Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DISPOSITIVO DE SEGURANÇA E MONITORAMENTO PARA APARELHOS ELETRÔNICOS II

João Pedro de Oliveira Silva

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Ricardo Zelenovsky

Brasília
2024

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DISPOSITIVO DE SEGURANÇA E MONITORAMENTO PARA APARELHOS ELETRÔNICOS II

João Pedro de Oliveira Silva

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Ricardo Zelenovsky (Orientador)
ENE/UnB

Prof. Dr. Daniel Chaves Café
ENE/UNB

Prof. Dr. Eduardo Peixoto Fernandes da Silva
ENE/UNB

Prof. Dr. João Luiz Azevedo de Carvalho
Coordenador do Curso de Engenharia da Computação

Brasília, 29 de janeiro de 2024

Dedicatória

Dedico este trabalho aos meus pais, que sempre se esforçaram para que eu tivesse o melhor da educação e da decência.

Agradecimentos

Agradeço primeiramente a Deus, que é o motivo do meu viver, foi Ele quem fez com que meus objetivos fossem alcançados, durante todos os meus anos de estudo. Aos meus pais e minha irmã, por todo o carinho, paciência, suporte e amor durante toda essa jornada.

Agradeço ao meu orientador Ricardo Zelenovsky, por toda a paciência, tempo e esforço dedicado em me orientar neste trabalho.

Agradeço a todos os meus amigos, por me apoiarem e estarem comigo durante esse tempo dificil de curso, pelos momentos de desconstração e principalmente em nosso crescimento na fé e pessoal.

Agradeço a todos os meus colegas de curso, que me apoiaram e proporcionaram um ambiente enriquecedor e colaborativo.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

O crescente índice de crimes demanda por medidas de segurança, as quais muito se beneficiam com a tecnologia moderna. Este trabalho tem por objetivo propor soluções para o rastreamento e monitoramento de objetos, além de melhorar a segurança e facilitar a recuperação de bens em situações de roubo, furto ou perda. O sistema é fundamentado no uso do microcontrolador MSP430F5529LP e incorpora alguns módulos - GPS, RTC, Wi-Fi, giroscópio/acelerômetro e memória flash - os quais operam de forma integrada para o rastreamento eficiente dos objetos.

O funcionamento do sistema inicia com o giroscópio e GPS, os quais identificam o estado de movimento do objeto. Adiante, com a implementação de um algoritmo, o qual determina se o objeto está sendo indevidamente removido ou furtado. Após essa detecção, o sistema captura e armazena informações cruciais como localização, data e hora na memória flash.

A fim de obter uma interação e um controle remoto do sitema, foi implementado um webservice utilizando Flask. Por meio desse webservice é permitido obter uma comunicação remota e eficiente com o hardware.

Por fim, foi desenvolvido uma interface de usuário utilizando o PySimpleGUI (uma biblioteca de Python), que se conecta ao dispositivo de segurança proposto via Wi-Fi, oferecendo uma comunicação simplificada e em tempo real.

Palavras-chave: Segurança, Microcontrolador, localização GPS, Comunicação Wi-Fi, Interface de usuário.

Abstract

The increasing crime rate calls for security measures, which greatly benefit from modern technology. This work aims to propose solutions for tracking and monitoring objects, as well as enhancing security and facilitating the recovery of assets in situations of theft, burglary, or loss. The system is based on the use of the MSP430F5529LP microcontroller and incorporates various modules - GPS, RTC, Wi-Fi, gyroscope/accelerometer, and flash memory - which work together for efficient object tracking.

The system's operation begins with the gyroscope and GPS, which detect the object's motion state. Subsequently, an algorithm is implemented to determine whether the object is being unlawfully removed or stolen. Following this detection, the system captures and stores crucial information such as location, date, and time in the flash memory.

To achieve interaction and remote control of the system, a webservice using Flask was implemented. This webservice allows for remote and efficient communication with the hardware.

Lastly, a user interface was developed using PySimpleGUI (a Python library), connecting to the proposed security device via Wi-Fi, offering simplified and real-time communication.

Keywords: Security, Microcontroller, GPS location, Wi-Fi Communication, User interface.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ADC Analog-Digital Converter.

AGPS Assisted Global Positioning System.

Bot Robot.

CPU Central Processing Unit.

DMA Direct Memory Access.

DMP Data Mangement Platform.

EEPROM Electrically Erasable Programmable Read-Only Memory.

GLONASS Global Navigation Satellite System.

GND Graduated Neutral Density filter.

GNSS Global Navigation Satellite System.

GPRS General Packet Radio Service.

GPS Global Positioning System.

GSM Global System for Mobile.

HTTP Hyper Text Transfer Protocol.

I2C Inter-Integrated Circuit.

IOT Internet of Things.

LBS Location-Based Services.

MPU Microprocessor Unit.

RAM Random Access Memory.

REST Representational State Transfer.

RTC Real-Time Clock.

SMTP Simple Mail Transfer Protocol.

SPI Serial Peripheral Interface.

UART Universal Asynchronous Receiver/Transmitter.

URI Uniform Resource Identifier.

USB Universal Serial Bus.

VCC Constant Current Voltage.

Capítulo 1

Introdução

1.1 Ambientação

A proteção do patrimônio é crucial, uma vez que furtos e roubos podem causar danos financeiros e emocionais significativos.

Em um mundo cada vez mais conectado pela tecnologia e onde as inovações digitais invadem o dia a dia, a segurança é uma grande preocupação. Ao mesmo tempo em que dispositivos e sistemas inteligentes aumentaram a praticidade e a eficiência, eles também criaram novas vulnerabilidades, tanto no âmbito virtual quanto no físico.

Tanto empresas quanto indivíduos estão procurando métodos inovadores para evitar a perca de seus pertences devido ao aumento dos casos de furtos e roubos. Câmeras de vigilância e rastreadores GPS são exemplos de sistemas de monitoramento que estão se tornando cada vez mais acessíveis, sofisticados e versáteis, o que permite que até mesmo os usuários comuns protejam seus pertences pessoais.

Além de sua função inicial de navegação, as aplicações de GPS agora são usadas para rastrear itens de valor, como automóveis, telefones celulares e até mesmo aparelhos domésticos.

Para enfrentar esse desafio contemporâneo, há uma crescente demanda por soluções inovadoras e sofisticadas. A escolha deste assunto foi motivada pela constatação dos problemas de segurança no cotidiano e que não há uma ampla disponibilidade de aparelhos de segurança eficientes. Ao combinar conceitos de sistemas embarcados com estratégias de segurança, busca uma oportunidade de fazer uma diferença significativa.

1.2 Descrição do problema

De que maneira um aparelho de segurança pequeno e sofisticado pode ajudar na prevenção de furtos e na recuperação de objetos perdidos? Esta é a questão principal que se pretende

discutir. O foco é desenvolver um dispositivo resistente, discreto e capaz de prevenir e resgatar objetos valiosos.

Este estudo se desdobrará em várias etapas para melhorar e inovar a segurança de bens pessoais. Começará com um estudo abrangente sobre o assunto, seguirá com a criação e teste do dispositivo e, finalmente, chegará às conclusões e recomendações finais.

1.3 Proposta da solução

A motivação por trás deste projeto é desenvolver um sistema sofisticado de monitoramento e segurança, não apenas para eletrodomésticos, mas também expandindo sua aplicação para veículos, dispositivos móveis e outros itens de valor. Ao rastrear esses objetos, têm-se o intuito de ampliar a probabilidade de recuperá-los em situações adversas como furtos ou roubos, proporcionando assim uma maior tranquilidade ao proprietário.

A proposta é desenvolver um hardware compacto e discreto que possa ser incorporado em diversos tipos de dispositivos domésticos, como aparelhos de som, televisões, máquinas de lavar roupa etc. Este dispositivo será ativado em um incidente de furto e armazenará informações importantes, como data, hora e coordenadas geográficas. O proprietário também receberá mensagens de atualização sobre o estado do dispositivo quando o sinal Wi-Fi estiver disponível.

Este item, ao ser levado, funcionará como uma isca tecnológica, similar ao conceito de "*honeypot*" no campo da segurança cibernética. Inspirado por essa analogia, foi decidido nomear o projeto de "ARAPUCA".

O projeto se encontra no repositório online do *Github* [1].

1.3.1 O que existe no mercado e preços

O projeto em questão não é algo único, existem diversas opções no mercado, deste algumas simples até outras caras e sofisticadas.

Opção 1

O dispositivo de rastreamento GPS mostrado na *Figura* 1.1 (objeto em forma de uma pequena caixa preta) custa R$ 179,99 e oferece uma solução eficiente e segura para localização de itens perdidos. Com configuração simples e intuitiva, os usuários podem registrar e localizar facilmente seus itens personalizando ícones e renomeando itens no aplicativo Find My. A tecnologia de banda ultralarga garante rastreamento preciso, encontrando diretamente objetos perdidos. Além disso, o aparelho possui alto-falantes de

posicionamento de som integrados e suporta controle de voz, aumentando sua praticidade para o uso diário.

Priorizando a segurança e a privacidade, todas as comunicações com a rede Find My são anônimas e criptografadas, garantindo que os dados de localização e histórico nunca sejam armazenados. A vasta rede Find My da Apple expande a cobertura de pesquisa e o Modo Perdido fornece notificações automáticas quando rastreadores são detectados. Apresentando uma bateria interna substituível que dura mais de um ano.

Figura 1.1: Mini Rastreador Gps Inteligente Smart Sem Fio Segurança

Fonte: Amazon.com

Disponivel em: Amazon. Acesso em: 09 de novembro de 2023

Opção 2

Custando R$ 149,05, este rastreador GPS avançado, conforme mostrado na *Figura* 1.2, projetado para fornecer localização precisa em tempo real. Com funcionalidades abrangentes, é ideal para diversas aplicações, como segurança pessoal e monitoramento de veículos. O dispositivo oferece não apenas rastreamento em tempo real, mas também grava um histórico de rotas, permitindo aos usuários um acompanhamento detalhado dos movimentos.

Além disso, o GF-22 possui funcionalidades como SOS para emergências, escuta ambiental e gravação, e a útil cerca eletrônica, que alerta o usuário se o dispositivo sair de uma área predeterminada. Suas opções de economia de bateria, a capacidade de ser carregado via USB, e o monitoramento da carga via aplicativo, asseguram eficiência e praticidade. A precisão do posicionamento, que varia de 10 a 50 metros, juntamente com o suporte a GPS, Wi-Fi e LBS, fazem deste rastreador uma ferramenta confiável para uma gama de necessidades de rastreamento.

Figura 1.2: Localizador GPS GF-22 Rastreador anti-roubo sem fio inteligente de posicionamento preciso

Fonte: Amazon.com

Disponivel em: Amazon. Acesso em: 09 de novembro de 2023

Opção 3

Por R$242,40, o APPEGO GPS VEÍCULAR *Figura* 1.3, representa uma solução de monitoramento veicular de alta tecnologia, ideal para quem busca segurança e controle sobre seus veículos. Este dispositivo utiliza uma tecnologia híbrida que combina GPS, AGPS (uma versão aprimorada do GPS) e LBS para monitoramento, juntamente com comunicação GSM/GPRS, garantindo uma localização precisa e confiável do seu veículo. Conectado facilmente a smartphones, tablets ou computadores, o APPEGO oferece tranquilidade e controle total, seja para carros, motos, ônibus, caminhões, barcos, bicicletas ou patinetes elétricos.

O que diferencia este dispositivo é a sua capacidade de capturar coordenadas de redes de satélite GNSS (GPS e GLONASS) e transmitir as informações para o aplicativo do usuário através da rede telefônica 2G usando um cartão SIM m2m incluído.

Figura 1.3: Rastreador Appego GPS Veicular

Fonte: Magazine Luiza

Disponivel em: Magazine Luiza. Acesso em: 09 de novembro de 2023

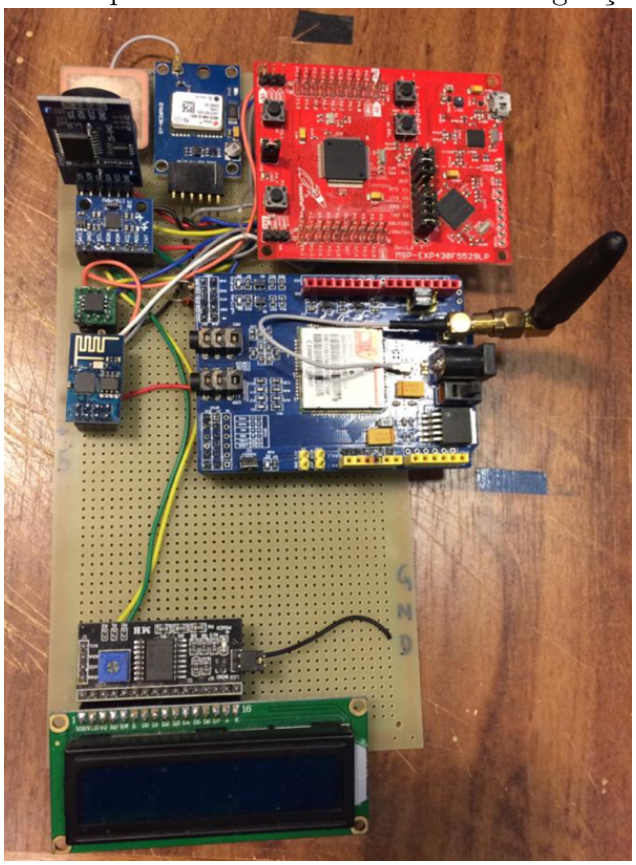
1.4 Projeto anterior

O trabalho aqui apresentado é uma continuação do projeto anterior [2], que desenvolveu o hardware e realizou os primeiros ensaios de funcionamento do dispositivo. Nesta primeira etapa a atenção foi dedicada principalmente às comunicações e ao funcionamento dos diversos componentes do sistema. Devido a dificuldade em operar o GPRS e o Wi-Fi, esta versão do projeto empregou a comunicação pela porta UART. A UART é uma tecnologia amplamente reconhecida e utilizada para a transmissão serial de dados entre dispositivos, garantindo transferências de informação de forma robusta e eficiente [3]. Contudo, essa maneira de comunicação, ainda que eficaz, tem suas limitações, especialmente quando se pensa em flexibilidade e adaptabilidade a cenários modernos de aplicação.

Com a evolução da tecnologia e as crescentes necessidades de sistemas mais adaptáveis e menos intrusivos [4], se justifica a atual etapa deste projeto .

A *Figura* 1.4 apresenta a versão de hardware presente no projeto anterior.

Figura 1.4: Hardware da primeira versão do sistema de segunça e monitoramento



Fonte: Trabalho anterior

Disponivel em: [2] Acesso em: 08 de novembro de 2023

Conforme destacado previamente, nesta etapa a utilização do GPRS e Wi-Fi não foi efetivamente implementada, sendo que, na fase subsequente, o GPRS foi desconsiderado para a implementação. Em contrapartida, optou-se por adotar o ESP32 como alternativa para a comunicação via Wi-Fi. Essa decisão foi tomada em função das dificuldades operacionais encontradas na etapa anterior, especificamente em relação à operação do GPRS e Wi-Fi com o ESP8266.

1.5 Projeto atual

Neste atual trabalho, foram implementados avanços significativos por meio de uma série de testes de hardware e análises de consistência, visando a otimização do dispositivo Arapuca. Um aspecto chave desses aperfeiçoamentos foi o desenvolvimento de um sistema de estados, que facilitou o controle e a operacionalidade do dispositivo. Além disso, o sistema de alerta, que emprega tanto o MPU quanto o GPS, passou por refinamentos,

resultando em uma eficácia e precisão aprimoradas na detecção de possíveis indícios de furto.

Além que nesta nova etapa, foi construída de forma completa a transição da comunicação, para o usuário, de UART para um modelo de comunicação via Wi-Fi, utilizando os protocolos HTTP e SMTP. A comunicação Wi-Fi permite que todas as mensagens geradas pelo sistema, incluindo status, alertas e outras informações pertinentes, sejam transmitidas sem a necessidade de conexões físicas [5]. Esta nova abordagem proporciona maior mobilidade e flexibilidade.

A adoção desta abordagem visa alinhar o projeto às tendências atuais em sistemas embarcados, assegurando que as soluções propostas estejam em consonância com os padrões tecnológicos de ponta e as expectativas modernas de desempenho e aplicabilidade [6].

Ademais, foi desenvolvida uma interface em Python, capaz de interagir com o dispositivo Arapuca através de conexão Wi-Fi. Essa interface permite o envio de comandos para o dispositivo de forma remota, além de fornecer um mecanismo eficiente para monitoração em tempo real.

Em resumo a *Tabela* 1.1 apresenta a diferença entre as duas versões do projeto *Arapuca*.

Tabela 1.1: Comparação entre os projetos

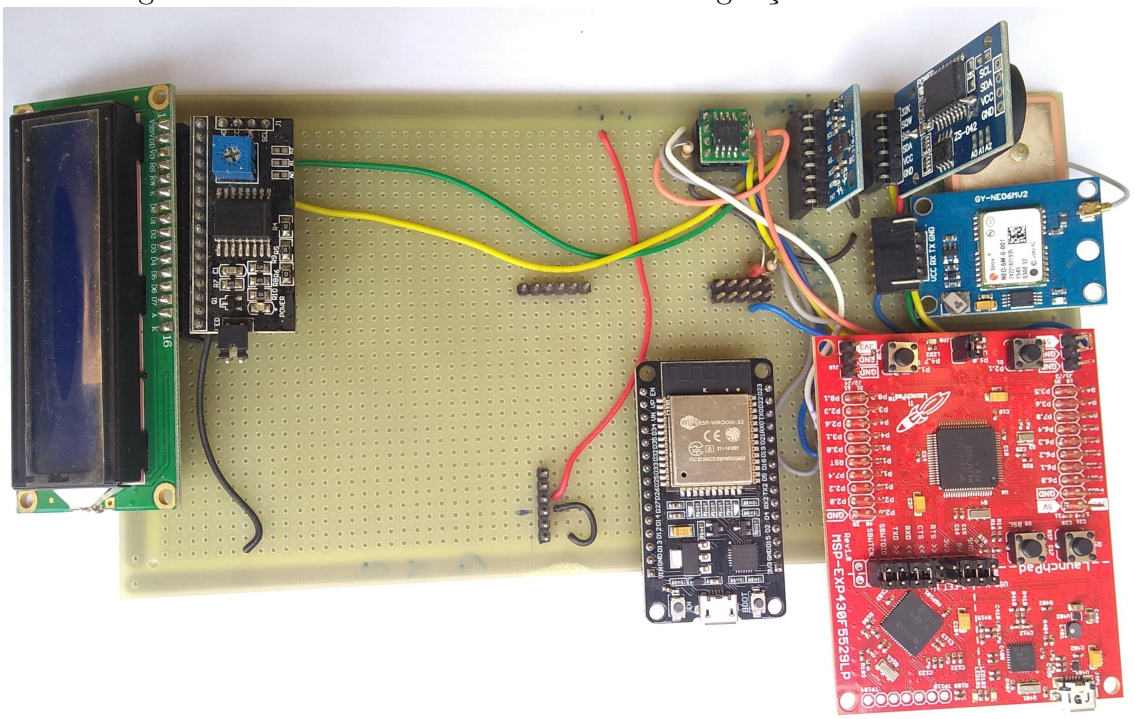
Projeto Anterior	Projeto Atual
• Desenvolvimento do Hardware. • Ensaios de funcionamento do dispositivo. • Não foi implementado o uso do GPRS e Wi-fi, porém foi realizado diversos testes com eles. • Comunicação com o usuário através da porta UART do MSP.	• Comunicação via Wi-Fi utilizando ESP32. • Sistema de estados. • Sistema de alerta, que emprega tanto o MPU quanto o GPS, passou por refinamentos. • Comunicação utilizando uma interface em Python com Rest, com os protocolos HTTP e SMTP. • Alerta de mudança de estado por whatsapp.

Fonte: Autoria própria.

1.6 Objetivo

Este trabalho tem como objetivo desenvolver um dispositivo eletrônico que possa auto-declarar sua localização, data e hora. Para isso, foi incorporado um conjunto de sensores como módulo Wi-Fi, relógio de tempo real (RTC), giroscópio e acelerômetro, módulo GPS e memória flash para armazenamento das posições. Para garantir um funcionamento autonomo, foi adcionado uma bateria que permite ao dispositivo funcionar de forma independente durante um período de tempo. No centro do sistema está um microcontrolador MSP430F5529LP da Texas Instruments e um ESP32 da Espressif Systems, responsável pela transmissão de mensagens em longas distâncias. Todos estes componentes foram cuidadosamente integrados numa única placa, culminando no protótipo final do projeto, conforme ilustrado na *Figura* 1.5.

Figura 1.5: Hardware atual do sistema de segunça e monitoramento



Fonte: Autoria própria.

1.7 Metodologia

Essa etapa de continuação do projeto envolveu uma análise aprofundada de tecnologias emergentes e suas aplicações em segurança.

Na fase de desenvolvimento de hardware, o projeto focou na integração de componentes essenciais, como um módulo GPS, módulo Wi-Fi para comunicação remota, um

módulo giroscópio e acelerômetro, Real-Time Clock e memória flash. A seleção desses componentes foi baseada em critérios de eficiência, custo-benefício e confiabilidade.

O desenvolvimento de software envolveu a programação do microcontrolador MSP430, escolhido por sua eficiência energética e capacidade de processamento adequada para aplicações de IOT. A programação focou em algoritmos para otimizar a coleta e transmissão de dados do GPS, bem como na comunicação eficiente com a memória flash e o módulo Wi-Fi [7].

Além disso, foi desenvolvida uma interface em Python com PySimpleGUI e um web service usando Python e Flask. Estes componentes foram essenciais para estabelecer uma conexão remota eficiente com o hardware, permitindo uma interação mais intuitiva e um gerenciamento de dados mais robusto [8], [9].

Para a integração e testes do sistema, o dispositivo foi montado e submetido a uma série de testes funcionais, incluindo a verificação da estabilidade do hardware e a eficácia da comunicação de dados via Wi-Fi. Testes de campo adicionais foram realizados para avaliar a precisão do monitoramento GPS e a confiabilidade da comunicação remota em diferentes cenários [10].

A fase de análise de dados focou na avaliação do desempenho do dispositivo em termos de precisão do monitoramento GPS e eficiência da comunicação de dados, utilizando métodos de análise quantitativos e qualitativos para compreender integralmente o funcionamento do dispositivo em condições reais de uso [11].

Finalmente, foram abordadas as limitações do estudo e potenciais áreas para futuras pesquisas, visando o contínuo aprimoramento e inovação na área de dispositivos de segurança e monitoramento GPS.

1.8 Descrição do Texto escrito

Capítulo 1 - Introdução: Neste capítulo, são introduzidas as noções essenciais referentes ao projeto, compreendendo os seus propósitos, a abordagem adotada e a sua estrutura.

Capítulo 2 - Embasamento teórico: Apresenta as abordagens, estratégias, metodologias e recursos a serem empregados no desenvolvimento do modelo de resolução do problema.

Capítulo 3 - Fundamentação de hardware: Apresenta breves explicações a respeito dos elementos que compõem o sistema de segurança e monitoração.

Capítulo 4 - Proposta de software: Contém uma ilustração dos estados de funcionamento do projeto, interface do software e seu funcionamento.

Capítulo 5 - Ensaios: Este capítulo contém as simulações e resultados sobre a realização do projeto.

Capítulo 6 - Conclusões: Neste trecho, estão descritas as conclusões obtidas a respeito do projeto, com uma avaliação da eficácia dos resultados, destacando as vantagens e limitações do projeto. Além disso, são sugeridas abordagens para aprimoramento do projeto.

Capítulo 2

Embasamento teórico

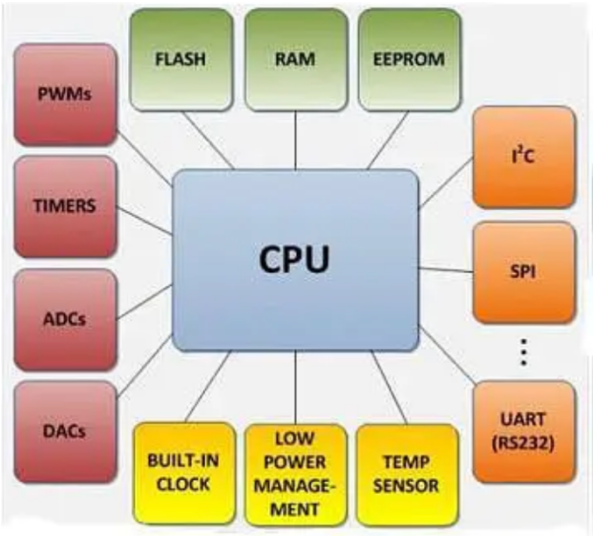
Os fundamentos teóricos e as tecnologias relevantes para este estudo serão abordados neste capítulo. Serão discutidos assuntos como microcontroladores, sistemas embarcados, entre outros.

2.1 Microcontroladores

Microcontroladores são componentes importantes de muitos sistemas eletrônicos, servindo como cérebro de vários dispositivos. São pequenos processadores contendo CPU, memória e periféricos de entrada/saída em um único circuito integrado. Esta integração permite que microcontroladores executem tarefas específicas em sistemas embarcados. A eficiência, versatilidade e baixo custo dos microcontroladores os tornam ideais para inúmeras aplicações, impulsionando a inovação em áreas tão diversas como a Internet das Coisas (IoT), onde desempenham um papel vital nas comunicações e no processamento de dados [12].

Além disso, os microcontroladores são fundamentais na implementação de sistemas de controle e monitoramento, graças à sua capacidade de processar sinais digitais e analógicos, controlar dispositivos e comunicar-se com outros sistemas. São programáveis, permitindo que os desenvolvedores criem soluções personalizadas para necessidades específicas. Com o avanço da tecnologia, os microcontroladores estão se tornando cada vez mais poderosos e eficientes, abrindo novos horizontes para aplicações inovadoras em campos como automação residencial, sistemas de segurança, dispositivos médicos e muito mais. A evolução contínua dos microcontroladores mostra sua importância e versatilidade em um mundo cada vez mais conectado e automatizado [13]. A *Figura* 2.1 mostra os componentes de um microcontrolador, seus periféricos, protocolos entre outros recursos.

Figura 2.1: Componentes de um microcontrolador



Fonte: Embarcados

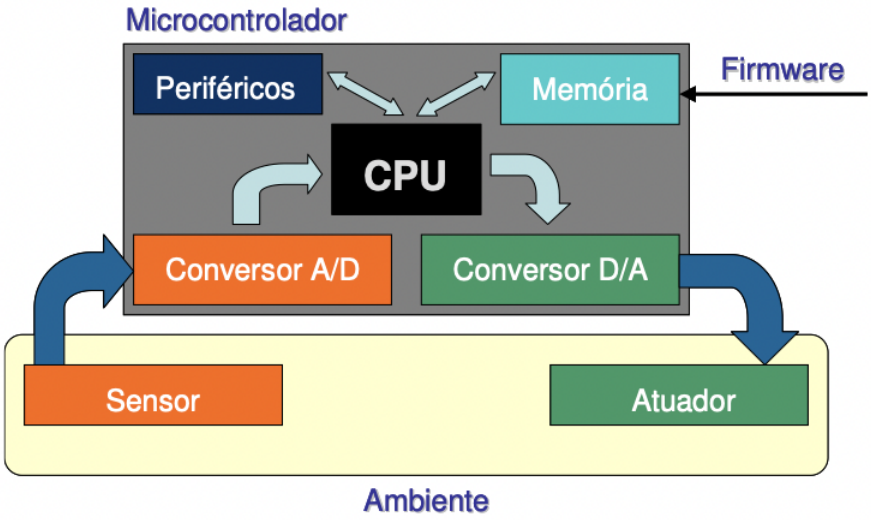
Disponivel em: Embarcados. Acesso em: 08 de novembro de 2023

2.2 Sistemas Embarcados

Os sistemas embarcados são uma parte importante da tecnologia moderna, integrando hardware e software para executar funções específicas dentro de um sistema maior. São projetados para executar uma série de tarefas especializadas, muitas vezes com requisitos em tempo real, e são encontrados em dispositivos que vão desde eletrodomésticos e automóveis até sistemas industriais complexos. A natureza compacta e eficiente dos sistemas embarcados os torna ideais para operações onde o espaço e a eficiência energética são críticos. Esses sistemas são otimizados para desempenho e consumo de energia e geralmente operam em ambientes com recursos limitados, como memória e poder de processamento [14].

A *Figura* 2.2 mostra um diagrama de um típico sistema embarcado, no qual as entradas são obtidas pelos sensores e conversores A/D, sendo processadas pelo programa embarcado que gera as saídas por meio dos conversores D/A e atuadores.

Figura 2.2: Diagrama de um sistema embarcado



Fonte: Max Pezzin

Disponivel em: Max Pezzin. Acesso em: 08 de novembro de 2023

À medida que a Internet das Coisas (IoT) avança, os sistemas embarcados tornaram-se mais predominantes, tornando-se a espinha dorsal de muitos dispositivos conectados. Eles desempenham um papel vital ao permitir a comunicação e automação inteligentes em todos os setores, impulsionando a inovação e melhorando a eficiência em muitos aspectos da vida diária [15].

2.3 SMTP

O SMTP é o protocolo padrão para transferência de e-mail na Internet. É responsável por enviar mensagens de um servidor para outro, garantindo que o e-mail chegue corretamente ao destinatário. O SMTP é uma parte importante da infraestrutura de e-mail, pois ele permite que os servidores de e-mail se comuniquem entre si para entregar mensagens. É executado na camada de aplicação do conjunto de protocolos da Internet e é usado para enviar mensagens, mas não para recebê-las. Para isso, outros protocolos como POP3 ou IMAP precisam ser usados [16].

2.4 HTTP

O HTTP é o protocolo básico de comunicação usado para transferir dados na World Wide Web. Este protocolo que define como as mensagens são formatadas e transmitidas e quais ações os servidores e navegadores devem realizar em resposta a vários comandos. O HTTP

é um protocolo sem estado, o que significa que cada solicitação do cliente para o servidor é tratada de forma independente, sem conhecimento de solicitações anteriores. Isto tornou a rede mais eficiente, contudo criaram-se desafios de segurança e privacidade que foram abordados em versões seguintes do protocolo, como HTTPS [17].

A evolução do HTTP tem sido objeto de pesquisas em um esforço para melhorar a segurança, eficiência e velocidade do protocolo. Por exemplo, a nova versão do HTTP/2 introduz melhorias significativas de desempenho, permitindo um carregamento de página mais rápido e uma comunicação mais eficiente entre clientes e servidores. Estas melhorias são fundamentais para apoiar a crescente procura de serviços de Internet rápidos e seguros, especialmente à medida que aumenta a utilização de aplicações Web e móveis [18].

2.5 Flask

Flask é um microframework para desenvolvimento web usando Python, conhecido por sua simplicidade e flexibilidade. Ao contrário de estruturas web mais pesadas, o Flask fornece as ferramentas básicas necessárias para iniciar um projeto web, como roteamento de URL, integração de banco de dados e suporte a modelos, mas deixa o esquema e outras extensões para os desenvolvedores. Isso permite que os desenvolvedores construam aplicações web de forma rápida e eficiente, com a liberdade de escolher as melhores ferramentas e práticas para seus projetos específicos [19].

A filosofia do Flask é baseada na simplicidade e extensibilidade. Segue o princípio "menos é mais" e fornece uma base enxuta para a construção de aplicações web. Essa característica torna o Flask mais atraente para pequenos projetos e prototipagem rápida, mas também é poderoso o suficiente para ser usado em grandes aplicações web. Com uma comunidade ativa e uma vasta gama de extensões disponíveis, Flask continua a ser uma escolha popular entre os desenvolvedores web [20].

2.6 Rest

Um sistema REST é uma abordagem de arquitetura para comunicação entre sistemas em uma rede, como a Internet. Devido à sua simplicidade, escalabilidade e eficiência, são amplamente utilizados para criar interfaces de programação de aplicativos (APIs). Em um sistema REST, a interação entre cliente e servidor é realizada por meio de solicitações HTTP padrão. Cada solicitação é independente e o servidor não mantém o estado da sessão do cliente, tornando o REST uma abordagem sem estado. Isso torna o sistema REST mais escalável e mais fácil de manter do que as abordagens baseadas em estado. [21].

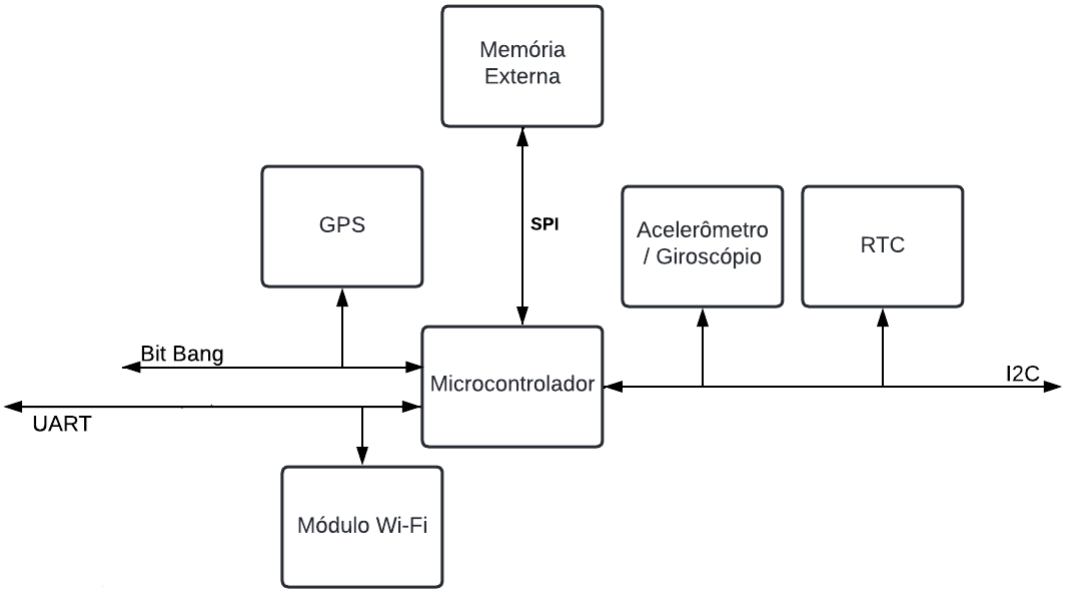
A arquitetura REST é baseada em recursos, onde cada recurso é identificado por um URI. As operações nesses recursos são realizadas usando métodos HTTP como GET, POST, PUT e DELETE, que correspondem a operações CRUD (criar, ler, atualizar, excluir). Esta abordagem padronizada promove a interoperabilidade entre diferentes sistemas e plataformas. Além disso, os sistemas REST podem retornar dados em vários formatos, como JSON ou XML, tornando-os flexíveis para diferentes tipos de aplicações. A popularidade dos sistemas REST se deve à sua simplicidade e eficiência, tornando-os uma escolha comum para o desenvolvimento de APIs web modernas. [22].

Capítulo 3

Fundamentação de hardware

Neste capítulo, é apresentado uma visão geral do projeto de hardware do sistema, detalhando algumas etapas de orientação e fornecendo explicações simplificadas sobre os módulos utilizados. São explorarados componentes eletrônicos essenciais, incluindo o MSP430F5529LP e seus protocolos (UART, I2C, SPI), a memória externa, o GPS, o acelerômetro, o Real-Time Clock e o módulo Wi-Fi. Na *Figura* 3.1, é apresentado um esquema abrangente das conexões dos barramentos dos periféricos com o microcontrolador. Em suma, este capítulo é dedicado às descrições dos componentes, o que permitirá uma compreensão mais aprofundada do sistema.

Figura 3.1: Esquemático completo das conexões dos componentes com o microcontrolador



Fonte: Autoria própria.

3.1 MSP430F5529LP

O microcontrolador MSP430F5529LP faz parte da família de produtos MSP430 da Texas Instruments e representa um marco importante em sistemas embarcados de baixo consumo de energia. Este microcontrolador tem sido objeto de pesquisas e aplicações em diversas áreas da eletrônica há muitos anos devido às suas características únicas e diferenciadas que atendem a uma ampla gama de necessidades industriais e acadêmicas.

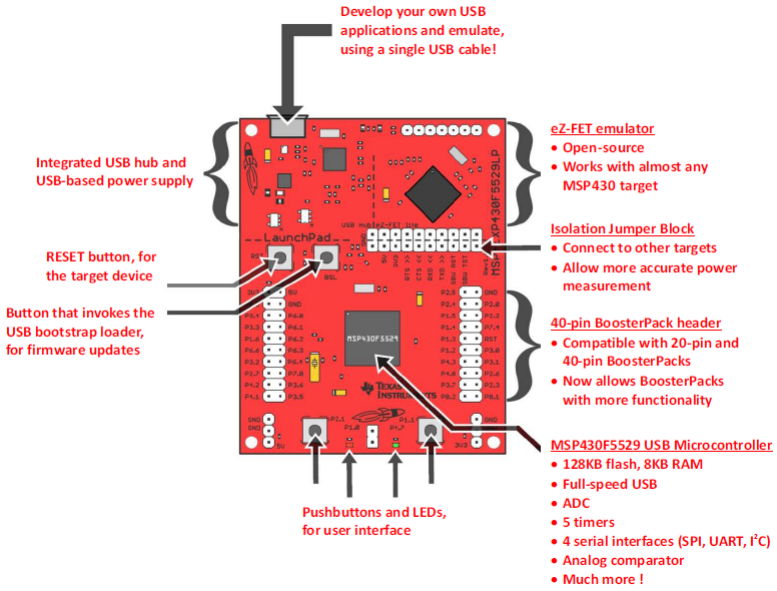
Do ponto de vista arquitetônico, o MSP430F5529LP funciona em arquitetura de 16 bits e vem com até 128 KB de memória flash e 8 KB de RAM. Essa combinação permite armazenar uma ampla gama de programas e executar algoritmos complexos. Além disso, vários periféricos integrados também destacam as suas funções de interface. Estas incluem interfaces USB, UART, SPI e I2C, permitindo a comunicação com uma variedade de dispositivos externos e facilitando a integração em sistemas mais complexos.

A estrutura modular do Texas Instruments MSP430F5529LP permite que vários módulos operem independentemente da CPU central. Isso facilita operações simultâneas de E/S sem sobrecarregar o processador. Além de melhorar o desempenho, também otimiza o consumo de energia, já que a CPU não precisa ser ativada para cada pequena tarefa.

O MSP430F5529LP se destaca não apenas por seus temporizadores de 16 bits, quatro portas seriais USCI, multiplicadores de hardware, controlador DMA, módulo RTC e 63 pinos, mas também por seu inovador conversor analógico-digital (ADC) de 12 bits.

A placa de desenvolvimento utilizada com o microcontrolador incorporado é ilustrada na *Figura* 3.2 de forma mais estruturada, na *Figura* 3.3 em sua forma física e a *Figura* 3.4 traz as conexões utilizadas no microcontrolador para controlar os periféricos do projeto.

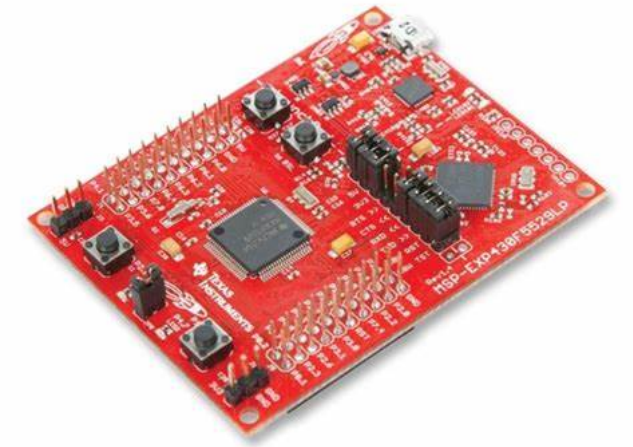
Figura 3.2: LaunchPad MSP430F5529LP e suas características



Fonte: Texas Instruments

Disponivel em: User's Guide. Acesso em: 08 de novembro de 2023

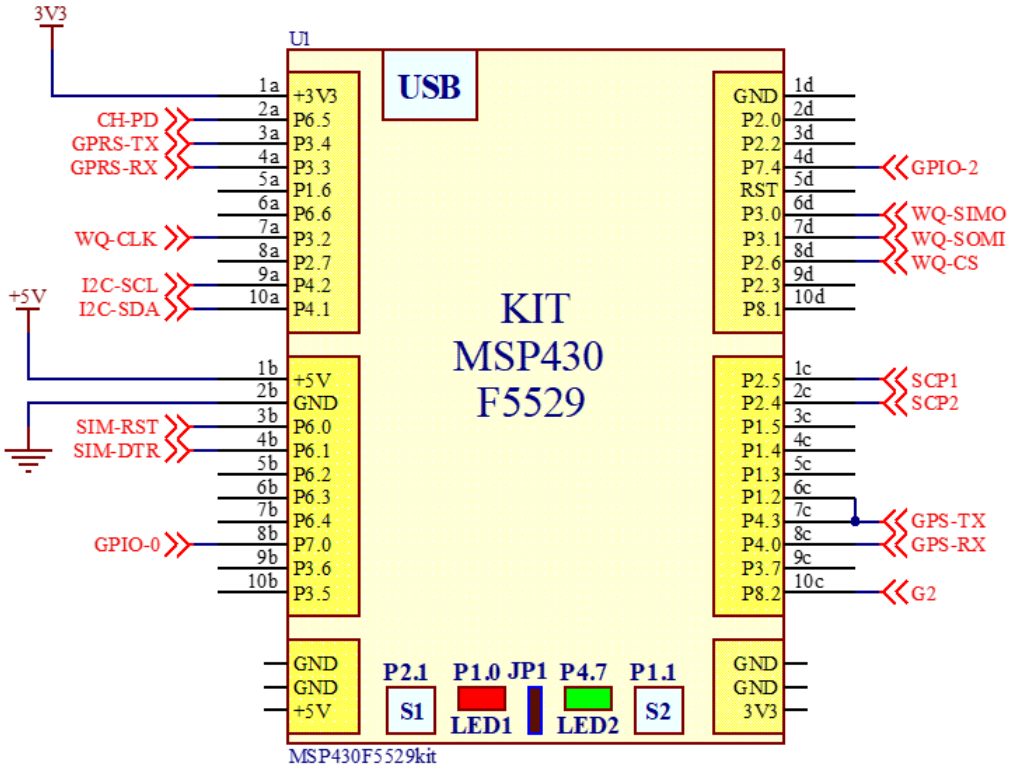
Figura 3.3: LaunchPad MSP430F5529LP



Fonte: Texas Instruments

Disponivel em: User's Guide. Acesso em: 08 de novembro de 2023

Figura 3.4: Conexões do MSP430F5529LP e os periféricos



Fonte: Autoria Própria.

3.2 Sensores

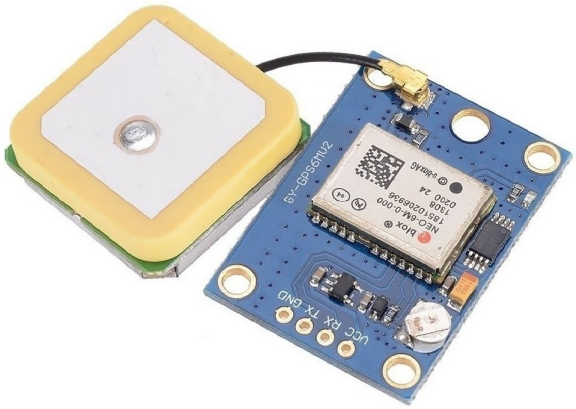
No âmbito deste projeto, além da utilização do microcontrolador MSP430F5529LP, são incorporados uma série de outros componentes essenciais para a sua completa funcionalidade. Estes incluem sensores especializados, um módulo GPS para posicionamento preciso, um RTC (relógio em tempo real) para gerenciamento em tempo real e uma interface Wi-Fi para suportar conectividade e comunicação de rede sem fio. Além disso, o projeto contará com recursos adicionais de memória para armazenamento de dados. Não menos importante, será implementada a funcionalidade de giroscópio e acelerômetro, que são ferramentas fundamentais para detecção de movimento e orientação.

3.2.1 GPS

O dispositivo de localização por GPS em questão é o modelo GY-GPS6MV2. Este não é apenas um módulo GPS comum, mas um que vem equipado com uma antena cerâmica de alta qualidade. Adicionalmente, ele também incorpora uma EEPROM, facilitando diversas aplicações. No que se refere à sua configuração de pinos, o GY-GPS6MV2 é equipado

com quatro deles: VCC, RX, TX e GND. A interface de comunicação do dispositivo é realizada por meio de uma conexão serial assíncrona, garantindo uma troca de dados eficiente. Um dos pontos fortes deste módulo é a sua precisão, que, combinada com sua facilidade de uso, o torna uma escolha excepcional para muitos projetos que requerem localização por GPS. A *Figura* 3.5 ilustra o modelo GPS utilizado no projeto.

Figura 3.5: Módulo GPS GY-GPS6MV2

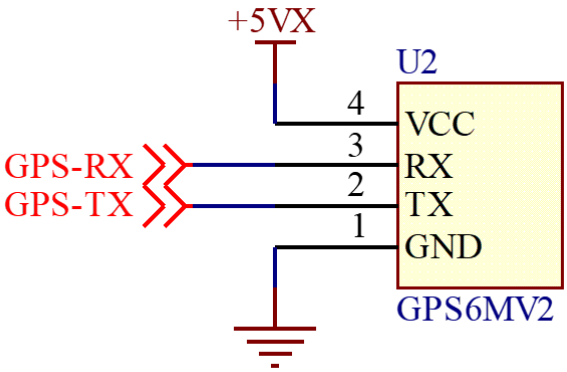


Fonte: HuInfinito

Disponivel em: Hu Infinito. Acesso em: 08 de novembro de 2023

Neste projeto, o módulo GPS e o microcontrolador se comunicam por meio do protocolo UART, porém foi adotado o Bit Banging, uma técnica de comunicação serial em que o software executa toda a comunicação simulando a transmissão e a recepção para acessar o GPS. A *Figura* 3.6 mostra os barramentos que o microcontrolador e o GPS utilizam.

Figura 3.6: Conexões do módulo GPS para o MSP430F5529LP



Fonte: Autoria Própria.

3.2.2 Memória

Conforme mencionado anteriormente, a capacidade de memória RAM do MSP430F5529LP é de 8 KB. Ao analisar sua estrutura interna, compreende-se que a sua capacidade efetiva para armazenar números inteiros, considerando-se de 16 bits, é limitada a 4 KB. Além disso, é fundamental entender que esta memória interna não serve apenas como depósito de dados. Parte dela é alocada para as variáveis utilizadas pelo programa e também para a gestão da pilha da CPU, o que torna sua capacidade efetiva ainda mais limitada.

Além dessas considerações, é preciso destacar que o presente projeto demanda por uma memória não volátil, pois não se tem como garantir o fornecimento contínuo de energia. Portanto, decidiu-se adcionar a memória W25Q32 para garantir a eficácia da solução proposta.

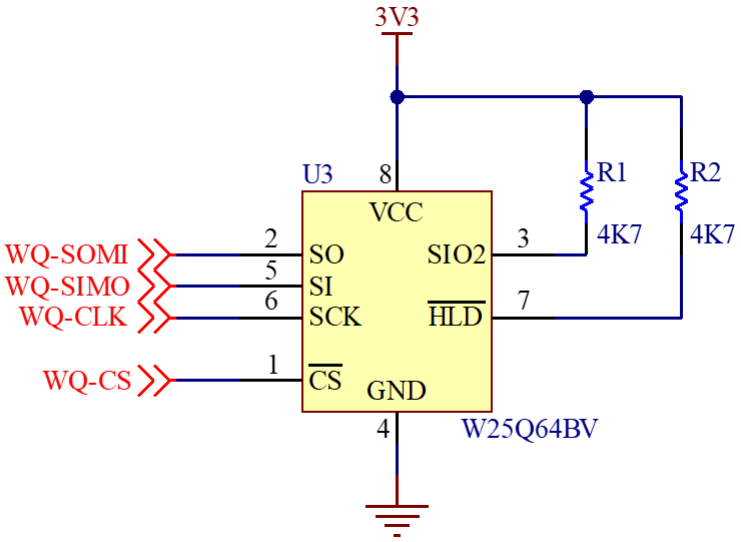
A memória W25Q32 opera com base no protocolo SPI e segue a dinâmica de comunicação mestre-escravo. É uma unidade flash que oferece 4 MB de espaço de armazenamento. Esta capacidade excede em muito o necessário para a presente proposta, garantindo espaço para futuras expansões ou modificações. Sua principal função é registrar informações cruciais, como dados relacionados à localização do módulo GPS e informações de tempo do RTC. Esta informação é essencial para o funcionamento do sistema e é posteriormente enviada ao usuário. Esta memória é eficiente quando se gravam dados dentro de setores de 256 bytes. Portanto, o tamanho dos registros foi especificado para 128 bytes. Dessa forma, dois registros ocupam um setor de 256 bytes. A *Figura* 3.7 mostra o modelo da memória externa utilizada no projeto enquanto a *Figura* 3.8 ilustra as conexões da memória externa utilizadas.

Figura 3.7: CHIP 25Q32FVSIG - Memória externa

Fonte: Mercado Livre

Disponivel em: Mercado Livre. Acesso em: 08 de novembro de 2023

Figura 3.8: Conexões do módulo da memória externa para o MSP430F5529LP



Fonte: Autoria Própria.

3.2.3 Real-Time Clock

O módulo DS3231 é um componente extremamente preciso e integrado de Relógio de Tempo Real (RTC), com oscilador compensado por variações de temperatura, além de um oscilador de cristal integrado (32kHz). Esse módulo foi projetado para proporcionar uma cronometragem precisa, com um erro mínimo devido à variação de temperatura, o que o torna ideal para aplicações que exigem uma contagem de tempo altamente confiável. O projeto faz uso de uma pequena placa que já traz o chip DS3231 soldado e com algumas facilidades para tensão e interrupção de bateria.

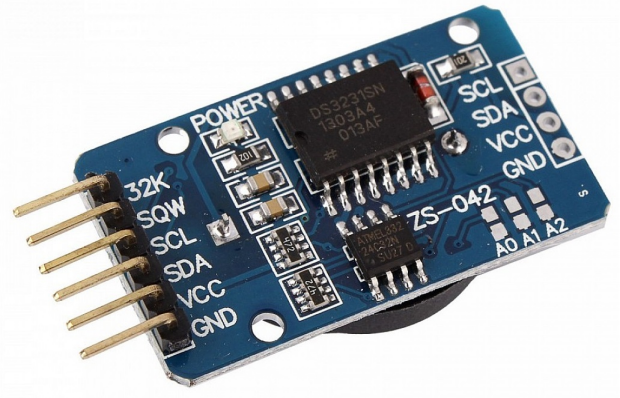
Em relação à sua interface de comunicação, o DS3231 utiliza o protocolo I2C, o que simplifica a integração com diversos microcontroladores e sistemas. Este método de comunicação bidirecional permite uma transferência de dados eficiente e confiável entre o módulo e outros componentes, ao mesmo tempo que mantém a pequena quantidade de pinos.

Em relação à configuração dos pinos e fonte de alimentação, o DS3231 é equipado com os seguintes pinos: VCC, GND, SDA, SCL, 32K e SQW. O pino VCC é responsável pela fonte de alimentação, enquanto GND é usado como referência de aterramento. Os pinos SDA e SCL são as linhas de dados e linhas de clock do protocolo I2C. Pelo pino denominado 32K, é disponibilizada uma onda quadrada de 32.678 Hz, e pelo pino SQW, é possível utilizar a função despertador para interromper a CPU. Vale ressaltar que o DS3231 pode operar em uma ampla faixa de tensões, o que o torna adequado para diversas aplicações em sistemas embarcados. O módulo tem a capacidade de ajustar automaticamente o final

dos meses que têm menos de 31 dias, também pode fazer a correção do ano bissexto e funcionar no formato de 12 ou 24 horas.

A *Figura* 3.9 mostra o modelo do RTC utilizado no projeto enquanto a *Figura* 3.10 ilustra as conexões do RTC utilizado. Deve ser ressaltado que esta placa já vem com uma bateria de backup tipo moeda. Portanto, seu uso é muito simples e resolve todos os problemas de data e hora do sistema.

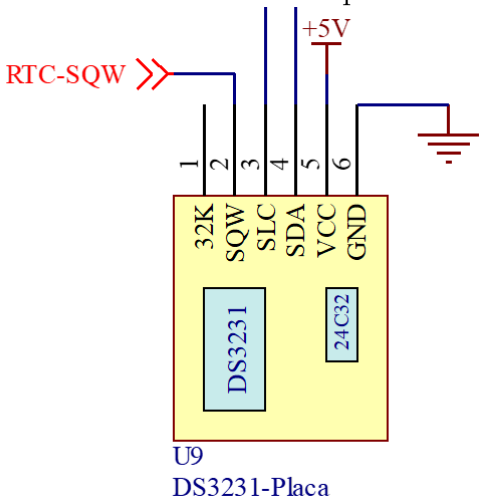
Figura 3.9: Módulo Real Time Clock RTC DS3231



Fonte: HuInfinito

Disponivel em: Hu Infinito. Acesso em: 08 de novembro de 2023

Figura 3.10: Conexões do módulo RTC para o MSP430F5529LP



Fonte: Autoria Própria.

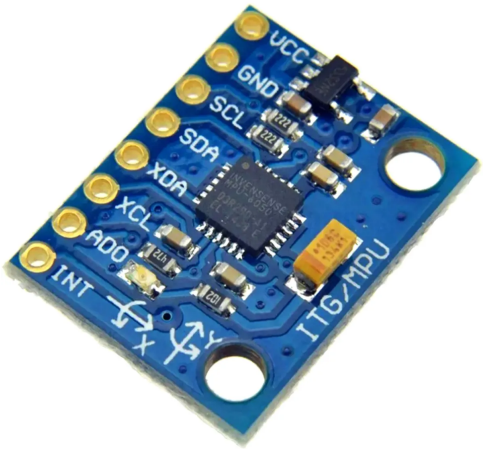
3.2.4 Giroscópio e Acelerômetro

O módulo MPU6050 possui uma arquitetura que inclui dois sensores integrados, um giroscópio e um acelerômetro, além de um recurso notável denominado Processador de

Movimento Digital (DMP). O processador realiza cálculos com base nos dados do sensor, o que permite a implementação em sistemas avançados de reconhecimento de gestos. Outro recurso que amplia a versatilidade do MPU6050 é o sensor de temperatura integrado, permitindo a medição de temperaturas de componentes internos na faixa de -40 a +85 graus Celsius.

Para se comunicar com outros dispositivos como o MSP430F5529LP, o MPU6050 utiliza uma interface I2C através dos pinos SCL e SDA. O módulo também possui pinos XDA e XCL, permitindo sua expansão da conexão I2C com outros componentes que também utilizam I2C, o que é interessante para sistemas que necessitam de orientação detalhada. O MPU6050 é flexível em relação à alimentação, admitindo a faixa de 3 a 5V. Porém, usar 5V é recomendado, pois melhoram as performances além de estar seguindo as recomendações do fabricante. A *Figura* 3.11 mostra o modelo do módulo utilizado no projeto, enquanto a *Figura* 3.12 ilustra suas conexões com o projeto.

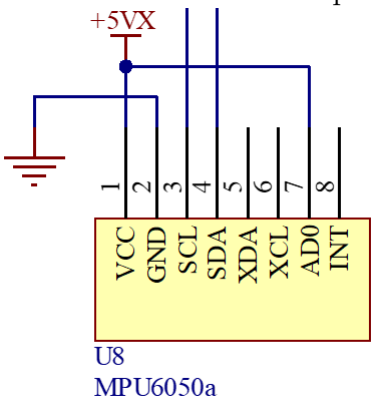
Figura 3.11: Módulo Acelerômetro/giroscópio MPU6050



Fonte: HuInfinito

Disponivel em: Hu Infinito. Acesso em: 08 de novembro de 2023

Figura 3.12: Conexões do módulo MPU6050 para o MSP430F5529LP



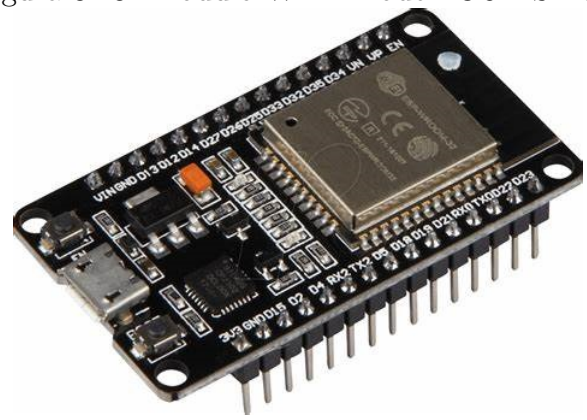
Fonte: Autoria Própria.

3.2.5 Wi-Fi

O ESP32 é um módulo altamente reconhecido na comunidade acadêmica e industrial. Embora o ESP32 seja frequentemente aclamado por suas capacidades multifuncionais e possa servir como módulo principal em muitas aplicações, no contexto do projeto em questão, sua utilidade foi primordialmente centrada em suas capacidades de conectividade Wi-Fi.

Desenvolvido pela Espressif Systems, o chip representa uma solução avançada para as necessidades da Internet das Coisas (IOT), combinando processamento de alto desempenho com a versatilidade das comunicações sem fio. A *Figura* 3.13 mostra o modelo do módulo utilizado no projeto.

Figura 3.13: Módulo Wi-Fi NodeMCU ESP-32



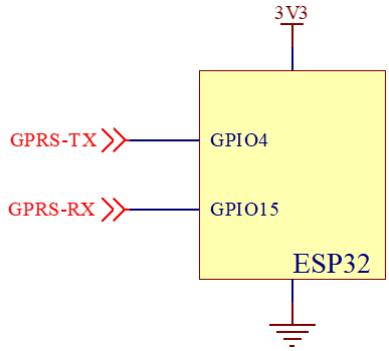
Fonte: HuInfinito

Disponivel em: Hu Infinito. Acesso em: 08 de novembro de 2023

A interface de comunicação do ESP32 suporta várias modalidades, incluindo I2C, SPI e GPIO. No entanto, a comunicação serial foi priorizada para a integração com o MSP430F5529LP, estabelecendo uma conexão confiável e eficaz entre os dois dispositivos. Esta opção de comunicação não diminui as outras capacidades do ESP32, mas enfatiza a necessidade de conectividade Wi-Fi neste projeto, demonstrando a rapidez e estabilidade que o módulo pode fornecer em transmissões de dados sem fio.

Quando se trata de conexão e alimentação, o ESP32 se destaca por seu baixo consumo de energia, geralmente operando a 3,3V. Esse módulo conta com um regulador interno, proporcionando flexibilidade ao garantir compatibilidade com variadas tensões. Para o projeto, os pinos essenciais de alimentação, VCC e GND, foram integrados à fonte de alimentação principal, garantindo uma unidade de fornecimento de energia. Ademais, os pinos TX e RX são cruciais para estabelecer a comunicação serial com o MSP430F5529LP, a *Figura* 3.14 apresenta as conexões com o projeto.

Figura 3.14: Conexões do módulo ESP32 para o MSP430F5529LP



Fonte: Autoria Própria.

3.3 Problemas com Wi-Fi

Um dos desafios enfrentado durante o desenvolvimento do projeto Arapuca foi relacionado à conectividade Wi-Fi, aspecto fundamental para o funcionamento eficiente do sistema. A dependência de uma conexão Wi-Fi estável pode resultar em significativas limitações, sobretudo quando a rede encontra-se inacessível ou instável. Em situações de sinal fraco ou intermitente o dispositivo pode enfrentar dificuldades ao enviar mensagens de alerta, comprometendo assim a eficiência do sistema de monitoramento e segurança.

Para que o dispositivo funcione corretamente, é necessário um ponto de acesso Wi-Fi estável. Portanto, o usuário deve garantir que haja uma rede Wi-Fi segura e confiável disponível. Em locais onde a cobertura Wi-Fi é limitada ou inexistente, isso pode ser um desafio.

A configuração do módulo ESP32 responsável pela conexão Wi-Fi do dispositivo requer acesso por meio de senha da rede Wi-Fi, o que agrega complexidade adicional à instalação e utilização do sistema. Esta dependência de uma rede Wi-Fi específica pode limitar a mobilidade do dispositivo e a sua adequação em diferentes contextos e ambientes.

Capítulo 4

Proposta de software

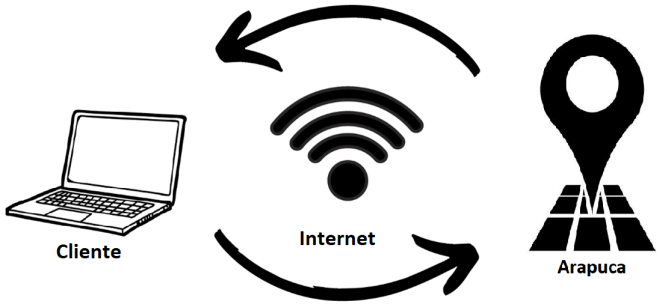
Neste capítulo serão abordados as principais ideias por trás dos códigos implementados, desde o emprego do WebService, interface do usuário etc.

Vale ressaltar que todas as configurações dos módulos já foram implementados no projeto anterior *Dispositivo de segurança para aparelhos eletrônicos* [2].

4.1 Diagrama geral do projeto Arapuca

Conforme mostrado na *Figura* 4.1, o sistema consiste em várias partes inter-relacionadas, cada uma desempenhando um papel vital na funcionalidade geral do sistema. Este diagrama desta figura fornece uma visão clara da estrutura e do fluxo operacional, essencial para a compreensão do sistema como um todo.

Figura 4.1: Diagrama conceitual de todo o sistema Arapuca



Fonte: Autoria própria.

No que diz respeito à interface de usuário, destacada pelo *cliente*, o sistema Arapuca utiliza uma abordagem centrada no usuário, garantindo facilidade de uso e interatividade. Permitindo enviar comandos e receber respostas em tempo real, o que facilita a gestão

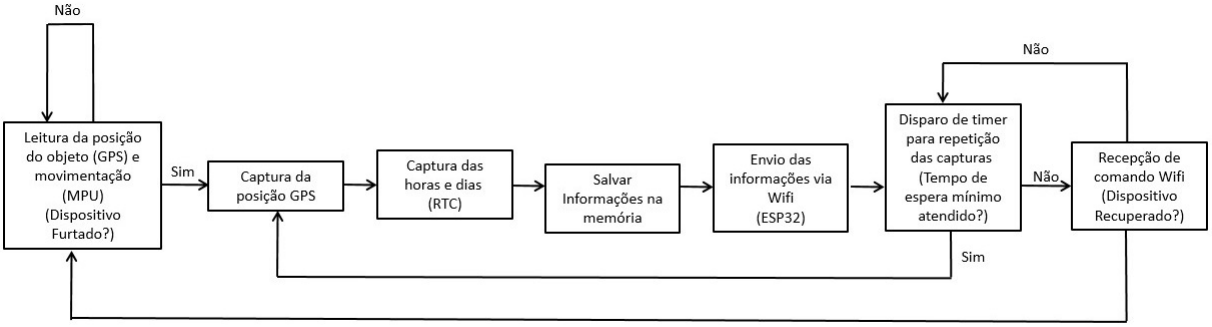
e o controle do sistema. Essa interface é um componente chave para garantir que o usuário possa interagir eficientemente com o sistema, independentemente de seu nível de habilidade técnica.

Em relação à conectividade e gerenciamento de dados, o sistema Arapuca incorpora um webservice através da *internet*. Este webservice desempenha um papel crucial na comunicação remota, permitindo a transmissão eficiente de dados entre o hardware e o usuário. Através deste serviço, o sistema pode enviar alertas e atualizações, além de permitir o acesso remoto aos dados coletados. Esta funcionalidade é fundamental para garantir que os usuários estejam sempre informados e possam responder prontamente a qualquer situação.

4.2 Funcionamento do hardware Arapuca

Na *Figura* 4.2, apresenta-se um diagrama de blocos que ilustra as diversas fases operacionais do dispositivo. Neste diagrama, é evidente o processo iniciado pelo dispositivo em caso de furto. Primeiramente (retângulo mais a esquerda), o dispositivo ativa a função de rastreamento GPS para determinar sua localização atual, além da leitura do MPU para determinar movimento. Em seguida, procede-se à captação da data e hora exatas do evento. Posteriormente, estas informações são transmitidas através do módulo Wi-Fi. Além disso, um timer integrado ao sistema é programado para disparar periodicamente, capturando e armazenando novos conjuntos de dados na memória flash. Este processo de captura e armazenamento de dados continua ativo até que o dispositivo seja efetivamente recuperado.

Figura 4.2: Diagrama de blocos do sistema de hardware



Fonte: Autoria própria.

4.2.1 Caracterização do furto

A identificação de um possível furto de um eletrodoméstico sob monitoramento do dispositivo, pode ser conseguida pela detecção de movimento via MPU ou através do GPS. Por exemplo, no caso de um aparelho de som portátil, a utilização do GPS para definir uma área segura de operação é mais eficaz, pois permite identificar deslocamentos fora dessa zona predefinida. Já para uma televisão, o MPU tende a ser mais adequado para sinalizar um furto, considerando que movimentos frequentes do aparelho são incomuns.

É importante destacar que o dispositivo pode permanecer ativo continuamente, mesmo em locais considerados seguros, enviando dados regularmente. Esta operação contínua é viável devido ao baixo consumo de energia do sistema, o que é aceitável na maioria das aplicações práticas.

4.2.2 Captura dos dados

Uma vez caracterizado o furto, o dispositivo inicia a coleta de dados de forma programada. Neste contexto, o papel do RTC (Real-Time Clock) é fundamental, pois é ele quem ativa o dispositivo periodicamente para repetir a coleta de dados. O RTC não só desperta o dispositivo, mas também fornece informações precisas de data e hora, essenciais para o registro dos eventos.

Após a conclusão da coleta, os dados são armazenados na memória flash. Esses dados são possíveis serem visualizados através da interface.

Para ilustrar a capacidade de armazenamento, considere que cada registro ocupa 128 bytes. Com uma memória de 4 MB, é possível armazenar até 32.768 registros. Isso representa uma quantidade significativa de dados:

- Com um registro por minuto, o dispositivo pode armazenar dados por aproximadamente 22 dias (32.768 registros divididos por 60 minutos).
- Com um registro por hora, a capacidade se estende para cerca de 3 anos (32.768 registros divididos por 24 horas).

Organização dos dados na memória flash

A padronização do armazenamento dos dados na memória flash, segue conforme o apresentado na *Tabela* 4.1, ocupando um total de 97 bytes.

Tabela 4.1: Organização dos dados armazenados em ASCII na memória flash

Modo	**Data**	**Hora**	**Latitude**	**-**
xxxx -	dd/mm/aa	hh:mm:ss	ddmm.mmmmm,	N/S
ALT2 -	15/11/23,	10:34:21,	4717.11437,	N
7,	9	9,	12	2

Longitude	**-**	**Acel**	**Giro**	**\n\r**
dddmm.mmmmm,	E/W	Ax=xxx Ay=xxx Az=xxx ,	Gx=xxx Gy=xxx Gz=xxx	CRLF
00833.91522,	E	A= 65 -2 3,	G= -1 -2 -1	0x130A
12	2	21	21	2

Fonte: Autoria própria.

Os dados do MPU são valores absolutos em 8 bits para cada um dos eixos x, y e z. Vale mencionar, que a quantidade de caracteres nem sempre segue conforme o número na linha 4 (última linha), pode acontecer de ter uma pequena variação em função da quantidade de dígitos necessária para cada parâmetro. O mais importante é que a quantidade total seja sempre inferior a 128 bytes, devido que na memória flash W25Q32, para evitar problemas com as fronteiras das páginas, o registro precisa ter tamanho que seja potência de 2. Dessa maneira, foi adequado utilizar um tamanho de 128 bytes (2^7).

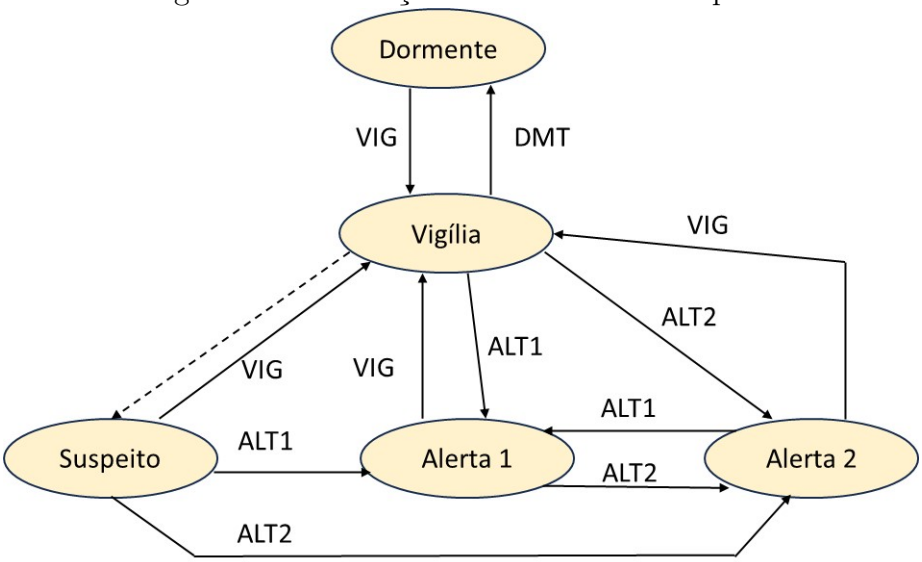
4.2.3 Confirmação da recuperação

Na fase final do processo, quando o dispositivo furtado é recuperado, o proprietário tem a possibilidade de reiniciar o ciclo operacional do sistema. Por exemplo, se um eletro-doméstico equipado com o dispositivo Arapuca é furtado e posteriormente recuperado, o proprietário pode enviar um comando ordenando que o sistema retorne à sua fase inicial e reinicie o ciclo de monitoramento.

4.3 Ilustração dos estados do Arapuca

A ilustração dos estados apresentado na *Figura* 4.3 ilustra as transições entre os diferentes estados do dispositivo. Cada estado reflete uma condição específica do sistema, desde a vigilância padrão até os níveis de alerta.

Figura 4.3: Ilustração dos estados do Arapuca



Fonte: Autoria própria.

O desafio principal na concepção do Arapuca foi desenvolver um mecanismo que permitisse a mudança automática de estado entre **Suspeito**, **Alerta 1** e **Alerta 2**. A complexidade residia em criar um sistema que não apenas detectasse condições anormais, mas também determinasse o nível apropriado de resposta com base na gravidade da situação. Essa transição automática é essencial para garantir que o dispositivo responda de maneira eficiente e eficaz, minimizando falsos alarmes e maximizando a segurança.

Diante do desafio apresentado, optou-se por desenvolver a mudança automática apenas entre os estados de **Vigilia** e **Suspeito**. Essa decisão foi tomada considerando que, na implementação atual deste trabalho, a transição entre os estados de **Suspeito**, **Alerta 1** e **Alerta 2** foi concebida para ser realizada manualmente pelo usuário. Cabendo ao usuário a responsabilidade de avaliar e decidir sobre a gravidade e a natureza de cada alerta. Essa funcionalidade manual, integrada ao sistema, assegura que o dispositivo Arapuca mantenha sua eficácia e flexibilidade, adaptando-se às necessidades específicas de segurança do usuário.

4.3.1 Dormente

Este modo foi projetado para operar de forma eficiente, com o objetivo de manter o dispositivo em estado de baixo consumo de energia e focar apenas no recebimento de mensagens. Esse recurso é essencial para prolongar a vida útil da bateria do dispositivo, garantindo que ele funcione por longos períodos sem a necessidade de carregamentos frequentes.

Ademais, o modo Dormente é crucial para manter o dispositivo alerta e responsivo a comandos externos, mesmo quando não está em uso ativo, assegurando que o sistema esteja sempre pronto para entrar em ação quando necessário.

4.3.2 Vigilia

Neste estado, o dispositivo não só recebe mensagens, mas também está preparado para transitar para os estados de alerta, dependendo das condições detectadas pelos sensores.

Este estado é especialmente sensível às variações nos sensores do acelerômetro, giroscópio ou GPS. Ao entrar neste modo, o dispositivo inicializa e configura os valores do MPU e do GPS, estabelecendo um ponto de referência. A partir daí, realiza medições constantes para verificar se há discrepâncias entre os valores atuais e os valores inicialmente configurados. Qualquer alteração significativa nos dados desses sensores pode indicar uma situação potencialmente suspeita, levando o Arapuca a entrar automaticamente no próximo estado **Suspeito**.

Este modo é uma medida para identificar e responder a possíveis ameaças ou irregularidades, como movimentos inesperados do objeto monitorado, garantindo uma resposta rápida e eficaz em situações de potencial furto ou deslocamento não autorizado do objeto.

4.3.3 Suspeito

Neste modo, o dispositivo aumenta sua vigilância e começa a enviar dados a cada três horas (este período pode ser programado pelo usuário), também armazenam essas informações na memória. Essa frequência de atualização garante que os dados sejam coletados e registrados regularmente, proporcionando um histórico detalhado e atualizado das condições e localização do objeto monitorado.

O aspecto mais crítico do Modo Suspeito é que a transição para outro estado operacional deve ser feita manualmente. Isso significa que, uma vez ativado, o dispositivo permanecerá neste modo até que uma intervenção direta seja realizada. Essa característica é essencial para manter um nível elevado de segurança, pois garante que o dispositivo continue monitorando e registrando dados em situações potencialmente arriscadas, sem ser desativado ou alterado automaticamente. Essa abordagem manual para mudar de estado assegura que o usuário tenha controle total sobre o funcionamento do dispositivo, permitindo uma resposta personalizada e consciente a situações suspeitas.

Uma adversidade não vencida, foi conseguir um meio em que houvesse mudaça desse estado para o de Alerta 1 de forma automártica.

4.3.4 Alerta 1

Quando ativado, este modo indica que o dispositivo foi furtado ou está em uma situação de alto risco. Neste estado, o Arapuca intensifica suas operações de monitoramento, enviando atualizações de posição a cada hora. Esta frequência aumentada de comunicação é vital para rastrear o movimento e a localização do objeto roubado de forma eficiente e contínua.

Além do envio de mensagens de posição, o dispositivo também grava essas informações na memória flash a cada hora.

4.3.5 Alerta 2

Representa um nível ainda mais elevado de vigilância e resposta em situações de furto. Este modo é ativado pelo usuário, quando há uma confirmação ou forte suspeita de que o dispositivo foi furtado. Neste estado, o Arapuca adota uma estratégia de monitoramento intensivo, enviando atualizações de posição a cada minuto. Esta alta frequência de comunicação é essencial para fornecer um acompanhamento em tempo quase real do movimento e localização do objeto furtado.

Paralelamente ao envio de mensagens de posição, o dispositivo também grava essas informações na memória flash a cada minuto. Este registro contínuo e detalhado é crucial para criar um histórico abrangente da trajetória do objeto, oferecendo informações valiosas para a recuperação do item e para eventuais investigações. O Modo de Alerta 2 é uma ferramenta vital no sistema Arapuca, proporcionando uma resposta imediata e detalhada em situações críticas de furto, maximizando as chances de localização e recuperação rápida dos bens monitorados.

4.4 Considerações importantes

Neste tópico, são abordados alguns ítens cruciais que desempenham papel fundamental no aprimoramento e na eficácia do sistema Arapuca. Esses ítens incluem o web service, o bot do WhatsApp e outras funcionalidades adicionais implementadas através dos botões da interface. Cada um desses elementos contribui de maneira única para a operação do sistema, oferecendo maior controle, segurança e interatividade ao usuário.

4.4.1 Web Service

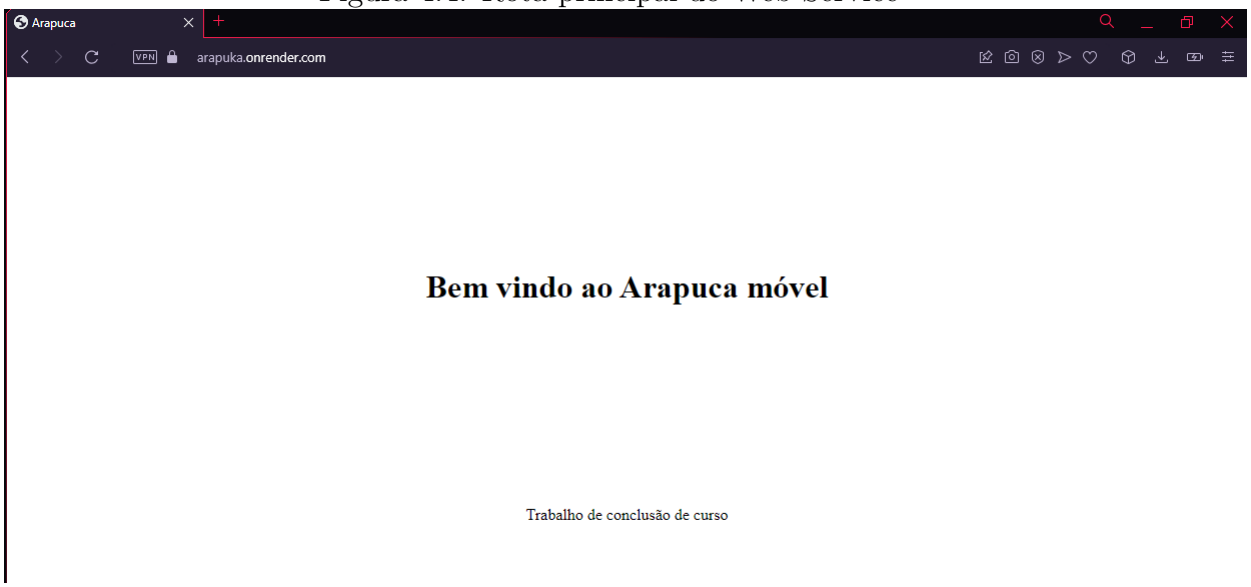
A implementação do web service, desenvolvido com Flask e Python, no sistema Arapuca é um aspecto crucial que traz uma série de benefícios significativos para a aplicação. O Flask, um microframework para web em Python, que oferece flexibilidade e eficiência necessária para criar um webservice robusto e confiável [23]. A escolha do Python, por sua

vez, devido a sua simplicidade de aprendizado além de fácil manutenção e de compatibilidade com futuras expansões no sistema [24]. O webservice atua como um intermediário essencial entre o hardware do dispositivo Arapuca e o usuário, permitindo a comunicação e o controle remotos de forma eficiente e segura [25].

Dentro deste web service, foram desenvolvidas três rotas principais que são fundamentais para a operação do sistema:

- **https://arapuka.onrender.com/**: apenas uma tela de apresentação do web service, como pode ser visto na *Figura* 4.4.

Figura 4.4: Rota principal do Web Service



Fonte: Autoria própria.

- **https://arapuka.onrender.com/comando**: é dedicada ao envio de comandos para o dispositivo. Esta funcionalidade permite que o usuário interaja diretamente com o Arapuca, enviando instruções específicas, como a ativação de funções ou a solicitação de dados.
- **https://arapuka.onrender.com/resposta**: é utilizada para receber as respostas do dispositivo. Essa rota é essencial para a coleta de informações do Arapuca, como a localização atual, o status do dispositivo e outros dados relevantes.

A implementação dessas rotas no webservice facilita a interação do usuário com o dispositivo, aproveitando as vantagens da conectividade web para melhorar a experiência do usuário e a funcionalidade do sistema Arapuca.

4.4.2 Alerta Whatsapp

Em muitas situações, o proprietário do dispositivo Arapuca pode não ter a disponibilidade ou a conveniência de monitorar constantemente a interface do sistema para verificar se ocorreu um furto ou alguma mudança de estado do objeto protegido. Essa realidade impõe a necessidade de um método mais prático e acessível de notificação. Nesse contexto, a integração de um *Bot de WhatsApp* [26] surge como uma solução inovadora e eficiente. Através deste bot, o proprietário pode receber alertas automáticos diretamente em seu smartphone, sempre que o dispositivo Arapuca detectar uma mudança de estado, como um deslocamento não autorizado ou uma tentativa de furto.

O uso do *Bot de WhatsApp* [26] para enviar notificações de alerta oferece uma série de vantagens. Para começar, oferece ao usuário a facilidade de obter atualizações em tempo real sem a necessidade de permanecer na interface do sistema em qualquer momento. Além disso, o WhatsApp, como uma plataforma comum para a maioria dos usuários [27], torna mais fácil aderir e entender o sistema de notificações. Essa abordagem assegura que o proprietário esteja sempre informado sobre o status de segurança de seus bens, aumentando a eficácia do sistema Arapuca na prevenção e no alerta de situações de furto ou movimentações suspeitas.

4.4.3 Interface Python

A *Figura* 4.5 representa a tela principal da interface desenvolvida.

Figura 4.5: Interface de usuário Arapuca

Fonte: Autoria própria.

Algumas das funcionalidades dos botões na interface já foram explicadas nos tópicos anteriores, a seguir são descritos o funcionamento e configuração dos demais comandos.

Console

Esta funcionalidade foi projetada para atender a uma necessidade específica: a verificação e atualização do estado do sistema após uma mudança já ter ocorrido.

Em cenários onde o objeto rastreado sofre uma alteração de estado - seja por movimento indevido, furto ou outra condição predefinida - o sistema automaticamente registra e armazena dados relevantes. O botão de console surge como um recurso intuitivo, permitindo ao usuário acessar uma tela de atualização constante, exibindo as mensagens recebidas, garantindo assim uma interação mais eficaz e uma resposta rápida às mudanças de estado detectadas.

Status

Possui o objetivo específico de informar o usuário sobre o estado atual do dispositivo Arapuca. Ao acioná-lo, o usuário obtém informações imediatas sobre o status do dispositivo,

desde informações do estado atual de operação, data e hora, localização GPS, além dos dados do acelerômetro e giroscópio.

Ler memoria

Oferece ao usuário a capacidade de acessar e analisar dados específicos armazenados no dispositivo. Esta opção é útil para a revisão e o monitoramento detalhado das atividades registradas pelo sistema. Dentro desta funcionalidade, existem comandos específicos que permitem uma leitura personalizada da memória, adaptando-se às necessidades do usuário.

- **RD_n**: é utilizado para ler os primeiros 'n' registros da memória.
- **RD_n_m**: oferece uma funcionalidade mais detalhada, permitindo ao usuário ler os registros da memória a partir do registro 'n' até o registro 'm'.

Para uma análise direcionada e eficiente, esta capacidade de leitura de dados é essencial. Permite que o usuário obtenha informações detalhadas sobre períodos de tempo ou eventos específicos. Assim, a função "Ler Memória" do Arapuca se destaca como uma ferramenta útil para gerenciamento e análise de dados, dando ao usuário acesso completo e controlado às informações que o dispositivo registrou.

Apagar memória

A capacidade de apagar toda a memória flash W25Q32 no sistema Arapuca representa um avanço no gerenciamento de dados e na segurança do dispositivo. Este recurso permite ao usuário limpar completamente a memória do dispositivo, garantindo que todas as informações armazenadas sejam excluídas de forma segura e eficaz. Este recurso é particularmente útil em situações onde o dispositivo precisa ser reutilizado para diferentes objetos ou onde a privacidade e a segurança dos dados armazenados são consideradas importantes. Ao limpar a memória, o usuário pode garantir que nenhuma informação residual ou sensível seja deixada no dispositivo, evitando acesso não autorizado ou recuperação inadequada de dados.

Além disso, a função de apagar a memória flash auxilia na manutenção e eficiência do sistema. Com o tempo, o acúmulo de dados afeta o desempenho do dispositivo, tornando as operações mais lentas e menos eficientes. Ao disponibilizar a opção de limpeza de memória, o sistema Arapuca garante que o aparelho possa operar em sua capacidade máxima, mantendo assim um alto nível de desempenho. Esse recurso também facilita a manutenção e gerenciamento dos equipamentos, permitindo ao usuário reiniciar o sistema de monitoramento com a memória limpa, pronta para novas operações e registros. Em

resumo, a capacidade de apagar a memória flash W25Q32 não apenas aumenta a segurança dos dados, mas também otimiza a funcionalidade geral do sistema Arapuca.

E-mail

No desenvolvimento do projeto Arapuca, uma das principais funcionalidades é a possibilidade de enviar e-mails contendo detalhes de localização do dispositivo. Este recurso é implementado em conformidade com o padrão SMTP para garantir a eficiência e segurança na transmissão de e-mail. Este protocolo permite que o usuário receba e-mail contendo informações importantes, incluindo um link direto para o Google Maps. Este link fornece uma visão precisa da localização atual do dispositivo, além da data e hora exatas em que esses dados foram registrados. Para realizar esta tarefa, o sistema utiliza o servidor *SMTP2GO* [28], uma escolha estratégica que garante confiabilidade e agilidade no envio de e-mails via SMTP.

É importante ressaltar que a funcionalidade de envio de e-mail no sistema Arapuca não é automática, mas é acionada pelo usuário. Esta decisão foi tomada com o intuito de dar ao usuário o controle total sobre quando queira receber essas informações, e evitar notificações desnecessárias ou excessivas. O usuário pode solicitar o envio de um e-mail a qualquer momento clicando no botão designado para esta função na interface do sistema. Ao solicitar um e-mail, além de inserir o assunto do corpo do e-mail, o usuário também pode inserir o endereço de destino do e-mail desejado.

Resete

Esta opção permite que o usuário envie um comando específico que resulta na reinicialização do microcontrolador MSP430 por meio de software. Este recurso é útil em situações onde o dispositivo encontra-se com algum erro operacional ou comportamento anômalo, necessitando de um reinício rápido para retomar seu funcionamento normal.

Após o processo de reinicialização, o Arapuca executa uma varredura automática na memória para determinar qual era o último estado operacional antes do reinício. Esta checagem é essencial para garantir que o dispositivo retome suas funções no mesmo modo em que se encontrava, mantendo a continuidade e a eficácia do monitoramento. O comando de **Resete** é, portanto, uma ferramenta valiosa para a manutenção da estabilidade e confiabilidade do sistema, assegurando que o Arapuca permaneça operacional e eficiente mesmo diante de eventuais falhas ou erros de sistema.

Set RTC

O sistema faz uso de uma placa RTC denominada DS3231 que já vem com uma bateria de backup, garantindo a precisão na marcação de data e hora. Este componente é essencial para o registro contínuo do tempo, recurso importante para um monitoramento eficaz. O funcionamento do RTC é relativamente simples e intuitivo: o usuário configura os valores iniciais de data e hora no registrador RTC, e a partir desse momento o dispositivo passa a cronometrar de forma autônoma. O módulo RTC utilizado neste projeto é particularmente eficiente porque possui uma fonte de alimentação interna dedicada, garantindo a continuidade da temporização mesmo que o sistema de segurança e vigilância não possua fonte de alimentação externa.

No entanto, é importante considerar que podem ocorrer erros na atualização da data e hora no dispositivo. Para amenizar esse problema, foi desenvolvida uma funcionalidade adicional: um botão que, quando acionado, captura a data e hora atuais do computador do usuário e as envia ao dispositivo via webservice. Essa ação permite que o RTC do dispositivo Arapuca atualize seus dados de data e hora, garantindo assim a precisão e a confiabilidade das informações registradas. Esta solução não apenas resolve o problema de potenciais desvios na contagem do tempo, mas também adiciona uma camada extra de flexibilidade e controle para o usuário, permitindo uma sincronização fácil e eficiente com a hora local.

Mostrar Mapa

A funcionalidade do botão **Mostrar Mapa** foi projetada para melhorar a experiência do usuário ao interagir com o dispositivo. Antes dessa implementação, o usuário recebia as coordenadas GPS do hardware e precisava inseri-las manualmente no *GoogleMaps* para localizar o dispositivo. Com a introdução deste botão, ao receber as coordenadas GPS, o usuário pode simplesmente clicar sobre ele para que uma nova janela se abra, exibindo uma imagem do local do dispositivo diretamente no Google Maps. Esta abordagem simplifica o processo de localização, oferecendo uma visualização rápida e clara do local onde o dispositivo se encontra.

Essa funcionalidade está disponível em algumas telas da interface, incluindo as relacionadas aos modos Dormente, Vigília, Console, Alerta 1, Alerta 2 e Status. É importante notar que para a exibição efetiva do mapa, é necessário que a interface já tenha recebido dados de coordenadas desde sua inicialização. Caso contrário, a imagem do mapa não poderá ser exibida.

Capítulo 5

Ensaios

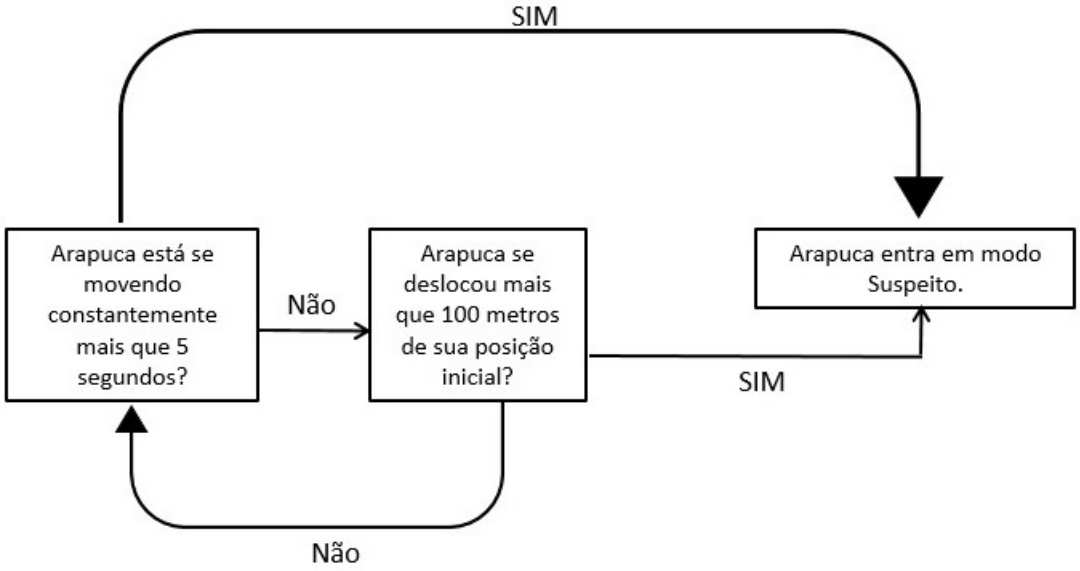
Neste capítulo, são explorados alguns testes realizados para avaliar as principais funcionalidades do Arapuca. Estas análises são fundamentais para compreender a eficácia, a eficiência do dispositivo e suas condições operacionais.

5.1 Testes para detecção de movimento do dispositivo

Para o desenvolvimento do sistema Arapuca utilizou-se dois métodos diferentes para a detecção de possíveis alertas de furto: MPU e GPS. O MPU pode detectar movimento e vibração usando um giroscópio integrado, fornecendo dados importantes para determinar se um dispositivo está se movendo de maneira suspeita. Além disso, o GPS ajuda a localizar o dispositivo com precisão, permitindo verificar se ele foi movido fora de uma área pré-determinada, o que pode indicar um furto.

A *Figura* 5.1 apresenta um pequeno diagrama a respeito do funcionamento da caracterização de furto.

Figura 5.1: Diagrama de caracterização de furto



Fonte: Autoria própria.

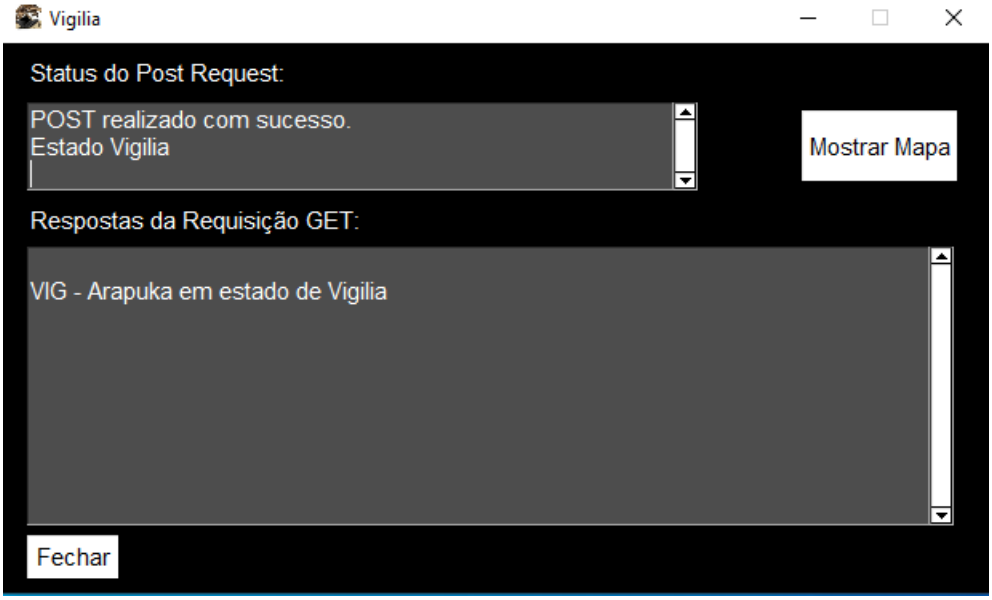
Baseando-se no projeto anterior [2], foi aprimorado o método de detecção de furtos utilizando o MPU6050. Através de uma calibragem cuidadosa, foi estabelecido um limite de ativação para o acelerômetro/giroscópio, considerando o ruído e a variação máxima detectada em estado de repouso. Definido um intervalo de tempo de 5 segundos para diferenciar movimentos normais de atividades suspeitas. Se o giroscópio detecta um movimento contínuo além desse período, o sistema inicia o processo de alerta de furto. Essa abordagem foi menos sensível em comparação com o projeto anterior, visando reduzir falsos positivos e aumentar a precisão na detecção de movimentos suspeitos.

Para a detecção de furto via GPS, foi adotado uma estratégia mais cautelosa devido à alta sensibilidade e ruído do sistema. Estabelecido um parâmetro de deslocamento de 100 metros como critério para o alerta de furto. Essa medida visa evitar falsos alarmes que poderiam ser causados por pequenas variações na localização GPS, garantindo que o alerta de furto seja acionado apenas quando há um deslocamento significativo do dispositivo.

5.2 Teste de mudança de estado

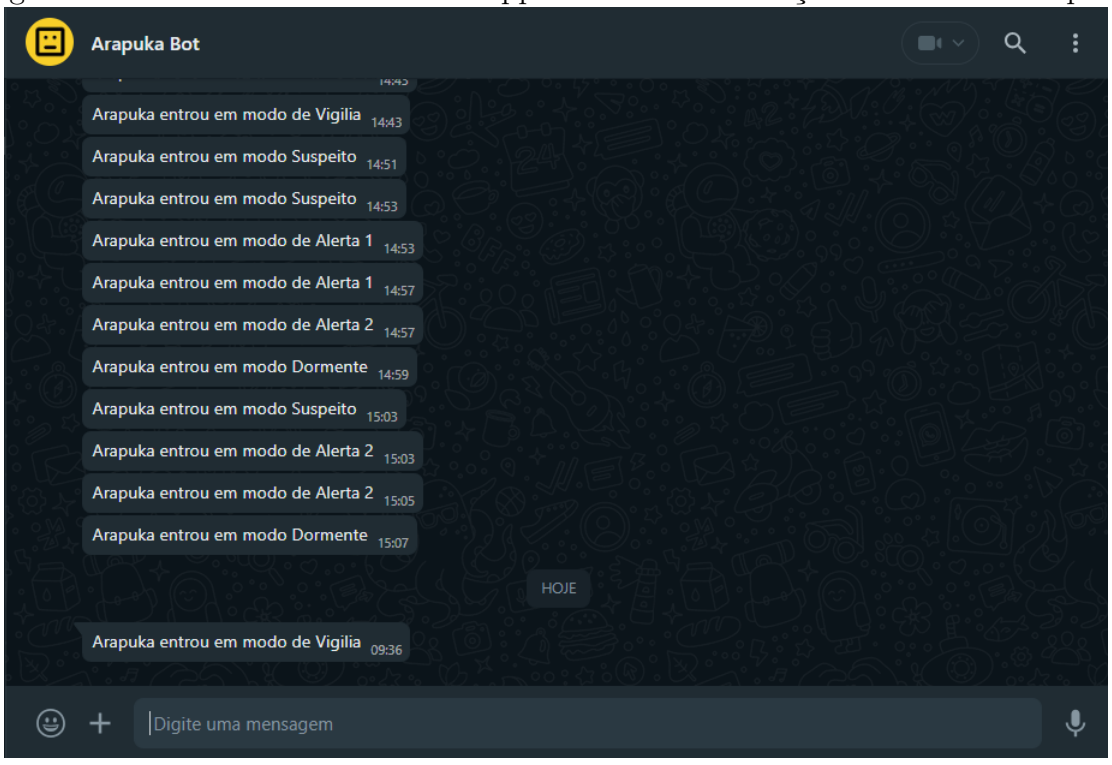
A *Figura* 5.2 demonstra a resposta do dispositivo ao selecionar a mudança para outro estado. Já a *Figura* 5.3 exibe o alerta recebido no WhatsApp, referente à mudança de estado realizada pelo Arapuca.

Figura 5.2: Pedido de mudança para outro estado do Arapuca



Fonte: Autoria própria.

Figura 5.3: Alerta recebido no whatsapp referente a mudança de estado do Arapuca

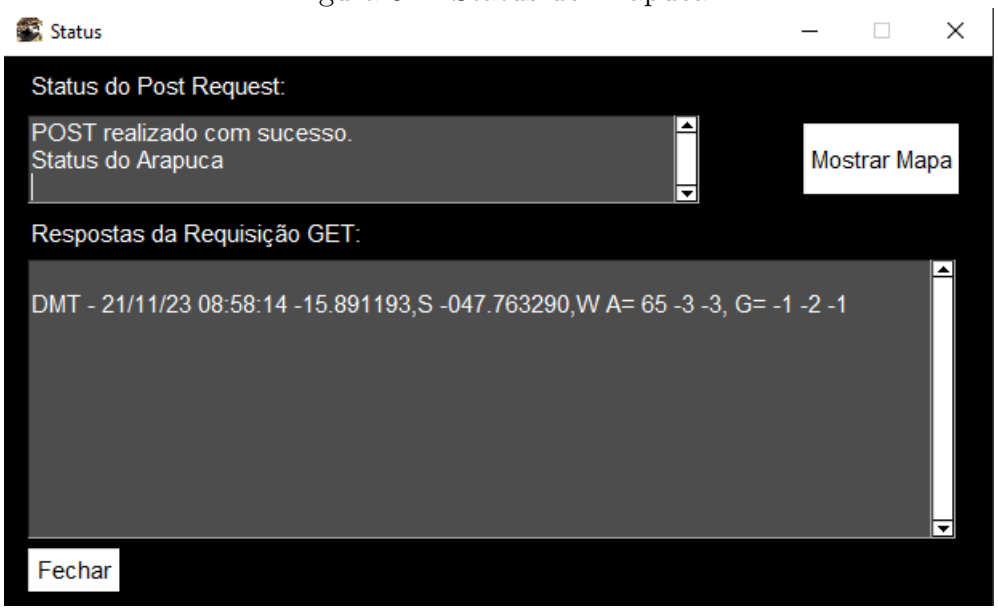


Fonte: Autoria própria.

5.3 Teste de Status

Conforme abordado no *tópico* 4.4.3, a função de status apresenta as informações atuais do dispositivo Arapuca, como ilustrado na *Figura* 5.4.

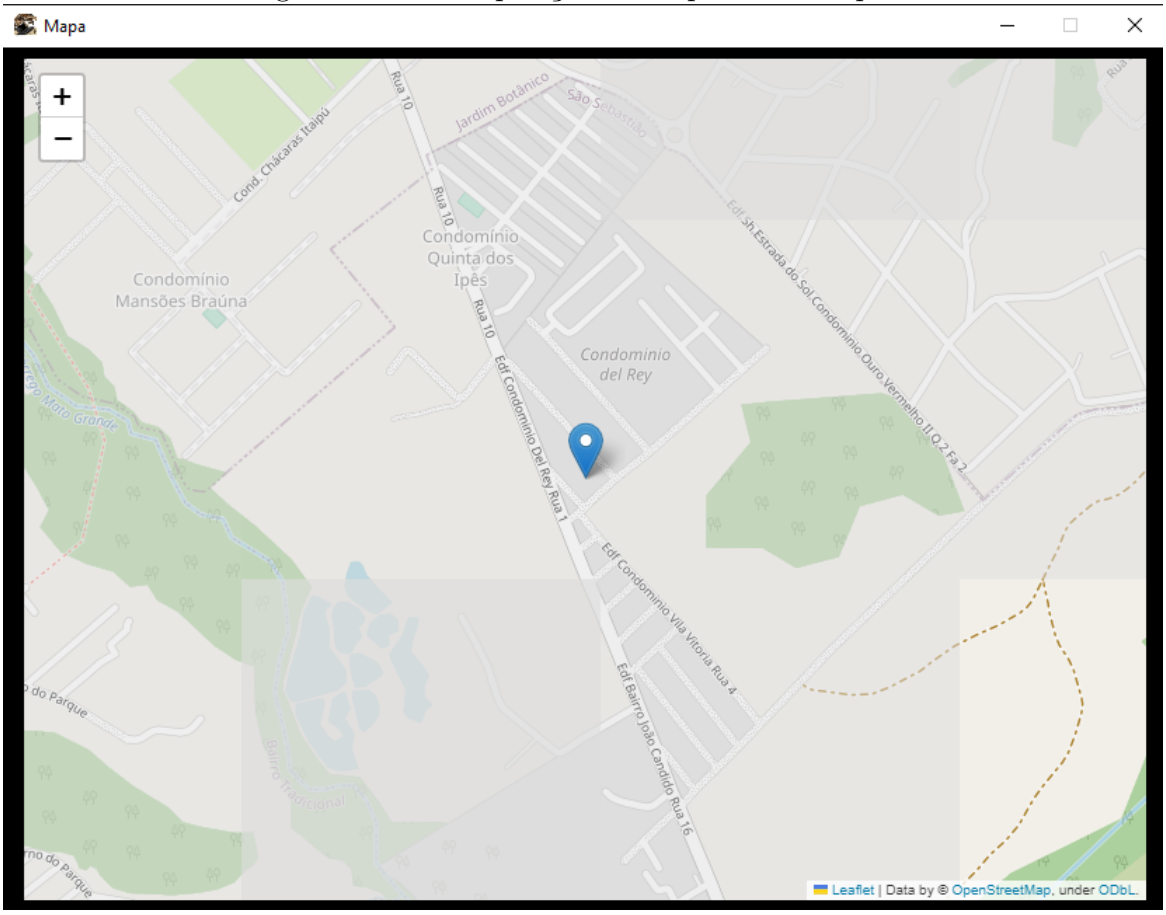
Figura 5.4: Status do Arapuca



Fonte: Autoria própria.

Conforme citado no *tópico* 4.4.3, para fazer o uso dessa funcionalidade, basta apertar o botão **Mostrar Mapa** no canto superior direito, a *Figura* 5.5 apresenta o exemplo do mapa, representando o local que o dispositivo está presente, no exemplo em questão o dispositivo está na casa do *João Pedro*, autor deste trabalho.

Figura 5.5: Atual posição do dispositivo Arapuca

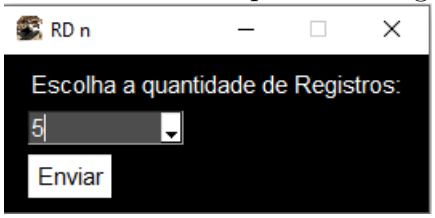


Fonte: Autoria própria.

5.4 Teste de memória

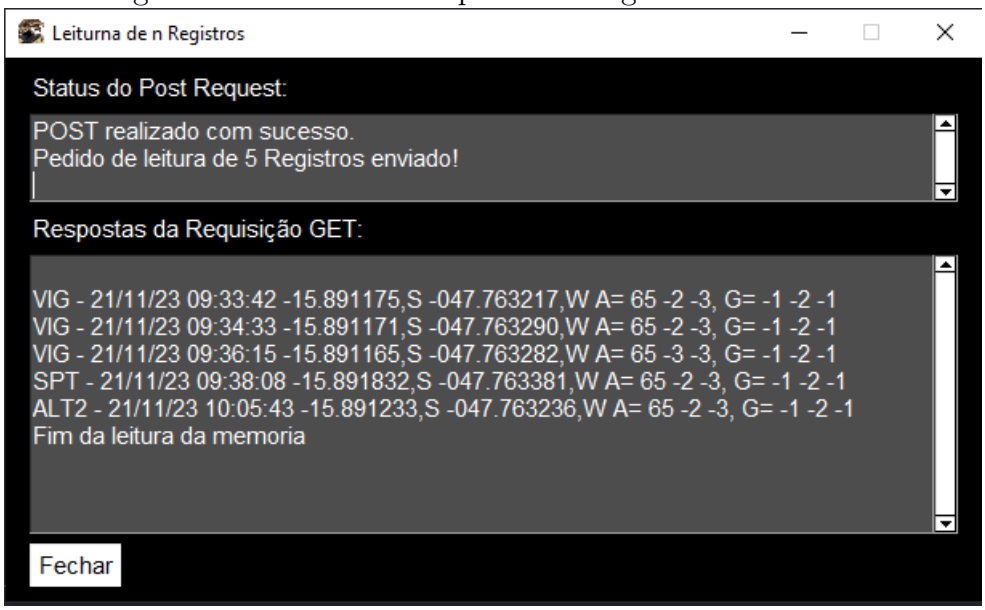
Nas *Figuras* 5.6 e 5.7, observa-se o teste para leitura dos primeiros n registros de memória, enquanto as *Figuras* 5.8 e 5.9 ilustram a leitura dos registros n até o m da memória flash.

Figura 5.6: Pedido de leitura dos 5 primeiros registros da memória



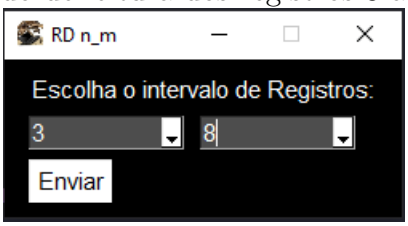
Fonte: Autoria própria.

Figura 5.7: Leitura dos 5 primeiros registros da memória



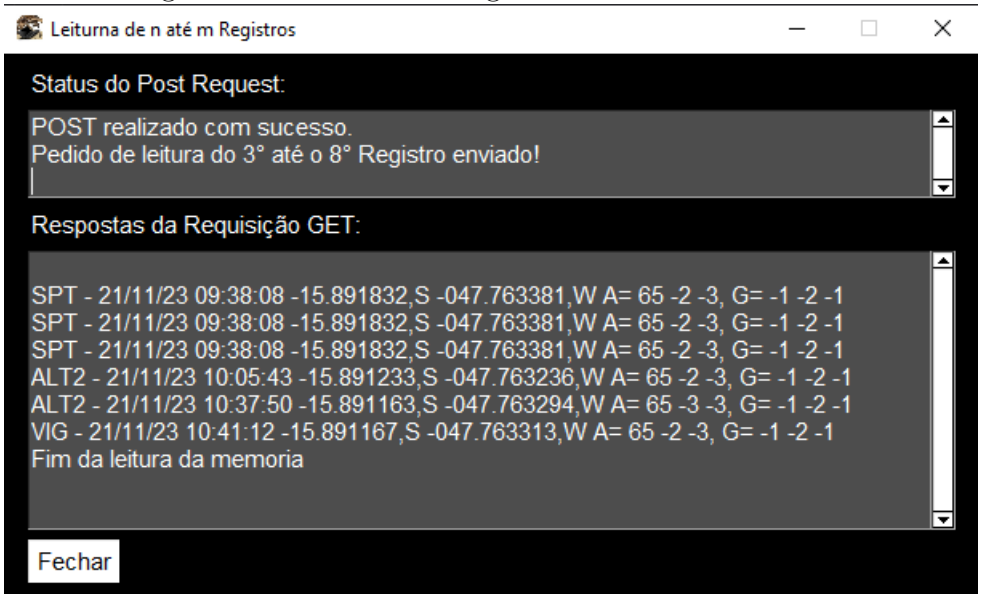
Fonte: Autoria própria.

Figura 5.8: Pedido de leitura dos registros 3 até 8 da memória



Fonte: Autoria própria.

Figura 5.9: Leitura dos registros 3 até 8 da memória

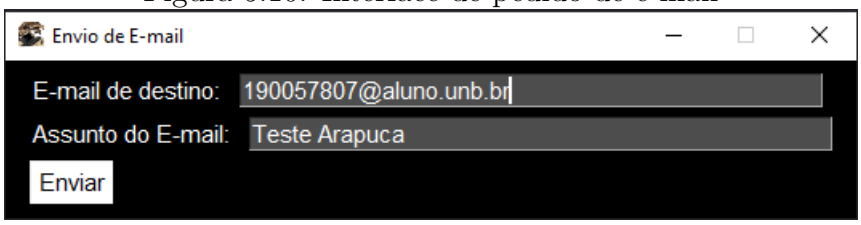


Fonte: Autoria própria.

5.5 Teste de envio de E-mail

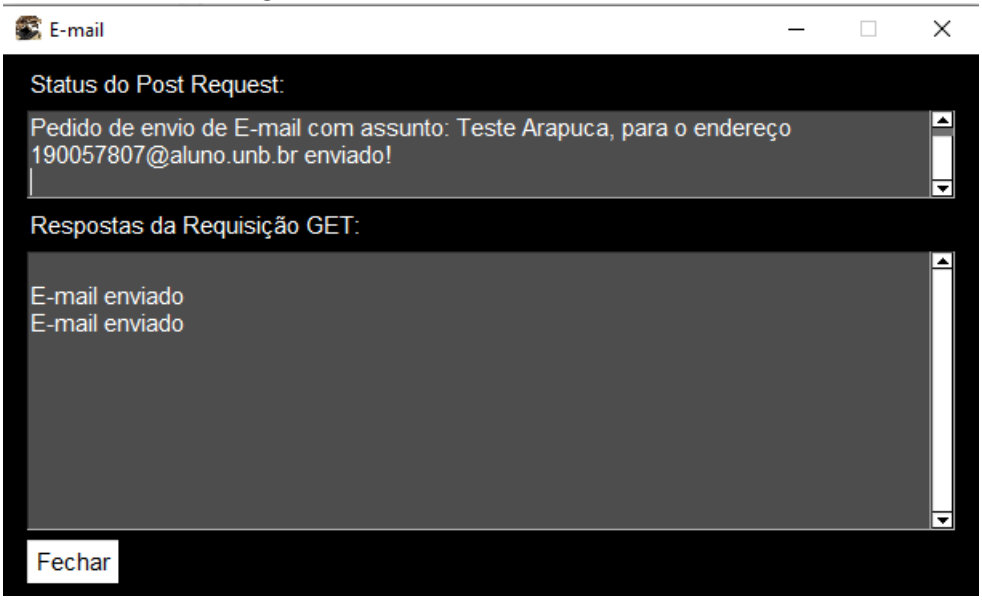
Neste exemplo, conforme as *Figuras* 5.10 e 5.11, enviou-se um e-mail com o Assunto **Teste Arapuca** para o endereço **190057807@aluno.unb.br**. *Figura* 5.12 mostra o e-mail recebido, exibindo a mensagem completa, que inclui um link do *GoogleMaps*.

Figura 5.10: Interface de pedido de e-mail



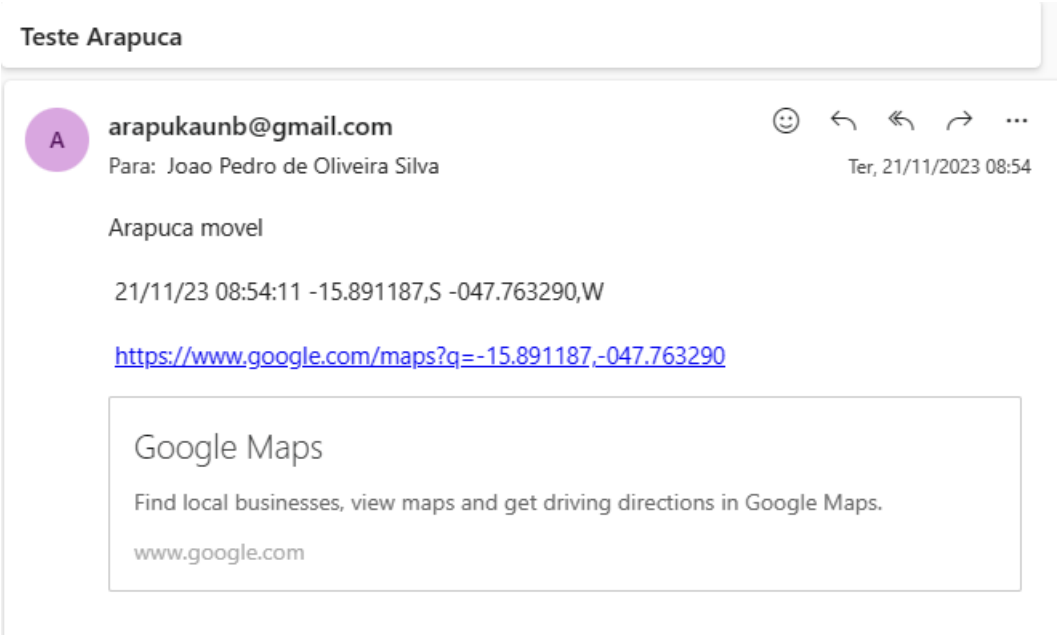
Fonte: Autoria própria.

Figura 5.11: Pedido de e-mail enviado



Fonte: Autoria própria.

Figura 5.12: E-mail recebido



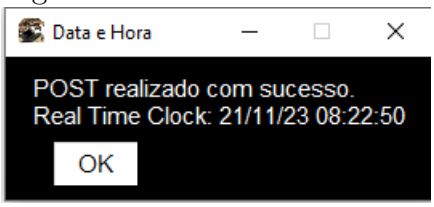
Fonte: Autoria própria.

5.6 Teste de Set RTC

No teste apresentado, o Arapuca estava com datas e horas desatualizadas. A *Figura* 5.13 representa o momento em que se clicou no botão **Set RTC**. Já a *Figura* 5.14 mostra a

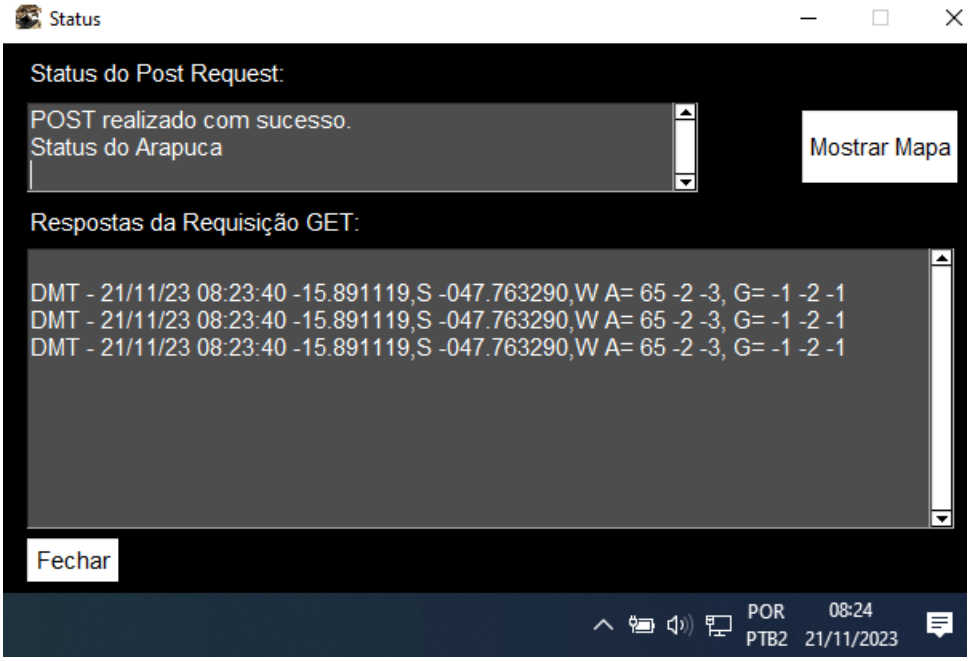
resposta atualizada do Arapuca, com valores de data e hora corrigidos.

Figura 5.13: Interface Set RTC



Fonte: Autoria própria.

Figura 5.14: Real Time CLock ajustado



Fonte: Autoria própria.

Capítulo 6

Conclusão

O tema principal do desenvolvimento do projeto Arapuca foi criar um dispositivo inovador capaz de detectar e alertar sobre furto de eletrodomésticos ou outros itens de valor. Este objetivo central orientou todas as fases de desenvolvimento desde a concepção do dispositivo até a sua implementação real. A motivação do projeto não é apenas recuperar itens furtados, mas também reduzir a incidência desse tipo de crime. Ao oferecer uma solução tecnológica eficaz para o monitoramento de bens, o projeto visava restaurar e fortalecer a confiança do usuário na segurança de seus pertences.

Os resultados obtidos com o dispositivo Arapuca foram notavelmente positivos, destacando especialmente quanto à eficiência do software e do webservice integrado. O sistema demonstrou uma capacidade eficiente de detecção e alerta em situações de furto, armazenando informações vitais como localização, data e hora. Esses dados provaram ser fundamentais para a recuperação dos objetos e para auxiliar nas investigações de furtos. O software, desenvolvido com foco na usabilidade e na interatividade, facilitou significativamente a operação do dispositivo pelo usuário. A implementação do webservice, utilizando Flask, proporcionou uma comunicação remota e eficiente com o hardware, permitindo o acesso e a gestão dos dados coletados de maneira simplificada e intuitiva. Essa integração de software e webservice não apenas melhorou a experiência do usuário, mas também aumentou a confiabilidade e a eficácia do sistema em situações reais de furto.

Em conclusão, o projeto Arapuca atingiu seu objetivo principal de desenvolver um dispositivo capaz de detectar e alertar furtos, proporcionando ao usuário uma maior segurança e a possibilidade de recuperação de seus bens.

Também é crucial avaliar o custo do hardware utilizado no projeto para verificar se sua aquisição é acessível para o cidadão comum. A *Tabela* 6.1 apresenta o orçamento dos componentes utilizados

Tabela 6.1: Orçamento do projeto Arapuca

Peça	Quantidade	Preço (R$)
Microcontrolador MSP430F5529LP	1	154,02
Módulo GPS GY-GPS6MV2	1	51,29
Memória externa 25Q32FVSIG	1	6,75
Módulo RTC DS3231	1	7,99
Módulo MPU6050	1	17,90
Módulo Wi-Fi ESP32	1	38,99
Power Bank	1	56,37
Total	7	333,31

Fonte: Autoria própria.

6.1 Propostas para futuras melhorias

Seguem-se algumas propostas que podem ser utilizadas para melhorar o projeto:

- Melhorar o funcionamento dos estados do Arapuca, implementar um meio que permitar de forma automática a transição entre os estados, por exemplo: estado **Suspeito** para o de **Alerta 1** e consequentemente do **Alerta 1** para o **Alerta 2**.
- Quando o dispositivo alertar possível furto, o usuário poder confirmar remotamente se realmente ocorreu, caso contrário envie mensagem para o dispositivo informando que foi um engano.
- Permitir o usuário alterar os valores referentes ao limite do MPU e GPS para alerta de furto, dentre a duração.
- Criação de um aplicativo mobile para facilitar ainda mais a monitoração.
- Integrar um modem USB ao hardware para garantir uma conexão constante à internet Wi-Fi.

Referências

[1] *Códigos e detalhes do projeto arapuka.* `https://github.com/Arapuka/arapukaunb2022`. 2, 59

[2] EDUARDO DE SOUSA MARTINS, VICTOR MENDES KOHL: *Dispositivo de segurança para aparelhos eletrônicos.* 2022. 5, 6, 28, 42

[3] Fang, Yi yuan e Xue jun Chen: *Design and simulation of uart serial communication module based on vhdl.* Em *2011 3rd International Workshop on Intelligent Systems and Applications*, páginas 1–4. IEEE, 2011. 5

[4] Depuru, Soma Shekara Sreenadh Reddy, Lingfeng Wang, Vijay Devabhaktuni e Nikhil Gudi: *Smart meters for power grid—challenges, issues, advantages and status.* Em *2011 IEEE/PES Power Systems Conference and Exposition*, páginas 1–7. IEEE, 2011. 5

[5] Alvi, Sheeraz A, Bilal Afzal, Ghalib A Shah, Luigi Atzori e Waqar Mahmood: *Internet of multimedia things: Vision and challenges.* Ad Hoc Networks, 33:87–111, 2015. 7

[6] Shin, Minho, Cory Cornelius, Dan Peebles, Apu Kapadia, David Kotz e Nikos Triandopoulos: *Anonysense: A system for anonymous opportunistic sensing.* Pervasive and Mobile Computing, 7(1):16–30, 2011. 7

[7] Sharma, Vijay Kumar, Vimal Kumar, Swati Sharma e Shashwat Pathak: *Python Programming: A Practical Approach.* CRC Press, 2021. 9

[8] Bawagan, Juan Miguel J: *A kidney exchange matching application using the blossom and hungarian algorithms for pairwise and multiway matching.* INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY, 13(02):229–247, 2020. 9

[9] Bonney, Matthew S, Marco De Angelis, Mattia Dal Borgo, Luis Andrade, Sandor Beregi, Nidhal Jamia e David J Wagg: *Development of a digital twin operational platform using python flask.* Data-Centric Engineering, 3:e1, 2022. 9

[10] Tadlaoui, Mouenis Anouar e Mohamed Chekou: *A blended learning approach for teaching python programming language: towards a post pandemic pedagogy.* International Journal of Advanced Computer Research, 11(52):13, 2021. 9

[11] Anggoro, Dimas Aryo e Nur Chudlori Aziz: *Implementation of k-nearest neighbors algorithm for predicting heart disease using python flask.* Iraqi Journal of Science, páginas 3196–3219, 2021. 9

[12] Al-Fuqaha, Ala, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari e Moussa Ayyash: *Internet of things: A survey on enabling technologies, protocols, and applications*. IEEE communications surveys & tutorials, 17(4):2347–2376, 2015. 11

[13] Esram, Trishan e Patrick L Chapman: *Comparison of photovoltaic array maximum power point tracking techniques*. IEEE Transactions on energy conversion, 22(2):439–449, 2007. 11

[14] R, Z.: *Embedded Systems*. Taylor and Francis Group, LLC, New York, 2006. 12

[15] Hochreiter, Sepp e Jürgen Schmidhuber: *Long short-term memory*. Neural computation, 9(8):1735–1780, 1997. 13

[16] Harris, Paul A, Robert Taylor, Brenda L Minor, Veida Elliott, Michelle Fernandez, Lindsay O'Neal, Laura McLeod, Giovanni Delacqua, Francesco Delacqua, Jacqueline Kirby *et al.*: *The redcap consortium: building an international community of software platform partners*. Journal of biomedical informatics, 95:103208, 2019. 13

[17] Berners-Lee, Timothy J e Robert Cailliau: *Worldwideweb: Proposal for a hypertext project*. 1990. 14

[18] Belshe, Mike e Roberto Peon: *Rfc 7540: hypertext transfer protocol version 2 (http/2)*, 2015. 14

[19] Grinberg, Miguel: *Flask: Web Development, One Drop at a Time*. " O'Reilly Media, Inc.", 2010. 14

[20] Grinberg, Miguel: *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2014. 14

[21] Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov e Liang Chieh Chen: *Mobilenetv2: Inverted residuals and linear bottlenecks*. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 4510–4520, 2018. 14

[22] Gao, Jianjiong, Bülent Arman Aksoy, Ugur Dogrusoz, Gideon Dresdner, Benjamin Gross, S Onur Sumer, Yichao Sun, Anders Jacobsen, Rileen Sinha, Erik Larsson *et al.*: *Integrative analysis of complex cancer genomics and clinical profiles using the cbioportal*. Science signaling, 6(269):pl1–pl1, 2013. 15

[23] Vogel, Patrick, Thijs Klooster, Vasilios Andrikopoulos e Mircea Lungu: *A low-effort analytics platform for visualizing evolving flask-based python web services*. Em *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, páginas 109–113. IEEE, 2017. 34

[24] Narayanan, Srikanth, NM Balamurugan, K Maithili e P Bini Palas: *Leveraging machine learning methods for multiple disease prediction using python ml libraries and flask api*. Em *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, páginas 694–701. IEEE, 2022. 35

[25] Chang, Yung Chia, Kuei Hu Chang e Yi Hsuan Huang: *A novel fuzzy credit risk assessment decision support system based on the python web framework*. Journal of Industrial and Production Engineering, 37(5):229–244, 2020. 35

[26] *Free api to send whatsapp messages*. https://www.callmebot.com/blog/free-api-whatsapp-messages/, Acesso em: 14 de nov. de 2023. 36, 59

[27] *Whatsapp é o aplicativo mais usado pelos brasileiros*. https://g1.globo.com/tecnologia/noticia/2022/01/11/whatsapp-e-o-aplicativo-mais-utilizado-por-metade-dos-brasileiros-confira-a-lista
ghtml, Acesso em: 14 de nov. de 2023. 36

[28] *Servidor smtp*. https://app.smtp2go.com/, Acesso em: 14 de nov. de 2023. 39, 55, 59, 61

[29] *Callmebot whatsapp*. https://www.callmebot.com/blog/free-api-whatsapp-messages/, Acesso em: 29 de nov. de 2023. 55, 59

[30] *Render*. https://dashboard.render.com, Acesso em: 27 de nov. de 2023. 55

[31] *site flask*. https://github.com/Arapuka/site, Acesso em: 27 de nov. de 2023. 55

[32] *Servidor gratuito para o site*. https://www.hashtagtreinamentos.com/servidor-gratuito-para-seu-site-railway-e-render-python, Acesso em: 29 de nov. de 2023. 55

[33] *Instalação selenium*. https://www.hashtagtreinamentos.com/automacao-web-no-python, Acesso em: 29 de nov. de 2023. 58

[34] *Download chromedriver*. https://chromedriver.chromium.org/downloads, Acesso em: 29 de nov. de 2023. 58

Apêndice A

Servidores, extensões e configurações

A.1 Bot Whatsapp

Foi utilizado um bot API para que fosse possível realizar o envio de mensagens para o whatsaap. Trata-se do [29], no site em questão demonstra os passos para sua instalação e uso.

A.2 Flask

Como dito no *tópico* 4.4.1, o web service foi desenvolvido com Flask, o qual pertence as 3 rotas. Para fazer o deploy dessa aplicação na web, é preciso um servidor, para tal foi utilizado o *Render* [30].

Para realizar a hospedagem dessa aplicação, basta logar com o *Github*. No local apropriado de realizar o deploy, fornecer qual o repositório que se deseja colocar na internet. No caso em questão é o repositório do web service desenvolvido [31].

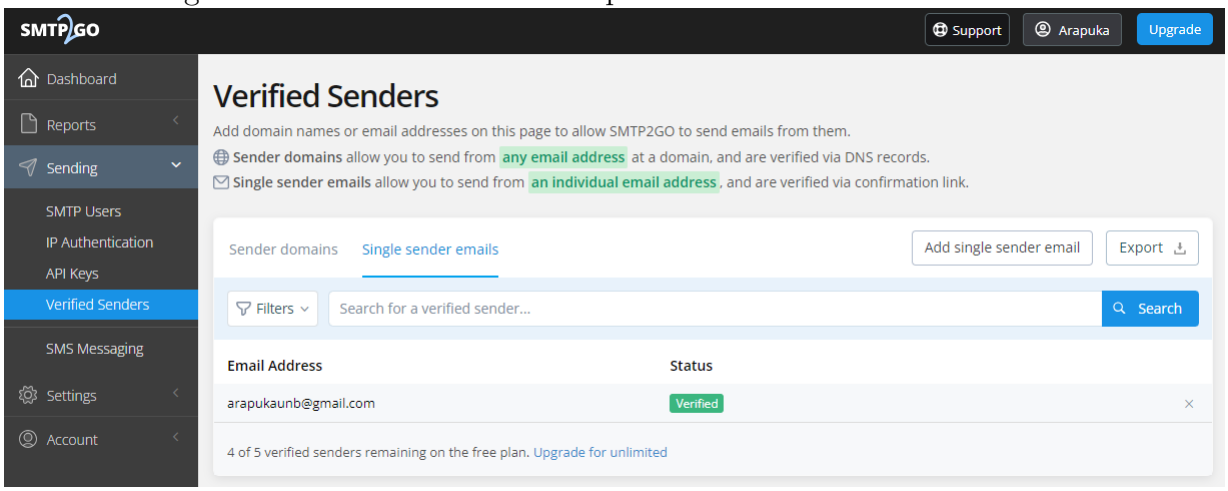
O site [32] demonstra como realizar a hospedagem de uma aplicação na web.

A.3 Servidor SMTP

Utilizando o servidor *SMTP2GO* [28], é preciso realizar a configuração, de colocar os e-mails autorizados para o envio de correios eletrônicos, dessa maneira evita qualquer usuário utilizar a conta e enviar e-mails indevidamente.

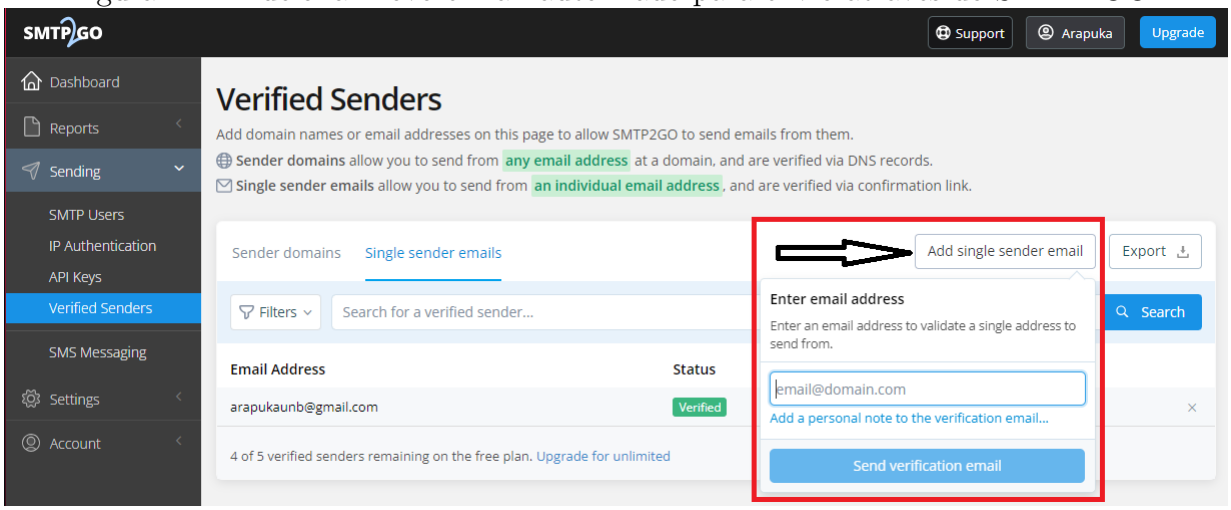
As *Figuras* A.1, A.2 aprensentam os passos para a configuração.

Figura A.1: E-mails autorizados para envio através do SMTP2GO



Fonte: Autoria própria.

Figura A.2: Adcionar novo e-mail autorizado para envio através do SMTP2GO



Fonte: Autoria própria.

A.4 Interface

Para o funcionamento da interface, segue algumas orientações.

A.4.1 Encerrar processo

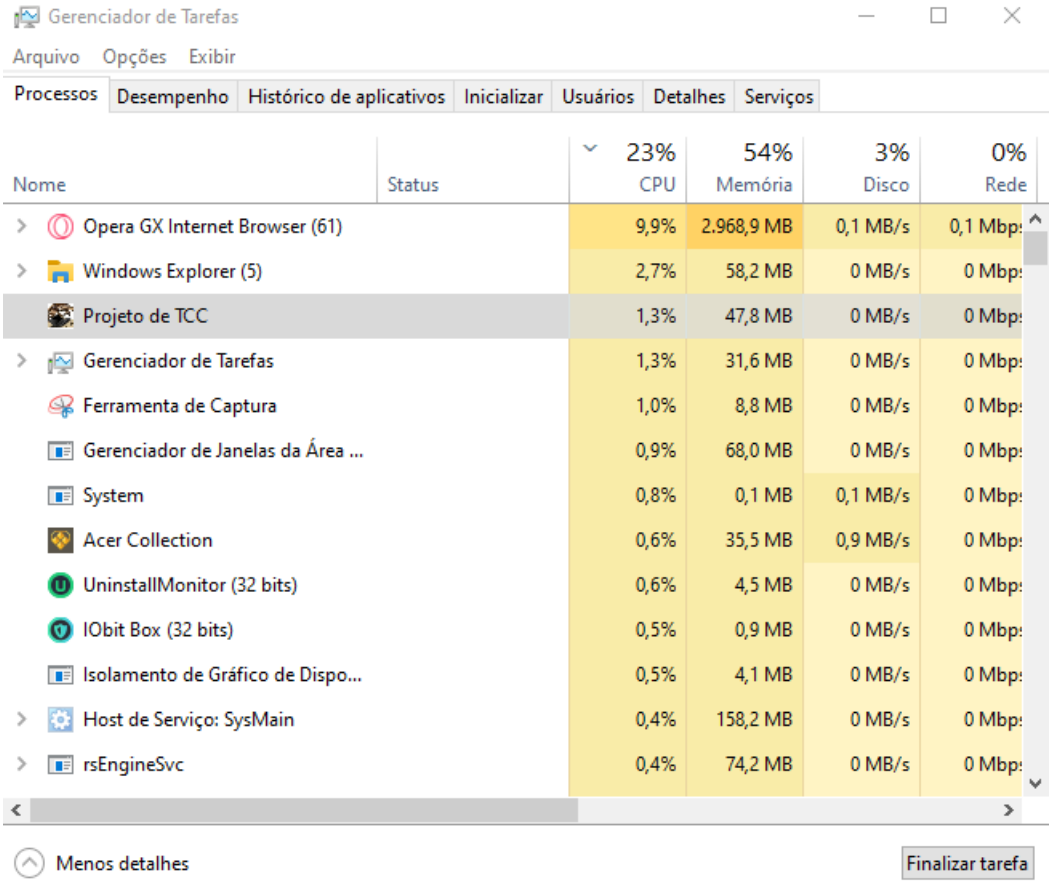
No software foi utilizado o principio de threads, que é uma programação concorrente. Um dos problemas no uso de threads deve-se que o software aprensentava muita lentidão no encerramento completo. Pensando nisso foi adotado o término de todas as threads apenas quando fosse fechado a interface. Todavia ainda assim teve-se um problema, pois mesmo

encerrando as devidas threads no encerramento da interface, o computador ainda alertava que o processo ainda estava em execução.

A razão de se encerrar esse processo, deve-se que caso abra outra janela da interface Arapuca, as informações irão se perder, pois poderão ser relacionadas ao processo que não foi encerrado anteriormente.

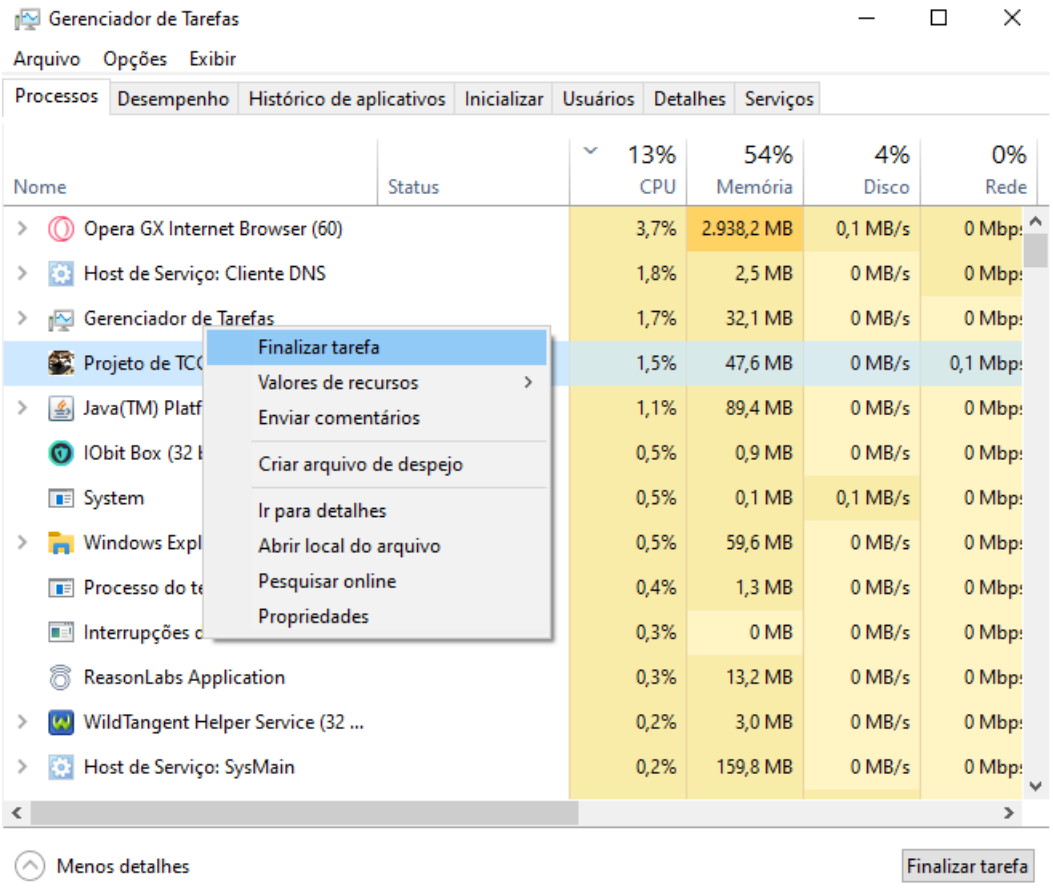
A *Figura* A.3 aprensenta o processo do software Arapuca aberto. A *Figura* A.4 apresenta a maneira de realizar o encerramento do processo.

Figura A.3: Processo da interface aberto



Fonte: Autoria própria.

Figura A.4: Finalizando processo da interface



Fonte: Autoria própria.

A.4.2 Selenium

Umas das funcionalidades da interface é a função de *Mostrar Mapa*, para que seja possível visualizar o mapa, é necessário ter o *Google Chrome instalado* além de instalar uma extensão "*Selenium*" no computador.

No site [33] demonstra como realizar o download e instalação do *Chromedriver* [34], extensão referente ao Selenium, necessária para realizar a visualização do mapa na tela do usuário.

Apêndice B

Códigos

Nesta seção será mostrada apenas alguns trechos de código. Caso tenha o desejo pode conferir todos eles no repositório do *GITHUB* [1].

B.1 ESP32

Nesta seção, segue alguns detalhes de implementação, desde ao bot de whatsapp utilizando o *Callmebot* [26], *SMTP* com o servidor *SMTP2GO* [28], comunicação com o msp, além de recebimento e envio de requisições *HTTP* com o Web Service.

Bibliotecas necessárias

```
/*
 * Intermediário para a comunicação do App com o Arapuca
 *
 */
#include <WiFi.h>
#include <HTTPClient.h>
#include <WiFiClientSecure.h>
#include <Base64.h>
#include <UrlEncode.h>
```

B.1.1 Bot whatsapp

Necessário colocar o número de telefone cadastrado no *CallmeBOT* [29], além da chave para conseguir receber mensagens no número cadastrado.

```
//Envia mensagem para o whatsApp (precisa inserir o numero e Key
   nas variaveis globais da main)
```

```
2  void WhatsAppMessages(String message){
3      int i=0;
4      //Serial.println("Iniciando Post");
                                                    //Debug
5      // Dados a serem enviados pelo método Post suportado pelo
          protocolo HTTP
6      String url = "https://api.callmebot.com/whatsapp.php?phone="
          + phoneNumber + "&apikey=" + apiKey + "&text=" + urlEncode(
          message);
7
8      HTTPClient http;

          // Cria um objeto HTTP
9      http.begin(url);

          // Inicia o objeto HTTP passando a string acima
10
11     http.addHeader("Content-Type", "application/x-www-form-
          urlencoded");         // Adiciona um cabeçalho
12
13     uint16_t httpResponseCode = http.POST(url);
                                          // Envia o método POST HTTP
           de requisição e lê o resultado
14     //Serial.println(url);
                                                                  //
          Debug
15     while(httpResponseCode != 200 && i<5){
16         httpResponseCode = http.POST(url);
17         i++;
18         Serial.println("Deu ruim");
                                                               //Debug
19     }
20
21     http.end();

          // Finaliza o objeto http
22 }
```

B.1.2 Envio de E-mail

Utiliza o servidor *SMTP2GO* [28]. Precisa realizar o cadastro e inserir o e-mail e senha de acesso ao servidor.

```
bool e_mail(String mensagem, String e_mail_to, String assunto,
   String coordenadas){
  // Configurações do servidor SMTP2GO
  char* server = "mail.smtp2go.com";
  int port = 465;

  // Configurações da conta de e-mail do servidor smtp2go
  char* email = "";
  char* email_password = "";

  // E-mail de envio(conforme configurado no smtp2go) e recepção(
     qualquer e-mail)
  char* email_from = "";
  String email_to = e_mail_to;

  // Inicializar a conexão segura com o servidor SMTP
  WiFiClientSecure client;
  client.setInsecure(); // Usar essa opção somente se o servidor
     SMTP não tiver certificado SSL válido

  if (!client.connect(server, port)) {
    Serial.println("Falha na conexão com o servidor SMTP!");
    return false;
  }
  Serial.println("Conectado ao servidor SMTP!");

  // Esperar a resposta do servidor SMTP
  waitForResponse(client);

  // Enviar comandos SMTP para enviar o e-mail
  client.println("EHLO 192.168.1.123");
  waitForResponse(client);

  client.println("AUTH LOGIN");
  waitForResponse(client);

```

```
34    client.println(base64::encode(email));
35    waitForResponse(client);
36
37    client.println(base64::encode(email_password));
38    waitForResponse(client);
39
40    client.println("MAIL FROM:<" + String(email_from) + ">");
41    waitForResponse(client);
42
43    client.println("RCPT TO:<" + String(email_to) + ">");
44    waitForResponse(client);
45
46    client.println("DATA");
47    waitForResponse(client);
48
49    client.println("Subject: " + String(assunto));
50    client.println("From: <" + String(email_from) + ">");
51    client.println("To: <" + String(email_to) + ">");
52    client.println("Content-Type: text/plain");
53    client.println("Arapuca movel\n\r" + String(mensagem) + "\n\r"
        + "https://www.google.com/maps?q="+ String(coordenadas)+ "\n\
        r att");
54    client.println(".");
55    waitForResponse(client);
56
57    client.println("QUIT");
58    waitForResponse(client);
59
60    // Fechar a conexão com o servidor SMTP
61    client.stop();
62    return true;
63  }
64
65  // Função para esperar a resposta do servidor SMTP com delays
      adequados
66  void waitForResponse(WiFiClientSecure& client) {
67    while (!client.available()) {
68      delay(100);
69    }
70    while (client.available()) {
```

```
71      String response = client.readStringUntil('\n');
72      Serial.println(response);
73    }
74  }
```

B.1.3 Comunicação com o MSP

Utilizando uma comunicação UART entre o ESP32 e MSP, utilizando os pinos GPIO 4(Tx) e GPIO 15(Rx) do ESP32, juntamente com os pinos P3.3 (Rx) e P3.4(Tx).

```
1  /*
2   * Funções referente ao Serial do MSP
3  */
4
5  //Separa o comando em duas strings, pega até o ultimo "#" para a
       primeira string e tudo depois do ultimo "#"
6  //até o primeiro ' ' para a segunda string, o restante na
       terceira string
7  //serivira principalmente para o e-mail
8  void splitMessage(const String &message, String result[]) {
9      int firstPos = message.indexOf('#');
10
11     if (firstPos != -1) {
12         int secondPos = message.indexOf('#', firstPos + 1);
13
14         if (secondPos != -1 && secondPos > firstPos) {
15             result[0] = message.substring(firstPos, secondPos +
                   1);
16             int spacePos = message.indexOf(' ', secondPos);
17
18             if (spacePos != -1) {
19                 result[1] = message.substring(secondPos + 1,
                       spacePos);
20                 result[2] = message.substring(spacePos + 1);
21             } else {
22                 result[1] = message.substring(secondPos + 1);
23                 result[2] = "";
24             }
25         } else {
26             result[0] = message;
```

```
27              result[1] = "";
28              result[2] = "";
29          }
30      } else {
31          result[0] = "";
32          result[1] = message;
33          result[2] = "";
34      }
35  }
36
37  //recebe o endereço da string que deverá ser enviada para o
       MSP430
38  void enviar_comando(const String &str) {
39      int tamanho = str.length();
40      for (int i = 0; i < tamanho; i++) {
41          Serial2.print(str[i]);
42          delay(2);
43      }
44      Serial2.print('\n');  //só para o ultimo caractere da
           mensagem ser um '\n', tem que ser aspas simples para
           reconhecer como unico caractere
45  }
46
47  // Receber tipo especifico de mensagem durante alguns segundos
48  String receber_mensagem_t_segundos(int tempoSegundos){
49    unsigned long startTime = millis();  // Armazena o tempo
         inicial
50    String response= "";
51
52    while ((millis() - startTime) < (tempoSegundos * 1000)) {  //
         Loop durante o tempo especificado em segundos
53      if (Serial2.available()) {  // Verifica se há dados disponí
           veis na Serial2
54        response = Serial2.readStringUntil('\0');  // Lê a resposta
             até encontrar o caractere nulo
55
56        // Verifica se o primeiro e ultimo caractere são iguais a
             '#'
57            if (response.charAt(0) == '#' && response.charAt(
                 response.length() - 1) == '#') {
```

```
58
59          response += response;
60        }
61  //      else{                           //significa que a mensagem
        não foi satisfeita
62  //        //response += "";
63  //        //return response;
64  //      }
65      }
66    }
67
68    Serial.println(response);
69    return response;
70  }
71  //// Receber caracteres por interrupção
                                  Nao deu certo, ficou ruim
72  //void IRAM_ATTR serial_Interrupt() {
73  //  noInterrupts(); // Desativa interrupções
74  //  char character = Serial2.read();
75  //  if (!startRecording && character >= ' ' && character <= 'z')
        {
76  //    startRecording = true;
77  //    recordingStartTime = millis();
78  //    resposta = "";
79  //  } else if (startRecording) {
80  //    resposta += character;
81  //    if (millis() - recordingStartTime >= recordingDuration) {
82  //      startRecording = false;
83  //    }
84  //  }
85  //  interrupts(); // ativa interrupções
86  //}
87
88  // Receber tipo especifico de mensagem
89  String receber_mensagem(){
90    String response= "";
91    if (Serial2.available()) {  // Verifica se há dados disponíveis
          na Serial2
92      response = Serial2.readStringUntil('\0');  // Lê a resposta
           até encontrar o caractere nulo
```

```
93      if (response.charAt(0) == '#') {
                                                  // Verifica se o
        primeiro caractere é igual a '#'
94        if (response.charAt(response.length() - 1) == '#') {
                                // Verifica se o último caractere é
          igual a '#'
95          Serial.println(response);
96          return response;
97        }
98      }
99      else{                           //significa que a mensagem não
         foi satisfeita
100       Serial.println(response);
101       response = "";
102       return response;
103     }
104   }
105 }
106
107 /*
108 //Recebe todo tipo de mensagem
109 String receber_mensagem(){
110   String response= "";
111   if (Serial2.available()) {  // Verifica se há dados disponíveis
        na Serial2
112     response = Serial2.readStringUntil('\0');  // Lê a resposta
         até encontrar o caractere nulo
113
114         return response;
115    }
116    return response;
117 }
118 */
119
120 // remove o primeiro e ultimo caractere
121 String removePrimeiroUltimoCaractere(const String& str) {
122   return str.substring(1, str.length() - 1);
123 }
124 // Remove todos os caracteres de # e substitui por nada
125 void remove_hashtag(String &input) {
```

```
126   input.replace("#", "");
127 }
128
129 //Separa o comando de RD, servira para saber a quantidade de
      iterações usar
130 void rd_string(const String &message, String result[]) {
131     // Encontra a posição do primeiro espaço em branco
132     int firstSpacePos = message.indexOf(' ');
133     int secondPos = message.indexOf('#', firstSpacePos);
134
135     // Verifica se a string começa com "#RD" ou "RD"
136     if (message.startsWith("#RD ") || message.startsWith("RD "))
          {
137         // Divide a string em partes usando o espaço em branco
              como delimitador
138         ///result[0] = "RD"; // A primeira parte é sempre "RD"
139
140         // Verifica se há um segundo espaço em branco
141         if (firstSpacePos != -1) {
142             // Encontra a posição do próximo espaço em branco após
                  o primeiro espaço
143             int secondSpacePos = message.indexOf(' ',
                  firstSpacePos + 1);
144
145             // Se houver um segundo espaço, divide a string em tr
                  ês partes
146             if (secondSpacePos != -1) {
147                 result[1] = message.substring(firstSpacePos + 1,
                      secondSpacePos);
148                 result[2] = message.substring(secondSpacePos + 1,
                      secondPos);
149             }
150             // Se não houver um segundo espaço, divide a string
                  em duas partes
151             else {
152                 result[1] = message.substring(firstSpacePos + 1,
                      secondPos);
153                 result[2] = "";
154             }
155         } else {
```

```
156             // Se não houver um segundo espaço, a segunda parte é
                    uma string vazia
157             result[1] = "";
158             result[2] = "";
159         }
160     }
161 }
```

B.1.4 Comunicação com Web Service

Utilizando requisições HTTP.

```
1  const char* url_comando = "https://arapuka.onrender.com/comando";
2  const char* url_resposta = "https://arapuka.onrender.com/resposta
       ";
3
4  // Pega o comando na pagWeb
5  String get_comando(){
6    int i=0;
7    // Criar um objeto HTTPClient
8    HTTPClient http;
9
10   String response = "";
11
12   // Enviar a requisição GET
13   http.begin(url_comando);
14   int httpCode = http.GET();
15
16   // Verificar o código de resposta
17   if (httpCode > 0) {
18     // Ler o conteúdo da resposta
19     response = http.getString();
20   }
21   while(httpCode < 1 && i < 5){
22     response = http.getString();
23     httpCode = http.GET();
24     i++;
25   }
26
27   return response;
```

```
28 }
29 // Deleta o ultimo comando no servidor, para evitar pegar mais de
       uma vez
30 void delete_post_comando(){
31   int i=0;
32   // Cria o objeto HTTPClient
33     HTTPClient http;
34
35     // Envia a requisição POST
36     http.begin(url_comando);
37     http.addHeader("Content-Type", "text/html");
38     int httpResponseCode = http.POST("");
39
40     while(httpResponseCode < 1 && i < 5){
41       // ReEnvia a requisição POST
42       Serial.println("Reenviando Requisição POST");
43       httpResponseCode = http.POST("");
44       i++;
45     }
46
47     // Libera os recursos do objeto HTTPClient
48     http.end();
49 }
50
51 // Envia a resposta do Arapuka para o servidor
52 void post_resposta(String mensagem){
53   Serial.println("Inicio do Post resposta = \n\r"+ String(
         mensagem));
54   int i = 0;
55   // Cria o objeto HTTPClient
56   HTTPClient http;
57
58   // Envia a requisição POST
59   http.begin(url_resposta);
60   http.addHeader("Content-Type", "text/html");
61   int httpResponseCode = http.POST(mensagem);
62
63   while(httpResponseCode < 1 && i < 5){
64     // ReEnvia a requisição POST
65     Serial.println("Reenviando Requisição POST");
```

```
66      httpResponseCode = http.POST(mensagem);
67      i++;
68    }
69    // Libera os recursos do objeto HTTPClient
70    http.end();
71    delay(100);
72    Serial.println("Fim do Post resposta");
73  }
```

B.2 Web Service

O web service foi desenvolvido com Flask e Python, abaixo segue o código para implementação.

```
1  from flask import Flask, request
2
3  app = Flask(__name__)
4
5  @app.route("/", methods=["GET"])
6  def homepage():
7      return '''
8          <html>
9          <head>
10             <title>Arapuka</title>
11         </head>
12         <body>
13             <br/><br/><br/><br/><br/><br/><br/><br/><br/>
14             <center><h1>Bem vido ao Arapuka móvel</h1></center>
15             <br/><br/><br/><br/><br/><br/><br/><br/><br/>
16             <center><p>Trabalho de conclusão de curso</p></center
                   >
17         </body>
18         </html>
19     '''
20
21 @app.route("/comando", methods=["GET", "POST"])
22 def comando():
23     if request.method == "GET":
```

```
24          if 'mensagem_comando' in app.config:
                                        #caso exista alguma
            mensagem_comando na variável de configuração
25              mensagem_comando = app.config['mensagem_comando']
26              return mensagem_comando
27          else:
28              return 'Nenhuma mensagem_comando foi postada ainda.'
29
30      elif request.method == "POST":
31          mensagem_comando = request.data.decode('utf-8')
                            # Obtém a mensagem do corpo (se for html)
            da requisição
32          app.config['mensagem_comando'] = mensagem_comando
                        # Armazena a mensagem_comando na variável de
            configuração
33          return 'Mensagem_comando postada'
34
35  @app.route("/resposta", methods=["GET", "POST"])
36  def resposta():
37      if request.method == "GET":
38          if 'mensagem_resposta' in app.config:
                                        #caso exista alguma
            mensagem_resposta na variável de configuração
39              mensagem_resposta = app.config['mensagem_resposta']
40              return mensagem_resposta
41          else:
42              return 'Nenhuma mensagem_resposta foi postada ainda.'
43
44      elif request.method == "POST":
45          mensagem_resposta = request.data.decode('utf-8')
                            # Obtém a mensagem_resposta do corpo (se
            for html) da requisição
46          app.config['mensagem_resposta'] = mensagem_resposta
                        # Armazena a mensagem na variável de configura
            ção
47          return 'Mensagem_resposta postada'
48
49  @app.route("/status", methods=["GET", "POST"])
50  def status():
51      if request.method == "GET":
```

```
52          if 'mensagem_status' in app.config:
                                        #caso exista alguma
            mensagem_status na variável de configuração
53              mensagem_status = app.config['mensagem_status']
54              return mensagem_status
                                                    # se chegar aqui, a
                mensagem é "ON"
55          else:
56              return 'Status OFF'
57
58      elif request.method == "POST":
59          mensagem_status = request.data.decode('utf-8')
                        # Obtém a mensagem_resposta do corpo (se
            for html) da requisição
60          app.config['mensagem_status'] = mensagem_status
                       # Armazena a mensagem na variável de
            configuração
61          return 'Status ON'
62
63  if __name__ == "__main__":
64      app.run(debug=True)
```

B.3 Interface Python

Nesta etapa, foi utilizado o PysimpleGUI, uma biblioteca em Python, que serve para desenvolvimento de interfaces gráficas.

B.3.1 Código principal

```
1  import PySimpleGUI as sg
2  from PIL import Image
3  import io
4  import time
5  import sys
6  import os
7
8  import eventos
9
10 # diretorio_atual = os.path.dirname(os.path.abspath(__file__))
```

```
11 # srcs_path = os.path.abspath(os.path.join(diretorio_atual,'.', '
       srcs'))
12 # sys.path.insert(1,srcs_path)
13
14 # from srcs.eventos import *
15 # from srcs.requisicoes import *
16
17 app_title = 'Arapuca'
18 themes = sorted(list(sg.LOOK_AND_FEEL_TABLE.keys()))
19 sg.ChangeLookAndFeel('DarkBlack')
20 sg.SetOptions(font='any 11', auto_size_buttons=True,
       progress_meter_border_depth=0, border_width=1)
21
22 ##################################################################
23 def introduction_animation():
24     # Carregar e exibir a imagem de introdução
25     intro_image = Image.open('arapuka.png')  # Substitua pelo
          caminho da sua imagem
26     intro_image.thumbnail((300, 300))  # Redimensionar a imagem
          para caber na tela
27     bio = io.BytesIO()
28     intro_image.save(bio, format="PNG")
29     layout_intro = [
30         [sg.Image(data=bio.getvalue())]
31     ]
32     window_intro = sg.Window('Arapuca', layout_intro, finalize=
          True,icon='arapuka.ico')
33
34     # Aguardar um tempo curto para exibir a imagem de introdução
35     sg.popup_quick_message('Carregando...')
36     time.sleep(1)
37     # Fechar a janela de introdução e abrir a janela principal
38     window_intro.close()
39 ##################################################################
40
41 # gui design
42 def create_window():
43     # main tab
44     layout = [[sg.T('', size=(17, 1)), sg.Text('Arapuca', font='
          Any 50')],
```

```
45
46                  [sg.Text('\n\n')],
47                  [sg.Button('Console', size=(10, 2), pad=(10, 10))
                        ],
48                  [sg.Text('Estados:')],
49                  # botões dos estados
50                  [sg.Button('Dormente', size=(10, 2), pad=(10, 10)
                        ), sg.Button('Vigilia', size=(10, 2), pad=(10,
                        10)), sg.Button('Suspeito', size=(10, 2), pad
                        =(10, 10)), sg.Button('Alerta 1', size=(10, 2),
                         pad=(10, 10)), sg.Button('Alerta 2', size=(10,
                         2), pad=(10, 10))],
51
52
53                  [sg.Text('\n\nAções:')],
54                  #botoes das açoes extras
55                  [sg.Button('Status', size=(10, 2), pad=(10, 20)),
                         sg.Button('Rd n', size=(10, 2), pad=(10, 20)),
                         sg.Button('Rd n_m', size=(10, 2), pad=(10, 20)
                        ), sg.Button('Apagar Memoria', size=(10, 2),
                        pad=(10, 20)), sg.Button('E-mail', size=(10, 2)
                        , pad=(10, 20))],
56                  [sg.Button('Resete', size=(10, 2), pad=(10, 20)),
                         sg.Button('Set RTC', size=(10, 2), pad=(10,
                        20)), sg.Button('Leds', size=(10, 2), pad=(10,
                        20))],
57
58              ]
59
60
61
62      # window
63      return sg.Window(title=app_title, layout=layout, size=(600,
            550), icon='arapuka.ico')
64
65  introduction_animation()
66  window = create_window()
67
68  while True:
69      event, values = window.read()
```

```
70
71      if event == sg.WINDOW_CLOSED:
72          break
73
74      elif event == 'Console':
75          window.hide()
76          eventos.log()
77          window.un_hide()
78
79      #Eventos dos Estados
80      elif event == 'Dormente':
81          window.hide()
82          eventos.dormente()
83          window.un_hide()
84
85      elif event == 'Vigilia':
86          window.hide()
87          eventos.vigilia()
88          window.un_hide()
89
90      elif event == 'Alerta 1':
91          window.hide()
92          eventos.alerta_1()
93          window.un_hide()
94
95      elif event == 'Alerta 2':
96          window.hide()
97          eventos.alerta_2()
98          window.un_hide()
99
100     elif event == 'Suspeito':
101         window.hide()
102         eventos.suspeito()
103         window.un_hide()
104
105     #Eventos das Ações
106     elif event == 'Status':
107         window.hide()
108         eventos.status()
109         window.un_hide()
```

```
110
111     elif event == 'Rd n':
112         window.hide()
113         eventos.rd_n()
114         window.un_hide()
115
116     elif event == 'Rd n_m':
117         window.hide()
118         eventos.rd_n_m()
119         window.un_hide()
120
121     elif event == 'Apagar Memoria':
122         window.hide()
123         eventos.apagar_memoria()
124         window.un_hide()
125
126     elif event == 'E-mail':
127         window.hide()
128         eventos.email()
129         window.un_hide()
130
131     elif event == 'Resete':
132         window.hide()
133         eventos.resete()
134         window.un_hide()
135
136     elif event == 'Set RTC':
137         window.hide()
138         eventos.rtc()
139         window.un_hide()
140
141     elif event == 'Leds':
142         window.hide()
143         eventos.leds()
144         window.un_hide()
145
146
147 if eventos.update_thread is not None and eventos.update_thread.
    is_alive():
148     eventos.update_thread.join()
```

```
149
150 window.close()
151 sys.exit()  # Encerra o processo Python
```

B.3.2 Requisições

```
1  import requests
2
3  url_status = 'https://arapuka.onrender.com/status'
4  url_comando = 'https://arapuka.onrender.com/comando'
5  url_resposta = 'https://arapuka.onrender.com/resposta'
6
7  #posta mensagem na url /comando
8  def post_comando(comando):
9      # Cabeçalhos da requisição
10     headers = {
11     'Content-Type': 'text/html'
12     }
13     try:
14         response = requests.post(url_comando, headers=headers,
               data=comando)
15         if response.status_code == 200:
16             return "POST realizado com sucesso."
17         else:
18             return f"Erro no POST request: {response.status_code}
                   "
19     except requests.exceptions.RequestException as e:
20         return f"Erro no POST request: {e}"
21
22
23 #posta mensagem em branco na url /resposta, serve para pegar só
       uma vez o conteúdo
24 def delete_resposta():
25     # Cabeçalhos da requisição
26     headers = {
27     'Content-Type': 'text/html'
28     }
29     for i in range(5):
```

```
30          response = requests.post(url_resposta, headers=headers,
               data='')
31          if response.status_code == 200:
32              return
33
34
35  #pega mensagem em qualquer url
36  # a url eh definida na threading.Thread(target=update_console,
       args=(new_window['OutputGet'], requisicoes.url_comando))
37  def get_resposta(url):
38      try:
39          response = requests.get(url)
40          if response.status_code == 200:
41              return response.text
42          else:
43              return f"Erro na solicitação: {response.status_code}"
44      except requests.exceptions.RequestException as e:
45          return f"Erro na solicitação: {e}"
```

B.3.3 Eventos

Este é o código responsável com a lógica de cada comando.

```
1   import PySimpleGUI as sg
2   import threading
3   import datetime
4   import queue
5   import requisicoes
6   import auxiliares
7
8   # global update_thread
9   # update_thread = None
10  update_queue = queue.Queue()
11  update_thread = None
12
13  def log():
14      estado_com_mapa("","Console", "") #tem apenas o argumento de
           Estado
15
16  # Funções para cada botão dos Estados
```

```
17 def dormente():
18     estado_duo_sem_mapa("#DMT#", "Dormente", "Estado Dormente")
19
20 def vigilia():
21     estado_duo_com_mapa("#VIG#", "Vigilia", "Estado Vigilia")
22
23 def alerta_1():
24     estado_duo_com_mapa("#ALT1#", "Alerta 1", "Estado de Alerta 1
       ")
25
26 def alerta_2():
27     estado_duo_com_mapa("#ALT2#", "Alerta 2", "Estado de Alerta 2
       ")
28
29 def suspeito():
30     estado_duo_com_mapa("#SPT#", "Suspeito", "Estado Suspeito")
31
32 # Funções para cada botão das Ações
33 def status():
34     estado_duo_com_mapa("#STAT#", "Status", "Status do Arapuca")
35
36 def rd_n():
37     window = auxiliares.rd_n()
38     while True:
39         event, values = window.read()
40
41         if event == sg.WINDOW_CLOSED:
42             break
43         elif event == 'Enviar':
44             window.close()
45             number_n = int(values['combo'])
46                             #chama a função que envia o POST
47             estado_duo_sem_mapa(f"#RD {number_n}#", "Leiturna de
                n Registros", f"Pedido de leitura de {number_n}
                Registros enviado!")
48
49     window.close()
50
51 def rd_n_m():
52     window = auxiliares.rd_n_m()
```

```
53      while True:
54          event, values = window.read()
55
56          if event == sg.WINDOW_CLOSED:
57              break
58          elif event == 'Enviar':
59
60              number_n = int(values['combo1'])
61              number_m = int(values['combo2'])
62              #chama a função que envia o POST
63      #
           ##################################################################

64              if(auxiliares.is_valid_RD_n_m(number_n, number_m)==
                   True):
65                  window.close()
66                  estado_duo_sem_mapa(f"#RD {number_n} {number_m}#"
                       , "Leiturna de n até m Registros", f"Pedido de
                       leitura do {number_n}   até o {number_m}
                       Registro enviado!")
67
68              else:
69                  sg.popup_error('O segundo numero precisa ser
                       maior que o primeiro.',title="Números inválidos
                       ", icon='arapuka.ico')
70
71      window.close()
72
73  def apagar_memoria():
74      estado_duo_sem_mapa("#APG#", "Apagar Memória", f"Pedido para
           Apagar Memória enviado!")
75
76
77  def email():
78      window = auxiliares.email_layout()
79      while True:
80          event, values = window.read()
81
82          if event == sg.WINDOW_CLOSED:
83              window.close()
```

```
84              break
85          elif event == 'Enviar':
86              email = values['email']
87              if auxiliares.is_valid_email(email):
88                  window.close()
89                  #chama a função que envia o POST com o email e
                        mostra o resultado no console
90                  estado_duo_sem_mapa(f"#MAIL#{email} {values['
                        assunto']}", "E-mail", f"Pedido de envio de E-
                        mail com assunto: {values['assunto']}, para o
                        endereço {email} enviado!")
91
92              else:
93                  sg.popup_error('E-mail inválido. Por favor,
                        digite novamente.', title="E-mail inválido",
                        icon='arapuka.ico')
94      window.close()
95
96  def resete():
97      estado_duo_sem_mapa(f"#RST#", "Resete", "Pedido de Resete do
            Arapuca enviado!")
98
99
100 def rtc():
101     agora = datetime.datetime.now()
102     data_hora_formatada = agora.strftime("%d/%m/%y %H:%M:%S")
103     #envia a requisição POST
104     post_result = requisicoes.post_comando(f"#RTC {
            data_hora_formatada}#")
105     sg.popup(f"{post_result}\nReal Time Clock: {
            data_hora_formatada}", title="Data e Hora", icon='arapuka.
            ico')
106
107 ##################################################################
108 ##################################################################
109 #Esta funçao fará com que acenda os leds da placa msp430
110 def leds():
111     window = auxiliares.leds_layout()
112     while True:
113         event, values = window.read()
```

```
114
115         if event == sg.WINDOW_CLOSED:
116             window.close()
117             break
118         elif event == 'led_verde_on':
119             requisicoes.post_comando("#green on#")
120 
121         elif event == 'led_verde_off':
122             requisicoes.post_comando("#green off#")
123 
124         elif event == 'led_vermelho_on':
125             requisicoes.post_comando("#red on#")
126 
127         elif event == 'led_vermelho_off':
128             requisicoes.post_comando("#red off#")
129 
130 ##################################################################
131 def estado_duo_sem_mapa(comando, estado, descricao):
132     #
        ##################################################################

133     #envia a requisição POST
134     post_result = requisicoes.post_comando(comando)
135 
136     #cria console
137     new_window = auxiliares.console_duo(estado)
138 
139     print(post_result, file=new_window["OutputPost"])   #Coloca
        se conseguiu enviar a mensagem ou não
140     print(descricao, file=new_window["OutputPost"])
141 
142     #Requisição GET
143     global update_thread
144     update_thread = threading.Thread(target=auxiliares.
        update_console, args=(new_window['OutputGet'], requisicoes.
        url_resposta, update_queue))
145     update_thread.start()
146 
147     while True:
148         event, values = new_window.read(timeout=100)
```

```
149         if event == sg.WINDOW_CLOSED or event == 'Fechar':
150             # if update_thread is not None and update_thread.
                    is_alive():
151             #     update_thread.join()
152             #update_thread.join()                #espera terminar
                    thread, mas enrola demais (não é boa pratica deixar
                     sem essa ação)
153             new_window.close()
154             break
155         try:
156             resposta = update_queue.get_nowait()
157             new_window['OutputGet'].update(value=new_window['
                    OutputGet'].get() + '\n' + resposta)
158         except queue.Empty:
159             pass
160     #
            ##################################################################


161
162 def estado_duo_com_mapa(comando, estado, descricao):
163     #
            ##################################################################

164     #envia a requisição POST
165     post_result = requisicoes.post_comando(comando)
166
167     #cria console
168     new_window = auxiliares.console_duo_map(estado)
169
170     print(post_result, file=new_window["OutputPost"])   #Coloca
            se conseguiu enviar a mensagem ou não
171     print(descricao, file=new_window["OutputPost"])
172
173     #Requisição GET
174     global update_thread
175     update_thread = threading.Thread(target=auxiliares.
            update_console, args=(new_window['OutputGet'], requisicoes.
            url_resposta, update_queue))
176     update_thread.start()
177
```

```
178     while True:
179         event, values = new_window.read(timeout=100)
180         if event == sg.WINDOW_CLOSED or event == 'Fechar':
181             # if update_thread is not None and update_thread.
                    is_alive():
182             #     update_thread.join()
183             #update_thread.join()                #espera terminar
                    thread, mas enrola demais (não é boa pratica deixar
                     sem essa ação)
184             new_window.close()
185             break
186         elif event == 'Mostrar Mapa':
187             # sg.popup(f"{auxiliares.localization}")
188             if auxiliares.localization != None:     #Se for
                    diferente de vazia tenta mostrar o mapa
189                 latitude, longitude = auxiliares.
                        extrair_latitude_longitude(auxiliares.
                        localization)
190                 if (latitude and longitude) != False:
191                     try:
192                         latitude = float(latitude)
193                         longitude = float(longitude)
194 
195                         thread = threading.Thread(target=
                            auxiliares.exibir_mapa, args=(latitude,
                             longitude))
196                         thread.start()
197 
198                         # Espera a thread terminar antes de abrir
                             a nova janela
199                         thread.join()
200 
201                         try:
202                             update_type, update_data = auxiliares
                                .update_queue.get_nowait()
203                             if update_type == 'show_map':
204                                 auxiliares.criar_janela_mapa(
                                    update_data)
205                             elif update_type == 'error':
206                                 sg.popup_error(update_data)
```

```
207                         except queue.Empty:
208                             pass
209
210                     except ValueError:
211                         sg.popup_error('Valores inválidos para
                               Latitude e Longitude.', title="Map
                               Error", icon='arapuka.ico')
212             else:
213                 sg.popup_error('Ainda Não possui dados da última
                       localização.',title="Map Error", icon='arapuka.
                       ico')
214         try:
215             resposta = update_queue.get_nowait()
216             new_window['OutputGet'].update(value=new_window['
                   OutputGet'].get() + '\n' + resposta)
217         except queue.Empty:
218             pass
219     #
        ##################################################################
220
221 def estado_com_mapa(comando, estado, descricao):
222     #cria console
223     new_window = auxiliares.console_get(estado)
224
225     #Requisição GET
226     global update_thread
227     update_thread = threading.Thread(target=auxiliares.
           update_console, args=(new_window['OutputGet'], requisicoes.
           url_resposta, update_queue))
228     update_thread.start()
229
230     while True:
231         event, values = new_window.read(timeout=100)
232         if event == sg.WINDOW_CLOSED or event == 'Fechar':
233             # if update_thread is not None and update_thread.
                   is_alive():
234             #     update_thread.join()
```

```
235             #update_thread.join()                #espera terminar
                    thread, mas enrola demais (não é boa pratica deixar
                     sem essa ação)
236             new_window.close()
237             break
238         elif event == 'Mostrar Mapa':
239             # sg.popup(f"{auxiliares.localization}")
240             if auxiliares.localization != None:     #Se for
                    diferente de vazia tenta mostrar o mapa
241                 latitude, longitude = auxiliares.
                        extrair_latitude_longitude(auxiliares.
                        localization)
242                 if (latitude and longitude) != False:
243                     try:
244                         latitude = float(latitude)
245                         longitude = float(longitude)
246
247                         thread = threading.Thread(target=
                            auxiliares.exibir_mapa, args=(latitude,
                             longitude))
248                         thread.start()
249
250                         # Espera a thread terminar antes de abrir
                             a nova janela
251                         thread.join()
252
253                         try:
254                             update_type, update_data = auxiliares
                                .update_queue.get_nowait()
255                             if update_type == 'show_map':
256                                 auxiliares.criar_janela_mapa(
                                    update_data)
257                             elif update_type == 'error':
258                                 sg.popup_error(update_data)
259                         except queue.Empty:
260                             pass
261
262                     except ValueError:
```

```
263                         sg.popup_error('Valores inválidos para
                               Latitude e Longitude.', title="Map
                               Error", icon='arapuka.ico')
264             else:
265                 sg.popup_error('Ainda Não possui dados da última
                       localização.',title="Map Error", icon='arapuka.
                       ico')
266         try:
267             resposta = update_queue.get_nowait()
268             new_window['OutputGet'].update(value=new_window['
                   OutputGet'].get() + '\n' + resposta)
269         except queue.Empty:
270             pass
```

B.3.4 Funções auxiliares, outras janelas

Funções auxiliares a fim de ajudar a interface, dentre o código responsável por atualizar as janelas da interface. Código da fila de mensagens, de mostrar mapa etc.

```
1
2  import PySimpleGUI as sg
3  import requisicoes
4  import time
5  import re
6  import threading
7  import folium
8  from selenium import webdriver
9  import os
10 import queue
11
12 #global que salva a localização para mostrar o mapa
13 localization = None
14
15 # Cria uma fila de mensagens para atualizações na janela
16 global update_queue
17 update_queue = queue.Queue()
18
19 '''
20 ###################################################################
21 #                                                      CONSOLE
```

```
22 #
   ##################################################################

23  Utiliza Threads
24 '''
25 #Dois consoles sem o mapa, mostrando a resposta do POST e do GET
26 def console_duo(Estado):
27     layout = [
28         [sg.Text('Status do Post Request:')],
29         [sg.Output(size=(70, 3), key='OutputPost')],
30         [sg.Text('Respostas da Requisição GET:')],
31         [sg.Output(size=(70, 10), key='OutputGet')],
32         [sg.Button('Fechar')]
33     ]
34
35     window = sg.Window(Estado, layout, finalize=True, icon='
          arapuka.ico')
36
37     return window
38
39 #Dois consoles com o mapa, mostrando a resposta do POST e do GET
40 def console_duo_map(Estado):
41     layout = [
42         [sg.Text('Status do Post Request:')],
43         [sg.Output(size=(50, 3), key='OutputPost'),sg.T('', size
              =(4, 1)), sg.Button('Mostrar Mapa', size=(10, 2), pad
              =(10, 10))],
44         [sg.Text('Respostas da Requisição GET:')],
45         [sg.Output(size=(70, 10), key='OutputGet')],
46         [sg.Button('Fechar')]
47     ]
48
49     window = sg.Window(Estado, layout, finalize=True, icon='
          arapuka.ico')
50
51     return window
52
53 #único console, mostrando a resposta do GET, com opção de mostrar
       mapa
54 def console_get(Estado):
```

```
55      layout = [
56          [sg.Button('Mostrar Mapa', size=(10, 2), pad=(10, 10))],
57          [sg.Text('Respostas da Requisição GET:')],
58          [sg.Output(size=(70, 10), key='OutputGet')],
59          [sg.Button('Fechar')]
60      ]
61
62      window = sg.Window(Estado, layout, finalize=True, icon='
            arapuka.ico')
63
64      return window
65
66  #atualiza toda parte GET do Console e coloca janela com mapa
        antes de apagar a mensagem
67  def update_console(output_elem, url, update_queue):
68      global localization
69      while True:
70          resposta = requisicoes.get_resposta(url)
71          #Restrições para que a mensagem possa ser mostrada no
                console
72          #Mensagem que mostra quando não foi postada nenhuma
                mensagem desde que o site foi iniciado
73          if resposta != "Nenhuma mensagem_resposta foi postada
                ainda.":
74              if len(resposta) > 30:
75                  localization = resposta
76              # output_elem.update(value=output_elem.get() + '\n' +
                    resposta)
77              # #apaga a ultima resposta, garante que vai pegar
                    apenas uma vez a resposta
78              update_queue.put(resposta)
79              requisicoes.delete_resposta()
80          time.sleep(2)
81
82  ##################################################################
83  #                                                  E-MAIL
84  #
        ##################################################################
85
```

```
86  #checa se o email eh valido
87  def is_valid_email(email):
88      # Utiliza uma expressão regular para validar o formato do e-
            mail
89      pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
90      return re.match(pattern, email)
91
92  def email_layout():
93      layout = [
94          [sg.Text('E-mail de destino:'), sg.InputText(key='email')
                ],
95          [sg.Text('Assunto do E-mail:'), sg.InputText(key='assunto
                ')],
96          [sg.Button('Enviar')],
97      ]
98      window = sg.Window('Envio de E-mail', layout, finalize=True,
            icon='arapuka.ico')
99
100     return window
101
102 ##########################################################################
103 #                                                    MEMORIA
104 #
        ##########################################################################
105
106 # Ler n primeiros Registros
107 def rd_n():
108     choices = list(range(1, 801))
109     layout = [
110         [sg.Text('Escolha a quantidade de Registros:')],
111         [sg.Combo(choices, default_value=1, size=(10, 1), key='
                combo')],
112         [sg.Button('Enviar')],
113     ]
114
115     window = sg.Window('RD n', layout, finalize=True, icon='
            arapuka.ico')
116
117     return window
```

```
118
119 # Ler n ate m Registros
120 def rd_n_m():
121     choices = list(range(1, 801))
122     layout = [
123         [sg.Text('Escolha o intervalo de Registros:')],
124         [sg.Combo(choices, default_value=1, size=(10, 1), key='
            combo1'), sg.Combo(choices, default_value=1, size=(10,
            1), key='combo2')],
125         [sg.Button('Enviar')],
126     ]
127
128     window = sg.Window('RD n_m', layout, finalize=True, icon='
        arapuka.ico')
129
130     return window
131
132 #checa se o primeiro valor é menor que o segundo para a ação Rd
    n_m
133 def is_valid_RD_n_m(number1, number2):
134     if(number2 > number1):
135         return True
136     else:
137         return False
138
139 #Layout para botões dos leds
140 def leds_layout():
141     layout = [
142         [sg.Text('Led Verde:'), sg.Text('  '), sg.Button('ON',
            button_color=('white', 'green'), key='led_verde_on'),
            sg.Button('OFF', button_color=('white', 'red'), key='
            led_verde_off')],
143         [sg.Text('Led Vermelho:'), sg.Button('ON', button_color=(
            'white', 'green'), key='led_vermelho_on'), sg.Button('
            OFF', button_color=('white', 'red'), key='
            led_vermelho_off')],
144     ]
145     window = sg.Window('Leds', layout, finalize=True, icon='
        arapuka.ico')
146
```

```
147     return window
148 ##########################################################################
149 #                                                          MAPA
150 #
    ##########################################################################
151
152 #retira a latitude e longitude da mensagem completa
153 def extrair_latitude_longitude(data):
154     try:
155         partes = data.split()
156         # Significa que mudou de estado
157         if(partes[0] == "DMT" or partes[0] == "VIG" or partes[0]
               == "ALT1" or partes[0] == "ALT2" or partes[0] == "SPT")
               :
158             latitude_str, longitude_str = partes[4].split(','),
                   partes[5].split(',')
159             latitude = float(latitude_str[0])
160             longitude = float(longitude_str[0])
161             return latitude, longitude
162         # Não mudou de estado
163         else:
164             latitude_str, longitude_str = partes[2].split(','),
                   partes[3].split(',')
165             latitude = float(latitude_str[0])
166             longitude = float(longitude_str[0])
167             return latitude, longitude
168     except:
169         return False, False
170
171 def exibir_mapa(latitude, longitude):
172     try:
173         m = folium.Map(location=[latitude, longitude], zoom_start
               =16)
174         folium.Marker([latitude, longitude]).add_to(m)
175
176         map_filename = 'map.html'
177         m.save(map_filename)
178
179         image_filename = 'map_screenshot.png'
```

```
180             capture_screenshot(map_filename, image_filename)
181
182             os.remove(map_filename)
183
184             update_queue.put(('show_map', image_filename))
185         except Exception as e:
186             update_queue.put(('error', str(e)))
187
188 def capture_screenshot(url, output_filename):
189     options = webdriver.ChromeOptions()
190     options.add_argument('--headless')
191     driver = webdriver.Chrome(options=options)
192
193     driver.get('file://' + os.path.abspath(url))
194     driver.save_screenshot(output_filename)
195
196     driver.quit()
197
198 def criar_janela_mapa(image_filename):
199     layout_mapa = [[sg.Image(filename=image_filename)]]
200     map_window = sg.Window('Mapa', layout_mapa, finalize=True,
        icon='arapuka.ico')
201
202     while True:
203         event, values = map_window.read()
204
205         if event == sg.WINDOW_CLOSED:
206             map_window.close()
207             os.remove(image_filename)
208             break
```