UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

Um dispositivo vestível e confiável para detectar quedas

JOÃO CARLOS BRITTO FILHO

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Microeletrônica.

Orientador: Prof. Dr. Marcelo Soares Lubaszewski

Porto Alegre
2022

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Filho, Joao Carlos Britto
Um dispositivo vestível e confiável para
detectar quedas / João Carlos Britto Filho. -- 2021.
102 f.
Orientador: Marcelo Soares Lubaszewski.
Dissertação (Mestrado) -- Universidade Federal do
Rio Grande do Sul, Instituto de Informática, Programa
de Pós-Graduação em Microeletrônica, Porto Alegre,
BR-RS, 2021.
1. Tolerância a falhas. I. Lubaszewski, Marcelo
Soares, orient. II. Doutor.

AGRADECIMENTOS

Quero agradecer inicialmente aos meus pais, João e Roseli, e irmãos, Marcos e Matheus, que me incentivaram nos momentos difíceis e compreenderam a minha ausência enquanto eu me dedicava à realização deste trabalho e durante todo o curso. Aos amigos Júnior e Stephan, que sempre estiveram ao meu lado, pela amizade incondicional e pelo apoio demonstrado ao longo de todo o período de tempo em que me dediquei a este trabalho. Ao meu sócio e amigo Rafael que sempre entendeu minha ausência para escrever esta dissertação. Ao professor Marcelo, por ter sido meu orientador e ter desempenhado tal função com dedicação, amizade e muito empenho, a quem também devo o amor à área de testes. A todos que participaram, direta ou indiretamente do desenvolvimento deste trabalho de pesquisa, enriquecendo o meu processo de aprendizado.

RESUMO

Uma queda tratada tardiamente pode causar danos irreparáveis ou até a morte. Esse problema é agravado quando é um idoso que sofre a queda. Existem soluções no mercado para a resolução desse problema, porém, não abordam adequadamente a confiabilidade dos dispositivos. Nenhuma das soluções encontradas durante as nossas pesquisas apresenta claramente suas capacidades em termos de teste, segurança de operação e tolerância a falhas. Este trabalho não apenas estrutura os testes off-line e on-line aos quais o dispositivo deve ser exposto, como também propõe o uso de redundância de hardware e software para fornecer uma solução mais confiável para o problema. Foi desenvolvido um protótipo de um dispositivo vestível para detecção de quedas, que fornece operação segura e tolera falhas, além de oferecer baixo consumo de corrente e baixo custo, que são especificações essenciais para atender às demandas do mercado por dispositivos vestíveis.

Palavras-chave: Detectores de queda. Dispositivos vestíveis. Testes. Segurança operacional. Tolerância a falhas.

A Reliable Wearable Device for Fall Detection

ABSTRACT

A late treated fall may cause irreparable damage or even the death of a human being. This problem gets worse with the older ages of the person suffering the fall. Solutions to this problem do exist in the market but do not appropriately address the reliability of using the proposed devices. None of the solutions found in our research clearly present their capabilities in terms of testing, operation safety and fault tolerance. This work not only gets structured the off-line and on-line tests to which the device should be exposed, but also proposes the use of hardware and software redundancy to provide for a more reliable solution for the problem. A prototype of a wearable device for fall detection has been developed that provides for safe operation and tolerates faults, while delivering very low current consumption and low cost, those that are essential specifications to meet the market demands for wearable devices.

Keywords: Fall detectors. Wearable devices. Testing. Operation safety. Fault tolerance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

TCE	Traumatismo CranioEncefálico
ADV	Atividades da vida diária
CPU	*Central Process Unit*
ROM	*Read-only memory*
PROM	*Programmable read-only memory*
EPROM	*Erasable programmable read-only memory*
RAM	*Random Access Memory*
SBC	*Single Board Computers*
I2C	*Inter-Integrated Circuit.*
SPI	*Serial Peripheral Interface*
UART	*Universal Asynchronous Receiver/Transmitter*
IoT	*Internet of things*
BLE	*Bluetooth Low Energy*
DfT	*Discrete Fourier Transform*
LFSR	*Linear Feedback Shift Registers*
BIST	*Built-In Self-Test*
TMR	*Triple Modular Redundancy*
NMR	*N-Modular Redundancy*
API	*Application Programming Interface*
DMP	*Digital Motion Processor*

SUMÁRIO

1 INTRODUÇÃO

QUEDAS são a quinta causa de morte e uma das principais causas de hospitalização (MEUCCI, 2019). O traumatismo cranioencefálico (TCE), causado por uma queda, é uma das principais causas de mortalidade e incapacidade no mundo. Mundialmente, a taxa de mortalidade por trauma é de 19 por 100.000 habitantes (BURGOS-FLÓREZ, 2020). O TCE representa pelo menos metade das mortes relacionadas ao trauma e produz altos custos para o sistema de saúde devido ao tratamento e reabilitação dos pacientes. Um estudo latino-americano recente relatou uma taxa de mortalidade de 37% em um total de 484 pacientes com TCE grave. O TCE é produzido por cargas de impacto no crânio, conhecidas como lesão direta, ou pela aceleração ou desaceleração da cabeça sem a aplicação direta de carga, conhecida como lesão difusa. No entanto, na maioria dos casos, uma combinação de cargas de impacto e aceleração está presente (BURGOS-FLÓREZ, 2020).

As quedas podem resultar em um trauma grave no tecido ósseo e cerebral (BURGOS-FLÓREZ, 2020), que geralmente piora quando o indivíduo é uma pessoa idosa. Por exemplo, estima-se que no Brasil, anualmente, 30% dos idosos com mais de 65 anos sofrem pelo menos uma queda não intencional (SIQUEIRA, 2011). Quedas podem incapacitar, ferir e até evoluir para o óbito (MEUCCI, 2019), sendo a causa de 15% das mortes de idosos no Brasil (WANDERLEY, 2019). Cabe ressaltar que os idosos sofrem o maior número de quedas fatais (SALEH, 2017). Outro ponto importante é que idosos podem sofrer quedas dentro de hospitais ou asilos, estas que podem ou não serem detectadas pelos profissionais que lá trabalham (COUTINHO, 2012). Dessa forma, o processo de cuidado e a qualidade de vida dos idosos podem ser melhorados com a adoção de sistemas de detecção automática de quedas (ABBATE, 2012). A queda de pacientes idosos ainda é um problema médico crítico, pois pode causar lesões ósseas irreversíveis devido à fraqueza dos ossos desses idosos (KERDJIDJ, 2020).

A letalidade da queda é diretamente proporcional à lesão e ao tempo que leva até que os primeiros cuidados sejam prestados à vítima (BURGOS-FLÓREZ, 2020). Assim, uma queda tratada tardiamente pode ser muito mais prejudicial do que uma queda mais grave tratada logo após à sua ocorrência (SALEH, 2017). Neste contexto, a detecção automática de quedas para idosos torna-se uma aplicação de saúde muito importante, pois permite uma intervenção médica oportuna (SALEH, 2017). O problema de detecção de quedas foi amplamente estudado na última década. No entanto, uma vez que os recursos de hardware de

dispositivos vestíveis são limitados, projetar algoritmos embutidos de alta precisão com custo computacional viável ainda é um desafio de pesquisa aberto (SALEH, 2017).

Atualmente, existem vários dispositivos e serviços de aplicativos no mercado para notificar um profissional de saúde, ou membro da família de uma pessoa idosa, quando ela cai. Entre eles, destacamos aquelas que são soluções baseadas em *smartphones* (ABBATE, 2012) e soluções dedicadas, que integram microprocessadores (ou microcontroladores) com vários sensores (VAN THANH, 2019). Os sensores normalmente usados para construir essas soluções são acelerômetros e giroscópios (WANG, 2018).

De acordo com (Fernandes, 2022), apesar de haver um crescente uso de *smartphones* de forma minimamente funcional pela população de idosos, a maioria ainda utiliza aparelhos mais simples principalmente para realizar ligações. Além disso, dentre os idosos que utilizam o celular de forma corriqueira existe a possibilidade de sofrerem acidentes, justamente pela utilização do próprio celular, enquanto realizam outras tarefas (LACERDA, 2021). Assim, as soluções baseadas em *smartphones* podem não ser tão eficazes ou seguras para as pessoas mais velhas, tornando a escolha mais apropriada o uso de dispositivos dedicados. Além disso, essas soluções devem ser dispositivos portáteis para facilitar o uso, conforme descrito em (WANG, 2018).

Embora já existam dispositivos comerciais para a detecção de quedas, eles tipicamente não tratam de forma adequada os problemas de teste, segurança e confiabilidade. Tais dispositivos, encarregados de monitorar condições físicas de pessoas idosas, que podem significar, inclusive, a diferença entre a vida e a morte do usuário, devem ser altamente confiáveis, devem ter altos padrões de segurança operacional e, na medida do possível, devem tolerar falhas durante a aplicação. Nenhuma das soluções encontradas durante a fase de pesquisa demonstra seus recursos de teste, segurança e tolerância a falhas (ALVES, 2016).

Este trabalho não apenas avalia e estrutura os testes *off-line* e *on-line* aos quais o dispositivo deve ser exposto ao longo da aplicação, como também propõe o uso de redundância de *hardware* e *software* para fornecer operação segura e tolerância a falhas do dispositivo.

Nessa linha, este trabalho propõe um dispositivo que executa a detecção de quedas e integra técnicas de teste e tolerância a falhas para garantir um uso altamente confiável. Faz parte da especificação do dispositivo que este seja uma solução de baixo custo e, como deve

ser vestível, a solução também deve consumir pouca energia e ter um bom equilíbrio entre consumo e disponibilidade do sistema.

O dispositivo dedicado para a detecção de quedas de idosos será aqui apresentado segundo a seguinte organização em capítulos: No Capítulo 2, serão descritos os componentes necessários para a composição do dispositivo e comparadas as características individuais de cada componente. No Capítulo 3, serão discutidas as técnicas de tolerância a falhas utilizadas no mesmo para que este torne-se um dispositivo mais seguro. Já no Capítulo 4, serão apresentadas as escolhas e justificativas de cada componente e técnica utilizados na composição do protótipo de detecção de quedas desenvolvido neste trabalho. No Capítulo 5, serão apresentados os resultados práticos da implementação do protótipo, e no Capítulo 6, as considerações finais e trabalhos futuros.

2 DETECÇÃO DE QUEDAS: DISPOSITIVOS E SEUS COMPONENTES

Atualmente, existem diversas soluções no mercado para a detecção de quedas, como as baseadas em *smartphones* (ABBATE, 2012), fones de ouvido (VAN THANH, 2019), pulseiras e relógios inteligentes (DHIAH EL DIEHN, 2016) e também cintos inteligentes, conforme descrito em (ABBATE, 2012).

Basicamente, estas soluções podem ser subdivididas em 3 grandes categorias, que serão brevemente descritas na sequência:

- soluções baseadas exclusivamente na utilização de um *smartphone*;
- soluções híbridas, contando com um dispositivo vestível como parte da solução baseada em *smartphone*; e
- dispositivos vestíveis completamente dedicados à detecção de queda.

As soluções baseadas exclusivamente em *smartphones* como dispositivos detectores de queda normalmente utilizam o sensor acelerômetro com giroscópio interno como fonte de entrada de dados e os valores por ele coletados são processados por algum *software* embarcado no *smartphone* (ABBATE, 2012). Estes dispositivos não dedicados estão sujeitos a diversos problemas, como principais é possível citar a limitação de hardware dedicado à tarefa, travamentos de *software* e, conforme anteriormente mencionado, a dificuldade de utilização por pessoas de idade avançada (AMORIM, 2018).

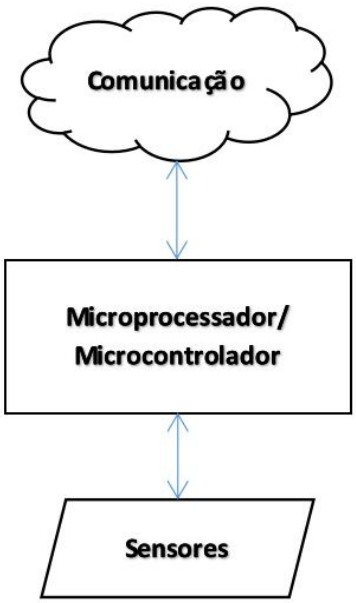
Diferentemente das soluções baseadas unicamente em *smartphones*, as soluções híbridas combinam um dispositivo vestível com um *smartphone*. Apesar de serem soluções que possam parecer mais robustas, são, na verdade, mais suscetíveis a falhas e erros. Por exemplo, em uma solução baseada em um *smartwatch*, onde este detecta a queda e envia um sinal ao *smartphone*, podem ocorrer falhas na comunicação entre os dois dispositivos. E, mesmo que falhas de comunicação entre os dispositivos não ocorram, o *smartphone* deve tratar com prioridade o sinal recebido do *smartwatch* para evitar que o idoso caia, danifique o *smartphone* e o aviso de queda do idoso não seja enviado a um terceiro, seu cuidador e/ou responsável. Como o objetivo do dispositivo desenvolvido neste trabalho é prestar um serviço seguro a idosos, sem complicar-lhe mais a vida, o fato de serem dois equipamentos que precisam estar constantemente com a bateria com carga suficiente para funcionarem adequadamente já é um ponto a mais de vulnerabilidade para a aplicação. Some-se a isso o

fato do idoso ter que lembrar de levar sempre consigo os dois equipamentos e também a possível dificuldade do mesmo para configurar ambos os dispositivos e a comunicação entre eles. Os sensores utilizados neste tipo de solução, normalmente, são os acelerômetros, giroscópios, barômetros muitas vezes em conjunto com medidores de pulso, de batimento cardíaco e/ou oxímetro disponíveis no *smartphone* e no *smartwatch*.

Soluções dedicadas, por sua vez, quando comparadas às soluções anteriores tendem a ser mais seguras do ponto de vista funcional e menos demandantes de atenção e habilidades do usuário idoso. Normalmente utilizam *hardware* dedicado para a detecção de quedas, o que auxilia na diminuição do problema de travamentos comum quando a plataforma utilizada é baseada em *smartphone*. Outro ponto é que, utilizando um microcontrolador dedicado, seu tempo de *reboot* é mínimo em comparação a um *smartphone*. No contexto de *reboots* inesperados por problemas de *software*, *hardware* ou ambos, esta característica torna-se uma vantagem, pois o sistema, potencialmente, tem a capacidade de recuperar-se com mais rapidez. Por último, estas soluções integram sensores como acelerômetros, giroscópios, sensores de inclinação, barômetros ou até câmeras (VAN THANH, 2019).

Essencialmente, qualquer das soluções anteriores é baseada em um ou mais sensores para a detecção de queda, em um elemento de processamento (tipicamente um microprocessador ou microcontrolador) que acopla estes sensores, mede, calcula, compara, classifica e, finalmente comunica ao dispositivo do cuidador, diretamente ou através de um repositório compartilhado de dados (na nuvem, por exemplo), a ocorrência ou não de uma queda do idoso. A grandeza medida pelo sensor deve ser tal que seja possível distinguir com precisão entre uma queda e outras situações diferentes que fazem parte do cotidiano do idoso, como deitar e sentar. Cabe ao microprocessador/microcontrolador converter os dados coletados do sensor em informações, com significado, e classificar a condição do usuário de acordo com movimentos como sentar, deitar, andar e cair. Também é necessário o uso de alguma comunicação externa para notificar terceiros sobre a ocorrência de uma queda. Essa comunicação deve ocorrer com segurança, rapidez e confiabilidade. A Figura 1 mostra como esses componentes tipicamente se interconectam em um dispositivo, para que possam, de maneira conjunta, detectar e sinalizar quedas do usuário.

Figura 1 – Componentes típicos de dispositivos para a detecção de quedas



Fonte: Produzida pelo Autor.

Nas sessões seguintes, serão apresentadas alternativas de implementação para cada um dos componentes apresentados na Figura 1.

2.1 Sensores

A gama de sensores utilizados na composição destas soluções é bastante variada, porém a grande maioria é baseada na utilização de acelerômetro juntamente com giroscópio. Outras possíveis alternativas são a utilização de barômetros, sensores de inclinação ou até câmeras.

2.1.1 Barômetro

O barômetro mede a pressão atmosférica, assim podendo ser utilizado para medir a pressão da coluna de ar sobre ele, realizando a medida indireta da taxa de queda de um objeto

ao qual o sensor esteja acoplado. O barômetro bme 280 [1](BOSH, 2018), por exemplo, mede a pressão em milibar, e em seu fundo de escala, consegue identificar a diferença de altura em relação a duas medidas com até 0,17 m de precisão. As soluções de detecção de queda baseadas neste tipo de sensor identificam uma diferença brusca entre duas ou mais medições. Como a pressão atmosférica não varia tão abruptamente, uma variação brusca é considerada uma queda. Estes sistemas de fato funcionam, mas apresentam falsos positivos, principalmente quando um idoso senta em uma cadeira mais baixa ou deita-se em uma cama, ou até mesmo quando utiliza um elevador. Assim, o uso de um barômetro mostra-se como uma solução que, isolada de outros sensores, pode não atender o nível necessário de precisão para implementar-se um instrumento adequado para a detecção de quedas.

2.1.2 Câmeras

Com o advento de câmeras mais baratas e melhor resolução somado ao fato de que a transmissão de dados ficou mais rápida e estável, tornou-se possível a utilização destas para a detecção de movimento, de face, objetos e poses. Outra utilização mais recente desta tecnologia é na detecção de quedas (OZCAN, 2013). A eficiência da detecção de quedas utilizando câmeras, no entanto, é particularmente complexa de mensurar, já que, normalmente, a detecção é feita com base em um modelo ou rede neural que deve ser treinada e criada de acordo com muitas variáveis e diferentes *datasets*. Outro desafio é que as imagens devem ser transmitidas e analisadas em tempo real, pois uma omissão pode significar uma queda não identificada. Não menos importante, destacam-se os falsos positivos ou os falsos negativos, estes provenientes de modelos insuficientemente treinados ou até de uma iluminação desfavorável. Por último, é importante equilibrar o consumo energético de uma solução desta natureza, que mesmo ligada a uma fonte de alimentação externa, consome consideravelmente mais energia do que outras soluções (partindo de 10 vezes mais e aumentando em comparação com soluções *low energy*), inclusive, com acurácia já comprovada.

[1] Disponível em: https://datasheetspdf.com/pdf/1096951/Bosch/BME280/1.

2.1.3 Acelerômetro

O acelerômetro é um sensor utilizado para medir a vibração ou aceleração de um movimento ao qual o sensor é exposto. Normalmente, é uma solução composta por um centro de massa equalizado como uma ponte de *wheatstone* composta por um sensor piezoelétrico. Quando a massa sofre uma aceleração no sentido em que o movimento é permitido, a mesma comprime o sensor piezo-elétrico, desbalanceando a ponte de *wheatstone*. Se esta ponte for alimentada, é possível mensurar a variação de tensão. Tendo como dados a quantidade de massa, deformação do extensômetro, alimentação do sistema e demais resistências presentes na ponte de *wheatstone*, é possível determinar a aceleração do sistema. Com estes dados, e um algoritmo específico, é possível determinar, com relativa precisão, se uma queda ocorreu. A precisão ainda pode ser melhorada com a utilização de componentes extras, como um giroscópio, na composição da solução.

2.1.4 Giroscópio

O giroscópio é um dispositivo que permite mensurar a velocidade de um objeto sobre o seu centro de rotação e seu envio de dados se dá em determinado intervalo de tempo. Ou seja, é um sensor que mede a velocidade angular em relação ao tempo, fato tal que permite estimar, de forma muito precisa, a posição angular do objeto (sensor) sobre seu eixo. O modelo de giroscópio escolhido para o projeto aqui apresentado tem a particularidade de ser fabricado com tecnologia MEMS. Este utiliza o efeito Coriolis, baseado na combinação do movimento oscilatório de uma massa e a rotação do sistema.

No giroscópio, internamente ao sensor, existe um centro de massa suspenso por uma quantidade de substrato flexível que tem uma função parecida com um sistema massa-mola. Quando ocorre uma rotação, a massa no centro do substrato flexível sofre o efeito *Coriolis*, que transfere uma força a um elemento resistivo. Este elemento tem o valor de sua resistência modificado de acordo com o deslocamento (encurtando ou esticando o material). Este valor resistivo varia de maneira proporcional à velocidade angular desta rotação e tem características similares às descritas anteriormente no acelerômetro. Com este sensor, é possível determinar com relativa precisão se uma queda ocorreu, porém, a utilização desse

com o acelerômetro é muito indicada na literatura para a diminuição de falsos positivos ou até utilizando este em dispositivos ADV - detectores de atividade da vida diária, que registram atividades variadas do usuário, e não somente quedas.

2.1.5 Inclinômetro

Além dos inclinômetros baseados na utilização em conjunto de acelerômetros e giroscópios, sensores já abordados anteriormente, existe o inclinômetro com ângulo fixo (conhecido como *Tilt*). Este atua de forma binária, acionando ou desativando o contato entre seus terminais de acordo com o ângulo previamente calibrado. Por exemplo, considere-se que o inclinômetro fosse calibrado para um ângulo de 10 graus. Uma vez que a inclinação medida atinja valores acima de 10 graus, o contato fecha-se. É possível utilizar quatro destes sensores para que uma queda seja detectada. O seu maior problema são os falsos negativos, pois, dependendo de como ocorra a queda, a mesma pode não ser detectada. Por exemplo, se o conjunto de sensores for calibrado para que seja detectada uma queda frontal, é possível que uma queda lateral ou para trás não seja detectada.

2.2 Microprocessadores e microcontroladores

Microcontroladores são compostos por uma CPU (*Central Processing Unit*), memória não volátil (ROM/PROM/EPROM/Flash), memória volátil (RAM) e unidades periféricas (portas de entrada e saída, portas de comunicação serial, paralela, etc) (GODSE, 2020). Já os microprocessadores, tipicamente só contém a CPU, necessitando que memórias voláteis e não voláteis sejam a eles conectadas para o seu pleno funcionamento (GODSE, 2020). Normalmente, os microcontroladores operam a frequências de relógio mais baixas e são mais comumente utilizados em sistemas dedicados que executam tarefas específicas. Por outro lado, os microprocessadores são utilizados para a realização de tarefas genéricas ou mais complexas. Em questão de consumo energético, os microcontroladores, geralmente, consomem menos do que os microprocessadores. Este fato está ligado principalmente a frequências de operação menores quando comparados aos microprocessadores.

Para a detecção de quedas, existem soluções baseadas em ambos, microprocessadores e microcontroladores. As soluções baseadas em *smartphones* e *single board computers* (SBC) tem como base um microprocessador. As dedicadas tem como base um microcontrolador. Num contexto de soluções vestíveis, a maior desvantagem na utilização de uma SBC para este tipo de aplicação é o seu consumo elevado de corrente em modo *standby* (quando não está executando nenhum código). A Tabela 1 ilustra o consumo médio , de acordo com o site https://www.raspberrypi.com/documentation/computers/raspberry-pi.html, de algumas SBC utilizadas na composição da solução para detecção de quedas.

Tabela 1 - Comparação de correntes de trabalho e em *standby* de SBCs

SBC	Corrente Máxima	Corrente em *Standby*
Raspberry Pi Model A	700 mA	220 mA
Raspberry Pi Model B	1,2 A	500 mA
Raspberry Pi Model A+	700 mA	180 mA
Raspberry Pi Model B+	1,8 A	330 mA
Raspberry Pi 2 Model B	1,8 A	350 mA
Raspberry Pi 3 Model B	2,5 A	400 mA
Raspberry Pi 3 Model A+	2,5 A	350 mA
Raspberry Pi 3 Model B+	2,5 A	500 mA
Raspberry Pi 4 Model B	3,0 A	600 mA
Raspberry Pi Zero	1,2 A	100 mA
Raspberry Pi Zero W/WH	1,2 A	150 mA

Fonte: Extraída de (RASPBERRYPI, 2022) [2]pelo autor

Por outro lado, como é possível perceber na Tabela 2, todos os microcontroladores comparados tem uma corrente de trabalho máxima e mínima muito inferiores às apresentadas pelas SBCs da Tabela 1.

Tabela 2 - Correntes de trabalho e em *deep-sleep* de microcontroladores

Microcontrolador	Corrente Máxima	Corrente em *deep-sleep*
Esp32	120 mA	0,011 mA
Esp8266	70 mA	0,165 mA
Arduino UNO (ATMEGA 328)	50 mA	7 mA
PIC 4550	48 mA	0,021 mA

Fonte: Extraído de (ESPRESSIF[3], 2022), (ARDUINO, 2022[4]) e (MICROCHIP[5], 2022) pelo autor

Outro ponto a ser destacado é o tempo de inicialização das SBCs relativamente longo em comparação a um microcontrolador. Como estas têm que executar um sistema operacional,

[2] Disponível em: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.
[3] Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32.pdf.
[4] Disponível em: https://docs.arduino.cc/retired/other/arduino-older-boards.
[5] Disponível em: https://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf.

e dentro deste sistema operacional executar a aplicação, acabam tendo um tempo de inicialização em torno de 40 segundos (utilizando como base o Raspberry pi 4 com 8Gb de RAM, armazenamento interno em cartão SD classe 10 e sistema operacional Raspberry pi OS na versão dezembro de 2020), utilizando a versão mais leve de seus sistemas operacionais. Por outro lado os microcontroladores, por realizarem uma tarefa específica, tem seu tempo de *reboot* e inicialização muito rápidos, normalmente alguns ciclos de *clock*.

2.3 Comunicação

A comunicação é um elemento fundamental em qualquer dispositivo digital. É o elemento que interliga diferentes componentes fazendo-os trocarem dados. Alguns exemplos são a comunicação entre memória e CPU, ou o envio de dados de um *smartphone* para a nuvem. Em termos gerais, a comunicação pode ser dividida em dois grandes grupos: comunicação interna e externa.

Para que a troca de informações transcorra de maneira adequada é necessário que exista um protocolo de comunicação. O protocolo de comunicação é uma espécie de moderador/tradutor, em que um elemento envia dados a outro e ambos compreendem esses dados de maneira a utilizá-los corretamente. Tipicamente, os protocolos têm características particulares e são otimizados para atender às necessidades de aplicações específicas.

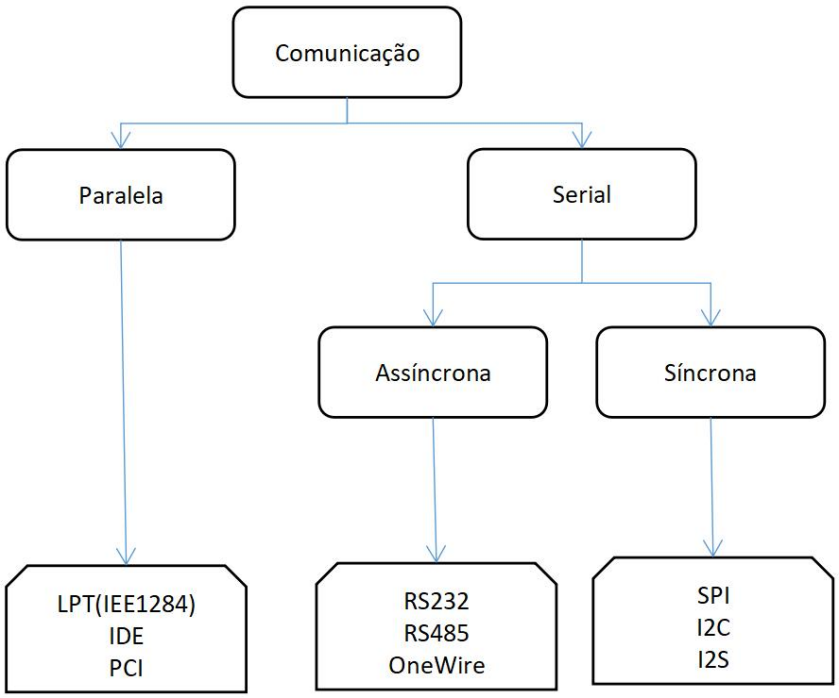
2.3.1 Comunicação entre elementos internos

A comunicação interna compreende a troca de informações entre elementos que compõem um chip, uma placa de circuito impresso ou um conjunto de placas. A comunicação interna se dá através de um meio físico.

O mais comum é que a comunicação entre elementos internos de um circuito seja realizada utilizando meios condutores de eletricidade. O uso de trilhas do circuito para a comunicação contribui para um menor consumo de energia quando comparado com a comunicação implementada pelo ar. Também o mais comum é que a comunicação seja realizada com base em protocolos seriais, exceto no caso da comunicação interna aos chips,

quando a utilização de barramentos e de protocolos paralelos é mais frequente. A Figura 2, abaixo, ilustra os tipos de comunicação interna e seus protocolos mais comuns.

Figura 2 – Comunicação interna e os protocolos mais comuns



Fonte: Produzida pelo Autor.

Para completar um ciclo de comunicação é necessário que dados sejam enviados e recebidos. Quando a comunicação entre as partes ocorre de forma simultânea, ou seja, recebe-se e enviam-se dados no mesmo período de tempo, a comunicação é dita *Full-Duplex*. Por outro lado, quando dados são obrigatoriamente transmitidos e recebidos em períodos de tempo distintos, a comunicação é conhecida como *Half-Duplex*.

Os protocolos seriais mais conhecidos e utilizados são o I2C (*Inter Integrated Circuit*), o SPI (*Serial Peripheral Interface*) e UART (*Universal Asynchronous Receiver Transmitter*). As características destes protocolos são comparadas na Tabela 3, abaixo.

Tabela 3 - Comparação entre os protocolos I2C, SPI e UART

Tecnologia	Barramento	Taxa máxima
UART(RS232)	2	115.200 bps
SPI	3 + nº de *Slaves*	2 Mbps
I2C	2 até 127	400 Kbps

Fonte: Produzida pelo autor

2.3.2 Comunicação entre elementos externos

Existem no mercado diversas tecnologias *wireless* que podem atender estas necessidades, as mais utilizadas, em se tratando de IoT (*Internet of Things*), são *bluetooth*, BLE (*Bluetooth Low Energy*), *Wi-Fi*, Zigbee e LoRa. Como é possível verificar na Tabela 4, para cada caso e cada aplicação existe uma ou mais tecnologias de comunicação sem fio que se encaixam.

Tabela 4 - Comparativo entre rádios usados na comunicação sem fio

Tecnologia	*Bluetooth Clássico*	*BLE*	*ZigBee*	*Wi-Fi*	*LoRa*
Frequências	2.4 GHz	2.4 GHz	868 MHz, 915 MHz, 2.4 GHz	2.4 GHz, 5 GHz	433 MHz, 868 MHz , 915 MHz e 923 MHz
Máxima Taxa de Bits (Mbps)	1 até 3	1	0.25	11(b) 54(g) 600(n)	0,003 a 0,05
Distância Máxima (m)	10-100	50	10-100	100-250	4000 a 15000
Consumo de Energia	Alto	Muito Baixo	Muito Baixo	Muito Alto	Extremamente baixo
Vida útil de uma bateria de 3300AH	Dias	Meses a Anos	Meses a anos	Horas	Meses a anos
Tamanho da rede (dispositivos)	até 7	Indefinido	64000+	255	2
Principa Desvantagens	Comunicação apenas local	Comunicação apenas local	Preço e Comunicação apenas local	Consumo elevado	Comunicação apenas com dispositivos LoRa.
Principal Vantagem	Retrocompati-bilidade	Baixo consumo energético	Baixo consumo energético	Comunicação com a internet	Baixo consumo energético

Fonte: Produzida pelo autor.

Um último aspecto relevante para este trabalho é que, não é incomum em aplicações de IoT, que algumas das tecnologias da Tabela 4 sirvam de suporte à comunicação com servidores externos para envio e registro de dados, e para o acesso a serviços de nuvem. Abordaremos a alternativa de uso de serviços na nuvem na próxima sessão que, conjuntamente com a tecnologia *Wi-Fi*, resultou na solução adotada no nosso protótipo para a comunicação externa

2.3.3 Servidor na nuvem

Existem diversas soluções em que serviços contratados na nuvem se encaixam em aplicações de IoT que utilizam dispositivos vestíveis, por exemplo. Estas podem ser baseadas em um servidor que processa os dados dos sensores e verifica se a queda ocorreu ou simplesmente num banco de dados onde o dispositivo detector sinaliza se a queda ocorreu. A maior vantagem de utilizar um servidor dedicado para este tipo de solução é que, possivelmente, o gasto energético do dispositivo dedicado seja menor por não necessitar realizar cálculos com os valores coletados e somente servir de ponte entre os sensores e o servidor onde a queda é detectada. Já a utilização de um banco de dados tem a vantagem de ter um preço mais atrativo, podendo ser, inclusive, gratuito. Outra grande limitação de servidores na nuvem é a quantidade de acessos simultâneos, que é diretamente proporcional ao valor do serviço contratado.

Quando utilizado um servidor na nuvem para calcular a probabilidade da queda ter ocorrido, é indispensável que haja uma comunicação ininterrupta dos dados pelo dispositivo que capta e envia os sinais dos sensores. No Brasil, este é um problema devido à má qualidade dos serviços de comunicação. Um outro ponto importante a ser destacado é que, para ser utilizado em uma aplicação de detecção de quedas, tanto o banco de dados na nuvem, quanto o servidor, devem ter um tempo de atualização/resposta baixo ou uma *flag* que sinaliza a atualização de um status/dado rapidamente, pois esse dado é enviado a um programa/aplicativo que avisa um terceiro sobre a queda.

Assim como no caso de tantos outros ADVs , embora existam várias alternativas para a detecção de quedas baseadas em diferentes sensores, microprocessadores/microcontroladores, diferentes dispositivos e serviços de comunicação, as soluções hoje adotadas tipicamente não tratam de forma adequada e completa os problemas de teste, segurança e confiabilidade do sistema resultante. Tais soluções, encarregadas de monitorar as condições físicas de pessoas, algumas delas podendo significar a diferença entre a vida e a morte do usuário, devem ser altamente confiáveis, devem ter altos padrões operacionais de segurança e, na medida do possível, tolerar falhas durante a aplicação. No próximo capítulo, trataremos de técnicas utilizadas para que estas soluções sejam menos suscetíveis a falhas que venham a ocorrer durante a aplicação do sistema e que tornem a experiência do usuário mais segura e confiável.

3 TESTE E TOLERÂNCIA A FALHAS

Os avanços tecnológicos têm sido tais que circuitos e sistemas eletrônicos a cada dia oferecem mais funcionalidades; tornam-se mais densos em termos de dispositivos, mas ao mesmo tempo mais compactos; consomem proporcionalmente menos energia e custam menos; e, concluem operações e cumprem sua missão muito mais rapidamente que suas versões mais antigas. Esse é um cenário de alta complexidade do ponto de vista do projeto destes circuitos e sistemas, considerando a complexidade da entrega de todos estes avanços com a qualidade que costuma requerer o exigente usuário final.

Esta garantia de qualidade é resultado das inúmeras verificações realizadas ao longo do projeto e da depuração dos primeiros protótipos, e consequência da triagem realizada durante os testes de produção e também daqueles realizados ao longo da vida destes circuitos e sistemas. Estas verificações e testes devem garantir o atendimento das especificações de projeto pelos diversos componentes eletrônicos, garantir que defeitos inerentes ao processo produtivo sejam identificados e as partes defeituosas descartadas, e garantir a confiabilidade do sistema durante seu uso na aplicação final.

Algumas aplicações mais modernas, como transações bancárias e o monitoramento de saúde, que são realizadas de maneira remota por dispositivos móveis, por exemplo, atingiram grande escala e são hoje utilizadas corriqueiramente por indivíduos comuns, apesar da sua alta complexidade devido à necessidade de segurança e confiabilidade das funções que realizam. Estas, assim como outras tantas aplicações, exigem a execução constante de testes de *software* e *hardware*, exigem a detecção e correção de erros durante seu uso, exigem redundância de componentes físicos ou a re-execução de funções críticas, dentre tantos outros métodos de teste que permitam assegurar a qualidade do serviço entregue ao usuário final.

Este capítulo apresenta e discute o impacto de alguns desses métodos, em particular daqueles que se aplicam ao dispositivo detector de quedas aqui estudado.

3.1 Métodos de teste

Em geral, independentemente da fase da vida do componente em que sejam aplicados, testes avaliam se o dispositivo físico fabricado atende às especificações previstas em seu

projeto. Quando é avaliado se o circuito tem um comportamento funcional correto, estamos realizando testes funcionais; quando se avalia a implementação física do circuito espelha seu esquemático, estamos realizando testes estruturais. Já que são realizados à velocidade nominal da aplicação, testes funcionais são amplamente utilizados para identificar problemas de desempenho que o circuito venha a apresentar, como atrasos excessivos em seus caminhos críticos de dados, por exemplo. Testes estruturais, por outro lado, abstraem a funcionalidade do circuito e baseiam-se em modelos de falhas, como veremos na próxima sessão. Por isso, podem prover uma medida de quantas falhas conseguem detectar de todas as falhas possíveis previstas (cobertura de falhas). Testes estruturais e testes funcionais costumam se complementar na vida real.

Desde a fabricação dos seus primeiros protótipos, um circuito passa necessariamente pelas etapas de depuração de projeto, de teste de produção, e possivelmente pela etapa de teste de manutenção quando já inserido na aplicação. Estas etapas identificam e isolam defeitos, ou até substituem as partes defeituosas. Como necessitam que a aplicação seja interrompida para a execução de procedimentos específicos, são conhecidos como testes fora de funcionamento (*off-line*). Sistemas que rodam aplicações críticas e, portanto, requerem segurança e confiabilidade na operação, devem tolerar mau funcionamento e, por isso, precisam detectar falhas concorrentemente à aplicação. Testes realizados nestas condições são conhecidos como testes em funcionamento (*on-line*). Assim como no caso dos testes estruturais e funcionais, na vida real de sistemas críticos em aplicações automotivas, aeronáuticas, bancárias ou de saúde, por exemplo, testes *on-line* e *off-line* costumam se complementar entre si.

Testes estruturais e funcionais, assim como testes *on-line* e *off-line*, serão objeto de análise ao estudarmos, na sequência, os métodos de teste mais comuns e também ao implementarmos, nos Capítulos 4 e 5, o protótipo confiável para a detecção de quedas.

3.1.1 Defeitos, Falhas e Erros

Antes de ingressarmos na discussão sobre os métodos para a realização de testes eficientes que garantam a qualidade do circuito produzido e a confiabilidade dos serviços prestados por este mesmo circuito ao usuário, é necessário o entendimento da origem da necessidade de realização desses testes.

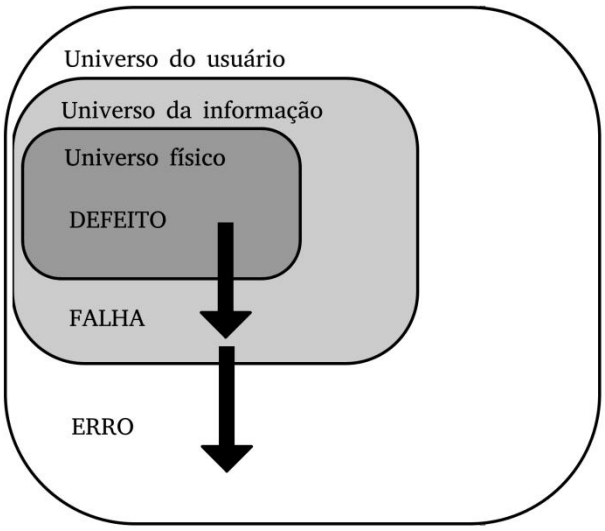
No mundo real, os processos produtivos são imperfeitos. Como consequência, a produção de circuitos está sujeita a possíveis diferenças entre a implementação física efetivamente obtida e aquela pretendida no projeto original, diferenças estas que, fora de limites tolerados, resultam em desvios da especificação do circuito. Estas diferenças físicas são o que chamamos de defeito de fabricação. Defeitos de fabricação tipicamente ocorrem devido a impurezas nos insumos, depósito de poeira sobre o substrato em processos de fabricação de circuitos integrados, mau funcionamento de equipamentos de produção, etc. Defeitos podem ocorrer, igualmente, ao longo da vida útil do circuito, uma vez que os materiais, com o uso, sofrem desgastes que podem evoluir para um mau funcionamento. Neste caso, os principais responsáveis são fenômenos eletromecânicos, de transporte, fatores térmicos e outros.

Um defeito pode estar presente no circuito e não ser percebido antes que uma determinada condição de funcionamento apareça. Quando há a manifestação interna da presença do defeito, traduzida, por exemplo, na alteração indevida de algum dado ou informação que esteja sendo tratado(a) pelo circuito, dizemos que o defeito se manifesta como uma falha. Falhas podem ter efeito permanente, se originadas em um defeito físico, ou transitórias, estas que ocorrem normalmente relacionadas a fenômenos aleatórios como radiação ou interferências eletromagnéticas. Falhas intermitentes, por outro lado, são caracterizadas pela ocorrência repetida ou temporária relacionada a um defeito e podem ser associadas a condições externas como aumento de temperatura, vibração, etc.(BALEN, 2005)

Sempre que uma falha interna se traduzir em mau funcionamento do circuito, que se manifeste no contexto da aplicação na qual o circuito está inserido, dizemos que um erro ocorreu.

A Figura 3 mostra a relação entre defeito, falha e erro.

Figura 3 – A relação entre defeito, falha e erro



Fonte: Figura original de (Nelson, 1990), modificada pelo autor.

É importante mencionar que um defeito pode ou não ocasionar uma falha que, por sua vez, pode ou não ocasionar um erro. Por exemplo, em um a condição que uma porta de um circuito integrado apresenta um defeito em uma de suas entradas, enquanto esta entrada não for utilizada, este defeito não se tornará uma falha. Se esta entrada for utilizada para realizar, por exemplo, a comunicação com outros circuitos, a falha pode ser ativada e, inclusive, gerar algum erro de comunicação. Por outro lado, a falha poderá ser mitigada (sinalizada ou corrigida) se houver redundância na implementação desta comunicação utilizando a duplicação de fios ou códigos de detecção/correção de erros embutidos no *hardware* e/ou *software* do circuito, por exemplo.

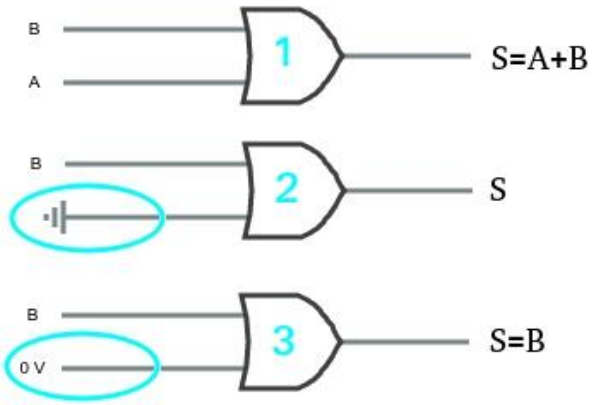
3.1.2 Modelos de Falhas

Conforme (LUBASZEWSKI; REIS, 2002), “Os testes só podem ser eficientes se falhas realistas forem modeladas a partir de mecanismos físicos e leiautes reais”. Considerar defeitos reais é, portanto, o caminho indicado para o sucesso da modelagem de falhas em qualquer aplicação.

Um dos modelos mais utilizados em testes estruturais de sistemas digitais é o modelo *stuck-at* (modelo de colagem). As falhas possíveis neste tipo de modelo são as de nível lógico 0 (*stuck-at-0*) e 1 (*stuck-at-1*). (LUBASZEWSKI; REIS, 2002)

Na Figura 4 abaixo, exemplifica-se o conceito e uso do modelo de falhas *stuck-at*.

Figura 4 – Exemplificação de *stuck-at-0*



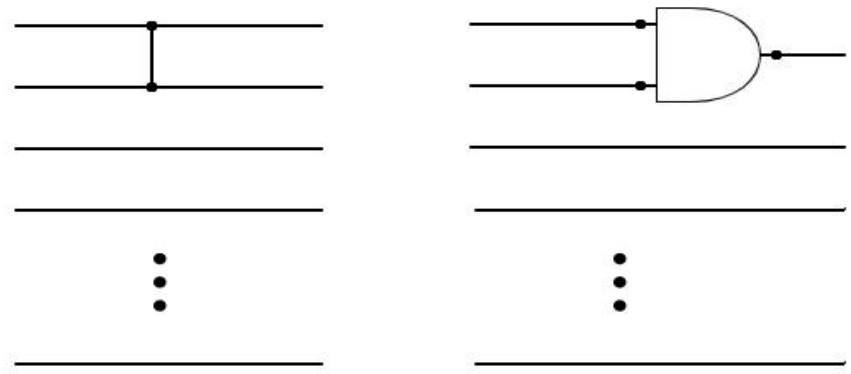
Fonte: Produzida pelo Autor.

Como é possível verificar na Figura 4, o circuito 1 é o *hardware* pretendido; o circuito 2 é a representação gráfica de um defeito de curto-circuito ocorrido entre a entrada A e o terra (tensão de referência); e, por fim, o circuito 3 apresenta a modelagem, no nível de porta, do comportamento do circuito na presença da falha *stuck-at-0* na entrada A da porta lógica. Se, no circuito 2, o defeito fosse de um curto-circuito da entrada A com Vdd, configurar-se-ia uma falha de *stuck-at-1*. O modelo *stuck-at* se aplica no nível de portas lógicas e, tipicamente, se limita às entradas e saídas destas portas.

Apesar do modelo *stuck-at* ser o mais comumente utilizado para o teste estrutural de circuitos digitais, outros modelos se somam a este, na medida que demonstram melhor representar o comportamento do circuito na presença de defeitos realistas. Por exemplo, no caso de barramentos e interconexões de uma maneira geral, circuitos-abertos (*opens*) e curtos-circuitos (*shorts*) representam os defeitos mais comumente encontrados, tanto em circuitos integrados, como em placas de circuito impresso. Circuitos abertos são tipicamente modelados no nível elétrico como resistores de alto valor de resistência. Já os curto-circuitos

são modelados no nível lógico como AND (*and-short*) ou OR (*or-short*) lógico das interconexões curto-circuitadas. A Figura 5 exemplifica o modelamento do curto-circuito entre duas linhas vizinhas de um barramento como um *and-short*. As características elétricas da implementação dos *buffers* que alimentam o barramento é que determinarão qual dos modelos, *and* ou *or-short*, deve ser o utilizado no modelamento de falhas.

Figura 5 – Modelo de falhas para interconexões aplicada a um barramento



Fonte: Produzida pelo Autor.

Não é incomum que modelos de falhas específicos demonstram-se mais adequados dependendo das particularidades da função ou da estrutura do circuito que se deseja testar. Incluem-se aí, por exemplo, modelos de falhas de memórias que, além de considerar comportamentos representados por *stuck-ats*, *opens* e *shorts*, agregam comportamentos lógicos do tipo acoplamento de células por vizinhança (ANDERSON, 1981). Outro exemplo muito comum é o de microprocessadores. Como estes são circuitos de grande complexidade do ponto de vista da função e do número de portas lógicas que os implementam, é comum simplificar o modelo considerando níveis mais altos de abstração na representação de falhas. Alguns exemplos são: o uso de modelos no nível de transferência de registradores ou de transição de estados, no caso de testes estruturais realizados de forma *off-line*; e, o uso de modelos no nível arquitetural ou algorítmico, no caso de testes funcionais realizados de forma *on-line*. Não existe "O" modelo nem "UM" modelo único que traga os melhores resultados para todos os casos. Na vida real, a combinação de vários modelos é a solução que termina imperando.

3.1.3 Simulação de Falhas

A simulação de falhas consiste, fundamentalmente, em injetar no circuito uma falha de maneira controlada considerando o modelo de falhas adotado, simular o circuito na presença desta falha e comparar o comportamento do circuito na presença e na ausência da falha.

Considerando um modelo de falhas, um determinado circuito e uma dada sequência de estímulos de teste, os passos envolvidos no processo de simulação de falhas são, basicamente, os seguintes (LUBASZEWSKI; REIS, 2002):

1 a lista de falhas do modelo (tipicamente *stuck-at*) é reduzida (*fault collapsing*) com base na remoção de falhas equivalentes (*fault equivalence*) e falhas que prevalecem sobre outras falhas (*fault dominance*);

2 o circuito sem falhas é então simulado para o próximo estímulo de entrada;

3 elege-se uma falha da lista restante de falhas ainda não simulada com o estímulo determinado no passo 2) e injeta-se a falha na descrição do circuito (*fault injection*);

4 simula-se o circuito com a falha e compara-se, ao final, o comportamento das saídas do circuito com e sem a falha simulada;

5 se houver diferença de comportamento entre as saídas do circuito sem e com a falha, a falha é marcada como detectada e é removida da lista de falhas (*fault dropping*); caso contrário, a falha permanece na lista;

6 se ainda houver falhas não simuladas na lista, retorna-se ao passo 3);

7 se ainda houver estímulos da sequência de teste não simulados, retorna-se ao passo 2).

A partir da própria descrição do procedimento acima é possível concluir que o objetivo da simulação de falhas é identificar quais falhas do modelo são detectadas quando o circuito é estimulado com uma sequência de vetores de teste. Este processo permite determinar a cobertura de falhas (percentual de falhas detectadas sobre o total de falhas do modelo) obtida pela sequência de teste, assim como quais falhas não são detectadas por esta sequência.

Encerrado o processo de simulação de falhas, para aquelas falhas não detectadas pela sequência de estímulos inicialmente utilizada, ferramentas de geração de vetores de teste podem ser utilizadas para encontrar estímulos que as detectem.

A partir da simulação de falhas é possível, igualmente, construir um dicionário de falhas (*fault dictionary*) que relaciona cada falha simulada aos vetores da sequência que a detectam, permitindo assim a realização do diagnóstico de falhas. Para a obtenção de um dicionário de falhas completo é necessária a desativação da função de *fault dropping* mencionada no passo 5) do procedimento acima.

3.2 Projeto visando o Teste

Integrar ao circuito, em tempo de projeto, funcionalidades normalmente realizadas pelo testador, com o intuito de obter melhores coberturas de falhas e/ou menores tempos de teste, são parte da disciplina denominada Projeto Visando o Teste (DfT, *Design for Test*) (LUBASZEWSKI; REIS, 2002).

As técnicas de DfT se dividem em 3 categorias básicas:

1 as que provêm o acesso eletrônico para melhoria da controlabilidade e da observabilidade de pontos internos do circuito (a técnica do *scan testing* é a mais difundida da categoria (EICHEBERGER, 1978);

2 as que transferem para dentro do circuito funções de aplicação de vetores de teste e/ou de análise das respostas de teste (as técnicas de geração pseudo-aleatória de vetores de teste e de análise de assinatura, ambas baseadas em LFSRs - *Linear Feedback Shift-Registers*, são as mais comumente utilizadas);

3 as que, de maneira intrínseca às funções já realizadas, agregam ao circuito informação e processamento utilizando redundância de *hardware* e/ou *software*, códigos detectores e/ou corretores de erro, etc, com a finalidade de testar o circuito constantemente durante o seu uso e atender requisitos de segurança e/ou confiabilidade da aplicação (o projeto de circuitos *self-checking*, assim como de circuitos e sistemas tolerantes a falhas, são representantes importantes desta categoria de técnicas (AMORIM, 2018)).

Das categorias de DfT acima mencionadas, são de particular interesse para este trabalho as técnicas que se enquadram nas categorias 2) e 3) acima. No rol de técnicas da

categoria 2), por exemplo, este trabalho faz uso de procedimentos de BIST (*Built-In Self Test*), ou autoteste integrado, dos sensores utilizados no protótipo aqui desenvolvido. Por outro lado, considerando a categoria 3), este trabalho utiliza largamente a redundância de *hardware* e de *software*, aliada a outras técnicas que serão descritas mais adiante.

3.3 Dependabilidade

Existem diversos tipos de aplicações que requerem variados níveis de segurança e de confiabilidade de seus componentes. Em aplicações como em um *desktop* utilizado por um usuário para edição de texto, por exemplo, o máximo de dano que pode causar um travamento do sistema é a perda de dados e a consequente perda de tempo do usuário para recuperá-los. De outro lado, se esse mesmo computador for utilizado por um sistema que realiza o monitoramento aéreo em um aeroporto, o seu travamento pode significar até a perda de muitas vidas caso a falha culmine com um acidente envolvendo uma ou mais aeronaves.

Estes exemplos evidenciam dois extremos do ponto de vista da criticidade da aplicação e, portanto, também do ponto de vista dos requisitos básicos de segurança e confiabilidade impostos a cada uma delas. Costumamos dizer que sistemas mais críticos visam a busca da dependabilidade. A dependabilidade, do inglês *dependability*, é o termo que define o nível de confiança e de qualidade de um serviço prestado por um determinado sistema. De acordo com (PRADHAN, 1996), as principais categorias de dependabilidade são:

- *Safety* (segurança de funcionamento): é a capacidade de um dado sistema de garantir a execução de forma correta de suas funções ou a paralisação destas de maneira a não propagar o dano aos usuários, sejam estes pessoas ou outros sistemas/dispositivos;
- *Security* (segurança de dados): é a capacidade de um dado sistema de garantir a proteção do usuário contra falhas ditas maliciosas, que tenham como alvo a privacidade, integridade, autenticidade e legalidade dos dados e informações, por exemplo;
- *Reliability* (confiabilidade): dentro de condições funcionais previamente definidas, esta é a capacidade de um dado sistema de atender estas condições integral e corretamente durante um período temporal pré-determinado, muitas vezes definido como tempo de missão; cabe ressaltar que a confiabilidade pressupõe que o sistema esteja operacional no início da missão;

- *Availability* (disponibilidade): é a capacidade de um determinado sistema se encontrar plenamente operacional num dado instante de tempo no futuro;
- *Mantainability* (mantenabilidade): é a medida da facilidade de realização de manutenções em um determinado sistema, quando este apresenta defeitos; quanto mais rápido for localizado e reparado o problema, sendo o sistema colocado em operação novamente, melhor será a sua mantenabilidade; sistemas facilmente testáveis tem baixos tempos de manutenção.

Em tempo de projeto e desenvolvimento de um novo sistema para uma aplicação crítica que exija algum nível de segurança (de funcionamento e/ou de dados), de confiabilidade, de disponibilidade e/ou mantenabilidade, diversas técnicas visando a dependabilidade podem ser empregadas (WENSLEY, et al., 1978) como, por exemplo:

- Previsão de falhas: estudo e estimativas sobre possíveis e prováveis falhas, e as consequências de sua ocorrência num determinado sistema;
- Prevenção de falhas: conceitualmente faz-se uso de métodos e tecnologias que impedem que uma falha ocorra ou seja introduzida no sistema;
- Validação: verificação se o sistema reage de forma correta quando ocorrem falhas.
- Tolerância a falhas: mesmo na ocorrência de falhas, o serviço é prestado como especificado utilizando mecanismos para a detecção de falhas, sua localização e confinamento, tratamento destas falhas, mascaramento, reconfiguração e recuperação do sistema.

Em termos de dependabilidade, conforme será visto no próximo capítulo, a solução para detecção de quedas apresentada nesta dissertação aplica conceitos de mantenabilidade, segurança de funcionamento e confiabilidade, e utiliza técnicas de tolerância a falhas, estas que serão apresentadas em mais detalhes na próxima sessão.

3.3.1 Tolerância a Falhas

A remoção e a prevenção de falhas podem não ser suficientes quando um sistema exige uma alta disponibilidade ou alta confiabilidade. Estes casos exigem que sejam

empregadas técnicas de tolerância a falhas, pois estas garantem o correto funcionamento mesmo na ocorrência de falhas. Sistemas tolerantes a falhas são baseados em algum nível de redundância. Utilizam mais componentes de *hardware* e/ou *software* e apresentam, consequentemente, maior consumo energético. Costumam coexistir com técnicas de remoção e prevenção de falhas. As mencionadas redundâncias, típicas de sistemas tolerantes a falhas, não dispensam que o sistema tenha sido desenvolvido, desde a origem, de maneira robusta, com base em processos confiáveis de verificação e validação de projeto, e a partir de componentes e processos de fabricação de reconhecida qualidade (LUBASZEWSKI; REIS, 2002).

As técnicas de tolerância a falhas seguem duas vertentes distintas: a primeira é baseada no mascaramento de falhas; e a segunda, baseada na detecção e localização do defeito, seguida da reconfiguração do sistema.

O mascaramento de falhas impede que estas se manifestem como erros, pois o comportamento incorreto do circuito na sua presença é confinado e corrigido antes de atingir alguma saída do sistema. O mascaramento geralmente emprega maior redundância do que a detecção, localização e reconfiguração, e tolera as falhas em tempo de execução. Existem diversas técnicas utilizadas para o mascaramento de falhas, as principais são: a replicação de componentes de *hardware*; o uso de códigos de correção de erros (ECC); a programação diversitária (n-*versions*); e, o uso de bloco de recuperação (AMORIM, 2018). No caso de falhas permanentes, a localização e o reparo destas são necessários para evitar que o sistema deixe de mascarar novas falhas que venham a ocorrer no sistema (LUBASZEWSKI; REIS, 2002).

O mascaramento é a estratégia preferencialmente empregada em sistemas críticos de tempo real, já que detectar, localizar e reconfigurar exigem a suspensão da aplicação para a execução de rotinas de manutenção. No caso desta segunda vertente, quatro são as etapas que devem ser percorridas pelas rotinas de manutenção (ANDERSON, 1981):

- Detecção de erros: os mecanismos utilizados são a replicação, testes de limites temporais, *watchdog timers* (temporizadores de cão de guarda), testes reversos, testes estruturais e de consistência, diagnóstico, teste de razoabilidade (de limites e de compatibilidades) e codificação (de detecção de erros, paridade, código de Berger…);
- Avaliação e confinamento: os mecanismos utilizados para impor limites e evitar a propagação de danos provenientes da falha são ações atômicas, operações primitivas auto

encapsuladas, isolamento de processos, hierarquia de processos, controle de recursos e regras como tudo que não é permitido é proibido;

- Tratamento da falha: emprega diagnóstico (localização da falha e, de forma precisa, da parte defeituosa) e reparo do sistema baseado no isolamento do componente defeituoso, no uso de partes sobressalentes (*spares*) e em reconfiguração do sistema;
- Recuperação de erros: costuma utilizar técnicas de recuperação por retorno (*backward error recovery*) e por avanço (*forward error recovery*), conforme detalhado a seguir.

Uma vez o sistema reparado, este deve voltar ao funcionamento em uma condição da aplicação de normalidade e correção de resultados. Portanto, a recuperação pressupõe o abandono do estado no qual a falha foi detectada (incorreto) e a transição para um estado em que o sistema esteja livre do risco de cometer erros. Duas são as possibilidades de recuperação:

- *forward error recovery*: conduz o sistema a um estado ainda não ocorrido desde a manifestação da falha/erro. Sua implementação é específica para cada aplicação e as perdas e danos devem ser cuidadosamente consideradas.
- *backward error recovery*: conduz o sistema ao estado imediatamente anterior à manifestação da falha/erro. Sua implementação é universal e pode ser baseada em conceitos como *checkpoints* (verificação), *audit trails* (pistas de auditoria), etc.

A recuperação é um processo complexo em um sistema onde existe um processamento distribuído e simples onde sua implementação se dá em um sistema com um único processo (JANSCH-PORTO, 1997). De uma maneira geral, técnicas de recuperação por retorno não devem ser utilizadas em sistemas de *real time*. Nestes, a técnica mais adequada costuma ser a de recuperação por avanço.

3.3.2 Redundância

Toda a técnica de tolerância a falhas utiliza algum tipo de redundância, seja de hardware, de software, de informação ou de tempo. Toda redundância impacta o sistema, seja no custo, no desempenho, nas dimensões físicas ou no consumo de energia. Logo, sua utilização deve ser pensada e comparada com outras opções cuidadosamente durante a fase de

desenvolvimento do projeto (LUBASZEWSKI; REIS, 2002). Dito isso, a redundância pode ser utilizada tanto para a detecção de falhas e manutenção do sistema, quanto para o mascaramento de falhas e erros.

Para que falhas sejam detectadas em um microprocessador, por exemplo, pode-se utilizar redundância de *hardware* e de *software* baseada em outro microprocessador idêntico, rodando o mesmo programa (duplicado) e sincronizado ao que será testado, além de um mecanismo de comunicação entre eles e com um elemento comparador dos resultados gerados por ambos. Uma falha será sinalizada quando houver desacordo de resultados entre um microprocessador e outro. A detecção fica assim, na maioria dos casos, assegurada. No entanto, falhas de modo comum que afetem, por exemplo, a alimentação (compartilhada) dos componentes, o relógio que os sincroniza ou o elemento comparador, podem conduzir a uma condição de acordo entre os microprocessadores apesar da saída do sistema ser incorreta (WENSLEY, et al., 1978).

A redundância de informação, por outro lado, aumenta o número de bits utilizados na representação e no processamento dos dados, de forma a embutir na aplicação códigos detectores, como a paridade, ou corretores de erro, como os códigos de Hamming (WENSLEY, et al., 1978).

Finalmente, a redundância temporal consiste em realizar repetidamente etapas do processamento de dados. A redundância temporal evita a utilização de hardware adicional, mas exige tempo adicional para a execução de uma determinada tarefa. Esta técnica é mais comumente utilizada quando o sistema apresenta alguma ociosidade na execução da tarefa, ociosidade esta que será preenchida com repetidas realizações da mesma tarefa seguidas da comparação entre os resultados obtidos em cada execução. A redundância temporal utilizando os mesmos dados de entrada é efetiva para a detecção de falhas transientes, que não perduram no tempo. Para a detecção de falhas permanentes, é necessário que a re-execução da tarefa seja realizada utilizando dados de entrada diferentes, mas cuja saída possa ser reconvertida, segundo alguma propriedade da função, antes da comparação entre as repetidas execuções. Por exemplo, no caso de uma rotina que multiplica dois operandos, as entradas originais podem ser processadas, em seguida estas mesmas entradas podem ser deslocadas de um bit à esquerda (multiplicadas por 2, cada uma) e depois novamente processadas pela rotina. O resultado da primeira execução pode ser então comparado com o resultado da segunda execução, este último deslocado de dois bits à direita (dividido por 4, portanto). Não havendo falha permanente, os resultados serão rigorosamente iguais. Caso contrário, serão diferentes.

Como a redundância de hardware e de software são predominantes na solução que apresentamos neste trabalho, é importante que aprofundemos um pouco mais a discussão sobre ambas a seguir.

3.3.2.1 Redundância de *hardware*

Conforme já mencionado, a redundância de *hardware* é baseada na utilização de múltiplos componentes em um sistema. O mais comum é que estes componentes sejam idênticos, mas há casos de uso de componentes diferentes executando rigorosamente a mesma função. Em qualquer caso, a redundância de *hardware* pode ser dita passiva (ou estática), ativa (ou dinâmica) ou híbrida.

O simples mascaramento de falhas, por exemplo, é um tipo de redundância passiva, pois não realiza nenhuma outra ação de identificação, nem remoção da falha. Essa técnica visa evitar o erro enquanto as falhas não se acumulam e até que não seja mais possível mascará-las. A técnica conhecida como TMR (*triple modular redundancy*) é um caso típico de redundância de hardware passiva (KHATRI, 2020). Nesta, todos os elementos, tipicamente idênticos, executam a mesma tarefa e o resultado é determinado por votação (LYONS, 1962), que pode ser uma votação por valor médio ou por maioria (LYONS, 1962). Normalmente a implementação de um votador é simples, já que ele não é utilizado para determinar qual módulo é o discordante, e tem baixo custo computacional. Dito isso, o votador é o ponto mais frágil dessa cadeia e, portanto, se for um elemento de baixa confiabilidade, todo o sistema terá uma baixa confiabilidade. Para garantir uma alta confiabilidade do votador é possível triplicar o próprio votador, construí-lo com base em componentes de alta confiabilidade e/ou realizar a votação via *software*.

A redundância ativa, por sua vez, não trabalha com mascaramento de falhas. Pressupõe ações adicionais de detecção, localização e recuperação, realizando assim o reparo de módulos do sistema. Tipicamente, a redundância ativa é a solução de preferência em aplicações que toleram permanecer num estado de erro durante um curto intervalo de tempo (enquanto realiza-se a manutenção e a recuperação) e/ou exigem longa vida útil do sistema (LUBASZEWSKI; REIS, 2002).

Por fim, a redundância híbrida mescla a redundância passiva e a redundância ativa. É, portanto, baseada em mascaramento e também na identificação e substituição do módulo com

falha. Um bom exemplo seria o uso de TMR ou NMR (generalização do TMR, com N>3) (LUBASZEWSKI; REIS, 2002), acompanhado de módulos sobressalentes (*spares*), com um votador com a capacidade de identificar o módulo faltoso e, havendo diagnóstico de falha permanente no módulo, de isolar o módulo defeituoso e reconfigurar o sistema para que um módulo sobressalente tome seu lugar.

3.3.2.2 Redundância de *software*

A simples duplicação de rotinas em *software* que rodam, uma após a outra, no mesmo *hardware*, é uma estratégia comum que funciona de maneira a suprir a demanda exigida nadetecção de falhas transientes.

No entanto, esta mesma abordagem apresenta sérias limitações para a detecção de falhas permanentes, já que rotinas idênticas resultarão em resultados idênticos entregues pelo *hardware* afetado que as roda.

Por outro lado, rotinas idênticas, que rodam em paralelo em réplicas diferentes do mesmo *hardware*, podem ser úteis e conduzirem à detecção de falhas transientes e permanentes nos respectivos *hardwares*.

A grande desvantagem de usar exatamente o mesmo *hardware* e a mesma rotina se encontra na sua limitada capacidade de detecção de falhas de modo comum, aquelas que afetam recursos de *hardware* e de *software* compartilhados entre as réplicas, como a alimentação, o relógio, etc.

Uma técnica muito conhecida que ajuda a contornar esta dificuldade é a diversidade de codificação, ou a programação de n-versões. A programação diversitária, ou diversidade de codificação, é a implementação de diversos programas diferentes, preferentemente desenvolvidos por equipes diferentes, para a resolução de um mesmo problema mas visto de maneira distinta, sob ângulos ligeiramente diferentes, pelos programadores. O resultado será tanto melhor quanto mais independente uma solução for da outra. Esta técnica, que pode ser igualmente usada na prevenção de falhas, pode ser utilizada em todas as fases do desenvolvimento de um programa. Na prática, os programas rodam de maneira independente e seus resultados são comparados uns aos outros ao final do processamento. Um sistema de votação opta pelo resultado provido pela maioria e o disponibiliza como a saída oficial do sistema.

O conceito de programação diversitária pode ser estendido às réplicas de *hardware*. A título de exemplo, a função executada por um filtro de sinal poderia ser implementada por software em um microprocessador, ou por um conjunto de módulos de memória, somadores e multiplicadores contidos em *hardware* específico para o processamento digital de sinais, ou ainda por um circuito analógico composto por amplificadores operacionais e componentes passivos, como resistores e capacitores. Para efeitos de comparação das saídas das 3 versões que, de maneira distinta, implementam a mesma função, o módulo analógico teria que ter sua saída convertida a digital por um conversor analógico digital. Assim poderia se construir um TMR diversitário para a função filtro com uma versão em software, outra em *hardware* digital e outra em *hardware* analógico.

À luz do exposto no Capítulo 2 e neste, no próximo capítulo apresentaremos a solução proposta neste trabalho para a detecção de quedas. Discutiremos os compromissos assumidos, considerando custos, dimensões físicas, consumo de energia, segurança e confiabilidade do sistema, compromissos estes que nortearam as escolhas de projeto em termos de componentes de *hardware* e de *software*, assim como das técnicas de tolerância a falhas integradas no dispositivo prototipado.

4 SOLUÇÃO PROPOSTA

O Capítulo 2 apresentou e discutiu diversas alternativas para a implementação de dispositivos de detecção de quedas. Como visto, soluções dedicadas tendem a ser mais seguras do ponto de vista funcional e menos demandantes de atenção e habilidades do usuário idoso. Por essas razões, optamos por uma solução baseada em *hardware* dedicado, que integra sensores específicos, microcontrolador com *software* dedicado à aplicação e comunicação para acesso ao serviço de nuvem. Estes elementos serão descritos na primeira metade deste capítulo.

O Capítulo 3, por sua vez, foi dedicado ao estudo de métodos de teste, implementados em *hardware* e *software*, que permitam assegurar a capacidade do sistema dedicado de detectar falhas, incluindo métodos para a detecção *off-line* de falhas, para a detecção *on-line* de erros e para a tolerância a falhas baseada em redundância de componentes ou na re-execução de funções críticas. Na segunda metade deste capítulo, apresentaremos os mecanismos de teste e tolerância a falhas implementados no nosso dispositivo detector de quedas de forma a impactar positivamente a dependabilidade final do sistema e da aplicação.

4.1 Componentes da Solução Dedicada:

Nesta seção descreveremos os componentes de *hardware* e de *software* escolhidos para compor nossa solução e abordaremos as razões pelas quais esses foram os escolhidos.

4.1.1 Sensores

Entre as diversas alternativas disponíveis de sensores citadas em 2.1, optou-se pelo acelerômetro e giroscópio MPU6050[6] (INVENSENSE, 2013). Ele contém, no mesmo encapsulamento, um giroscópio e um acelerômetro de 3 eixos. Este modelo possui uma resolução de 16 bits e é equipado com um *Digital Motion Processor* (DMP), que executa cálculos complexos antes de entregar o valor medido pelos sensores ao microprocessador.

[6] Disponível em: https://www.cdiweb.com/products/detail/mpu6050/422200/.

Isso ajuda a reduzir o consumo de energia do sistema, este que consome apenas 2 mA/H. Portanto, neste quesito, o sensor escolhido é compatível com implementações vestíveis.

O MPU 6050 tem a capacidade de executar diversas leituras por segundo (aproximadamente uma a cada 50 ms: frequência de 20Hz) e se comunica com o microprocessador por meio de um barramento I2C.

Outro fator determinante na escolha deste componente foi a possibilidade de realização de auto-teste integrado, este que é executado individualmente, em cada componente mecânico e elétrico que compõe o chip.

O auto-teste dos sensores é utilizado para a detecção de defeitos no dispositivo. Estes podem ser de natureza mecânica ou elétrica. O resultado deve ser lido acessando os registradores internos do dispositivo, como descrito na tabela da Figura 6:

Figura 6 – Pacote para leitura dos valores do MPU-6050

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0D	13	XA_TEST[4-2]			XG_TEST[4-0]				
0E	14	YA_TEST[4-2]			YG_TEST[4-0]				
0F	15	ZA_TEST[4-2]			ZG_TEST[4-0]				
10	16	RESERVED		XA_TEST[1-0]		YA_TEST[1-0]		ZA_TEST[1-0]	

Fonte: *datasheet* do MPU-6050 (INVENSENSE, 2013)

O autoteste do giroscópio é realizado acionando componentes internos do sensor que, por sua vez, movem a massa de prova do sensor por uma distância equivalente a um *Coriolis* de força (INVENSENSE, 2013). Quando há este deslocamento, uma alteração ocorre na saída do valor do sensor. Este valor é utilizado para verificar se o sensor responde corretamente.

Essa resposta é dada pela equação:

$$Resposta\ do\ autoteste = \frac{Saída\ do\ acelerômetro\ com\ o\ autoteste\ habilitado}{Saída\ do\ acelerômetro\ com\ o\ autoteste\ desabilitado} \quad \text{Eq. (01)}$$

Existe um valor pré-definido de calibração de lote da fábrica, que é obtido a partir da seguinte expressão:

$$Mudança\ de\ valor\ da\ calibração\ do\ autoteste\ de\ fábrica(\%) = (STR - FT)$$ Eq. (02)

em que:

$$FT = Valor\ setado\ de\ fábrica\ do\ trim\ no\ autoteste.$$

$$STR = Resposta\ do\ autoteste$$

Esse valor é gravado na Flash do microprocessador e é utilizado durante as rotinas de teste *off-line*, como veremos mais adiante neste mesmo capítulo.

De maneira semelhante ao realizado com o giroscópio, também é possível realizar o autoteste do acelerômetro. A equação utilizada para a obtenção da resposta do auto-teste (STR) é a própria Eq. (COUTINHO, 2012) dada acima.

Os valores de calibração de fábrica são aqueles que constam na Figura 7:

Figura 7 – Equações utilizadas no auto-teste do acelerômetro do MPU6050.

When performing accelerometer self test, the full-scale range should be set to ±8*g*.

$$\begin{cases} FT[Xa] = 4096 * 0.34 * \frac{0.92}{0.34}^{\left(\frac{XA\_TEST-1}{2^5-2}\right)} & \text{if XA\_TEST} \neq 0. \\ FT[Xa] = 0 & \text{if XA\_TEST} = 0. \end{cases}$$

$$\begin{cases} FT[Ya] = 4096 * 0.34 * \frac{0.92}{0.34}^{\left(\frac{YA\_TEST-1}{2^5-2}\right)} & \text{if YA\_TEST} \neq 0. \\ FT[Ya] = 0 & \text{if YA\_TEST} = 0. \end{cases}$$

$$\begin{cases} FT[Za] = 4096 * 0.34 * \frac{0.92}{0.34}^{\left(\frac{ZA\_TEST-1}{2^5-2}\right)} & \text{if ZA\_TEST} \neq 0. \\ FT[Za] = 0 & \text{if ZA\_TEST} = 0. \end{cases}$$

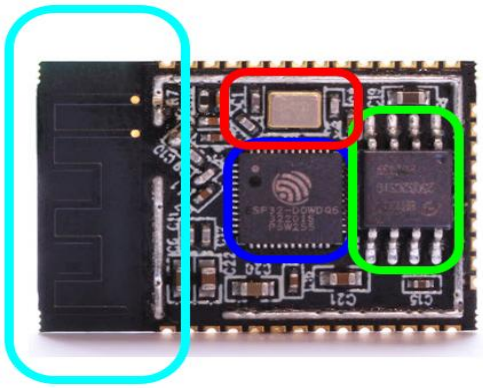
Fonte: *datasheet* do MPU-6050 (INVENSENSE, 2013)

Com os valores da Figura 7 é possível determinar se o acelerômetro está funcionando de forma adequada.

4.1.2 Microcontrolador

O microcontrolador escolhido foi o ESP32 (ESPRESSIF, 2022). Seu *hardware* mínimo é construído a partir de uma placa de circuito impresso, incluindo uma antena interna com tamanho de 25,5 mm x 10,8 mm, tornando possível o projeto de um dispositivo realmente vestível. O ESP 32, Figura 8, é composto de um microprocessador *LX6 de 32 bits ®Dual-Core Xtensa* (em azul escuro na Figura 8), memória *flash* em chip separado, variável de 1, 4, 8 ou 16 Mb dependendo da versão (em verde), rádios *Wi-Fi* e *bluetooth* incorporados no processador, cristal de *clock* (em vermelho na Figura 8) e, por fim, a antena dos rádios (na imagem abaixo, em azul claro, é embutida no chip, mas pode ser externa).

Figura 8 – Estrutura interna do ESP 32



Fonte: modificada pelo autor a partir de (ESPRESSIF, 2022)

Os rádios *Wi-Fi* e *Bluetooth* BLE (baixa energia), fazem com que o ESP suporte atualização remota de *firmware* e, entrando no modo *deep-sleep*, ambos são desligados e o ESP consome menos de 1 mA/h.

Entre outras razões, a escolha do microcontrolador ESP 32 foi feita considerando que deveria haver redundância no processamento de dados. Cada ESP 32 possui dois núcleos internos e independentes, e um terceiro processador dedicado à execução das rotinas de interrupção internas e externas ou até implementações *low-energy* durante momentos em que os núcleos principais estão em modo *sleep*.

4.1.3 *Software*

Do ponto de vista de *software*, o sistema é dividido em 4 grupos de rotinas: rotinas de testes dos sensores e barramentos, rotinas de comunicação e de validação dos rádios, rotina principal de detecção de queda e de testes *on-line* e rotina de interrupção, conforme abaixo:

- as rotinas de teste e autoteste dos sensores e barramentos visam o ajuste e a realização dos testes nos sensores e nas conexões físicas entre os sensores e os microprocessadores;
- as rotinas de comunicação e de validação dos rádios tem o objetivo de enviar a um servidor os resultados obtidos nas outras rotinas, ao mesmo tempo que testam a conexão *Bluetooth*, com a rede *Wi-Fi*, com a internet e com o servidor de nuvem;
- as rotinas de leitura dos sensores, processamento e tomada de decisão constituem o código principal da aplicação e são responsáveis por detectar se a queda ou outro evento ocorreu com o usuário do dispositivo e também por detectar eventuais falhas que se manifestem *on-line*; e,
- por fim, existe a rotina de interrupção, que têm a função de acordar o microprocessador de seu estado de *deep-sleep* para economia de energia, ou de retirá-lo de uma situação de *looping* na execução do código da aplicação.

Estes grupos de rotinas serão apresentados em mais detalhes a seguir, incluindo informações detalhadas sobre testes *off-line*, testes *on-line*, votação e mascaramento de erros.

4.1.4 Servidor da Nuvem

O ambiente em nuvem usado em nossa solução é o *Google Firebase*®[7]. Este foi escolhido por várias razões. Trata-se de um banco de dados em tempo real[8] , que tem a capacidade de executar internamente funções como consulta e análise de dados (reduzindo a necessidade de processamento local). Em sua API para desenvolvimento de *software* em celular, tem uma função que sinaliza quando um dado é modificado, assim consumindo apenas 2 *bytes* caso registre-se uma queda ou envie-se sinais entre o aplicativo e o microprocessador. O *Google Firebase*® suporta aprendizado de máquina nativo no servidor,

[7] Disponível em: https://firebase.google.com/?hl=pt-br.
[8] Com baixa latência em ações de leitura e escrita. (FIREBASE, 2022)

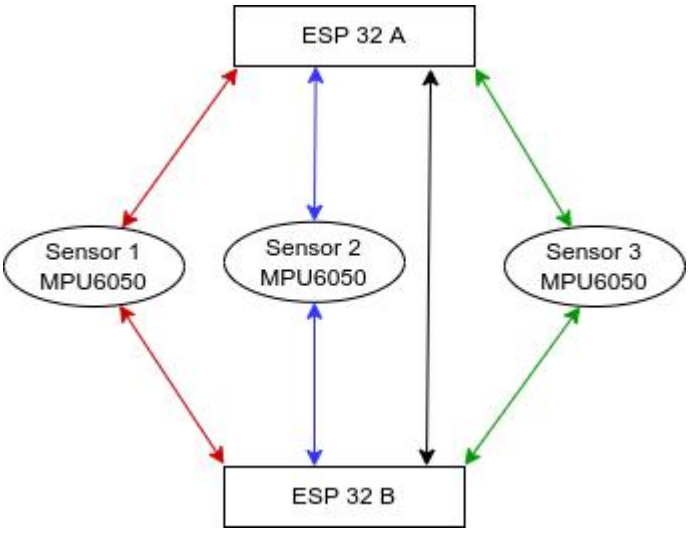
além de ofertar o uso de até 1 Gb de dados gravados no banco e 10 Gb de operações de *download* + *upload*, no pacote gratuito.

4.2 Mecanismos de Teste e Tolerância a Falhas

Embora defeitos e falhas façam parte da vida real dos sistemas eletrônicos, suas consequências podem ser minimizadas, ou até suprimidas, se esses sistemas forem projetados adequadamente. Portanto, conhecendo os possíveis defeitos e falhas antes que ocorram, recursos adicionais podem ser acrescidos ao sistema e ações podem ser tomadas para contornar suas consequências caso ocorram erros durante o seu uso. O resultado é um sistema que pode operar com segurança e tolerar erros, oferecendo resultados confiáveis, mesmo sob condições adversas.

Como visto anteriormente, as técnicas mais usadas para tolerância a falhas são baseadas em redundância, votação e mascaramento. Assim, procurando uma solução de baixo custo e com o menor consumo de energia possível, optamos por usar um sistema de votação com três sensores e dois microcontroladores, a fim de garantir alta confiabilidade do dispositivo vestível aqui proposto. A Figura 9 apresenta a abordagem de redundância de *hardware* utilizada no projeto.

Figura 9 – Redundância de *Hardware*



Fonte: Produzida pelo autor

Os três sensores são interconectados individualmente aos dois microcontroladores. Existem quatro barramentos I2C, cada um representado por uma cor diferente na Figura 9. Três barramentos atendem às conexões sensor-microcontroladores, e o quarto barramento I2C (Em preto) conecta os microcontroladores entre si.

Levando em consideração o sistema redundante proposto, um modelo de falhas misto, combinando falhas de tipo *stuck-at*, curto-circuito, circuito aberto e falhas funcionais, foi considerado no desenvolvimento dos mecanismos de teste e tolerância a falhas. Estes mecanismos conduzem a um alto nível de segurança e confiabilidade na operação. As falhas consideradas e discutidas na próxima seção são divididas em quatro grupos principais: falhas nos sensores MPU6050, falhas nos núcleos dos ESP32, falhas de comunicação e falha na alimentação.

Do ponto de vista da detecção de falhas e do mascaramento de erros, o *software* é composto por dois arranjos, um de testes *off-line* que roda durante a recarga de bateria, e outro da aplicação propriamente dita, que inclui os testes *on-line*, a votação entre sensores e ESPs, e o mascaramento de erros.

Conforme anteriormente mencionado, ambos arranjos são construídos a partir de rotinas de teste e autoteste dos componentes, de rotinas de comunicação e de validação dos rádios, de rotinas de leitura dos sensores, processamento dos valores medidos e tomada de decisão, e da rotina de interrupção por estouro do temporizador *watchdog*. A visão geral mostrando como estas rotinas estão organizadas no software é dada na Figura 10.

O fluxograma da Figura 10, inicia por configurar o preset do *watchdog timer*, de forma que um eventual estouro de contagem, devido ao travamento da aplicação, gere um pedido de interrupção.

Na sequência, verifica-se se o dispositivo se encontra em uso ou em carga de bateria. Durante a carga da bateria, são realizados os testes *off-line*, que serão detalhados na próxima seção. Tendo sido identificado, durante estes testes, algum defeito que impeça o dispositivo de ser utilizado, sinaliza-se o servidor na nuvem e um sinal luminoso piscante aparecerá no próprio dispositivo. Neste ponto o sistema fica em *loop* infinito, até que o sistema seja desligado e religado. No momento em que se sinaliza o servidor da nuvem, um elemento externo é acionado (médico, plano de saúde, hospital, cuidador, responsável…)

Por outro lado, quando a bateria atingir um nível pré-estabelecido de carregamento, o dispositivo poderá entrar em uso. Caso o carregamento seja interrompido antes de chegar-se a

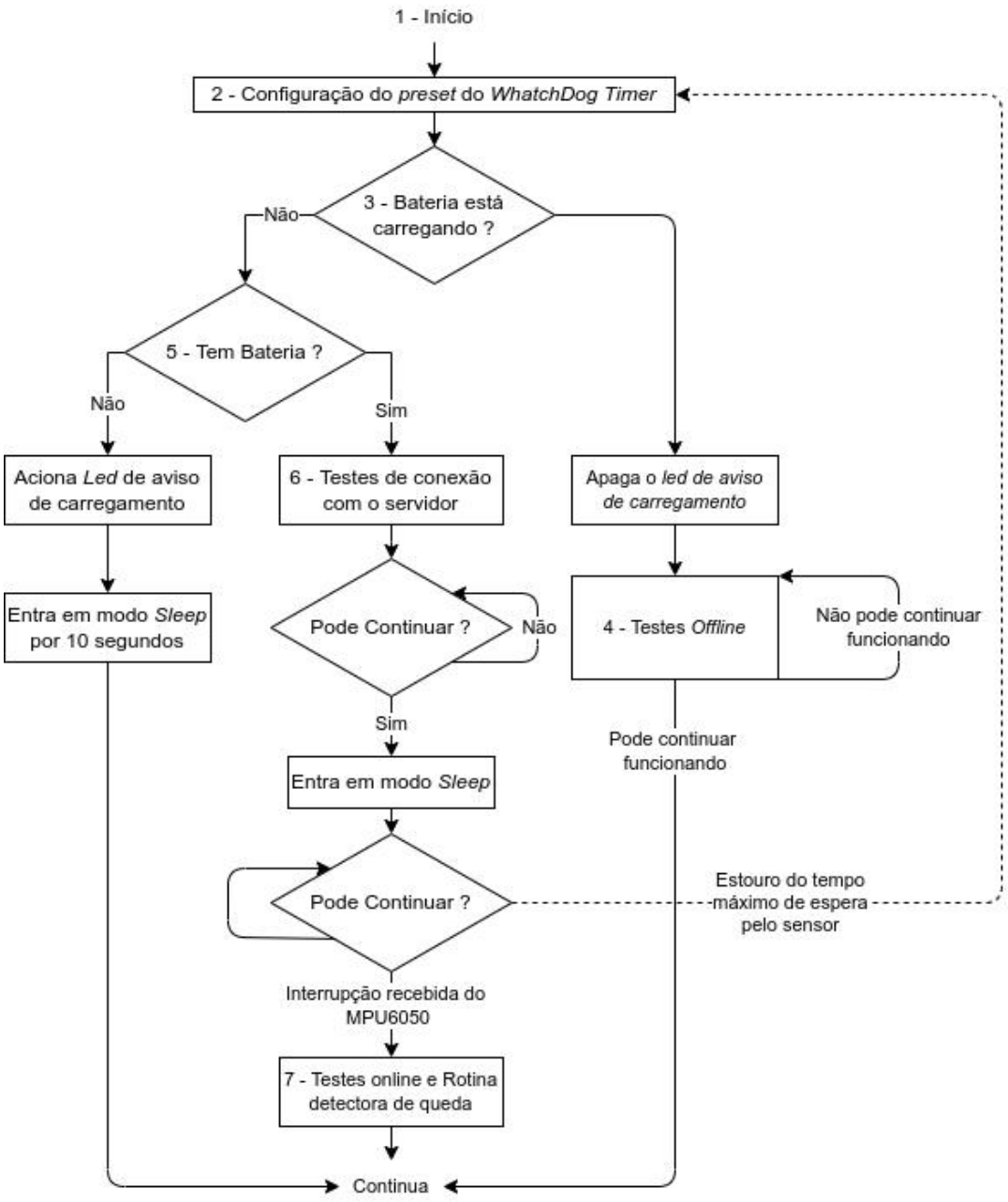
este nível, um *led* será energizado para indicar ao usuário que há a necessidade de completar a carga antes do uso do dispositivo para que o idoso saiba que o aparelho necessita de mais tempo carregando para que possa funcionar corretamente.

Neste caso, os ESPs entrarão em modo *sleep* para economizar energia e terão seu despertar (*wake up*) programado para 10 segundos para realizar uma nova avaliação da carga da energia e de entrada nos outros blocos da programação.

Quando o dispositivo estiver apto a entrar em uso, primeiramente será confirmada a funcionalidade de acesso ao servidor na nuvem, muito embora esta funcionalidade já tenha sido verificada nos testes *off-line* durante o carregamento da bateria. A partir daí, ambos os ESPs entram em modo *sleep* e aguardam que ocorra uma interrupção advinda de um dos sensores ou, que ocorra o estouro do *watchdog timer*. O estouro do temporizador é o mecanismo utilizado pelo dispositivo para que, em não havendo interrupção dos sensores, ainda assim envie um sinal de *keep alive* através do teste da conexão com o servidor na nuvem. Por outro lado, uma vez ocorra uma interrupção por parte de algum dos sensores, a rotina de detecção de queda iniciará e será executada de forma concomitante ao teste *on-line* dos microprocessadores e dos sensores, além dos próprios mecanismos de votação e mascaramento de erros.

As rotinas indicadas na Figura 10 serão descritas em mais detalhes posteriormente neste capítulo.

10 – Visão geral do *software*: estruturação das rotinas



Fonte: Produzida pelo autor

4.2.1 Falhas nos Sensores, Microcontroladores e Alimentação

Os sensores MPU6050 usados se comunicam com os microprocessadores através do barramento I2C. Nesse cenário, é possível identificar diferentes fontes de erros. São elas: ângulo(s) incorreto(s) (leitura válida, mas o sensor não está calibrado, ou o sensor está com

defeito), leituras incorretas ou não válidas (devido a defeitos no MPU6050), e pacotes de dados corrompidos (devido a defeitos nas linhas do barramento I2C ou fatores externos ao circuito). A queima do sensor não está contabilizada, pois a rotina de verificação de erro no pacote de dados engloba a não resposta do sensor com um *timeout* de 65 ms. Este número foi arbitrado pois o tempo de leitura máximo especificado é de 50 ms.

Durante o carregamento da bateria, são realizados testes *off-line* para garantir que os MPU6050s estejam calibrados, que funcionem de maneira correta e que as interconexões da placa de circuito impresso não apresentem falhas. O recurso de autoteste dos sensores é utilizado para calibração e para garantir que os mesmos encontram-se em bom estado operacional.

Os microprocessadores, por sua vez, podem apresentar vários problemas enquanto em uso, sendo os mais comuns aqueles que causam o travamento durante a execução de um código de programa. Normalmente é utilizada a técnica de *watchdog timer* (estouro de temporizador) para identificar a existência de problemas de processamento ou de código, e para reinicializá-los.

Outro ponto a ser destacado é que foi definida a utilização de um processador de núcleo duplo para auxiliar o *watchdog timer* a resolver possíveis bloqueios. Portanto, se um dos núcleos trava de uma maneira que a rotina de interrupção do *watchdog timer* não consegue lidar, o núcleo secundário assume como principal. O núcleo travado é logicamente isolado dos outros núcleos e não participa mais de futuras rodadas de leitura dos sensores, como veremos mais adiante. A situação de isolamento lógico só poderá ser revertida quando, por ocasião do próximo carregamento da bateria, os testes *off-line* puderem reverter a falha ou defeito que gerou o erro (por um simples *reset* ou pela substituição da parte defeituosa do dispositivo).

Como cada um dos 2 ESP32s usados possui seu próprio rádio *Wi-Fi* e *Bluetooth*, nosso dispositivo vestível explora a redundância natural para garantir a comunicação externa com o serviço em nuvem. A capacidade de cada microcontrolador para acessar a rede *Wi-Fi* é testada durante o carregamento da bateria e constantemente durante o próprio uso de seus núcleos. Estando os dois rádios livres de falhas, um dos microcontroladores assume a posição de mestre e o outro, de escravo. Caso apenas um deles esteja sem falhas, o microcontrolador de cujo rádio encontra-se sem falhas será o mestre da comunicação.

Enquanto o dispositivo vestível estiver em uso, o ESP mestre é responsável por atualizar periodicamente a nuvem com as informações mais recentes do usuário. Tanto durante o teste *off-line*, quanto durante o uso do dispositivo, a comunicação entre os dois microcontroladores é preferencialmente feita via I2C, evitando assim um gasto desnecessário de energia na utilização dos rádios. Caso o canal I2C fique indisponível devido a uma falha de *stuck-at*, por exemplo, a comunicação entre os dois microcontroladores mudará para os recursos *Bluetooth*.

Uma situação comum com dispositivos portáteis é a falta de fonte de alimentação devido a algum problema não programado. As principais causas são: bateria insuficiente para enviar pacotes de dados via *Wi-Fi*; bateria mal conectada; ou interconexões danificadas entre a bateria e o restante do circuito. Como duplicar a bateria tornaria impossível obter um dispositivo pequeno o suficiente para caber em uma solução vestível, decidiu-se usar um supercapacitor como reserva de energia. Estas foram as características essenciais identificadas para esse supercapacitor adicional:

- Capacidade de envio de dados para a nuvem por uma duração mínima de 2 horas, caso a bateria seja desconectada;
- Auxiliar a bateria durante os envios (fornecendo corrente), assim podendo diminuir o tamanho da bateria;
- Ter uma dimensão física compatível com o sistema proposto;
- Habilitar uma carga rápida do sistema, caso seja necessário.

Mais detalhes sobre o super capacitor e o consumo do dispositivo vestível serão apresentados no capítulo sobre resultados experimentais.

4.2.2 Testes *Off-line*

Os testes *off-line* são realizados exclusivamente enquanto a carga do super capacitor encontra-se plena e o dispositivo está conectado à fonte de energia para o carregamento da sua bateria principal. O testes *off-line* são compostos pelos testes dos rádios *Bluetooth*, rádios *Wi-Fi*, pelo auto teste e calibração de cada sensor MPU 6050, pelo teste das trilhas de interconexão dos barramentos I2C (caso os sensores não respondam) e pela verificação da conexão com o servidor na nuvem. Quando o teste identifica uma parte defeituosa, uma *flag*

específica sinaliza que a respectiva parte está indisponível e que não deve ser utilizada durante o processamento da rotina principal detectora de quedas. O defeito é sempre sinalizado ao usuário com sinal piscante no dispositivo e com uma *flag* de erro no servidor. Isto ocorre, independentemente de o dispositivo, em função de seu *hardware* e *software* redundantes, poder continuar operando corretamente em virtude do isolamento lógico da parte defeituosa e/ou do mascaramento de valores intrínseco ao processo de votação utilizado para tolerar erros.

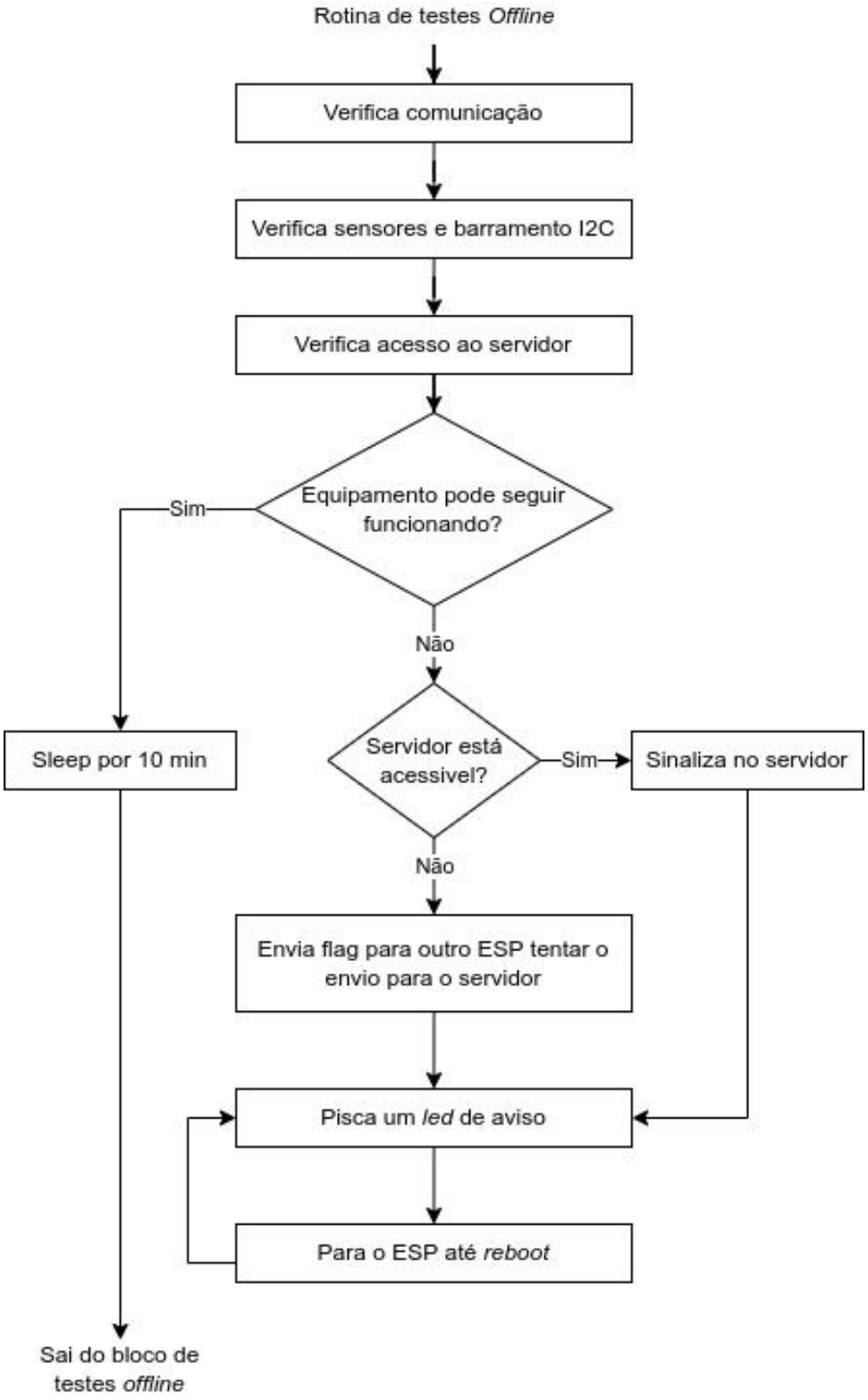
Dando início ao detalhamento das rotinas expressas à Figura 10, apresentamos, nesta seção, os testes *off-line*. Todos os testes aqui apresentados poderiam ser realizados durante o funcionamento do circuito, mas optou-se por realizá-los de maneira *off-line*, aproveitando a periodicidade da recarga da bateria e considerando duas razões principais:

1. para evitar o uso de fonte adicional de energia que suportasse maior consumo durante a aplicação e que garantisse a utilização do dispositivo sem a necessidade de recargas muito frequentes;

2. para manter o programa da aplicação o mais simples e direto possível, de forma a reduzir a probabilidade de ocorrência de falhas de travamento de *software*, típicas de códigos mais complexos, além de facilitar a atividade de manutenção do próprio código.

A sequência destes testes realizados *off-line* é apresentada na Figura 11.

Figura 11 – Sequência de testes *off-line*



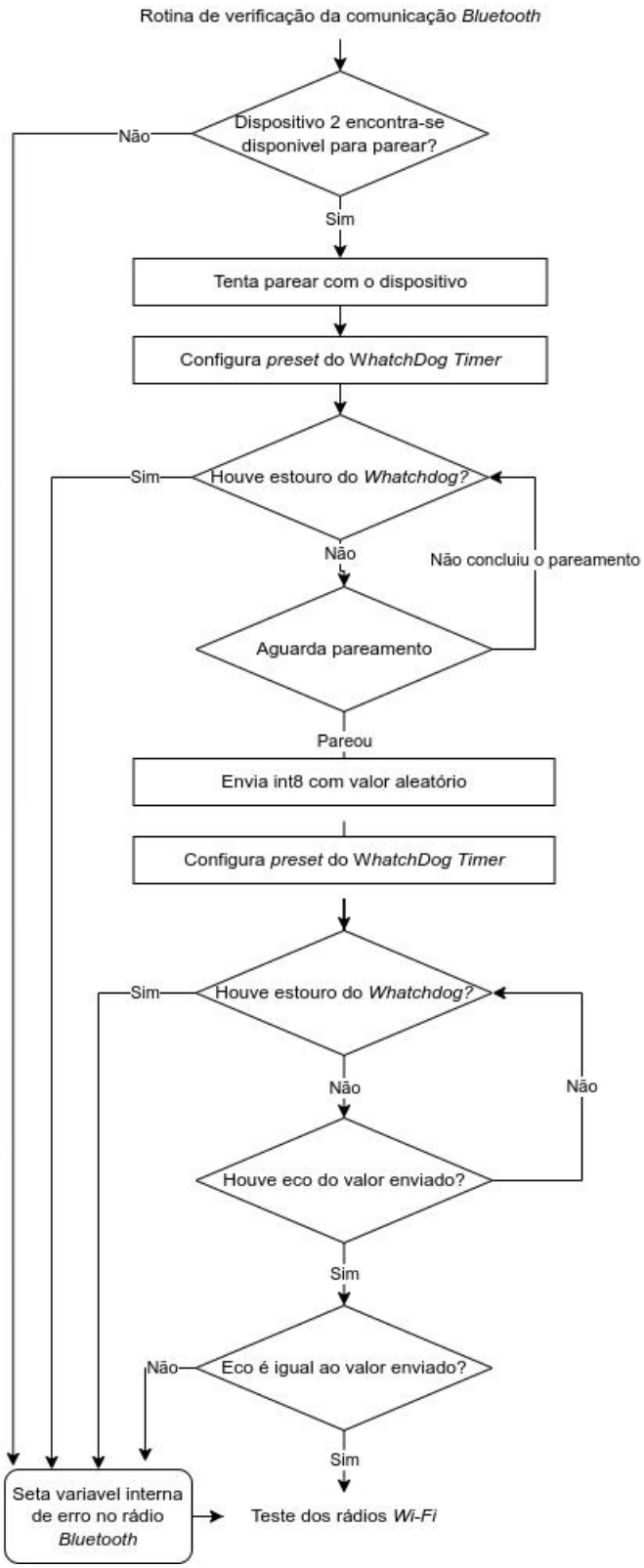
Fonte: Produzida pelo autor

O primeiro bloco da rotina consiste nos testes de comunicação. Estes testes primeiramente verificam os rádios *Bluetooth*. Os rádios *Bluetooth* são emergencialmente utilizados para comunicação entre os ESPs, caso o barramento I2C, que é o canal principal de comunicação entre eles, não esteja operacional.

O seu teste consiste simplesmente na tentativa de parear os rádios dos dois ESPs. O rádio do ESP A é pré-definido como mestre da comunicação via programação com o MAC correspondente. Ele procura parear-se com o rádio do ESP B e programa um tempo de *time-out* para o pareamento em seu *WatchDog Timer*. Caso o temporizador estoure, um dos dois rádios encontra-se defeituoso, já que não foi possível estabelecer a comunicação entre eles. Neste caso, uma *flag* é ativada para posterior envio da informação de erro de comunicação ao banco de dados na nuvem.

Caso receba o sinal do rádio do ESP B, uma variável com valor aleatório é enviada pelo rádio do ESP A que aguarda um eco como resposta. Neste caso, o temporizador volta a ser programado para monitorar a reação do rádio do ESP B. Se este temporizador estourar ou o valor recebido for diferente do enviado, um dos rádios encontra-se sem condições de comunicar-se. Caso tudo transcorra como esperado, sem estouros de temporizador ou erros de comunicação, o teste dos rádios *Wi-Fi* é realizado. A rotina de verificação do rádio *bluetooth* é vista na Figura 12.

Figura 12 – Rotina de testes realizados com os rádios *Bluetooth*



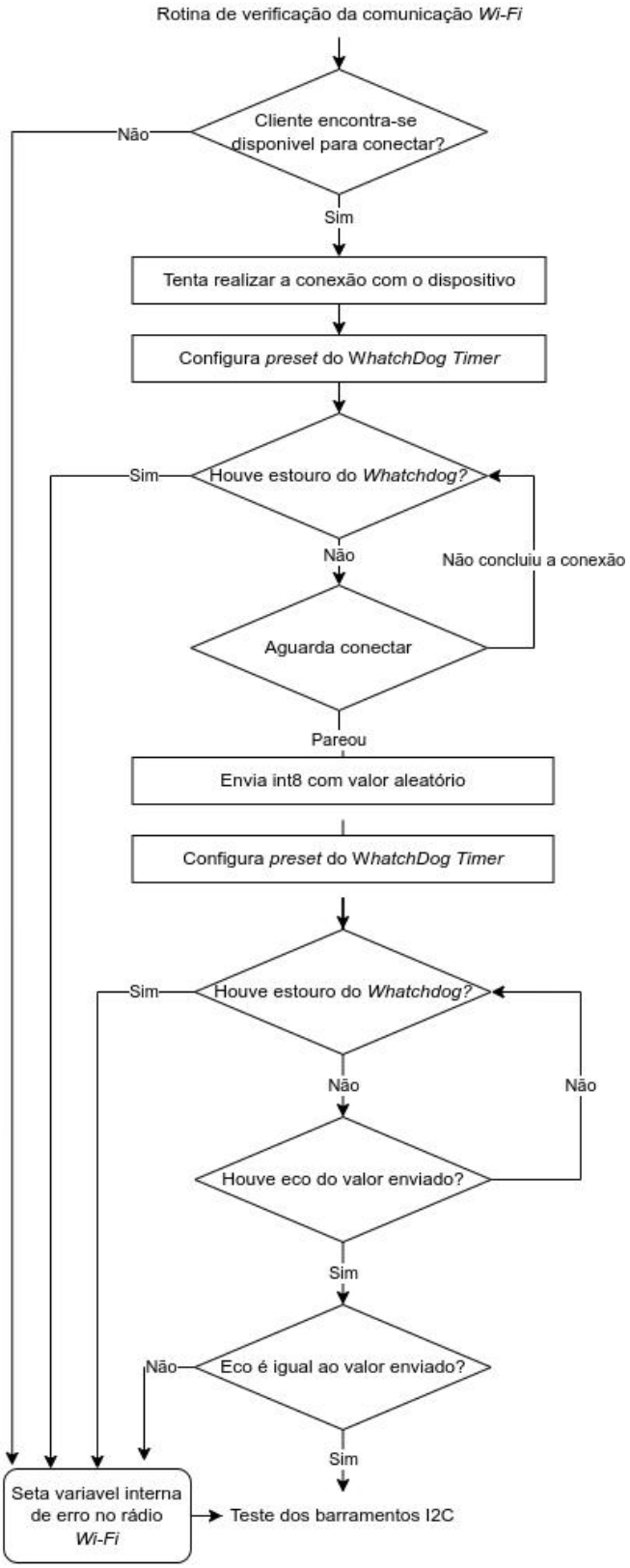
Fonte: Produzida pelo autor

A validação do funcionamento dos rádios *Wi-Fi* é feita de maneira muito semelhante à dos rádios *Bluetooth*. Como pode ser visto na Figura 13, esse teste basicamente conecta um ESP ao outro ESP via *Wi-Fi*. Caso o *watchdog timer* sinalize que a comunicação não se estabeleceu ou que não fluiu corretamente de um ESP a outro, uma *flag* de erro é ativada. Independentemente de haver sido ou não identificado um problema com os rádios *Wi-Fi*, todos os testes *off-line* que restam serão realizados em sequência, primeiro no ESP A e depois no ESP B.

Finalizados os testes dos rádios, passa-se ao teste/calibração dos sensores e ao teste dos barramentos I2C. Primeiramente, procura-se estabelecer a comunicação entre os ESPs e os sensores MPU 6050. Caso esta comunicação não prospere, então testa-se explicitamente os barramentos I2C. A ideia é diagnosticar se a falha que impediu a comunicação encontra-se no sensor ou nas trilhas do barramento.

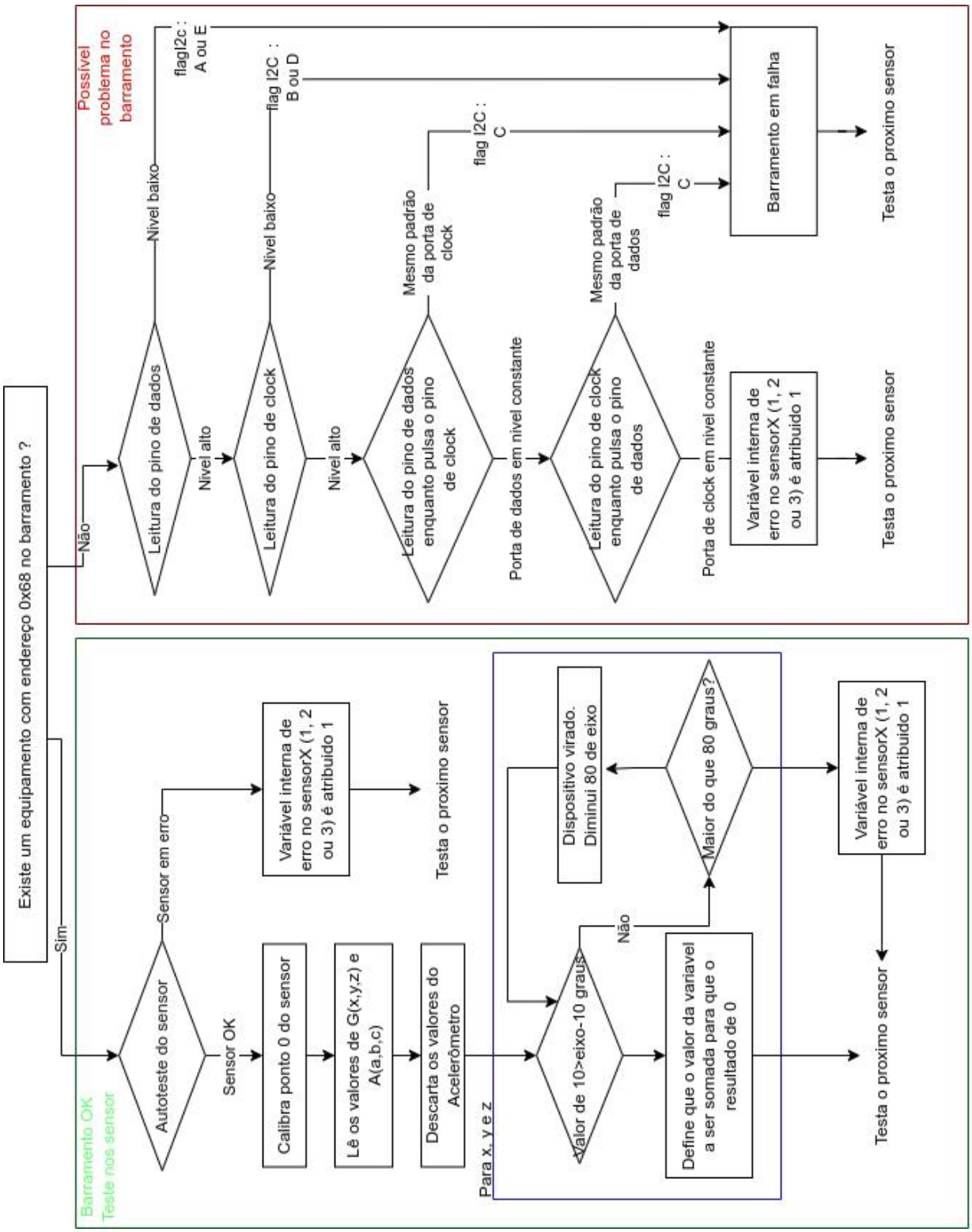
A Figura 14 apresenta o fluxograma da rotina que realiza estes testes, iniciando com a varredura dos MPU6050 a partir do endereço 0X68. Para o teste de cada dispositivo, é enviado no barramento um pacote contendo o endereço e o comando 0x00. Este comando solicita um *keep alive* do dispositivo endereçado. Cada pacote enviado tem um *timeout* de resposta (encontrado/não encontrado) de 65 mS. Caso o dispositivo endereçado retorne alguma informação válida, o contador de pulsos de *clock* interrompe a contagem e passa-se a realizar o auto-teste do sensor, assim como a sua calibração. Caso o dispositivo endereçado não retorne nenhuma informação válida e o contador de pulsos de *clock* estoure, existe algum problema no sensor ou no barramento I2C. Neste caso, passa-se a testar as trilhas do barramento deste sensor. Como cada microcontrolador ESP acessa cada MPU6050 através de um barramento I2C próprio (Figura 9), o teste serial dos sensores (primeiro testa-se o sensor 1, depois o 2 e por fim o 3) será realizado duas vezes, a primeira partindo do ESP32 A e a segunda, do ESP32 B. A duplicidade do teste é necessária para garantir a cobertura de falhas que venham a ocorrer em todos os barramentos I2C.

Figura 13 – Testes *off-line* dos rádios *Wi-Fi*



Fonte: Produzida pelo autor

Figura 14 – Teste e calibração dos sensores, e testes dos barramentos I2C



Fonte: Produzida pelo autor

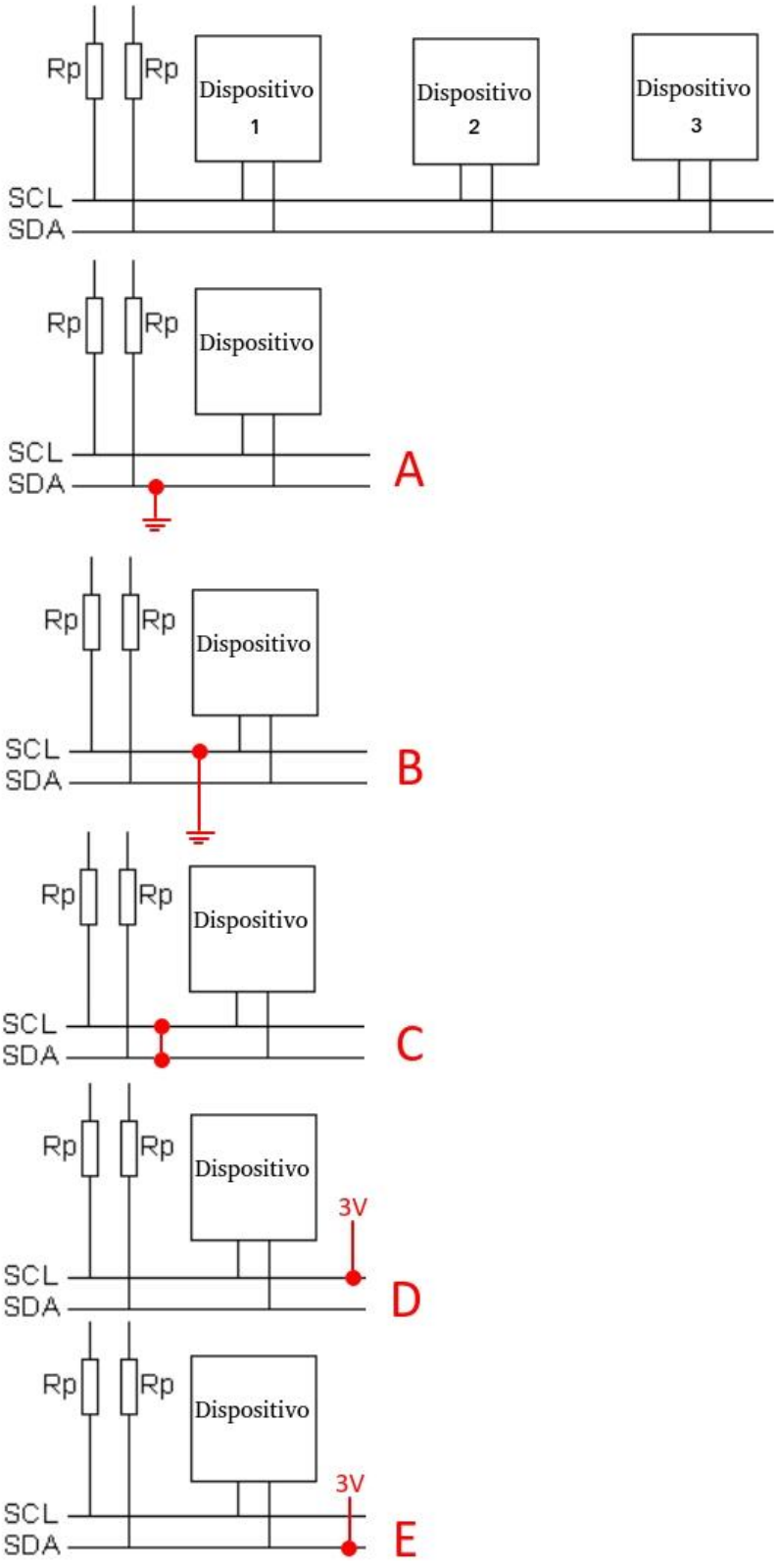
O autoteste do MPU 6050 retorna se o sensor está em erro ou não. Como mencionado anteriormente, este autoteste verifica apenas a integridade física dos componentes internos que compõem o acelerômetro e o giroscópio. Caso estes

componentes estejam em perfeito funcionamento, o ponto zero do sensor é calibrado. Para tanto, lê-se os valores do giroscópio e verifica-se se o valor apresenta variação de +-10 graus (máximo aceitável, pois foi construída uma base de carregamento que, em princípio, deixa o aparelho em 0 graus nos eixos x e y). Caso o usuário posicione o carregador em 90 graus em algum dos eixos, a leitura dos sensores pode resultar em valores nas proximidades de 90 graus. Neste caso, volta-se a testar o sensor descontando-se 80 graus (= 90 graus - 10 graus de tolerância). Se, ainda assim, o valor medido, mesmo descontando-se 80 graus, estiver fora da faixa de +-10 graus, então o sensor provavelmente está com defeito e esta situação será indicada em uma *flag* de falha do sensor. Se o valor medido encontrar-se no intervalo desejado, este será adotado como o ponto de angulação zero do sensor. Assim, o sistema registra o valor medido como o valor a compensar (somar ou subtrair) quando novas medidas forem realizadas.

Caso a comunicação entre o ESP e o MPU não tenha funcionado, nem o auto teste, nem a calibração do sensor são possíveis de realizar. É quando testa-se explicitamente o barramento I2C para identificar se o problema encontra-se nas trilhas que os conectam. Lembrando que o barramento I2C é composto por 4 conexões: alimentação de 3V, referência/terra, trilha de dados (*Data*) e trilha de relógio (*Clock*).

A verificação destas trilhas consiste basicamente em testar as interconexões entre os sensores e os microcontroladores para a detecção de falhas do tipo *stuck-at*, curto-circuito e circuito-aberto. A Figura 15 exemplifica os tipos de falhas que podem afetar as trilhas de *Data* (SDA) e de *Clock* (SCL). Este teste tem caráter estrutural e parte da observação do comportamento do barramento quando aplicados estímulos à porta do microprocessador que implementa o protocolo I2C, conforme veremos a seguir.

Figura 15 – Barramento I2C e possíveis falhas de interconexão: (A) SDA *stuck-at 0*; (B) SCL *stuck-at 0*; (C)and-short entre SCL e SDA; (D) SCL *stuck-at 1*; (E) SDA *stuck-at 1*.



Fonte: Produzida pelo autor

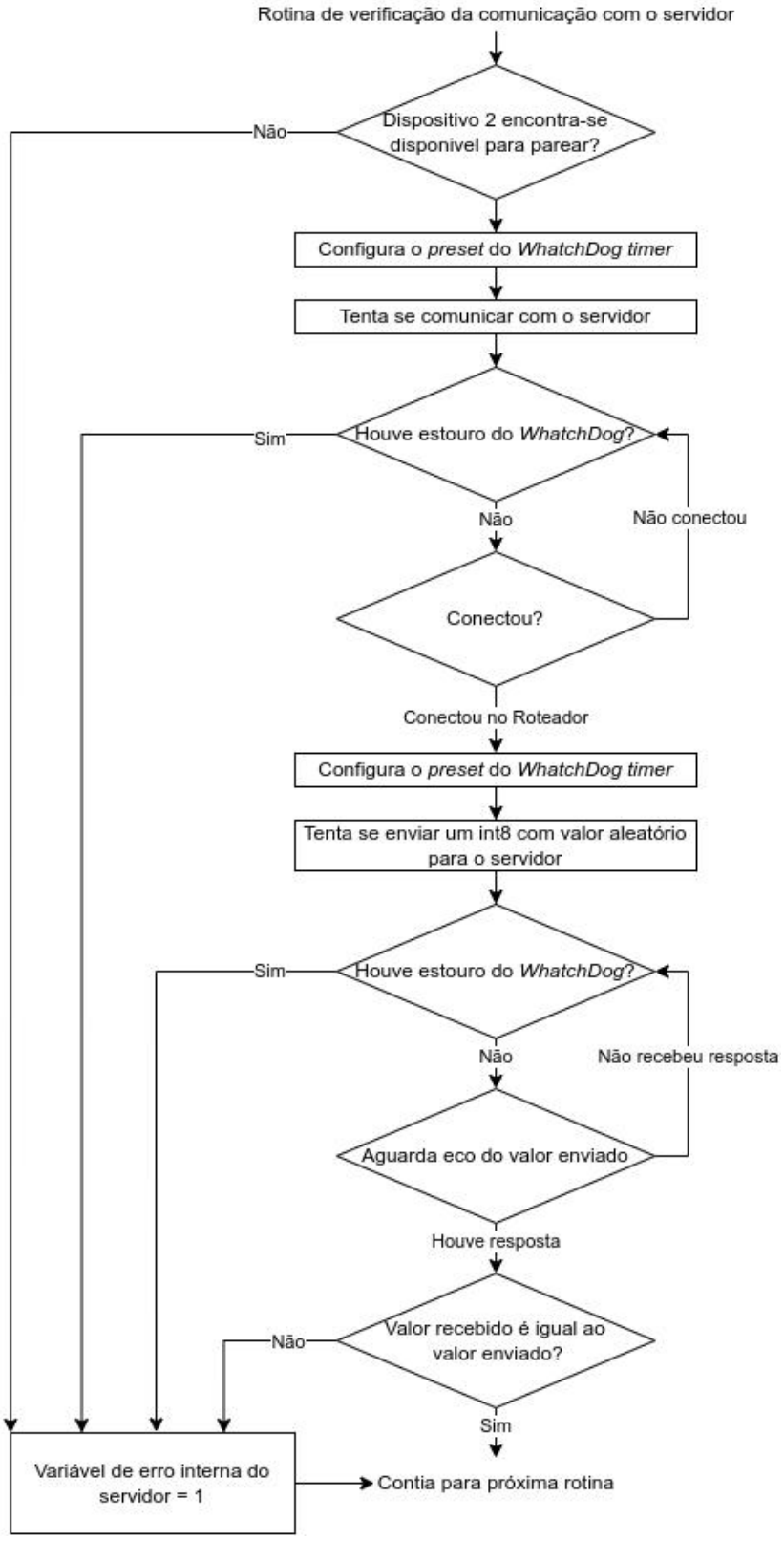
Na Figura 15, estão representados os dois tipos de *stuck-at* e o modelo de curto-circuito é o de *and-short*, em função do comportamento imposto pelos resistores de *pull-up* internos às portas dos ESPs que implementam os barramentos I2C. Em função da existência desses resistores de *pull-up*, qualquer circuito-aberto, em SCL ou SDA, do ponto de vista do ESP, corresponderia ao mesmo comportamento de um *stuck-at 1* na trilha correspondente. Muito embora haja a possibilidade que ocorra mais de uma dessas falhas simultaneamente, os testes utilizados para detectá-las são baseados na hipótese de falha simples.

Assim sendo, para o teste das falhas de tipo *stuck-at 1*(0), deve-se escrever em SCL e SDA o valor oposto ao da falha 0(1) e realizar-se a leitura dos valores escritos. Estes testes são realizados através da escrita e leitura na porta de dados e na porta de *clock*, conforme indicado à direita no fluxograma da Figura 14. São estes testes que detectam as condições de falha A, B, D e E da Figura 15.

Para o teste de curto-circuito entre SCL(SDA) e SDA(SCL), optou-se por aplicar um trem de pulsos no primeiro e ler-se o segundo. Caso SDA(SCL) acompanhe o sinal aplicado à SCL(SDA), as duas trilhas encontram-se em curto-circuito. Este procedimento detecta a condição de falha C da Figura 15 e também aparece indicado no fluxograma da Figura 14.

Por fim, o último teste realizado de maneira *off-line* é o teste de comunicação com o servidor que, conforme pode ser observado na Figura 16 e como era de se esperar, guarda muitas semelhanças com os testes dos rádios *Bluetooth* e *Wi-Fi* já apresentados. Basicamente esta rotina tenta conectar o dispositivo ao servidor na nuvem, via roteador, e testa a conexão. Do ponto de vista da comunicação, esta conexão é uma funcionalidade essencial para o dispositivo pois, caso qualquer defeito, falha ou erro ocorra, é no servidor que ficarão registrados e é de onde partirá a informação que permitirá a tomada de decisão, por parte do cuidador, com relação aos cuidados a prestar ao idoso portador do detector de quedas. Por conta de sua criticidade, este teste será repetido durante o uso do dispositivo .

Figura 16 – Teste de comunicação com o servidor



Fonte: Produzida pelo autor

4.2.3 Testes *On-line*

Os testes realizados concomitantemente com a aplicação são aqueles essenciais à garantia da segurança e confiabilidade contínua da função do dispositivo. Por tratar-se de um dispositivo vestível e que, portanto, deve ter baixíssimo consumo, devemos limitar os testes *on-line* ao menor número possível de verificações de componentes, sempre lembrando que todos serão periodicamente testados *off-line*. Testam-se, portanto, a comunicação *Wi-Fi* do dispositivo com o banco de dados do servidor na nuvem; testam-se os sensores e barramentos através de leituras redundantes (as partes defeituosas são isoladas, logicamente falando, e os valores medidos, com erro, não participam dos cálculos que levam à tomada de decisão); testam-se os diversos microprocessadores (quatro núcleos dos dois ESPs, no total) através de mecanismos de passagem de *token* e de estouro de temporizador (*watchdog timer*). Aliam-se a todos estes testes *on-line* os próprios mecanismos de tolerância a falhas baseados em votação e mascaramento de erros.

O primeiro destes testes *on-line* é o de comunicação com o servidor, este que já foi apresentado na seção anterior, juntamente com os testes *off-line*.

O segundo teste *on-line* realiza leituras redundantes nos sensores através dos seus respectivos barramentos I2C. São 3 os MPU6050 que compõem o sistema redundante e cada um contém um acelerômetro e um giroscópio, como visto. O microprocessador realiza três leituras consecutivas dos valores medidos pelo acelerômetro e pelo giroscópio, conforme mostra a Figura 17. Assim, os valores de leitura dos sensores preenchem um buffer de 18 posições, estas compostas por 3 leituras (sensor 1, 2 e 3) de dois valores (acelerômetro e giroscópio) para 3 momentos de tempo diferentes, separados por aproximadamente 1500 ms (leitura 1, leitura 2, leitura 3). A leitura de cada sensor individualmente respeita o tempo máximo de transmissão de dados de cada sensor, o que resulta em uma frequência máxima de 20 Hz.

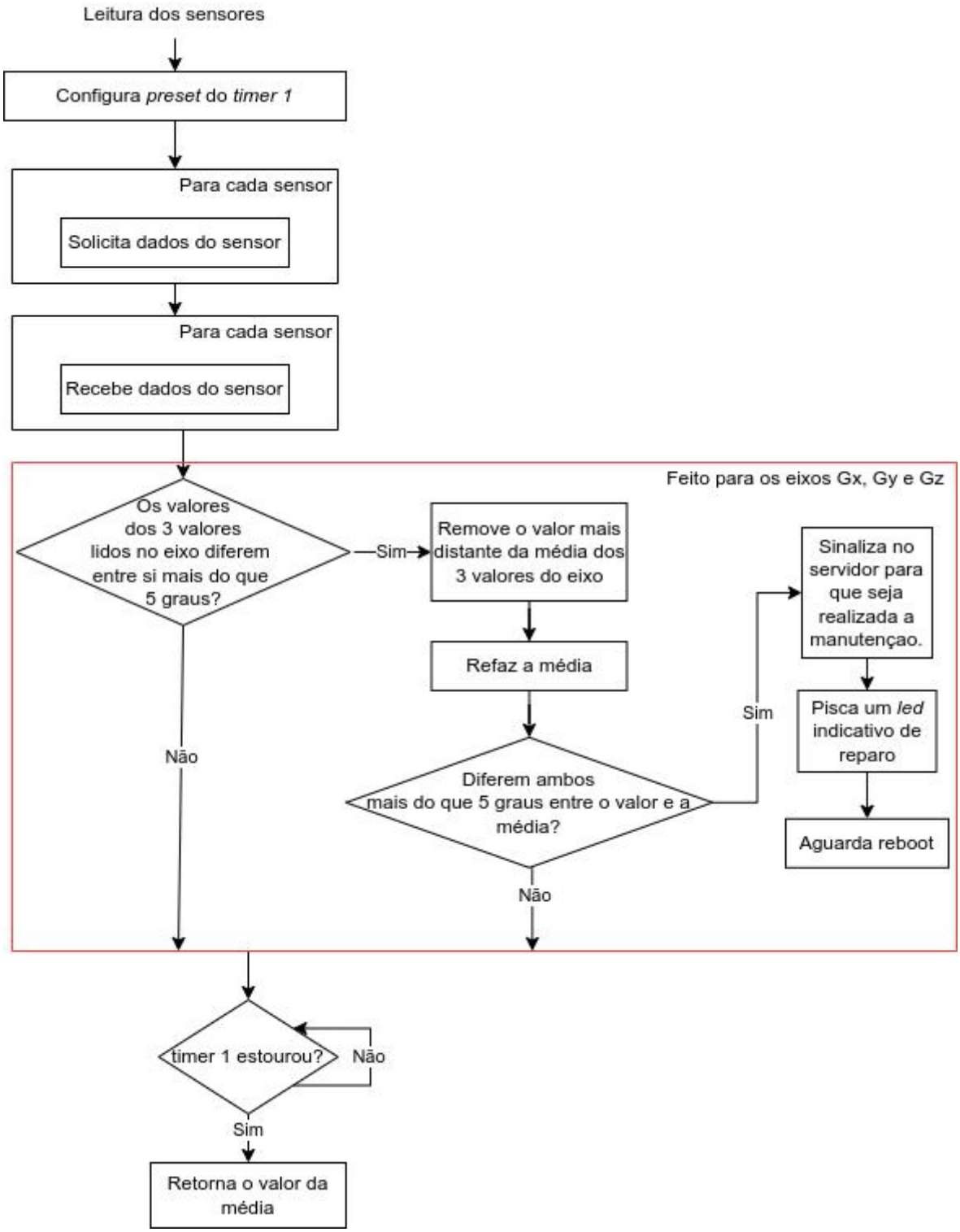
Independentemente da aplicação necessitar ou não da medida de ambas as grandezas físicas, no uso do MPU 6050 não há como ler a medida realizada pelo giroscópio sem também ler a realizada pelo acelerômetro. No entanto, como veremos no capítulo 5, no nosso dispositivo somente os valores medidos pelo giroscópio são utilizados para detectar o padrão de evolução de ângulos que caracteriza uma queda.

Assim, e dando sequência ao teste *on-line* baseado na leitura redundante dos sensores, o microprocessador calcula a média das leituras do giroscópio. As mesmas três leituras são realizadas em cada um dos três MPUs, que tem suas respectivas médias calculadas. Dessas leituras, resultam 3 médias referentes aos giroscópios. Um sistema de votação é aplicado para verificar se, das 3 médias de cada giroscópio, há alguma que divirja por mais de 5 graus das outras duas médias. Conforme resultado de testes de campo, uma diferença de 5 graus corresponderia a uma velocidade angular inatingível pelo idoso em qualquer movimento que estivesse realizando (1,70 rad/s ou 16,233 RPM) e, portanto, identificaria um sensor operando em condição de erro. Sendo identificada uma tal diferença, o respectivo sensor é “marcado” como defeituoso, sua média de valores medidos é rejeitada e o núcleo do ESP32 segue o processamento considerando exclusivamente a média da média de valores dos dois giroscópios restantes.

No próximo ciclo de leitura dos sensores, se um sensor específico estiver com o status “defeituoso”, este voltará a ser considerado no sistema de votação, mas com 5 leituras, conforme mostra a Figura 17. O fato de o sensor ter sido “marcado” como defeituoso em algum momento pode ter sido ocasionado por um problema de calibração (que deverá ser resolvido por teste *off-line* na próxima carga de bateria) ou por um falha transiente, por isso optou-se por não descartar definitivamente o sensor, mas por mascarar suas leituras como descrito, enquanto o erro persistir.

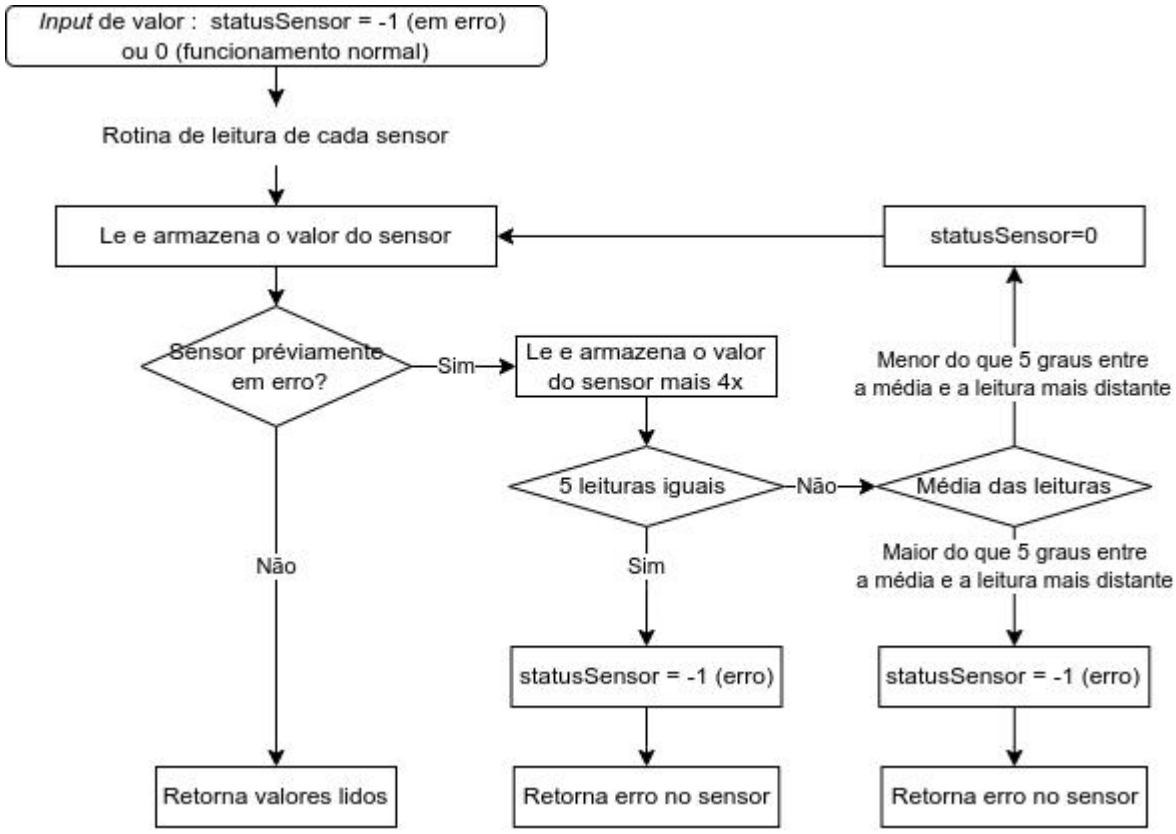
Na Figura 18, pode parecer estranho considerar que 5 leituras iguais do sensor resultem em marcá-lo como defeituoso. No entanto, é necessário ponderar que, conforme demonstraram os testes de campo, mesmo que o idoso permaneça "imóvel" durante leituras consecutivas dos giroscópios, diferenças superiores a 0,01 graus (essa é a resolução do sensor) necessariamente surgirão nas medidas. Portanto, tantas leituras consecutivas absolutamente iguais terminam por indicar uma condição de erro na operação do giroscópio, pois esse sensor gera um pequeno ruído na leitura dos eixos.

Figura 17 – Esquema de leitura e teste *on-line* dos sensores.



Fonte: Produzida pelo autor

Figura 18 – Rotina de leitura individual de cada sensor.



Fonte: Produzida pelo autor

Testada a comunicação com o servidor na nuvem e os sensores/barramentos, resta-nos testar os microprocessadores do sistema. Como mencionado anteriormente, a consequência mais comum advinda de falhas em microprocessadores é o travamento do código em execução.

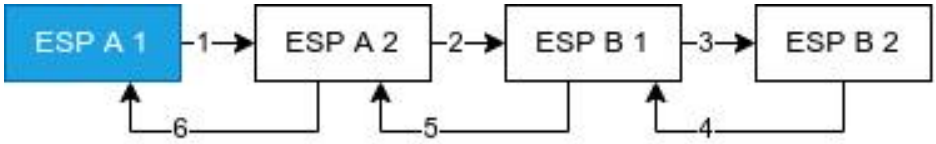
Considerando que no nosso dispositivo existem 2 microprocessadores (núcleos) em cada um dos ESP 32 utilizados, 4 núcleos microprocessadores precisam ser testados *on-line*. Os 4 núcleos rodam praticamente o mesmo *software*, exceto o núcleo mestre que, adicionalmente, roda a rotina que faz a gestão das leituras nos outros núcleos e centraliza a decisão final do dispositivo.

Basicamente, o teste dos quatro microprocessadores se ampara em um mecanismo de passagem de *token*, conforme ilustra a Figura 19, monitorado por *watchdog timers* que

sinalizam o travamento do *software* nos núcleos a partir do estouro de contagem dos temporizadores (Figura 21).

O esquema da Figura 19 apresenta o ESP A 1 (núcleo 1 do ESP A) como o mestre da comunicação entre os núcleos. O código que roda neste núcleo é muito semelhante ao dos outros três núcleos. A primeira diferença é que o núcleo mestre é o responsável pela votação final da sinalização ou não de uma queda, decisão que é tomada a partir dos votos recebidos de cada um dos outros núcleos operacionais e comunicada ao servidor na nuvem. A segunda diferença é que, considerando que o *token* avança na cadeia até o último núcleo e retorna ao mestre passando novamente pelos núcleos por onde passou anteriormente (Figura 20), o tempo limite programado em seu *watchdog timer* é quatro vezes maior que o tempo armazenado no *watchdog timer* do último núcleo da cadeia do *token* (ESP B 2). Pela mesma razão, o tempo limite programado no *watchdog timer* do ESP A 2 é três vezes maior que do ESP B 1 e o tempo limite programado no *watchdog timer* do ESP B 1 é duas vezes maior que no *watchdog timer* do ESP B 2, este que é o último núcleo da cadeia de *token*. A comunicação para a passagem de *token* e para a transmissão do voto ao mestre, acontecerá via barramento interno do próprio ESP quando se tratar de núcleos pertencentes ao mesmo ESP (ESP A 1 e ESP A 2; ESP B 1 e ESP B 2). Já a comunicação entre núcleos de ESPs diferentes (ESP A 2 e ESP B 1) é realizada via barramento I2C caso este esteja operacional. Caso contrário, a comunicação ocorrerá via rádio *Bluetooth* ou, em último caso, via rádio *Wi-Fi*.

Figura 19 – Esquema inicial de passagem de *token*: ESP A 1 é o mestre

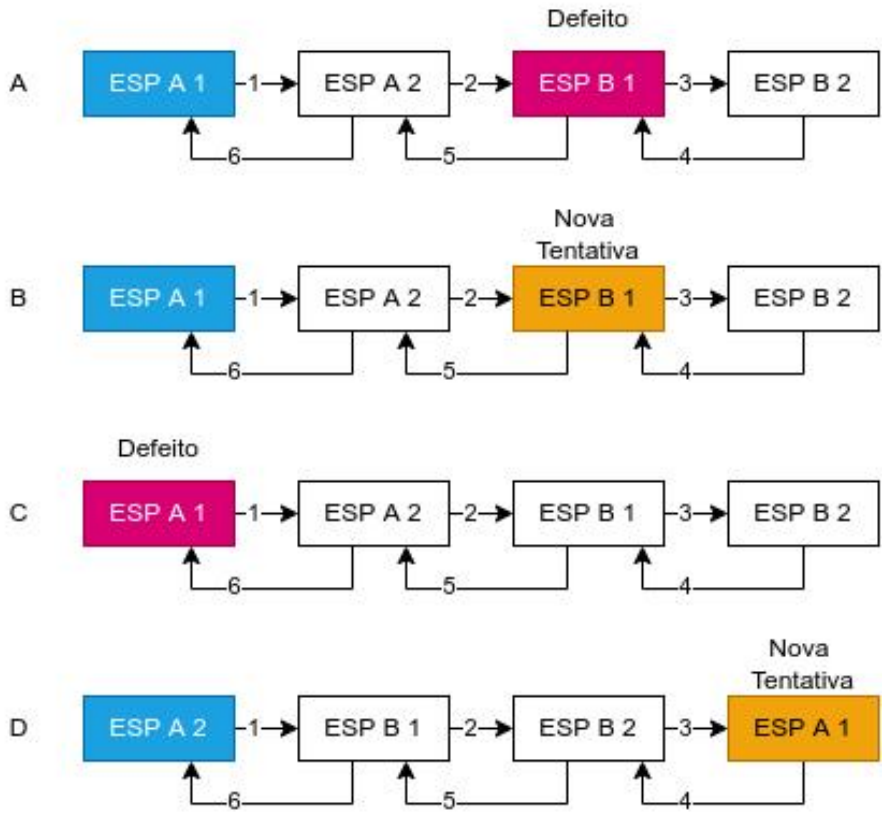


Fonte: Produzida pelo autor

Na inicialização do sistema, o núcleo ESP A 1 é arbitrado como mestre e a cadeia de passagem de *token* é aquela mostrada na Figura 19. Caso o *watchdog timer* de um núcleo estoure por falta de passagem do *token* do núcleo que lhe antecede na cadeia, este último será isolado logicamente e permanecerá assim até a próxima rodada de leituras, quando for efetivada uma nova tentativa de passagem de *token*. No exemplo da Figura 20 (a) e (b), em virtude do núcleo ESP B 1 não passar o *token* ao ESP B 2, o *watchdog timer* deste último estoura e força o mestre ESP A 1 a isolar logicamente o núcleo ESP B 1 supostamente com

defeito. Caso o mestre apresente defeito, respeitando a sequência demonstrada na Figura, o núcleo que o sucede na cadeia de passagem de *token* assumirá o seu papel dali por diante e isolará logicamente o antigo mestre. Na Figura 20, em (c) e (d), exemplifica-se o caso em que o papel de mestre é passado ao núcleo ESP A 2, por ausência de passagem de *token* pelo ESP A1.

Figura 20 – Situação de núcleo defeituoso



Fonte: Produzida pelo autor

O fluxograma da Figura 21 ilustra o *software* de monitoramento da passagem de *token* a partir do estouro do temporizador de *watchdog* de cada núcleo. O fluxograma considera o código programado no núcleo mestre ESP A 1.

Consideremos primeiro uma situação sem falhas, quando o *token* passa de núcleo a núcleo sem estouro do *watchdog timer*. Neste caso, o núcleo mestre toma a decisão de sinalizar ou não uma queda baseada nos votos de todos os outros núcleos do dispositivo. Durante o processamento, cada núcleo calcula o próprio voto e envia para o outro ESP via barramento I2C e para o núcleo interno do mesmo ESP via compartilhamento de endereço de

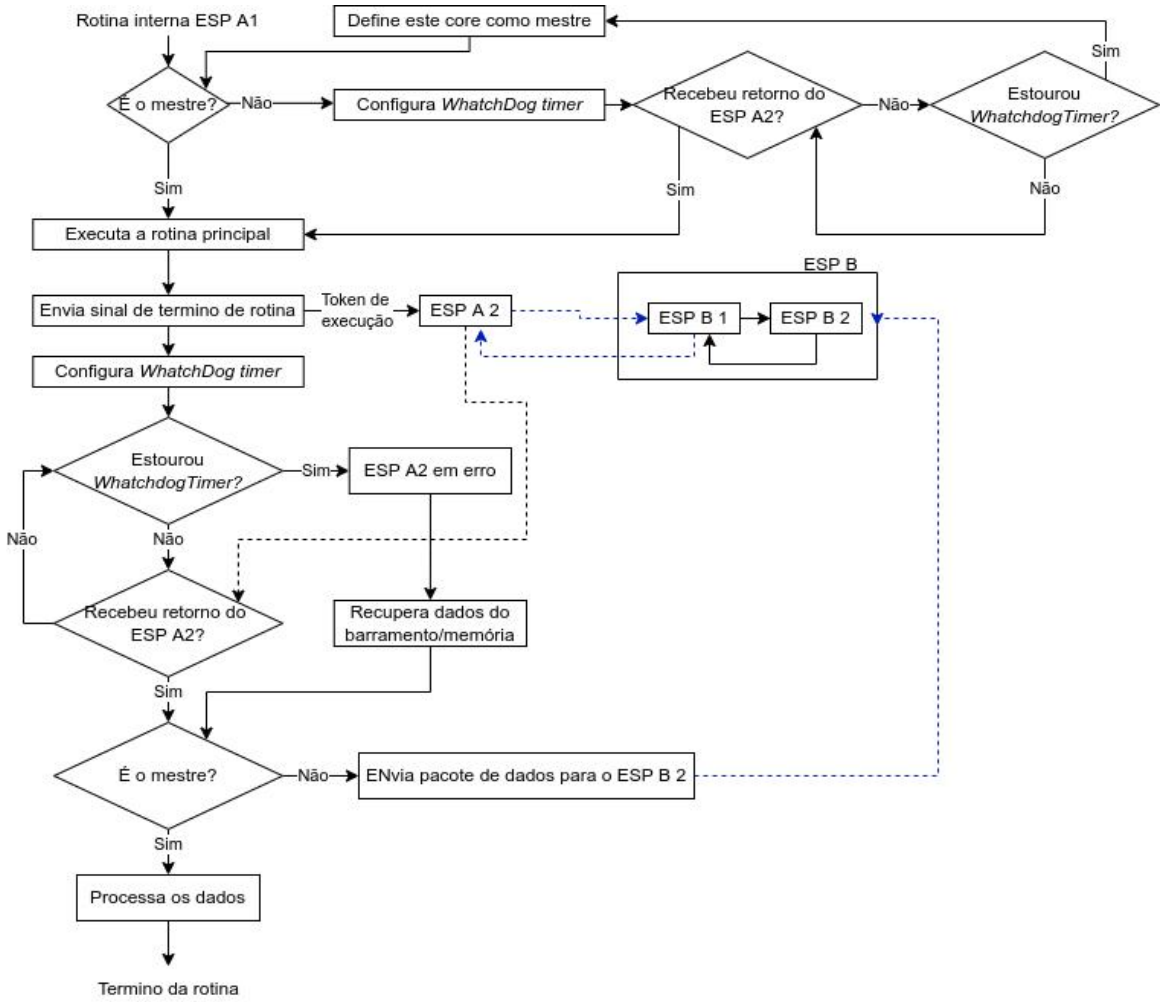
memória, e passa o *token* ao núcleo que lhe sucede na cadeia. Assim, seu sucessor, a seu tempo, também processa seu voto. Chegando ao final da cadeia, o voto de cada núcleo, juntamente com os votos dos que lhe sucedem, são computados no mestre, conforme indicado pelas setas inferiores nas Figuras 19 e 20. Encerrada a votação e comunicado o resultado ao servidor na nuvem, o dispositivo retorna ao estado de *sleep*.

Considerando agora uma situação de falha, por exemplo, no núcleo ESP A 2. Neste caso, o mestre ESP A 1 verá seu *watchdog timer* estourar e precisará resgatar os votos dos núcleos ESP B 1 e ESP B 2 enviados e não recebidos pelo núcleo defeituoso ESP A 2. Como o barramento I2C implementado entre os dois ESPs (assim como os rádios dos dois microcontroladores) possui um *buffer* de dados recebidos que pode ser acessado por ambos os núcleos do mesmo ESP, muito embora o núcleo ESP A 2 não tenha encaminhado os votos recebidos ao mestre ESP A 1, o próprio mestre, no estouro de seu *timer*, pode buscar a informação no *buffer* compartilhado.

Outra situação de falha que pode ocorrer, ilustrada na Figura 20 (A) e (B), é a do ESP B 1 apresentar defeito e a contagem do *watchdog timer* do ESP A 2 estourar. Neste caso, o ESP A 2 irá verificar, através do barramento I2C que comunica os dois ESPs, se o ESP B 1 chegou a depositar seu voto (e eventualmente também o voto do ESP B 2) no *buffer* do ESP B (a falha pode ter ocorrido após o envio dos votos) e, em caso afirmativo, retornará ao ESP A 1 a informação encontrada no *buffer* juntamente com o seu próprio voto. Caso não identifique no *buffer* o(s) referido(s) voto(s), o ESP A 2 retornará apenas o seu próprio voto ao mestre ESP A 1. De maneira similar, se o ESP B 2 apresentar defeito e a contagem do *watchdog timer* do ESP B 1 estourar, o ESP B 1 irá verificar se o ESP B 2 chegou a depositar seu voto no *buffer* I2C antes da falha ocorrer e, em caso afirmativo, retornará ao ESP A 2 a informação do ESP B 2 juntamente com o seu próprio voto. Caso não identifique no *buffer* o voto do ESP B 2, retornará apenas o seu próprio voto ao ESP A 2.

Finalmente, se o núcleo mestre ESP A 1 sofrer uma falha em meio ao processo, a rodada de leituras, processamento e votação não será concluída e, na próxima rodada, o seu núcleo vizinho assumirá o papel de mestre, conforme ilustram as Figuras 20 (C) e (D), e a Figura 21.

Figura 21 – O fluxograma da passagem de *token* do ponto de vista do núcleo mestre ESP A 1



Fonte: Produzida pelo autor

Considerando uma condição sem falhas, cada núcleo, a seu tempo, realiza a leitura dos sensores, calcula a média destas leituras, toma sua decisão com relação ao padrão observado para a evolução dos ângulos de movimento do usuário (conforme veremos na próxima seção na figura 22) e envia sua indicação de queda ou não-queda ao núcleo mestre, caso ele próprio não o seja. E, segundo o esquema descrito nas Figuras 19 e 21, envia o *token* ao próximo núcleo para que este realize exatamente o mesmo procedimento. Este procedimento é repetido até que todos os 4 núcleos tenham sido percorridos e tenham disponibilizado suas indicações ao núcleo mestre. Resulta de todo este processo um conjunto de 4 indicações independentes que são comparadas entre si pelo núcleo mestre e, havendo duas ou mais indicações de padrão de queda, a decisão do sistema será de registrar a queda no servidor em nuvem e de disparar

um alerta ao cuidador do idoso. Quando a queda é confirmada, o ESP A envia um sinal para o servidor via *Wi-Fi* e um sinal pelo barramento para que o ESP B realize o mesmo processo de sinalização, garantindo a redundância na comunicação com o servidor. Caso a queda não seja confirmada, o sistema é novamente colocado em modo *sleep* e só realizará novas leituras quando voltar a ser interrompido por algum dos sensores.

A Tabela 5 apresenta o comportamento do dispositivo em condições com e sem falhas nos núcleos. As condições com falhas tabeladas se limitam àquelas que ainda apresentam uma operação segura, ou seja, com pelo menos dois núcleos operando no sistema. Observa-se que a indicação de sem queda só ocorre quando a maioria dos núcleos chega à mesma conclusão. Em caso de empate na votação, em nome da segurança do usuário, a resposta do dispositivo é conservadora mesmo que a indicação de queda signifique um falso positivo.

Tabela 5 - Resultado oferecido pelo dispositivo em condições com e sem falhas nos núcleos

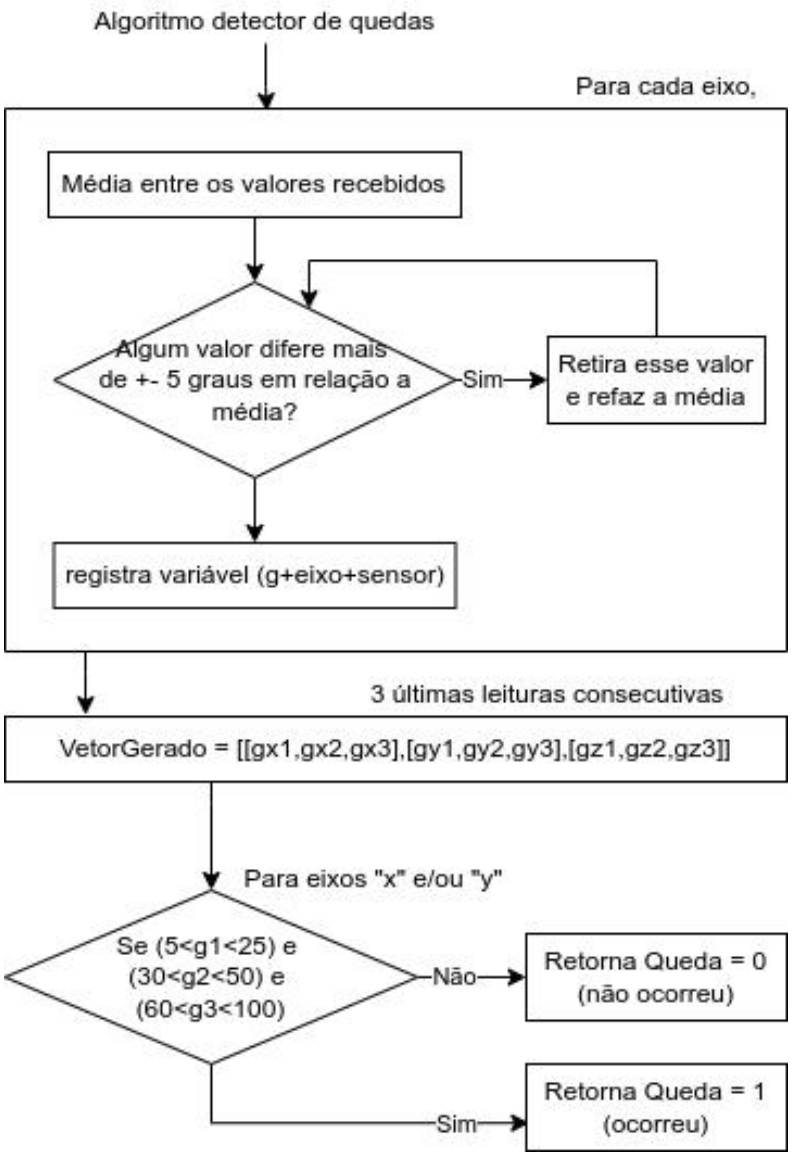
	Resultado com 4 núcleos ativos	Resultado com 3 núcleos ativos	Resultado com 2 núcleos ativos
4 resultados positivos para queda	Queda	Situação inválida	Situação inválida
3 resultados positivos para queda	Queda	Queda	Situação inválida
2 resultados positivos para queda	Queda	Queda	Queda
1 resultado positivo para queda	Sem queda	Sem queda	Queda
0 resultados positivos para queda	Sem queda	Sem queda	Sem queda

Fonte: Produzida pelo autor.

4.2.4 A Rotina de Detecção de Quedas e o Mascaramento de Erros

As rotinas de leitura dos sensores, já apresentadas anteriormente, aliadas ao processamento dos valores medidos e à tomada de decisão constituem o código principal da aplicação e são responsáveis por contribuir para a classificação da condição do usuário do dispositivo e a eventual detecção de quedas. A Figura 22 ilustra, sob a forma de fluxograma, a parte do código principal que processa os valores medidos e conclui sobre a condição do usuário, esta que, juntamente com a leitura dos sensores apresentada anteriormente nas Figuras 16 e 17, rodam em cada um dos 4 núcleos do dispositivo (ESP A 1, ESP A 2, ESP B 1 e ESP B 2) que compõem o hardware do nosso dispositivo.

Figura 22 – Processamento das médias de leituras dos sensores e de classificação da condição do usuário (em queda ou não)



Fonte: Produzida pelo autor

Neste cenário, o sistema valida os valores lidos por votação e trabalha com a média dos valores validados. As médias processadas para os 3 sensores são então comparadas com os ângulos alvo (15, 40, 80), para que seja verificado se a queda ocorreu. Este processo está descrito no fluxograma da figura 22. No capítulo 5, onde abordaremos os resultados experimentais deste trabalho, explicaremos a origem deste padrão (15, 40, 80) de evolução de ângulos que foi identificado como o padrão que caracteriza uma queda.

Conforme a Figura 17, os valores dos sensores 1, 2 e 3 são sequencialmente lidos e armazenados na memória do núcleo. Após cada leitura, obtém-se dados relativos aos eixos x, y e z para o giroscópio (Gx,Gy,Gz) e para o acelerômetro (Ax,Ay,Az). Conforme antecipado,

os valores fornecidos pelo acelerômetro não são considerados na nossa implementação de detector de quedas e, portanto, são descartados pela rotina.

Realizado um ciclo completo de leituras dos 3 sensores (com espaçamento de 50 ms entre leituras), tem-se, no caso dos giroscópios, (Gx1,Gy1,Gz1), (Gx2,Gy2,Gz2) e (Gx3,Gy3,Gz3). Para cada um dos eixos, calcula-se então a distância D entre pares de medidas de sensores diferentes, conforme:

D1 = |Gx1-Gx2|

D2 = |Gx2-Gx3|

D3 = |Gx1-Gx3|

Ordena-se as distâncias D1, D2 e D3, e, caso nenhuma exceda a 5 graus, a média simples de Gx1, Gx2 e Gx3 será o valor resultante do processo de votação das leituras dos 3 sensores.

Caso uma das medidas divirja em mais de 5 graus das outras duas, duas das três distâncias acima resultarão em um valor superior a 5 graus. A distância que não extrapola o limite máximo de 5 graus identifica as duas medidas concordantes. Suponha, por exemplo, que Gx2 se distancie em mais de 5 graus de Gx1 e Gx3. Neste caso, D1 e D2 resultarão maiores que 5 graus. Somente D3 terá valor inferior a 5 graus. Portanto, a média simples de Gx1 e Gx3 será o valor resultante do processo de votação para a medida do eixo x dos 3 giroscópios.

Quando as 3 distâncias resultarem fora do limite de tolerância, será emitido um alerta de que o sistema só pode continuar operando de maneira segura após a realização de manutenção nos MPU6050.

Esse mesmo procedimento é feito para os valores lidos nos outros eixos dos giroscópios (Gy,Gz). Ao término, os valores médios obtidos para cada eixo dos giroscópios são salvos na posição 0 de um vetor de médias. Este mesmo procedimento é realizado até que o mencionado vetor tenha todas as suas posições preenchidas, com médias que resultam das 3 leituras de cada um dos sensores formando uma matriz.

Com as 3 posições do vetor de médias preenchidas, a cada novo ciclo de leituras aplica-se um algoritmo que calcula os ângulos de movimento do usuário e compara com a sequência de ângulos que serve como referência para a identificação de uma queda. O algoritmo em

questão funciona comparando o módulo individual das últimas 3 médias de leituras do ângulo x com os valores, consecutivamente, 15(+-10), 40(+-10) e 80(+-20). Caso o padrão de queda não seja identificado no eixo x, ele testa as médias dos valores medidos no eixo y e considerando o mesmo padrão de ângulos. Se não houver coincidência de padrões no eixo x ou no eixo y, uma queda não foi identificada. Neste caso, o sistema entra em modo *sleep*, volta a zerar o vetor de médias de 3 posições e volta a fazer novas leituras somente quando for interrompido pelo MPU6050.

O padrão de ângulos (15+-10, 40+-10, 80+-20) que identificam uma queda, bem como o intervalo de 500 ms entre leituras foram obtidos experimentalmente, conforme explicaremos na primeira seção do próximo capítulo, que trata de resultados experimentais.

A partir da solução apresentada e discutida neste capítulo, foi possível desenvolver e testar um protótipo vestível de detector de quedas. No próximo capítulo, descreveremos a implementação física e os experimentos, incluindo os de injeção de falhas, que foram realizados no protótipo e que serviram para avaliar tanto a sua funcionalidade e desempenho, quanto a sua capacidade de detectar e tolerar falhas através do mascaramento de erros.

5 RESULTADOS EXPERIMENTAIS

No capítulo anterior, apresentamos em detalhes a solução proposta para o dispositivo detector de quedas. Foram cobertos os aspectos de *hardware*, de *software*, assim como as estratégias para detecção e tolerância a falhas. Neste capítulo, será abordada a implementação física do dispositivo, os experimentos com ele realizados em laboratório e em campo, assim como os resultados obtidos em termos de sua funcionalidade e desempenho, e também em termos de segurança e confiabilidade de operação do dispositivo.

5.1 Validação da função de detecção de queda

Para a validação da funcionalidade fim do dispositivo, montou-se, inicialmente, um *hardware* simples de teste, composto por um único sensor MPU6050, um único ESP32 (com o *software* inicial rodando exclusivamente em um de seus núcleos) e bateria. Com esta configuração mínima de *hardware* e *software*, desenvolveu-se a rotina de detecção de quedas e, através dela, foram feitos experimentos e coletados dados que permitiram não só validar a parte lógica do código, como também calibrar a lógica programada considerando as grandezas físicas efetivamente medidas em campo.

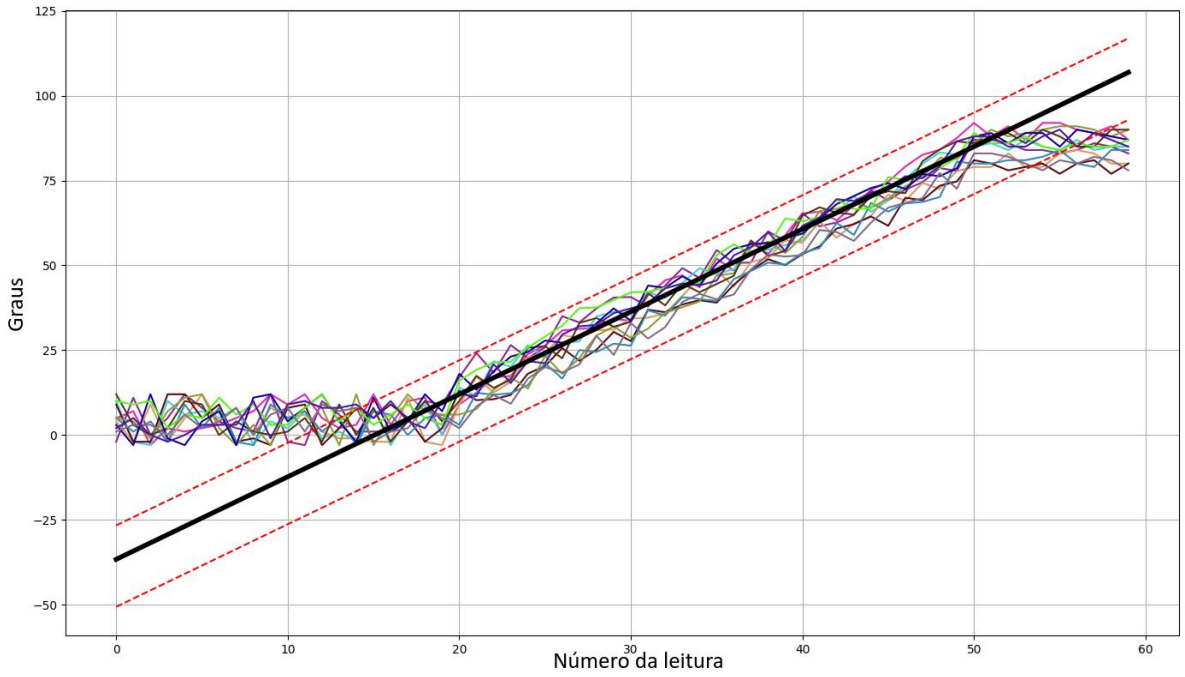
O principal experimento realizado nesta fase de desenvolvimento do protótipo foi o que resultou na identificação do padrão de medidas que caracterizaria uma queda. Iniciou-se emulando quedas e observando o padrão de evolução dos ângulos medidos pelo giroscópio do MPU 6050 nos eixos x, y e z. Desenvolveu-se uma rotina em *Python* para auxiliar na identificação deste padrão, tendo como condição de contorno reduzir ao máximo o número de falsos positivos.

Emulou-se 12 quedas e executou-se esta rotina coletando dados ininterruptamente à taxa máxima de leitura dos MPUs, que é de aproximadamente 20Hz (ou uma a cada 50ms). Esta rotina gerou o gráfico da Figura 23 para os valores medidos pelo giroscópio.

Considerando um tempo médio de 1,5s para uma queda (HSIEH, 2017), inicialmente os dados foram coletados e arranjados de maneira que a queda iniciasse perto da leitura 20 e terminasse próximo à leitura 50, conforme se pode ver na Figura 23. Claramente, são três as

regiões de interesse, na Figura 23, para a identificação de um padrão de queda: a região dos ângulos que antecedem e que dão início à queda, a região onde se observa a evolução destes ângulos ao longo da queda e a região dos ângulos ao final e que sucedem a queda. Portanto, 3 pontos de comparação seriam o mínimo necessário para identificação de um padrão evolutivo de queda.

Figura 23 – Resultado dos primeiros experimentos



Fonte: Produzida pelo autor

Considerando, na sequência, os componentes utilizados na nossa solução e as redundâncias de *hardware* e *software* (em particular as leituras múltiplas dos sensores) para a tolerância a falhas, foi necessário validar a compatibilidade dos tempos de processamento do dispositivo com os da queda propriamente dita. Partindo da premissa de um número mínimo de 3 leituras para comparação com o padrão de referência, como trabalhamos com 4 núcleos teríamos que cada núcleo realiza 3 leituras de cada um dos 3 sensores à taxa de uma leitura a cada 50ms. Ora, considerando um processamento completamente serializado, este nível de redundância resultaria em um ciclo total de 1800ms (4 x 3 x 3 x 50ms), superior à duração média de uma queda que é de 1500ms.

Para solucionar esta aparente incompatibilidade, a cada sensor foi atribuído seu próprio barramento I2C. Assim, a leitura destes sensores por cada núcleo passou a ser realizada com algum nível de paralelismo, para ajustar o tempo de processamento do sistema

(gasto com leituras, validação, cálculos de médias, passagens de *token*, votação, tomada de decisão e comunicação com o servidor) ao tempo real de ocorrência de uma queda.

Para encontrar valores de referência para as 3 regiões da Figura 23, a primeira tentativa realizada foi a de utilizar a média simples de cada uma das leituras em cada ponto das curvas da Figura 23, o que resultou ao que se assemelhava a retas para cada região, mas disformes. Recorremos, então, à técnica computacional de regressão linear para obter o resultado expresso na Figura 23 para a região de evolução da queda, representada pela reta em preto e suas tolerâncias, para mais e para menos, em vermelho pontilhado.

O resumo das medidas realizadas nas 3 regiões de interesse da Figura 23 aparece na Tabela 6. A Tabela apresenta a evolução dos ângulos nos eixos x, y e z do giroscópio, evidenciando os valores mínimos, máximos e a média simples para cada medida relativa às 12 quedas.

Tabela 6 - Resultados das medidas obtidas pelo protótipo mínimo para 12 quedas

valor giroscópio	primeira medida eixo (x,y,z)	segunda medida eixo (x,y,z)	terceira medida eixo (x,y,z)
média simples	(13.45,14.65,3.85)	(42.35,43.55,5.25)	(79.85,82.90,4.20)
valor máximo	(18,16,9)	(57,51,7)	(87,91,7)
valor mínimo	(6,8,-2)	(32,31,-4)	(67,71,-2)

Fonte: Produzida pelo autor.

O resultado dos testes foi que os ângulos medidos no eixo x OU y, ou nos eixos x E y, devem apresentar a evolução de +15 (+- 10 de tolerância) para +40 (+-10 de tolerância) e, por fim, +80 graus (+-20 graus de tolerância), para que seja uma queda. Outro ponto a ser destacado é que, como o valor medido no eixo z permaneceu sempre entre -5 e +10 graus, à primeira vista a sua participação na tomada de decisão pode parecer desnecessária. No entanto, a medida do eixo z e o seu enquadramento no intervalo acima, aumenta a probabilidade de que falsos positivos não ocorram quando realizadas ações de deitar e sentar, conforme veremos a seguir.

O experimento que foi a seguir realizado visava a validação do padrão identificado como exclusivo para quedas, perante a evolução de ângulos obtida para outras ações cotidianas realizadas pelo usuário, em particular a de deitar-se e sentar-se. A Tabela 7 abaixo traz um extrato desse experimento, exemplificando 3 situações diferentes de queda, de deitar e de sentar.

Tabela 7 - Padrões de ângulos obtidos em experimentos de queda, deitar e sentar

	Ângulo queda (X,Y,Z)	**Queda 1**	**Queda 2**	**Queda 3**
Leitura 0	(5 a 25, 5 a 25, 0 a 10)	(7,1,2)	(3,10,5)	(16,9,4)
1	(30 a 50, 30 a 50, 0 a 10)	(32,4,3)	(8,36,1)	(38,4,3)
2	(60 a 100, 60 a 100, 0 a 10)	(62,2,6)	(17,70,4)	(81,21,7)
	Ângulo queda (X,Y,Z)	**Deitar 1**	**Deitar 2**	**Deitar 3***
0	(5 a 25, 5 a 25, 0 a 10)	(14,2,22)	(5,22,1)	(6,18,5)
1	(30 a 50, 30 a 50, 0 a 10)	(12,4,21)	(2,14,17)	(4,19,23)
2	(60 a 100, 60 a 100, 0 a 10)	(34,2,5)	(1,48,12)	(4,34,50)
	Ângulo queda (X,Y,Z)	**Sentar 1**	**Sentar 2**	**Sentar 3***
0	(5 a 25, 5 a 25, 0 a 10)	(14,7,3)	(14,10,3)	(7,9,90)
1	(30 a 50, 30 a 50, 0 a 10)	(44,8,6)	(35,10,3)	(41,3,50)
2	(60 a 100, 60 a 110, 0 a 10)	(21,5,6)	(11,7,1)	(15,6,30)

Fonte: Produzida pelo autor.

Nos itens Deitar 3* e Sentar 3*, cujos valores medidos nos eixos y e x, respectivamente, se aproximam do padrão de queda, é possível verificar um comportamento no eixo z que se diferencia, em muito, dos outros padrões de Deitar e Sentar, e também do padrão esperado em uma queda. No caso da ação de Deitar, isso ocorre porque se partiu da posição sentado na cama à posição deitado, girando o corpo no próprio eixo. Já no caso da ação de Sentar, esta foi realizada ao ingressar em um carro, girando o corpo a 90 graus para sentar-se.

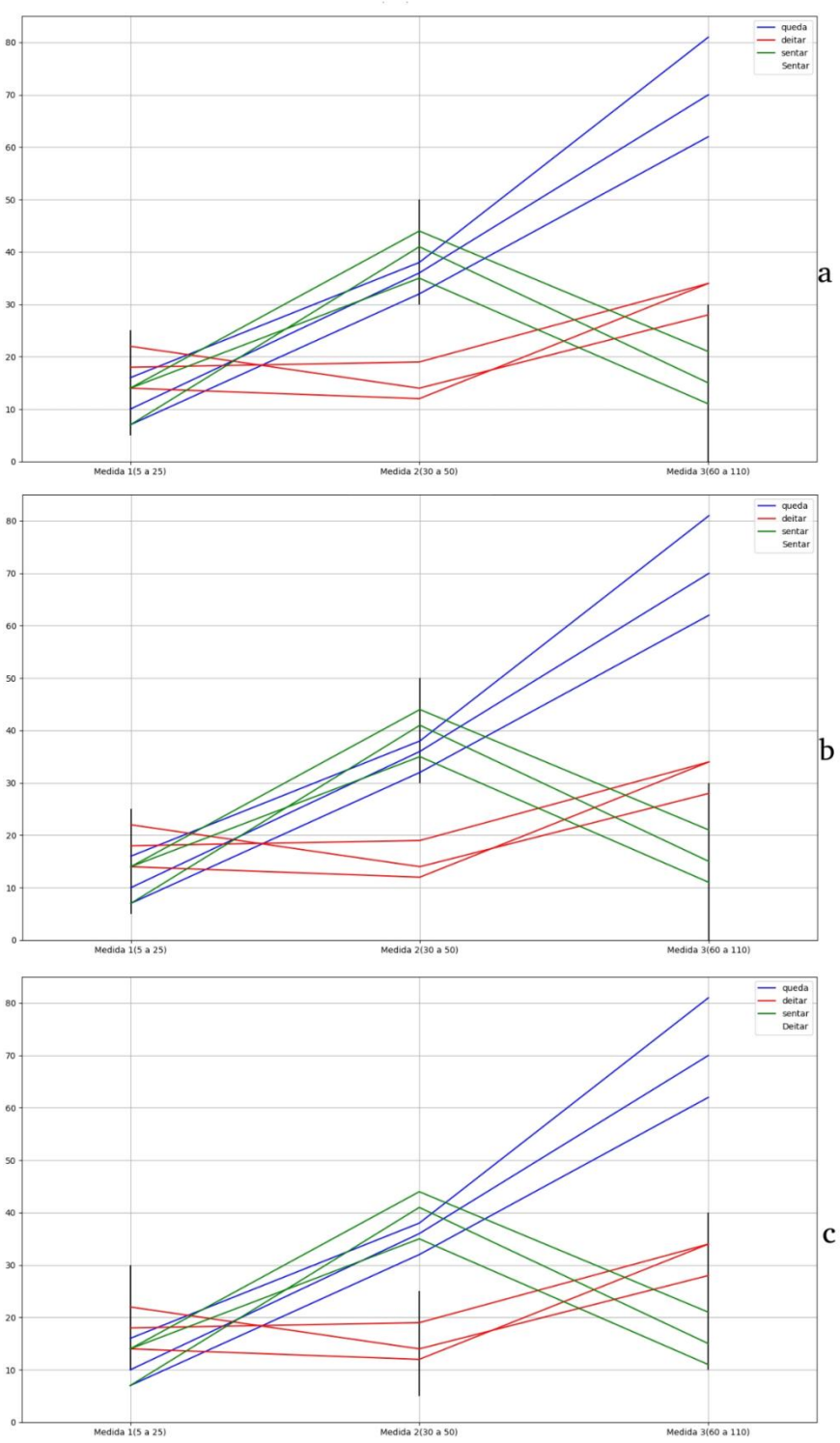
Os gráficos da Figura 24 demonstram as diferenças entre os padrões destas 3 condições do usuário. Séries em azul são quedas, em verde quando o usuário deita-se e, em vermelho,

quando ele senta. Nas barras pretas, encontra-se a sequência alvo de ângulos, com suas respectivas tolerâncias superiores e inferiores. Também é possível notar que o sistema sempre inicia com um valor abaixo de 25 graus e acima de 5 (primeira barra preta). Este intervalo de valores foi quem determinou como programar os MPU 6050 para a solicitação de interrupção aos ESPs que dispara a leitura dos sensores, conforme vimos no capítulo anterior. As leituras dão início quando o MPU envia o sinal de variação de ângulo.

Na Figura 24, em (a), (b) e (c) , as barras pretas servem para comparar as ações de queda, sentar e deitar relativamente ao limite considerado pelo algoritmo para a detecção da respectiva ação. Neles é possível verificar que o padrão de evolução dos ângulos é muito distinto de uma ação a outra.

Uma vez desenvolvido e validado o algoritmo e o seu padrão de evolução dos ângulos, passamos à inclusão das técnicas de detecção e tolerância a falhas no protótipo.

Figura 24 – Comparação entre padrões de queda, sentar e deitar: (a) com os limites ajustados para detecção de queda; (b) com os limites ajustados para a ação de sentar; (c) com os limites ajustados para a ação de deitar.



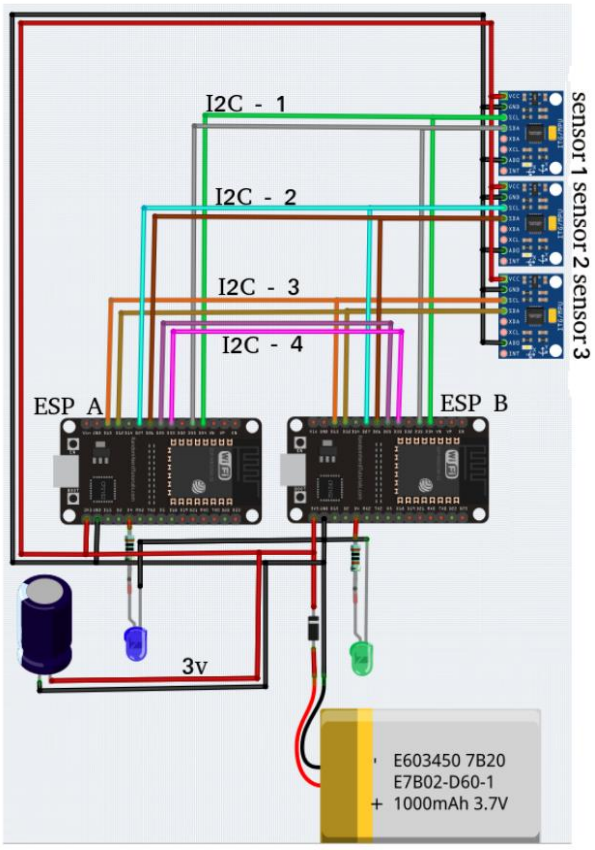
Fonte: Produzida pelo autor

5.2 Protótipo físico

Como visto no Capítulo 3, tolerar falhas implica em algum nível de redundância espacial, lógica ou temporal, incluindo aí componentes adicionais de *hardware*, de *software* ou de processamento. A avaliação de todos estes aspectos, combinados com restrições de consumo e custo, resultaram na proposta de uma arquitetura com sensores e conexões físicas triplicadas, com processadores quadruplicados (dois ESPs com dois núcleos processadores, cada núcleo rodando praticamente o mesmo código), e com recursos de comunicação e de alimentação duplicados, conforme antecipado no Capítulo 4.

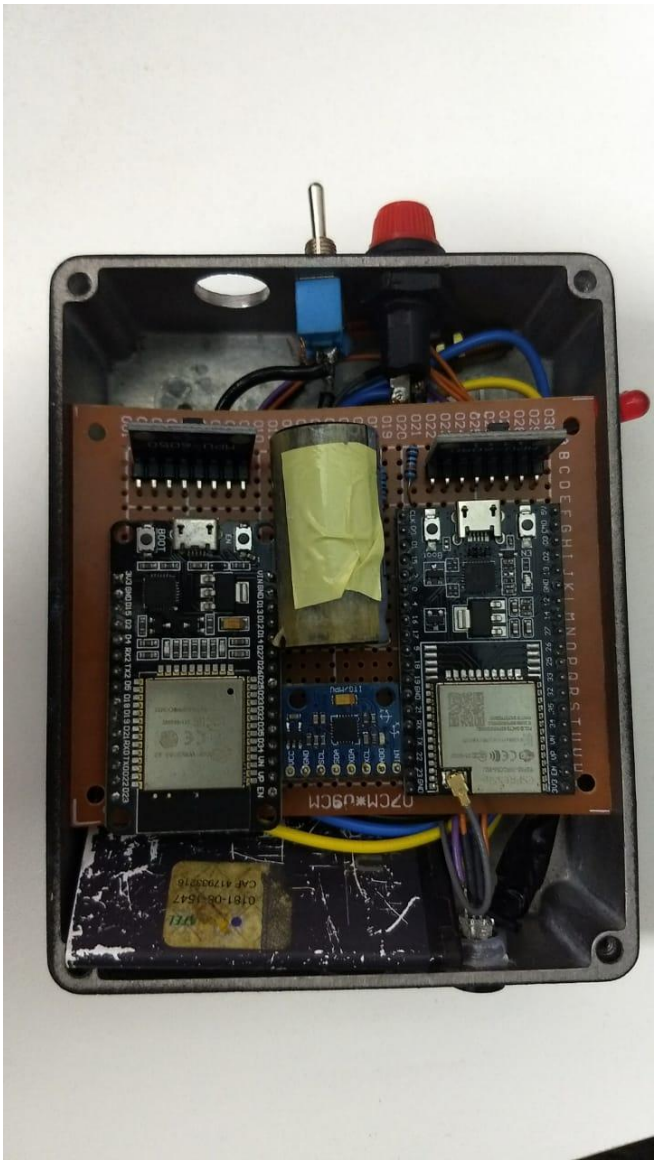
O esquemático do protótipo final do dispositivo de detecção de queda vestível é mostrado na Figura 25 e na figura 26 é mostrado o protótipo construido. Na Figura é possível identificar os 3 sensores MPU 6050, os 3 barramentos I2C, os 2 ESP32, as baterias e o supercapacitor de *backup*. A estrutura escolhida foi justificada em capítulos anteriores.

Figura 25 – *Hardware* final do protótipo



Fonte: Produzida pelo autor

Figura 26 – Protótipo final



Fonte: Produzida pelo autor

A redundância de hardware exige inerentemente mais energia. Como nosso protótipo deve ser vestível, o consumo de energia é um ponto crítico e, portanto, o dimensionamento da fonte de alimentação deve resultar de uma equalização entre o tempo mínimo de uso entre recargas consecutivas, as dimensões físicas do dispositivo e a maneira como o *software* é executado no *hardware* redundante.

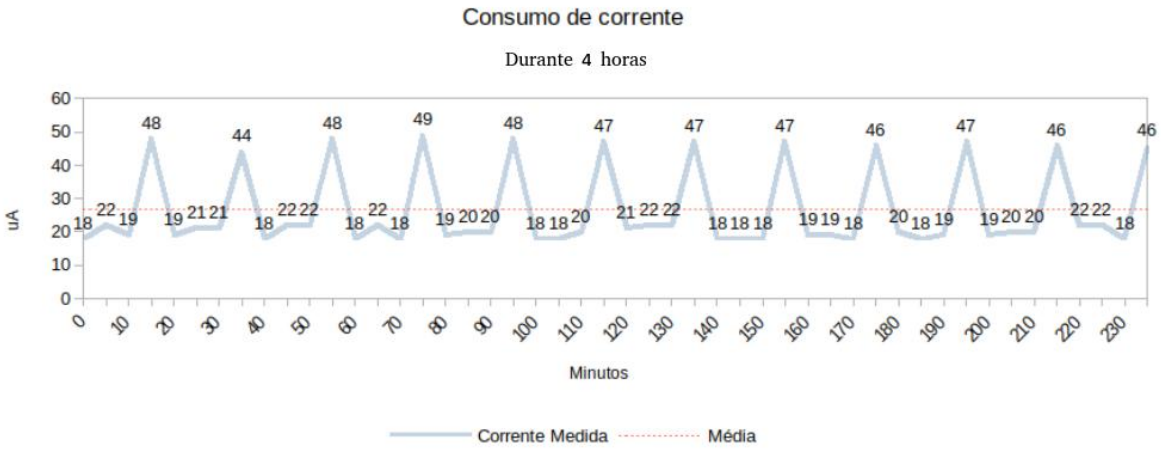
No protótipo implementado, conforme já apresentado no capítulo anterior, foi decidido executar o mesmo código em cada núcleo de cada microprocessador, mas de maneira serial e circular e, não simultaneamente. Ou seja, o próximo núcleo começa a executar o *software*

quando o anterior terminar de ler os sensores e calcular as médias de medição. Quando um núcleo está ativo, os outros 3 ficam inativos e aguardam sua vez de entrar em operação.

Um ciclo completo de 3 leituras de sensor em todos os 4 núcleos é realizado a cada 2 s. Em cada um desses ciclos, os rádios *Bluetooth* e *Wi-Fi* não são acionados, o que também reduz o pico de corrente e reduz a especificação da bateria e do supercapacitor. O tempo de execução da rotina de leitura do sensor é determinístico e sua duração é conhecida previamente. Portanto, se um núcleo em particular enfrentar um problema e não confirmar o fim de sua atividade, o próximo núcleo perceberá que, através do estouro de seu *watchdog timer*, deve assumir o papel de mestre. Nesse contexto de *hardware* redundante e execução de *software* serial e redundante, o consumo de corrente do sistema foi medido usando o circuito integrado ACS 712 [9].

A utilização do dispositivo vestível tem seu gráfico de consumo de corrente mostrado na Figura 26 (corrente média= 26,4792 uA). Os valores expressos no gráfico correspondem a uma janela de 5 minutos, de uma bateria de medições que totalizou 4 horas. Esta janela de 300 segundos é composta por médias simples calculadas a cada segundo a partir de 100 amostragens de corrente. Pontilhado em vermelho, encontra-se o valor de 27 uA, que é o valor da média simples entre todos os resultados somados e divididos por 4 horas.

Figura 27 – Consumo de corrente do protótipo medido durante 4 horas



Fonte: Produzida pelo autor

[9] Disponível em: https://www.allegromicro.com/en/products/sense/current-sensor-ics/zero-to-fifty-amp-integrated-conductor-sensor-ics/acs712.

Esse baixíssimo valor de corrente ocorre porque os dois microprocessadores consomem juntos aproximadamente 20uA no modo *deep-sleep* e outros 3uA são fornecidos aos sensores. Os picos são provenientes da transmissão utilizando o rádio *Wi-Fi* para envio do *keep-alive*. Estes se dão durante a execução do programa. No gráfico não aparece o momento da transmissão da informação de queda, pois ela será enviada uma única vez (quando o usuário de fato cair). O valor energético da transmissão de informação de uma queda é praticamente o mesmo dos períodos de maior consumo expressos no gráfico.

Também cabe lembrar que o sistema fica em *deep-sleep* até que qualquer um dos 3 sensores acorde os microprocessadores por meio de interrupção externa ou até que este tenha seu temporizador estourado. Os picos de corrente chegam a 120 mA, porém durante um período muito curto, cerca de 22 segundos. Este tempo elevado ocorre pois, para que seja realizada a conexão com a rede, o ESP leva 4 segundos, depois mais 2 segundos para a transmissão do *keep alive*, em seguida mais 5 segundos para transmitir o resultado do último teste e, por último, mais um tempo igual para que o segundo ESP execute o mesmo procedimento e ainda envie os seus próprios dados.

Conforme demonstrado a seguir, para atender à especificação acima, o valor encontrado para o supercapacitor foi de 1 F em 3V. Considerando um consumo de corrente típico de 27 uA/h para o dispositivo vestível, uma tensão de alimentação do sistema de 3V e uma constante de tempo de 24 horas, esse capacitor garante uma operação sem bateria por um período de 30 horas e 52 minutos aproximadamente, conforme pode se observar pelos cálculos a seguir.

Inicialmente, foi calculado o valor da resistência aproximada do sistema utilizando $V = R\ x\ I$.

Substituindo os valores de V por 3 V e I por 27 uA.

$R = 3/27u \simeq$ 111111,1 Ohm.

Agora é preciso calcular o tempo de descarga do capacitor.

$$t = RxC$$

onde t= 24 h (86400 s) e R $\simeq$ 111,1 K Ohm

$$C = 86400/111{,}1K \simeq 0{,}7776\,F$$

Como inexiste o valor 0,7776 F, o valor mais próximo encontrado (em 3 V) foi o de 1 F. Com esses valores é possível calcular o tempo ao longo do qual esse capacitor pode alimentar o circuito.

$$t = C\,x\,R, substituindo$$

$$t = 1F\,x\,111{,}1K = 111111{,}1\,s\,ou\,30{,}86\,h$$

Esse tempo de 30,86 h é o tempo estimado para que o capacitor descarregue 100% de sua carga (0 V). O ESP 32 trabalha com tensões de até 2,4 V, dessa maneira, durante os testes funcionais, o tempo que o circuito de fato ficava em funcionamento somente utilizando o capacitor foi entre 15 e 16 horas.

Além disso, o tempo necessário para recarregar o capacitor na ausência total de uma bateria é de aproximadamente 1 h. Ou seja, considerando um nível inicial de carga em 0% (totalmente descarregado), ao final de uma hora de carga o capacitor chegará a 100%. Com essas características e as devidas precauções na recarga, torna-se muito seguro o uso do dispositivo alimentado exclusivamente pelo capacitor até que o idoso possa substituir a bateria.

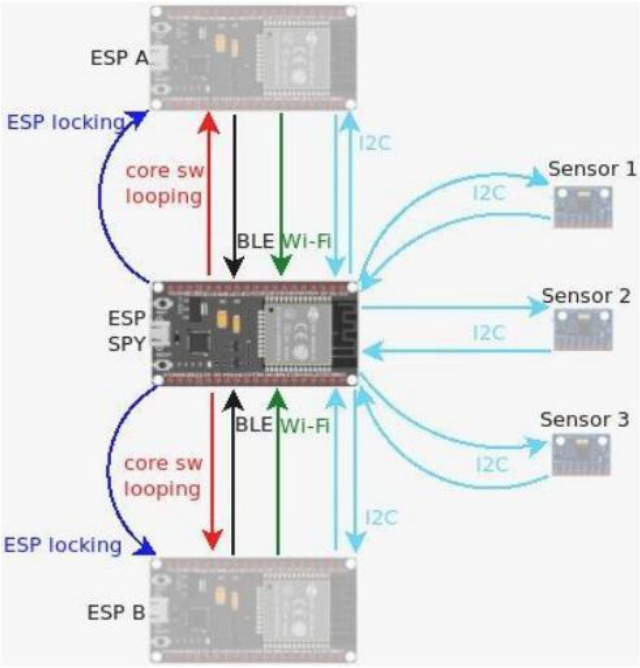
5.3 Injeção de falhas

Uma plataforma para injeção de falhas foi construída para validar todos os mecanismos implementados em nosso dispositivo vestível para testar, diagnosticar e tolerar defeitos nos sensores, microprocessadores, falhas de comunicação e fornecimento de energia. Essa plataforma é baseada em um espião/invasor ESP 32, como mostrado na Figura 27.

Para emular a ocorrência de leituras discrepantes/erradas de sensor, defeitos de trilhas I2C, núcleos individuais ou *loop* e travamento de todo o *software* ESP, rádios de comunicação inoperantes, falta de energia da bateria e também para validar os mecanismos implementados para mascaramento de erros, isolamento lógico de peças defeituosas e sinalização da manutenção necessária, um ESP 32 espião foi inserido como um tipo de *buffer* entre os ESP32s existentes e os sensores MPU6050.

Como pode ser visto na Figura 27, o intruso ESP *Spy* divide os barramentos I2C, começa a intermediar a comunicação *bluetooth* e *Wi-Fi* entre ESP A e ESP B e ganha controle para provocar *loop* de *software* em núcleos e travamento em ESPs através de fios de conexão adicionais com ESP A e ESP B .

Figura 28 – Plataforma de injeção de falhas



Fonte: Produzida pelo autor

Dentro desta plataforma e através do ESP *Spy*, as seguintes falhas foram injetadas em diversos experimentos:

- Corrupção de pacotes de dados ou de sinais de relógio para emular a leitura incorreta do sensor e falhas do tipo *stuck-at*, curto-circuito e circuitos aberto nos fios do barramento I2C;

- Travamento de *software* e estouro do *watchdog timer* nos núcleos individuais do ESP A e do ESP B, emulando assim falhas nos microprocessadores;

- Desligamento dos rádios *bluetooth* e *Wi-Fi* para emular falhas na comunicação sem fio entre os ESPs e destes com o servidor em nuvem.

Em um primeiro momento, todas essas falhas e também a condição de falta de bateria (esta foi realizada desconectando-se manualmente um fio de alimentação) foram injetadas em nosso dispositivo vestível, considerando uma ocorrência simples de falhas. Em todos os

experimentos realizados durante a recarga da bateria (quando os testes *off-line* são realizados) e durante a execução da aplicação (testes *on-line* e votação), os mecanismos adotados no protótipo implementado mostraram-se adequados para a finalidade de detectar e tolerar falhas.

Em um segundo momento, adotou-se a ocorrência múltipla de falhas de uma mesma categoria (dois ou mais sensores, duas ou mais trilhas de barramento, etc) durante a execução da aplicação, e avaliou-se o nível de degradação da confiabilidade, da segurança e da própria operação do dispositivo. Os testes foram organizados em:

1 Falha de barramento (em todos os 4 individualmente)

1.a *Stuck-at* 0

a.i Barramento de dados

a.ii Barramento de *clock*

1.b *Stuck-at* 1

b.i Barramento de dados

b.ii Barramento de *clock*

2 Falha de *core*

2.a Simulando travamento individual em cada *core*

a.i ESP A1

a.ii ESP A2

a.iii ESP B1

a.iv ESP B2

2.b Simulado travamento em *cores* do mesmo ESP

b.i A1 e A2

b.ii B1 e B2

2.c Simulado travamento em *cores* de ESPs diferentes

c.i A1 B1

c.ii A1 B2

c.iii A2 B1

c.iv A2 B2

3 Simulação de combinação de fatores adversos:

3.a ESP B 1 ou B 2 ou A 1 ou A 2 em falha somado a

a.i Sensor 1 em falha

a.ii Sensor 2 em falha

a.iii Sensor 3 em falha

a.iv Sensores 1 e 2 em falha

a.v Sensores 2 e 3 em falha

4 Sensores 1 e 3 EM falha

4.a ESP B 1 e B 2 ou A 1 e A 2 em falha somado a

4.b Sensor 1 em falha

4.c Sensor 2 em falha

4.d Sensor 3 em falha

4.e Sensores 1 e 2 em falha

4.f Sensores 2 e 3 em falha

4.g Sensores 1 e 3 falha

Durante todos os testes o protótipo foi capaz de utilizar todos os resultados coletados de maneira correta e não perdeu nenhum deles. Após verificado que o sistema proposto

funcionou na retenção dos valores obtidos foi a hora de testar o mesmo durante 20 quedas para validar o modelo e confirmar que o mesmo encontra-se funcional mesmo adicionando este processamento a mais.

A validação do modelo em campo foi feita arbitrando 5 quedas para frente, 5 para trás, 5 laterais para a esquerda e mais 5 para a direita. Durante todos os testes as quedas foram detectadas de maneira satisfatória e sem erros. Foi medido o tempo de execução de cada rotina e a rotina principal como um todo. O acréscimo de código/processamento equivale a menos de 2 ms na execução total da rotina. A Tabela 8 resume os resultados experimentais obtidos da campanha de injeção de falhas.

Tabela 8 - Campanha de injeção de falhas e capacidade de resiliência resultante

Modelo de falha		Componente(s) defeituoso(s)	Capacidade de resiliência			
			Tolerante a falhas	Operação segura	Operação insegura	Inoperante
Sensores	Leitura incorreta (descalibrado ou defeito físico)	Nenhum	☑			
		1 sensor		☑		
		2 sensores			☑	
		todos os sensores				☑
	Defeitos no barramento I2C (stuck-at, circuito aberto ou curto circuito)	Nenhum	☑			
		1 trilha de dados		☑		
		1 trilha de clock		☑		
		1 trilha de clock e dados (mesmo barramento)		☑		
		2 trilhas de dados e/ou 2 trilhas de clock (2 barramentos)			☑	
		3 trilhas de dados ou 3 trilhas de clock(todos os barramentos)				☑
Microprocessadores	Núcleo em loop	Nenhum	☑			
		1 núcleo	☑			
		2 núcleos		☑		
		3 núcleos			☑	
		Todos os núcleos				☑
	ESP travado (watchdog)	Nenhum	☑			
		1 ESP 32		☑		
		2 ESP 32				☑
Comunicação	Radio Bluetooth inoperante	Nenhum	☑			
		1 rádio		☑		
		2 rádios		☑		
	Radio Wi-Fi inoperante	Nenhum	☑			
		1 rádio			☑	
		2 rádios				☑
	Ambos os radios	Sem Bluetooth e sem Wi-Fi				☑
Alimentação	Problemas na alimentação	Nenhum		☑		
		Bateria desconectada /descarregada			☑	
		Bateria e super capacitor descarregados/desconectados				☑

Fonte: Produzida pelo autor.

Observe que, sob diferentes condições de falha, a capacidade de resiliência do dispositivo vestível muda. São possíveis quatro condições diferentes:

- tolerante a falhas - o dispositivo tem recursos suficientes para continuar mascarando erros e tolerando novas falhas;

- operação segura - pode garantir uma operação segura, sendo capaz de sinalizar valores de leitura discrepantes quando uma nova falha/erro ocorrer;
- operação insegura - o dispositivo pode continuar operando, mas sem garantias de que os valores de leitura e as decisões tomadas estejam corretas; na ocorrência de uma nova falha, ele poderá tornar-se inoperante;
- Inoperante - não funciona, necessita de manutenção.

5.4 Testes de campo

Durante 9 meses um idoso foi monitorado utilizando o dispositivo proposto. Este idoso efetivamente sofreu seis quedas, que testemunhou terem sido três delas no banheiro, uma no pátio da casa, outra tentando levantar-se da cama e outra em local não revelado. Durante os testes de campo também foram emuladas 3 quedas controladas.

Das 6 quedas sofridas pelo idoso, 3 foram frontais e resultaram nas medidas à Tabela 9. Em todos os casos, o padrão de evolução dos ângulos característico de uma queda apareceu nas medidas do eixo x.

Tabela 9 - Médias das leituras dos giroscópios em 3 quedas frontais sofridas

Valores alvo (eixo x)	Frontal 1	Frontal 2	Frontal 3
15	10	13	21
40	30	50	53
80	97	78	94

Fonte: Produzida pelo autor.

O idoso também sofreu 2 quedas laterais, e 1 foi emulada em ambiente controlado, sobre uma superfície macia. A Tabela 10 traz os resultados das médias calculadas para as leituras realizadas pelos giroscópios. Já nesta classe de quedas, o padrão de ângulos foi observado pelas medidas do eixo y.

Tabela 10 - Médias das leituras dos giroscópios em 3 quedas laterais (2 sofridas e 1 emulada)

Valores alvo (eixo y)	Lateral 1	Lateral 2	Lateral (Emulada)
15	17	15	9
40	49	55	41
80	95	87	95

Fonte: Produzida pelo autor.

Por fim, o idoso sofreu uma queda para trás (de costas) e outras duas do mesmo tipo foram emuladas. Em todos os casos, o padrão de evolução dos ângulos característico de uma queda voltou a aparecer nas medidas do eixo x, como esperado e como demonstrada à Tabela 11.

Tabela 11 - Médias das leituras dos giroscópios em 3 quedas de costas (1 sofrida e 2 emuladas)

Valores alvo (eixo x)	Costas 1	Costas (Emulada 1)	Costas (Emulada 2)
15	25	13	11
40	60	53	61
80	97	95	94

Fonte: Produzida pelo autor.

Durante os testes de campo, conforme se pode observar, o sistema mostrou-se capaz de identificar corretamente quedas e distingui-las das ações de sentar e deitar do idoso. Não observaram-se falsos positivos.

O sistema demonstrou-se útil e vital para a vida e liberdade do idoso que submeteu-se aos testes de campo. Este que teve seus parentes avisados na ocorrência de todas as quedas que sofreu no alcance da rede *wireless* de sua residência. De fato, os testes de campo ficaram limitados a esta região por uma restrição funcional do protótipo vestível. Portanto, um ponto a ser melhorado consiste justamente na integração de um meio de comunicação com o servidor além do próprio *Wi-Fi*.

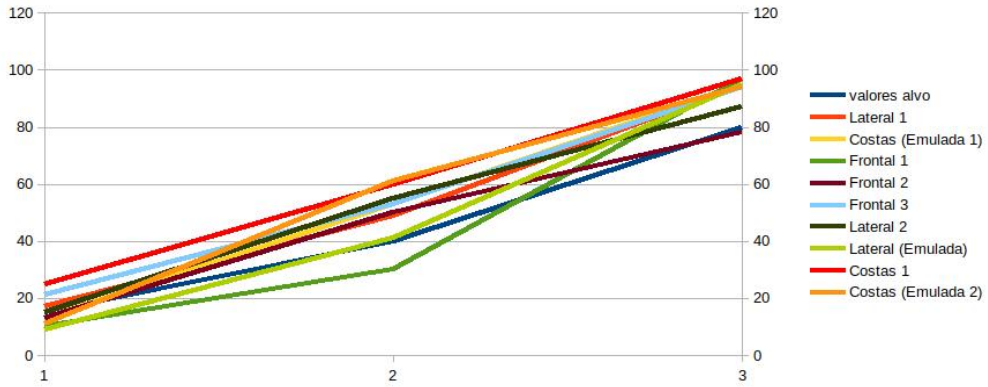
No caso específico deste usuário, outro ponto a ser destacado é que, como o sistema desenvolvido não é compatível com *smartphones* com o sistema operacional IOS, limitou-se o número de parentes que poderiam monitorar o idoso. Quatro de um total de oito parentes não puderam participar dos testes.

Outro destaque dos testes realizados foi que o idoso perdeu o carregador do protótipo em uma ocasião. Assim, muitas vezes o protótipo acabou ficando sem bateria. Cabe ressaltar que todos os 10 alarmes de bateria fraca foram vistos pela família do idoso e o mesmo foi visitado assim que possível para verificar o que havia ocorrido.

Durante quatro das seis quedas sofridas pelo idoso, o sistema sofreu avarias que foram consertadas após alguns dias. Graças às rotinas de teste *off-line,* o diagnóstico e a manutenção das partes defeituosas ficaram muito facilitados.

Após a primeira queda, não foi possível substituir um dos três sensores e o idoso acabou utilizando somente dois durante um mês. Neste mês, ocorreu outra queda. Esta, que foi detectada pelo sistema e é descrita como queda frontal 1 na Tabela 9. No gráfico da Figura 28, é possível perceber que o comportamento desta queda, em particular, é menos linear do que das outras quedas ocorridas, mas ainda assim detectada pelo protótipo. O sistema, portanto, provou ser capaz de tolerar falhas em um contexto de operação segura, conforme anteriormente colocado pela Tabela 8.

Figura 29 – Padrão de ângulos observados nas 9 quedas experimentadas pelo idoso



Fonte: Produzida pelo autor

6 CONCLUSÕES

Esta dissertação propôs, desenvolveu e testou um protótipo de um dispositivo vestível, altamente confiável, para a detecção de quedas de idosos. A principal motivação para conduzir este estudo foi o fato de que, além de quedas serem a causa de 15% das mortes de idosos no Brasil (WANDERLEY, 2019), elas podem ferir ou até incapacitar, se o tempo para prestar atendimento e os primeiros socorros à vítima for longo (BURGOS-FLÓREZ, 2020).

Na escolha da plataforma para implementar o dispositivo, nos defrontamos com a dificuldade de idosos nem sempre fazerem o uso mais adequado de um smartphone. Daí a decisão de partir para uma solução dedicada e vestível. Além disso, pesquisando os dispositivos disponíveis no mercado, concluímos que as soluções existentes não propunham características que garantem uma operação segura e confiável do detector de quedas na presença de falhas permanentes ou transientes. A título de exemplo, quando dos testes de campo realizados neste trabalho, vimos-nos na situação de conviver com avarias no dispositivo causadas pela própria queda do idoso e a impossibilidade de realizar uma manutenção imediata no protótipo. Como ele foi construído para, em alguma medida, tolerar falhas, foi possível seguir usando o dispositivo de maneira segura e confiável, apesar dos danos que sofrera.

Durante o desenvolvimento da solução, conforme descreve o Capítulo 2, avaliamos diversas possibilidades em termos de componentes para compor a arquitetura do nosso protótipo. Estudamos vários sensores para a detecção de queda, vários microprocessadores e microcontroladores, assim como formas de comunicação para garantir a conexão do dispositivo que monitora o idoso ao seu cuidador. Resultou, conforme justificado no capítulo 4, na escolha do sensor MPU6050, que contém um acelerômetro e um giroscópio; do microcontrolador ESP32, que possui microprocessador dual core, rádios bluetooth e Wi-Fi; e do ambiente de nuvem Google Firebase® usado para a comunicação com o cuidador do idoso.

No momento de endereçar os aspectos de segurança e confiabilidade da operação, recorremos aos conceitos e técnicas de teste e tolerância a falhas apresentados no Capítulo 3. A partir desses conceitos e técnicas adotamos um modelo de falhas, predominantemente funcional, e, a partir dele, desenvolvemos rotinas de *software* para o teste e diagnóstico dos

componentes de hardware, alguns deles realizados no momento da recarga da bateria do dispositivo (testes *off-line*) e outros, realizados concomitantemente com o código principal da aplicação (testes *on-line*). Chegada a hora de propor mecanismos para tolerar falhas, lançamos mão de redundância de hardware, triplicando sensores, barramentos e duplicando rádios de comunicação, e de redundância de software, quadruplicando microprocessadores que executam, serialmente, o mesmo código. Aliando as mencionadas redundâncias à técnica de mascaramento de erros por votação, o dispositivo passou a ser capaz de realizar sua missão mesmo na presença de defeitos ou falhas transientes em seus componentes. O teste e a tolerância a falhas, tal qual implementados no protótipo de de detector de quedas, foram objeto do Capítulo 4.

Finalmente, descrevemos e analisamos, no Capítulo 5, os experimentos de laboratório realizados para a validação da rotina detectora de quedas, a implementação física do protótipo redundante, a plataforma de injeção de falhas e a respectiva campanha de injeção feita sobre o dispositivo, além dos testes de campo do protótipo que foram realizados com um voluntário idoso ao longo de 9 meses. Todos estes experimentos serviram para avaliar tanto a funcionalidade e o desempenho do protótipo, quanto a sua capacidade de detectar e tolerar falhas através do mascaramento de erros, e permitiram concluir que a solução proposta atingiu o objetivo principal deste trabalho que era o de proporcionar segurança e confiabilidade à operação de um dispositivo detector de quedas.

Como não podia ser diferente, ao longo do desenvolvimento e dos testes experimentais, foram identificados diversos pontos de melhoria, assim como oportunidades para trabalhos futuros, que elencamos abaixo:

- Além de melhorias diversas na estratégia de codificação das rotinas de aquisição e processamento de dados, outras configurações de hardware utilizando outros modelos de giroscópios (GY-291, Bmi160, Mma8452q, MPU-6500, GY-LSM6DS3, Icm 20948, entre outros), outros microprocessadores (ESP32-C3, ESP32-S2, ESP8285, ESP8266, entre outros) e outros barramentos ou formas de comunicação entre estes componentes (SPI, CAN, RS232, entre outros), poderiam ter resultado em uma solução menos complexa e, ainda assim, satisfatória do ponto de vista do objetivo principal deste trabalho;

- A implementação de um rádio GPRS para ocasiões em que o usuário sai de sua residência ou quando o sinal de Wi-Fi está instável ou indisponível, teria atendimento melhor às necessidades do idoso que se voluntariou aos testes de campo ;

- Nesta mesma linha, a implementação de uma malha LoRa (Long Range Wi-Fi), usando 3/4/5G ou até uma rede Sigfox®, para que o sistema seja independente das atuais restrições de alcance das redes Wi-Fi comuns;

- Implementação de um servidor próprio, a fim de evitar a crescente limitação dos serviços hoje prestados gratuitamente pela plataforma Google Firebase;

- Utilização de um sistema de comunicação mesh com rádios ble 5.x acoplados a cada sensor e um concentrador conectado diretamente em um meio de comunicação sem fio como Wi-Fi ou 2/3/4/5G, assim reduzindo significativamente o tempo de manutenção do dispositivo já que o próprio usuário pode substituir o sensor defeituoso (caso o aparelho seja composto por um rádio ble 5.0, um giroscópio e uma bateria);

- A implementação de um protocolo MQTT para que a solução proposta seja compatível com fornecedores globais de produtos e serviços de automação residencial.

- Implementação do *hardware* proposto em um único *chip* englobando toda a aplicação e as estruturas e funcionalidades necessárias à tolerância a falhas, desta maneira proporcionando uma série de vantagens como: dimensão reduzida do *hardware* final, redução significativa do consumo de energia, maior disponibilidade do sistema como um todo e menor custo em escala.

REFERÊNCIAS

ABBATE, Stefano; AVVENUTI, Marco; BONATESTA, Francesco; COLA, Guglielmo; CORSINI, Paolo; VECCHIO, Alessio. A smartphone-based fall detection system. **Pervasive and Mobile Computing**, v. 8, n. 6, p. 883-899, 2012. Disponível em: https://doi.org/10.1016/j.pmcj.2012.08.003. Acesso em: 8 out. 2020.

ALVES, Ana Honorato Cantalice; PATRÍCIO, Anna Cláudia Freire de Araújo; ALBUQUERQUE, Karla Fernandes de; DUARTE, Marcella Costa Souto; SANTOS, Jiovana de Souza; OLIVEIRA, Michelle Salles de. Occurrence of falls among elderly institutionalized: prevalence, causes and consequences. **Revista de Pesquisa**: cuidado é fundamental online, v. 8, n. 2, 2016. Disponível em: https://doi.org/10.9789/2175-5361.2016.v8i2.4376-4386. Acesso em: 30 nov. 2020.

AMORIM, Diane Nogueira Paranhos; SAMPAIO, Luísa Veríssimo Pereira; CARVALHO, Gustavo de Azevedo; VILAÇA, Karla Helena Coelho. Aplicativos móveis para a saúde e o cuidado de idosos. **Revista Eletrônica de Comunicação, Informação e Inovação em Saúde**, v. 12, n. 1, 2018. Disponível em: https://doi.org/10.29397/reciis.v12i1.1365. Acesso em: 4 nov. 2021.

BALEN, Tiago Roberto; LUBASZEWSKI, Marcelo Soares. Teste e projeto visando a testabilidade de circuitos e sistemas integrados. *In*: SAWICKI, Sandro; REIS, Ricardo Augusto da Luz. **Tópicos em micro e nanoeletrônica**. Ijuí: Editora Unijuí, 2014, v.1. p. 6-48(páginas do capítulo). acesso em 05-04-2020, 09:26

BORRI, Simone; HAGE-HASSAN, Magali; DILILLO, Luigi; GIRARD, Patrick; PRAVOSSOUDOVITCH, Serge; VIRAZEL, Arnaud. Analysis of dynamic faults in embedded-SRAMs: implications for memory test. **Journal of Electronic Testing**, v. 21, n. 2, 2005. Disponível em: https://doi.org/10.1007/s10836-005-6146-1. Acesso em: 26 jun. 2021.

BURGOS-FLÓREZ, Francisco; GARZÓN-ALVARADO, Diego Alexander. Stress and strain propagation on infant skull from impact loads during falls: a finite element analysis. **International Biomechanics**, v. 7, n. 1, 2020. Disponível em: https://doi.org/10.1080/23335432.2020.1719196. Acesso em: 4 jun. 2021.

COUTINHO, Evandro Silva Freire; BLOCH, Katia Vergetti; COELI, Claudia Medina. One-year mortality among elderly people after hospitalization due to fall-related fractures: comparison with a control group of matched elderly. **Cadernos de Saúde Pública**, v. 28, n. 4, 2012. Disponível em: https://doi.org/10.1590/S0102-311X2012000400019. Acesso em: 10 maio. 2021.

EICHELBERGER, Edward; WILLIAMS, Thomas. A Logic Design Structure for LSI Testability. In: DESIGN AUTOMATION CONFERENCE, 14., 1977, New Orleans. **Anais** [...]. Nova Jersey: IEEE Press, 1977. Disponível em: https://dl.acm.org/doi/abs/10.5555/800262.809170. Acesso em: 25 maio. 2020. p. 462-468.

FERNANDES, Nathan Martins. **A experiência de uso de smartphones por indivíduos idosos e o desenvolvimento de requisitos para interfaces de smartphones mais amigáveis**. 2022. Dissertação (Mestrado em Design) – Faculdade de Arquitetura, Artes, Comunicação e Design, Universidade Estadual Paulista, São Paulo, 2022.

GODSE, Atul; GODSE, Deepali. **Microprocessor and Interfacing**. Pune: Technical Publications, 2020.

HSIEH, Chia-Yeh; LIU, Kai-Chun; HUANG, Chih-Ning; CHU, Woei-Chyn; CHAN, Chia-Tai. Novel Hierarchical Fall Detection Algorithm Using a Multiphase Fall Model. **Sensors**, v. 17, n. 2, 2017. Disponível em: 10.3390/s17020307. Acesso em: 22 set. 2021.

KERDJIDJ, Oussama; RAMZAN, Naeem; GHANEM, Khalida; AMIRA, Abbes; CHOUIREB, Fatima. Fall detection and human activity classification using wearable sensors and compressed sensing. **Journal of Ambient Intelligence and Humanized Computing**, v. 11, n. 1, 2020. 349-361. Disponível em: https://doi.org/10.1007/s12652-019-01214-4. Acesso em: 14 jun. 2021.

KHALIFEH, Ala; SALEH, Adham; AL-NUIMAT, Mahmoud; ABOU-TAIR, Dhiah el Diehn; ALNUMAN, Nasim. Design and implementation of internet of things and cloud based platform for remote health monitoring and fall detection. *In*: FARDOUN, Habib; HASSAN, Ahlam, DE LA GUÍA, Elena. **New technologies to improve patient rehabilitation**: 4th workshop, REHAB 2016, Lisbon, Portugal, October 13-14, 2016, revised selected papers. Springer, 2019. v. 1002. p. 84-97. Disponível em: 10.1007/978-3-030-16785-1_7. Acesso em: 26 nov. 2021. (Communications in Computer and Information Science).

KHATRI, Abdul Rafay. Overview of fault tolerance techniques and the proposed TMR generator tool for FPGA designs. **International Journal of Advanced Computer Science and Applications**, v. 11, n. 4, 2020. Disponível em: 10.14569/IJACSA.2020.0110497. Acesso em: 29 maio. 2020.

KOYANAGI, Fernando. Do you know about ESP32 ADC adjustment? **Instructables**, 2019. Disponível em: https://www.instructables.com/Do-You-Know-About-ESP32-ADC-Adjustment/. Acesso em: 5 jun. 2020.

LACERDA, Vanessa de Sousa; BRAGA, Ana Carolina Aguirres. **Uso de smartphones em dupla-tarefa aumenta riscos de quedas e de acidentes em idosos**: resultados de uma revisão sistemática. 2021. Trabalho de conclusão de curso (Graduação em Fisioterapia) – Faculdade de Fisioterapia, Universidade Federal do Mato Grosso do Sul, Mato Grosso do Sul, 2021.

LAPRIE, Jean-Claude. Dependable computing and fault tolerance: concepts and terminology. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 15., 1985, Ann Arbor. **Anais** [...]. Nova Jersey: IEEE, 1985. p. 2-11.

LEE, Peter Alan; ANDERSON, Thomas. **Fault tolerance**: principles and practice. Englewood Cliffs, Nova Jersey: Prentice-Hall, 1981. v.3. (Dependable Computing and Fault-Tolerant Systems).

LUBASZEWSKI, Marcelo Soares. Teste e Projeto Visando o Teste de Circuitos e Sistemas Integrados. *In*: REIS, Ricardo Augusto da Luz. **Concepção de circuitos integrados**. Porto Alegre: Editora Sagra Luzzatto, 2002. v. 2. p. 167-189(páginas do capítulo). (Série Livros didáticos do Instituto de Informática da UFRGS).

LYONS, Richard.; VANDERKULK, Wouter. The Use of Triple-Modular Redundancy to Improve Computer Reliability. **IBM Journal of Research and Development**, v. 6, n. 2, 1962. Disponível em: 10.1147/rd.62.0200. Acesso em: 28 fev. 2021.

MCCLUSKEY, Edward. Built-in self-test techniques. **IEEE Design & Test of Computers**, v. 2, n. 2, 1985. Disponível em: 10.1109/MDT.1985.294856. Acesso em: 14 maio. 2020.

MEUCCI, Rodrigo; RUNZER-COLMENARES, Fernando; PARODI, José; MOLA, Christian Loret de. Falls among the elderly in Peruvian Andean Communities and the rural far South of Brazil: prevalence and associated factors. **Journal of community health**, v. 45, n. 2, 2020. Disponível em: 10.1007/s10900-019-00751-5. Acesso em: 5 jul. 2021.

NELSON, Victor. Fault-tolerant computing: fundamental concepts. **Computer**, v. 23, n. 7, 1990. Disponível em: 10.1109/2.56849.Acesso em: 29 jan. 2020.

OZCAN, Koray; MAHABALAGIRI, Anvith Katte; CASARES, Mauricio; VELIPASALAR, Senem. Automatic fall detection and activity classification by a wearable embedded smart camera. **IEEE journal on emerging and selected topics in circuits and systems**, v. 3, n. 2, 2013. Disponível em: 10.1109/JETCAS.2013.2256832. Acesso em: 29 jul. 2020.

PAVLIDIS, Antonios; LOUËRAT, Marie-Minerve; FAEHN, Eric; KUMAR, Anand; STRATIGOPOULOS, Haralampos-G. SymBIST: Symmetry-based analog and mixed-signal built-in self-test for functional safety. **IEEE Transactions on Circuits and Systems I**: Regular Papers, v. 68, n. 6, 2021. Disponível em: 10.1109/TCSI.2021.3067180. Acesso em: 10 ago. 2021.

PRADHAN, Dhiraj. **Fault-tolerant computer system design**. Nova Jersey: Prentice Hall, 1996.

SALEH, Majd; JEANNÈS, Régine Le Bouquin. Elderly fall detection using wearable sensors: a low cost highly accurate algorithm. **IEEE Sensors Journal**, v. 19, n. 8, 2017.

SIQUEIRA, Fernando Vinholes; Facchini, Luiz Augusto; SILVEIRA, Denise Silva da; PICCINI, Roberto Xavier; TOMASI, Elaine; THUMÉ, Elaine; SILVA, Suele Manjourany, DILÉLIO, Alitéia. Prevalence of falls in elderly in Brazil: a countrywide analysis. **Cadernos de Saúde Pública**, v. 27, n. 9, 2011. Disponível em: https://doi.org/10.1590/S0102-311X2011000900015. Acesso em: 6 mar. 2020.

VAN THANH, Pham; TRAN, Duc-Tan; NGUYEN, Dinh-Chinh; ANH, NGUYEN Duc Ahn; DINH, Dang Nhu; EL-RABAIE, Sayed; SANDRASEGARAN, Kumbesan. Development of a real-time, simple and high-accuracy fall detection system for elderly using 3-DOF accelerometers. **Arabian Journal for Science and Engineering**, v. 44, n. 4, 2019. Disponível em: https://doi.org/10.1007/s13369-018-3496-4. Acesso em: 6 mar. 2020.

WADSACK, Ronald. Fault Modelling and Logic Simulation of CMOS and MOS Integrated Circuits. **The Bell System Technical Journal**, v. 57, n. 5, 1978. Disponível em: 10.1002/j.1538-7305.1978.tb02106.x.Acesso em: 28 ago. 2021.

WANDERLEY, Preite Sobrinho. Mortes por queda quadruplicam e se aproximam do número de homicídios em SP. **Uol**, São Paulo, 2019. Disponível em:

https://noticias.uol.com.br/saude/ultimas-noticias/redacao/2019/07/07/mortes-por-queda-quadruplicam-e-se-aproximam-do-numero-de-homicidios-em-sp.htm. Acesso em: 6 mar. 2020. Notícias.

WANG, Fu-Tai; CHAN, Hsiao-Lung; HSU, Ming-Hung; LIN, Cheng-Kuan; CHAO, Pei-Kuang; CHANG, Ya-Ju. Threshold-based fall detection using a hybrid of tri-axial accelerometer and gyroscope. **Physiological measurement**, v. 39, n. 10, 2018. Disponível em: 10.1088/1361-6579/aae0eb. Acesso em: 19 nov. 2020.

WENSLEY, John; LAMPORT, Leslie; GOLDBERG, Jack; GREEN, Milton; LEVITT, Karl; MELLIAR-SMITH, Peter Michael; SHOSTAK, Robert; WEINSTOCK, Charles. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. **Proceedings of the IEEE**, v. 66, n. 10, 1978. Disponível em: 10.1109/PROC.1978.11114. Acesso em:19 nov. 2020.

YHDEGO, Haben; LI, Jiang; MORRISON, Steven; Audette, Michel; PAOLINI, Christopher; SARKAR, Mahasweta; OKHRAVI, Hamid. Towards musculoskeletal simulation-aware fall injury mitigation: transfer learning with deep CNN for fall detection. *In*: MODELING AND SIMULATION IN MEDICINE SYMPOSIUM, 19.; SPRING SIMULATION CONFERENCE, 19., 2019, Tucson. **Anais** [...]. San Diego: Society fot Computer Simulation International, 2019. Disponível em: 10.23919/SpringSim.2019.8732857. Acesso em: 4 abr. 2020. p. 1-12.