

ALMA MATER STUDIORUM - UNIVERSITA DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria - DISI

Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA

in

ELETTRONICA

**RILEVAZIONE ED ELABORAZIONE DEI
DATI DEL MONDO REALE PER LA
REALTÀ VIRTUALE: VRBIKE**

CANDIDATO

Lorenzo Andraghetti

RELATORE

Chiar.mo Prof. Bruno Riccò

CORRELATORE

Prof. Massimo Lanzoni

Anno Accademico 2016/17

Sessione II

Ai miei nonni e ai miei genitori.

I miei pilastri.

Indice

1 Introduzione	3
1.1 Struttura della tesi	4
2 Realtà Virtuale	5
2.1 Come si crea la Realtà Virtuale	6
3 Architettura del sistema Hardware	9
3.1 Sensori del manubrio	10
3.2 Sensori del volano	11
3.3 Microcontrollore	12
3.3.1 Stringa composta dal microcontrollore	12
3.4 Ricevitore Bluetooth	12
3.5 OSVR	13
3.5.1 Caratteristiche	13
3.5.2 Integrazione	14
4 Architettura del sistema Software	15
4.1 Blender	15
4.1.1 Mesh della bicicletta	16
4.1.2 Mesh dell'ambientazione	18
4.2 Unity	19
4.2.1 Schermata Principale	19
4.2.2 Scenario Principale	21
4.3 DataReceiver script	24
4.4 Motocontroller script	27
4.5 OSVR	32

5 Sperimentazione e correzioni	33
5.1 Realizzazione del sistema	33
5.2 OSVR e le impostazioni iniziali	35
5.3 Calibrazioni della fisica virtuale	35
5.3.1 Velocità	36
5.3.2 Frenata e Inerzia	37
6 Direzioni future di ricerca e conclusioni	39
6.1 Soluzione	40
6.2 Conclusioni	41
A Codice script DataReceiver	43
B Codice script MotoController	47

Capitolo 1

Introduzione

La tesi si prefigge la realizzazione di una applicazione che rilevi le informazioni del mondo reale per elaborarle così da modellare una realtà virtuale.

Si tratta quindi di un programma che permetta all'utente, dotato di visore per la realtà virtuale, la possibilità di muoversi all'interno di un mondo tridimensionale. La possibilità di movimento è data da una cyclette, in modo da dare all'utilizzatore la sensazione di muoversi lungo un percorso su una bicicletta. La cyclette è fornita di particolari sensori che ne determinano velocità di pedalata e rotazione del manubrio. Lo scopo della tesi è quello di far comunicare questi sensori con un computer in modo da pilotare la bicicletta virtuale.

In futuro si cercherà di utilizzare una bicicletta vera, applicata su rulli che diano un ritorno di forza per implementare la sensazione di fatica in casi di salita o discesa.

1.1 Struttura della tesi

La tesi è strutturata nel modo seguente.

Nella sezione *Realtà Virtuale* si illustrerà lo stato dell'arte nell'ambito della realtà virtuale. Lo studio di questo ambito è stato reso necessario per la progettazione.

Nelle sezioni *Architettura del sistema Hardware* e *Architettura del sistema Software* si mostrerà il progetto nella sua interezza con la descrizione di tutti i moduli.

Nella sezione *Sperimentazione e correzioni* si descriveranno i processi della sperimentazione, tra cui problemi incontrati ed eventuali correzioni ad essi. Si descriveranno inoltre alcune prove di calibrazione dei valori relativi alla fisica del mondo virtuale.

Nella sezione *Direzioni future di ricerca e conclusioni* si riassumono gli scopi, le valutazioni di questi e le prospettive future.

Capitolo 2

Realtà Virtuale



Ultimamente si parla sempre più spesso di *Realtà Virtuale* ma cos'è, come si crea e a cosa serve? Il termine *Realtà Virtuale* fu coniato nel 1989 da Jaron Lanier, una delle prime persone che iniziò a lavorarci. Con il termine Realtà virtuale (Virtual Reality) si indica una Realtà simulata che si realizza attraverso la ricostruzione di ambienti o oggetti in modo da dare al soggetto una percezione il più possibile realistica della loro esistenza. A questa definizione tecnica si può affiancare una più generale, che vede la Realtà virtuale come una Realtà parallela. è possibile distinguere due tipi

di Realtà virtuale: una immersiva e una non immersiva. Si definisce immersiva la Realtà virtuale che è in grado di assorbire l'utente, realizzando una vera e propria immersione dei sensi nell'ambiente tridimensionale generato dal computer. Ciò è possibile attraverso l'utilizzo di dispositivi particolari: un visore, che permette la visualizzazione delle immagini tridimensionali e l'isolamento dall'ambiente esterno, ed un tracker, per il rilevamento di posizione e movimento dell'utente. Oltre a questi ne esistono molti altri, di cui alcuni ancora in fase di studio e perfezionamento. La Realtà virtuale non immersiva, invece, è caratterizzata dall'utilizzo di un monitor per la visualizzazione delle immagini tridimensionali e non si serve dell'ausilio di un tracker determinando nell'utente la sensazione di vedere il mondo tridimensionale, creato dal computer, come attraverso "una finestra" e in maniera quindi non partecipativa.

2.1 Come si crea la Realtà Virtuale

La Realtà Virtuale è possibile grazie a strumenti, programmi e linguaggi di programmazione appositi. Nella tesi proposta, si è scelto di utilizzare una Realtà Virtuale di tipo immersivo. Questa tipologia richiede una strumentazione più complessa e performante rispetto alla tipologia non immersiva. Gli strumenti che richiede, sono i seguenti:

- **Computer:** tutte le periferiche utilizzate fanno capo a un hardware, se non integrato, che funge da fulcro per l'elaborazione e lo smistamento dei dati. Questo fulcro è spesso rappresentato da un computer dalle prestazioni elevate, in termini di CPU e GPU. La sua funzione, oltre a mantenere uno stato aggiornato, è quella di ricevere dati da sensori esterni, utili per recepire le azioni dell'utilizzatore, e inviare informazioni di feedback delle azioni elaborate. Ad esempio l'aggiornamento delle immagini visualizzate e l'attivazione di attuatori per altri tipi di sensazioni (ad esempi tattili).
- **Visore:** costituito da due pannelli LCD impostati per la vista binoculare, questo isola dal mondo esterno e costituisce una specie di casco. Questa impostazione dà la sensazione di tridimensionalità. Il visore può essere dotato di sensori in grado di rilevare i movimenti dell'utente come la rotazione del capo per inquadrare un'altra area del mondo virtuale. In alternativa, esistono visori economici che non

sono altro che contenitori, muniti di lenti, che permettono di inserire lo smartphone e utilizzarlo come schermo LCD e come fulcro di tutte le funzioni hardware e software.

- **Auricolari:** solitamente integrati nei visori, permettono di udire i suoni emessi dal mondo virtuale. Il software deve variare i suoni emessi in base alla posizione dell'utilizzatore.
- **Sensori:** per il riconoscimento dei movimenti e delle azioni dell'utilizzatore. I sensori presenti nel visore danno informazioni sulla posizione, sul movimento della testa e del corpo. I sensori sono di vario tipo e sono in continuo sviluppo. I più conosciuti sono: guanti, in sostituzione dei canonici gestori di input (joystick, mouse, tastiere, ecc.); tute, in grado di trasferire le posture e i movimenti dell'utente nella rappresentazione e Virtuix Omni, una piattaforma che permette di muoversi camminando nell'ambiente virtuale.

Capitolo 3

Architettura del sistema Hardware

In questo capitolo e nel successivo si descriverà la progettazione del sistema creato. Inizialmente si tratterà l'analisi del progetto e le scelte effettuate per quanto riguarda gli strumenti da utilizzare. Successivamente si descriveranno tutti i moduli che compongono l'architettura del sistema. La trattazione è divisa in parte hardware e parte software.

L'obiettivo del progetto è quello di creare uno strumento che permetta all'utilizzatore di pedalare, sterzare e osservare un luogo in un mondo virtuale. In primo luogo, era necessario creare un sistema simile ad una bicicletta. Ai fini di sperimentazione della tipologia di progetto, si è scelto di utilizzare una cyclette. La suddetta, permette di semplificare notevolmente il sistema elettronico e il sistema software, poiché questi non devono tenere conto dell'attrito e del ritorno di forza, in quanto una pedalata farà sempre ruotare il volano. La cyclette è inoltre sprovvista di freni, i quali potrebbero essere utilizzati per frenare la bicicletta virtuale. Si è quindi scelto di ottenere solo le informazioni relative alla pedalata e alla posizione del manubrio. Queste informazioni devono essere elaborate da un microcontrollore che ottiene i dati da tutti i sensori e genera una macro-informatione da inviare al sistema software.

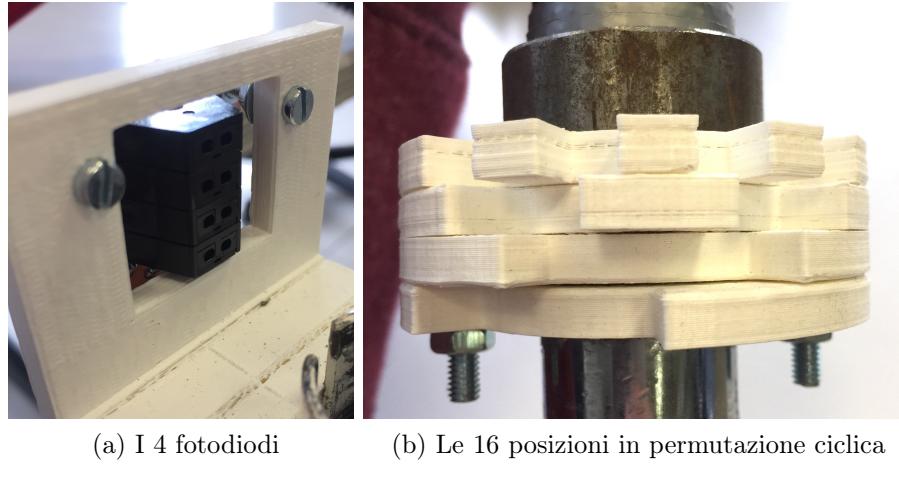


Figura 3.1: Sensori del manubrio

3.1 Sensori del manubrio

Per ottenere le informazioni relative alla posizione del manubrio è stato scelto un angolo massimo di rotazione di 120° . Questo angolo è stato diviso in 16 posizioni.

Per poter rilevare la posizione corrente è stata necessaria una codifica in 4 bit. Per ottenere questa codifica, si è fatto uso di 4 fotodiоди¹ che permettono di rilevare i 4 spazi che compongono una delle 16 configurazioni. I fotodiоди permettono di distinguere uno spazio vuoto da uno spazio pieno creando un segnale elettrico. Le configurazioni sono state realizzate con un oggetto stampato con stampante 3D, come si vede in figura 3.1b, in permutazione ciclica (figura a lato), che permette di accorpare gli *uni* e gli *zeri* e ridurre l’alternanza di *vuoti* e *pieni*.



¹Il fotodiодо è un particolare tipo di diodo fotorilevatore che funziona come sensore ottico sfruttando l’effetto fotovoltaico, in grado cioè di riconoscere una determinata lunghezza d’onda dell’onda elettromagnetica incidente e di trasformare questo evento in un segnale elettrico di corrente applicando ai suoi estremi un opportuno potenziale elettrico. Esso è dunque un trasduttore da un segnale ottico ad un segnale elettrico.

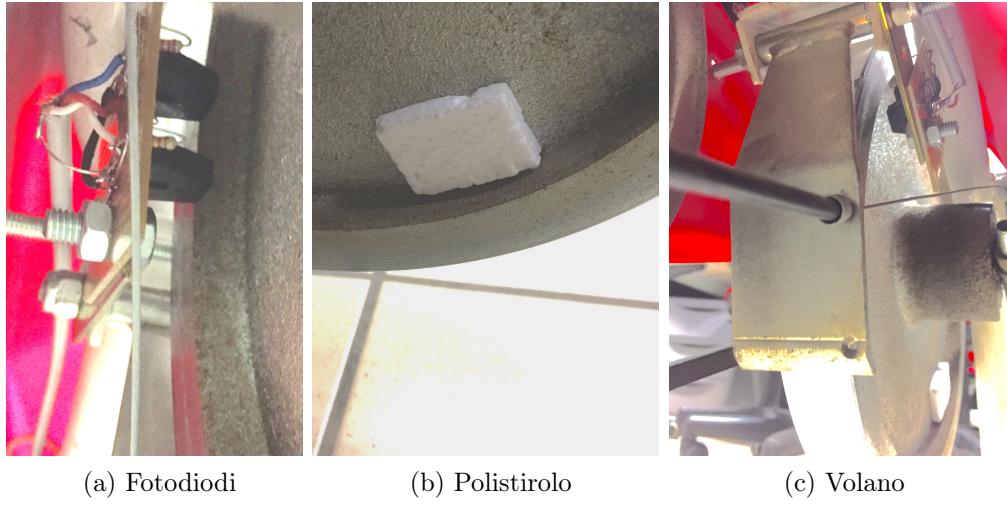


Figura 3.2: Sensori del volano

3.2 Sensori del volano

La bicicletta virtuale deve muoversi ad una velocità consona alla rotazione del volano della cyclette. Quest'ultimo è stato quindi munito di un sistema che fornisce informazioni sulla velocità di rotazione. Il sistema di sensori è composto da due fotodiodi (figura 3.2a) che permettono di ottenere la velocità di rotazione e distinguere se questa è oraria o antioraria. A questo fine, è stato installato sul volano un pezzo di polistirolo (figura 3.2b) che permette di attivare i fotodiodi al passaggio. Grazie a questo, il microcontrollore può ottenere il tempo impiegato dal volano per coprire lo spazio che intercorre tra un fotodiodo e l'altro. La velocità di rotazione è quindi ottenuta tramite formula fisica della velocità, ovvero spazio percorso fratto tempo impiegato. La direzione di rotazione viene ottenuta distinguendo quale dei due fotodiodi viene attivato per primo. Ad esempio, se consideriamo due fotodiodi disposti orizzontalmente:

- Se un oggetto passa da sinistra a destra, si oscurerà prima il fotodiodo sinistro poi il destro, quindi assumendo un oggetto che abbia il fulcro di rotazione sotto i fotodiodi, la sua rotazione è **oraria**
- in caso contrario, la rotazione è **antioraria**.

In figura 3.2c si può osservare l'intero sistema del volano.

3.3 Microcontrollore

Il cuore della parte hardware è la scheda Arduino che elabora tutti i dati ricevuti via cavo o via bluetooth. La scheda con il microcontrollore è in continua comunicazione in input e in output con il modulo bluetooth poiché deve ricevere l'informazione sulla rotazione dall'accelerometro e deve inviare all'elaboratore esterno una stringa contenente tutte le informazioni sotto forma di stringa.



Figura 3.3: La scheda Arduino Micro e il modulo bluetooth

3.3.1 Stringa composta dal microcontrollore

La stringa elaborata dall'Arduino è composta come segue:

- **Posizione del manubrio:** un valore compreso tra 0 e 15 che indica la posizione del manubrio
- **Velocità di rotazione:** un valore compreso tra 0 e 40 che indica la velocità di rotazione della pedalata.
- **Senso di rotazione:** un segno + o - che indica senso orario o antiorario di rotazione della pedalata.

Un esempio pratico è **12+27.5**: significa che si sta registrando una pedalata in senso orario con velocità 27.5 e il manubrio è nella posizione 12.

3.4 Ricevitore Bluetooth

La ricezione della stringa avviene tramite un ricevitore bluetooth collegato ad una porta USB del computer generale in cui viene eseguito il software. Utilizzando un programma terminale apposito, chiamato HyperTerminal è stato possibile leggere la porta seriale COM a cui vengono inviati i dati a 9600 bit/s e testare la parte hardware.

3.5 OSVR



OSVR è l'acronimo di *Open-Source Virtual Reality* ed identifica la piattaforma software open-source per la Realtà Virtuale e quella aumentata. È stata sviluppata dalla Razer e da Sensics. La prima è l'azienda leader nel settore del gaming, mentre la seconda lo è nel settore della realtà virtuale. Affinché questa piattaforma sfondasse nel mercato, le società sopracitate hanno deciso di rendere open-source sia il software sia l'hardware per i programmatori. È infatti reperibile su github il codice sorgente. OSVR consente in modo davvero semplice la configurazione e la gestione dei vari dispositivi: occhiali VR, inseguitori di posizione, periferiche di gioco e vari altri.

3.5.1 Caratteristiche

L'OSVR è composto da un display montato su una visiera, o meglio head-mounted (HMD), da 2 lenti amovibili, da un display OLED da 5.5 pollici con risoluzione 1920 x 1080 pixel, 60 fps di refresh rate e campo visivo di 100 gradi. Contiene una scheda madre riprogrammabile con accelerometro e giroscopio integrati. Il visore OSVR dispone di doppie lenti che consentono di ridurre la distorsione delle immagini. A differenza di altri visori, questo dispone di una cintura elastica che porta i cavi fino all'altezza del muscolo trapezio dell'utilizzatore, in modo da non limitarne il movimento e quindi l'esperienza di gioco.



Figura 3.4: I componenti del visore OSVR HDK1

Il punto forza di questa piattaforma è la facile integrazione con dispositivi e con software aggiuntivi. Ad esempio, utilizzando una telecamera eye-tracking è possibile adoperare il software fornito dal produttore della fotocamera per calcolare la direzione dello sguardo. È inoltre possibile utilizzare il visore in praticamente il 90% dei sistemi operativi. In questo modo lo sviluppatore non ha più bisogno di scegliere anticipatamente un particolare sistema operativo per la sua applicazione, la cui realizzazione richiederà meno tempo sfruttando i plug-in dei quali l'OSVR dispone. Si permette così allo sviluppatore di concentrarsi su essa piuttosto che sull'interfacciamento. Sono inoltre reperibili all'interno di Github², tutti i plug-in di integrazione con i vari motori grafici quali Unity e Unreal Engine.

3.5.2 Integrazione

L'integrazione dell'OSVR verrà trattata nella sezione *Architettura del sistema Software* in cui si descriverà il procedimento per installare, configurare ed utilizzare il visore.

²GitHub è un servizio di hosting per progetti software. Il nome deriva dal fatto che è un servizio sostitutivo del software dell'omonimo strumento di controllo versione distribuito, Git.

Capitolo 4

Architettura del sistema Software

La parte software è stata scritta interamente nel linguaggio C# e si avvale del motore grafico Unity. L'applicazione crea un'ambientazione virtuale in cui viene posizionata una bicicletta a cui viene collegato uno script che legge continuamente la porta COM su cui il microcontrollore invia la stringa contenente l'informazione di movimento. Lo script parsifica la stringa e dà i valori in pasto al motore grafico che muove la bicicletta virtuale.

4.1 Blender

Blender è un software libero e multipiattaforma di modellazione, animazione, compositing e rendering di immagini tridimensionali. Inoltre dispone di funzionalità per mapature UV e simulazioni di rivestimenti adatte alla creazione di applicazioni e giochi 3D. All'interno di Blender tutte le funzioni e le interfacce possono essere richiamate con scorciatoie e per questo motivo quasi tutti i tasti sono collegati a numerosi comandi. La sua interfaccia si basa essenzialmente sulla finestra principale di lavoro, in cui è possibile visualizzare un modello 3D e modificarlo.



L'*Object Mode* è la modalità che permette di visualizzare le mesh¹ dell'og-

¹Una mesh poligonale, anche detta maglia poligonale, è una collezione di vertici, spigoli e facce che definiscono la forma di un oggetto poliedrico nella computer grafica 3D e nella modellazione solida.

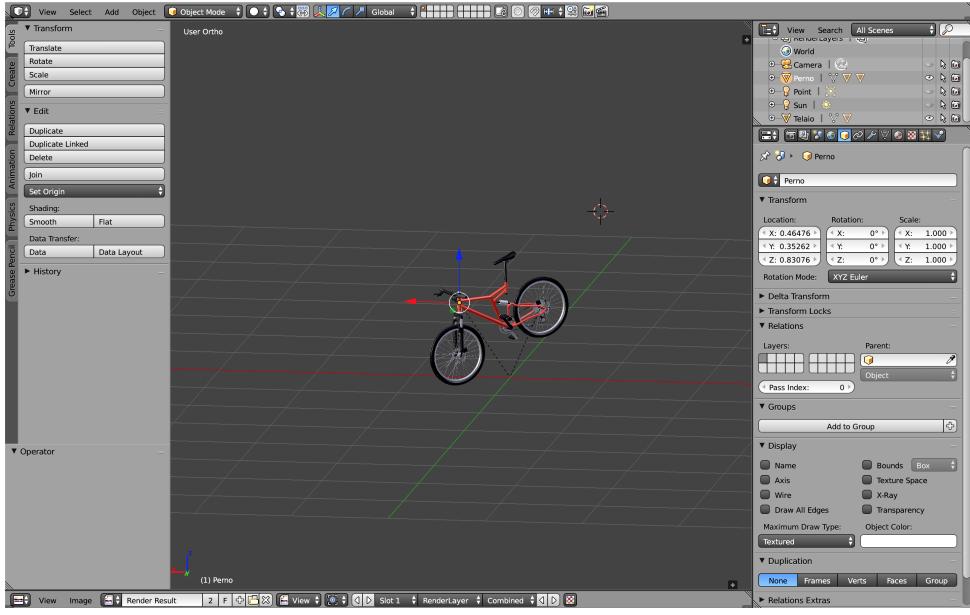


Figura 4.1: Interfaccia Principale di Blender

getto, mentre l'*Edit Mode* permette di modificarlo. Vi sono altre modalità che non sono state utilizzate, ma risultano utili per una modellazione più dettagliata. Per dotare il modello di texture o di materiali, ovvero l'applicazione di colori, trasparenza, lucentezza o altro, Blender implementa diverse finestre come la *UV/Image Editor* nella quale è possibile scegliere la texture da applicare all'oggetto. Blender è stato utilizzato per creare la mesh della bicicletta virtuale da manovrare attraverso script collegati ad un motore grafico.

4.1.1 Mesh della bicicletta

La mesh proviene dal sito TurboSquid. È stata scaricata gratuitamente con estensione FBX² ed è stata adattata per l'utilizzo necessario ai fini del progetto. È stata ridotta in scala ed è stata centrata alle coordinate (0,0,0). La mesh scaricata era già suddivisa in tutti i componenti di una bicicletta classica. Le uniche parti che devono essere dissociate ai fini della tesi sono il manubrio e il telaio. Sono state quindi assemblate tutte le

²FBX è un formato di file di Autodesk che viene supportato dai più comuni software di grafica 3D in quanto è in grado di immagazzinare non solo geometrie, ma anche dati di texture e di animazioni.



mesh che compongono il telaio e tutte quelle che compongono il manubrio. Blender permette di associare un centro ad ogni mesh con la funzione **Set Origin > Origin to Geometry**. Questa permette di fornire, a tempo di esecuzione, una rotazione attorno al centro così definito. Sono state quindi lasciate libere le ruote e sono state centrate nel loro origine, in modo da poter dare una rotazione attorno al proprio asse. La mesh della bicicletta è stata quindi suddivisa come segue:

- **Telaio:** comprende tutto il telaio della bicicletta escluso il manubrio e le ruote.
 - **Ruota posteriore:** è stata separata per darle possibilità di ruotare sul suo asse.
- **Perno:** consiste nell'unione tra ruota e manubrio. Questo oggetto permette di ruotare tutto il manubrio mantenendo però un asse di rotazione inclinato. Comprende:
 - **Manubrio:** contiene il manubrio stesso, la forcella e l'asse di sterzo.
 - **Ruota anteriore:** è stata separata per darle la possibilità di ruotare sul suo asse.

Il telaio è stato separato dal manubrio solo nell'asse di sterzo, ma è stato mantenuto il canotto di sterzo, in modo da poter ruotare il manubrio attorno all'asse passante per il centro del canotto. La mesh così ottenuta è stata esportata ed è stata importata in Unity come oggetto da manovrare tramite script.

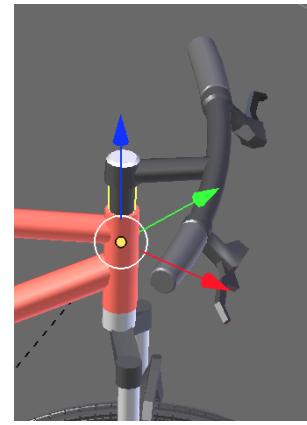
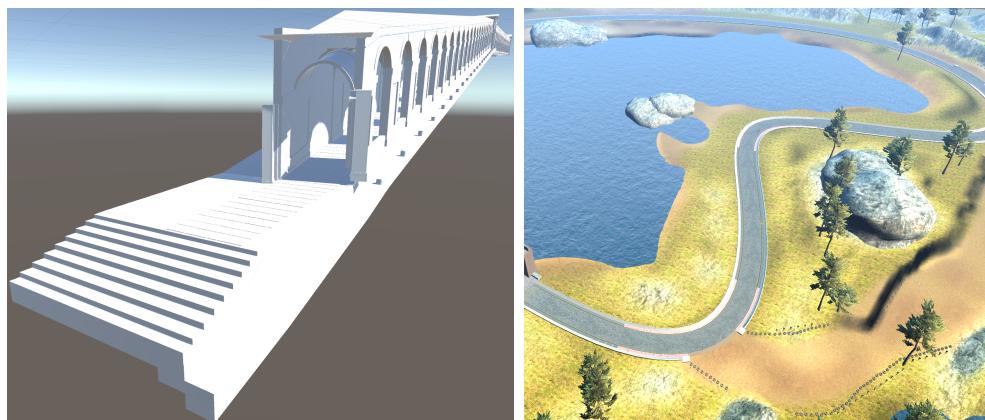


Figura 4.2: L'asse del canotto della bicicletta

4.1.2 Mesh dell'ambientazione

Sono state utilizzate due diverse ambientazioni. La prima delle due è stata fornita dal Cineca: si tratta di un modello 3D dei portici di San Luca (Bologna) ottenuto tramite rilevazione laser. Il modello riporta fedelmente tutti i dettagli spaziali dei portici, ma non riportano la texture. La seconda ambientazione è stata scaricata gratuitamente dallo store di Unity. Per il progetto che è stato creato, l'ambientazione non ha alcuna rilevanza: è possibile asportare la bicicletta e posizionarla in un qualsiasi altro mondo virtuale, purché sia adatto al movimento di una bicicletta al suo interno. L'ambientazione può essere munita di luce principale sia su blender che sul motore grafico. L'importante è non sovrapporre troppe luci per non rischiare la sovraesposizione. In figura 4.3 possiamo notare un esempio delle due ambientazioni.



(a) I portici di San Luca

(b) Il tracciato di gara

Figura 4.3: Le due ambientazioni del progetto

4.2 Unity

Unity è un motore grafico integrato multipiattaforma per la creazione di videogiochi o altri contenuti interattivi 3D, quali visualizzazioni architettoniche o animazioni 3D in tempo reale. Unity permette di modellare l'ambientazione 3D e il modello della bicicletta virtuale attraverso script. Permette inoltre di integrare il visore per la Realtà Virtuale ed è per questo che la scelta è ricaduta su questo motore grafico: per via della facile integrazione dei visori quali OSVR e Oculus Rift.



4.2.1 Schermata Principale

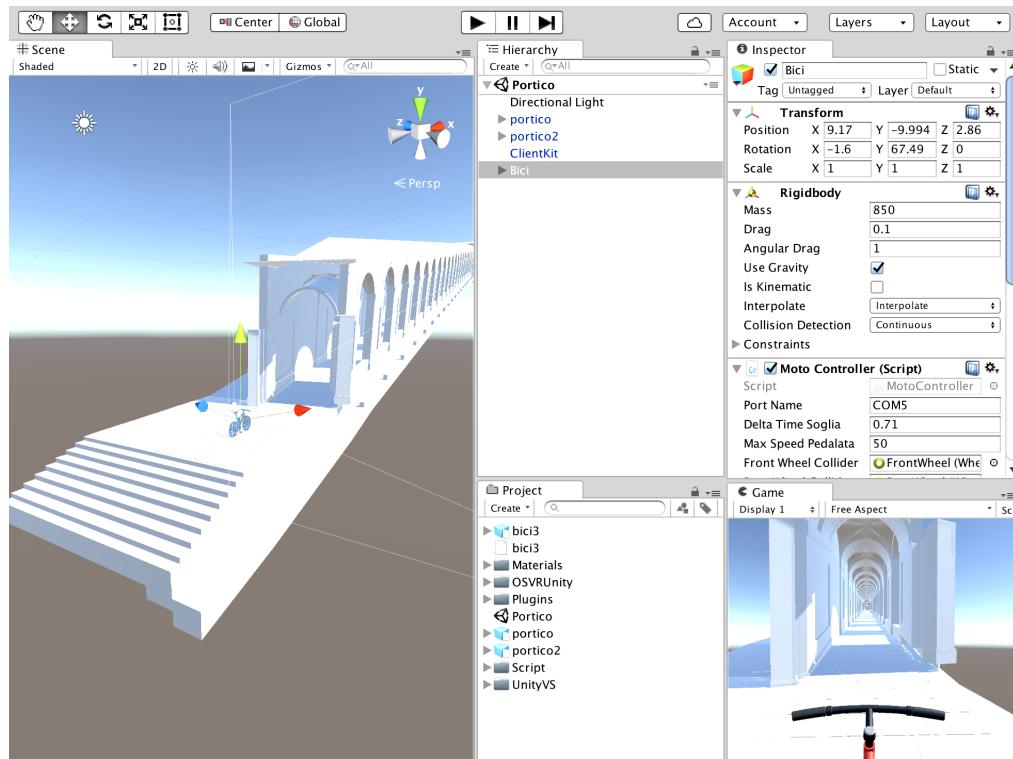


Figura 4.4: La schermata principale di Unity

La schermata principale di figura La schermata principale di Unity si compone di 5 riquadri suddivisi in:

- **Scene:** dove viene mostrata l'architettura dell'ambientazione e degli oggetti. All'interno di quest'area si ha la possibilità di modificare gli

oggetti, scalarli, muoverli e/o ruotarli. È inoltre possibile cambiare visualizzazione e vista utilizzando mouse e tastiera.

- **Game:** in questo riquadro è possibile osservare l'inquadratura della telecamera o del visore quando la simulazione è avviata. È inoltre possibile interagire con tastiera per muovere gli oggetti o spostare la visuale.
- **Inspector:** questo pannello permette allo sviluppatore di modificare tutti i valori relativo ad un oggetto selezionato. Permette inoltre di assegnare script che modellino a tempo di esecuzione l'oggetto, come ad esempio modificandone la posizione e/o creando animazioni per muoverlo.
- **Hierarchy:** in questo punto sono elencati tutti gli oggetti presenti nella scena in ordine di parentela o di directory. Ogni componente può averne al suo interno altre (figlie) e può essere usata per trasferire delle caratteristiche quali la posizione, la rotazione e/o vincolare queste ultime ai comportamenti del padre. Grazie al pulsante Create si possono aggiungere nuovi elementi alla scena. Ogni elemento aggiunto può essere messo in gerarchia semplicemente trascinandolo su un altro con il mouse, questi possono essere oggetti 2D e 3D, telecamere, luci, ecc.
- **Project:** in questa sezione è possibile rintracciare tutte le cartelle e i file del progetto. Questa sezione può interagire con Hierarchy: è infatti possibile trascinare oggetti prefabbricati (come ad esempio la bicicletta creata in Blender) e posizionarli nella scena.

Ogni oggetto che fa parte del progetto è elencato all'interno del riquadro Project ed ogni oggetto che fa parte della scena si può trovare all'interno di Hierarchy. Gli oggetti all'interno di Hierarchy vengono chiamati GameObject e ognuno di questi può essere salvato come componente Prefab, con tutti i suoi figli, e riutilizzato in tutte le scene come oggetto identicamente uguale al GameObject che lo ha originato. Questa funzione risulta utile nel caso servano istanze multiple dello stesso GameObject all'interno di una o più scene. Quando viene modificato il GameObject originario, le variazioni si propagano a tutte le sue istanze nel programma. Selezionando

un GameObject da Hierarchy o anche dalla sezione Project la finestra Inspector si adatta alla caratteristiche in esso contenute. Ogni GameObject possiede al suo interno dei Component che ne definiscono caratteristiche e comportamento.

4.2.2 Scenario Principale

Durante la progettazione sono stati eseguiti due distinti test su due scenari diversi. Il primo test è stato eseguito nel modello dei portici di San Luca ed è servito per testare il comportamento della bicicletta, in relazione alla fisica dell'attrito e dell'inerzia. Il motivo di questa scelta è dovuto alla pendenza dei portici, che portano la bicicletta a discendere per la gravità. Il secondo test è invece stato eseguito nel modello di un percorso di gara, per verificare l'efficenza della pedalata, la velocità e la pendenza della bicicletta durante una sterzata. Tutte le calibrazioni riguardanti la fisica si tratteranno nella sezione *Sperimentazione e correzioni*. In questa sezione si descriverà lo scenario creato inizialmente per testare gli script che modellano il movimento della bicicletta a tempo di esecuzione. La gerarchia

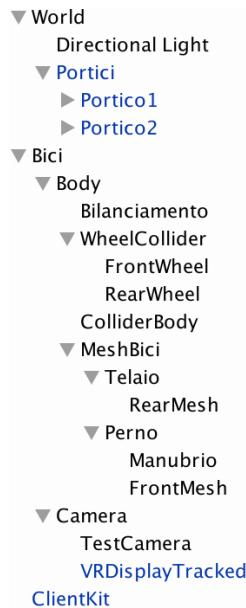


Figura 4.5: La gerarchia di GameObject nello scenario

di GameObject nello scenario è la seguente.

- **World:** Questo oggetto contiene tutti gli elementi del mondo virtuale. Contiene una luce direzionale e può contenere sia il modello dei portici, sia il modello del percorso di gara. Il modello dei portici di San Luca è stato fornito diviso in 2 parti, perciò questi sono stati riuniti in un unico oggetto padre che contiene i due sottogruppi di mesh. Il modello del percorso di gara è stato scaricato da internet come prefabbricato.
- **Bici:** l'oggetto padre contenente altri due oggetti:
 - **Body:** oggetto contenente le mesh e le meccaniche della bicicletta; i figli saranno vincolati a muoversi in concomitanza dei cambiamenti del suo Component Transform. Il body è un RigidBody: componente utilizzato per rappresentare la fisica dei GameObject composto da diversi dati. Tra i principali ritroviamo la massa assegnata, l'utilizzo o meno della gravità sul dato GameObject e le limitazioni di traslazione e rotazione. Al suo interno vi sono:
 - * *Bilanciamento*: un oggetto che rappresenta il centro di massa della bicicletta su cui si concentrerà il peso. Questo è utile quando la bicicletta effettua salti, infatti le permette di non rovesciarsi.
 - * *WheelCollider*: oggetto che contiene i collider delle due ruote, ovvero i GameObject che permettono di collidere con altri oggetti dello stesso tipo. Questi non permettono solamente alla bicicletta di rimanere ancorata ad un terreno e di non attraversarlo, ma ruotando, forniscono anche il movimento.
 - * *ColliderBody*: è stato aggiunto come involucro esterno per la collisione con l'ambiente circostante. Lo scopo è quello di rendere più realistico l'impatto, al momento della collisione della bici con i muri, evitando che le mesh visibili entrino per metà negli oggetti con cui si scontrano, prima di fermarsi.
 - * *MeshBici*: contiene tutte le mesh della bicicletta. È suddiviso in due elementi: *Telaio* e *Perno*. Il telaio contiene la mesh della ruota posteriore e di tutto il telaio escluso il manubrio. Il perno è l'oggetto che contiene il manubrio e

la mesh della ruota anteriore. È una mesh separata poiché permette in base a dove si deciderà di voltare, di avere un feedback visivo della scelta effettuata.

- **Camera:** oggetto contenente due oggetti camera.
 - * *TestCamera*: camera solitamente disattivata. Utilizzata solamente in test senza il visore.
 - * *VRDisplayTracked*: il Prefab³ del visore che sostituisce un normale oggetto camera. Questo oggetto camera permettere di essere ruotata tramite i movimenti del visore.
- **ClientKit:** Prefab di OSVR che si occupa del collegamento con il server e i relativi controlli tra il visore e il calcolatore.

Il funzionamento del WheelCollider è vincolato ad avere almeno un Rigid Body come padre, questo sarà il componente sul quale verranno applicate tutte le forze di spostamento date dalla torsione delle ruote. Il Rigidbody è stato impostato opportunamente ed è stato aggiunto uno script al GameObject per ricevere i segnali di input, per applicare forze alle ruote e per risolvere i limiti della fisica di Unity3D connessi ad esse. Lo script è stato generato vuoto aggiungendo il *Component New Script* attraverso il tasto *Add Component* nel riquadro *Inspector*. Questo script è il *MotoController* e fa uso di una classe *DataReceiver* che si occupa di ricevere i dati dalla cyclette.

³Oggetto prefabbricato di Unity, pronto per essere importato

4.3 DataReceiver script

Questa sezione descrive il comportamento dello script DataReceiver. Questo è stato fornito in versione completa nell'appendice Codice script - DataReceiver a pagina 43. Lo script *DataReceiver* si occupa di leggere i dati inviati dal microcontrollore della cyclette alla porta COM del computer e di interpretarli. È costituito da un costruttore, due funzioni private e due pubbliche:

- **Costruttore:** permette di inizializzare la classe *DataReceiver* per utilizzarla nella classe *MotoController*. Esso inizializza il nome della porta COM utilizzata dal ricevitore Bluetooth e imposta la variabile pubblica *go* a true. Questa variabile accessibile anche dal *MotoController*, determina se il thread associato allo script deve continuare o no la lettura dalla porta.

```
public bool go;
private SerialPort serial;
private string _com;

public DataReceiver(string ComName)
{
    go = true;
    _com = ComName;
}
```

- **Funzioni private:** queste funzioni si occupano di gestire la porta seriale COM.

- **openPort:** si occupa di aprire la porta. Inizializza un oggetto *SerialPort* a 9600bit/s di velocità, senza bit di parità, 8bit di dato e con un solo bit di stop. A questo oggetto assegna un buffer di lettura di dimensione 4096bit e abilita il Data Terminal Ready, un bit di controllo che può indicare l'intenzione di comunicare da parte del dispositivo. È stato necessario configurare un *timeout* di 2 secondi alla lettura della porta. Questo per evitare che, alla chiusura dell'applicazione, il thread associato alla lettura rimanesse in attesa di un dato e bloccasse tutto

il sistema. Infine viene fatta l'apertura della porta. La parte principale del metodo è riportata in seguito.

```
serial = new SerialPort(_com, 9600,
    Parity.None, 8, StopBits.One);
serial.ReadBufferSize = 4096;
serial.DtrEnable = true;
serial.ReadTimeout = 2000;
serial.Open();
```

- **closePort**: si occupa della chiusura della porta seriale COM. Non fa altro che tentare di eseguire `serial.Close()`; catturando eventuali eccezioni.
- **getData**: riceve e restituisce i dati della seriale sotto forma di stringa solo nel caso in cui lo script sia abilitato alla lettura, ovvero la variabile *go* sia impostata a *true*. Il codice eseguito è il seguente:

```
String tmp = "";
if (serial.IsOpen && go)
    tmp = serial.ReadLine();
return tmp;
```

- **Funzioni pubbliche**: si basano sullo stato della variabile booleana *go*. Se è *true*, allora il processo abilitato alla lettura deve procedere, altrimenti deve chiudere la porta seriale e terminare la sua esecuzione. Queste funzioni sono richiamate ricorsivamente in *MotoController* e utilizzano le funzioni private sopra definite.

- **start**: apre la porta con `openPort` ed esegue `getData` iterativamente con un ciclo *while*, finché *go* è *true*. Ogni volta che legge il dato, lo immagazzina in una stringa e lo suddivide in due valori: *sterzo* e *velocità* che poi invia ad Unity tramite l'evento *PedalataTrovata*. Nel caso in cui *go* sia *false*, il processo esce dal ciclo ed esegue la chiusura della porta tramite `closePort`. La parsificazione eseguita all'interno del ciclo è descritta nel modo seguente(rimando alla sottosezione *Stringa composta dal microcontrollore* di pagina 12 per la composizione della stringa inviata dal microcontrollore):

```

//gestione sterzo
tmp = "";
if (data[0] != '0')
tmp += data[0];
tmp += data[1];
sterzo = int.Parse(tmp);

//gestione velocita'
tmp = "";
if (data[3] != '0')
tmp += data[3];
tmp += data.Substring(4);
speed = float.Parse(tmp);
if (speed > 0 && data[2] == '-')
speed *= -1;

//Invio dati a unity
PedalataTrovata(sterzo, speed);

```

- **stop**: semplicemente imposta la variabile *go* a *false*: *go = false*;.. Questo obbliga il processo a terminare l'esecuzione uscendo dal ciclo *while* sopra descritto.

4.4 Motocontroller script

Il MotoController è lo script direttamente applicato al GameObject *Bici* e lo modella in relazione ai dati ricevuti dalla cyclette. Al suo interno sono state inserite delle variabili pubbliche affinché siano accessibili direttamente dall'*Inspector* in modo tale da poter aggiungere i vari GameObject di cui si necessita senza doverli cercare da codice. Unity3D al momento del lancio dell'applicazione si preoccupa di collegare alle variabili gli oggetti che sono stati registrati attraverso la schermata di progettazione, come si può vedere dall'immagine seguente.



Figura 4.6: Le variabili impostabili direttamente dall'inspector di Unity

Lo script è una classe che estende `MonoBehaviour`, una classe di default di Unity per modellare il comportamento di un `GameObject`. I campi dello Script `MotoController.cs` visibili nella schermata *Inspector* di Unity3D per il testing, sono:

- Front/RearWheelCollider, rispettivamente i `WheelCollider` della ruota anteriore e posteriore.
- Front/RearWheelTransform, classi `Transform` collegate ai `GameObject` delle mesh delle ruote per i movimenti nello spazio e la rotazione della ruota anterore, in modo da dare l'effetto di movimento.

- SteeringHandleBar, Transform collegato alla mesh del manubrio per la rotazione nelle curve.
- Centro, componente per il calcolo del centro di massa.
- Body, GameObject che racchiude in sè tutti i componenti mobili appartenenti alla bici i quali dovranno inclinarsi durante una curva.
- Body Vertical Lean, vincolo sull'inclinazione verticale.
- Body Horizontal Lean, vincolo per l'inclinazione orizzontale.
- Engine Torque, forza di torsione della ruota.
- Steer Angle, vincolo sull'angolo di sterzata massima.
- High Speed Steer Angle, angolo di sterzata ad alta velocità.
- High Speed Steer Angle At Speed, angolo di sterzata per velocità.
- Max Speed, limitatore di velocità.
- Brake, forza di frenata.
- Friction, forza di attrito.

Tutte le variabili giocano un ruolo fondamentale per il calcolo del movimento e la visualizzazione conseguente. All'interno delle funzioni che si mostreranno sotto, possiamo notarne il comportamento. Si è tuttavia omesso gran parte del codice per rendere più semplice la lettura di questa sezione. Si rimanda quindi all'appendice Codice script MotoController a pagina 47.

- La funzione **Start** è la prima ad essere invocata subito dopo il popolamento delle variabili pubbliche, grazie ad essa è possibile:
 - Salvare il collegamento al *RigidBody*, in questo caso il *GameObject*, che contiene tutti gli altri, nella variabile *rigid*.
 - Imporre vincoli sul movimento della rotazione dell'asse z, impedendo la rotazione.
 - Fissare il centro di massa calcolandolo partendo da quello standard e dal *GameObject Bilanciamento*

- Stabilire un limite alla velocità angolare
 - Salvare la preferenza per l'angolo di sterzata
 - Inizializzare il Receiver e impostare l'evento PedalataTrovata
 - Creare un thread separato che esegua il codice del Receiver.
- **FixedUpdate** area che va utilizzata per la modifica dei GameObject che possiedono il Component RigidBody. All'interno invochiamo tre funzioni per il controllo degli input, quali l'attivazione del movimento delle ruote, l'eventuale arresto o retromarcia. Queste funzioni sono state definite sotto e sono:
 - Inputs
 - Engine
 - Braking
 - **Update** viene richiamato in concomitanza con **FixedUpdate** ed è indispensabile in quanto è l'unico metodo disponibile per l'aggiornamento dalla grafica. Al suo interno troviamo altre due funzioni utili per aggiornare i movimenti delle mesh su schermo e per controllare l'angolo del nostro assetto verticale:
 - WheelAlign
 - Lean
 - **Inputs** controlla lo stato degli input e modifica le variabili che serviranno per calcolare il movimento che dovrà essere effettuato, nel dettaglio:
 - Salva la velocità corrente convertita.
 - Riporta i dati delle rotazioni imponendo quella sull'asse z a 0.
 - Imposta la variabile che discrimina il movimento in avanti o all'indietro.
 - Inserisce il valore della variabile per l'angolo di sterzata
 - Controlla e imposta la variabile per la retromarcia.
 - **Engine** mette in moto i WheelCollider e permette quindi alla bici-letta di sostenere un moto accelerato. Il procedimento sarà quello di:

- Calcolare l’angolo di sterzata contenuta nelle variabili precedentemente ottenute grazie agli ingressi e imporle al Collider della ruota anteriore.
- Trovare la velocità da inserire. In base al valore che questa assume distinguiamo due comportamenti:
 - * Il primo cessa l’accelerazione per il superamento della velocità massima, subendo l’inerzia fino al momento in cui non si torni al di sotto della soglia consentita riaccelerando.
 - * Il secondo applica la forza di accelerazione il parametro *motorTorque*
- Controllare sul reversing: nel caso in cui l’input sia una velocità negativa, si impone questa pari a:
 - * 0 per velocità maggiori di quella desiderata
 - * una velocità di retromarcia ridotta.
- **Braking** in caso di frenata deceleriamo la bici fino al punto di invertire la marcia. Le opzioni sono:
 - Se la velocità è inferiore da una certa soglia, rilasciando il tasto di input viene applicato un attrito che permetterà alla bici di fermarsi. Esso mantiene il mezzo fermo anche in caso di dislivello.
 - Se applico un valore negativo di input e non siamo in retromarcia, si avrà una frenata più rapida.
 - Se non si è in nessuno dei due precedenti casi, la forza di decelerazione viene resa nulla.
- **WheelAlign** calcola i movimenti che dovranno eseguire le mesh delle ruote e del manubrio in accordo con l’input e i WheelCollider. Le fasi operative saranno:
 - Calcolare il punto di contatto con il terreno.
 - Controllare, tracciando una linea immaginaria dal centro della ruota, se lungo il percorso si incontra un altro Collider.
 - Disegnare la mesh nel punto calcolato in base ai controlli effettuati.

- Adeguare la rotazione in base alla velocità e al tempo del movimento.
- Applicare la torsione alla ruota anteriore in base alla rotazione del FrontWheelCollider.
- Compire i passaggi precedenti anche per la ruota posteriore, ovviamente trascurando la torsione derivante dal manubrio.
- Ruotare il manubrio in base all’angolo di sterzata. Siccome il manubrio è inclinato e non perfettamente perpendicolare al piano di appoggio, è stato necessario utilizzare Eulero attraverso i parametri *eulerAngles*. Questo è quanto viene applicato:
 - * Si ottiene la differenza di angolo tra la posizione corrente (*euler.z*) e l’angolo del FrontWheelCollider.
 - * Si applica la formula $\text{euler.z} = (\text{euler.z} - \text{angleDifference}) \% 360$ per fornire la rotazione sull’asse inclinato
 - * Si applica il valore ottenuto agli *eulerAngles* del manubrio.
- **Lean** calcola e applica l’inclinazione verticale a fronte di una sterzata.
Le operazioni contenute in essa sono:
 - Calcolare l’angolo rispetto alla verticale in base ai dati e al tempo.
 - Salvare i dati del contatto con il terreno del WheelCollider.
 - Utilizzare i dati del contatto per ottenere l’angolo normalizzato in corrispondenza del terreno. Se quest’ultimo, dovesse essere maggiore, rispetto a quello desiderato, verrà limitato da un lato o dall’altro.
 - Calcolare l’angolo orizzontale.
 - Generare gli angoli di rotazione e applicarli al Body.
 - Ricalcolare il centro di massa per non avere effetti anomali, per esempio di trascinamento orizzontale.
- **ReceiverPedalataTrovata** funzione per gestire l’oggetto evento di *DataReceiver* e aggiornare le variabili che immagazzinano i valori di sterzo e velocità di pedalata.

- `OnApplicationQuit` funzione che ridefinisce la chiusura dell'applicazione. Richiama la funzione `stop` di `DataReceiver` e si sospende in attesa della terminazione del thread associato.
- `msgToDebug` funzione utile per debug. Permette di scrivere direttamente nella console di debug, una stringa passata come argomento.

Un volta scritto, lo script `Motocontroller` deve essere associato all'oggetto Bici, come in figura 4.6.

4.5 OSVR

Il visore OSVR deve semplicemente sostituire l'oggetto camera con un suo prefabbricato. Questo prefabbricato è impostato per muoversi in relazione ai movimenti del visore. Per utilizzare i prefabbricati e per poter utilizzare il visore è stato necessario installare il software fornito dal portale sviluppatori su Github. I pacchetti che sono stati utilizzati sono tre: il primo per la configurazione del visore e per il controllo del suo funzionamento, il secondo per l'abilitazione dell'accelerometro e del giroscopio tramite l'utilizzo del server (un processo locale) e il terzo per importare sia i componenti da utilizzare sia le scene di testing all'interno di Unity. Utilizzando il secondo pacchetto contenente il server, un processo locale che utilizza un file .json per la configurazione del visore (e quindi è possibile crearsene di personalizzati), si è abilitato il tracker per il rilevamento della posizione e per lo spostamento dell'accelerometro. Completata la configurazione del visore, è stato possibile integrarlo con Unity. Dei componenti importati e necessari su Unity sono stati adoperati il `ClientKit`, che si occupa del collegamento con il server e i relativi controlli tra il visore e il calcolatore, e il `VRVRDisplayTracked` già dotato della camera, che consente di visualizzare la simulazione, e degli script per il movimento all'interno del mondo.

Capitolo 5

Sperimentazione e correzioni

5.1 Realizzazione del sistema

Il sistema assemblato e impostato per il funzionamento si presenta come in figura.



Figura 5.1: Il sistema assemblato

Per avere un corretto funzionamento di tutto il sistema è necessario seguire obbligatoriamente una determinata serie di operazioni nell'esatto ordine:

- il computer principale deve essere acceso, deve avere il ricevitore bluetooth collegato e deve avere il progetto di Unity già pronto per essere avviato
- collegare alla corrente il microcontrollore della cyclette
- prendere l'hub dell'OSVR e collegarlo alla corrente
- collegare il visore all'hub
- prendere il cavo HDMI-USB dell'OSVR e collegare HDMI e USB all'hub
- collegare l'usb al computer
- collegare l'HDMI al computer
- eseguire il server OSVR nel computer
- far partire l'applicazione premendo il tasto *play* su Unity.
- una volta partito, è necessario spostare la schermata *game* nel monitor virtuale a destra, poiché l'OSVR viene rilevato dal computer come un monitor.
- a questo punto è possibile indossare il visore e cominciare ad utilizzare la bicicletta.

5.2 OSVR e le impostazioni iniziali

Nei primi test eseguiti con l'OSVR appena installato, si è riscontrato un bug che, all'avviamento della simulazione, posizionava la telecamera ruotata di 90° verso destra e un'inversione di rotazione tra roll (su Z) e pitch (su Y). La chiarificazione di questi movimenti si ha in figura 5.2. Il problema è

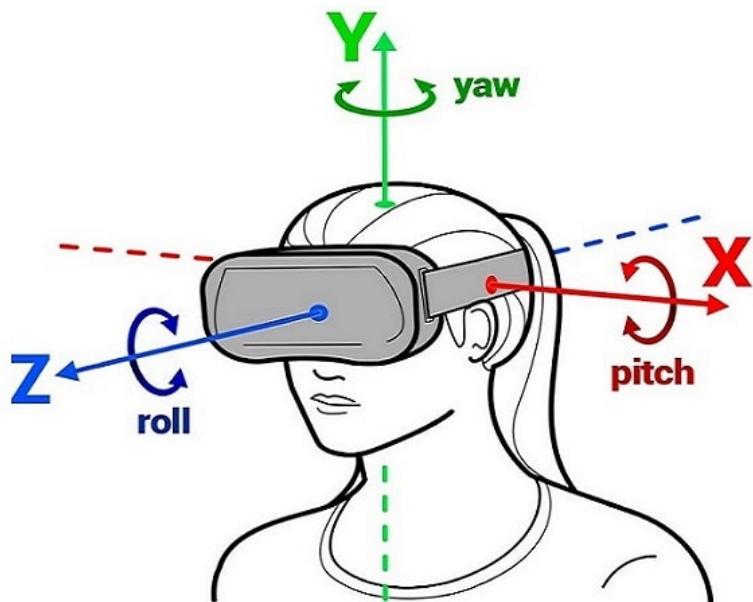


Figura 5.2: Le varie rotazioni della testa

che l'OSVR necessita di una calibrazione iniziale che non era stata eseguita dopo l'installazione. Inizialmente si è risolto il problema posizionando la bicicletta a (0,0,0) con l'oggetto *VRDisplayTracked* interno alla bicicletta con posizione (0,0,0) e tutte le rotazioni a 0. In questo modo le rotazioni a tempo di esecuzione erano tutte corrette. Tuttavia, esiste un tool fornito dal portale di sviluppo di OSVR che permette di resettare il sistema di coordinamento del visore, chiamato *OSVRResetYaw*. Lanciando l'eseguibile di questo tool, è possibile resettare la posizione e le rotazioni del visore, portandolo ai valori impostati su Unity.

5.3 Calibrazioni della fisica virtuale

Testando la bicicletta nel tracciato di gara citato nella sottosezione Le due ambientazioni del progetto a pagina 18 si sono potuti definire al meglio

i valori della fisica del mondo virtuale. I problemi principali erano: la velocità da fornire dopo una pedalata, l'inerzia e la frenata in caso di pedalata nulla.

5.3.1 Velocità

La velocità è stata limitata al valore *40* che rappresenta una velocità poco superiore alla realtà, ma rende realistico il modo in cui la bicicletta si muove nel mondo virtuale. Il valore *MaxSpeedPedalata* è stato impostato a *20* e rappresenta, in modo inversamente proporzionale, quanto una pedalata può influire sulla velocità: più piccolo è, maggiore sarà l'accelerazione che ne deriva. Il valore che principalmente influenza la velocità della bicicletta è *Engine Torque*. Questo è stato impostato a *1500* per avere una buona accelerazione ad ogni pedalata, ma è stato raggiunto per tentativi. Sotto i *1000* si otteneva una scarsa accelerazione, mentre sopra i *3000* si otteneva una velocità non realistica che portava la bicicletta a impennarsi. Per evitare l'effetto impennata è stato spostato l'oggetto *Bilanciamento* fino ad un punto in cui la bicicletta non si ribaltava neanche durante dei salti.



Figura 5.3: Il punto in cui viene applicato il centro di massa

5.3.2 Frenata e Inerzia

I valori che influenzano la frenata sono *Brake* e *Friction*. Il primo è stato impostato a 2500 poiché con valori inferiori la bicicletta frena troppo lentamente e con valori superiori, faticava addirittura a partire. Il secondo è stato impostato a 6, poiché per valori molto superiori, la ruota posteriore slitta. Entrambi i valori sono stati impostati uno relativo all'altro durante la sperimentazione. L'inerzia viene gestita tramite questi valori nello script `MotoController` nella funzione `Braking`.

Capitolo 6

Direzioni future di ricerca e conclusioni

L'obiettivo di questo progetto è creare un simulatore che permetta all'utente di pedalare ed immedesimarsi in un mondo virtuale. I risultati ottenuti sono discreti, ma non tanto realistici quanto voluto. Per quanto possa essere accessibile, l'OSVR ha prestazioni piuttosto ridotte se viene utilizzato con ambientazioni con molti dettagli. Per la realtà virtuale è necessario un computer con elevate prestazioni e un visore con un'ottima risoluzione. Per la realizzazione del progetto non è stato possibile utilizzare strumenti con prestazioni migliori. Non usando queste caratteristiche, il risultato che si ottiene sono simulazioni non molto realistiche e spesso fastidiose: dopo diverse ore l'utente può accusare affaticamento alla vista, stanchezza e nel-
lo specifico cefalea e nausea.

Un altro problema è dovuto alla cyclette e al fatto che i dati ricavati dal suo utilizzo siano poco realistici. Non è infatti possibile fermare i piedi sui pedali affidandosi all'inerzia come in una reale bicicletta, poiché la cyclette ha i pedali direttamente ancorati al sistema del volano. Fermare i piedi sulla cyclette porta a fermare direttamente il volano, ma l'applicazione genera comunque l'inerzia sulla bicicletta virtuale. Inoltre, se nel modello vi fosse una salita, la bicicletta virtuale salirebbe più lentamente, ma l'utente che utilizza la cyclette non avrebbe nessun ritorno di forza.

6.1 Soluzione

Per sopperire a questi due problemi è in corso la progettazione di un sistema più sofisticato, che fa uso di una bicicletta vera e di rulli che permettano di dare un ritorno di forza in caso di pendenze. Un esempio è quello in figura.



Figura 6.1: Un esempio di bicicletta con rulli

Inoltre, in futuro il sistema verrà aggiornato con visore molto più sofisticato: l'Oculus Rift. Questo visore ha una risoluzione più elevata e garantisce un'esperienza utente molto più realistica dell'OSVR. Tuttavia, necessita un computer con prestazioni molto più elevate. Verrà infatti installata una scheda video ASUS Strix NVidia GeForce GTX 1080, che è VR Ready, ovvero pronta per poter supportare la realtà virtuale e portare l'esperienza utente ai massimi livelli. Grazie alla popolarità di Unity, l'Oculus Rift è perfettamente integrabile ed esistono pacchetti che permettono di farlo con grande facilità.

6.2 Conclusioni

Durante la progettazione, il lavoro è stato svolto con un occhio di riguardo alle future implementazioni. Si è quindi mantenuto un profilo generale anche nei dettagli, che permettesse una facile integrazione di una bicicletta vera. Nonostante la inevitabile eliminazione o variazione di tutto ciò che riguarda la cyclette e l'hardware associato, il simulatore software è già pronto per essere riutilizzato nel sistema con i rulli. L'unica modifica, di facile realizzazione, sarà quella di modificare la velocità in base alle informazioni che il sistema hardware fornirà riguardo alla pendenza. Gli sviluppi futuri e le integrazioni che si stanno applicando, porteranno il simulatore ad un livello di realismo considerevole, il quale potrà contribuire a portare la realtà virtuale nella vita di tutti i giorni.

Appendice A

Codice script DataReceiver

```
using System;
using UnityEditor;
using System.IO;
using System.IO.Ports;
using System.Threading;

namespace SerialLibrary
{
    public class DataReceiver
    {
        public bool go;
        private SerialPort serial;
        private string _com;
        public event Action<int, float> PedalataTrovata;

//=====COSTRUTTORE=====
        public DataReceiver(string ComName)
        {
            go = true;
            _com = ComName;
        }

//==FUNZIONI PRIVATE per gestione porta seriale==
        private bool openPort()
        {
            try
            {
                serial = new SerialPort(_com, 9600, Parity.None, 8, StopBits.One);
                serial.ReadBufferSize = 4096;
                serial.DtrEnable = true;
                serial.ReadTimeout = 2000;

                serial.Open();
            }
            catch (IOException e)
            {
                MotoController.msgToDebug(e.ToString());
                return false;
            }
        }
    }
}
```

```

        }
        MotoController.msgToDebug("Is open: " + serial.IsOpen);

        return serial.IsOpen;
    }
    private bool closePort()
    {
        try
        {
            serial.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine(e.ToString());
            return false;
        }

        return true;
    }
    private string getData()
    {
        String tmp = "";
        if (serial.IsOpen && go)
            tmp = serial.ReadLine();
        return tmp;
    }

//==FUNZIONI PUBBLICHE per gestione dati ==
    public void stop()
    {
        go = false;
    }

    public void start()
    {
        go = openPort();

        string data;
        string tmp = "";
        int sterzo;
        float speed;

        while (go)
        {
            try
            {
                //ricezione dati dalla porta seriale
                data = getData();

                if (data != "")
                {
                    //gestione sterzo
                    tmp = "";
                    if (data[0] != '0')
                        tmp += data[0];
                    tmp += data[1];

```

```

sterzo = int.Parse(tmp);

//gestione velocita'
tmp = "";
if (data[3] != '0')
    tmp += data[3];
tmp += data.Substring(4);
speed = float.Parse(tmp);
if (speed > 0 && data[2] == '-')
    speed *= -1;

//Invio dati a unity
PedalataTrovata(sterzo, speed);
}

}
catch (TimeoutException)
{
    if (!go)
        closePort();
}
closePort();
MotoController.msgToDebug("Thread: processo terminato.");
}
}
}

```


Appendice B

Codice script MotoController

```
using UnityEngine;
using SerialLibrary;
using System.Threading;
using System.IO;

public class MotoController : MonoBehaviour {

    private SerialLibrary.DataReceiver receiver;
    private float pedalata;
    private float deltaTimePedalata;

    public string PortName = "COM5";

    public float deltaTimeSoglia = 0.71f;
    public float maxSpeedPedalata = 50.0f;

    private Rigidbody rigid;

    public WheelCollider FrontWheelCollider;
    public WheelCollider RearWheelCollider;
    public Transform FrontWheelTransform;
    public Transform RearWheelTransform;
    public Transform SteeringHandlebar;
    public Transform Centro;

    //Bike Body Lean
    public GameObject body;
    public float bodyVerticalLean = 10.0f;
    public float bodyHorizontalLean = 10.0f;
    private float horizontalLean = 0.0f;
    private float verticalLean = 0.0f;

    //Configurations
    public float EngineTorque = 1500f;

    [HideInInspector]
    public float SteerAngle = 30f;
```

```

public float Speed;
public float highSpeedSteerAngle = 40f;
public float highSpeedSteerAngleAtSpeed = 80f;
public float maxSpeed = 180f;
public float Brake = 2500f;
public float friction = 6f;

private float motorInput = 0f;
private float defsteerAngle = 0f;
private float RotationValue1 = 0f;
private float RotationValue2 = 0f;

[HideInInspector]
public float steerInput = 0f;
private bool reversing = false;
private float sterzata;
private int lastInputSterzata;
private float lastSterzata;
private Vector3 euler;
private Vector3 FrontEuler;
private Thread thread;

void Start() {
    //Rigidbody
    rigid = GetComponent<Rigidbody>();
    rigid.constraints = RigidbodyConstraints.FreezeRotationZ;
    rigid.centerOfMass = new Vector3(Centro.localPosition.x *
        transform.localScale.x, Centro.localPosition.y * transform.localScale.y,
        Centro.localPosition.z * transform.localScale.z);
    rigid.maxAngularVelocity = 2f;

    defsteerAngle = SteerAngle;
    lastInputSterzata = 7;

    receiver = new DataReceiver(PortName);
    receiver.PedalataTrovata += Receiver_PedalataTrovata;
    thread = new Thread(receiver.start);

    euler = SteeringHandlebar.localEulerAngles;
    FrontEuler = FrontWheelTransform.localEulerAngles;

    thread.Start();
}

private void Receiver_PedalataTrovata(int inputSterzo, float inputPedalata)
{
    if (lastInputSterzata == 0 && inputSterzo == 15)
        inputSterzo = 0;

    pedalata = inputPedalata/maxSpeedPedalata;

    sterzata = (inputSterzo - 7.0f) / 8; // Mathf.Lerp(lastSterzata,
    (inputSterzo - 7.0f) / 8, 0.2f);

    lastInputSterzata = inputSterzo;
    lastSterzata = sterzata;
}

```

```

}

void FixedUpdate()
{
    Inputs();
    Engine();
    Braking();
}

void Update()
{
    WheelAlign();
    Lean();
}

void Inputs()
{
    Speed = rigid.velocity.magnitude * 3.6f;

    transform.eulerAngles = new Vector3(transform.eulerAngles.x,
    transform.eulerAngles.y, 0);

    motorInput = pedalata + Input.GetAxis("Vertical");
    steerInput = sterzata + Input.GetAxis("Horizontal");

    if (motorInput < 0)
        reversing = true;
    else
        reversing = false;
}

void Engine()
{
    FrontWheelCollider.steerAngle = SteerAngle * steerInput;

    if (Speed > maxSpeed)
    {
        RearWheelCollider.motorTorque = 0;
    }
    else if (!reversing)
    {
        RearWheelCollider.motorTorque = EngineTorque * Mathf.Clamp(motorInput,
        0f, 1f);
    }

    if (reversing)
    {
        if (Speed < maxSpeed)//test con 10
        {
            RearWheelCollider.motorTorque = (EngineTorque * motorInput) / 5f;
        }
        else
        {
            RearWheelCollider.motorTorque = 0;
        }
    }
}

```

```

        }

    }

    public void Braking()
{
    // Deceleration.
    if (Mathf.Abs(pedalata) <= .05f)
    {
        FrontWheelCollider.brakeTorque = (Brake) / friction; // 25f;
        RearWheelCollider.brakeTorque = (Brake) / friction; // 25f;
    }
    else if (motorInput < 0 && !reversing)
    {
        FrontWheelCollider.brakeTorque = (Brake) * (Mathf.Abs(motorInput) / 5f);
        RearWheelCollider.brakeTorque = (Brake) * (Mathf.Abs(motorInput));
    }
    else
    {
        FrontWheelCollider.brakeTorque = 0;
        RearWheelCollider.brakeTorque = 0;
    }
}

void WheelAlign()
{
    RaycastHit hit;
    WheelHit CorrespondingGroundHit;
    float extension_F;
    float extension_R;

    Vector3 ColliderCenterPointFL =
    FrontWheelCollider.transform.TransformPoint(FrontWheelCollider.center);
    FrontWheelCollider.GetGroundHit(out CorrespondingGroundHit);

    if (Physics.Raycast(ColliderCenterPointFL,
    -FrontWheelCollider.transform.up, out hit,
    (FrontWheelCollider.suspensionDistance + FrontWheelCollider.radius) *
    transform.localScale.y))
    {
        if (hit.transform.gameObject.layer != LayerMask.NameToLayer("Bici"))
        {
            FrontWheelTransform.transform.position = hit.point +
            (FrontWheelCollider.transform.up * FrontWheelCollider.radius) *
            transform.localScale.y;
            extension_F = (
            -FrontWheelCollider.transform.InverseTransformPoint(
            CorrespondingGroundHit.point).y - FrontWheelCollider.radius) /
            FrontWheelCollider.suspensionDistance;
        }
    }
    else
    {
        FrontWheelTransform.transform.position = ColliderCenterPointFL -
        (FrontWheelCollider.transform.up * FrontWheelCollider.suspensionDistance)
    }
}

```

```

        * transform.localScale.y;
    }
    //rotazione ruota anteriore: x fissa, y data dal movimento, z data dal perno
    RotationValue1 += FrontWheelCollider.rpm * (6) * Time.deltaTime;
    FrontEuler.x = 0;
    FrontEuler.y = -RotationValue1;
    FrontWheelTransform.localEulerAngles = FrontEuler;

    Vector3 ColliderCenterPointRL =
    RearWheelCollider.transform.TransformPoint(RearWheelCollider.center);
    RearWheelCollider.GetGroundHit(out CorrespondingGroundHit);

    if (Physics.Raycast(ColliderCenterPointRL,
    -RearWheelCollider.transform.up, out hit,
    (RearWheelCollider.suspensionDistance + RearWheelCollider.radius) *
    transform.localScale.y))
    {
        if (hit.transform.gameObject.layer != LayerMask.NameToLayer("Bici"))
        {
            RearWheelTransform.transform.position = hit.point +
    (RearWheelCollider.transform.up * RearWheelCollider.radius) *
    transform.localScale.y;
            extension_R = (
            -RearWheelCollider.transform.InverseTransformPoint(
            CorrespondingGroundHit.point).y - RearWheelCollider.radius) /
    RearWheelCollider.suspensionDistance;
        }
    }
    else
    {
        RearWheelTransform.transform.position = ColliderCenterPointRL -
    (RearWheelCollider.transform.up * RearWheelCollider.suspensionDistance) *
    transform.localScale.y;
    }
    RotationValue2 += RearWheelCollider.rpm * (6) * Time.deltaTime;
    RearWheelTransform.transform.rotation =
    RearWheelCollider.transform.rotation * Quaternion.Euler(RotationValue2,
    RearWheelCollider.steerAngle, RearWheelCollider.transform.rotation.z+90);

    //rotazione del manubrio
    if (SteeringHandlebar) {
        //ottengo la differenza di angolo tra la posizione corrente (euler.z) e l'angolo della wheelcollider
        float angleDifference = euler.z - FrontWheelCollider.steerAngle;

        euler.z = (euler.z - angleDifference) % 360; //applico la differenza
        SteeringHandlebar.localEulerAngles = euler; // aggiorno la rotazione
    }
}

void Lean()
{
    verticalLean = Mathf.Clamp(Mathf.Lerp(verticalLean,
    transform.InverseTransformDirection(rigid.angularVelocity).x *
    bodyVerticalLean, Time.deltaTime * 5f), -10.0f, 10.0f);
}

```

```

WheelHit CorrespondingGroundHit;
FrontWheelCollider.GetGroundHit(out CorrespondingGroundHit);

float normalizedLeanAngle =
Mathf.Clamp(CorrespondingGroundHit.sidewaysSlip, -1f, 1f);

if (transform.InverseTransformDirection(rigid.velocity).z > 0f)
    normalizedLeanAngle = -1;
else
    normalizedLeanAngle = 1;

horizontalLean = Mathf.Clamp(Mathf.Lerp(horizontalLean,
(transform.InverseTransformDirection(rigid.angularVelocity).y *
normalizedLeanAngle) * bodyHorizontalLean, Time.deltaTime * 3f), -50.0f,
50.0f);

Quaternion target = Quaternion.Euler(verticalLean,
body.transform.localRotation.y + (rigid.angularVelocity.z),
horizontalLean);
body.transform.localRotation = target;

rigid.centerOfMass = new Vector3((Centro.localPosition.x) *
transform.localScale.x, (Centro.localPosition.y) *
transform.localScale.y, (Centro.localPosition.z) *
transform.localScale.z);
}

void OnApplicationQuit()
{
    receiver.stop();
    thread.Join();
    Debug.Log("Chiusura del programma: " + (thread.ThreadState));
}

public static void msgToDebug(string txt)
{
    Debug.Log(txt);
}
}

```

Elenco delle figure

3.1	Sensori del manubrio	10
3.2	Sensori del volano	11
3.3	La scheda Arduino Micro e il modulo bluetooth	12
3.4	I componenti del visore OSVR HDK1	14
4.1	Interfaccia Principale di Blender	16
4.2	L'asse del canotto della bicicletta	18
4.3	Le due ambientazioni del progetto	18
4.4	La schermata principale di Unity	19
4.5	La gerarchia di GameObject nello scenario	21
4.6	Le variabili impostabili direttamente dall'inspector di Unity	27
5.1	Il sistema assemblato	33
5.2	Le varie rotazioni della testa	35
5.3	Il punto in cui viene applicato il centro di massa	36
6.1	Un esempio di bicicletta con rulli	40

Bibliografia

- [1] **Arduino.** <https://www.arduino.cc>.
- [2] **Blender.** <https://www.blender.org>.
- [3] **HyperTerminal.** <http://www.hilgraeve.com/hyperterminal-serial-port/>.
- [4] **Oculus Rift.** <https://www3.oculus.com/en-us/rift/>.
- [5] **OSVR Guide.** <http://osvr.github.io/>.
- [6] **OSVR Reset Yawn.** <http://resource.osvr.com/docs/OSVR-Core/OSVRResetYaw.html>.
- [7] **OSVR site.** <http://www.osvr.org>.
- [8] **TurboSquid.** <http://www.turbosquid.com/Search/3D-Models/free>.
- [9] **Unity3D.** <https://unity3d.com>.
- [10] **Virtux Omni.** <http://www.virtuix.com>.