Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Classificação de poses e movimentos do boxe para interação com jogos eletrônicos

Autor: Ariel Vieira Lima Serafim
Orientador: Professor Doutor Renato Coral Sampaio

Brasília, DF

2024

Ariel Vieira Lima Serafim

Classificação de poses e movimentos do boxe para interação com jogos eletrônicos

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Professor Doutor Renato Coral Sampaio

Brasília, DF

2024

Este trabalho é dedicado a todos aqueles que almejam expandir os limites do conhecimento humano.

Agradecimentos

Agradeço aos meus pais, Adilson Serafim de Oliveira e Maria Emília Vieira Lima Serafim, que possuem suas missões na Terra ligadas à minha pela Lei Divina.

Resumo

Este projeto busca incentivar a prática de saúde e esporte por meio da criação de um sistema que identifica poses e técnicas do boxe para interagir com videogames. Para classificar poses e técnicas, foi utilizada uma unidade de medida inercial, que captura ângulos em 3 dimensões e publicá-os no formato de quatérnios. O sistema operacional robótico versão 2, Ros2, foi utilizado para construir uma arquitetura de microsserviços e integrar o módulo de telemetria com o módulo de classificação, que utiliza K vizinhos mais próximos e classificador centróide como estratégias de classificação.

Palavras-chaves:Knn. classificador. ros2. IMU, boxe, jogos

Abstract

This project seeks to incentivize health and sports practice by creating a system that identifies boxing poses and techniques to interact with video games. To classify poses and techniques, an Inertial measurement unit was used to capture angles in 3 dimensions and publish it as a quaternion. The Robotic operation system version 2, Ros2, was used to build a microservice architecture and integrate the telemetry module and the classifier module, that uses K nearest neighbors and centroid classifier.

Key-words: Knn. classifier. ros2. IMU, boxing, games

1 Introdução

Cerca de 40.3% dos brasileiros acima de 18 anos são sedentários (IBGE, 2019), é o que indica a Pesquisa Nacional de Saúde, PNS, divulgados pelo Instituto Brasileiro de Geografia e Estatística, IBGE. São considerados sedentários aqueles que realizam atividades físicas por menos de 150 minutos por semana, já com contabilizado deslocamento, lazer e trabalho. Sedentarismo é um poderoso agravante e potencializador de problemas de saúde.

Os riscos provenientes do sedentarismo estão ligados à um estado de desequilíbrio geral do organismo (INNOVATION2YOU, 2021), isto é, os mecanismos sobre os quais o corpo humano opera são perturbados na ausência de uma vida ativa. Temos perda de massa muscular, ganho de gordura, diminuição de capacidade respiratória, alteração hormonal, e, para cada fator citado, uma série de problemas de saúde consequentes.

A prevenção padrão do sedentarismo é, naturalmente, praticar atividade física (MACEDO et al., 2003), comumente, esportes. Para o praticante de atividade física, é interessante participar de um esporte formal, aqui me refiro à qualquer modalidade esportiva organizada (TIPOS. . . , 2021), pois, em comparação com uma atividade não organizada, uma brincadeira de criança por exemplo, temos com o esporte formal um conjunto de estudos e orientações que, quando devidamente seguidas, garantem a segurança do praticante.

O esporte boxe é uma dessas modalidades muito bem organizadas e largamente estudadas (BRITANNICA, 2022), que podemos usar como base para escolha dos movimentos e poses que serão identificados, pois usar esse esporte garante que o usuário dessa aplicação não realizará movimentos que possa levar a uma lesão. O boxe foi escolhido pela compatibilidade de seus movimentos com os movimentos vistos em jogos eletrônicos.

Será apresentado nesse trabalho, técnicas que buscam promover uma atividade física, inspirada nos movimentos do boxe, com aplicação lúdica voltada para jogos eletrônicos. A importância de uma aplicação lúdica é justamente o pri-

Tabela 1 – Movimentos básicos do boxe

Movimento	**Nome Inglês**	**Tipo**	**Descrição**
Jab	Jab	Golpe	Golpe reto desferido com a mão frontal
Cruzado	Cross	Golpe	Golpe cruzado desferido horizontalmente
Gancho	Uppercut	Golpe	Golpe cruzado desferido verticalmente, de baixo para cima.
Esquiva	Slip	Desvio	Movimento diagonal dos ombros e cabeça, de cima para baixo.
Pêndulo	Duck	Desvio	movimento circular realizado com o corpo todo, em que a cabeça caminha formando um 'u'.
Bloqueio	block	Defesa	Defesa da cabeça realizada com os braços.

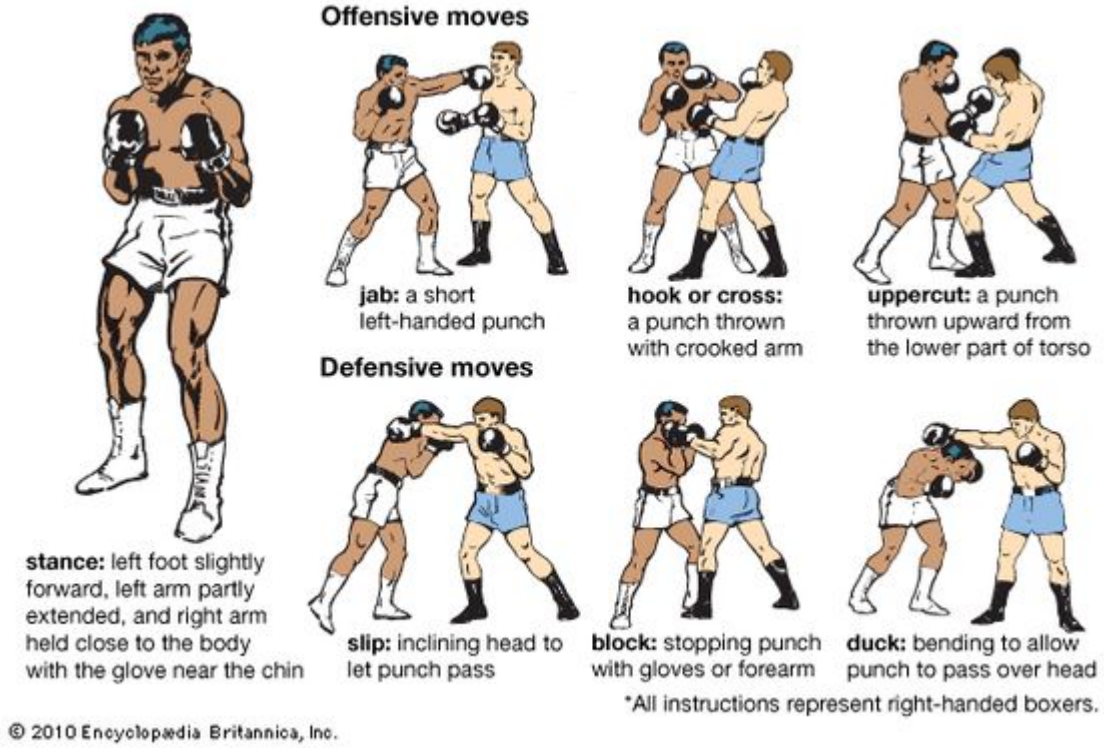
meiro dado apresentado nessa introdução, a quantidade de pessoas sedentárias no Brasil, bem como as consequências disso, implementar um elemento lúdico à atividade física é uma forma de, incentivar a criação de habito dessa importantíssima prevenção de problemas de saúde.

1.1 Descrição do problema

Vimos que o sedentarismo é comum no brasil, e que pode ser a causa e/ou agravante de diversas condições médicas, vimos também que a solução direta para essa condição é a prática de atividade física, com destaque para esportes. Então, de forma simples e direta, o problema central é o desinteresse em praticar esportes, e com isso buscamos uma forma de criar um incentivo lúdico para a prática desportiva, a partir do mapeamento dos movimentos básicos do boxe para interação com jogos eletrônicos.

Considere o conjunto de movimentos da Figura 1, estes são os movimentos básicos do boxe, detalhados na 1 . Há também variações de lado do movimento, então por exemplo, um golpe cruzado pode ser realizado com o braço direito e esquerdo, bem como os outros golpes e defesas podem ser realizadas com ambos os braços, variações de angulo são relevantes em uma análise mais profunda do esporte boxe, mas para aplicação desse trabalho, consideraremos um conjunto básico de 6 movimentos, realizados a partir de uma base padrão, vista na figura a seguir, com pés e tronco na diagonal e braços levantados, próximos ao rosto.

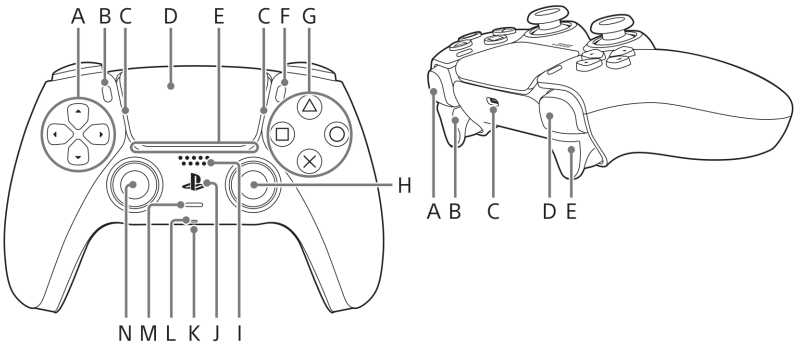
Figura 1 – Movimentos básicos do boxe



Fonte: autor

Vamos então entender o contexto de interação com jogos eletrônicos. Podemos usar como exemplo o controle DualSense (PLAYSTATION, 2020), da Sony PlayStation. O controle possui 16 botões, e consequentemente todos os jogos disponíveis para a plataforma da Sony são construídos para funcionar de forma plena com essa interface. A ação equivalente ao acionamento de um botão ou conjunto de botões é dependente do contexto no jogo. Vemos na Figura 2 a vista frontal e superior do DualSense, usamos esse controle como exemplo, mas controles de outras plataformas, como o do Xbox Microsoft e controles dedicados para jogos disponíveis para computador, possuem estrutura e número de botoes semelhante.

Figura 2 – Controle DualSense



Fonte: (PLAYSTATION, 2020)

1.2 Objetivos

1.2.1 Objetivo Geral

Uma vez que conhecemos a importância da pratica desportiva, os movimentos básicos do boxe e o contexto de interação com jogos eletrônicos, o objetivo desse trabalho é Identificar em tempo-real por meio de sensores inerciais, com técnicas de fusão sensorial, IMU(inertial measurement unit),unidade de medição inercial, em software, os movimentos básicos do boxe e mapeá-los para possível interação com jogos eletrônicos.

1.2.2 Objetivos Específicos

- Obter dados de telemetria de uma IMU.
- Estimar a posição angular dos braços de uma pessoa.
- Identificar poses e movimentos de uma pessoa, utilizando ângulos, aceleração linear e velocidade angular.
- Mapear as poses e movimentos identificados em comandos, possíveis de serem usados para interagir com um jogo eletrônico.

2 Fundamentação Teórica

2.1 Identificação de poses e movimentos

Nesse trabalho será proposto uma técnica de identificação de poses humanas, com aplicação em jogos eletrônicos, aplicação que, por sua vez, impõe um requisito de tempo para a completa identificação de uma determinada pose, já que para uma interação com um jogo ser considerada responsiva, é necessário computar o comando do jogador em uma curta janela de tempo, e também, garante a não necessidade de certo detalhamento na identificação das poses, pois os jogos atuais são construídos de forma a funcionar para controles com cerca de 15 botões distintos (PLAYSTATION, 2020), (XBOX, 2020). Dada essa análise inicial, estudou-se a possibilidade de uso de 2 tecnologias para identificação de poses e movimentos: Sensores inerciais e Visão computacional.

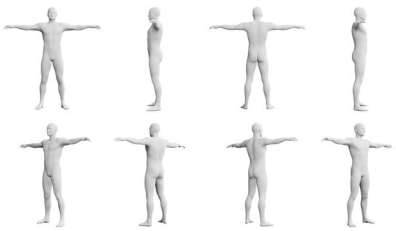
Um software de modelagem 3D permite a criação de uma representação matemática de um objeto ou forma tridimensional em computador. O objeto criado é chamado de modelo 3D, e esses modelos tridimensionais são usados em vários setores. (AUTODESK, 2023)

Em modelagem 3D, há o conceito de T pose, ou pose T, em que o personagem apresenta a posição de pernas esticadas,tronco ereto e braços esticados lateralmente (AUTODESK, 2022). Podemos pensar nessa pose como um ponto de repouso, ou ponto inicial, e usar técnicas de visão computacional e/ou medição inercial para identificar outras poses que uma pessoa possa assumir. A seguir vamos explorar as técnicas citadas, e discorrer sobre a escolha de tecnologia escolhida.

2.1.1 Visão computacional

Visão computacional é uma área multidisciplinar da ciência e engenharia, que busca extrair informações a partir de imagens digitais. Uma aplicação comercial que usa essa tecnologia para identificação de poses é o Kinect (CORPORATION, 2014) ,produto da Microsoft que usa 3 câmeras distintas para identificar

Figura 3 – (a) pessoa com traje de captura de movimento na pose 'T', (b) Rigging equivalente à pose em 'a', (c) personagem 3d em posição de acordo com a captura feita.



Fonte: (STOCK, 2020)

movimentos do jogador e mapear esses movimentos para diversos jogos.

Captura de movimentos realizada por um conjunto de câmeras em posições estratégicas é o estado da arte na área de identificação de poses e movimento, muito utilizado na área de animação 3d e criação de jogos eletrônicos, entretanto, é necessário também o uso de trajes com marcações de alto contraste para a adequada identificação das poses, o que dificulta a implementação dessa tecnologia para uso do consumidor final e, consequentemente, impossibilita que este trabalho seja desenvolvido com o uso dessa técnica.

A Figura 3 mostra a pose 'T', citada anteriormente, à esquerda, uma pessoa com traje de captura de movimento,ao centro, o resultado do mapeamento dessa posição em 3D, mais especificamente o resultado em *rigging*, um processo em que se define uma estrutura análoga ao esqueleto (ADOBE, 2023), com hierarquia de controle das 'articulações' do personagem 3D, e à direita temos o resultado disso no personagem 3D.

2.1.2 Sensores Inerciais

Sensores inerciais identificam forças aplicadas sobre um corpo móvel em repouso, a partir desse valor inicial, podemos obter medidas de aceleração e/ou velocidade do sistema em questão(KEMPE, 2011).

Os giroscópios triaxiais informam o valor da aceleração angular em 3 eixos, ao integrar os valores de velocidade angular, temos uma estimativa da rotação em

relação a um ponto inicial. Os acelerômetros triaxiais informam a aceleração linear aplicada em 3 eixos, que podem ser relacionadas para estimar o ângulo em 3 eixos em relação a um estado inercial acordado (KEMPE, 2011).

Devido à praticidade e versatilidade dos sensores inerciais atuais, optou-se por prosseguir os trabalhos com o uso desses sensores, que, como veremos a seguir, complementam-se para construir uma medição mais acurada.

2.2 IMU '3 Space Sensor'

Usar o filtro de Kalman é então, uma excelente opção para para se obter dados de telemetria, porém, como explorado na seção anterior, a técnica possui sua complexidade teórica e prática, por ser vastamente usada comercialmente, existem opções comerciais acessíveis de componentes com filtro de Kalman embutido, para esse projeto foi disponibilizado a IMU '3 Space Sensor' (LABS, 2017).

"O 3-Space Sensor™ Wireless integra um Sistema de Referência de Altitude e Inclinação (AHRS) em miniatura, de alta precisão e alta confiabilidade, com uma interface de comunicação DSSS de 2,4 GHz e uma solução de bateria de polímero de lítio recarregável em uma única unidade pronta para uso de baixo custo. O Sistema de Referência de Altitude e Inclinação (AHRS) usa sensores triaxiais de giroscópio, acelerômetro e bússola em conjunto com algoritmos avançados de filtragem e processamento embarcados para determinar a orientação em relação a uma referência absoluta em tempo real"

O parágrafo anterior é uma tradução livre do parágrafo inicial da introdução do documento oficial de documentação da IMU usada nesse projeto, a '3 Space Sensor', da Yost Labs.

2.3 Protocolo de Comunicação da IMU

O sensor 3-Space recebe mensagens do sistema controlador na forma de sequências de bytes de comunicação serial chamadas pacotes. Para facilitar o uso e a flexibilidade de operação, são fornecidos dois métodos de codificação de comandos: binário e texto. A codificação binária é mais compacta, mais eficiente e mais fácil de

acessar programaticamente. A codificação em texto ASCII é mais verbosa e menos eficiente, porém é mais fácil de ler e acessar através de uma interface de terminal tradicional. Ambos os métodos de codificação compartilham uma estrutura de comando idêntica e suportam todo o conjunto de comandos do sensor 3-Space.

O sensor 3-Space armazena em buffer o fluxo de comandos recebidos e só executa uma ação quando o pacote inteiro foi recebido e o checksum foi verificado como correto (os comandos em modo ASCII não utilizam checksums por conveniência). Pacotes incompletos e pacotes com checksums incorretos serão ignorados. Isso permite que o sistema controlador envie dados de comando de forma contínua sem perda de funcionalidade. No entanto, o buffer de comandos será limpo sempre que o sensor 3-Space for reiniciado ou ligado/desligado.

2.3.1 Formato do Pacote Binário

O tamanho do pacote binário pode ter três ou mais bytes de comprimento, dependendo da natureza do comando sendo enviado ao controlador. Cada pacote consiste em um byte inicial de "início do pacote", seguido por um byte especificador de "valor do comando", seguido por zero ou mais bytes de "dados do comando", e terminado por um byte de "valor do checksum do pacote".

A estrutura geral de um pacote de comando binário é a seguinte:

- Byte de início do pacote
- Byte de valor do comando
- Zero ou mais bytes de dados do comando
- Byte de valor do checksum

2.3.2 Comunicação Sem Fio

Para se comunicar com um sensor através de um computador sem fio, o dongle 3-Space deve estar conectado ao computador via USB. O dongle se apresenta como uma porta COM, assim como o sensor 3-Space. Cada dongle pode ser associado a até 15 unidades de sensor sem fio. Para associar uma unidade de sensor

a um dongle, o usuário deve colocar o número de série do sensor desejado em um dos 15 slots lógicos da tabela de wireless do dongle. Qualquer sensor 3-Space sem fio ao alcance que tenha um endereço atribuído em um slot do dongle pode então ser comunicado usando o dongle.

A comunicação sem fio suporta os mesmos comandos que a comunicação com fio, mas não são idênticos. Isso permite que o protocolo sem fio inclua características específicas da natureza da comunicação sem fio, como endereçamento, confiabilidade e tratamento de perda de pacotes.

2.3.2.1 Formato do Pacote Binário Sem Fio

O formato do pacote binário sem fio é similar ao do com fio, com a adição de campos específicos para endereçamento e controle de confiabilidade. Cada pacote consiste em:

- Byte de início do pacote
- Byte de valor do comando
- Zero ou mais bytes de dados do comando
- Byte de valor do checksum
- Bytes adicionais para endereçamento e controle de confiabilidade

2.3.2.2 Resposta a Comandos Binários

As respostas a comandos binários seguem uma estrutura similar, permitindo que o sistema controlador verifique a integridade e o sucesso da execução dos comandos enviados. As respostas incluem:

- Byte de início do pacote de resposta
- Byte de valor de resposta
- Zero ou mais bytes de dados de resposta
- Byte de valor do checksum

2.3.2.3 Comandos Binários de Amostra

Exemplos de comandos binários incluem solicitações para leitura de dados de orientação, calibração do sensor e configuração de parâmetros de comunicação. Cada comando é composto por uma sequência específica de bytes conforme definido na documentação do protocolo.

2.3.3 Cabeçalho de Resposta com Fio

O cabeçalho de resposta com fio inclui campos para verificar a integridade e a sequência dos dados recebidos, assegurando que o sistema controlador possa validar cada resposta.

2.4 Representação de Ângulos

Os ângulos de Euler, introduzidos por Leonhard Euler, servem como uma ferramenta popular para representar rotações no espaço tridimensional. Através de três rotações consecutivas em torno de eixos específicos, comumente chamados de yaw, pitch e roll, eles oferecem uma maneira intuitiva e compreensível de descrever a orientação de um objeto.

No entanto, apesar da sua praticidade, os ângulos de Euler apresentam algumas desvantagens importantes que podem afetar a precisão e confiabilidade dos cálculos. Uma delas é a singularidade de Gimbal, também conhecida como bloqueio de Gimbal. Imagine um giroscópio: quando dois dos seus eixos de rotação se alinham perfeitamente, ocorre a singularidade, resultando na perda de um grau de liberdade e na impossibilidade de representar determinadas orientações. No sistema cartesiano XYZ, por exemplo, isso acontece quando o ângulo de pitch atinge $\pm 90°$.

Outro problema reside na acumulação de erros durante cálculos sucessivos de rotações utilizando ângulos de Euler. Em aplicações que exigem alta precisão, como simulações de voo ou animações 3D, esses erros podem se manifestar como distorções ou desvios na orientação final do objeto, comprometendo a qualidade do resultado final.

2.4.1 Quaterniões

Os quaterniões são frequentemente utilizados para superar as limitações dos ângulos de Euler. São uma extensão dos números complexos e fornecem uma maneira robusta e eficiente de representar rotações no espaço tridimensional.

Um quaternião pode ser representado na forma:

$$q = w + xi + yj + zk \tag{2.1}$$

onde w, x, y, z são números reais e i, j, k são unidades imaginárias que obedecem às seguintes regras de multiplicação:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.2}$$

$$ij = k, \quad ji = -k \tag{2.3}$$

$$jk = i, \quad kj = -i \tag{2.4}$$

$$ki = j, \quad ik = -j \tag{2.5}$$

Os quaterniões não sofrem de singularidade de Gimbal e evitam a acumulação de erros de arredondamento que são comuns em representações de ângulos de Euler.Enquanto os ângulos de Euler fornecem uma maneira simples e intuitiva de representar rotações, eles apresentam limitações que podem ser problemáticas em muitas aplicações práticas. Os quaterniões oferecem uma alternativa poderosa, superando esses problemas com uma representação matemática robusta e eficiente para rotações no espaço tridimensional.

2.5 Teoria da Classificação

Para o ser humano é natural reconhecer padrões complexos a partir de dados brutos simples, reconhecemos rostos, músicas, o estado de amadurecimento

de uma fruta, tudo isso usando características simples, como formas, cores e sons. Em classificação, buscamos entender como replicar o processo de identificação de características e classificação de padrões a partir delas.

2.5.1 Os Sub-problemas da Classificação de Padrões

Diversos sub-problemas são cruciais na classificação de padrões. Entre eles estão a extração de características, o tratamento de ruídos, o sobreajuste, a seleção de modelos, o uso de conhecimento prévio, a gestão de características ausentes, a mereologia, a segmentação e o contexto.

2.5.1.1 Característica

Extração de característica é o processo de tomar um padrão e produzir valores de características. O número de características é geralmente escolhido para ser menor do que o necessário para descrever completamente o alvo de interesse, o que leva a uma perda de informação. Contudo, a extração de características visa preservar todas as informações relevantes para a tarefa em questão. A extração de características é fundamental para reduzir a informação bruta em um conjunto de dados que seja manejável e significativo para a classificação.

2.5.1.2 Mereologia

A mereologia é o estudo das relações de parte/todo. Um exemplo clássico é a leitura da palavra "BEATS". Por que não lemos palavras que são subconjuntos perfeitamente válidos do padrão completo, como "BE", "BEAT", "EAT", "AT"e "EATS"? Ou quando vimos o "B", por que não lemos um "P"ou um "I", que estão "presentes"dentro do "B"? Esse é o problema dos subconjuntos e superconjuntos. A mereologia está intimamente relacionada ao conhecimento prévio e à segmentação. Em suma, como reconhecemos ou agrupamos o número "correto"de elementos – nem muito poucos, nem muitos? Parece que os melhores classificadores tentam incorporar o máximo de entrada na categorização que "faz sentido", mas não em excesso.

2.5.1.3 Segmentação

A segmentação é um dos problemas mais profundos no reconhecimento automático de fala e outras aplicações. Considere o exemplo dos peixes na esteira rolante. Muitas vezes, eles estariam sobrepostos ou tocando-se, e nosso sistema precisaria determinar onde um peixe termina e o próximo começa. Este problema é semelhante na fala, onde precisamos segmentar sons individuais (como fonemas) e depois combiná-los para determinar a palavra. A segmentação adequada é crucial para uma classificação correta dos padrões.

2.5.2 Algoritmos usados

Para esse projeto foram usados dois algoritmos de classificação. K vizinhos mais próximos(KNN) e classificador de centroide. O KNN consiste em classificar um ponto, que é definido a partir do conjunto de caractericas, a partir dos K pontos com menor distancia a esse ponto: Considere um conjunto de 2 grupos com classificação conhecida, digamos grupo A e grupo B, considere também um ponto x, calculamos a distância euclidiana do ponto x para cada um dos membros dos grupos A e B, o resultado é um conjunto de N distâncias, em que N é o numero de pontos em A + o numero de pontos em B. Ao ordenar essas N distâncias, o valor K, é o numero de 'votos' que serão considerados para classificar x, o grupo com mais votos dentre as K primeiras distâncias ordenadas, é o o grupo para qual x sera classificado.

Para o classificador de centroide considere o mesmo cenário, dois grupos e um ponto que se deseja classificar. Essa técnica consiste em calcular o centroide de cada um dos grupos, e então calcular a distancia do ponto x a cada um destes centroides. A menor distância é usada para classificar o ponto.

3 Desenvolvimento

Esse projeto foi feito do ponto de vista da engenharia de software, de maneira geral, considerou-se 5 grandes etapas do processo de desenvolvimento de um produto de software, sendo elas:

- Processo de desenvolvimento de software
- Requisitos de software
- Design e Arquitetura
- Desenvolvimento
- Testes

3.1 Processo de desenvolvimento de software

É mais comum se falar em processo quando se trabalha em equipe; Nesse projeto tive o apoio de meu orientador, professores da banca de avaliação, e pesquisadores do Laboratório de Automação e Robótica(Lara) da Universidade de Brasília(UnB), porém, o desenvolvimento em si foi feito por uma pessoa. Mesmo assim, optou-se por adaptar princípios de Kanban e Scrum para gerenciar o fluxo de trabalho, priorizar tarefas e garantir entregas regulares.

A natureza experimental do projeto demandou a utilização de uma estratégia de prova de conceito (PoC) para validar ideias e conceitos. Através da criação de versões mínimas viáveis e testes iterativos, foi possível identificar falhas e oportunidades de melhorias desde as etapas iniciais do desenvolvimento. As PoCs, documentadas no diretório "concepts"do repositório do projeto, serviram como base para a construção da versão final do sistema.

3.2 Requisitos de software

"Os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que deve oferece e as restrições a seu funcionamento."
(SOMMERVILLE, 2011)

O ponto originário de todo e qualquer requisito desse projeto, é que se quer classificar movimentos e poses do boxe e usar esse resultado como um controle de jogos eletrônicos.

Ao entender os detalhes desse objetivo, os requisitos do projeto foram elicitados, listados a seguir, em requisitos funcionais (RF) e requisitos não funcionais (RNF).

- RF1: O Sistema deve identificar poses do boxe.
- RF2: O Sistema deve identificar movimentos do boxe.
- RF3: O Sistema deve interpretar sequências de poses e movimentos do boxe como comandos.
- RNF1: A classificação de uma pose deve ser feita a cada amostra.
- RNF2: A classificação de um movimento deve ser feita em tempo menor ou igual ao tempo médio de reação humana.
- RNF3: A captura de dados de telemetria deve ser feita de forma confortável.
- RNF4: O envio de dados de telemetria deve ser feito de forma sem fio.

O conjunto de requisitos funcionais desse projeto, é a própria premissa, o próprio objetivo desse trabalho, é simplesmente um forma de definir a proposta desse projeto em termos de requisitos.

O que se pode detalhar sobre o conjunto de requisitos funcionais, é definir o que se entende por pose, movimentos e 'interpretar como comandos'.

A pose, no contexto de estudos do movimento humano e esportes, refere-se a uma posição estática adotada pelo corpo. É uma configuração específica do corpo

em um determinado instante, onde todas as partes do corpo estão posicionadas de uma forma particular.

> "Uma pose pode ser entendida como uma configuração momentânea e estática do corpo, frequentemente analisada para entender a postura e a prontidão para movimento"(WULF, 2016).

(WULF, 2016).

No boxe, por exemplo, a pose inclui posições como a posição de guarda e a posição de defesa, onde cada uma tem um propósito específico relacionado à preparação e proteção do boxeador

O movimento, por outro lado, refere-se à mudança de posição ou postura de uma ou mais partes do corpo ao longo do tempo.

> "Movimento é a mudança de posição ou postura ao longo do tempo, essencial para a execução de tarefas motoras e atividades físicas".

(SCHMIDT; LEE, 2019)

Vimos então que pose é independente do tempo, e que movimento é a passagem de uma pose para outra, no tempo.

Por ser independente do tempo, a classificação de poses precisa ser feita a cada amostra, que é o RNF1.

O RNF2 vem do objetivo de se usar as poses e movimentos para interagir com jogos eletrônicos, limita o tempo máximo de classificação do movimento ao tempo médio de reação humana, é uma questão prática, um comando para um jogo precisa ser visto como responsivo.

O RNF3 vem do uso humano e potencialmente comercial do projeto; Existem diversas maneiras de se medir movimentos humanos, algumas delas invasivas inclusive, como por exemplo a Eletromiografia (EMG) Intramuscular, porém, é necessário um método não invasivo por questões éticas e comerciais.

O RNF4 complementa o RF3 e o RNF3; durante o uso do projeto, vários golpes serão usados, usar fios para a transmissão de dados de telemetria seria problemático.

3.3 Design e Arquitetura

Devido ao requisito de responsividade do módulo de telemetria, utilizou-se o ROS2, que possui ferramentas de manipulação de threads, criação de publicadores e subscritores, além de suporte à troca de mensagens. Esses conceitos são organizados de forma adequada com a arquitetura de microserviços, que foi escolhida para o projeto.

3.3.1 ROS2 (Robot Operating System 2)

O ROS2 (Robot Operating System 2) é uma plataforma flexível de software para robótica, projetada para simplificar o desenvolvimento de aplicações robóticas. O ROS2 oferece uma infraestrutura robusta para comunicação entre diferentes componentes de software, facilitando a criação de sistemas modulares e escaláveis.

- **Manipulação de Threads**: ROS2 fornece suporte nativo para a criação e gerenciamento de threads, permitindo que diferentes partes do sistema operem de maneira assíncrona e eficiente.
- **Publicadores e Subscritores**: A arquitetura de comunicação do ROS2 baseia-se em um modelo de publicador-subscritor, onde os nós (nodes) podem publicar mensagens em tópicos ou se inscrever para receber mensagens desses tópicos.
- **Troca de Mensagens**: A comunicação entre nós no ROS2 é facilitada pela troca de mensagens, que pode ocorrer de forma síncrona ou assíncrona, permitindo a integração suave de componentes distribuídos.
- **Middleware DDS**: ROS2 utiliza o Data Distribution Service (DDS) como middleware para a comunicação, oferecendo alta performance, baixa latência e suporte a comunicação em tempo real.

3.3.2 Arquitetura de Microserviços

A arquitetura de microserviços é um estilo de design de software onde uma aplicação é estruturada como um conjunto de serviços pequenos, independentes e que se comunicam entre si. Cada serviço é desenvolvido, implantado e mantido de forma independente, permitindo uma escalabilidade e manutenção mais fácil do sistema.

- **Escalabilidade**: Microserviços podem ser escalados de forma independente, permitindo alocação de recursos específicos conforme a demanda.
- **Desenvolvimento Independente**: Equipes podem trabalhar em diferentes serviços de forma paralela e independente, acelerando o ciclo de desenvolvimento.
- **Resiliência**: Falhas em um serviço não afetam o sistema como um todo, aumentando a robustez e resiliência da aplicação.
- **Deploy Contínuo**: Permite a implantação contínua de novas versões de serviços sem interrupção do sistema completo.

3.3.3 Implementação com ROS2 e Microserviços

Para o projeto de telemetria, utilizamos ROS2 e a arquitetura de microserviços devido às suas vantagens em termos de modularidade, escalabilidade e manutenção. A implementação inclui a criação de diversos nós do ROS2 que funcionam como microserviços independentes. Cada nó é responsável por uma funcionalidade específica, como coleta de dados de sensores, processamento de dados, e comunicação com outros módulos.

Na Figura 4 podemos ver a macro organização do projeto: Iniciamos com o usuário, que utiliza uma pulseira que carrega o sensor da IMU, esse sensor comunica de forma wireless com o dongle, que por sua vez transmite os dados via USB para o computador. A IMU, representada em azul, é um módulo fechado, e esse projeto é um cliente desse módulo, todo os drivers necessários para

telemetria são nativos da IMU usada no projeto. O módulo de telemetria inicia a implementação deste trabalho, após processar os dados recebidos pela IMU, publica-os no tópico Ros2 'imu/Angles', e fornece serviços (Ros2 services) para o usuário. O módulo de classificação possui o nó(Ros2 node) de classificação de poses, inscrito no tópico 'imu/Angles', e publicador do tópico 'pose', que recebe o resultado de classificação de poses. Por fim, o nó de classificação de técnicas, é inscrito do tópico 'pose', e utiliza as informações recebidas para classificar técnicas.

3.3.3.1 Nós e Comunicação

Os nós do ROS2 se comunicam através de tópicos, serviços e ações, utilizando o middleware DDS para garantir uma comunicação eficiente e em tempo real. A arquitetura de microserviços permite que novos nós sejam facilmente adicionados ou substituídos, sem impactar os outros componentes do sistema.

Sobre o design dos módulos e classes, seguiu-se os princípios SOLID, e veremos com mais detalhes nas seções dedicadas a cada módulo.

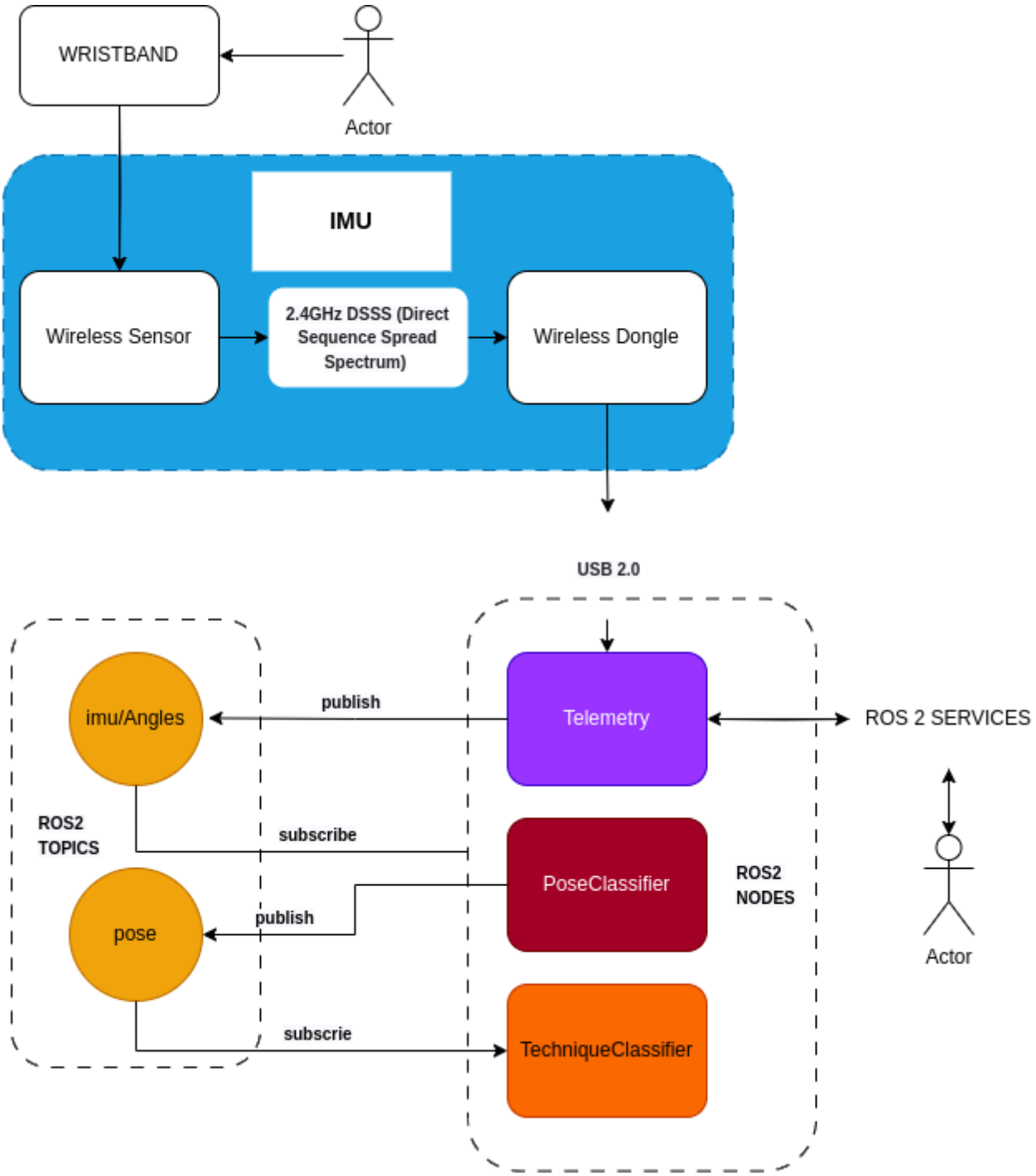
3.4 Desenvolvimento e testes Automatizados

O desenvolvimento foi feito em conjunto com os testes automatizados, da seguinte maneira: Inicialmente era feito uma prova de conceito, como já citado anteriormente, então foi aplicado Desenvolvimento Orientado por Comportamento (Behavior Driven Development - BDD), em que primeiro de descrito o comportamento esperado de uma classe, implementa-se a classe, e então o teste executado para garantir que a classe se comporta como esperado.

A biblioteca Catch2 foi usada para criação dos testes, que podem ser encontrados na pasta "test", dentro de cada módulo do repositório desse projeto.

Foi construído um mock do módulo de telemetria para agilizar o desenvolvimento e para testes dos módulos dependentes. Um mock é um módulo auxiliar que simula um módulo real, implementado sua interface. O uso desse mock foi essencial para o desenvolvimento, por conseguir simular uma mensagem enviada

Figura 4 – Diagrama da arquitetura do projeto



Fonte: próprio autor

pela IMU, diminuindo assim a necessidade de ir ao laboratório fazer testes.

3.4.1 Módulo de Telemetria

A módulo de telemetria foi baseado em um projeto python fornecido pela equipe do Laboratório de automação e robótica(LARA), da Universidade de Brasília. Ros2 fornece suporte para pyhton e c++, porém, considera-se mais adequado fazer grandes projetos com o suporte de c++, o módulo de telemetria implementado para esse projeto foi pensado de maneira a contribuir com os projetos do LARA, e seu design foi planejado de maneira a facilitar a expansão e manutenção do módulo. A IMU oferece diversas opções de streaming de telemetria, e para esse projeto foram implementadas apenas 2, mas o objetivo do módulo de telemetria é que seja fácil de se expandir a classes para obter os demais dados.

O 3-Space-Sensor pode ser utilizado via USB, ou via wireless, devido ao requisito de identificação de poses e movimentos do boxe, optou-se pela comunicação wireless. Em termos de protocolo de comunicação, existe a possibilidade de comunicação com bytes, ou com strings, devido à confiabilidade e simplicidade do protocolo com bytes, optou-se pela primeira opção.

Para utilizar comunicação wireless com mensagens em bytes, basta enviar os comandos com bytes, o 3-space-sensor envia a resposta do comando no mesmo formato.

3.4.1.1 Classe 'SpaceSensor'

O design de cada classe foi pensado de maneira à atender os princípios SOLID. O princípio de responsabilidade única foi aplicado de maneira a modularizar o escopo de cada classe. A classe 'SpaceSensor' é uma representação direta das estruturas definidas na documentação da IMU usada nesse projeto, com um conjunto de 'enums' que mapeiam bytes documentados, com significados acordados, são eles os comandos, comandos de 'streaming', modos de validação, e tamanhos esperados de respostas. Essa classe define também duas estruturas, comando binário 2 e resposta binária 3, além de métodos dedicados ao parse de

cada resposta usada.

Para o envio de comandos binário, se criou uma estrutura(C++ struct) fiel à definição dada pela documentação da IMU. Os membros foram organizados de forma descritiva para o usuário dos comandos, e, com um método específico, é possível serializar um objeto dessa estrutura, obtendo-se assim um conjunto de bytes que representam um comando válido.

Tabela 2 – Estrutura comando binário

Membro	**Descrição**	**Tipo de dado**
StartOfPacket	Define o modo de validação aplicado.	ValidateMode(enum)
LogicalId	Identifica o ID lógico (e.g., IMU) associado ao comando.	$uint8_t$
Command	Especifica o comando a ser usado.	$uint8_t$
CommandData	Campo opcional de dados associado ao comando (pode ser vazio).	std::vector<$uint8_t$ >
Checksum	Byte de validação do comando, soma dos membros anteriores, módulo 256.	$uint8_t$

De forma análoga ao comando binário, foi criada uma estrutura dedicada para a resposta que se espera da IMU, que recebe um conjunto de bytes, e cria um objeto, já com validação da resposta e com fácil acesso ao dado de resposta em si.

Tabela 3 – Estrutura Resposta binária

Membro	**Descrição**	**Tipo de Dados**
Status	Indica o resultado da execução do comando no dispositivo.	$int8_t$
LogicalId	Identifica o dispositivo lógico (por exemplo, IMU) que enviou a resposta.	$uint8_t$
DataLength	Especifica o comprimento dos dados opcionais da resposta.	$uint8_t$
ResponseData	Contém os dados reais da resposta do dispositivo (pode estar vazio).	std::vector<$uint8_t$ >

Uma vez que foi enviado um comando válido, e recebeu-se uma resposta válida, resta então, fazer o parse do campo ResponseData, cada comando vai ter ou não esse campo, e cada comando com esse campo, vai ter um tamanho de

interpretação diferente, a exemplo temos o resposta dos ângulos de Euler, que é um conjunto de 12 bytes, cada conjunto consecutivo de 4, é um ângulo radiano e em big endian.

As estruturas de comando binário e resposta binária são capazes de enviar e receber qualquer comando, sem qualquer modificação, o que segue o aspecto 'fechado' do princípio 'aberto-fechado', porém, é necessário que se faça o parse dessa resposta. A classe foi construída de maneira que, para se obter uma tipo de resposta nova da IMU, basta implementar um parse específico para esta resposta nova, sem alterar o código já existente, o que atende ao aspecto 'aberto' do princípio 'aberto fechado'.

3.4.1.2 Classe 'SpaceSensor': Implementação

Definição do Header: A documentação da IMU define a interface de comandos, explicita o protocolo de comunicação binária (usada no projeto) e a em formato de texto. Como explicado anteriormente, a classe 'SpaceSensor' mapeia, em código, as interfaces definidas da IMU. O recurso 'enum' foi utilizado para mapear os bytes de cada recurso definido na documentação. Começamos pelos comandos usados no projeto.

```
#pragma once

#include <string>
#include <vector>
#include <cstddef>
#include <array>
#include <numeric>

namespace telemetry
{
    class SpaceSensor
    {
    public:
```

```
enum Commands
{
    SetEulerAngleDecompositionOrder = 0x10,
    OffsetWithCurrentOrientation = 0x13,
    SetBaseOffsetWithCurrentOrientation = 0x16,
    SetStreamingSlots = 0x50,
    SetStreamingTiming = 0x52,
    StartStreaming = 0x55,
    StopStreaming = 0x56,
    TareWithCurrentOrientation = 0x60,
    TareWithQuaternion = 0x61,
    SetCompassEnabled = 0x6D
};
```

Comandos de streaming: Há também a organização dos comandos de streaming de dados

```
enum StreamingCommand
{
    ReadTaredOrientationAsQuaternion = 0x00,
    ReadTaredOrientationAsEulerAngles = 0x01,
    ReadTaredOrientationAsRotationMatrix = 0x02,
    ReadTaredOrientationAsAxisAngle = 0x03,
    ReadTaredOrientationAsTwoVector = 0x04,
    ReadDifferenceQuaternion = 0x05,
    ReadUntaredOrientationAsQuaternion = 0x06,
    ReadUntaredOrientationAsEulerAngles = 0x07,
    ReadUntaredOrientationAsRotationMatrix = 0x08,
    ReadUntaredOrientationAsAxisAngle = 0x09,
    ReadUntaredOrientationAsTwoVector = 0x0A,
    ReadTaredTwoVectorInSensorFrame = 0x0B,
    ReadUntaredTwoVectorInSensorFrame = 0x0C,
    ReadAllNormalizedComponentSensorData = 0x20,
```

```
    ReadNormalizedGyroscopeVector = 0x21,
    ReadNormalizedAccelerometerVector = 0x22,
    ReadNormalizedCompassVector = 0x23,
    ReadAllCorrectedComponentSensorData = 0x25,
    ReadCorrectedGyroscopeVector = 0x26,
    ReadCorrectedAccelerometerVector = 0x27,
    ReadCorrectedCompassVector = 0x28,
    ReadCorrectedLinearAcceleration = 0x29,
    ReadTemperatureC = 0x2B,
    ReadTemperatureF = 0x2C,
    ReadConfidenceFactor = 0x2D,
    ReadAllRawComponentSensorData = 0x40,
    ReadRawGyroscopeVector = 0x41,
    ReadRawAccelerometerVector = 0x42,
    ReadRawCompassVector = 0x43,
    ReadBatteryVoltage = 0xC9,
    ReadBatteryPercentage = 0xCA,
    ReadBatteryStatus = 0xCB,
    ReadButtonState = 0xFA,
    NoCommand = 0xFF
};
```

Modos de validação: Os modos de validação definem qual algoritmo será usado para validar o buffer.

```
enum ValidateMode
{
    Simple = 0xf8,
    PrependResponseHeader = 0xfa
};
```

Tamanho de respostas: Cada resposta de sreaming possui um tamanho pré definido. Foram mapeadas as respostas usadas. Vale lembrar que o código está

feito de forma que mapear uma resposta nova, exige apenas uma expansão, e não uma modificação do código.

```
enum ResponsesSizes
{
    Header = 3,
    EulerAngle = 12,
    Quaternion = 16
};
```

Comando binário: A estrura 'BinaryCommand' é definida dentro da classe 'SpaceSensor'. Foi implementada de maneira a fazer uma interface intuitiva entre o programador e a IMU, uma forma de construir o buffer de commando de maneira descritiva. Observe que o byte de validade, 'CheckSum' é construído automaticamente quando se define os demais componentes do comando, e que o método 'Get' retorna o comando no formato de buffer. Ou seja, é uma estrutura de serializar um comando.

```
struct BinaryCommand
{
    uint8_t StartOfPacket = ValidateMode::Simple;
    uint8_t LogicalId = 0x03; // Imu available for use.
    uint8_t Command;
    std::vector<uint8_t> CommandData;
    uint8_t CheckSum;

    BinaryCommand(uint8_t startOfPacket,
                  uint8_t logicalId,
                  uint8_t command,
                  std::vector<uint8_t> commandData = {})
        : StartOfPacket(startOfPacket),
          LogicalId(logicalId),
          Command(command),
```

```
        CommandData(commandData)
    {
        CheckSum = (LogicalId +
                    Command   +
                    std::accumulate(commandData.begin(),
                    commandData.end(), 0)) % 256;
    }

    std::vector<uint8_t> SpaceSensor::BinaryCommand::Get()
    {
        std::vector<uint8_t> commandBuffer{StartOfPacket, LogicalId, Command};

        std::ranges::copy(CommandData.begin(), CommandData.end(),
                          std::back_inserter(commandBuffer));

        commandBuffer.push_back(CheckSum);

        return commandBuffer;
    }

    bool operator==(const BinaryCommand &other) const = default;
};
```

Resposta binária: A estrura 'BinaryResponse' é um definida dentro da classe 'SpaceSensor'. Foi implementada de maneira a fazer uma interface intuitiva entre o programador e a IMU, uma forma de obter um objeto com os membros componentes da resposta, a partir de um buffer. Observe que o construtor da classe valida o buffer recebido e disponibiliza os dados interessantes ao usuário, ou seja, a classe faz a desserialização do buffer.

```
struct BinaryResponse
{
    int8_t Status;
```

```
uint8_t LogicalId;
uint8_t DataLength = 0x00;
std::vector<uint8_t> ResponseData;

SpaceSensor::BinaryResponse::BinaryResponse(
    const std::vector<uint8_t> &buffer,
    const ResponsesSizes responseSize)
{
    std::memcpy(&Status,
    buffer.data(), sizeof(uint8_t));

    if (Status != 0x00)
    {
        return;
    }

    std::memcpy(&LogicalId,
                buffer.data() + sizeof(uint8_t),
                sizeof(uint8_t));

    if (LogicalId != 0x03)
    {
        return;
    }

    std::memcpy(&DataLength,
                buffer.data() + 2 * sizeof(uint8_t),
                sizeof(uint8_t));

    if (DataLength != responseSize)
    {
        return;
    }
```

```
        std::ranges::copy(buffer.begin() + (ResponsesSizes::Header * sizeof(uint
                          std::back_inserter(ResponseData));

        m_isValid = true;
    }

    bool operator==(const BinaryResponse &other) const = default;

    bool IsValid() { return m_isValid; }

private:
    bool m_isValid = false;
};
```

Parser: O método de 'Parser' é responsável por converter o dado em si da resposta para formato válido em código, isto é, reinterpretar um conjunto de bytes em big-endian para um vetor de floats. Para isto é preciso também saber o número de floats que se espera interpretar.

```
std::vector<float> SpaceSensor::Parser(
    const std::vector<uint8_t> &responseData,
    const uint8_t dataSize)
{
    std::vector<float> angles;

    const auto steps = dataSize / 4; // float size

    std::ranges::for_each(std::views::iota(0, steps),
        [&angles, &responseData](const auto step)
        {
            float data;
            std::memcpy(&data, responseData.data() +
```

```
            (step * sizeof(float)), sizeof(float));

            std::reverse(reinterpret_cast<uint8_t *>(&data),
                         reinterpret_cast<uint8_t *>(&data) + sizeof(float));

            angles.push_back(data);
        });

    return angles;
}
```

ParseQuaternion/ParseEulerAngle: O métodos 'ParseQuaternion' e 'ParseEulerAngle' são clientes do método 'Parser', apenas definem o tamanho esperado da resposta. Nesse context, o parser é específico para ângulos, porém, possíveis parsers de outras respostas fornecidas pela IMU podem seguir o mesmo princípio de reinterpretação de endianismo.

```
std::vector<float> SpaceSensor::ParseEulerAngle(
    const std::vector<uint8_t> &responseData)
{
    std::vector<float> angles;

    if (responseData.size() < ResponsesSizes::EulerAngle)
    {
        return angles;
    }

    angles = Parser(responseData, ResponsesSizes::EulerAngle);

    return angles;
}

std::vector<float> SpaceSensor::ParseQuaternion(
```

```
    const std::vector<uint8_t> &responseData)
{
    std::vector<float> angles;

    if (responseData.size() < ResponsesSizes::Quaternion)
    {
        return angles;
    }

    angles = Parser(responseData, ResponsesSizes::Quaternion);

    return angles;
}
```

3.4.1.3 Classe 'ImuNode'

Em projetos ROS2, módulos de publicadores e/ou subscritores são chamados nós(node), a classe 'ImuNode' foi criada para fazer a interface entre o usuário e os comandos da IMU, mapeados na classe SpaceSensor. Ao ser inicializada, são enviados comandos para a IMU que definem os os espaços de streaming desejados, o tempo de streaming(frequência), ordem da resposta que será lida, e então é disponibilizado comandos no formato ROS2. Os comandos utilizados são os de definir o zero das coordenadas(tare),começar streaming e parar streaming. Uma vez iniciado o streaming, na frequancia determiada, se recebe da IMU a resposta da telemetria, que passa pelo parse definido na classe SpaceSensor, e então o resultado é publicado em um tópico (ros topic) dedicado.

Construção do Nó de Telemetria: A classe 'ImuNode' é construida para expor serviços Ros2 para o usuário, que comunicam com a IMU usando a bilioteca boost/asio e a classe 'SpaceSensor'. O código a seguir é o construtor da classe ImuNode: Iniciamos com a tentativa de conectar com uma porta USB, em que o receptor da IMU está conectado e recebendo dados, em seguida definimos a frequência da porta, criamos o tópico de publicação dos ângulos, e então definimos os serviços disponíveis para o usuário.

```
ImuNode::ImuNode() : Node("imuNode"), m_io()
{
    try
    {
        m_serialPort =
            std::make_shared<boost::asio::serial_port>(m_io,
                                                       "/dev/ttyACM0");
    }
    catch (const boost::system::system_error &e)
    {
        RCLCPP_ERROR(get_logger(),
                     "Failed to open serial port: %s", e.what());
        throw;
    }

    m_serialPort->set_option(
        boost::asio::serial_port_base::baud_rate(115200));

    m_publisher =
        create_publisher<std_msgs::msg::Float32MultiArray>("imu/Angles", 1);

    m_startService = create_service<std_srvs::srv::Empty>(
        "imu/start",
        std::bind(&ImuNode::StartStreaming,
            this,
            std::placeholders::_1,
            std::placeholders::_2),
            rmw_qos_profile_services_default);

    m_stopService = create_service<std_srvs::srv::Empty>(
        "imu/stop",
        std::bind(&ImuNode::StopStreaming,
            this, std::placeholders::_1,
```

```
            std::placeholders::_2));

    m_serviceTare = create_service<std_srvs::srv::Empty>(
        "imu/tare", std::bind(&ImuNode::TareSensor,
        this, std::placeholders::_1,
        std::placeholders::_2));

    m_serviceTareQuaternion = create_service<std_srvs::srv::Empty>(
        "imu/tareQuaternion",
        std::bind(&ImuNode::TareSensorQuaternion,
            this,
            std::placeholders::_1,
            std::placeholders::_2));

    m_serviceOffset = create_service<std_srvs::srv::Empty>(
        "imu/offset",
        std::bind(&ImuNode::OffsetWithCurrentOrientation,
            this,
            std::placeholders::_1,
            std::placeholders::_2));

    m_serviceBaseOffset = create_service<std_srvs::srv::Empty>(
        "imu/baseOffset",
        std::bind(&ImuNode::SetBaseOffsetWithCurrentOrientation,
                  this,
                  std::placeholders::_1,
                  std::placeholders::_2));

    SetStreamingSlots({SpaceSensor::StreamingCommand::ReadTaredOrientationAsQuat
                       SpaceSensor::StreamingCommand::NoCommand,
                       SpaceSensor::StreamingCommand::NoCommand,
                       SpaceSensor::StreamingCommand::NoCommand,
                       SpaceSensor::StreamingCommand::NoCommand,
```

```
                        SpaceSensor::StreamingCommand::NoCommand,
                        SpaceSensor::StreamingCommand::NoCommand,
                        SpaceSensor::StreamingCommand::NoCommand});

    SetStreamingTiming(5);

    SetCompassEnabledToZero();

    m_timer = create_wall_timer(std::chrono::milliseconds(5),
                                std::bind(&ImuNode::LoopCallback, this));
}
```

Aplicação de comandos: Para a aplicação de comandos, recebe-se um 'BinaryCommand' e aplica-se o comando com o auxilio da biblioteca boost/asio. A resposta é então apresentada de forma opcional.

```
void ImuNode::ApplyCommand(
    SpaceSensor::BinaryCommand &binaryCommand,
    bool showResponse)
{
    m_serialPort->write_some(boost::asio::buffer(binaryCommand.Get()));

    std::this_thread::sleep_for(std::chrono::milliseconds(100));

    if (showResponse)
    {
        std::vector<uint8_t> responseBuffer(128);

        int bytesRead =
            m_serialPort->read_some(boost::asio::buffer(responseBuffer));

        if (bytesRead > 0)
        {
```

```
            std::ranges::for_each(responseBuffer,
                [&](const auto byte)
                    { RCLCPP_INFO(get_logger(), ">> %X", byte); });
        }
    }
}
```

Comando de início de streaming: A implementação a seguir do método 'StarStreaming' demonstra o uso de um comando binário, dos enums definidos, e do método anterior de aplicar comandos

```
void ImuNode::StartStreaming(
    [[maybe_unused]] const std::shared_ptr<std_srvs::srv::Empty::Request> reques
    [[maybe_unused]] std::shared_ptr<std_srvs::srv::Empty::Response> response)
{
    if (m_streaming)
    {
        RCLCPP_WARN(get_logger(), "Already streaming");

        return;
    }

    RCLCPP_INFO(get_logger(), "Start Streaming");

    SpaceSensor::BinaryCommand binaryCommand(SpaceSensor::ValidateMode::Simple,
                                             0x03,
                                             SpaceSensor::Commands::StartStream
    ApplyCommand(binaryCommand);

    m_streaming = true;
}
```

Leitura em loop de dados de telemetria: Por fim, a lógica central de leitura

de dados é definido com o método 'LoopCallback'. Esse método é chamado na frequência determinada pelo construtor da classe, a cada 5 milisegundos, e é responsável por adquirir os dados fornecidos pela IMU e publicar no tópico 'imu/angles'

```
void ImuNode::LoopCallback()
{
    if (not m_streaming)
    {
        RCLCPP_INFO(get_logger(), "Not streaming");
        std::this_thread::sleep_for(std::chrono::seconds(1));
        return;
    }

    int bytesAvailable;

    if (ioctl(m_serialPort->native_handle(),
              FIONREAD, &bytesAvailable) == -1)
    {
        RCLCPP_ERROR(get_logger(), "native_handle failed");
        return;
    }

    constexpr int quaternionBinaryResponseSize =
        (SpaceSensor::ResponsesSizes::Header +
         SpaceSensor::ResponsesSizes::Quaternion);

    if (bytesAvailable < quaternionBinaryResponseSize)
    {
         return;
    }

    std::vector<uint8_t> responseBuffer(quaternionBinaryResponseSize);
```

```
    const auto bytesRead =
        m_serialPort->read_some(
            boost::asio::buffer(responseBuffer,
                                quaternionBinaryResponseSize));

    if (bytesRead <= 0)
    {
        RCLCPP_INFO(get_logger(), "No bytes read");
        return;
    }

    SpaceSensor::BinaryResponse
        binaryResponse(responseBuffer,
        SpaceSensor::ResponsesSizes::Quaternion);

    if (not binaryResponse.IsValid())
    {
        m_serialPort->read_some(
            boost::asio::buffer(responseBuffer, 1));
        return;
    }

    const auto angles =
        SpaceSensor::ParseQuaternion(binaryResponse.ResponseData);


    std_msgs::msg::Float32MultiArray message;
    message.data = {angles[0], angles[1], angles[2], angles[3]};

    m_publisher->publish(message);
}
```

3.4.2 Módulo de Classificação

O módulo de telemetria, abastecido pela Imu, fornece 3 conjuntos de informações que podem ser usadas para classificar poses e movimentos do boxe, orientação, giroscópio e acelerômetro. Classificar movimentos exige resolver uma questão, que em teoria de classificação se denomina mereologia, que é o estudo do todo pela parte. Essa questão aparece pois não se tem,inicialmente, um ponto claro que define o início e o fim de um golpe em termos dos dados disponíveis, porém, existem definições de golpe do ponto de vista teórico, pode-se considerar que um golpe do boxe é:

Um movimento que vai da guarda à um ponto final específico, com uma trajetória determinada.

Porém, vamos explorar apenas a primeira parte da sentença para esse trabalho, não usaremos informação de trajetória. Consideramos então que um golpe do boxe é:

Um movimento que vai da guarda à um ponto final específico

A sentença anterior define a ideia central, prática, de solução do problema proposto nesse projeto. A descrição do experimento feito para desenvolver essa premissa, vem a seguir.

Classificação de poses: O módulo de classificação de poses é inscrito no tópico 'imu/angles' e publica no tópico 'pose'. A estrutura de definição de nó funciona de forma análoga ao módulo de telemetria. Segue a implementação do método proposto para classificar poses. Repare na linha comentada de resultado obtido a partir de knn, isso foi mantido neste documento para mostrar que nesse contexto as estratégias de classificação tem o mesmo papel, isto é, usar o knn ou kmeans não altera a lógica do módulo de classificação de poses.

```
void PoseClassification::topicCallback(
    const std_msgs::msg::Float32MultiArray::SharedPtr message)
{

    std::vector<float> angles = message->data;
```

```
classes::Data dataPoint(angles, classes::Data::Poses::Unclassified);

// const auto result = m_knn.Classify(dataPoint);
const auto result = m_kMeans.Classify(dataPoint);

switch (result.Label)
{
case classes::Data::Poses::Guard:

    if (m_lastPose != classes::Data::Poses::Guard)
    {

        RCLCPP_INFO(get_logger(), "GUARD");
    }

    m_lastPose = classes::Data::Poses::Guard;
    break;

case classes::Data::Poses::JabEnd:

    if (m_lastPose != classes::Data::Poses::JabEnd)
    {

        RCLCPP_INFO(get_logger(), "JAB END");
    }

    m_lastPose = classes::Data::Poses::JabEnd;
    break;

case classes::Data::Poses::HookEnd:

    if (m_lastPose != classes::Data::Poses::HookEnd)
```

```
        {

            RCLCPP_INFO(get_logger(), "HOOK END");
        }

        m_lastPose = classes::Data::Poses::HookEnd;
        break;

    case classes::Data::Poses::UppercutEnd:

        if (m_lastPose != classes::Data::Poses::UppercutEnd)
        {

            RCLCPP_INFO(get_logger(), "UPPERCUT END");
        }

        m_lastPose = classes::Data::Poses::UppercutEnd;
        break;

    case classes::Data::Poses::Unknown:

        if (m_lastPose != classes::Data::Poses::Unknown)
        {

                RCLCPP_INFO(get_logger(), "UNKNOWN");
        }

        m_lastPose = classes::Data::Poses::Unknown;
        break;

    default:
        break;
    }
```

```
    std_msgs::msg::Int32 pose;

    pose.data = m_lastPose;

    m_publisher->publish(pose);
}
```

Implementação do Knn: Na fundamentação teórica foi explicado o algorítmo K vizinhos mais próximos, veremos uma nova explicação, dessa vez focada na implementação:

Estrutura de Distância classificada: Para entender a implementação desse algoritmo, vamos entender as classes básicas usadas no módulo de telemetria, a classe LabeledDistance agrega uma valor em ponto flutuante e uma classificação, e define operadores sobre que utilizam o valor em ponto flutuante como alvo, ou seja, é uma forma de fazer operações sobre um ponto e manter sua classificação.

```
struct LabeledDistance
{
    LabeledDistance(const float distance,
        const uint8_t label)
        : Distance(distance), Label(label)
    {
    }

    bool operator<(const LabeledDistance &other) const
    {
        return Distance < other.Distance;
    }

    bool operator==(const LabeledDistance &other) const = default;

    float Distance;
```

```
    uint8_t Label;
};
```

Estrutura central de dados: A estrutura central que agrega os dados para classificação é a struct 'data', que reune um conjunto de dados, isto é, um std::vector de floats: 'Features', uma classificação, 'Label', isto é, esta estrutura, reune um ponto, ou uma amostra, um dado para classificar, isto é, um conjunto de características que conhecemos a classificação. O algoritmo principal dessa estrutura é a distância euclidiana, que é simplesmente a distancia euclidiana clássica, mas que carrega a informação de classificação, label, ao ser calculada, isto é, em um contexto de classificação de poses do boxes, posso ter um ponto 'A' com classificação conhecida como 'guarda', receber um ponto 'B' com classificação desconhecida, e ao usar o método de distância euclidiana, saberei o valor de distância, e que essa distância é referente a um ponto sabidamente classificado como guarda. Repare que há um enum de Poses nessa estrutura, esse enum é específico para esse projeto, porém, para classificar um outro conjunto de poses, pode-se alterar esse conjunto, e também pode-se estudar reformulação desse design para, por exemplo, carregar o conjunto de poses a partir de um arquivo de configuração.

```
struct Data
{
    enum Poses
    {
        Guard = 0,
        JabEnd = 1,
        HookEnd = 2,
        UppercutEnd = 3,
        Unknown = 100,
        Unclassified = 255
    };

    std::vector<float> Features;
```

```
    uint8_t Label;

    Data() : Features({}), Label(0) {}

    Data(const std::vector<float> &features,
         const uint8_t label) : Features(features),
                                Label(label) {}

    bool operator==(const Data &other) const = default;

    Data operator+(const Data &other) const;
    Data operator/(float divisor) const;

    LabeledDistance EuclideanDistance(const Data &other);
};
```

Classe KNN A classe Knn agrega um conjunto de amostras com classificação conhecida, isto é, um std::vector da estrutura data, com label diferente de desconhecido. **O conjunto de dados (vector de data) é o conjunto de treinamento do projeto.** A classe também possui o K usado para classificação, e, por fim, o um método de classificação.

```
class Knn
{
public:
    Knn() : m_k(3) {}
    Knn(const uint8_t k) : m_k(k) {}

    void AddData(const std::vector<classes::Data> &data);

    std::vector<classes::Data> GetData();

    classes::Data Classify(classes::Data &data);
```

```
private:
    uint8_t m_k;

    std::vector<classes::Data> m_data;
};
```

Classificação com KNN Por fim, temos a classificação com Knn: Iniciamos obtendo os vizinhos, que são obtidos a calcular a distancia euclidiana do ponto recebido a todos os pontos do conjunto de treinamento. Então ordenamos todos os vizinhos, obtemos os candidatos, que são os k vizinhos mais próximos, e então contamos o label que mais aparece entre a lista de candidatos. O label com mais ocorrências dentre os K, é usado para classificar o ponto.

```
classes::Data Knn::Classify(classes::Data &data)
{

    std::vector<classes::LabeledDistance> neighbors;

    std::ranges::transform(m_data,
                           std::back_inserter(neighbors),
                           [&data](const auto &dataPoint)
                           { return data.EuclideanDistance(dataPoint); });

    std::ranges::sort(neighbors, [](const classes::LabeledDistance &a,
                                    const classes::LabeledDistance &b)
                      { return a.Distance < b.Distance; });

    std::vector<classes::LabeledDistance> candidates;

    candidates.insert(candidates.begin(),
                      neighbors.begin(),
                      std::next(neighbors.begin(),
```

```
                                    std::min(neighbors.size(),
                                    static_cast<size_t>(m_k))));

    std::vector<uint8_t> labels(m_k);

    std::ranges::transform(candidates, labels.begin(),
                           [](const classes::LabeledDistance &neighbor){
                               return neighbor.Label;
                           });

    std::unordered_map<uint8_t, size_t> labelCounts;

    for (uint8_t label : labels)
    {
        labelCounts[label]++;
    }

    uint8_t mostFrequentLabel = labels[0];

    size_t maxCount = labelCounts[mostFrequentLabel];

    for (const auto &[label, count] : labelCounts)
    {
        if (count > maxCount)
        {
            maxCount = count;
            mostFrequentLabel = label;
        }
    }

    data.Label = mostFrequentLabel;
    return data;
}
```

Classe de grupos A alternativa de classificação é a classificação com centroides. Essa classificação usa uma classe de grupos, que usa a estrutura de dados, e fornece Média e desvio padrão.

```
class Group
{
public:
    Group(const std::vector<Data> &points);

    std::optional<Data> Mean();
    std::optional<float> StandardDeviation();

    std::optional<Data> GetMean();
    std::optional<float> GetStandardDeviation();

private:
    std::vector<Data> m_points;
    std::optional<Data> m_mean;
    std::optional<float> m_standardDeviation;
}
```

Kmeans ou classificador de centroide A Classificação com centroides é similar ao Knn, porém possui uma etapa anterior, em que os centroides são calculados assim que o conjunto de treinamento é fornecido, e então é calculado a distância euclidiana do ponto que se deseja classificar ao centroide de cada grupo. Isso permite que esse algoritmo use um conjunto maior de dados, pois são feitas apenas comparações para cada grupo, não para cada membro do grupo.

Classificação com Centroide A classificação com centroide também permite um uso de parâmetro de limiar de classificação, que faz com que apenas os pontos com distância dentro do limiar sejam classificados, pontos além desse limiar ficam com classificação indefinida.

```
class KMeans
```

```
{
public:
    struct Candidate
    {
        Candidate(const classes::LabeledDistance &point, float threshold)
            : Point(point), Threshold(threshold) {}

        classes::LabeledDistance Point;
        float Threshold;
    };

    KMeans() {}

    void AddGroup(const classes::Group &group);
    void AddGroup(const std::vector<classes::Data> &points);

    classes::Data Classify(classes::Data &data);

    float GetLastMinimumDistance();

private:
    float m_lastMinimumDistance;
    std::vector<classes::Group> m_groups;
};

classes::Data KMeans::Classify(classes::Data &data)
{
    std::vector<Candidate> candidates;

    std::ranges::transform(m_groups,
                std::back_inserter(candidates),
                    [&data](auto &group)
                    { return Candidate(data.EuclideanDistance(group.GetMean().va
```

```
                                    group.GetStandardDeviation().value()); });

    std::ranges::sort(candidates, [](const Candidate &a, const Candidate &b)
            { return a.Point.Distance < b.Point.Distance; });

    ((2 * candidates.front().Point.Distance) < candidates.front().Threshold)
        ? data.Label = candidates.front().Point.Label
        : data.Label = classes::Data::Poses::Unknown;

    return data;
}
```

Classificação de técnicas: O módulo de classificação de técnicas também é um nó. É inscrito no tópico 'pose' e que apresenta a classificação obtida para o usuário. Baseado no modelo escolhido para definir um movimento/técnica do boxe, segue a implementação:

```
void TechniqueClassification::topicCallback(const std_msgs::msg::Int32 message)
{
    if ((message.data == classes::Data::Poses::Guard) and
        (LastStablePose != classes::Data::Poses::Guard))
    {
        RCLCPP_INFO(get_logger(), "ON GUARD");

        LastStablePose = classes::Data::Poses::Guard;
        return;
    }

    if (LastStablePose == classes::Data::Poses::Guard)
    {
        switch (message.data)
        {
        case classes::Data::Poses::JabEnd:
```

```
            RCLCPP_INFO(get_logger(), "JAB");
            LastStablePose = classes::Data::Poses::JabEnd;

            break;

        case classes::Data::Poses::HookEnd:

            RCLCPP_INFO(get_logger(), "HOOK");
            LastStablePose = classes::Data::Poses::HookEnd;

            break;

        case classes::Data::Poses::UppercutEnd:

            RCLCPP_INFO(get_logger(), "UPPERCUT");
            LastStablePose = classes::Data::Poses::UppercutEnd;

            break;

        case classes::Data::Poses::Unknown:
            break;
        default:
            break;
        }
    }
}
```

4 Resultados

Repositório do projeto

https://github.com/ArielSixwings/ImuBoxing

O requisito não funcional 2: 'A classificação de um movimento deve ser feita em tempo menor ou igual ao tempo médio de reação humana.' foi cumprido em sua totalidade. Como visto anteriormente, os dados de telemetria são lidos a cada 5 milissegundos, e uma pose é classificada assim que a posição se torna próxima o suficiente de uma das poses de interesse, e todo o processo de classificação ocorre dentro do intervalo de leitura, ou seja, quando um dado N é publicado em um determinado tópico, todo o processamento do dado N-1 já ocorreu.

Considere que podemos assumir a pose de guarda como o ponto inicial do boxe para classificação. Considere também que para cada golpe, existe uma posição final do braço, antes de iniciar o movimento de volta para a guarda, então para os movimentos Jab, cruzado e uppercut, existem respectivamente as poses Jab-final, cruzado-final e uppercut-final. Temos então, inicialmente, um conjunto de 4 poses para se classificar.

Vimos que uma pose é, por definição, um estado sem tempo, isto é, pode ser definida por uma única posição. Inicialmente foi usado apenas o dado de ângulo de Euler fornecido pela IMU para definir o conjunto de características de uma pose.

$$\textbf{Caracteristica} = \boldsymbol{\Theta} = \{\theta_1, \theta_2, \theta_3\} \tag{4.1}$$

Temos então uma definição matemática para o que é uma pose, uma pose pose ser representada pela equação 4.1.

4.1 Conjuntos de treinamento, validação e teste

Com a definição foi feito um conjunto de treinamento: Para cada uma das 4 poses que se deseja classificar, foi obtido 2048 amostras, esse número é fácil de se obter com o aparato montado para essa etapa do projeto, os dados de telemetria são obtidos a cada 5 milissegundos, então obter 2048 dados de uma pose, significa ficar nessa pose por cerca de 2 segundos, período em que a integridade da pose dificilmente se é perdida. O conjunto de treinamento é obtido ao colocar a IMU no pulso e se posicionar da maneira adequada. Então para obter o conjunto de treinamento completo:

- A IMU é posicionada no pulso esquerdo do atleta.
- A IMU é ligada.
- Inicia-se o streaming de dados.
- O atleta se posiciona na pose de guarda 5
- O sistema obtém a quantidade desejada de dados na pose (a cada 5 milissegundos).
- O atleta descansa e então, entra na pose de final do jab. 6
- O sistema obtém a quantidade desejada de dados na pose (a cada 5 milissegundos).
- O atleta descansa e então, entra na pose de final do cruzado. 7
- O sistema obtém a quantidade desejada de dados na pose (a cada 5 milissegundos).
- O atleta descansa e então, entra na pose de final do gancho/uppercut. **??**
- O sistema obtém a quantidade desejada de dados na pose (a cada 5 milissegundos).

Figura 5 – Posição de guarda

Fonte: (COSTA, 2020)

Figura 6 – Posição final do jab

Fonte: (NGUYEN, 2012)

Figura 7 – Posição final do Cruzado

Fonte: (COSTA, 2020)

Figura 8 – Posição final do Gancho

Fonte: (COSTA, 2020)

$$\textbf{Grupo} = \boldsymbol{\Omega} = (\Theta_1, \Theta_2, \Theta_3, \ldots, \Theta_{2048}) \tag{4.2}$$

O conjunto de validação foi usado nos testes automatizados: Foi feito um teste usando a ferramenta catch2, com a estratégia BDD, Behavior driven development, em que se é verificado que: dado um ponto não classificado, mas que se sabe pertencer a um grupo x, quando o algoritmo de classificação em questão executa, o ponto recebe a classificação indicativa do grupo x. Para validação foi usado metade de cada conjunto, 1024 para cada grupo.

Queremos também manter um padrão mínimo para classificação de uma pose, isto é, determinar um critério que diga que um dado recebido/ponto que se quer classificar, esta próximo o suficiente para poder ser classificado como pertencente à determinado grupo, e aqui proximidade é literal, pois usamos distância euclidiana como parâmetro de comparação.

Um exemplo: sabemos que um ponto criado a partir do centroide de um grupo, +- um valor menor que o critério decidido, deve ser classificado como pertencente ao grupo, o teste desse cenário vem a seguir.

```
SCENARIO("Should Classify the data according to the Knn rule",
         "[Unit][strategy][Knn][Classify]")
{
    GIVEN("a vector of Data centered at 90.0, 0.0, 45.0")
    {
```

```
std::vector<std::vector<float>> features1;

for (int i = 0; i < 10; ++i)
{

    std::vector<float> point = {90.0, 0.0, 45.0};

    std::random_device randomDevice;
    std::mt19937 generator(randomDevice());
    std::uniform_real_distribution<float>
        distribution(-5.0, 5.0);

    float randomValue = distribution(generator);

    std::ranges::for_each(point, [&randomValue](auto &entry)
                          { entry += randomValue; });

    features1.push_back(point);
}

std::vector<classifier::classes::Data> group;

std::ranges::transform(features1,
                       std::back_inserter(group),
                       [](const auto &feature)
                       { return classifier::classes::Data(
                            feature,
                            5);
                       });

AND_GIVEN("a vector of Data centered at 0.0, 90.0, 90.0")
{
```

```
std::vector<std::vector<float>> features2;
for (int i = 0; i < 10; ++i)
{

    std::vector<float> point = {0.0, 90.0, 90.0};

    std::random_device randomDevice;
    std::mt19937 generator(randomDevice());
    std::uniform_real_distribution<float>
        distribution(-5.0, 5.0);

    float randomValue = distribution(generator);

    std::ranges::for_each(point, [&randomValue](auto &entry)
                          { entry += randomValue; });

    features2.push_back(point);
}

std::ranges::transform(features2,
                       std::back_inserter(group),
                       [](const auto &feature)
                       {
                       return classifier::classes::Data(
                                   feature,
                                   30);
            });

AND_GIVEN("a knn Object")
{

    classifier::strategy::Knn knn(3);
    knn.AddData(group);
```

```
                AND_GIVEN("A Data at 0.0, 85.0, 85.0")
                {
                    classifier::classes::Data
                    dataPoint({0.0, 85.0, 85.0}, 100);

                    WHEN("Classify is called")
                    {
                        const auto result = knn.Classify(dataPoint);

                        THEN("Resulting data is classified to group 2")
                        {
                            classifier::classes::Data
                            dataPointCompare({0.0, 85.0, 85.0}, 30);

                            CHECK(dataPoint == dataPointCompare);
                        }
                    }
                }
            }
        }
    }
}
```

4.2 Experimento

Foram usadas duas estratégias de classificação nesse projeto, o K vizinhos mais próximos (KNN), e o centroide mais próximo. Esses 2 algorítimos funcionam rápido o suficiente para passar no critério de tempo máximo de classificação, e são eficientes, o detalhamento de quão eficiente será tratado mais a frente, com as taxas de acerto do processo final. O critério de tempo é a frequência de leitura de dados de telemetria, 5 milissegundos.

O resultado inicial de classificação de poses foi satisfatório, cada uma das 4 poses foi corretamente identificada pelos algoritmos escolhidos e, essa etapa do experimento evidenciou a necessidade de adicionar uma nova classificação ao sistema.

Considere a 'pose T', citada anteriormente, no contexto de sistemas de captura de movimento, é uma pose válida e muito importante, mas no contexto do boxe, não é uma pose válido, isto é, não tem significado, ou simplesmente, não pertence ao conjunto. O mesmo vale qualquer outra pose não considerada para o sistema, não pertence ao conjunto. A partir disso, foi acrescentado o conceito de pose 'desconhecida'.

Para classificar uma pose como desconhecida, considere novamente os grupos de características, ao calcularmos o centroide de um grupo, podemos considerar esse centroide como representante do grupo, e então escolhemos um valor de limite de classificação, que é a distância euclidiana máxima que um ponto pode ter do centroide para ser classificado como membro do grupo, isto é, definimos um critério extra, além da resposta dos algoritmos de classificação, para classificar um ponto. E pontos que não atendem a esse critério, são classificados para a pose desconhecida.

Foi usado o desvio padrão como limite de classificação. A escolha foi feita pela definição de desvio padrão, representa o desvio médio dos pontos que compõem o grupo.

Com esse ajuste, as 4 poses que se desejava classificar, foram corretamente identificadas com 99% de taxa de acerto. Ao realizar uma das poses propostas, o classificador corretamente mantêm a classificação que espera enquanto se manter a pose. O experimento foi realizado apenas pelo autor desse projeto, não foram realizados mais testes, pois os resultados obtidos com o teste de uma pessoa já foram suficientes para apontar os acertos e falhas do modelo proposto, porém, testes com um grupo maior de indivíduos é crucial para o refinamento do projeto, em etapas futuras.

O ciclo de classificação de poses segue da seguinte forma:

- A IMU é posicionada no pulso esquerdo do atleta.
- A IMU é ligada.
- Inicia-se o streaming de dados.
- Uma pose é realizada
- A IMU transmite informações de ângulos (a cda 5 milissegundos)
- O modulo de telemetria recebe, trata e publica essa informação de angulo no tópico 'imu/angle'.
- O modulo de classificação de poses recebe esse dado e: utiliza o Knn ou o classificador de centroide para classificar **cada dado recebido**
- O modulo de classificação de poses informa a classificação de toda vez que a classificação do ponto atual é diferente da do ponto anterior. E **sempre publica a classificação no tópico 'pose'**
- O modulo de classificação de técnicas recebe todos os dados de pose, e classifica uma tecnica sempre que recebe sequencia de poses: guarda, pose final 'X', como técnica 'X'.

Munido do resultado de classificação de poses, testou-se a hipótese para classificação de movimentos:

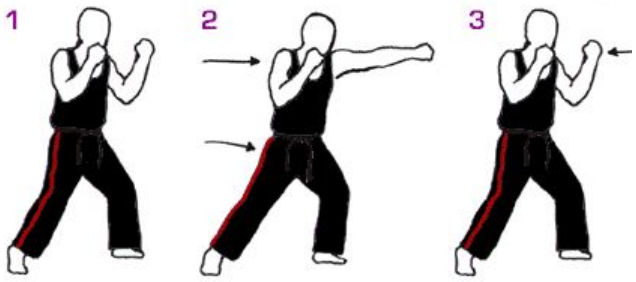
A sequencia de poses: 'guarda', 'desconhecido' e 'final golpe x', corresponde ao golpe x

Isto é, usamos aqui uma abstração simples de o que é um movimento, definido apenas a partir da identificação de poses, não consideramos a trajetória para essa análise.

Temos então a expectativa de classificar um movimento a partir de um padrão como o da figura 9, isto é, queremos identificar a pose 1 como 'guarda', então a pose 2 como 'fim do jab', e considerar esse conjunto como 'jab'.

São 4 técnicas que se deseja classificar, os golpes jab, cruzado e gancho, e o estado de guarda. O conceito de desconhecido está presente para movimentos

Figura 9 – Representação simples de um jab



Fonte:(JONES, 2017)

também, mas assume o nome de 'em movimento', pois espera-se que represente o estado intermediário entre a guarda e o final de um golpe.

O classificador comete dois erros: O jab muitas vezes é classificado como um cruzado, pois a trajetória entre a posição de guarda e a posição final do jab, é muito parecida (tem ângulos próximos) à posição final do cruzado. O gancho é muitas vezes é classificado como a posição de guarda, pois essas posições são semelhantes em termos de ângulos, então um ponto que visualmente diríamos que corresponde à posição final do gancho, pode ainda estar dentro do limite de classificação da guarda.

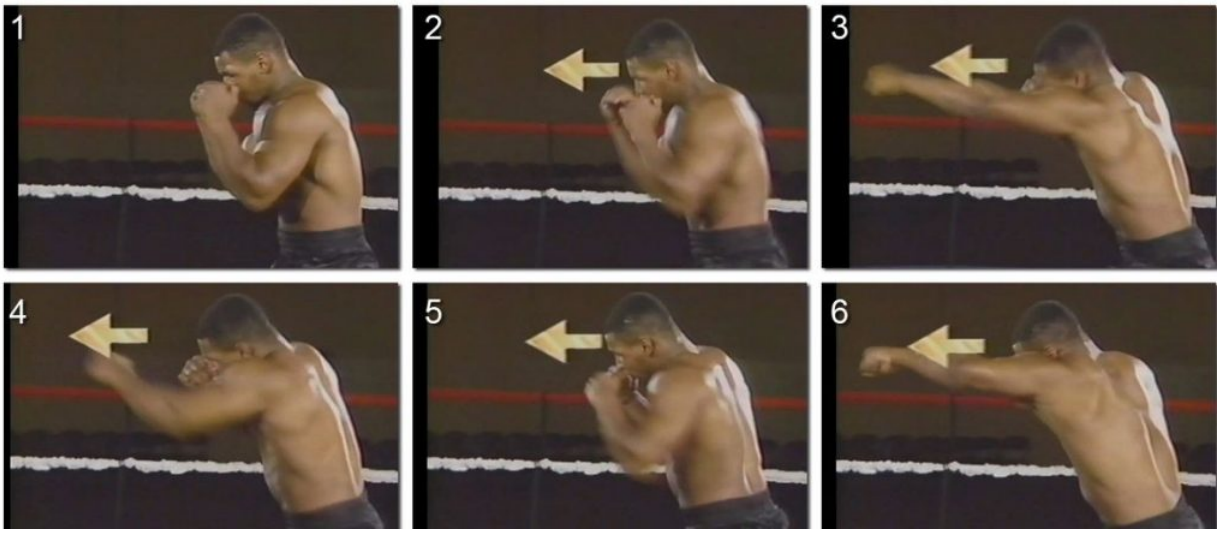
Podemos visualizar o caso de confusão entre cruzado e jab na figura 10: Repare que em um movimento real de jab, o braço passa por uma posição intermediária extremamente parecida com um cruzado, vemos isso no ponto 4 da figura 10, isto é, a expectativa de conseguir classificar uma pose pela passagem da posição de guarda para a posição final do golpe, não é atendida, 1 a 3 na figura, já que posições intermediárias, como em 4, afetam a classificação.

O mesmo caso acontece para classificar as técnicas Gancho e 'em guarda', entre a guarda e o final do Gancho existem posições intermediárias que afetam a classificação.

Para melhorar os resultados, usou-se a leitura de ângulos em formato de quatérnios, que altera a definição, para o sistema, de caraterística para

$$\textbf{Caracteristica} = \boldsymbol{\Theta} = \{\theta_1, \theta_2, \theta_3, \theta_4\}$$

Figura 10 – Visão real de um jab



Fonte: (JONES, 2017)

Tabela 4 – Taxa de acerto de diferentes técnicas.

Técnica	**Taxa de acerto**
Em Guarda	90%
Jab	10%
Cruzado	60%
Gancho	90%

Infelizmente, a nova representação não surtiu efeito, mantendo então o exato mesmo resultado para a classificação técnicas.

5 Etapas futuras e Considerações finais

A método proposto foi eficaz para classificação de poses, já para a classificação de movimentos, o método inicial teve resultados consideráveis, porém apresentava erros frequentes, causados por limitações de teóricas, o que exigiu o refinamento de modelo matemático, o uso de quatérnios, que foi suficiente para melhorar os resultados, mas não para atingir taxas de acerto tão elevadas quanto as obtidas para classificação de poses.

O modelo usado para definir um movimento, identificação de pose inicial e pose final, apenas do pulso, é de fato uma grande simplificação do movimento real. A forma que se poderia avançar os resultados desse projeto, é acrescentar ao método de classificação, informações da trajetória feita durante o movimento, os próprios valos de aceleração linear e velocidade angular, podem ser significativos para a correta identificação de um movimento.

Esse projeto teve sucesso em integrar componentes que eram a primeira vista não relacionados. Vimos a importância da atividade física para construção de saúde, a popularidade dos video jogos, a possibilidade de incentivar a prática esportiva a partir da integração com video jogos, ferramentas para capturar os dados necessários para fazer essa integração e modelos capazes de entender os componentes básicos de uma prática esportiva. Uma vez que temos um ciclo completo de integração dessas ideias, naturalmente, a forma de evoluir esse projeto é voltar ao princípio e refinar cada um de seus componentes, isto é.

- **Teoria de boxe e modelagem:** Os conceitos de boxe usados para o projeto foram simplificados. Existe vasta literatura científica sobre o esporte em questão, e pode-se usar desa literatura para melhorar o modelo de pose e técnica.

- **Telemetria:** Foram usados ângulos em 3 dimensões para classificação, porém, mais dados estão disponíveis a partir da IMU, uma expansão do

módulo de telemetria possibilita o uso de mais informações para classificação.

- **Uso de mais sensores:** Foi usado apenas uma IMU em uma posição, usar mais de uma IMU possibilita estudar o movimento de mais de uma parte do corpo durando a realização de uma técnica, e estudar a relação entre as IMUs, possibilitando aprimorar os modelos e consequentemente os resultados de classificação.

Referências

ADOBE. *What Is Rigging in Animation? Skeletal Animation Explained | Adobe*. 2023. Disponível em: <https://www.adobe.com/uk/creativecloud/animation/discover/rigging.html>. Citado na página 14.

AUTODESK. *Bind pose | Maya LT 2018 | Autodesk Knowledge Network*. 2022. Disponível em: <https://knowledge.autodesk.com/support/maya-lt/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/MayaLT-CharacterAnimation/files/GUID-36808BCC-ACF9-4A9E-B0D8-B8F509FEC0D5-htm.html>. Citado na página 13.

AUTODESK. *3D Modeling Software | Free Trials & Tutorials | Autodesk*. 2023. Disponível em: <https://www.autodesk.com/solutions/3d-modeling-software>. Citado na página 13.

BRITANNICA. *boxing*. 2022. Disponível em: <https://kids.britannica.com/scholars/article/boxing/108498>. Citado na página 9.

CORPORATION, M. *Kinect_sensor Datasheet | Microsoft Corporation - Datasheetspdf.com*. 2014. Disponível em: <https://datasheetspdf.com/datasheet/Kinect_sensor.html>. Citado na página 13.

COSTA, M. *Boxing Techniques and Movements*. 2020. Disponível em: <https://www.dicaseducacaofisica.info/pt-pt/tecnicas-movimentos-boxe/>. Citado 2 vezes nas páginas 61 e 62.

IBGE, p. *Sistema IBGE de Recuperação Automática - SIDRA*. 2019. Disponível em: <https://sidra.ibge.gov.br/pesquisa/pns/pns-2019>. Citado na página 9.

INNOVATION2YOU. *A MAIORIA DOS MALEFÍCIOS DO SEDENTARISMO SÃO VELHOS CONHECIDOS*. 2021. Disponível em: <https://socesp.org.br/publico/noticias/area-medica-/a-maioria-dos-maleficios-do-sedentarismo-sao-velhos-conhecidos/>. Citado na página 9.

JONES, L. *Boxing Jab Technique For Beginners | RDX Sports*. 2017. Disponível em: <https://blogs.rdxsports.com/boxing-jab-technique/>. Citado 2 vezes nas páginas 68 e 69.

KEMPE, V. *Inertial MEMS: principles and practice*. [S.l.]: Cambridge University Press, 2011. Citado 2 vezes nas páginas 14 e 15.

LABS, Y. 3 space sensor. 2017. Disponível em: <https://yostlabs.com/wp-content/uploads/pdf/3-Space-Sensor-Users-Manual-1.pdf>. Citado na página 15.

MACEDO, C. d. S. G. et al. Benefícios do exercício físico para a qualidade de vida. *Revista Brasileira de Atividade Física & Saúde*, v. 8, n. 2, p. 19–27, 2003. Citado na página 9.

NGUYEN, J. *Como lancar um jab*. 2012. Disponível em: <https://www.expertboxing.com.br/tecnicas-do-boxe/como-lancar-um-jab>. Citado na página 61.

PLAYSTATION, S. *DualSense Register*. 2020. Disponível em: <https://www.playstation.com/content/dam/global_pdc/en/corporate/support/manuals/accessories/ps5-accessories/dualsense-wireless-controller-cfi-zct1w/PT_AMER_DualSense_Wireless_Controller_Instruction_Man_Web.pdf>. Citado 3 vezes nas páginas 11, 12 e 13.

SCHMIDT, R. A.; LEE, T. D. Motor control and learning: A behavioral emphasis. *Human Kinetics*, 2019. Disponível em: <https://us.humankinetics.com/products/motor-control-and-learning-6th-edition-with-web-resource>. Citado na página 25.

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson Prentice Hall, 2011. 529 p. ISBN 8579361087, 9788579361081. Citado na página 24.

STOCK, A. *Xbox Wireless Controller | Xbox*. 2020. Disponível em: <https://stock.adobe.com/br/search/images?k=t-pose>. Citado na página 14.

TIPOS de esportes: conheça as modalidades esportivas – UniToledo. 2021. Disponível em: <https://unitoledo.br/tipos-de-esportes/>. Citado na página 9.

WULF, G. Posture and movement: Fundamentals of human movement science. *Journal of Human Kinetics*, v. 53, p. 5–18, 2016. Disponível em: <https://content.sciendo.com/view/journals/hukin/hukin-overview.xml>. Citado na página 25.

XBOX. *Xbox Wireless Controller | Xbox*. 2020. Disponível em: <https://www.xbox.com/en-US/accessories/controllers/xbox-wireless-controller>. Citado na página 13.

Apêndices

APÊNDICE A – Repositório do projeto

https://github.com/ArielSixwings/ImuBoxing