UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

PEDRO HENRIQUE BUENO XAVIER E SILVA

REABILITAÇÃO ROBÓTICA: DESENVOLVIMENTO DE UM DISPOSITIVO INTERATIVO PARA PACIENTES COM DEFICIÊNCIAS NEUROMOTORAS

SÃO CARLOS
2023

PEDRO HENRIQUE BUENO XAVIER E SILVA

REABILITAÇÃO ROBÓTICA: DESENVOLVIMENTO DE UM DISPOSITIVO INTERATIVO PARA PACIENTES COM DEFICIÊNCIAS NEUROMOTORAS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Mecânica da Universidade Federal de São Carlos, para obtenção do título de Bacharel em Engenharia Mecânica.

Orientador: Prof. Leonardo Marquez Pedro

SÃO CARLOS
2023

Esta folha deve ser substituída pela folha de aprovação digitalizada

AGRADECIMENTOS

Primeiramente, expresso minha eterna gratidão aos meus pais, Donizeth e Valda. Suas forças e ensinamentos foram o alicerce que sustentou todas as minhas decisões, desafios e conquistas. Agradeço por sempre terem sido minha torre de suporte e incentivo incondicional em todas as etapas da minha vida.

Aos meus irmãos, Daniel e Mariana, devo mais do que palavras podem expressar. Vocês foram, e continuarão sendo, minhas principais referências de caráter, força e determinação. Vocês sempre iluminaram meu caminho, fazendo-me almejar ser alguém melhor a cada dia.

Dirijo um agradecimento especial ao grupo de docentes do DEMEC pela orientação, conhecimento e experiência que compartilharam ao longo da minha trajetória acadêmica. Estendo meus agradecimentos ao Centro Acadêmico, Baja, Tênis UFSCar, e, com um carinho especial, à Atlética UFSCar. As experiências, aprendizados e momentos vividos nestes grupos foram cruciais para minha formação pessoal e profissional.

Por fim, mas não menos importante, sou imensamente grato a todos os amigos que fiz ao longo dos anos na universidade. Alguns de vocês se tornaram mais do que amigos, tornaram-se família. Aos que serão meus amigos para sempre, levo no coração a certeza de que os laços construídos durante esta fase serão eternos e indestrutíveis. Juntos, vivenciamos sonhos, superamos desafios e compartilhamos alegrias incontáveis.

A todos vocês, meu mais profundo e sincero agradecimento.

RESUMO

Este trabalho apresenta o desenvolvimento de um dispositivo eletrônico interativo para a reabilitação de membros superiores de pacientes com sequelas decorrentes de doenças neurológicas, como acidente vascular cerebral, doença de Parkinson e paralisia cerebral. Diante da crescente incidência dessas doenças no Brasil e das consequências debilitantes associadas a elas, a necessidade de sistemas de reabilitação eficazes tornou-se premente. O dispositivo utiliza um sensor de movimento, em particular um acelerômetro, para monitorar movimentos de transladação em três eixos. Testes iniciais confirmaram a robustez e precisão do dispositivo. Além disso, experimentos práticos demonstraram a aplicabilidade do sensor em cenários de atividades cotidianas, como a movimentação de um copo sobre uma mesa. Os resultados reforçam o potencial do dispositivo em auxiliar na reabilitação neurológica, promovendo melhorias no controle motor dos pacientes. Apesar do progresso promissor, mais investigações são necessárias para maximizar o potencial do dispositivo. Além do desenvolvimento técnico, este estudo contribui para a literatura ao oferecer insights sobre a integração de tecnologia e terapia em contextos clínicos.

Palavras-chave: Reabilitação de membros superiores. Doenças neurológicas. Dispositivo interativo.

ABSTRACT

This study introduces the development of an interactive electronic device aimed at rehabilitating the upper limbs of patients suffering from sequels of neurological diseases such as stroke, Parkinson's disease, and cerebral palsy. Given the rising incidence of these diseases in Brazil and the debilitating outcomes associated with them, the need for effective rehabilitation systems has become pressing. The device employs a motion sensor, specifically an accelerometer, to monitor translational movements in three axes. Initial tests confirmed the device's robustness and accuracy. Furthermore, practical experiments showcased the sensor's applicability in everyday activity scenarios, such as moving a cup across a table. The findings underscore the device's potential in aiding neurological rehabilitation, enhancing patient motor control. Despite promising progress, further inquiries are needed to fully harness the device's potential. Beyond the technical development, this research contributes to the literature by offering insights into the integration of technology and therapy in clinical settings.

Keywords: Upper limb rehabilitation. Neurological diseases. Interactive device.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Descrição

Seguindo um padrão mundial, a pirâmide etária brasileira sofre mudanças no que se refere ao perfil da população, onde a expectativa de vida do brasileiro deve atingir 81 anos em 2050 (IBGE, 2018). Tal fato positivo se torna preocupante, em decorrência de uma maior presença de comorbidades relacionadas ao envelhecimento presentes nessa população (DAMATA et al., 2016). Além disso, alguns problemas de saúde, como o acidente vascular encefálico (AVE), a doença de Parkinson (DP) e a paralisia cerebral (PC) podem prejudicar a utilização dos membros superiores e reduzir a participação do indivíduo nas atividades de vida diária (AVD), as quais requerem um amplo espectro de habilidades manuais, tais como realização de cuidados pessoais, alimentação, atividades profissionais dentre outras (MORENO et al., 2023).

O AVE é apontado como uma das comorbidades mais prevalentes da atualidade, sendo considerado uma das principais causas de lesões no membro superior (ONAOLAPO; ONAOLAPO; NATHANIEL, 2019). O Brasil é o país da América Latina onde a doença fica em primeiro lugar como a maior causadora de morte e incapacitação (ALVES; SANTANA; AOYAMA, 2020). No entanto, a promoção e a prevenção da saúde são estratégias primárias de cuidados que visam reduzir as morbimortalidades relacionadas ao AVE, a fim de evitar a doença.

A DP é uma patologia de origem neurológica, sendo considerada a segunda enfermidade neurodegenerativa mais comum na população idosa, manifestando-se de forma crônica e progressiva, em decorrência da diminuição do neurotransmissor dopamina nos gânglios da base (CHOU, 2020). As perdas cognitivas na DP são fatores que comprometem significativamente os domínios de responsabilidade como déficit de memória operacional, além da acentuada redução do desempenho de funções executivas (BALESTRINO; SCHAPIRA, 2019). O declínio no desempenho motor afeta principalmente a prática de atividades cotidianas desses pacientes, de forma a limitar ou até mesmo impedir tais ações, causando comprometimento na qualidade de vida do paciente (MALAK et al., 2017).

Por outro lado, a PC é o termo utilizado para descrever os distúrbios que envolve déficits motores frequentemente relacionados a disfunções sensoriais e cognitivas, resultante de uma lesão não progressiva (COSTA; SANTOS, 2021). Essa lesão acaba desencadeando alterações no crescimento do encéfalo, a qual acomete o desenvolvimento e maturação estrutural e funcional do sistema nervoso central, podendo ocorrer no período pré-natal, perinatal ou pós-natal, acometendo membros e o tronco do indivíduo, sendo classificada topograficamente de maneira individual (ELAD et al., 2018).

A reabilitação é essencial para pacientes com essas condições neurológicas, pois

uma intervenção bem-sucedida pode melhorar significativamente a qualidade de vida e promover maior independência. Ao aprimorar habilidades motoras, cognitivas e sensoriais, a reabilitação permite aos pacientes enfrentar os desafios do dia a dia com maior autoconfiança e autonomia.

Com o intuito de favorecer a utilização dos membros, minimizar sequelas e ampliar o grau da participação de pacientes nas AVD e no contexto social, várias abordagens e recursos terapêuticos estão disponíveis, porém a maneira de avaliar resultados de processos de reabilitação muitas vezes é subjetiva e podem variar de um profissional da área para outro, pois poucas medidas quantitativas estão disponíveis, apresentando desta forma poucos resultados para determinar o grau de evolução do paciente (ANG et al., 2014).

Além disso, a literatura aponta que a inserção de sistemas robóticos voltados para a reabilitação também podem ser utilizados como auxílio para diagnóstico e tratamento de tais doenças, com base no segmento do membro superior que recebe a terapia, ombro, cotovelo, punho e mão, ou uma combinação de vários desses segmentos, atingindo até mesmo o membro superior como um todo (POLI et al., 2013).

Os dispositivos robóticos possibilitam a realização de tarefas específicas repetidas vezes, de forma controlada e confiável, o que tem sido demonstrado na literatura como fator determinante para a facilitação da reorganização cortical, com concomitante aumento da habilidade motora e melhora do desempenho das atividades funcionais (CARDOSO, 2020). A terapia assistida por robô possibilita aos pacientes realizar o treinamento de forma mais independente, ou seja, prática intensiva com supervisão mínima pelo terapeuta (POLI et al., 2013). Além disso, permite que o profissional programe um protocolo de reabilitação para que o paciente o execute em múltiplas sessões no dia.

Por outro lado, os equipamentos disponíveis atualmente apresentam algumas desvantagens, limitando sua aplicação e restringindo a sua utilização em larga escala. Essas desvantagens envolvem principalmente o alto custo, impossibilitando a adesão ao equipamento, ao manejo hospital adequado em quesitos de higienização, além dos robôs serem importados, o que dificulta a veiculação do dispositivo no Brasil.

Outra desvantagem significativa dos equipamentos robóticos atuais é a falta de acesso em áreas remotas ou carentes de recursos. Isso limita ainda mais a possibilidade de um tratamento abrangente e eficaz para pacientes nessas regiões, destacando a necessidade de soluções mais acessíveis e de fácil implementação. Com isso, há a necessidade de criar sistemas de avaliações mais acessíveis, para que tornar o diagnóstico e o tratamento de doenças como AVC mais fácil e humanizado.

Diante do exposto, o presente trabalho buscou desenvolver um dispositivo, inspirado no modelo proposto por (ROCHA et al., 2016) com o objetivo de auxiliar no diagnóstico e reabilitação de pacientes, em relação à habilidade de realizar tarefas funcionais que envolvem a manipulação de objetos. Pretende-se com esse dispositivo interativo proporcionar uma ferramenta eficaz para os pacientes praticarem e aprimorarem suas

habilidades motoras, contribuindo assim para uma avaliação mais completa e eficaz.

O dispositivo proposto funciona como uma plataforma interativa, onde os pacientes poderão praticar movimentos específicos relacionados às habilidades motoras dos membros superiores. Isso será alcançado por meio de tarefas e desafios que simulam atividades da vida diária. O dispositivo foi equipado com sensores que permitem monitorar e registrar o desempenho do paciente, fornecendo informações importantes sobre sua evolução ao longo do tempo.

1.2 Objetivos

O objetivo geral deste trabalho foi desenvolver um dispositivo eletrônico interativo destinado à reabilitação de membros superiores em pacientes com sequelas decorrentes de doenças neurológicas, como acidente vascular cerebral, doença de Parkinson e paralisia cerebral. O foco principal de seu desenvolvimento foi pautado em sua concepção e na construção de uma ferramenta que possa potencialmente facilitar o processo de reabilitação e, no futuro, contribuir para melhorar a qualidade de vida desses pacientes.

Para que esse objetivo seja atingido, os seguintes objetivos específicos foram propostos:

- Projeto e desenvolvimento de um dispositivo que seja fácil de utilizar e possa ser usado em diferentes ambientes de reabilitação, como hospitais, clínicas de fisioterapia e até mesmo em casa.
- Conduzir testes preliminares do dispositivo em condições controladas, com o objetivo de garantir sua funcionalidade e eficácia potencial na reabilitação dos membros superiores.
- Projeto e implementação de uma interface que seja intuitiva para facilitar o uso do dispositivo, considerando que o dispositivo será operado por profissionais de saúde e potencialmente pelos próprios pacientes.

1.3 Estrutura do Texto

Esta monografia está organizada da seguinte forma:

- O Capítulo 2 apresenta os conceitos gerais e a fundamentação teórica sobre doenças neurológicas crônicas e suas principais implicações, apresentando também possíveis reabilitações para membros superiores, terapias convencionais e o uso de dispositivos eletrônicos e robóticos;
- O Capítulo 3 aborda os materiais e métodos utilizados para o desenvolvimento do dispositivo proposto, descrevendo em detalhe as etapas de seu processo de construção e como as diferentes tecnologias foram integradas para a sua efetiva funcionalidade. Este capítulo também apresenta as motivações por trás das escolhas tecnológicas

feitas, o papel de cada componente no dispositivo como um todo e como eles interagem entre si para produzir o resultado final, descrito no Capítulo 4.

- O Capítulo 5 oferece a discussão dos resultados obtidos e o Capítulo 6 as conclusões finais do trabalho, sintetizando as principais descobertas e realçando a relevância e as implicações destas para a área de reabilitação neurológica. Este capítulo também revisita os objetivos iniciais do estudo, avaliando em que medida foram atingidos e oferecendo uma análise crítica sobre as limitações do trabalho. Finalmente, propõe futuras direções de pesquisa e possíveis melhorias no dispositivo, com base nas experiências e aprendizados adquiridos ao longo do desenvolvimento deste projeto.

2 REVISÃO BIBLIOGRÁFICA

2.1 Doenças neurológicas e suas implicações

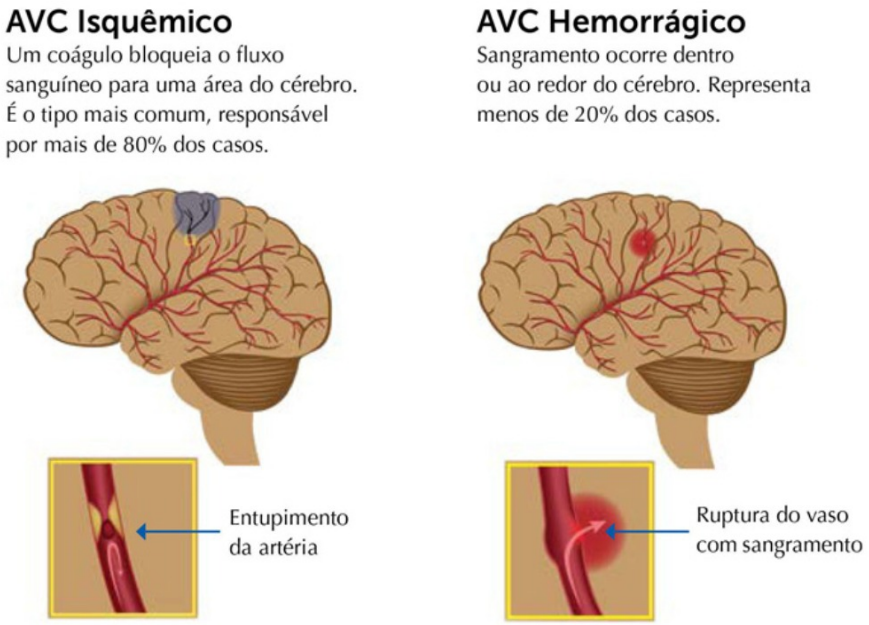
As doenças neurológicas afetam mais de 1,8 milhão de pessoas na América Latina e o número deve subir para 9,1 milhões até 2040, índices que se aproximam da América do Norte, que possui cerca de 9,2 milhões de ocorrências (PRINCE et al., 2013). Essas condições, que não podem ser curadas, podem levar à paralisia parcial ou completa e ao comprometimento da função cognitiva, pois as células cerebrais são danificadas ou destruídas devido a condições como a demência (CEDARO et al., 2020).

A importância de descobrir novos tratamentos para reduzir ou adiar os sintomas de doenças neurodegenerativas é imensurável, pois as implicações dessas doenças podem causar impactos econômicos e sociais incapacitantes para os pacientes e familiares (SCHRODER; ARAUJO; IGNáCIO, 2020). Dentre elas, pode-se destacar o Acidente Vascular Encefálico (AVE), Doença de Parkinson (DP) e a Paralisia Cerebral (PA).

2.1.1 Acidente Vascular Encefálico (AVE)

O Acidente Vascular Encefálico (AVE) é uma condição clínica caracterizada pela interrupção súbita do fluxo sanguíneo para o cérebro, podendo ser classificado em duas categorias principais: isquêmico e hemorrágico. Independentemente do tipo, o AVE pode causar danos cerebrais irreversíveis e, em muitos casos, resultar em incapacidades motoras e cognitivas. Segundo (ALMEIDA, 2012), a taxa de mortalidade causada pelo AVE varia entre 0,05%, sendo 35% dos 99.174 óbitos ocorrendo com pacientes com mais de 80 anos. Além disso, a alta incidência do AVE é frequentemente associada a fatores ligados ao envelhecimento populacional e a própria transição epidemiológica (ALMEIDA, 2012). A Figura abaixo ilustra as diferenças entre o AVE isquêmico e hemorrágico.

Figura 1 – Tipos de AVE



Fonte: Chester(2003).

Apesar de se representar como uma doenças de alto risco e prevalência, o AVE pode ser prevenível adotando mudanças de hábitos de vida mais saudáveis, visto que seus fatores de risco incluem a Hipertensão Arterial Sistêmica (HAS), Diabetes Mellitus (DM), hipercolesterolemia, tabagismo e sedentarismo (RIBEIRO et al., 2018). Outra forma de prevenção se dá através de exercícios físicos, buscando a prevenção da recorrência da doença e exercendo influência direta, seja na melhora após a exposição da doença diante de sua reabilitação ou na redução de seus fatores de risco .

Atualmente, o AVE se tornou a principal causa de morte no Brasil, sendo responsável por aproximadamente 32% das mortes por doenças do aparelho circulatório, apresentando incidência de 81,7 a 180 casos por 100 mil habitantes. A Sociedade Brasileira de Doenças Cerebrovasculares ressalta ainda, que cerca de 70% da população acometida não conseguem retornar ao trabalho devido a sequelas da doença, tendo 50% da população também acometida dependentes em suas atividades de vida diária (AVD) em decorrência de seus efeitos colaterais (CEREBROVASCULARES, 2023). Por meio de seus efeitos negativos, o AVE pode ser considerado como um grande problema de saúde pública, em decorrência de suas consequências físicas, econômicas e sociais para seus pacientes(CARVALHO et al., 2014).

2.2 Reabilitação de membros superiores

A reabilitação de membros superiores é essencial para melhorar a funcionalidade e a qualidade de vida dos pacientes com condições neurológicas que afetam a mobilidade e a força desses membros . As terapias de reabilitação visam recuperar ou melhorar

as habilidades motoras, a coordenação, a amplitude de movimento e a força muscular, permitindo que os pacientes realizem atividades diárias com maior independência.

No estudo realizado por (LAMERS et al., 2016), foi constatado que a terapia assistida por robôs para membros superiores pode contribuir para o aumento da recuperação motora em pacientes no período subagudo após um acidente vascular cerebral (AVC). No ensaio clínico randomizado realizado pelo grupo, 53 pacientes com AVC foram submetidos a terapia padrão e um grupo adicional recebeu 30 sessões de terapia assistida por robôs. Melhorias significativas foram encontradas no grupo experimental em relação à escala de Ashworth modificada para ombro e cotovelo e à amplitude total de movimento passivo para ombro/cotovelo. Ambos os grupos, experimental e controle, apresentaram melhorias significativas na escala de avaliação de Fugl-Meyer e no Índice de Motricidade, com uma maior melhora observada no grupo experimental.

2.2.1 Terapias convencionais

As terapias convencionais de reabilitação de membros superiores incluem exercícios de fortalecimento e alongamento, treinamento de coordenação e equilíbrio, terapia ocupacional e fisioterapia . Estas terapias são geralmente realizadas sob a supervisão de um terapeuta especializado e envolvem a realização de exercícios e atividades específicas para melhorar a função dos membros afetados.

Os terapeutas também podem empregar técnicas como a estimulação elétrica neuromuscular, a terapia espelho e a terapia de restrição do movimento induzida (CIMT) para ajudar na recuperação dos pacientes . A escolha das terapias e a abordagem utilizada são adaptadas às necessidades específicas de cada paciente e à gravidade das suas condições.

2.2.2 Uso de dispositivos eletrônicos e robóticos

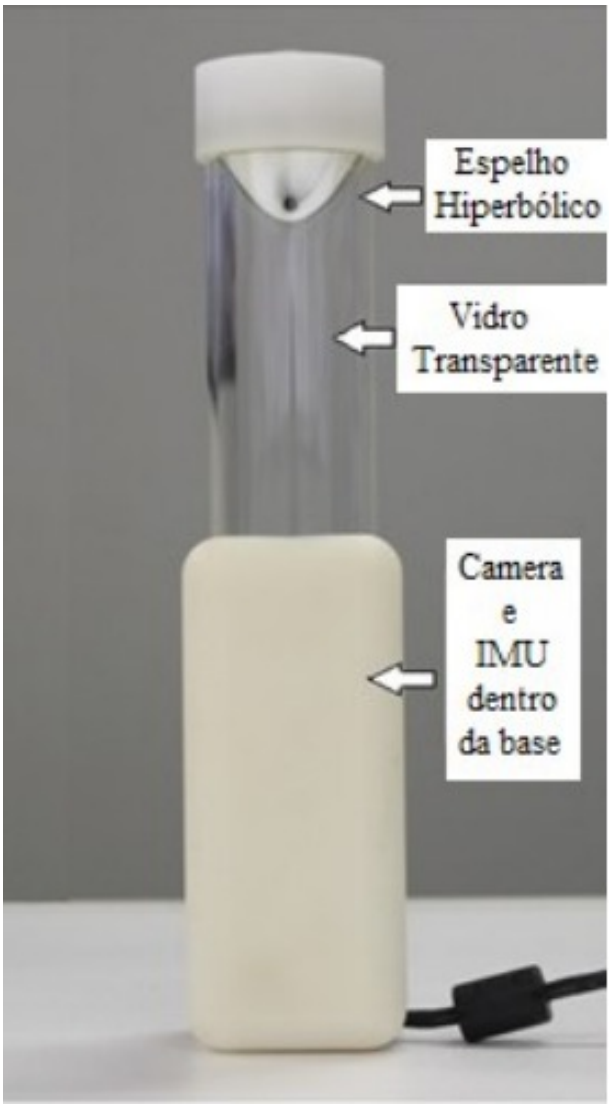
Nos últimos anos, dispositivos eletrônicos e robóticos têm sido cada vez mais utilizados na reabilitação de membros superiores. Esses dispositivos podem auxiliar os pacientes a realizar exercícios e movimentos específicos, proporcionando suporte e resistência adequados, além de monitorar o progresso do paciente ao longo do tempo.

De acordo com uma revisão sistemática sobre reabilitação do membro superior em pessoas com EM, diferentes tipos de estratégias de reabilitação podem melhorar a função do membro superior em pacientes com EM. Isso inclui abordagens multidisciplinares e reabilitação baseada em robôs, bem como treinamento de força e resistência (SALE et al., 2014). No entanto, o conteúdo e a dosagem da terapia podem variar bastante entre os diferentes estudos, e mais pesquisas são necessárias para comparar diretamente os efeitos das diferentes estratégias de reabilitação e para investigar a dosagem ideal de terapia de acordo com o nível de incapacidade do membro superior.

2.3 Sensor de Preensão e Movimentos da Mão (SPMM)

O desenvolvimento de dispositivos de assistência à reabilitação neurológica, especialmente na área de controle motor, tem sido um campo de pesquisa em rápida expansão nos últimos anos. Este crescimento é impulsionado tanto pela necessidade de tratamentos mais eficazes para as condições neurológicas crescentes quanto pelo avanço contínuo nas tecnologias de sensores e computação. A Figura 2 ilustra um protótipo do SPMM utilizado no estudo de Rocha et al., 2016 e replicado no estudo de Appel (2016).

Figura 2 – Protótipo do SPMM



Fonte: (ROCHA et al., 2016).

O Sensor de Preensão e Movimentos da Mão (SPMM) é um exemplo de tal tecnologia. O SPMM é projetado para capturar a funcionalidade da mão e o desempenho de preensão, fornecendo dados valiosos para avaliação e reabilitação em casos de danos neurais. Para entender o contexto do SPMM e sua relevância, é essencial examinar os

avanços recentes em sensores para a avaliação da função motora, bem como a importância da reabilitação da mão em pacientes neurológicos.

Porém, Appel (2016) ressalta que alguns pontos precisam ser considerados. Como o protótipo SPMM foi utilizado em crianças, foi observado que a faixa etária de 3 anos de idade precisou utilizar ambas as mãos para executar o protocolo de beber água, em decorrência do peso do dispositivo, visto que o mesmo pesa 0,550kg. Além disso, os dados do equipamento foram enviados via USB, onde o cabo poseria ser substituído caso fosse conectado via wireless.

2.4 Hardware

2.4.1 ESP32

O ESP32 é um microcontrolador fabricado pela Espressif Systems. É especialmente popular na comunidade de desenvolvedores de Internet das Coisas (IoT) devido ao seu baixo custo, alta capacidade de processamento e conectividade Wi-Fi e Bluetooth integrada (CARDUCCI, 2019). Seu ambiente de desenvolvimento é o ESP-IDF, mas também suporta a linguagem de programação MicroPython, o que permite a prototipagem rápida e o desenvolvimento de software.

O ESP32 possui alta capacidade de comunicação e processamento. Ele permite que os dados do sensor MPU6050 sejam lidos em tempo real e transmitidos para o PC via conexão serial. Além disso, a capacidade do ESP32 de executar MicroPython oferece a oportunidade de usar uma linguagem de programação de alto nível, o que facilita o desenvolvimento e a depuração do software.

2.4.2 MPU6050

O MPU6050, como mencionado anteriormente, é um sensor de movimento que combina um acelerômetro e um giroscópio em um único chip, permitindo a medição da aceleração linear e da velocidade angular (NAVYA, 2018). É uma ferramenta ideal para este projeto, pois oferece uma maneira precisa e acessível de detectar o movimento e a orientação. Além disso, sua compatibilidade com o ESP32 através da comunicação I2C simplifica a integração dos componentes de hardware do projeto.

2.5 Software

2.5.1 MicroPython

O MicroPython é uma implementação compacta e eficiente da linguagem de programação Python 3, otimizada para rodar em microcontroladores como o ESP32 (RODRíGUEZ-PONCE, 2019). Ele oferece uma interface de programação familiar para

desenvolvedores acostumados com Python e facilita a prototipagem rápida e o desenvolvimento de software para microcontroladores.

Escolhemos o MicroPython para este projeto porque ele oferece uma maneira eficiente e intuitiva de programar o ESP32. Além disso, o MicroPython simplifica tarefas complexas, como a leitura de dados do sensor MPU6050 e a comunicação serial, tornando o desenvolvimento do software mais eficiente e acessível.

2.5.2 Python e PySerial

Python é uma linguagem de programação de alto nível muito popular por sua facilidade de uso e versatilidade. PySerial é uma biblioteca Python que permite a comunicação serial entre um PC e dispositivos externos, como o ESP32 (PéREZ, 2011).

Neste projeto, Python e PySerial são usados para ler os dados do sensor MPU6050 que são transmitidos pelo ESP32. A escolha do Python permite uma implementação rápida e eficiente, enquanto PySerial oferece uma maneira confiável de comunicar com o ESP32.

3 MATERIAIS E MÉTODOS

3.1 Seleção de componentes e materiais

Nesta seção, são apresentados os componentes e materiais escolhidos para o desenvolvimento do dispositivo, explicando as razões por trás de cada escolha e como eles se integram ao projeto. No caso deste projeto, a tarefa era criar um sistema de sensoriamento de movimento com baixa latência usando o módulo ESP32 e o sensor MPU6050. Além disso, uma interface de usuário no PC foi desenvolvida para exibir e armazenar os dados em tempo real.

3.1.1 Módulo ESP32

Este microcontrolador foi escolhido devido à sua capacidade de executar o MicroPython, um subconjunto compacto do Python que é ideal para microcontroladores. O ESP32 é capaz de operar com baixo consumo de energia e tem capacidade suficiente para lidar com operações de sensoriamento de movimento.

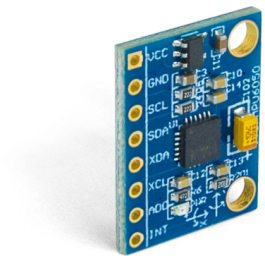
Figura 3 – Módulo ESP32

Fonte: Robocore(2023).

3.1.2 Sensor MPU6050

O MPU6050 é um sensor popular e acessível que combina um acelerômetro e um giroscópio. Ele é capaz de medir a aceleração linear ao longo de três eixos (X, Y, Z) e a velocidade de rotação (também em torno de X, Y, Z), tornando-o ideal para o sensoriamento de movimento.

Figura 4 – Sensor MPU6050



Fonte: Robocore(2023).

3.1.3 Computador

Para este projeto, um notebook com as seguintes características foi utilizado:

- Sistema operacional: Linux mint
- Processador: i7-7700
- Memoria ram: 16Gb

3.1.4 Cabo micro USB

O cabo micro USB foi usado para conectar o módulo ESP32 ao computador, permitindo que os dados do sensor sejam transmitidos.

3.1.5 Bibliotecas Python

Neste projeto, utilizamos duas bibliotecas Python importantes: Pygame e PySerial.

PySerial é uma biblioteca Python que permite a comunicação serial entre o computador e o módulo ESP32. Ela é usada neste projeto para transmitir os dados do sensor MPU6050 do microcontrolador para o computador. Isso permite que os dados do sensor sejam processados e visualizados em tempo real na interface do usuário.

Pygame, por outro lado, é uma biblioteca para desenvolvimento de jogos e aplicações multimídia. Ela fornece uma série de funcionalidades que tornam mais fácil criar interfaces gráficas interativas. No nosso caso, a Pygame é usada para criar uma interface que visualiza os dados do sensor em tempo real e permite ao usuário interagir com esses dados.

3.2 Metodologia utilizada

Uma vez que os materiais foram selecionados, a etapa seguinte foi definir a metodologia adequada para alcançar o objetivo do projeto. Este passo é vital para garantir

que os componentes individuais funcionem em conjunto de maneira eficaz.

O projeto foi dividido em três partes: a configuração do ambiente, a coleta de dados pelo ESP32 e criação de uma interface de usuário no PC.

3.2.1 Configuração do ambiente

Utilizar o MicroPython é uma excelente maneira de explorar ao máximo as potencialidades da placa ESP32. Este chip oferece uma ótima plataforma para a implementação do MicroPython.

O primeiro passo é baixar o arquivo .bin mais recente do firmware MicroPython para carregar em seu dispositivo ESP32. Este arquivo pode ser obtido na página de downloads do MicroPython.

Atualmente, o único suporte para copiar o firmware é o esptool.py. Você pode instalá-la usando pip:

```
pip install esptool
esptool.py --port /dev/ttyUSB0 erase_flash
esptool.py --chip esp32 --port /dev/ttyUSB0 write_flash -z 0x1000
    esp32-20180511-v1.9.4.bin
```

Usando o esptool.py, você pode apagar o flash com o comando acima e então implementar o novo firmware. Se os comandos acima forem executados sem erros, então o MicroPython deve estar instalado em sua placa!

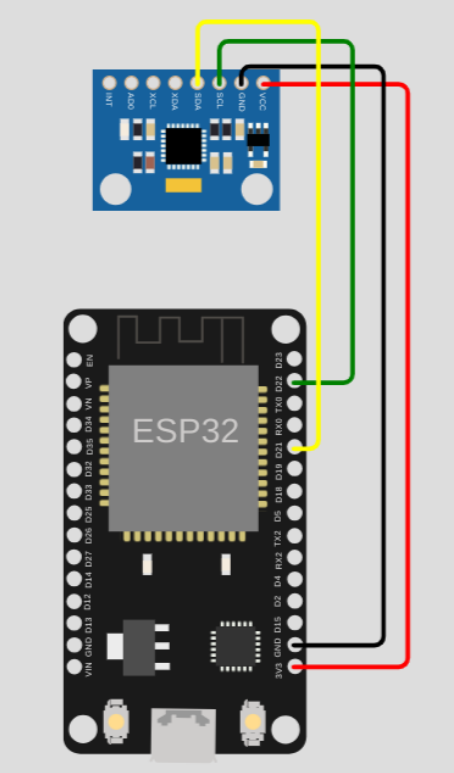
Para se conectar a placa esp32 via terminal, você precisa utilizar um programa de emulação de terminal. Por exemplo, no Linux, você pode tentar executar:

```
picocom /dev/ttyUSB0 -b115200
```

3.2.2 Configuração dos hardwares

O hardware para este projeto consiste no módulo ESP32 e no sensor MPU6050. Estes componentes estão interconectados através de uma conexão I2C, que permite a transmissão de dados entre o sensor e o microcontrolador. Uma imagem representando a configuração física dos componentes é apresentada a seguir:

Figura 5 – Conexões dos hardwares



Fonte: Autor.

A linguagem MicroPython foi utilizada na etapa de coleta de dados no ESP32 para acessar e ler os dados do sensor MPU6050. Em seguida, esses dados foram empacotados e transmitidos pela porta serial para o computador. Para ilustrar esse procedimento, apresentamos um pseudocódigo que exemplifica a operação:

```
1. Importar as bibliotecas necessrias
2. Definir os pinos para a comunicao I2C
3. Definir o barramento no qual o dispositivo est conectado
4. Criar uma instncia do sensor MPU6050

Procedimento "run":
5. Iniciar um loop infinito:
    5.1 Tentar ler e imprimir os dados do sensor
    5.2 Se ocorrer um erro, imprimir o erro e continuar o loop

6. Se este script  o programa principal, executar o procedimento "run"
```

No pseudocódigo acima, as bibliotecas necessárias são inicialmente importadas. A classe Pin da biblioteca 'machine' é utilizada para configurar os pinos no ESP32 que são usados para a comunicação I2C. A classe MPU6050 é importada do módulo esp32.mpu6050,

que contém a implementação dos métodos necessários para interagir com o sensor MPU6050.

O procedimento ”run”é definido para iniciar um loop infinito que constantemente tenta ler e imprimir os dados do sensor MPU6050. Caso ocorra uma exceção durante a leitura ou impressão dos dados, essa exceção é capturada e sua mensagem é impressa na tela, mas o loop continua a executar. Isso garante que falhas temporárias na leitura de dados do sensor não interrompam a coleta contínua de dados.

Assim, os dados do sensor MPU6050 são continuamente lidos e impressos no console, estando prontos para serem coletados pelo sistema no PC.

3.2.2.1 Leitura dos dados do dispositivo

A leitura dos dados do sensor MPU6050 foi realizada utilizando um script Python personalizado que se comunica com o sensor via uma porta serial. O script gerencia a comunicação serial e processa os dados do acelerômetro e giroscópio. O processo de coleta de dados é simplificado em um pseudocódigo abaixo para uma melhor compreensão:

```
Classe CustomSerial:
  Inicializar:
    Estabelecer conexo serial
    Iniciar buffers para acelermetro e giroscpio
    Definir status de  gravao  como False
    Definir o tempo de incio como nulo

    Funo   start_recording:
    Definir status de  gravao  como True
    Marcar o tempo de incio

    Funo   stop_recording:
    Definir status de  gravao  como False
    Limpar o tempo de incio

    Funo   get_raw_data:
    Se dados disponveis na porta serial:
      Ler e decodificar linha
      Se tipo de sensor est na linha:
        Extrair e converter valores dos dados para float
        Retornar lista de valores dos dados

    Funo   get_data:
    Obter dados brutos com get_raw_data
    Calcular a mdia  mvel
    Atualizar buffers com dados brutos mais recentes
```

```
Funo  save_data:
Se  gravao  est  ativa:
  Obter dados brutos e mdia mvel de ambos os sensores
  Calcular o tempo decorrido desde o incio da gravao
  Adicionar todos os valores  lista all_sensor_data

Funo  download_csv:
Criar um nome de arquivo nico
Escrever cabealho do arquivo
Gravar cada linha de dados na lista all_sensor_data no arquivo
```

Cada uma dessas etapas foi encapsulada dentro de funções para facilitar a manipulação dos dados. O script se iniciou estabelecendo uma conexão serial, e então procede para coletar, processar e armazenar os dados do sensor. Durante a coleta de dados, foi possível iniciar e parar a gravação conforme necessário, e os dados coletados foram salvos em um arquivo CSV para análise posterior.

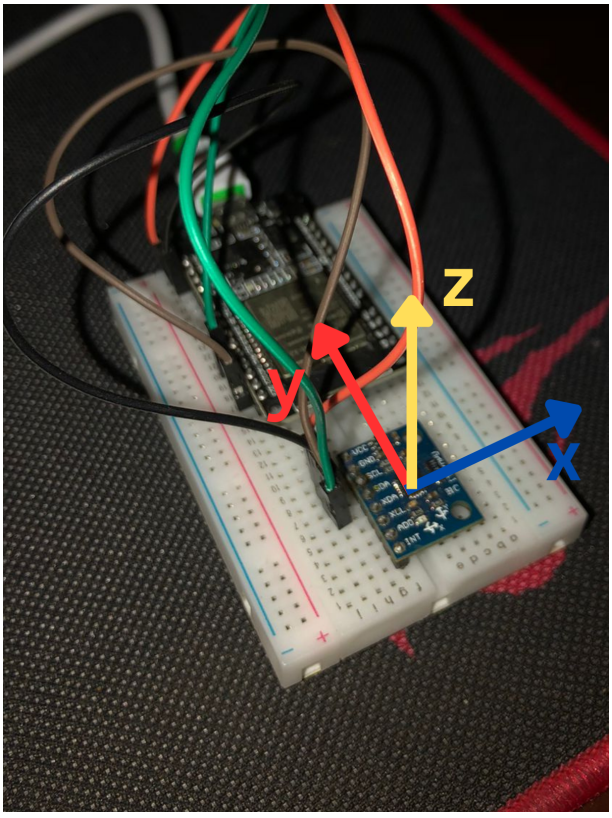
4 RESULTADOS

Antes de discutir as análises dos experimentos realizados, é importante esclarecer a orientação adotada do sensor nos experimentos. A orientação do sensor foi crucial para a interpretação correta dos dados coletados.

A orientação do sensor foi definida de acordo com um sistema de coordenadas cartesianas tridimensionais, onde cada eixo (x, y e z) representa uma direção no espaço.

O eixo X representa a direção lateral, da esquerda para direita. O eixo Y representa a direção longitudinal, de frente para trás. O eixo Z representa a direção vertical, de cima para baixo. Cada movimento do sensor ao longo de um desses eixos resulta em uma leitura correspondente do acelerômetro e do giroscópio, que são os principais sensores utilizados em nossos experimentos. Os eixos podem ser observados na figura abaixo.

Figura 6 – MPU6050 e seus eixos



Fonte: Autor.

No contexto dos nossos experimentos, um movimento positivo ao longo de um eixo corresponde a um movimento na direção positiva desse eixo no sistema de coordenadas, enquanto um movimento negativo corresponde a um movimento na direção oposta.

4.1 Média Móvel

Na implementação do presente trabalho, a média móvel foi empregada para suavizar os dados brutos obtidos dos sensores. Para cada componente do sensor (x, y e z), mantivemos um buffer que armazena um conjunto de dados cujo tamanho é determinado pela nossa janela de média móvel. Este buffer é continuamente atualizado com novas leituras do sensor.

A média móvel foi calculada da seguinte maneira:

$$MA = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{1}$$

- MA - A Média Móvel. Este é o valor resultante calculado pela fórmula.
- N - O número de pontos na janela da média móvel. Este é o tamanho da janela e determina quantos dos últimos pontos de dados são usados para calcular a média móvel.
- x_i - O valor do ponto de dados no índice i. Estes são os valores individuais dos pontos de dados que são usados para calcular a média móvel.

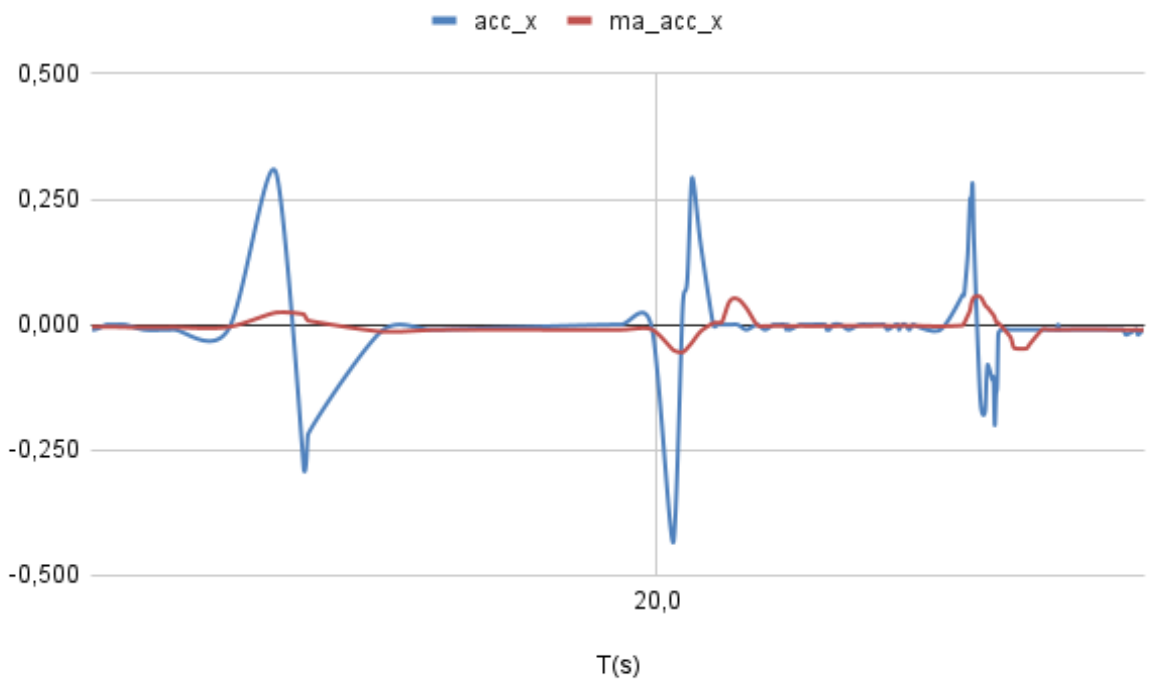
Quando novos dados são coletados do sensor, essas leituras são adicionadas ao final do buffer. Se o buffer já estiver preenchido, ou seja, se já contiver o número máximo de leituras permitido pela nossa janela de média móvel, o valor mais antigo é automaticamente descartado. A média móvel é então recalculada.

A implementação desta média móvel resultou em um conjunto de dados suavizados, que favorecem uma análise mais precisa. A suavização proporcionada pela média móvel minimiza o impacto de ruídos ou flutuações de curto prazo nos dados brutos.

Um aspecto importante a considerar é o tamanho da janela adotado na média móvel. Para determinar um tamanho de janela adequado, foram realizados testes com dois tamanhos distintos, $n = 40$ e $n = 10$, em uma amostra de dados coletada durante 180 segundos.

Para $n = 40$, foi observada uma suavização significativa dos dados, como pode ser visto na Figura abaixo.

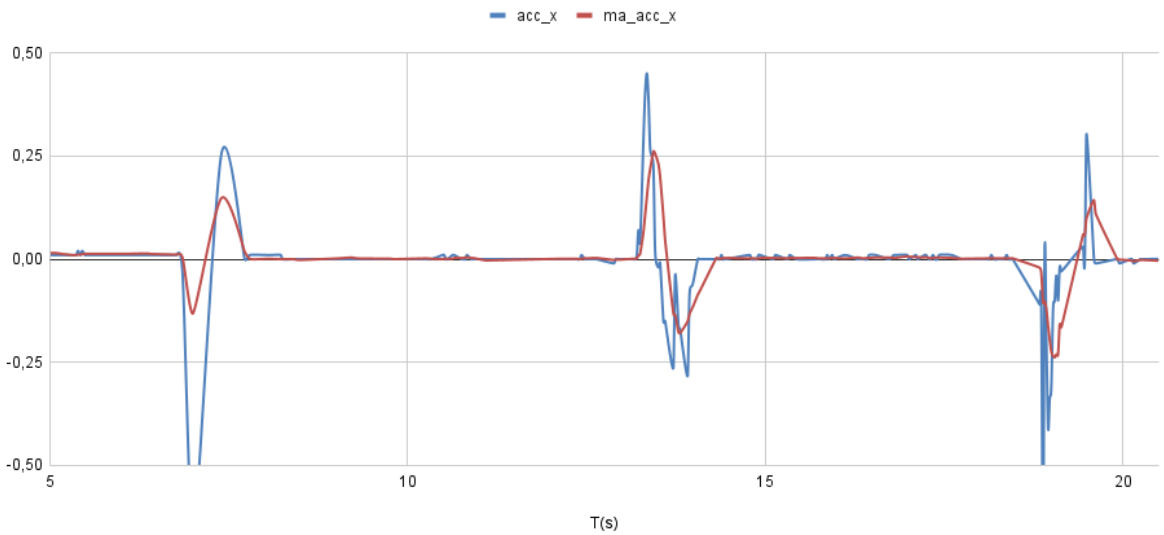
Figura 7 – Gráfico dos dados brutos e média móvel com uma janela de 40 leituras.



Fonte: Autor.

Em contraste, para $n = 10$, a suavização ainda estava presente, mas em um grau menor. No entanto, essa suavização mostrou-se adequada para as necessidades do projeto, como ilustrado na Figura abaixo.

Figura 8 – Gráfico dos dados brutos e média móvel com uma janela de 10 leituras.



Fonte: Autor.

Em suma, a escolha do tamanho da janela da média móvel é um aspecto crucial

que deve ser considerado. Uma janela maior pode resultar em uma suavização excessiva, possivelmente perdendo alguns detalhes importantes dos dados. Por outro lado, uma janela menor pode não ser suficiente para atenuar o ruído presente nos dados brutos.

Os experimentos realizados com diferentes tamanhos de janela de média móvel ($n = 40$ e $n = 10$) demonstraram que uma janela menor ($n = 10$) proporcionou uma suavização adequada para o nosso projeto, preservando simultaneamente as características essenciais dos dados.

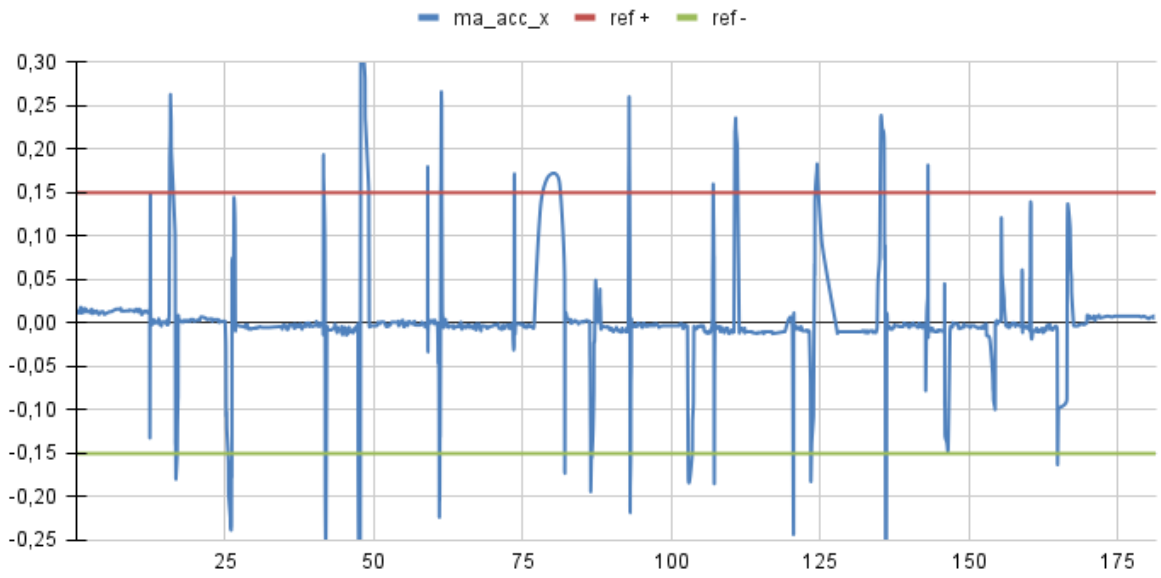
4.2 Identificação dos Limiares de Movimento

A identificação dos limiares de movimento é uma etapa crucial na análise dos dados do acelerômetro. Tais limiares permitem distinguir entre movimentos e estados estacionários, assim como facilitam a categorização de diferentes tipos de movimento. Para a definição desses limiares, foram realizados experimentos detalhados, que envolveram a translação do sensor nas direções lateral, vertical e longitudinal. Os detalhes desses experimentos, incluindo os gráficos dos dados coletados, estão disponíveis no Apêndice A.

Após a análise dos dados coletados, foram estabelecidos limiares de -0.15 e 0.15 para os eixos x e y, e -0.85 e 1.15 para o eixo z.

Na figura abaixo, é possível observar a aplicação desses limiares na translação lateral.

Figura 9 – Limiares do movimento lateral

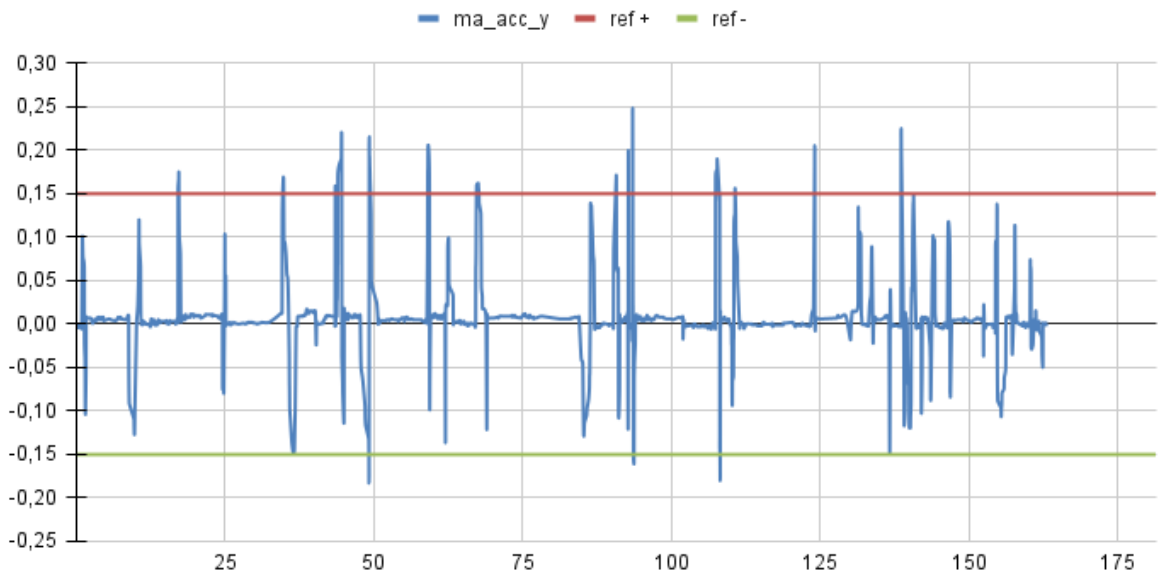


Fonte: Autor.

A maior parte dos dados está contida dentro dos limites estabelecidos, confirmando a eficácia dos limiares determinados para a translação lateral.

A figura seguinte ilustra a aplicação dos limiares na translação longitudinal. Apesar de alguns dados apresentarem-se abaixo dos limites estabelecidos, grande parte está contida dentro dos limiares, sugerindo que a padronização dos limites é aplicável neste caso.

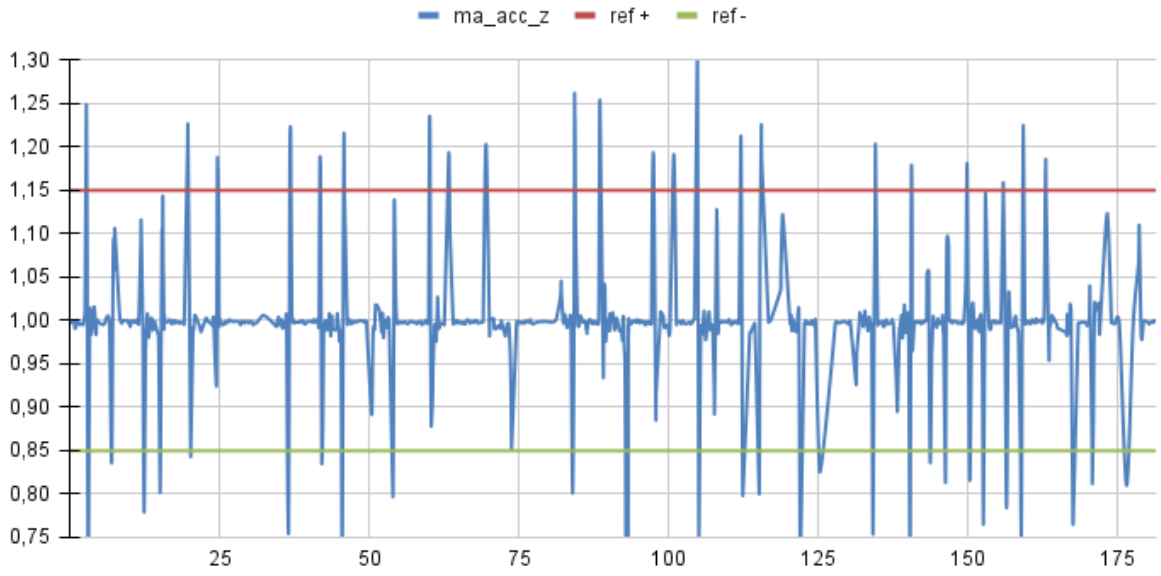
Figura 10 – Limiares do movimento longitudinal



Fonte: Autor.

Por fim, a figura abaixo demonstra a aplicação dos limiares na translação vertical. Assim como nas demais direções de movimento, os dados estão de acordo com os limites estabelecidos.

Figura 11 – Limiares do movimento vertical



Fonte: Autor.

A consistência observada na aplicação dos limiares em todas as direções de movimento reforça a sua validade e utilidade no processamento dos dados do acelerômetro.

4.3 Utilização do dispositivo para atividades ordinárias

Para testar a eficácia do dispositivo em atividades diárias, foram realizadas simulações de tarefas comuns que demandam a utilização dos membros superiores, em especial, os movimentos da mão e do braço. A aplicação do dispositivo em cenários de atividades cotidianas visa a demonstrar sua viabilidade prática e como ele pode ser usado para ajudar na reabilitação de pacientes com condições neurológicas.

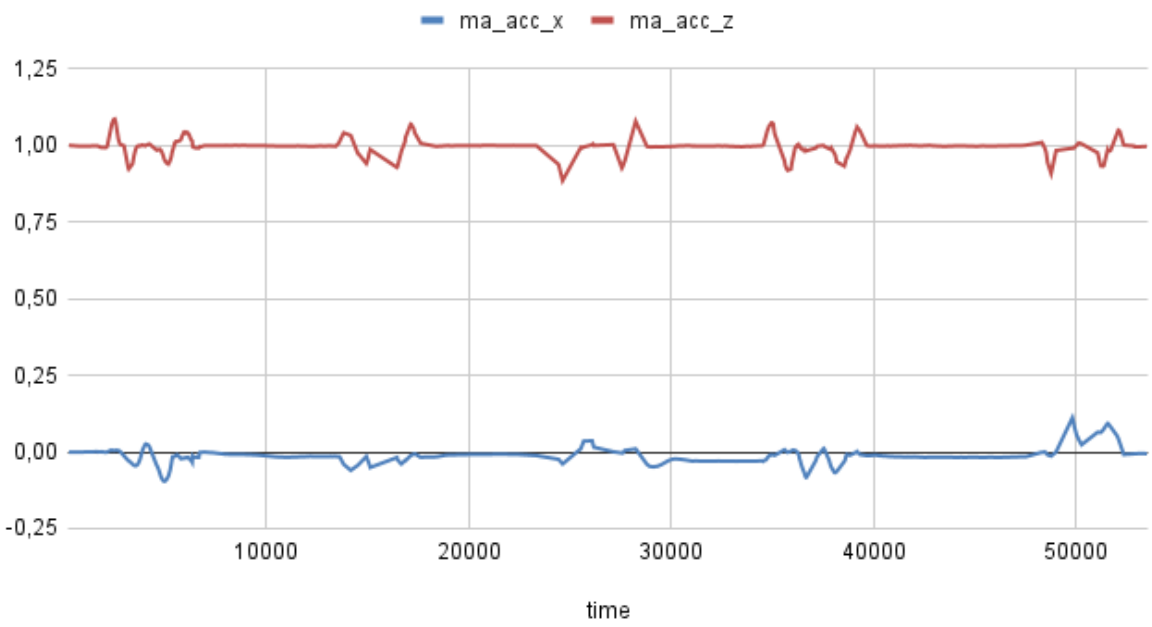
A tarefa simulada foi a movimentação de um copo em cima de uma mesa. Nesta simulação, o dispositivo foi transladado verticalmente e lateralmente, a fim de detectar a sequencia de movimentos.

Figura 12 – Primeira simulação - Movimento de um copo com uma só mão

Fonte: Autor.

Os dados captados no primeiro experimento podem ser observados no gráfico abaixo.

Figura 13 – Limiares do movimento vertical



Fonte: Autor.

Por fim é possível identificar os movimentos de subida e translação lateral do dispositivo.

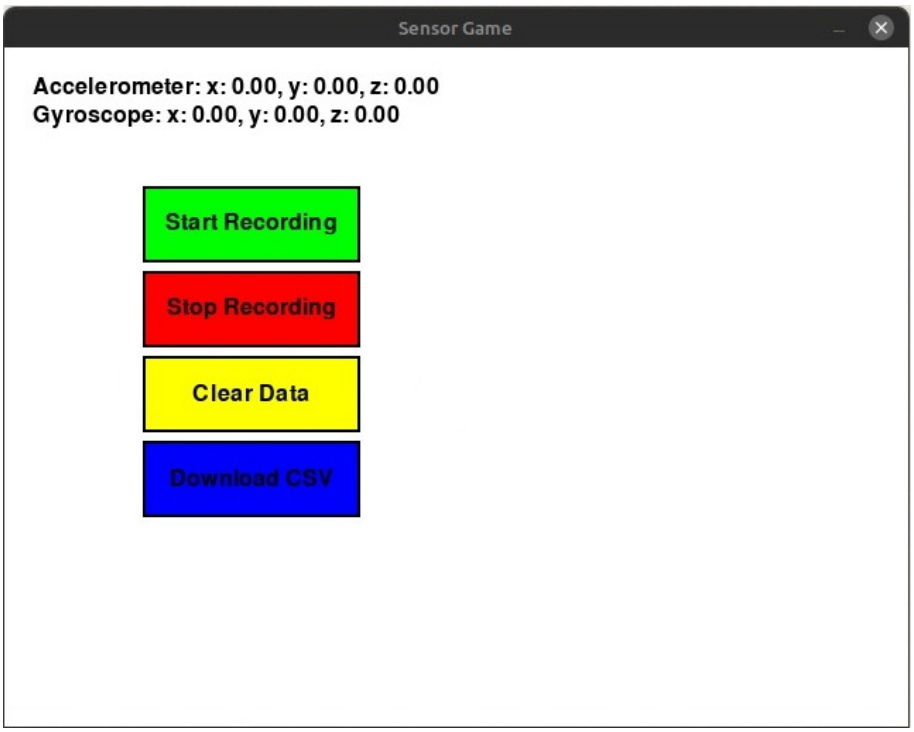
4.4 Interface para captura dos dados

A interface de captura de dados foi concebida para ser o mais intuitiva e funcional possível, permitindo a captura, limpeza e exportação dos dados captados pelo dispositivo de maneira prática e eficiente. Essa interface consiste em quatro botões principais:

- **Start Recording**: Este botão inicia a gravação dos dados do dispositivo. Quando acionado, o dispositivo começa a capturar os movimentos realizados e a registrar esses dados para análise futura. Durante o período de gravação, é essencial que os movimentos realizados estejam de acordo com o que se deseja analisar para garantir a precisão e relevância dos dados capturados.
- **Stop Recording**: Este botão termina o processo de gravação de dados. Após a gravação ser interrompida, os dados capturados ficam armazenados na memória do dispositivo para posterior análise.
- **Clean Data**: Este botão permite ao usuário limpar todos os dados registrados até o momento. Este recurso é útil quando o usuário deseja iniciar um novo conjunto de gravações sem a interferência dos dados anteriores.
- **Download CSV**: Este botão permite ao usuário baixar os dados registrados no formato CSV. Isso facilita a análise dos dados em softwares de terceiros, como planilhas ou programas de análise de dados.

Dessa forma, a interface oferece ao usuário todas as ferramentas necessárias para realizar uma captura de dados bem-sucedida, fornecendo um meio eficaz de registrar e analisar a reabilitação do paciente.

Figura 14 – Interface para captura dos dados



Fonte: Autor.

Além dos quatro botões mencionados, a interface também apresenta a visualização dos dados do acelerômetro e do giroscópio em tempo real na parte superior da tela. Estes indicadores são essenciais para acompanhar e entender os movimentos que estão sendo realizados pelo usuário em tempo real.

5 DISCUSSÃO

Este estudo envolveu a análise e aplicação prática de um dispositivo de sensor de movimento, com foco particular no uso de um acelerômetro para monitorar e interpretar os movimentos de transladação em três eixos: lateral (x), vertical (y) e longitudinal (z). O objetivo principal foi investigar como o dispositivo pode ser eficaz ao ser aplicado para auxiliar na reabilitação de pacientes com condições neurológicas que afetam os membros superiores.

Durante a fase inicial do estudo, foram conduzidos experimentos para testar a resposta do sensor de aceleração a movimentos controlados. Os resultados demonstraram um comportamento previsível e consistente do sensor em todas as direções de movimento, confirmando a robustez do dispositivo e a sua capacidade de responder de maneira precisa às variações de movimento. As observações também permitiram o estabelecimento de limiares de movimento que ajudam a diferenciar entre estados de movimento e de inatividade. Esses limiares mostraram-se aplicáveis e eficazes em todos os eixos de movimento, contribuindo para uma interpretação precisa dos dados do sensor.

Na etapa seguinte, o dispositivo foi aplicado em cenários de atividades cotidianas, especificamente na simulação de movimentação de um copo sobre uma mesa. Esses testes de uso prático mostraram a viabilidade do dispositivo para monitorar movimentos complexos e sequenciais. Os dados obtidos nestes testes ainda precisam ser analisados detalhadamente, o que será um foco de trabalho futuro. No entanto, os primeiros indícios apontam para uma correspondência positiva entre os dados do sensor e os movimentos executados.

Os resultados deste estudo reforçam o potencial do dispositivo para auxiliar na reabilitação neurológica, especialmente na melhoria do controle motor dos membros superiores. A capacidade de monitorar e interpretar movimentos precisamente e em tempo real pode ajudar os profissionais de saúde a desenvolver e ajustar programas de reabilitação mais eficazes, permitindo uma recuperação mais rápida e eficiente para os pacientes.

Apesar do progresso promissor, este estudo representa apenas um primeiro passo em direção ao pleno aproveitamento do potencial do dispositivo. Ademais, trabalhos futuros são necessários para aprofundar a análise dos dados coletados nas simulações de atividades cotidianas e para expandir o alcance dos testes para incluir uma gama maior de movimentos e atividades. Além disso, uma avaliação mais detalhada da resposta do dispositivo em diferentes condições ambientais e em resposta a variações individuais nos movimentos também seria benéfica.

6 CONCLUSÃO

Em conclusão, os resultados deste estudo confirmam o potencial do dispositivo de sensor de movimento como uma ferramenta útil para a reabilitação neurológica. Espera-se que os insights gerados a partir desta pesquisa contribuam para o desenvolvimento de abordagens mais eficazes e personalizadas para a reabilitação de pacientes com condições que afetam os membros superiores.

REFERÊNCIAS

ALMEIDA, S. R. M. Análise epidemiológica do acidente vascular cerebral no brasil. **Revista Neurociência**, v. 20, n. 4, p. 481–482, 2012.

ALVES, C. L.; SANTANA, D. S. de; AOYAMA, E. de A. Acidente vascular encefálico em adultos jovens com ênfase nos fatores de risco. **Revista Brasileira Interdisciplinar de Saúde**, 2020.

ANG, K. K. et al. Brain computer interface based robotic end effector system for wrist and hand rehabilitation: results of a three-armed randomized controlled trial for chronic stroke. **Frontiers in Neuroengineering**, v. 7, p. 30, 2014.

BALESTRINO, R.; SCHAPIRA, A. H. Parkinson disease. **European Journal of Neurology**, v. 27, n. 1, p. 27–42, 2019.

CARDOSO, L. R. L. ispositivo robótico instrumentado para reabilitação bimanual: estudo de caso com aprendizagem motora em pessoas típicas. **USP**, 2020.

CARDUCCI, C. G. C. Enabling esp32-based iot applications in building automation systems. **2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0IoT)**, 2019.

CARVALHO, M. I. F. et al. Acidente vascular cerebral: Dados clínicos e epidemiológicos de uma clínica de fisioterapia do sertão nordestino brasileiro. **Revista Interfaces: Saúde, Humanas e Tecnologia**, v. 2, n. 6, 2014.

CEDARO, J. J. et al. Doença neurodegenerativa rara: itinerário de portadores de doença de huntington em busca de diagnóstico e tratamento. **Brazilian Journal of Health Review**, v. 3, n. 5, p. 13182–13197, 2020.

CEREBROVASCULARES, S. B. de D. **Acidente Vascular Cerebral**. 2023. Acessado em: 15 de março de 2023. Disponível em: <http://www.sbdcv.org.br/publica_avc.asp>.

CHOU, K. L. **Clinical manifestations of Parkinson disease**. 2020. UpToDate.

COSTA, E. L. D.; SANTOS, C. C. T. Gameterapia na reabilitação de pacientes com paralisia cerebral. **Revista Coleta Científica**, v. 5, n. 10, p. 60–68, 2021.

DAMATA, S. R. R. et al. Perfil epidemiológico dos idosos acometidos por acidente vascular cerebral. **R. Interd**, v. 9, n. 1, p. 107–117, 2016.

ELAD, D. et al. Sense of autonomy and daily and scholastic functioning among children with cerebral palsy. **Research in Developmental Disabilities**, v. 80, p. 161–169, 2018.

IBGE. **Número de idosos cresce 18% em 5 anos e ultrapassa 30 milhões em 2017**. 2018. <https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/20980-numero-de-idosos-cresce-18-em-5-anos-e-ultrapassa-30-milhoes-em-2017>.

LAMERS, I. et al. Upper limb rehabilitation in people with multiple sclerosis. **Neurorehabilitation and Neural Repair**, v. 30, p. 773 – 793, 2016.

MALAK, A. L. S. B. et al. Symptoms of depression in patients with mild cognitive impairment in parkinson's disease. **Dement Neuropsychol**, v. 11, n. 2, p. 145–153, 2017.

MORENO, J. C. O. et al. **Efeitos Clínicos e terapêuticos de micro doses de canabinoides para tratamento da paralisia cerebral não progressiva da infância: um estudo clínico, duplo-cego, randomizado, prospectivo e controlado por placebo**. Dissertação (Mestrado) — Universidade de São Paulo, 2023.

NAVYA, J. Non-invasive fall detection system for parkinson's disease. **2018 International CET Conference on Control, Communication, and Computing (IC4)**, 2018.

ONAOLAPO, A. Y.; ONAOLAPO, O. J.; NATHANIEL, T. I. Cerebrovascular disease in the young adult: Examining melatonin's possible multiple roles. **Journal of experimental neuroscience**, v. 13, 2019.

POLI, P. et al. Robotic technologies and rehabilitation: new tools for stroke patients' therapy. **BioMed Research International**, v. 2013, 2013.

PRINCE, M. et al. The global prevalence of dementia: a systematic review and metaanalysis. **Alzheimer's & Dementia**, v. 9, n. 1, p. 63–75.e2, 2013. PMID: 23305823.

PéREZ, F. Python: An ecosystem for scientific computing. **Computing in Science Engineering**, 2011.

RIBEIRO, S. et al. Determinants of stroke in brazil: A cross-sectional multivariate approach from the national health survey. **Journal of Stroke and Cerebrovascular Diseases**, v. 27, n. 6, p. 1616–1623, 2018.

ROCHA, A. C. P. et al. A novel device for grasping assessment during functional tasks: Preliminary results. **Frontiers in Bioengineering and Biotechnology**, v. 4, 2016. ISSN 2296-4185. Disponível em: <https://www.frontiersin.org/articles/10.3389/fbioe.2016.00016>.

RODRíGUEZ-PONCE, R. Controlador de servomotores industriales mediante un microcontrolador utilizando micropython. **Revista de Simulación y Laboratorio**, 2019.

SALE, P. et al. Effects of upper limb robot-assisted therapy on motor recovery in subacute stroke patients. **Journal of NeuroEngineering and Rehabilitation**, v. 11, 2014.

SCHRODER, J. D.; ARAUJO, J. A. B. de; IGNáCIO, Z. M. O efeito da redução telomérica sobre as doenças neurodegenerativas. **Simpósio de Neurociência Clínica e Experimental**, v. 1, n. 1, 2020.

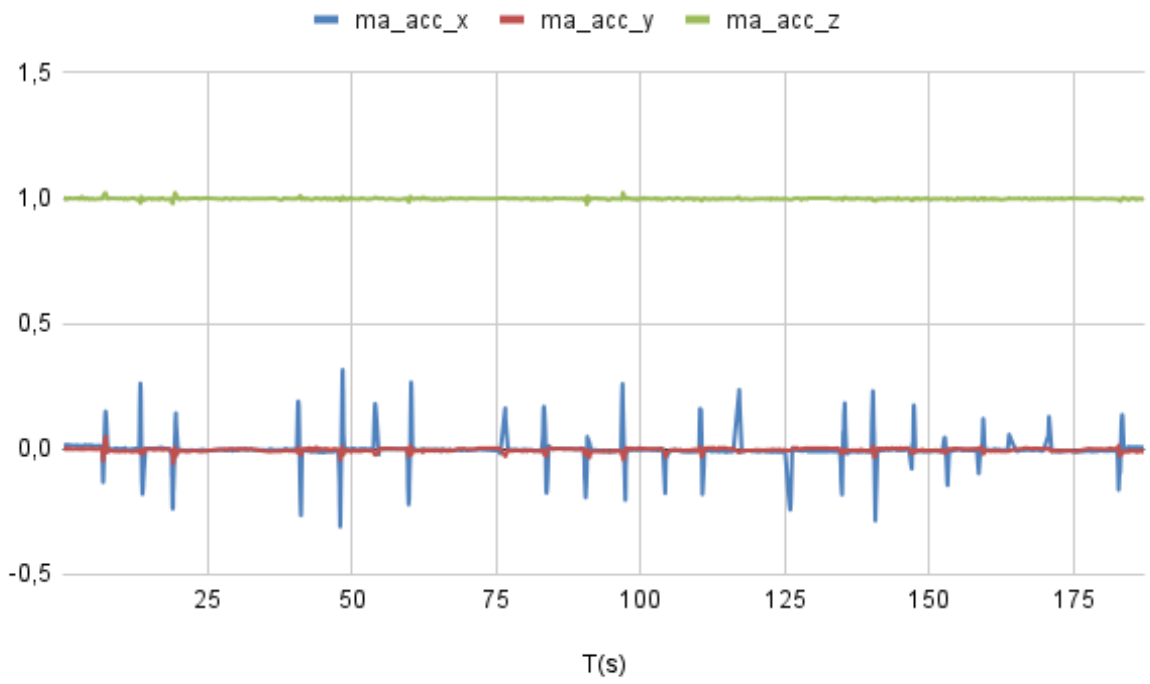
Apêndices

APÊNDICE A – Detalhamento dos Experimentos de Translação do Sensor

A.1 Detecção da translação lateral

O primeiro experimento consiste na translação lateral do sensor, ou seja, ao longo do eixo x. Durante 180 segundos o sensor foi transladado de um lado para outro a cada 5 segundos. Dessa forma e possível identificar os padrões dos dados na leitura do sensor. Os dados podem ser visualizados no gráfico abaixo

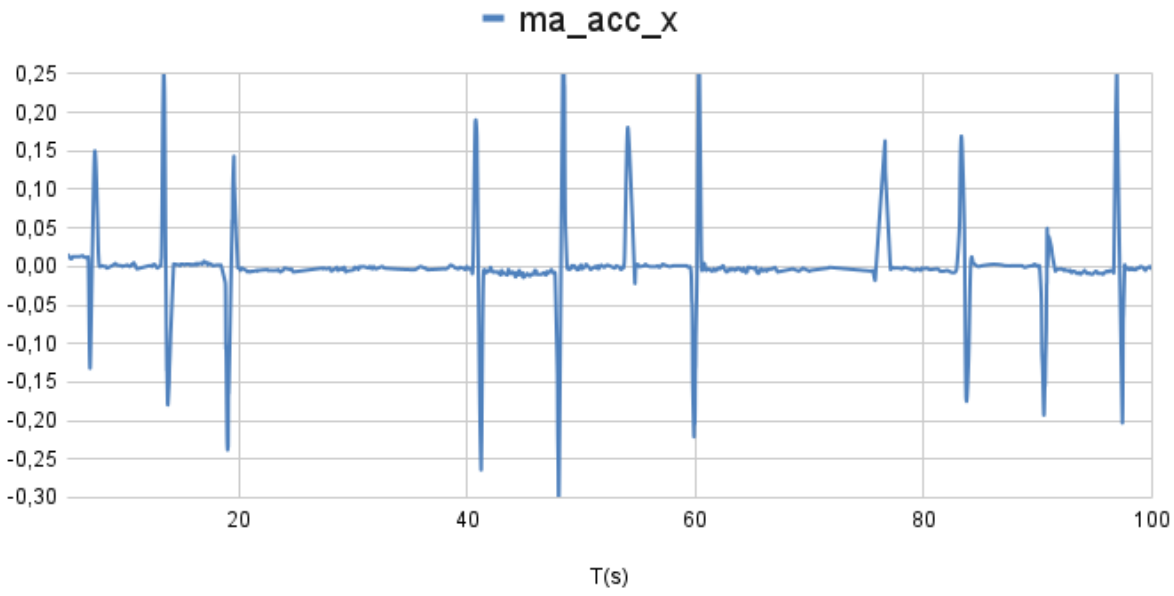
Figura 15 – Gráfico dos dados na translação lateral do sensor



Fonte: Autor.

No gráfico abaixo podemos observar em detalhe o comportamento apenas no eixo x.

Figura 16 – Gráfico de um trecho dos dados em x do acelerômetro



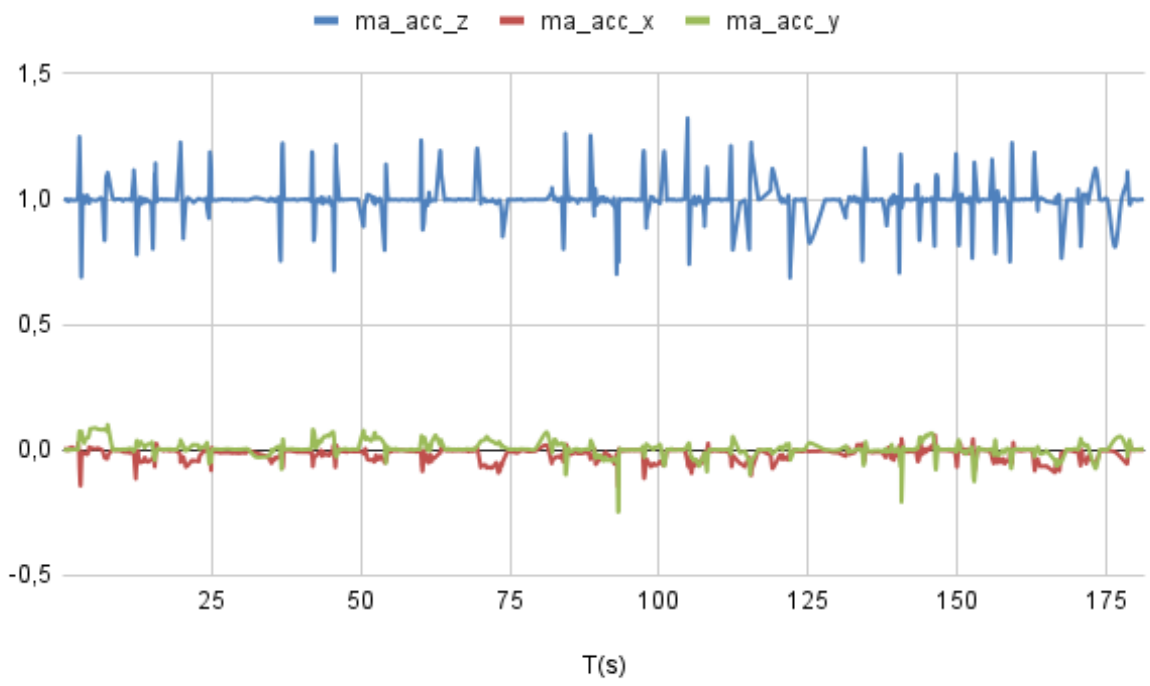
Fonte: Autor.

Por fim, é possível verificar que o sensor reage conforme o esperado durante a translação lateral. Ao mover o sensor na direção positiva do eixo x, os dados apresentam um pico positivo seguido por um pico negativo, indicando a aceleração e a desaceleração respectivamente. Quando o sensor é movido na direção oposta, o fenômeno ocorre de forma invertida. Este entendimento é fundamental para entendermos o comportamento do sensor e como interpretar seus dados em aplicações futuras.

A.2 Detecção da translação vertical

O próximo experimento consistiu na translação vertical do sensor, ou seja, ao longo do eixo y. O sensor foi movido verticalmente para cima e para baixo por 180 segundos, alternando a cada 5 segundos. O objetivo é identificar os padrões nos dados da leitura do sensor quando ocorre a translação vertical. Os dados são apresentados no gráfico abaixo.

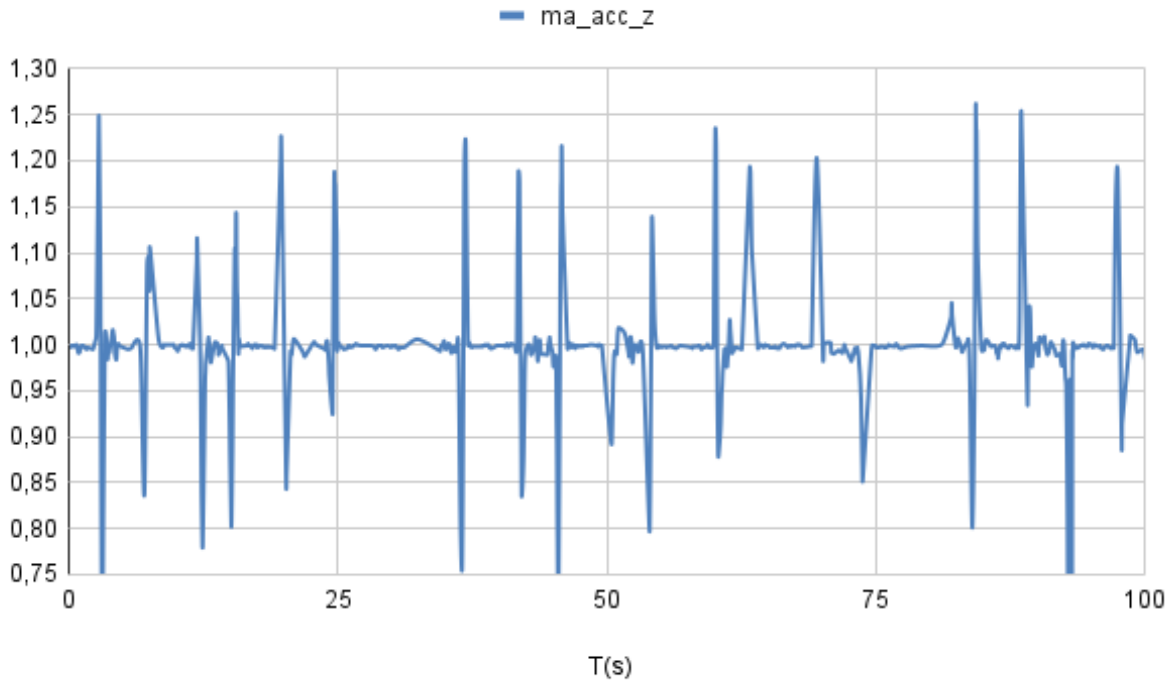
Figura 17 – Gráfico dos dados na translação vertical do sensor



Fonte: Autor.

No gráfico abaixo, podemos observar em detalhe o comportamento apenas no eixo y.

Figura 18 – Gráfico de um trecho dos dados em y do acelerômetro



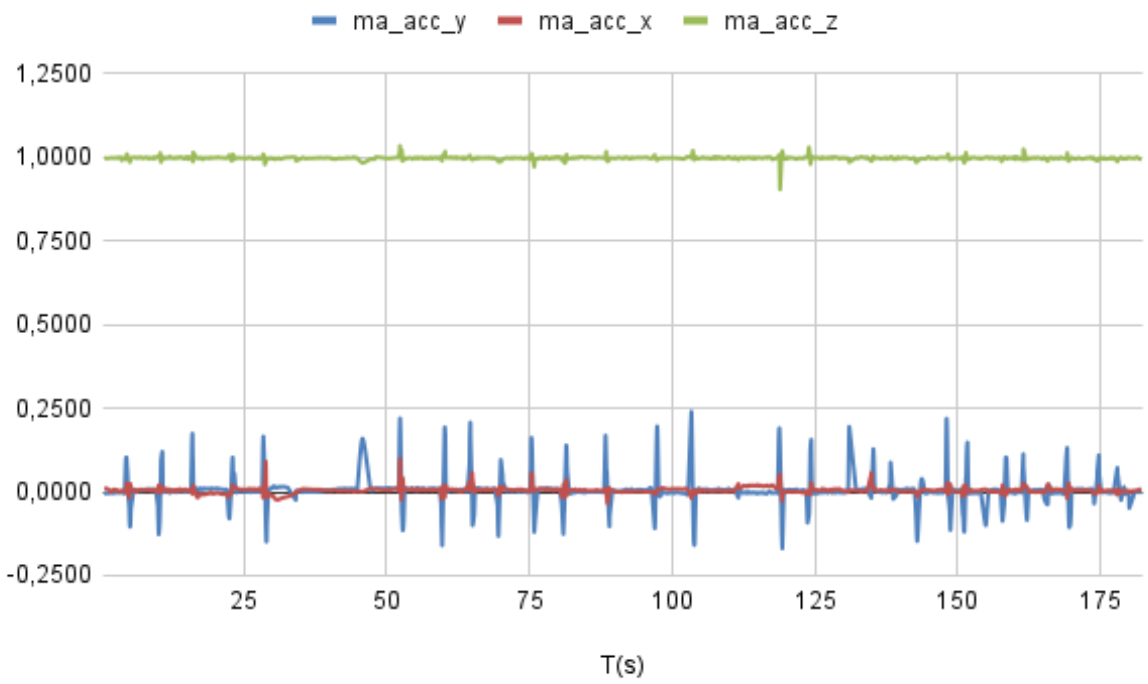
Fonte: Autor.

Similar ao experimento de translação lateral, é possível observar que o sensor responde de maneira previsível durante a translação vertical. Quando o sensor é movido para cima, os dados apresentam um pico positivo seguido por um pico negativo, indicando a aceleração e desaceleração, respectivamente. O oposto é observado quando o sensor é movido para baixo. Este comportamento ajuda na compreensão de como o sensor reage a movimentos em diferentes direções e será útil em aplicações futuras.

A.3 Detecção da translação longitudinal

Por último, o experimento envolve a translação longitudinal do sensor, ou seja, ao longo do eixo z. Durante 180 segundos, o sensor foi transladado de frente para trás a cada 5 segundos. Os dados são visualizados no gráfico abaixo.

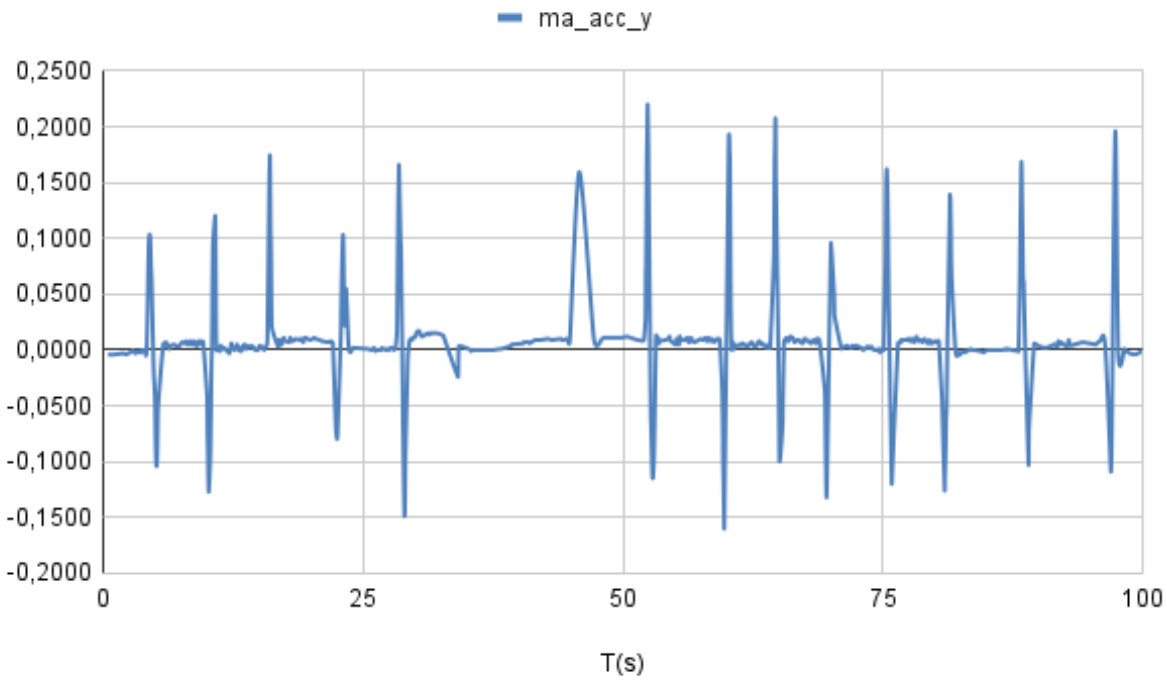
Figura 19 – Gráfico dos dados na translação longitudinal do sensor



Fonte: Autor.

O gráfico a seguir apresenta um detalhamento dos dados apenas no eixo z.

Figura 20 – Gráfico de um trecho dos dados em z do acelerômetro



Fonte: Autor.

Assim como nas translações lateral e vertical, o sensor exibe um comportamento consistente durante a translação longitudinal. Ao mover o sensor para frente (direção positiva do eixo z), os dados apresentam um pico positivo seguido por um pico negativo, denotando aceleração e desaceleração. O inverso é observado quando o sensor é movido para trás (direção negativa do eixo z). Estes resultados confirmam a previsibilidade do sensor em diferentes direções, o que é crucial para o desenvolvimento de aplicações precisas no futuro.

APÊNDICE B – Código do Software no computador

```
import serial
import time
import csv
from collections import deque
import os


SERIAL_PORT = '/dev/ttyUSB0'
BAUD_RATE = 115200
MOVING_AVERAGE_SIZE = 10

class CustomSerial:
    """Handles serial communication with an accelerometer and gyroscope."""

    def __init__(self, ser=serial.Serial(SERIAL_PORT, BAUD_RATE),
        moving_average_size=MOVING_AVERAGE_SIZE):
        self.ser = ser
        self.moving_average_size = moving_average_size
        self.acc_data = [0, 0, 0]
        self.gyro_data = [0, 0, 0]
        self.acc_buffer = [deque(maxlen=self.moving_average_size) for _ in
            range(3)]
        self.gyro_buffer = [deque(maxlen=self.moving_average_size) for _ in
            range(3)]
        self.all_sensor_data = []
        self.is_recording = False # Whether data is currently being saved.
        self.start_time = None

    def start_recording(self):
        self.is_recording = True
        self.start_time = time.time()

    def stop_recording(self):
        self.is_recording = False
        self.start_time = None

    def clear_data(self):
        self.all_sensor_data = []
```

```
def _get_raw_data(self, sensor_type):
    """Retrieves and processes raw data from a specified sensor type."""

    try:
        if self.ser.in_waiting > 0:
            line = self.ser.readline().decode('utf-8').strip()
            if sensor_type.upper() in line:
                clean_data = line.split()
                data = [float(clean_data[i]) for i in range(3, 8, 2)]
                return data
    except Exception as e:
        print(f"Error occurred while reading data: {e}")
        return [0, 0, 0]

def get_data(self, sensor_type):
    """Retrieves raw and moving average data from a specified sensor
        type."""

    raw_data = self._get_raw_data(sensor_type)
    if raw_data is None:
        return None, None

    if sensor_type.upper() == "ACCELEROMETER":
        buffer = self.acc_buffer
        for i, value in enumerate(raw_data):
            buffer[i].append(value)
    else: # Assume GYROSCOPE
        buffer = self.gyro_buffer
        for i, value in enumerate(raw_data):
            buffer[i].append(value)

    ma_data = [sum(d)/len(d) if len(d) > 0 else 0 for d in buffer]
    return raw_data, ma_data

def save_data(self):
    if not self.is_recording: # Skip saving if not currently recording.
        return
    """Saves raw and moving average data from both sensors."""

    acc_raw_data, acc_ma_data = self.get_data("ACCELEROMETER")
    gyro_raw_data, gyro_ma_data = self.get_data("GYROSCOPE")
```

```
        # Save data
        try:
            elapsed_time = (time.time() - self.start_time) * 1000
            self.all_sensor_data.append([elapsed_time] + acc_raw_data +
                gyro_raw_data + acc_ma_data + gyro_ma_data)
        except:
            pass

    def download_csv(self, filename_base="output"):
        """Writes all stored sensor data to a CSV file."""

        # Find a unique filename by incrementing a number suffix.
        i = 1
        while os.path.exists(f"{filename_base}_{i}.csv"):
            i += 1
        filename = f"{filename_base}_{i}.csv"

        header = ["time", "acc_x", "acc_y", "acc_z", "gyro_x", "gyro_y",
            "gyro_z", "ma_acc_x", "ma_acc_y", "ma_acc_z", "ma_gyro_x",
            "ma_gyro_y", "ma_gyro_z"]
        with open(filename, "w", newline="") as file:
            writer = csv.writer(file)
            writer.writerow(header)
            for data in self.all_sensor_data:
                writer.writerow(data)

        print(f"Data saved to {filename}")

    def get_serial(self):
        """Returns the serial object."""

        return self.ser
```

Listing B.1 – *Custom_serial.py*

```
import pygame
from pygame.locals import *
from elements import Button
import time

class SensorGame:
```

```
def __init__(self, custom_serial):
    pygame.init()
    pygame.display.set_caption("Sensor Game")
    self.WIDTH, self.HEIGHT = 640, 480
    self.WHITE = (255, 255, 255)
    self.screen = pygame.display.set_mode((self.WIDTH, self.HEIGHT))
    self.download_button = Button((0, 0, 255), 100, 280, 150, 50, 'Download
        CSV')
    self.clear_button = Button((255, 255, 0), 100, 220, 150, 50, 'Clear
        Data')
    self.start_button = Button((0, 255, 0), 100, 100, 150, 50, 'Start
        Recording')
    self.stop_button = Button((255, 0, 0), 100, 160, 150, 50, 'Stop
        Recording')
    self.clear_button.draw(self.screen, (0, 0, 0))
    self.start_button.draw(self.screen, (0, 0, 0))
    self.stop_button.draw(self.screen, (0, 0, 0))
    self.download_button.draw(self.screen, (0, 0, 0))
    self.accelerometer_data = [0, 0, 0]
    self.gyroscope_data = [0, 0, 0]
    self.player = Player(self.WIDTH, self.HEIGHT)
    self.custom_serial = custom_serial # custom_serial is the CustomSerial
        instanc
    self.font = pygame.font.Font(None, 24) # Create a font object.
    self.running = True
    self.last_movement_time = time.time()
    self.last_movement_direction = None

def run(self):
    while self.running:
        self.handle_events()
        self.update_game_state()
        self.draw_game_state()
    pygame.quit()

def handle_events(self):
    for event in pygame.event.get():
        pos = pygame.mouse.get_pos()

        if event.type == pygame.QUIT:
            self.running = False
```

```
            if event.type == pygame.MOUSEBUTTONDOWN:
                if self.start_button.is_over(pos):
                    self.custom_serial.start_recording()
                elif self.stop_button.is_over(pos):
                    self.custom_serial.stop_recording()
                elif self.clear_button.is_over(pos):
                    self.custom_serial.clear_data()
                elif self.download_button.is_over(pos):
                    self.custom_serial.download_csv()

            if event.type == pygame.MOUSEMOTION:
                for button in [self.start_button, self.stop_button,
                    self.clear_button, self.download_button]:
                    if button.is_over(pos):
                        button.color = (255, 0, 0)
                    else:
                        button.color = (0, 255, 0)

    def process_movement(self, data, label, positive_direction,
        negative_direction):
        if label == 'z':
            if data > 1.05 and abs(self.accelerometer_data[0]) < 0.1 and
                abs(self.accelerometer_data[1]) < 0.1 and time.time() -
                self.last_movement_time > 1:
                self.last_movement_time = time.time()
                self.last_movement_direction = positive_direction
                print(f"Going {positive_direction} - {data} - x:
                    {self.accelerometer_data[0]}, y:
                    {self.accelerometer_data[1]}")
            elif data < 0.95 and abs(self.accelerometer_data[0]) < 0.1 and
                abs(self.accelerometer_data[1]) < 0.1 and time.time() -
                self.last_movement_time > 1:
                self.last_movement_time = time.time()
                self.last_movement_direction = negative_direction
                print(f"Going {negative_direction} - {data} - x:
                    {self.accelerometer_data[0]}, y:
                    {self.accelerometer_data[1]}")
        if label == 'x' or label == 'y':
            if data > 0.05 and time.time() - self.last_movement_time > 1 and
                abs(self.accelerometer_data[2]) < 1.05 and
                abs(self.accelerometer_data[2]) > 0.95:
                self.last_movement_time = time.time()
```

```
                self.last_movement_direction = positive_direction
                print(f"Going {positive_direction} - {data} - y:
                    {self.accelerometer_data[1]}, z:
                    {self.accelerometer_data[2]}")
            elif data < -0.05 and time.time() - self.last_movement_time > 1 and
                abs(self.accelerometer_data[2]) < 1.05 and
                abs(self.accelerometer_data[2]) > 0.95:
                self.last_movement_time = time.time()
                self.last_movement_direction = negative_direction
                print(f"Going {negative_direction} - {data} - y:
                    {self.accelerometer_data[1]}, z:
                    {self.accelerometer_data[2]}")

    def update_game_state(self):
        try:
            # Get accelerometer data from custom_serial
            raw_accel_data, ma_accel_data =
                self.custom_serial.get_data("ACCELEROMETER")
            self.custom_serial.save_data()
            if ma_accel_data is not None and all(element is not None for
                element in ma_accel_data):
                self.accelerometer_data = ma_accel_data
                self.player.move(self.accelerometer_data[0],
                    -self.accelerometer_data[1])

                # Print messages based on accelerometer data
                self.process_movement(self.accelerometer_data[2], 'z', 'up',
                    'down')
                self.process_movement(self.accelerometer_data[0], 'x', 'right',
                    'left')
                self.process_movement(self.accelerometer_data[1], 'y',
                    'foward', 'backward')

            # Get gyroscope data from custom_serial
            raw_gyro_data, ma_gyro_data =
                self.custom_serial.get_data("GYROSCOPE")
            if ma_gyro_data is not None and all(element is not None for element
                in ma_gyro_data):
                self.gyroscope_data = raw_gyro_data
        except Exception as e:
            print(f"Error occurred while reading data: {e}")
            self.accelerometer_data = [0, 0, 0]
```

```
            self.gyroscope_data = [0, 0, 0]

    def draw_game_state(self):
        self.screen.fill(self.WHITE)
        for button in [self.start_button, self.stop_button, self.clear_button,
            self.download_button]:
            button.draw(self.screen, (0, 0, 0))
        self.screen.blit(self.player.surf, self.player.rect)
        accel_text = "Accelerometer: x: {:.2f}, y: {:.2f}, z:
            {:.2f}".format(*self.accelerometer_data)
        gyro_text = "Gyroscope: x: {:.2f}, y: {:.2f}, z:
            {:.2f}".format(*self.gyroscope_data)
        text_surface_acc = self.font.render(accel_text, True, (0, 0, 0))
        text_surface_gyro = self.font.render(gyro_text, True, (0, 0, 0))
        self.screen.blit(text_surface_acc, (20, 20))
        self.screen.blit(text_surface_gyro, (20, 40))
        pygame.display.flip()

class Player(pygame.sprite.Sprite):
    def __init__(self, WIDTH, HEIGHT):
        super().__init__()
        self.surf = pygame.Surface((50, 50))
        pygame.draw.circle(self.surf, (0, 0, 255), (25, 25), 25)
        self.surf.set_colorkey((0, 0, 0))
        self.rect = self.surf.get_rect(center=(WIDTH/2, HEIGHT/2))
        self.speed = 5.0
        self.original_surf = self.surf.copy()

    def move(self, x, y):
        self.rect.move_ip(x * self.speed, y * self.speed)

    def change_size(self, z):
        factor = 0.5 + z # scales the size according to z axis
        self.surf = pygame.transform.rotozoom(self.original_surf, 0, factor)
        self.rect = self.surf.get_rect(center=self.rect.center)
```

Listing B.2 – *game.py*

```
    from custom_serial import CustomSerial
    from game import SensorGame
```

```
def main():
    custom_serial = CustomSerial()
    game = SensorGame(custom_serial)
    game.run()


if __name__ == "__main__":
    main()
```

Listing B.3 – *main.py*

APÊNDICE C – Código do Software no ESP32

```
from machine import Pin
from esp32.mpu6050 import MPU6050
import ustruct

sda = Pin(21)
scl = Pin(22)
bus = 1 # or whichever bus your device is connected to

sensor = MPU6050(bus, sda, scl)

def run():
    while True:
        try:
            sensor.print_data()
        except Exception as e:
            print(e)
            pass

if __name__ == "__main__":
    run()
```

Listing C.1 – *main.py*

```
from machine import Pin, I2C
from micropython import const
from collections import namedtuple
import struct, utime, math

_R2D          = 180/math.pi

NONE          = const(0x00)

#filter type flags
ANGLE_KAL     = const(0x01)
ANGLE_COMP    = const(0x02)
ANGLE_BOTH    = const(0x03)

#filter 'which' flags
```

```
FILTER_ACCEL   = const(0x01)
FILTER_GYRO    = const(0x02)
FILTER_ANGLES  = const(0x04)
FILTER_ALL     = const(0x07)

#accel
ACCEL_FS_2     = const(0x00)
ACCEL_FS_4     = const(0x01)
ACCEL_FS_8     = const(0x02)
ACCEL_FS_16    = const(0x03)

#gyro
GYRO_FS_250    = const(0x00)
GYRO_FS_500    = const(0x01)
GYRO_FS_1000   = const(0x02)
GYRO_FS_2000   = const(0x03)

#dlpf
DLPF_BW_256    = const(0x00)
DLPF_BW_188    = const(0x01)
DLPF_BW_98     = const(0x02)
DLPF_BW_42     = const(0x03)
DLPF_BW_20     = const(0x04)
DLPF_BW_10     = const(0x05)
DLPF_BW_5      = const(0x06)

#clock
CLK_INTERNAL   = const(0x00)
CLK_PLL_XGYRO  = const(0x01)
CLK_PLL_YGYRO  = const(0x02)
CLK_PLL_ZGYRO  = const(0x03)
CLK_PLL_EXT32K = const(0x04)
CLK_PLL_EXT19M = const(0x05)
CLK_KEEP_RESET = const(0x07)


# Data Structure
_D   = namedtuple('D', ('acc_x', 'acc_y', 'acc_z', 'gyro_x', 'gyro_y',
    'gyro_z'))
_A   = namedtuple('A', ('roll', 'pitch'))

# Data Formatting
```

```
_SEP = '-'*60
_W   = (' ', '-')
_C   = '{}\n{}'
_F   = '\t{} failed Self-Test'
_OUT = '[{:<16}] x: {}{:<10.2f} y: {}{:<10.2f} z: {}{:<10.2f}'


#__> I2C
class __I2CHelper(object):
    def __init__(self, bus:int, sda, scl, addr:int, freq:int=400000) ->
        None:
        if isinstance(sda, int): sda = Pin(sda)
        if isinstance(scl, int): scl = Pin(scl)
        self.__addr   = addr
        self.__bus    = I2C(bus, scl=scl, sda=sda, freq=freq)
        self.__buffer = bytearray(16)
        self.__data   = [memoryview(self.__buffer[0:n]) for n in range(1,
            17)]

    def __writeBytes(self, reg:int, buff) -> None:
        self.__bus.writeto_mem(self.__addr, reg, buff)

    def __writeWords(self, reg:int, length:int, val) -> None:
        if isinstance(val, int):
            L = int(length * 2)
            val = bytearray(val.to_bytes(L, 'big'))

        if isinstance(val, (list, tuple)):
            val = bytearray(val)

        if isinstance(val, (bytearray, bytes, memoryview)):
            self.__writeBytes(reg, val)

    def __writeByte(self, reg:int, val) -> None:
        if isinstance(val, int):
            val = bytearray([val])

        if isinstance(val, (bytearray, bytes, memoryview)):
            self.__writeBytes(reg, val)

    def __writeBit(self, reg:int, bit:int, data:int) -> None:
        b = self.__readByte(reg)
```

```
        self.__data[0][0] = (b | (1 << bit)) if data else (b & ~(1 << bit))
        self.__writeByte(reg, self.__data[0][0])

    def __writeBits(self, reg:int, bitstart:int, length:int, data:int) ->
        None:
        shift = (bitstart - length + 1)
        mask  = ((1 << length) - 1) << shift
        self.__readByte(reg)
        data <<= shift
        data &= mask
        self.__data[0][0] &= ~(mask)
        self.__data[0][0] |= data
        self.__writeByte(reg, self.__data[0])

    def __readBytes(self, reg:int, length:int) -> int:
        if length > 0:
            if length in range(1, 17):
                self.__bus.readfrom_mem_into(self.__addr, reg,
                    self.__data[length-1])
                return self.__data[length-1]
            else:
                buff = bytearray([0x00]*length)
                self.__bus.readfrom_mem_into(self.__addr, reg, buff)
                return buff
        else:
            return bytearray()

    def __readWords(self, reg:int, length:int) -> int:
        fmt = '>{}'.format('h'*length)
        return struct.unpack(fmt, self.__readBytes(reg, length*2))

    def __readByte(self, reg:int) -> int:
        return self.__readBytes(reg, 1)[0]

    def __readBit(self, reg:int, bit:int) -> int:
        return (self.__readByte(reg) & (1 << bit))

    def __readBits(self, reg:int, bitstart:int, length:int) -> int:
        shift = (bitstart - length + 1)
        mask  = ((1 << length) - 1) << shift
        return ((self.__readByte(reg) & mask) >> shift)
```

```
#__> Kalman Filter
class __Filters(object):
    def __init__(self, R:float, Q:float, alpha:float) -> None:
        self.__cov = float('nan')
        self.__x  = float('nan')
        self.__c  = float('nan')
        self.__A, self.__B, self.__C = 1, 0, 1
        self.__R, self.__Q = R, Q

        self.__alpha = alpha
        self.__time = utime.ticks_us()
        self.__delta = utime.ticks_diff(utime.ticks_us(),
            self.__time)/1000000

    def kalman(self, angle:float) -> float:
        u = 0
        if math.isnan(self.__x):
            self.__x  = (1 / self.__C) * angle
            self.__cov = (1 / self.__C) * self.__Q * (1 / self.__C)
        else:
            px        = (self.__A * self.__x) + (self.__B * u)
            pc        = ((self.__A * self.__cov) * self.__A) + self.__R
            K         = pc * self.__C * (1 / ((self.__C * pc * self.__C) +
                self.__Q))
            self.__x  = px + K * (angle - (self.__C * px))
            self.__cov = pc - (K * self.__C * pc)

        return self.__x

    def complementary(self, rate:float, angle:float) -> float:
        if math.isnan(self.__c):
            self.__c = angle

        self.__delta = utime.ticks_diff(utime.ticks_us(),
            self.__time)/1000000
        self.__time = utime.ticks_us()
        self.__c    = (1-self.__alpha) * (self.__c + rate * self.__delta) +
            self.__alpha * angle
        return self.__c

#__> MPU6050
```

```
class MPU6050(__I2CHelper):

    #__>       PROPERTIES      <__#

    @property
    def device_id(self) -> int:
        return self.__readBits(0x75, 0x6, 0x6)

    @property
    def data(self) -> namedtuple:
        data = None
        ax, ay, az, gx, gy, gz = 0, 0, 0, 0, 0, 0
        if self.__usefifo:
            data = self.__get_fifo_packet(12)
            if not data is None:
                ax = data[0] / self.__accfact
                ay = data[1] / self.__accfact
                az = data[2] / self.__accfact
                gx = data[3] / self.__gyrofact
                gy = data[4] / self.__gyrofact
                gz = data[5] / self.__gyrofact

        if (not self.__usefifo) or (data is None):
            data = struct.unpack(">hhhhhhh", self.__readBytes(0x3b, 14))
            ax = data[0] / self.__accfact
            ay = data[1] / self.__accfact
            az = data[2] / self.__accfact
            gx = data[4] / self.__gyrofact
            gy = data[5] / self.__gyrofact
            gz = data[6] / self.__gyrofact

        if self.__filtered & (FILTER_GYRO | FILTER_ACCEL):
           if self.__filtered & FILTER_GYRO:
               gx = self.__fil_gx.kalman(gx)
               gy = self.__fil_gy.kalman(gy)
               gz = self.__fil_gz.kalman(gz)

           if self.__filtered & FILTER_ACCEL:
               ax = self.__fil_ax.kalman(ax)
               ay = self.__fil_ay.kalman(ay)
               az = self.__fil_az.kalman(az)
```

```
        return _D(ax, ay, az, gx, gy, gz)

    @property
    def angles(self) -> tuple:
        if (self.__aftype & ANGLE_BOTH) and (self.__filtered &
            FILTER_ANGLES):
            return self.__filtered_angles()

        ax, ay, az, gx, gy, gz = self.data
        z2    = az**2
        roll  = math.atan(ax/math.sqrt(ay**2+z2))*_R2D
        pitch = math.atan(ay/math.sqrt(ax**2+z2))*_R2D

        return _A(roll, pitch)

    @property
    def int_angles(self) -> tuple:
        if (self.__aftype & ANGLE_BOTH) and (self.__filtered &
            FILTER_ANGLES):
            return self.__filtered_angles(True)

        ax, ay, az, gx, gy, gz = self.data
        z2    = az**2
        roll  = math.atan(ax/math.sqrt(ay**2+z2))*_R2D
        pitch = math.atan(ay/math.sqrt(ax**2+z2))*_R2D

        return _A(int(roll), int(pitch))

    @property
    def connected(self) -> bool:
        return self.device_id == 0x34

    @property
    def passed_self_test(self) -> bool:
        if self.connected:
            self.__enable_tests(True)                    #enable tests
            accel  = self.__test(*self.__accel_st_data) #test accelerometer
            gyro   = self.__test(*self.__gyro_st_data) #test gyroscope
            self.__enable_tests(False) #disble tests and revert to pre-test
                states
            if not (accel and gyro):
                if not gyro:
```

```
                print(_F.format('Gyroscope'))
            if not accel:
                print(_F.format('Accelerometer'))
            return False
        else:
            print('No Connection To Device')
            return False
        return True

    @property
    def celsius(self) -> float:
        return self.__temperature/340 + 36.53

    @property
    def fahrenheit(self) -> float:
        return self.celsius * 1.8 + 32

    #__> CONSTRUCTOR
    def __init__(self, bus:int, sda, scl, ofs:tuple=None, intr=None,
        callback=None, angles:bool=False, clock:int=CLK_PLL_XGYRO,
        gyro:int=GYRO_FS_500, accel:int=ACCEL_FS_2, dlpf:int=DLPF_BW_188,
        rate:int=4, filtered:int=NONE, anglefilter:int=NONE, R:float=0.003,
        Q:float=0.001, A:float=0.8, addr:int=0x68, freq:int=400000) -> None:
        super().__init__(bus, sda, scl, addr, freq)
        self.__accsense , self.__accfact , self.__accfs = 0, 0, accel
        self.__gyrosense, self.__gyrofact, self.__gyrofs = 0, 0, gyro
        self.__rate    , self.__dlpf                = rate, dlpf
        self.__intr    , self.__usefifo             = None, False
        self.__fil_r   , self.__fil_p               = None, None
        self.__fil_gx  , self.__fil_gy , self.__fil_gz = None, None, None
        self.__fil_ax  , self.__fil_ay , self.__fil_az = None, None, None
        self.__useangles, self.__filtered, self.__aftype = angles,
            filtered, anglefilter

        if filtered & FILTER_ANGLES:
            self.__fil_r , self.__fil_p             = __Filters(R, Q, A),
                __Filters(R, Q, A)
        if filtered & FILTER_GYRO:
            self.__fil_gx, self.__fil_gy, self.__fil_gz = __Filters(R, Q,
                A), __Filters(R, Q, A), __Filters(R, Q, A)
        if filtered & FILTER_ACCEL:
            self.__fil_ax, self.__fil_ay, self.__fil_az = __Filters(R, Q,
```

```
            A), __Filters(R, Q, A), __Filters(R, Q, A)

        # self.__enable_interrupts(False)
        self.__enable_fifo (False)
        self.__enable_tests(False)
        self.__enable_sleep(False)

        self.__set_clock(clock)
        self.__set_gyro (gyro )
        self.__set_accel(accel)
        self.__set_rate (rate )
        self.__set_dlpf (dlpf )

        utime.sleep_ms(100)                 #a moment to stabilize
        if (filtered & (FILTER_ANGLES | FILTER_GYRO)) or (anglefilter &
            ANGLE_KAL):
            for _ in range(50): self.angles #this primes the Kalman filters
        elif not anglefilter & NONE:
            for _ in range(10): self.angles #primes delta for complimentary
                filter

        self.__time = utime.ticks_us()
        self.__delta = utime.ticks_diff(utime.ticks_us(),
            self.__time)/1000000
        self.__cx, self.__cy = self.angles

        if isinstance(ofs, tuple):
            self.__set_offsets(*ofs) if (len(ofs) == 6) else
                self.__calibrate(6)
        else:
            self.__calibrate(6)

        if (not intr is None) and (not callback is None):
            self.__intr    = Pin(intr, Pin.IN, Pin.PULL_DOWN) if
                isinstance(intr, int) else Pin(sda)
            self.__intr.irq(self.__handler, Pin.IRQ_RISING)
            self.__callback = callback
            self.__usefifo = True
            self.__reset_fifo()
            self.__enable_fifo(True)

    #__>       PUBLIC METHODS     <__#
```

```
#INTERRUPT CONTROL____>
def start(self) -> None:
    if not self.__usefifo is None:
        self.__enable_interrupts(True)

def stop(self) -> None:
    if not self.__usefifo is None:
        self.__enable_interrupts(False)

#PRINT_________________>
def print_offsets(self) -> None:
    ax, ay, az = self.__readWords(0x6 , 3)
    gx, gy, gz = self.__readWords(0x13, 3)
    print('ofs=({}, {}, {}, {}, {}, {})'.format(ax, ay, az, gx, gy, gz))

def print_data(self):
    self.print_from_data(self.data)

def print_from_data(self, data:tuple) -> None:
    ax, ay, az, gx, gy, gz = data
    a = _OUT.format('ACCELEROMETER', _W[ax<0], abs(ax), _W[ay<0],
        abs(ay), _W[az<0], abs(az))
    g = _OUT.format('GYROSCOPE' , _W[gx<0], abs(gx), _W[gy<0], abs(gy),
        _W[gz<0], abs(gz))
    print(_C.format(a, g))

def print_angles(self, asint:bool=False) -> None:
    if asint:
        self.print_from_angles(self.int_angles)
        return
    self.print_from_angles(self.angles)

def print_from_angles(self, angles:tuple) -> None:
    r, p = angles[0:2]
    print('[{:<16}] roll: {}{:<10.2f} pitch:
        {}{:<10.2f}'.format('ANGLES', _W[r<0], abs(r), _W[p<0], abs(p)))

def print_celsius(self) -> None:
    print('[{:<16}] {:>6.2f} C'.format('TEMPERATURE', self.celsius))

def print_fahrenheit(self) -> None:
```

```
        print('[{:<16}] {:>6.2f} F'.format('TEMPERATURE', self.fahrenheit))

    def print_all(self):
        self.print_celsius()
        self.print_fahrenheit()
        self.print_angles()
        self.print_data()

    #__>       PRIVATE PROPERTIES   <__#

    @property
    def __temperature(self) -> int:
        return struct.unpack('>h', self.__readBytes(0x41, 2))[0]

    @property
    def __fifo_count(self) -> int:
        msb = self.__readByte(0x72)
        lsb = self.__readByte(0x73)
        return (msb << 8) | lsb

    @property
    def __accel_st_data(self) -> tuple:         #self-test data
        a  = self.__readBit(0x1c, 0x7)
        b  = self.__readBit(0x1c, 0x6)
        c  = self.__readBit(0x1c, 0x5)
        v  = self.__readByte(0x10)
        x_ = self.__readByte(0xd)
        x  = ((x_>>3) & 0x1C) | ((v>>4) & 0x03)
        y_ = self.__readByte(0xe)
        y  = ((y_>>3) & 0x1C) | ((v>>2) & 0x03)
        z_ = self.__readByte(0xf)
        z  = ((z_>>3) & 0x1C) | (v & 0x03)
        return (a, b, c), (x, y, z)

    @property
    def __gyro_st_data(self) -> tuple:          #self-test data
        a = self.__readBit(0x1b, 0x7)
        b = self.__readBit(0x1b, 0x6)
        c = self.__readBit(0x1b, 0x5)
        x = self.__readByte(0xd) & 0x1F
        y = self.__readByte(0xe) & 0x1F
        z = self.__readByte(0xf) & 0x1F
```

```
        return (a, b, c), (x, y, z)

    #__>       PRIVATE METHODS   <__#

    #SETTERS_____________>
    def __set_dlpf(self, dlpf:int):
        self.__writeBits(0x1a, 0x2, 0x3, dlpf)

    def __set_rate(self, rate:int) -> None:
        self.__writeByte(0x19, rate)

    def __set_gyro(self, rng:int) -> None:
        self.__gyrosense = [250, 500, 1000, 2000][rng]
        self.__gyrofact = [131, 66.5, 32.8, 16.4][rng]
        self.__writeBits(0x1b, 0x4, 0x2, rng)

    def __set_accel(self, rng:int) -> None:
        self.__accsense = [2, 4, 8, 16][rng]
        self.__accfact = 32768.0/self.__accsense
        self.__writeBits(0x1c, 0x4, 0x2, rng)

    def __set_clock(self, source:int) -> None:
        self.__writeBits(0x6b, 0x2, 0x3, source)

    #MISC_________________>
    def __handler(self, pin:Pin) -> None:       #interrupt handler
        if (not self.__intr is None) and (not self.__callback is None):
            if self.__useangles:
                self.__callback(self.angles)
                return
            self.__callback(self.data)

    def __filtered_angles(self, asint:bool=False):              #manages
        all angle filtering
        smps  = self.__rate//2
        fx, fy = [0.00]*(smps), [0.00]*(smps)
        for s in range(smps):
            ax, ay, az, gx, gy, gz = self.data
            z2    = az**2
            ay2z2 = math.sqrt(ay**2+z2)
            ax2z2 = math.sqrt(ax**2+z2)
            roll  = 0 if not ay2z2 else math.atan(ax/ay2z2)*_R2D
```

```
            pitch = 0 if not ax2z2 else math.atan(ay/ax2z2)*_R2D
            if self.__aftype & ANGLE_KAL:
                roll  = self.__fil_r.kalman(roll)
                pitch = self.__fil_p.kalman(pitch)
            if self.__aftype & ANGLE_COMP:
                roll  = self.__fil_r.complementary(gx, roll )
                pitch = self.__fil_p.complementary(gy, pitch)
            fx[s], fy[s] = roll, pitch
            utime.sleep_us(100)
        roll  = sum(fx)/smps
        pitch = sum(fy)/smps

        if asint:
            return _A(int(roll), int(pitch))

        return _A(roll, pitch)

    def __enable_sleep(self, e:bool) -> None:
        self.__writeBit(0x6b, 0x6, e)

    def __enable_interrupts(self, e:bool) -> None:
        self.__writeByte(0x38, (0x11 if e else 0x00))

    #SELF-TEST____________>
    def __enable_tests(self, e:bool) -> None:
        #accelerometer test
        self.__set_accel(ACCEL_FS_8 if e else self.__accfs)
        self.__writeBit(0x1c, 0x7, e)
        self.__writeBit(0x1c, 0x6, e)
        self.__writeBit(0x1c, 0x5, e)
        #gyroscope test
        self.__set_gyro(GYRO_FS_500 if e else self.__gyrofs)
        self.__writeBit(0x1b, 0x7, e)
        self.__writeBit(0x1b, 0x6, e)
        self.__writeBit(0x1b, 0x5, e)

    def __test(self, st_data:tuple, trim:tuple) -> bool:
        spec = range(-14, 15) #factory min/max specs
        return sum([True if not t else ((s-t)//t in spec) for t, s in
            zip(trim, st_data)]) == 3

    #FIFO_________________>
```

```
    def __reset_fifo(self) -> None:
        self.__writeBit(0x6a, 0x6, False) #disable FIFO
        self.__writeBit(0x6a, 0x2, True) #reset FIFO
        self.__writeBit(0x6a, 0x6, True) #enable FIFO

    def __enable_fifo(self, e:bool) -> None:
        self.__writeBit(0x6a, 0x6, e)               #enable FIFO
        self.__writeByte(0x23, (0x78 if e else 0x00)) #enable Gyro and
            Accel registers

    def __fifo_bytes(self, length:int) -> bytearray:
        #read from FIFO buffer
        return self.__readBytes(0x74, length) if length > 0 else None

    def __get_fifo_packet(self, length:int=12) -> bytearray:
        fifoC = self.__fifo_count
        extra = fifoC%length
        self.__fifo_bytes(extra)
        fifoC = self.__fifo_count
        return None if fifoC < length else struct.unpack(">hhhhhh",
            self.__fifo_bytes(fifoC)[-length:])

    #DMP_________________>
    def __reset_dmp(self) -> None:
        self.__writeBit(0x6a, 0x3, True)

    def __enable_dmp(self, e:bool) -> None:
        self.__writeBit(0x6a, 0x7, e)

    #CALIBRATE___________>
    def __calibrate(self, loops:int) -> None:
        x = (100 - int((loops - 1) * (0 - 20) / (5 - 1) + 20)) * .01
        kP, kI = 0.3*x, 90*x
        self.__pid(0x43, kP, kI, loops) #calibrate gyroscope
        kP, kI = 0.3*x, 20*x
        self.__pid(0x3b, kP, kI, loops) #calibrate accelerometer
        self.print_offsets()

    def __pid(self, readaddr:int, kP:float, kI:float, loops:int) -> None:
        saveaddr = 0x06 if readaddr == 0x3B else 0x13
        data , reading, bitzero = 0 , 0.0, [0]*3
        shift, esample, esum  = 2  , 0  , 0
```

```
error, pterm , iterm  = 0.0, 0.0, [0.0]*3

for i in range(3):
    data = self.__readWords(saveaddr + (i * shift), 1)[0]
    reading = data
    if not (saveaddr == 0x13):
        bitzero[i] = data & 1
        iterm[i]  = reading * 8
    else:
        iterm[i]  = reading * 4

for L in range(loops):
    esample = 0
    for c in range(100):
        esum = 0
        for i in range(3):
            data     = self.__readWords(readaddr + (i * 2), 1)[0]
            reading  = data

            if ((readaddr == 0x3B) and (i == 2)): reading -= 16384

            error    = -reading
            esum    += abs(reading)
            pterm    = kP * error
            iterm[i] += (error * 0.001) * kI
            data     = (round((pterm + iterm[i]) / 8) & 0xFFFE) |
                bitzero[i] if saveaddr != 0x13 else round((pterm +
                iterm[i]) / 4)

            self.__writeWords(saveaddr + (i * shift), 1, data)

        c = 0 if (c == 99) and (esum > 1000) else c

        if (esum * (.05 if readaddr == 0x3B else 1)) < 5: esample+=
            1
        if (esum < 100) and (c > 10) and (esample >= 10): break
        utime.sleep_ms(1)

    kP *= .75
    kI *= .75
    for i in range(3):
        data = (round(iterm[i] / 8) & 0xFFFE) | bitzero[i] if not
```

```
                (saveaddr == 0x13) else round(iterm[i] / 4)
            self.__writeWords(saveaddr + (i * shift), 1, data)

    self.__reset_fifo()
    self.__reset_dmp()

def __set_offsets(self, ax:int, ay:int, az:int, gx:int, gy:int, gz:int)
    -> None:
    self.__writeWords(0x6 , 1, ax)
    self.__writeWords(0x8 , 1, ay)
    self.__writeWords(0xa , 1, az)
    self.__writeWords(0x13, 1, gx)
    self.__writeWords(0x15, 1, gy)
    self.__writeWords(0x17, 1, gz)
```

Listing C.2 – *mpu6050.py*