UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di laurea triennale

in Ingegneria Informatica e dell'Automazione

Studio e sviluppo di copie digitali di robot sottomarini automatici

Digital Twin systems for marine and underwater robotics

Relatore

Prof.re David Scaradozzi

Correlatore

Ing. Nicolò Ciuccoli

Tesi di laurea di

Angelo D'Agostino

Bonomi

Anno Accademico 2020/2021

Ringraziamenti

Ringrazio il mio relatore David Scaradozzi, il mio correlatore Nicolò Ciuccoli e la Dott.ssa Veronica Bartolucci per i consigli e la disponibilità dimostrati durante lo sviluppo del progetto.

Un particolare ringraziamento va ai miei genitori, che con il loro amore ed il loro supporto mi hanno permesso di raggiungere questo traguardo.

Infine, un ringraziamento va a mio fratello Alessio che mi è sempre stato vicino, soprattutto nei momenti più difficili.

INDICE

Introduzione………………………………………………………………………..1

1. Stato dell'Arte……………………………………………………….………..3

1.1. Sviluppo di Digital Twins per simulazione, analisi e controllo…………...3

1.2. Principali veicoli subacquei……………………………………………….4

1.2.1. Veicoli sottomarini………………………………………………4

1.2.1.1. ROV…………………………………………………4

1.2.1.2 AUV…………………………………………………7

1.2.2. Veicoli sommergibili……………………………………………8

1.2.2.1 DPV ………………………………….....................8

2. Ambiente di lavoro……………………………………………………………11

2.1. Catena di funzionamento…………………………………………………..11

2.2. Software per analisi ed elaborazione dati………………………………….12

2.2.1. Matlab………………………………………………………………12

2.2.1.1. Simulink………………………………………………..12

2.3. Software per il rendering grafico ad alta precisione……………………….13

2.3.1. Unity3D……………………………………………………………..13

2.3.1.1. Interfaccia……………………………………………15

2.4. Risolutore multifisico……………………………………………………...17

2.4.1. AGX Dynamics………………………………………………….17

2.4.1.1. Licenza AGX per Unity……………………………..17

2.5 Comunicazione tra i software……………………………………………18

2.5.1. Protocollo UDP………………………………………………..18

3. Studio di un oggetto semplice…………………………………………...21

3.1. Scelta della Capsula……………………………………………………..21

3.1.1. Modello matematico ……………………………………….22

4. Applicazione e simulazione del modello…………………………………..23

4.1. Sviluppo ambiente marino………………………………………………23

4.1.1. Stage…………………………………………………………...24

4.1.2. AGXunity.WindAndWaterManager……………………………..26

4.1.3. Player…………………………………………………………27

4.1.3.1. Script: PlayerController………………………………..28

4.2. Schema di controllo……………………………………………………..29

4.2.1 Regolatori PID…………………………………………………...31

4.2.1.1. Calibrazione PID……………………………………32

4.2.1.1.1. Prima tecnica di Zigler - Nichols…………….32

4.2.1.1.2. Seconda tecnica di Zigler - Nichols………....34

4.2.2. Implementazione su Simulink……………………………………34

4.3. Implementazione della comunicazione tra Simulink e Unity……………...35

4.4. Presentazione dei casi di studio analizzati…………………………………41

4.4.1. Caso di studio con densità pari a 500………………………….. 42

4.4.2. Caso di studio con densità pari a 1000…………………………..45

5. Conclusioni e sviluppi futuri……………………………………………………48

Bibliografia…………………………………………………………………………49

Introduzione

Lo sviluppo tecnologico che ha caratterizzato questo secolo trova una delle sue massime espressioni nella robotica, sempre più utilizzata in diversi ambiti: uno dei campi di ricerca in cui la robotica ha un ruolo fondamentale è lo sviluppo ed il controllo di veicoli sottomarini privi di equipaggio. Questi veicoli consentono l'ispezione di piattaforme petrolifere, condutture e tubazione, precedentemente effettuate dai sommozzatori; permettono inoltre piccoli lavori di costruzione, come apertura e chiusura di valvole, e possono essere utilizzati anche per scansionare il fondale marino.

Per lo sviluppo di tali veicoli è necessario fare dei test in ambienti sottomarini ma spesso fare questi test risulta difficoltoso a causa della disponibilità delle risorse. Per far fronte a questo problema negli ultimi anni sono stati sviluppati software in grado di simulare la realtà permettendo così lo sviluppo di una copia virtuale del modello reale al fine di analizzarne il comportamento. Questa copia che prende il nome di Gemello digitale, o Digital Twin, consente di ottimizzare la sperimentazione eliminando tutti gli errori che potrebbero danneggiare il sistema fisico in fase di test, fornisce inoltre la possibilità di ripetizione dei test in maniera semplice e veloce.

L'obiettivo di questo elaborato è quello di creare un gemello digitale di un oggetto semplice, una capsula, mediante il software di rendering grafico Unity3D e utilizzando il risolutore multifisico AGX per Unity per simularne il comportamento in acqua; verranno inoltre utilizzati i software di analisi ed elaborazioni di dati Matlab e Simulink per implementare un sistema di controllo tramite PID.

Nella prima parte dell'elaborato, si può trovare un'introduzione al problema dei gemelli digitali per il controllo e una panoramica sui principali veicoli sottomarini e sulle loro caratteristiche.

Successivamente, nel secondo capitolo, viene descritto il processo di interazione tra i diversi software; vengono inoltre descritti i diversi software illustrandone le principali caratteristiche e le motivazioni che hanno portato alla loro scelta. In questo capitolo si tratta anche del protocollo UDP per la trasmissione di dati tra i diversi software.

Nel terzo capitolo, vengono introdotte le motivazioni che hanno portato alla scelta della capsula come oggetto da simulare e ne viene introdotto il modello matematico.

Nel quarto capitolo, è spiegato in maniera dettagliata come è stato costruito l'ambiente marino su Unity3D, come si è implementato lo schema di controllo utilizzando Simulink con una panoramica sull'utilizzo dei regolatori PID e come è stato utilizzato il protocollo UDP per permettere la trasmissione di dati da Simulink a Unity3D e viceversa. Successivamente vengono illustrati due casi di studio: il primo considerando che il corpo abbia densità pari a 1000 ed il secondo considerando che il corpo abbia densità pari a 500: per entrambi i casi verranno mostrati diversi grafici del comportamento del corpo.

Nelle conclusioni verranno confrontati gli obiettivi di questo elaborato con i risultati ottenuti. Successivamente verranno proposti degli sviluppi futuri.

1. STATO DELL’ARTE

1.1. Sviluppo di Digital Twins per simulazione, analisi e controllo

Il concetto di Digital Twin è stato presentato per la prima volta dal Dr. Michael Grieves dell’Università del Michigan che, durante una conferenza di Product Lifecycle Management nel 2003, descrive un Gemello digitale come la copia virtuale di un prodotto fisico. Come viene descritto nell’articolo [1] non c’è una definizione univoca del concetto di Digital Twin ma tra tutte se ne distinguono quattro:

- “*Un Digital Twin è una simulazione probabilistica, multiscala e multifisica di un sistema, che utilizza i migliori modelli matematici e le migliori tecnologie disponibili per replicare la vita del gemello reale*”. Glaessgen e Stargel, 2012.
- “*Un Digital Twin è un insieme di costrutti di informazioni virtuali che descrivono completamente un prodotto fisico dal livello microscopico al livello macroscopico*”. Grieves e Vickers, 2017.
- “*Un Digital Twin è composto da tre parti: un prodotto fisico nello spazio reale, un prodotto virtuale nello spazio virtuale, la comunicazione bidirezionale di dati e informazioni che lega i due prodotti insieme*”. Grieves, 2014.
- “*Un Digital Twin è composto da tre parti: prodotto fisico, prodotto virtuale e una connessione che li lega insieme. Le caratteristiche che un Digital Twin deve avere sono: replica in tempo reale e lo spazio virtuale deve mantenere un alto livello di sincronizzazione e fedeltà allo spazio reale*”. Tao et al, 2018.

Pur essendo diverse tra di loro, è possibile identificare un tratto comune nelle diverse definizioni, infatti tutte fanno riferimento alla necessità di integrare diverse fonti di dati per permettere una rappresentazione virtuale dell’oggetto fisico o del processo per tutta la durata del suo ciclo vitale.

Nell’ambito della robotica marina i Digital Twins vengono utilizzati al fine di analizzarne e predire il comportamento per rifinire il controllo o per ottimizzare il sistema. In particolar modo è possibile usare un Digital Twin per studiare la correttezza dei programmi da caricare sul veicolo, per evitare errori che danneggerebbero il sistema fisico, e per la diagnostica di eventuali anomalie: dato che modello funziona in parallelo

con il sistema reale, se quest'ultimo si discosta troppo dalla simulazione significa che c'è un errore

1.2. Principali veicoli subacquei

I veicoli subacquei sono quei mezzi concepiti per operare in immersione. Esistono principalmente due categorie in cui suddividere i veicoli subacquei in base al loro modo di operare:

- Sono detti sottomarini quei veicoli le cui prestazioni in immersione sono superiori a quelle in emersione, sono progettati per operare unicamente e in modo prolungato in immersione.
- Sono detti sommergibili quei veicoli le cui prestazioni in emersione sono superiori a quelle in immersione, possono operare sia in immersione che in emersione

1.2.1. Veicoli sottomarini

1.2.1.1. ROV

Il ROV (Remotely Operated Vehicle) è un tipo di veicolo sottomarino controllato da remoto che permette l'esplorazione degli oceani senza la necessità di impiegare sommozzatori. Questi veicoli sono controllati da delle stazioni in superficie, che possono essere fisse o mobili, mediante un collegamento effettuato utilizzando un cavo ombelicale, detto anche tether, che permette lo scambio di dati tra il ROV e la stazione.

Tipicamente i ROV sono dotati di videocamere analogiche, per l'acquisizione di video e foto ad alta risoluzione, di luci e laser per la quantificazione della dimensione degli organismi o delle morfologie ispezionate. Il sistema di propulsione può essere elettrico o idraulico, in entrambi i casi sono permessi movimenti in verticale, per poter scendere in profondità o risalire in superficie, e in orizzontale, per potersi spostare.

È possibile suddividere i ROV in cinque differenti classi riconosciute dall'IMCA (*International Marine Contractors Association)*:

- **Classe I-ROV da osservazione:** dispositivi dalle dimensioni ridotte equipaggiate unicamente con una telecamera, delle luci e in alcuni casi anche un sonar. Sono utilizzati solamente per l'ispezione.

Figura 1: ROV da osservazione

- **Classe II- ROV da osservazione con Payload:** dispositivi dalle medie dimensioni, oltre a essere equipaggiati con telecamere e sonar sono dotati anche di un manipolatore

Figura 2: ROV da osservazione con Payload

- **Classe III- ROV da lavoro:** dispositivi dalle elevate dimensioni, i veicoli di questa classe sono abbastanza grandi da poter essere equipaggiati con diversi sensori e grandi manipolatori. La presenza di sensori e strumenti permette di poter lavorare in un certo range di autonomia. Possono lavorare a profondità molto elevate.

Figura 3: ROV da lavoro

- **Classe IV- veicoli trainanti e cingolati:** detti anche ROV trencher, sono dispositivi dalle elevate dimensioni e dal notevole peso, hanno poca libertà di manovra e si muovono sul fondale marino utilizzando dei cingoli. Solitamente sono progettati per uno specifico scopo, ad esempio per l'interramento dei cavi.

Figura 4: ROV trencher

- **Classe V:** i veicoli in questa classe sono ancora in fase di sviluppo o sono ancora dei prototipi. Sono inoltre, in questa classe tutti quei veicoli che non rientrano nelle classi precedenti. In questa classe sono inclusi anche gli AUV.

Figura 5: Veicolo di classe V

1.2.1.2. AUV

Un AUV (Autonomous Underwater Vehicle) è un tipo di veicolo sottomarino in grado di operare in maniera autonoma. Al contrario del ROV, che sono collegati ad una stazione in superficie, l'AUV non presenta collegamenti fisici con un operatore ma sono controllati da dei programmi. Gli AUV sono equipaggiati con attrezzature destinate all'osservazione e l'analisi come telecamere, sonar e sensori di profondità. Non essendo direttamente collegati con un operatore gli AUV non trasmettono dati in tempo reale ma hanno bisogno di un'unità di memoria in cui salvarli. Essendo veicoli autonomi sono alimentati con batterie a ioni di litio ricaricabili, solitamente ne montano due, una principale ed una secondaria.

Come descritto nel testo [2], tipicamente gli AUV si muovono planando lungo delle traiettorie lineari segmentate che giacciono prevalentemente sul piano orizzontale. Tuttavia, per alcune specifiche applicazioni, come nel raccoglimento di dati ad alte profondità, è possibile sfruttare il movimento sul piano verticale.

Figura 6: AUV

Le possibili applicazioni di un AUV sono molteplici: sono molto utilizzati per i rilievi idrografici in quanto grazie alla loro mobilità possono accedere ad aree difficili da raggiungere per altri veicoli, possono essere usati anche per la scoperta di depositi sottomarini di petrolio o gas, si utilizzano per la manutenzione degli scafi delle navi e grazie alle più recenti scoperte ultimamente si utilizzano anche in ambito militare eliminando i rischi di utilizzare veicoli con equipaggio.

1.2.2 Veicoli sommergibili

1.2.2.1 DPV

Il DPV (Diver Propulsion Vehicle), detto anche scooter subacqueo, è un tipo di veicolo sommergibile utilizzato dai sommozzatori durante le immersioni per spostamenti a lungo raggio. Tipicamente sono composti da una struttura resistente alla pressione ed impermeabile in cui si trova un motore elettrico alimentato a batteria che aziona un'elica. I DPV sono strutturati in modo da avere un assetto neutro quando viene immerso, ovvero se viene lasciato in immersione esso tende a rimanere nella stessa posizione.

È possibile suddividere i DPV in quattro classi, due di esse sono per i DPV ad uso ricreativo, le altre due sono per i DPV ad uso militare:

- **Manta Board:** è la forma più semplice di DPV, non è dotato di motore, pertanto, si utilizza collegandolo, per mezzo di corde, ad una barca in superficie. La struttura dei Manta board è molto semplice, è composto da due tavole inclinabili collegate tra di loro (che ricordano le ali di una manta). Il sommozzatore si mantiene a queste tavole e viene trascinato dalla barca, inclinando le tavole può immergersi o risalire. Questi DPV sono utilizzati principalmente a scopo ricreativo essendo molto semplici e disponibili a basso costo.

Figura 7: Manta Board DPV

- **Scooters:** è il tipo più comune di DPV, consiste in una struttura impermeabile a forma di siluro al cui interno è presente un motore elettrico e una batteria che azionano un'elica. La struttura inoltre è pensata in modo da proteggere l'elica in modo da non essere pericolosa per il subacqueo. Il principale vantaggio di questi veicoli sta nella loro dimensione ridotta, nella loro maneggevolezza e nel fatto che il subacqueo viene trascinato in modo da porre poca resistenza all'acqua consumando meno batteria possibile.

Figura 8: Scooter DPV

- **Siluri subacquei:** Questo tipo di DVP è diventato popolare soprattutto in Italia, durante la Seconda Guerra Mondiale in quanto permettevano il trasporto di due militari contemporaneamente. I siluri al giorno d'oggi sono stati migliorati notevolmente. Tipicamente permettono il trasporto di due persone equipaggiate con attrezzature da subacquei, i sommozzatori possono sedersi direttamente sul siluro non dovendo rimanere attaccati e farsi trascinare. I modelli più recenti sono dotati di GPS e sonar.

Figura 9:Militari a bordo di un Siluro subacqueo

- **Subskimmers:** Questo tipo di DPV viene utilizzato prevalentemente in abito militare a causa delle sue grandi dimensioni e della sua struttura. Consiste in una barca gonfiabile che si può immergere in acqua, fino a che è in superficie il subskimmer viene alimentato a gasolio, mentre quando si trova sott'acqua il motore a gasolio viene spento e si passa ad un'alimentazione a batterie.

Figura 10: Subskimmer DPV

2. Ambiente di lavoro

2.1. Catena di funzionamento

Per l’esecuzione del progetto sono stati utilizzati diversi software che devono interagire tra di loro scambiandosi informazioni in tempo reale. Di seguito è proposto uno schema per illustrare come i diversi software sono connessi per comprendere al meglio l’impiego delle tecnologie utilizzate per lo sviluppo del Digital Twin.

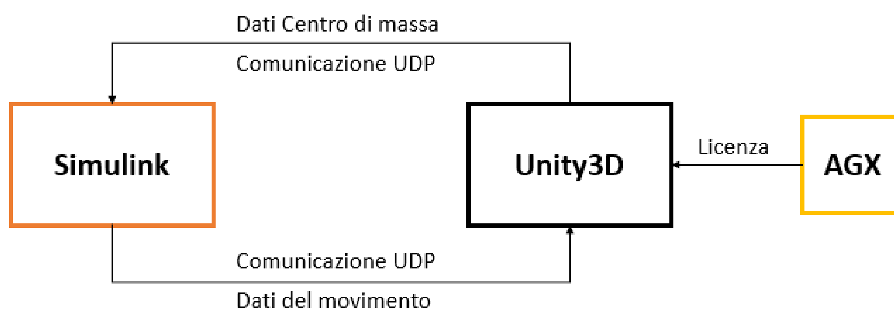

Figura 11: Schema di interazione dei software

Unity3D, l’ambiente di rendering grafico in cui è stato modellato l’ambiente in cui si studia il comportamento della capsula, sfrutta il risolutore multifisico AGX, utilizzando una licenza fornita da quest’ultima, per simulare la fisica dell’acqua; successivamente utilizza il protocollo UDP per inviare a Simulink i dati relativi al centro di massa. Simulink elabora i dati ricevuti e utilizzando anch’esso il protocollo UDP invia a Unity3D i dati relativi al movimento che la capsula deve compiere. Il ciclo di comunicazione tra i due software si ripete fino a che la capsula non raggiunge il punto designato.

2.2 Software per l’analisi e l’elaborazione di dati

2.2.1. Matlab

Matlab è un ambiente di calcolo numerico e l’analisi statistica scritto in C creato dalla MathWorks. Esso è uno dei software più utilizzati per l’analisi dei dati, lo sviluppo di algoritmi e la creazione di modelli e simulazioni, questo anche grazie ai numerosi toolbox messi a disposizione dell’utente [3].

Come suggerisce il nome, infatti Matlab è l’abbreviazione di Matrix Laboratory, l’elemento fondamentale di Matlab è la matrice; ogni dato dichiarato costituisce una matrice. Matlab inoltre mette a disposizioni diverse funzioni per la manipolazione delle matrici, risultando molto utile per la gestione di notevoli quantità di dati contemporaneamente.

Tra i diversi toolbox messi a disposizione da Matlab i più utilizzati per la modellazione e la simulazione dei sistemi sono il System Identification Toolbox, utilizzato per ottenere un modello matematico da un sistema utilizzando l’approccio black-box, e Simulink, principalmente usato per la simulazione dei sistemi.

Per implementare il controllo del Digital Twin sviluppato per questo progetto è stato utilizzato il toolbox Simulink.

2.2.1.1 Simulink

Simulink è un toolbox creato utilizzando i comandi di Matlab utilizzato per la modellazione e la simulazione dei sistemi senza l’utilizzo di codice. Tra i principali vantaggi dell’utilizzo di Simulink si trovano: l’interfaccia grafica semplice ed intuitiva, la possibilità di utilizzare dei blocchi predefiniti da connettere tra di loro, un’elevata flessibilità nella variazione di un progetto e la riduzione di tempo e costo rispetto ad un test pratico.

L’interfaccia di Simulink si presenta suddivisa in due sezioni principali: il workspace e il Simulink Library Browser. Nel Simulink Library Browser è possibile trovare i blocchi elementari utilizzabili nel progetto mentre il workspace è dove si costruisce il progetto interconnettendo i blocchi. Il Simulink Library Browser è una schermata di sola lettura, infatti per poter modificare il valore dei blocchi bisogna importarli nel workspace, una

volta importato un blocco nel workspace cliccando su di esso è possibile aprire una finestra in cui si possono impostare i valori dei parametri che lo caratterizzano.

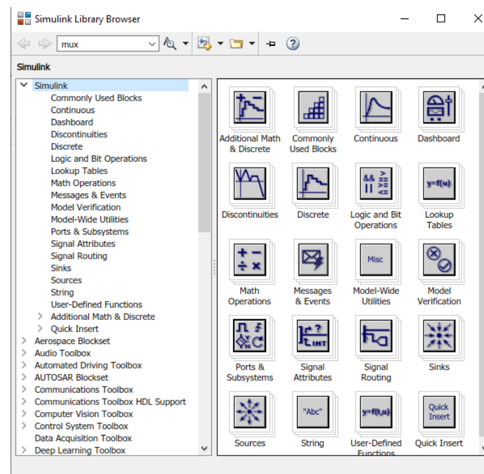

Figura 12: Simulink Library Browser

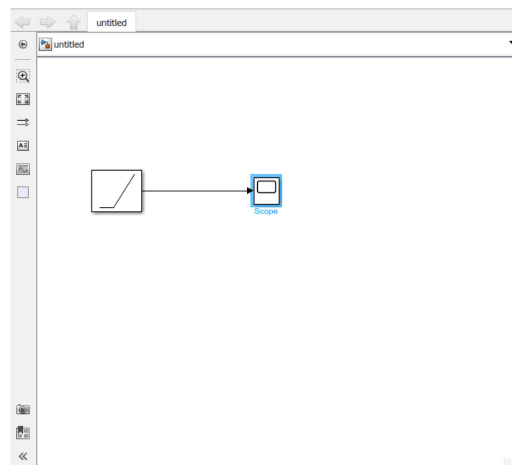

Figura 13: Workspace di Simulink

2.3 Software per il Rendering grafico ad alta precisione

2.3.1. Unity3D

Unity3D è un motore grafico multipiattaforma che consente lo sviluppo di videogiochi e contenuti interattivi come animazioni 3D e visualizzazioni architettoniche. L'ambiente di sviluppo Unity è disponibile per Windows, MacOS e Linux ed i giochi prodotti possono essere eseguiti su diversi dispositivi, quali PC, Xbox 360, Xbox One, Playstation 3 e 4, Playstation Vita, Wii U, Nintendo Switch, iPhone, iPad e dispositivi Android.

All’interno dell’ambiente di sviluppo si possono fornire le definizioni di alcuni elementi fondamentali:

- Scena: è un livello del gioco, più scene compongono il mondo di gioco. Il passaggio da una scena all’altra elimina tutti gli elementi presenti in quella scena e carica tutti i successivi.
- gameObject: ogni elemento in una scena, compresi luci e telecamera, prende il nome di gameObject. È possibile installare diversi componenti su un gameObject per attribuirgli delle proprietà.
- Componenti: caratteristiche che si possono assegnare ad ogni gameObject per dargli delle proprietà. Di base su ogni gameObject è preinstallato il componente Transform, che rappresenta tutte le trasformazioni 3D (scala, rotazione, posizione). È possibile assegnare come componente anche uno script.

Insieme a Unity viene installato anche Visual Studio, un ambiente di sviluppo integrato sviluppato dalla Microsoft. Per mezzo di Visual Studio è permessa la programmazione e la scrittura di script, dei file in C# che possono essere assegnati ad un oggetto per fargli compiere delle azioni. Di base ogni volta che viene creato uno script esso avrà due metodi predefiniti che vengono eseguiti automaticamente:

- Start(): viene eseguito all’apertura della scena e solitamente si utilizza per inizializzare le variabili dell’oggetto.
- Update(): viene eseguito ad ogni frame e permette di aggiornare i valori delle variabili dell’oggetto in base alle sue interazioni con la scena.

2.3.1.1. *Interfaccia*

L'interfaccia di Unity può essere suddivisa in cinque sezioni [4]: Hierarchy, Scene, Game, Project, Inspector.

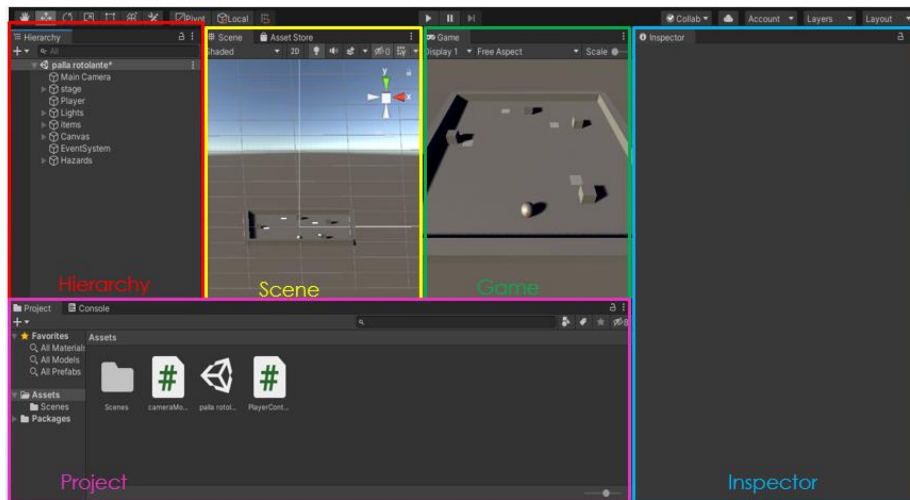

Figura 14: Suddivisone in sezioni dell'interfaccia di Unity

- *Hierarchy*: in questa sezione sono contenuti tutti i gameObject presenti nella scena corrente, si può utilizzare questo pannello per aggiungere o rimuovere gameObject alla scena. La finestra di hierarchy può essere utilizzata anche per organizzare e raggruppare i gameObject sfruttando il Parenting. Unity sfrutta il parenting per raggruppare gli oggetti secondo una gerarchia Genitore-Figlio, un gameObject figlio eredita tutti i componenti assegnati al genitore inoltre è possibile modificare delle caratteristiche del genitore per modificare anche tutti i figli, ad esempio raddoppiando la scala del genitore raddoppierà anche quella di tutti i figli.
- *Scene*: rappresenta la finestra in cui compare la scena attuale. È una delle schermate più utilizzate in quanto in essa si costruisce la scena di gioco agendo direttamente sui gameObject per modificarne la posizione, la rotazione o le dimensioni. Per facilitare la costruzione del progetto nella scena sono presenti una griglia e degli assi cartesiani che permettono di orientarsi nel posizionamento di un oggetto.
- *Game*: è una schermata che mostra l'output delle telecamere presenti nella scena, essendo una schermata di solo output non è possibile modificare il progetto in

alcun modo da questa finestra. Tuttavia, la funzione principale di questa schermata è testare il gioco prima di buildarlo per vedere se ci sono eventuali errori, sia di programmazione che di costruzione, da correggere.

- *Inspector*: come suggerisce il nome questa è la schermata di ispezione di un oggetto, su questa schermata compaiono tutte le proprietà di un oggetto selezionato e tutti i componenti che gli sono stati assegnati. Da questa schermata è possibile modificare le proprietà di un oggetto, come ad esempio la posizione, in maniera precisa potendo inserire i valori desiderati, è possibile anche assegnare nuovi componenti ad un oggetto, attivarlo o disattivarlo, assegnarli un tag e selezionare il layer in cui deve stare l'oggetto. Tramite la schermata inspector è possibile ispezionare e modificare non solo i gameObject ma anche gli assets.
- *Project*: la schermata di project contiene tutti i file che vengono utilizzati nel progetto, sono in questa schermata tutte le textures, i suoni, i modelli e gli script. Ad affiancare la schermata di project c'è il pannello Console, in esso vengono stampati tutti gli output testuali degli script; inoltre, su questa schermata vengono visualizzati anche gli errori.

2.4. Risolutore multifisico

2.4.1. AGX Dynamics

AGX è un SDK (Software Development Kit) per la modellazione e la simulazione di sistemi meccanici complessi in movimento. AGX può modellare facilmente robot, veicoli pesanti e veicoli da lavoro, come escavatrici e gru, ma può anche modellare i materiali con cui questi oggetti interagiscono come la terra o nel caso di questo progetto l'acqua [5].

I moduli che mette a disposizione permettono di creare progetti complessi senza dover semplificare eccessivamente i modelli sei sistemi fisici. In particolare, per lo sviluppo di questo progetto è stato utilizzato il modulo *AGX Hydrodynamycs/Wind* che calcola tutte le forze a cui un corpo è sottoposto se si trova in acqua o in aria. Tra i vantaggi di questo modulo ci sono la possibilità di calcolare le spinte di drag, lift e di calcolare la galleggiabilità, la possibilità di definire le correnti marine o aeree da parte dell'utente e che questo modulo lavora direttamente sulle geometrie del corpo in modo da avere forze accurate.

AGX è disponibile in C++, C# e Python su Windows, macOS e Linux ma mette a disposizione delle licenze per interfacciarsi con dei motori grafici come Unity3D e Unreal Engine.

2.4.1.1. Licenza AGX per Unity

AGX Dynamics for Unity nasce dall'idea di fornire agli utenti un unico servizio che combina le tecnologie più avanzate di rendering grafico fornite da Unity con la possibilità di simulare sistemi complessi data da AGX, infatti mediante l'estensione AGX Dynamics for Unity si riesce ad ottenere un simulatore che riesce ad utilizzare la fisica reale dei corpi.

Uno dei componenti forniti da AGX Unity più utilizzati è AGX.RigidBody che va a sostituire il componente Rigidbody fornito da Unity. Mediante AGX.RigidBody è possibile controllare ed assegnare valori alle caratteristiche fisiche dei corpi, come ad esempio la massa, scegliere il MotionControl del corpo (ovvero cosa lo fa muovere) e applicargli forze e torsioni. Si preferisce utilizzare il Rigidbody fornito da AGX rispetto a quello di Unity per la precisione con cui vengono calcolate e applicate le forze sul corpo.

Tra i vari servizi messi a disposizione dalla licenza AGX c'è la possibilità di utilizzare i componenti che si trovano sotto la voce *Shapes* (*Box, Capsule, Cone, Cylinder, Height Field, HollowCone, HollowCylinder, Mesh, Plane, Sphere*) per creare delle rappresentazioni 3D delle principali forme su cui AGX può lavorare facilmente.

Per creare un modello 3D basta assegnare il componente al gameObject e selezionare la voce: *Create a visual representation for the physical shape*; la schermata che si aprirà permetterà di scegliere il nome della rappresentazione e il materiale da assegnargli. Una volta cliccato su "Create" verrà la visualizzazione verrà creata sottoforma di oggetto figlio dell'oggetto a cui è stato assegnato il componente. Utilizzando la voce Half Extents è possibile selezionare le dimensioni dell'oggetto (per le visualizzazioni di AGX non è possibile utilizzare la voce *Scale* del componente *Transform*), mentre, mediante la voce Collision Enabled si può attivare e disattivare il collider.

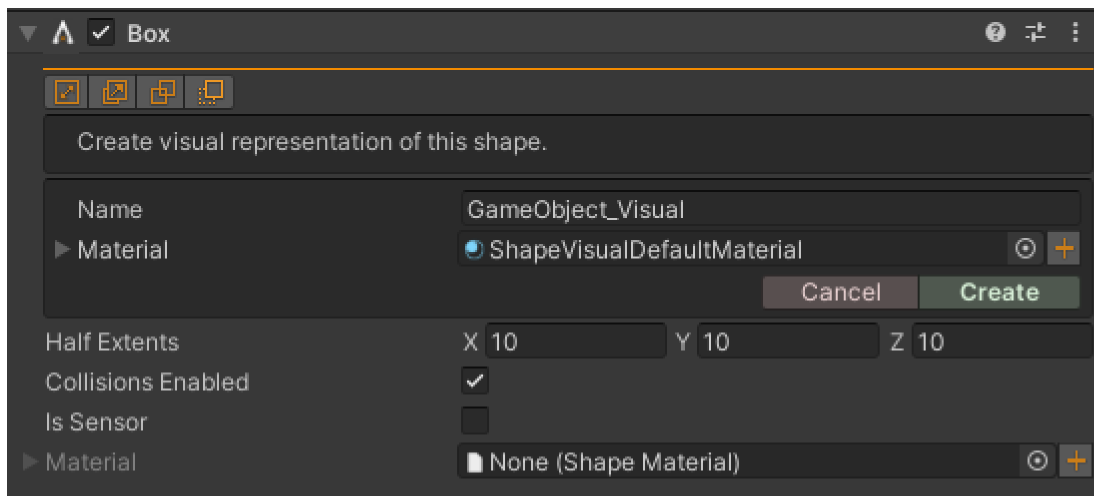

Figura 15: Componente Box di AGX

2.5. Comunicazione tra i software

2.5.1. Protocollo UDP

UDP (User Datagaram Protocol) è un protocollo di trasporto dati nelle reti IP non orientato alla connessione. È un tipo di protocollo molto semplice, infatti, come descritto nel testo [6] esso fornisce una procedura ad un programma di inviare dati ad un altro programma utilizzando in minima parte il meccanismo del protocollo.

Le caratteristiche principali dell'UDP sono:

- **Assenza di connessione**: il trasporto di dati avviene senza stabilire una connessione tra mittente e destinatario, ovvero è possibile inviare i dati specificando solamente una porta di destinazione ed un indirizzo IP senza necessità di ricevere una conferma di ricezione da parte del destinatario.
- **Utilizzo delle porte**: Il protocollo UDP fa utilizzo delle porte per trasferire i pacchetti di dati ai riceventi. Le porte sono definite utilizzando un modello a numeri in cui le porte dalla 0 alla 1023 sono destinate a dei servizi fissi.
- **Comunicazione rapida**: grazie alla configurazione senza connessione è garantita una trasmissione rapida dei dati in quanto questo protocollo non gestisce la suddivisone in segmenti ed il loro riordinamento. La sua rapidità lo rende ideale per applicazioni time-sensitive o per la trasmissione di audio e video in streaming in tempo reale
- **Possibile inaffidabilità**: l'implementazione particolarmente semplice e delle ridotte dimensioni delle PDU del protocollo UDP lo rendono particolarmente rapido a discapito tuttavia della sua affidabilità. UDP, infatti, è uno di quei protocolli definiti inaffidabili in quanto, pur contenendo dei codici di identificazione degli errori, non effettua nessuna gestione o correzione degli errori. Dunque, nonostante si guadagni in velocità si perde in qualità poiché a differenza di altri protocolli non garantisce l'integrità dei dati ricevuti.

La Protocol Data Unit (PDU), ovvero l'unità di informazione, del protocollo UDP si chiama datagramma, ogni datagramma è diviso in due parti: *header* e *body*. Il body è la parte del datagramma che contiene il messaggio vero e proprio da inviare al processo destinatario. L'header è quella parte del datagramma che contiene le informazioni di controllo necessarie al funzionamento della rete. Normalmente lo si posiziona all'inizio del datagramma ma in alcuni casi può essere posizionato anche in coda, in questo caso prende il nome di trailer. La lunghezza in bit dell'header è fissa e ha una struttura molto semplice:

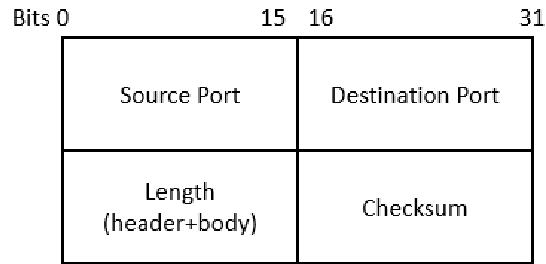

Figura 16: Struttura dell'header

Nello specifico, con *Source port* e *Destination port* si intendono rispettivamente il numero di porta del processo sorgente ed il numero di porta del processo di destinazione, *Length* rappresenta la lunghezza totale del datagramma (header+body) espressa in byte, la lunghezza massima di un datagramma è di 65536 byte compreso l'header; e *Checksum* è un numero di 16 bit, calcolato utilizzando degli specifici algoritmi, che indica l'integrità del datagramma ricevuto.

3. Studio di un oggetto semplice

3.1. Scelta della capsula

Come precedentemente introdotto, l'obiettivo di questa tesi è quello di sviluppare un Digital Twin di un oggetto semplice ideale, di cui si conoscono tutte le caratteristiche, al fine di studiarne il comportamento in acqua per sviluppare e migliorare delle strategie di controllo per veicoli subacquei reali più complessi.

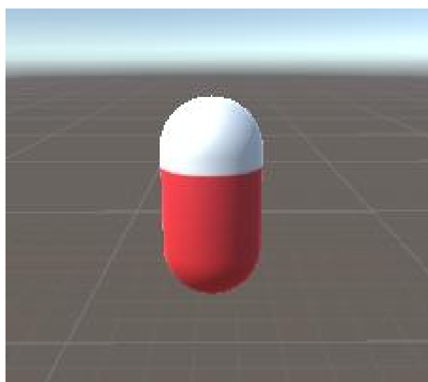

Figura 17: Modello 3D di una capsula

Per lo sviluppo di questo progetto è stato scelto come oggetto da modellare una capsula, una figura tridimensionale composta dall'unione di un cilindro di altezza h e due semisfere di raggio r. La scelta della capsula è stata fatta considerando che questa forma, che ricorda quella di un siluro, approssima la forma di diversi veicoli sottomarini, tra cui gli AUV e diversi tipi di DPV.

Successivamente verranno proposti due casi di studio: il primo si ha quando la densità della capsula è uguale a 500, mentre per il secondo la densità del corpo è stata impostata a 1000. In entrambi i casi sono state fatte due assunzioni sulle caratteristiche del corpo: l'altezza h ed il raggio r sono legati dalla relazione $h = 2r$ e la massa del corpo è di 1 kg.

3.1.1. Modello matematico

La scelta dello studio del comportamento di un corpo ideale è dovuta alla possibilità di conoscere completamente il modello matematico permettendo di controllare il corpo con precisione senza dover stimare un modello.

In generale la capsula risponde alla seconda legge di Newton, $\vec{F} = m\vec{a}$, a cui bisogna aggiungere la forza di attrito viscoso $\vec{A} = \eta\vec{v}$. Pertanto, la legge che regola il moto del corpo è:

$$\vec{F} - \vec{A} = m\vec{a} \Leftrightarrow \vec{F} = m\vec{a} + \eta\vec{v}$$

In cui:

- $\vec{F}$: forza applicata
- $\vec{A}$: forza di attrito viscoso dinamico
- m : massa del corpo
- $\vec{a}$: accelerazione del corpo
- η : coefficiente di attrito viscoso
- $\vec{v}$: velocità del corpo

Considerando che $a = \frac{d^2s}{dt^2}$ e $v = \frac{ds}{dt}$ è possibile esprimere tutto in funzione dello spazio come:

$$\vec{F} = m\frac{d^2\vec{s}}{dt^2} + \eta\frac{d\vec{s}}{dt}$$

Nello specifico, in questo progetto è stato controllato il movimento del corpo considerando solamente lo spostamento lungo l'asse z, pertanto vale la relazione $dx = dy = 0$, da cui possiamo ricavare che $d\vec{s} = dx\,\hat{\imath} + dy\,\hat{\jmath} + dz\,\hat{k} = dz\,\hat{k}$. Utilizzando questi dati possiamo riscrivere l'equazione nel seguente modo:

$$\vec{F} = m\frac{d^2z}{dt^2}\hat{k} + \eta\frac{d\,z}{dt}\hat{k}$$

4. Applicazione e simulazione del modello

4.1. Sviluppo ambiente marino

Nella creazione dell'ambiente di simulazione è stata utilizzata la licenza AGX for Unity per cui tutti i componenti che verranno nominati, eccetto quelli per la telecamera, le luci e gli scripts, sono forniti da AGX.

Il progetto è formato da un'unica scena in cui sono presenti sei *gameObject*: *Main Camera, Directional Light, Player, Stage, AGXunity.WindAndWaterManager* e *UDP Control*; di questi sei *Main Camera* e *Directional lights* sono oggetti utilizzati da Unity per la visualizzazione mentre, *Player, Stage, AGXunity.WindAndWaterManager* e *UDP Control* sono gli oggetti che rappresentano l'ambiente vero e proprio.

In questa sezione verranno analizzati i gameObject sopracitati ponendo particolare riferimento a *Player, Stage* e *AGXunity.WindAndWaterManager*; un'analisi più approfondita di *UDP Control* verrà invece inclusa nella sezione **4.3.**

- **Main Camera**: gameObject che rappresenta la telecamera, ovvero l'oggetto che permette di visualizzare la scena in Play mode. È dotato di tre componenti standard: *Camera*, che permette la cattura delle immagini, *Audio Listener*, che permette la cattura dei suoni e *Transform*. Per far sì che venisse incluso l'intero stage la Main camera ha le seguenti caratteristiche: *Position* $(0, 45, -15)$, *Rotation* $(45, 0, 0)$, *Scale* $(1, 1, 1)$.
- **Directional Light**: gameObject che rappresenta la fonte di luce che illumina lo stage. È dotato di due componenti standard: *Transform* e *Light.*
- **UDP Control**: è il gameObject che si occupa dell'utilizzo del protocollo UDP per far comunicare Unity con Simulink, è dotato di tra script: *UDP Transmitter, UDP Receiver* e *UDP Controller*. Questi tre script verranno approfonditi in **4.3.**

4.1.1. Stage

La struttura dello stage è molto semplice: è una vasca rettangolare di dimensioni (*lunghezza*, *larghezza*, *profondità*) pari a $60m \times 24m \times 22m$ con al suo interno un blocco cui sono state assegnate le proprietà dell'acqua.

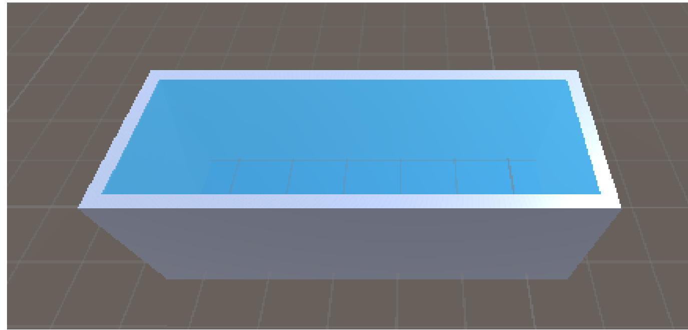

Figura 18: Stage - vista laterale

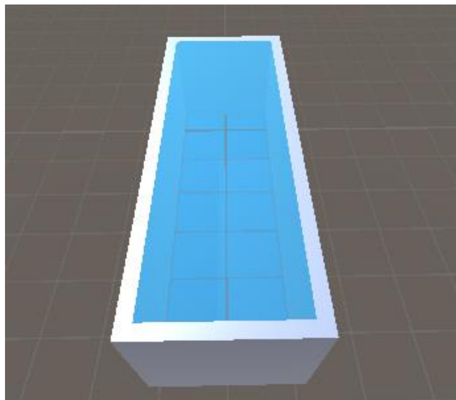

Figura 19: Stage - Vista dall'alto

Per la realizzazione dello stage è stato utilizzato un gameObject vuoto chiamato *Stage*, questo oggetto funge da genitore per tutti i gameObject che compongono lo stage, che erediteranno i componenti ad esso assegnati.

Nello specifico a questo gameObject ha due componenti: *Transform* e *Rigidbody*; dal primo si è selezionata la posizione di riferimento per tutti gli oggetti figli, ovvero $(0, 0, 25)$, mentre mediante il secondo si tratta lo stage come un oggetto fisico. Per evitare che all'avvio lo stage sia sottoposto alla forza di gravità e continui a sprofondare bisogna assegnare al parametro *Motion Control* la voce *STATIC,* in questo modo lo stage potrà muoversi.

Per la composizione della vasca sono stati utilizzati cinque gameObject, uno per il pavimento e quattro per i muri, di seguito verranno elencati gli oggetti utilizzati e le loro

caratteristiche principali. Tutte le posizioni sono calcolate considerando come origine la posizione del gameObject genitore, pertanto, è possibile ricondurle alle coordinate globali utilizzando la relazione: $Posizione_{globale\ figlio} = posizione_{genitore} + posizone_{locale\ figlio}$.

- *Floor*: gameObject utilizzato come pavimento della vasca, a questo oggetto sono stati assegnati due componenti: *Transform* e *Box*. *Floor* si trova nella posizione locale $(0, -1,\ 0)$, mentre, le sue dimensioni sono state impostate dall' *Half Extents* del componente *Box* come: $(10, 1\ , 30)$.
- *Muri*: per la creazione dei muri sono stati utilizzati quattro gameObject, rispettivamente chiamati *Wall_1*, *Wall_2*, *Wall_3* e *Wall_4*. Tutti e quattro i gameObject hanno solamente due componenti installati: *Transform* e *Box*. Per quanto riguarda le dimensioni e la posizione dei muri esse sono:
  - *Wall_1:* La posizione locale è $(11,\ 9\ ,\ 0)$, la rotazione è di $(0, 90,\ 0)$, mentre Half Extents è di $(30,\ 11,\ 1)$.
  - *Wall_2:* La posizione locale è $(-11,\ 9\ ,\ 0)$, la rotazione è di $(0, 90,\ 0)$, mentre Half Extents è di $(30,\ 11,\ 1)$.
  - *Wall_3:* La posizione locale è $(0,\ 9\ , -31)$, la rotazione è di $(0,\ 0,\ 0)$, mentre Half Extents è di $(12,\ 11,\ 1)$.
  - *Wall_4:* La posizione locale è $(0,\ 9\ ,\ 31)$, la rotazione è di $(0,\ 0,\ 0)$, mentre Half Extents è di $(12,\ 11,\ 1)$.

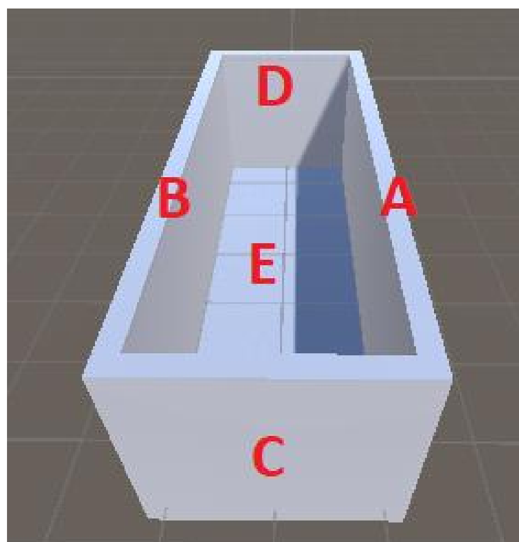

Figura 20: Stage- Vasca. (A) Wall_1, (B) Wall_2, (C) Wall_3, (D) Wall_4 e (E) Floor

Per l'acqua invece è stato creato un gameObject chiamato *Pool* a cui sono stati assegnati due componenti: *Transform* e *Box*. Mediante il componente *Transform* si impostata la posizione $(0, 10, 0)$, mentre, con *Box* si sono impostate le dimensioni $(10, 10, 30)$ e si è selezionato il materiale di render: è stato utilizzato *WaterMaterial*, fornito da AGX. Per l'implementazione della fisica dell'acqua è stato utilizzato *AGXunity.WindAndWaterManager*.

4.1.2. AGXunity.WindAndWaterManager

Il manager di Acqua e Vento è un'istanza di un oggetto che deve essere aggiunto alla scena per abilitare l'idrodinamica o l'aereodinamica di un oggetto [7]. Per aggiungere questo manager bisogna cliccare su: *AGXUnity → Managers→ Wind.*

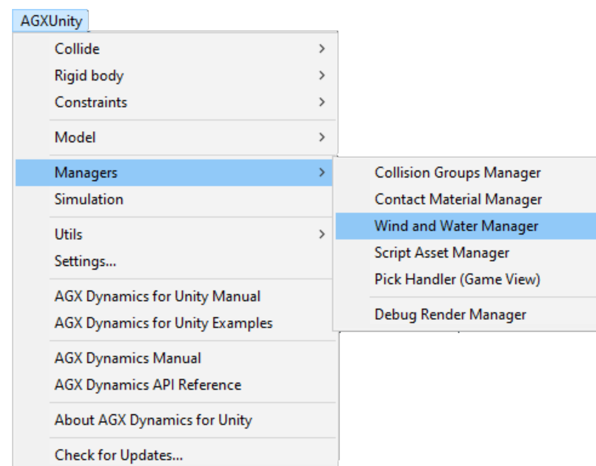

Figura 21: Path per l'importazione del manager.

Verrà creato automaticamente un oggetto chiamato AGXUnity.WindAndWaterManager dotato di cui componenti: *Transform* e *Wind and Water Manager*. Il componente Wind and Water Manager è molto semplice, infatti, è formato da tre parametri:

- *Water:* Oggetto che rappresenta l'acqua. Di default è impostato a ***null***. A *Water* è stato assegnato il gameObject *Pool.*
- *Water Velocity:* parametro che indica la velocità delle correnti marine, di default è impostato a $(\mathbf{0}, \mathbf{0}, \mathbf{0})$. In questo progetto non è stata utilizzata nessuna corrente marina, pertanto, il suo valore è rimasto invariato.
- *Wind Velocity:* parametro che indica la velocità delle correnti aeree, di default è impostato a $(\mathbf{0}, \mathbf{0}, \mathbf{0})$. Essendo questo progetto finalizzato allo studio del comportamento in acqua questo parametro non è stato utilizzato.

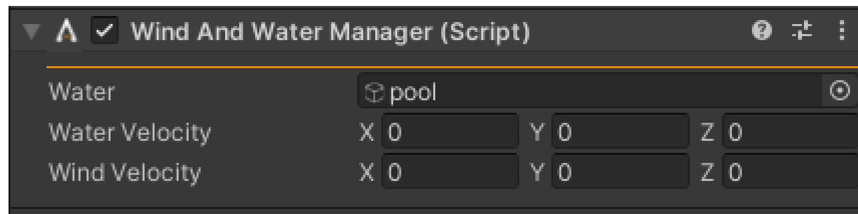

Figura 22: Wind and Water Manager con i valori assegnati

4.1.3. Player

Come descritto nel capitolo **3.1.** il player è rappresentato da un oggetto tridimensionale detto capsula. Per la costruzione del Player si è utilizzato un gameObject vuoto a cui sono stati assegnati cinque componenti: *Capsule, RigidBody, Hydrodynamics Parameters, Transform* e lo script *PlayerController*.

Mediante *Capsule* e *Transform* sono state decise tutte le trasformazioni 3D relative all'oggetto nei diversi casi di studio considerati e per ognuno di essi è stata creata la rispettiva visualizzazione 3D. I parametri di questi componenti sono diversi per ogni caso di studio, tuttavia, entrambi i casi di studio hanno delle caratteristiche in comune: La posizione iniziale, nei diversi casi, differisce solamente per la componente Y, pertanto, in generale la posizione della capsula è $(\mathbf{0},\ \boldsymbol{Y},\ \mathbf{0})$, La rotazione della capsula è uguale per tutti i casi considerati ed è pari a $(-\mathbf{90},\ \mathbf{0},\ \mathbf{0})$. I parametri relativi alle dimensioni, ovvero *Radius* e *Height* variano nei casi considerati in accordo con le caratteristiche del corpo descritte nel paragrafo **3.1.** ma è stato utilizzato in tutti i casi esaminati il materiale *Capsule_Material* per selezionarne la densità. Per entrambi i casi di studio la capsula è orientata secondo una terna destrorsa avente come assi orizzontali gli assi X, Z e come asse verticale L'asse Y.

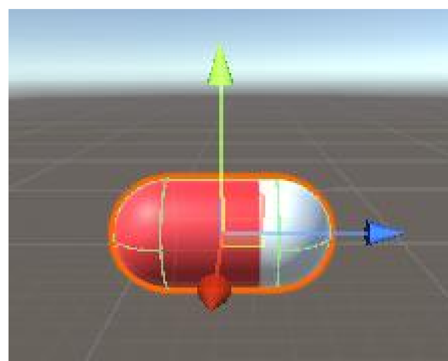

Figura 23: sistema di riferimento della capsula. in rosso l'asse x, in blu l'asse z ed in verde l'asse y

Una trattazione più approfondita del valore dei parametri per i diversi casi di studio verrà illustrata nei paragrafi ***4.4.*** e ***4.5.***

Il componente *RigidBody* è stato utilizzato per conferire al corpo tutte le proprietà dinamiche necessarie al suo movimento, in particolare, è stato impostato il parametro *Motion Control* a *DYNAMICS* per fare in modo che il movimento del corpo sia determinato dalle forze e che la posizione e la velocità e la posizione siano aggiornate dal sistema.

Hydrodynamics Parameters è un componente che deve essere assegnato ai corpi che si muovono attraverso i fluidi per simulare gli effetti idrodinamici come le spinte di lift e drag. Questo componente ha tre parametri principali: *Pressure Drag*, impostato a 0.6, *Viscous Drag*, impostato a 0.1 e *Lift*, impostato a 0.01.

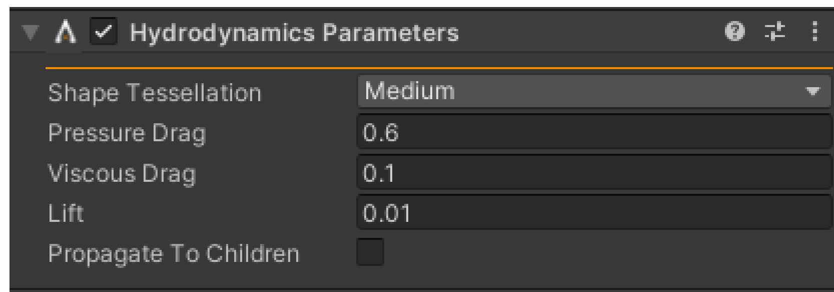

Figura 24: vista del componente Hydrodynamics Parameters

4.1.3.1. Script: PlayerController

Lo script PlayerController è un componente assegnato a *Player* per poterne controllare il movimento. È uno script molto semplice, infatti conta solamente due variabili globali e tre metodi, le variabili utilizzate sono: *public GameObject capsule* e *agx.RigidBody player*.

Per poter utilizzare i metodi e gli attributi del componente *RigidBody* di AGX è necessario estrapolare l'istanza nativa di quest'ultimo; per fare ciò la funzione *Start()* è stata definita in questo modo:

```
void Start()
{
    player = capsule.GetComponent<AGXUnity.RigidBody>().GetInitialized<AGXUnity.RigidBody>().Native;
}
```

Figura 25: Estrapolazione dell'istanza nativa di RigidBody

Per l'ottenimento dei dati relativi alla posizione del centro di massa si utilizza la funzione *getCMPosition()*; questa funzione restituisce un *Vector3* che rappresenta le coordinate spaziali del centro di massa.

```
public Vector3 getCMPosition()
{
    print(transform.position);
    return transform.position;
}
```

Figura 26: codice di getCMPosition()

L'ultima funzione all'interno di questo script è *movement(float force)*, questa funzione utilizza la funzione *AddForce(agx.Vec3 force)* per applicare al corpo una forza lungo la componente z del corpo ricevuta in input da Simulink.

```
public void movement (float force)
{
    agx.Vec3 Forza = new agx.Vec3(0,0, force);
    player.addForce(Forza);

}
```

Figura 27: Codice della funzione movement(float force)

4.2. Schema di controllo

Lo schema di controllo del movimento del corpo proposto è stato sviluppato mediante Simulink; Lo schema è quello di un sistema retroazionato controllato utilizzando un regolatore PID:

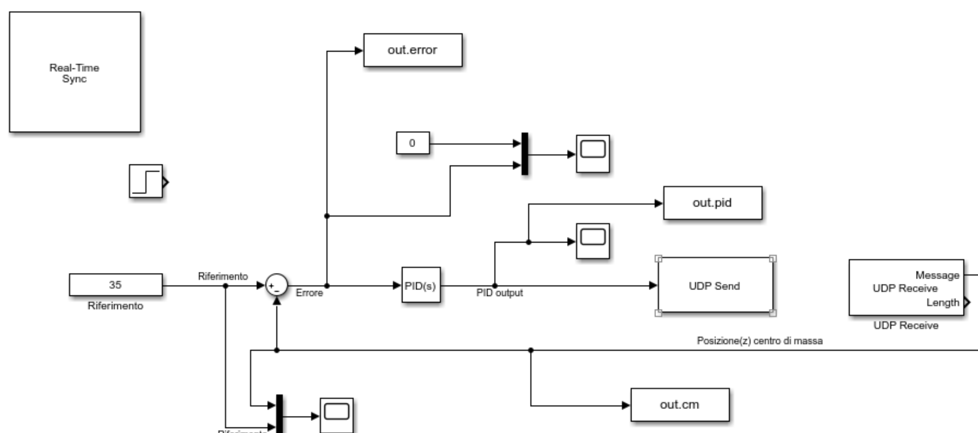

Figura 28: Schema di controllo implementato su Simulink

Dello schema a blocchi presentato precedentemente non tutti i blocchi vengono effettivamente utilizzati per modellare il controllo del movimento, alcuni di essi sono stati utilizzati solamente per il raccoglimento dei dati e per fare grafici. Di seguito viene proposta una versione semplificata dello schema che permette di capirne al meglio il funzionamento:

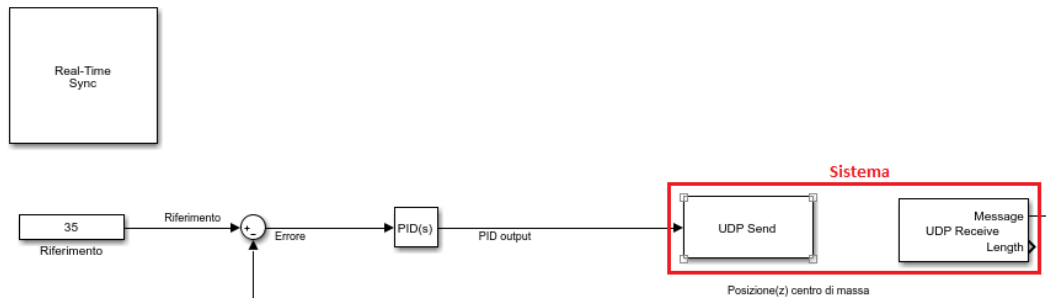

Figura 29: Schema a blocchi semplificato

Tra i blocchi più importanti utilizzati in questo progetto c'è il *Real-Time Synchroniation*, blocco necessario per lavorare in Real-Time, dato che Simulink di default non lo fa; per poter utilizzare questo blocco bisogna installare il Real-Time Kernel da Matlab. I parametri identificativi di questo blocco sono due: *Sample Time* e *Maximum missed ticks*, che rappresentano rispettivamente il tempo di campionamento (in secondi) e il massimo numero di tick mancati; nello specifico di questo progetto il valore impostato per *Sample time* è 0.005, mentre quello di *Maximum missed ticks* è di 100000, scelto così alto per evitare che il programma si blocchi in caso di lag

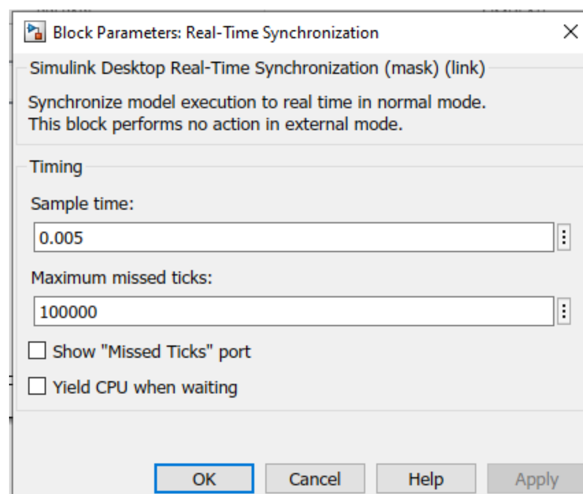

Figura 30: parametri del blocco Real-Time Syncrhonization

I blocchi *out.error, out.cm* e *out.pid* sono blocchi che permettono di salvare il valore rispettivamente dell'errore, della posizione del centro di massa e dell'output dei PID. Nello schema sono presenti anche tre blocchi di tipo *Scope* utilizzati per ottenere i grafici della posizione del centro di massa confrontata con il riferimento a cui deve arrivare, l'errore confrontato con lo zero e dell'uscita dei PID.

I blocchi *UDP Send* e *UDP Receive* sono stati utilizzati per implementare la comunicazione con Unity tramite il protocollo UDP, un approfondimento sull'utilizzo di questi blocchi sarà presentato nel paragrafo **4.3.**

4.2.1. Regolatori PID

Il PID (Proporzionale-Integrale-Derivativo) è sistema in retroazione negativa comunemente utilizzato nei sistemi di controllo, è particolarmente utilizzato all'interno delle industrie specialmente la versione PI. Tra i principali vantaggi dell'utilizzo de PID c'è la di controllare in modo soddisfacente un'ampia gamma di processi, la possibilità di tarare i suoi parametri anche in assenza del modello matematico del processo da controllare ed i costi ridotti di realizzazione.

Come precisamente descritto nel testo [9] i PID sono sistemi dalla struttura prefissata caratterizzata dalla presenza di tre parametri liberi e regolabili, si basano sul principio che l'ingresso u(t) del sistema sia la somma di tre contributi: uno proporzionale all'errore, uno proporzionale all'integrale dell'errore ed uno proporzionale alla derivata dell'errore, pertanto, la legge di controllo del PID è la seguente:

$$u(t) = K_P e(t) + K_I \int_{t_0}^{t} E(\tau) d\tau + K_D \frac{de(t)}{dt}$$

Il termine $K_P e(t)$ permette di diminuire l'errore ma non permette di avere errore nullo a regime con segnali di riferimento costanti.

Il termine $K_I \int_{t_0}^{t} E(\tau) d\tau$ permette di annullare l'errore a regime con segnali di riferimento costante introducendo tuttavia, un ritardo di fase che rallenta la risposta del sistema

Il termine $K_D \frac{de(t)}{dt}$ ha l'obiettivo di anticipare l'andamento futuro dell'errore migliorando la stabilità, non viene utilizzato mai da solo poiché in generale peggiora le prestazioni a regime

La tipica implementazione di un PID è l'implementazione in parallelo, consiste nel trattare ciascuna modalità di controllo separatamente e unirle tramite un nodo sommatore:

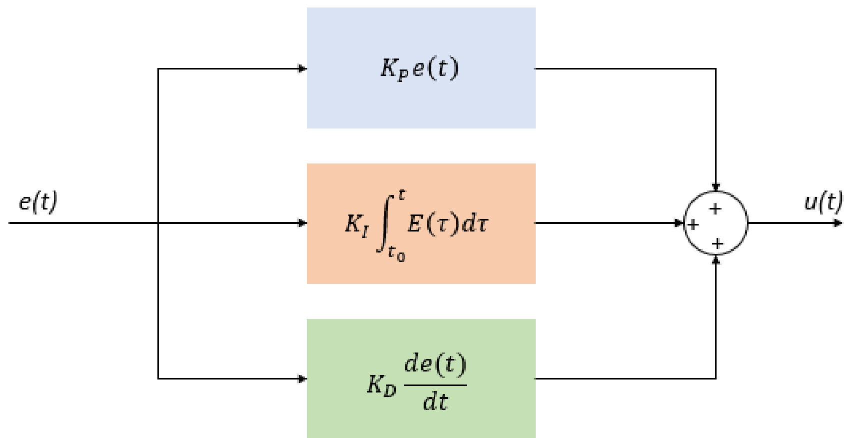

Figura 31: Schema a blocchi dell'implementazione in parallelo

A partire da questo schema è possibile calcolare la funzione di trasferimento del PID,

$$G(s) = K_P \left(1 + \frac{1}{sT_I} + \frac{T_D s}{1 + \frac{T_D}{N} s} \right)$$

Con $T_I = \frac{K_P}{K_I}$, Tempo integrale, $T_D = \frac{K_D}{K_P}$ e N coefficiente di filtraggio, un numero che si introduce per rendere fisicamente realizzabile la componente derivativa, tipicamente ha valori compresi tra 5 e 20

4.2.1.1. Calibrazione PID

Esistono due tecniche per la determinazione sperimentale delle impostazioni del controllore che permettono di ottenere i coefficienti ottimi per i PID, queste sono il metodo di Ziegler-Nichols ad anello chiuso ed il metodo di Ziegler-Nichols ad anello aperto

4.2.1.1.1. Prima Tecnica di Ziegler-Nichols

La prima tecnica di Ziegler-Nichols è detta anche metodo di Ziegler-Nichols ad anello chiuso poiché la taratura del PID avviene testando il sistema ad anello chiuso. Per applicare questo metodo si utilizza la risposta del sistema al segnale *sca(t):*

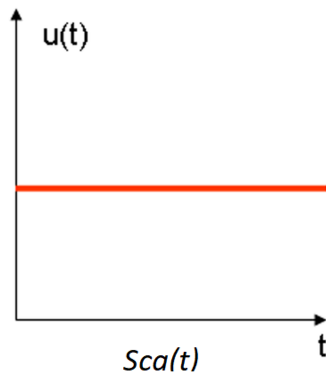

Figura 32: Segnale Scalino

Si considera l'azione proporzionale considerando un K_P piccolo, e si annullano le azioni derivativa e integrale; a questo punto si aumenta il guadagno fino a $K_P = K_0$, ovvero il valore in cui il sistema entra in oscillazione e si determina il periodo T_0. Da questi valori si possono ottenere i valori per l'implementazione di controllori P, PI e PID:

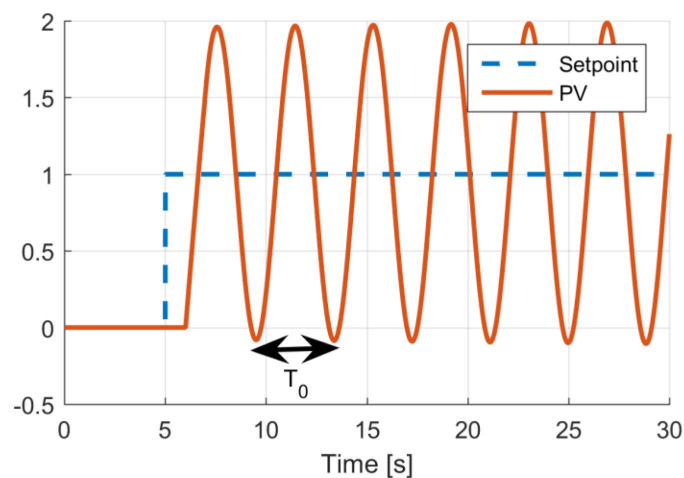

Figura 33: Grafico del sistema in oscillazione

- *Controllore P:* $K_P = 0.5\,K_0$
- *Controllore PI:* $K_P = 0.45\,K_0,\ T_I = 0.8\,T_0$
- *Controllore PID:* $\mathrm{K_P} = 0.6\ \mathrm{K_0},\ \mathrm{T_I} = 0.5\ \mathrm{T_0},\ T_D = 0.125\,T_0$

È possibile calcolare i valori dei parametri $\mathrm{K_I}$ e $\mathrm{K_D}$ utilizzando le relazioni $\mathrm{K_I} = \frac{K_P}{T_I}$ e $\mathrm{K_D} = K_P\,T_D$

4.2.1.1.2. Seconda tecnica di Ziegler-Nichols

Il secondo metodo di Ziegler-Nichols per la taratura dei PID è detto anche metodo di Ziegler-Nichols ad anello aperto. Come spiegato nel testo [10] questa tecnica è utilizzata tipicamente per i sistemi del prim'ordine caratterizzati da una risposta allo scalino di questo tipo:

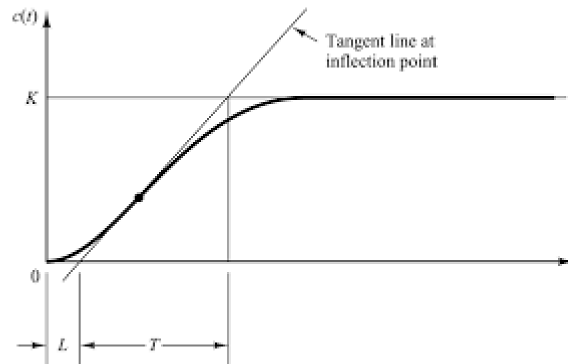

Figura 34: Risposta allo scalino per il metodo ad anello aperto

Come si può vedere dalla figura si possono identificare due parametri caratteristici: il ritardo L e la costante di tempo T; entrambi i parametri si ottengono tracciando la tangente alla curva di risposta allo scalino nel suo punto di flesso. Utilizzando i valori di T ed L è possibile calcolare i valori dei parametri caratteristici dei controllori P, PI, PID:

- *Controllore P:* $K_P = \frac{\mathrm{T}}{\mathrm{L}}$
- *Controllore PI:* $K_P = 0.9\frac{\mathrm{T}}{\mathrm{L}}, T_I = \frac{L}{0.3}$
- *Controllore PID:* $K_P = 1.2\frac{\mathrm{T}}{\mathrm{L}}, T_I = 2\mathrm{L}\,, T_D = 0.5\mathrm{L}$

4.2.2. Implementazione su Simulink

Per la calibrazione dei PID è stato utilizzato il metodo di Ziegler-Nichols ad anello chiuso. Inizialmente è stato utilizzato un blocco per la generazione di un segnale a scalino utilizzato al posto del blocco *Riferimento* presentato in figura 28. Successivamente si modifica il valore P del blocco PID, mantenendo nulli i valori di I e D, per trovare un valore di P per cui il sistema inizia ad oscillare. Una volta trovato il valore P corretto e calcolato il valore di T_0, si inseriscono questi valori all'interno di uno script creato per calcolare i parametri ottimi per il funzionamento dei PID.

Parametri calcolati

```
K0 %inserire valore iniizale
t0 %inserire valore iniziale
```

Parametri PI

```
Kp_PI=0.45*K0;
ti_PI=0.8*t0;
ki_PI=Kp_PI/ti_PI;
```

Parametri PID

```
Kp_PID=0.6*K0;
ti_PID=0.5*t0;
td_PID=0.125*t0;
ki_PID=Kp_PID/ti_PID;
kd_PID=Kp_PID*td_PID;
```

Figura 35: Script per il calcolo del valore dei PID

Una volta calcolati i parametri dei controllori PI e PID, se ne sceglie uno e si settano i valori del blocco *PID(s)*

4.3. Implementazione della comunicazione tra Simulink e Unity

Per permettere la comunicazione tra i software è stato utilizzato il protocollo UDP, pertanto è stato necessario integrare un metodo per utilizzare questo protocollo sia su Simulink che su Unity.

Per quanto riguarda Simulink, il software stesso mette a disposizione dei blocchi che permettono di utilizzare UDP, questi blocchi sono *UDP Send*, per inviare dati da Simulink ad un programma esterno, ed *UDP Receive,* per ricevere i dati da un programma esterno a Simulink.

I parametri identificativi di un blocco send sono solo tre: *Remote IP Address*, *Remote IP Port* e *Send buffer size* che rappresentano rispettivamente l'indirizzo IP e la porta a cui inviare il messaggio e la dimensione massima del buffer del blocco send espressa in bytes. Per questi parametri sono stati scelti i seguenti valori:

- *Remote IP Address:* 127.0.0.1
- *Remote IP Port:* 25000
- *Send Buffer:* 8192

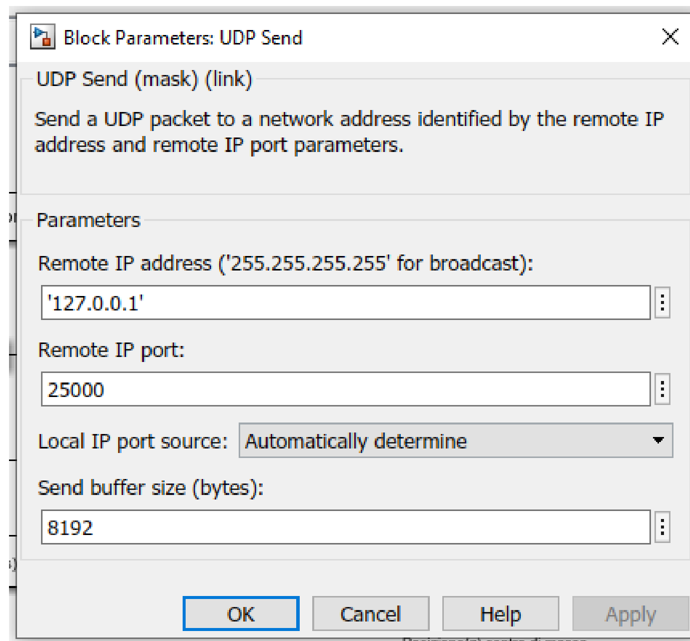

Figura 36: Parametri del blocco UDP Send

I parametri di un blocco Receive invece sono cinque, essi sono *Local IP Port*, *Local IP Address, Receive buffer size, Maximum lenght for Message* e *Sample time;* questi parametri indicano rispettivamente la porta e l'indirizzo a cui inviare il messaggio, la dimensione del buffer di receive espressa in bytes, la dimensione massima del messaggio da ricevere ed il tempo di campionamento espresso in secondi. I valori assegnati a questi parametri sono:

- *Local IP Port:* 26000
- *Local IP Address:* 0.0.0.0
- *Receive buffer size:* 8192
- *Maximum lenght for Message:* 1
- *Sample time*: 0.005

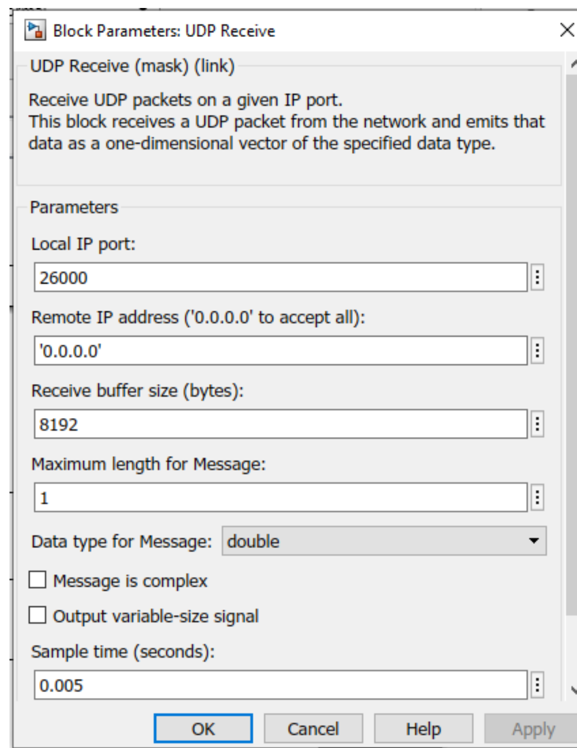

Figura 37: parametri caratteristici del blocco receive

Per l'implementazione del protocollo UDP su Unity si è creato un gameObject vuoto chiamato UDP Control a cui sono stati assegnati come componenti tre script: *UDP Transmitter*, studiata per implementare una Send, *UDP Receiver*, che implementa una Receive, e *UDP Controller*, che permette di utilizzare le Send e Receive per inviare e ricevere messaggi. Per l'implementazione di questi script si è partiti da quelli forniti da [11].

UDP Transmitter è una classe che implementa le funzionalità di una send; è composta da quattro attributi e cinque metodi. Gli attributi sono:

- *String IP:* una variabile utilizzata per determinare l'indirizzo IP a cui inviare i dati, è stata inizializzata a "127.0.0.1".
- *Int TransmitPort:* variabile che indica a quale porta bisogna inviare i dati, è stata dichiarata come pubblica per poter cambiare il suo valore da Unity.
- *IPEndPoint RemoteEndPoint:* è un tipo di variabile composta da indirizzo IP e porta che serve per determinare la destinazione dei dati, viene inizializzata utilizzando le variabili *IP* e *TransmitPort*.

- *UDPClient Transmit:* variabile che consente l'utilizzo dei servizi del protocollo UDP.

Il metodo *Start()* è utilizzato per l'inizializzazione delle variabili, in particolare esso richiama il metodo *Initialize()* che effettua questa inizializzazione:

```
void Start()
{
    Initialize();
}
public void Initialize()
{

    RemoteEndPoint = new IPEndPoint(IPAddress.Parse(IP), TransmitPort);
    Transmit = new UdpClient();
}
```

Figura 38: funzioni Start() e Initialize() di UDP Transmitter

Successivamente è definita la funzione *Send(double val)* che permette l'invio di una variabile di tipo double all'endpoint precedentemente inizializzato. Questa funzione sfrutta la funzione *Send* fornita da *Transmit:*

```
public void Send(double val)
{
    try
    {
        byte[] MessageByteArray = BitConverter.GetBytes(val);
        Transmit.Send(MessageByteArray, MessageByteArray.Length, RemoteEndPoint);
    }
    catch (Exception error)
    {
        Debug.Log("<color=red>" + error.Message + "</color>");
    }
}
```

Figura 39: Funzione Send(double val) di UDP Transmitter

È stata sviluppata una versione della funzione Send che possa inviare dati del tipo *Vector3* molto utilizzati in Unity, questa è *Send(Vector3 val):*

```
public void Send(Vector3 val)
{

        try
        {

            byte[] message = new byte[24];
            Buffer.BlockCopy(BitConverter.GetBytes((double)val.x), 0, message, 0, 8);
            Buffer.BlockCopy(BitConverter.GetBytes((double)val.y), 0, message, 8, 8);
            Buffer.BlockCopy(BitConverter.GetBytes((double)val.z), 0, message, 16, 8);
            Transmit.Send(message, message.Length, RemoteEndPoint);

        }
        catch (Exception error)
        {
            Debug.Log(error.Message);
        }
```

Figura 40: funzione Send(Vector3 val) di UDP Transmitter

L'ultima funzione sviluppata è *OnApplicationQuit()* che permette la deinizializzazione dei dati alla chiusura per evitare errori in utilizzi successivi:

```
private void OnApplicationQuit()
{
    try
    {
        Transmit.Close();
    }
    catch(Exception error)
    {
        Debug.Log("<color=red>" + error.Message + "</color>");
    }
}
```

Figura 41: funzione OnApplicationQuit() di UDP Transmitter

UDP Receive implementa le caratteristiche di una receive, è una classe dotata di quattro attributi e cinque metodi. In particolare, gli attributi sono:

- *Int Port*: variabile che specifica la porta da cui si deve ricevere il messaggio. Si è definita come pubblica in modo da poterla modificare da Unity.
- *UDPClient Client:* variabile che consente l'utilizzo dei servizi del protocollo UDP.
- *Thread receiveThread:* variabile che definisce il thread su cui la receive lavorerà.
- *Float receive*: variabile in cui viene salvato il dato ricevuto, è inizializzata a 0.

La funzione *Start()* è utilizzata per richiamare la funzione *Initialize()* che si occupa dell'inizializzazione delle variabili, nello specifico attiva il thread e lo mette in background.

```
void Start()
{
    Initialize();
}

// Update is called once per frame
void Initialize()
{
    receiveThread = new Thread(new ThreadStart(ReceiveData));
    receiveThread.IsBackground=true;
    receiveThread.Start();

}
```

Figura 42: funzioni Start() e Initalize() di UDP Receive

La funzione *ReceiveData()* si occupa della ricezione dei dati a partire da uno specifico indirizzo IP e una porta, mentre è possibile modificare il valore della porta da Unity,

l'indirizzo IP è stato scelto 0.0.0.0 che sta a significare che è possibile ricevere da qualsiasi indirizzo IP. Una volta ricevuti i dati essi vengono inseriti all'interno della variabile *receive*:

```
private void ReceiveData()
{
    Client = new UdpClient(port);
    while (true)
    {
        try
        {
          IPEndPoint anyIp = new IPEndPoint(IPAddress.Any, 0);
           byte[] data = Client.Receive(ref anyIp);


         receive=(float)BitConverter.ToDouble(data,0);



        }
        catch(Exception error)
        {
            Debug.Log(error.Message);
        }
    }
}
```

Figura 43: Funzione ReceiveData() di UDP Receive

Per poter utilizzare la variabile *receive* è stata inserita la funzione *getReceiver()* che permette di accedere in lettura alla variabile *receive:*

```
public float getReceiver()
{
     //print(receive);
    return receive;
}
```

Figura 44: funzione getReceiver() di UDP Receiver

Come ultima funzione di *UDP Receiver* è *OnApplicatinQuit()* che deinizializza le variabili per evitare errori in caso di utilizzi futuri:

```
public void OnApplicationQuit()
{
    try
    {
        receiveThread.Abort();
        receiveThread = null;
        Client.Close();
    }
    catch(Exception error)
    {
        Debug.Log(error.Message);
    }
}
```

Figura 45: OnApplicatioQuit() di UDP Receiver

UDP Controller è la classe che si occupa della gestione delle classi sopracitate per la trasmissione e la ricezione dei messaggi. È una classe molto semplice, infatti è composta solamente da tre attributi e due metodi; gli attributi sono:

- *UDPTransmitter Transmitter:* istanza della classe *UDPTransmitter*, viene usata per inviare i messaggi
- *UDPReceiver Receiver:* istanza della classe *UDPReceiver* utilizzata per la ricezione di dati
- *gameObject Player:* oggetto che rappresenta il corpo da cui si devono estrapolare i dati da inviare a Simulink e su cui si devono inviare i dati ricevuti da Simulink

Per l'inizializzazione delle variabili si utilizza il metodo *Awake()*, questo è un metodo di Unity che si attiva prima di qualsiasi altro metodo:

```
void Awake()
{
    Transmitter = GetComponent<UDPTransmitter>();
    Receiver = GetComponent<UDPReceiver>();
}
```

Figura 46: Funzione Awake di UDPControl

Lo scambio di messaggi tra Unity e Simulink viene gestito all'interno della funzione *FixedUpdate()*, all'interno di questa funzione viene settata la forza da applicare in base ai dati ricevuti da Simulink e viene inviata la componente z della posizione del centro di massa a Simulink:

```
void FixedUpdate()
{
    player.GetComponent<PlayerController>().movement(Receiver.getReceiver());
    Transmitter.Send(player.GetComponent<PlayerController>().getCMPosition().z);
}
```

Figura 47: FixedUpdate di UDPControl

4.4. Presentazione casi di studio analizzati

In questo paragrafo verranno presentati due casi di studio per la simulazione del Digital Twin: il primo è ottenuto considerando che la densità del corpo è pari a 500 mentre, il secondo è ottenuto impostando la densità del corpo a 1000. Per entrambi i casi di studio

verranno illustrate le dimensioni della capsula, i parametri ottenuti con la taratura dei PID ed i grafici relativi al controllo, l'errore e l'output dei PID.

Le simulazioni sono state effettuate con un laptop HP Notebook che monta un processore *AMD A8-7410 APU* con scheda video integrata *Radeon R5 Graphics* e 8 GB di RAM. Date le caratteristiche del computer è possibile notare, nei grafici presentati, la presenza di piccoli errori dovuti alla perdita di dati, accentuati dall'utilizzo del protocollo UDP.

4.4.1. Caso di studio con densità pari a 500

Come già spiegato nel paragrafo **4.1.3.** le dimensioni della capsula e la sua posizione iniziale variano a seconda del caso considerato per rispettare delle caratteristiche definite; nel caso del corpo con densità pari a 500 il raggio delle semisfere che compongono la capsula è pari a 0.57m mentre l'altezza del cilindro è di 1.151m.

Utilizzando il metodo di Ziegler-Nichols ad anello aperto è stato calcolato il periodo di oscillazione T_0 e il guadagno iniziale K_0, per questo caso di studio sono stati selezionati $K_0 = 200$ e $T_0 = 14.008$

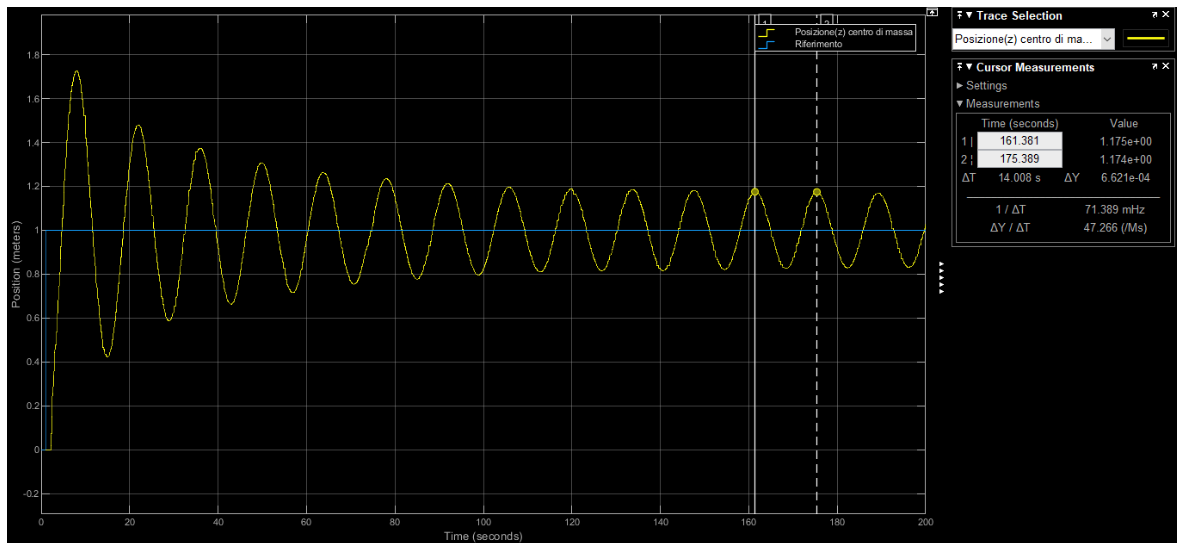

Figura 48: Risposta allo scalino con $K_0 = 200$ e $T_0 = 14.008$.

In blu il riferimento mentre in giallo il valore dell'uscita

Utilizzando questi dati è stato possibile calcolare il valore dei parametri per i controllori PI e PID, i loro valori sono:

- *Controllore PI:*
    - $K_P = 90$
    - $T_I = 11.2064 \Rightarrow K_I = 8.0311$

- *Controllore PID:*
  - $K_P = 120$
  - $T_I = 7.0040 \implies K_I = 17.1331$
  - $T_I = 1.7516 \implies K_I = 210.1200$

Per la simulazione del comportamento della capsula è stato scelto un controllore di tipo PI. Utilizzando il blocco PID(s) viene calcolato l'ingresso da dare al sistema istante per istante, di seguito si allega il grafico che mostra l'andamento dell'uscita del PI nel tempo; è possibile notare che nei primi secondi il valore dell'uscita del PI ha un andamento brusco verso l'alto, questo è dovuto a dei lag, che si hanno nel far partire la simulazione, dovuti al computer utilizzato.

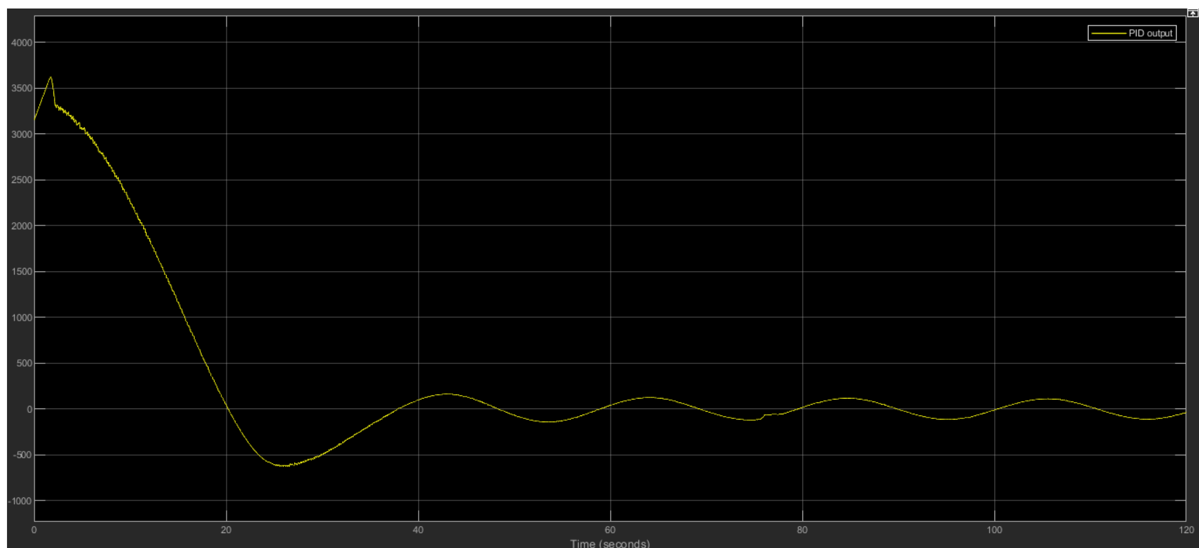

Figura 49: Grafico dell'output del PI nel tempo

La posizione utilizzata come riferimento è pari a $(0, 20, 35)$ mentre il corpo parte dalla posizione $(0, 20, 0)$; la simulazione è durata 120s, tempo in cui il corpo riesce a stabilizzarsi attorno alla posizione di riferimento. Di seguito è allegato il grafico della posizione del corpo confrontato con il riferimento selezionato.

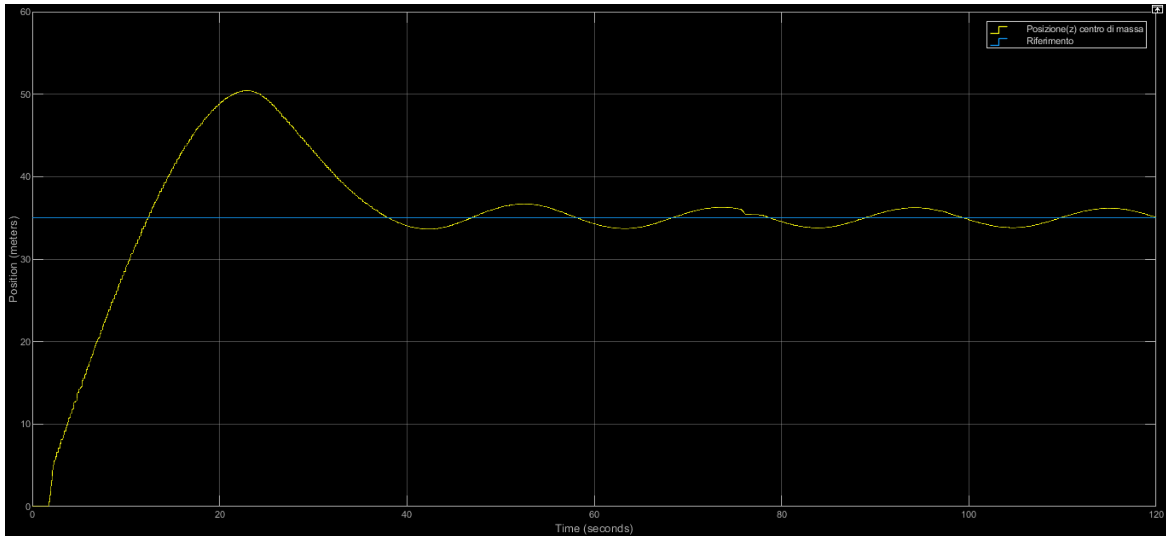

Figura 50:grafico del confronto tra la posizione del centro di massa del corpo e il riferimento.

In blu il riferimento mentre in giallo la coordinata z della posizione del centro di massa

Dal grafico si può osservare che il corpo inizia a stabilizzarsi attorno alla posizione di riferimento già a 40s dall'esecuzione della simulazione, e continua ad avvicinarsi all'obiettivo nel tempo rimanente.

Per comprendere al meglio il comportamento del corpo si sono studiate le caratteristiche dell'errore, in particolare l'errore ottenuto dalla simulazione ha una *media* pari a -0.0319 e una *varianza* di 67.2257. Di seguito è presentato il grafico dell'andamento dell'errore confrontato con lo zero:

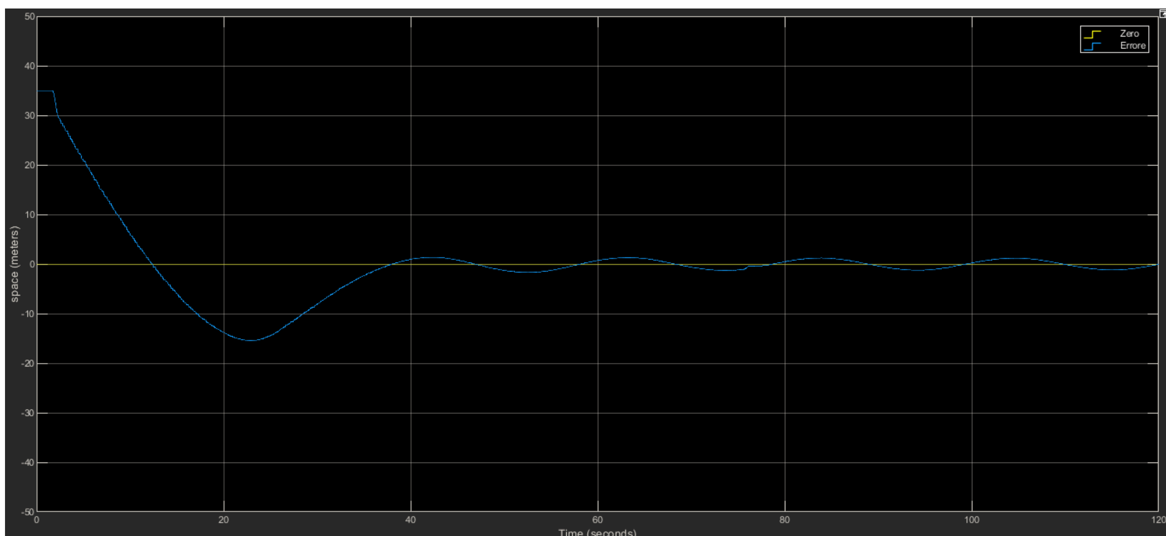

Figura 51: andamento dell'errore nel tempo: in blu l'errore, in giallo lo zero

4.4.2. Caso di studio con densità pari a 1000

Per considerare questo caso di studio le dimensioni della capsula che rispettano le caratteristiche del corpo elencate precedentemente sono: raggio delle semisfere pari a 0.45m e altezza del cilindro pari a 0.91m.

Il calcolo dei coefficienti K_0 e T_0 utilizzando il metodo di Ziegler-Nichols ad anello chiuso ha dato come risultati $K_0 = 300$ e $T_0 = 10.805$; da questi sono stati calcolati i valori dei parametri per i controllori PI e PID:

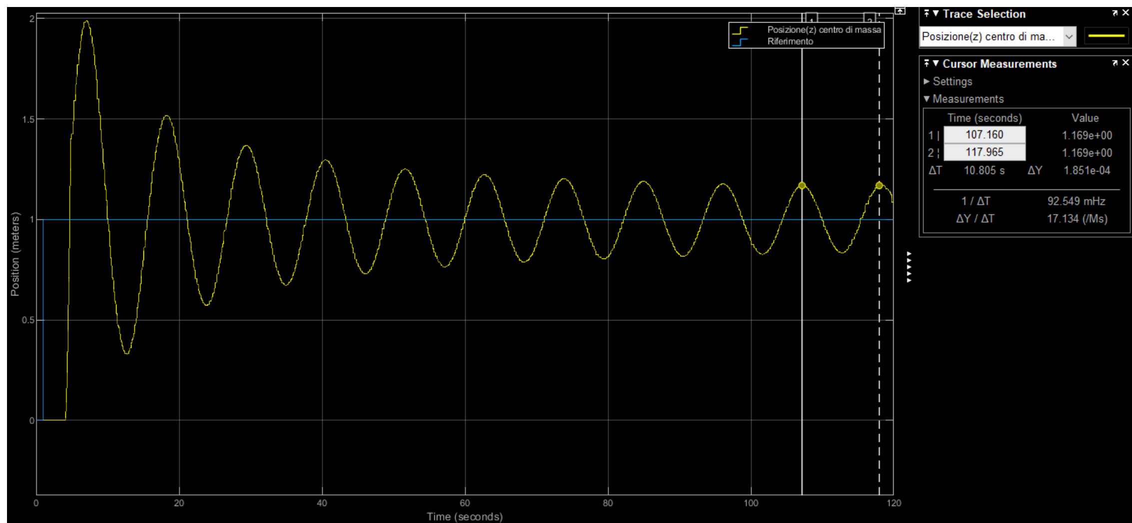

Figura 52: Risposta allo scalino del sistema.

In blu il riferimento, in giallo la risposta

- *Controllore PI:*
  - $K_P = 135$
  - $T_I = 8.6440 \Longrightarrow K_I = 15.6178$
- *Controllore PID:*
  - $K_P = 243.1125$
  - $T_I = 5.4025 \Longrightarrow K_I = 33.3179$
  - $T_D = 1.3506 \Longrightarrow K_D = 243.1125$

Anche in questo caso per il controllo è stato impiegato un regolatore PI. Di seguito è presentato il grafico dell'andamento del controllore rispetto al tempo, anche in questo caso ci sono degli errori all'inizio a causa dei lag.

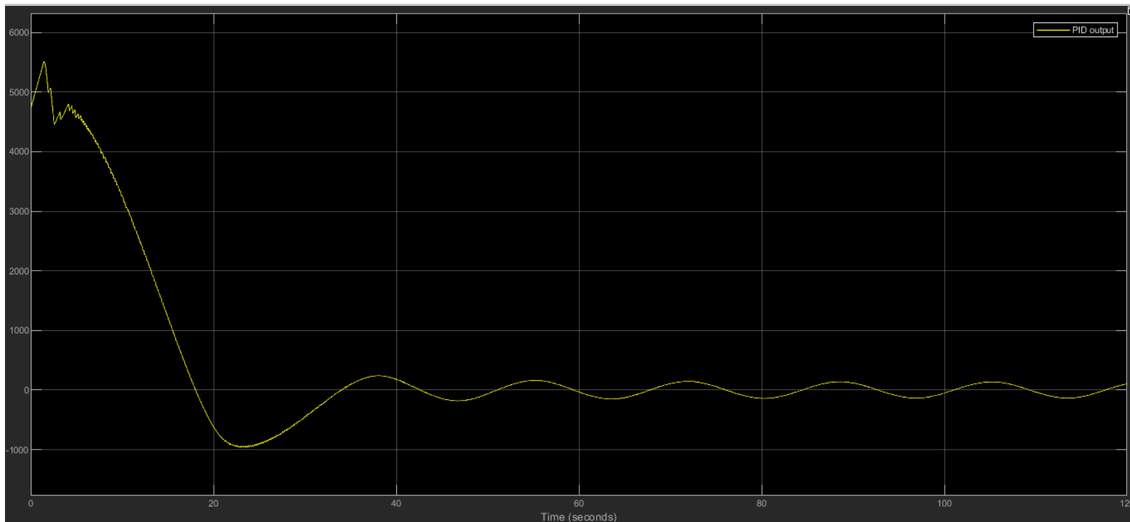

Figura 53: Andamento dell'output del PI nel tempo

In questo caso il corpo parte con una posizione iniziale di $(0, 15, 0)$, ovvero in immersione completa, e come posizione di riferimento è stata scelta la posizione $(0, 15, 35)$. Anche in questo caso la simulazione dura 120s; l'andamento della posizione nel tempo è:

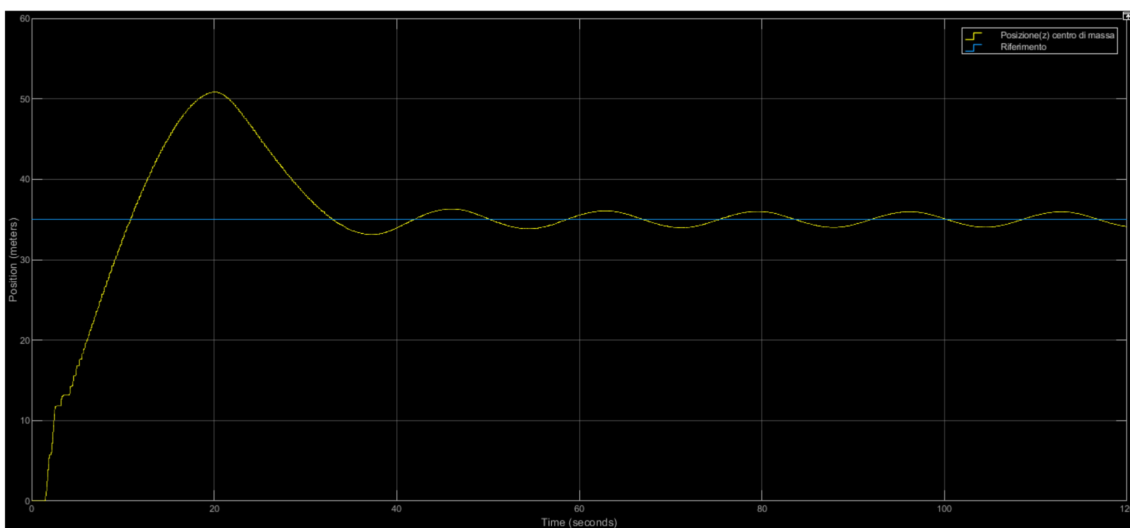

Figura 54: coordinata z della posizione del centro di massa confrontata con il riferimento.

In giallo la posizione del centro di massa, in blu il riferimento

In questo caso si può osservare che il corpo inizia a stabilizzarsi attorno alla posizione di riferimento prima di quanto facesse nel caso precedente.

Lo studio dell'errore in questo caso ha mostrato una *media* pari a -0.0078 e una *varianza* di 57.4899. Di seguito si allega il grafico che mostra l'andamento dell'errore nel tempo confrontato con lo zero:

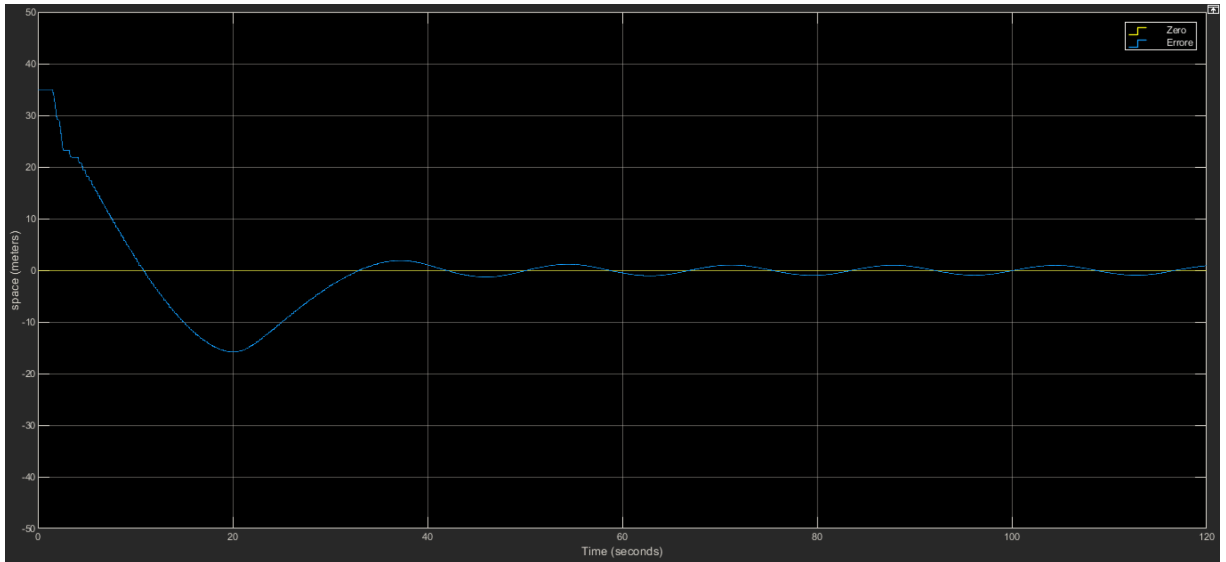

Figura 55: Errore di controllo nel tempo. in blu l'errore, in giallo il riferimento

5. Conclusioni e sviluppi futuri

Per la realizzazione di questo progetto sono stati considerati i movimenti della capsula solamente lungo la direzione z; comunque è stata implementata una funzione che permette il trasferimento di vettori di dati tramite UDP, pertanto, una possibile applicazione futura potrebbe essere quella di riuscire a controllare il movimento della capsula in tutte e tre le direzioni spaziali.

Un altro ambito in cui si potrebbero fare delle modifiche è proprio nella comunicazione tra Simulink e Unity; se pur vero che l'utilizzo del protocollo UDP permette di inviare dati ad una velocità elevata, esso è soggetto a perdite di dati che diventano elevate in caso di utilizzo di dispositivi poco potenti. In futuro si potrebbe pensare di sostituire l'utilizzo del protocollo UDP con un protocollo che implementi delle tecniche di sicurezza e recupero dati come ad esempio il TCP.

L'obiettivo principale di questo progetto è stato la simulazione del modello di un oggetto semplice in acqua per studiarne il comportamento; lo studio di un caso ideale come questo è utile per potersi cimentare nello studio di casi reali, infatti questo progetto potrebbe essere utilizzato come base per la modellazione e simulazione di sistemi di navigazione più complessi.

BIBLIOGRAFIA

[1] Hendrik van der Valk, Frederik Möller, Jan-Luca Henning, Hendrik Haße, Michael Arbter and Boris Otto. *A Taxonomy of Digital Twins.*

[2] J. W. Nicholson, A. J. Healey. *The Present State of Autonomous Underwater Vehicles (AUV) Application and Technologies.* March 2008 – Marine Technology Society Journal (42): 44-51

[3] Sito Matlab: https://www.mathworks.com/products/matlab.html

[4] Documentazione Unity3D: https://docs.unity3d.com/Manual/index.html

[5] Sito Algoryx: https://www.algoryx.se/agx-dynamics/?utm_term=agx%20dynamics&utm_campaign=AGX&utm_source=adwords&utm_medium=ppc&hsa_acc=3676762440&hsa_cam=10062947755&hsa_grp=102831328442&hsa_ad=435384703433&hsa_src=g&hsa_tgt=kwd-906704179615&hsa_kw=agx%20dynamics&hsa_mt=b&hsa_net=adwords&hsa_ver=3&gclid=Cj0KCQjw24qHBhCnARIsAPbdtlIetMhxBqgE7CosgsBr-kei1vjcNAmUBzjDLR0Sf7RAo8elXiU1uCwaAhQKEALw_wcB

[6] J. Postel. *RFC 768.* 28 August 1980

[7] Documentazione AGX Unity: https://us.download.algoryx.se/AGXUnity/documentation/current/editor_interface.html

[8] AGX Hydro- and aereodynamic: https://www.algoryx.se/documentation/complete/agx/tags/latest/UserManual/source/hydro__and_aerodynamics.html

[9] Christos Tsironis and K. Iordanis Giannopoulos. *Automatic Control Systems in Tokamak Plasmas: Current Status and Needs.* 2009 Athens,Greece

[10] Manoj Kushwah and Ashis Patra. *PID Controller Turning using Ziegler-Nichols Method for Speed Control of DC Motor.* June 2014 – International Journal of Scientific Engineering and Technology Research, Volume 03, pages: 2924-2929.

[11] UDP Example project: https://github.com/CihadDogan/UDPExample