UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

Juan Henrique dos Santos

Acessibilidade de leitura para cegos com o projeto Celta numa Raspberry PI

São Carlos - SP

2023

Juan Henrique dos Santos

Acessibilidade de leitura para cegos com o projeto Celta numa Raspberry PI

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação da UNIVERSIDADE FEDERAL DE SÃO CARLOS, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientação Profa. Dra. Marilde Terezinha Prado Santos

São Carlos - SP

2023

Dedico este trabalho à minha querida família e aos meus amigos. Sem o apoio de vocês, esta jornada acadêmica teria sido muito mais desafiadora. Agradeço por estarem sempre presentes, compartilhando alegrias e aliviando as dificuldades. Que este documento possa, de alguma forma, representar a minha gratidão que sinto por cada um de vocês.

Agradecimentos

Agradeço profundamente aos meus pais, Nilson e Magda, pelo apoio ao longo de todas as etapas da minha vida. Sem a orientação e o encorajamento de vocês, eu não teria alcançado o ponto em que estou hoje.

Quero expressar minha gratidão especial ao meu tio Anderson, que se tornou um verdadeiro modelo de que a educação constrói caminhos e que a vitória só é possível com muita determinação. Também desejo agradecer ao meu tio Luiz, por me encaminhar na direção da computação e por me ensinar, desde cedo, a importância do foco e da perseverança. Sem a sua ajuda e seus contatos, eu não teria conseguido manter-me em São Carlos. Vocês dois foram meus mentores mais influentes e desempenharam um papel fundamental na minha jornada acadêmica.

Agradeço a todos os meus amigos, mas em especial ao Alexandre, Bruno, Amanda, Fernanda e Vitor. A vida nos presenteia com muitos desafios, altos e baixos, mas ter amigos como vocês ao meu lado torna tudo mais fácil e significativo.

A minha namorada, Luri, agradeço pela paciência, companhia, incentivo e parceria para concluir essa jornada. Obrigado por ser a minha companheira, por enfrentar os desafios ao meu lado e por celebrar as alegrias comigo. Você é a razão pela qual meu coração sorri todos os dias.

Também desejo expressar minha sincera gratidão à Professora Dra. Marilde Terezinha Prado Santos, que me encaminhou para um projeto verdadeiramente extraordinário e, em seguida, guiou-me ao longo desse percurso, mostrando que o processo de aprendizado pode se tornar uma jornada leve e divertida.

Expresso também minha profunda gratidão aos Professores Glauber e Emerson, por terem aceitado participar da banca e por suas valiosas contribuições que enriqueceram ainda mais este trabalho. Quero estender meus agradecimentos especiais ao Professor Glauber, que não apenas desempenhou um papel crucial como membro da banca, mas também foi meu coordenador no projeto Celta, permitindo-me compreender a verdadeira importância de projetos de acessibilidade.

"Lutar pelos direitos dos deficientes é uma forma de superar as nossas próprias deficiências."

(John F. Kennedy)

Resumo

A leitura em braile desempenha um papel fundamental na inclusão e no acesso à informação para pessoas cegas ou com baixa visão. Ela é uma forma de alfabetização e comunicação capaz de incluir pessoas para que possam participar plenamente da sociedade, permitindo que tenham acesso ao conhecimento e a cultura. O Projeto Celta, elaborado pela equipe da SEaD-UFSCar com apoio financeiro da CAPES, teve como objetivo o desenvolvimento de um *hardware* e *software* de baixo custo para a leitura braile através de uma célula tátil. Ao final do projeto, foram elaborados manuais com instruções para a replicação e na elaboração de material de divulgação. O presente trabalho tem como objetivo dar continuidade ao desenvolvimento do projeto Celta, promovendo uma atualização tecnológica desse equipamento. Essa atualização envolve a substituição do microcontrolador Arduíno pelo computador *Raspberry Pi*. Através dessa substituição, torna-se possível a implementação de novos avanços, como a troca da comunicação serial pela comunicação sem fio utilizando *sockets*. No *Raspberry Pi*, a programação das portas é realizada utilizando a linguagem *Python*, enquanto na aplicação *desktop*, um *software* desenvolvido em Java é utilizado para extrair texto de arquivos nos formatos .PDF e .TXT. A utilização das linguagens *Python* e Java torna o projeto mais amigável e acessível aos desenvolvedores, aproveitando a popularidade dessas comunidades. Essas mudanças resultam em uma maior praticidade no uso do equipamento aos usuários.

Palavras-chave: *acessibilidade, braile, projeto Celta, comunicação socket, Java, Python, Raspberry PI.*

Abstract

Braile reading is fundamental to include visually impaired people in the access to information, it's a type of literacy and communication able to include everyone in society allowing then access to knowledge and culture. This tactile reading can be found in diffrents formats such as embossed paper or even in a much more complex way, like a braile display. The Celta project, elaborated by SEaD-UFSCar squad funded by CAPES, had as objective the development of a low cost hardware and software to braile reading. By the end of project were elaborated intructions manuals for replications and advertising material, there was a contextualization and clarification about the use and applicability of Celta. The present work aims to continue the development of the Celta project by promoting a technological update of this equipment. This update involves replacing the Arduino microcontroller with the Raspberry Pi computer. Through this replacement, it becomes possible to implement new advancements, such as replacing serial communication with wireless communication using sockets. In the Raspberry Pi, port programming is carried out using the Python language, while in the desktop application, a software developed in Java is used to extract text from .PDF and .TXT files. The use of Python and Java languages makes the project more user-friendly and accessible to developers, taking advantage of the popularity of these communities. These changes result in greater convenience for equipment users.

Keywords: *accessibility, braile, Celta project, socket programming, Java, Python, Raspberry PI.*

Lista de ilustrações

Lista de tabelas

Sumário

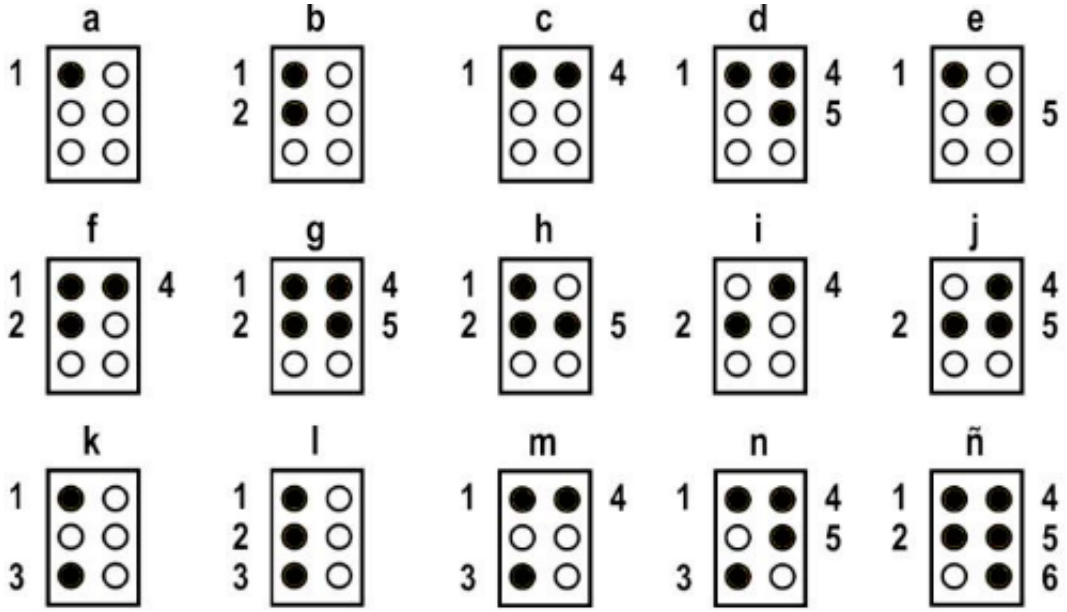
1 Introdução

A acessibilidade é um conceito dinâmico e fundamental em nossa sociedade, que pode estar associado não só ao desenvolvimento tecnológico mas também ao desenvolvimento da sociedade, permitindo que todos possam ter oportunidades iguais e acesso a todos os recursos disponíveis. Este conceito envolve aspectos do espaço físico, assim como também pode envolver o espaço digital (TORRES; MAZZONI; ALVES, 2002).

As tecnologias de informação e comunicação têm criado diversas possibilidades no espaço digital, no entanto, é fundamental garantir o atendimento adequado às diferentes formas de interação das pessoas com a informação, levando em consideração suas preferências e limitações, tanto em relação aos dispositivos utilizados quanto às suas capacidades físicas através da acessibilidade. Assim, a acessibilidade no meio digital refere-se a disponibilizar, de maneira autônoma, todas as informações acessíveis ao usuário, independentemente de suas limitações físicas, garantindo que o conteúdo seja acessível sem qualquer discriminação. Isso é alcançado ao apresentar a informação de maneiras diversas, seja através de redundância simples ou por meio de um sistema automático de transcrição de mídias, com a utilização de ajudas técnicas, como sistemas de leitura de tela, reconhecimento de fala, simuladores de teclado, entre outros, para otimizar a experiência dos usuários que possuam limitações associadas a deficiências. Uma vez que o espaço digital abrange diversos meios de comunicação como televisão digital, computadores e redes telemáticas. É nesse espaço que o direito à informação, de forma acessível, deve ser defendido e promovido (TORRES; MAZZONI; ALVES, 2002).

A acessibilidade desempenha um papel importante na inclusão e no acesso à informação para pessoas cegas ou com baixa visão, sendo o sistema Braille uma ferramenta fundamental nesse processo. Em 1784, foi fundada em Paris a primeira escola para cegos (*Instituiton Nationale des Jeunes Aveugles*) por Valentin Hauvy (1745-1822), reconhecido como o "pai da educação dos cegos". Hauvy desenvolveu uma escrita em relevo para possibilitar a leitura tátil por pessoas cegas, assim como um ensaio sobre a educação desse grupo. Louis Braille (1809-1852), um discípulo de Hauvy, ingressou no instituto de cegos de Paris aos sete anos, quatro anos após perder a visão. Ele se destacou como músico e se tornou professor no mesmo instituto. Braille dedicou-se ao estudo de uma escrita para uso dos cegos, conhecida atualmente como sistema braile. Ele criou 63 combinações com seis pontos em relevo, representando letras, acentuações, pontuações, algarismos, sinais algébricos, contrações estenográficas e sinais musicais. O sistema braile é uma forma de codificação textual projetada para pessoas com deficiência visual acentuada. No Brasil, utiliza-se o sistema braile de 6 pontos, com duas colunas de três pontos, possibilitando os 63 sinais distintos representados na Figura 1 (BRITANNICA DO BRASIL, 1987).

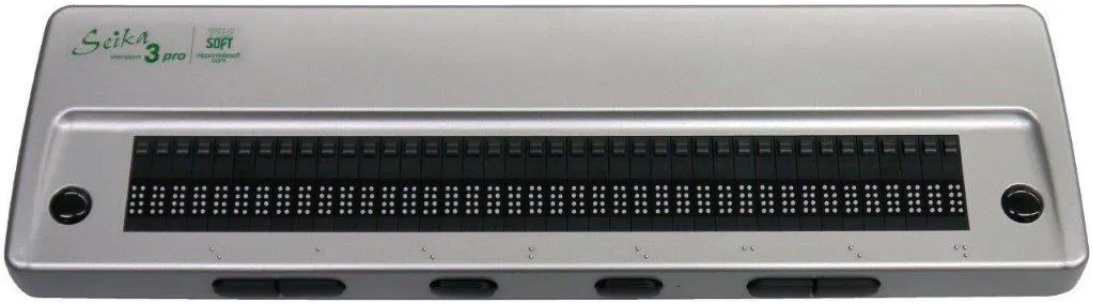
Figura 1 – Representação de letras no sistema braile de 6 pontos.



Fonte: (INSTRUCTOR. . . , 2023).

Para criar a escrita braile em diferentes dispositivos, são utilizadas um conjunto de células táteis em um *display* braile, conforme ilustrado na Figura 2. Essas células podem variar em preço dependendo do tipo de marca, qualidade e características específicas do dispositivo. Podem ser desde pequenas células táteis individuais usadas em papel braile até *displays* eletrônicos em braile mais complexos, empregados em dispositivos como *smartphones*, *tablets* e computadores. Para a utilização do sistema braile, é necessário um meio físico, como papel com pontos em alto relevo, que incluem regletes, máquinas de escrever braile, impressoras braile e os mencionados *displays* braile. Estes últimos são recomendados para o módulo de ensino EaD (educação a distância), pois permitem a leitura imediata do conteúdo digital, embora apresentem um alto valor comercial que dificultam suas aquisições (SANTIAGO et al., 2020).

Figura 2 – Representação de um *display* braile com 40 células táteis de 8 pinos.



Fonte: (LOJA CIVIAM, 2023).

Portanto, emerge a necessidade de desenvolver dispositivos compatíveis ao sistema

braile que sejam economicamente acessíveis. Desta forma, com o objetivo de promover a inclusão de pessoas com deficiência visual, foi proposto o desenvolvimento de uma alternativa de difusão de um *display* braile customizado e de baixo custo, tornando a acessibilidade mais viável para um número maior de pessoas. A equipe da SEaD-UFSCar desenvolveu o Projeto Celta, também conhecido como Sistema de célula tátil para leitura braile, com o apoio financeiro da CAPES através do edital nº 3/2018 – Ferramentas de acessibilidade (processo: 88887.185276/2018-00). O principal objetivo do projeto foi criar um *hardware* e *software* para a leitura braile e disponibilizá-lo livremente, juntamente com toda a documentação necessária para sua reprodução. Além disso, o projeto incluiu a elaboração de manuais (*site*, *e-book*, vídeos etc.) com instruções para a replicação e a criação de materiais de divulgação, contextualização e esclarecimento sobre o uso e aplicação do Celta (SECRETARIA GERAL DE EDUCAÇÃO A DISTÂNCIA, SEAD, 2018).

É estimado que cada unidade possa ser elaborada com menos de R$300,00, considerando os materiais utilizados. Quanto à utilização, é vislumbrado que o Celta possa facilitar atividades de letramento em braile, permitindo a disseminação e proporcionando treinamento para um maior número de pessoas nessa área de leitura. Acredita-se também que, por ser um projeto aberto, possa contribuir para o estímulo do desenvolvimento tecnológico em acessibilidade no Brasil (SANTIAGO et al., 2020).

O Celta é um sistema computacional que atua como um periférico conectado ao computador e composto por *hardware* (máquina) e *software* (programa). É constituído por um único *display* braile e dois botões de controle. Seu funcionamento consiste em conectar o dispositivo ao computador através de um cabo USB (*Universal Serial Bus*) e, com o *software* de interface aberto, permite ao usuário selecionar e copiar um texto desejado para a área de transferência do computador (utilizando os comandos de copiar, Ctrl+c, e colar, Ctrl+v). Posteriormente, o texto copiado é exibido em caracteres braile, sendo possível ler um caractere por vez, ao tocar com os dedos na superfície do *hardware*. A progressão ou retrocesso dos caracteres é controlada pelo usuário através dos dois botões disponíveis (SANTIAGO et al., 2020).

1.1 Objetivos

O principal objetivo deste projeto é aprimorar a funcionalidade do Celta, buscando estabelecer uma comunicação sem fio entre o *hardware* e o *software*, utilizando o *wi-fi* como meio de transmissão. Atualmente o Celta se comunica por meio da comunicação serial através de um microcontrolador Arduíno, conforme ilustrado na Figura 3. Com a implementação do novo protocolo *socket*, é possível uma conexão mais ágil e eficiente. Para viabilizar essa comunicação, é necessário efetuar modificações no *hardware*, optando pela

adoção do *Raspberry PI* conforme ilustrado na Figura 4, com foco no modelo *Raspberry PI 3 Model B v1.2*. Essa adaptação permite uma integração perfeita e livre de fios, conferindo ao Celta uma maior flexibilidade e compatibilidade com as tecnologias de comunicação mais avançadas disponíveis até o momento.

Para alcançar os objetivos propostos, foi adotada a linguagem *Python* no *hardware* do Celta, considerando sua ampla utilização e facilidade de aprendizagem o que torna a replicação do projeto mais acessível. Também foi levada em consideração sua popularidade e vasta comunidade de desenvolvedores, o que facilita o acesso a recursos, suporte e aprimoramentos futuros. Sua utilização permite uma boa eficiência no controle e gerenciamento dos componentes do dispositivo, proporcionando uma integração mais fluida entre o *hardware* e o *software*. O controle dos pinos do equipamento foi realizado através das portas GPIO (*General Purpose Input/Output*), assim como era realizado na versão anterior do Arduíno.

Com a combinação da linguagem *Python* e a utilização das portas GPIO, o Celta se beneficia de uma programação mais intuitiva e flexível, facilitando seu desenvolvimento e manutenção ao longo do tempo. Essa abordagem torna o projeto mais amigável aos desenvolvedores e permite que novas funcionalidades e melhorias sejam incorporadas de maneira eficiente, contribuindo para a evolução contínua do Celta como um produto tecnologicamente avançado e adaptável às necessidades dos usuários.

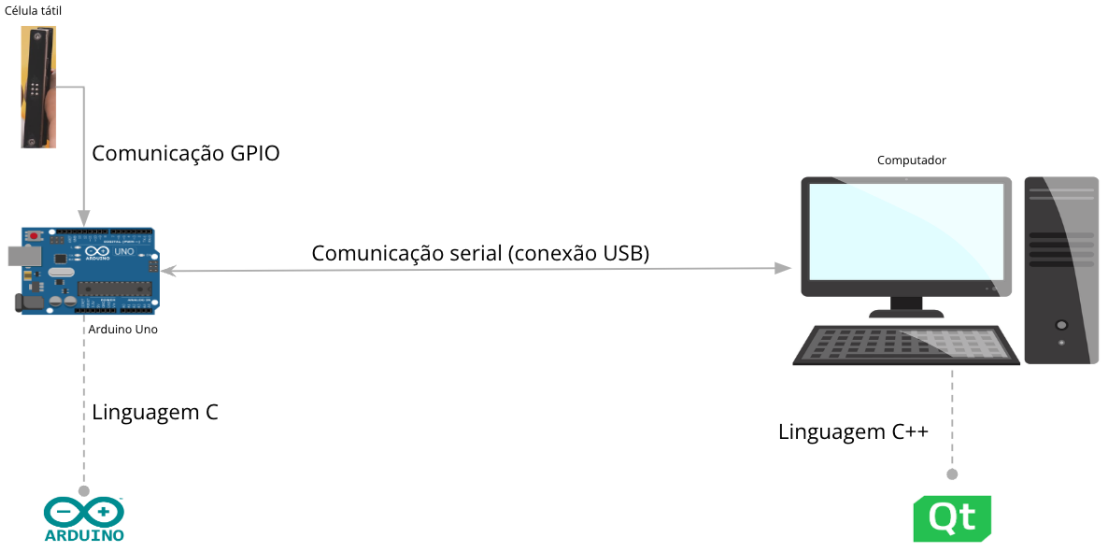
Além disso, para garantir que o *software* funcione de maneira adequada em diversos sistemas operacionais, foi efetuada uma recriação completa do programa utilizando a linguagem *Java*. Essa abordagem oferece a vantagem de eliminar restrições de compatibilidade, possibilitando que o *hardware* se comunique de forma fluida com o *software*, independentemente do sistema operacional utilizado pelo usuário.

Outra característica importante do projeto é a integração de bibliotecas disponíveis na internet, projetadas especialmente para a leitura de arquivos PDF's *(Portable Document Format*). Com a incorporação dessas bibliotecas adicionais, o Celta ganha uma funcionalidade valiosa ao permitir a leitura de documentos nesse formato específico. Isso torna o dispositivo ainda mais versátil e amplia suas possibilidades de uso para os usuários, que poderão interagir com arquivos PDFs de forma prática e eficiente. Com essas melhorias e atualizações, o Celta se posiciona como uma ferramenta mais poderosa e adaptável, pronta para atender às necessidades e demandas dos usuários em diferentes contextos e ambientes operacionais.

Na Tabela 1 são realizadas comparações entre aspectos referentes à versão atual do Celta, em relação à nova versão. Apesar da primeira versão do Celta incluir *Linux* e *Windows* como sistema operacional, existem algumas dificuldades quanto à geração do executável. Com a nova versão do *software*, esta dificuldade foi superada. Através do *YouTube* é possível assistir a explicação detalhada sobre a comunicação do *hardware* com o

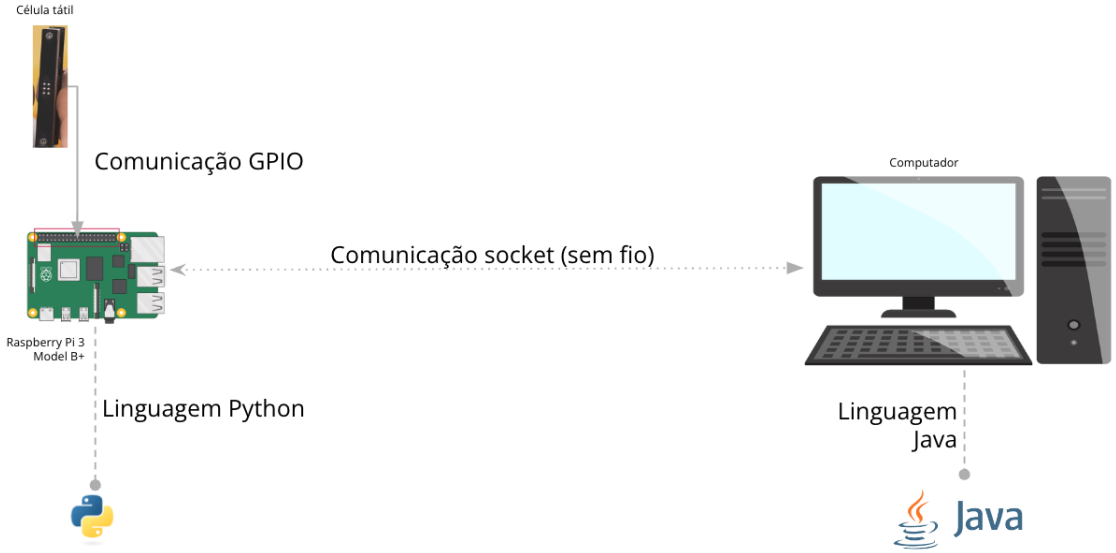
software, disponibilizada pelo autor[1].

Figura 3 – Panorama geral do projeto atual com o uso do microcontrolador Arduíno. A comunicação entre *hardware* e *software* acontece via cabo através da comunicação serial com USB. A linguagem utilizada no Arduíno é C, enquanto que no *software* presente no *desktop* é C++.



Fonte: Do Autor.

Figura 4 – Panorama geral do novo projeto com o uso do computador *Raspberry Pi*. A comunicação entre *hardware* e *software* acontece via wi-fi através da comunicação socket. A linguagem utilizada no Raspberry Pi é Python, enquanto que no *software* presente no *desktop* é Java.



Fonte: Do Autor.

[1] <https://youtu.be/60GOgeH97_s>

Tabela 1 – Comparação entre Celta atual e Celta aprimorado.

Características	**Celta Atual**	**Celta Aprimorado**
Hardware	*Arduíno Uno*	*Raspberry PI 3 Model B v1.2*
Comunicação *hardware* x *software*	Comunicação serial do *Arduino* através do uso do cabo USB	Conexão sem fio via *Wi-Fi*, utilizando protocolo *Socket*
Linguagem de Programação no embarcado	Linguagem C	*Python* 3.2
Linguagem de Programação no *software*	Linguagem C++	Java
Leitura de textos	Obtenção dos textos através do Ctr+V disponibilizados no *clipboard* do usuário	Obtenção dos textos através do Ctr+V disponibilizados no *clipboard* do usuário. Caso o usuário possua o caminho de um arquivo *.TXT ou *.PDF, o *software* irá extrair textos desses arquivos para leitura do *hardware*
Sistema operacional	*Linux* e *Windows*	TODOS os Sistemas Operacionais suportados pelo *Java*

Fonte: Do Autor.

1.2 Organização do Trabalho

Este trabalho foi estruturado em seis capítulos, cada um com um propósito específico. No Capítulo 1, é apresentada a proposta desta pesquisa, delineando de maneira clara os objetivos que norteiam todo o estudo. Esse capítulo é fundamental para estabelecer as bases sobre as quais o trabalho se desenvolveu.

Já o Capítulo 2 abarca a fundamentação teórica que serve como alicerces essenciais para a condução desta pesquisa. Aqui, serão abordados os conceitos que sustentam a comunicação por meio de *sockets* entre *hardware* e *software* do Celta. A compreensão destes conceitos é importante para a exploração e execução deste projeto. Além disso, é traçada uma introdução sobre o uso dos computadores *Raspberry Pi*.

No Capítulo 3 tor uma análise mais profunda do *hardware* do Celta é conduzida. Esse capítulo oferece um entendimento mais aprofundado para replicar o novo *hardware* do Celta, desenvolvido na placa *Raspberry PI 3 B v1.2*. Além disso, é explorado o código fonte do embarcado, escrito em *Python*, proporcionando uma visão abrangente dessa parte do projeto.

Prosseguindo, no Capítulo 4 é detalhado o *software* do Celta. Neste ponto, uma explicação abrangente da aplicação *Java* é oferecida, fornecendo uma visão completa da abordagem adotada na parte de *software* do projeto.

O Capítulo 5 revela os resultados deste trabalho, oferecendo uma síntese das descobertas alcançadas ao longo da pesquisa. Por fim, no Capítulo 6 há a conclusão do trabalho, onde são apresentados possíveis tópicos de pesquisa que poderiam ser explorados no futuro, visando uma melhoria contínua do projeto Celta.

2 Fundamentação Teórica

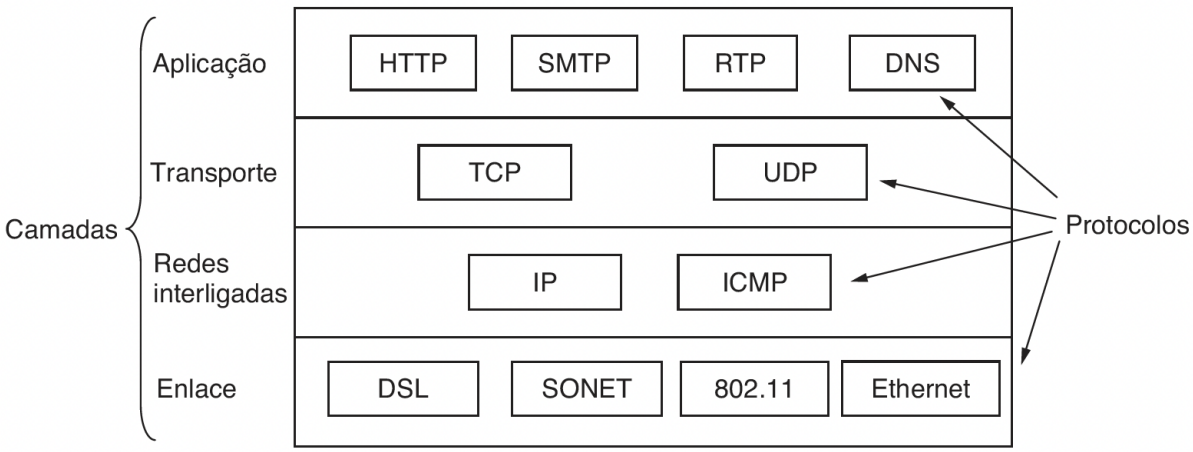
Este capítulo aborda os conceitos fundamentais que embasam este trabalho. São explorados os conceitos teóricos relacionados ao modelo TCP/IP (seção 2.1); arquitetura cliente-servidor (subseção 2.1.2) e a comunicação *socket* (subseção 2.1.3), utilizada para estabelecer a comunicação entre o *hardware* e o *software* do projeto Celta, permitindo a ativação/desativação dos pinos. Além disso, é apresentada uma introdução ao computador *Raspberry Pi* (seção 2.2). A compreensão desses temas serviu como base para a implementação do projeto.

2.1 Modelo TCP/IP

2.1.1 Camadas TCP/IP

A arquitetura TCP/IP opera por meio da divisão funcional em camadas distintas. Esse princípio proporciona uma estrutura organizada que facilita a troca de informações e recursos. As camadas estão representadas na Figura 5 retiradas do livro de (TANENBAUM; WETHERALL, 2011).

Figura 5 – Camadas do TCP/IP.



Fonte: (TANENBAUM; WETHERALL, 2011)

2.1.1.1 Camada de aplicação

O TCP/IP reúne protocolos que entregam uma gama diversificada de serviços de comunicação tanto para o sistema quanto para o usuário. Ele engloba não somente

os aspectos das camadas de apresentação, sessão e aplicação conforme estabelecido no Modelo OSI, mas também incorpora os protocolos essenciais para os Serviços Básicos DNS e DHCP, assim como protocolos voltados aos serviços ao usuário, tais como Telnet, FTP (*File Transfer Protocol*), HTTP (*Hypertext Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*) e muitos outros (JÚNIOR, 2018).

2.1.1.2 Camada de transporte

Encarregada do transporte direto dos dados de uma ponta a outra, a camada de transporte no modelo TCP/IP opera com foco no trajeto completo, sem se preocupar com detalhes intermediários como endereços e rotas. Dentre suas atribuições tem-se a garantia da qualidade do serviço, com atenção à confiabilidade da entrega, gerenciamento eficiente do fluxo de pacotes e a habilidade de identificar e corrigir eventuais erros que possam surgir durante a transmissão.

Dentre os principais protocolos nesta camada tem-se o UDP (*User Datagram Protocol*), que se destaca por sua eficiência e agilidade na troca de informações, e o TCP (*Transmission Control Protocol*), que se sobressai na garantia da integridade e sequência correta dos dados durante a transmissão (JÚNIOR, 2018).

2.1.1.3 Camada internet (camada de rede)

Encarregada do gerenciamento das operações de movimentação (comutação) e roteamento dos pacotes pela rede, a camada de rede assume a responsabilidade fundamental de garantir que os pacotes originados de qualquer sub-rede sejam direcionados até seu destino final, superando os obstáculos do trajeto e das redes percorridas. Tais tarefas são efetivadas através da atribuição de um identificador único, conhecido como endereço IP.

A finalidade central dessa camada é permitir a conectividade de ponto a ponto, independentemente do caminho e das diversas redes que tomem para chegar lá. Isso é alcançado graças ao funcionamento do principal protocolo desta camada, o IP (*Internet Protocol*). Além disso, outros protocolos são empregados, incluindo o ICMP (*Internet Control Message Protocol*) para propósitos de comunicação de diagnóstico e *feedback*, e o IGMP (*Internet Group Management Protocol*) para otimizar a gestão de grupos em redes *multicasting* (JÚNIOR, 2018).

2.1.1.4 Camada de enlace

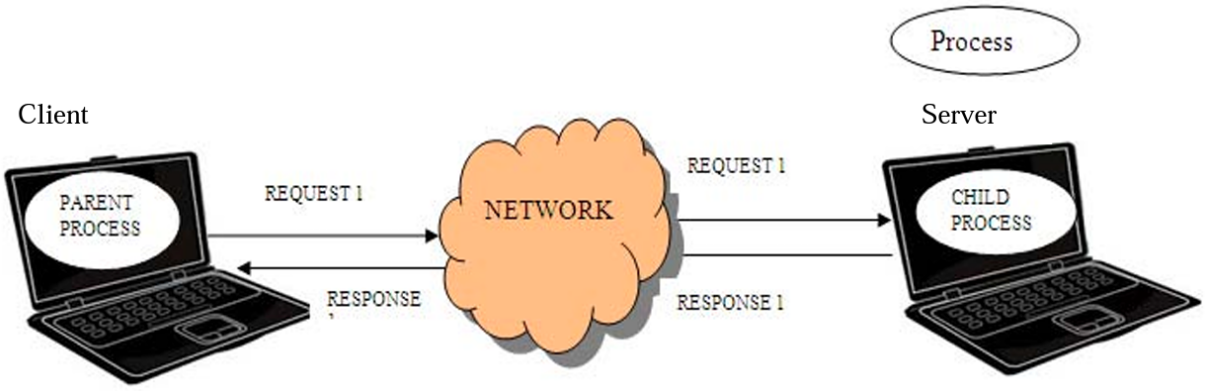
Segundo (JÚNIOR, 2018), esse nível engloba tanto o *driver* de dispositivo no sistema operacional quanto a placa de rede correspondente, bem como outros componentes de *hardware* necessários para a conexão física com a rede (como V24, V35, RS422 e outras tecnologias).

Essa camada trata de todos os elementos cruciais para estabelecer de fato um elo físico entre o ponto de partida e o destino desejado de um pacote de dados. Ela engloba uma série de detalhes que incluem as tecnologias usadas em redes LAN (*Local Area Network*) e redes WAN (*Wide Area Network*). Dessa forma, ele é responsável por garantir que o fluxo de dados seja adequadamente adaptado para a transmissão no meio físico escolhido. Dentre os exemplos de protocolos que operam nessa camada estão *X25*, *Frame Relay*, ATM (*Asynchronous Transfer Mode*), PPP (*Point-to-Point Protocol*), *Ethernet*, *Token Ring*, ARP (*Address Resolution Protocol*) e RARP (*Reverse Address Resolution Protocol*).

2.1.2 Cliente servidor

Segundo a (KALITA, 2014), uma rede é composta por computadores, cada um atuando como cliente ou servidor. Um servidor é um programa que disponibiliza um serviço específico, enquanto um cliente é um programa que solicita um serviço. Servidores são computadores ou processos robustos designados para gerenciar unidades de disco (servidores de arquivos), impressoras (servidores de impressão) ou o tráfego da rede (serviços de rede). Por outro lado, clientes são computadores ou estações de trabalho nos quais os usuários executam aplicativos. Clientes dependem dos servidores para acessar recursos como arquivos, dispositivos e até mesmo capacidade de processamento. Quando esses programas são executados, um processo de cliente e um processo de servidor são criados simultaneamente, e esses dois processos interagem lendo e escrevendo em soquetes, conforme ilustrado na Figura 6.

Figura 6 – Comunicação entre cliente e servidor.



Fonte: (KALITA, 2014).

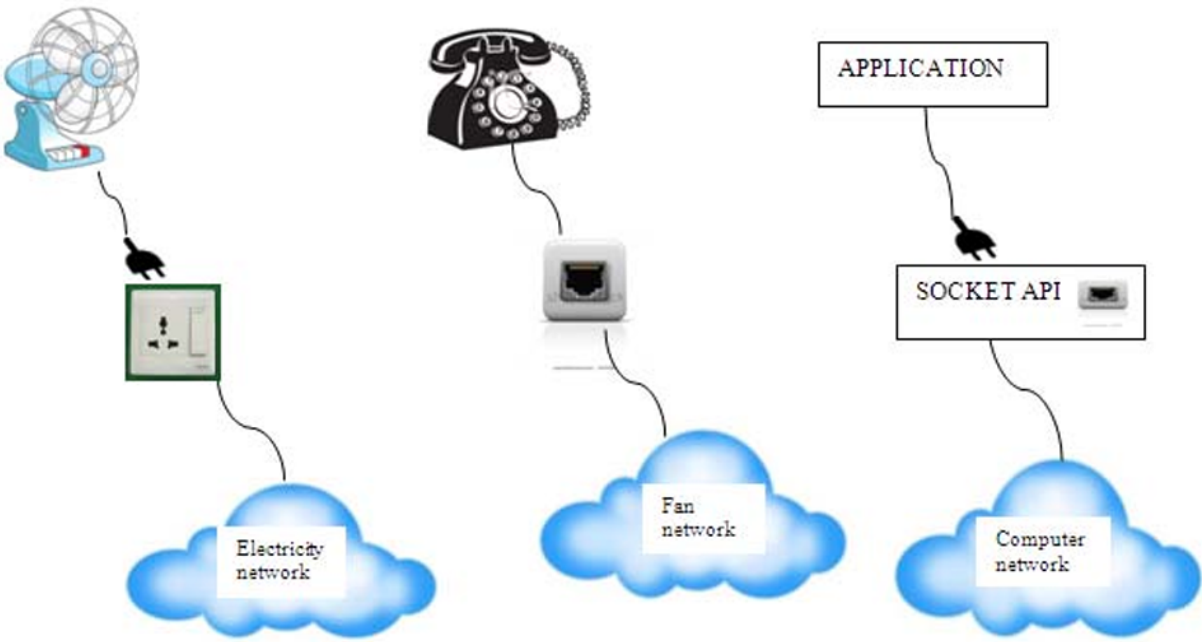
Esses *sockets* são as interfaces de programação fornecidas pelos protocolos TCP e UDP, respectivamente, para comunicação contínua e comunicação de datagrama. Tais protocolos fazem parte da camada de transporte no conjunto TCP/IP. Ao criar um aplicativo de rede, a principal tarefa do desenvolvedor é escrever o código tanto para os programas cliente quanto para os programas servidor. No contexto aqui abordado,

trata-se de um aplicativo cliente/servidor proprietário. Um único desenvolvedor ou equipe de desenvolvimento é responsável pela criação de ambos os programas cliente e servidor, destacando-se que ele tem total controle sobre o que é incorporado no código. No entanto, como o código não implementa um protocolo de domínio público, outros desenvolvedores independentes não podem desenvolver código que seja interoperável com o aplicativo. Quando se desenvolve um aplicativo proprietário, é crucial evitar a utilização de números de porta amplamente conhecidos definidos nos RFCs (*Request for Comments*).

2.1.3 *Socket*

O termo *socket* foi inspirado no uso de uma tomada elétrica/telefone, onde as conexões atuam como interfaces que se encaixam uns nos outros por meio de uma rede, conforme exemplificado na Figura 7.

Figura 7 – Exemplos de *socket*.



Fonte: (KALITA, 2014).

De acordo com (KALITA, 2014), os *sockets* podem ser definidos de diversas maneiras, com múltiplas abordagens. Um *socket* de rede refere-se a um ponto terminal em um fluxo de comunicação entre processos através de uma rede de computadores, formado por um endereço IP e número de porta. Também podem ser interpretados como os pontos finais de conexões entre dois computadores, identificados por um endereço IP e um número de porta correspondente. Outra definição sugere que sejam uma abstração de *software* representando os "terminais" de conexões entre máquinas. Além disso, podem ser concebidos como abstrações oferecidas a programadores de aplicativos para envio e

recebimento de dados entre processos. Os *sockets* são, na prática, interfaces entre aplicativos e redes, viabilizando comunicações eficazes e transferências de dados.

Figura 8 – Brincadeira dos copos ligados com fio de barbante.

Fonte: (FREEPIK, 2023).

Uma analogia no uso do *socket*, exemplificada por (KALITA, 2014) e ilustrado na Figura 8, é a brincadeira do telefone construído com dois copos unidos por um fio de barbante. Para essa brincadeira, são necessárias duas pessoas: seu amigo pegaria um dos telefones e caminharia até o outro lado da sala, falando no telefone (copo). Você aproximaria seu ouvido no outro copo e, assim, seria capaz de ouvir seu amigo. Os copos neste exemplo representam os *sockets*. Portanto, é possível se comunicar com seu amigo falando no copo (obtendo um fluxo de saída do *socket* e enviando *bytes*) e aproximando seu ouvido do copo para ouvir seu amigo falar (obtendo um fluxo de entrada do soquete e lendo dados dele). Um fluxo é uma sequência contínua de caracteres que flui para dentro ou para fora de um processo.

2.1.3.1 Operações com *socket*

Um soquete realiza quatro operações fundamentais:

1. Conectar-se a uma máquina remota;
2. Enviar dados;
3. Receber dados;

4. Fechar a conexão

Um *socket* não pode estar conectado a mais de um *host* ao mesmo tempo. Entretanto, um *socket* pode se comunicar em duas direções com o *host* ao qual está conectado, o que significa que ele pode enviar e receber dados (KALITA, 2014).

2.1.3.2 Porta

Uma porta na rede de computadores é um ponto de comunicação dentro de um computador. Uma porta está associada a um endereço IP do *host*, bem como ao tipo de protocolo utilizado para essa comunicação. O propósito das portas é identificar de maneira única diferentes aplicativos ou processos em execução no mesmo computador. Os protocolos que usam as portas estão na camada de transporte. Conforme visto na subseção 2.1.1.2, tem-se os protocolos TCP e UDP. Eles usam portas para mapear dados de entrada para um processo específico em execução num computador. Portanto, uma porta é como um nome que diz o que está acontecendo dentro do computador.

2.1.4 Programando *socket* com TCP

Com base nas ilustrações apresentadas (Figuras 9 e 10), foi viabilizada a construção da Tabela 2, que exemplifica a funcionalidade dos métodos nos diagramas:

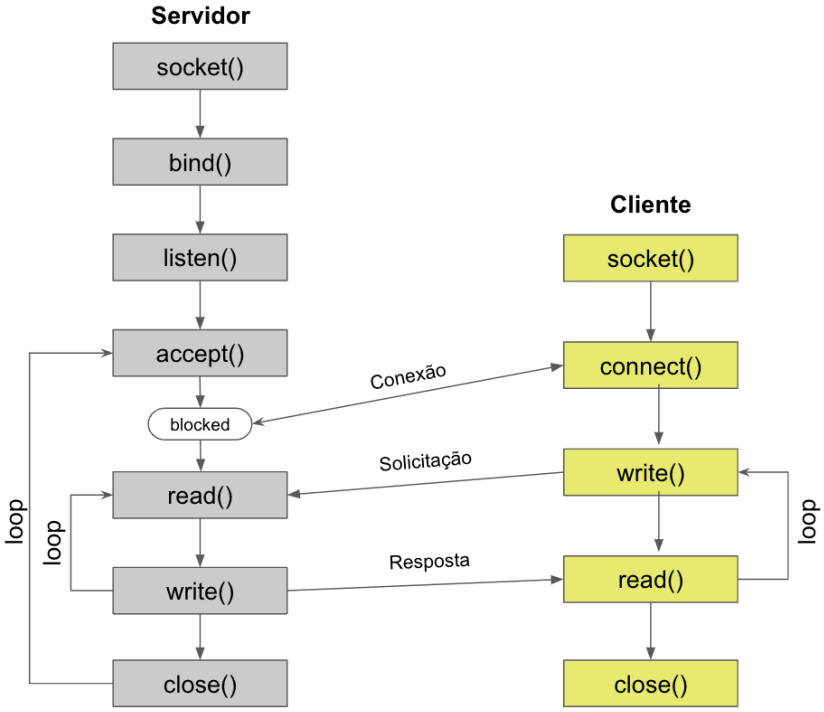
Tabela 2 – Principais Funções da API de *Sockets* no Python e no Java.

Python	Java	Descrição do método
socket	**socket**	Cria um novo descritor para comunicação
connect	**connect**	Iniciar conexão com o servidor
send	**write**	Escreve dados em uma conexão
recv	**read**	Lê dados de uma conexão
close	**close**	Fecha a conexão
bind	**bind**	Atribui um endereço IP e uma porta a um socket
listen	**listen**	Coloca o *socket* em modo passivo, para "escutar"portas
accept	**accept**	Bloqueia o servidor até a chegada de requisição de conexão

Fonte: Do Autor.

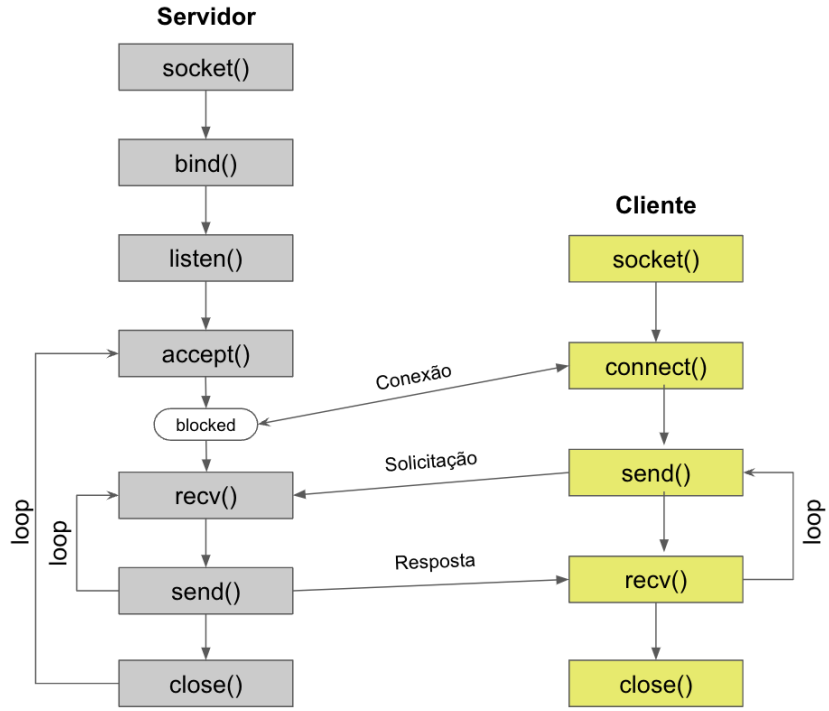
1. **Servidor:** Um servidor atribui um endereço IP e uma porta a um *socket*. Na sequência, o *socket* fica em modo passivo, escutando portas e esperando conexões. Enquanto não houver a requisição de conexão de um cliente, o restante do fluxo ficará bloqueado. Quando finalmente houver uma conexão, o restante do fluxo seguirá com a leitura de possíveis dados que vierem do cliente e com possíveis envios de dados ao cliente. Aqui nesse ponto, é possível que haja um *loop* de leitura/escrita, até que

Figura 9 – Programando *sockets* no *Java*.



Fonte: (ARAKAKI, 2023).

Figura 10 – Programando *sockets* no *Python*.



Fonte: Do Autor.

determinada condição seja satisfeita ou até o cliente estar conectado. Ao final desse fluxo, é encerrada a conexão e/ou o aplicativo entra em modo passivo novamente, no aguardo de novas conexões de clientes;

2. **Cliente:** Em paralelo ao servidor, existem N clientes querendo se conectar com o servidor. Ao se conectar com ele, torna-se possível o envio de dados e a leitura de dados ao servidor. Aqui nesse ponto, é possível que haja um *loop* de leitura/escrita, até que determinada condição seja satisfeita ou que tenhamos um servidor ativo. Ao final desse fluxo, é encerrado a conexão;

2.2 *Raspberry Pi*

O intuito dessa seção é apresentar o *Raspberry Pi*, destacando e mostrando aspectos fundamentais do seu uso em diversas aplicações.

2.2.1 História

O *Raspberry Pi* é uma série de pequenos computadores que possuem o tamanho de um cartão de crédito, desenvolvidos pela *Raspberry Pi Foundation*. A fundação foi criada em 2006 por Eben Upton, Rob Mullins, Jack Lang e Alan Mycroft, com o objetivo de promover o ensino de ciência da computação básica em escolas e países em desenvolvimento. A jornada se iniciou quando Eben Upton se encontrou com seus colegas no laboratório de informática da Universidade de Cambridge. A intenção era debater maneiras de introduzir programação de forma mais acessível para crianças, permitindo-lhes adquirir familiaridade com algoritmos. Essa abordagem se preocupava em prepará-las para um futuro mais fácil na programação quando ingressassem no ambiente universitário (CIRIACO, 2015).

Os primeiros modelos do *Raspberry Pi*, o *Raspberry Pi 1 Model A* e *Raspberry Pi 1 Model B*, foram lançados em fevereiro de 2012. Foram projetados como dispositivos de baixo custo para estimular o interesse em programação e tecnologia entre estudantes e entusiastas. Desde então, várias versões foram lançadas, cada uma com melhorias de desempenho e recursos adicionais. O *Raspberry Pi* ganhou popularidade em diversas áreas, incluindo educação, projetos de DIY (*Do It Yourself*), automação residencial, servidores de baixo consumo de energia e entre outros.

Dentre os objetivos da *Raspberry Pi Foundation* (ABOUT..., 2023), é citado:

- **Educação:** Com o intuito de viabilizar o ensino de computação e a criação com tecnologias digitais em diversas escolas, é proporcionada uma estrutura de currículo, recursos e treinamento de alta qualidade aos professores.

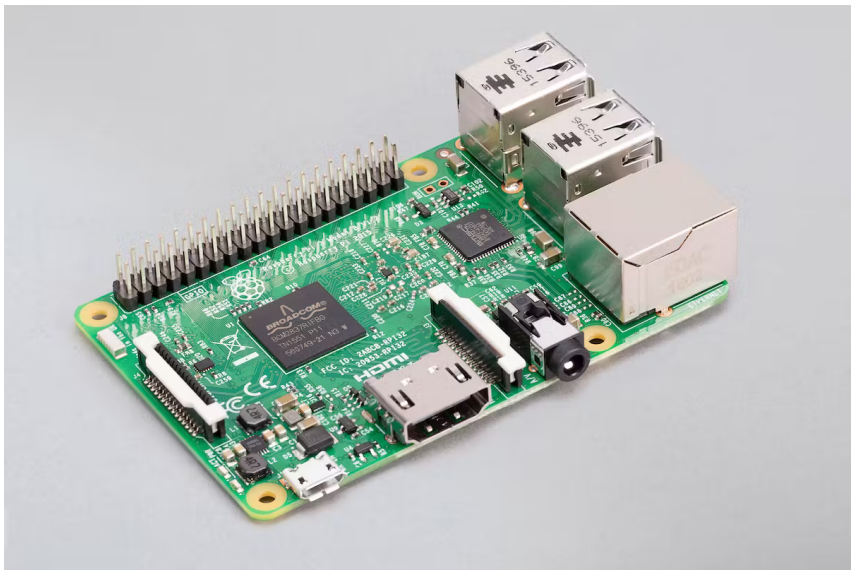
- **Aprendizagem não formal:** visa engajar um grande número de jovens na compreensão da computação e na habilidade de criar com tecnologias digitais, fora do ambiente escolar, através da disponibilização de recursos e aplicativos online, clubes, competições e colaborações com organizações juvenis.

- **Pesquisa:** Aprofundar a compreensão acerca do processo pelo qual os jovens adquirem conhecimento sobre computação e habilidades de criação com tecnologias digitais, de modo a aplicar essa perspicácia para potencializar o impacto das ações e promover progressos no domínio da educação em computação.

2.2.2 Especificações

Este projeto utiliza o modelo *Raspberry Pi 3 Model B V1.2* dessa série de computadores, conforme ilustrado na Figura 11. De acordo com (RASPBERRY PI, 2017), foi criada a Tabela 3, para detalhar a especificação do computador.

Figura 11 – Computador *Raspberry Pi 3 Model B V 1.2* .



Fonte: (RASPBERRY PI, 2017).

2.2.3 GPIO

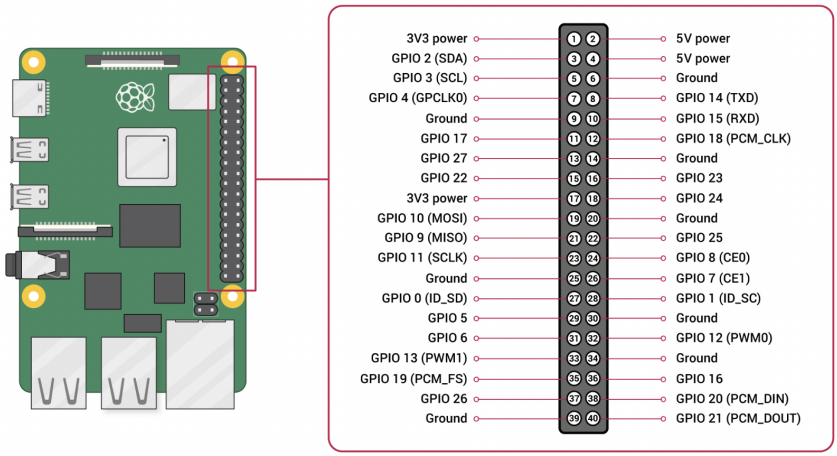
A GPIO (*General Purpose Input/Output*) em uma *Raspberry Pi* refere-se aos pinos de propósito geral, que permitem a interação entre o computador e o mundo exterior. Os pinos podem ser configurados como entradas ou saídas durante o desenvolvimento do *software* e são utilizados para conectar tanto sensores, dispositivos, atuadores quanto outros componentes eletrônicos. Através desses pinos, o computador consegue interagir com o ambiente externo. Neste modelo em questão é disponibilizado um total de quarenta pinos, conforme é ilustrado nas figuras 12 e 13.

Tabela 3 – Especificação *Raspberry Pi 3 Model B v1.2.*

Especificação
Processador Broadcom BCM2837 64bit Quad Core
Clock de 1.2GHz
Memória RAM de 1GB
Wi-fi 802.11n integrado 2.4 Ghz
Bluetooth 4.1 BLE integrado
Conector ethernet
GPIO de 40 pinos
4 portas USB 2.0
Conector de áudio e vídeo
Conetor de vídeo HDMI
Interface para câmera (CSI)
Interface para display (DSI)
Slot para cartão microSD

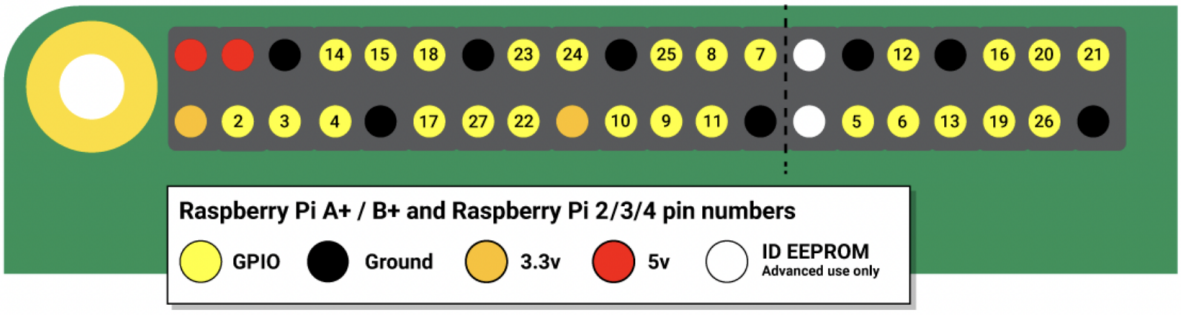
Fonte: (RASPBERRY PI, 2017).

Figura 12 – GPIOs da *Raspberry Pi 3 Model B V 1.2.*



Fonte: (RASPBERRY PI, 2023a).

Figura 13 – Disposição das GPIOs da *Raspberry Pi 3 Model B V 1.2* distribuída por cores.



Fonte: (RASPBERRY PI, 2023a).

2.2.4 Sistema Operacional

O *Raspberry Pi* não possui um disco rígido, mas possui um *slot* para conectar um cartão de memória SD como seu principal meio de armazenamento. Todos os arquivos (incluindo o sistema operacional) são gravados no cartão. O *slot* para a inserção do cartão SD está localizado na parte inferior da placa *Raspberry Pi*.

De acordo com (FROMAGET, 2019), existem várias distribuições Linux que funcionam no *Raspberry Pi*. Em sua publicação, foram elencados alguns sistemas populares que funcionam nesse computador. Neste projeto é utilizado o sistema operacional *Raspberry Pi OS*, sendo a distribuição oficial do dispositivo. Ele tem como base o sistema operacional *Debian* e segue a mesma filosofia, focando em estabilidade e desempenho. Muitos dos pacotes do Debian também estão disponíveis para o *Raspberry Pi*.

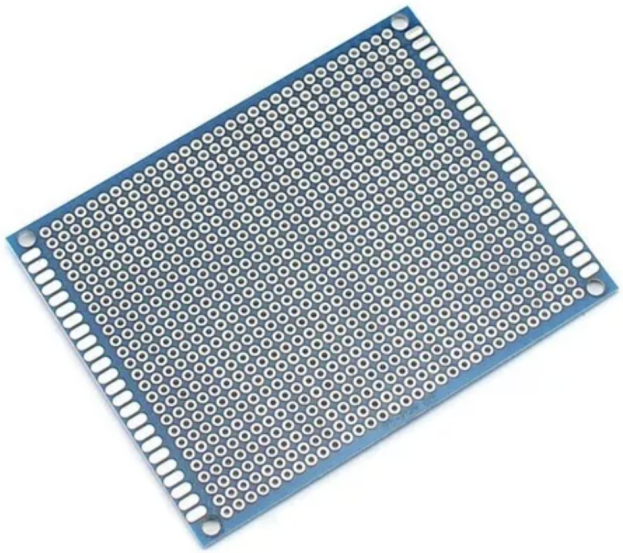
3 Hardware

Neste capítulo é detalhado o processo de desenvolvimento do novo *hardware* do Celta, explicando ajustes realizados no circuito elétrico, a integração do circuito com a *Raspberry Pi* e o *software* do embarcado escrito em *Python* que atua como um servidor *socket*.

3.1 Circuito elétrico

O circuito elétrico do novo Celta foi baseado no manual do (SANTIAGO et al., 2020), passando por pequenas alterações em sua construção. A principal diferença em relação a anterior está na construção do circuito numa placa PCB conforme ilustra a Figura 14.

Figura 14 – Placa de circuito impresso PCB, com ilhas nos dois lados da placa (superior e inferior).



Fonte: (MERCADO LIVRE, 2023a).

3.1.1 Componentes do circuito

Para a construção do novo Celta foram necessários:

- 1x Placa de Circuito impresso PCB

- 3x Circuitos integrados L293D
- 2x *push button*
- fio para conexão das ilhas

3.1.2 Montagem do circuito

Inicialmente um circuito elétrico foi criado na ferramenta *Tinkercad*, um sistema de prototipação *online* (TINKERCAD, 2023). Para essa montagem, tornou-se necessário o estudo do *datasheet* representado na Figura 15. Cada um dos CIs de modelo L293D possuem a capacidade de controlar dois motores através da ponte H, um circuito eletrônico usado para controlar a direção e a velocidade de motores elétricos.

Tabela 4 – Datasheet do CI modelo L293D

Pin Number	*Pin Name*	*Description*
1	*Enable 1,2*	*This pin enables the input pin Input 1(2) and Input 2(7)*
2	*Input 1*	*Directly controls the Output 1 pin. Controlled by digital circuits*
3	*Output 1*	*Connected to one end of Motor 1*
4	*Ground*	*Ground pins are connected to ground of circuit (0V)*
5	*Ground*	*Ground pins are connected to ground of circuit (0V)*
6	*Output 2*	*Connected to another end of Motor 1*
7	*Input 2*	*Directly controls the Output 2 pin. Controlled by digital circuits*
8	*Vcc2 (Vs)*	*Connected to Voltage pin for running motors (4.5V to 36V)*
9	*Enable 3,4*	*This pin enables the input pin Input 3(10) and Input 4(15)*
10	*Input 3*	*Directly controls the Output 3 pin. Controlled by digital circuits*
11	*Output 3*	*Connected to one end of Motor 2*
12	*Ground*	*Ground pins are connected to ground of circuit (0V)*
13	*Ground*	*Ground pins are connected to ground of circuit (0V)*
14	*Output 4*	*Connected to another end of Motor 2*
15	*Input 4*	*Directly controls the Output 4 pin. Controlled by digital circuits*
16	*Vcc2 (Vss)*	*Connected to +5V to enable IC function*

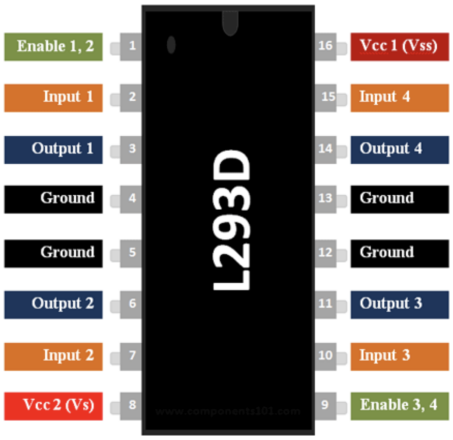
Fonte: (COMPONENTS 101, 2023).

Conforme exibido na Tabela 4, o pino *enable1,2* ativa o motor 1, já o pino *enable3,4* é o responsável por ativar o motor 2. Os pinos nomeados como *input* são responsáveis por controlar o sentido do giro do motor, enquanto que a saída *output* executa a rotação no motor. Desse modo, os pinos *input1* e *input2* controlam o giro no motor 1, enquanto que os pinos *input3* e *input4* controlam o giro no motor 2. As saídas que executam o giro no motor 1 são controladas pelos pinos *input1* e *input2*, enquanto que no motor 2 os pinos *input3* e *input4* são os responsáveis. O resultado da saída ocorrerá de acordo com o pino de ativação e o sentido do respectivo motor.

Os pinos *Vcc* e *ground* são os responsáveis por alimentar o motor. Aqui o *Vcc1* alimenta o motor 1, enquanto que o *Vcc2* alimenta o motor 2. Como a célula tátil possui

seis pinos no total, são necessários três CIs L293D para o controle efetivo conforme a Figura 17.

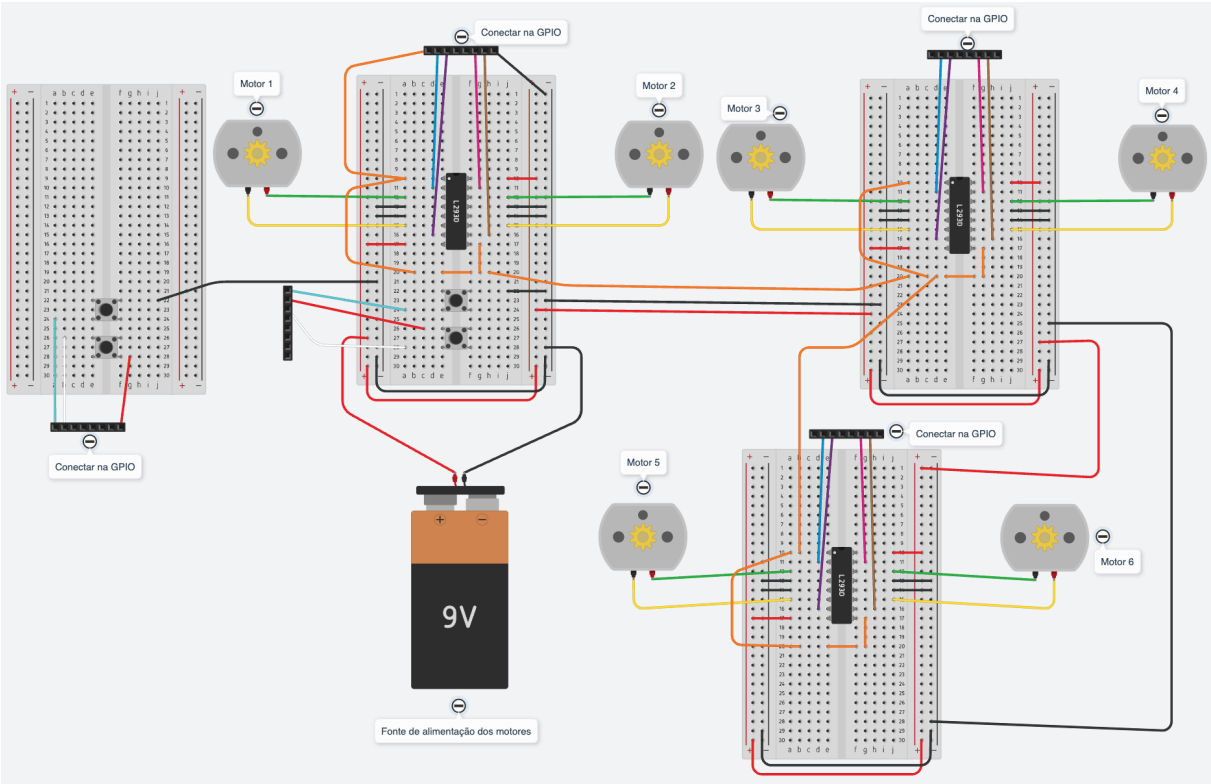
Figura 15 – Datasheet do circuito integrado L293D.



Fonte: (COMPONENTS 101, 2023).

No *Tinkercad* foram conectados todos os elementos necessários para a reprodução do novo Celta. Ao final da montagem, foi obtido o circuito da Figura 16. No total, foram utilizados quatro *protoboards* para melhor distribuição e explicação de todo o projeto.

Figura 16 – Circuito elétrico do novo Celta na ferramenta *Tinkercad*.



Fonte: (TINKERCAD, 2023).

Figura 17 – Célula tátil com seis pinos, que se movimentam de acordo com o sentido dos motores.

Fonte: (SANTIAGO et al., 2020).

Ao final da prototipação, a ferramenta *Tinkercad* permitiu o *download* da vista esquemática, exibidos na Figura 18 e Figura 19.

3.1.3 Circuito final

Após a prototipação do novo Celta no Tinkerpad, houve a soldagem dos componentes na placa PCB da Figura 14. Após a soldagem, foi possível chegar no resultado final conforme a Figura 20 e Figura 21. Na próxima sessão, é abordada a conexão dos pinos com a *Raspberry Pi*.

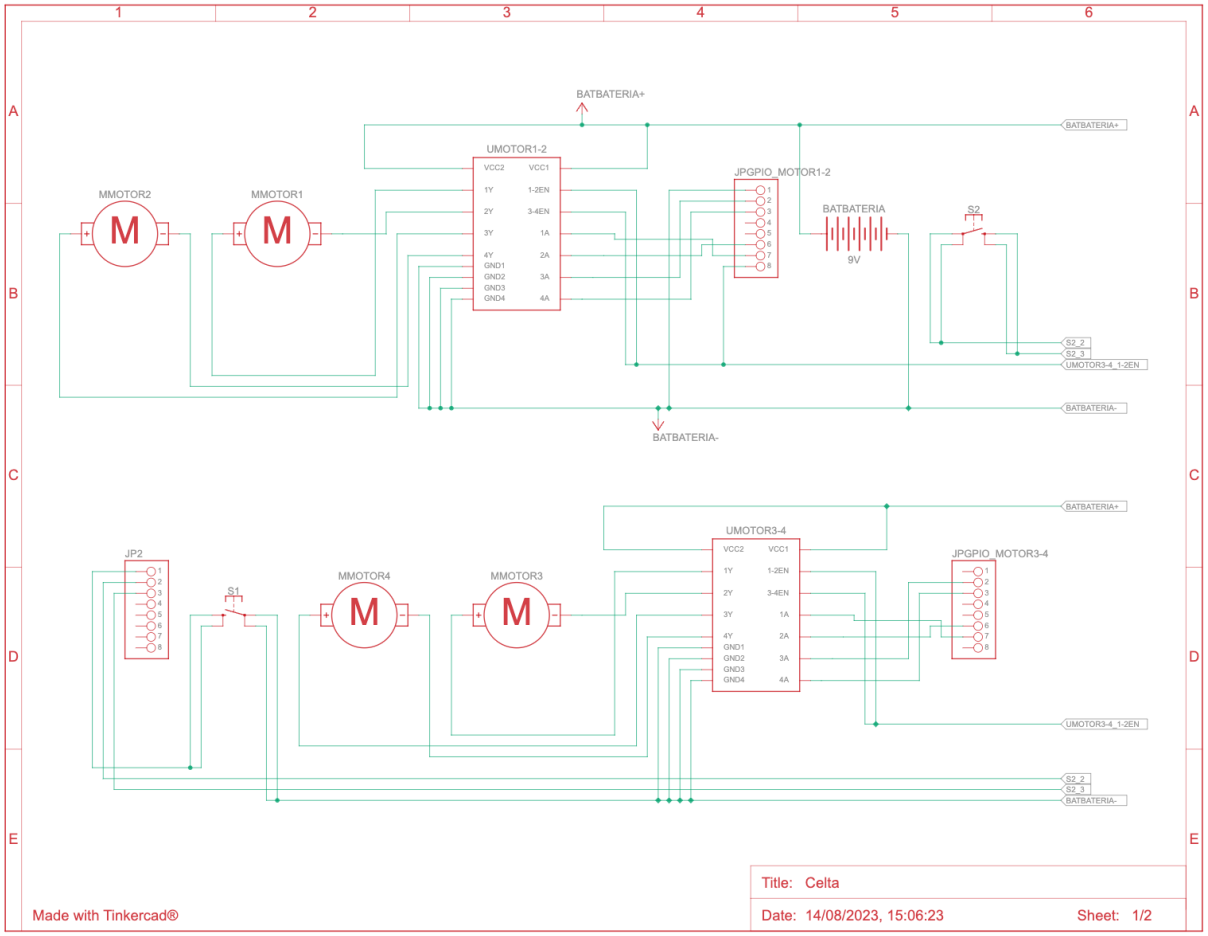
3.2 *Raspberry PI*

Conforme explicado na seção 2.2, foi utilizado o computador *Raspberry Pi 3 Model B v 1.2* para a construção do novo Celta.

3.2.1 Configuração da *Raspberry Pi*

Para que o projeto possa funcionar corretamente, houve a instalação do sistema operacional *Raspberry Pi OS* conforme a explicação no *site* (RASPBERRY PI, 2023b). Para tal, foi necessário instalar o programa *Raspberry Pi Imager*. Com o programa instalado num computador, ele irá realizar o *download* da imagem do sistema operacional e carregar em um cartão *Micro SD* previamente conectado neste mesmo computador. Após

Figura 18 – Circuito elétrico do novo Celta (página 1).



Fonte: (TINKERCAD, 2023).

o processo informado acima, o *Raspberry Pi* pode ser ligado e ter as configurações do usuário realizadas.

O sistema operacional traz consigo o Python 3.2 instalado, de maneira que não foi necessário instalar qualquer outro *software*.

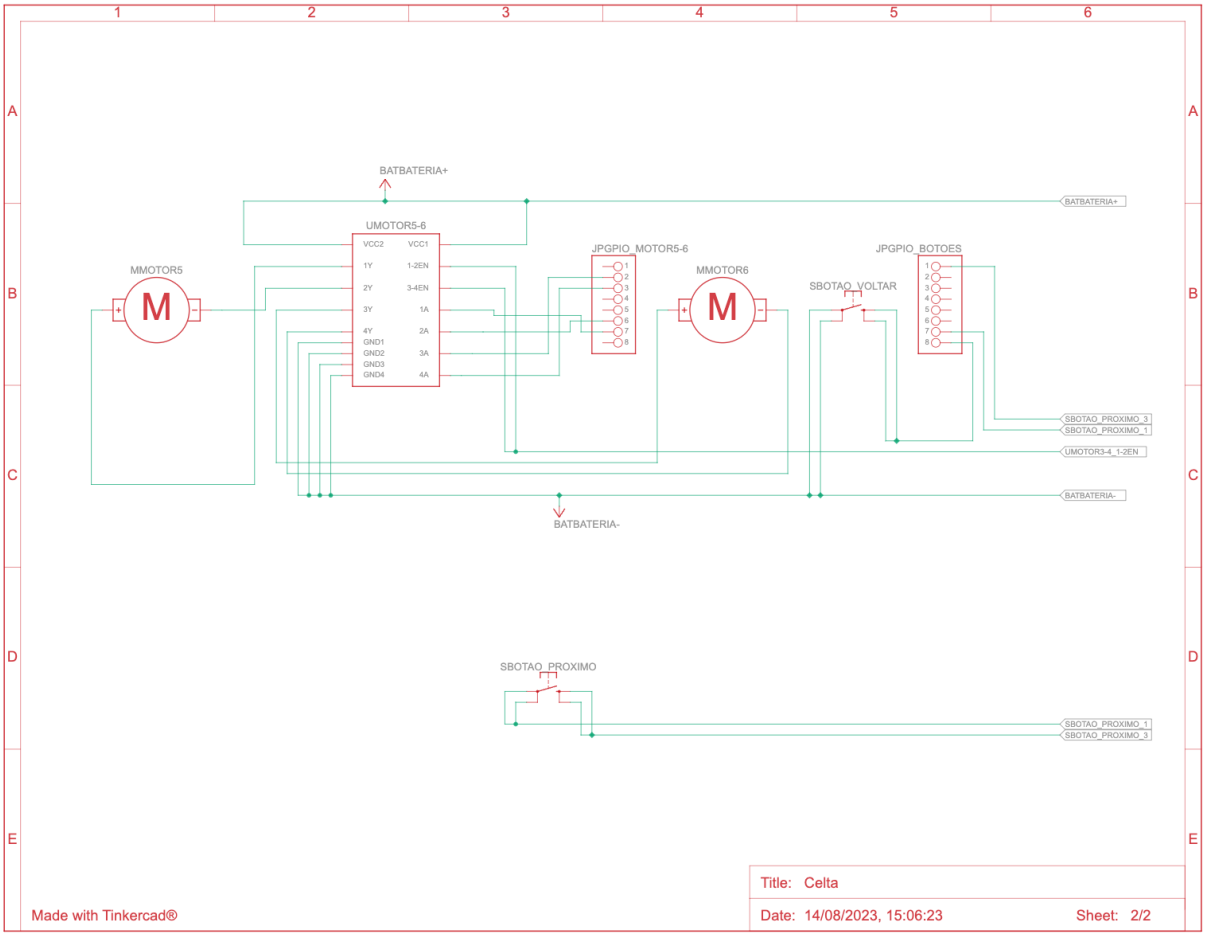
3.2.2 *Software* do embarcado

Conforme visto no código presente em subseção A.1.1 e no fluxograma da Figura 25, é possível destacar alguns pontos:

1. Na linha 202, há a definição do método ___main___ do programa *Python*. O programa se inicia nesse ponto.

2. Logo no início do programa é criado um *socket* no IP 192.168.15.30, na porta 50000. Este IP foi configurado para sempre ser disponibilizado ao *Raspberry Pi*. Ele foi escolhido de forma aleatória, por ser um IP livre na rede no momento da configuração.

Figura 19 – Circuito elétrico do novo Celta (página 2).



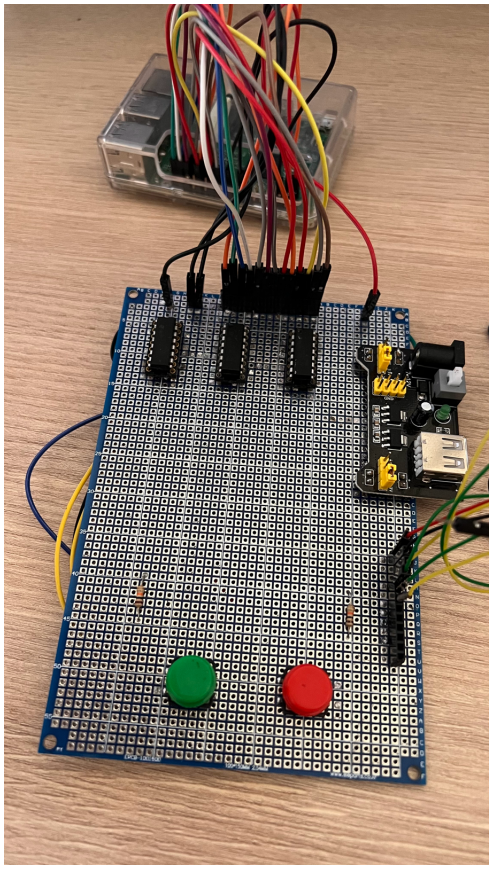
Fonte: (TINKERCAD, 2023).

3. Este programa configura um servidor *socket* que irá se conectar a um único cliente, conforme a linha 205, seguindo a arquitetura cliente-servidor visto na subseção 2.1.3.

4. Na linha 207 o método `setup()` é chamado. Neste método é realizada a configuração de todas as portas GPIO da *Raspberry Pi*.

5. Nas linhas 210 e 211, há a configuração dos métodos `callback_botao1()` e `callback_botao2()`. Quando os botões para voltar/avançar letras são pressionados, seus respectivos métodos são acionados e as variáveis de controle recebem valores de acordo com o tempo do pressionamento do botão. É importante ressaltar o uso do *bouncetime* de 75 mili-segundos. Quando um botão físico é pressionado (ou solto), ele pode ocasionar pequenas oscilações elétricas devido à natureza mecânica do componente. Com isso, múltiplas leituras "falsas" do estado do botão em um curto período de tempo podem ser interpretadas, mesmo que o usuário tenha pressionado o botão apenas uma vez. Assim, esse tempo configurado é necessário para a estabilização das oscilações.

Figura 20 – Circuito elétrico do novo Celta visto do lado superior.



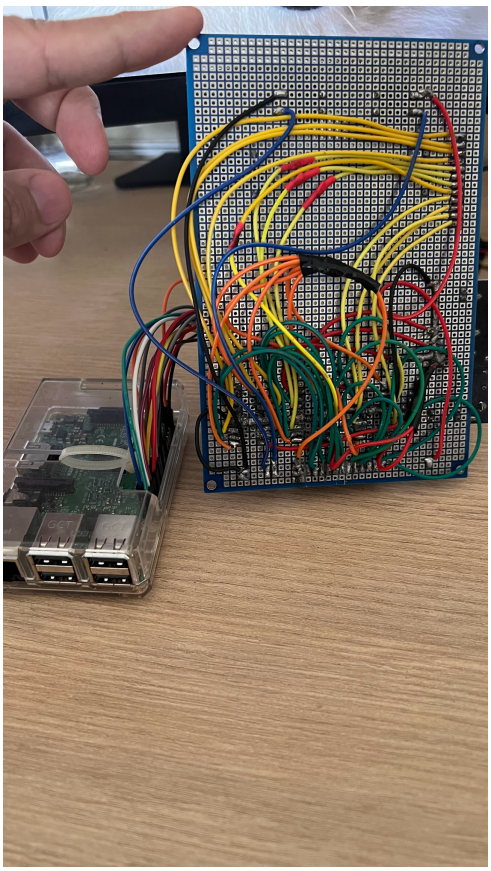
Fonte: Do Autor.

6. Ao final do programa, é exibida a mensagem no console e se inicia o *loop*. Caso o usuário pressione as teclas Ctrl + C, o programa é encerrado através da captura de exceção. Consequentemente ocorre o encerramento da conexão *socket* e a limpeza das portas GPIO.

7. O método `loop()` aguarda a conexão de um cliente num *loop* inicial. Internamente a este *loop*, existe um segundo *loop* para que o programa fique constantemente verificando se alguma mensagem foi enviada pelo cliente. Caso o cliente se desconecte, a *thread* é encerrada e com isso o segundo *loop* é encerrado no mesmo momento. O *software* reinicia o fluxo com o primeiro *loop* e fica no aguardo de uma nova conexão de um cliente.

8. Diferente da versão antiga do embarcado do Celta, o programa atual aguarda uma *string* de 6 posições. Cada um desses caracteres é responsável por seu respectivo pino. Dessa forma, o primeiro caractere é responsável pelo pino 1, o segundo caractere é responsável pelo pino 2 e assim por diante. O conteúdo esperado é binário, ou seja, “0” ou “1”. O valor “0” fará com que o motor rode no sentido com que o pino abaixe, enquanto que o valor “1” fará com que o motor gire no sentido com que o pino se

Figura 21 – Circuito elétrico do novo Celta visto do lado inferior.



Fonte: Do Autor.

Figura 22 – Tela inicial do computador no *Raspberry Pi OS*.

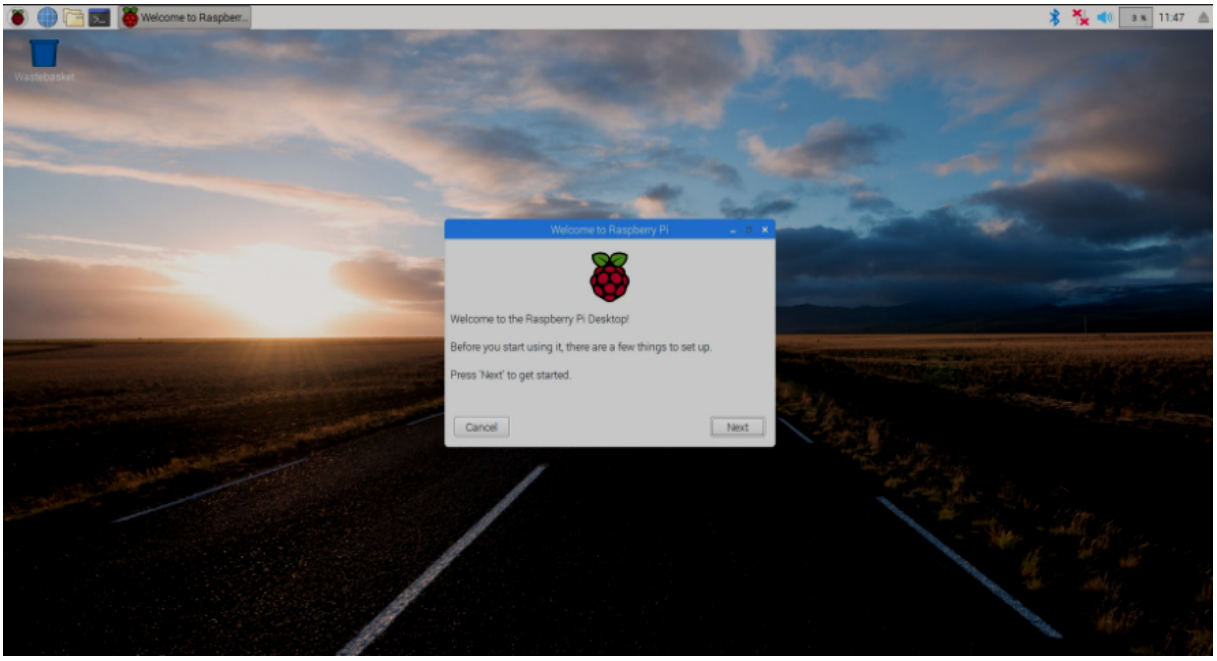


Figura 23 – Fonte: Do Autor.

Figura 24 – Fluxo de comunicação entre o *hardware* e *software*. De acordo com o botão pressionado e o tempo de pressionamento, haverá uma interpretação do comando pelo *software* no *desktop* e o envio do código da letra atual a ser representada no *hardware*.

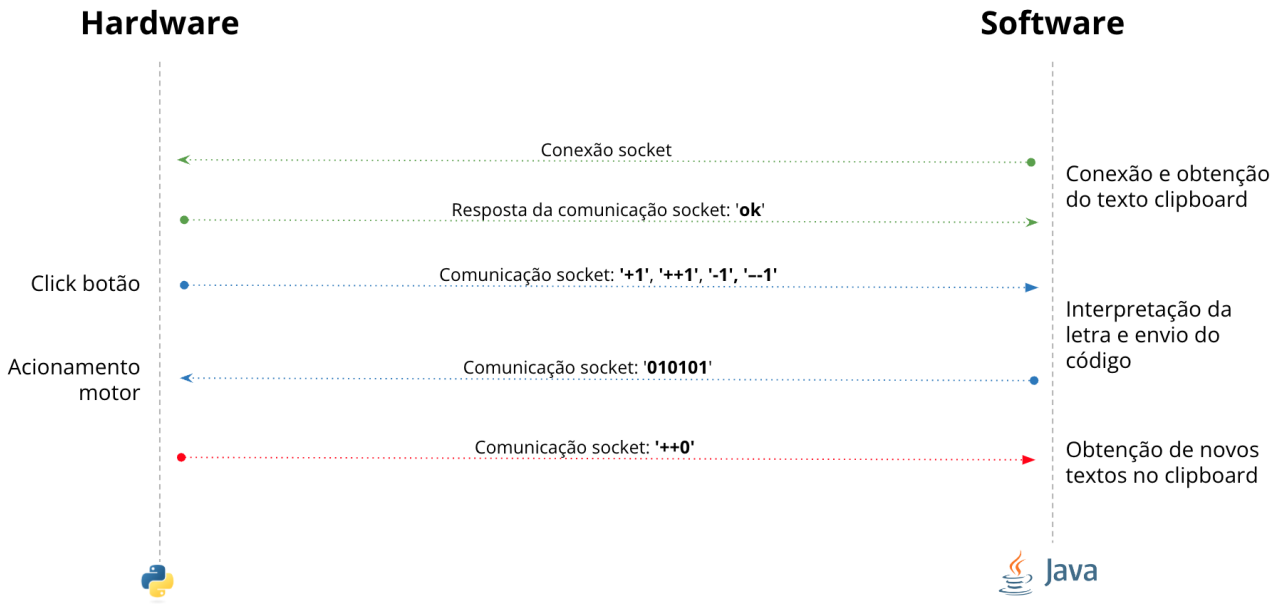


Figura 25 – Fonte: Do Autor.

levante. Ao receber a *string* "001001", a Célula tátil do Celta irá ativar o pino 3 e o pino 6, enquanto que todos os outros pinos (1, 2, 4 e 5) serão desativados.

9. Qualquer outro caractere que não seja "0" ou "1" é desconsiderado.

10. Outro ponto importante dessa comunicação está nas ações dos botões: No hardware existem os botões voltar e avançar. De acordo com o tempo de pressionamento e da combinação dos botões pressionados, o cliente recebe uma *string* como resposta via *socket*, conforme exemplificado na Figura 25 e na tabela Tabela 5.

11. Para determinar o final da *string* nos textos enviados ao cliente, o caractere "\n" indica o final do texto.

3.2.3 Conexão do circuito com a *Raspberry Pi*

Nesta subseção são abordados alguns aspectos sobre a comunicação entre circuito e *Raspberry Pi*.

3.2.3.1 Teste de conexão das portas

É importante testar a conexão entre os equipamentos, de modo a verificar se o sentido das entradas corresponde ao sentido do giro do motor. Ao receber "0", o pino deve ser desabilitado. Ao receber "1" pino será ativado.

Figura 26 – Fluxo de execução do programa *Python* executado no *hardware*. Neste diagrama é exemplificado em alto nível quanto ao ciclo de vida de execução do programa.

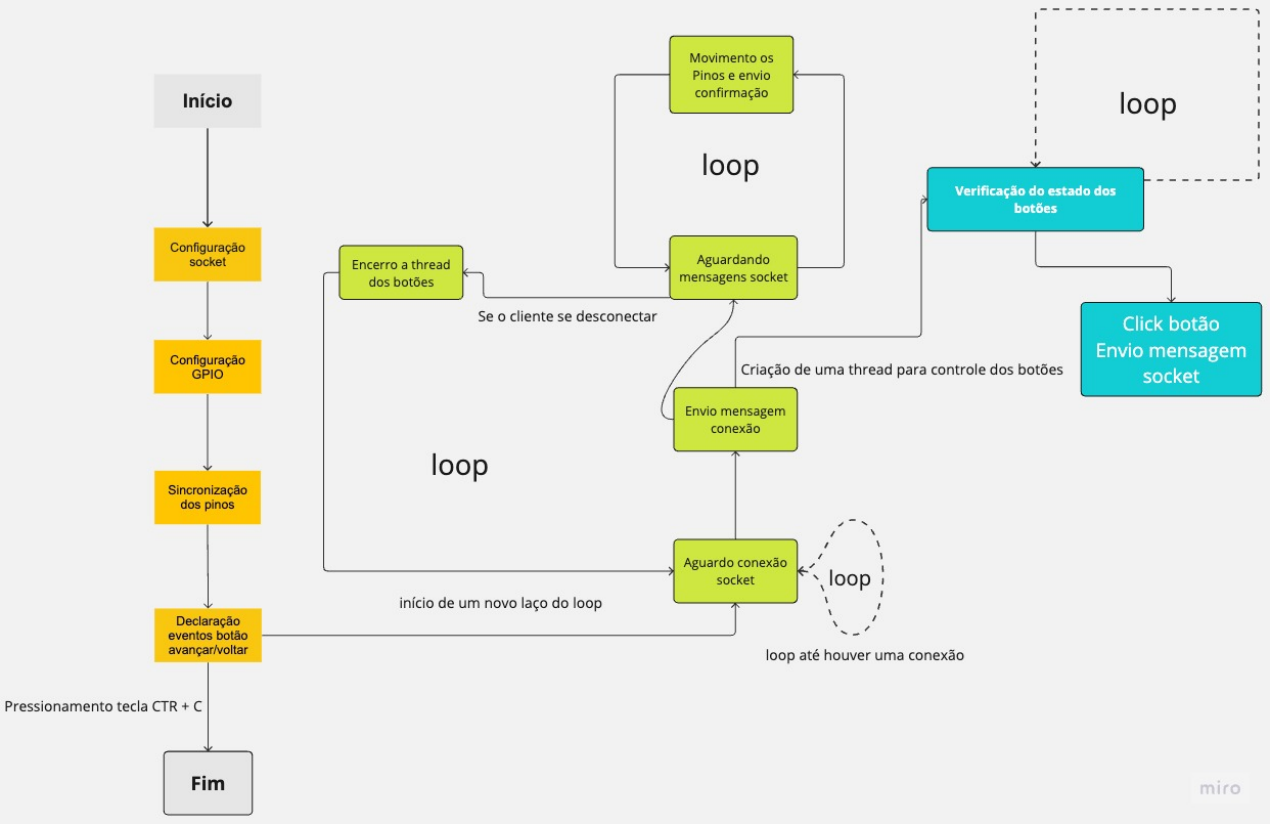


Figura 27 – Fonte: Do Autor.

Para testar se o pino está correto, basta enviar essa combinação de comandos de forma a verificar a situação de cada um dos 6 pinos. Caso esteja incorreto, é necessário trocar as portas do motor (*output*) e verificar se ação está ocorrendo conforme o combinado nesta próxima tentativa.

3.2.3.2 Fluxo de conexão dos pinos

Neste ponto, conforme comentado anteriormente no Item 7, existem duas camadas de *loop*: no *loop* inicial (*loop* externo) o servidor fica no aguardo de uma conexão oriunda pelo cliente. Quando um cliente se conecta, o *loop* com a observação dos botões (*loop* interno) é iniciado e executa até o cliente ser desconectado. Ao encerrar o *loop* interno, o *loop* externo é retomado e volta a esperar uma nova conexão de um cliente.

Tabela 5 – Tabela com as respostas enviadas via *socket* ao cliente de acordo com as combinações de pressionamento dos botões.

Botão pressionado	Pressionamento curto (<0,3s)	Pressionamento longo (>=0,3s)
Anterior	-1	-1
Próximo	+1	++1
Anterior + Próximo	+0	++0

Fonte: Do Autor.

3.2.3.3 Fluxo de observação dos botões

3.2.4 Repositório

O código fonte do embarcado visto nesse capítulo, está disponível no GitHub do autor[1].

[1] <https://github.com/juan-santos/celta_ufscar/tree/feature/embarcado-python/new_celta/raspberry/ponto.py>

4 Software

Neste capítulo, é apresentado o desenvolvimento do *software* na linguagem Java, que apresenta um cliente *socket* e desempenha a função de recuperar textos da área de transferência do usuário, também conhecida como *clipboard*.

4.1 *Maven*

De acordo com o *site* (APACHE MAVEN PROJECT, 2023), o *Apache Maven* é uma ferramenta de gerenciamento de projetos de *software* utilizada na comunidade de desenvolvimento. Com ele é possível automatizar tarefas relacionadas à construção, compilação, teste, documentação e distribuição de projetos.

4.1.1 POM

Uma das principais características do Maven é o seu modelo de projeto, conhecido como POM (*Project Object Model*), que é um arquivo XML que descreve as configurações, dependências e outras informações essenciais do projeto. O código está disponibilizado na subseção A.2.2.

4.2 Classe FormCelta

A classe FormCelta é a responsável por exibir uma interface gráfica através da biblioteca *Swing*. O *Swing* está incluído na plataforma *Java Standard Edition* (Java SE) e é empregado na criação de interfaces gráficas de usuário para aplicativos Java. Com o *Swing*, é possível desenvolver interfaces gráficas interativas e visualmente atraentes para os programas. O código está disponibilizado na subseção A.2.2.

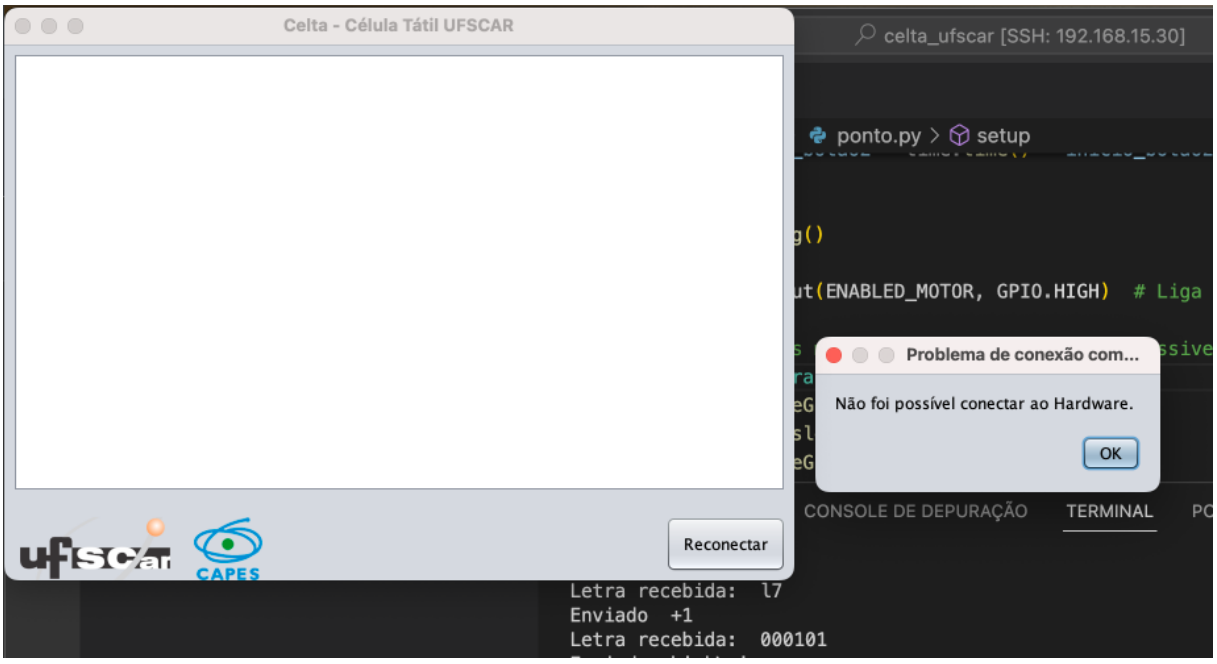
4.3 Classe Utils

A prática de criar uma classe com o nome “Utils” é comum em muitos projetos que seguem a programação orientada a objetos. Serve para agrupar funções, métodos e utilidades que não se encaixaram diretamente em uma única categoria ou classe específica. Esse tipo de abordagem facilita na organização e encapsulamento de funcionalidades, mesmo quando elas não têm um escopo mais específico. O código está disponibilizado na subseção A.2.3.

4.4 Classe Comunicação

Na classe Comunicacao, ocorre a criação de um cliente *socket*, sendo este o responsável pela comunicação bidirecional com o *hardware*. Caso ocorra algum erro de comunicação com o servidor, ele habilitará um botão para uma nova tentativa e exibirá uma mensagem de erro, conforme ilustrado na Figura 28. Para ler ou escrever dados, é necessário que a conexão *socket* seja realizada com sucesso. O código está disponibilizado na subseção A.2.4.

Figura 28 – Erro de conexão entre o *software* e o embarcado. No momento em que o erro é disparado, o botão para reconectar é exibido na interface, além da mensagem de erro que também é proferida.



Fonte: Do Autor.

4.5 Classe *Raspberry*

A classe *Raspberry* implementa a interface *Runnable*, uma classe que é frequentemente utilizada no Java para criar *threads* e realizar programação paralela. A interface no Java é um contrato que define um conjunto de métodos que uma classe deve implementar se quiser cumprir essa interface. No caso da interface *Runnable*, ela possui um único método chamado `run()`. A classe que implementar a interface *Runnable* precisará fornecer uma implementação para o método `run()`, que contém o código que será executado quando a *thread* for iniciada.

```
1 /**
2  * Metodo responsavel por executar a thread de conexao com o
      raspberry
```

```
3  */
4 @Override
5 public void run() {
6
7     this.clearHardware();
8     this.copyValuesClipboard();
9
10    while (!Thread.currentThread().isInterrupted()) {
11        String dado = this.controlePorta.lerDados();
12
13        if(dado != null){
14            this.execCmd(dado);
15        } else {
16            this.pnlButton.revalidate();
17            this.pnlButton.repaint();
18            Thread.currentThread().interrupt();
19        }
20
21    }
22
23    this.clearHardware();
24    this.controlePorta.close();
25 }
```

Por implementar a interface *Runnable*, o método `run()` é executado em paralelo com a *thread* principal. Tal artifício possibilita a constante obervação de dados obtidos via *socket*, oriundos do servidor. Caso haja alguma ação que deva ser realizada, este aplicativo Java interpretará através de um *switch* o respectivo comando, conforme a Tabela 6. O código está disponibilizado na subseção A.2.5.

Tabela 6 – Tabela com as possíveis ações a serem realizadas, de acordo com a *string* recebida via *socket* do servidor.

Mensagem do embarcado	**Descrição da execução no software**
“-1”	Retroceder a posição da letra atual em uma unidade
“- -1”	Voltar para a primeira letra da palavra anterior
“+1”	Avançar a posição da letra atual em uma unidade
“++1”	Avançar para a primeira letra da próxima palavra
“+0”	Nenhuma função foi atribuída para esse comando até o momento
“++0”	Atualizar o texto que será lido

Fonte: Do Autor.

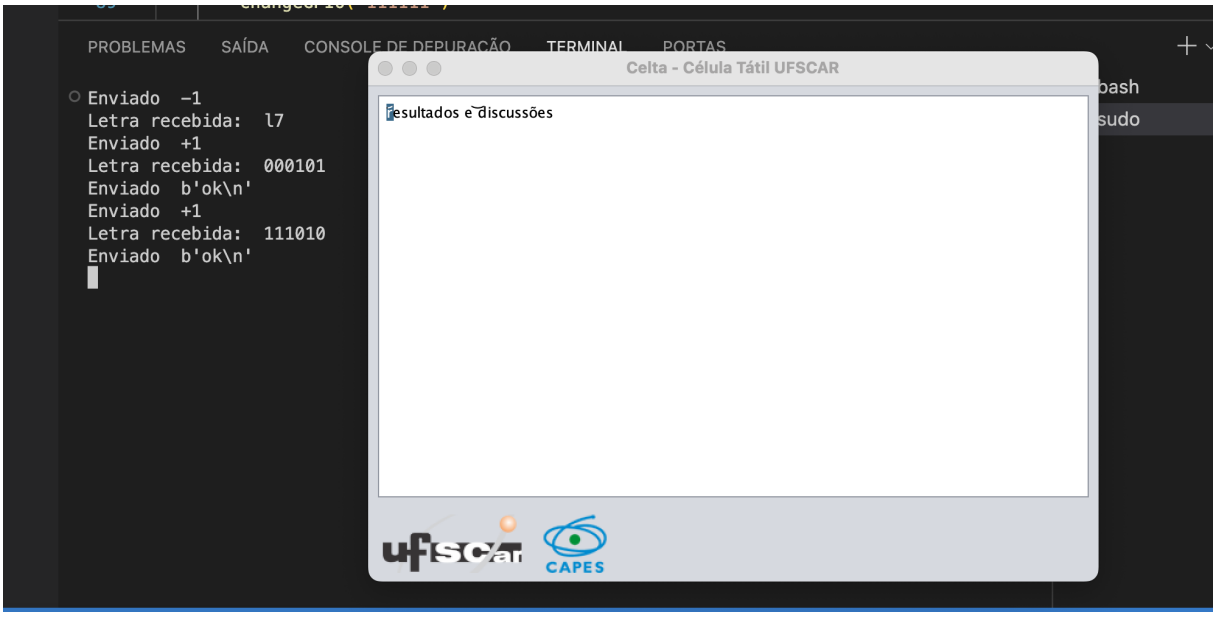
Esta classe é a responsável por transformar letras e números no formato conhecido

pelo servidor, para que os pinos representem o caractere desejado. Como este *software* é uma transição tecnológica em relação à versão anterior, houve a troca do padrão de comunicação. Para ativar um pino, basta posicionar o caractere “1” no pino que deve ser ativado, enquanto que para desativar, basta posicionar o caractere “0” no pino desejado.

Assim, para ativar todos os pinos na versão anterior, era necessário enviar a *string* “l1l2l3l4l5l6”, na qual a letra “l”, seguida do número, indicava a porta que deveria ser ligada, enquanto que para desativar era utilizado a letra “d”. Para ativar apenas o pino 4 e 6, seria necessário a *string* “d1d2d3l4d5l6”, enquanto que no padrão atual é necessário apenas o envio da *string* “000101”, conforme exemplificado na Tabela 7.

Outra característica importante para auxiliar o desenvolvedor está na identificação do caractere atual através do foco no elemento JTextArea, conforme a Figura 29 e de acordo com a Figura 30.

Figura 29 – Comunicação do *software* com o embarcado, navegando através dos botões para ler o texto "Resultados e Discussões".



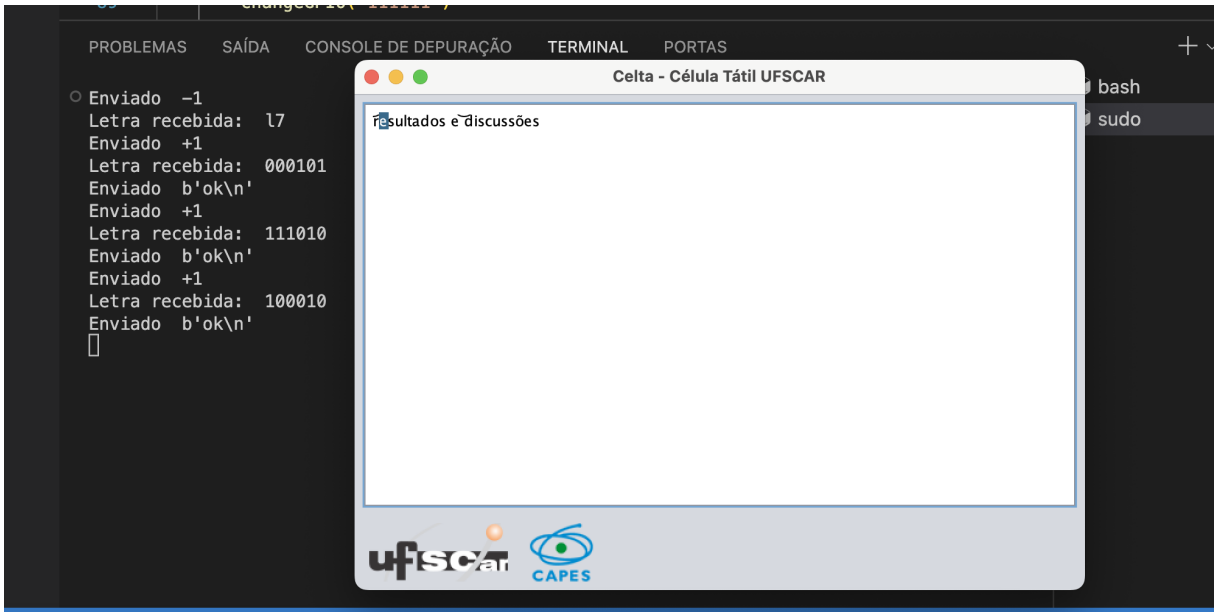
Fonte: Do Autor.

Tabela 7 – Tabela comparativa entre a versão anterior e a nova versão, com exemplos de comandos em cada uma das versões.

Pinos ativados	Comando anterior	Comando novo
Ativar pino 2, 4 e 6	“d1l2d3l4d5l6”	“010101”
Ativar pino 1, 3 e 5	“l1l3l5d2d4d6”	“101010”
Ativar pino 3 e 4	“l3l4d1d2d5d6”	“001100”
Nenhum pino ativado	“d1d2d3d4d5d6”	“000000”
Todos os pinos ativados	“l1l2l3l4l5l6”	“111111”

Fonte: Do Autor.

Figura 30 – Comunicação do *software* com o embarcado, avançando através do botão para ler o segundo caractere do texto "Resultados e Discussões".



Fonte: Do Autor.

4.6 Interface *Reader* para leitura de arquivos

Através do uso das interfaces no Java, o programa torna-se apto para que em futuras melhorias possa receber diversos tipos de formatos de arquivos. Atualmente os formatos .TXT e .PDF foram implementados. O código está disponibilizado na subseção A.2.6 e seu diagrama é exibido na Figura 31.

4.6.1 Classe TXTReader

Implementação para que seja possível a leitura em arquivos .TXT. O código está disponibilizado na subseção A.2.7.

4.6.2 Classe PDFReader

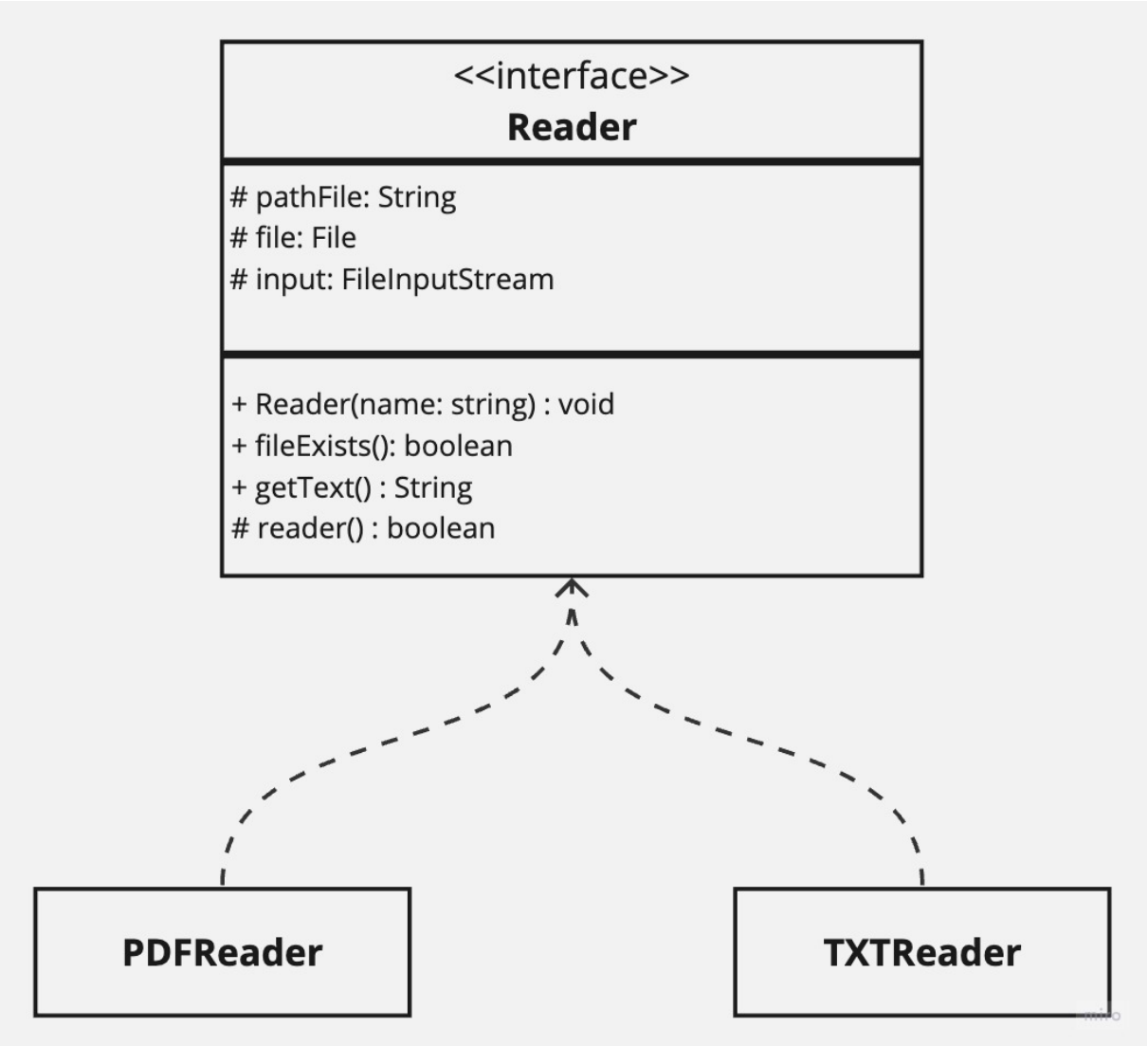
Implementação para que seja possível a leitura em arquivos .PDF. O código está disponibilizado na subseção A.2.8.

4.7 Repositório

O código fonte do aplicativo Java está disponível no GitHub do autor[1].

[1] <https://github.com/juan-santos/celta_ufscar/tree/feature/embarcado-python/new_celta/java/Celta>

Figura 31 – Interface Reader.



Fonte: Do Autor.

5 Resultados e discussões

Neste capítulo são discutidos os resultados obtidos com as alterações propostas ao longo deste trabalho. Apesar da *Raspberry Pi* possuir um valor relativamente superior a um Arduíno, seu uso permite avanços consideráveis ao projeto. Enquanto o Arduíno é conhecido por sua acessibilidade econômica, a *Raspberry Pi* é uma opção mais versátil, mas também pode ser um pouco mais cara. O preço de ambos os dispositivos pode variar significativamente de acordo com o modelo e suas características específicas. Numa busca realizada no *marketplace* **Mercado Livre**, foi possível encontrar a versão 1 por por R$ 239,00 (MERCADO LIVRE, 2023b), enquanto que o modelo 4, mais avançado e atualizado, pode ser encontrado na faixa dos R$720,00 (MERCADO LIVRE, 2023c). A escolha entre *Raspberry Pi* e Arduíno não deve ser estritamente baseada no preço, mas sim nas necessidades específicas do projeto. Ambos têm seu lugar no mundo da eletrônica e da automação, e a decisão depende de fatores como a complexidade do projeto, o orçamento disponível e a capacidade de processamento necessária.

A atualização tecnológica do Celta trouxe diversas facilidades e melhorias para o equipamento. Dentre elas:

Comunicação sem fio: A troca da comunicação serial com fio para a comunicação sem fio através do uso do *wi-fi* permitiu uma maior mobilidade e flexibilidade ao equipamento. Dispositivos podem se comunicar sem estarem fisicamente conectados, permitindo uma maior liberdade de movimento.

Programação em Java: Com essa melhoria, o Celta se posiciona como uma ferramenta mais poderosa e adaptável, pronta para atender às necessidades e demandas dos usuários em diferentes contextos e sistemas operacionais.

Leitura de arquivos: Através dessa melhoria, o Celta passa a ler documentos no formato *.TXT* e *.PDF*. Isso torna o dispositivo ainda mais versátil e amplia suas possibilidades de uso para os usuários, que poderão interagir com arquivos PDFs de forma prática e eficiente. Tal artifício permite um grande salto de acessibilidade a conteúdos, sendo este um dos maiores ganhos desta versão.

Programação em *Python*: A escolha da linguagem *Python* para o *hardware* do Celta baseou-se em sua ampla popularidade e comunidade de desenvolvedores, o que simplifica o acesso a recursos, suporte e melhorias futuras. Isso proporciona uma eficiente gestão dos componentes do dispositivo, promovendo uma conexão fluida entre *hardware* e *software*. Ao combinar *Python* com portas GPIO, o Celta se beneficia de programação flexível e intuitiva, simplificando seu desenvolvimento e manutenção.

Essa abordagem torna o projeto mais amigável para desenvolvedores, facilitando a incorporação eficaz de novas funcionalidades e melhorias, contribuindo para o seu progresso contínuo como um produto tecnologicamente avançado e adaptado às necessidades dos usuários.

Otimização na comunicação: Conforme visto na seção 4.5, houve uma otimização no protocolo de comunicação entre o embarcado e o *software*. Com essa mudança, o tamanho da mensagem foi reduzida para a metade do tamanho, tendo um ganho de 50% de otimização.

6 Conclusões

A acessibilidade é um conceito dinâmico e fundamental em nossa sociedade, permitindo que todos possam ter oportunidades iguais aos recursos disponíveis. Com o objetivo de promover a inclusão de pessoas com deficiência visual a um baixo custo, o projeto Celta facilita atividades de letramento em braile, disseminando e proporcionando treinamento para um maior número de pessoas nessa área de leitura.

Através dos avanços propostos nesse trabalho, houve um grande salto na busca pela acessibilidade ao permitir a leitura de arquivos .PDF e .TXT. Tal avanço torna o dispositivo ainda mais versátil e amplia suas possibilidades de uso para os usuários. Com a mudança de abordagem em seu protocolo de comunicação, foi possível trazer ao projeto uma maior mobilidade e flexibilidade ao eliminar a comunicação por fio. Por meio da nova versão do software em Java, o Celta se posiciona como uma ferramenta adaptável e pronta para atender às necessidades dos usuários em diferentes contextos e sistemas operacionais.

Apesar de todos esses avanços exibidos, existem novos caminhos possíveis para aprimorar ainda mais este projeto, tornando-o cada vez mais acessível para o seu público. Na seção 6.1, são apresentados possíveis caminhos para o avanço deste projeto.

6.1 Trabalhos Futuros

Melhoria na navegabilidade: Através de um estudo multidisciplinar que se concentre no *design*, avaliação e implementação de interfaces para pessoas com deficiência visual, que busque melhorar a interação entre os usuários e o dispositivo. Tal estudo pode ser viabilizado pelo uso do campo IHC (Interação Humano-Computador).

Melhoria na segurança: Na configuração atual, a comunicação entre o dispositivo embarcado e o *software* carece de medidas de segurança mais adequadas. Isso resulta no tráfego de dados através da rede sem qualquer forma de criptografia, que pode acabar expondo esses dados à leitura e interpretação por qualquer pessoa com acesso à rede. A implementação de melhorias nesse aspecto é de grande importância, pois garante não somente a confidencialidade dos dados, mas também a privacidade e uma série de outros benefícios essenciais aos usuários.

Aumento do número de células braile: Através do aumento do número de células braile no equipamento, é possível viabilizar a leitura de caracteres que requerem mais de uma célula para serem exibidos corretamente. Essa melhoria pode ser alcançada por meio da implementação de um multiplexador. O multiplexador, pertencente à área de circuitos e sistemas digitais, tem a finalidade de selecionar um dos diversos

sinais de entrada e direcioná-lo para uma única saída, com base nos sinais de controle fornecidos. Nesse contexto, o multiplexador atua como um componente controlado eletronicamente, comparável a um interruptor, permitindo a escolha do sinal de entrada a ser transmitido para a saída (ABDALA, 2023).

Inclusão de novos formatos: Neste projeto, o Celta foi incrementado com a leitura de arquivos nos formatos .TXT e .PDF. Contudo, seria benéfico expandir a variedade de formatos suportados pelo dispositivo. Para alcançar esse objetivo, é necessário realizar uma análise a fim de determinar quais tipos de arquivos são utilizados por uma considerável base de usuários.

Integração com leitores de tela: A acessibilidade na *web* pressupõe que os *sites* e portais sejam projetados de modo a permitir que todas as pessoas possam perceber, compreender, navegar e interagir de maneira efetiva com as páginas. O termo Tecnologia Assistiva faz referência aos recursos que contribuem para que indivíduos com deficiência possam desfrutar de maior independência e autonomia. Esses recursos abrangem desde elementos simples até sistemas computacionais bastante complexos. Ao tratar-se de acessibilidade na *web*, enfoca-se nos recursos de Tecnologia Assistiva, os quais oferecem soluções para que pessoas com algum tipo de deficiência possam acessar os recursos de *hardware* e/ou *software* que o mundo digital proporciona (CONSELHO REGIONAL DE BIOMEDICINA - 3ª REGIÃO, 2021). Além de ampliar o número de formatos suportados, a capacidade de extrair conteúdo de leitores de tela representaria um notável progresso para o dispositivo. Esse avanço teria um impacto significativo ao garantir uma navegação mais acessível para pessoas com deficiência visual, melhorando assim a experiência e a usabilidade do equipamento.

A Apêndice

A.1 *Software*

A.1.1 Código *Python*

```
#!usr/bin/env python3
import RPi.GPIO as GPIO
import time
from typing import List, Tuple
import socket
import threading

# Variaveis para gerenciar os botoes
inicio_botao1 = 0
inicio_botao2 = 0

final_botao1 = 0
final_botao2 = 0

duploClick = False

# pino para ativar todos os motores
ENABLED_MOTOR = 2 #PIN 3

# pinos dos botoes
BUTTON_BACK = 24 #PIN 18
BUTTON_NEXT = 25 #PIN 22

# pinos responsaveis por fazer o motor girar no sentido que
    ative o braile
ATIVAR1 = 0 #PIN 27
ATIVAR2 = 5 #PIN 29
ATIVAR3 = 6 #PIN 31
ATIVAR4 = 13 #PIN 33
ATIVAR5 = 19 #PIN 35
ATIVAR6 = 26 #PIN 37

# pinos responsaveis por fazer o motor girar no sentido que
    desative o braile
```

```
33 DESATIVAR1 = 7 #PIN 26
34 DESATIVAR2 = 1 #PIN 28
35 DESATIVAR3 = 12 #PIN 32
36 DESATIVAR4 = 16 #PIN 36
37 DESATIVAR5 = 20 #PIN 38
38 DESATIVAR6 = 21 #PIN 40
39
40 EnabledList: Tuple[int] = (ATIVAR1, ATIVAR2, ATIVAR3,
       ATIVAR4, ATIVAR5, ATIVAR6)
41 DisabledList: Tuple[int] = (DESATIVAR1, DESATIVAR2,
       DESATIVAR3,
42                 DESATIVAR4, DESATIVAR5, DESATIVAR6)
43
44 tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
45
46 # Funcao de callback para o Botao 1
47 def callback_botao1(channel):
48     global inicio_botao1
49     global final_botao1
50     global duploClick
51
52     estado_botao2 = GPIO.input(BUTTON_NEXT)
53
54     if not GPIO.input(BUTTON_BACK):  # Botao pressionado
55         inicio_botao1 = time.time()
56
57         if estado_botao2:
58             duploClick = True
59
60     else:  # Botao solto
61         final_botao1 = time.time() - inicio_botao1
62
63 # Funcao de callback para o Botao 2
64 def callback_botao2(channel):
65     global inicio_botao2
66     global final_botao2
67     global duploClick
68
69     estado_botao1 = GPIO.input(BUTTON_BACK)
70
71     if GPIO.input(BUTTON_NEXT):  # Botao pressionado
72         inicio_botao2 = time.time()
```

```
73
74          if not estado_botao1:
75              duploClick = True
76
77      else:  # Botao solto
78          final_botao2 = time.time() - inicio_botao2
79
80  def setup():
81      GPIOConfig()
82
83      GPIO.output(ENABLED_MOTOR, GPIO.HIGH)  # Liga todos os
            motores
84
85      #inicio os motores, de forma a abaixar possiveis pinos
            que estejam levantados
86      for i in range(1000):
87          changeGPIO('000000')
88          time.sleep(0.4)
89          changeGPIO('111111')
90          time.sleep(0.4)
91
92  # Metodo responsavel por configurar a placa do raspberry
93  def GPIOConfig():
94      GPIO.setmode(GPIO.BCM)
95
96      GPIO.setup(ENABLED_MOTOR, GPIO.OUT) # pino responsavel
            por ativar/desativar todos os motores
97
98      GPIO.setup(BUTTON_BACK, GPIO.IN,
            pull_up_down=GPIO.PUD_UP)    # Pino do botao como
            saida e aciona o pull-up
99      GPIO.setup(BUTTON_NEXT, GPIO.IN,
            pull_up_down=GPIO.PUD_DOWN)
100
101     GPIO.setup(EnabledList + DisabledList, GPIO.OUT) #
            configuro todos os pinos
102
103 # Metodo responsavel por ficar escutando os botoes e enviar
      o status ao cliente
104 def loop_button(encerrar_event, client_socket):
105     global final_botao1
106     global final_botao2
```

```
107     global duploClick
108 
109     while not encerrar_event.is_set():
110         if duploClick:
111             while (not final_botao1) or (not final_botao2):
112                 pass
113 
114             time.sleep(0.2)
115 
116             if final_botao1 > 0.3 or final_botao2 > 0.3:
117                 client_socket.send("++0\n".encode())
118                 print("Enviado ", "++0")
119             else:
120                 client_socket.send("+0\n".encode())
121                 print("Enviado ", "+0")
122 
123             final_botao1 = 0
124             final_botao2 = 0
125             duploClick = False
126 
127         if final_botao1:
128             if final_botao1 > 0.3:
129                 client_socket.send("++1\n".encode())
130                 print("Enviado ", "++1")
131             else:
132                 client_socket.send("+1\n".encode())
133                 print("Enviado ", "+1")
134 
135             final_botao1 = 0
136 
137         elif final_botao2:
138             if final_botao2 > 0.3:
139                 client_socket.send("--1\n".encode())
140                 print("Enviado ", "--1")
141             else:
142                 client_socket.send("-1\n".encode())
143                 print("Enviado ", "-1")
144 
145             final_botao2 = 0
146 
147         pass
148 
```

```
149 # Loop principal
150 def loop():
151     while True:
152         client_socket, address = tcp.accept()
153         message = "Conectado\n"
154 
155         client_socket.send(message.encode())
156 
157         encerrar_event = threading.Event()
158         button_thread = threading.Thread(daemon=True,
               target=loop_button, args=(encerrar_event,
               client_socket,))
159         button_thread.start()
160 
161         while True:
162             decodeMessage = client_socket.recv(1024).decode()
163 
164             if not decodeMessage:
165                 encerrar_event.set()
166                 button_thread.join()
167                 break
168 
169             resposta = changeGPIO(decodeMessage)
170             if resposta:
171                 client_socket.send(resposta.encode())
172                 print("Enviado ", resposta.encode())
173 
174 #Metodo responsavel por ler o codigo da letra e
      ativar/desativar os pinos correspondentes
175 def changeGPIO(message: str) -> str :
176     print("Letra recebida: ", message)
177     enabled: List[int] = []
178     disabled: List[int] = []
179 
180     if len(message) == 6:
181         for value in message:
182             if value in ['1', '0']:
183                 enabled.append(GPIO.HIGH if value == '1'
                       else GPIO.LOW)
184                 disabled.append(GPIO.LOW if value == '1'
                       else GPIO.HIGH)
185             else:
```

```
186                 return ''
187     else:
188         return '' # caso o codigo recebido pelo aplicativo
               nao esteja adequado ao padrao
189
190     GPIO.output(EnabledList + DisabledList, enabled +
           disabled)
191     time.sleep(0.4)
192
193     GPIO.output(EnabledList + DisabledList, GPIO.LOW) #Apos
           os motores exibirem a
194                                                     #letra eu
                                                           desativo
                                                           todos os
                                                           motores
195     return "ok\n"
196
197 # Metodo responsavel por limpar as placas GPIO e encerrar a
      conexao socket
198 def destroy():
199     GPIO.cleanup()
200     tcp.close()
201
202 if __name__ == '__main__':     # Program start from here
203     # Variavel responsavel pela conexao socket
204     tcp.bind(('192.168.15.30', 50000))
205     tcp.listen(1)
206
207     setup()
208
209     # Registro dos callbacks para os eventos de nivel alto
           (RISING)
210     GPIO.add_event_detect(BUTTON_BACK, GPIO.BOTH,
           callback=callback_botao1, bouncetime=75)
211     GPIO.add_event_detect(BUTTON_NEXT, GPIO.BOTH,
           callback=callback_botao2, bouncetime=75)
212
213     try:
214         print('Pressione Ctrl+C para sair')
215         loop()
216     except KeyboardInterrupt:
217         destroy()
```

A.2 *Hardware*

A.2.1 POM

```
1
2 <?xml version="1.0" encoding="UTF-8"?>
3 <project xmlns="http://maven.apache.org/POM/4.0.0"
4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <groupId>br.ufscar.sead</groupId>
8     <artifactId>Celta</artifactId>
9     <version>1.0-SNAPSHOT</version>
10    <packaging>jar</packaging>
11    <properties>
12        <project.build.sourceEncoding>
13            UTF-8
14        </project.build.sourceEncoding>
15        <maven.compiler.source>17</maven.compiler.source>
16        <maven.compiler.target>17</maven.compiler.target>
17        <exec.mainClass>celta.FormCelta</exec.mainClass>
18    </properties>
19
20    <dependencies>
21        <dependency>
22            <groupId>org.apache.tika</groupId>
23            <artifactId>tika</artifactId>
24            <version>1.18</version>
25            <type>pom</type>
26        </dependency>
27
28        <dependency>
29            <groupId>org.apache.tika</groupId>
30            <artifactId>tika-core</artifactId>
31            <version>1.18</version>
32        </dependency>
33
34        <dependency>
35            <groupId>org.apache.tika</groupId>
```

```
36              <artifactId>tika-parsers</artifactId>
37              <version>1.18</version>
38          </dependency>
39      </dependencies>
40
41      <build>
42          <plugins>
43            <plugin>
44              <!-- Build an executable JAR -->
45              <groupId>org.apache.maven.plugins</groupId>
46              <artifactId>maven-jar-plugin</artifactId>
47              <version>3.1.0</version>
48              <configuration>
49                <archive>
50                  <manifest>
51                    <addClasspath>true</addClasspath>
52                    <classpathPrefix>lib/</classpathPrefix>
53                    <mainClass>celta.FormCelta</mainClass>
54                  </manifest>
55                </archive>
56              </configuration>
57            </plugin>
58          </plugins>
59      </build>
60  </project>
```

A.2.2 *Classe FormCelta*

```
1  package celta;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.PrintWriter;
7  import java.net.ServerSocket;
8  import java.net.Socket;
9  import javax.swing.ImageIcon;
10
11 public class FormCelta extends javax.swing.JFrame {
12
13     /**
14      * Creates new form FormCelta
```

```
15      */
16     public FormCelta() {
17         initComponents();
18     }
19
20     /**
21      * This method is called from within the constructor to
           initialize the form.
22      * WARNING: Do NOT modify this code. The content of this
           method is always
23      * regenerated by the Form Editor.
24      */
25     @SuppressWarnings("unchecked")
26     // <editor-fold defaultstate="collapsed" desc="Generated
          Code">
27     private void initComponents() {
28         imgUfscar = new javax.swing.JLabel();
29         imgCapes = new javax.swing.JLabel();
30         pnlClipboardArea = new javax.swing.JScrollPane();
31         txtClipboardArea = new javax.swing.JTextArea();
32         pnlButton = new javax.swing.JPanel();
33         btnRetry = new javax.swing.JButton();
34
35         setDefaultCloseOperation(
36             javax.swing.WindowConstants.EXIT_ON_CLOSE
37         );
38         setTitle("Celta - Celula Tatil UFSCAR");
39         setAlwaysOnTop(true);
40         addWindowListener(new java.awt.event.WindowAdapter()
               {
41             public void
                   windowClosing(java.awt.event.WindowEvent evt)
                   {
42                 formWindowClosing(evt);
43             }
44             public void
                   windowOpened(java.awt.event.WindowEvent evt) {
45                 formWindowOpened(evt);
46             }
47         });
48
49         imgUfscar.setIcon(
```

```
50              new javax.swing.ImageIcon(
51                  getClass().getResource("/images/ufscar.png")
52              )
53          ); // NOI18N
54          imgUfscar.setName("imgUfscar"); // NOI18N
55
56          imgCapes.setIcon(
57              new javax.swing.ImageIcon(
58                  getClass().getResource(
59                      "/images/capes.png"
60                  )
61              )
62          ); // NOI18N
63          imgCapes.setName("imgCapes"); // NOI18N
64
65          txtClipboardArea.setEditable(false);
66          txtClipboardArea.setColumns(20);
67          txtClipboardArea.setRows(5);
68          pnlClipboardArea.setViewportView(txtClipboardArea);
69
70          btnRetry.setText("Reconectar");
71          btnRetry.setEnabled(false);
72          btnRetry.setName(""); // NOI18N
73          btnRetry.addActionListener(new
               java.awt.event.ActionListener() {
74              public void
                   actionPerformed(java.awt.event.ActionEvent
                   evt) {
75                  btnRetryActionPerformed(evt);
76              }
77          });
78
79          javax.swing.GroupLayout pnlButtonLayout = new
               javax.swing.GroupLayout(pnlButton);
80          pnlButton.setLayout(pnlButtonLayout);
81          pnlButtonLayout.setHorizontalGroup(
82              pnlButtonLayout.createParallelGroup(
83                  javax.swing.GroupLayout.Alignment.LEADING
84              ).addGroup(
85                  javax.swing.GroupLayout.Alignment.TRAILING,
86                  pnlButtonLayout.createSequentialGroup()
87                  .addContainerGap(
```

```
88                      javax.swing.GroupLayout.DEFAULT_SIZE,
89                      Short.MAX_VALUE
90                  )
91                  .addComponent(btnRetry)
92                  .addContainerGap()
93                  )
94          );
95          pnlButtonLayout.setVerticalGroup(
96              pnlButtonLayout.createParallelGroup(
97                  javax.swing.GroupLayout.Alignment.LEADING
98              )
99              .addGroup(
100                 pnlButtonLayout.createSequentialGroup()
101                 .addContainerGap()
102                 .addComponent(
103                     btnRetry,
104                     javax.swing.GroupLayout.DEFAULT_SIZE,
105                     43,
106                     Short.MAX_VALUE
107                 ).addContainerGap()
108             )
109         );
110
111         javax.swing.GroupLayout layout = new
               javax.swing.GroupLayout(getContentPane());
112         getContentPane().setLayout(layout);
113         layout.setHorizontalGroup(
114             layout.createParallelGroup(
115                 javax.swing.GroupLayout.Alignment.LEADING
116             )
117             .addGroup(layout.createSequentialGroup()
118                 .addContainerGap()
119                 .addGroup(
120                     layout.createParallelGroup(
121                         javax.swing.GroupLayout.Alignment.LEADING
122                     )
123                     .addGroup(
124                         layout.createSequentialGroup()
125                         .addComponent(
126                             pnlClipboardArea,
127                             javax.swing.GroupLayout.DEFAULT_SIZE,
128                             588,
```

```
129                         Short.MAX_VALUE
130                     )
131                     .addContainerGap()
132                 ).addGroup(
133                 layout.createSequentialGroup()
134                     .addComponent(imgUfscar)
135                     .addGap(32, 32, 32)
136                     .addComponent(imgCapes)
137                     .addPreferredGap(
138                         javax.swing.LayoutStyle.
139                         ComponentPlacement.RELATED,
140                         javax.swing.GroupLayout.DEFAULT_SIZE,
141                         Short.MAX_VALUE
142                     )
143                     .addComponent(
144                     pnlButton,
145                     javax.swing.GroupLayout.PREFERRED_SIZE,
146                     javax.swing.GroupLayout.DEFAULT_SIZE,
147                     javax.swing.GroupLayout.PREFERRED_SIZE
148                     )
149                 ))
150             )
151         );
152         layout.setVerticalGroup(
153             layout.createParallelGroup(
154                 javax.swing.GroupLayout.Alignment.LEADING
155             )
156             .addGroup(
157                 layout.createSequentialGroup()
158                 .addContainerGap()
159                 .addComponent(
160                     pnlClipboardArea,
161                     javax.swing.GroupLayout.DEFAULT_SIZE,
162                     333,
163                     Short.MAX_VALUE
164                 )
165                 .addPreferredGap(
166                     javax.swing.LayoutStyle.
167                     ComponentPlacement.RELATED
168                 )
169                 .addGroup(layout.createParallelGroup(
170                     javax.swing.GroupLayout.Alignment.LEADING
```

```
171                 )
172                 .addComponent(imgCapes)
173                 .addComponent(
174                     pnlButton,
175                     javax.swing.GroupLayout.PREFERRED_SIZE,
176                     javax.swing.GroupLayout.DEFAULT_SIZE,
177                     javax.swing.GroupLayout.PREFERRED_SIZE
178                 )
179                 .addComponent(imgUfscar))
180                 .addGap(0, 0, 0)
181             )
182         );
183 
184         pack();
185     }// </editor-fold>
186 
187     private void formWindowOpened(java.awt.event.WindowEvent
        evt) {
188         this.hideButton();
189         raspberry = new Thread(new
            Raspberry(this.txtClipboardArea));
190         raspberry.start();
191     }
192 
193     private void
        formWindowClosing(java.awt.event.WindowEvent evt) {
194         this.raspberry.interrupt();
195     }
196 
197     private void
        btnRetryActionPerformed(java.awt.event.ActionEvent
        evt) {
198         this.hideButton();
199         this.raspberry = new Thread(new
            Raspberry(this.txtClipboardArea, this.btnRetry,
            this.pnlButton));
200     }
201 
202     public void hideButton() {
203         this.btnRetry.setVisible(false);
204         this.btnRetry.setEnabled(false);
205         this.pnlButton.revalidate();
```

```
206         this.pnlButton.repaint();
207 
208         this.revalidate();
209         this.repaint();
210     }
211 
212     /**
213      * @param args the command line arguments
214      */
215     public static void main(String args[]) {
216         /* Set the Nimbus look and feel */
217         //<editor-fold defaultstate="collapsed" desc=" Look
                and feel setting code (optional) ">
218         /* If Nimbus (introduced in Java SE 6) is not
                available, stay with the default look and feel.
219          */
220         try {
221             for (javax.swing.UIManager.LookAndFeelInfo info
                    :
                    javax.swing.UIManager.getInstalledLookAndFeels())
                    {
222                 if ("Nimbus".equals(info.getName())) {
223                     javax.swing.UIManager.setLookAndFeel(
224                         info.getClassName()
225                     );
226                     break;
227                 }
228             }
229         } catch (ClassNotFoundException ex) {
230             java.util.logging.Logger.getLogger(
231                 FormCelta.class.getName()
232             ).log(
233                 java.util.logging.Level.SEVERE, null, ex
234             );
235         } catch (InstantiationException ex) {
236             java.util.logging.Logger.getLogger(
237                 FormCelta.class.getName()
238             ).log(
239                 java.util.logging.Level.SEVERE, null, ex
240             );
241         } catch (IllegalAccessException ex) {
242             java.util.logging.Logger.getLogger(
```

```
243                 FormCelta.class.getName()
244             ).log(
245                 java.util.logging.Level.SEVERE, null, ex
246             );
247         } catch (javax.swing.UnsupportedLookAndFeelException
               ex) {
248             java.util.logging.Logger.getLogger(
249                 FormCelta.class.getName()
250             ).log(
251                 java.util.logging.Level.SEVERE, null, ex
252             );
253         }
254         //</editor-fold>
255 
256         /* Create and display the form */
257         java.awt.EventQueue.invokeLater(new Runnable() {
258             public void run() {
259                 new FormCelta().setVisible(true);
260             }
261         });
262     }
263 
264     public Thread raspberry;
265 
266     // Variables declaration - do not modify
267     private javax.swing.JButton btnRetry;
268     private javax.swing.JLabel imgCapes;
269     private javax.swing.JLabel imgUfscar;
270     private javax.swing.JPanel pnlButton;
271     private javax.swing.JScrollPane pnlClipboardArea;
272     private javax.swing.JTextArea txtClipboardArea;
273     // End of variables declaration
274 }
```

A.2.3 *Classe Utils*

```
1 package celta;
2 
3 import celta.reader.PDFReader;
4 import celta.reader.Reader;
5 import celta.reader.TXTReader;
6 import java.awt.Toolkit;
```

```
7  import java.awt.datatransfer.DataFlavor;
8  import java.awt.datatransfer.UnsupportedFlavorException;
9  import java.io.IOException;
10
11 public class Utils {
12
13     /**
14      * Metodo responsavel por copiar dados da area de
           transferencia
15      * @return Retorna uma string com os dados da area de
           transferencia
16      */
17     public static String copyClipboard() throws
           UnsupportedFlavorException, IOException{
18         return (String) Toolkit.getDefaultToolkit()
19         .getSystemClipboard()
20         .getData(
21             DataFlavor.stringFlavor
22         );
23     }
24
25     public static boolean identifyWhenIsFile(String
           clipboard){
26         return
              clipboard.matches("[a-zA-Z]:[\\\\\\/](?:[a-zA-Z0-9]+
27         [\\\\\\/])*([a-zA-Z0-9]+\\.[A-Za-z]{3}$)");
28     }
29
30     public static Reader typeOfFile(String pathFile) {
31         if(pathFile.isEmpty()){
32             return null;
33         }
34
35         String format = pathFile.substring(pathFile.length()
              - 3, pathFile.length());
36
37         switch(format){
38             case "txt":
39                 return new TXTReader(pathFile);
40             case "pdf":
41                 return new PDFReader(pathFile);
42             default:
```

```
43                  return null;
44          }
45
46      }
47  }
```

A.2.4 Classe Comunicacao

```
1  package celta;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStream;
6  import java.io.InputStreamReader;
7  import java.io.OutputStream;
8  import java.net.Socket;
9  import javax.swing.JButton;
10 import javax.swing.JOptionPane;
11
12 public class Comunicacao {
13
14     private OutputStream serialOut;
15     private InputStream serialIn;
16     private final JButton btnReconectar;
17     Socket socket;
18
19     InputStreamReader inputStreamReader;
20     BufferedReader bufferedReader;
21
22     /**
23      * Construtor da classe Comunicacao
24      *
25      * @param btnReconectar Botao responsavel por ativar o
            comando de reconectar
26      */
27     public Comunicacao(JButton btnReconectar) {
28         this.btnReconectar = btnReconectar;
29         this.initialize();
30     }
31
32     /**
```

```
33      * Metodo que verifica se a comunicacao com o hardware
            esta funcionando
34      */
35     private void initialize() {
36         try {
37             this.socket = new Socket("192.168.15.30", 50000);
38             this.serialOut = socket.getOutputStream();
39             this.serialIn = socket.getInputStream();
40
41             this.inputStreamReader = new
                   InputStreamReader(this.serialIn);
42             this.bufferedReader = new
                   BufferedReader(this.inputStreamReader);
43
44         } catch (IOException e) {
45             this.btnReconectar.setVisible(true);
46             this.btnReconectar.setEnabled(true);
47
48             JOptionPane.showMessageDialog(null, "Nao foi
                   possivel conectar ao Hardware. ",
49                     "Problema de conexao com o Celta",
                           JOptionPane.PLAIN_MESSAGE);
50             e.printStackTrace();
51         }
52     }
53
54     /**
55      * Metodo que encerra a comunicacao
56      */
57     public void close() {
58         try {
59
60             if (this.serialOut != null) {
61                 this.serialOut.close();
62             }
63
64             if (this.serialIn != null) {
65                 this.serialIn.close();
66             }
67
68             if (this.socket != null) {
69                 this.socket.close();
```

```
70              }
71
72          } catch (IOException e) {
73              e.printStackTrace();
74          }
75      }
76
77      /**
78       * Metodo que realiza o envio de dados via socket
79       *
80       * @param opcao - Valor a ser enviado pela porta serial
81       */
82      public void enviaDados(String opcao) {
83
84          if (this.serialOut != null) {
85              try {
86                  if (this.serialOut != null) {
87                      this.serialOut.write(opcao.getBytes());
88                      return;
89                  }
90
91              } catch (IOException ex) {
92                  ex.printStackTrace();
93                  JOptionPane.showMessageDialog(null, "Nao foi
                        possivel enviar o dado. ",
94                              "Enviar dados",
                                JOptionPane.PLAIN_MESSAGE);
95              }
96          }
97      }
98
99      /**
100      * Metodo que obtem dados recebidos via socket
101      *
102      * @return string recebido via comunicacao socket
103      */
104     public String lerDados() {
105
106         if (this.bufferedReader != null) {
107
108             try {
109                 String mensagemDoServidor;
```

```
110
111                 while ((mensagemDoServidor =
                        bufferedReader.readLine()) != null) {
112                     System.out.println("Mensagem recebida: "
                            + mensagemDoServidor);
113                     return mensagemDoServidor;
114                 }
115
116             } catch (IOException ex) {
117                 ex.printStackTrace();
118             }
119         }
120
121         return null;
122     }
123 }
```

A.2.5 Classe *Raspberry*

```
1  package celta;
2
3  import celta.reader.Reader;
4  import javax.swing.JTextArea;
5  import javax.swing.JButton;
6  import javax.swing.JPanel;
7
8  public class Raspberry implements Runnable {
9
10     private final Comunicacao controlePorta;
11     private final JTextArea textArea;
12     JPanel pnlButton;
13
14     private int atual = -1;
15     private String texto = "";
16
17     /**
18      * Construtor da classe Raspberry
19      *
20      * @param textArea Text area com o texto a ser exibido
            pelo raspberry
21      * @param btnReconectar Botao responsavel por tentar
            reconectar o socket
```

```
22      * @param pnlButton Painel que contem o botao
23      */
24     public Raspberry(JTextArea textArea, JButton
           btnReconectar, JPanel pnlButton) {
25         this.textArea = textArea;
26         this.pnlButton = pnlButton;
27         this.controlePorta = new Comunicacao(btnReconectar);
28     }
29 
30     /**
31      * Metodo responsavel por executar a thread de conexao
           com o raspberry
32      */
33     @Override
34     public void run() {
35 
36         this.clearHardware();
37         this.copyValuesClipboard();
38 
39         while (!Thread.currentThread().isInterrupted()) {
40             String dado = this.controlePorta.lerDados();
41 
42             if(dado != null){
43                 this.execCmd(dado);
44             } else {
45                 this.pnlButton.revalidate();
46                 this.pnlButton.repaint();
47                 Thread.currentThread().interrupt();
48             }
49 
50         }
51 
52         this.clearHardware();
53         this.controlePorta.close();
54     }
55 
56     /**
57      * Altero o texto que sera lido pelo raspberry
58      *
59      * @param texto
60      */
61     public void setText(String texto) {
```

```
62          this.atual = -1;
63
64          this.texto = this.convertTextToBraile(texto);
65          this.textArea.setText(this.texto);
66      }
67
68      private void atualizaCursor() {
69          this.textArea.setSelectionStart(atual);
70          this.textArea.setSelectionEnd(atual + 1);
71      }
72
73      /**
74       * Metodo responsavel por resetar possiveis pinos ativos
75       */
76      private void clearHardware() {
77          this.controlePorta.enviaDados("000000");
78      }
79
80      /**
81       * Metodo responsavel por emitir som no hardware
82       */
83      private void acionarBeep() {
84          this.controlePorta.enviaDados("17");
85      }
86
87      /**
88       * Metodo responsavel por executar os comandos de acordo
             com a opcao
89       * recebida pelo parametro
90       *
91       * @param value comando a ser executado
92       */
93      private void execCmd(String value) {
94          int aux;
95          switch (value) {
96              case "++1":
97                  aux = this.proximaPalavra(this.atual);
98
99                  if (aux != -1) {
100                     atual = aux;
101                     System.out.println("Avancar palavra");
102                 } else if (atual + 1 < this.texto.length()) {
```

```
103                     atual++;
104                     System.out.println("Avancar");
105                 }
106 
107                 this.controlePorta.enviaDados(
108                     this.escreveLetra(
109                         this.texto.charAt(this.atual)
110                     )
111                 );
112                 this.atualizaCursor();
113                 break;
114             case "+1":
115                 if (this.texto.length() == this.atual + 1) {
116                     this.acionarBeep();
117                     System.out.println("Ja estamos na ultima
                        letra");
118                     return;
119                 }
120 
121                 System.out.println("Avancar");
122                 this.controlePorta.enviaDados(
123                     this.escreveLetra(
124                         this.texto.charAt(++this.atual)
125                     )
126                 );
127                 this.atualizaCursor();
128 
129                 break;
130 
131             case "--1":
132                 aux = this.palavraAnterior(this.atual);
133 
134                 if (aux != -1) {
135                     this.atual = aux;
136                     System.out.println("Voltar palavra");
137                 } else if (this.atual > 0) {
138                     this.atual--;
139                 }
140 
141                 this.controlePorta.enviaDados(
142                     this.escreveLetra(
143                         this.texto.charAt(this.atual)
```

```
144                     )
145                 );
146                 this.atualizaCursor();
147                 break;
148
149             case "-1":
150
151                 if(this.atual < 0){
152                     this.acionarBeep();
153                     System.out.println("Leitura nao
                        iniciada");
154                     return;
155                 }
156
157                 if (this.atual == 0) {
158                     this.acionarBeep();
159                     System.out.println("Ja estamos na
                        primeira letra: acionar 2x beep");
160                     return;
161                 }
162
163                 System.out.println("Voltar");
164                 this.controlePorta.enviaDados(
165                     this.escreveLetra(
166                         this.texto.charAt(--this.atual)
167                     )
168                 );
169                 this.atualizaCursor();
170                 break;
171
172             case "++0":
173                 this.copyValuesClipboard();
174                 break;
175
176             case "clear":
177                 this.controlePorta.enviaDados(
178                     this.escreveLetra(' ')
179                 );
180                 break;
181             default:
182                 System.out.println(value);
183                 break;
```

```
184         }
185     }
186 
187     /**
188      * Metodo responsavel por traduzir nosso caracter para
           os pinos em braile
189      *
190      * @param letra letra no formato ASCII
191      * @return retorna string indicando quais pinos devem
           estar
192      * ativados/desativados
193      */
194     private String escreveLetra(char letra) {
195         System.out.println("Letra: " + letra);
196         switch (letra) {
197             case ' ':
198                 return "000000";
199             case '1':
200             case 'a':
201                 return "100000";
202             case '2':
203             case 'b':
204                 return "110000";
205             case '3':
206             case 'c':
207                 return "100100";
208             case '4':
209             case 'd':
210                 return "100110";
211             case '5':
212             case 'e':
213                 return "100010";
214             case '6':
215             case 'f':
216                 return "110100";
217             case '7':
218             case 'g':
219                 return "110110";
220             case '8':
221             case 'h':
222                 return "110010";
223             case '9':
```

```
224             case 'i':
225                 return "010100";
226             case 'O':
227             case 'j':
228                 return "010110";
229             case 'k':
230                 return "101000";
231             case 'l':
232                 return "111000";
233             case 'm':
234                 return "101100";
235             case 'n':
236                 return "101110";
237             case 'o':
238                 return "101010";
239             case 'p':
240                 return "111100";
241             case 'q':
242                 return "111110";
243             case 'r':
244                 return "111010";
245             case 's':
246                 return "011100";
247             case 't':
248                 return "011110";
249             case 'u':
250                 return "101001";
251             case 'v':
252                 return "111001";
253             case 'w':
254                 return "010111";
255             case 'x':
256                 return "101101";
257             case 'y':
258                 return "101111";
259             case 'z':
260                 return "101011";
261             case '\u02E9':
262                 return "001111";
263             case '\u035D':
264                 return "000101";
265             case '\u00E1':
```

```
266                 return "111011";
267             case '\u00E0':
268                 return "110101";
269             case '\u00E2':
270                 return "100001";
271             case '\u00e3':
272                 return "001110";
273             case '\u00e7':
274                 return "111101";
275             case '\u00E9':
276                 return "111111";
277             case '\u00EA':
278                 return "110001";
279             case '\u00ED':
280                 return "001100";
281             case '\u00F3':
282                 return "001101";
283             case '\u00F4':
284                 return "100111";
285             case '\u00F5':
286                 return "010101";
287             case '\u00FA':
288                 return "011111";
289             case '\u00FC':
290                 return "110011";
291             case '(':
292                 return "110001";
293             case ')':
294                 return "001110";
295             case '[':
296                 return "111011";
297             case ']':
298                 return "011111";
299             case ',':
300                 return "010000";
301             case ';':
302                 return "011000";
303             case ':':
304                 return "010010";
305             case '.':
306                 return "001000";
307             case '`':
```

```
308                 return "001000";
309             case '?':
310                 return "010001";
311             case '!':
312                 return "011010";
313             case '-':
314                 return "001001";
315             case '+':
316                 return "011010";
317             default:
318                 return "";
319         }
320     }
321 
322     /**
323      * Metodo responsavel por obter o texto a ser lido pelo
            raspberry. Se for o
324      * caminho de um arquivo ele ira abrir o conteudo do
            arquivo.
325      */
326     private void copyValuesClipboard() {
327         try {
328             String data = Utils.copyClipboard();
329             System.out.println("Copiado da area de
                transferencia: " + data);
330 
331             if (Utils.identifyWhenIsFile(data)) {
332                 System.out.println("Abrindo arquivo: " +
                    data);
333                 Reader text = Utils.typeOfFile(data);
334 
335                 if (text != null && text.fileExists()) {
336                     data = text.getText();
337                 }
338 
339                 System.out.println("Texto Colado: " + data);
340             }
341 
342             this.setText(data);
343         } catch (Exception ex) {
344 
345             for (int i = 0; i < 3; i++) {
```

```
346                 this.acionarBeep();
347             }
348         }
349     }
350 
351     private String convertTextToBraile(String texto) {
352         StringBuilder aux = new StringBuilder();
353 
354         for (int i = 0; i < texto.length(); i++) {
355 
356             //caracteres numericos
357             if (texto.charAt(i) >= 48 && texto.charAt(i) <=
                    57) {
358                 aux.append('\u02E9');
359                 aux.append(texto.charAt(i));
360                 continue;
361             } else {
362 
363                 //caracteres maiusculos
364                 if (texto.charAt(i) >= 65 && texto.charAt(i)
                        <= 90) {
365 
366                     aux.append('\u035D');
367                     aux.append(
368                         Character.toLowerCase(texto.charAt(i))
369                     );
370                     continue;
371                 }
372             }
373 
374             aux.append(texto.charAt(i));
375         }
376 
377         return aux.toString();
378     }
379 
380     private int proximaPalavra(int start) {
381         int atual = start;
382 
383         while ((atual = this.texto.indexOf(" ", atual)) !=
                -1) {
384 
```

```
385             if (atual + 1 < this.texto.length() &&
                    this.texto.charAt(atual + 1) != ' ') {
386                 return atual + 1;
387             }
388             atual++;
389         }
390
391         return -1;
392     }
393
394     private int palavraAnterior(int end) {
395         int atual = 0, anterior = 0;
396
397         String aux = this.texto.substring(0, end < 1 ? 0 :
                end - 1);
398         while ((atual = aux.indexOf(" ", atual)) != -1) {
399
400             if (atual + 1 < this.texto.length() &&
                    this.texto.charAt(atual + 1) != ' ') {
401                 anterior = atual + 1;
402             }
403             atual++;
404         }
405
406         return anterior;
407     }
408
409 }
```

A.2.6 Interface *Reader*

```
1  package celta.reader;
2
3  import java.io.File;
4  import java.io.FileInputStream;
5  import java.util.Scanner;
6
7  public abstract class Reader {
8      protected String pathFile;
9      protected File file;
10     protected FileInputStream input;
11
```

```
12      public Reader(String name){
13          this.pathFile = name;
14          this.file = new File(this.pathFile);
15      }
16
17      public boolean fileExists() {
18          return this.file.exists();
19      }
20
21      public String getText() throws Exception {
22          if(this.reader()){
23              String data;
24              try (Scanner s = new
                    Scanner(this.input).useDelimiter("\\A")) {
25                  data = s.hasNext() ? s.next() : "";
26                  s.close();
27              }
28
29              return data;
30          }
31
32          return "";
33      }
34
35      abstract protected boolean reader() throws Exception;
36  }
```

A.2.7 Classe TXTReader

```
1  package celta.reader;
2
3  import java.io.FileInputStream;
4  import java.io.FileNotFoundException;
5
6  public class TXTReader extends Reader {
7
8      public TXTReader(String name){
9          super(name);
10     }
11
12     @Override
13     protected boolean reader() throws FileNotFoundException{
```

```
14          this.input = new FileInputStream(this.file);
15          return true;
16      }
17  }
```

A.2.8 Classe PDFReader

```
1  package celta.reader;
2
3  import java.io.FileInputStream;
4
5  import org.apache.tika.metadata.Metadata;
6  import org.apache.tika.parser.ParseContext;
7  import org.apache.tika.parser.pdf.PDFParser;
8  import org.apache.tika.sax.BodyContentHandler;
9
10
11
12 /**
13  *
14  * @author MendeSantos
15  */
16 public class PDFReader extends Reader {
17
18      private BodyContentHandler ch;
19
20      public PDFReader(String name){
21          super(name);
22      }
23
24      @Override
25      public String getText() throws Exception {
26          if(this.reader()){
27              return ch.toString();
28          }
29
30          return "";
31      }
32
33      @Override
34      protected boolean reader() throws Exception {
35          this.input = new FileInputStream(this.file);
```

```
36          this.ch = new BodyContentHandler();
37
38          Metadata md = new Metadata();
39          ParseContext pc = new ParseContext();
40          PDFParser pp = new PDFParser();
41
42          pp.parse(this.input, ch, md, pc);
43          return true;
44      }
45  }
```

Referências

ABDALA, D. D. *Circuitos multiplexadores*. 2023. Universidade Federal de Uberlândia - UFU. Disponível em: <https://www.facom.ufu.br/~abdala/sd/mat/multiplexadores.pdf>. Citado na página 60.

ABOUT us. 2023. Raspberry Pi Foundation. Disponível em: <https://www.raspberrypi.org/about/>. Citado na página 34.

APACHE MAVEN PROJECT. *Welcome to Apache Maven*. 2023. Disponível em: <https://maven.apache.org/>. Citado na página 51.

ARAKAKI, J. *Programação com "Sockets"*. 2023. PUC-SP. Disponível em: <https://www.pucsp.br/~jarakaki/lp4/ProgramacaoSockets.pdf>. Citado na página 33.

BRITANNICA DO BRASIL. *Enciclopédia Mirador Internacional*. 1987. Citado na página 19.

CIRIACO, D. *O que é Raspberry Pi?* 2015. Canaltech. Disponível em: <https://canaltech.com.br/hardware/o-que-e-raspberry-pi/>. Citado na página 34.

COMPONENTS 101. *L293D Motor Driver IC*. [S.l.], 2023. L293D Pin Configuration. Disponível em: <https://components101.com/ics/l293d-pinout-features-datasheet>. Citado 2 vezes nas páginas 40 e 41.

CONSELHO REGIONAL DE BIOMEDICINA - 3ª REGIÃO. *Tecnologia Assistiva: Sugestões de leitores de tela para deficientes visuais*. 2021. Disponível em: <https://www.crbm3.gov.br/ajuda/acessibilidade>. Citado na página 60.

FREEPIK. *Copos Plásticos Presos com Barbante*. 2023. JPG. Fonte: Freepik.com. Disponível em: <https://br.freepik.com/fotos-gratis/copos-plasticos-presos-com-barbante_11684275.htm#query=lata%20com%20fio%20de%20barbante&position=10&from_view=search&track=ais>. Citado na página 31.

FROMAGET, P. *15 Best Operating Systems for Raspberry Pi (with pictures)*. 2019. Disponível em: <https://raspberrytips.com/best-os-for-raspberry-pi/>. Citado na página 37.

INSTRUCTOR Braille. 2023. Disponível em: <http://instructorbraille.blogspot.com/p/el-codigo-braille.html>. Citado na página 20.

JÚNIOR, E. S. Modelo TCP/IP. 2018. Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro. Disponível em: <https://esj.eti.br/IFTM/Disciplinas/Grau02/RC/RC_Unidade_03.pdf>. Citado na página 28.

KALITA, L. Socket programming. *International Journal of Computer Science and Information Technologies*, Citeseer, v. 5, n. 3, p. 4802–4807, 2014. Disponível em: <https://ijcsit.com/docs/Volume%205/vol5issue03/ijcsit20140503462.pdf>. Citado 4 vezes nas páginas 29, 30, 31 e 32.

LOJA CIVIAM. *Linha Braille - DIsplay Braille Seika V3 Pro*. 2023. Disponível em: <https://www.lojaciviam.com.br/linha-braille-display-braille-seika-v3-pro>. Citado na página 20.

MERCADO LIVRE. *Placa De Circuito Impresso Pcb Azul Fibra 10x15 2188 Furos*. 2023. Disponível em: <https://produto.mercadolivre.com.br/MLB-2784427110-placa-de-circuito-impresso-pcb-azul-fibra-10x15-2188-furos-_JM?quantity=1>. Citado na página 39.

MERCADO LIVRE. *Raspberry Pi 1 B+ Sem Caixa*. 2023. Disponível em: <https://produto.mercadolivre.com.br/MLB-2853858842-raspberry-pi-1-b-sem-caixa-_JM?matt_tool=18956390&utm_source=google_shopping&utm_medium=organic>. Citado na página 57.

MERCADO LIVRE. *Raspberry Pi4 Pi 4 Model B 4gb Ram Novo Na Caixa C/ Manual*. 2023. Disponível em: <https://produto.mercadolivre.com.br/MLB-3345104417-raspberry-pi4-pi-4-model-b-4gb-ram-novo-na-caixa-c-manual-_JM#position=9&search_layout=grid&type=item&tracking_id=6766b3d8-e182-4682-8ce2-79f0319d51ae>. Citado na página 57.

RASPBERRY PI. *Raspberry Pi 3 Model B*. [S.l.], 2017. Disponível em: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b/>. Citado 2 vezes nas páginas 35 e 36.

RASPBERRY PI. *Raspberry Pi Documentation*. 2023. Disponível em: <https://www.raspberrypi.com/documentation/computers/os.html>. Citado na página 36.

RASPBERRY PI. *Raspberry Pi OS*. 2023. Disponível em: <https://www.raspberrypi.com/software/>. Citado na página 42.

SANTIAGO, G. et al. *CELTA: Sistema de célula tátil para leitura Braille*. São Carlos, 2020. Disponível em: <https://celta.ufscar.br/wp-content/uploads/2020/04/Manual-de-confecÃğÃčo-Celta.pdf>. Citado 4 vezes nas páginas 20, 21, 39 e 42.

SECRETARIA GERAL DE EDUCAÇÃO A DISTÂNCIA, SEAD. 2018. Disponível em: <https://www.sead.ufscar.br/pt-br/acoes-e-servicos/acoes-1/projeto-celta>. Citado na página 21.

TANENBAUM, A. S.; WETHERALL, D. *Redes de Computadores*. 5. ed. São Paulo: Pearson-Prentice Hall, 2011. Citado na página 27.

TINKERCAD. *Fácil de todas as maneiras*. 2023. Disponível em: <https://www.tinkercad.com/circuits>. Citado 4 vezes nas páginas 40, 41, 43 e 44.

TORRES, E. F.; MAZZONI, A. A.; ALVES, J. B. D. M. A acessibilidade à informação no espaço digital. 2002. Disponível em: <https://doi.org/10.1590/S0100-19652002000300009>. Citado na página 19.