Universidade de Brasília
Faculdade de Tecnologia

Estimação de velocidade de caminhada por sensores inerciais de movimento em ambiente imersivo de realidade virtual

Ricardo Hideki Ito

PROJETO FINAL DE CURSO
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília
2023

Universidade de Brasília
Faculdade de Tecnologia

Estimação de velocidade de caminhada por sensores inerciais de movimento em ambiente imersivo de realidade virtual

Ricardo Hideki Ito

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Roberto de Souza Baptista - FGA/UnB

Brasília
2023

Ito, Ricardo Hideki.
I89e Estimação de velocidade de caminhada por sensores inerciais de movimento em ambiente imersivo de realidade virtual / Ricardo Hideki Ito; orientador Roberto de Souza Baptista - FGA/UnB. –– Brasília, 2023.
86 p.

Projeto Final de Curso (Engenharia de Controle e Automação) –– Universidade de Brasília, 2023.

1. Realidade virtual. 2. Unidade de medição inercial. 3. Reabilitação. 4. Análise de movimento. I. - FGA/UnB, Roberto de Souza Baptista, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Estimação de velocidade de caminhada por sensores inerciais de movimento em ambiente imersivo de realidade virtual

Ricardo Hideki Ito

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 13 de Fevereiro de 2023:

Prof. Roberto de Souza Baptista FGA/UnB
Orientador

Prof. Ana Cristina de David, FEF/Unb
Examinador interno

Prof. Renato Coral Sampaio, FGA/UnB
Examinador interno

Brasília
2023

Este trabalho é dedicado à minha família e amigos que nunca desistiram de mim, mesmo quando eu mesmo já havia desistido.

Agradecimentos

Agradeço principalmente a minha mãe, por todo o carinho e sufoco que passamos juntos, por me aguentar até hoje em casa e por tudo que já fez e ainda faz por mim. Ao meu pai por sempre acreditar em mim, por ser uma pessoa compreensível e de confiança quando eu mais precisei. Aos meus irmãos, que apesar das brigas, a gente sempre vai torcer para o sucesso um do outro. Aos meus amigos também que chamo de "melhores"por não serem sempre a melhor influência nos meus momentos de estudo, mas com certeza são a melhor influência que tenho para a vida. Deixar citado alguns nomes de muita importância para mim, mas sem ordem de preferência: Tsuzaki, Luan, Halafi, Helton, Paulo, Rômulo, Tutu, Piapi, Babidi, André, Tailândia, Takata, Tonin, Carlos, Bichara, Luísa, Gustavo, Lemos. E o Lucas Tsuzaki também pelos energéticos, ao grupo das Qualquerzinhas por todas as risadas, e ao meu gato Shogun por ser um ótimo companheiro. Obrigado, sem vocês eu não teria conseguido... ou teria também, mas foi para dar ênfase na importância de vocês... haha. Obrigado por tudo.

Ricardo Hideki Ito

“There is no harm in doubt and skepticism,
for it is through these that new discoveries are made.”
(Richard Feynman)

Resumo

Neste estudo, explora-se o uso da tecnologia de realidade virtual (VR) e da unidade de medida inercial (IMU) para determinar a velocidade de caminhada no contexto de reabilitação. É utilizado o motor de desenvolvimento de jogos Unity para criar um ambiente virtual que simula a experiência de caminhar a baixa velocidade e com os movimentos do corpo inteiro para uma melhor imersão do usuário. A velocidade é calculada a partir dos dados de orientação dos sensores enquanto a pessoa caminha e a movimentação do avatar é obtida pelo conjunto de equipamentos de VR e sensores. Os resultados mostram que essa abordagem permite uma experiência mais imersiva e realista, bem como uma estimação precisa da velocidade de caminhada.

Este projeto demonstra o potencial da VR e da IMU como ferramentas de reabilitação, bem como para proporcionar experiências imersivas para uma ampla gama de aplicações. Ele também destaca a utilidade das plataformas de desenvolvimento de jogos para criar ambientes virtuais realistas e a importância de se considerar o corpo inteiro em simulações virtuais. O objetivo é criar um sistema de comunicação entre um ambiente/objeto virtual e dispositivos físicos para visualização e análise tridimensional de movimentos do corpo em tempo real, permitindo também a análise de pacientes em qualquer localidade.

Palavras-chave: Realidade virtual. Unidade de medição inercial. Reabilitação. Análise de movimento.

Abstract

In this study, we explore the use of virtual reality (VR) and inertial measurement unit (IMU) technology to determine walking velocity in the context of rehabilitation. The Unity game development engine is used to create a virtual environment that simulates the experience of walking at low speed and with the movements of the whole body for better immersion of the user. The velocity is calculated from the orientation data of the sensors while the person walks and the avatar's movement is obtained by the combination of VR equipment and sensors. The results show that this approach allows for a more immersive and realistic experience, as well as a precise estimate of walking velocity.

This project demonstrates the potential of VR and IMU as rehabilitation tools, as well as providing immersive experiences for a wide range of applications. It also highlights the usefulness of game development engines to create realistic virtual environments and the importance of considering the entire body in virtual simulations. The goal is to create a system for communication between a virtual environment/object and physical devices for real-time three-dimensional visualization and analysis of body movements, allowing for the analysis of patients in any location.

Keywords: Virtual reality. Inertial measurement unit. Rehabilitation. Movement analysis.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Este capítulo contextualiza a motivação do projeto, a definição do problema e o processo de desenvolvimento de uma solução.

1.1 Situação atual

A cada ano, milhões de pessoas são afetadas por acidentes de trabalho, guerras, doenças e outras adversidades que podem causar danos temporários ou permanentes à saúde e bem-estar físico. De acordo com o Ministério da Saúde do Brasil, em 2018 foram registrados mais de 400 mil acidentes de trabalho no país, resultando em mais de 10 mil mortes (SAÚDE, 2018). Além disso, a Organização Mundial da Saúde (OMS) estima que cerca de 15% da população mundial vive com algum tipo de deficiência, sendo que a maior parte desses indivíduos foi afetada por algum tipo de acidente ou doença (OMS, 2011). Já a Pesquisa Nacional de Saúde (PNS), em 2021, coordenada pelo IBGE, contabilizou 17 milhões de pessoas com deficiência no país (PNS, 2021).

O impacto de um dano na coordenação motora pode ser devastador, afetando a capacidade de realizar atividades básicas do dia a dia e a qualidade de vida em geral. A perda dessa habilidade pode levar a dificuldades na realização de tarefas simples, como escovar os dentes ou vestir-se, e pode até mesmo comprometer a capacidade de trabalho. Além disso, a perda de coordenação motora também pode levar a problemas psicológicos, como depressão e baixa autoestima. Para pessoas que sofreram perdas graves, como a amputação de membros, o impacto pode ser ainda mais significativo, especialmente em regiões com poucos recursos e infraestrutura para atender às necessidades dessa parcela da população. Além disso, danos ao cérebro ou à coluna vertebral podem ter consequências irreversíveis, como perda de memória, habilidades motoras e personalidade.

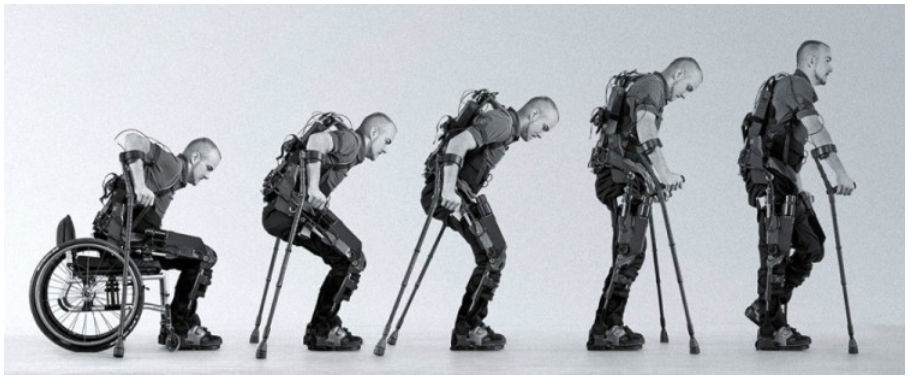

Figura 1.1 – *Exoesqueleto Rewalk*

A robótica tem sido vista como uma solução cada vez mais eficaz para tratar esses problemas. O uso de dispositivos robóticos e sistemas de realidade virtual pode ajudar a melhorar a coordenação motora e a fornecer uma terapia mais personalizada e eficaz. Uma das principais aplicações da robótica é a utilização de exoesqueletos, que são dispositivos robóticos que cobrem o corpo e permitem o movimento das pernas ou braços. Um exemplo disso é o exoesqueleto ReWalk do artigo (RIZZO et al., 2004), que foi desenvolvido para auxiliar pessoas com lesão medular na caminhada. Outra aplicação da robótica é a utilização de dispositivos de realidade virtual (VR) e sensores inercias de movimento (IMU). A VR permite a criação de um ambiente virtual imersivo, enquanto os IMUs permitem o rastreamento dos movimentos do corpo. Essa combinação permite ao paciente realizar exercícios de reabilitação de forma lúdica e interativa, como em (GÓMEZ-SÁNCHEZ et al., 2017), com o equipamento G-EO System. Além disso, a VR também pode ser utilizada para criar ambientes virtuais que permitem ao paciente praticar atividades do dia a dia, como o Virtual Reality Therapy (GÓMEZ-SÁNCHEZ et al., 2018), que permite ao paciente realizar atividades como cozinhar ou dirigir em um ambiente virtual seguro.

Figura 1.2 – *G-EO System*

Na localidade de Universidade de Brasília (UnB), grupos como o do "Projeto Empoderando Mobilidade Autonomia"(EMA) utilizam destas tecnologias para tornar possível tratamentos cada vez mais eficientes e sofisticados. Como exemplo dos trabalhos desenvolvidos, tem-se a utilização de próteses e outros sensores para medição da orientação e do sinal eletromiográfico (EMG) demonstrado de Barbosa (BARBOSA, 2021), além da utilização de robôs adicionados de técnicas como a teleoperação, demonstrado no trabalho de (BALBINO, 2016).

Figura 1.3 – Alguns integrantes do Projeto EMA (2020)

A reabilitação adicionada a realidade virtual com atividade gamificada demonstra o potencial de melhoria da saúde, força, flexibilidade e equilíbrio do corpo, citados em (MACEDO et al., 2012), e do desenvolvimento do paciente em práticas como transferência de peso corporal, mudanças de postura e independência física para atividades do cotidiano de acordo com os trabalhos de (BELLE; MACHADO; BOTARELI, 2021) e (ESPOSTO et al., 2017). Assim, com foco na reabilitação e nas tecnologias para análise do movimento do paciente, desenvolveu-se um plano de projeto para imersão e representação de um corpo virtual com movimentação similar a do usuário em uma simulação de caminhada em marcha lenta, além da estimação real da velocidade do paciente pelo dados das IMUs.

1.2 Definição do problema

Frente ao problema de dano físico corporal, ou mesmo cerebral que comprometa parte da habilidade motora do indivíduo, a reabilitação tende a ajudar na melhora do paciente, porém há diversos fatores que podem fazer com que o este desgoste e/ou abandone o tratamento. Clínicas de fisioterapia podem ser às vezes muito distantes, ou muito caras para o acesso da população necessitada. Uma própria deficiência pode dificultar a locomoção do paciente no processo de reabilitação fazendo com que ele desista. E mesmo se quando realizado o tratamento, os exercícios podem ser monótonos e a evolução dos resultados a olho nu podem ser demoradas e difíceis de se perceber.

Para percepção do corpo de uma pessoa minimamente próxima da original, é necessário representar pelo menos a cabeça, os braços, e as pernas. O equipamento de realidade virtual possibilita a simulação das mãos e da cabeça de uma pessoa, porém carece da representação dos braços e das pernas. Além disso, a estimação da velocidade de caminhada é um aspecto importante da reabilitação, mas pode ser difícil de ser determinada apenas

com base nos movimentos dos sensores e por não ser o modo de movimentação padrão no desenvolvimento de jogos.

1.3 Objetivos do projeto

O objetivo principal deste projeto é desenvolver um sistema de comunicação entre dispositivos eletrônicos como uma ferramenta de reabilitação baseada em tecnologia de realidade virtual (VR) e unidade de medida inercial (IMU) para determinar a velocidade de caminhada de indivíduos que tiveram algum comprometimento da movimentação. Outro objetivo é demonstrar o potencial da VR e da IMU como ferramentas de reabilitação, proporcionando diversas experiências para uma ampla gama de aplicações e em qualquer localidade.

Para tal propósito, objetiva-se:

1. Criar um ambiente virtual imersivo através de modelos 3D;
2. Simular a experiência de caminhada em marcha lenta com movimentos do corpo inteiro;
3. Calcular a velocidade a partir dos dados de orientação dos sensores IMU;
4. Proporcionar uma experiência mais imersiva e realista para o paciente;

1.4 Cronograma de execução

Feito inicialmente a revisão bibliográfica de trabalhos antigos feitos por Raphael Barbosa e Henrique Balbino, alunos passados do projeto EMA, por também utilizarem de sensores de posição para desenvolvimento de seus respectivos trabalhos. Obteve-se uma base teórica para o trabalho e uma perspectiva das próximas atividades a serem tomadas, realizou-se também a leitura de artigos relacionados ao tema sugeridos pelo professor e por avaliação própria, além da leitura de livros; acesso a sites e fóruns da comunidade Unity; e visualizações de vídeos do Youtube abordando o assunto.

1. Revisão bibliográfica;
2. Familiarização com equipamentos e sensores;
3. Desenvolvimento da estratégia de captura e representação de dados;
4. Implementação em software da estratégia desenvolvida ;
5. Experimentos e testes para análise;

6. Conclusão de resultados.

2 Fundamentação Teórica

Este capítulo traz uma base teórica e uma breve introdução dos assuntos e das principais tecnologias utilizadas para a realização do projeto.

2.1 Hardware

Será inicialmente apresentado os equipamentos e dispositivos utilizados, em seguida os programas e então os conceitos teóricos envolvendo estas tecnologias.

2.1.1 Sensores Inerciais de Movimento - Yost Labs

Estes sensores, também chamados de unidades de medição inercial (IMU), são dispositivos que provêem dados de orientação em tempo real, utilizando uma combinação de acelerômetros, giroscópios e magnetômetros para medir velocidade angular, aceleração linear, e direção do campo magnético no espaço. As medições nos 3 eixos ortogonais XYZ obtêm medidas com até 9 graus de liberdade e a partir deles temos a representação da orientação por ângulos de Euler, representado por 3 variáveis (x, y, z); ou por quatérnions que possuem 4 variáveis (x, y, z, w) através da configuração de parâmetros de comunicação.

Figura 2.4 – Linha de sensores inercias da Yost Labs, dongle a direita

Usados em uma variedade de indústrias e aplicações, incluindo realidade virtual e aumentada, robótica, esportes e automação industrial, estes sensores desta linha Yost Labs são considerados de confiança por diversas empresas de tecnologia e instituições de pesquisa ao redor do mundo.

2.1.1.1 Giroscópio, Acelerômetro, Magnetômetro

O giroscópio é um sensor que mede a variação angular de um corpo pelo tempo, mais precisamente, a sua velocidade angular por segundos, porém não é capaz de determinar sua orientação absoluta em um espaço. É possível fazer uma estimativa da orientação integrando

o valor ao longo do tempo, porém o erro será acumulado e para esta correção, utiliza-se o acelerômetro e o magnetômetro.

O acelerômetro é capaz de medir as acelerações lineares em atuação sobre um corpo. Há dois tipos: as estáticas, como a gravitacional; e as dinâmicas, como deslocamentos e perturbações. Devido à gravidade sempre se manter a uma direção constante, para baixo, esta pode ser usada como referência absoluta para estimação da orientação espacial. As acelerações dinâmicas causam um ruído que atrapalham esta estimativa, além disso, não é possível medir a variação em um dos 3 eixos, o ângulo de guinada, por ser paralela ao vetor de influência da gravidade. Por isso usa-se o magnetômetro para o cálculo deste terceiro eixo.

O magnetômetro é capaz de medir a direção e magnitude do campo magnético ao qual o corpo está submetido. Similar ao acelerômetro, este detecta o campo magnético da Terra, e age como uma referência absoluta para estimação de orientação espacial. Porém suas medidas podem ser afetadas por materiais ferrosos ou magnéticos, dependendo mais uma vez da composição dos outros sensores para manter uma boa estimativa da orientação.

Figura 2.5 – Variações angulares em torno de cada eixo. Imagem tirada de (HANGAR33, s.d.)

2.1.2 Equipamento de Realidade Virtual - HTC

HTC Vive é um headset para realidade virtual (RV) desenvolvido pela "HTC and Valve Corporation". Lançado em 2016, foi um dos primeiros produtos de RV para o consumidor de ponta no mercado. Este headset foi construído para ser utilizado com o PC e requer uma placa de vídeo de alta qualidade para rodar com pouco atraso (lag) e de maneira fluida. A boa resolução, de até 2160 x 1200 pixels, e alta taxa de frame (90Hz), ajudam a reduzir o enjoo e melhorar a experiência de imersão da RV. É possível também utilizar fones de ouvido e o microfone embutido para aumentar ainda mais a sensação de presença dentro do ambiente virtual. O equipamento vem junto de dois controles sem fio e duas estações-base para rastreamento do movimento destes e do headset, chamado também de Head-Mounted Display (HMD), em um ambiente delimitado, permitindo que o usuário ande ao redor e interaja com objetos e ambientes virtuais.

Figura 2.6 – Equipamento de VR - HTC Vive

As estações-base são aparelhos estacionários colocados na sala onde a atividade está sendo realizada, estas utilizam de lasers infravermelho que constantemente emitem sinais para os sensores de infravermelho posicionados no headset e nos controles. Utilizando-se a medição do tempo para o laser ser emitido e em seguida, detectado, as estações-base são capazes de determinar com grande acurácia a posição e orientação do headset e dos controles em tempo real.

2.1.3 Esteira - MT200

A esteira da Spirit Medical Systems Group, de modelo MT200 é um aparelho de exercício que possibilita a realização de atividades físicas como caminhada ou corrida em um ambiente controlado. Elas são compostas por uma superfície de corrida plana e lisa, que pode ser ajustada em relação à velocidade. Possui características adicionais, como programas de treinamento pré-definidos, barreiras horizontais para segurança, e monitores de frequência cardíaca, que proporcionam uma maior variedade de opções de exercícios e acompanhamento da condição física. A utilização regular de uma esteira pode ajudar a melhorar o condicionamento cardiovascular e contribuir para a perda de peso.

Figura 2.7 – Esteira Spirit Medical MT200

2.2 Softwares

2.2.1 3-Space Suite Application

É a aplicação fornecida pela empresa Yost Labs para configuração e análise gráfica dos dados sobre a orientação do sensor e parâmetros de calibração. Além disso, é possível analisar gráficos ao longo do tempo dos valores do acelerômetro, do giroscópio e do compasso, para representação da variação angular em ângulos de Euler, ou quatérnions se desejado. Através da aplicação, obtêm-se o número serial correspondente de cada IMU que será associada a um ID nas configurações do Dongle para reconhecimento e, em seguida, manipulação em código python.

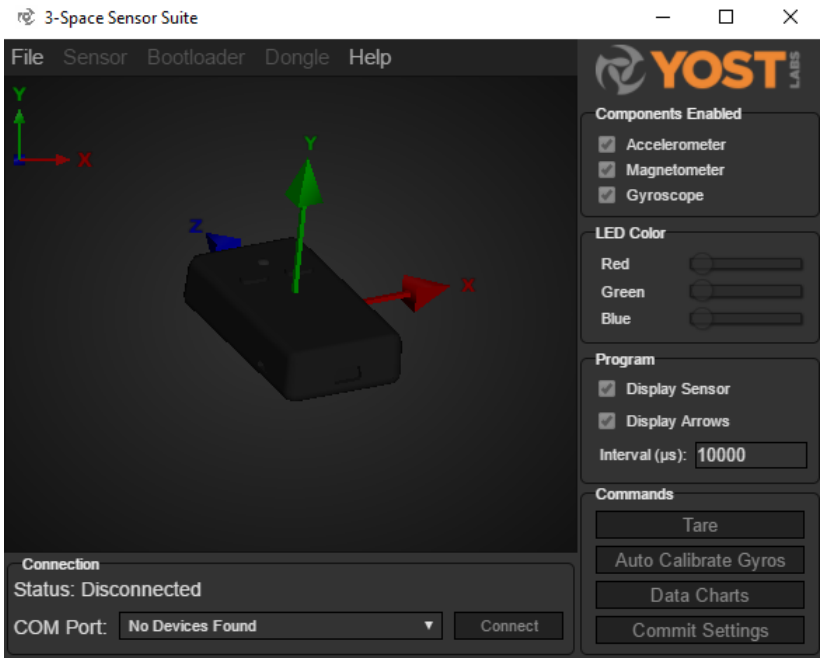

Figura 2.8 – *3-Space Suite Application*

2.2.2 Ambiente de desenvolvimento integrado - Visual Studio e Pycharm

Um ambiente de desenvolvimento integrado (IDE) é um programa de computador que fornece ferramentas para facilitar o processo de desenvolvimento de software. Ele geralmente inclui um editor de código, um depurador e um gerenciador de versão, além de outras ferramentas que podem ser úteis para o desenvolvedor. Um IDE é projetado para aumentar a produtividade do desenvolvedor, pois ele permite que todas as ferramentas necessárias para o desenvolvimento de software sejam acessadas em um único lugar.

2.2.3 3D Modelling Software - Blender

O Blender é um programa gratuito e de código aberto usado na criação de modelos 3D, animação, simulação, renderização, rigging, edição de vídeo e criação de jogos. É uma

ferramenta poderosa capaz de criar gráficos tão bons quanto os usados pelos profissionais, sendo ideal para indivíduos e pequenos estúdios. Há um enorme suporte da comunidade e está sempre sendo atualizada. Alguns modelos e testes de animações foram brevemente feitos com o uso deste programa para utilização no ambiente virtual, tal como a pista de corrida e outros objetos de teste.

2.2.4 Game Engine - Unity

É uma plataforma de desenvolvimento de jogos, também chamados de Engine, ela é recomendada para desenvolvedores independentes devido às suas múltiplas funcionalidades como edição e manipulação de terreno, animação, materiais, iluminação, simulação de física e suporte para realidade virtual de maneira mais simples se comparada a Unreal Engine, porém grandes empresas utilizam desta plataforma,. Não é incomum o uso destes softwares para simulações e testes em institutos de pesquisa.

Figura 2.9 – Logos da Unreal Engine e da Unity

Para fim de comparação, o projeto inicial utilizava-se de outra engine chamada Unreal Engine. Em comparação com a Unity, eles se diferem em alguns aspectos, principalmente em relação aos gráficos, no qual a Unreal consegue atingir um grau de realidade maior, porém ao custo do aumento de processamento, e portanto, aumento do delay e diminuição do FPS (frame per second). A Unity possui uma interface mais simples e leve sendo recomendada para iniciantes com menos rigor gráfico ou para pequenas equipes com menos experiência, mas que ainda são capazes de criar jogos de boa qualidade e alto realismo no mercado profissional.

2.2.5 SteamVR e OpenXR

A Unity utiliza um plugin para facilitar o desenvolvimento de aplicações que utilizem a RV chamado de OpenXR. Esta API permite que os desenvolvedores criem aplicações de RV e RA que poderão ser acessadas por diversos dispositivos diferentes, sem se preocuparem muito sobre as especificidades de cada hardware. Junto a este plugin, utiliza-se o software SteamVR, uma plataforma já consolidada e desenvolvida pela Valve Corporation, para suporte e acesso do hardware aos diversos conteúdos da RV.

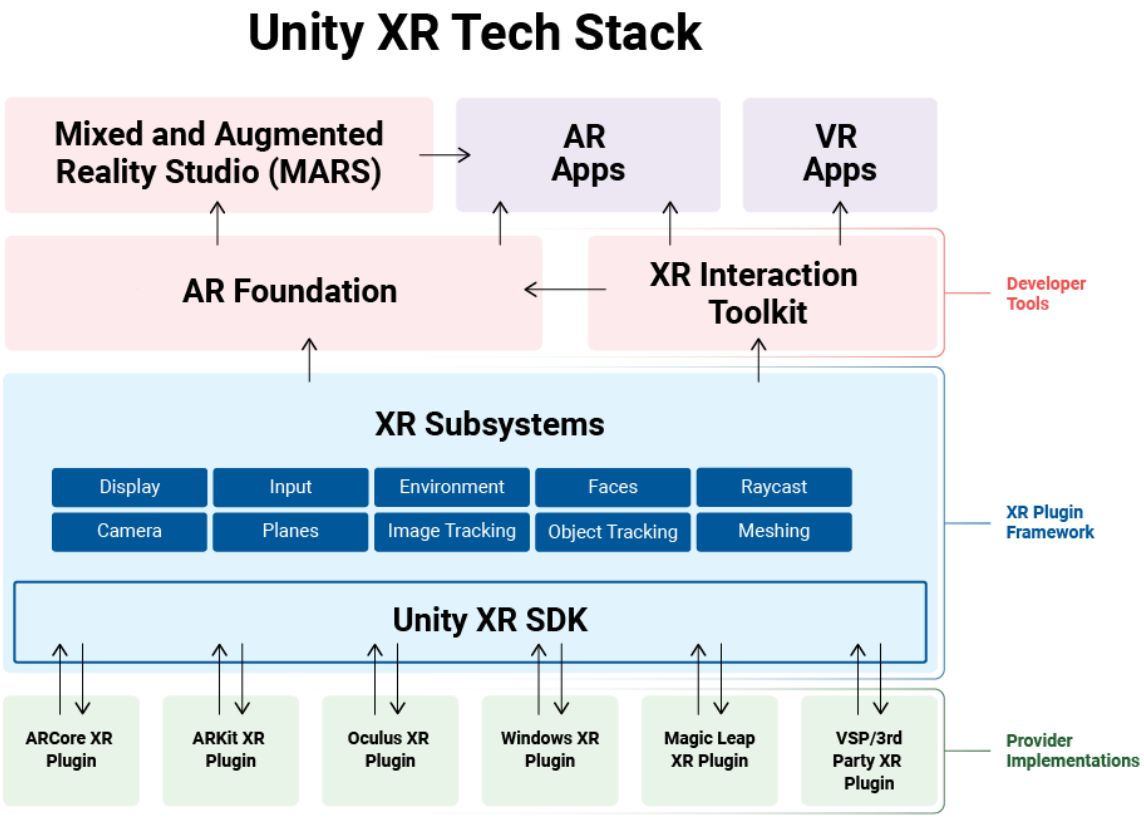

Figura 2.10 – *Diagrama para desenvolvimento rápido de aplicações de realidade virtual*

A imagem mostrada na Figura 2.10 exemplifica em forma de diagrama como funciona a comunicação de um dispositivo de realidade virtual com aplicações do tipo : MARS , AR, e VR.

2.3 Conceitos teóricos

O hardware e software utiliza de conceitos de engenharia para o funcionamento destes, esta seção explica o estudo sobre o assuntos.

2.3.1 Tipos de comunicação

Cada tipo de comunicação tem suas próprias características e considerações de projeto, e o tipo mais adequado para uma determinada aplicação dependerá de fatores como a distância, a velocidade, a confiabilidade, a segurança e o custo.

2.3.1.1 Comunicação sem fio: Bluetooth

O Bluetooth é um padrão de comunicação sem fio que permite a troca de dados entre dispositivos em uma curta distância. Ele é amplamente utilizado em dispositivos eletrônicos

como smartphones, tablets, laptops, fones de ouvido, e muito mais. A comunicação Bluetooth é possível graças à utilização de rádio frequência (RF) em uma faixa de frequência de 2,4 GHz e à utilização da técnica de modulação FHSS (Frequency-Hopping Spread Spectrum).

Figura 2.11 – Comunicação sem fio : Bluetooth

Para se comunicar através do Bluetooth, os dispositivos precisam ser pareados e autenticados mutuamente. Depois disso, eles podem se comunicar de forma privada e segura através de uma conexão Bluetooth. É uma opção popular para a comunicação sem fio devido à sua simplicidade, baixo consumo de energia e compatibilidade com uma ampla variedade de dispositivos. No entanto, a distância máxima de comunicação é limitada em relação a outros padrões de comunicação sem fio, como WiFi.

2.3.1.2 Comunicação serial: USB

A comunicação serial USB (Universal Serial Bus) é um tipo de comunicação serial que utiliza o padrão USB para conectar dispositivos a um computador ou outro dispositivo host. Ela é amplamente utilizada em aplicações que precisam transferir dados a curtas distâncias, como a conexão de periféricos a um computador ou a comunicação entre dispositivos internos em um sistema embarcado.

Figura 2.12 – Diversos tipos de entradas USB

O USB é um padrão de comunicação serial que foi desenvolvido para simplificar a conexão de dispositivos eletrônicos, substituindo os muitos tipos de conectores e cabos diferentes que existiam antes dele. Ele é compatível com uma ampla variedade de dispositivos

e oferece alta velocidade de transmissão de dados, baixo consumo de energia e facilidade de uso. Para se comunicar através da comunicação serial USB, os dispositivos precisam estar conectados por um cabo USB ou por uma porta USB em ambos os lados.

2.3.1.3 Comunicação de redes: TCP e UDP

A comunicação pela rede se dá por requisições e respostas. Um dos modelos mais conhecidos para descrição destes protocolos são o OSI, que possui 7 camadas, e o modelo TCP/IP, que possui 5 camadas. Como mostra a Figura 2.13

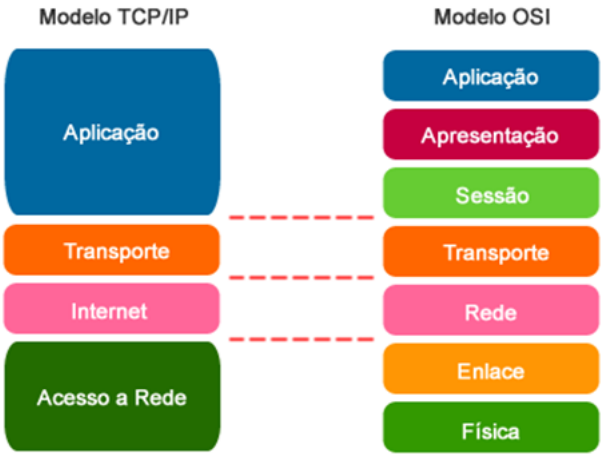

Figura 2.13 – Modelos OSI e TCP/IP. Imagem tirada de (RAIN, s.d.)

O TCP é um protocolo orientado a conexão, o que significa que os dispositivos precisam estabelecer uma conexão explícita antes de começar a trocar dados. Isso garante que os dados sejam transmitidos de forma fiel e completa, mas pode ser mais lento do que outros protocolos em algumas situações.

O UDP (User Datagram Protocol) é um protocolo de camada de transporte semelhante ao TCP, mas é orientado a mensagem e não requer uma conexão explícita. Isso torna o UDP mais rápido, mas também significa que os dados podem ser perdidos ou entregues de forma incorreta em algumas situações. O UDP é frequentemente usado em aplicações que precisam transmitir dados em tempo real, como jogos online ou telefonia IP.

2.3.2 Representação da orientação 3D

A orientação tridimensional de um objeto pode ser representada de várias maneiras, cada uma com suas próprias vantagens e desvantagens. Algumas das maneiras mais comuns de representação incluem:

2.3.2.1 Ângulos de Euler

Os ângulos de Euler são uma forma comum de representar a orientação tridimensional de um objeto, especialmente em sistemas de coordenadas cartesianas. Eles são fáceis de entender e interpretar, pois são compostos por três ângulos que descrevem a rotação em torno dos eixos x, y e z como mostrado na Figura 2.14. Este tipo de representação é muito estudado e amplamente utilizado em aplicações de dinâmica de corpos rígidos, pois permite demonstrar a relação entre dois sistemas de coordenadas, um local e outro geralmente fixo.

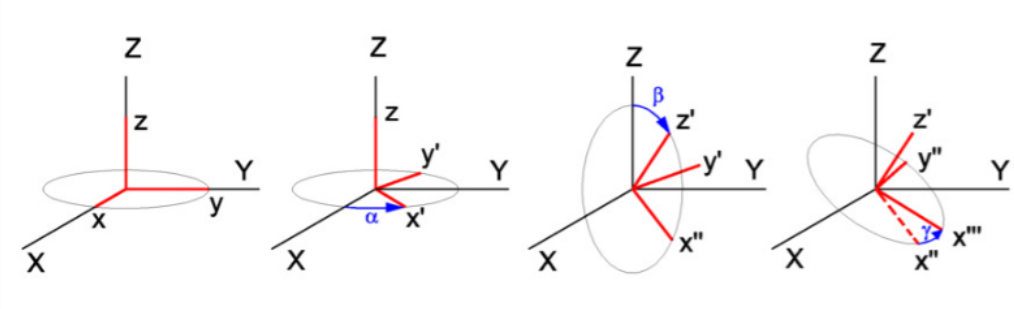

Figura 2.14 – Representação em ângulos de Euler. Imagem tirada de (PNG, s.d.)

No entanto, é importante lembrar que a representação dos ângulos de Euler pode ter problemas de ambiguidade em algumas situações, onde um objeto pode ser rotacionado em 360 graus de volta ao seu estado original sem mudar sua orientação. Além disso, os ângulos de Euler também podem sofrer com o problema conhecido como "Gimbal lock", onde a rotação em torno de um eixo fica limitada devido às restrições dos outros eixos.

2.3.2.2 Quatérnions

Outra forma de representação da orientação são os quatérnions. O quatérnion é formado por quatro dimensões e pode ser entendido como uma parte real, formada por um valor e outra imaginária, formada por três valores.

$$q = q_0 + iq_1 + jq_2 + kq_3 \tag{2.1}$$

Por possuir quatro dimensões, sua representação 3D e interpretação de seus valores são considerados de difícil compreensão. A Figura 2.15 é mostrada. Mesmo assim, estes são geralmente utilizados para fins de robótica, criação de jogos e gráficos visuais devido a sua maior velocidade de cálculo em computadores para determinação de uma nova rotação, se comparado ao cálculo por matrizes de rotação utilizados pela representação em ângulos Eulers. Além disso, os quatérnions permitem a interpolação entre transformações de rotação aplicadas a um corpo, de forma suave e contínua, também evitando assim, o problema de Gimball Lock.

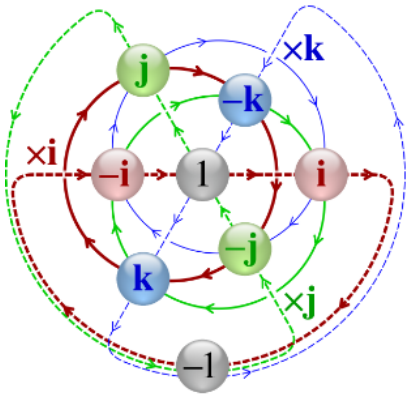

Figura 2.15 – Representação em quatérnions. Imagem tirada de (WIKIPEDIA, s.d.)

É possível converter entre quatérnions e ângulos de Euler através da multiplicação matricial. A conversão de quatérnions para ângulos de Euler é dada por uma matriz 4x3, enquanto que a conversão no sentido oposto é dada por um vetor 3x1.

Para converter de quatérnions para ângulos de Euler, basta multiplicar os quatérnions pela matriz de conversão. Por exemplo, se q = (h, i, j, k) é o quatérnion a ser convertido e M é a matriz de conversão, os ângulos de Euler são dados por:

$$q * M = (x, y, z) \tag{2.2}$$

Já para converter de ângulos de Euler para quatérnions, basta multiplicar o vetor de ângulos de Euler pelo inverso da matriz de conversão. Por exemplo, se (x, y, z) são os ângulos de Euler a serem convertidos e M é a matriz de conversão, o quatérnion resultante é dado por:

$$(x, y, z) * M^{-}1 = q \tag{2.3}$$

Além disso, as matrizes de conversão específicas podem variar dependendo do sistema de coordenadas e da convenção de Euler utilizados.

2.3.3 Cinemática

Cinemática é estudo do movimento e de trajetórias em que se desconsidera as causas, como forças e torques. No contexto particular em que o objeto sendo estudado está conectado a uma sequência de elos e juntas, como um braço robótico, Figura 2.16, é que se definem os conceitos de cinemática direta e inversa.

2.3.3.1 Cinemática direta

A cinemática direta é o ramo da cinemática que estuda o movimento de um corpo a partir das forças e torque aplicados. Ela é importante em diversas áreas, como a mecânica dos

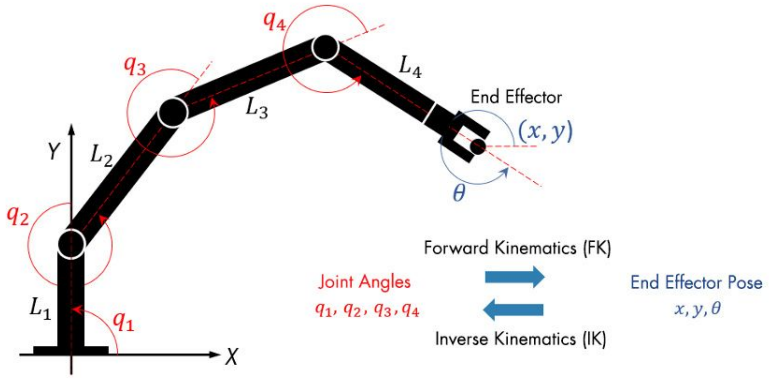

Figura 2.16 – Sequência de elos e juntas em braço robótico. Imagem tirada de (WORKS, s.d.)

sólidos, a dinâmica dos gases, a mecânica dos fluidos e a robótica, entre outras. A cinemática direta permite calcular a posição, velocidade e aceleração de um corpo a partir das forças e torque que atuam nele. Isso possibilita, por exemplo, o projeto de sistemas de controle de movimento ou a simulação de sistemas mecânicos. Para estudar o movimento de um corpo em cinemática direta, é preciso conhecer as leis de Newton, que estabelecem as relações entre as forças que atuam em um corpo, e a equação de Euler, que relaciona as forças de torque que atuam em um corpo rígido com sua aceleração angular.

2.3.3.2 Cinemática inversa

Já a cinemática inversa é o ramo da cinemática que estuda o movimento de um corpo a partir da posição desejada, permitindo calcular as forças e torque necessários para mover um corpo.

Geralmente, a cinemática inversa é mais complexa que a cinemática direta, pois envolve o cálculo de variáveis que são função de outras, o que pode exigir o uso de métodos iterativos ou otimização. Além disso, a cinemática inversa pode enfrentar problemas como a colisão, que é o contato entre dois ou mais corpos.

2.4 Tecnologias Gráficas

2.4.1 Realidade Virtual

A realidade virtual (VR) é uma das mais grandes e importantes tendências, e que pode ser implementada nos mais diferentes setores como o empresarial, industrial e em pesquisas. Antigamente, por volta de duas décadas atrás, os equipamentos utilizados para a VR apresentavam alto custo, dificuldades de implementação e exigiam uma curva de aprendizado grande para a operação deles. Porém, a evolução da ciência da computação propiciou uma evolução que se tornou mais viável no mercado de produtos eletrônicos, como cita (RODRIGUES; PORTO, 2013).

Figura 2.17 – Equipamento de realidade virtual da Playstation

Aplicado principalmente no setor de games, porém não limitado apenas ao lazer, a realidade virtual é capaz de criar ambientes virtuais por meio de um sistema computacional. É uma tecnologia recente que utiliza efeitos visuais e sonoros, além de permitir o usuário incorporar uma entidade e dentro do ambiente virtual, interagir com este por meio de aparatos tecnológicos em primeira pessoa com uma percepção bastante semelhante de espaço da realidade.

2.4.2 Modelagem 3D

Segundo (BASTOS, 2015) , a modelagem consiste em todo o processo de criação de um modelo *mesh*, seja um objeto ou uma cena, em sua forma tridimensional e utilizando softwares especializados. Modelos 3D podem ser usados para uma variedade de propósitos, incluindo visualizar designs e produtos, criar animações e efeitos especiais para cinema e televisão, e criar experiências de realidade virtual. Quando em formato 2D, como nos desenhos ou jogos antigos, os modelos são chamados de sprites.

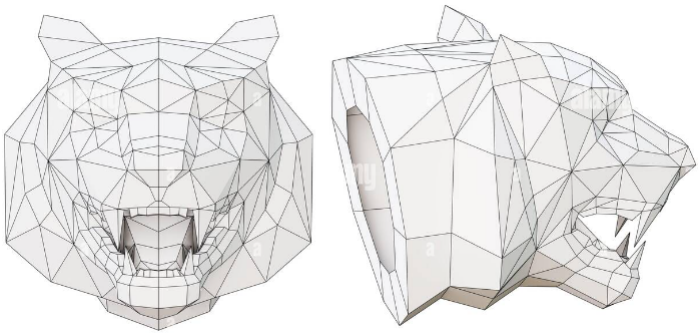

Figura 2.18 – Modelo 3D em modo de visão "Wireframe"

O modelo 3D é formado por vértices, arestas e lados. A unidade mínima que define a complexidade do modelo é o número de triângulos necessários para formação deste. Quanto mais detalhado, maior a quantidade de triângulos necessários para representá-lo e maior poder de processamento necessário para manipulá-lo. Para se ter uma ideia, meshes

com baixo número de polígonos (low poly), podem ter por volta de centenas de triângulos, enquanto outros mais complexos podem passar de milhões.

2.4.3 Texturização

Na texturização, define-se a cor e a textura que o modelo deverá representar. O processo é semelhante a envolver uma imagem 2D em volta de um objeto 3D. Ao posicionar e dimensionar corretamente a imagem em relação ao modelo e definir as qualidades de reflexão da luz sobre o objeto, é possível criar a ilusão de diferentes materiais, como vidro, tijolo ou metal. Como mostrado na Figura 2.19.

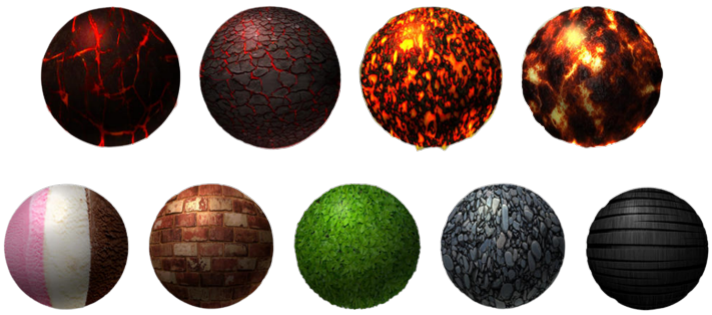

Figura 2.19 – Diferentes exemplos de texturas aplicadas ao mesmo modelo 3D. Imagem retirada de (PRINTING, s.d.)

Para reduzir ou manter a complexidade do modelo, a texturização é capaz de adicionar detalhes sem aumento do número de polígonos, criando a impressão de textura numa superfície plano.

2.4.4 Rigging

Este processo usa uma série de ossos (*bones*) digitais interconectados representando o esqueleto dos personagens, Figura 2.20, para simular a movimentação e conexão entre os modelos. Um modelo 3D pode ser criado pela junção de diversas partes e definindo suas relações de dependências. Ou seja, caso alguma parte do modelo se mova, outra parte conectada deve, ou não, se mover a partir de um ponto de referência.

Rigging é uma etapa importante no processo de animação 3D, pois permite que os animadores dêem vida aos seus modelos e criem movimentos complexos.

2.5 Terminologia de desenvolvimento de jogos

Conceitos utilizados pela plataforma da Unity para interação e modificação de seus componentes.

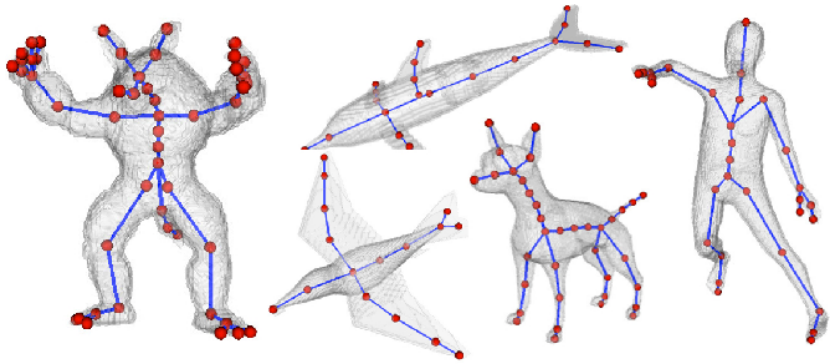

Figura 2.20 – Modelos com seus diferentes esqueletos e partes conectadas

2.5.1 GameObject, Components, Prefabs e Scripts

Os *GameObjects* são a unidade fundamental para representação de personagens, adereços e cenário. Estes, funcionam como contêineres para adição de *Components* que darão a funcionalidade para o objeto. O *Script* pode ser entendido como um código e tipo de componente no qual se pode criar funções personalizadas para o comportamento do *GameObject*. E depois de criado um objeto de jogo, com seus componentes e códigos, caso se queira reutilizá-lo de forma rápida, sem ter que refazer um novo do zero, é possível transformá-lo em um *Prefab* para posterior uso.

Figura 2.21 – Layout principal da Unity e seus componentes

3 Métodos e Desenvolvimento

Os passos para a estimação da velocidade do usuário e da representação, criação e movimentação de um avatar completo em ambiente virtual com auxílio dos sensores e do equipamento de realidade virtual são desenvolvidos e explicados neste capítulo.

3.1 Metodologia

Utilizando o software de desenvolvimento de jogos Unity para demonstração do avatar, deseja-se estimar a velocidade de caminhada em marcha lenta a partir dos dados de orientação dos sensores. Além disso, deseja-se inserir a representação e o movimento da cabeça e todos os seus membros em primeira pessoa para um experiência mais imersiva com um número limitado de sensores. A metodologia desenvolvida é composta pelas seguintes etapas:

- Estudar e calibrar o sensor de posição disponível, além de extrair, tratar e enviar seus dados.
- Aprender a usar a ferramenta Unity, configurar e instalar os pacotes necessários para comunicação dos equipamento de realidade virtual com o programa.
- Criar os códigos para captura, processamento, e plotagem da velocidade, assim como definir a lógica e funcionamento do sistema.
- Representar o avatar completo, junto aos membros superiores, inferiores e a cabeça.
- Simular o sistema.
- Analisar e validar os resultados.

3.2 Sistemas Hardware e Software

Separado em 3 conjuntos principais de hardware, tem-se os sensores da Yost Labs, os equipamentos de realidade virtual da HTC Vive, e o computador com sistema operacional Windows rodando todos os softwares necessários para o desenvolvimento do projeto. O sistema previsto é demonstrado na Figura 3.22, além destes, será utilizado também uma esteira para testes e determinação dos períodos das senoides para uma velocidade constante.

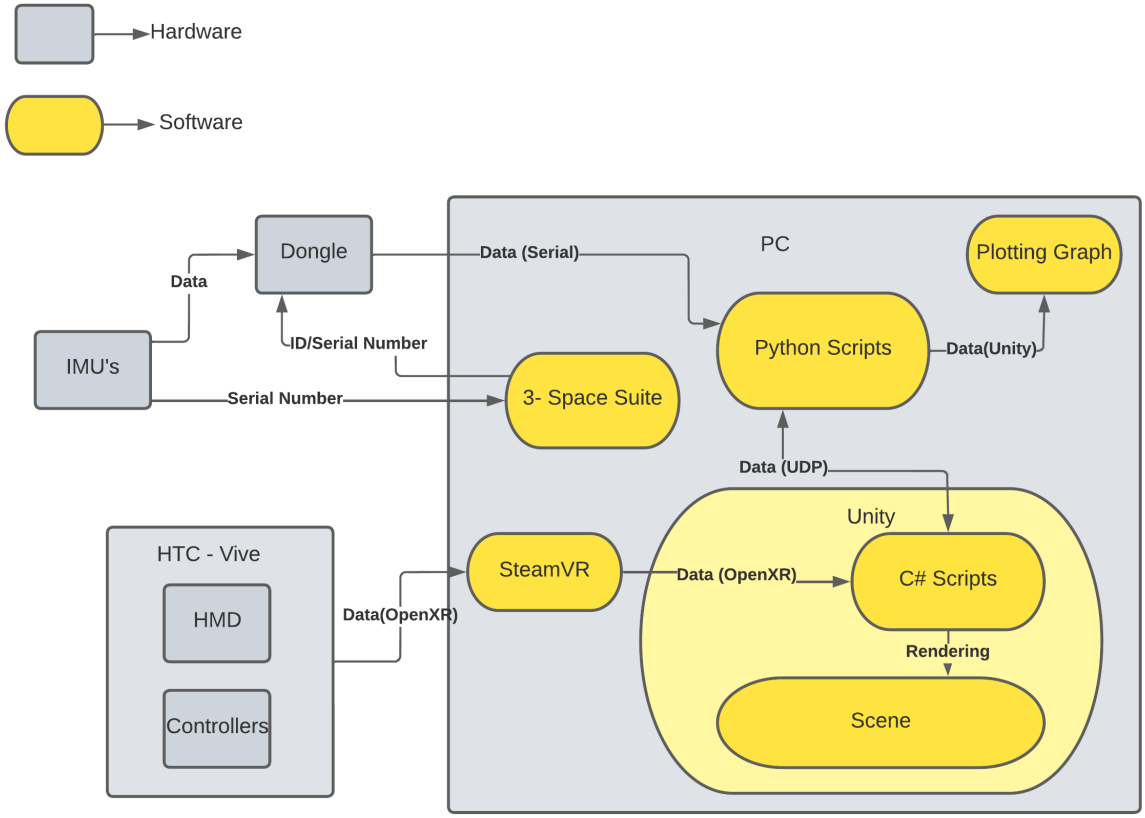

Figura 3.22 – Diagrama do sistema de hardware e software

3.3 Calibração dos sensores

Para análise inicial do sensor, realizou-se testes com rotações em volta de cada um dos 3 eixos por vez no sentido horário usando o *Data Chart* do aplicativo, anexos A.48, A.49, A.50. Na Figura 3.23, pode-se perceber inconsistências em sua orientação. Os valores do acelerômetro e do compasso são representados por matrizes $3x3$, e um vetor de *Bias* para cálculo de sua orientação. Este problema pode ser resolvido calibrando o acelerômetro e o compasso.

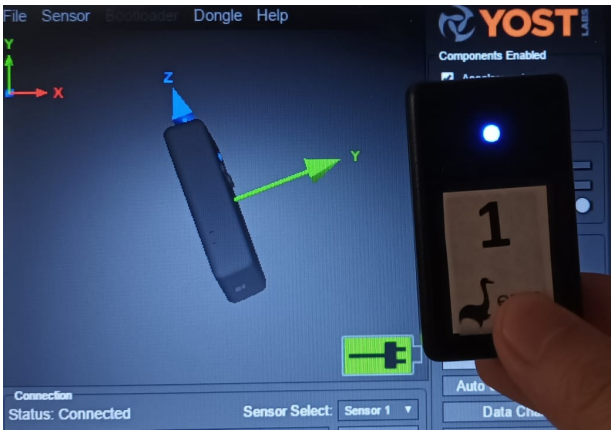

Figura 3.23 – Desalinhamento da posição real e representação no software da IMU

Na aba *Sensor* » *Calibration* » *Run Gradiente Descent Wizard*. Seguindo o tutorial, a IMU deve ser posicionada como na demonstra no aplicativo e em seguida, apertar *Next*. Repete-se para todas as 24 orientações possíveis e por fim, salvar os valores para cada sensor.

No início, pensou-se que a representação da IMU girando fora do alinhamento do eixo fosse um erro esperado devido a imperfeições naturais dos sensores. Porém com o tempo, os erros se tornaram cada vez maiores ao ponto de haverem rotações de até 90 graus fora da orientação esperada. Devido a isso, percebeu-se a falta de calibração do equipamento e após a configuração, os novos gráficos das rotações com os mesmos testes são mostrados nos anexos A.51, A.52, A.53.

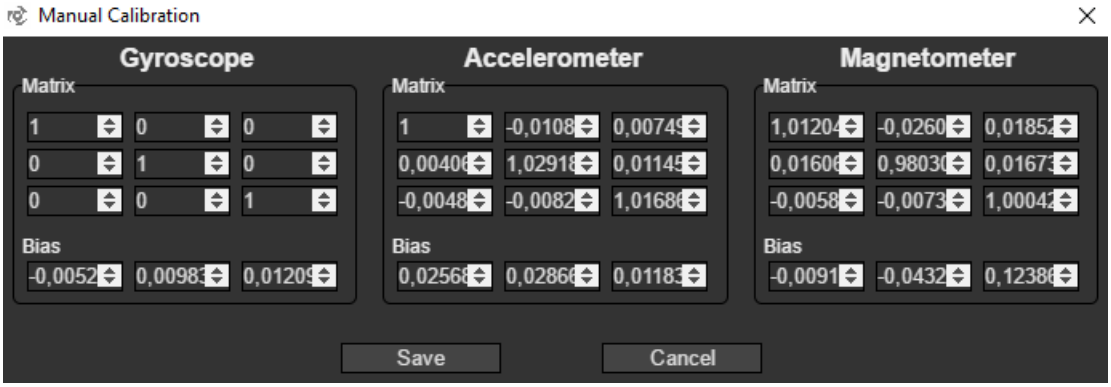

Figura 3.24 – Valores de Bias, do acelerômetro e do compasso calibrados

Depois de calibrado, pode-se ver os novos valores na calibração manual pela figura 3.24. Pelos gráficos, também pode-se perceber o comportamento de suas variáveis. Para representação dos valores dos ângulos de Euler, os valores possuem uma descontinuação ao se atingir um valor máximo e então trocados de sinal, com aumento positivo contínuo. Já a dos quatérnions pode-se ver um movimento contínuo em todas as suas orientações devido a sua forma de representação.

Os gráficos antes e depois de calibrados podem ser vistos em Anexo A.

3.4 Comunicação entre dispositivos

Os inputs principais do projeto são os sensores e o equipamento de RV. Para que o projeto se desenvolve-se, foi necessário entender como eles se comunicavam para obtenção de seus dados.

3.4.1 IMU's e Dongle

O Dongle é um dispositivo de hardware que se conecta a um computador, para fornecer o acesso aos dados das IMU's. Normalmente, um dongle é conectado a uma porta USB.

Conectar IMUs com o Dongle é um processo que permite que os dados coletados pelos sensores IMU sejam transmitidos para um computador para processamento e análise. As IMUs se conectam com o Dongle via Bluetooth, sendo assim uma forma conveniente e sem fio para a aplicação proposta, já que ela requer um certo grau de movimentação. O Dongle se conecta ao computador via USB e permite que os dados sejam transmitidos de forma segura e confiável.

Para conectar um IMU ao dongle, é necessário seguir alguns passos. Primeiro, é preciso instalar o programa *3-Space Suite* da Yost Labs e o software de driver do dongle no computador. Em seguida, para estabelecer comunicação wireless entre eles, é preciso inicialmente que eles se reconheçam pelo número serial.

3.4.1.1 Identificação Serial das IMU's

Conectando inicialmente a IMU com o notebook, o programa reconhece automaticamente ao selecionar a porta COM do sensor. Na aba com informações sobre o sensor, encontra-se o número serial dele para se utilizar na comunicação wireless do sensor com o Dongle. Repete-se o processo para outras IMU's.

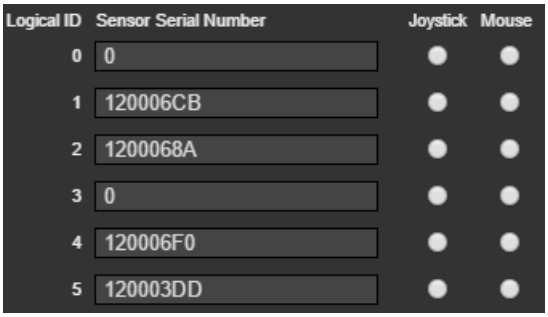

Figura 3.25 – *Dongle Wireless Setup*

Em seguida, desconectando-se os sensores e com apenas o Dongle conectado, é necessário abrir a aba de informações deste e escrever o número serial de cada sensor associado a um ID como na Figura 3.25. Assim é possível receber os valores de múltiplos IMU's utilizando uma porta COM conectado ao computador.

3.4.2 Equipamento VR e computador

Para conexão do equipamento de realidade virtual para testes na plataforma, é necessário a instalação de alguns softwares e seguir alguns procedimentos mostrados a seguir.

3.4.2.1 OpenXR Plugin

Nas configurações de projeto, instalar o Gerenciador de Plug-in XR, assim como aceitar o novo sistema de input da Unity. Alterar tipo de renderização para *Multi Pass* devido a utilização de duas telas, uma para cada olho, e escolher os perfis de interação, ou seja, a

marca e o tipo de controle que serão utilizados. Neste caso, utilizou-se o Valve Index da HTC Vive e adicionou-se também para controles da marca Oculus por ser o headset mais popular. A Figura 3.26 demonstra a tela de configuração para o procedimento.

Figura 3.26 – Configurações do Sistema do OpenXR

3.4.2.2 New User Input System

O novo sistema de entradas da Unity lançado em 2019 separa a entrada de dispositivos com as ações de código. No antigo, era necessário o jogador saber qual dispositivo e qual botão ele estava pressionando, desta nova forma, o jogador apenas precisa saber que ações o jogador aciona.

Para ativar este novo sistema, deve-se baixar pelo gerenciador de pacotes no registro da Unity o pacote *Input System*. Depois de baixado, um novo tipo de *Input Action Asset* é criado e dentro dele uma nova aba com o mapeamento das ações e suas respectivas teclas.

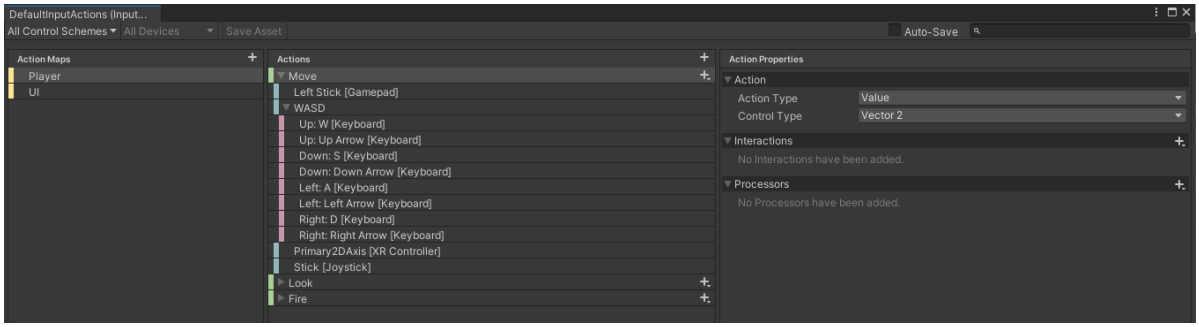

Figura 3.27 – *Input Action Map*

Agora, os inputs para mover o jogador assim como os para navegar no *UI* (User Interface) podem ser programados separadamente sem dependência do dispositivo, o que facilita no desenvolvimento para quando o equipamento não está disponível.

3.4.2.3 XR Interaction Toolkit

Para a tradução dos inputs do controle para reconhecimento por ações na Unity, utiliza-se o pacote baixado através da *URL: com.unity.xr.interaction.toolkit*. Este pacote possui diversas pré-configurações como movimento contínuo, rotação do personagem, controle da

mão esquerda e direita, entre outras funcionalidades para execução mais rápida do nosso projeto.

Por fim, nas configurações de projeto, no gerenciador de pré-configurações, fazer a diferenciação pelo campo de filtros da parte direita e esquerda de cada *script* da ação de controle de cada mão.

3.5 Desenvolvimento dos códigos

Os scripts python são responsáveis por abrir a comunicação serial do Dongle para obtenção dos dados das IMUs, além de decodificar, tratar e então enviá-los para a Unity por comunicação UDP para posterior cálculo de velocidade. Os scripts também recebem os valores calculados e realizam a plotagem de gráficos.

Já os scripts em *C sharp* são responsáveis por receber estes dados parcialmente tratados provenientes dos códigos em python, determinar a velocidade relacionada ao movimento angular do sensor, e realizar a conexão do movimento do avatar virtual com o dos sensores e equipamentos de realidade virtual. Os cálculos obtidos são reenviados para os scripts em python.

Todos os códigos utilizados para o desenvolvimento do projeto podem ser encontrados em $https://github.com/lara-unb/VR\_velocity$.

3.5.1 Criação de um ambiente de desenvolvimento

A utilização do software Anaconda é recomendada por possibilitar que outros e futuros desenvolvedores repliquem o código e saibam as versões das bibliotecas instalados para caso haja algum erro de compatibilidade. Logo, criou-se um ambiente para todo o processo de captura e envio de dados. Foram instalada as dependências e pacotes inicialmente, tais como *matplotlib, pyserial, numpy* e outros pacotes internos. As versões de cada uma estão mostrados na tabela 3.1.

Tabela 3.1 – Pacotes e versionamentos

PACOTES	time, traceback, serial, socket	numpy	pyserial	matplotlib
VERSÂO	Python v. 3.9.12	1.21.2	3.5	3.5.1

3.5.2 Configuração de parâmetros

Dentro da pasta *getIMUQuat*, o código *main_imu.py* realiza a configuração, conexão, decodificação e tratamento dos dados dos sensores do Dongle e envia em forma de string para o código da Unity. O código utiliza de outras bibliotecas essenciais como *serial_operations.py*,

desenvolvida por antigos membros e modificada para uso personalizado, que envia parâmetros para a aplicação, ou não, de diversos tipos de filtros; define a forma de representação angular dos dados (Euler ou quatérnion); e ativa a auto calibração dos sensores; como mostra a Figura 3.28. Para comunicação sincronizar, a taxa padrão do sensor deve ser de 115200 *bits* por segundo e o ID de reconhecimento da porta do Dongle deve ser 4128.

```
# Create UDP socket to use for sending (and receiving)
sock = U.UdpComms(udpIP="127.0.0.1", portTX=8000, portRX=8001, enableRX=True, suppressWarnings=True)
# Set parameters that will be configured
imu_configuration = {
    "disableCompass": True,
    "disableGyro": False,
    "disableAccelerometer": False,
    "gyroAutoCalib": True,
    "filterMode": 1,
    "tareSensor": True,
    "logical_ids": [1, 2, 3],
    "streaming_commands": [1, 255, 255, 255, 255, 255, 255, 255]
}
```

Figura 3.28 – Parâmetros de configuração de comunicação serial e UDP

A biblioteca *UdpComms.py* é responsável pela comunicação UDP entre os dados obtidos pelo programa em Python para os scripts em C sharp da Unity. Setando os valores de IP = 127.0.0.1, os números das portas de transferência e recebimento, portRX = 8000 e portTX = 8001, consegue-se enviar e receber dados através deste tipo de comunicação. O localhost 127.0.0.1 é um endereço virtual, utilizado principalmente para comunicação interna para diagnóstico e testes locais.

Os códigos *main_imu.py*, *serial_operations.py* e *UdpComms.py* estão anexados em anexo B ao final do documento.

3.5.3 Tratamento de dados

O tratamento inicial é a aplicação do filtro de Kalman nos valores dos sensores. Para tal, deve-se setar nos parâmetros do trecho de código na Figura 3.28, o valor da variável *filterMode* para 1 de acordo com o protocolo do sensor retirado do manual do usuário do sensor (3-SPACE…, 2007-2017).

Os dados de orientação são então recebidos na forma de linguagem de máquina, e devem ser traduzidos e tratados para que se possa fazer sentido e uso de seus valores para o processamento de informações. A função de *extract_euler_angles*, figura 3.29, manipula os dados brutos que o sensor envia para obter os ângulos de Euler. O primeiro passo na função é decodificar os dados usando o método decode(). Isso é tipicamente usado quando os dados estão no formato de bytes e precisam ser convertidos para o formato de string. Em seguida, a função divide os dados decodificados em uma lista, substituindo caracteres específicos e dividindo a string por espaços. Depois disso, usa-se a função filter() para remover quaisquer

elementos vazios da lista e então extrai o vetor de ângulos de Euler da lista de dados limpos. O vetor de ângulos de Euler é o primeiro elemento, começando pela terceira coluna em diante da lista e é representado como uma string de valores separados por vírgulas. Por fim, a função converte cada uma das strings em float usando o método np.array().

```python
def extract_eulers(data):
    """ Manipulate data to obtain eulers angles
        data: Raw data that sensor send
    Returns:
        eulers angles vector
    """
    decoded_data = data.decode()
    list_data = decoded_data.replace('\r\n',' ').split(' ')
    cleaned_list_data = list(filter(None, list_data))
    euler_vector = cleaned_list_data[0][3:].split(',')
    euler_vector = np.array(euler_vector, dtype=np.float64)
    return euler_vector
```

Figura 3.29 – Decodificação e tratamento dos dados brutos do sensor

Os dados em ângulos de Euler, que estão em radianos, são convertidos para graus, arredondados e enviados para Unity. Em seguida, cria-se um código em C sharp, chamado *UdpEuler.cs*, que realiza o tratamento da string recebida para determinação da velocidade.

3.5.4 Determinação da velocidade

Ao se iniciar o teste, os braços devem estar parados e perpendiculares com o chão para calibração do sensor. Os valores recebidos começam em 0 e variam positivamente ou negativamente até 180, representando a angulação em graus. Como estratégia para determinação da velocidade, percebe-se um formato de onda senoidal dos ângulos de entrada do sensor. Para a posição do sensor adotada no braço, e pela análise dos gráficos, utilizou-se dados da angulação do eixo Y por sua maior variância.

```
if (y_data > media && ascending == false)   // Calculating number of steps
{
    ascending = true;
    steps++;
    velocity = SetVelocity(timer, prevTime);
    velocityList.Add(velocity);     // Velocity list to calculate median

    if (velocityList.Count == 10)
    {
        foreach (float x in velocityList)
        {
            sum = sum + x;
        }
        velocityMedia = sum / 10;
        sum = 0;
        velocityList.RemoveAt(0);
    }
    prevTime = timer;
}

if (y_data < media && ascending == true)
{
    ascending = false;
    steps++;
}
```

Figura 3.30 – Definição dos períodos e velocidade média

3.5.4.1 Tempo por passada

Criando-se variáveis booleanas para determinação do momento de subida, ou descida de uma onda, criou-se condições para se determinar os períodos de ciclo de onda completa, que está relacionado a dois passos, direito e esquerdo, de uma pessoa caminhando, Figura 3.30. Logo, a ideia é que cada período esta associado a uma velocidade e portanto, realizou-se testes com o uso da esteira para diferentes velocidades, de 1 a 4, em intervalos de 0,5 quilômetros por hora. A média para 10 valores de período para cada velocidade é mostrada na Tabela 4.3 em resultados.

3.5.4.2 Regressão linear

A regressão linear modela a relação entre os pontos de dados ajustando uma equação linear aos dados observados. O objetivo da regressão linear é encontrar a linha de melhor ajuste que descreve a relação entre as variáveis o mais próximo possível para então estimar o valor da velocidade com base no valor da frequência.

A partir dos dados de períodos para cada velocidade, calculou-se também suas frequências e desenvolveu-se o código *vel_graph.py* para regressão linear dos dados obtidos. A saída do código e os coeficientes obtidos é mostrado na Figura 3.31 O gráfico e equação resultante são mostrados nos resultados.

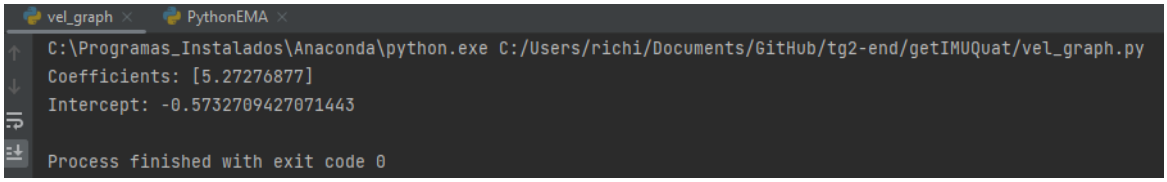

Figura 3.31 – Coeficientes da regressão linear da saída

3.5.5 Plotagem dos gráficos

Para se plotar o gráfico de velocidade estimada pelo tempo, o código em Unity utiliza de rotinas, Coroutine(), um método que permite executar um conjunto de operações em um período de tempo, em vez de tudo de uma vez como mostrado no trecho de código da Figura 3.32 Ao se apertar a tecla "Enter", a rotina envia em formato de string: dados do tempo desde que se deu Play na simulação, dados da velocidade estimada por ciclo de onda pela regressão linear, e os valores dos ângulos de rotação do eixo Y.

Afim de se evitar um número muito grande de amostras para se plotar o gráfico, se utiliza a função WaitforSeconds(), que faz com que a rotina se repita a cada 0,2s, ao invés de cada quadro. O código *main_imu.py* então recebe estes dados, salva em uma lista, e quando o número de amostras chegar numa certa quantidade programada, o código plota um gráfico e salva em um arquivo .txt os valores de velocidade estimada para cálculo posterior da

```
    // Sending data starts on Enter.
    if (Input.GetKey(KeyCode.KeypadEnter))
    {
        //timer = 0f;
        Debug.Log("Timer Started");
        StartCoroutine(SendDataCoroutine()); // Added to show sending data from Unity to Python via UDP
    }
}

1 referência
IEnumerator SendDataCoroutine() //  Added to show sending data from Unity to Python via UDP
{
    while (true)
    {
        SendData(timer.ToString("F2") + ":" + velocity.ToString("F3") + ":" + y_data.ToString("F1"));
        yield return new WaitForSeconds(0.2f);
    }
}
```

Figura 3.32 – Acionamento da rotina para plotagem de gráfico

velocidade média. O arquivo se chama *velocity.txt* e pode ser encontrado no repositório do Github.

Os gráficos de regressão linear, assim como os de velocidade estimada pelo tempo são mostrados nos resultados.

3.6 Criação do avatar virtual

Para aplicação da estratégia adotada para representação do corpo, utilizou-se diferentes técnicas para assimilar o corpo do indivíduo com o avatar virtual. O modelo Robot Kyle foi baixado gratuitamente pelo *Asset Store* da Unity, no qual se tem a acesso a diversos arquivos para o desenvolvimento de jogos. Na Figura 3.33 tem-se o modelo 3D pronto com esqueleto e ligações de cada uma das juntas utilizado.

3.6.1 Representação da parte superior

Para representação da cabeça e das mãos, utiliza-se o equipamento de realidade virtual. Para representação dos braços e dos antebraços, utiliza-se técnicas como cinemática inversa.

3.6.1.1 Cabeça

Para simulação da cabeça, o seu movimento deve causar uma mudança na visão do jogador, ou seja, os dados de rotação adquiridos pelo HMD devem orientar a câmera para representação do movimento do pescoço. O giroscópio interno informa a rotação em quatérnions de acordo com o *Input Actions*, enquanto a posição do capacete é calculada pelos sensores base instalados no teto e representada em um vetor de 3 dimensões.

Na prática, deve-se anexar o GameObject que contém a câmera principal ao script do *XR Origin*, um prefab do pacote *XR Interaction Toolkit*, como mostrado na Figura 3.34

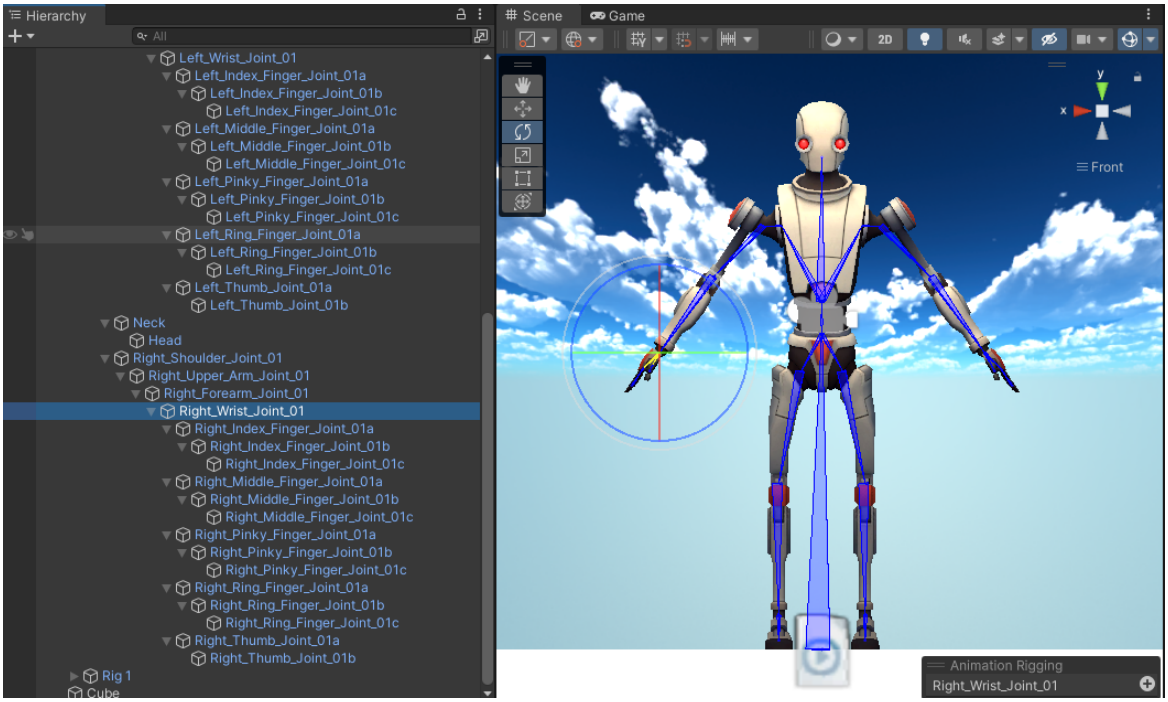

Figura 3.33 – Árvore do esqueleto e avatar virtual, Robot Kyle.

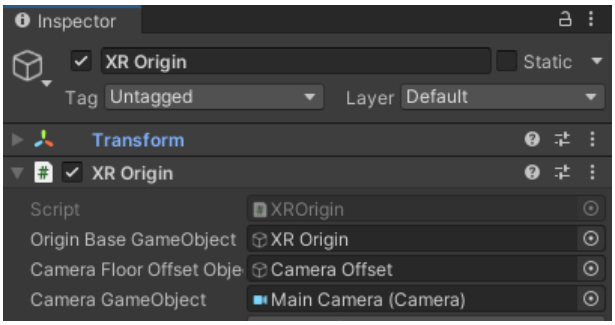

Figura 3.34 – Informações do GameObject *XR Origin* e conexão de seus componentes.

Com a câmera conectada ao HMD, resta conectar a câmera ao modelo 3D. Para isso, cria-se um objeto dentro do esqueleto e adiciona-se o Componente *Multi-Parent Constrain*. Este possui a função de conectar a cabeça e mover também os modelos dos próximos ossos de forma gradual para evitar deformações bruscas do modelo.

3.6.1.2 Mãos e braços

Os controles, por serem segurados pelas mãos, permitem saber a localização e orientação no mundo virtual apenas destas. Ao anexar um modelo de mão a posição do controle, este se comporta como "mãos flutuantes". Deseja-se então, adicionar braços e antebraços ligados ao ombro sem variações de comprimento, mas com a variação dos ângulos de rotação para alcançar determinada posição. Para isto, aplica-se a cinemática inversa por meio de um componente chamado *Two Bone IK Constraint*.

Para uso, deve-se criar um esqueleto de animação (*Rigging Animation*) inicial e

parentear a ele um GameObject com o componente *Two Bone IK Constraint*. Dentro deste componente, associa-se o pulso, o antebraço e o braço com os parâmetros *Tip, Mid, Root*, respectivamente. O componente então cria dois GameObjects automaticamente, o *target* e o *hint*. O primeiro é usado para localização da mão, logo, alinha-se a suas transformadas. O segunda é usado como referência para saber qual direção o cotovelo deve apontar. Como uma posição no espaço pode ser alcançada por múltiplas combinações de ângulos, o *hint* ajuda o braço a evitar posições não naturais.

E por fim, cria-se um script atrelado ao modelo do robô para conexão do HMD ao *target* da cabeça e dos controles com as mãos chamado *VRRig.cs*, anexo B. O código tem função de calcular o deslocamento de offset das posições de partes do modelo com a posição dos controles e do HMD, para então sobrepor ambos. Para correção da orientação e de posição de offset, roda-se o jogo com o equipamento ligado e simultaneamente altera-se os valores para alinhamento de posição e orientação mais próximos de como o corpo físico realmente está. Os valores são salvos e mostrados na Figura 3.35.

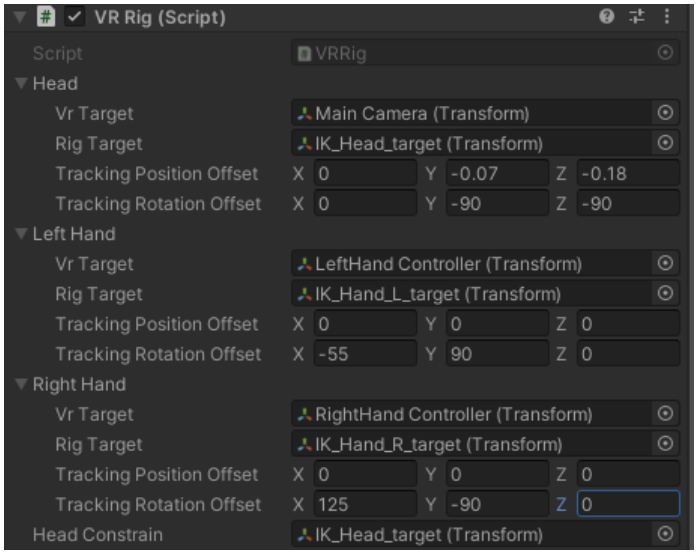

Figura 3.35 – Informações de offset da cabeça e dos controles.

3.6.2 Representação da parte inferior

Desenvolveu-se 3 técnicas diferentes como tentativa inicial de representação das pernas. Cada uma com suas vantagens, desvantagens e nível de complexidade que serão mostrados a seguir. Os códigos criados estão mostrados no Anexo B.

3.6.2.1 Por máquina de estado:

A primeira técnica utiliza o sistema de animação de jogos tradicionais representado por uma máquina de estados e sobreposição de animações para um conjunto específico de ossos de um esqueleto como na Figura 3.36.

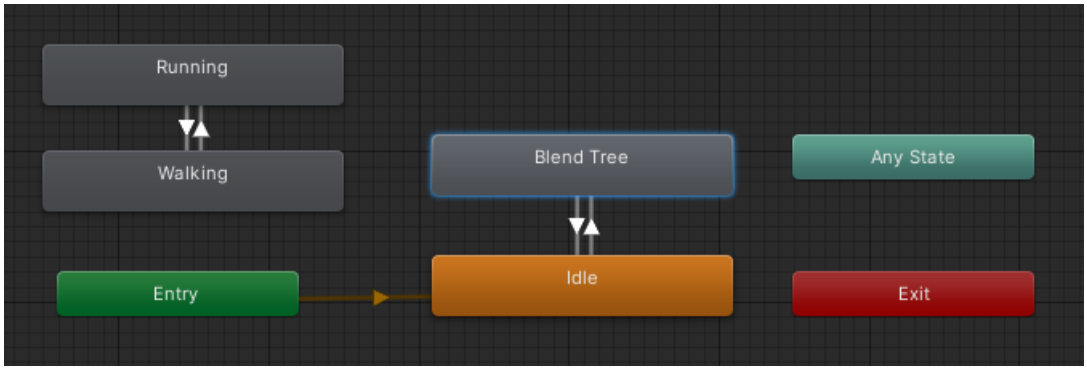

Figura 3.36 – Aba de animação por máquina de estados

Esta técnica requer baixar animações representando cada tipo de ação que se quer simular e então acessá-los por uma mudança de estado, na qual a animação irá começar caso uma condição seja atendida. Os scripts *AnimationStateController.cs* e *BlendTreeController.cs* foram desenvolvidos e os movimentos do corpo resultantes são bem mais naturais e semelhantes aos de uma pessoa real. Porém devido a falta de sincronização do movimento do braço com as pernas, por ser totalmente independente do balanço do braço, esta técnica falta de imersão e houve dificuldades na mesclagem de movimentos da parte superior com a inferior.

3.6.2.2 Por quatérnions:

A segunda técnica utiliza de 4 sensores de posição para orientação das coxas e das canelas em quatérnions de cada perna como mostra a tabela 3.2.

Tabela 3.2 – ID da IMU associada a partes das pernas

ID	1	2	4	5
POSIÇÃO	Coxa Direita	Canela Direita	Coxa Esquerda	Canela Esquerda

Este forma de representação requer tratamento dos dados e diferenciação da ID de cada sensor, além de um estudo profundo para compreensão desta forma de orientação. Com os dados em quatérnions tratados e com a ID da IMU correspondente, conecta-se cada sensor a uma parte diferente da perna do avatar pelo script *LegsRotation.cs*.

Existe uma posição padrão inicial que se deseja alcançar representado por um quatérnion q_I, um quatérnion de *offset* q_O e o quatérnion do sensor q_D representando uma rotação absoluta à orientação do mundo. A equação 3.1 demonstra que um quatérnion q_3 é a aplicação de um quatérnion q_2 em um q_1.

$$\begin{aligned} q_3 &= q_1 * q_2 \\ q_I &= q_D * q_O \end{aligned} \tag{3.1}$$

Encontrando o quatérnion de *offset* pela multiplicação do inverso do quatérnion dos dados pela posição inicial como mostra a equação 3.2

$$\begin{aligned} q_D^{-1} * q_I &= q_D^{-1} * q_D * q_O \\ q_O &= q_D^{-1} * q_I \end{aligned} \tag{3.2}$$

Após criado os códigos para o cálculo de um quatérnion de *offset* e multiplicá-los com o quatérnion dos dados. Tem-se a posição e o sentido de orientação corretas e alinhadas com o do ambiente virtual.

A dificuldade para se trabalhar com quatérnions, assim como definir um plano ou ponto intermediário de delimitação do fim de ciclo de onda para o cálculo de seu período, fez-se com que esta técnica fosse muito difícil de ser realizada, além de utilizar demasiados sensores e obter uma lenta resposta da variação dos ângulos por algum motivo desconhecido.

3.6.2.3 Por ângulos de Euler:

Já a terceira técnica se faz uso da mesma IMU que determina a velocidade para o movimento das pernas e aplica a angulação em uma das pernas e inverte na outra utilizando ângulos de Euler. Para movimentação das pernas, deve-se salvar o vetor de posição inicial que a perna se encontra. E então adicionar a esta orientação os ângulos recebidos do sensor. Devido a posição inicial de uma perna em relação a outra já ser invertida, a adição do vetor na orientação de cada perna não precisa ser invertida como mostra o trecho de código 3.37

```
void Update()
{
    leftLeg.transform.eulerAngles = leftStartRotation + new Vector3(0, 0, -udp.y_data);
    rightLeg.transform.eulerAngles = rightStartRotation + new Vector3(0, 0, -udp.y_data);

    //Debug.Log(leftLeg.transform.eulerAngles);
}
```

Figura 3.37 – Adição da variação da angulação para cada perna

Apesar do fato de esta ser a solução mais simples, esta foi a que produziu um resultado mais satisfatório em relação a conforto visual, imersão e sincronia com a caminhada.

3.7 Design do ambiente virtual

Para finalizar, com a utilização do software Blender, modelou-se uma pista de corrida com marcações azuis a cada 100 metros para acompanhamento da distância percorrida. E aplicou-se um céu com nuvens, além de outros cenários para testes.

4 Resultados

Este capítulo decorre sobre os resultados finais da estimação da velocidade e representação do avatar.

4.1 Sistema final

O sistema completo final, com foco no software desenvolvido e no fluxo dos dados para a reprodução do projeto, é demonstrado na Figura 4.38. As formas abaloadas representam as entradas dos dados pelo movimento de uma pessoa; os retângulos representam os códigos principais criados em suas linguagens de programação; o losangolo, um plugin instalado na plataforma Unity; e os cilindros, os gameobjects no cenário ativo.

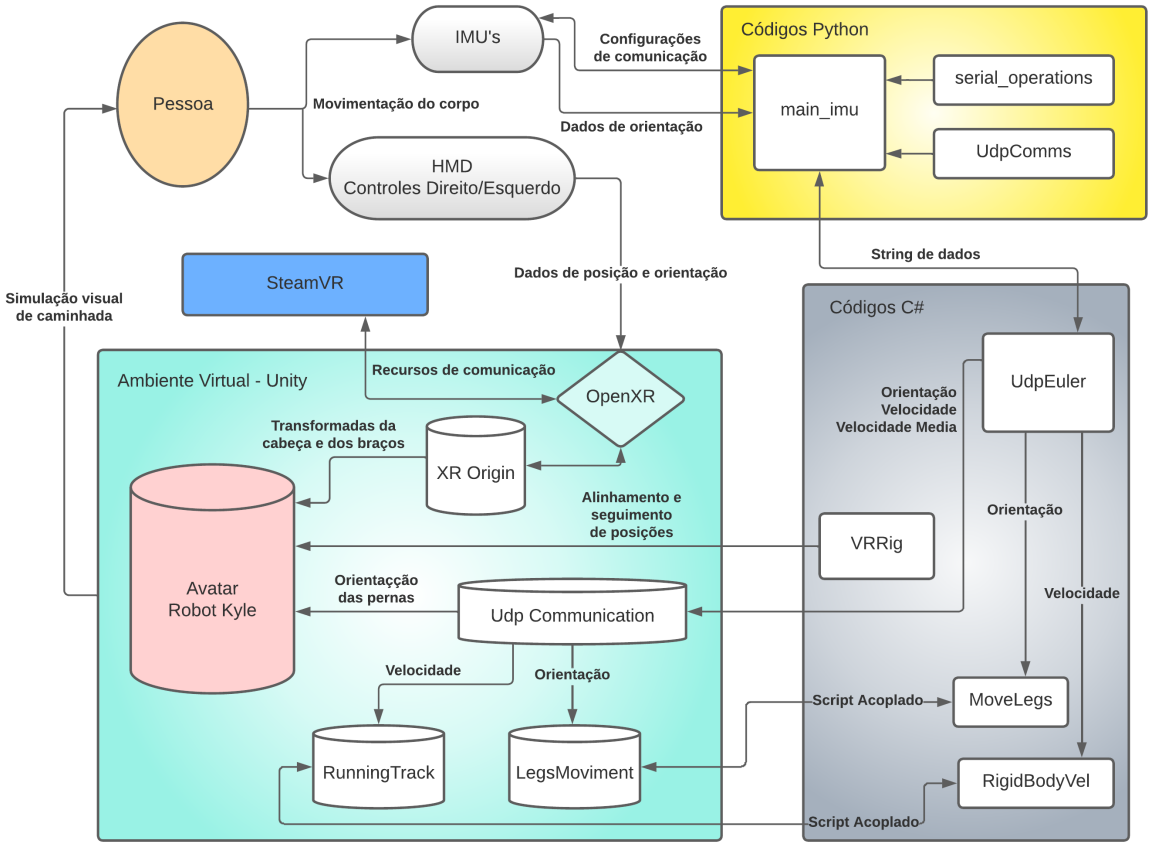

Figura 4.38 – Diagrama do fluxo de dados

4.2 Velocidade estimada

São mostrados as saídas dos códigos desenvolvidos para determinação e validação da velocidade estimada.

4.2.1 Testes iniciais

Os valores dos períodos médios dos últimos 10 valores encontrados, e suas frequências correspondentes para diferentes velocidades setadas pela esteira é mostrada na tabela 4.3.

Tabela 4.3 – Período e frequência dos passos por velocidade

Velocidade (km/h)	1	1.5	2	2.5	3	3.5	4
Período (s)	3.42	2.64	1.95	1.67	1.48	1.29	1.17
Frequência (1/s)	0.29	0.38	0.51	0.60	0.67	0.78	0.85

4.2.2 Regressão linear

A representação da regressão linear da velocidade pela frequência é mostrada no gráfico 4.39.

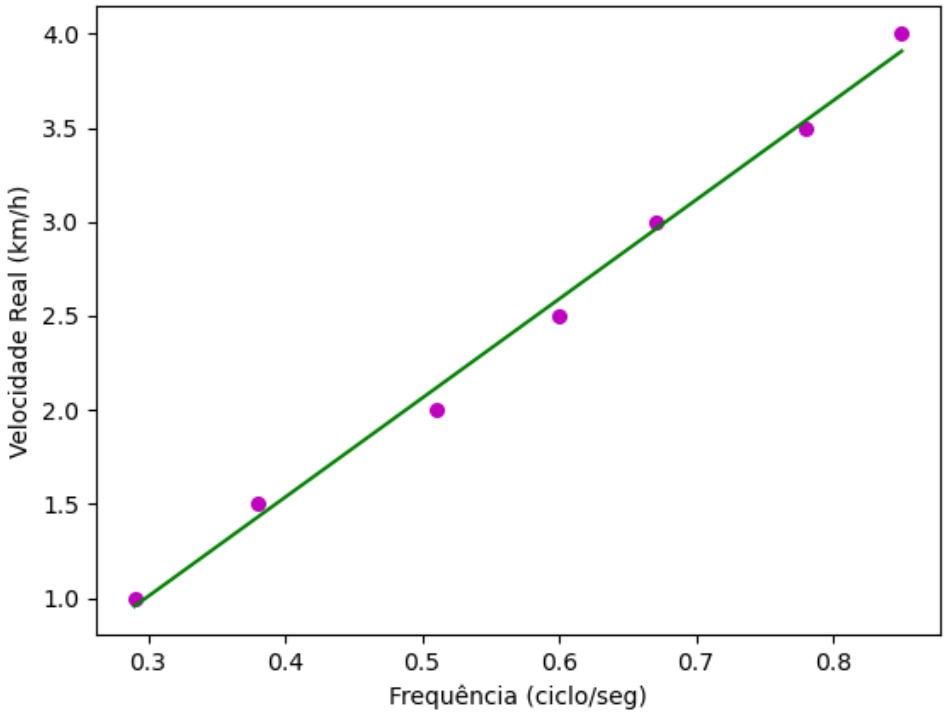

Figura 4.39 – Gráfico de regressão linear da velocidade pela frequência

A equação de regressão linear resultante aplicada ao conjunto de dados é mostrada abaixo:

$$y = 5.27276877x - 0.5732709427 \tag{4.1}$$

4.2.3 Validação do modelo

Os valores de ângulo e da velocidade estimada a cada meio ciclo de onda pelo tempo para velocidades de 1, 2.5 e 4 quilômetros por hora são mostrados nas Figuras 4.40, 4.41, 4.42.

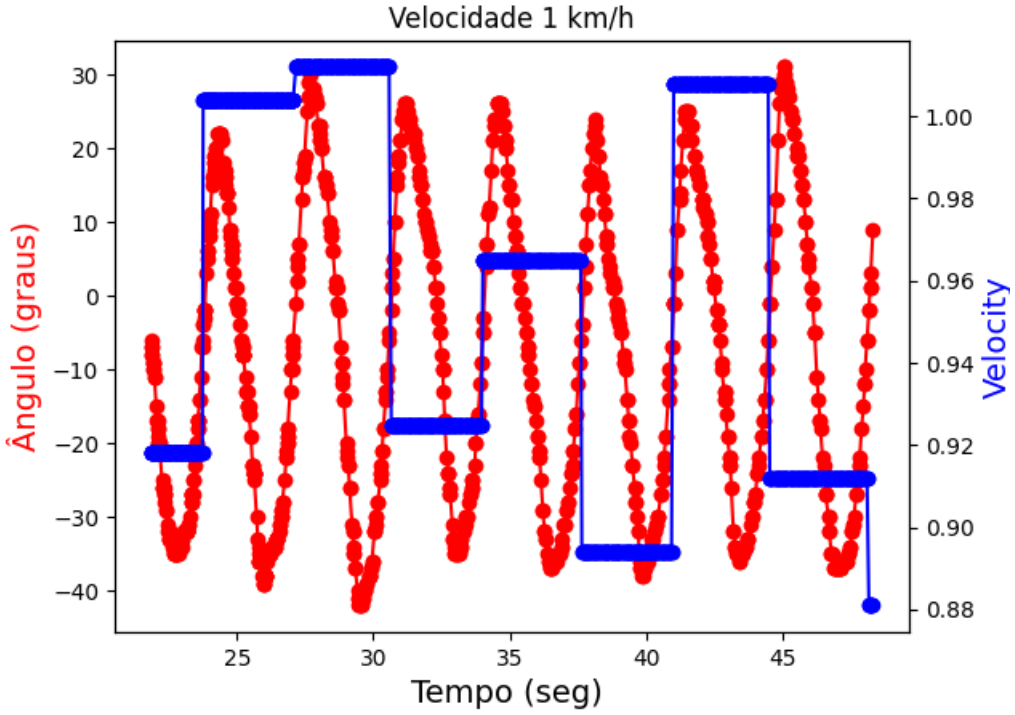

Figura 4.40 – Gráfico da velocidade estimada para velocidade de 1 km/h

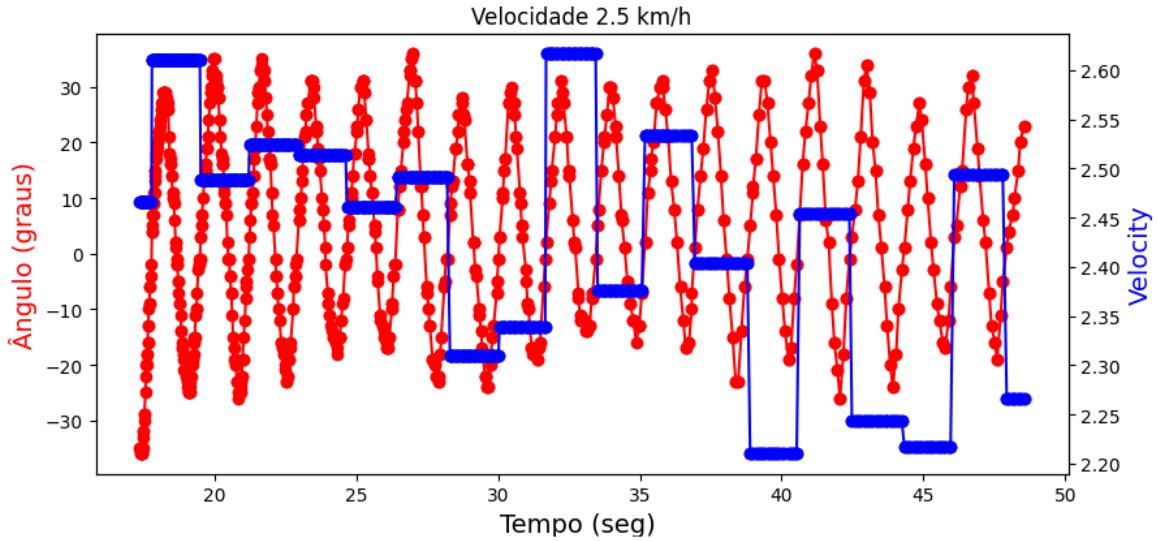

Figura 4.41 – Gráfico da velocidade estimada para velocidade de 2,5 km/h

Foi utilizado 10 amostras de velocidade estimadas por ciclo de onda, e então salvas no arquivo *velocity.txt* e a partir deles, pode-se calcular uma velocidade estimada média mais consistente e aproximada da velocidade real constante. A Tabela 4.4 compara as velocidades reais com as estimadas pela regressão linear adotada.

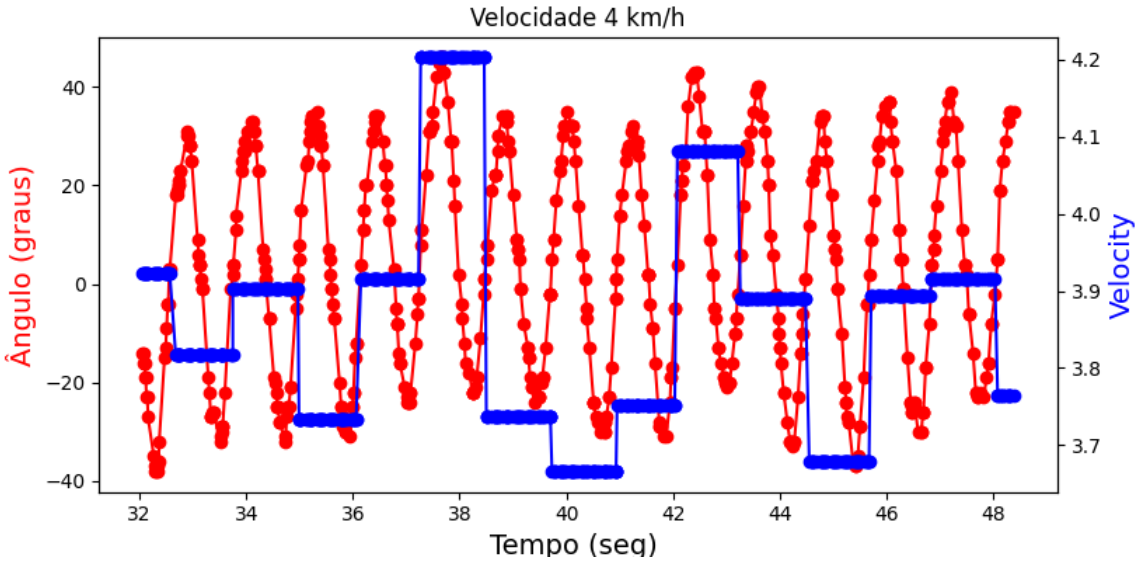

Figura 4.42 – Gráfico da velocidade estimada para velocidade de 4 km/h

Tabela 4.4 – Tabela de velocidades reais e estimadas

Velocidade Real (km/h)	1	2.5	4
Velocidade Estimada	0.96	2.45	3.91

É possível fazer a média móvel para análise da velocidade média em tempo-real. Criou-se uma lista com até n valores, a escolha do programador, para serem armazenados e feito o cálculo da média. Para cada novo valor para quando a lista atinge o número máximo, o mais antigo é removido do cálculo.

4.3 Avatar virtual

A representação e comparação do posicionamento do corpo do usuário e de seus movimentos em terceira e primeira pessoa, em um ambiente virtual, são mostrados nas imagens 4.43 4.44 4.45

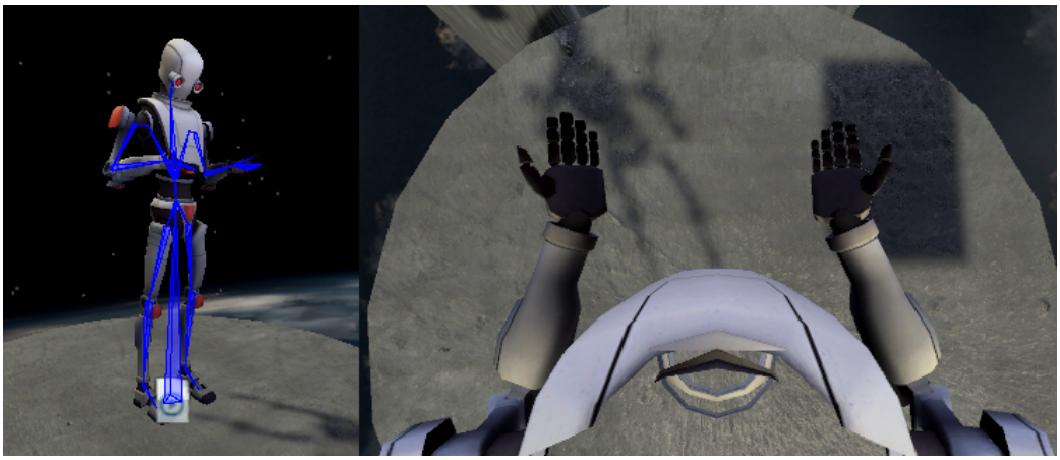

Figura 4.43 – Visualização em 3ª e 1ª pessoa do avatar

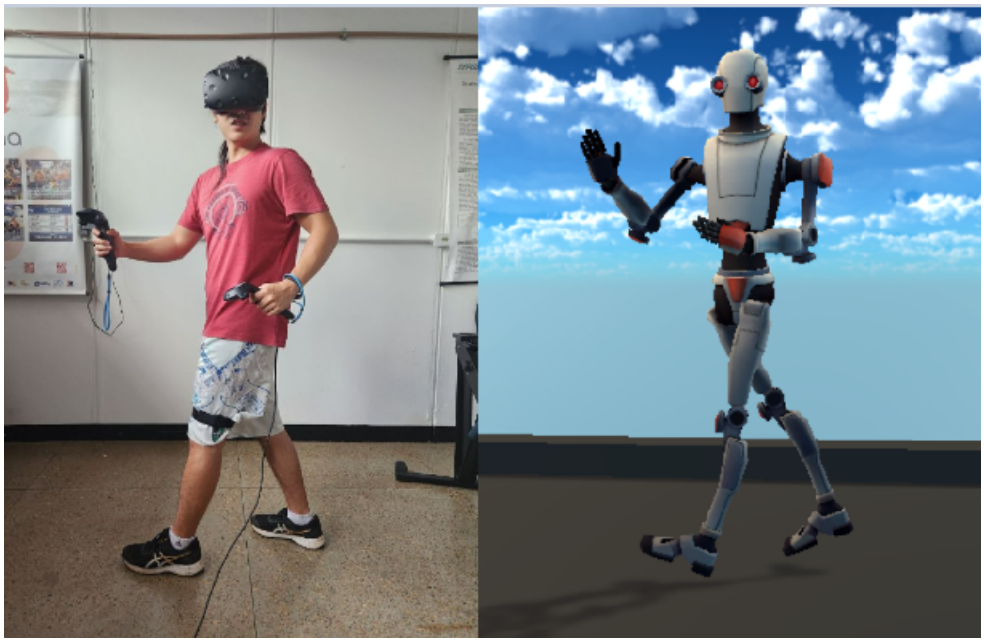

Figura 4.44 – Exemplo 1: Comparação do usuário com o avatar

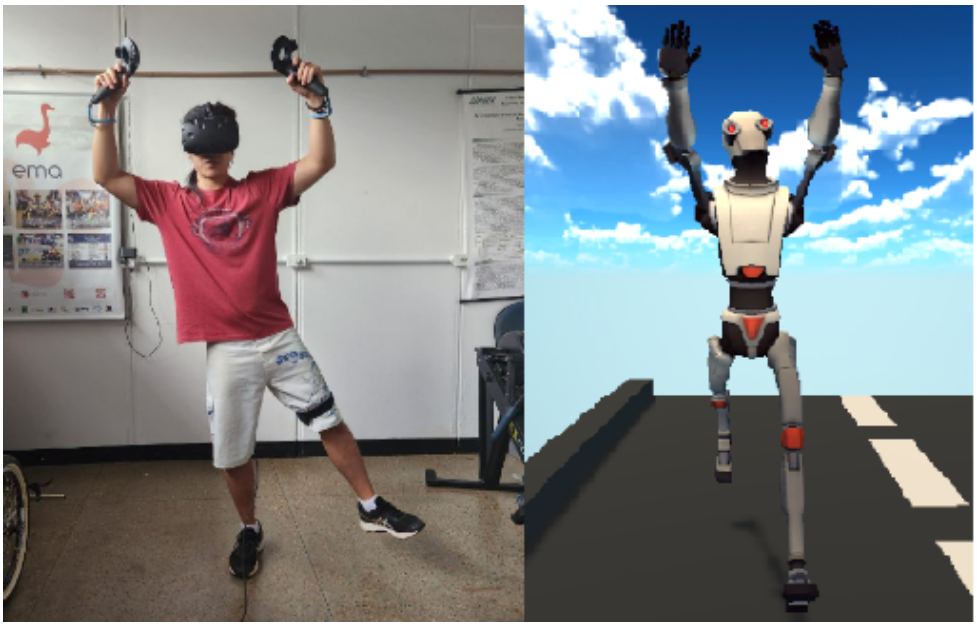

Figura 4.45 – Exemplo 2: Comparação do usuário com o avatar

Devido a natureza da atividade de caminhada, a representação do avatar é satisfatória para o conjunto de movimentos apresentados na simulação. Porém, também percebe-se algumas inconsistências de posicionamento.

Os resultados mostram que o movimento das pernas ficou sincronizado com a caminhada, porém para certas posições das mãos, o avatar é incapaz de reproduzir corretamento as angulações do usuário. Os erros mais agravantes são para movimentos de grande amplitude e acima da cabeça. Um dos motivos é que os braços do avatar são curtos demais para a distância dos controles, fazendo com que eles fiquem em posição esticada antes mesmo das do usuário. Os membros inferiores também possuem limitações de 2 dos 3 eixos de

movimento possíveis.

4.4 Simulação completa

A aparência final do experimento simulado para um cadeirante, para a validação da velocidade estimada e para a visualização da simulação virtual criada é demonstrada nas imagens 4.46 e 4.47.

Figura 4.46 – Experimentos para simulação de cadeirante e verificação do modelo

Figura 4.47 – Simulação final em 3ª e 1ª pessoa

O experimento final simulado para representação de um cadeirante, obteve resultados satisfatórios e não muito diferentes com o experimento na esteira na estimação da velocidade. O único ponto a se tomar cuidado é na diferença de altura que o usuário sente em cada um dos experimentos, causando assim uma pequena sobreposição ou flutuação das pernas com a pista de corrida. Este problema é facilmente consertado ajustando transformada de altura da pista. Apesar da falta de visão do mundo real durante o experimento na esteira, devido a baixa velocidade, foi consideravelmente seguro realizá-lo. As travas laterais servem como guia e a chave de emergência desliga a esteira caso o paciente se afaste demais do meio dela. Um outro ponto importante foi na aplicação da velocidade na pista, ao invés do avatar. Foi feito dessa maneira pois devido ao rastreamento do avatar com as posições globais dos controles e da cabeça, ao se mover o personagem, este apresentava falhas de movimento e tremores ao se deslocar. Logo, com a aplicação da velocidade na pista, este problema foi resolvido, e a sensação de movimentação para a frente se manteve, pois a velocidade relativa entre eles continua a mesma. Os resultados como um todo tiveram seus objetivos propostos concluídos.

5 Conclusão

5.1 Conclusões e discussões

Durante o desenvolvimento, é esperado o surgimento de erros inesperados. Os dados e gráficos obtidos para compreensão de seus resultados foram essenciais para solução de dificuldades enfrentadas.

O projeto demonstra o potencial da realidade virtual e da robótica como uma tecnologia de assistência para tratamentos. A pesquisa e desenvolvimento dessa tecnologia é um passo importante para melhorar a qualidade de vida das pessoas com deficiências de mobilidade, e quanto mais atrativa, mais pessoas interessadas para solucionar esse problema.

As IMUs permitiram a inclusão e representação dos movimentos das pernas, um processo ainda pouco utilizado atualmente no jogos atuais. Apesar de simplória, essa forma para controlar o movimento de um personagem virtual, permitiu que indivíduos com deficiência simulem ações de caminhada de maneira sincronizada e que não é possível com métodos tradicionais. Apesar da estimativa da velocidade por cada ciclo apresentar divergências medianas do valor real, a estimativa da velocidade média para pelo menos 10 ciclos já possui um erro médio menor e condizente com a realidade. Para diferentes indivíduos, a estimativa da velocidade pode também variar devido a diferentes estaturas e formas de caminhar. Poderia se calibrar para cada pessoa individualmente para maior acurácia da estimativa da velocidade, ou também a criação de uma tabela relacionando altura com a frequência dos passos para diferentes indivíduos para simplificar o processo de calibrar toda vez.

Este projeto é importante não só para o campo da reabilitação, mas também abre uma ampla gama de possibilidades de aplicação. O uso de IMUs com tecnologia de realidade virtual também tem o potencial de ser usado em outros campos, como jogos, treinamento esportivo e até mesmo exploração espacial, além de poder proporcionar uma maneira única e envolvente para indivíduos praticarem e melhorarem seus movimentos.

Resumidamente, o interesse pelo assunto sobre realidade virtual e criação de jogos foi também um fator importante e relevante para a escolha do trabalho e dos desafios propostos. o projeto envolveu diversas áreas da mecatrônica durante seu processo como: robótica, instrumentação, modelagem 3D, sistemas operacionais, tratamento de dados e comunicação entre dispositivos para realização deste. Os resultados obtidos, os estudos realizados e o desenvolvimento do projeto foram de grande satisfação pessoal e positiva.

5.2 Trabalhos futuros

Como sugestão para possíveis trabalhos futuros, poderia-se realizar testes para outros pacientes, tanto para se ter mais opiniões sobre a experiência e imersão, quanto para a análise das diferenças de altura e formas de se caminhar no quanto elas impactam para a estimação a velocidade. O projeto abre oportunidades para qualquer forma de atividade física, ou simulação de situações com a realidade virtual e seria interessante ver o que o próximo aluno poderá criar com esta tecnologia.

Referências

BALBINO, H. S. S. Desenvolvimento de sistema de teleoperação para robô humanoide. **Trabalho de Graduação em Engenharia de Controle e Automação**, Faculdade de Tecnologia, Universidade de Brasília, DF, Publicação TG-030/2016, p. 0-49, 2016. Citado na p. 15.

BARBOSA, R. R. C. Classificação de Movimentos de Mão para Próstese Robótica Utilizando Sinais Mio-elétricos. **Trabalho de Graduação em Engenharia de Controle e Automação**, Faculdade de Tecnologia, Universidade de Brasília, DF, Publicação TG-06/2016, p. 67, 2021. Citado na p. 15.

BASTOS, V. B. Desenvolvimento e integração de ambiente 3D de simulação para algoritmos de navegação por imagem. **Unicamp Campinas,SP**, v. 0, p. 21–21, jan. 2015. Citado na p. 30.

BELLE, F.; MACHADO, K. M.; BOTARELI, F. G. Os benefícios da gameterapia na reabilitação de idosos com diagnóstico de Acidente Vascular Cerebral. **Revista Neurociências**, v. 29, p. 1–15, out. 2021. DOI: 10.34024/rnc.2021.v29.12274. Disponível em: <https://periodicos.unifesp.br/index.php/neurociencias/article/view/12274>. Citado na p. 16.

ESPOSTO, D. S.; VERRI, E. D.; FABRIN, S.; REGALO, S.; FIOCO, E. M.; ZANELLA, C. Beneficios da realidade virtual no processo de reabilitação de indivíduos Pós-AVE: revisão sistemática da literatura. **Ling. Acadêmica**, v. 7, p. 41–52, 2017. Citado na p. 16.

GÓMEZ-SÁNCHEZ, E.; MARTÍNEZ-VELASCO, M.; RIBERA, A.; BESCÓS, R. Virtual Reality and Inertial Measurement Units for Gait Rehabilitation: A Systematic Review. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 17, n. 4, p. 888, 2017. Citado na p. 15.

GÓMEZ-SÁNCHEZ, E.; MARTÍNEZ-VELASCO, M.; RIBERA, A.; BESCÓS, R. Virtual Reality and Inertial Measurement Units for Walking Rehabilitation: A Systematic Review. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 18, n. 5, p. 1392, 2018. Citado na p. 15.

HANGAR33. **Arfagem, Guinada, Rolagem**. Disponível em: <http://blog.hangar33.com.br/quais-sao-os-principais-fatores-que-influenciam-o-voo/>. Citado na p. 20.

MACEDO, C. d. S. G.; GARAVELLO, J. J.; OKU, E. C.; MIYAGUSUKU, F. H.; AGNOLL, P. D.; NOCETTI, P. M. BENEFÍCIOS DO EXERCÍCIO FÍSICO PARA A QUALIDADE DE VIDA. **Revista Brasileira de Atividade Física amp; Saúde**, v. 8, p. 19–27, out. 2012. Disponível em: <https://rbafs.org.br/RBAFS/article/view/875>. Citado na p. 16.

OMS. **World report on disability**. 2011. Disponível em: <http://www.who.int/publications/i/item/9789241564359>. Acesso em: 26 dez. 2022. Citado na p. 14.

PNG, G. **Ângulos de Euler**. Disponível em: <https://www.gratispng.com/png-pooiq1>. Citado na p. 27.

PNS. **Portadores de deficiência no Brasil**. 2021. Disponível em: <https://brasil.estadao.com.br/blogs/vencer-limites/pessoas-com-deficiencia-no-censo-2022/>. Acesso em: 26 dez. 2021. Citado na p. 14.

PRINTING, A. A. 3. **Textures**. Disponível em: <https://all3dp.com/2/best-sites-for-free-3d-textures>. Citado na p. 31.

RAIN, D. **Modelos OSI e TCP/IP**. Disponível em: <https://www.datarain.com.br/blog/qual-diferenca-entre-modelo-osi-e-modelo-tcpip>. Citado na p. 26.

RIZZO, A.; KIM, G.; CASEY, K.; ROBBINS, M.; DIFEDE, J. The Use of Virtual Reality for Rehabilitation: A Review. **Physical Medicine and Rehabilitation Clinics of North America**, Elsevier, v. 15, n. 3, p. 729–739, 2004. Citado na p. 15.

RODRIGUES, G. P.; PORTO, C. d. M. Realidade Virtual: conceitos, evolução, dispositivos e aplicações. **EDUCAÇÃO**, v. 1, n. 3, p. 97–109, jun. 2013. DOI: 10.17564/2316-3828.2013v1n3p97-109. Disponível em: <https://periodicos.set.edu.br/educacao/article/view/909>. Citado na p. 29.

SAÚDE, M. da. **Acidentes de trabalho no Brasil**. 2018. Disponível em: <http://www.saude.gov.br/saude-de-a-z/acidentes-de-trabalho>. Acesso em: 26 dez. 2022. Citado na p. 14.

WIKIPEDIA. **Quaternions**. Disponível em: <https://en.wikipedia.org/wiki/Quaternion>. Citado na p. 28.

WORKS, M. **Robotic Arm Joints Positioning**. Disponível em: <https://www.mathworks.com/discovery/inverse-kinematics.html>. Citado na p. 29.

YOST LABS. **3-Space Sensor System User's Guide**. 2007-2017. Disponível em: <https://yostlabs.com/wp/wp-content/uploads/pdf/3-Space-Sensor-Users-Manual-1.pdf>. Citado na p. 39.

Anexos

Anexo A – Gráficos dos sensores

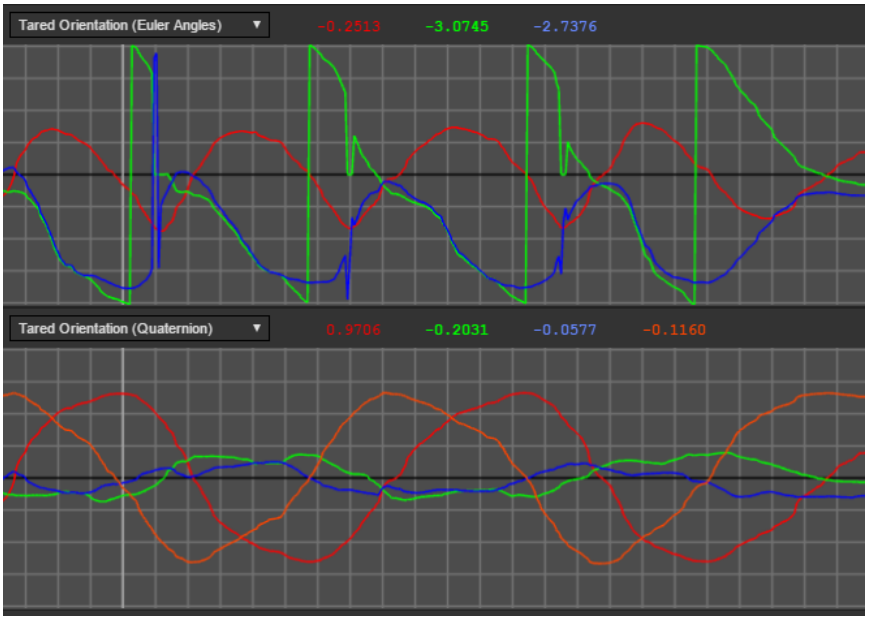

Figura A.48 – Data Chart: Rotação em eixo X - Unidades: Ãngulos de Euler(Em cima), Quatérnions (Embaixo)

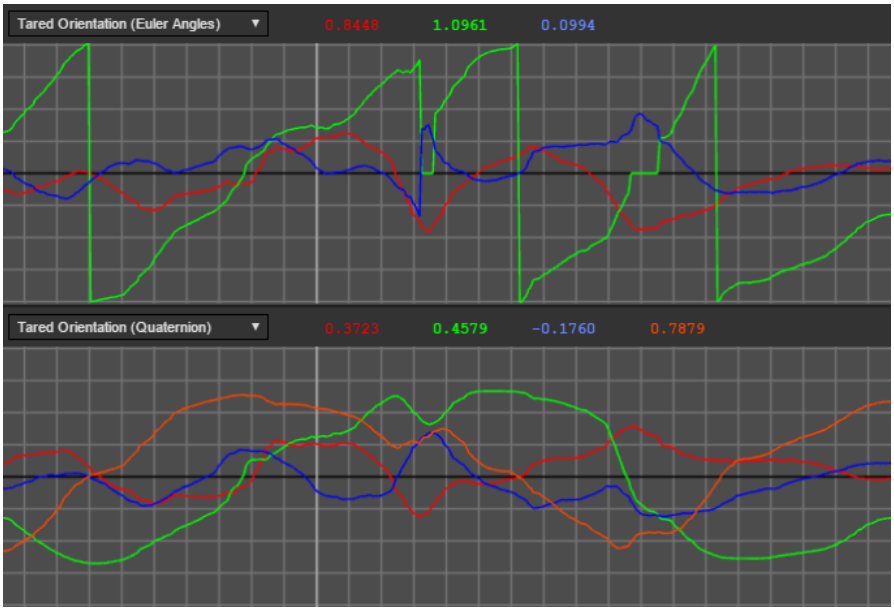

Figura A.49 – Data Chart: Rotação em eixo Y - Unidades: Ãngulos de Euler(Em cima), Quatérnions (Embaixo)

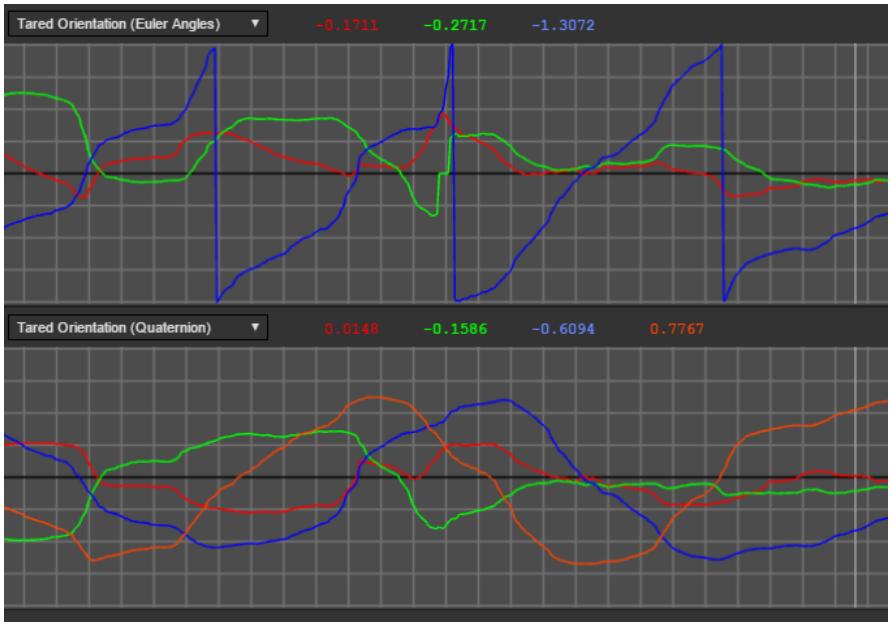

Figura A.50 – Data Chart: Rotação em eixo Z - Unidades: Ãngulos de Euler(Em cima), Quatérnions (Embaixo)

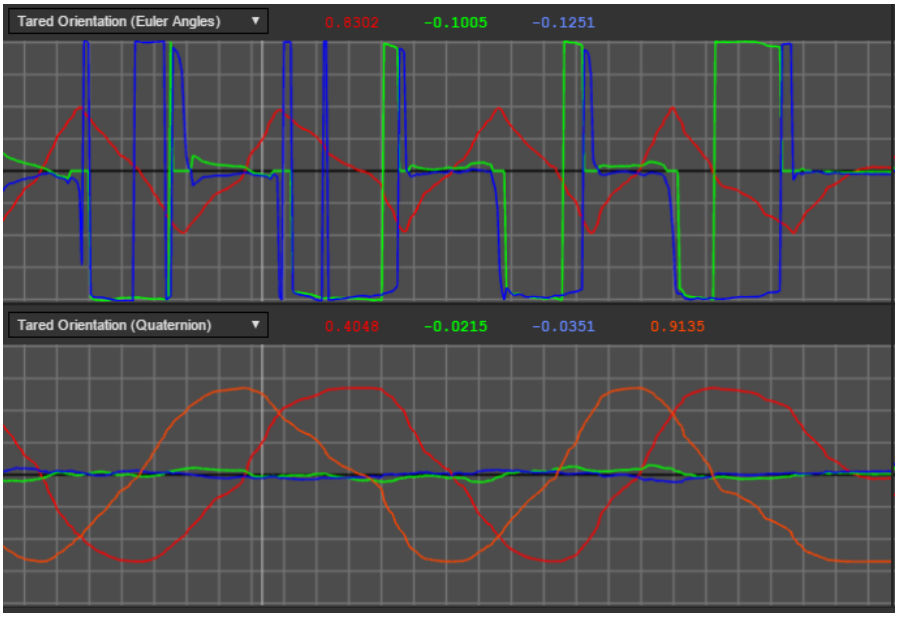

Figura A.51 – Data Chart: Calibrada rotação em eixo X

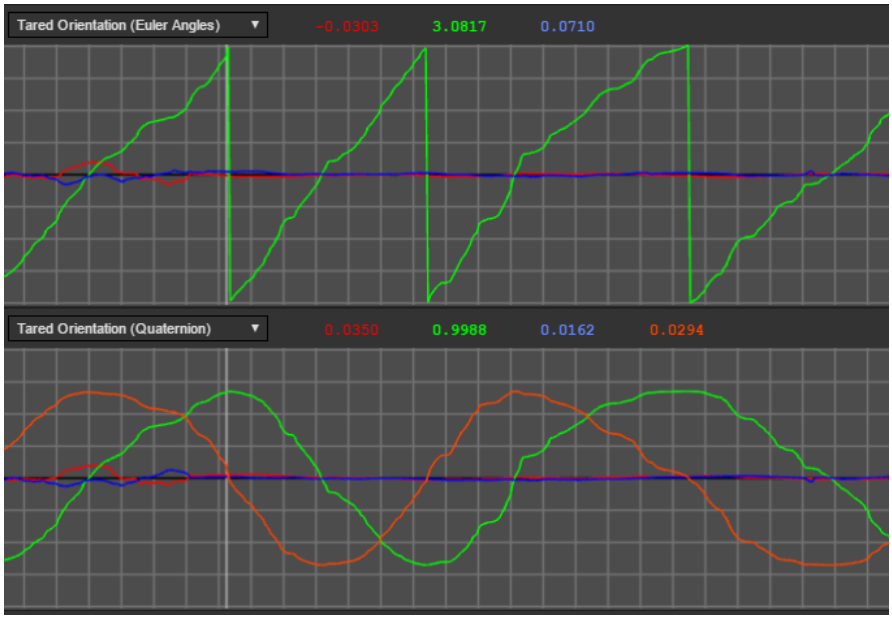

Figura A.52 – Data Chart: Calibrada rotação em eixo Y

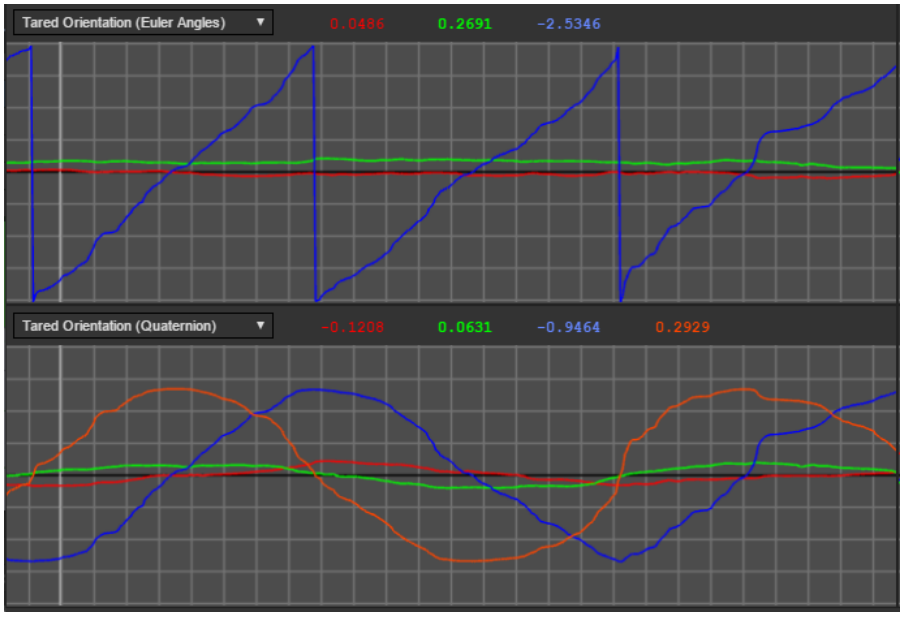

Figura A.53 – Data Chart: Calibrada rotação em eixo Z

Anexo B – Códigos de programação

B.1 Código principal: *main_imu.py*

Código B.1 – Código em Python

```
"""Getting IMU Angles

"""

import numpy as np
import traceback
import serial_operations as serial_op
from colors import *
import UdpComms as U
import matplotlib.pyplot as plt


# Create UDP socket to use for sending (and receiving)
sock = U.UdpComms(udpIP="127.0.0.1", portTX=8000, portRX=8001,
    enableRX=True, suppressWarnings=True)
# Set parameters that will be configured
imu_configuration = {
    "disableCompass": True,
    "disableGyro": False,
    "disableAccelerometer": False,
    "gyroAutoCalib": True,
    "filterMode": 1,
    "tareSensor": True,
    "logical_ids": [1, 2, 3],
    "streaming_commands": [1, 255, 255, 255, 255, 255, 255, 255]
}
prev_angle = 0
count_value = 0
timer, velocity, angle = [], [], []

#Starting serial connection
serial_port = serial_op.initialize_imu(imu_configuration)

while True:
    try:
        data = sock.ReadReceivedData()

        if data != None:
            #print(data)
            #print(type(data))

            data_splited = (data.replace(",",".")).split(':')
```

```
42
43              #Saving data for plot
44              timer.append(float(data_splited[0]))
45              velocity.append(float(data_splited[1]))
46              #Saving velocity values for median calculation
47              with open("velocity.txt", "w") as file:
48                  for i in velocity:
49                      file.write(str(i) + '\n')
50              angle.append(float(data_splited[2]))
51
52              print(f"Count: {count_value} Timer:
                   {timer[count_value]} Velocity:
                   {velocity[count_value]} Y Angle:
                   {angle[count_value]}")
53
54              count_value += 1
55
56          #When samples data is enough, plot graph
57          if count_value == 600:
58              print( "Plotting Graph ")
59
60              fig, ax = plt.subplots()
61              ax.plot(timer, angle, color="red", marker="o")
62              ax.set_xlabel("Tempo (seg) ", fontsize=14)
63              ax.set_ylabel(" ngulo  (graus)", color="red",
                   fontsize=14)
64              ax2 = ax.twinx()
65              ax2.plot(timer, velocity, color="blue", marker="o")
66              ax2.set_ylabel("Velocity", color="blue", fontsize=14)
67              plt.title('Velocidade 4 km/h')
68              plt.show()
69              fig.savefig('./angle_vel_time.jpg', format='jpeg',
                   dpi=100, bbox_inches='tight')
70              timer, velocity, angle = [], [], []
71              count_value = 0
72
73          bytes_to_read = serial_port.inWaiting()
74
75          # Read data from sensors and sent to Unity in better
               format.
76          if 0 < bytes_to_read > 80:
77              data = serial_port.read(bytes_to_read)
78              if data[0] != 0:
79                  continue
80
81              euler_data = serial_op.extract_eulers(data)
82              y_data = round(euler_data[1] * 180 / 3.14)
83              # check for 0 error values
84              if y_data == 0:
85                  y_data = prev_angle
86
87              # print(f"IMU{data[1]}:" + str(y_data))
```

```
88
89              sock.SendData(str(y_data))
90              prev_angle = y_data
91
92      except KeyboardInterrupt:
93          print(GREEN, "Keyboard excpetion occured.", RESET)
94          serial_port = serial_op.stop_streaming(serial_port,
               imu_configuration['logical_ids'])
95          break
96      except Exception:
97          print(RED, "Unexpected exception occured.", RESET)
98          print(traceback.format_exc())
99          print(GREEN, "Stop streaming.", RESET)
100         serial_port = serial_op.stop_streaming(serial_port,
               imu_configuration['logical_ids'])
101         break
102
103
104
105 # def filter(q):
106 #     mf_window.pop(0)
107 #     mf_window.append(q)
108 #
109 #     x_list = [quat[0] for quat in mf_window]
110 #     y_list = [quat[1] for quat in mf_window]
111 #     z_list = [quat[2] for quat in mf_window]
112 #     w_list = [quat[3] for quat in mf_window]
113 #
114 #     x_list.sort()
115 #     y_list.sort()
116 #     z_list.sort()
117 #     w_list.sort()
118 #
119 #     qx = x_list[len(x_list) // 2]
120 #     qy = y_list[len(x_list) // 2]
121 #     qz = z_list[len(x_list) // 2]
122 #     qw = w_list[len(x_list) // 2]
123 #
124 #     return [qx, qy, qz, qw]
```

B.2 Comunicação serial: *serial_operations.py*

Código B.2 – Código em Python

```
1 import serial.tools.list_ports
2 import time
3 import numpy as np
4
5 #from colors import *
6
```

```
7  SMALL_IMU_DONGLE_PORT = 4128
8
9  # If permission denied error occurs in Linux try:
10 # sudo chmod 666 /dev/ttyACM0 -> with the correspondent COM port
11 def get_dongle_object():
12     """ Create a serial port object to operate with imu dongle
13     Returns:
14         serial_port: PySerial object
15     """
16     # Lists available ports and select dongle port
17     ports_list = serial.tools.list_ports.comports()
18     print("Ports available: ")
19     for wire in ports_list:
20         text = "Port: {0}\tSerial#:{1}\tDesc:{2} PID
               {3}".format(wire.device, wire.serial_number,
               wire.description, wire.pid)
21         print(text)
22         if wire.pid == SMALL_IMU_DONGLE_PORT:
23             portIMU = wire.device
24
25     # Instantiate seria port object
26     serial_port = serial.Serial(port=portIMU, baudrate=115200,
           timeout=0.01)
27     manual_flush(serial_port)
28
29     return serial_port
30
31 def manual_flush(serial_port):
32     """ Clean serial port buffer
33
34     Args:
35         serial_port: PySerial Object
36     """
37     while not serial_port.inWaiting() == 0:
38         serial_port.read(serial_port.inWaiting())
39
40 def create_imu_command(logical_id, command_number, arguments = []):
41     """ Create imu command string
42
43     Args:
44         logical_id: integer represents imu sensor configure to
               dongle
45         (configure in sensor suit)
46         command_number: integer represents a command (see all
               commands in
47         user manual)
48         arguments: list with arguments, if necessary
49
50     Return:
51         encoded string with the command in the correct format
52     """
53     # Create command
```

```
54      command = ">"+str(logical_id)+","+str(command_number)
55      if(len(arguments) != 0):
56          arguments_string = ","
57          for  argument in arguments:
58              arguments_string += str(argument)
59              arguments_string += ","
60          arguments_string = arguments_string[:-1]
61          command += arguments_string
62      command += '\n'
63      return command.encode()
64
65  def apply_command(serial_port, command, showResponse=False):
66      """ Apply command in sensor
67
68      Args:
69          serial_port: PySerial Object
70          command: encoded string with the command
71          showResponse: boolean that decides if output will be
                displayed
72      """
73      serial_port.write(command)
74      time.sleep(0.1)
75      if(showResponse):
76          while serial_port.inWaiting():
77              out = '>> ' +
                    serial_port.read(serial_port.inWaiting()).decode()
78          print(out)
79      time.sleep(0.1)
80      # return out
81
82
83
84  def stop_streaming(serial_port, logical_ids):
85      """ Apply stop streaming operation
86
87      Args:
88          serial_port: PySerial Object
89          logical_ids: list of sensors that the command should be
                applied
90      """
91      for id in logical_ids:
92          command = create_imu_command(id, 86)
93          apply_command(serial_port, command)
94
95  def start_streaming(serial_port, logical_ids):
96      """ Apply start streaming operation
97
98      Args:
99          serial_port: PySerial Object
100         logical_ids: list of sensors that the command should be
                applied
101     """
```

```
102     for id in logical_ids:
103         command = create_imu_command(id, 85)
104         apply_command(serial_port, command)
105     return serial_port
106 def clean_data_vector(data):
107     decoded_data = data.decode()
108     cleaned_data = decoded_data.replace('\r\n',' ').split(' ')
109     cleaned_data = list(filter(None, cleaned_data))[0].split(",")
110     cleaned_data = [float(d) for d in cleaned_data]
111     return cleaned_data
112 def write_command_read_answer(serial_port, logical_ids,
    command_number, arguments=[]):
113     # If it's streaming stop it
114     # stop_streaming(serial_port, logical_ids)
115     time.sleep(0.1)
116     manual_flush(serial_port)
117 
118     # Apply imu command to each logical id and saves it's answers
119     answer = []
120     for id in logical_ids:
121         time.sleep(0.1)
122         command = create_imu_command(id, command_number, arguments)
123         apply_command(serial_port, command)
124         time.sleep(0.1)
125         cleaned_data =
              clean_data_vector(serial_port.read(serial_port.inWaiting()))
126         answer.append(cleaned_data)
127 
128     # start_streaming(serial_port, logical_ids)
129     return answer
130 
131 def set_streaming_slots(serial_port, logical_ids, commands):
132     """ Set streaming slots
133 
134     Args:
135         serial_port: PySerial Object
136         logical_ids: list of sensors that the command should be
              applied
137         commands: list of integer that commands slots should be
              filled
138     """
139     for id in logical_ids:
140         command = create_imu_command(id, 80, commands)
141         print(command)
142         apply_command(serial_port, command, True)
143     return serial_port
144 
145 def configure_sensor(serial_port, configDict):
146     """ Apply common sensor configuration
147 
148     Args:
149         serial_port: PySerial Object
```

```
150         configDict: dictionary with sensor basic configuration {
151                 "disableCompass": Boolean,
152                 "disableGyro": Boolean,
153                 "disableAccelerometer": Boolean,
154                 "gyroAutoCalib": Boolean,
155                 "tareSensor": Boolean,
156                 "filterMode": Integer (see user's manual)
157                 "logical_ids": list of logical ids,
158                 "streaming_commands": list of streaming slots
159         }
160     """
161     if(configDict["disableGyro"]):
162         for id in configDict["logical_ids"]:
163             command = create_imu_command(id, 107)
164             apply_command(serial_port, command)
165 
166     if(configDict["disableAccelerometer"]):
167         for id in configDict["logical_ids"]:
168              command = create_imu_command(id, 108)
169              apply_command(serial_port, command)
170 
171     if(configDict["disableCompass"]):
172         for id in configDict["logical_ids"]:
173             command = create_imu_command(id, 109)
174             apply_command(serial_port, command)
175 
176     filterMode = configDict["filterMode"]
177     for id in configDict["logical_ids"]:
178         command = create_imu_command(id, 123, [filterMode])
179         apply_command(serial_port, command)
180 
181     if configDict["gyroAutoCalib"]:
182         for id in configDict["logical_ids"]:
183             command = create_imu_command(id, 165)
184             apply_command(serial_port, command)
185 
186     if configDict["tareSensor"]:
187         for id in configDict["logical_ids"]:
188             command = create_imu_command(id, 96)
189             apply_command(serial_port, command)
190 
191     # Forcing axis configuration to standard
192     # for id in configDict["logical_ids"]:
193     #     command = create_imu_command(id, 116, [0])
194     #     apply_command(serial_port, command)
195 
196 def tare_sensor(serial_port, logical_ids):
197     """ Apply tare sensor operation
198 
199     Args:
200         serial_port: PySerial Object
```

```
201         logical_ids: list of sensors that the command should be
                applied
202     """
203     for id in logical_ids:
204         command = create_imu_command(id, 96)
205         apply_command(serial_port, command)
206 
207 def get_sensor_information(serial_port, logical_ids):
208     """ Get some sensor current information such as filter mode
            and trust values
209 
210     Args:
211         serial_port: PySerial Object
212         logical_ids: list of sensors that the command should be
                applied
213     """
214     # Current filter mode
215     for id in logical_ids:
216         command = create_imu_command(id, 152)
217         apply_command(serial_port, command)
218     # Current accelerometer trust values
219     for id in logical_ids:
220         command = create_imu_command(id, 130)
221         apply_command(serial_port, command)
222 
223 # Get rotation matrix streamed in first slot.
224 def extract_rotation_matrix(data):
225     """ Manipulate data to obtain rotation matrix
226 
227     Args:
228         data: Raw data that sensor send
229 
230     Returns:
231         rotation matrix dictionary
232     """
233     decoded_data = data.decode()
234     list_data = decoded_data.replace('\r\n',' ').split(' ')
235     cleaned_list_data = list(filter(None, list_data))
236     rotatation_vector = cleaned_list_data[0][3:].split(',')
237     rotatation_vector = np.array(rotatation_vector,
            dtype=np.float64)
238     rotation_matrix = rotatation_vector.reshape((3,3))
239     return {'rotation_matrix': rotation_matrix}
240 
241 def extract_euler_angles(data):
242     """ Manipulate data to obtain rotation matrix
243 
244     Args:
245         data: Raw data that sensor send
246 
247     Returns:
248         rotation matrix dictionary
```

```
249     """
250     decoded_data = data.decode()
251     list_data = decoded_data.replace('\r\n',' ').split(' ')
252     cleaned_list_data = list(filter(None, list_data))
253     euler_vector = cleaned_list_data[0][3:].split(',')
254     euler_vector = np.array(euler_vector, dtype=np.float64)
255     return {euler_vector}
256 
257 def extract_eulers(data):
258     """ Manipulate data to obtain eulers angles
259         data: Raw data that sensor send
260     Returns:
261         eulers angles vector
262     """
263     decoded_data = data.decode()
264     list_data = decoded_data.replace('\r\n',' ').split(' ')
265     cleaned_list_data = list(filter(None, list_data))
266     euler_vector = cleaned_list_data[0][3:].split(',')
267     euler_vector = np.array(euler_vector, dtype=np.float64)
268     return euler_vector
269 
270 def initialize_imu(configuration_dict):
271     """ Initialize imu dongle and sensor
272 
273     Args:
274         configDict: dictionary with sensor basic configuration {
275                 "disableCompass": Boolean,
276                 "disableGyro": Boolean,
277                 "disableAccelerometer": Boolean,
278                 "gyroAutoCalib": Boolean,
279                 "tareSensor": Boolean,
280                 "filterMode": Integer (see user's manual)
281                 "logical_ids": list of logical ids,
282                 "streaming_commands": list of streaming slots
283         }
284 
285     Returns:
286         serial_port: PySerial Object
287 
288     """
289     # Find and open serial port for the IMU dongle
290     print("Getting imu object:")
291     serial_port = get_dongle_object()
292     print("Done.")
293 
294     # Clean outputs
295     manual_flush(serial_port)
296 
297     # Stop streaming
298     print("Stoping streaming.")
299     stop_streaming(serial_port, configuration_dict['logical_ids'])
300 
```

```
301
302     # Setting streaming slots, this means that while streaming
           sensors will send
303     # this data to the dongle as in page 29 - User manual:
304     # 0 - Differential quaternions;
305     # 1 - tared orientation as euler angles;
306     # 2 - rotation matrix
307     # 255 - No data
308     # See user manual to get more information
309     set_streaming_slots(serial_port,
310                         configuration_dict['logical_ids'],
311                         configuration_dict['streaming_commands'])
312
313     # Set magnetometer(explain it better), calibGyro if
           calibGyro=True and Tare sensor
314     print('Starting configuration: ')
315     configure_sensor(serial_port, configuration_dict)
316     print("Done.")
317
318
319     print("Starting streamnig.")
320     # Start streaming
321     start_streaming(serial_port, configuration_dict['logical_ids'])
322
323     print("IMU's ready to use.")
324
325     return serial_port
326
327
328 def reinitialize_imu(configuration_dict):
329     """ Initialize imu dongle and sensor
330
331     Args:
332         configDict: dictionary with sensor basic configuration {
333                 "disableCompass": Boolean,
334                 "disableGyro": Boolean,
335                 "disableAccelerometer": Boolean,
336                 "gyroAutoCalib": Boolean,
337                 "tareSensor": Boolean,
338                 "filterMode": Integer (see user's manual)
339                 "logical_ids": list of logical ids,
340                 "streaming_commands": list of streaming slots
341         }
342
343     Returns:
344         serial_port: PySerial Object
345
346     """
347
348     # Clean outputs
349     manual_flush(serial_port)
350
```

```
351     # Stop streaming
352     print("Stoping streaming.")
353     stop_streaming(serial_port, configuration_dict['logical_ids'])
354 
355     # Setting streaming slots, this means that while streaming
            sensors will send
356     # this data to the dongle as in page 29 - User manual:
357     # 0 - Differential quaternions;
358     # 1 - tared orientation as euler angles;
359     # 2 - rotation matrix
360     # 255 - No data
361     # See user manual to get more information
362     set_streaming_slots(serial_port,
363                         configuration_dict['logical_ids'],
364                         configuration_dict['streaming_commands'])
365 
366     # Set magnetometer, calibGyro if calibGyro=True and Tare sensor
367     print('Starting configuration: ')
368     configure_sensor(serial_port, configuration_dict)
369     print("Done.")
370 
371     print("Starting streamnig.")
372     # Start streaming
373     start_streaming(serial_port, configuration_dict['logical_ids'])
374 
375     print("Serial Reset")
376 
377     return serial_port
```

B.3 Comunicação UDP: *UdpComms.py*

Código B.3 – Código em Python

```
1  #Communication via UDP protocol from the Python Code to Unity.
2  
3  class UdpComms():
4      def
           __init__(self,udpIP,portTX,portRX,enableRX=False,suppressWarnings=Tru
5          """
6          Constructor
7          :param udpIP: Must be string e.g. "127.0.0.1"
8          :param portTX: integer number e.g. 8000. Port to transmit
9          :param portRX: integer number e.g. 8001. Port to receive
10         """
11         import socket
12 
13         self.udpIP = udpIP
14         self.udpSendPort = portTX
15         self.udpRcvPort = portRX
16         self.enableRX = enableRX
```

```
17          self.suppressWarnings = suppressWarnings # when true
               warnings are suppressed
18          self.isDataReceived = False
19          self.dataRX = None
20
21          # Connect via UDP
22          self.udpSock = socket.socket(socket.AF_INET,
               socket.SOCK_DGRAM) # internet protocol, udp (DGRAM)
               socket
23          self.udpSock.setsockopt(socket.SOL_SOCKET,
               socket.SO_REUSEADDR, 1) # allows the address/port to be
               reused immediately instead of it being stuck in the
               TIME_WAIT state waiting for late packets to arrive.
24          self.udpSock.bind((udpIP, portRX))
25
26          # Create Receiving thread if required
27          if enableRX:
28              import threading
29              self.rxThread =
                   threading.Thread(target=self.ReadUdpThreadFunc,
                   daemon=True)
30              self.rxThread.start()
31
32      def __del__(self):
33          self.CloseSocket()
34
35      def CloseSocket(self):
36          # Function to close socket
37          self.udpSock.close()
38
39      def SendData(self, strToSend):
40          # Use this function to send string to C#
41          self.udpSock.sendto(bytes(strToSend,'utf-8'), (self.udpIP,
               self.udpSendPort))
42
43      def ReceiveData(self):
44          """
45          Should not be called by user
46          Function BLOCKS until data is returned from C#. It then
               attempts to convert it to string and returns on
               successful conversion.
47          An warning/error is raised if:
48              - Warning: Not connected to C# application yet.
                   Warning can be suppressed by setting
                   suppressWarning=True in constructor
49              - Error: If data receiving procedure or conversion to
                   string goes wrong
50              - Error: If user attempts to use this without enabling
                   RX
51          :return: returns None on failure or the received string on
               success
52          """
```

```
53          if not self.enableRX: # if RX is not enabled, raise error
54              raise ValueError("Attempting to receive data without
                    enabling this setting. Ensure this is enabled from
                    the constructor")
55
56          data = None
57          try:
58              data, _ = self.udpSock.recvfrom(1024)
59              data = data.decode('utf-8')
60          except WindowsError as e:
61              if e.winerror == 10054: # An error occurs if you try
                    to receive before connecting to other application
62                  if not self.suppressWarnings:
63                      print("Are You connected to the other
                            application? Connect to it!")
64                  else:
65                      pass
66              else:
67                  raise ValueError("Unexpected Error. Are you sure
                        that the received data can be converted to a
                        string")
68
69          return data
70
71      def ReadUdpThreadFunc(self): # Should be called from thread
72          """
73          This function should be called from a thread [Done
                automatically via constructor]
74                  (import threading -> e.g. udpReceiveThread =
                        threading.Thread(target=self.ReadUdpNonBlocking,
                        daemon=True))
75          This function keeps looping through the BLOCKING
                ReceiveData function and sets self.dataRX when data is
                received and sets received flag
76          This function runs in the background and updates class
                variables to read data later
77
78          """
79
80          self.isDataReceived = False # Initially nothing received
81
82          while True:
83              data = self.ReceiveData()  # Blocks (in thread) until
                    data is returned (OR MAYBE UNTIL SOME TIMEOUT AS
                    WELL)
84              self.dataRX = data # Populate AFTER new data is
                    received
85              self.isDataReceived = True
86              # When it reaches here, data received is available
87
88      def ReadReceivedData(self):
89          """
```

```
90          This is the function that should be used to read received
               data
91          Checks if data has been received SINCE LAST CALL, if so it
               returns the received string and sets flag to False (to
               avoid re-reading received data)
92          data is None if nothing has been received
93          :return:
94          """
95
96          data = None
97
98          if self.isDataReceived: # if data has been received
99              self.isDataReceived = False
100             data = self.dataRX
101             self.dataRX = None # Empty receive buffer
102
103         return data
```

B.4 Regressão linear: *vel_graph.py*

Código B.4 – Código em Python

```
1  from sklearn.linear_model import LinearRegression
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  # Set the x and y values for your data set
6  x = np.array([0.29, 0.38, 0.51,  0.60,  0.67,   0.78,
       0.85]).reshape(-1,1)
7  y = np.array([   1,  1.5,    2,   2.5,     3,    3.5,     4])
8
9  # Create a LinearRegression object
10 reg = LinearRegression()
11
12 # Fit the model to the data
13 reg.fit(x, y)
14
15 # Print the coefficients
16 print("Coefficients:", reg.coef_)
17 print("Intercept:", reg.intercept_)
18
19 # Predict the y values for the x values
20 y_pred = reg.predict(x)
21
22 # Plot the data with the regression line
23 plt.scatter(x, y, color = "m", marker = "o", s = 30)
24 plt.plot(x, y_pred, color = "g")
25 plt.ylabel('Velocidade Real (km/h)')
26 plt.xlabel('Frequ ncia (ciclo/seg)')
27 plt.show()
```

B.5 Comunicação e processamento dos dados: *UdpEuler.cs*

Código B.5 – Código em C Sharp

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;

public class UdpEuler : MonoBehaviour
{
    [HideInInspector] public bool isTxStarted = false;

    [SerializeField] string IP = "127.0.0.1"; // local host
    int rxPort = 8000; // port to receive data from Python on
    int txPort = 8001; // port to send data to Python on

    // Create necessary UdpClient objects
    UdpClient client;
    IPEndPoint remoteEndPoint;
    Thread receiveThread; // Receiving Thread

    //String Received
    //public static string[] textArray;
    public float  y_data;
    [SerializeField] private int    steps = 0;
    [SerializeField] private float  timer = 0.0f;
    [SerializeField] private bool   ascending = true;
    [SerializeField] private float  media = 0;

    public  float   velocity = 0.0f;
    public  float   velocityMedia = 0.0f;
    private float   prevTime = 0.0f;
    private float   sum  = 0.0f;

    private float   minY, maxY = 0.0f;
    List<float> velocityList = new List<float>();

    void Awake()
    {
        // Create remote endpoint
        remoteEndPoint = new IPEndPoint(IPAddress.Parse(IP),
            txPort);

        // Create local client
        client = new UdpClient(rxPort);
```

```
46
47          // local endpoint define (where messages are received)
48          // Create a new thread for reception of incoming messages
49          receiveThread = new Thread(new ThreadStart(ReceiveData));
50          receiveThread.IsBackground = true;
51          receiveThread.Start();
52
53          // Initialize (seen in comments window)
54          print("UDP Comms Initialised");
55
56
57      }
58
59      private void Update()
60      {
61          timer += Time.deltaTime;
62
63          if(y_data < minY) // Calculating Min and Max values
64          {
65              minY = y_data;
66          }
67          if(y_data > maxY)
68          {
69              maxY = y_data;
70          }
71          media = (minY + maxY) / 2; // Calculating Media
72
73
74          if (y_data > media && ascending == false)   // Calculating
               number of steps
75          {
76              ascending = true;
77              steps++;
78              velocity = SetVelocity(timer, prevTime);
79              velocityList.Add(velocity);     // Velocity list to
                   calculate median
80
81              if (velocityList.Count == 10)
82              {
83                  foreach (float x in velocityList)
84                  {
85                      sum = sum + x;
86                  }
87                  velocityMedia = sum / 10;
88                  sum = 0;
89                  velocityList.RemoveAt(0);
90              }
91              prevTime = timer;
92          }
93
94          if (y_data < media && ascending == true)
95          {
```

```
96              ascending = false;
97              steps++;
98          }
99
100
101         // Sending data starts on Enter.
102         if (Input.GetKey(KeyCode.KeypadEnter))
103         {
104             //timer = 0f;
105             Debug.Log("Timer Started");
106             StartCoroutine(SendDataCoroutine()); // Added to show
                    sending data from Unity to Python via UDP
107         }
108     }
109
110     IEnumerator SendDataCoroutine() //  Added to show sending data
        from Unity to Python via UDP
111     {
112         while (true)
113         {
114             SendData(timer.ToString("F2") + ":" +
                    velocity.ToString("F3") + ":" +
                    y_data.ToString("F1"));
115             yield return new WaitForSeconds(0.2f);
116         }
117     }
118
119
120     private float SetVelocity(float timer, float prevTime)
121     {
122         // Calculating period between steps
123
124         velocity = (5.27276877f * (1 / (timer - prevTime))) -
                0.5732709427f;
125         //frequency = (1 / (timer - prevTime)) ;
126         //period = (timer - prevTime);
127
128
129         if (velocity < 0.0f)
130         {
131             velocity = 0.0f;
132         }
133         if (velocity > 100.0f)
134         {
135             velocity = 0.0f;
136         }
137
138
139         return velocity;
140     }
141
142
```

```
143     // Receive data, update packets received
144     private void ReceiveData()
145     {
146         while (true)
147         {
148             try
149             {
150                 IPEndPoint anyIP = new IPEndPoint(IPAddress.Any,
                        0);
151                 byte[] data = client.Receive(ref anyIP);
152                 string text = Encoding.UTF8.GetString(data);
153                 y_data = float.Parse(text); // Getting the y value
                        in angles
154                 //Debug.Log(y_data);
155 
156                 ProcessInput(text);
157             }
158             catch (Exception err)
159             {
160                 print(err.ToString());
161             }
162         }
163     }
164 
165     private void ProcessInput(string input)
166     {
167 
168         if (!isTxStarted) // First data arrived so tx started
169         {
170             isTxStarted = true;
171         }
172     }
173 
174 
175     void OnDisable()  //Prevent crashes - close clients and
            threads properly!
176     {
177         if (receiveThread != null)
178             receiveThread.Abort();
179 
180         client.Close();
181     }
182 
183 
184     public void SendData(string message) // Use to send data to
            Python
185     {
186         try
187         {
188             byte[] data = Encoding.UTF8.GetBytes(message);
189             client.Send(data, data.Length, remoteEndPoint);
190         }
```

```
191         catch (Exception err)
192         {
193             print(err.ToString());
194         }
195     }
196
197
198 }
```

B.6 Mapeamento do avatar com as entradas: *VRRig.cs*

Código B.6 – Código em C Sharp

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5
6  [System.Serializable]
7  public class VRMap
8  {
9      public Transform vrTarget;
10     public Transform rigTarget;
11     public Vector3 trackingPositionOffset;
12     public Vector3 trackingRotationOffset;
13
14     public void Map()
15     {
16         rigTarget.position =
               vrTarget.TransformPoint(trackingPositionOffset);
17         rigTarget.rotation = vrTarget.rotation *
               Quaternion.Euler(trackingRotationOffset) ;
18     }
19 }
20
21
22 public class VRRig : MonoBehaviour
23 {
24     public VRMap head;
25     public VRMap leftHand;
26     public VRMap rightHand;
27
28
29     public Transform headConstrain;
30     public Vector3 headBodyOffset;
31
32     void Start()
33     {
34         headBodyOffset = transform.position -
               headConstrain.position;
```

```
35
36      }
37
38      // Update is called once per frame
39      void Update()
40      {
41          transform.position = headConstrain.position +
                headBodyOffset;
42          transform.forward =
                Vector3.ProjectOnPlane(headConstrain.up,
                Vector3.up).normalized;
43
44          head.Map();
45          leftHand.Map();
46          rightHand.Map();
47      }
48  }
```

B.7 Representação por animações: *AnimationStateController.cs*

Código B.7 – Código em C Sharp

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class AnimationStateController : MonoBehaviour
6  {
7      Animator animator;
8      int isWalkingHash;
9
10     void Start()
11     {
12         animator = GetComponent<Animator>();
13         isWalkingHash = Animator.StringToHash("isWalking");
14     }
15
16     // Update is called once per frame
17     void Update()
18     {
19         bool isWalking = animator.GetBool(isWalkingHash);
20         bool isRunning = animator.GetBool("isRunning");
21         bool forwardPressed = Input.GetKey("w");
22         bool runPressed = Input.GetKey(KeyCode.LeftShift);
23
24         if (!isWalking && forwardPressed)
25         {
26             animator.SetBool(isWalkingHash, true);
```

```
27          }
28
29          if (isWalking && !forwardPressed)
30          {
31              animator.SetBool(isWalkingHash, false);
32          }
33
34          if (!isRunning && (forwardPressed && runPressed))
35          {
36              animator.SetBool("isRunning", true);
37          }
38
39          if (isRunning && (!forwardPressed || !runPressed))
40          {
41              animator.SetBool("isRunning", false);
42          }
43      }
44  }
```

B.8 Representação por quatérnions: *LegsRotation.cs*

Código B.8 – Código em C Sharp

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class LegsRotation : MonoBehaviour
6   {
7       // Get the transform of each parts of the legs.
8       public Transform r_legTransf;
9       public Transform r_kneeTransf;
10      public Transform l_legTransf;
11      public Transform l_kneeTransf;
12
13      // Acess the string from another script
14
15      UdpSocket udpSocket;
16      [SerializeField] GameObject udp;
17      public string[] s_text;
18
19      Quaternion rLeg_StartQuat;
20      Quaternion rKnee_StartQuat;
21      Quaternion lLeg_StartQuat;
22      Quaternion lKnee_StartQuat;
23
24      Quaternion quat_RLeg_Offset;
25      Quaternion quat_RKnee_Offset;
26      Quaternion quat_LLeg_Offset;
27      Quaternion quat_LKnee_Offset;
```

```
28
29      public Quaternion q_RL ;
30      public Quaternion q_RK ;
31      public Quaternion q_LL ;
32      public Quaternion q_LK ;
33
34
35      public Quaternion quaternionRLegOffset;
36
37
38      Quaternion rotQuatZ = Quaternion.Euler(0, 0, 90);
39
40
41      void Awake()
42      {
43          // Create a udp component to grab the data from de
               UdpSocket script
44          udpSocket = udp.GetComponent<UdpSocket>();
45
46          // Grab the initial orientation from the legs
47          rLeg_StartQuat = r_legTransf.rotation;
48          rKnee_StartQuat = r_kneeTransf.rotation;
49          lLeg_StartQuat = l_legTransf.rotation;
50          lKnee_StartQuat = l_kneeTransf.rotation;
51
52
53          Debug.Log(s_text);
54      }
55
56      // Update is called once per frame
57      void Update()
58      {
59          s_text = udpSocket.str_text.Split(':');
60
61
62          // Compare the IMU number to each part of the leg.
63          // 1 = RLeg, 2 = RKnee, 3 = LLeg, 4 = LKnee;
64
65          if (Input.GetKey(KeyCode.Space))
66          {
67              if (int.Parse(s_text[0]) == 1)
68              {
69                  quat_RLeg_Offset =
                       Quaternion.Inverse(StringToQuaternion(s_text[1]))
                       * rLeg_StartQuat;
70              }
71
72              if (int.Parse(s_text[0]) == 2)
73              {
74                  quat_RKnee_Offset =
                       Quaternion.Inverse(StringToQuaternion(s_text[1]))
                       * rKnee_StartQuat;
```

```
75              }
76
77              if (int.Parse(s_text[0]) == 4)
78              {
79                  quat_LLeg_Offset =
                       Quaternion.Inverse(StringToQuaternion(s_text[1]))
                       * lLeg_StartQuat;
80              }
81
82              if (int.Parse(s_text[0]) == 5)
83              {
84                  quat_LKnee_Offset =
                       Quaternion.Inverse(StringToQuaternion(s_text[1]))
                       * lKnee_StartQuat;
85              }
86
87          }
88
89          if (Input.GetKey(KeyCode.C))
90          {
91              //Send data to python to tare sensor
92              udpSocket.SendData("c");
93          }
94
95
96          if (int.Parse(s_text[0]) == 1)
97          {
98              Quaternion current = r_legTransf.localRotation;
99
100             q_RL = StringToQuaternion(s_text[1]);
101
102             r_legTransf.rotation = Quaternion.Slerp(current, q_RL
                    * quat_RLeg_Offset, Time.deltaTime) ;  //Right Leg
103
104
105         }
106
107         if (int.Parse(s_text[0]) == 2)
108         {
109             Quaternion current = r_kneeTransf.localRotation;
110
111             r_kneeTransf.localRotation = Quaternion.Slerp(current,
                    StringToQuaternion(s_text[1]) * quat_RKnee_Offset,
                    Time.deltaTime);  //Right Knee
112
113         }
114
115         if (int.Parse(s_text[0]) == 4)
116         {
117             l_legTransf.rotation = StringToQuaternion(s_text[1]) *
                    quat_LLeg_Offset;   //Left Leg
118         }
```

```
119
120         if (int.Parse(s_text[0]) == 5)
121         {
122             l_kneeTransf.rotation = StringToQuaternion(s_text[1])
                    * quat_LKnee_Offset;  //Left Knee
123         }
124
125
126     }
127
128
129     public static Quaternion StringToQuaternion(string sQuaternion)
130     {
131         // Split the items
132         string[] sArray = sQuaternion.Split(',');
133         // Store as a Quaternion
134         Quaternion result = new Quaternion(
                float.Parse(sArray[0]), float.Parse(sArray[1]),
135                                                 float.Parse(sArray[2]),
                                                    float.Parse(sArray[3]));
136
137         return result;
138     }
139
140 }
```

B.9 Representação por ângulos de Euler: *MoveLegs.cs*

Código B.9 – Código em C Sharp

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class MoveLegs : MonoBehaviour
6  {
7      public UdpEuler udp;
8      public GameObject leftLeg;
9      public GameObject rightLeg;
10
11     Vector3 leftStartRotation;
12     Vector3 rightStartRotation;
13
14     void Start()
15     {
16         leftStartRotation = leftLeg.transform.eulerAngles;
17         rightStartRotation = rightLeg.transform.eulerAngles;
18     }
19
20     // Update is called once per frame
```

```
21      void Update()
22      {
23          //Debug.Log(udp.y_data);
24
25          leftLeg.transform.eulerAngles = leftStartRotation + new
                Vector3(0, 0, -udp.y_data);
26          rightLeg.transform.eulerAngles = rightStartRotation + new
                Vector3(0, 0, -udp.y_data);
27
28          //Debug.Log(leftLeg.transform.eulerAngles);
29
30
31      }
32  }
```