



Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia Eletrônica

Desenvolvimento de Sistema de Captura de Movimento Humano Utilizando IMU

Autor: Eduardo Sousa Sales Rodrigues
Orientadora: Profa. Dra. Lourdes Mattos Brasil

Brasília, DF

2018



Eduardo Sousa Sales Rodrigues

Desenvolvimento de Sistema de Captura de Movimento Humano Utilizando IMU

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

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientadora: Profa. Dra. Lourdes Mattos Brasil

Coorientador: MSc. Roberto Aguiar Lima

Brasília, DF

2018

Eduardo Sousa Sales Rodrigues

Desenvolvimento de Sistema de Captura de Movimento Humano Utilizando IMU/
Eduardo Sousa Sales Rodrigues. – Brasília, DF, 2018-
89 p. : il.

Orientadora: Profa. Dra. Lourdes Mattos Brasil

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2018.

1. IMU.2. Prevenção de lesão.I. Profa. Dra. Lourdes Mattos Brasil.II. Universidade de
Brasília.III. Faculdade UnB Gama.IV. Desenvolvimento de Sistema de Captura de Movi-
mento Humano Utilizando IMU

CDU 02:141:005.6

Eduardo Sousa Sales Rodrigues

Desenvolvimento de Sistema de Captura de Movimento Humano Utilizando IMU

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

Trabalho aprovado. Brasília, DF, 10 de dezembro de 2018.

Profa. Dra. Lourdes Mattos Brasil
Orientadora

MSc. Roberto Aguiar Lima
Coorientador

Bel. Ithallo Junior Alves Guimarães
Convidado

Bel. Renata Menezes Lopes
Convidada

Brasília, DF
2018

*"Não vos conformeis com este mundo,
mas transformai-vos pela renovação do vosso espírito,
para que possais discernir qual é a vontade de Deus:
o que é bom, o que Lhe agrada e o que é perfeito."
(Bíblia Sagrada, Romanos 12, 2)*

RESUMO

O objetivo deste trabalho é o desenvolvimento de uma Unidade de Medição Inercial (IMU) para aquisição de parâmetros de movimento e posições corporais em humanos, a fim de auxiliar pesquisas futuras na área de prevenção de lesões musculoesqueléticas. Existem muitos sensores capazes de adquirir dados sobre movimento, mas a maioria não é acessível ou não consegue detalhar certos parâmetros úteis, por exemplo, velocidade e aceleração. A prevenção de lesão é muito importante para atletas de alto desempenho, pois os esportistas conseguem permanecer mais tempo treinando e atingem melhores marcas. O MPU6050 é um sensor inercial feito a partir de tecnologia *Micro Electro Mechanical System* (MEMS), que integra em um único *chip*, 3 sensores de aceleração e 3 sensores de velocidade angular. Este sensor comunica-se por protocolo *I²C*, o que permite a utilização das plataformas Arduíno e Wemos, que possuem microcontroladores programáveis. O microcontrolador do Arduíno se comunica serialmente ao computador, por meio de um cabo *Universal Serial Bus* (USB). O do Wemos pode se comunicar via *wireless* com um protocolo *web*. A linguagem de programação *Python* pode se comunicar com o Arduino por meio de protocolo de comunicação serial, ou com o Wemos, com protocolo HTTP, e assim possibilitar que os dados adquiridos pelo sensor MPU6050 sejam armazenados no computador. Os dados podem ser manipulados para obtenção de diversas informações: aceleração linear, velocidade angular, posições e ângulos. Então, foi feito um protótipo funcional com uma *interface* via terminal, para utilização dos pesquisadores do laboratório LIS da UnB/FGA, necessitando de futuros testes com movimentação em humanos para validar a precisão e acurácia do IMU.

Palavras-chaves: Unidade de Medição Inercial, Movimento Humano, Eletrônica.

ABSTRACT

The objective of this work is the development of an Inertial Measurement Unit (IMU) to acquire parameters of human body movement and positions to assist research in the area of preventing musculoskeletal injury. There are many sensors capable of acquiring data on motion, but most of them are not accessible or do not detail certain useful parameters(e.g. speed and acceleration). Injury prevention is very important for high-performance athletes, because of preventing injuries, athletes can stay longer in training and achieve better marks. The MPU6050 is an inertial sensor made with Micro Electro Mechanical System (MEMS) technology, integrating on a single chip, 3 acceleration sensors and 3 angular speed sensors. This sensor communicates by I^2C protocol, which allows the use of the Arduino and Wemos platforms, which have programmable micro-controller. This microcontrollers. The Arduino communicates serially to the computer via a Universal Serial Bus (USB) cable. Wemos can communicate wirelessly with a web protocol. The Python programming language can communicate with the Arduino through a serial communication protocol or Wemos with HTTP protocol and thus allow the data acquired by the MPU6050 sensor to be stored on the computer. The data can be manipulated to obtain various information (e.g. linear acceleration, angular velocity, positions and angles). So, a functional prototype was done with a simple interface for the use of the researchers of the laboratory LIS of the UnB/FGA, needing of future tests to verify the accuracy and precision of the IMU.

Key-words: Inertial Measurement Unit, IMU, Preventing Musculoskeletal Injury, Electronic.

LISTA DE ILUSTRAÇÕES

| | |
|---|----|
| Figura 1 – Ilustração esquemática de componentes MEMS | 26 |
| Figura 2 – Um motor MEMS ao lado de uma fio de cabelo humano. | 26 |
| Figura 3 – Eixos de orientação. Adaptada de INVENSENSE (2013). | 27 |
| Figura 4 – Sensor inercial baseado em 2 sensores | 27 |
| Figura 5 – Princípio de funcionamento do acelerômetro. Adaptada de Sanjeev (2018). . | 29 |
| Figura 6 – Imagem de microscopia eletrônica por varredura de um giroscópio diapasão. | 29 |
| Figura 7 – Arduino Nano | 33 |
| Figura 8 – WeMos D1 - R2 | 34 |
| Figura 9 – Esquemático de conexão do MPU6050 com Arduíno Nano (Imagen do autor). | 39 |
| Figura 10 – Transmissão de dados do MPU6050 para Arduíno (Imagen do autor). . . . | 39 |
| Figura 11 – Transferência de dados com o protocolo I^2C | 40 |
| Figura 12 – Fluxograma do <i>software</i> de leitura dos dados. (Imagen do autor). | 42 |
| Figura 13 – Fluxograma do <i>software</i> que salva os dados lidos (Imagen do autor). | 45 |
| Figura 14 – Demonstração da posição do sensor (Imagen do autor). | 46 |
| Figura 15 – Demonstração direção de giro do sensor (Imagen do autor). | 46 |
| Figura 16 – Protótipo na <i>Protoboard</i> (Imagen do autor). | 49 |
| Figura 17 – MPU6050 na <i>Protoboard</i> (Imagen do autor). | 49 |
| Figura 18 – Segundo Protótipo com o Velcro (Imagen do autor). | 50 |
| Figura 19 – Protótipo Fixado ao Punho e a um Cubo (Imagen do autor). | 50 |
| Figura 20 – Tela Inicial (Imagen do autor). | 51 |
| Figura 21 – Lista com as escalas (Imagen do autor). | 51 |
| Figura 22 – Programa mostrando os resultados (Imagen do autor). | 51 |
| Figura 23 – Gráficos com os 3 Eixos alinhados com a Normal. (Imagen do autor). . . . | 52 |
| Figura 24 – Gráficos para Teste do Giroscópio (Imagen do autor). | 53 |
| Figura 25 – Placa Perfurada (Imagen do autor). | 53 |
| Figura 26 – Cabo Flat com Sensor (Imagen do autor). | 53 |
| Figura 27 – 3 sensores Multiplexados (Imagen do autor). | 54 |
| Figura 28 – Final da Execução dos 3 Sensores (Imagen do autor). | 54 |
| Figura 29 – Wemos com Sensor MPU6050 (Imagen do autor). | 54 |
| Figura 30 – Wemos Configurada como Ponto de Acesso WiFi (Imagen do autor). . . . | 55 |
| Figura 31 – Final da Execução da Leitura Sem Fio (Imagen do autor). | 55 |
| Figura 32 – Frequêcia x Distância (Imagen do autor). | 55 |

LISTA DE TABELAS

| | |
|---|----|
| Tabela 1 – Tabela de comparação dos modelos de Arduíno. Adaptado de Smith (2016). | 32 |
| Tabela 2 – Conexões entre Arduíno e MPU6050 | 38 |
| Tabela 3 – Conexões entre Arduíno e 3 Sensores | 40 |
| Tabela 4 – Conexões entre Wemos e Sensor | 40 |
| Tabela 5 – História de Usuário | 44 |

LISTA DE ABREVIATURAS E SIGLAS

| | |
|-------|---|
| ADC | <i>Analog Digital Converter</i> - Conversor Analógico-Digital |
| DC | <i>Direct Current</i> - Corrente Contínua |
| DMP | <i>Digital Motion Processor</i> - Processador Digital de Movimento |
| FGA | Faculdade Gama |
| HARM | <i>High-Aspect-Ratio Micro-machining</i> - Micromaqinação de Alta Proporção |
| HTTP | <i>HyperText Transfer Protocol</i> - Protocolo de Transferência de Hipertexto |
| I2C | <i>Inter-Integrated Circuit</i> - Circuito Inter-Integrado |
| IC | <i>Integrated Circuit</i> - Circuito Integrado |
| IDE | <i>Integrated Development Environment</i> - Ambiente de Desenvolvimento Integrado |
| IETF | <i>Internet Engineering Task Force</i> - Força Tarefa de Engenharia da Internet |
| IMU | <i>Inertial Measurement Unit</i> - Unidade de Medição Inercial |
| LIS | Laboratório de Informática em Saúde |
| MEMS | <i>Micro Electro Mechanical System</i> - Sistema Micro-eletromecânico |
| MOCAP | <i>Motion Capture</i> - Captura de Movimento |
| MST | <i>Microsystems Technology</i> - Tecnologia de Microssistemas |
| RFC | <i>Request for Comments</i> - Pedido para Comentário |
| PDIP | <i>Plastic Dual-In-line Package</i> - Pacote de Dupla Linha de Plástico |
| PQFP | <i>Plastic Quad Flat Pack</i> - Pacote de Encapsulamento Quadrado de Plástico |
| SCL | <i>Serial Clock</i> - "Relógio" Serial |
| SDA | <i>Serial Data</i> - Dados Seriais |
| sEMG | <i>Surface Electromyography</i> - Eletromiografia dE Superfície |
| UnB | Universidade de Brasília |

| | |
|-----|--|
| URI | <i>Uniform Resource Identifier</i> - Identificador Uniforme de Recurso |
| URL | <i>Uniform Resource Locator</i> - Localizador Uniforme de Recursos |
| USB | <i>Universal Serial Bus</i> - Barramento Serial Universal |

LISTA DE SÍMBOLOS

ω Velocidade angular em rad/s

\oslash Diâmetro

\forall Para todo

π Constante pi

SUMÁRIO

| | | |
|------------|--|-----------|
| 1 | INTRODUÇÃO | 21 |
| 1.1 | Objetivos | 22 |
| 1.1.1 | Objetivo Geral | 22 |
| 1.1.2 | Objetivos Específicos | 22 |
| 1.2 | Justificativa | 22 |
| 2 | REFERENCIAL TEÓRICO | 23 |
| 2.1 | Prevenção de Lesões | 23 |
| 2.2 | O processo de amostragem | 24 |
| 2.3 | MEMS | 25 |
| 2.4 | IMU | 26 |
| 2.4.1 | Acelerômetro | 28 |
| 2.4.2 | Giroscópio | 28 |
| 2.4.3 | MPU6050 | 30 |
| 2.4.4 | Arduíno | 31 |
| 2.4.5 | Esp8266 | 33 |
| 2.5 | Python | 34 |
| 2.6 | Protocolo HTTP | 35 |
| 3 | METODOLOGIA | 37 |
| 3.1 | Projeto de Hardware | 37 |
| 3.1.1 | Materiais Utilizados | 37 |
| 3.1.2 | Métodos | 38 |
| 3.2 | Projeto de Software | 41 |
| 3.2.1 | Softwares para o Arduíno | 41 |
| 3.2.2 | Software para o Wemos D1 | 42 |
| 3.2.3 | Software para Salvar os Dados | 43 |
| 3.2.4 | Software para Teste de Comunicação Sem Fio | 44 |
| 3.3 | Teste | 44 |
| 4 | RESULTADOS E DISCUSSÃO | 49 |
| 4.1 | Resultados | 49 |
| 4.2 | Discussão | 56 |
| 5 | CONCLUSÃO | 57 |

| | |
|--|----|
| REFERÊNCIAS | 59 |
| APÊNDICES | 63 |
| APÊNDICE A – CÓDIGO FONTE DO ARDUÍNO COM UM SENSOR | 65 |
| APÊNDICE B – CÓDIGO ARDUINO COM 3 SENsoRES | 69 |
| APÊNDICE C – CÓDIGO WEMOS D1 COMO PONTO DE ACESSO | 73 |
| APÊNDICE D – CÓDIGO DO PROGRAMA EM PYTHON COM UM SENSOR | 77 |
| APÊNDICE E – CÓDIGO DO PROGRAMA EM PYTHON COM 3 SEnsoRES | 83 |
| APÊNDICE F – FUNÇÕES DE COMUNICAÇÃO | 87 |
| APÊNDICE G – CÓDIGO DO PROGRAMA EM PYTHON PARA LEI- TURA HTML | 89 |

1 INTRODUÇÃO

O interesse em analisar por meio de conceitos físicos o movimento humano é bastante antigo. Estudos clássicos como os de Aristóteles deixam claro que a relevância de analisar o movimento por meio de ensaio físico data do século III a.C. Porém, mesmo com o estudo do movimento sendo antigo, a Biomecânica se consolidou como uma ciência e disciplina acadêmica no Brasil recentemente. Historicamente falando, apenas na década de 60 com influência do governo da República Federal da Alemanha que apoiou algumas universidades brasileiras a introduzir a Biomecânica nos cursos de Educação Física (ACQUESTA et al., 2008).

É essencial, para associar medidas físicas ao movimento, estudos relacionados à cinética e cinemática do movimento precisos. Com os dados adquiridos é possível realizar diagnóstico de desempenho em atletas, pesquisas para prevenção de lesão musculoesquelético e adquirir melhor entendimento do movimento humano como um todo (MCGINNIS, 2013).

Dessa forma, o estudo da biomecânica do movimento se tornou importante para desenvolver atletas em diversos esportes. No caso de um velocista, a coleta e a análise de dados sobre a velocidade, aceleração, postura e qualidade de movimento são úteis para possibilitar a evolução de seu desempenho (OKAZAKI et al., 2012)

São necessários equipamentos adequados para realizar a coleta de dados que possibilitem aos profissionais da área de esportes procurar por diferentes estratégia e métodos de treinamento para obtenção de melhores resultados no desempenho dos atletas. Estes equipamentos estão sendo desenvolvidos (OKAZAKI et al., 2012).

As unidades de medição inercias (IMU - *Inertial Measurement Units*) são um exemplo desses equipamentos e tornaram-se ferramentas muito úteis para aquisição de informações relacionadas ao movimento corporal. Esses sensores são baratos, pequenos e permitem mobilidade quando integrados com módulos de comunicação sem fio. Porém, sua utilização necessita de um conhecimento técnico e matemático específico (OBERLANDER, 2015).

A evolução desses sensores permitiu que a movimentação corporal humana fosse estudada em diversos ambientes e situações, sendo necessário apenas algumas unidades de sensores colocados nos pontos adequados do corpo. E, assim, os dados adquiridos são capazes de proporcionar algumas variáveis importantes como: aceleração, velocidade angular, velocidade linear, altura, direção e ângulos, todos de forma não invasiva e sem a necessidade de ambientes fechados (CHANG; GEORGY; EL-SHEIMY, 2016). A partir desses dados é possível fazer análises preditivas, diagnósticos de lesões, movimentos assíncronos de membros corporais, má postura entre outros.

1.1 OBJETIVOS

Os principais objetivos desse trabalho de conclusão de curso são os seguintes:

1.1.1 OBJETIVO GERAL

Desenvolvimento de um IMU para aquisição de movimentos e posições corporais em humanos, a fim de auxiliar em pesquisas na área de prevenção de lesões musculoesqueléticas.

1.1.2 OBJETIVOS ESPECÍFICOS

- Identificar quais serão parâmetros que o IMU deve ser capaz de calcular;
- Desenvolver e realizar testes para leitura de mais de um sensor simultaneamente;
- Estabelecer a comunicação sem fio entre o IMU e um computador;
- Desenvolver Kit's para utilização em pesquisas futuras do Laboratório de Informática em Saúde (LIS) da Universidade de Brasília (UnB), Faculdade Gama (FGA).

1.2 JUSTIFICATIVA

O sensor IMU tem inúmeras aplicações na área da saúde e dos esportes, como avaliação de postura, assimetria de movimentos, análise de marcha, movimentos específicos de esportes entre outras. É um sensor essencial para pesquisas nessas áreas. São poucos os trabalhos que tentam tornar esse sensor mais amigável para utilização por pesquisadores de diversas áreas, que não tem o conhecimento matemático e de linguagens de programação necessários para trabalhar com ele (OBERLANDER, 2015)(CHANG; GEORGY; EL-SHEIMY, 2016).

Além disso, este sensor, exige um custo financeiro menor do que outros equipamentos utilizados nessa área de estudo, como o *Motion Capture* (MOCAP), por exemplo, é um equipamento que exige um ambiente interno com espaço específico, muitas câmeras para melhor precisão e sistemas que normalmente tem um custo financeiro alto (CHANG; GEORGY; EL-SHEIMY, 2016).

Esse sensor deverá futuramente ser utilizado para pesquisas no laboratório LIS da UnB e auxiliar estudantes de graduação e mestrado em seus trabalhos.

2 REFERENCIAL TEÓRICO

2.1 PREVENÇÃO DE LESÕES

O desempenho máximo de atletas pode ser afetado por lesões musculoesqueléticas, fazendo com que seus resultados não sejam os melhores, impedindo de participar em competições ou causando a saída do esporte precocemente. O estudo do corpo humano permite obter uma análise operacional dos atletas, ou seja, demonstrar diferentes parâmetros, tais como do sistema proprioceptivo, da estabilidade articular e da força muscular durante movimentos específicos, a mecânica dos tecidos biológicos utilizados no esporte, bem como do estudo morfológico dos atletas. Podendo assim promover a prevenção de lesões (MIZIARA, 2014).

O esforço físico feito pelo atleta durante práticas esportivas lhes proporcionam diversas consequências fisiológicas. O ramo do conhecimento dedicado ao estudo de efeitos fisiológicos agudos e crônicos dos exercícios físicos sobre os diversos sistemas corporais é a Fisiologia do Exercício. Contudo, os métodos de medição ou determinação destes estados do corpo humano em pleno emprego do seu esforço são importantes para diagnosticar e prevenir lesões, fortalecimento de musculatura e tendões, ou seja, todo o desempenho do atleta (ROCHA, 2005).

Estes métodos utilizados analisam algumas características importantes, tais como velocidade, aceleração, posicionamento no espaço e necessitam, de certa maneira, mais do que um profissional com olhar treinado para uma análise real do movimento do corpo humano, dado que a prevenção de lesões e o desempenho dos atletas estão ligados à biomecânica esportiva deles (AMADIO, 2000).

Dessa forma, as avaliações biomecânicas devem ir além do olhar clínico ao analisar os parâmetros associados a lesões de atletas e estruturarem seus argumentos em sistemas que possam ser capazes de traduzir estes parâmetros (MIZIARA, 2014). Os métodos utilizados para estudar as diversas formas de movimento são: eletromiografia, antropometria, dinamometria, cinemetria (AMADIO; SERRÃO, 2007).

A eletromiografia é a medição da atividade elétrica de músculos que estão sendo utilizados durante a prática esportiva. A antropometria estabelece as propriedades físicas do corpo humano estudado como as medidas geométricas. A dinamometria mede parâmetros como força e distribuição de pressão com equipamentos específicos para cada atividade física. E, por fim, a cinemetria utiliza parâmetros como posição e orientação de partes do corpo humano, que é o tipo de medição tratado neste trabalho (AMADIO; SERRÃO, 2007)(MEDEIROS, 2012).

2.2 O PROCESSO DE AMOSTRAGEM

A cinemetria pode ser realizada através da captura por imagens ou da medição de determinados pontos com sensores inerciais. E para os dois métodos é preciso saber qual a frequência de amostragem, no caso das imagens, o número de *frames* por segundo, necessários para captar o movimento com precisão suficiente para afirmar que ele representa o deslocamento real (AMADIO; SERRÃO, 2007).

Para definição dessa frequência existe o teorema de amostragem Nyquist-Shannon, que é a base para determinar a frequência mínima de amostragem de um sinal. Através do processo de amostragem, um sinal contínuo no tempo, é modificado para um sinal discreto no tempo. O teorema da amostragem assegura que as amostras discretas uniformemente espaçadas são uma representação completa do sinal, se sua frequência máxima é menor do que a metade da taxa de amostragem (MADEIRO; LOPES, 2011).

Para formalizar o conceito, seja $x(t)$ um sinal contínuo no tempo e $X(f)$ sua transformada de Fourier (ou seja, representação do sinal em frequência):

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-i2\pi ft} dt \quad (2.1)$$

O sinal $x(t)$ é limitado em banda, B, se:

$$X(f) = 0 \quad \forall |f| > B \quad (2.2)$$

A condição suficiente para uma exata reconstrução a partir das amostras em uma taxa de amostragem uniforme f_s (em amostras por unidade de tempo) é:

$$f_s > 2B \quad (2.3)$$

$2B$ é a chamada de Taxa de Nyquist e é uma propriedade do sinal limitado em banda, e $f_s/2$ é chamado de Frequência de Nyquist e é um propriedade desse sistema de amostragem.

O tempo T entre as sucessivas amostras é o intervalo de amostragem.

$$T = \frac{1}{f_s} \quad (2.4)$$

O teorema da amostragem garante que é possível a reconstrução exata do sinal $x(t)$ original desde que sejam respeitadas as condições iniciais (OPPENHEIM; WILLSKY; NAWAB, 2010).

A velocidade do movimento humano varia de acordo com o local analisado. Pode ser a velocidade do corpo inteiro, ou de órgãos separadamente. Esta velocidade pode ser medida em frequência e varia entre 10 e 160 Hz (SONG; GODOY, 2016).

Logo a Taxa de Nyquist para amostragem de sinais de movimento humano é de 320 Hz. E o sensor para medir os movimentos deve ter uma frequência de amostragem maior que essa (SONG; GODOY, 2016).

2.3 MEMS

MEMS (*Micro Electro Mechanical Systems* - Sistemas Microeletromecânicos) é uma tecnologia de processamento usada para criar dispositivos integrados ou sistemas que combinam componentes mecânicos e elétricos. Eles são fabricados usando técnicas de processamento em lote de circuitos integrados e podem variar em tamanho entre alguns micrômetros para milímetros. Esses dispositivos (ou sistemas) têm a capacidade de detectar, controlar e atuar na escala micro e gerar efeitos em escala macro (PRIME, 2002).

O termo MEMS, é um acrônimo originado nos Estados Unidos. Também é conhecido como *Microsystems Technology* (MST) na Europa e *Micromachines* no Japão. Independentemente da terminologia, o fator que define um dispositivo MEMS está na maneira como é feito. Enquanto os aparelhos eletrônicos são fabricados usando tecnologia IC (*Integrated Circuit - chip* de computador), os componentes micromecânicos são fabricados por sofisticadas manipulações com silício e outros substratos usando processos de microusinagem. Processos como microusinagem a granel e de superfície, bem como micromaquinção de alta proporção (HARM - *High Aspect Ratio Micromachining*) remove seletivamente partes do silício ou adiciona camadas estruturais adicionais para formar os componentes mecânicos e eletromecânicos. Enquanto circuitos integrados são projetados para explorar as propriedades elétricas do silício, os MEMS aproveitam as propriedades mecânicas do silício ou suas propriedades elétricas e mecânicas (PRIME, 2002).

Na forma mais geral, os MEMS consistem em microestruturas mecânicas, microssensores, microatuadores e microeletrônica, todos integrados no mesmo *chip* de silício (Figura 1). Microsensores detectam mudanças no ambiente do sistema medindo informações ou fenômenos mecânicos, térmicos, magnéticos, químicos ou eletromagnéticos. A microeletrônica processa essa informação e sinaliza aos microatuadores para reagirem e criarem alguma forma de mudanças no meio ambiente (PRIME, 2002).

Os dispositivos MEMS são extremamente pequenos, seus componentes são geralmente microscópicos (Figura 2). Alavancas, engrenagens, pistões, motores e até motores a vapor foram todos fabricados por MEMS. No entanto, a tecnologia MEMS não se refere apenas à miniaturização de componentes mecânicos ou à fabricação de coisas a partir do silício (na verdade, o termo MEMS é enganoso, já que vários dispositivos micromaquinados não são mecânicos em nenhum sentido). MEMS é uma tecnologia de fabricação, um paradigma para projetar e criar dispositivos e sistemas mecânicos complexos, bem como sua eletrônica integrada, utilizando técnicas de fabricação em lote (PRIME, 2002).

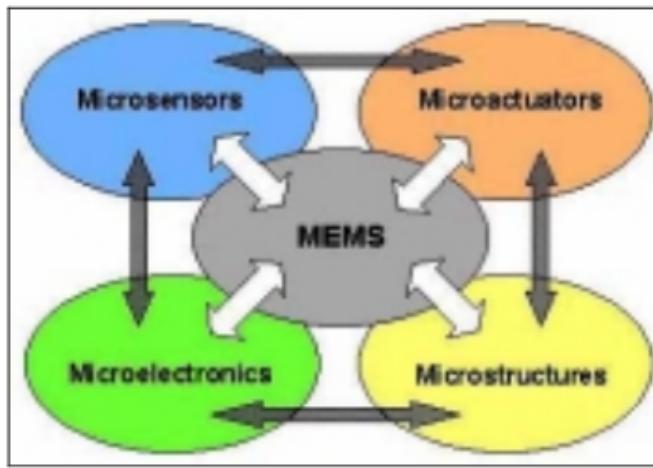


Figura 1 – Ilustração esquemática de componentes MEMS

Fonte: (PRIME, 2002)

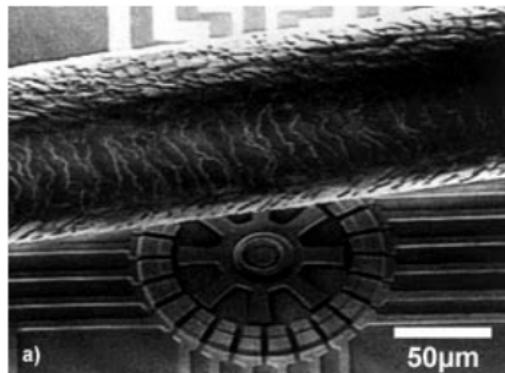


Figura 2 – Um motor MEMS ao lado de uma fio de cabelo humano.

Fonte: (PRIME, 2002)

2.4 IMU

A história do sensor IMU começou em 1930, quando foi usado para auxiliar a navegação de aeronaves e outros dispositivos de grande porte. Por causa de suas restrições, principalmente em tamanho, custo e consumo de energia, o uso do IMU naquele momento era restrito a aplicações em dispositivos grandes e, portanto, impopular para equipamentos de tamanho menor e em grande escala de consumo. Porém, recentemente, o sensor IMU feito por MEMS foi introduzido com uma característica atraente de baixo custo e com poder de processamento. A demanda aumentou e as áreas de aplicação também. Atualmente, diversos fabricantes estão competindo nos melhores projetos de IMU, como *Invensense*, *Honeywell*, *STMicroelectronics*, *Microstrain* e *X-Sens* (AHMAD; GHAZILLA; KHAIRI, 2013).

Os sensores de IMU podem ser capazes de medir diversas variáveis. Os mais comuns são feitos para medir 6 graus de liberdade. São 3 medidas de aceleração linear a partir de um acelerômetro e 3 medidas de aceleração angular feitas por um giroscópio (Figura 3) (SANTOS; VIEIRA; JUNIOR, 2016).

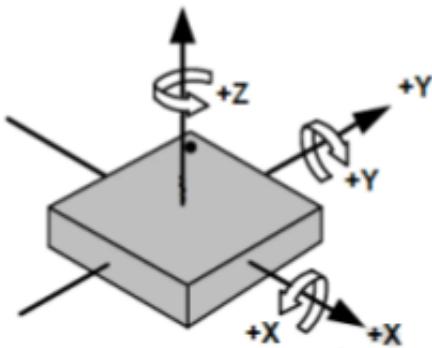


Figura 3 – Eixos de orientação. Adaptada de INVENSENSE (2013).

A vantagem de usar este tipo de IMU é que não será interferido pelo campo magnético externo em torno do sensor quando é usado próximo demais de material ferromagnético. Por outro lado, dependendo do acelerômetro e do giroscópio, pode não ser suficiente para aumentar a precisão da medição devido ao ruído dos sensores e à questão do desvio do giroscópio. Os dados adquiridos pelo IMU são integrados como mostrado na Figura 4 (AHMAD; GHAZILLA; KHAIRI, 2013).

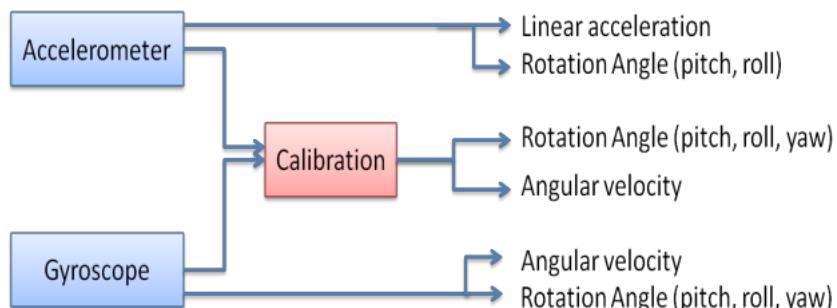


Figura 4 – Sensor inercial baseado em 2 sensores

Fonte: (AHMAD; GHAZILLA; KHAIRI, 2013)

Com essas variáveis adquiridas um padrão de movimento pode ser traçado, bem como podem ser feitas análises sobre o movimento humano. Assim, junto com a sEMG (*Surface Electromyography - Eletromiografia de Superfície*), o uso de sensores inercias tem sido mais solicitado na área de monitoramento de práticas esportivas (HOWARD, 2016).

Porém, a utilização de IMUs com equipamentos de comunicação sem fio, têm mostrado uma certa vantagem sobre o uso de sensores com sEMG. Isso ocorre por conta de a integração entre acelerômetros, giroscópios e dispositivos de transmissão sem fio ser mais simplificada do que a aquisição de dados por eletromiôgrafos sem fio (HOWARD, 2016).

Os testes que foram feitos com uso de sensores inerciais, segundo Howard (2016) foram capazes de produzir dados relacionados com fadiga muscular, desempenho, postura e velocidade. O que possibilita melhores técnicas de treinamento e até mesmo prevenção de lesão.

2.4.1 ACELERÔMETRO

Os sensores de movimento são feitos para aferir taxa de variação de posição, ou seja, o deslocamento que estiver ocorrendo. Assim, se a posição, $x(t)$, de um corpo varia ao longo do tempo, então será obtida sua velocidade, $v(t)$, derivando essa mudança de posição ao longo do tempo como mostra a Equação 2.5. A Equação 2.6 mostra que ao derivar a variação de velocidade ao longo do tempo será calculada a aceleração, $a(t)$, do corpo (NUSSENZVEIG, 2013).

$$v(t) = \frac{dx(t)}{dt} \quad (2.5)$$

$$a(t) = \frac{dv(t)}{dt} = \frac{d^2x(t)}{dt^2} \quad (2.6)$$

Um acelerômetro é capaz de medir a aceleração, e a partir desse dado é possível obter, também, a velocidade e a posição. Basta fazer a operação inversa da derivada, que pode ser vista nas Equações 2.7 e 2.8 (NUSSENZVEIG, 2013).

$$v(t) = v(0) + \int_0^t (a(t)dt) \quad (2.7)$$

$$x(t) = x(0) + \int_0^t (v(t)dt) \quad (2.8)$$

Acelerômetros podem ser utilizados em muitas áreas, como na automotiva (com *airbags*), na navegação, no monitoramento de máquinas, na saúde, nos jogos e em outras. São diversos os processos físicos utilizados para desenvolver um sensor para medir a aceleração. Em aplicações que envolvem voo, aviões e satélites, acelerômetros são baseados em propriedades de massas rotativas. Na indústria, porém, o projeto mais comuns são feitos a partir da combinação da Lei de Newton de aceleração de massa e a Lei de Hooke de ação de mola (CARNEIRO, 2003).

Existem os que funcionam a partir do efeito de cristais piezoelétricos, que é o princípio no qual alguns cristais geram uma corrente elétrica como resposta a uma pressão mecânica. Então, o acelerômetro funciona como na Figura 5 demonstra, como uma pequena caixa com uma esfera dentro, e as paredes são os cristais piezoelétricos. Sempre que a caixa é alterada de posição a esfera é forçada a se mover em direção de uma das paredes devido a força da gravidade ou de qualquer outra força de aceleração em outro sentido que não a normal. Cada par de paredes paralelas correspondem a um eixo no espaço. Assim, a partir da corrente gerada em cada parede é possível determinar a direção do movimento acelerado e sua magnitude (SANJEEV, 2018).

2.4.2 GIROSCÓPIO

Os giroscópios , também, são sensores de movimento, porém eles não medem aceleração linear como os acelerômetros, eles disponibilizam ao usuário a velocidade angular de um

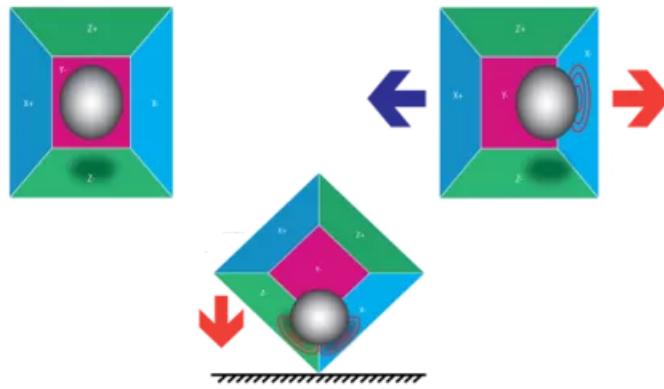


Figura 5 – Princípio de funcionamento do acelerômetro. Adaptada de Sanjeev (2018).

objeto em torno de um eixo. Em geral, os giroscópios feitos com tecnologia MEMS utilizam do efeito Coriolis, no qual um objeto que se encontra em um movimento de rotação imprime na massa um deslocamento ortogonal a direção de rotação. Os giroscópios têm um princípio bem mais complexo que os acelerômetros. Este é um dos motivos fizeram que eles demorarem mais a parecer como dispositivos MEMS (ALMEIDA, 2014).

O giroscópio do tipo diapasão é constituído por duas massas paralelas que oscilam com a mesma amplitude, direção igual e sentidos opostos mostrado na Figura 6.

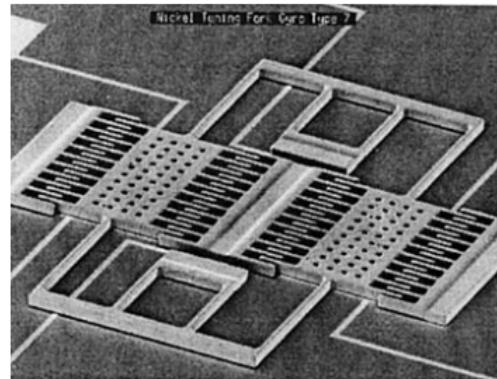


Figura 6 – Imagem de microscopia eletrônica por varredura de um giroscópio diapasão.

Fonte: (FORHAN, 2010)

Ao acontecer uma rotação, a força de Coriolis gera uma vibração ortogonal ao sentido do movimento de cada massa. A amplitude desse movimento pode ser medida de forma capacitiva. A utilização de dois giroscópios na mesma direção e em sentidos opostos aumenta a precisão da medição. Com a medida da força de Coriolis F_c , a medida da massas paralelas m , se movendo a um determinada velocidade v , com relação a um referencial fixo que possua uma velocidade angular ω é possível obter a aceleração de Coriolis a_c como mostram as Equações 2.9 e 2.10 (FORHAN, 2010).

$$F_c = 2mv\omega \quad (2.9)$$

$$a_c = 2v\omega \quad (2.10)$$

Com aceleração de Coriolis, que é obtido pelo sensor, a unidade de processamento do sensor é capaz de calcular a velocidade angular em °/s. Com a velocidade angular em relação a cada eixo do espaço tridimensional sendo obtida, é possível encontrar posição angular do sensor e do objeto, ou corpo, em que ele estiver acoplado (FORHAN, 2010)(NUSSENZVEIG, 2013).

A unidade de medida padrão de velocidade angular é rad/s. Então, com o intuito de converter o valor obtido para o padrão internacional, tem-se as Equações 2.11, 2.12, 2.13 e 2.14:

$$1\pi rad = 180^\circ \quad (2.11)$$

$$1\pi rad/s = 180^\circ/s \quad (2.12)$$

$$\frac{\pi}{180} rad/s = (\frac{180}{180})^\circ/s \quad (2.13)$$

$$\frac{\pi}{180} rad/s = 1^\circ/s \quad (2.14)$$

2.4.3 MPU6050

O MPU6050, da série MPU60X0, foi a primeira interface de movimento a integrar 6 eixos de leitura em um dispositivo único. Esse sensor de movimento, integra 3 eixos de um acelerômetro e 3 eixos de um giroscópio e um DMP (*Digital Motion Processor* - Processador Digital de Movimento) tudo em um pequeno componente medindo 4x4x0.9mm. As suas principais qualidades são seu tamanho reduzido, o baixo consumo de energia, a alta precisão e confiabilidade, a alta tolerância a choques mecânicos, ter seu desempenho programável para aplicações específicas e ainda um baixo custo (INVENSENSE, 2013).

O protocolo de comunicação utilizado pelo MPU6050 é o *I²C* (*Inter-Integrated Circuit* - Circuito Inter-Integrado). Que é um protocolo de barramento, e com os mesmos dois fios podem ser conectados vários dispositivos, um sendo o *master* e os outros como *slave*. Esta é uma característica boa para reduzir a quantidade de pinos necessários para conexão de mais dispositivos no microcontrolador (INVENSENSE, 2013).

As aplicações sugeridas pelo fabricante para o MPU6050 são diversas. Abaixo estão listadas algumas:

- Tecnologia *BlurFree^{TM1}*;
- Controles para jogos baseados em movimento;
- Sensores em roupas *TouchAnywhere* para aplicação na saúde e esportes;
- Tecnologia *MotionCommand^{TM2}*;
- Em brinquedos

Características do MPU6050

- O giroscópio MEMS de 3 eixos do MPU6050, possui as seguintes características segundo INVENSENSE (2013):
 - Saídas digitais com os valores de velocidade angular para os eixos X, Y e Z com escalas programáveis entre $\pm 250, \pm 500, \pm 1000$ e ± 2000 °/seg;
 - Conversores ADCs (*Analog Digital Converter* - Conversor Analógico Digital) de 16 bits integrados permitem amostragem simultânea do giroscópio
 - Corrente de operação: 3.6mA;
 - Bom desempenho com ruído de baixa frequência;
 - Filtro passa-baixa programável;
 - Fator de escala de sensibilidade calibrado de fábrica.
- O acelerômetro MEMS de 3 eixos do MPU6050, possui as seguintes características segundo INVENSENSE (2013):
 - Saídas digitais dos 3 eixos do acelerômetro com escalas programáveis entre $\pm 2g, \pm 4g, \pm 8g$ e $\pm 16g$. Sendo ‘g’ uma constante que equivale a aceleração da gravidade³;
 - Conversores ADCs de 16 bits integrados permitem amostragem simultânea do acelerômetro sem a necessidade de um multiplexador;
 - Orientação, detecção e sinalização.

2.4.4 ARDUÍNO

O Arduíno é uma plataforma de desenvolvimento de código aberto que contém elementos de *hardware* e *software* especificamente adaptados à simplicidade de uso, permitindo uma rápida prototipagem. Placas Arduíno têm grande potencial de desenvolvimento e uma ampla gama de recursos para uso em conjunto com sensores, transceptores de dados e atuadores, os quais são de grande interesse para este projeto (SMITH, 2016).

¹ Para estabilização de imagens e vídeos.

² Para comandos de movimento curtos.

³ aproximadamente $9,81m/s^2$.

Uma das maiores vantagens do Arduíno sobre outras plataformas de prototipagem com microcontroladores é a facilidade para utilizá-la, que permite que pessoas que não necessariamente são da área de tecnologia, possam aprender rapidamente o básico e criar seus próprios projetos. E existe um grande compartilhamento de projetos de forma livre utilizando essa plataforma. Isso é relevante para esse projeto em específico, pois ele está sendo desenvolvido para utilização de pesquisadores de diversas áreas do conhecimento, os quais poderão aprender a adaptar o projeto para suas necessidades e demandas futuras (ROBERTS, 2011).

Existem diversos modelos de placas Arduíno no mercado, sendo as mais utilizadas LilyPad, Uno, Mini Pro, Mega e Nano. Cada uma dessas placas possuem suas particularidades em relação ao número de recursos, formato físico, memória e capacidade de processamento. Na Tabela 1 estão listados alguns modelos e suas características (SMITH, 2016).

Tabela 1 – Tabela de comparação dos modelos de Arduíno. Adaptado de Smith (2016).

| Modelos | LilyPad | Uno | Pro | Mega | Nano |
|---------------------|------------|---------------|-----------|--------------|---------------|
| Microcontrolador | ATmega32u4 | ATmega328P | ATmega238 | ATmega2560 | ATmega328P |
| Tensão de Operação | 3.3V | 5V | 3.3/5V | 5V | 5V |
| Tensão de Entrada | 3.8-5V | 7-12V | 5-12V | 7-12V | 5-12V |
| Pinos Digitais | 9 | 14 | 14 | 54 | 14 |
| Entradas Analógicas | 4 | 6 | 6 | 16 | 8 |
| Memória Flash | 32kB | 32kB | 32kB | 256kB | 32kB |
| Velocidade do Clock | 8MHz | 16MHz | 8/16MHz | 16MHz | 16MHz |
| Dimensões Físicas | 50mmØ | 68.6 x 53.4mm | 52 x 53mm | 101.5x53.5mm | 18,5 x 43,2mm |
| Custo Aproximado | R\$ 22,00 | R\$ 54,00 | R\$ 50,00 | R\$ 80,00 | R\$ 24,00 |

O Arduíno Nano (Figura 7) tem funcionalidades semelhantes às do Arduíno Duemilanove, mas com um formato diferente. O Nano possui o microcontrolador ATmega328P embutido, o mesmo que o Arduíno UNO. A principal diferença entre eles é que a placa UNO é apresentada em formato PDIP (*Plastic Dual-In-line Package*) com 30 pinos e Nano está disponível em PQFP (*Plastic Quad Flat Pack*) com 32 pinos. Os 2 pinos extras do Arduíno Nano servem para as funcionalidades do ADC, enquanto o UNO possui 6 portas ADC, o Nano possui 8. A placa Nano não vem com um conector para fonte DC (*Direct Current - Corrente Contínua*) como outras placas Arduíno, mas possui uma porta mini-USB. Esta porta é usada para programação, monitoramento serial e alimentação (se necessário). A característica mais fascinante do Nano é que ele escolherá a fonte de energia com maior diferença de potencial, e o *jumper* de seleção de fonte de energia não precisa ser utilizado (JOHN, 2018).

Para se comunicar com qualquer microcontrolador ou computador, é necessário usar uma linguagem de programação. No caso do Arduíno, é necessário usar a linguagem C++⁴, que é uma linguagem tradicional e já bastante conhecida. Porém, essa linguagem ainda não é a que a máquina comprehende, é uma linguagem de alto nível, e para a máquina comprehendê-la é necessário o uso de um compilador, que converte a linguagem de alto nível para a linguagem

⁴ Com alguma modificações.

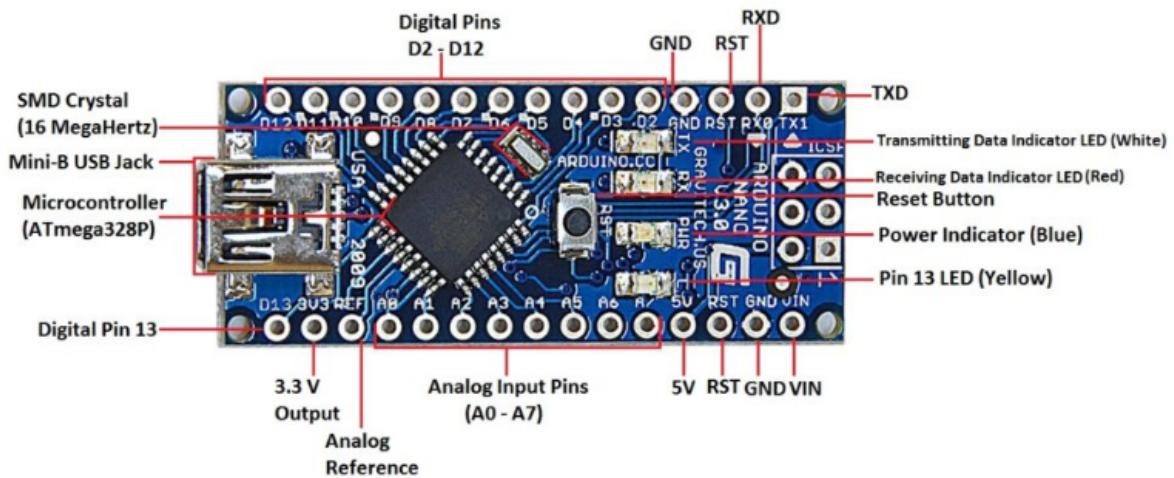


Figura 7 – Arduino Nano

Fonte: (JOHN, 2018)

de máquina. Em geral, para compilar um programa, é necessário a utilização de um IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado), um aplicativo de computador que possui um compilador integrado. A plataforma do Arduíno utiliza o Arduíno IDE (CHAVIER, 2016).

2.4.5 ESP8266

O ESP8266 é um microcontrolador fabricado pela empresa *Espressif Systems*. Possui um sistema de comunicação *Wi-Fi* próprio, podendo ser utilizado tanto para acessar a uma rede existente, como para criar o próprio ponto de acesso. Essa funcionalidade é seu grande diferencial em relação ao Arduíno, porém existem outras diferenças (OLIVEIRA, 2017).

A tensão das portas digitais é de 3,3 V, possui apenas uma porta para leitura analógica, 11 portas digitais, memória *flash* de 4Mb e o processador com velocidade de *clock* de 80MHz, todas essas características são diferentes nas placas Arduíno (WEMOS, 2017).

Existe uma grande variedade de placas disponíveis no Brasil. Mas em todas elas é utilizado o mesmo processador ESP8266. O que distingue os modelos um do outro é o número de pinos GPIO (*General Purpose Input/Output - Entrada e Saída para Uso Geral*) expostos, a quantidade de memória *flash* fornecida, o estilo dos pinos do conector e várias outras considerações relacionadas à fabricação. Em relação a programação, elas são todas iguais e podem ter seu código embarcado escrito com a IDE do Arduíno, sendo apenas necessário carregar sua biblioteca de placas, através do próprio programa, e a linguagem de programação utilizada é o C++ (KOLBAN, 2016).

Um dos modelos é a placa Wemos D1 (Figura 8), que tem o mesmo formato do Arduíno UNO, o que deixa a utilização mais confortável para usuários do Arduíno. Esse modelo possui

todos os seus pinos descritos na própria placa, também possui pinos para comunicação I^2C , conector micro USB e um conector de carga que aceita tensão entre 9 e 24 V (KOLBAN, 2016)(WEMOS, 2017).

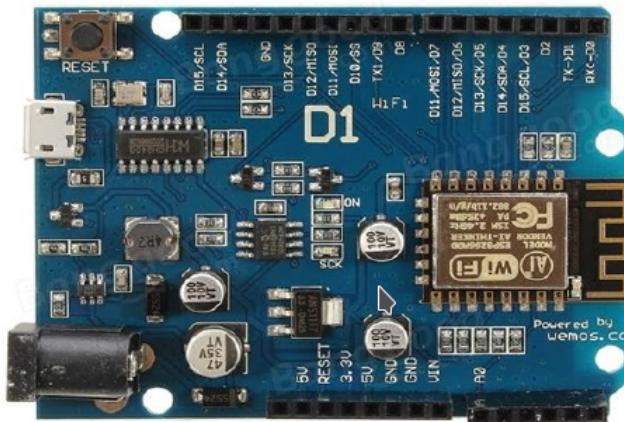


Figura 8 – WeMos D1 - R2

Fonte: (WEMOS, 2017)

2.5 PYTHON

Python é uma linguagem de programação poderosa, mas fácil de usar, desenvolvida por Guido van Rossum, em 1991. Com o *Python*, você pode escrever rapidamente um pequeno projeto e também, por se adaptar bem, pode ser usado para aplicativos comerciais. Uma prova de como é uma linguagem comercial é ela ser utilizada por empresas como *Google*, *IBM*, *NASA*, *Xerox*, *Yahoo* e outras grandes empresas (DAWSON, 2010).

O principal objetivo de qualquer linguagem de programação é fazer uma ponte entre o programador e a máquina. E a maioria das linguagens de programação tornam essa comunicação mais próxima da humana, mas o *Python* é tão direto que, segundo DAWSON (2010), é chamada de "programação na velocidade do pensamento". Essa facilidade se traduz na alta produtividade de programadores profissionais os quais com ele conseguem fazer programas mais curtos e mais rápidos de escrever do que com outras linguagens de programação.

Python é uma boa linguagem para aplicações científicas porque é fácil traduzir raciocínio em algoritmos através dela, bem como é simples ler um código em *Python* e conseguir entender o raciocínio por trás dele, por ser uma linguagem com poucos caracteres especiais, poucas palavras chaves utilizadas apenas para compilar e ter uma sintaxe muito próxima da linguagem falada (REITZ, 2018). Além disso, é uma linguagem de propósito geral. Muitas vezes, é necessário lidar com tarefas laterais: buscar dados em um banco de dados remoto, ler uma página na internet, exibir graficamente os resultados, criar uma planilha, etc. Linguagens de cunho especificamente científico têm um sério problema aí, mas, uma vez que *Python* é utilizada

em praticamente todos os tipos de tarefa, encontram-se módulos prontos para realizar essas tarefas que podem se tornar complicadas. Ou seja, é uma preocupação a menos para quem está desenvolvendo aplicações científicas (DOWNEY, 2012).

2.6 PROTOCOLO HTTP

Hypertext Transfer Protocol (HTTP) é o método utilizado para enviar e receber informações na *web*. A versão mais utilizada atualmente é a 1.1, definida pela especificação RFC 2616. RFCs (*Request for Comments*) são documentos técnicos desenvolvidos e mantidos pelo IETF (*Internet Engineering Task Force*), instituição que especifica os padrões que serão implementados e utilizados em toda a internet (VIEIRA, 2007).

O protocolo é baseado em requisições e respostas entre clientes e servidores. O cliente, navegador ou dispositivo que fará a requisição (também é conhecido como *user agent*), solicita um determinado *resource* (recurso) enviando um pacote de informações contendo alguns cabeçalhos (*headers*) a um URI (*Uniform Resource Identifier* - Identificador Uniforme de Recurso) ou, mais especificamente, URL (*Uniform Resource Locator* - Localizador Uniforme de Recursos). O servidor recebe estas informações e envia uma resposta, que pode ser um recurso ou um simplesmente um outro cabeçalho (VIEIRA, 2007).

O HTTP é *stateless*, ou seja, ele não é capaz por si só de reter informações entre requisições diferentes. Para manter informações é necessário utilizar *cookies*, sessões, campos de formulário ou variáveis na própria URL (VIEIRA, 2007).

Quando um requisição é realizada, é preciso especificar qual método deve ser utilizado. O HTTP possui 8, conhecidos por verbos, identificam a ação a ser executada por certo recurso. Os 5 métodos mais utilizados são: *get*, *post*, *delete*, *put* e *head* (VIEIRA, 2007)(SZYGALSKI, 2018).

Get solicita a representação de um recurso. É definido como um método seguro e não deve ser usado para disparar uma ação. *Post* é a informação enviada ao corpo da requisição e é utilizado para criar um novo recurso. *Delete* remove um recurso. *Put* atualiza um recurso na URI especificada. Caso o recurso não exista, ele pode criar um. A principal diferença entre *post* e *put* é que o primeiro pode lidar não somente com recursos, mas pode fazer processamento de informações, por exemplo. *Head* retorna informações sobre um recurso. Na prática, funciona semelhante ao método *get*, mas sem retornar o recurso no corpo da requisição. É considerado um método seguro (VIEIRA, 2007).

Toda requisição recebe um código de resposta conhecido como *status*. Assim é possível saber se uma operação foi realizada com sucesso (código 200), se ele foi movida e agora existe em outro lugar (código 301) ou se não existe mais (código 404), e existem diversos outros códigos (VIEIRA, 2007).

O HTTP é um protocolo extensível que é simples de usar. A arquitetura cliente-servidor, combinada com a habilidade de simplesmente adicionar cabeçalhos, permite que o HTTP avance suas funcionalidades juntamente com a elasticidade da *web* (SZYGALSKI, 2018).

3 METODOLOGIA

Nesta seção será apresentado quais foram os materiais e métodos utilizados para implementação do projeto, tanto da parte de *hardware* (parte física) quanto a parte de *software*. A principal metodologia utilizada foi baseada no *Scrum*, que é um método ágil iterativo incremental, onde o projeto foi dividido em ciclos de 15 dias chamados de "*sprints*". A cada *sprint*, algumas atividades são retiradas do *backlog*¹, colocadas no campo "*do*"² e tem de ser resolvidas até o início da próxima *sprint*, onde as atividades realizadas serão avaliadas, validadas ou não, se estiverem atendendo ao requisito são enviadas para o "*done*"³, se não estiverem atendendo as necessidades, elas voltam ao *backlog* ou para o *do*.

Os requisitos do projeto, que compuseram o *backlog* e definiram os objetivos desse projeto foram relacionados com o auxílio dos pesquisadores do LIS e alunos de mestrado da UnB/FGA.

O trabalho foi dividido em duas etapas, a primeira etapa foi focada em realizar a comunicação via porta serial e com o microcontrolador Arduíno, com ele foram adquiridos os parâmetros de um sensor MPU6050, em seguida foi realizada a multiplexação de 3 sensores para poder obter medidas de 3 pontos ao mesmo tempo através de um único Arduíno. A segunda etapa foi tentar comunicar o sensor com o computador por meio de transmissão sem fio e com protocolo HTTP, utilizando o ESP8266 e seu módulo de *Wi-Fi*.

3.1 PROJETO DE HARDWARE

3.1.1 MATERIAIS UTILIZADOS

- Sensor MPU6050;
- Arduíno Nano V.3;
- WeMos D1 -R2;
- Fios (*jumpers*);
- Cabo USB/Micro USB;
- Cabo USB/Mini USB;
- Computador com Arduíno IDE e *Python* instalados;

¹ Conjunto de requisitos que ainda não foram solucionados.

² Campo onde estão as atividades a serem realizadas durante a *sprint*.

³ Conjunto de atividades que já foram feitas e validadas.

- *Jupyter Notebook*;
- Tiras de velcro;
- Cabo *flat* (8 fios);
- *Protoboard*;
- Placa de Cobre Perfurada - 15x10cm;
- Bateria 5V - 2A - conector - Mini USB .

3.1.2 MÉTODOS

Para realizar a comunicação do sensor MPU6050 com o Arduíno foi feita a conexão conforme o esquemático mostrado na Figura 9 e na Tabela 2. Logo abaixo, onde os pinos Vcc e Gnd do MPU6050, responsáveis por ligar o sensor, foram conectados aos pinos 5V e Gnd do Arduíno. O pino SDA (*Serial Data* - Dados Seriais)⁴ foi conectado ao pino SDA do Arduíno, que também é o pino de entrada analógica A4 e o pino SCL (*Serial Clock* - Sinal de "Relógio" Serial)⁵ ao pino SCL do Arduíno, que é o pino de entrada analógica A5.

O modelo escolhido para este projeto foi o Arduíno Nano, pois mesmo sendo apenas um protótipo, haverá a necessidade de que o tamanho do IMU seja reduzido para facilitar a fixação em diversas partes do corpo e o custo⁶ para desenvolvimento do protótipo foi levado em consideração.

Para os primeiros testes foi utilizada uma *protoboard* para facilitar as conexões, pois ela permite que os componentes sejam conectados sem a necessidade de solda entre eles. Mas em um segundo momento para os teste de movimento serem realizados com maior acurácia, houve a necessidade de retirar os componentes da *protoboard* e soldá-los em uma placa furada de cobre, e isso faria o ruído devido a conexões instáveis fosse menor.

O sensor foi fixado em uma faixa de velcro, para permitir a fixação do IMU em diversas partes do corpo, e para conectá-lo ao Arduíno foi soldado ao sensor um cabo *flat* com 8 fios.

Tabela 2 – Conexões entre Arduíno e MPU6050

| Arduíno | MPU6050 |
|---------|---------|
| 5V | VCC |
| GND | GND |
| A4 | SDA |
| A5 | SCL |

⁴ Pino de transferência de dados.

⁵ Pino de sincronização do *clock*.

⁶ Preços pesquisados em lojas de Brasília.

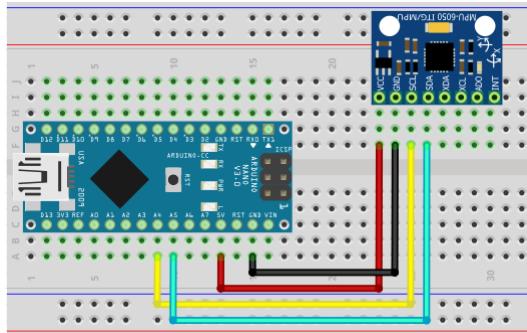


Figura 9 – Esquemático de conexão do MPU6050 com Arduíno Nano (Imagen do autor).

A comunicação entre o sensor MPU6050 e o Arduíno foi realizada através do protocolo I^2C , onde os dados eram enviados de forma serial através de apenas um fio e os dados organizados como mostrado na Figura 10, onde é evidenciado quais são os dados transmitidos do MPU6050 para o Arduíno e a velocidade do *clock*, e Figura 11, que mostra com mais detalhe a relação dos dados com o *clock*.

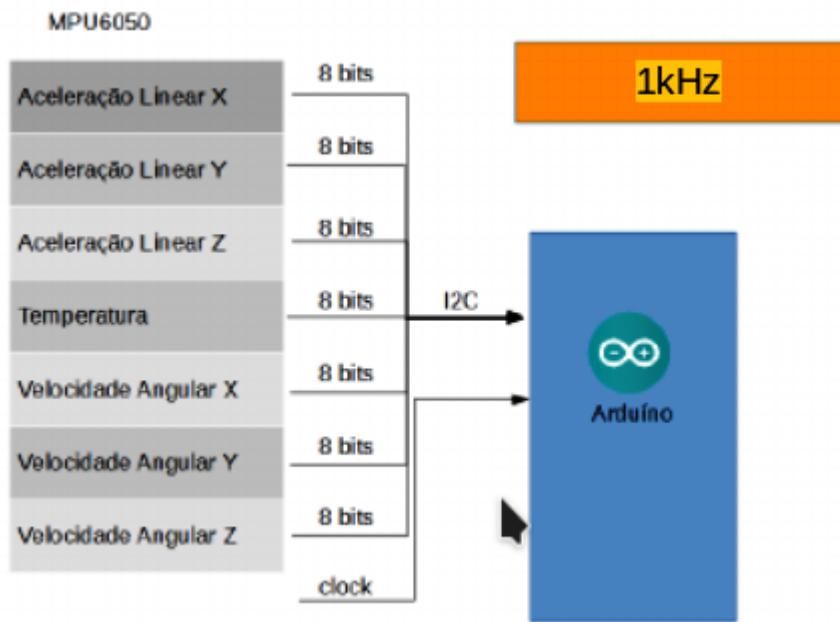


Figura 10 – Transmissão de dados do MPU6050 para Arduíno (Imagen do autor).

O Arduíno Nano foi conectado ao computador por meio de um cabo USB 2.0, que além de ser responsável pela comunicação, também era por meio dele que era realizada a alimentação de energia com 5V de tensão. O protocolo de comunicação utilizado com um cabo USB foi o serial. Então, os dados transmitidos do Arduíno para o computador são enviados em série.

Ainda na primeira etapa foi realizada a montagem utilizando 3 sensores. Para isso foi preciso adicionar mais um pino do MPU6050 a montagem, para poder realizar a multiplexação deles. Foi utilizado o pino AD0, responsável por definir um nome de endereço para o sensor. Quando o é conectado com 0 V (Nível Lógico Baixo), o seu endereço de registrador se torna

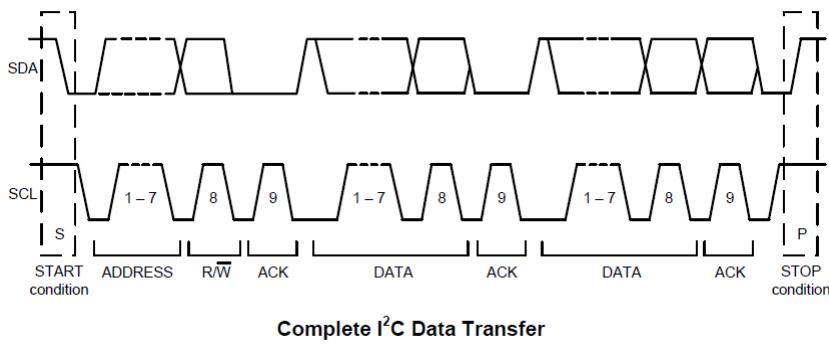


Figura 11 – Transferência de dados com o protocolo I^2C .

Fonte: (INVENSENSE, 2013).

0x68, e quando ligado entre 3.3 e 5 V (Nível Lógico Alto), é alterado para 0x69. Dessa forma foi possível realizar a leitura dos sensores de forma sequencial. A tabela 3 mostra as conexões necessárias.

Tabela 3 – Conexões entre Arduíno e 3 Sensores

| Arduíno | MPU6050 A | MPU6050 B | MPU6050 C |
|---------|-----------|-----------|-----------|
| 5V | VCC | VCC | VCC |
| GND | GND | GND | GND |
| A4 | SDA | SDA | SDA |
| A5 | SCL | SCL | SCL |
| D2 | AD0 | - | - |
| D3 | - | AD0 | - |
| D4 | - | - | AD0 |

Para a segunda etapa, foi escolhido a placa Wemos D1, como apresentado no referencial teórico, todas as placas que possuem o processador ESP8266 apresentam características semelhantes, a Wemos foi escolhida pela possibilidade de conectá-la ao computador sem a necessidade de um cabo USB, após configurar o *firmware* para criar uma conexão *wireless*, ou se conectar a uma rede.

A Tabela 4 mostra a conexão do MPU6050 com a ESP8266, a qual, é semelhante à primeira montagem realizada com o Arduino, sendo alterado apenas o nome dos pinos que representam SCL e SDA, na Wemos, que são, respectivamente, os pinos digitais D1 e D2.

Tabela 4 – Conexões entre Wemos e Sensor

| Wemos D1 | MPU6050 |
|----------|---------|
| 5V | VCC |
| GND | GND |
| D2 | SDA |
| D1 | SCL |

Foi necessário para carregar o código do *firmware*, a conexão do Wemos através do cabo micro USB, após o programa ser compilado e salvo no microcontrolador, este foi desconectado

já com o sensor ligado a ele e foi adicionada uma bateria de 5V, 2A e carga de 10000 mAh, para fornecer energia à placa. Essa fonte garante energia suficiente para placa configurada como ponto de acesso e o sensor. O consumo energético máximo do MPU6050 é 3,9 mA e da placa Wemos é de 215 mA, logo essa bateria, estando totalmente carregada, garante a utilização do conjunto por mais de 4 horas.

3.2 PROJETO DE SOFTWARE

Para esse trabalho foram necessários seis projetos de *softwares*, dois para Arduíno, uma para o Wemos, dois em *Python*, responsáveis por realizar o tratamento dos dados adquiridos do sensor e salvá-los para análises futuras e um terceiro apenas para testar a comunicação sem fio e verificar a velocidade da transmissão.

3.2.1 SOFTWARES PARA O ARDUÍNO

Os requisitos do *software* salvo no microcontrolador, conectado com apenas um sensor, são:

- Inicializar o MPU6050;
- Receber os dados do MPU6050;
- Enviar os dados para interface serial.

Para isso foram utilizadas as bibliotecas *Wire.h*, *I2Cdev.h* e *MPU6050.h*, que são responsáveis por realizar a comunicação em I^2C , enviar o comando de inicialização, a variável escolhida pelo usuário para definir os limites de leitura do acelerômetro e do giroscópio. São estas bibliotecas que realizam a leitura dos registradores que tem o valor de cada sensor armazenado neles.

O fluxograma do *software* responsável pela leitura dos dados do MPU6050 e envio para a *interface* de comunicação serial é o mostrado na Figura 12.

Para realizar a transmissão dos dados do Arduíno para o programa em *Python*, o programa salvo no Arduíno imprime no terminal serial os dados obtidos do sensor. O programa é fechado sempre que uma nova leitura é realizada.

O código feito para conectar o Arduíno a 3 sensores e transmitir os dados adquiridos, foi feito sem o auxílio de todas as bibliotecas utilizadas no anterior, pois esse precisa de uma velocidade de processamento mais alta, para não perder a qualidade do sinal.

São configurados os pinos D2, D3 e D4 como saídas digitais, estes serão os pinos que definirão qual sensor está estabelecendo comunicação. Em seguida D2 é alterado para o estado

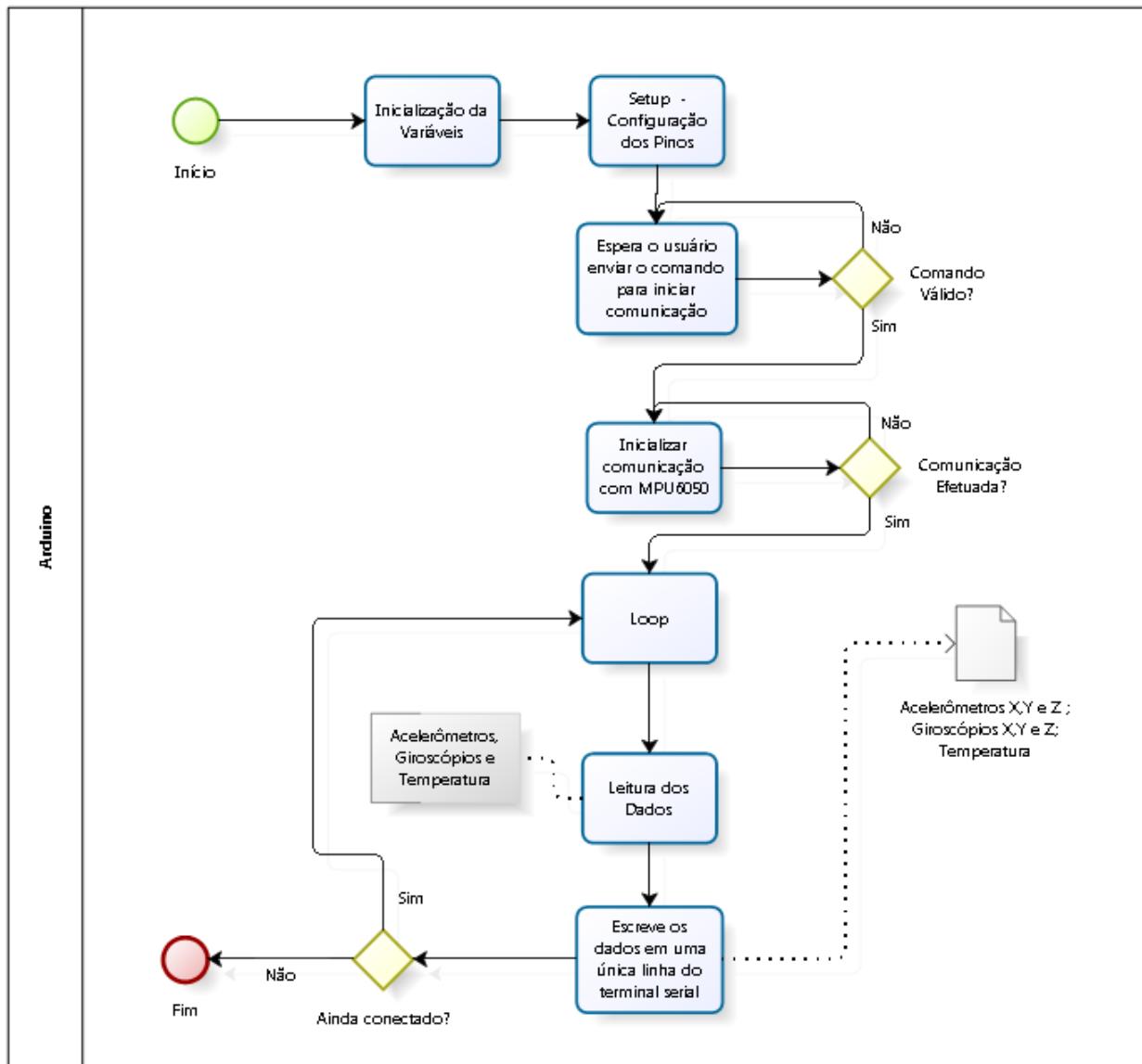


Figura 12 – Fluxograma do *software* de leitura dos dados. (Imagen do autor).

LOW (nível lógico baixo), e outros dois permanecem em nível lógico alto e o sensor conectado ao pino D2 é inicializado. Esse mesmo procedimento é realizado com nos pinos D3 e D4, assim todos os sensores têm comunicação iniciada.

Dentro da função *loop*, foi implementado mais um laço para realizar multiplexação e alterar o identificador do sensor. Nessa função os sensores são lidos um e enviados ao terminal serial a cada execução do laço. Foi colocada uma função *timer*, para garantir a frequência de 1kHz na leitura dos sensores.

3.2.2 SOFTWARE PARA O WEMOS D1

O *software* ou *firmware* do microcontrolador ESP8266, precisa de uma biblioteca específica para ele poder ser escrito na IDE do Arduíno, a *ESP8266WiFi.h*, responsável pela

configuração do módulo *wireless*.

O código primeiro configura o módulo como servidor *Wi-Fi* habilitando a porta 80. As variáveis são inicializadas e dentro da função *setup* e são iniciados os modos de comunicação serial e *wire*, o sensor MPU6050 é ligado e configurado para transmissão dos parâmetros de movimento.

Os endereços para utilização do protocolo HTTP são escritos, o microcontrolador é configurado como ponto de acesso, seu nome e senha são salvos. E os endereços escritos anteriormente são salvos nas configurações do módulo.

A transmissão *wireless* é habilitada antes da execução da função *loop*, a qual fica na espera a requisição de um cliente para iniciar leitura e transmissão dos dados ao endereço de IP configurado no início do programa. Assim que a requisição é feita é verificado se o endereço requisitado é estabelecido, se for é escrito o código 200, para confirmar que foi uma comunicação bem sucedida e os dados então são escritos.

Em seguida o programa volta a esperar mais uma requisição para ler e enviar os dados.

3.2.3 SOFTWARE PARA SALVAR OS DADOS

Esse programa, foi feito para realizar a leitura dos dados transmitidos pelo Arduíno, e salvá-los em um arquivo de texto, para futuras análises. Esse *software* ainda dá a possibilidade do usuário salvar os dados sem nenhum tratamento ou os valores com medidas físicas aproximadas das medições realizadas.

O *software* foi escrito em *Python 3* e interpretado com o *Jupyter Notebook* e com a IDE *Spyder*, os dois programas são do pacote *Anaconda*.

O programa inicia mostrando a lista de dispositivos USB conectados ao computador. Se não houver nenhum dispositivo conectado aparece a mensagem "Conekte o Arduíno". Se houver, a mensagem que aparece pede para digitar o número correspondente ao dispositivo que o usuário pretende verificar os dados.

Após a escolha do Arduíno, o usuário deve escolher entre as 4 possibilidades de escala dos dados obtidos. E em seguida pressionar "*enter*" para iniciar a leitura e "*Ctrl+C*"⁷ para pausar e salvar os valores. O programa então imprime na tela os valores dos sensores que estão sendo lidos e ao fim do código, quando o usuário pressionar "I" duas vezes, os valores são salvos em dois arquivos nomeados com a data de quando foi finalizada a leitura, como mostrado na Figura 13.

Para definir melhor os requisitos do *software*, foi feita a seguinte história de usuário descrita na Tabela 5, e a partir dela foi possível desenvolver o *software* inicial e com ele realizar os testes preliminares do protótipo.

⁷ No *Jupyter Notebook* o "*Ctrl+C*" não funcionou, precisando assim pressionar o ícone de "*stop*" disponível na tela.

Tabela 5 – História de Usuário

| Usuário | O que quer fazer? | Resposta do Sistema |
|-------------|------------------------------|--|
| Pesquisador | Iniciarizar o sistema | Abre o <i>prompt</i> de comando Mostra a lista de dispositivos conectados |
| Pesquisador | Selecionar o sensor | Estabelecer a conexão com sensor Mostra a lista de intervalo de leitura |
| Pesquisador | Selecionar o intervalo | Mensagem de qual intervalo foi selecionado Aguarda comando para iniciar a coleta |
| Pesquisador | Iniciar coleta | Inicia a coleta Aguarda comando para pausar coleta |
| Pesquisador | Pausar coleta e salvar Dados | Para a coleta Fecha a conexão Salva os dados coletados Mensagem com nome dos arquivos |

O segundo *software* em *Python* foi escrito para receber os dados de 3 sensores. Esse programa funciona basicamente da mesma forma que o primeiro, com as seguintes diferenças: recebe um dado a mais a cada laço de execução, o número de identificação do sensor que varia de 1 a 3; possui uma variável a mais para armazenar esse valor; não exibe a escolha de escalas de medição diferentes; e no fim da execução pergunta ao usuário se deseja salvar os dados ou não.

3.2.4 SOFTWARE PARA TESTE DE COMUNICAÇÃO SEM FIO

Esse código foi escrito apenas para receber os dados escritos pelo Wemos no endereço especificado e verificar o tempo que de duração entre cada requisição.

O programa utilizou a biblioteca a *urllib*, para realizar a requisição; a 'bs4' para salvar apenas os dados transmitidos, e não os provenientes do protocolo HTTP; e a *timeit* para calcular o tempo de execução do código.

Era executado um laço, que realiza a requisição, salva e imprimi o texto recebido, durante a execução do laço um variável armazenar o número de vezes que o laço foi executado. Para interromper o programa o usuário precisa pressionar a tecla 'I' duas vezes.

Após a interrupção é calculado o tempo médio de execução de cada laço. Esse valor, o tempo total e número de execuções são impressos no terminal.

3.3 TESTE

Para testar o funcionamento dos acelerômetros do IMU foi feito um teste simples. Pois sempre que o sensor estiver "parado", ele estará sobre efeito da força do peso e consequentemente da aceleração da gravidade de aproximadamente $9,8 \text{ m/s}^2$. Então para verificar, foi necessário testar o sensor com um eixo de cada vez alinhado com a Normal, como na Figura

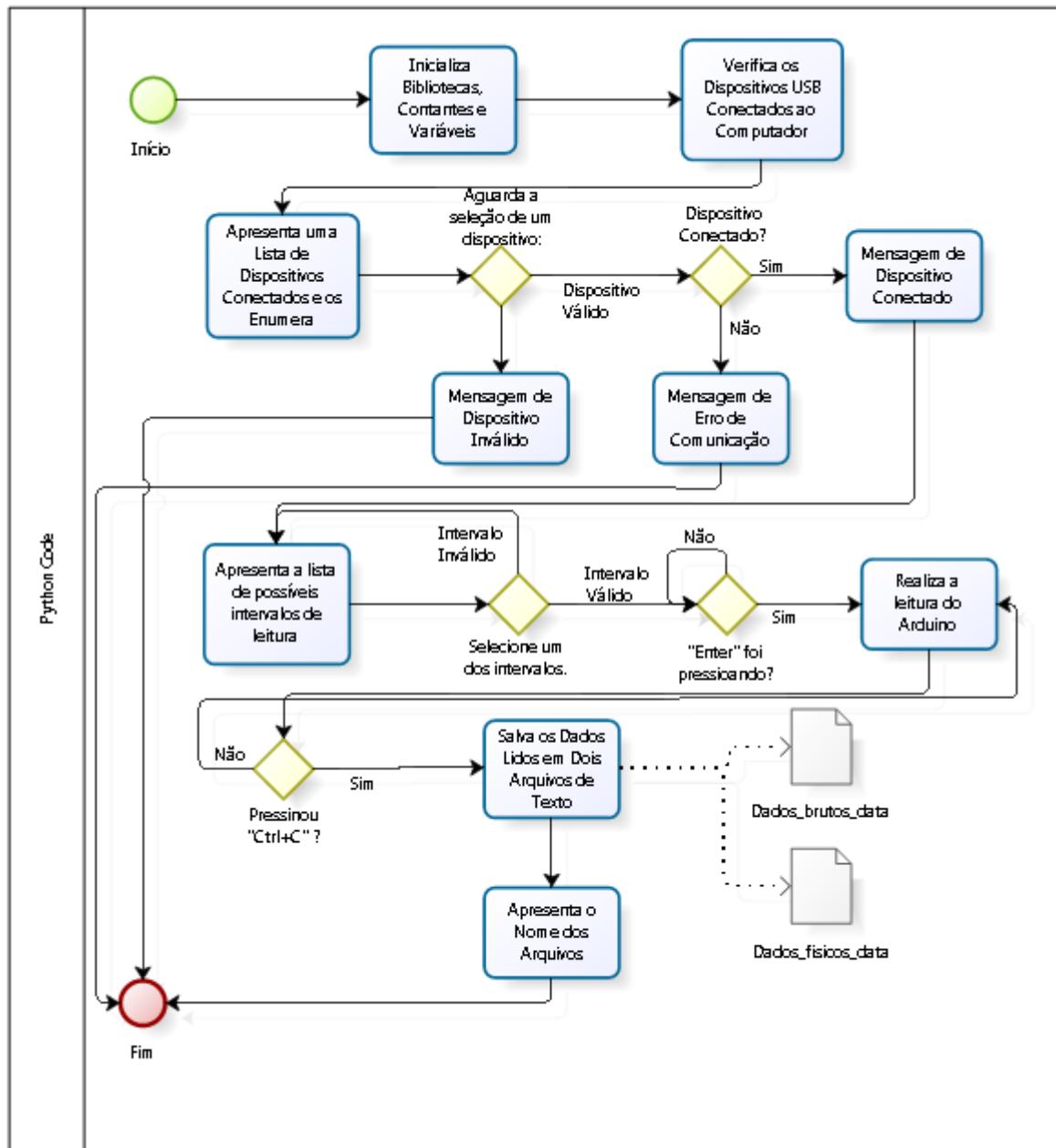


Figura 13 – Fluxograma do *software* que salva os dados lidos (Imagen do autor).

14, onde a imagem mais à direita mostra o MPU6050 com o eixo X paralelo a Normal, a imagem à direita com o eixo Y paralelo a Normal e ao centro o eixo Z.

Após realizar as leitura, os dados foram plotados em um gráfico simples, onde o eixo horizontal representa o tempo e o vertical a aceleração medida. Foram colocados os gráficos dos 3 eixos na mesma imagem.

Para verificar se os giroscópios estavam funcionando, sem ainda se atentar a precisão deles e confiabilidade das medidas, foi realizado um movimento de rotação de 90° em relação a cada um dos eixos, como na Figura 15 e ao fim da leitura do sensor o resultado foi plotado em um gráfico para facilitar a análise.

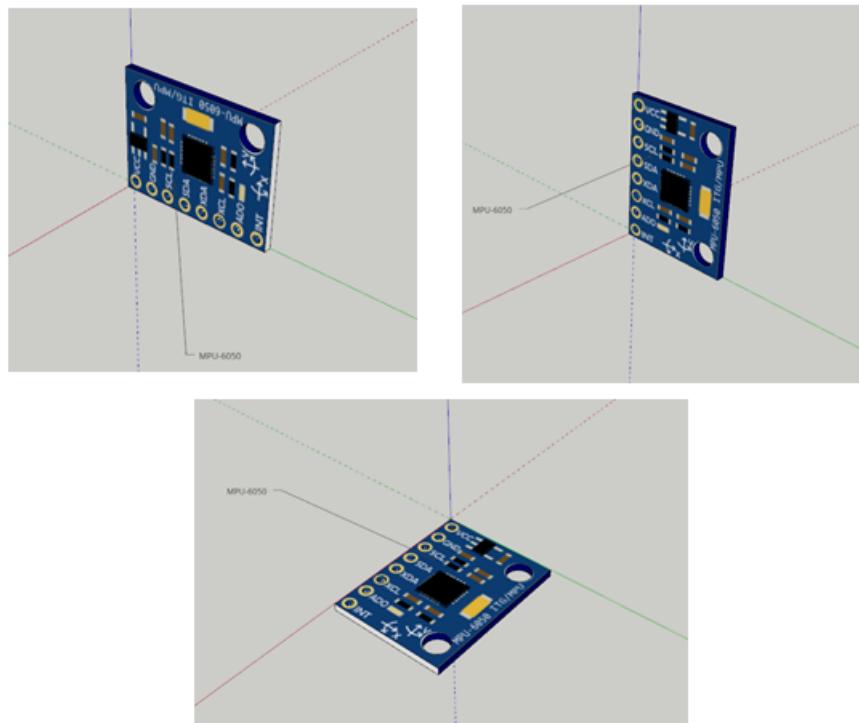


Figura 14 – Demonstração da posição do sensor (Imagen do autor).

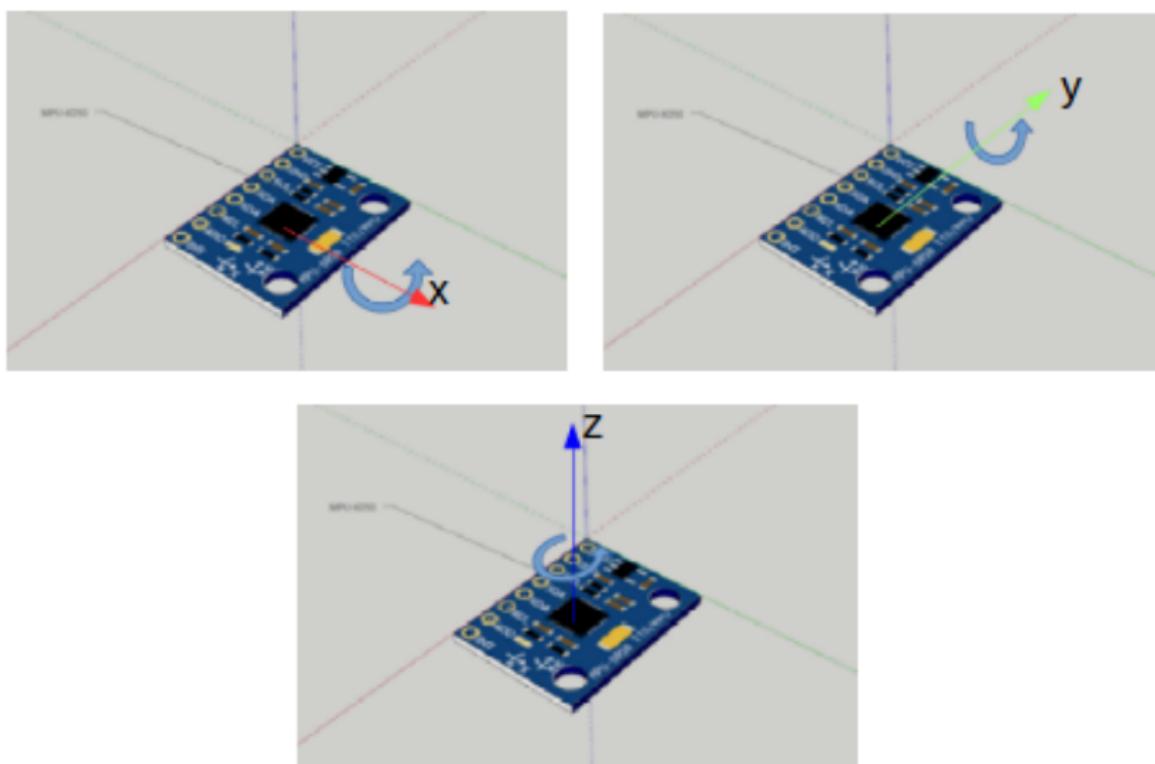


Figura 15 – Demonstração direção de giro do sensor (Imagen do autor).

Para testar o funcionamento do programa que recebia os dados do 3 sensores multipleados, o programa foi executado e os foi verificado o tempo transcorrido durante a execução do

programa e o resultado foi dividido pelo total de execuções. Para verificar se a velocidade de processamento era suficiente para garantir o que os dados representam os parâmetros reais.

O funcionamento da transmissão sem fio foi monitorado ligando a placa Wemos à bateria, conectando o computador a rede *wireless* gerada pela placa. Em seguida o programa feito em Python foi executado em diferentes distâncias entre a placa e computador conectado a ela, para verificar a perda de velocidade de processamento com o distanciamento.

Para confirmar o funcionamento do MPU6050 e se as leituras são confiáveis, ou não, serão necessários outros testes além dos realizados até o momento, como a comparação dos valores obtidos medindo os parâmetros que descrevem os movimentos ao mesmo tempo que uma gravação de vídeo é realizada, e verificar se os valores obtidos com o IMU são condizentes com as imagens gravadas.

Este trabalho teve aprovação do comitê de ética CAAE (Certificado de Apresentação para Apreciação Ética) número: 82432618.1.0000.8093.

E para validar a medição do sensor com fim de utilizá-lo em pesquisas com movimento musculoesquelético humano é necessário ao menos um estudo de caso.

4 RESULTADOS E DISCUSSÃO

4.1 RESULTADOS

Os resultados obtidos são os seguintes: protótipo funcional, *software* para o microcontrolador, *software* para ler, tratar e salvar os dados, tabelas e gráficos parciais do funcionamento do IMU. Os códigos aqui desenvolvidos estão disponibilizados nos Apêndices A, B, C, D, E, F e G.

As Figuras 16 e 17 mostram a primeira montagem feita do IMU para programar o microcontrolador e verificar a comunicação entre o sensor e o Arduíno. As Figuras 18 e 19 mostram a segunda montagem realizada do IMU, onde ele foi soldado com um cabo *flat* e preso em uma tira de velcro.

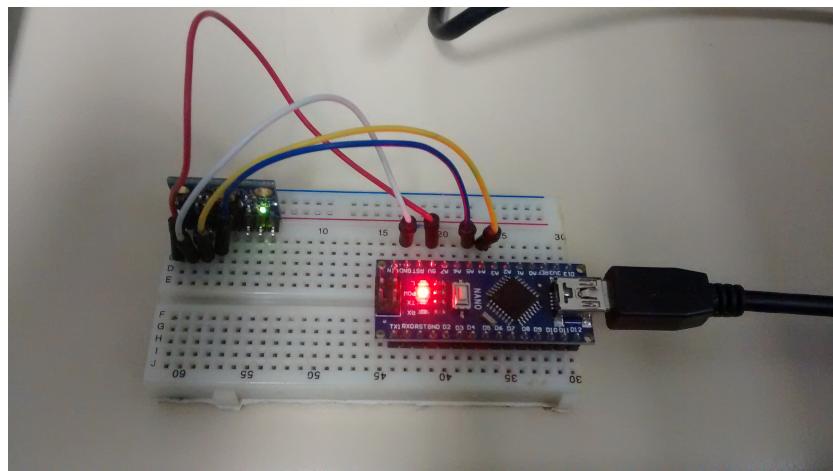


Figura 16 – Protótipo na *Protoboard* (Imagen do autor).

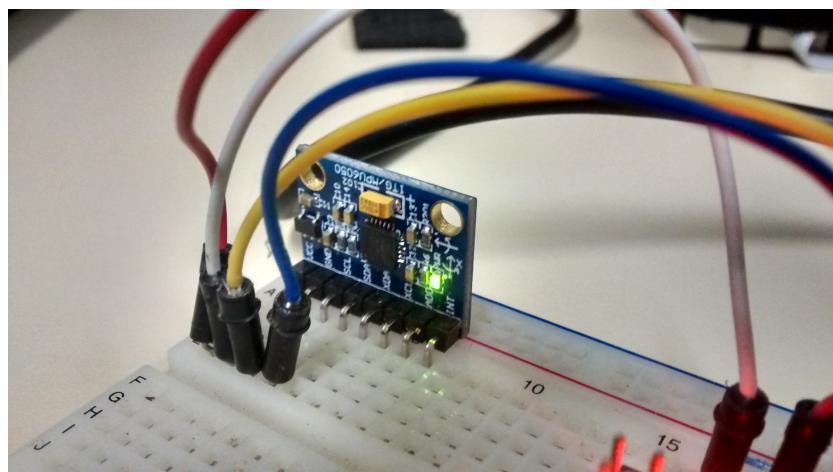


Figura 17 – MPU6050 na *Protoboard* (Imagen do autor).

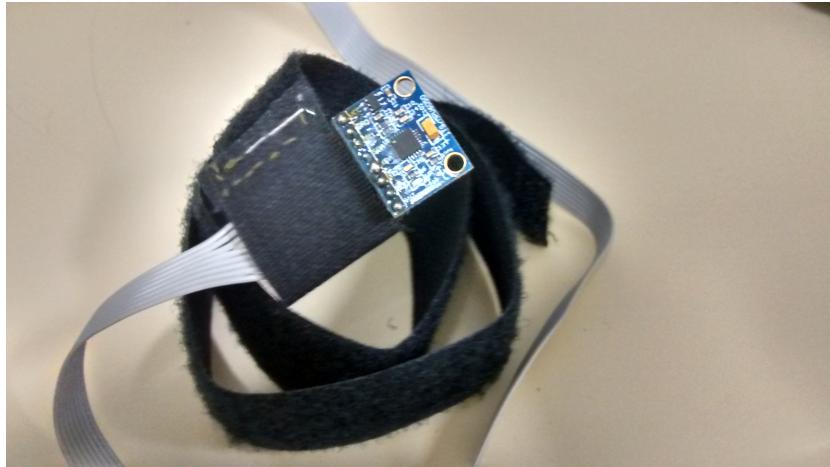


Figura 18 – Segundo Protótipo com o Velcro (Imagen do autor).



Figura 19 – Protótipo Fixado ao Punho e a um Cubo (Imagen do autor).

O programa em funcionamento é mostrado nas Figuras 20, 21 e 22 que, por enquanto, apenas no *prompt* de comando. A Figura 20 mostra a mensagem que aparece assim que a pessoa coloca o programa para funcionar, no caso, aparece também a lista de dispositivos conectados e pede para a pessoa escolher um deles. Se nenhum estivesse conectado, apareceria a mensagem "Conecte o Arduíno".

A Figura 21 mostra o que aparece logo depois de escolher o Arduíno. A confirmação de conexão com Arduíno, uma mensagem, que vem do próprio Arduíno, pedindo para selecionar uma das opções mostradas logo abaixo em um lista de possíveis escalas de leitura para o IMU e aguarda a escolha do usuário.

E na Figura 22, aparece a mensagem para iniciar a leitura do sensor e alguns dos dados lidos são mostrados para o usuário, precisando apenas pressionar 'I' duas vezes para pausar a leitura e salvar os dados em dois arquivos. Apenas para os testes foram acrescentados ao final do programa a geração de dois gráficos, um com os dados obtidos dos acelerômetros e o outro para o giroscópio.

```
===== Lista de dispositivos USB =====
0 - /dev/ttyUSB0
Escolha a porta do Arduino (e.g. 0): |
```

Figura 20 – Tela Inicial (Imagen do autor).

```
Comunicação Serial Estabelecida.
Digite o comando.(0,1,2 ou 3)
```

```
Intervalos de leitura:
0 -----> +- 250 deg/s e +- 2g
1 -----> +- 500 deg/s e +- 4g
2 -----> +- 1000 deg/s e +- 8g
3 -----> +- 2000 deg/s e +- 16g
```

```
Escolha o intervalo de leitura (e.g. 0): |
```

Figura 21 – Lista com as escalas (Imagen do autor).

```
Escolha o intervalo de leitura (e.g. 0): 0
Pressione 'Enter' para iniciar a leitura e "Ctrl+C" para pausar:
Comando 0 recebido.

Erro de valor.
-296   -4     16160   -3584   -294    -197    -68
-324   20     16104   -3568   -287    -143    -69
-308   56     16140   -3552   -270    -161    -90
-368   -68    16172   -3536   -248    -175    -85
-224   112    16116   -3568   -262    -179    -56
-272   120    16268   -3552   -296    -168    -73
-356   96     16156   -3520   -283    -214    -79
```

Figura 22 – Programa mostrando os resultados (Imagen do autor).

Os gráficos apresentados na Figura 23, mostram o resultado parcial do teste para medir a aceleração da gravidade com cada um dos 3 acelerômetros. É possível a partir deles verificar que os acelerômetros estão obtendo valores realistas para as medidas feitas, pois todos os gráficos apresentaram um valor próximo ao da aceleração da gravidade. Os gráficos ainda não estão com aspecto ideal, dando a sensação de presença de ruído, mas isso pode ser corrigido com um

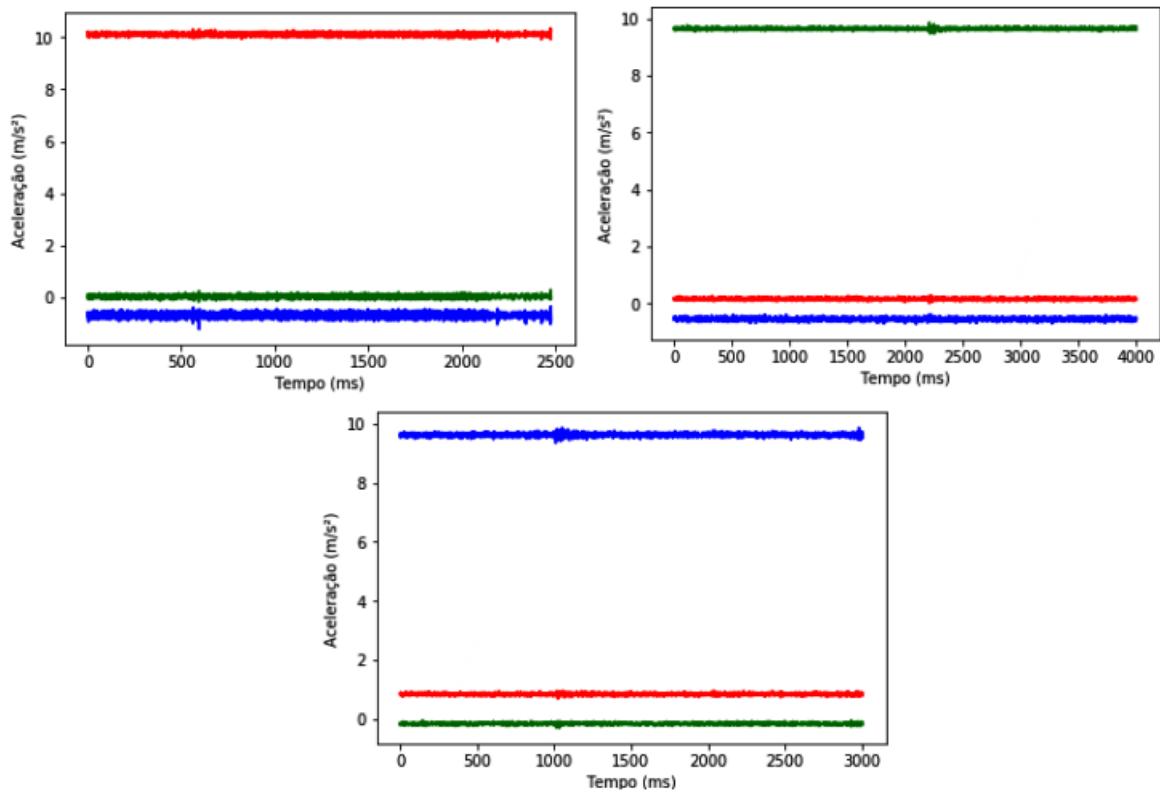


Figura 23 – Gráficos com os 3 Eixos alinhados com a Normal. (Imagem do autor).

Na imagem à esquerda o Eixo X está alinhado, à direita o Eixo Y e ao centro o Eixo Z.

buffer para carregar uma quantidade de valores e mostrar apenas a média deles, porque o sensor é sensível a pequenas variações de aceleração.

Os gráficos da Figura 24 representam à esquerda, à direita e ao centro, respectivamente, a rotação em relação aos Eixos X,Y e Z. O gráfico apresenta vários picos e variações que parecem ruidosos, mas na realidade é porque o sensor é preciso o bastante para captar pequenos impactos. E na hora de realizar o teste precisava de uma movimentação mais suave e constante, para melhor visualização no gráfico, mas como o teste foi realizado movimentando o sensor com a mão, ele captou a vibração natural da mão e o impacto com a superfície no momento que ele é solto, o que explica a movimentação em todos os sensores e não apenas em um.

Porém, mesmo com o gráfico não apresentando o resultado ideal, é possível verificar a variação nas velocidades de rotação em cada giroscópio separadamente, o que demonstra que os giroscópios funcionam.

Foi obtido como resultado, também, a montagem da placa formada pelo Arduíno, cabos *flat* e os 3 sensores vistos nas Figuras 25 , 26 e 27. O cabo *flat* utilizado nesses sensores, não foi soldado, como o primeiro utilizado, e sim *crimpado*. Dessa forma ele ficou menos sensível a movimentação. O que havia sido soldado, acabou quebrando após algumas utilizações.

O resultado do fim da execução do programa utilizando os 3 sensores foi o mostrado na

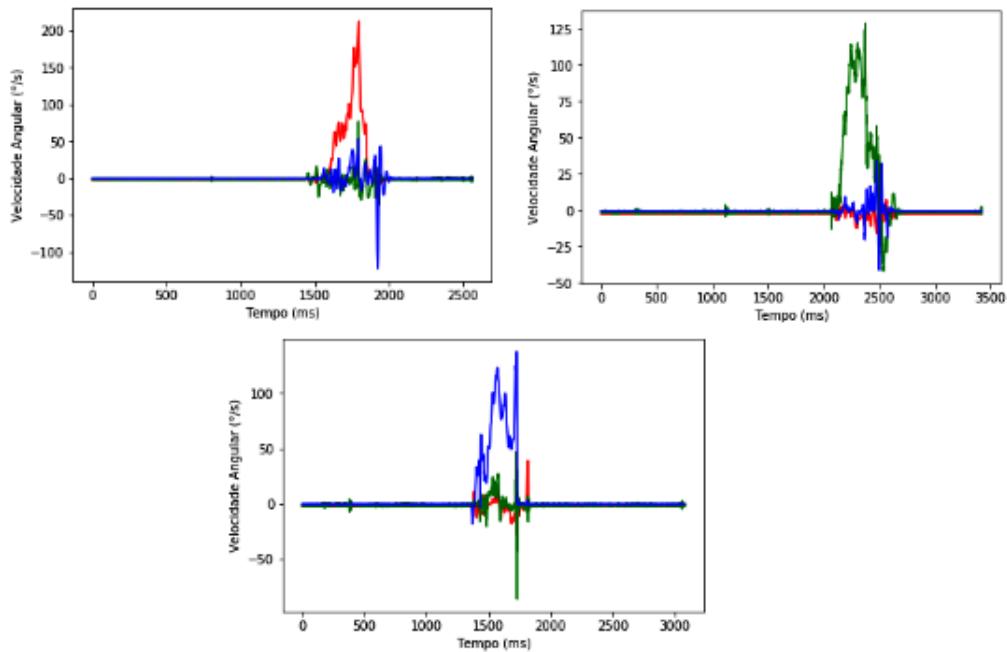


Figura 24 – Gráficos para Teste do Giroscópio (Imagen do autor).

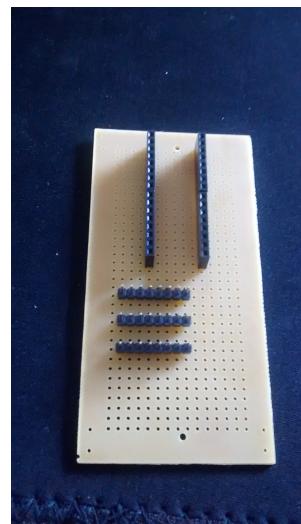


Figura 25 – Placa Perfurada (Imagen do autor).



Figura 26 – Cabo Flat com Sensor (Imagen do autor).

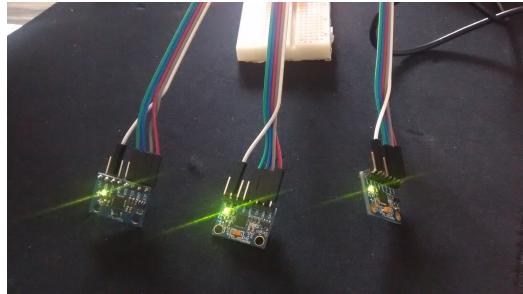


Figura 27 – 3 sensores Multiplexados (Imagen do autor).

figura 28. O início do programa é parecido com o primeiro. O resultado obtido foi mostrado em relação ao tempo de execução em segundos e foi igual a 0,00235 s. Sabendo que $f = 1/s$ então $f = 425,5\text{Hz}$.

```

1      -9912.00      -32768.00      11284.00      -3184.00      -1053.00      -17293.00      112.00
2      -3860.00      -276.00 14316.00      -2432.00      157.00      -223.00  -99.00
3      -2748.00      -4112.00      15736.00      -2400.00      -695.00  -71.00  -152.00
1      -9884.00      -32768.00      11280.00      -3152.00      -1064.00      -17309.00      86.00
Tempo médio de 2147 execuções foi de  0.002352392672101629  s
O tempo total foi: 5.050587067002198
Voce interrompeu este programa.
1

```

Figura 28 – Final da Execução dos 3 Sensores (Imagen do autor).

Seriam gerados, como resultado, os gráficos de cada um dos sensores. Não foi implementado porque antes de escrever esta parte do código, a qual exigia algumas manipulações nos dados, foi iniciada a segunda etapa do projeto.

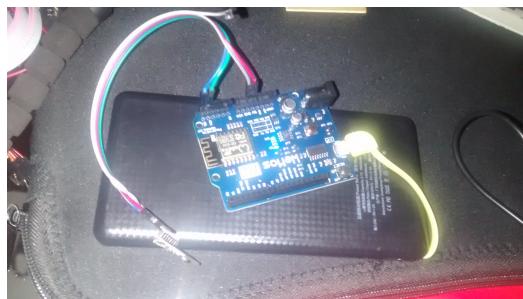


Figura 29 – Wemos com Sensor MPU6050 (Imagen do autor).

Os últimos resultados obtidos foram o sensor conectado à placa Wemos, a comunicação sem fio (Figura 29 e 30), a alimentação por bateria recarregável, a execução do programa para obtenção dos dados (Figura 31) e a frequência de execução do código.

A Figura 31 mostra o final da execução do programa quando a distância entre a placa Wemos e o computador era de 1 metro. Para ficar mais clara a essa variação, foi feito o gráfico mostrado na Figura 32, onde o eixo x representa a distância e o eixo y a frequência calculada com os intervalos de tempo obtidos após cada execução.

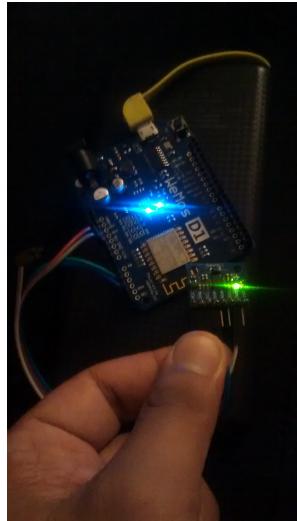


Figura 30 – Wemos Configurada como Ponto de Acesso WiFi (Imagen do autor).

```

Resultado: 57232      896     11904    62896   65501   65348   65438
Resultado: 57164      972     12008    62912    60       65303   65424
Resultado: 57028      856     11920    62896   109      65316   65427
Resultado: 57112      1012    11888    62912   69       65345   65449
Tempo médio de 502 execuções foi de  0.020698393938246625  s
O tempo total foi: 10.390593756999806
Laço interrompido.

```

Figura 31 – Final da Execução da Leitura Sem Fio (Imagen do autor).

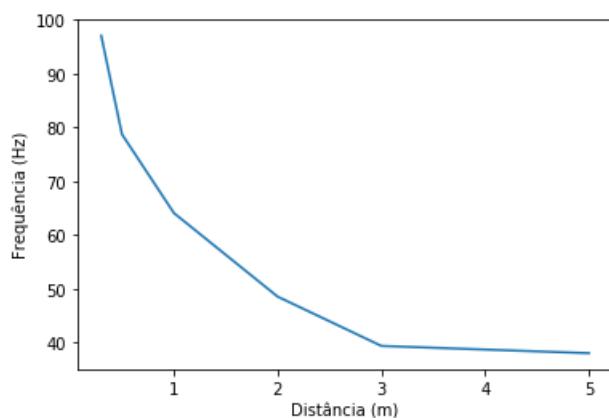


Figura 32 – Frequêcia x Distância (Imagen do autor).

A frequêcia atingida através da conexão sem fio não atingiu a taxa de Nyquist para amostragem do movimento humano, mostrado no referencial teórico. E quanto maior a distância entre o transmissor e receptor, menor foi a frequêcia obtida. Com esse resultado o sinal a ser amostrado pode ter uma frequêcia máxima de 40Hz.

4.2 DISCUSSÃO

O primeiro protótipo, na *protoboard*, foi útil por facilitar os testes de conexão, pois com a *protoboard* não era necessário soldar os pinos, mas por ser uma conexão instável está propensa a ruídos. E por isso foi feito o segundo protótipo. O cabo *flat* foi escolhido por permitir soldar o sensor e ainda mantê-lo com um tamanho confortável para fixação.

A *interface* do programa ainda precisa ser melhorada, mas atende a aplicação, porque foi feita para utilização de pesquisadores do laboratório que receberam o treinamento e saberão lidar com linhas de comando, ou receberão o treinamento necessário para a devida utilização. Com a melhoria da *interface*, será mais intuitivo utilizar o programa e ele se tornará ainda mais acessível, podendo ser utilizado por outros profissionais que tiverem interesse no movimento humano.

A partir dos resultados preliminares, mostrados nos gráficos, é possível verificar que o IMU com fio e comunicação serial, pode ser utilizado para o propósito estabelecido inicialmente, porém, este construído ainda é apenas um protótipo, vários testes ainda serão necessários. Os gráficos ficariam melhores se os testes não tivessem sido realizados com os métodos que foram utilizados, onde, o sensor acabou medindo a movimentação da mão que o manipulou, tornando o movimento irregular, e também acabou medindo os impactos sobre o mesmo.

A multiplexação dos sensores através do próprio *firmware* do Arduíno, se mostrou viável para utilização, pois a frequência obtida estava dentro da Taxa de Nyquist, porém ainda necessita de otimização devido estar próxima da frequência mínima;

O comunicação sem fio foi estabelecida, os dados são todos transmitidos e com uma boa velocidade, mas não suficiente para esta aplicação. Provavelmente essa taxa de transmissão que foi atingida pode ser elevada a uma frequência muito mais alta. Os dados foram transmitidos em forma de *String* (um vetor de caracteres), cada caractere ocupa 1 *byte*. esse vetor que estava sendo transmitido possui um tamanho variável entre 7 e *bytes*. Os dados obtidos através do sensor são enviados no barramento I^2C através de um vetor de inteiros com 14 *bytes*. Portanto, conseguindo transmitir os dados com mesmo formato que eles são obtidos, haveria um ganho na taxa de transmissão e processamento.

5 CONCLUSÃO

A medição dos movimentos humanos é bem importante para estudos sobre prevenção de lesão em atletas, correção de postural e outros tratamentos fisioterápicos. Para realizar essas medições são necessários equipamentos específicos. E o IMU surge como uma solução viável para facilitar o acesso dos pesquisadores a sensores de utilização menos complexas e de custo mais baixo.

Foi concluído, neste trabalho, o protótipo de um sensor IMU capaz de medir a aceleração linear em três eixos, e a velocidade angular em três eixos. Os parâmetros que podem ser calculados com esse dados são velocidade linear, deslocamento e ângulos em relação aos 3 eixos. É possível utilizar mais de um sensor conectado a um único microcontrolador para obtenção dos parâmetros de movimento em até 3 pontos de interesse simultaneamente com a multiplexação dos canais de entrada. A partir dos resultados obtidos é possível afirmar que os sensores podem ser utilizados para realizar coleta de parâmetros de movimento. Sendo necessário validar os dados obtidos através do sensor, o que não foi o objetivo desse trabalho.

A comunicação sem fio pode ser útil para utilização do sensor com maior grau de liberdade de movimentos, mas o protocolo utilizado nesse projeto ainda não é o ideal, devendo ser otimizado para realização de coletas mais precisas.

Como proposta para trabalhos futuros fica a validação dos dados obtidos, podendo ser realizado um estudo de caso e comparação de resultados com outros mecanismos de medição que já tem sua acurácia comprovada. A otimização do protocolo de comunicação sem fio e do sistema de multiplexação para que permita a utilização de mais de 3 sensores em uma única coleta. Melhorar a *interface* para uso de profissionais de diversas áreas. E tentar tornar o IMU mais miniaturizado, tentando integrar uma única placa o transmissor sem fio, o microcontrolador e o sensor de movimento.

REFERÊNCIAS

- ACQUESTA, F. M. et al. Estudo da biomecânica do movimento humano no brasil. *Revista Brasileira de Biomecânica*, 2008. Citado na página 21.
- AHMAD, N.; GHAZILLA, R. A. R.; KHAIRI, N. M. Reviews on various inertial measurement unit (imu) sensor applications. *International Journal of Signal Processing Systems*, 2013. Citado 2 vezes nas páginas 26 e 27.
- ALMEIDA, V. M. de. Sensores inerciais. 2014. Acesso em: 06/07/2018. Disponível em: <<http://www.decom.ufop.br/imobilis/sensores-inerciais/>>. Citado na página 29.
- AMADIO, A. C. Metodologia biomecânica para o estudo das forças internas ao aparelho locomotor: importância e aplicaçõesno movimento humano. In: *A Biomecânica do Movimento e Suas Relações Interdisciplinares*. São Paulo, Brasil: [s.n.], 2000. p. 45–70. Citado na página 23.
- AMADIO, A. C.; SERRÃO, J. C. Conyextualização da biomecânica para a investigação do movimento: fundamentos, métodos e aplicações para análise da técnica esportiva. In: *Revista Brasileira de Educação Física e Esporte*. São Paulo, Brasil: [s.n.], 2007. v. 21, p. 61–85. Citado 2 vezes nas páginas 23 e 24.
- CARNEIRO, F. M. Levantamento bibliográfico das tecnologias dos acelerômetros comerciais. 2003. Acesso em: 05/07/2018. Disponível em: <http://www.fem.unicamp.br/~lotavio/tgs/2003_BibliografiaDeAcelerometros_TG_FelipeCarneiro.pdf>. Citado na página 28.
- CHANG, H.; GEORGY, J.; EL-SHEIMY, N. Improved cycling navigation using inertial sensors measurements from portable devices with arbitrary orientation. *IEEE Transactions on Instrumentation and Measurement*, 2016. Acesso em: 29/06/2018. Disponível em: <<https://ieeexplore.ieee.org/document/7112512/citations?tabFilter=papers>>. Citado 2 vezes nas páginas 21 e 22.
- CHAVIER, L. F. Programação para arduino. 2016. Acesso em: 05/07/2018. Disponível em: <<<https://www.circuitar.com.br/tutoriais/programacao-para-arduino-primeiros-passos/>>>. Citado na página 33.
- DAWSON, M. *Python® Programming for the Absolute Beginner, Third Edition*. Massachusetts - USA: Cengage Learning, 2010. 480 p. Citado na página 34.
- DOWNEY, A. *Think Python: How to Think Like a Computer Scientist*. Massachusetts - USA: Green Tea Pess, 2012. 240 p. Citado na página 35.
- FORHAN, N. A. E. Giroscópios mems. São José dos Campos - SP, Brazil, 2010. Acesso em: 05/07/2018. Disponível em: <<<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/01.25.18.42>>>. Citado 2 vezes nas páginas 29 e 30.
- HOWARD, R. Wireless sensor devices in sports performance. *IEEE Potentials*, 2016. Citado na página 27.

INVENSENSE, I. *MPU-6000 and MPU-6050 Product Specification Revision 3.4.* [S.l.], 2013. Citado 5 vezes nas páginas 11, 27, 30, 31 e 40.

JOHN, D. Arduino nano tutorial - pinout and schematics. 2018. Acesso em: 09/07/2018. Disponível em: <<http://www.circuitstoday.com/arduino-nano-tutorial-pinout-schematics>>. Citado 2 vezes nas páginas 32 e 33.

KOLBAN, N. *Kolban's Book on ESP8266.* [S.l.: s.n.], 2016. v. 1. 436 p. Citado 2 vezes nas páginas 33 e 34.

MADEIRO, F.; LOPES, W. T. A. Introdução à compressão de sinais. *Revista de Tecnologia da Informação e Comunicação*, 2011. Citado na página 24.

MCGINNIS, R. S. *Advancing Applications of IMUs in Sports Training and Biomechanics.* Tese (Doutorado) — University of Michigan, 2013. Citado na página 21.

MEDEIROS, M. F. Identificação de assimetrias bilaterais dos membros inferiores por meio de salto vertical. Minas Gerais, Brasil, 2012. Citado na página 23.

MIZIARA, I. M. Proposta de um sistema para avaliação biomecânica de atletas de taekwondo. Uberlândia - MG, Brasil, 2014. Citado na página 23.

NUSSENZVEIG, H. M. *Curso de Física Básica: Mecânica.* [S.l.]: Blucher, 2013. v. 1. 336 p. Citado 2 vezes nas páginas 28 e 30.

OBERLANDER, K. D. *Inertial Measurement Unit (IMU) Technology: Inverse Kinematics: Joint Considerations and the Maths for Deriving Anatomical.* Tese (Doutorado) — University of Koblenz-Ladau, 2015. Citado 2 vezes nas páginas 21 e 22.

OKAZAKI, V. H. A. et al. Ciência e tecnologia aplicada à melhoria do desempenho esportivo. *Revista Mackenzie de Educação Física e Esporte*, 2012. Citado na página 21.

OLIVEIRA, R. R. de. Uso do miconcontrolador esp8266 para automação residencial. 2017. Citado na página 33.

OPPENHEIM, A. V.; WILLSKY, A. S.; NAWAB, S. H. *Sinais e Sistemas.* Brasil: Pearson, 2010. 592 p. Citado na página 24.

PRIME. *An Introduction to MEMS (Micro-electromechanical Systems).* [S.l.]: Prime Faraday Technology Watch, 2002. 56 p. Citado 2 vezes nas páginas 25 e 26.

REITZ, K. *Python Guide Documentation.* [S.l.: s.n.], 2018. v. 1. 129 p. Citado na página 34.

ROBERTS, M. M. *Arduino Básico.* São Paulo, Brazil: Novatec, 2011. 456 p. Citado na página 32.

ROCHA, M. Fisiologia do exercício. In: *Atlas do esporte no Brasil.* Rio de Janeiro, Brasil: [s.n.], 2005. p. 657 – 659. Citado na página 23.

SANJEEV, A. How to interface arduino and the mpu 6050 sensor. 2018. Acesso em: 08/07/2018. Disponível em: <<https://maker.pro/arduino/tutorial/how-to-interface-arduino-and-the-mpu-6050-sensor>>. Citado 3 vezes nas páginas 11, 28 e 29.

- SANTOS, C. P.; VIEIRA, M. E. M.; JUNIOR, S. L. S. Sensores inercias aplicados à marcha humana no esporte. *SEA-Seminário de Eletrônica e Automação*, 2016. Citado na página 26.
- SMITH, S. *Wearable Technology and Gesture Recognition for Live Performance Augmentation*. Tese (Doutorado) — University of Southern Queensland, 2016. Citado 3 vezes nas páginas 13, 31 e 32.
- SONG, M.; GODOY, R. I. How fast is your body motion? determining a sufficient frame rate for an optical motion tracking system using passive markers. *Plos One*, 2016. Citado 2 vezes nas páginas 24 e 25.
- SZYGALSKI, C. A. Uma visão geral do http. 2018. Acesso em: 05/12/2018. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>>. Citado 2 vezes nas páginas 35 e 36.
- VIEIRA, N. Entendendo um pouco mais sobre o protocolo http. 2007. Acesso em: 05/12/2018. Disponível em: <<https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http>>. Citado na página 35.
- WEMOS. D1: An arduino uno compatible wifi board based on esp8266ex. 2017. Acesso em: 05/12/2018. Disponível em: <<<https://wiki.wemos.cc/products:d1:d1>>>. Citado 2 vezes nas páginas 33 e 34.

Apêndices

APÊNDICE A – CÓDIGO FONTE DO ARDUÍNO COM UM SENSOR

```
// Programa para leitura do Módulo MPU 6050  
// Autor: Eduardo-ssr
```

```
//Carrega as bibliotecas  
#include<Wire.h>  
#include <I2Cdev.h>  
#include <MPU6050.h>  
  
//Endereco I2C do MPU6050  
const int MPU=0x68;  
  
//Variaveis para armazenar valores dos sensores  
double AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;  
char command;  
int test = 0;  
int c = 0;  
  
void setup()  
{  
Serial.begin(115200);  
Serial.print("Digite o comando.(0,1,2 ou 3)&\n");  
delay(1000);  
Wire.begin();  
do{  
if (Serial.available()) {  
  
command = Serial.read();  
  
if(command == '0'){  
Serial.print("Comando 0 recebido.\n");  
c = 0;
```

```
test = 1;
delay(2);
}
else if(command == '1'){

    Serial.print("Comando 1 recebido.\n");
c = 1;
test = 1;
delay(2);
}
else if (command == '2'){
Serial.print("Comando 2 recebido.\n");
c = 2;
test = 1;
delay(2);
}
else if (command == '3'){
Serial.print("Comando 3 recebido.\n");
c = 3;
test = 1;
delay(2);
}
else{
Serial.print("Comando invalido. Tente outro.\n");
delay(1000);
test = 0;
}
}

}while(test == 0);
Wire.beginTransmission(MPU);
Wire.write(0x6B);

//Inicializa o MPU-6050
Wire.write(c);
Wire.endTransmission(true);
}
void loop()
{
```

```
Wire.beginTransmission(MPU);
Wire.write(0x3B); // starting with register
0x3B (ACCEL_XOUT_H)
Wire.endTransmission(false);

//Solicita os dados do sensor
Wire.requestFrom(MPU,14,true);

//Armazena o valor dos sensores nas variaveis correspondentes
AcX=Wire.read()<<8|Wire.read(); //0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
AcY=Wire.read()<<8|Wire.read(); //0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
AcZ=Wire.read()<<8|Wire.read(); //0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
Tmp=Wire.read()<<8|Wire.read(); //0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
GyX=Wire.read()<<8|Wire.read(); //0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
GyY=Wire.read()<<8|Wire.read(); //0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
GyZ=Wire.read()<<8|Wire.read(); //0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

//Envia valor X do acelerometro para a serial
Serial.print(AcX);
Serial.print("\t");

//Envia valor Y do acelerometro para a serial
Serial.print(AcY);
Serial.print("\t");

//Envia valor Z do acelerometro para a serial
Serial.print(AcZ);
Serial.print("\t");

//Envia valor da temperatura para a serial
//Calcula a temperatura em graus Celsius
Serial.print(Tmp);//(Tmp/340.00+36.53)
Serial.print("\t");

//Envia valor X do giroscopio para a serial
Serial.print(GyX);
Serial.print("\t");
```

```
//Envia valor Y do giroscópio para a serial  
Serial.print(GyY);  
Serial.print("\t");
```

```
//Envia valor Z do giroscópio para a serial  
Serial.println(GyZ);  
//Serial.print("\t");  
//Serial.println();  
// delay(100);
```

```
}
```

APÊNDICE B – CÓDIGO ARDUINO COM 3 SENsoRES

```
#include <Wire.h>

uint8_t i2c_data[14];

double accX, accY, accZ;
double tmp;
double gyroX, gyroY, gyroZ;

uint32_t timer;

void setup() {

    Serial.begin(115200);
    Wire.begin();

#if if ARDUINO >= 157
    Wire.setClock(400000UL); // Freq = 400kHz.
#else
    TWBR = ((F_CPU/400000UL) - 16) / 2; // Freq = 400kHz
#endif

    i2c_data[0] = 7;      /* 0x19 - Taxa de amostragem 8kHz/(7 + 1) = 1000Hz */
    i2c_data[1] = 0x00;   /* 0x1A - Desabilitar FSYNC, Configurar o Filtro de ACC 260Hz */
    i2c_data[2] = 0x00;   /* 0x1B - Configurar o fundo de escala do Gyro 250deg/s - Faixa */
    i2c_data[3] = 0x00;   /* 0x1C - Configurar o fundo de escala do Acelerometro para 2g */

    while(i2cWrite(0x19, i2c_data, 4, false));

    /* PLL tenha como referencia o gyro de eixo X, Desabilitando Sleep Mode */
    while(i2cWrite(0x6B, 0x01, true));

    /* */
    while(i2cRead(0x75, i2c_data, 1));

    if(i2c_data[0] != 0x68 && i2c_data[0] != 0x69){
        Serial.print("Erro .\u2014 Placa\u2014 desconhecida\n");
        while(1){
            Serial.print("Erro .\u2014 Conecte\u2014 a\u2014 MPU6050\u2014 no\u2014 barramento\u2014 i2c\n");
        }
    }
}
```

```

}

/* Tempo de estabilizacao do Sensor MPU6050 */
delay(100);

/* 1 - Leitura dos dados de Acc XYZ */
while(i2cRead(0x3B, i2c_data, 14));

/* 2 - Organizar os dados de Acc XYZ */
accX = (int16_t)((i2c_data[0] << 8) | i2c_data[1]); // ([ MSB ] [ LSB ])
accY = (int16_t)((i2c_data[2] << 8) | i2c_data[3]); // ([ MSB ] [ LSB ])
accZ = (int16_t)((i2c_data[4] << 8) | i2c_data[5]); // ([ MSB ] [ LSB ])

timer = micros();

}

void loop() {

/* Leitura dos Dados de Aceleracao e Gyro do sensor MPU6050 */
while(i2cRead(0x3B, i2c_data, 14));

/* Aceleracao */
accX = (int16_t)((i2c_data[0] << 8) | i2c_data[1]); // ([ MSB ] [ LSB ])
accY = (int16_t)((i2c_data[2] << 8) | i2c_data[3]); // ([ MSB ] [ LSB ])
accZ = (int16_t)((i2c_data[4] << 8) | i2c_data[5]); // ([ MSB ] [ LSB ])

/* Temperatura */
tmp = (int16_t)((i2c_data[6] << 8) | i2c_data[7]);

/* Giroscopio */
gyroX = (int16_t)((i2c_data[8] << 8) | i2c_data[9]); // ([ MSB ] [ LSB ])
gyroY = (int16_t)((i2c_data[10] << 8) | i2c_data[11]); // ([ MSB ] [ LSB ])
gyroZ = (int16_t)((i2c_data[12] << 8) | i2c_data[13]); // ([ MSB ] [ LSB ])

/* Calculo do Delta Time */
double dt = (double)(micros() - timer)/1000000;
timer = micros();

/* Transmissao dos dados para porta Serial*/
Serial.print(dt); Serial.print("\t");
Serial.print(accX); Serial.print("\t");
Serial.print(accY); Serial.print("\t");
Serial.print(accZ); Serial.print("\t");
Serial.print(tmp); Serial.print("\t");
}

```

```
Serial.print(gyroX); Serial.print("\t");
Serial.print(gyroY); Serial.print("\t");
Serial.print(gyroZ); Serial.print("\n");
}
```


APÊNDICE C – CÓDIGO WEMOS D1 COMO PONTO DE ACESSO

```
#include <ESP8266WiFi.h>
#include <Wire.h>

const char *ssid = "Teste_de_AP";
const char *password = "mpu6050";

WiFiServer server(80);

String html;// String que armazena o corpo do site.
char dados[100];
uint8_t i2c_data[14];

int accX, accY, accZ;
int tmp;
int gyroX, gyroY, gyroZ;

void setup() {
    // put your setup code here, to run once:
    // Serial.begin(115200);
    Serial.begin(2000000);
    Wire.begin();
}

#if ARDUINO >= 157
Wire.setClock(400000UL); // Freq = 400kHz.
#else
TWBR = ((F_CPU/400000UL) - 16) / 2; // Freq = 400kHz
#endif

i2c_data[0] = 7;      /* 0x19 - Taxa de amostragem 8kHz/(7 + 1) = 1000Hz */
i2c_data[1] = 0x00;   /* 0x1A - Desabilitar FSYNC, Configurar o Filtro de ACC 260Hz */
i2c_data[2] = 0x00;   /* 0x1B - Configurar o fundo de escala do Gyro 250deg/s - Faixa */
i2c_data[3] = 0x00;   /* 0x1C - Configurar o fundo de escala do Acelerometro para 2g */

/* Configuracoes do i2c*/
while(i2cWrite(0x19, i2c_data, 4, false));

/* PLL tenha como referencia o gyro de eixo X, Desabilitando Sleep Mode */
while(i2cWrite(0x6B, 0x01, true));

/* */
```

```

while(i2cRead(0x75, i2c_data , 1));

if(i2c_data[0] != 0x68 && i2c_data[0] != 0x69){
    // Serial.print("Erro ._Placa_desconhecida\n");
    while(1){
        // Serial.print("Erro ._Conecte_a_MPU6050_no_barramento_i2c\n");
    }
}

IPAddress staticIP(192, 168, 4, 2); // IP set to Static
IPAddress gateway(192, 168, 4, 1); // gateway set to Static
IPAddress subnet(255, 255, 255, 0); // subnet set to Static

WiFi.mode(WIFI_AP); // Working mode only as Acess Point

WiFi.softAP(ssid, password, 2, 0);
WiFi.config(staticIP, gateway, subnet);

server.begin();

// Serial.println("Server_started");
// Serial.println(WiFi.softAPIP());
}

void loop() {

while(i2cRead(0x3B, i2c_data , 14));

/* Aceleracao */
accX = (int)((i2c_data[0] << 8) | i2c_data[1]); // ([ MSB ] [ LSB ])
accY = (int)((i2c_data[2] << 8) | i2c_data[3]); // ([ MSB ] [ LSB ])
accZ = (int)((i2c_data[4] << 8) | i2c_data[5]); // ([ MSB ] [ LSB ])

/* Temperatura */
tmp = (int)((i2c_data[6] << 8) | i2c_data[7]);

/* Giroscopio */
gyroX = (int)((i2c_data[8] << 8) | i2c_data[9]); // ([ MSB ] [ LSB ])
gyroY = (int)((i2c_data[10] << 8) | i2c_data[11]); // ([ MSB ] [ LSB ])
gyroZ = (int)((i2c_data[12] << 8) | i2c_data[13]); // ([ MSB ] [ LSB ])

WiFiClient client = server.available();
if (!client) {
    return;
}

// Wait until the client sends some data

```

```
while (!client.available()) {
//  delay(1);
}

String req = client.readStringUntil('\r');
req = req.substring(req.indexOf("/") + 1, req.indexOf("HTTP") - 1);
// Serial.println(req);
client.flush();

// Match the request

client.print("HTTP/1.1 200 OK\n\n");
if (req.indexOf("D") != -1)
{
    sprintf(dados, "%d %d %d %d %d %d\n", (accX), (accY), (accZ), (t
    client.print(dados);
    client.print('/n')
}
else if (req.indexOf("R") != -1)
{
    client.print("REcebido seu dado R\n");
}
else {
    client.print("Invalid Request");
    client.flush();
    client.stop();
    return;
}

// client.print("HTTP/1.1 200 OK\n\n");
client.flush();

}
```


APÊNDICE D – CÓDIGO DO PROGRAMA EM PYTHON COM UM SENSOR

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

Created on Fri Jun 29 09:31:48 2018

@author: eduardo-ssr
"""

import sys
import serial
import glob
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from drawnow import drawnow
from time import sleep
from datetime import datetime

read1 = []
read2 = []
read3 = []
read4 = []
read5 = []
read6 = []
read7 = []

acelX = []
acelY = []
acelZ = []
gyroX = []
gyroY = []
gyroZ = []

constant_Calib_Acel = (16384/9.81)
constant_Calib_Gyro = 131

```

```

t=0

os.system('clear')
print ("=====Lista_de_dispositivos_USB=====")
serial_ports = glob.glob('/dev/ttyUSB*')
while(len(serial_ports)==0):
    print ("Conecte_o_Arduino...\n")
    sleep(5);
    serial_ports = glob.glob('/dev/ttyUSB*')

for i in range(len(serial_ports)):
    print (i, " - ", serial_ports[i])
    port = input("Escolha_a_porta_do_Arduino_(e.g._0):_")

ser = serial.Serial(serial_ports[int(port)], 115200,timeout=1)

def main():
    global contador
    global t
    global constant_Calib_Acel
    global constant_Calib_Gyro

    if ser.is_open:
        print ("Comunicacao_Serial_Estabelecida.\n")
    else :
        print ("Erro_na_Comunicacao_Serial.\n")

    while(t==0):
        sleep(3)
        print(ser.readline().decode("utf-8"))
        print("\n")
        print("Intervalos_de_leitura : \n")
        print("0---->+-250_deg/s_e+-2g\n")
        print("1---->+-500_deg/s_e+-4g\n")
        print("2---->+-1000_deg/s_e+-8g\n")
        print("3---->+-2000_deg/s_e+-16g\n")

        command = (input("Escolha_o_intervalo_de
leitura_(e.g._0):_"))

        sleep(1.8)

        if command == '0':
            ser.write(b'0')
            constant_Calib_Acel = (16384/9.81)
            constant_Calib_Gyro = 131
            t = 1

```

```

elif command == '1':
    ser.write(b'1')
    constant_Calib_Acel = (8192/9.81)
    constant_Calib_Gyro = 65.5
    #sleep(2)
    t = 1
elif command == ('2'):
    ser.write(b'2')
    constant_Calib_Acel = (4096/9.81)
    constant_Calib_Gyro = 32.8
    #sleep(2)
    t = 1
elif command == ("3") :
    ser.write(b'3')
    constant_Calib_Acel = (2048/9.81)
    constant_Calib_Gyro = 16.4

    t = 1
else:
    print("Comando invalido , tente outro .\n")
    t = 0

input("Pressione 'Enter' para iniciar a leitura e
        Ctrl+C" para pausar:.\n")

while(True):
    try:
        line = ser.readline().decode("utf-8")

        print(line)
        try:
            entry = line.split("\t")
            AcX = np.float(entry[0])
            AcY = np.float(entry[1])
            AcZ = np.float(entry[2])
            temp = np.float(entry[3])
            Gx = np.float(entry[4])
            Gy = np.float(entry[5])
            Gz = np.float(entry[6])

            #entre --20 m/s^2 e + 20 m/s^2
            ACX = AcX/constant_Calib_Acel

            #entre --20 m/s^2 e + 20 m/s^2
            ACY = AcY/constant_Calib_Acel

```

```

#entre --20 m/s^2 e + 20 m/s^2
ACZ = AcZ/constant_Calib_Acel

Temp = temp/340.00 + 36.53

# entre +250deg/s e -250deg/s
GX = Gx/constant_Calib_Gyro
# entre +250deg/s e -250deg/s
GY = Gy/constant_Calib_Gyro
# entre +250deg/s e -250deg/s
GZ = Gz/constant_Calib_Gyro

read1.append(AcX)
read2.append(AcY)
read3.append(AcZ)
read4.append(Gx)
read5.append(Gy)
read6.append(Gz)
read7.append(temp)

acelX.append(ACX)
acelY.append(ACY)
acelZ.append(ACZ)
gyroX.append(GX)
gyroY.append(GY)
gyroZ.append(GZ)

except (ValueError):
    print("Erro_de_valor.")
    pass

except (KeyboardInterrupt):
    now = datetime.now()
    print ("Voce_pressionou_Ctrl+C_para_interromper
este_programa!_Seus_dados_foram_salvos
em '_Dados_%s.csv'"%(str(now)[-7]))

ser.close()

plt.plot(acelX, "-r")
plt.plot(acelY, "-g")
plt.plot(acelZ, "-b")
plt.xlabel('Tempo_(ms)');
plt.ylabel('Aceleracao_(m/s^2)');
plt.show()

```

```
plt.plot(gyroX, "-r")
plt.plot(gyroY, "-g")
plt.plot(gyroZ, "-b")
plt.xlabel('Tempo_(ms)');
plt.ylabel('Velocidade_Angular_(deg/s)');
plt.show()

x = np.vstack((read1, read2, read3, read4, read5, read6,
               read7))
y = np.vstack((acelX, acelY, acelZ, gyroX, gyroY, gyroZ))

np.savetxt('Dados_Brutos_%s.csv'%str(now)[-7]),
          np.transpose(x), delimiter=';')
np.savetxt('Dados_Fisicos_%s.csv'%str(now)[-7]),
          np.transpose(y), delimiter=';')

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


APÊNDICE E – CÓDIGO DO PROGRAMA EM PYTHON COM 3 SENSORES

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""

Created on Fri Jun 29 09:31:48 2018

@author: eduardo-ssr
"""

import sys
import serial
import glob
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from drawnow import drawnow
from time import sleep
from datetime import datetime
import timeit
from scipy import signal

%matplotlib inline

read0 = []
read1 = []
read2 = []
read3 = []
read4 = []
read5 = []
read6 = []
read7 = []

acelX = []
acelY = []
acelZ = []
gyroX = []
gyroY = []
gyroZ = []

```

```

fx = []
#constant_Calib_Acel = (16384/9.81)
constant_Calib_Acel = (16384/9.81)
constant_Calib_Gyro = 131

os.system('clear')
print("=====Lista de dispositivos USB=====")
serial_ports = glob.glob('/dev/ttyUSB*')
while(len(serial_ports)==0):
    print("Conekte o Arduino...\n")
    sleep(5);
    serial_ports = glob.glob('/dev/ttyUSB*')

for i in range(len(serial_ports)):
    print(i, " - ", serial_ports[i])
    port = input("Escolha a porta do Arduino (e.g., 0):\n")

ser = serial.Serial(serial_ports[int(port)], 2000000, timeout=1)

def main():
    global contador
    global t
    global constant_Calib_Acel
    global constant_Calib_Gyro

    if ser.is_open:
        print("Comunicacao Serial Estabelecida.\n")
    else :
        print("Erro na Comunicacao Serial.\n")

    input("Pressione 'Enter' para iniciar a leitura e \"I\" duas vezes para pausar:\n")

    i = 0
    inicio = timeit.default_timer()

    while(True):
        try:

            line = ser.readline().decode("utf-8")
            print(line)

            try:
                entry = line.split("\t")
                if len(entry) != 8 :

```

```

print('Erro_de_valor.')
else:
    n = np.int(entry[0])
    AcX = np.float(entry[1])
    AcY = np.float(entry[2])
    AcZ = np.float(entry[3])
    temp = np.float(entry[4])
    Gx = np.float(entry[5])
    Gy = np.float(entry[6])
    Gz = np.float(entry[7])

    ACX = AcX/constant_Calib_Acel # entre --20 m/s2 e + 20 m/s2
    ACY = AcY/constant_Calib_Acel # entre --20 m/s2 e + 20 m/s2
    ACZ = AcZ/constant_Calib_Acel # entre --20 m/s2 e + 20 m/s2

    Temp = temp/340.00 + 36.53

    GX = Gx/constant_Calib_Gyro # entre +250deg/s e -250deg/s
    GY = Gy/constant_Calib_Gyro # entre +250deg/s e -250deg/s
    GZ = Gz/constant_Calib_Gyro # entre +250deg/s e -250deg/s

    read0.append(n)
    read1.append(AcX)
    read2.append(AcY)
    read3.append(AcZ)
    read4.append(Gx)
    read5.append(Gy)
    read6.append(Gz)
    read7.append(temp)

    acelX.append(ACX)
    acelY.append(ACY)
    acelZ.append(ACZ)
    gyroX.append(GX)
    gyroY.append(GY)
    gyroZ.append(GZ)

    fx.append(i)
    i = i+1

except (ValueError):
    print("Erro_de_valor.")
    pass

except (KeyboardInterrupt):
    fim = timeit.default_timer()
    tempo = fim - inicio

```

```
now = datetime.now()

print('Tempo_medio_de', i, 'execucoes_foi_de', tempo/i, 's', '\n')
print('O_tempo_total_foi:', tempo)

print ("Voce_interrompeu_este_programa.\n")

ser.close()

x = np.vstack((read0, read1, read2, read3, read4, read5, read6, read7))
y = np.vstack((read0, acelX, acelY, acelZ, gyroX, gyroY, gyroZ))

r = input('Deseja_salvar_os_dados_adquiridos_(S/N)? ')
if r == 'S' or r == 's':

    np.savetxt('Dados_Brutos_%s.csv'%(str(now)[:-7]), np.transpose(x), delimiter=',')
    np.savetxt('Dados_Fisicos_%s.csv'%(str(now)[:-7]), np.transpose(y), delimiter=',')
    print ('Seus_dados_foram_salvos_como: Dados_Fisicos_%s.csv'%(str(now)[:-7]))

break

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

APÊNDICE F – FUNÇÕES DE COMUNICAÇÃO

```

const uint8_t IMUAddress = 0x68; // AD0 is logic low on the PCB
const uint16_t I2C_TIMEOUT = 1000; // Used to check for errors in I2C communication

uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop) {
    return i2cWrite(registerAddress, &data, 1, sendStop); // Returns 0 on success
}

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop)
{
    Wire.beginTransmission(IMUAddress);
    Wire.write(registerAddress);
    Wire.write(data, length);
    uint8_t rcode = Wire.endTransmission(sendStop); // Returns 0 on success
    if (rcode) {
        Serial.print(F("i2cWrite failed: "));
        Serial.println(rcode);
    }
    return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
}

uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {
    uint32_t timeOutTimer;
    Wire.beginTransmission(IMUAddress);
    Wire.write(registerAddress);
    uint8_t rcode = Wire.endTransmission(false); // Don't release the bus
    if (rcode) {
        Serial.print(F("i2cRead failed: "));
        Serial.println(rcode);
        return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
    }
    Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true); // Send a repeated start and the address
    for (uint8_t i = 0; i < nbytes; i++) {
        if (Wire.available())
            data[i] = Wire.read();
        else {
            timeOutTimer = micros();
            while (((micros() - timeOutTimer) < I2C_TIMEOUT) && !Wire.available())
                if (Wire.available())
                    data[i] = Wire.read();
                else {
                    Serial.println(F("i2cRead timeout"));
                }
        }
    }
}

```

```
        return _5; // This_error_value_is_not_already_taken_by_endTransmission
    }
}
return _0; // Success
}
```

APÊNDICE G – CÓDIGO DO PROGRAMA EM PYTHON PARA LEITURA HTML

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import timeit

try:
    inicio = timeit.default_timer()
    cont = 0
    while(True):
        cont+=1
        url = "http://192.168.4.1/D"
        html = urlopen(url).read()
        soup = BeautifulSoup(html)

        # get text
        text = soup.get_text()

        print('Resultado:', text, '\n')

except KeyboardInterrupt:
    fim = timeit.default_timer()
    tempo = fim - inicio

    print('Tempo_medio_de', cont, 'execucoes_foi_de', tempo/cont, 's', '\n')
    print('O_tempo_total_foi:', tempo)

    print('Laco_interrompido.\n')
```