UNIVERSIDADE FEDERAL DO PARANÁ

GRR20141294 RENATO FRAGA TREVIZAN

SISTEMA EMBARCADO DE DETECÇÃO DE QUEDAS INTELIGENTE ATRAVÉS DE SENSORES UTILIZANDO ESP32

CURITIBA

2019

GRR20141294 RENATO FRAGA TREVIZAN

SISTEMA EMBARCADO DE DETECÇÃO DE QUEDAS INTELIGENTE ATRAVÉS DE SENSORES UTILIZANDO ESP32

Projeto apresentado como requisito à matrícula na disciplina de Trabalho de Conclusão de Curso B, do Curso de Engenharia Elétrica com Ênfase em Sistemas Eletrônicos Embarcados.
Orientador: Prof. Dr. Marcelo Eduardo Pellenz

CURITIBA

2019

AGRADECIMENTOS

Se tornar engenheiro foi um projeto que se iniciou em uma criança apaixonada pela ciências exatas e fascinada pelas novas tecnologias que surgiam com o passar dos anos. E nessa altura, esse sonho está muito próximo de se tornar realidade e tenho que agradecer por todas as pessoas que acompanharam essa trajetória e que foram essencial em todos os momentos.

Primeiramente, gostaria de agradecer a minha família, em especial aos meus pais, Ricardo Trevizan e Jaqueline Fraga Trevizan, aos meus irmãos, Rodrigo Trevizan e Marina Trevizan, que mesmo com todas as dificuldades, entenderam as minhas incalculáveis ausências e não mediram esforços para esse sonhos se tornasse realidade, sempre com muito amor e carinho.

Agradeço aos meus amigos, em especial ao Lin Ting Yen, Matheus Elias, Eduardo Henrique, Rafaela Quirino, Leandro Fontana, Wiviane Maneira e ao Gustavo Censi, que batalharam lado a lado durante a toda essa árdua trajetória e que vibraram comigo por cada conquista ao longo desses inúmeros semestres.

Agradeço aos professores e servidores da Universidade Federal do Paraná, na qual apresentaram seus criteriosos métodos e ensinamentos que, sem eles, esse trabalho não seria possível. Agradeço em especial ao Professor Marcelo Pellenz pela sua valiosa orientação e contribuição para a melhoria desse projeto.

E por fim, sou grato a todos que de alguma forma, direta ou indiretamente, participaram da realização desse projeto. Tenho certeza que sem o comprometimento e a paciência de todos os envolvidos para a execução deste projeto, tudo seria muito mais desgastante.

“Se cheguei até aqui foi porque me apoiei no ombro dos gigantes.”

Isaac Newton

RESUMO

Saúde é uma área muito delicada, importante e ampla na nossa sociedade, onde um dos meios de monitoramentos passivo mais utilizado é por meio de sensoriamentos. Sabe-se que a rápida ação em determinados casos, pode evitar grandes lesões e até mesmo salvar vidas, como por exemplo acionamentos de alarmes para detecção de quedas em usuários específicos. Os dados divulgados pela Organização Mundial de Saúde deixa claro que as quedas e as consequentes lesões resultantes constituem um problema de saúde pública e de grande impacto social, principalmente em pessoas idosas. Dessa forma, o objetivo desse projeto é propor um sistema de detecção de quedas através de sensores, que estabeleça a melhor relação precisão/desempenho num sistema embarcado de baixo custo. Visto que o mercado atualmente é carente ou até mesmo pouco acessível de um sistema inteligente de monitoramento capaz de perceber quedas e com bom tempo de resposta a um custo acessível.

Palavras-chave: Detector. Quedas. Embarcado. Aceletômetro. Giroscópio. Magnetômetro. ESP32. *Android*

ABSTRACT

Health is a very delicate and important area in our society where one of the most commonly used monitoring is through sensors. Rapid action in certain cases can prevent major injuries and even save lives, such as triggering alarms for detect falls in specific users. The report released by the World Health Organization make it clear that falls and their resulting injuries are a public health problem and of major social impact, especially in the elderly. Thus, the main objective of this project is to propose a sensor-based fall detection system that establishes the best accuracy / performance ratio in a low-cost embedded system. Since the market is currently lacking of option or even inaccessible to an intelligent monitoring system that can detect falls and with good response time at an accessible cost.

Key-words: *Detector*. *Fall*. *Acelerometer*. *Gyroscope.Magnetometer*. ESP32. *Android*

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analog-to-Digital Converter*
ADL	*Activities of Daily Living*
BLE	*Bluetooth Low Energy*
CPU	*Central Processing Unity*
CSV	*Comma-separated values*
GPIO	*General Purpose Input/Output*
I2C	*Inter-Integrated Circuit*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
IP	*Internet Protocol address*
LE	*Low Energy*
LED	*Light-emitting Diode*
PHY	*Physical Layer*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
ROM	*Read Only Memory*
RF	*Rádio Frequência*
SO	*Sistema Operacional*
SMS	*Short Message Service*
SRAM	*Static Random Access Memory*
SPI	*Serial Peripheral Interface*

UART	*Universal Asynchrounous Receiver/Transmiter*
UUID	*Universally Unique Identifier*
USB	*Universal Serial Bus*
Wi-Fi	*Wireless Fidelit*

SUMÁRIO

1 INTRODUÇÃO

Observando os dados divulgado pela Organização Mundial de Saúde fica claro que as quedas e as consequentes lesões resultantes constituem um problema de saúde pública e de grande impacto social enfrentado hoje por todos os países em que ocorre expressivo envelhecimento populacional (WHO, 2017).

Aproximadamente 28-35% das pessoas com 65 anos ou mais caem a cada ano , aumentando para 32-42% para as pessoas com mais de 70 anos . A frequência das quedas aumenta com a idade e o nível de fragilidade. Aproximadamente 30-50% das pessoas que vivem em instituições de longa permanência caem a cada ano e 40% delas experimentam quedas recorrentes (WHO, 2017).

O mercado atualmente é carente ou pouco acessível de um sistema inteligente de monitoramento capaz de perceber detecção e com bom tempo de resposta a um custo acessível. Os sistemas atuais estão restritos ao uso de tecnologias caras, e as vezes inacessível para a população necessitada.

Pode-se tirar conclusões e fazer levantamentos de dados através de dispositivos *wearable*, embarcados de sensores, capaz de monitorar atividades e trazer melhorias para área de saúde e outras área de interesse.

Portanto, o desenvolvimento de projetos de detecção de quedas inteligentes é interessante sob vários aspectos. Sendo capaz de gerar um monitoramento robusto, com alarmes acionados através de análises de sensores para buscar melhor segurança para as pessoas necessitadas. Dessa forma, a implantação de um sistema inteligente de detecção de quedas pode contribuir com uma menor preocupação das pessoas de interesse.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Propor um sistema de detecção de quedas através de sensores, que estabeleça a melhor relação precisão/desempenho num sistema embarcado de baixo custo.

1.1.2 Objetivos Específicos

Dentre os principais objetivos específicos destacam-se:

- Realizar uma revisão teórica do tema com as palavras chave "*accelerometer*","*embedded*", "*gyroscope*","*magnetometer*", "*detector*";
- Propor uma solução para a detecção de quedas de pessoas, utilizando preferencialmente bibliotecas *open-source*;
- Desenvolver um dispositivo portátil com as funcionalidades necessárias para o processamento da detecção e transmissão de dados;
- Implementar um aplicativo para realizar os alertas necessários das consequências geradas pelo sistemas;
- Realizar uma análise comparativa de quedas e falsas quedas;
- Avaliar o desempenho do detector quando aplicado com pessoas em ambiente controlado.

1.2 JUSTIFICATIVA

À medida que as tecnologias de predição e mobilidade são desenvolvidas são realizados estudos para verificar a viabilidade de utilizá-las para detecção de queda e *ADL (Activities of Daily Living)* .

Sabe-se que a rápida ação em determinados casos, pode evitar grandes lesões e até mesmo salvar vidas. Com a constatada fragilidade e as deficiências da saúde humana, o sistema de embarcados de quedas inteligentes abordado neste documento apresenta-se como uma possível solução ao problema.

2 REVISÃO BIBLIOGRÁFICA

2.1 ESTADO DA ARTE

A utilização de um detector de quedas como uma medida preventiva contra aumento da gravidade do estado de saúde de idosos ou pessoas sujeitas é utilizada hoje em dia. Os mecanismos que visam a utilização deste sistema são mais abrangentes para idosos com maiores de 65 anos, onde é o público mais afetado por quedas (WHO, 2017).

As soluções de detecção de queda existentes podem ser divididas em duas classes. A primeira classe analisa apenas a aceleração para detectar quedas. A segunda classe de soluções utiliza informações de aceleração e orientação corporal para detectar quedas.

Para a primeira classe, Shi aplicou o método utilizando três eixos do acelerômetro utilizando técnicas *threshold* para detecção de quedas e diferenciar as *ADL* com o movimento da queda. Shi usou como parâmetro o valor de pico da aceleração, o comprimeto da base dado em tempo, e a velocidade pós impacto (SHI XINGMING SUN; LIU, 2016).

Lim realizou um trabalho semelhante com a utilização apenas dos três eixos do acelerômetro porém utilizando técnicas do modelo de *Markov*. Lim utilizou o acelerômetro para calcular alguns parâmetros como o ângulo e o módulo da aceleração num primeiro momento, para então aplicar o algoritmo de *Markov* combinado com as técnicas do *threshold* para detecção de quedas (LIM CHULHO PARK; YU, 2014).

Na segunda classe de soluções de detecção de quedas, Noury desenvolveu um detector de queda composto por três sensores: um sensor de inclinação (*tilt*) para monitorar a orientação do corpo, um acelerômetro piezoelétrico para monitorar o choque de aceleração vertical e um sensor de vibração para monitorar os movimentos do corpo (NOURY et al., 2000).

Quiang em seu artigo, aplicou seu método de detecção de quedas utilizando acelerômetro e giroscópio. No método, leve-se uma consideração especial da postura da pessoa através do dado da taxa do ângulo dado pela leitura da aceleração da gravidade. O método é capaz de diminuir os falsos positivos gerado por motivo como

sentar rápido, pular, deitar sobre cama rapidamente (LI JOHN A. STANKOVIC, 2009).

Em contra partida, nos dispositivos comercialmente disponível, vê-se que uso de *Smart Watch* é uma das tecnologias que realiza algumas dessas funções no nível mais avançado.

Tem-se como exemplo, uma das tecnologias mais avançadas nesse requisito, o *Apple Watch Series* que, além da função de detecção de quedas, tem diversas outras funções (APPLE, 2019). Ainda no ano de 2019, esse dispositivo foi capaz de salvar a vida de um idoso da Noruega de 65 anos, que sofreu uma queda durante uma madrugada enquanto caminhava em direção ao banheiro. O dispositivo acionou o socorro e o norueguês foi encontrado sangrando, inconsciente e caído no chão do banheiro pela sua filha.

No caso em questão, o dispositivo pergunta se está tudo bem com o usuário após detectar sinais de queda. Caso não haja nenhuma resposta dentro de 1 minuto, o relógio realiza uma ligação para um serviço de emergência automaticamente, compartilhando a localização do evento (NRK, 2019).

Um caso semelhante aconteceu também com uma idosa de 80 anos, na Alemanha, na qual foi encontrada pelo filho dela e atendida pelo médicos sem nenhum ferimento grave (UOL, 2019). Por padrão, essa função é automaticamente habilitada para pessoas com maiores de 65 anos de idade.

2.2 ESP SERIES

2.2.1 Hardware

As placas de desenvolvimento com o chip da família ESP Series é desenvolvida pela empresa chinesa *Espressif*, localizada em Shanghai. Hoje a empresa conta com mais de 100 milhões de chips conectados com dispositivos IoT no mundo. Os principais valores que *Espressif* trabalha são: confiabilidade, economia, robustez, segurança e eficiência energética assim como informa o próprio site. Nessa seção será abordado dois modelo mais facilmente encontrado e usado no mercado, as especificações de ambos serão apresentadas, a fim de comparação entre os dois modelos.

2.2.1.1 *ESP8266*

O ESP8266 da Espressif oferece solução Wi-Fi altamente integrada para atender às demandas contínuas dos usuários por uso eficiente de energia, design compacto e desempenho confiável na indústria de IoT.

Possui vários modelos, entre os mais utilizados, está o ESP-01. O que a princípio chama muito atenção neste módulo são dois aspectos. O primeiro, é seu tamanho muito reduzido. E o segundo, é seu preço baixo comparado com outros microcontroladores. Vale destacar também a facilidade com que o mesmo pode ser integrado a demais soluções, bastando, por exemplo, o uso de uma comunicação serial UART.

Dentre as principais configurações do ESP8266, pode-se destacar:

- System-On-Chip com Wi-Fi embutido;
- Conectores GPIO, barramentos I2C, SPI, UART, entrada ADC, saída PWM e sensor interno de temperatura;
- CPU que opera em 80MHz;
- 32KBytes de RAM para instruções;
- 96KBytes de RAM para dados;
- 64KBytes de ROM para boot;

Atualmente, é possível encontrar vários modelos do ESP8266, que varia principalmente com o tamanho, podendo encontrar modelos extremamente pequenos.

2.2.1.2 *ESP32*

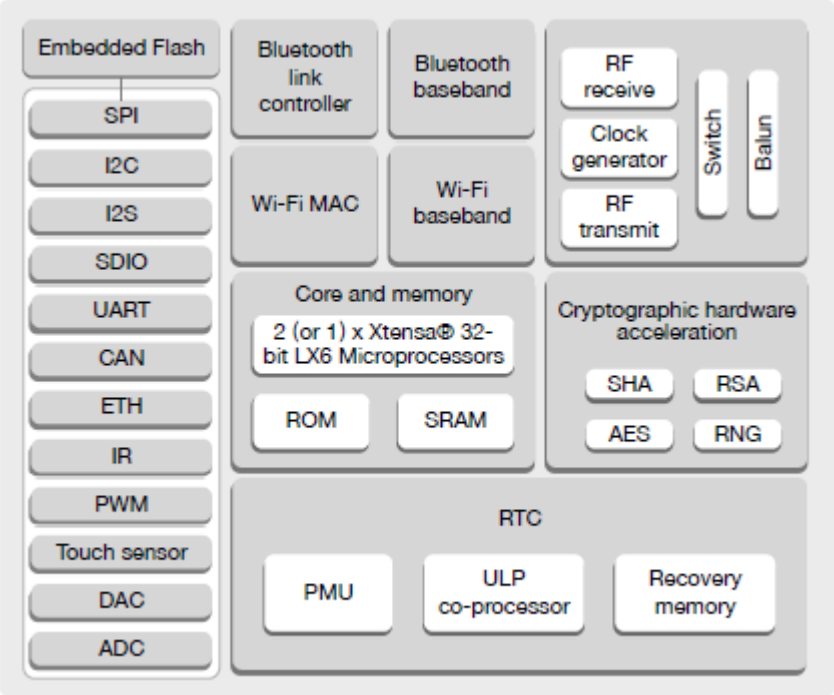
O ESP32 é um único chip combinado de Wi-Fi e Bluetooth de 2,4 GHz projetado com a tecnologia de ultra-baixa potência. Ele foi projetado para obter o melhor desempenho de potência e RF, mostrando robustez, versatilidade e confiabilidade em uma ampla variedade de aplicações e cenários de energia (SYSTEMS, 2019).

Basicamente, o ESP32 é o sucessor do ESP8266. Ele adiciona um núcleo de CPU extra, Wi-Fi mais rápido, mais GPIOs e suporta Bluetooth 4.2 e Bluetooth de baixa energia. Além disso, o ESP32 vem com pinos sensíveis ao toque que podem ser

usados para acordar o ESP32 do modo *deep sleep* um sensor de efeito hall embutido e um sensor de temperatura embutido. O diagrama de blocos funcionais pode ser observado na Figura 2.1.

Com o objetivo de ter uma solução completa para Wifi e Bletooth, o ESP32 é designado para celulares, eletrônicos *wearable* e aplicações IoT, com o foco em ter uma solução *Ultra-low-Power*.

FIGURA 2.1 – Diagrama de blocos funcionais do ESP32



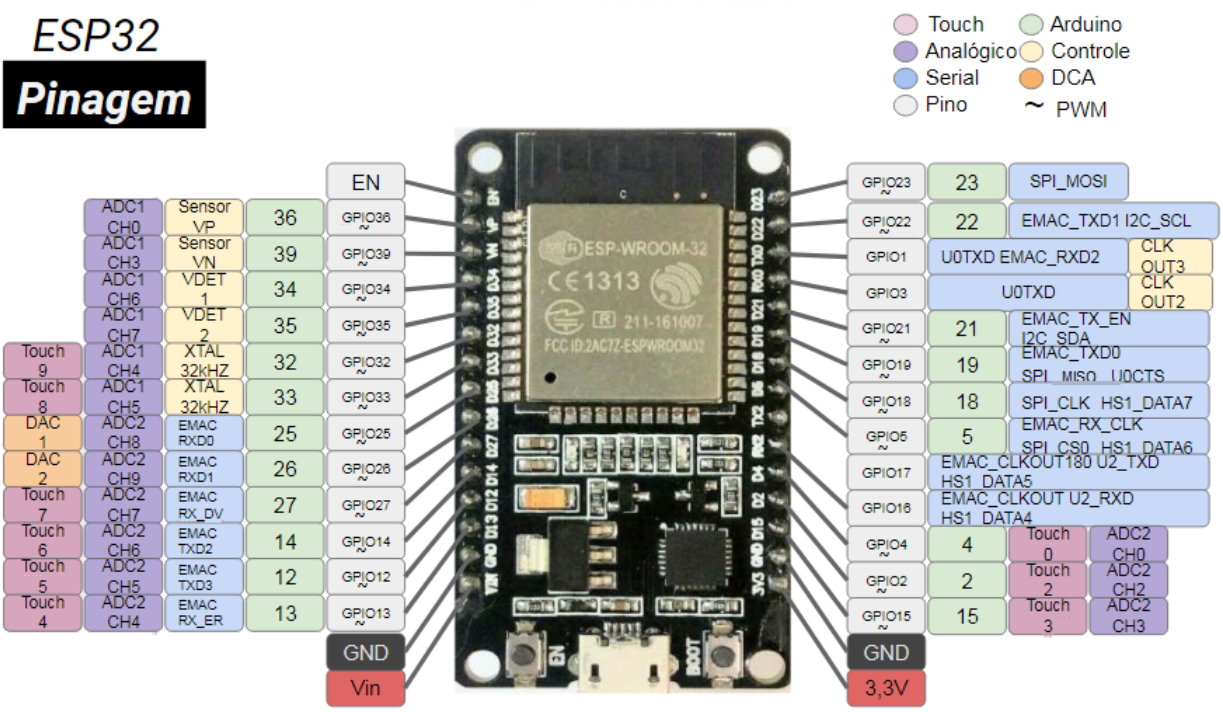
Fonte: (SYSTEMS, 2019)

2.2.2 Software

2.2.2.1 *Arduíno Software IDE*

O ambiente de desenvolvimento integrado (IDE) do Arduino é um aplicativo de plataforma cruzada (para Windows, macOS, Linux) escrito na linguagem de programação Java. É usado para escrever e fazer upload de programas em placas compatíveis com Arduino, mas também, com a ajuda de núcleos de terceiros, outras placas de desenvolvimento de fornecedores, como por exemplo, o ESP32.

FIGURA 2.2 – Pinagens detalhadas do ESP32



Fonte: (CURTOCIRCUITO, 2018)

A IDE do Arduino é conhecida por ser uma plataforma estruturada, bom boa curva de aprendizado e acessível para desenvolvedores, não apenas com os dispositivos do Arduíno, como também para outras placas de desenvolvimento.

2.3 SENSORES

Nessa sessão será apresentado o módulo que será usado no projeto e também, os sensores embutidos no mesmo.

2.3.1 MPU-9250

O MPU-9250 é um módulo multi-chip (MCM) fabricado pela empresa *Invensense* (INVENSENSE, 2016) constituído de basicamente 3 sensores diferente: acelerômetro, giroscópio e magnetômetro. Esse dispositivo combina na primeira casa 3 eixos do giroscópio e os 3 eixos do acelerômetro. Na segunda casa, o AK8963 como define o *datasheet*, há o magnetômetro também composto por 3 eixos. Portanto, o MPU-9250 é um dispositivo que rastreia movimento em 9 eixos que conta com o sensor dedicado I²C e *MotionFusion* integrado, que melhoram a performance de calibração do *firmware*,

reduzindo custo e complexidade de seleção para determinados níveis de sistemas.

Além disso, o MPU-9250 é composto por 3 conversores 16-bit analógico-digital (ADC) para cada um das funções (acelerômetro, giroscópio, magnetômetro) para digitalizar as saídas dos mesmos.

Para um preciso rastreamento dos rápidos e lentos movimentos, as características para programação do usuário permite uma configuração completa de escala do giroscópio num range de ±250, ±500, ±1000, e ±2000°/sec (dps), configuração completa de escala do acelerômetro num range de ±2g, ±4g, ±8g, e ±16g, e finalmente, configuração completa de escala do magnetômetro num range de ±4800µT.

A *Invensense* garante uma precisão do clock de 1% dentro do range de temperatura de -40ºC e 85ºC.

O módulo opera dentro do range de tensão de 2.4V a 3.6V e possui uma comunicação do I²C em 400kHz e comunicação SPI em 1MHz.

As principais aplicações dedicada do MPU-9250 são: controle remoto 3D para dispositivos conectados, sensores *wearable* para saúde e esportes, jogos portáteis e serviços baseados em localização de interesses. (INVENSENSE, 2016)

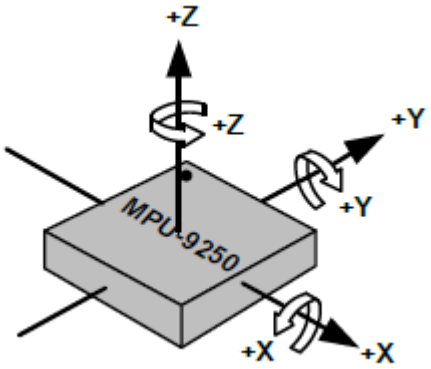
O MPU-9250 é o menor dispositivo MotionTracking 9 eixos do mundo, o que permite reduzir drasticamente o tamanho do chip e a potência de consumo, com o consumo de apenas 9.3µA e com o tamanho de 44% menor do que a primeira geração lançada pela *Invensense*. Ao mesmo tempo, isso aprimora a performance e o custo do projeto.

2.3.2 Giroscópio

O módulo MPU-9250 consiste de 3 vibradores independentes para o giroscópio que detecta a taxa de rotação sobre os eixos X-, Y- e Z-, como mostra a figura 2.3. Basicamente como explica o datasheet, quando o giroscópio é rotacionado em qualquer eixo, o *Coriolis Effect* causa uma vibração que detecta uma capacidade de decolagem. O resultado do sinal é amplificado, demodulado, e filtrado para produzir uma tensão proporcional o taxa angular. A tensão é digitalizada usando um ADC individual de 16 bit para cada eixo. A faixa de escala completa dos sensores giroscópicos pode ser programada digitalmente para ± 250, ± 500, ± 1000 ou ± 2000 graus por segundos (*dps*). A taxa de amostragem do ADC é programada de 8000 amostra por segundos para 3.9

amostra por segundos, e filtros passa-baixas selecionáveis pelo usuário permitem uma ampla variedade de frequências de corte.

FIGURA 2.3 – Orientação dos eixos do Acelerômetro e Giroscópio



Fonte: (INVENSENSE, 2016)

2.3.3 Acelerômetro

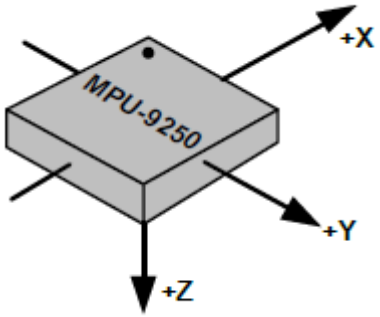
O acelerômetro de 3 eixos do MPU-9250 usa massas de prova separadas para cada eixo. A aceleração ao longo de um eixo específico induz deslocamento na massa de prova correspondente, e os sensores capacitivos detectam o deslocamento de maneira diferente. A arquitetura do MPU-9250 reduz a suscetibilidade dos acelerômetros às variações de fabricação, bem como à deriva térmica. Quando o dispositivo é colocado em uma superfície plana, ele mede 0g nos eixos X e Y e + 1g no eixo Z, assim conforme a figura 2.3. O fator de escala dos acelerômetros é calibrado na fábrica e é nominalmente independente da tensão de alimentação. Cada sensor possui um ADC sigma-delta dedicado para fornecer saídas digitais. A faixa de escala completa da saída digital pode ser ajustada para ± 2g, ± 4g, ± 8g ou ± 16g.

2.3.4 Magnetômetro

O magnetômetro de 3 eixos usa a tecnologia de sensor Hall altamente sensível. A parte do magnetômetro do circuito integrado incorpora sensores magnéticos para detectar magnetismo terrestre nos eixos X, Y e Z, um circuito de acionamento do sensor, uma cadeia de amplificadores de sinal e um circuito aritmético para processar o sinal de cada sensor. Cada ADC possui uma resolução de 16 bits e uma faixa de

escala total de ± 4800 T. Na figura 2.4 pode-se observar as orientação dos eixos da bússola que constituí o magnetômetro.

FIGURA 2.4 – Orientação dos eixos da sensibilidade da bússola



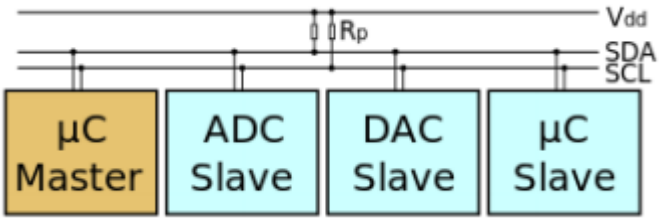
Fonte: (INVENSENSE, 2016)

2.3.5 Protocolo de Comunicação I2C

O protocolo de comunicação I2C *(Inter-Integrated Circuit)* descreve o funcionamento de um barramento de comunicação serial que utiliza apenas dois fios. É um protocolo muito utilizado para conectar periféricos de baixa velocidade a placas-mãe, microcontroladores e afins (CAMARA, 2013).

No protocolo I2C, encontra-se dois tipos de dispositivos: Master e Slave. Onde o Master é a unidade de controle responsável por coordenar todos os periféricos (Slaves). A linha SCL é responsável pelo clock do barramento, e a linha SDA pela transmissão de dados, como mostra a figura 2.5.

FIGURA 2.5 – Barramento I2C



Fonte: (CAMARA, 2013)

2.4 BLE

O Bluetooth Low Energy (LE) foi projetado para operação com energia muito baixa. Para permitir uma operação confiável na faixa de frequência de 2,4 GHz, onde utiliza uma abordagem robusta de espectro de salto de frequência que transmite dados em mais de 40 canais. O Bluetooth LE oferece aos desenvolvedores uma enorme flexibilidade, incluindo várias opções PHY que suportam taxas de dados de 125 Kb/s a 2 Mb/s, vários níveis de potência, de 1 mW a 100 mW, bem como várias opções de segurança (BLUETOOTH, 2019).

2.5 ANDROID

O Sistema Operacional móvel mais conhecido e mais usada no mundo, Android é utilizado não só nos smartfones, como também em relógios, carros e TV e outros dispositivos que pode implementar o sistema operacional. O SO é baseado em no núcleo Linux e desenvolvido pela empresa de tecnologia Google, é seguro, gerenciável e flexível uma vez que possui portabilidade entre dispositivos (GOOGLE, 2019).

2.5.1 Android Studio

Android Studio é a IDE oficial para desenvolvimento de aplicativos Android baseado em *IntelliJ*. Logo, toda a interface textual do software é baseado em Java. Ela fornece interface gráfica e textual para o usuário realizar o design gráfico do aplicativo. Android Studio é desenvolvido pela Google em colaboração com JetBrains (HOHENSEE, 2013).

A plataforma é completa e independente para desenvolvimento de aplicativo, sendo permitido desenvolver em linguagem de programação *Java* e *Kotlin*.

2.5.2 Linguagem Java

Java é uma linguagem de programação orientada a objetos, amplamente utilizada, que foi desenvolvida na década de 90 por uma equipe de programadores da empresa *Sun Microsystems*. Entretanto, em 2008 o Java foi adquirido pela empresa *Oracle Corporation* onde pertence até os dias de hoje (CAELUM, 2014).

Java conta com uma demanda mundial expressiva, e está entre as 3 linguagem mais utilizada no mundo no ano de 2019, segundo o site do *Stack Overflow*, devido a sua popularidade e aplicabilidade. Para o caso em questão, será utilizada essa linguagem para desenvolvimento de aplicativo Android para esse projeto.

3 DESENVOLVIMENTO

3.1 METODOLOGIA

3.1.1 Seleção do Hardware

O ESP8266 é mais barato que o ESP32. Embora não tenha tantas funcionalidades, ele funciona bem para a maioria dos projetos simples de IoT. No entanto, ele tem algumas limitações quando se trata do mapeamento GPIO e pode não ter pinos suficientes para determinados objetivos. O ESP32 é muito mais poderoso que o ESP8266, pois contém mais GPIOs com várias funções Wi-Fi mais rápido e também suporta *Bluetooth*. Também conta com a compatibilidade e todas as facilidades de programar na IDE do Arduíno.

FIGURA 3.1 – Comparação do ESP32 e ESP8266

Especificações	ESP8266	ESP32
MCU	xtensa® single 32-bit l106	xtensa® dual-core 32-bit lx6 600 DMIPS
802.11 b/g/n Wi-Fi	HT20	HT40
Bluetooth	Não	Bluetooth 4.2 Le
Frequência	80 MHz	160 MHz
SRAM	160 Kbytes	512 Kbytes
Flash	SPI Flash, 16 Mbytes	SPI Flash, 16 Mbytes
GPIO	17	36
Hardware/ Software PWM	Não/ 8 canais	1 / 16 canais
SPI/ I2C/I2S/UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	Não	1
Interface Ethernet Mac	Não	1
Sensor Capacitico	Não	Sim
Sensor de Temperatura	Não	Sim
Temperatura de Trabalho	-40°C a 125°C	-40°C a 125°C

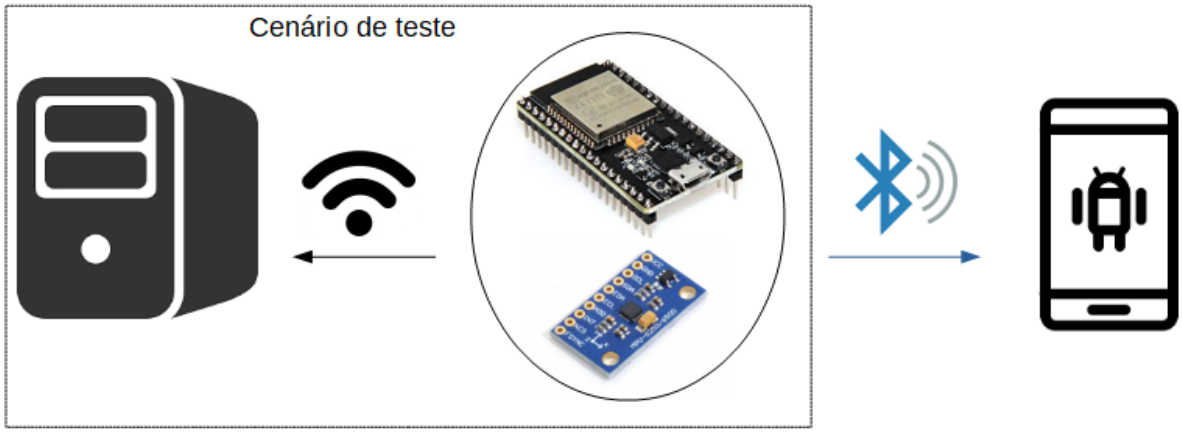
Fonte: (CURTOCIRCUITO, 2018)

A figura 3.1 mostra um quadro comparativo entre o ESP32 e o ESP8266. Visto que será necessário um poder de processamento maior, e também a presença de *bluetooth* para realizar a comunicação com o aplicativo, foi selecionado o ESP32 para esse projeto.

3.2 ESQUEMÁTICO GERAL

Afim de torna mais claro a visão geral do projeto, na figura 3.2 mostra a esquemático geral do projeto. Pode-se observar que dentro do cenário geral, existe um cenário de teste que é a interação do dispositivo com o servidor via *Wi-Fi*. E um outro cenário que o dispositivo conecta com o *smartphone* via *BLE*.

FIGURA 3.2 – Esquemático geral do projeto



Fonte: Autor

O cenário de teste foi criado para facilitar a validação do resultado e ajudar na análise de configuração de detecção. No caso, o servidor atua no armazenamento das leituras realizadas pelo dispositivos, na qual receberá esses dados através do *Wi-Fi*. O armazenamento é realizado em arquivos *CSV*, e será abordado com mais detalhes na seção 3.3. Entretanto, esse cenário não será utilizado após validado os testes, tornando o sistema independente de uma rede *Wi-Fi* para funcionamento, uma vez que o dispositivo final usará apenas comunicação *bluetooth*.

Uma vez que o teste foi realizado e validado, o dispositivo atuará apenas com o *bluetooth*, na qual irá se interagir com o *Android* do *smartphone* para acionar o alarme programado.

3.3 SERVIDOR PYTHON

Foi necessário criar um servidor para se comunicar com o ESP32 afim de monitorar o comportamento da leitura dos sensores e poder avaliar o resultado realizando a plotagem de gráficos dos eventos específicos pré determinado.

Para tal, foi criado um servidor em na linguagem de programa *Python* na qual aplica-se o conceito de socket que combina o endereço de IP e a porta de comunicação. A porta de comunicação, por padrão, foi utilizada a 8090.

Python tem uma biblioteca chamada *socket* que foi utilizada nesse servidor. Para facilitar a comunicação da rede, usou-se a função *bind* que realiza a ligação com o cliente/servidor.

Ainda no servidor *Python*, usou-se a biblioteca *csv* para armazenar os dados recebido do servidor do ESP32. O arquivo separador através vírgula permite que seja mais facilmente interpretado por programas posteriores na análise gráfica.

Para o teste em questão, o servidor *Python* é utilizado com cliente, pois recebe os dados, e o ESP32 é o servidor que fornece os dados. O código utilizado está disponível no apêndice A.

Uma vez adquiridos os dados, optou-se por analisar os dados através de interfaces gráficas da bibliotecas *open source matlplotlib*, disponível na *webpage* matplotlib.org.

3.4 LEITURA DOS SENSORES

Conforme explicado na seção 2.3.1, o módulo MPU-9250 utilizado nesse projeto, realiza a leitura dos sensores em 9 eixos (3 eixos do giroscópio, 3 eixos do acelerômetro e o magnetômetro também composto por 3 eixos), que são os 9 dados disponível para processamento do desenvolvedor em sua aplicação, podendo ser expandida em outros dados de interesse.

Nesse projeto, as leituras dos sensores será feita na frequência de 15 Hz. Essa frequência foi adotada porque também está presente nas leituras dos sensores das referências do estado da arte. Portanto, com a leitura de 15 amostram em 1 segundos, no intervalo de 6 segundos, tem-se um total de 90 amostras de cada eixo. Sabe-se que os 3 sensores fazem leituras em 3 eixos, logo tem-se 135 medidas por segundo, e 810 medidas dentro de um intervalo de tempo de 6 segundos (intervalo estipulado para

análise da queda).

3.4.1 Biblioteca utilizada

Encotra-se disponível uma biblioteca open source , feita pela japonesa Asuki Kono, na *webpage* do *GitHub* da autora, que realiza a leitura e a calibração dos sensores atravês dos pinos *SDA* e *SCL* do sensor MPU-9250.

Será utilizada essa biblioteca afim de otimizar a leitura e os processos do projeto com relação ao sensor MPU-9250, uma vez que a mesma fornece a calibração dos sensores do acelerômetro e gisroscópio.

3.5 ANÁLISE DE DETECÇÃO

Esse projeto utilizou algumas ferramentas matemáticas para a melhor representação do resultado. Para o cálculo do módulo das componentes X, Y e Z dos sensores utilizados, foi necessário aplicar os conceitos de análise vetorial. Por exemplo, para o cálculo do módulo da aceleração, foi utilizada a equação 3.1:

$$|a| = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{3.1}$$

onde a_x, a_y e a_z são as componentes da aceleração do eixo X, Y e Z respectivamente; e $|a|$ é o módulo da aceleração. E para o cálculo do módulo do giroscópio, foi utilizada a equação 3.2:

$$|g| = \sqrt{g_x^2 + g_y^2 + g_z^2} \tag{3.2}$$

onde g_x, g_y e g_z são as componentes do giroscópio do eixo X, Y e Z respectivamente; e $|g|$ é o módulo da giroscópio. Os três eixos tanto do acelerômetro, quanto do giroscópio, julgam-se de interesse para a análise de detecção de quedas.

3.5.1 Técnicas de detecção por *threshold*

O objetivo deste estudo foi adaptar metodologia para estimar o ponto a partir do qual pode-se utilizar como parâmetro para detecção de quedas. Dentre os métodos sensoriais analíticos discriminativos podem ser citados os testes de sensibilidade ou *threshold*, os quais são definidos como sendo o limite da capacidade sensorial (CIVILLE B. THOMAS CARR, 1999).

A tecnologia de *threshold* é estudada a partir do valor de pico da magnitude dos vetores obtidos do acelerômetro e giroscópio. A magnitude desses valores é feita pela equações 3.1 e 3.2.

3.5.2 Variação do magnetômetro

O uso do magnetômetro será utilizado de maneira diferente aos demais sensores devido ao seu funcionamento na aplicabilidade para detecção de quedas. O magnetômetro, como mencionado em 2.3.1, utiliza o magnetismo terrestre para o cálculo das suas componentes. Dessa forma, as variações da componentes é resultando do deslocamento do eixo respectivo X, Y e Z.

Para a análise de detecção de quedas nesse projeto, será utilizado apenas a componente do eixo Z, onde representa a variação magnética da altura do usuário. Dessa forma, o magnetômetro irá ajudar a evitar falsos alarmes quando ocorrer a validação do *threshold* na mesma altura. Sabe-se que para ocorrer uma queda é necessário ter variação no eixo Z do magnetômetro , ou seja, variação da altura. Dessa forma, a diferença entre o valor máximo e valor mínimo do eixo Z do magnetômetro será um dos parâmetros para detecção de quedas.

3.6 MÉTODOS PARA AVALIAÇÃO DE ALGORITMOS DE DETECÇÃO DE QUEDA

Observou-se nas literaturas que cada autor possui seu método para detecção de quedas. Dessa forma, afim de avaliar melhor cada diferentes tipos de métodos, Noury et al. (2007) apresentou cenários e uma forma generalizada de avaliação dos algoritmos de detecção de quedas desenvolvidos. Segundo Noury et al. (2007) existem 4 casos possíveis:

- Verdadeiro positivo (VP): a queda ocorre e o dispositivo a detectou;
- Falso positivo (FP): o dispositivo acusa uma queda, mas ela, de fato, não ocorreu;
- Verdadeiro negativo (VN): uma atividade diária ocorre, e o dispositivo, de fato, não detecta uma queda;
- Falso negativo (FN): uma queda ocorre, mas o dispositivo não a detecta.

Dessa forma, para avaliar essas 4 situações, 3 critérios são propostos: sensibilidade, especificidade e exatidão. Sensibilidade é a capacidade do sistema de detectar a ocorrência de uma queda. Especificidade é a capacidade de detectar a ausência das características de uma queda quando de fato não houve uma queda. E, por fim, exatidão é a proporção de resultados verdadeiramente positivos e negativos dentro de todo o espaço amostral. O cálculo é apresentado pelas equações 3.3, 3.4 e 3.5, respectivamente (NOURY, 2007):

$$Sensibilidade = \frac{VP}{VP + FN} \tag{3.3}$$

$$Especificidade = \frac{VN}{VN + FP} \tag{3.4}$$

$$Exatidão = \frac{VP + VN}{N} \tag{3.5}$$

onde N é número de amostras.

Para realizar levantamento de dados e calcular os critérios citados, foi proposto 13 ADL para gerar dados amostrais e poder avaliar o método. As ADL proposta, baseado em (NOURY, 2007), são:

- Queda para trás;
- Queda para frente finalizando de joelhos;
- Queda para frente finalizando deitado;
- Andar poucos metros;
- Correr poucos metros;
- Se abaixar rápido e pegar algo no chão;
- Subir escada andando;
- Descer escada andando;
- Subir escada correndo;
- Descer escada correndo;

- Pular;
- Sentar rápido;
- Levantar rápido;

Foi realizado 20 testes de cada ADL, totalizando um espaço amostral de 260 testes realizado por voluntários do sexo masculino e feminino.

3.7 ALIMENTAÇÃO E DURAÇÃO DA BATERIA

O dispositivo tem um consumo médio de 50mAh e a tensão de entrada deve ser maior do que 5V porém menor do que 10V. A placa tem um regulador de tensão para 3.3V para qualquer tensão de entrada dentro do range de tensão citado.

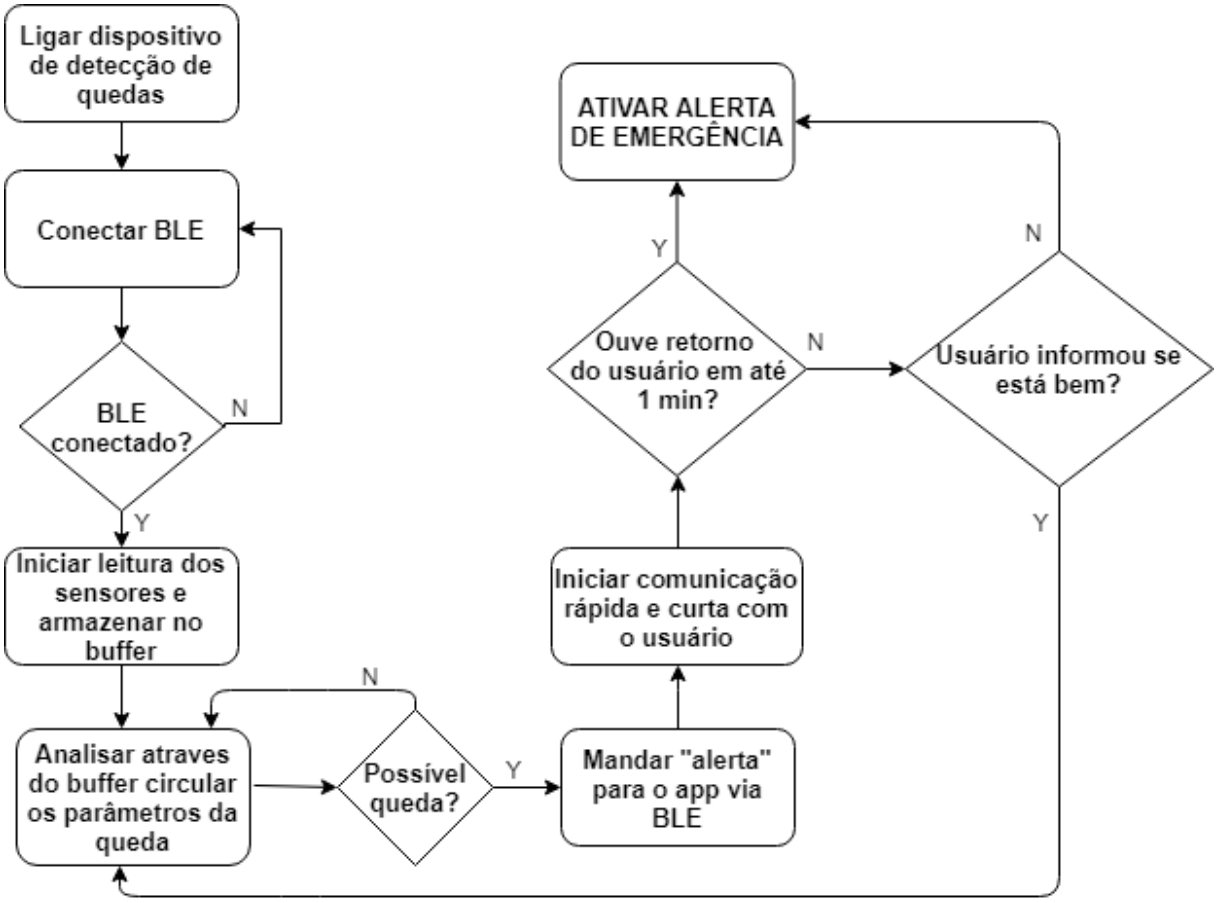
Dessa forma, foi escolhido a bateria de 9V do modelo 6F22 facilmente encontrado no mercado nos dias atuais. A carga da bateria pode variar de acordo com o fabricante e com a especificação. É possível também encontrar bateria do mesmo modelos recarregáveis no mercado sem grandes dificuldades. Para estimativa de duração, uso-se uma bateria com carga de 450mA. Para o consumo citado, estima-se que a duração é de 9h de uso contínuo.

3.8 FLUXOGRAMA GERAL DO PROJETO

Na figura 3.3 pode-se observar o fluxograma do funcionamento geral do projeto, assim como a interação do dispositivo com o aplicativo. Inicialmente é necessário fazer o pareamento e a conexão do aplicativo com o ESP32. Após concretizado, a leitura e a analise da mesma é iniciada para perceber uma possível queda.

Assim que detectado uma possível queda, o ESP32 envia um sinal de alerta para o app através do *bluetooth* que então realiza uma interface com o usuário com a pergunta sobre o estado do usuário. Em caso se não houver resposta do usuário, ou o usuário informar que precisa de ajuda, um SMS é enviado para um número de celular programado. Em um caso do usuário informar que está tudo bem, o sistemas volta a analisar as leituras.

FIGURA 3.3 – Fluxograma geral do projeto



Fonte: Autor

4 RESULTADOS

4.1 PRIMEIROS RESULTADOS

Os primeiros resultados atingidos foram as primeiras modelagem das ADL, ou seja, algumas simulações de atividades do dia a dia. Umas das principais preocupações do projeto está no fato de não confundir o evento de quedas, com outros eventos, como pular, correr, subir escada, descer escada, caminhar e outros.

Portanto, tendo em vista essa primeira preocupação, foi realizado a leitura dos dados dos sensores enquanto essas atividades eram realizadas. As imagens abaixo pode-se observar os dados obtidos dentro do cenário de teste, onde foi plotado dentro de um código escrito em *python* através da biblioteca *matplotlib*.

Os valores impressos nos gráficos do acelerômetro e giroscópio são os valores de picos dos módulos registrados. E os valores impressos nos gráficos do magnetômetro são os valores de máximo e mínimo registrados do eixo Z.

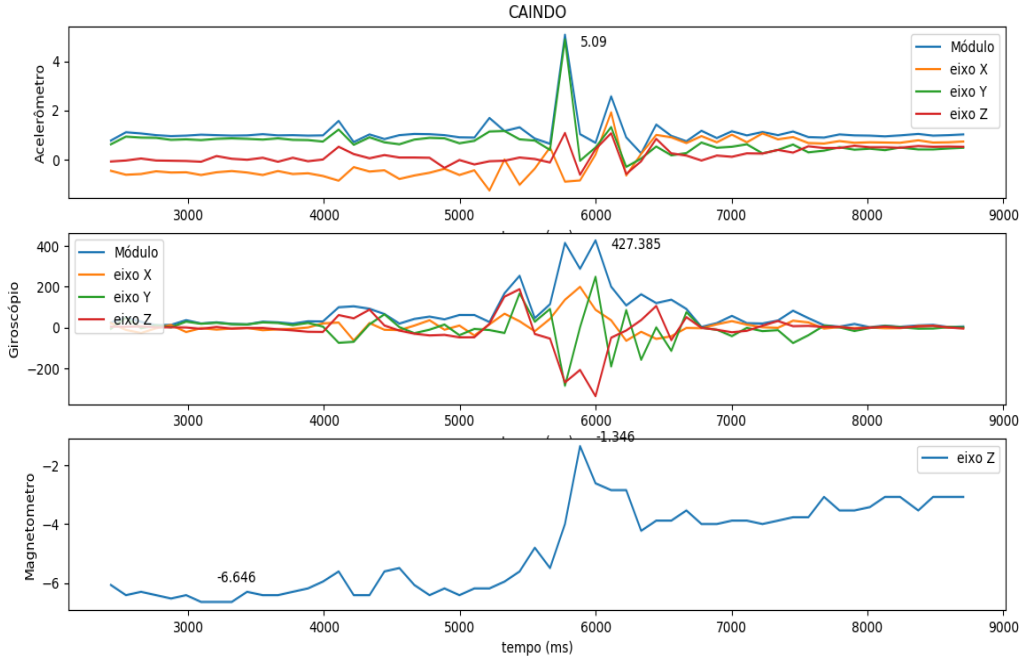


FIGURA 4.1 – Dados do evento 'CAINDO'

Fonte: Autor

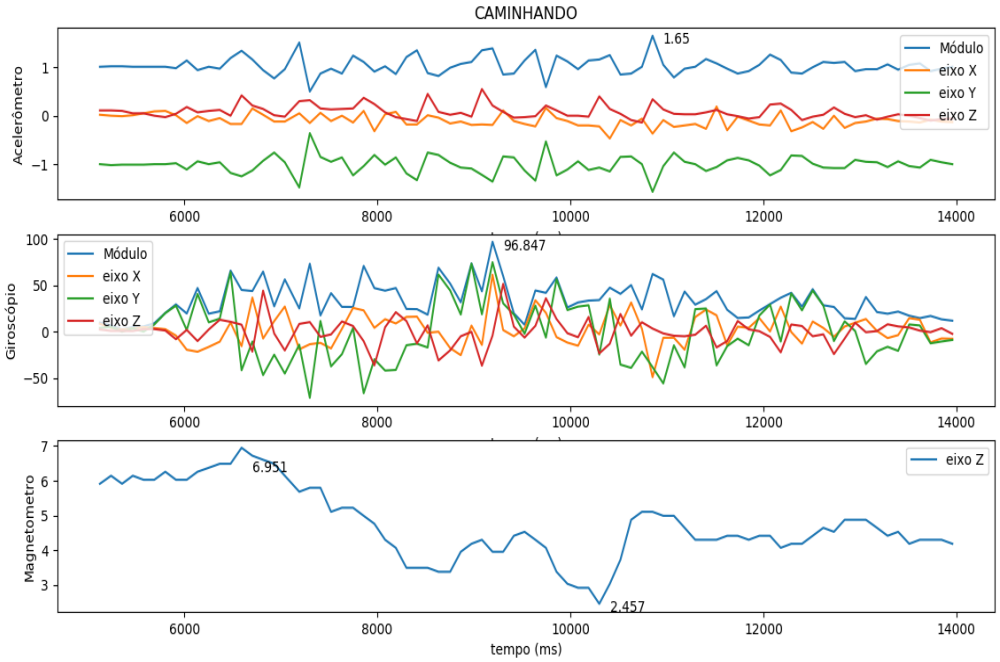


FIGURA 4.2 – Dados do evento ’CAMINHANDO’

Fonte: Autor

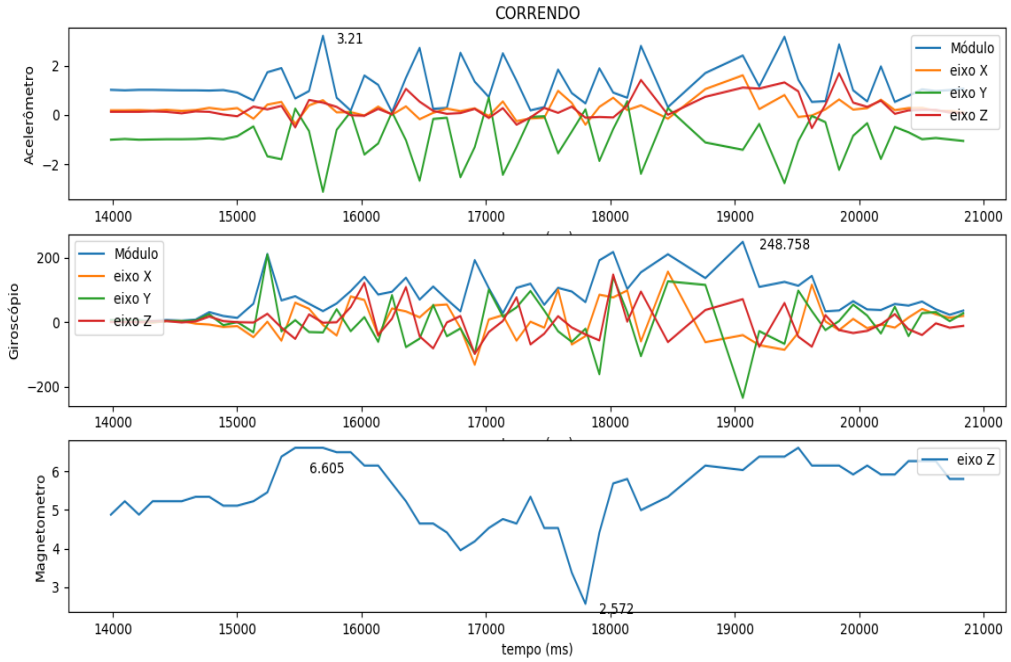


FIGURA 4.3 – Dados do evento ’CORRENDO’

Fonte: Autor

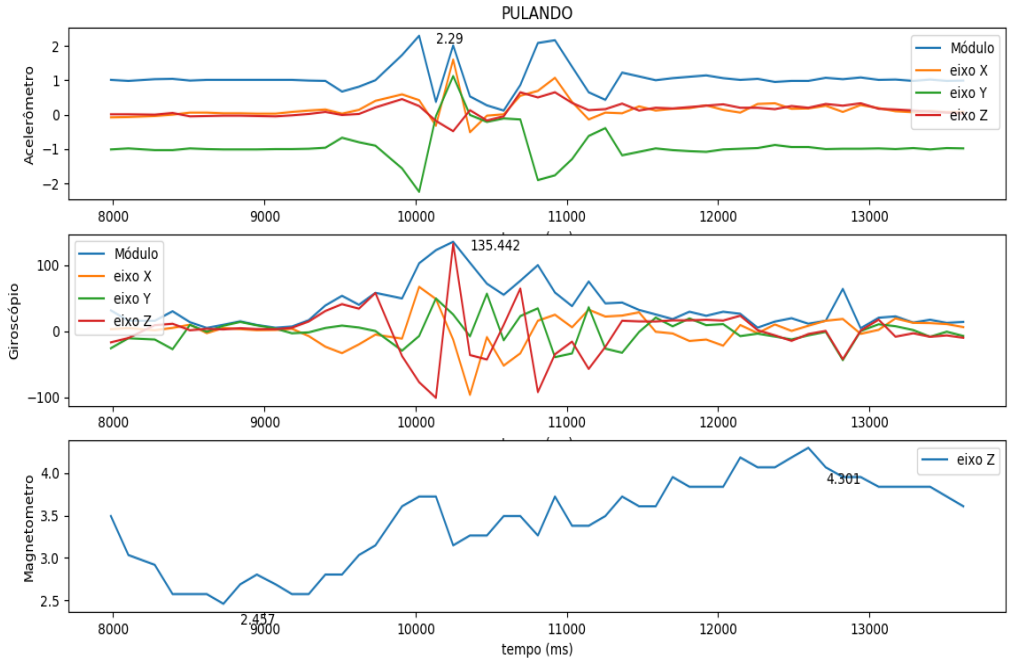


FIGURA 4.4 – Dados do evento ’PULANDO’

Fonte: Autor

Evidentemente que o foco do projeto é detectar quedas. Dessa forma foi feito um estudo e testes preliminares afim de caracterizar os diferentes tipos de ADL e encontrar um padrõa para os eventos de quedas. Observa-se que, dentro dos eventos estudados, o acelerômetro registrou pico de leitura próximo ou maiores que 2 vezes a aceleração da gravidade o evento: ’caindo’, ’correndo’, ’descendo a escada’ e ’pulando’. O destaque maior fica é registrado no evento ’correndo’ e ’caindo’ na qual o valor de pico chegou a 3 vezes a aceleração da gravidade.

Para a análise do giroscópio, observa-se que apenas 3 eventos, dos 6 testados, geraram valores maiores do que 100º de taxa angula momentânea. Esses eventos são: ’caindo’, ’correndo’ e ’pulando’. Entretanto, apresentam comportamento característicos plausível para diferenciar. O evento ’pulando’ chegou ao valor de pico próximo a 120º apenas o início do momento, enquanto o ’corrrendo’, chegou a 200º tendo em vista que esse tem maior duração. E finalmente, o evento ’caindo’, na qual é o foco do projeto, atingiu o maior valor comparado com todos os outros eventos analisados, próximo a 400º.

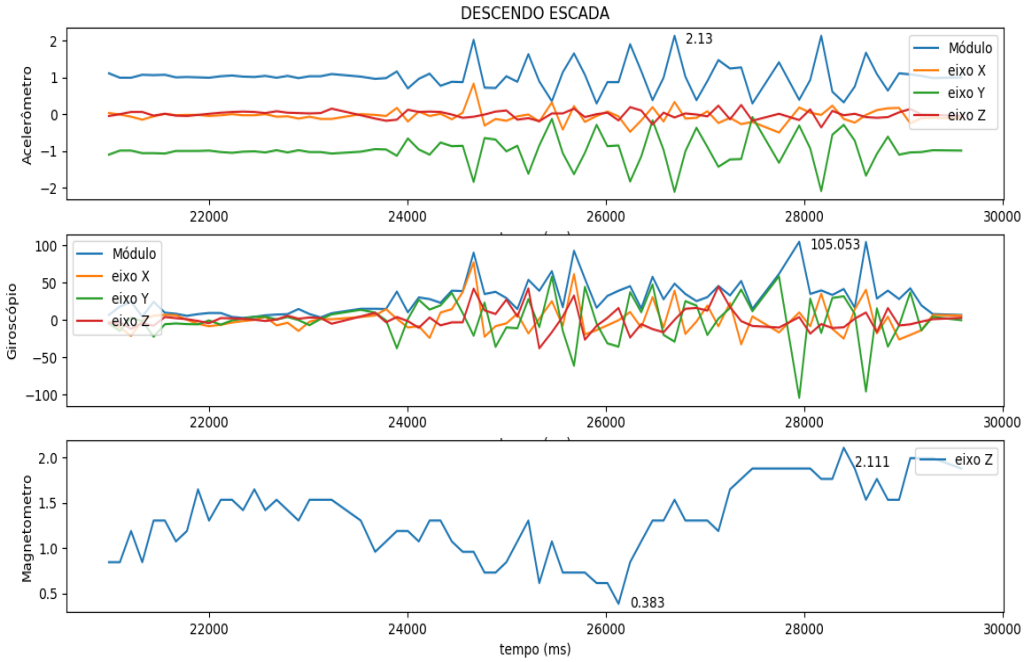


FIGURA 4.5 – Dados do evento ’DESCENDO A ESCADA’

Fonte: Autor

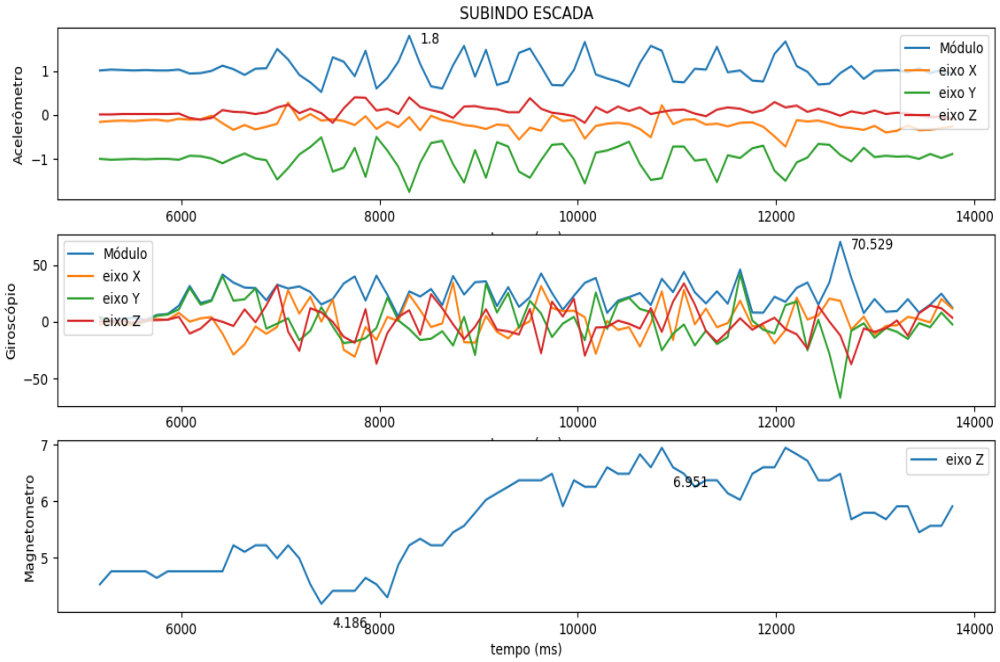


FIGURA 4.6 – Dados do evento ’SUBINDO A ESCADA’

Fonte: Autor

4.2 PARÂMETROS DE DETECÇÃO

Para fazer a análise de detecção proposta na sessão 3.5, foi necessário pré determinar alguns parâmetros para aplicar o método a ser implementado em um algoritmo. São basicamente três parâmetros definido pelo autor baseado nas análises dos primeiros testes realizados conforme mostrado na sessão 4.1.

Os parâmetros utilizado para esse projeto são:

- $|a| > 3m/s^2$;
- $|g| > 300$;
- $\Delta m_z > 5T$;

Onde Δm_z é a componente do eixo Z do magnetômetro, $|a|$ e $|g|$ o módulo dos 3 eixos do acelerômetro e do giroscópio, respectivamente.

4.3 O APLICATIVO ANDROID - *SMART FALL*

A plataforma de desenvolvimento Android Studio permite que se desenvolva aplicativo em duas linguagem de programação: *kotlin* e *Java*. Para o projeto em questão, o aplicativo foi desenvolvido em *Java*. Nessa sessão será mostrado como foi desenvolvido e os resultados obtidos.

O aplicativo foi densenvolvida em cima da versão do *Android* 4.0, que tem o apelido de *Android Ice Cream Sandwich*. Essa versão foi escolhida justamente para ter mais compatibilidade com todos as versões do SO *Android*. Segundo a plataforma de desenvolvimento utilizada, essa versão garate 100% de compatibilidade com todos os dispostivios *Android*, uma vez que não será necessário utilizar os recursos mais novos que só estão disponíveis nas últimas versões.

Basicamente, o aplicativo tem a sua funcionalidade resumida em 2 *Activities* principais: *MainActivity* e *Alerta*. Além dessas, existem mais duas *Activities* secundárias que atua no *bluetooth*, porém não há *layout* que faça uma interface com o usuário. Para facilitar a nomenclatura e codificação, foi dado um nome para o aplicativo: *Smart Fall*.

FIGURA 4.7 – Exemplo de como é feito o acesso a permissões

Fonte: Autor

4.3.1 Permissões

Assim como grande parte dos aplicativos utilizados hoje em dia, o app *Smart Fall* faz-se necessário o uso das permissões para que tenha o funcionamento completo. No caso, será necessário o de 3 permissões: acesso ao *bluetooth*, acesso a localização e autorização para enviar SMS. Via de regra e por questões de segurança, nenhum aplicativo pode acessar informações do *smatphone* sem as devidas autorizações do usuário.

Essas permissões são declaradas no arquivo *AndroidManifest.xml* dentro da *tag <uses-permission />*, e são indicadas a seguir:

```
"android.permission.BLUETOOTH"
"android.permission.BLUETOOTH_ADMIN"
"android.permission.ACCESS_FINE_LOCATION"
"android.permission.ACCESS_COARSE_LOCATION"
"android.permission.SEND_SMS"
```

As duas primeiras dão acesso ao gerenciamento do *bluetooth*, as duas seguintes dão acesso a localização (latitude e longitude) e por fim, a ultima permite que o app envie SMS. A figura 4.7 mostra um exemplo de como essas permissões são concedidas para o *back-end* do aplicativo. Pode-se observar nos códigos que ocorre um checagem constante nas permissões para não ocorrer falhas de funcionamentos.

4.3.2 Funcionamento

Ao inicializar o aplicativo, o Android solicitas as permissões conforme mostrado em 4.3.1. Na primeira parte da tela, tem um *TextView*, que é um componente do Android na qual apresenta texto no *layout*, que tem como função orientar o usuário do que deve ser feito para configurar.

Tem-se um botão único chamado "ESCANEAR/CONECTAR"que realiza várias funções automatizadas para o usuário, justamente para facilitar o uso, uma vez que, o usuário não precisa saber o que está acontecendo por trás da tela, ou saber as sequências de ações realizadas que não fazem diferenças,no sentido técnico, para o usuário.

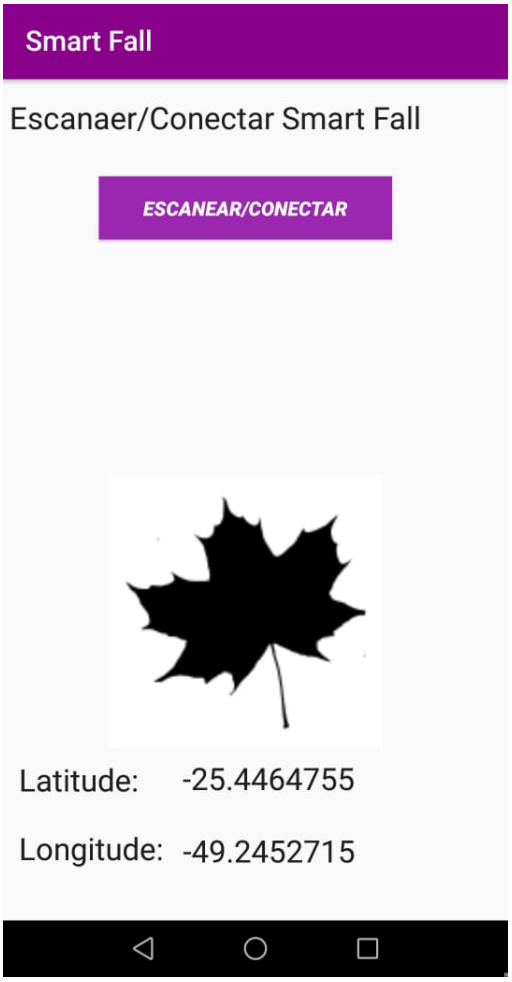
A figura 4.8 mostra a tela inicial na qual orienta o usuário a pressionar o botão para procurar e conectar com o dispositivo. Assim que as informação de localização são concedidas, já é possível acessar as informações de latitude/longitude do aplicativo. Essas informações são mostradas na parte inferior da tela, conforme pode observar na figura 4.8.

Após o usuário pressionar o botão, o aplicativo emitirá uma mensagem "Procurando dispositivos..."e "Parear com o Smart Fall". O dispositivo foi renomeado de *Smart Fall* para ser encontramo por esse nome na *ListView*. Essa etapa, o *MainActivity* chama a função *onClickDiscover()* que começa a buscar todos os dispositivos disponíveis na área de cobertura, ou seja, o aplicativo busca por dispositivos não pareados, como pode observar na figura 4.9.

Assim que o app faz a busca de todos os dispositivos disponível, é necessário fazer o pareamento. A lista *(ListView)* de todos os dispositivos disponível aparece logo abaixo do botão. O pareamento é feito por sucessivos métodos *BroadcastReceiver()* desde a busca até o pareamento propriamente dito. Sabe-se que, apenas o pareamento é necessário, porém não suficiente para realizar comunicação.

Dessa forma, é necessário fazer a conexão para efetivamente realizar a comunicação entre o ESP32 e o aplicativo. Essa comunicação é feita via *socket bluetooth*, através da biblioteca nativa do Android *BluetoothSocket*. O *socket* do *bluetooth* faz uma combinação entre o UUID e o nome do dispositivo. O UUID do inglês - *universally unique identifier*, é o Identificador Único Universal formado de caracteres de 128 bits

FIGURA 4.8 – Tela inicial



Fonte: Autor

usado para identificar informações em sistemas de computação. O UUID do aplicativo é dado a seguir:

00001101−0000−1000−8000−00805F9B34FB

Essa configuração é feita automaticamente pelo *Activity BluetoothConnectionService* através do comando *mSocket.connect()*, onde o *mSocket* é o UUID. Assim, a aplicativo é capaz de realizar a conexão através do método *startConnection()* invocado no *MainActivity* e replicado no *Activity BluetoothConnectionService*.

O processo de pareamento e conexão demora alguns segundos, e é observado como mostra na tela 4.10. E após finalizar a conexão, o sistema está pronto para uso, conforme pode ser observado na figura 4.11.

O ESP32 fará a leitura e o processamento dos dados conforme mostra o código

FIGURA 4.9 – Instruções para conectar

Fonte: Autor

no Apêndice B. Dessa forma, quando o ESP32 detectar uma queda, ele enviará um sinal para invocar o alarme no aplicativo.

Portanto, assim que o aplicativo receber o sinal oriundo do ESP32, ele chama uma nova *activity* do alarme. Essa *activity* pode ser observada na figura 4.12 e no código no Apêndice E.

O alarme consiste em dizer que uma possível queda foi detectada e tentar realizar uma comunicação rápida e direta com o usuário perguntando se o mesmo está bem ou se ele precisa de ajuda. Na figura 4.12 observa-se depois botões que são as respostas da pergunta "Você está bem?".

Dentro desse cenário, tem-se 3 opções: a primeira é o usuário responder que está bem, a segunda é o mesmo responder que não está bem e a ultima é não responder. No primeiro cenário, o aplicativo apenas volta a monitorar normalmente conforme

FIGURA 4.10 – Pareando com o ESP32

Fonte: Autor

anteriormente. No caso do segundo cenário, um SMS é enviado automaticamente para o número estrategicamente pré-programado com a localização onde ocorreu a queda. Um exemplo de como é enviado esse SMS pode ser observado na figura 4.13. E por fim, no ultimo caso de não houver nenhuma resposta, um SMS é enviado automaticamente em 1 minuto para o número programado.

O SMS contém um link com a latitude e longitude na plataforma do Google Maps. Dessa forma, basta abrir o link que o sistema operacional irá redirecionar para o aplicativo do Google Maps informando a localização no mapa.

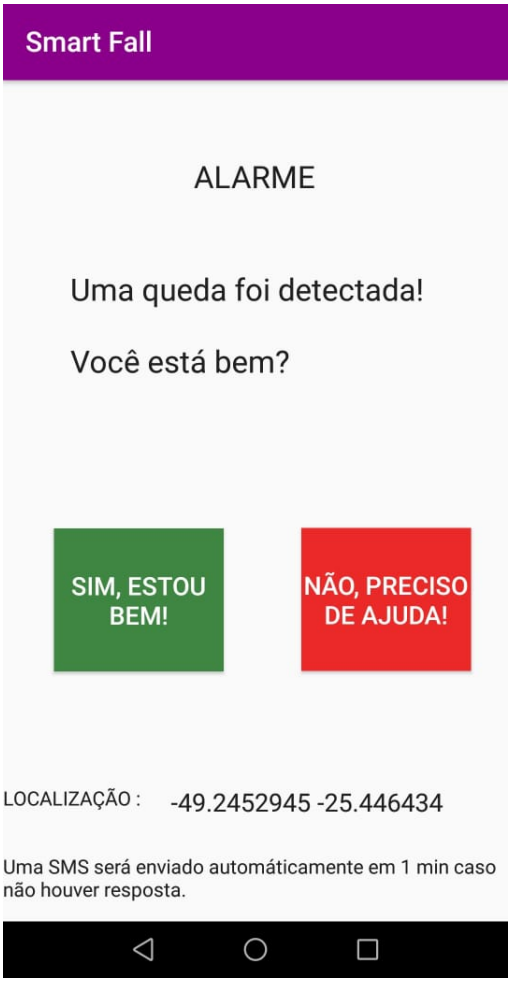
FIGURA 4.11 – Conectado e pronto para uso

Fonte: Autor

FIGURA 4.13 – SMS

Fonte: Autor

FIGURA 4.12 – Alarme



Fonte: Autor

4.4 RESULTADOS DAS SIMULAÇÕES

Conforme citado na sessão 3.6, tem-se um campo amostral de 260 amostras de testes, que foram simulados os 13 eventos repetindo em 20 vezes, que serviram para aplicar o método de avaliação proposto do projeto. As simulações foram divididas e realizadas por voluntário do sexo masculino e feminino. O resultado dos testes é representado nas Tabelas 1 e 2.

TABELA 1 – RESULTADO DAS SIMULAÇÕES DOS ADL - parte 1

ADL – SIMULAÇÕES	**VP**	**FN**	**FP**	**VN**
Queda para frente finalizando deitado	20	0	0	0
Queda para frente finalizando de joelhos	16	4	0	0
Queda para trás	13	7	0	0
Andar poucos metros	0	0	0	20
Correr poucos metros	0	0	0	20
Se abaixar rápido e pegar algo no chão	0	0	0	20
Subir escada andando	0	0	0	20
Descer escada andando	0	0	0	20
Subir escada correndo	0	0	13	7
Descer escada correndo	0	0	3	17
Pular	0	0	12	8
Sentar rápido	0	0	0	20
Levantar rápido	0	0	0	20
TOTAL	**49**	**11**	**28**	**172**

TABELA 2 – RESULTADO DAS SIMULAÇÕES DOS ADL - parte 2

ADL – TESTE	**Sensibilidade**	**Especificidade**	**Exatidão**
Queda para frente finalizando deitado	100%	0%	100%
Queda para frente finalizando de joelhos	80%	0%	80%
Queda para trás	65%	0%	65%
Andar poucos metros	0%	100%	100%
Correr poucos metros	0%	100%	100%
Se abaixar rápido e pegar algo no chão	0%	100%	100%
Subir escada andando	0%	100%	100%
Descer escada andando	0%	100%	100%
Subir escada correndo	0%	35%	35%
Descer escada correndo	0%	85%	85%
Pular	0%	40%	40%
Sentar rápido	0%	100%	100%
Levantar rápido	0%	100%	100%
TOTAL	**82%**	**86%**	**85%**

Aplicando as fórmulas 3.3, 3.4 e 3.5 da seção 3.6 para todo o campo amostral, tem-se:

$$Sensibilidade = \frac{VP}{VP + FN} = \frac{49}{49 + 11} = 82\% \tag{4.1}$$

$$Especificidade = \frac{VN}{VN + FP} = \frac{172}{172 + 28} = 86\% \tag{4.2}$$

$$Exatidão = \frac{VP + VN}{N} = \frac{49 + 172}{260} = 85\% \tag{4.3}$$

Podemos observar que o método adotado tem uma exatidão de 85%. Entretanto, percebe-se que o dispositivo pode confundir com outros movimentos, como o movimento de subir e descer a escada correndo, e o movimento de pular.

Teve-se um êxito de 100% para quedas para frente que finalizam deitados, na qual teve-se os parâmetros baseados, e algumas falhas para quedas para trás e as que finalizam de joelhos.

A definição do parâmetro teve foco as análises feitas nas quedas para frente que finalizam deitado, caso comum que ocorre na quedas com términos inconscientes. Entretanto, é notável que a análise de detecção pode ser aprimorada e torna-se mais sofisticado com análises mais complexas, uma vez que, apenas se analisa o valor de pico para aceleração e giroscópio. Acredita-se que o projeto apresenta muita variáveis que permitem um aprofundamento na área de análise de detecção.

Se considerarmos ADL aplicadas em apenas idosos, provavelmente aumentaria a exatidão das simulações, onde as ADL que resultou em falsos alarmes não são frequêntemente praticadas por pessoas idosas. As outras ADL mais comuns obteve-se êxito satisfatórios mesmo com a técnica de detecção não muito aprofundada.

5 CONCLUSÃO E TRABALHOS FUTUROS

5.1 CONCLUSÃO

A área de saúde é uma área muito delicada e importante na nossa sociedade, onde um dos meios de monitoramentos passivo mais utilizado é por meio de sensoriamentos. Sabe-se que a rápida ação em determinados casos, pode evitar grandes lesões e até mesmo salvar vidas, como por exemplo acionamentos de alarmes para detecção de quedas em usuários específicos. Os dados divulgados pela Organização Mundial de Saúde deixa claro que as quedas e as consequentes lesões resultantes constituem um problema de saúde pública e de grande impacto social, principalmente em pessoas idosas.

O objetivo desse projeto é propor um sistema de detecção de quedas através de sensores, que estabeleça a melhor relação precisão/desempenho num sistema embarcado de baixo custo.

Esse projeto contou com o desenvolvimento de um sistema completo, que inclui o desenvolvimento do dispositivo envolvendo o microcontrolador, sensores e a programação dos mesmos, até o desenvolvimento do aplicativo *Android* que faz a comunicação via *bluetooth* com o dispositivo. Pode-se concluir que a integração obteve êxito em sua comunicação que contemplou o funcionamento do projeto.

Pode-se observar também que as quedas apresentam comportamento característico capaz de se diferenciar de outros eventos. Observa-se que esse evento foi o único que apresentou no gráfico do giroscópio intensidade próximas a 400º de taxa angular momentânea para quedas para frente comparado com os primeiros testes feitos.

Os sensores se mostraram ser recursos satisfatório para detecção das quedas, entretanto, a análise dos dados, no que se refere a interpretação de quedas ou não, pode ser melhorada em um trabalho futuro, uma vez que o processamento só faz a análise do valores de picos e a variação dos valores do magnetômetro.

E por fim, o projeto teve início em agosto de 2019 na qual contou aplicações de várias áreas de conhecimento, mostrando-se uma abordagem multidisciplinar, e com poucos meses de trabalho e um elemento na equipe, foi possível atender os requisitos

do escopo previsto no plano de trabalho que contemplam a disciplina de Trabalho de Conclusão de Curso.

5.2 TRABALHOS FUTUROS

No que se refere aos trabalhos futuros, pode-se mencionar primeiramente um investimento mais investigativo na análise de detecção de quedas utilizando os recursos já apresentados nesse projeto. Essas análises pode ser abordados conceitos mais profundos de simulações matemáticas onde se aplicam algoritmos de análises inteligentes que leva em consideração o comportamento inteiro da queda, e não apenas os valores de pico. Como o exemplo citado em 2.1, Lim aplicou o algoritmo de *Markov* nos três eixos do acelerômetro (LIM CHULHO PARK; YU, 2014). Esse projeto tem a vantagem que possuir mais informações sobre a queda além do acelerômetro. Acredita-se que esse trabalho pode melhorar a exatidão então obtida nesse projeto.

O dispositivo pode ter a sua usabilidade melhorada tentando otimizar principalmente o seu tamanho. É possível, por exemplo, fazer o uso de impressoras 3D de forma que otimize o espaço vazio interno, modelar uma nova placa de tamanho otimizado com os chips embutido e consequentemente otimizar o consumo de energia para o uso de uma bateria também mais otimizada.

E por fim, os sensores permite que esse projeto se abrange a outras aplicações além de detecção de quedas. É possível que se encontre aplicações em situações específicas que fazem monitoramento que podem, de certa forma, trazer melhorias na área de saúde.

REFERÊNCIAS

APPLE. *Apple Watch Series 4*. 2019. <https://www.apple.com/br/apple-watch-series-4/>. Acesso em 09 set. 2019. Citado na página 14.

BLUETOOTH. *Bluetooth Low Energy (LE)*. 2019. <https://www.bluetooth.com/bluetooth-technology/radio-versions/>. Acesso em 03 set. 2019. Citado na página 21.

CAELUM. *Java e Orientação a Objetos*. 2014. <https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/. Acesso em 04 set. 2019. Citado na página 21.

CAMARA, R. *PROTOCOLO I2C*. 2013. <http://www.univasf.edu.br/ romulo.camara/novo/wp-content/uploads/2013/11/Barramento-e-Protocolo-I2C.pdf>. Acesso em 11 set. 2019. Citado na página 20.

CIVILLE B. THOMAS CARR, M. C. G. V. Sensory evaluation techniques. *CRC Press*, 1999. Citado na página 26.

CURTOCIRCUITO. *Conhecendo o ESP32*. 2018. <https://www.curtocircuito.com.br/blog/conhecendo-esp32/>. Acesso em 03 setembro. 2019. Citado 2 vezes nas páginas 17 e 23.

GOOGLE. *Android*. 2019. <https://www.android.com/. Acesso em 04 set. 2019. Citado na página 21.

HOHENSEE, B. *Getting Started with Android Studio*. [S.l.]: ISBN, 2013. Citado na página 21.

INVENSENSE. *MPU-9250 Product Specification*. 2016. <https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/>. Acesso em 01 set. 2019. Citado 4 vezes nas páginas 17, 18, 19 e 20.

LI JOHN A. STANKOVIC, M. H. A. B. J. L. G. Z. Q. Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information. *IEEE*, 2009. Citado na página 14.

LIM CHULHO PARK, N. H. K. S.-H. K. D.; YU, Y. S. Fall-detection algorithm using 3-axis acceleration: Combination with simple threshold and hidden markov model. *Hindawi*, 2014. Citado 2 vezes nas páginas 13 e 46.

NOURY, A. F. N. Fall detection – principles and methods. *IEEE*, Conference of the IEEE EMBS, n. FrA05.6, 2007. Citado na página 28.

NOURY, N. et al. Monitoring behavior in home using a smart fall sensor and position sensors. In: . [S.l.: s.n.], 2000. p. 607 – 610. ISBN 0-7803-6603-4. Citado na página 13.

NRK. *Smartklokken reddet Toralv*. 2019. <https://www.nrk.no/norge/smartklokken-reddet-toralv-lordag-natt-1.14412266>. Acesso em 03 set. 2019. Citado na página 14.

SHI XINGMING SUN, Z. X. L. C. T.; LIU, J. Fall detection algorithm based on triaxial accelerometer and magnetometer. *Engineering Letters*, 2016. Citado na página 13.

SYSTEMS, E. *Datasheet ESP32, Version 3.1*. 2019. <https://www.espressif.com/sites/default/files/documentation/esp32$_datasheet_en.pdf$ > $.Acessoem13agosto.2019.Citado2vezesnasp\acute{a}ginas15e$ 16.

UOL. *RPi Apple Watch Series 4 detecta queda de idosa de 80 anos na Alemanha*. 2019. <https://macmagazine.uol.com.br/post/2019/04/16/apple-watch-series-4-detecta-queda-de-idosa-de-80-anos-na-alemanha/>. Acesso em 03 set. 2019. Citado na página 14.

WHO. Who global report on falls prevention in older age. 2017. Citado 2 vezes nas páginas 11 e 13.

APÊNDICE A – SERVIDOR PYTHON

```
import socket
import csv

s = socket.socket()

s.bind(('0.0.0.0', 8090 ))
s.listen(0)

while True:
    client, addr = s.accept()

    while True:
        content = client.recv(80)

        if len(content) ==0:
           break

        else:
            with open("esp32.txt","a") as f:
                f.write(content.decode("utf-8")+"\n")
            print(content)
```

APÊNDICE B – CÓDIGO DE LEITURA

```
#include <MPU9250_asukiaaa.h>
#include <math.h>
MPU9250_asukiaaa mySensor;
#define SDA_PIN 21
#define SCL_PIN 22
#include "BluetoothSerial.h"
#define CALIB_SEC 10

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it
#endif

BluetoothSerial SerialBT;

float max_value_A;
float max_value_G;
float gyro_module;
int count, alerta;

uint8_t sensorId;
float mDirection, mX, mY, mZ;
float max_mZ, min_mZ;
float deltaZ;


void setup()
{

  Serial.begin(115200);
```

```
  SerialBT.begin("Smart Fall"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with bluetooth!");

  //setup MPU9250
  Wire.begin(SDA_PIN, SCL_PIN); //sda, scl
  mySensor.setWire(&Wire);
  while (mySensor.readId(&sensorId) != 0) {
    Serial.println("Cannot find device to read sensorId");
    delay(2000);
  }

  mySensor.beginAccel();
  mySensor.beginMag();
  mySensor.beginGyro();

  // offset for mag values
  float magZ, magZMin, magZMax;

  Serial.println("Start scanning values of magnetometer to get offset values.");
  Serial.println("Rotate your device for " + String(CALIB_SEC) + " seconds.");
  setMagMinMaxAndSetOffset(&mySensor, CALIB_SEC);
  Serial.println("Finished setting offset values.");

  max_mZ= NULL;
  min_mZ= NULL;

  max_value_A = 0;
  max_value_G = 0;
  count = 0;
  alerta = 1;

  pinMode(2, OUTPUT);
}
```

```
void setMagMinMaxAndSetOffset(MPU9250_asukiaaa* sensor, int seconds) {
  unsigned long calibStartAt = millis();
  float magX, magXMin, magXMax, magY, magYMin, magYMax, magZ, magZMin, magZMax;

  sensor->magUpdate();
  magXMin = magXMax = sensor->magX();
  magYMin = magYMax = sensor->magY();
  magZMin = magZMax = sensor->magZ();

  while(millis() - calibStartAt < (unsigned long) seconds * 1000) {
    delay(100);
    sensor->magUpdate();
    magX = sensor->magX();
    magY = sensor->magY();
    magZ = sensor->magZ();
    if (magX > magXMax) magXMax = magX;
    if (magY > magYMax) magYMax = magY;
    if (magZ > magZMax) magZMax = magZ;
    if (magX < magXMin) magXMin = magX;
    if (magY < magYMin) magYMin = magY;
    if (magZ < magZMin) magZMin = magZ;
  }

  sensor->magXOffset = - (magXMax - magXMin) / 2;
  sensor->magYOffset = - (magYMax - magYMin) / 2;
  sensor->magZOffset = - (magZMax - magZMin) / 2;

  Serial.println("mySensor.magXOffset = " + String(mySensor.magXOffset) + ";");
  Serial.println("mySensor.maxYOffset = " + String(mySensor.magYOffset) + ";");
  Serial.println("mySensor.magZOffset = " + String(mySensor.magZOffset) + ";");

}
```

```
void loop()
{

    mySensor.accelUpdate();
    mySensor.gyroUpdate();
    mySensor.magUpdate();


    mDirection = mySensor.magHorizDirection();


    gyro_module = pow( pow(mySensor.gyroX(),2) + pow(mySensor.gyroY(),2) +
    pow(mySensor.gyroZ(),2) , 0.5);


    if (float(mySensor.magZ())<min_mZ ||  max_mZ == NULL){
      min_mZ = float(mySensor.magZ());
      }
    if (float(mySensor.magZ())>max_mZ ||  max_mZ == NULL){
      max_mZ = float(mySensor.magZ());
      }
    deltaZ = max_mZ-min_mZ;

    if (float(mySensor.accelSqrt())>max_value_A){
      max_value_A = float(mySensor.accelSqrt());
    }
    if (float(gyro_module)>max_value_G){
      max_value_G = gyro_module;
    }

    if (max_value_A > 3 && max_value_G > 300 && deltaZ > 10){
```

```
      digitalWrite(2, HIGH);
      if (alerta == 1) {
        SerialBT.write('1'); //manda sinal para o App
        alerta = 0 ;
        count = 0 ;
      }
    }

    count++;

    if (count >= 90){
      count = 0;
      alerta = 1;
      max_value_A = 0;
      max_value_G = 0;
      digitalWrite(2, LOW);
    }

    Serial.println("DeltaZ: "+String(deltaZ));
    Serial.println("Alerta: "+String(alerta));
    Serial.println("Count: "+String(count));

    delay(66.6);
}
```

APÊNDICE C – MAIN ACTIVTIY - ANDROID

```
package com.example.bluetooth;

import android.Manifest;
import android.annotation.SuppressLint;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.pm.PackageManager;
import android.content.Intent;
import android.content.IntentFilter;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Build;
import android.support.annotation.RequiresApi;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
```

```
import java.lang.reflect.Method;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.Set;
import java.util.UUID;

public class MainActivity extends AppCompatActivity implements
AdapterView.OnItemClickListener {

    private static final String TAG = "MainActivity";
    int checkScanner ;

    Button btOnOff, btDiscoverable, btSend, btStartConnection;
    EditText etSend;
    BluetoothConnectionService mBluetoothConnection;
    private static final
    UUID UUID_INSECURE = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
    BluetoothDevice mBTdevice;
    BluetoothAdapter bluetoothAdapter;
    public ArrayList<BluetoothDevice> devicesBT=new ArrayList<BluetoothDevice>();
    public DeviceListAdapter deviceListAdapter;
    ListView lvNewDevides;
    TextView incomingMessage;
    StringBuilder messages;
    TextView txtLatitude, txtLongitude;
    TextView txtStatus;


    int alerta = 0;

    @RequiresApi(api = Build.VERSION_CODES.M)
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    lvNewDevides = findViewById(R.id.listViewNewDevices);
    devicesBT = new ArrayList<>();
    txtLatitude = (TextView) findViewById(R.id.txtLatitude);
    txtLongitude = (TextView) findViewById(R.id.txtLongitude);
    txtStatus = (TextView) findViewById(R.id.txtStatus);


    IntentFilter filter = new
    IntentFilter(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
    registerReceiver(broadcastReceiver4, filter);

    bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    lvNewDevides.setOnItemClickListener(MainActivity.this);
    messages = new StringBuilder();

    LocalBroadcastManager.getInstance(this).registerReceiver(mReceiver,
    new IntentFilter("incomingMessage"));

    txtStatus.setText("Verificando as permissões!");
    ligarBT();
    permissaoSMS();
    pedirPermissoes();
    checkBTPermissions();
}

private final BroadcastReceiver broadcastReceiver1 = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
```

```
            final String action = intent.getAction();

            if(action.equals(bluetoothAdapter.ACTION_STATE_CHANGED)){
                final int state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,
                bluetoothAdapter.ERROR);

                switch (state){
                    case BluetoothAdapter.STATE_OFF:
                        Log.d(TAG, "onReceive: STATE_OFF");
                        break;
                    case BluetoothAdapter.STATE_TURNING_OFF:
                        Log.d(TAG,"broadcastReceiver1: STATE_TURNING_OFF");
                        break;
                    case BluetoothAdapter.STATE_ON:
                        Log.d(TAG,"broadcastReceiver1: STATE_ON");
                        break;
                    case BluetoothAdapter.STATE_TURNING_ON:
                        Log.d(TAG, "broadcastReceiver1: STATE_TURNING_ON");
                        break;
                }
            }
        }
    };

    private final BroadcastReceiver broadcastReceiver2 = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();

            if(action.equals(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED)){
                final int mode = intent.getIntExtra(BluetoothAdapter.EXTRA_SCAN_MOD

                switch (mode){
```

```
                case BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE:
                    Log.d(TAG, "broadcastReceiver2: Discoverability Enabled");
                    break;
                case BluetoothAdapter.SCAN_MODE_CONNECTABLE:
                    Log.d(TAG,"broadcastReceiver2: Discoverability Disabled.
                    Able to receive connections");
                    break;
                case BluetoothAdapter.SCAN_MODE_NONE:
                    Log.d(TAG, "broadcastReceiver2: Discoverability Disabled.
                    Not able to receive connections");
                    break;
                case BluetoothAdapter.STATE_CONNECTING:
                    Log.d(TAG,"broadcastReceiver2: Connecting...");
                    break;
                case BluetoothAdapter.STATE_CONNECTED:
                    Log.d(TAG, "broadcastReceiver2: Connected");
                    break;
            }
        }
    }
};

private BroadcastReceiver broadcastReceiver3 = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        Log.d(TAG, "onReceive: ACTION FOUND");

        if(action.equals(BluetoothDevice.ACTION_FOUND)){
            BluetoothDevice device = intent.
            getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            devicesBT.add(device);
            Log.d(TAG, "onReceive: " + device.getName() + ": " +
```

```
                device.getAddress());
                deviceListAdapter = new DeviceListAdapter(context,
                R.layout.device_adapter_view, devicesBT);
                lvNewDevides.setAdapter(deviceListAdapter);
            }
        }
    };

    private final BroadcastReceiver broadcastReceiver4 = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();

            if(action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)){
                BluetoothDevice device =
                intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

                if(device.getBondState() == BluetoothDevice.BOND_BONDED){
                    Log.d(TAG, "broadcastReceiver4: BOND_BONDED");
                    mBTdevice = device;
                    txtStatus.setText("Smart Fall pareado!");
                    startConnection();
                    txtStatus.setText("Smart Fall Conectado e pronto para uso!");
                    checkScanner = 1;
                }
                if(device.getBondState() == BluetoothDevice.BOND_BONDING){
                    Log.d(TAG, "broadcastReceiver4: BOND_BONDING");
                    txtStatus.setText("Pareando com o Smart Fall...");
                }
                if(device.getBondState() == BluetoothDevice.BOND_NONE){
                    Log.d(TAG, "broadcastReceiver4: BOND_NONE");
                    txtStatus.setText("Procurando dispositivos...
                    \nParear com o Smart Fall");
```

```
                checkScanner = 0;
            }
        }
    }
};

@Override
protected void onDestroy() {
    Log.d(TAG,"onDestroy: called");
    super.onDestroy();
    unregisterReceiver(broadcastReceiver1);
    unregisterReceiver(broadcastReceiver2);
    unregisterReceiver(broadcastReceiver3);
    unregisterReceiver(broadcastReceiver4);
}


BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

        quedaDetectada();
        String text = intent.getStringExtra("theMessage");
        messages.append(text); //+ "\n");
        incomingMessage.setText(messages);
    }
};

public void quedaDetectada(){
    Log.d(TAG, "quedaDetectada: Initializing");

    Intent intent = new Intent(this, Alerta.class);
    startActivity(intent);
```

```
}

public void startBTConnection(BluetoothDevice device, UUID uuid){
    Log.d(TAG, "startConnection: Initializing RFCOM bluetooth connection");

    mBluetoothConnection.startClient(device, uuid);
    txtStatus.setText("Smart Fall Conectado! \nPronto para uso!");
}



public void startConnection(){
    startBTConnection(mBTdevice, UUID_INSECURE);
}

public void onClickSend(View view){
    byte[] bytes = etSend.getText().toString().
    getBytes(Charset.defaultCharset());
    mBluetoothConnection.write(bytes);
    etSend.setText("");
}

public void ligarBT() {
    Log.d(TAG, "onClickOnOff: enabling/disabling bluetooth");
    if(bluetoothAdapter == null){
        Log.d(TAG, "onClickOnOff: Does not have BT capabilities");
        Toast.makeText(getApplicationContext(),"Bluetooth is not supported",
        Toast.LENGTH_SHORT).show();
    }
    if(!bluetoothAdapter.isEnabled()){
        Log.d(TAG, "onClickOnOff: enabling BT");
        txtStatus.append("Ligar o bluetooth");
```

```
            Intent enableBTIntent = new
            Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivity(enableBTIntent);

            IntentFilter BTintent = new
            IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
            registerReceiver(broadcastReceiver1, BTintent);
        }

    }


    private void permissaoSMS(){
        Log.d(TAG, "permissaoSMS: para o SMS");

        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.SEND_SMS)
        != PackageManager.PERMISSION_GRANTED) {
            Log.d(TAG, "permissaoSMS: permissão SMS não concedida");
            Toast.makeText(MainActivity.this, "Sem permissão para o SMS ",
            Toast.LENGTH_SHORT).show();
            ActivityCompat.requestPermissions(this, new
            String[]{Manifest.permission.SEND_SMS}, 1);
        }
        else{
            Log.d(TAG, "permissaoSMS: OK !");
        }
    }

    // para a Localização
    private void pedirPermissoes() {
        Log.d(TAG, "pedirPermissoes: para o GPS");

        if (ActivityCompat.checkSelfPermission(this,
```

```
        Manifest.permission.ACCESS_FINE_LOCATION) !=
        PackageManager.PERMISSION_GRANTED
                && ActivityCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_COARSE_LOCATION) !=
                PackageManager.PERMISSION_GRANTED) {

            ActivityCompat.requestPermissions(this, new
            String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
            ActivityCompat.requestPermissions(this, new
            String[]{Manifest.permission.ACCESS_COARSE_LOCATION}, 1);
        }

        else
            txtStatus.setText("Permissões concedidas\n");
            configurarServico();
            ligarBT();
            if (checkScanner == 0){
                txtStatus.setText("Escanaer/Conectar Smart Fall");
            }
    }

    // para a Localização - invoca automático se o usuário não der permissão
    @Override
    public void onRequestPermissionsResult(int requestCode, String permissions[],
    int[] grantResults) {
        Log.d(TAG, "onRequestPermissionsResult: para o GPS");
        switch (requestCode) {
            case 1: {
                if (grantResults.length > 0 && grantResults[0] ==
                PackageManager.PERMISSION_GRANTED) {
                    configurarServico();
                } else {
                    Toast.makeText(this, "Não vai funcionar!!!",
```

```
                Toast.LENGTH_LONG).show();
            }
            return;
        }
    }
}

// Método que lê o serviço de GPS do Android
public void configurarServico(){
    Log.d(TAG, "configurarServico: para o GPS");
    try {
        LocationManager locationManager = (LocationManager)
        getSystemService(Context.LOCATION_SERVICE);

        LocationListener locationListener = new LocationListener() {
            public void onLocationChanged(Location location) {
                atualizar(location);
            }

            public void onStatusChanged(String provider, int status, Bundle
            extras) { }

            public void onProviderEnabled(String provider) { }

            public void onProviderDisabled(String provider) { }

        };

        //locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
        0, 0, locationListener);
        locationManager.requestLocationUpdates
        (LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
    }catch(SecurityException ex){
```

```
            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_LONG).show();
            Log.d(TAG, "configurarServico: erro");
        }

    }

    // Joga as informações da LAT/LONG no TextView
    public void atualizar(Location location) {
        Log.d(TAG, "atualizar: para o GPS");

        Double latPoint = location.getLatitude();
        Double lngPoint = location.getLongitude();

        txtLatitude.setText(latPoint.toString());
        txtLongitude.setText(lngPoint.toString());
    }

    @RequiresApi(api = Build.VERSION_CODES.M)
    public void onClickDiscover(View view){
        Log.d(TAG, "btDiscover: Trying unpaired all devices");
        unpairDevice();

        Log.d(TAG, "btDiscover: Looking for unpaired devices");

        if(bluetoothAdapter.isDiscovering()){
            bluetoothAdapter.cancelDiscovery();
            Log.d(TAG, "btDiscover: Canceling discovery");

            checkBTPermissions();

            bluetoothAdapter.startDiscovery();
            IntentFilter discoverDevicesIntent = new
            IntentFilter(BluetoothDevice.ACTION_FOUND);
```

```
            registerReceiver(broadcastReceiver3, discoverDevicesIntent);
        }
        if(!bluetoothAdapter.isDiscovering()){
            checkBTPermissions();

            bluetoothAdapter.startDiscovery();
            IntentFilter discoverDevicesIntent = new
            IntentFilter(BluetoothDevice.ACTION_FOUND);
            registerReceiver(broadcastReceiver3, discoverDevicesIntent);
        }
    }

    @RequiresApi(api = Build.VERSION_CODES.M)
    private void checkBTPermissions() {
        if(Build.VERSION.SDK_INT > Build.VERSION_CODES.LOLLIPOP){
            int permissionCheck =
            this.checkSelfPermission("Manifest.permission.ACCESS_FINE_LOCATION");
            permissionCheck +=
            this.checkSelfPermission("Manifest.permission.ACCESS_COARSE_LOCATION");
            if(permissionCheck != 0){
                this.requestPermissions(new
                String[]{Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.ACCESS_COARSE_LOCATION}, 1001);
            }else{
                Log.d(TAG, "checkBTPermissions: No need to check permissions. SDK
                version < LOLLIPOP");
            }
        }
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
    long id) {
```

```
        bluetoothAdapter.cancelDiscovery();
        Log.d(TAG,"onItemClick: Clicked on a device");

        String deviceName = devicesBT.get(position).getName();
        String deviceAddress = devicesBT.get(position).getAddress();
        if(Build.VERSION.SDK_INT > Build.VERSION_CODES.JELLY_BEAN_MR2){
            Log.d(TAG, "onItemClick: Trying to pair with " + deviceName);
            devicesBT.get(position).createBond();

            mBTdevice = devicesBT.get(position);
            mBluetoothConnection = new
            BluetoothConnectionService(MainActivity.this);
        }
    }

    @SuppressLint("LongLogTag")
    private void unpairDevice() {
        Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
        //if (pairedDevices.size() > 0) {
        for (BluetoothDevice device : pairedDevices) {
            try {
                Method m = device.getClass()
                        .getMethod("removeBond", (Class[]) null);
                m.invoke(device, (Object[]) null);
            } catch (Exception e) {
                Log.e("Removing has been failed.", e.getMessage());
            }
        }
    }

    // para a SMS
    private void pedirPermissaoSMS() {
        Log.d(TAG, "pedirPermissaoSMS: para o SMS");
```

```
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.SEND_SMS)
        != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this, new
            String[]{Manifest.permission.SEND_SMS}, 1);
        }
        else
            Log.d(TAG, "pedirPermissoes: OK");
    }

    @RequiresApi(api = Build.VERSION_CODES.M)
    public void EnviarSMS (View v){
        Log.d(TAG, "EnviarSMS: checando as permissões");
        pedirPermissaoSMS();

        try{
            Log.d(TAG, "EnviarSMS: tentando mandar SMS");
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendTextMessage("041998949563",null,"Uma queda foi
            detectada : maps.google.com/maps?q="+txtLatitude.getText()+","+
            txtLongitude.getText(),null,null);
            Toast.makeText(MainActivity.this, "SMS Sent Successfully",
            Toast.LENGTH_SHORT).show();
        }
        catch (Exception e){
            Toast.makeText(MainActivity.this, "SMS Failed to Send, Please try
            again", Toast.LENGTH_SHORT).show();
        }
    }

}
```

APÊNDICE D – BLUETOOTH CONNECTION SERVICE - ANDROID

```
package com.example.bluetooth;

import android.annotation.SuppressLint;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.charset.Charset;
import java.util.UUID;

public class BluetoothConnectionService {
    private static final String TAG = "BluetoothConnectionService";
    private static final String appName = "MYAPP";
    private static final UUID UUID_INSECURE = UUID.fromString("00001101-0000-1000-8

    private final BluetoothAdapter mBluetoothAdapter;
    Context mContext;

    private AcceptThread mInsecureAcceptThread;
    private ConnectThread mConnectThread;
    private BluetoothDevice mDevice;
```

```
private UUID deviceUUID;
ProgressDialog mProgressDialog;

private ConnectedThread mConnectedThread;

public BluetoothConnectionService(Context context){
    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    mContext = context;
}

private class AcceptThread extends Thread{
    private final BluetoothServerSocket mServerSocket;

    @SuppressLint("LongLogTag")
    public AcceptThread(){
        BluetoothServerSocket tmp = null;

        try{
            tmp =
            mBluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord(
            appName, UUID_INSECURE);
            Log.d(TAG, "AcceptThread: Setting up Server using: " +
            UUID_INSECURE);
        }catch (IOException e){
            Log.e(TAG,"AcceptThread: IOException: " + e.getMessage());
        }
        mServerSocket = tmp;
    }

    @SuppressLint("LongLogTag")
    public void run(){
        Log.d(TAG,"run: AcceptThread Running");
        BluetoothSocket socket = null;
```

```
            try{
                Log.d(TAG,"run: RFCOM server socket start...");
                socket = mServerSocket.accept();
                Log.d(TAG,"run: RFCOM server socket accepted connection");
            }catch (IOException e){
                Log.e(TAG,"AcceptThread: IOException: " + e.getMessage());
            }

            if(socket != null){
                connected(socket,mDevice);
            }
            Log.i(TAG, "END mAcceptThread");
        }

        @SuppressLint("LongLogTag")
        public void cancel(){
            Log.d(TAG, "cancel: Cancelling AcceptThread");
            try{
                mServerSocket.close();
            }catch (IOException e){
                Log.e(TAG, "cancel: Close of AcceptThread ServerSocket failed: " +
                e.getMessage());
            }
        }
    }

    private class ConnectThread extends Thread {
        private BluetoothSocket mSocket;

        @SuppressLint("LongLogTag")
        public ConnectThread(BluetoothDevice device, UUID uuid){
            Log.d(TAG, "ConnectThread: started");
```

```
        mDevice = device;
        deviceUUID = uuid;
    }

    @SuppressLint("LongLogTag")
    public void run() {
        BluetoothSocket tmp = null;
        Log.i(TAG, "RUN mConnectThread");

        try{
            Log.d(TAG, "ConnectThread: Trying to create InsecureRFcommSocket
            using UUID: " + UUID_INSECURE);
            tmp = mDevice.createRfcommSocketToServiceRecord(deviceUUID);
        }catch (IOException e){
            Log.e(TAG, "ConnectThread: Could not create InsecureRFcommSocket "
            + e.getMessage());
        }
        mSocket = tmp;

        mBluetoothAdapter.cancelDiscovery();
        try {
            mSocket.connect();
            Log.d(TAG, "run: ConnectThread connected");
        } catch (IOException e) {
            try {
                mSocket.close();
                Log.d(TAG, "run: Closed Socket");
            } catch (IOException e1) {
                Log.e(TAG, "mConnectThread: run: Unable to close connection in
                socket");
            }
        }
```

```
            connected(mSocket,mDevice);
        }

        @SuppressLint("LongLogTag")
        public void cancel(){
            try{
                Log.d(TAG, "cancel: Closing Client Socket");
                mSocket.close();
            }catch (IOException e){
                Log.e(TAG, "cancel: close() of mSocket n ConnectThread failed");
            }
        }
    }

    @SuppressLint("LongLogTag")
    public synchronized void start(){
        Log.d(TAG, "start");

        if(mConnectThread != null){
            mConnectThread.cancel();
            mConnectThread = null;
        }
        if(mInsecureAcceptThread == null){
            mInsecureAcceptThread = new AcceptThread();
            mInsecureAcceptThread.start();
        }
    }

    @SuppressLint("LongLogTag")
    public void startClient(BluetoothDevice device, UUID uuid){
        Log.d(TAG, "startClient: Started");

        mProgressDialog = ProgressDialog.show(mContext, "Connecting Bluetooth",
```

```
        "Please wait...", true);
        mConnectThread = new ConnectThread(device, uuid);
        mConnectThread.start();
    }

    private class ConnectedThread extends Thread{
        private final BluetoothSocket mSocket;
        private final InputStream mInputStream;
        private final OutputStream mOutputStream;

        @SuppressLint("LongLogTag")
        public ConnectedThread(BluetoothSocket socket){
            Log.d(TAG, "ConnectedThread: Starting");

            mSocket = socket;
            InputStream tmpIn = null;
            OutputStream tmpOut = null;

            try{
                mProgressDialog.dismiss();
            }catch(NullPointerException e){
                e.printStackTrace();
            }

            try {
                tmpIn = mSocket.getInputStream();
                tmpOut = mSocket.getOutputStream();
            } catch (IOException e) {
                e.printStackTrace();
            }

            mInputStream = tmpIn;
            mOutputStream = tmpOut;
```

```
}

@SuppressLint("LongLogTag")
public void run() {
    byte[] buffer = new byte[1024];
    int bytes;

    while(true){
        try {
            bytes = mInputStream.read(buffer);
            String incomingMessage = new String(buffer,0,bytes);
            Log.d(TAG, "InputStream: " + incomingMessage);

            Intent incomingMessageIntent = new Intent("incomingMessage");
            incomingMessageIntent.putExtra("theMessage", incomingMessage);
            LocalBroadcastManager.getInstance(mContext).
            sendBroadcast(incomingMessageIntent);
        } catch (IOException e) {
            Log.e(TAG, "write: Error reading to inputstream " +
            e.getMessage());
            break;
        }
    }
}

@SuppressLint("LongLogTag")
public void write(byte[] bytes){
    String text = new String(bytes, Charset.defaultCharset());
    Log.d(TAG, "write: Writing to outputStream: " + text);
    try{
        mOutputStream.write(bytes);
    }catch (IOException e){
        Log.e(TAG, "write: Error writing to outputstream " +
```

```
                e.getMessage());
            }
        }

        @SuppressLint("LongLogTag")
        public void cancel(){
            try{
                mSocket.close();
            }catch (IOException e){
                Log.e(TAG, "cancel: Error to close socket");
            }
        }
    }

    @SuppressLint("LongLogTag")
    private void connected(BluetoothSocket mSocket, BluetoothDevice mDevice){
        Log.d(TAG, "connected: Starting");

        mConnectedThread = new ConnectedThread(mSocket);
        mConnectedThread.start();
    }

    @SuppressLint("LongLogTag")
    public void write(byte[] out){
        ConnectedThread r;

        Log.d(TAG, "write: Write Called");
        mConnectedThread.write(out);
    }
}
```

APÊNDICE E – ALERTA - ANDROID

```
package com.example.bluetooth;

import android.Manifest;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.telephony.SmsManager;
import android.widget.Toast;
import android.os.Handler;




public class Alerta extends AppCompatActivity {
    private static final String TAG = "Alerta";
    TextView txtLatitude2, txtLongitude2;
    int alerta = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.alerta);
        Log.d("Alerta:", "startAlerta");

        txtLatitude2 = (TextView) findViewById(R.id.txtLatitude2);
        txtLongitude2 = (TextView) findViewById(R.id.txtLongitude2);



        permissaoSMS();
        pedirPermissoes();

        final Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    // Do something after 5s = 60000ms
                    if (alerta == 1) {
                        EnviarSMS();
                        alerta = 0;}
                }
            }, 60000);

    }

    public void onClickbNAO (View view) {
        EnviarSMS ();
    }

    public void onClickbSIM(View view) {
        alerta = 0;
        chamarMain();
    }
```

```
public void chamarMain(){
    finish();
}

private void permissaoSMS(){
    Log.d(TAG, "permissaoSMS: para o SMS");

    if (ActivityCompat.checkSelfPermission(this, Manifest.permission.SEND_SMS)
    != PackageManager.PERMISSION_GRANTED) {
        Log.d(TAG, "permissaoSMS: permissão SMS não concedida");
        Toast.makeText(Alerta.this, "Sem permissão para o SMS ",
        Toast.LENGTH_SHORT).show();
        ActivityCompat.requestPermissions(this, new
        String[]{Manifest.permission.SEND_SMS}, 1);
    }
    else{
        Log.d(TAG, "permissaoSMS: OK !");
    }
}

// para a Localização
private void pedirPermissoes() {
    Log.d(TAG, "pedirPermissoes: para o GPS");

    if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_FINE_LOCATION) !=
    PackageManager.PERMISSION_GRANTED
            && ActivityCompat.checkSelfPermission(this,
            Manifest.permission.ACCESS_COARSE_LOCATION) !=
            PackageManager.PERMISSION_GRANTED) {

        ActivityCompat.requestPermissions(this, new
        String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
```

```
            ActivityCompat.requestPermissions(this, new
            String[]{Manifest.permission.SEND_SMS}, 1);
        }
        else{
            Log.d(TAG, "pedirPermissoes: OK !");
            configurarServico();
        }
    }

    // para a Localização - invoca automático se o usuário não der permissão
    @Override
    public void onRequestPermissionsResult(int requestCode, String permissions[],
    int[] grantResults) {
        Log.d(TAG, "onRequestPermissionsResult: para o GPS");
        switch (requestCode) {
            case 1: {
                if (grantResults.length > 0 && grantResults[0] ==
                PackageManager.PERMISSION_GRANTED) {
                    configurarServico();
                } else {
                    Toast.makeText(this, "Não vai funcionar!!!",
                    Toast.LENGTH_LONG).show();
                }
                return;
            }
        }
    }

    // Método que lê o serviço de GPS do Android
    public void configurarServico(){
        Log.d(TAG, "configurarServico: para o GPS");
        try {
            LocationManager locationManager = (LocationManager)
```

```
            getSystemService(Context.LOCATION_SERVICE);

            LocationListener locationListener = new LocationListener() {
                public void onLocationChanged(Location location) {
                    atualizar(location);
                }

                public void onStatusChanged(String provider, int status, Bundle
                extras) { }

                public void onProviderEnabled(String provider) { }

                public void onProviderDisabled(String provider) { }

            };

            //locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
            0, 0, locationListener);
            locationManager.requestLocationUpdates
            (LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
        }catch(SecurityException ex){
            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_LONG).show();
            Log.d(TAG, "configurarServico: erro");
        }

    }

    // Joga as informações da LAT/LONG no TextView
    public void atualizar(Location location) {
        Log.d(TAG, "atualizar: para o GPS");

        Double latPoint = location.getLatitude();
        Double lngPoint = location.getLongitude();
```

```
        txtLatitude2.setText(latPoint.toString());
        txtLongitude2.setText(lngPoint.toString());
    }

    public void EnviarSMS (){

        try{
            Log.d(TAG, "EnviarSMS: tentando mandar SMS");
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendTextMessage("041998949563",null,"Uma queda foi
            detectada :
            maps.google.com/maps?q="+txtLatitude2.getText()+",
            "+txtLongitude2.getText(),null,null);
            Toast.makeText(Alerta.this, "SMS Sent Successfully",
            Toast.LENGTH_SHORT).show();
            //chamarMain();
        }
        catch (Exception e){
            Toast.makeText(Alerta.this, "SMS Failed to Send, Please try again",
            Toast.LENGTH_SHORT).show();
        }

    }
}
```