si valuta il contatto con un oggetto e non il ritorno dell'informazione sonora (figura 6.12).

Inoltre l'ambiente simulativo per facilità di renderizzazione non va a rilevare il riflesso dell'onda acustica con l'acqua e il fondale marino, elementi che genererebbero enorme rumore sull'immagine.

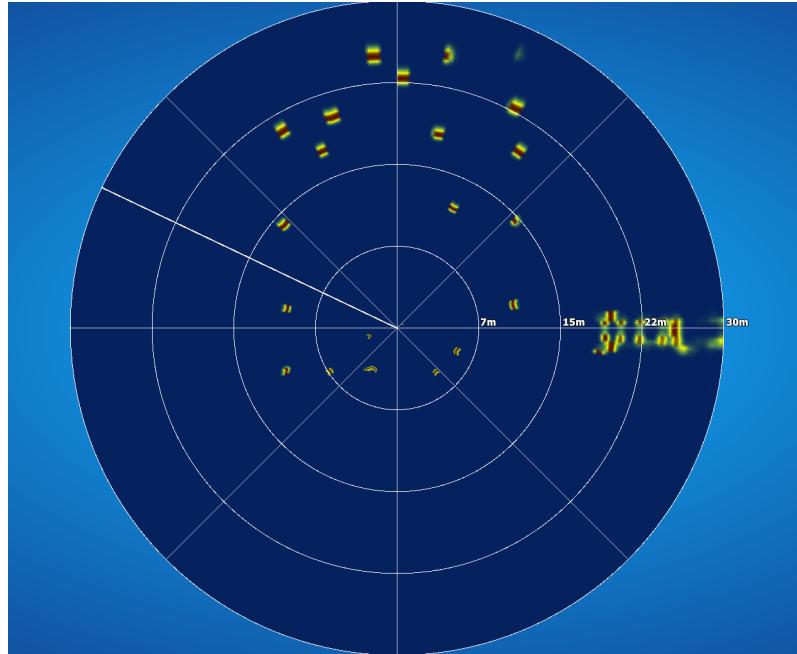


Figura 6.12 – Osservazione di un ambiente con palafitte.

Infine la rappresentazione grafica avviene dal punto di vista del sistema di Unity e quindi nel caso di cambio dell’assetto di Zeno, ad esempio veicolo imbarda, la rappresentazione successiva non dipenderà da dove punta la prua, ma sarà sempre la stessa. Questo elemento è importante per migliorare la navigazione del veicolo, infatti il software di *BlueRobotics* accede all’informazione sull’assetto per modificare la sua immagine e mostrare il miglior punto di vista. Per accedere a questo dettaglio in più, probabilmente è necessario acquisire i dati dalla IMU montata sul robot poiché non indicato nel protocollo di comunicazione Ping Protocol.

Conclusioni

I risultati delle simulazioni dimostrano che è possibile simulare in modo molto preciso il comportamento del sensore e del segnale acustico in ambiente marino in molte situazioni. Ovviamente il modello si basa su assunzioni che vanno a semplificare l'ambiente simulativo e a presentare differenze dal caso reale, ma con una possibile implementazione dell'acustica in Unity e la gestione del rumore sonoro dato da tutti i materiali è possibile ottenere immagini più veritieri di quanto lo sono già. Come riportato nel capitolo precedente, la miglior rappresentazione avviene tramite il software Ping Viewer ed è consigliato per eventuali simulazioni.

I passi da compiere per migliorare il modello del Ping360 sono quello di aggiustare il suo aspetto fisico e dell'ambiente simulativo, magari introducendo uno studio migliore dei materiali, e a livello di comunicazione passare il pacchetto di bytes che permette la rotazione della visualizzazione in base all'orientazione della prua del ROV.

In un futuro, sarebbe sicuramente interessante poter utilizzare i risultati di una simulazione per confrontarli con quelli sul campo reale per poter classificare i dati derivanti da elementi necessari per la navigazione o l'ispezione, come ad esempio capire quale segnale acustico arriva dall'insonificazione di una barca o da una parete del fondale marino.

Il progetto quindi può essere visto come un supporto per diverse applicazioni, tipo:

- guida e navigazione: la simulazione può essere utilizzata per sviluppare algoritmi di guida avanzati per veicoli sottomarini autonomi basandosi su informazioni noti a priori dell'ambiente;
- obstacle avoidance: generare algoritmi che consentono agli AUV - ROV di rilevare e evitare in modo efficace elementi sul loro percorso;
- ispezione di ambienti e oggetti: utilizzando la simulazione, è possibile sviluppare strategie per intercettare e seguire determinati elementi;
- collaborazione tra veicoli: coordinamento efficace per missioni di ricerca e monitoraggio subacqueo.

Elenco delle figure

1.1	Ping360.	2
1.2	Ping360 e BlueROV2.	2
1.3	Impulso sonoro generato dal sonar.	2
1.4	Rappresentazione di zone insonificate e non.	3
1.5	Rotazione del fascio acustico.	3
1.6	Esempio di scansione polare.	3
1.7	Ambiente simulativo Unity.	5
1.8	Modello Ping360.	5
1.9	Fascio acustico in ambiente simulativo.	5
2.1	Elemento generico contenuto nella variabile <i>distances_matrix</i>	8
3.1	Schema Simulink per l'elaborazione e invio dei dati.	10
3.2	Rappresentazione grafica in MATLAB.	11
3.3	Rappresentazione grafica in PingViewer.	11
3.4	Esempio di colorazione parziale della matrice <i>ColorMap</i>	13
3.5	Porzione di acquisizione dell'ambiente virtuale.	13
3.6	Rappresentazione degli oggetti presenti in figura 3.5 utilizzando l'ambiente MATLAB.	14
3.7	Rappresentazione degli oggetti presenti in figura 3.5 utilizzando il software Ping Viewer.	14
4.1	Finestra di Ping Viewer in cui inserire i parametri di configurazione della comunicazione col sensore.	16
4.2	Formato del messaggio: intestazione, payload e checksum.	17
4.3	Costruzione del payload.	18
4.4	Esempio messaggio di Log.	20
4.5	Documentazione definita in <i>PingProtocol</i> per il messaggio di handshake di bytes che il Ping Viewer invia al sensore.	21
4.6	Array di bytes del messaggio di handshake che invia l'interfaccia Ping Viewer.	21
4.7	Array di bytes del messaggio di handshake che invia il sensore Ping360.	21
5.1	Blocco Simulink UDP Receive per comunicazione con ambiente simulativo.	22
5.2	Sottosistema Simulink UDP per comunicazione con Ping Viewer.	23
6.1	Aereo militare naufragato presente in ambiente virtuale.	25
6.2	Rappresentazione dell'aereo militare naufragato con Ping Viewer.	26
6.3	Rappresentazione dell'aereo militare naufragato con MATLAB.	26
6.4	Statua del discobolo presente in ambiente virtuale.	27
6.5	Rappresentazione del discobolo con Ping Viewer.	27
6.6	Rappresentazione del discobolo con MATLAB.	28
6.7	Oggetto a basso valore di riflettività.	28
6.8	Oggetto ad alto valore di riflettività.	29
6.9	Aeroplano da insonificare su Unity.	29

6.10 Rappresentazione di zone di ombra acustica.	30
6.11 Palafitte in Unity.	30
6.12 Osservazione di un ambiente con palafitte.	31

Elenco delle tabelle

1.1 Elettrico Ping360.	4
1.2 Comunicazione Ping360.	4
1.3 Acustica Ping360.	4
1.4 Meccanica Ping360.	5
1.5 Parametri modificabili per la propria missione.	6
4.1 Parametri scelti per caratterizzare l'intestazione.	17
4.2 Parametri scelti per caratterizzare il payload.	18
5.1 Indirizzi IP e porte.	23

Listings

2.1 Creazione dell'array distances_array_bin.	7
3.1 Divisione delle informazioni ricevute dall'ambiente simulato.	9
3.2 Utilizzo del modello Gaussiano.	11
3.3 Costruzione della variabile <i>data</i>	12

Bibliografia

- [1] BlueRobotics. *Datasheet sensore Ping360*. 2024. URL: <https://bluerobotics.com/store/sonars/imaging-sonars/ping360-sonar-r1-rp/>.
- [2] BlueRobotics. *Installing application PingViewer*. 2024. URL: <https://docs.bluerobotics.com/ping-viewer/#installing-and-running-the-application>.
- [3] BlueRobotics. *PingProtocol*. 2024. URL: <https://docs.bluerobotics.com/ping-protocol/>.
- [4] BlueRobotics. *Scanning Sonar theory*. 2024. URL: <https://bluerobotics.com/learn/understanding-and-using-scanning-sonars/>.
- [5] Woen-Sug Choi et al. “Physics-Based Modelling and Simulation of Multibeam Echosounder Perception for Autonomous Underwater Manipulation”. In: *Frontiers in Robotics and AI* 8 (2021), p. 706646.