UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

Lucas Serafim Morelato

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE SOFTWARE E INTERFACE GRÁFICA PARA COMANDAR UM MONOCICLO ASSISTIVO MOTORIZADO E ESTÁTICO

Vitória-ES

Dezembro/2018

Lucas Serafim Morelato

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE SOFTWARE E INTERFACE GRÁFICA PARA COMANDAR UM MONOCICLO ASSISTIVO MOTORIZADO E ESTÁTICO

Parte manuscrita do Projeto de Graduação do aluno Lucas Serafim Morelato, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Vitória-ES

Dezembro/2018

Lucas Serafim Morelato

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE SOFTWARE E INTERFACE GRÁFICA PARA COMANDAR UM MONOCICLO ASSISTIVO MOTORIZADO E ESTÁTICO

Parte manuscrita do Projeto de Graduação do aluno Lucas Serafim Morelato, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Prof. Dr. Teodiano Bastos Filho
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. André Ferreira
Universidade Federal do Espírito Santo

Prof.ª Dra. Eliete Maria de Oliveira Caldeira
Universidade Federal do Espírito Santo

Vitória-ES

Dezembro/2018

RESUMO

Lesões na medula espinhal e AVC causam sérios danos que podem provocar a paralisia, completa ou incompleta, dos membros do corpo humano. Novas terapias alternativas com tecnologias emergentes surgem em busca de melhores resultados na recuperação dessas pessoas. Uma pesquisa em desenvolvimento no Núcleo de Tecnologia Assistiva (NTA) da Universidade Federal do Espírito Santo (UFES), motivada por resultados impressionantes de pesquisas anteriores, estuda a reabilitação de pacientes com paralisia nos membros inferiores, causadas por lesão na medula, com a utilização de sinais cerebrais (Eletroencefalograma - EEG) realimentando uma máquina assistiva e um avatar em ambiente de realidade virtual para simular o movimento imaginado pela pessoa. Este trabalho propõe o desenvolvimento do sistema de controle da máquina assistiva com a implementação de um *software* e de uma interface gráfica do usuário.

Palavras-chave: Reabilitação; Banco de Dados; Python; Interface; Máquina Assistiva

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

UFES	Universidade Federal do Espírito Santo
NTA	Núcleo de Tecnologia Assistiva
EEG	Eletroencefalografia
LME	Lesão da Medula Espinhal
ASIA	*American Spinal Injury Association*
VR	Realidade Virtual
RPi	*Raspberry Pi*
GUI	*General User Interface*
SQL	*Structured Query Language*
HCI	*Human-Computer Interaction*
IMU	*Inertial Measurement Unit*

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

A medula espinhal é uma massa cilindroide de tecido nervoso situada dentro do canal vertebral (MACHADO, 1993). Esta massa é um composto de células nervosas que compreendem parte do sistema nervoso central. Verticalmente localizada entre o bulbo cerebral e a vértebra lombar, a medula espinhal apresenta um formato de cordão cilíndrico não uniforme que proporciona a comunicação entre o cérebro e o corpo. Apesar de protegida no interior da coluna vertebral, acidentes originados por compulsões, incisões ou contusões podem acarretar lesões ao longo da extensão da medula.

A lesão da medula espinhal (LME) consiste no dano em alguma parte da medula espinhal que pode ocasionar sérias deficiências. De acordo com a Organização Mundial da Saúde, um estudo publicado em Genebra em 2013 mostrou que todo ano cerca 500 mil pessoas no mundo sofrem uma LME, e 90% das lesões são causadas por traumas que são passíveis de serem prevenidos como, por exemplo, acidentes de carro (WORLD HEALTH ORGANIZATION, 2013).

De acordo com a gravidade da lesão, a LME pode ser dividida em dois tipos conhecidos como (WORLD HEALTH ORGANIZATION, 2013):

- Completa: provoca perda total da capacidade sensitiva e função motora abaixo do nível da lesão na medula espinhal.
- Incompleta: provoca perda parcial, ou seja, ainda há alguma função motora ou sensora abaixo do nível da lesão neurológica.

A classificação da LME é realizada pela *American Spinal Injury Association - ASIA*, conforme o Quadro 1. A localidade da LME, em relação à posição na vértebra, determina as partes do corpo humano onde perdas sensoriais e motoras podem ocorrer. Desta forma, existem duas classificações (WORLD HEALTH ORGANIZATION, 2013):

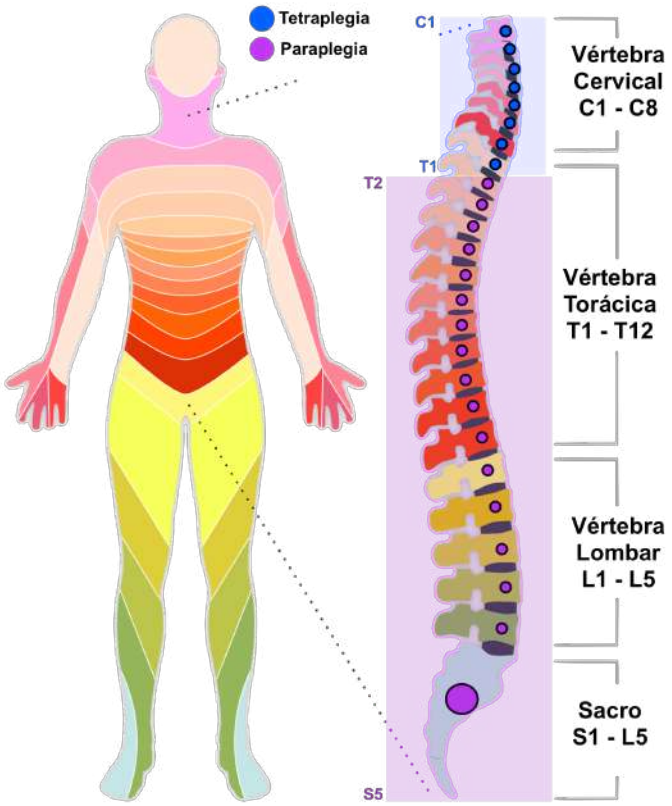
- Paraplegia: ocorre quando a lesão atinge um nível na medula espinhal na qual perdas sensoriais e motoras ocorrem, em diversos níveis de gravidades, no tronco e nas partes inferiores do corpo, livrando de danos os membros superiores do corpo como braços e ombros, por exemplo. São as lesões que atingem a área torácica (T2-T12), lombar (L1-L5) ou sacral (S1-S5) como representado na Figura 1 pelos círculos roxos.
- Tetraplegia: causada por uma LME que provoca perdas sensoriais e motoras, em diversos níveis, nas partes superiores e inferiores do corpo, incluindo o tronco e o

Quadro 1 – Classificações das lesões da medula espinhal segundo ASIA.

Classificação	Descrição
ASIA A - Completa	Falta total de funções motoras e sensoriais abaixo do nível da lesão, incluindo a região anal
ASIA B - Incompleta	Alguma capacidade de sensoriamento abaixo do nível da lesão, incluindo a região anal
ASIA C - Incompleta	Alguns movimentos musculares, mas mais de 50% dos músculos abaixo do nível da lesão estão impossibilitados de se mover contra a gravidade
ASIA D - Incompleta	Mais de 50% dos músculos abaixo do nível da lesão conseguem se mover contra a gravidade
ASIA E - Normal	Todas as funções motoras e sensoriais estão normais

Fonte: Produção do próprio autor.

Figura 1 – Medula espinhal e região de paraplegia e tetraplegia nas vértebras lesionadas.



Fonte: THE BACK UP TRUST (2017) - Adaptada pelo autor.

pescoço. Essas lesões atingem os segmentos cervicais da medula espinhal (C1-T1) como representado na Figura 1 pelos círculos azuis.

A paraplegia, tetraplegia, as perdas ou alterações motoras podem ser sequelas também de acidentes vasculares cerebral (AVC). O AVC é um processo súbito que ocorre quando

as artérias que alimentam o cérebro sofrem uma obstrução ou uma ruptura, provocando a morte do tecido cerebral que por ela era alimentado. O AVC é Isquêmico quando essa artéria sofre uma obstrução reduzindo subitamente o fluxo sanguíneo. Se a artéria sofrer um rompimento, o AVC é chamado de Hemorrágico e provoca uma hemorragia no interior do cérebro (NATIONAL HEART, LUNG, AND BLOOD INSTITUTE, 2018).

A reabilitação de pessoas que sofreram algum tipo de perda ou alteração motora ainda é um desafio extremamente difícil para a Medicina, principalmente para os casos classificados como ASIA A. Estudos anteriores a 2016 que utilizaram apenas sistemas com equipamentos de suporte ao corpo humano, robôs assistivos e estimulação elétrica da perna, sem uma realimentação de sinais cerebrais, para assistir o andar de pacientes que sofreram uma LME, nunca apresentaram melhoras consistentes nas funções neurológicas (percepção sensorial de tato, temperatura, dor e o controle voluntário dos músculos) (DONATI et al., 2016).

Contudo, conforme explicado em (DONATI et al., 2016), estudos realizados inicialmente em ratos e macacos, e posteriormente em humanos, com a utilização de métodos que envolviam realimentação de sinais cerebrais com máquinas assistivas (*BMI: Brain-Machine Interface* - Interface Cérebro-Máquina), surgiram como potenciais opções para restaurar a mobilidade e a sensibilidade de pacientes que sofrem de perdas motoras ou sensoriais dos membros devido a acidentes na coluna vertebral ou doenças degenerativas.

Segundo (DONATI et al., 2016), estudos indicaram que de 60 a 80% dos pacientes que sofreram "lesão completa" na coluna vertebral ainda apresentam vestígios de axônios que ultrapassam o caminho cérebro-membro na altura da lesão da coluna. Desta forma, os axônios remanescentes podem trazer algum grau de melhora na recuperação neurológica desses pacientes.

As alterações motoras e sensitivas nas pessoas que sofreram sequelas pós AVC ou LME apresentam um impacto muito grande na vida pessoal. Ações que para muitos são simples como andar, vestir-se, alimentar-se e autocuidar-se ficam extremamente difíceis, ou até mesmo impossíveis, para pessoas que apresentam déficit no controle motor.

Uma pesquisa realizada em 2016 (DONATI et al., 2016) utilizou um sistema com interface entre o cérebro e uma máquina assistiva destinado a restabelecer a mobilidade em oito indivíduos paralíticos (LME com pelo menos 3 anos), mostrando resultados motivadores na reabilitação motora e sensorial. Com uma duração de 12 meses, o estudo realimentou uma máquina assistiva e um óculos VR com sinais cerebrais ECG para realizar a movimentação dos membros inferiores e animar um avatar do paciente na realidade virtual de acordo

com a intenção de executar o movimento dos músculos.

Como resultado, todos os oito indivíduos apresentaram melhoras neurológicas no sensoriamento (localização de dor e toque) das áreas do corpo atingidas pela paralisia (inferior à altura da lesão na coluna). Os indivíduos também conseguiram restabelecer movimentos voluntários dos músculos, permitindo uma melhora no índice de caminhada (*The Walking Index*, em inglês), índice composto por 19 níveis que capturam a extensão e a natureza da assistência para caminhar em pessoas com lesões na medula espinhal (DITUNNO JR et al., 2013). Por fim, metade dos indivíduos evoluíram na classificação ASIA, saindo de lesão completa (ASIA A) para incompleta (ASIA B, C ou D), de acordo com o Quadro 1. Em resumo, esta técnica, promoveu:

- Melhora na sensibilidade de dor e toque nas áreas afetadas em todos os indivíduos.
- Evolução de lesão completa para incompleta na metade dos indivíduos.
- Restabelecimento voluntário dos músculos em alguns indivíduos.

O resultado obtido por essa pesquisa motiva buscar o potencial que este tipo de recuperação pode apresentar para esses pacientes. Desta forma, com o objetivo de promover um tratamento alternativo a pessoas com déficit motor nos membros inferiores, esse trabalho sugere-se em um sistema de baixo custo, com interface cérebro-máquina, realidade virtual e máquina assistiva, que está em desenvolvimento no Núcleo de Tecnologia Assistiva/UFES para estudar a reabilitação de indivíduos que sofreram a lesão na coluna ou sofreram sequela pós AVC, e que apresentam perda ou alteração motora.

2 OBJETIVOS

2.1 Objetivo geral

Desenvolvimento do sistema de controle de uma máquina assistiva, um monociclo estático e motorizado, que inclui um *software* programado na linguagem Python, interface gráfica para utilização do sistema, e *hardware* necessário para realizar a interface *software*-máquina no sistema de reabilitação que está em desenvolvimento no NTA/UFES.

2.2 Objetivos específicos

Como objetivos específicos, este Projeto de graduação contempla os seguintes pontos a seguir.

- Desenvolver o *software*, em Python, responsável pelo controle do monociclo e pela comunicação com periféricos necessários.
- Desenvolver a interface gráfica do programa.
- Implementar o banco de dados, MySQL, a ser utilizado para armazenar dados pertinentes ao *software*.

3 EMBASAMENTO TEÓRICO

O desenvolvimento do *software* para controle da máquina monociclo foi realizado em Python, linguagem frequentemente utilizada em microcomputadores embarcados, como o proposto para o projeto: RaspberryPi (RPi) Modelo 3. A RPi é responsável por enviar comandos ao *hardware* do monociclo para controlar a máquina e, também, em paralelo, fornecer uma interface gráfica (*Graphical User Interface* - GUI) para o usuário do sistema.

Para o controle de acesso ao programa, um banco de dados MySQL é utilizado. Devido à necessidade de execução de códigos que se relacionam e que precisam ser executados simultaneamente, o conceito de *threading* é aplicado no *software*.

3.1 Python

Python é uma programação de linguagem de alto nível, interpretada, interativa e orientada a objeto. É uma linguagem conhecida por ser potente em resolução de problemas com uma sintaxe muito limpa e clara. Incorpora módulos, exceções, estruturas de dados de alto nível e classes (PYTHON SOFTWARE FOUNDATION, 2017).

A filosofia do Python é realizar ações de forma clara e simples com uma abordagem direta na escrita do programa. Além disso, a linguagem foi desenvolvida para ser intuitiva, com sintaxe próxima da língua inglesa e com pouco uso de pontuações (FOURMENT; GILLINGS, 2008). Assim, uma programação realizada em Python favorece a legibilidade do código fonte, o que torna a linguagem mais produtiva.

Além disso, a linguagem apresenta estruturas de alto nível (como listas, dicionários, data/hora, complexos e outras) e uma vasta coleção de módulos prontos para uso (BORGES, 2010). Esses recursos agilizam muito na execução de um programa que precisa trabalhar com os dados de uma maneira simples e produtiva.

Segundo (BORGES, 2010), a linguagem foi criada em 1990 por Guido Van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda e tinha originalmente foco em usuários como físicos e engenheiros. O Python foi concebido a partir de outra linguagem existente na época, chamada ABC. Atualmente, Python é utilizada em diversas empresas internacionais de alta tecnologia como Google, Yahoo, Microsoft e até a Disney nas animações 3D.

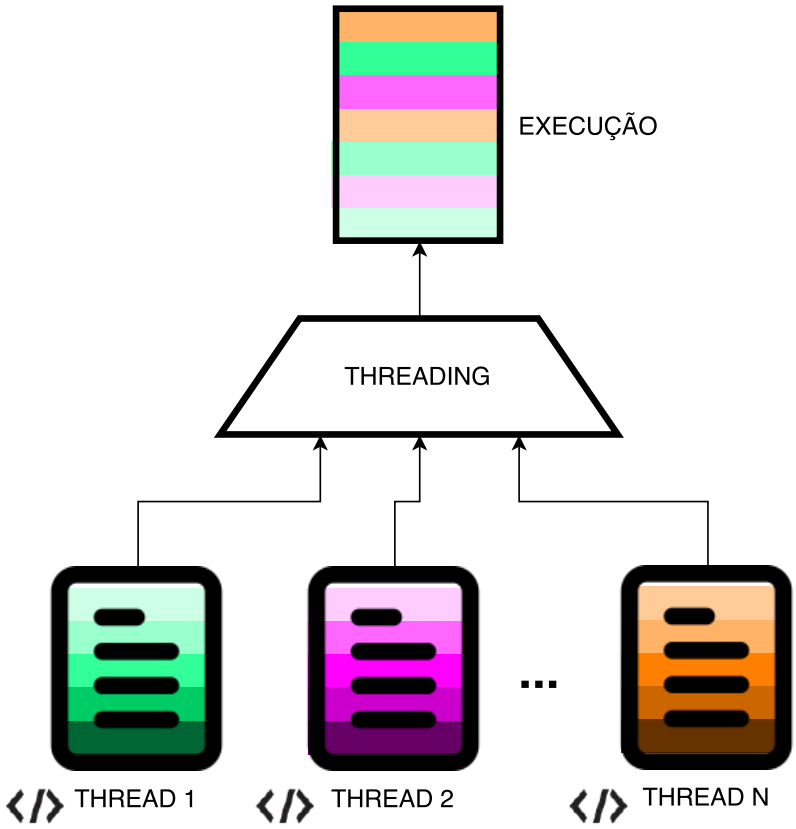
Para maiores informações a respeito da linguagem e suas estruturas, consulte o Apêndice

A deste documento.

3.2 Threading

Para a execução concorrente dos códigos no projeto, a programação por *threading* é realizada no *software* desenvolvido. *Threading* é uma adaptação direta do modelo sequencial de computação para sistemas paralelos (LEE, 2006). O modo de programação de *threads* consiste na execução de códigos que são gerenciados por um agendador que define a ordem e o tempo de execução de pequenas partes de cada *thread*, criando uma ilusão de programação paralela entre as tarefas. Um diagrama básico do funcionamento pode ser visto na Figura 2 representando a execução de partes das *threads*, de acordo com um agendamento.

Figura 2 – Diagrama do funcionamento básico de *threading*.



Fonte: Produção do próprio autor.

Conforme (BORGES, 2010), a utilização de *threads* oferece diversas vantagens quando comparadas aos processos convencionais:

Quadro 2 – Quadro exemplo com tabela de dados.

Nome	Número	Tipo
Lucas	27909090909	Móvel
Pedro	27999990909	Móvel
João	2730909090	Fixo

Fonte: Produção do próprio autor.

- Consomem menos recursos de máquina.
- Podem ser criadas e destruídas mais rapidamente.
- Podem ser chaveadas entre si mais rapidamente.
- Podem se comunicar com outras *threads* de forma mais fácil.

A justificativa para o uso de *threads* neste projeto se dá pela necessidade de realizar processamento paralelo. Como o projeto apresenta uma interface, o uso de *threading* permite que o usuário continue interagindo com a interface enquanto a aplicação realiza o controle da máquina assistiva.

3.3 Banco de dados

Banco de dados é uma coleção de informações organizadas e indexadas em linhas e colunas de tabelas que podem ser facilmente acessadas, gerenciadas e atualizadas (ROUSE, 2017). Uma tabela deve possuir um nome e o tipo de dado que será armazenado na coluna. Para cada linha inserida, tem-se um registro com seus atributos (um atributo para cada coluna), por exemplo, uma tabela de telefones com os atributos: número, o nome da pessoa e o tipo (fixo ou móvel). Assim, cada linha ou registro apresentará o telefone com seu respectivo dono e o tipo, conforme o Quadro 2.

O banco de dados escolhido para a realização deste projeto é o MySQL, capaz de fornecer um servidor de banco de dados SQL (Linguagem de Consulta Estruturada ou *Structured Query Language*), muito rápido e robusto, com múltiplos usuários e *threads* (ORACLE CORPORATION, 2017a). Por meio da linguagem SQL, as tabelas podem ser criadas, editadas e atualizadas. Cada tabela apresenta um conjunto de informações pertinentes a uma atividade como, por exemplo, usuários e senhas para acesso de um programa. Diferentes tabelas podem se relacionar entre si, de acordo com os dados que apresentam em comum, gerando um banco de dados com tabelas interligadas.

3.4 Interface

A interface, conhecida como Interface Gráfica do Usuário (*Graphical User Interface* - GUI), é um modo do homem interagir com a máquina de maneira gráfica, a qual utiliza janelas, ícones e menus que podem ser manipulados por dispositivos como, por exemplo, o mouse (THE LINUX INFORMATION PROJECT, 2004). É importante que a interface de um programa seja bem estruturada e planejada, pois se a GUI não for fácil de aprender e de usar, o produto pode não aderir ao mercado (GUTIERREZ MIRANDA, 2011).

As interfaces gráficas geralmente utilizam a metáfora do *desktop*, um espaço em duas dimensões que apresenta janelas retangulares que representam aplicativos, documentos e outros. As janelas podem apresentar diversos elementos que podem ser do tipo controle, que são objetos que interagem com o usuário ou apresentam informações, ou do tipo *containers*, que servem como repositório para coleção de outros objetos (BORGES, 2010).

Desta forma, com todos seus elementos, a interface gráfica interage com o usuário esperando e respondendo por eventos de acordo com seus objetos. Esses eventos surgem dessa interação e podem ser através de alguma informação digitada ou de algum botão pressionado, por exemplo. Para tratar esses eventos, os elementos da interface devem ser associados a funções chamadas de *callback*, as quais são executadas toda vez que um evento específico associado ao elemento ocorrer.

De acordo com (BORGES, 2010), os controles e *containers* mais usados são:

Controles:

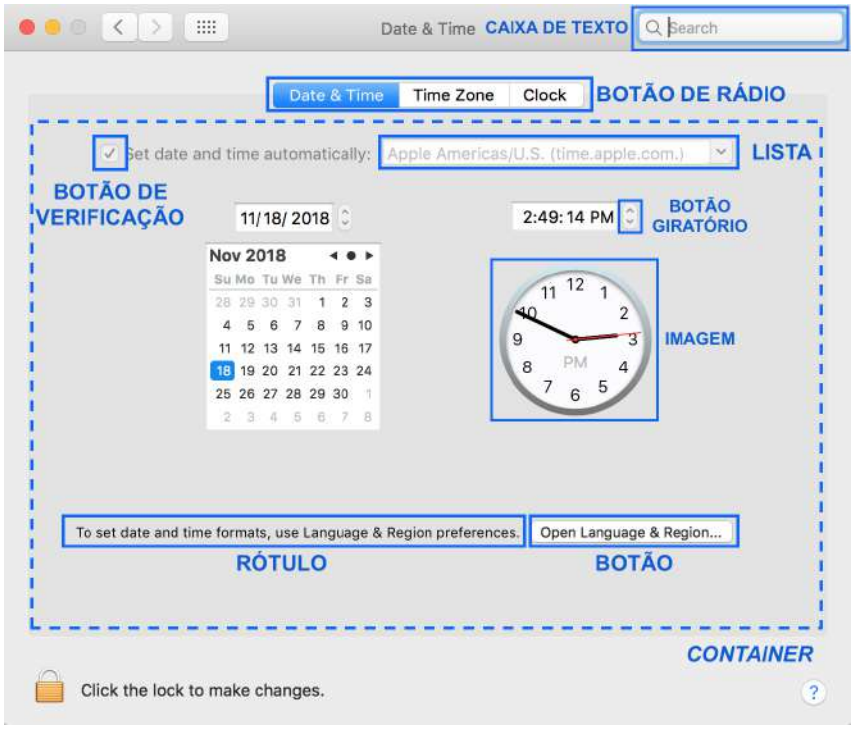
- Rótulo: retângulo que exibe texto.
- Caixa de texto: área para inserir texto.
- Botão: elemento interativo que pode ser clicado.
- Botão de rádio: um conjunto de botões, no qual apenas um pode ser ativo de cada vez.
- Botão de verificação: tipo de botão para ser marcado ou não.
- Botão giratório: composto por uma caixa de texto que tem seu valor alterado por dois botões (setas) que aumentam e diminuem o valor da caixa.
- Imagem: área destinada para imagens.

Containers:

- Barra de menu: área de menu normalmente localizada na área superior.
- Caixas horizontais, verticais ou tabelas: compartimentos para fixar objetos em seu interior.
- Caderno: várias áreas que podem ser visualizadas, uma de cada vez, quando selecionadas através de abas, geralmente na parte superior.

A Figura 3 mostra alguns exemplos dos elementos citados acima, os quais são exibidos em uma janela simples de hora e data.

Figura 3 – Exemplificação elementos e *containers* em uma janela exemplo.



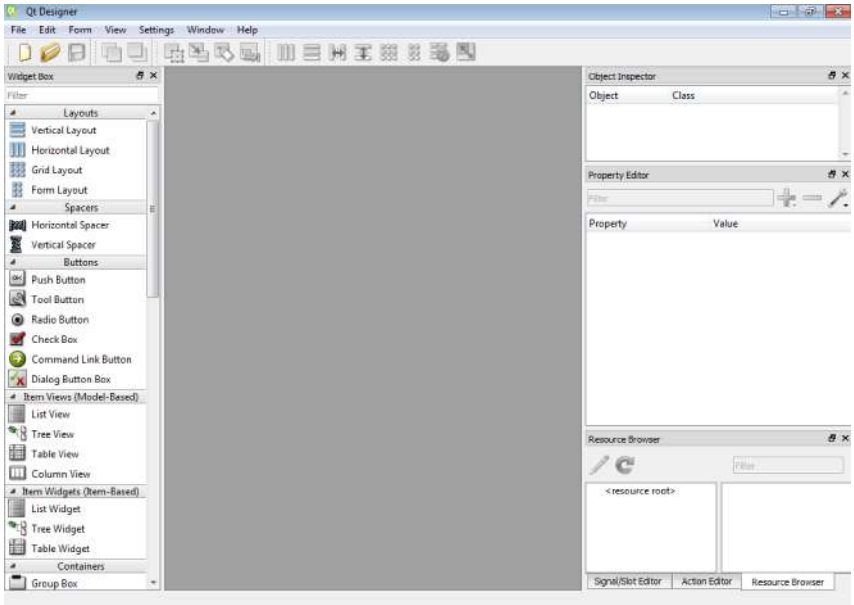
Fonte: Produção do próprio autor.

O Python apresenta o módulo chamado PyQT que permite a criação de interfaces realizando ligações das estruturas de aplicativos Qt da *The Qt Company*. Qt é conhecido por oferecer mais do que um simples conjunto de ferramentas para uma GUI, apresentando ligações com banco de dados SQL, navegadores de internet e uma vasta coleção de ferramentas para construir interfaces mais elaboradas (RIVERBANK COMPUTING, 2016), sendo mais usado para criação de aplicativos GUI (BORGES, 2010).

O módulo PyQT permite o uso do Qt no Python sob licença GPL. A interface em Qt pode ser diretamente programada por código em Python, porém, a ferramenta dispõe

do QtDesigner, um programa capaz de criar a interface de maneira gráfica, obtendo diretamente o resultado esperado sem ter que editar diretamente o código. Desta maneira, QtDesigner exporta o código Python a partir da interface criada no programa (RIVERBANK COMPUTING, 2016). Na Figura 4 a janela principal do programa QtDesigner está representada.

Figura 4 – *Software* QtDesigner para criação de interface gráfica.



Fonte: Produção do próprio autor.

Com o QtDesigner é possível utilizar os elementos (mostrados na coluna esquerda da Figura 4) e, literalmente, de forma gráfica, montar uma interface desejada. O *software* é capaz de transformar essa montagem em um arquivo XML que pode ser facilmente transformado em código Python utilizando o utilitário pyuic (BORGES, 2010), o que torna a criação de uma GUI no Python bastante prática. Consulte o Apêndice A para maiores detalhes e exemplos.

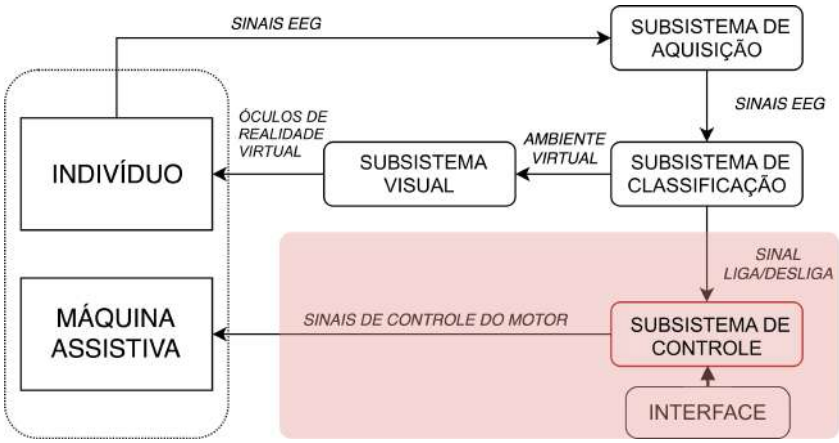
4 MATERIAIS E MÉTODOS

Este trabalho se caracteriza como projeto, isto é, tem o objetivo de criar e desenvolver um sistema capaz de atender as propostas discutidas neste documento.

Conforme discutido anteriormente, este projeto de graduação faz parte do desenvolvimento de um sistema em desenvolvimento no NTA da UFES. O sistema final apresenta um subsistema de aquisição de sinais EEG, os quais serão processados por um subsistema de classificação em um computador. O sub-sistema de classificação alimentará um subsistema visual composto por um óculos de realidade virtual que será usado pelo paciente, e o subsistema de classificação também será responsável por enviar comandos de ligar/desligar ao sub-sistema de controle de uma máquina assistiva, conforme mostrado na Figura 5.

Este projeto de graduação consiste no desenvolvimento do subsistema de controle para comandar a máquina assistiva por meio do sinal recebido pelo subsistema de classificação ou pelos comandos da interface desenvolvida. O método consiste em um *software* escrito em Python que gerencia a interface criada e o *hardware* desenvolvido para controlar a máquina assistiva (um monociclo motorizado estático). Desta forma, os comandos realizados na interface pelo operador são processados pelo *software* em um computador de placa única, a *Raspberry Pi 3*, que transfere sinais para a placa do *hardware* e aciona a máquina assistiva. Um banco de dados simples de uma única tabela no MySQL foi utilizado para acesso/controle de acesso ao *software* por meio de código de acesso e senha.

Figura 5 – Diagrama do método do sistema do NTA/UFES.



Fonte: Produção do próprio autor.

4.1 Raspberry Pi

Para o processamento do sub-sistema de controle foi utilizado o computador de bordo de placa única, Raspberry Pi modelo 3B (RPi3), do NTA da UFES. A escolha pela RPi3 se deu devido a sua praticidade, seu tamanho e sua capacidade de processar o necessário para o projeto. Por ter um tamanho físico reduzido, a RPi3 pode ser acoplada à máquina assistiva, promovendo uma maior portabilidade do projeto.

4.2 Máquina Assistiva

O projeto como um todo propõe como parte de atuação física no paciente apenas os membros inferiores. Desta forma, a escolha de uma máquina assistiva se deu por um monociclo que atua apenas com pedais para acionar o movimento das pernas do paciente. O monociclo utilizado no projeto é o ACTIVcycle da Paradigm Health Wellness, disponível no NTA.

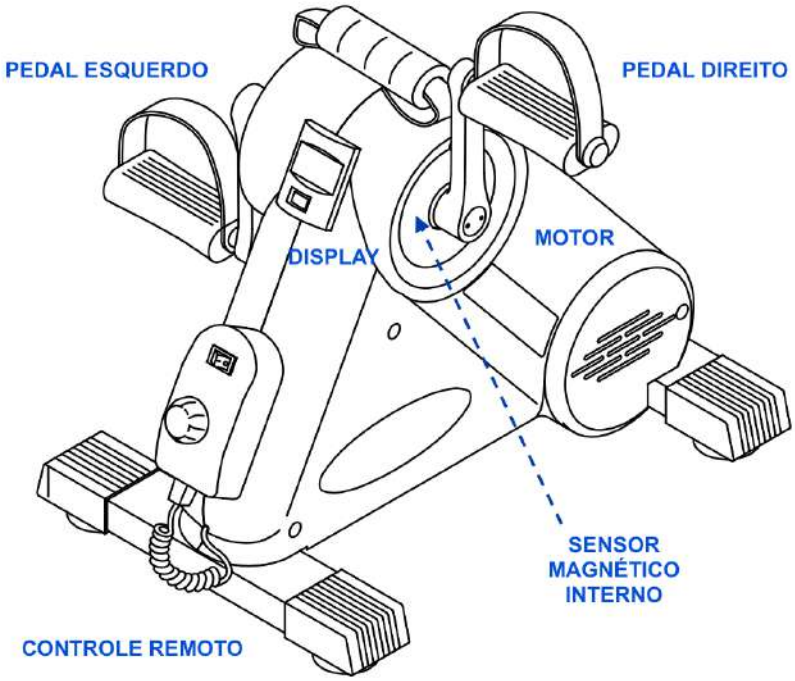
O monociclo apresenta um motor interno de corrente contínua de $6,5W$ de potência, um *display* acoplado que realiza contagem de tempo e voltas, pedais, um controle remoto para ligar a máquina e alterar sua velocidade, e um sensor magnético interno conforme a Figura 6.

O sensor magnético, acoplado no pedal direito no interior da máquina, permite realizar a contagem de voltas realizadas, o qual é composto por um contato normalmente aberto e um ímã que está acoplado ao pedal que gira conforme o movimento do mesmo. Quando o ímã passa pela chave que está fixa na estrutura do monociclo, o contato é fechado devido à intensidade local do campo magnético do ímã, conforme a Figura 7. A posição de contato fechado, ou seja, quando o ímã passa sobre os contatos, ocorre na posição de 90 graus vertical (representada na Figura 6) do pedal direito da máquina assistiva.

Removendo o *display* original da máquina assistiva (Figura 6) é possível ter acesso aos fios responsáveis pelo retorno do sensor magnético conforme o funcionamento explicado. Assim, se os dois fios estiverem com o mesmo potencial significa que o pedal direito está sobre ou passando pela posição vertical de 90 graus. Na Figura 8 os fios que retornam o *status* do sensor magnético na máquina assistiva podem ser vistos.

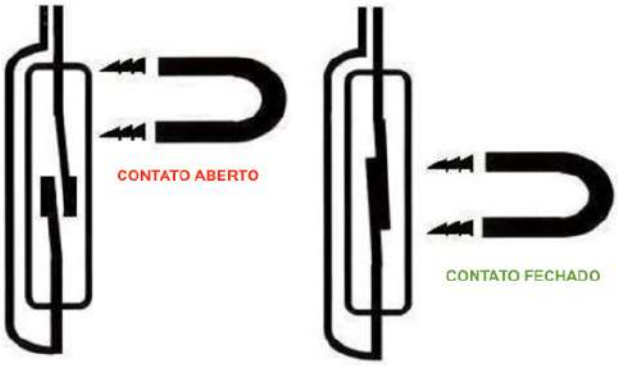
A máquina assistiva conta ainda com um controle remoto, conforme a Figura 9, o qual permite ligar e desligar o motor por meio de um interruptor e alterar a velocidade dos

Figura 6 – Máquina assistiva.



Fonte: (WELLNESS, 2005).

Figura 7 – Diagrama do funcionamento do sensor magnético.



Fonte: Produção do próprio autor.

pedais com um botão giratório. Esses comandos são realizados pelo programa desenvolvido em Python e controlado pela interface. Desta forma o controle foi retirado da máquina assistiva sendo substituído pelo *hardware* desenvolvido, o que será explicado mais adiante.

A máquina apresenta ainda uma placa de controle interna para controlar o motor, de acordo com seu controle remoto. Esta placa foi mantida no projeto visto que seu princípio de funcionamento foi aproveitado e não houve necessidade de alterações para incluir novas

Figura 8 – Fios retorno do sensor magnético na máquina assistiva.

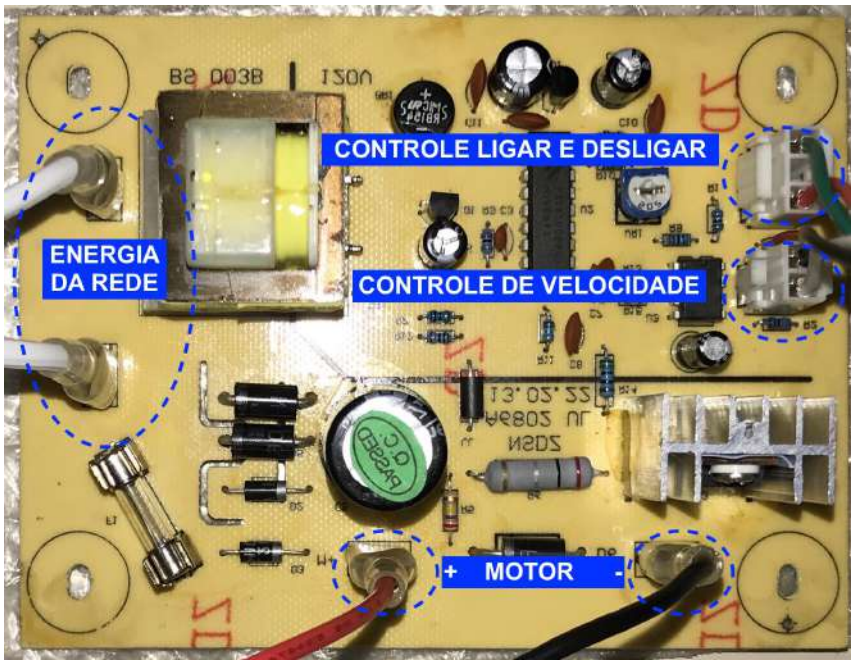
Fonte: Produção do próprio autor.

Figura 9 – Controle remoto da máquina assistiva e seus fios de comando.

Fonte: Produção do próprio autor.

funcionalidades. A placa interna pode ser vista na Figura 10.

Figura 10 – Placa de controle interna da máquina assistiva.



Fonte: Produção do próprio autor.

A alimentação da rede é 127 AC. A placa realiza a transformação para o circuito de controle utilizando um transformador (trafo) que pode ser visto na Figura 10. Os dois pares de cabos destacados à direita correspondem ao sinal que o controle remoto envia para a placa interna para ligar, desligar e alterar a velocidade. Por fim, os cabos vermelhos e pretos abaixo correspondem ao sinal enviado para ligar o motor.

Após uma análise detalhada do circuito de controle da placa do motor, foi observado que para ligar a máquina é necessário conectar os cabos vermelho (+) e verde (-) da Figura 10 entre si. O controle de velocidade realizado originalmente pelo controle remoto utiliza um potenciômetro de $10k\Omega$, sendo que a alteração de velocidade ocorria pela alteração da resistência entre os cabos de controle de velocidade de forma que quanto maior o valor, maior a velocidade.

Utilizando a estrutura original da máquina assistiva, foi levantada uma tabela de correspondência de valores de impedância utilizados nos cabos de controle de velocidade para obter determinadas velocidades. Para isso, o controle remoto foi utilizado para controlar a velocidade da máquina de forma que o valor da velocidade foi coletado do *display* original da mesma. Com o objetivo de discretizar as velocidades para serem utilizadas no *software* escrito em Python, foram escolhidas oito velocidades para serem trabalhadas no projeto: 30, 35, 40, 45, 50, 55, 60 e 70 rpm, visto que a velocidade máxima testada foi de 74 rpm e a mínima de 26 rpm. O método utilizado para levantar as resistências foi ligar a máquina e ajustar a velocidade até obter as 8 desejadas acima no *display*. A cada velocidade, o

Quadro 3 – Quadro com as correspondências impedância e velocidade em rpm da máquina assistiva.

Resistência medida	Resistência comercial	Velocidade
160Ω	150Ω	30 rpm
723Ω	820Ω	35 rpm
1378Ω	$1k2\Omega$	40 rpm
$2.14k\Omega$	$2k2\Omega$	45 rpm
$2.92k\Omega$	$3k\Omega$	50 rpm
$3k8\Omega$	$3k9\Omega$	55 rpm
$4.95k\Omega$	$5k1\Omega$	60 rpm
$8.96k\Omega$	$9k1\Omega$	70 rpm

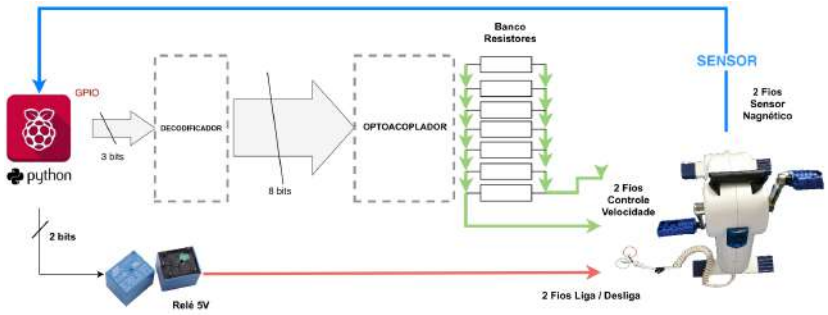
Fonte: Produção do próprio autor.

potenciômetro foi removido e sua impedância foi medida com um multímetro até obter as velocidades conforme o Quadro 3. Por questão de eficiência, o potenciômetro foi dessoldado e utilizado com fios conectores para sua fácil remoção e colocação entre as medições.

4.3 *Hardware* para controle da máquina assistiva

Para realizar o controle de velocidade, ligar e desligar a máquina assistiva utilizando o código em Python, foi desenvolvido um *hardware* para realizar a interface Raspberry Pi - Máquina assistiva. Uma preocupação considerada no desenvolvimento do *hardware* foi manter o circuito eletrônico da máquina assistiva isolado do circuito eletrônico do *hardware* desenvolvido por medidas de segurança. A metodologia do *hardware* pode ser vista na Figura 11.

Figura 11 – Método do *hardware* de controle da máquina assistiva.



Fonte: Produção do próprio autor

Para ligar a máquina assistiva, um relé de $5V$ foi utilizado para realizar o chaveamento dos dois fios que ligam e desligam o motor. Para o sensor magnético, um dos fios é aterrado na RPi e o outro é conectado a uma entrada configurada como *pull-up*. Desta forma, toda vez que o pedal direito passar pela posição de 90 graus que fecha o contato entre os dois

Quadro 4 – Correspondência dos 3 *bits* da RPi para os 8 bits do decodificador, a resistência acionada pelo optacoplador e a velocidade da máquina assistiva.

Saídas RPi	Saída Decodificador	Resistência	Velocidade
000	10000000	150Ω	30 rpm
001	01000000	820Ω	35 rpm
010	00100000	$1k2\Omega$	40 rpm
011	00010000	$2k2\Omega$	45 rpm
100	00001000	$3k\Omega$	50 rpm
101	00000100	$3k9\Omega$	55 rpm
110	00000010	$5k1\Omega$	60 rpm
111	00000001	$9k1\Omega$	70 rpm

Fonte: Produção do próprio autor.

fios um sinal de *ground* será enviado para a entrada da RPi, sendo assim possível realizar uma contagem de voltas sempre que um sinal de nível baixo for identificado nesta porta.

Para realizar o controle de velocidade, foi utilizado um sistema decodificador - optoacoplador. Como foram selecionadas oito velocidades para variar na máquina assistiva, três saídas digitais (3 *bits*) da RPi foram utilizadas para representar 8 valores diferentes ($2^3 = 8$). Assim, com o uso de um decodificador, obtêm-se oito saídas digitais que são acionadas de acordo com a combinação de 3 *bits* na entrada do *chip*, conforme o Quadro 4.

As oito saídas digitais obtidas alimentam oito optoacopladores, onde cada um apresenta em sua saída uma das resistências obtidas no Quadro 3. Como somente uma das oito saídas digitais estará ativa em cada momento, apenas um optoacoplador funcionará por vez, de acordo com a velocidade decodificada. Portanto, quando o *software* enviar para o decodificador um sinal de 3 *bits*, este será decodificado em um sinal +5V em uma das oito saídas, o qual ativará um dos oitos optoacopladores aplicando entre os cabos de controle de velocidade da máquina assistiva um dos oitos resistores, alterando a velocidade.

No total foram utilizados 5 pinos de saída da Raspberry Pi no modo GPIO, sendo dois pinos de alimentação, um de +5V e outro Terra, e um pino de entrada no modo GPIO com *pull-up* para analisar o sensor magnético. Além disso, o relé é alimentado em +5V por uma porta de alimentação exclusiva da RPi, que não é compartilhada com o resto do circuito, devido aos picos de corrente que o chaveamento da bobina proporciona. Observe que o sensor magnético se comporta como uma chave e apresenta dois fios, um em cada extremidade. Um dos fios é conectado ao terra da RPi e o outro é o pino de entrada comentado. No Quadro 5 as portas utilizadas na RPi e correspondência no *hardware* são exibidas.

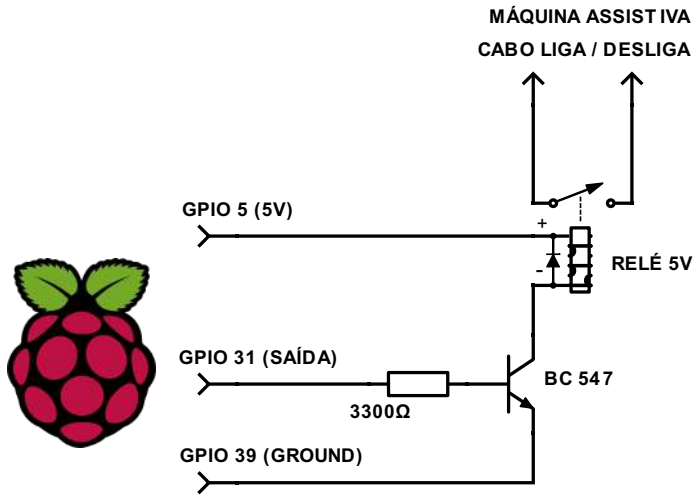
Quadro 5 – Correspondências das portas da RPi e o *hardware*.

Porta	Configuração	Função
GPIO 2	Alimentação	+5V Alimentação Relé
GPIO 4	Alimentação	+5V Alimentação circuito
GPIO 39	Alimentação	Terra do circuito
GPIO 31	Saída	Comando para acionar relé e ligar/desligar máquina
GPIO 33	Saída	bit 0 para controle de velocidade
GPIO 35	Saída	bit 1 para controle de velocidade
GPIO 37	Saída	bit 2 para controle de velocidade
GPIO 40	Entrada	Leitura status sensor magnético

Fonte: Produção do próprio autor.

Para ligar e desligar a máquina foi utilizado um relé de $5V$, com um respectivo diodo de roda livre, um transistor NPN BC547 e uma resistência de base de 3300Ω, de acordo com a Figura 12. Como os pinos de $+5V$ apresentam uma capacidade maior de fornecer corrente quando comparada aos pinos GPIO, foi montado o circuito para que a alimentação do relé fosse feita pelos pinos $+5V$ e terra da RPi. O acionamento ocorre quando um sinal alto ($5V$) é aplicado no pino GPIO 31. Desta forma, a corrente que circula na base do transistor é suficiente para fazê-lo operar saturado, funcionando como uma chave.

Figura 12 – Circuito do acionamento da máquina assistiva.

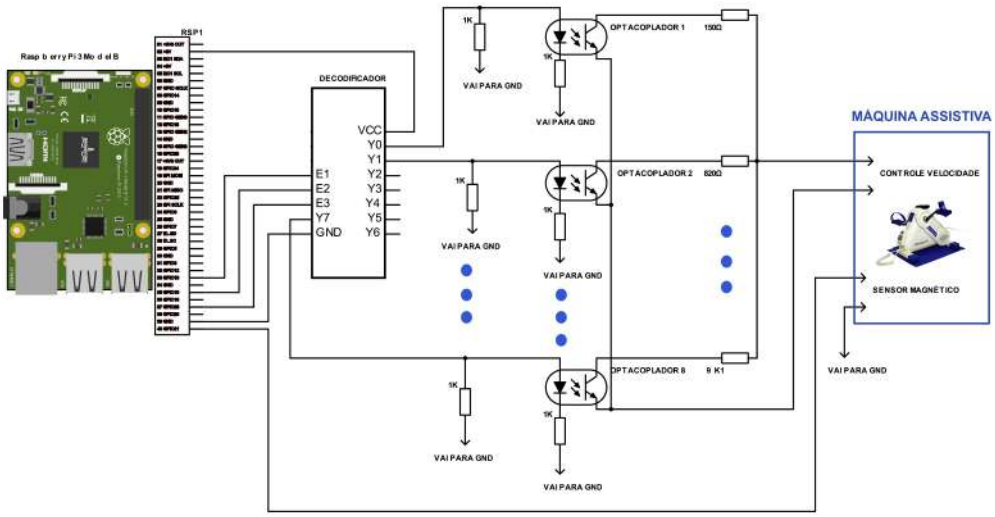


Fonte: Produção do próprio autor.

Para a decodificação do sinal de 3 *bits* e controle de velocidade dos pinos da RPi, conforme Quadro 5, foi utilizado o *chip* decodificador da Motorola MC14051B, dois chips BC847 com 4 optoacopladores cada, 16 resistores de $1k\Omega$ (8 para limitar corrente e 8 para *pull-down*) e as oito resistências encontradas para as 8 velocidades distintas, conforme a Figura 13.

Note também que uma das saídas do sensor magnético presente na máquina foi aterrado no pino GND da RPi, e o outro na entrada, conforme explicado anteriormente.

Figura 13 – Circuito do decodificador e optacopladores para controle de velocidade selecionando resistências.

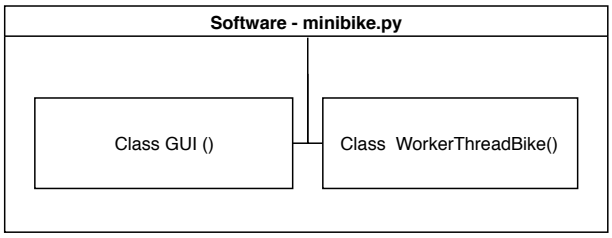


Fonte: Produção do próprio autor.

4.4 *Software* para o controle da máquina assistiva e a GUI

O *software* desenvolvido em Python para o controle da máquina assistiva e para a interface gráfica apresenta a estrutura de classes conforme a Figura 14. Observe que são duas classes: GUI() (*General User Interface*) e *WorkerThreadBike* (). A primeira é responsável por toda a interface gráfica do *software*, e a segunda pelo controle da máquina assistiva. Ambas se comunicam por meio de sinal, conforme explicado anteriormente, e *WorkerThreadBike* () é executada como uma *thread*.

Figura 14 – Estrutura de classes no *software*.



Fonte: Produção do próprio autor.

4.4.1 Interface - Classe GUI()

A classe GUI () do *software* desenvolvido é responsável por importar dois métodos desenvolvidos no programa QtDesigner, os quais correspondem a duas janelas: a janela de acesso (*loginwindow*) ao *software* e a janela principal que controla a máquina assistiva em três estilos diferentes. Todas as interfaces foram desenhadas no QtDesigner com todos os elementos pertinentes. Conforme explicado, o programa da interface foi desenhada em um arquivo (.ui) que foi convertido em Python. Os códigos então foram importados no *software* desenvolvido (minibike.py) como módulos para serem executados na classe GUI ().

O método da interface pode ser visto na Figura 15. A inicialização do *software* abre inicialmente uma tela de acesso com usuário e senha. Após a entrada, uma janela principal é aberta com 4 abas: *MiniBike Command*, *Free Mode*, *Fixed Mode* e *Test Mode*.

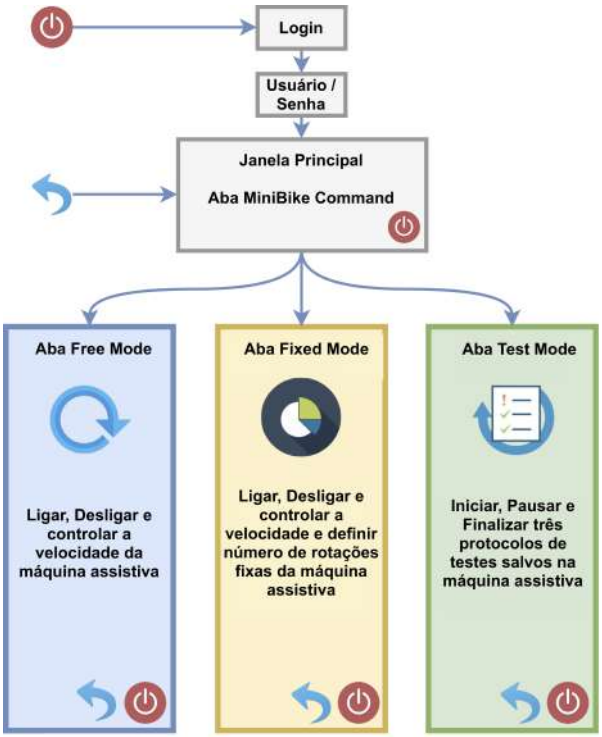
A primeira aba apresenta botões para acessar as três outras abas, e um botão vermelho para fechar a janela principal e retornar à tela de *login*. A segunda aba abre uma janela que permite ao usuário ligar e desligar a máquina assistiva e controlar sua velocidade, escolhendo uma das 8 disponíveis, conforme o Quadro 4. A terceira aba permite controlar a máquina de forma a determinar o número de voltas que ela realizará na velocidade escolhida na própria aba. A quarta e última aba disponibiliza para o usuário poder aplicar três protocolos de testes pré-definidos, possibilitando iniciar, pausar e finalizar os testes.

A classe apresenta algumas funções que são responsáveis pela gerência das ações que o usuário realiza na interface. Na função de inicialização (__init__()), as duas janelas de interface são inicializadas criando os objetos desenhados; os elementos das interfaces que sofrem ações são associados às funções criadas na própria classe GUI () usando os objetos de cada janela; o objeto *thread* da classe *WorkerThreadBike* é inicializado; e os sinais que comunicam as duas classes são associados às funções respectivas definidas na classe GUI () (funções dos sinais). A estrutura da classe pode ser vista na Figura 16.

4.4.1.1 Janela de acesso - *login window*

A janela de acesso é a responsável pelo acesso ao *software* desenvolvido. O método desenvolvido consiste no uso de um usuário e senha que deve ser inserido corretamente, o qual está representado na Figura 17. A verificação do usuário/senha é realizada acessando o banco de dados em MySQL procurando pela combinação de entrada pelo usuário após

Figura 15 – Método da interface do *software*.



Fonte: Produção do próprio autor.

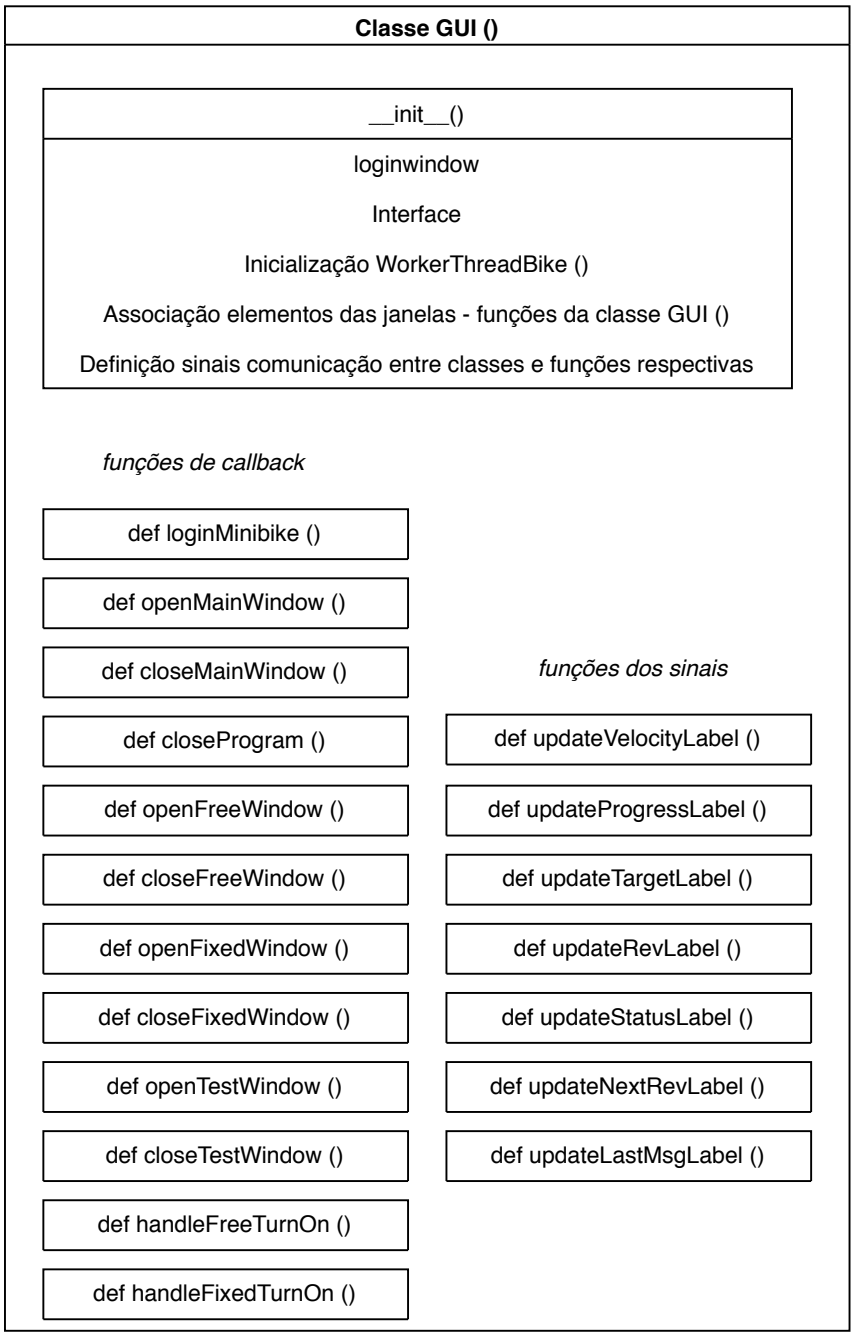
clicar em um botão de acesso chamado *Login*. Caso a combinação não exista, será exibido para o usuário uma mensagem de senha incorreta ou usuário inválido. A janela de acesso também conta com um botão *Exit* para terminar o programa caso o usuário desista de acessar o mesmo.

4.4.1.2 Janela principal - *Main Window*

Realizado o acesso ao programa pela janela de *login*, a janela principal é aberta na aba inicial (*MiniBike Command*) e a janela de acesso é terminada. O objetivo desta aba é mostrar para o usuário as opções disponíveis para o comando, e direcionar para as abas específicas cada modo disponível. O método desta janela pode ser visto na Figura 18, a qual apresenta ainda uma opção de terminar o programa por meio de um botão vermelho, retornando à janela de acesso.

A escolha de um dos três métodos leva a janela para a aba do método correspondente. A aba *Free Mode* apresenta elementos que permitem ao usuário ligar e desligar a máquina assistiva a qualquer momento, e alterar sua velocidade, se desejado, durante o funcionamento. Além disso, a interface permite a escolha da velocidade com que a máquina será ligada, de acordo

Figura 16 – Estrutura da classe GUI ().



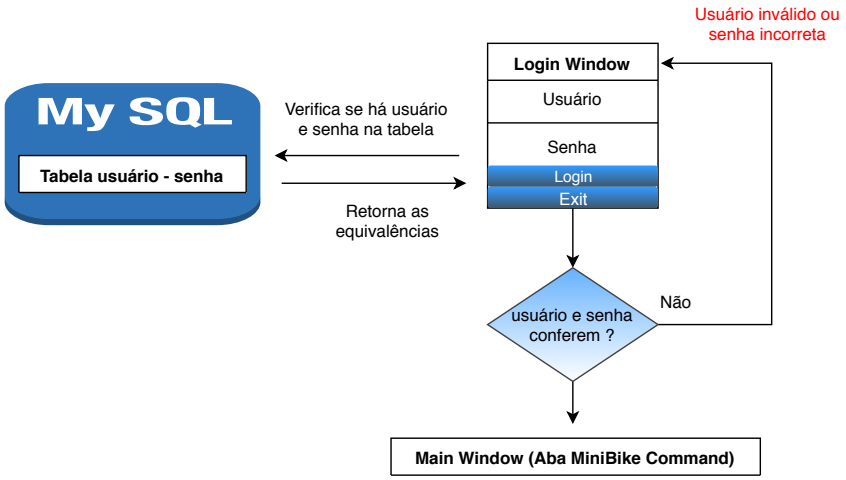
Fonte: Produção do próprio autor.

com as velocidades definidas no Quadro 6. O diagrama que representa essa interface pode ser visto na Figura 19.

Desta forma, para utilizar este método, o usuário precisa:

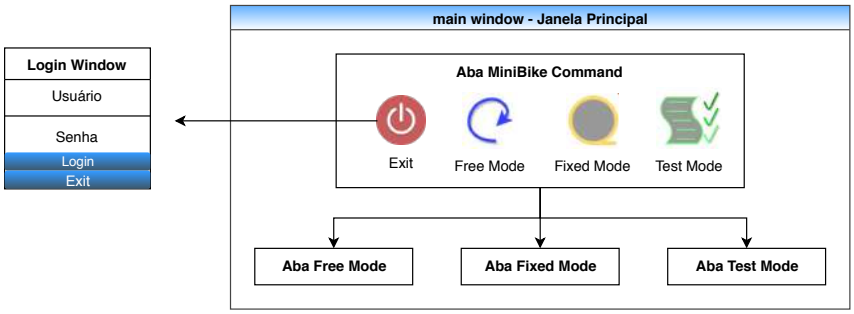
1. Escolher a velocidade desejada inicial (que a máquina será ligada) na lista de velocidades disponível.

Figura 17 – Método da janela de acesso - *login window*.



Fonte: Produção do próprio autor.

Figura 18 – Método da aba *Minibike Command* da janela principal.



Fonte: Produção do próprio autor.

2. Ligar a máquina assistiva no botão *TURN ON*.

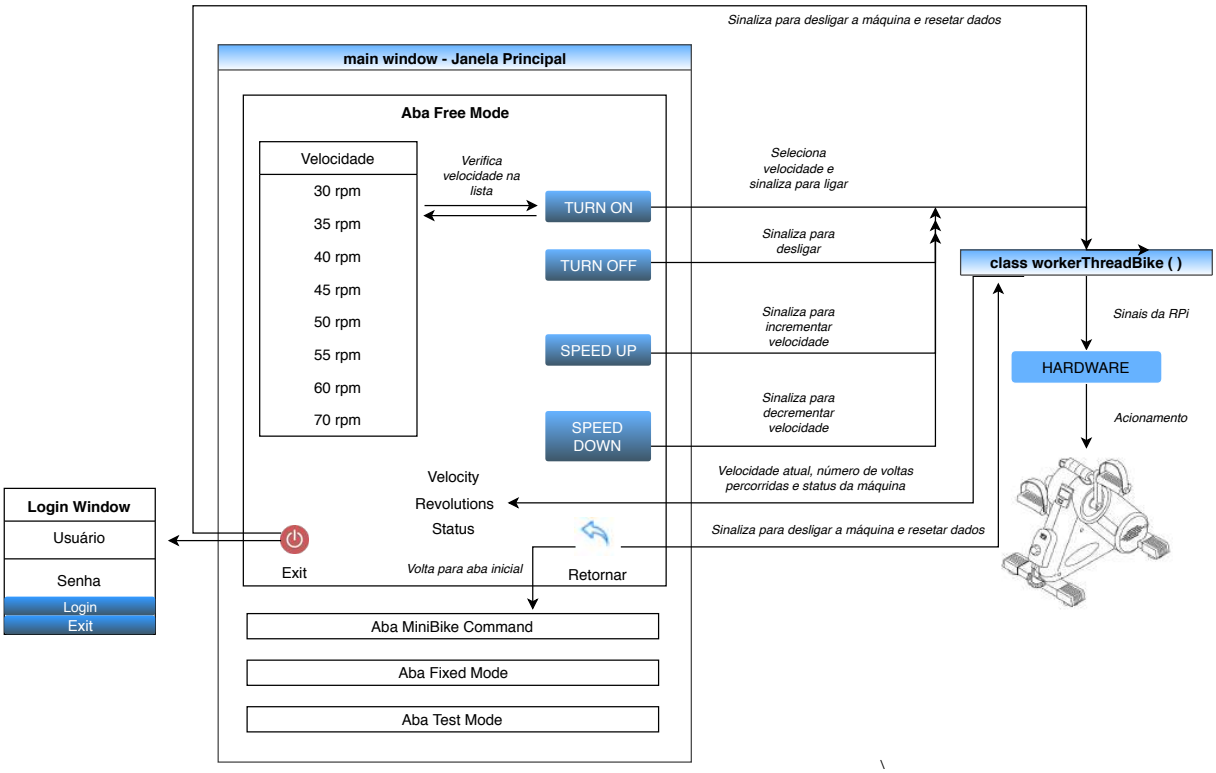
3. Se desejado, aumentar ou diminuir a velocidade a partir da atual, através dos botões *SPEED UP* e *SPEED DOWN*. Observe que os botões aumentam ou diminuem a velocidade de uma para outra em sequência, de acordo com o Quadro 6.

4. Desligar, a qualquer momento, a máquina através do botão *TURN OFF*.

5. Acompanhar a velocidade atual, número de voltas e estado da máquina na área de informações.

Observa-se pelo diagrama da Figura 19 que a interface mostra para o usuário o atual *status* da máquina assistiva (ON - ligada ou OFF - desligada), o número de revoluções já executadas e a velocidade atual de execução. Note também que a comunicação realizada da interface com a *thread* que controla a máquina é realizada por sinais, conforme explicado anteriormente.

Figura 19 – Método da aba *Free Mode* da janela principal.



Fonte: Produção do próprio autor.

Quadro 6 – Quadro com as velocidades na lista da janela principal.

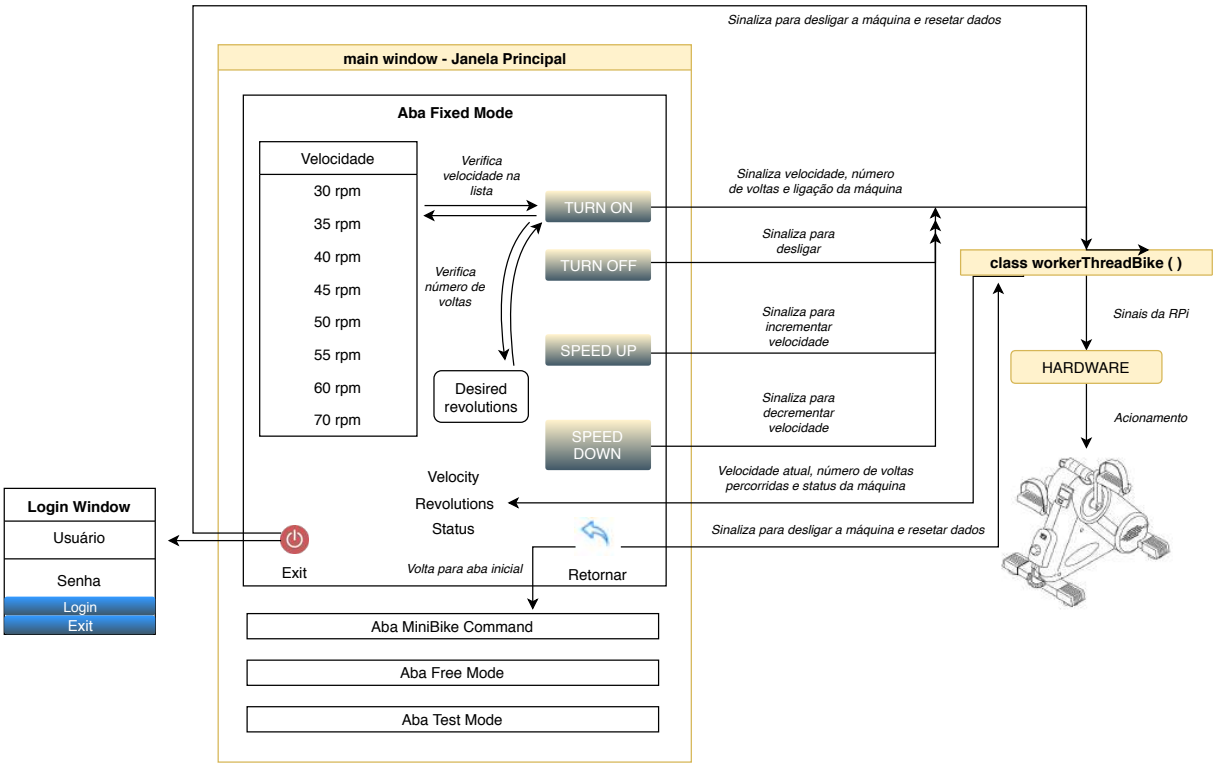
Velocidade	RPM
Velocity 1	30 rpm
Velocity 2	35 rpm
Velocity 3	40 rpm
Velocity 4	45 rpm
Velocity 5	50 rpm
Velocity 6	55 rpm
Velocity 7	60 rpm
Velocity 8	70 rpm

Fonte: Produção do próprio autor.

A segunda aba, *Fixed Mode*, apresenta um diagrama de funcionamento muito semelhante à primeira, que pode ser visto na Figura 20. O objetivo principal desta aba é permitir que o usuário escolha a quantidade de voltas a ser realizada. É possível também escolher a velocidade inicial a ser desempenhada nas voltas, ligar e desligar a máquina a qualquer momento, e incrementar ou decrementar a velocidade durante a operação.

Desta forma, para utilizar este método, o usuário precisa:

Figura 20 – Método da aba *Fixed Mode* da janela principal.



Fonte: Produção do próprio autor.

1. Escolher a velocidade desejada inicial (que a máquina será ligada) na lista de velocidades.

2. Digitar a quantidade de voltas a ser realizada pela máquina em um campo disponível para este fim.

3. Ligar a máquina assistiva através do botão *TURN ON*.

4. Se desejado, aumentar ou diminuir a velocidade a partir da atual através dos botões *SPEED UP* e *SPEED DOWN*.

5. Acompanhar a velocidade atual, número de voltas e estado da máquina.

6. Esperar a máquina completar as voltas desejadas ou desliga-lá a qualquer momento através do botão *TURN OFF*.

A diferença principal desta aba está no fato da necessidade do usuário inserir o número de voltas desejadas, sendo que um erro será emitido ao usuário se outro tipo de dado diferente de inteiro for inserido. O restante se comporta da mesma maneira que a aba *Fixed Mode*.

Por último, a aba *Test Mode* pode ser vista na Figura 28. O objetivo principal desta aba é permitir que o usuário realize três protocolos de testes pré definidos, os quais foram utilizados em outras pesquisas de mestrado do NTA.

O primeiro teste realiza uma sequência de 25 quantidades de voltas separadas por um intervalo randômico de $2.0s$ a $5.0s$ (espaçados de $0.5s$ cada). O número de voltas em cada uma das 25 quantidades é pré determinado pelo sistema conforme o vetor *teste1voltas* na Equação 4.1. A cada vez que a quantidade de voltas de uma entrada do vetor *teste1voltas* é realizada, a máquina é parada durante um dos tempos em segundos no vetor *teste1tempo*, conforme a Equação 4.2. A escolha do tempo é realizada de forma aleatória pelo sistema.

$$teste1voltas = [3, 1, 5, 2, 4, 2, 1, 5, 4, 3, 5, 1, 3, 2, 4, 2, 3, 5, 4, 1, 2, 5, 3, 4, 1, 0] \tag{4.1}$$

$$teste1tempo = [2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0] \tag{4.2}$$

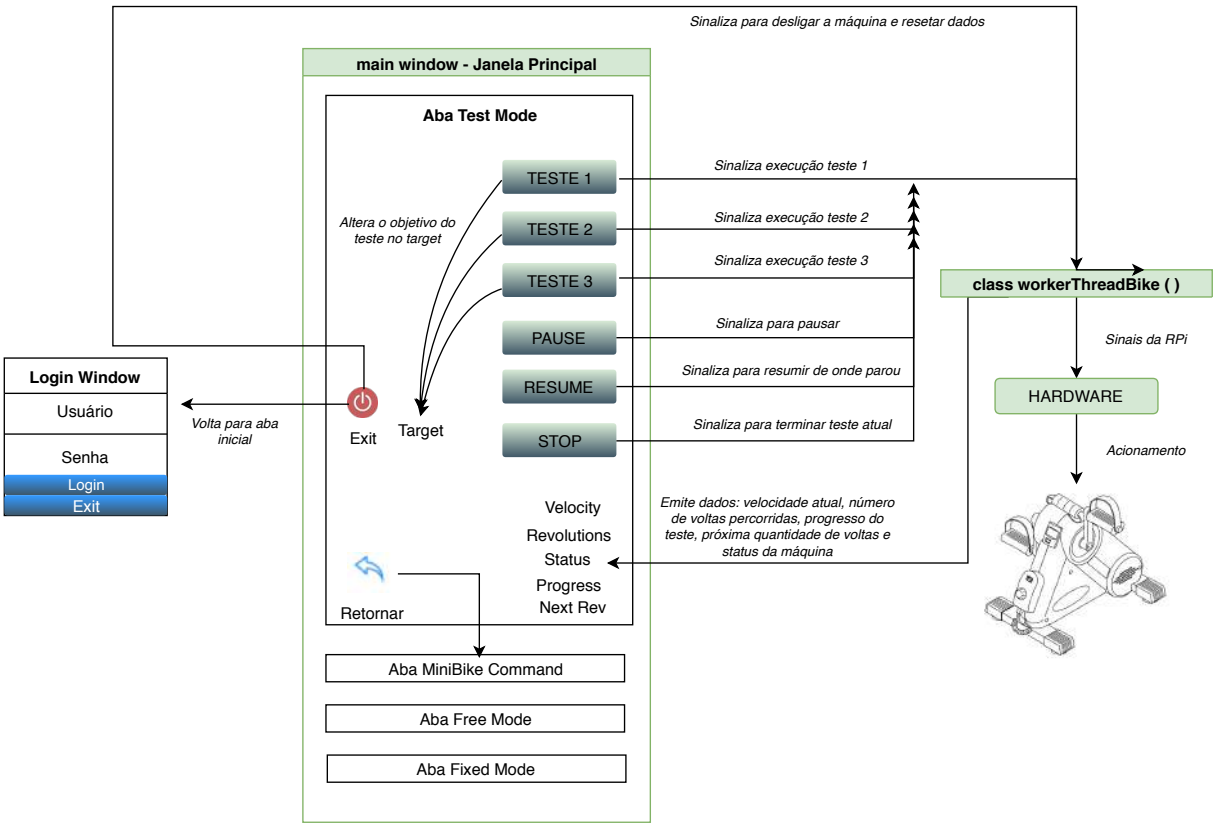
O segundo teste consiste na execução de 60 voltas, sendo que entre cada volta a máquina fica parada uma quantidade de segundos, seguindo a lógica do teste 1, utilizando os valores da Equação 4.2. Por fim, o terceiro teste liga a máquina assistiva durante 2 min e desliga após o tempo decorrido. O método desta aba pode ser visto na Figura 21.

Assim, para utilizar este método, o usuário precisa:

1. Escolher o teste a ser realizado pela máquina, clicando em um dos três botões correspondentes (Teste 1, Teste 2 e Teste 3), disponíveis na interface final.
2. Esperar o teste acabar ou, se precisar, pausar, recomeçar ou terminar o teste antes do seu fim, através dos botões: *Pause*, *Resume* e *Stop*.
3. Se optado por esperar o teste ser completado, a máquina irá parar após o fim do mesmo.

Para mostrar o progresso e o *status* da execução dos testes, essa interface também apresenta o atual estado de progresso do teste que está sendo realizado, e qual o seu objetivo, os quais são exibidos em *Progress* e *Target* na Figura 21. Observa-se ainda a necessidade de mostrar a próxima quantidade de voltas a ser realizada pela máquina quando está realizando o Teste 1, de forma a preparar o usuário para a próxima execução.

Figura 21 – Método da aba *Test Mode* da janela principal.



Fonte: Produção do próprio autor.

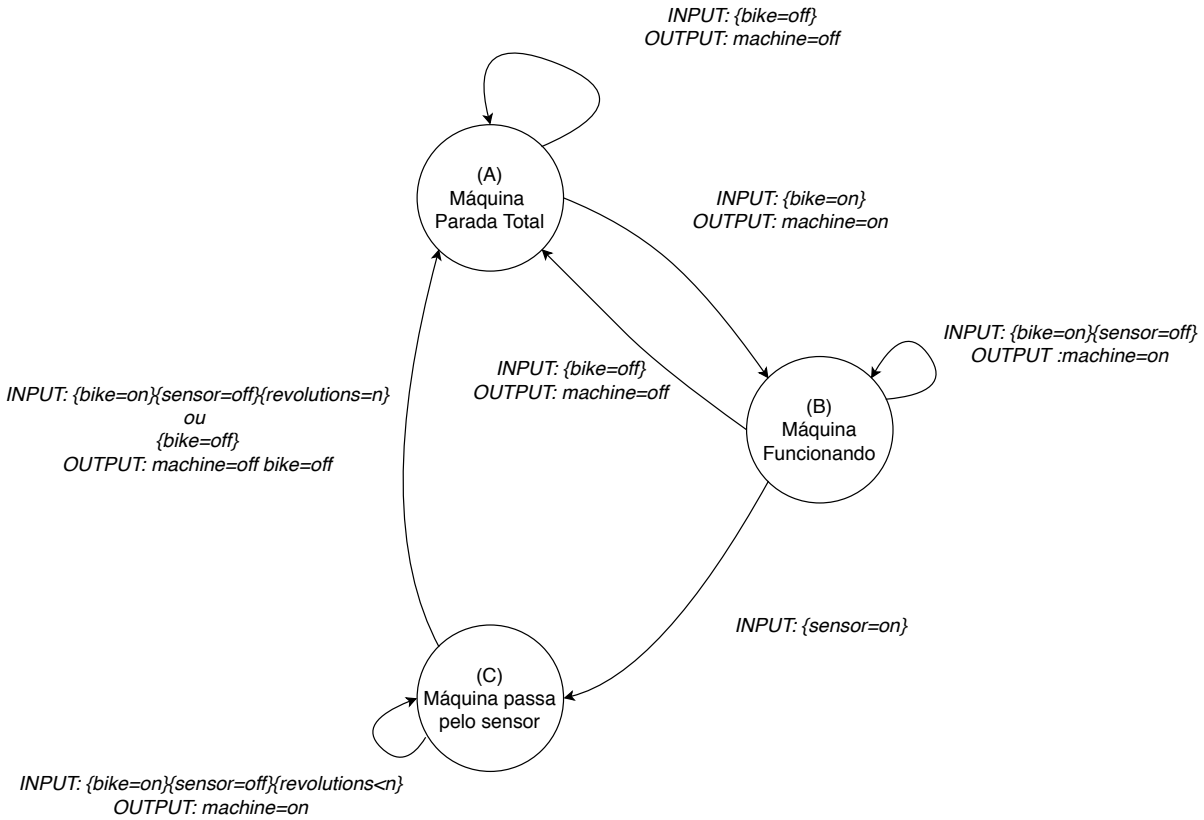
4.4.2 Lógica do controle - Classe *WorkerThreadBike* ()

A lógica para a execução e controle da máquina assistiva é processada por uma classe operando como *threading*, a *WorkerThreadBike* (). Como explicado, a necessidade da utilização da programação por *threading* se dá pelo fato da interface e do controle precisarem operar em paralelo no programa, pois a interface precisa sempre estar de prontidão para detectar e atuar nas ações dos usuários e, ao mesmo tempo, a máquina precisa executar os comados recebidos e permanecer na lógica de funcionamento.

A lógica utilizada no controle da máquina em todos os modos consiste em uma máquina de estados. A máquina apresenta três estados: máquina parada (A), máquina funcionando (B), e máquina passa pelo sensor (C). O primeiro estado (A) é o estado em que a máquina assistiva está parada, em seu estado inicial, ou seja, antes de começar sua operação ou após ter finalizado alguma quantidade de voltas. O segundo estado (B) corresponde ao seu funcionamento quando os pedais estão em rotação executando a atividade desejada pelo usuário. Quando a máquina passa pelo sensor magnético, ela está no estado (C), o qual é importante para verificar se a máquina já realizou a quantidade de rotações definidas, se for o caso.

Na Figura 22 são mostrados os três estados e as entradas/saídas de cada um, e a Figura 23 exibe os respectivos sinais de entrada, saída e estados.

Figura 22 – Método da máquina de estados para o controle da máquina assistiva.



Fonte: Produção do próprio autor.

Observe que a entrada *bike* é uma variável que sofre constante alteração por meio de uma lógica, para representar quando a máquina assistiva deve ser ligada ou não. Isso ocorre devido aos diferentes métodos que o sistema pode operar. Assim, deve ser verificado se está operando no modo teste, *free* ou *fixed*. Como cada modo apresenta suas peculiaridades referentes à rotação (quantidade e tempo de intervalo entre rotações, por exemplo), a variável *bike* representa a ação a ser tomada pelo funcionamento da máquina assistiva e atua como entrada na máquina de estados.

Figura 23 – Estados, entradas e saídas da máquina de estados.

Estados
(A) Máquina parada (B) Máquina em funcionamento (C) Máquina passa pelo sensor

Saída
machine: ação de ligar (on) ou desligar (off) a máquina assistiva

Entradas
bike: entrada que diz quando a máquina é ligada ou desligada. Esta entrada é controlada pelo programa de acordo com o estilo de funcionamento da máquina: fixed, teste ou free. (on -> maquina ligada; off=> desligada) *sensor: sinal de entrada que indica quando o pedal passa pelo sensor (on) ou quando não passa pelo sensor (off)* *revolutions: entrada que indica se a máquina já realizou o número de voltas que era preciso (n voltas)*

Fonte: Produção do próprio autor.

5 RESULTADOS

5.1 Interface

Nas seções que seguem as interfaces criadas para as janelas explicadas anteriormente são exibidas. Todas as janelas com as interfaces apresentam a característica de se moldarem de acordo com o tamanho que o usuário deseja. Desta forma, é possível aumentar e diminuir o tamanho de cada janela de forma livre, e todos os elementos acompanham essa alteração. Esta é uma característica importante e foi implementada, pois o programa pode vir a ser executado em diferentes monitores com resoluções distintas, assim, sua visualização é garantida de forma nítida em diferentes situações.

5.1.1 Janela de acesso - Login

A interface criada para a janela de acesso pode ser vista na Figura 24. O requisito principal desta interface é oferecer ao usuário uma forma de acesso clara e concisa ao *software*. Desta forma, foram implementados somente os elementos necessários para acessar e verificar a combinação de usuário e senha. O tamanho da tela, como comentado anteriormente, pode ser alterado livremente. Porém, ela é iniciada em uma resolução menor que as outras (265px, 426px) para focar o usuário apenas no necessário, e não apresentar muitos espaços brancos não utilizados. Caso o usuário e a senha estejam errados, aparecerá uma mensagem em vermelho na tela avisando o tipo de erro e os campos digitados da senha e do *login* são apagados.

Conforme explicada sua lógica, os elementos que a compõe representam:

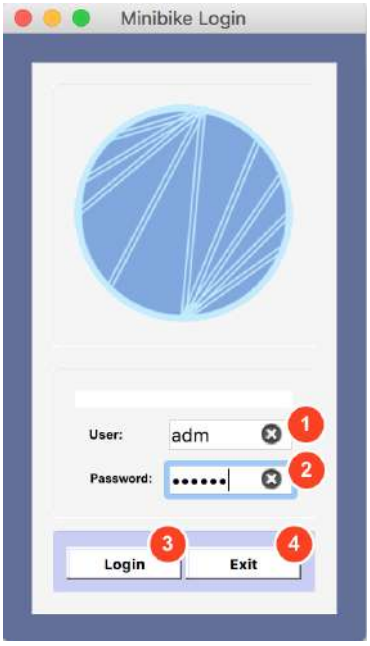
1 O campo para inserir o nome de usuário.

2 O campo para inserir a senha do usuário.

3 O botão de acesso ao programa caso o usuário e a senha estejam corretos.

4 O botão de término do programa (e da janela).

Figura 24 – Janela de acesso ao programa - *login window*.



Fonte: Produção do próprio autor.

5.1.2 Janela principal - *Minibike Command*

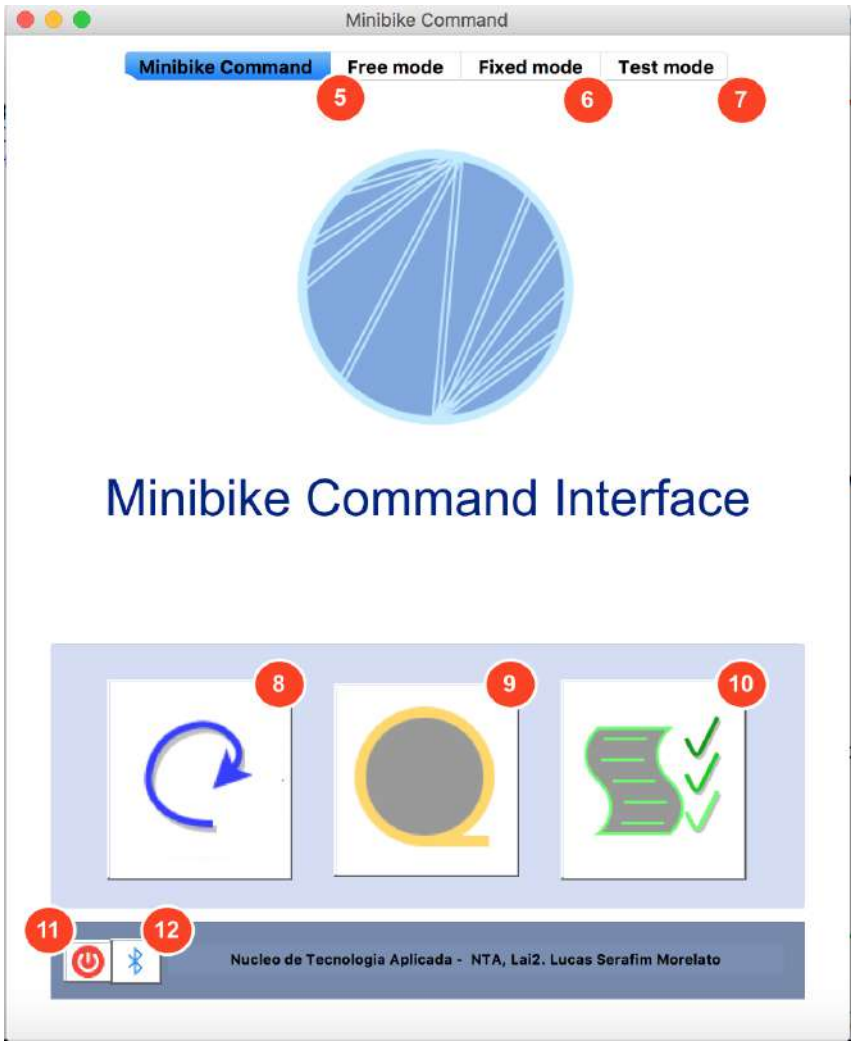
A Figura 25 exibe a interface janela principal do *software* e é executada logo após o acesso no programa pela interface da Figura 24. Com o objetivo principal de direcionar o usuário para um dos três modos de operação da máquina assistiva, uma solução encontrada para essa interface foi utilizar três botões com ícones que representam os três modos de operação. Os botões destacados utilizam um fundo de cor azul, de forma a realizar um contraste na área da tela. Essa técnica foi utilizada para atrair a atenção do usuário para os botões de acesso aos modos de operação, que é o principal objetivo desta aba.

Foi mantida também a mesma logo utilizada na tela de acesso, e um título para identificar a interface inicial do programa. Pode-se observar a barra inferior, a qual apresenta os botões [11], [12] e os créditos. Optou-se por colocar essas informações no final da interface para não serem muito destacadas e não tirarem o foco dos botões dos modos de operação. Abaixo os elementos da aba são descritos.

5 Etiqueta *Free Mode* informa se o programa está na aba de controle *Free* quando azul.

6 Etiqueta *Fixed Mode* informa se o programa está na aba de controle *Fixed* quando azul.

Figura 25 – Janela principal do programa - *Main Window* ou *Minibike Command*.



Fonte: Produção do próprio autor.

7 Etiqueta *Test Mode* informa se o programa está na aba de controle *Test* quando azul.

8 Botão responsável por acessar a aba que controla a máquina assistiva em seu modo *Free*.

9 Botão responsável por acessar a aba que controla a máquina assistiva em seu modo *Fixed*.

10 Botão responsável por acessar a aba que controla a máquina assistiva em seu modo *Test*.

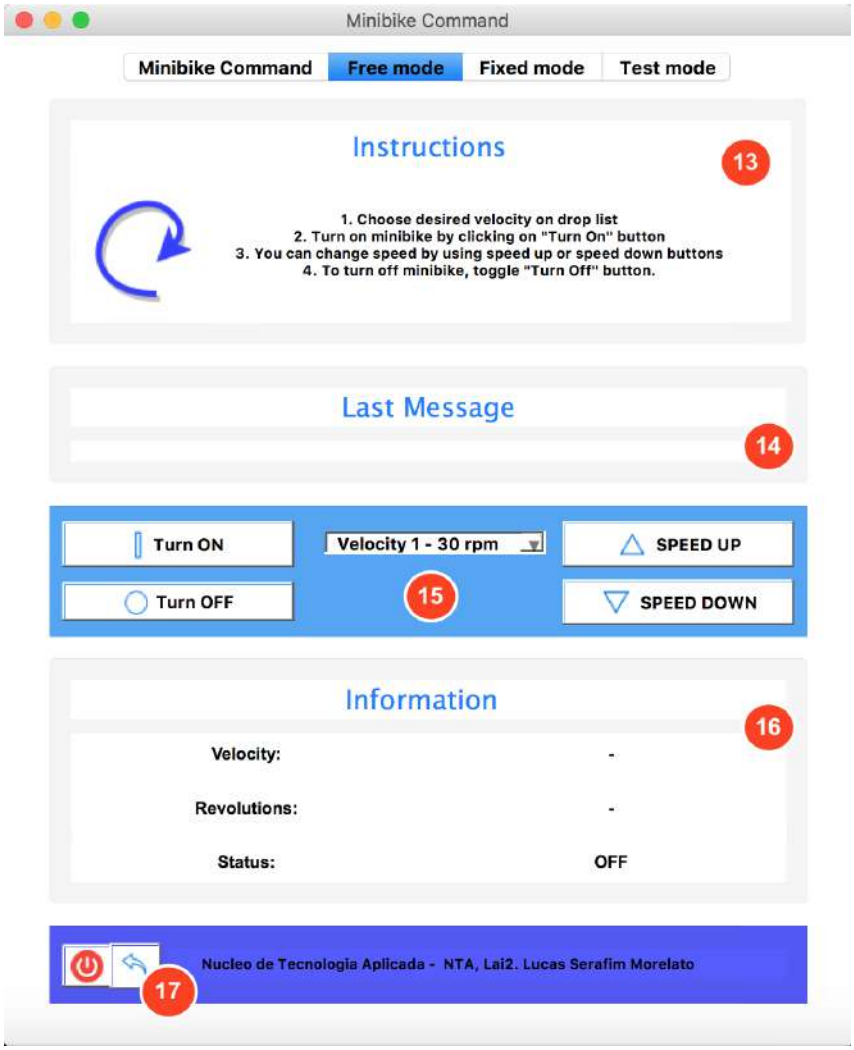
11 Botão responsável por fechar a janela principal e retornar a janela de acesso da Figura 24.

12 Botão responsável por abrir uma janela de conexão *bluetooth* utilizada em outro projeto que será explicado mais a frente.

5.1.3 Aba *Free Mode - Minibike Command*

A interface da aba *Free Mode* pode ser vista na Figura 26, a qual é aberta após clicar no botão [9] da janela principal na Figura 25. A estratégia usada para a interface dos modos foi utilizar abas na própria janela principal. Isso mantém uma padronização e une os modos em uma mesma janela, entregando ao usuário uma sensação de maior facilidade na movimentação entre os diferentes modos disponíveis. Pode-se observar que quando uma determinada aba está aberta, sua identificação no topo da interface fica em azul, mostrando ao usuário que esta aba está selecionada no momento.

Figura 26 – Aba *Free Mode* da janela principal.



Fonte: Produção do próprio autor.

Conforme explicada sua lógica, os elementos compõe a janela são:

13 Etiqueta *Instructions*, que informa como se deve proceder para controlar a máquina assistiva neste modo de operação.

14 Etiqueta *Last Message*, que informa ao usuário as situações pertinentes que aconteceram momentos antes da situação atual. Por exemplo, se a máquina acabou de ser ligada, aparecerá uma mensagem informando que a máquina foi acionada.

15 Área com quatro botões de controle e uma lista de velocidades. Nesta parte, o botão *TURN ON* é responsável por ligar a máquina na velocidade selecionada na lista entre os quatro botões. O botão *TURN OFF* realiza o desligamento da máquina, e os botões *SPEED UP* e *SPEED DOWN* incrementam e decrementam, respectivamente, a velocidade de rotação em um nível de acordo com as velocidades disponíveis no Quadro 6.

16 Quadro *Information*, que informa o *status* (Ligada - ON ou Desligada - OFF), a velocidade atual em *Velocity*, e o número de voltas que o pedal realizou em *Revolutions*.

17 Botão de retorno. Este botão realiza o retorno para a aba inicial *Minibike Command* na Figura 25.

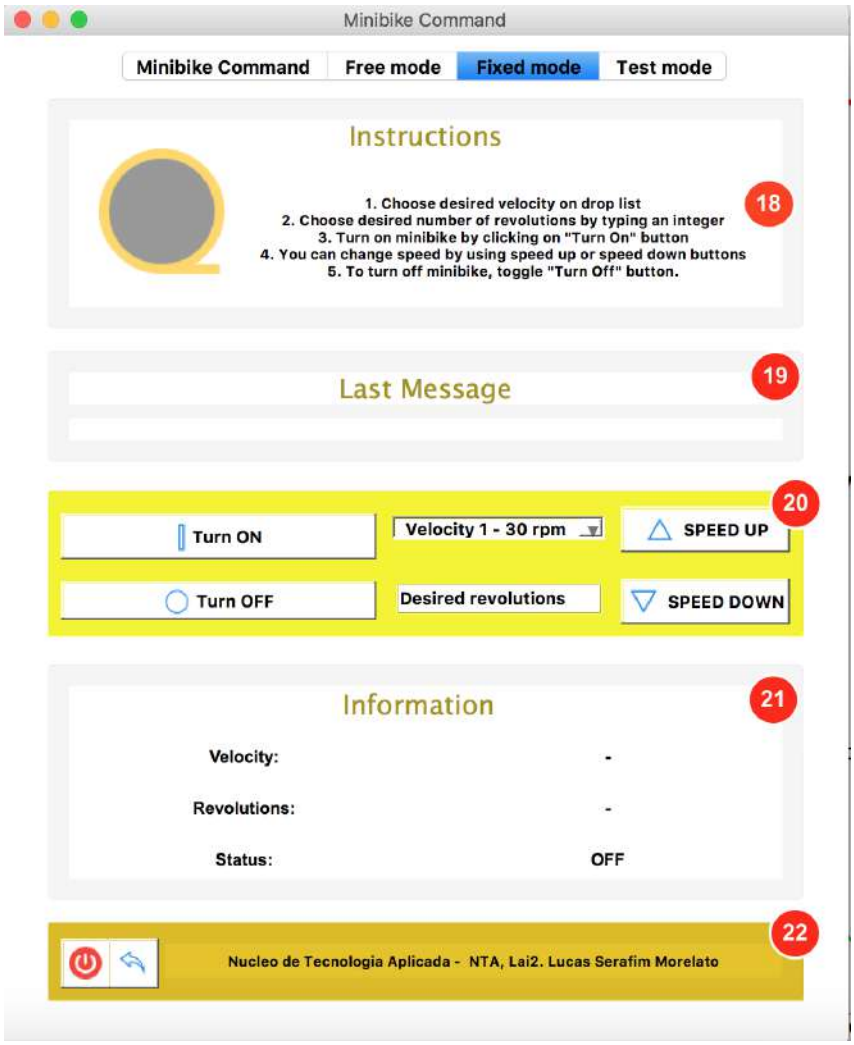
A ideia aplicada nessa e nas outras abas foi destacar os botões de operação utilizando um fundo de cor única que une todos os elementos que o usuário precisa utilizar/atuar conforme destacado no número [15] na Figura 26. Nas demais áreas foram utilizadas cores neutras de fundo branco para manter uma interface clara e limpa. Pode-se observar que para este modo a cor predominante utilizada foi azul, e duas outras cores são utilizadas nas outras duas abas. Isso é importante para levar ao usuário uma assimilação de cores versus modo de operação, a qual incorpora uma caracterização de cada modo em uma cor única, além de atrair o usuário e não tornar todas as interfaces monótonas.

A lógica da informação das interfaces das três abas é vertical, de cima para baixo. Isso é, tudo que o usuário precisa fazer está em ordem de colocação da parte superior para a parte inferior, desde as instruções até as informações das operações. Por fim, a mesma barra de botões para voltar e sair do programa é colocada na mesma posição, de forma a manter uma uniformidade nas abas.

5.1.4 Aba *Fixed Mode - Minibike Command*

A interface da aba *Fixed Mode* pode ser vista na Figura 27 e seus elementos explicados logo abaixo.

Figura 27 – Aba *Fixed Mode* da janela principal.



Fonte: Produção do próprio autor.

18 Etiqueta *Instructions*, que informa como se deve proceder para controlar a máquina assistiva neste modo de operação.

19 Etiqueta *Last Message*, que informa aos usuários situações pertinentes que aconteceram momentos antes da situação atual, igual à aba anterior.

20 Área com quatro botões de controle, uma lista de velocidades e uma caixa de texto de entrada. Os botões e a lista de velocidade se comportam da mesma maneira como explicado na aba anterior; a diferença está na caixa de entrada do texto, *desired revolutions*, que permite ao usuário colocar um número inteiro de voltas a ser realizada. Se inserido um valor diferente de inteiro, uma mensagem de erro em vermelho aparecerá no campo *Last Message* e não iniciará a máquina assistiva.

21 Quadro *Information* que informa o *status* (Ligada - ON ou Desligada - OFF), a velocidade atual em *Velocity* e o número de voltas que o pedal realizou em *Revolutions*

similar à aba anterior.

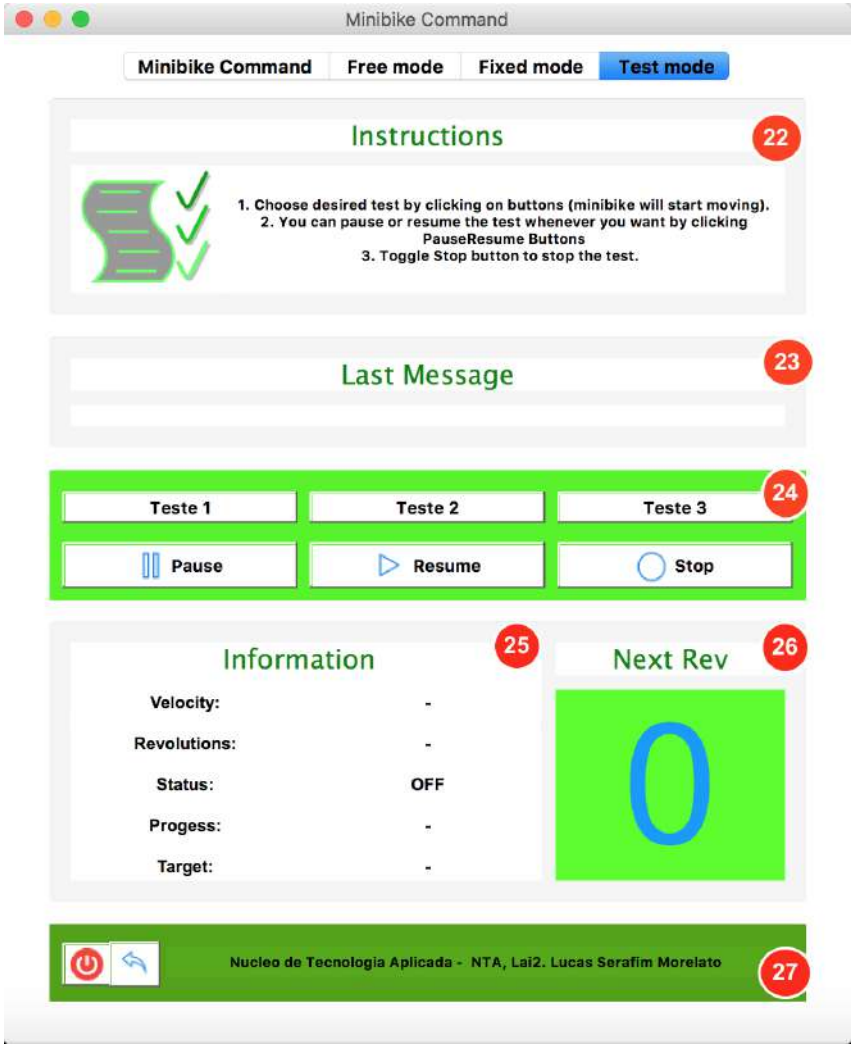
22 Área com botão de retorno e de desligar conforme explicado anteriormente.

Observa-se que a mesma estrutura de interface da aba *Free Mode* foi utilizada nesta, diferenciando-se apenas a cor do foco, que é amarela. Novamente os botões de controle (*Turn ON*, *Turn OFF*, *SPEED UP* e *SPEED DOWN*) são destacados para atrair o usuário.

5.1.5 Aba *Test Mode - Minibike Command*

Por fim, a interface da última aba *Test Mode* pode ser vista na Figura 28, e os respectivos elementos envolvidos são listados abaixo:

Figura 28 – Aba *fixed Mode* da janela principal.



Fonte: Produção do próprio autor.

22 Etiqueta *Instructions*, que informa como se deve proceder para controlar a máquina assistiva neste modo de operação.

23 Etiqueta *Last Message*, que informa ao usuário as situações pertinentes que aconteceram momentos antes da situação atual igual nas abas anteriores.

24 Área com 6 botões de controle: Teste 1, Teste 2 e Teste 3, os quais iniciam os três testes quando são clicados. A qualquer momento os testes podem ser pausados (parando o movimento da máquina) ao clicar no botão *Pause*; podem ser prosseguidos de onde pararam ao clicar no botão *Resume*; ou podem ser abortados ao clicar no botão *Stop*.

25 Quadro *Informations*, que informa o *status*, a velocidade atual e o número de voltas, assim como explicado nas abas anteriores. Apresenta ainda duas novas informações, *Progress* e *Target*. A primeira exibe o atual progresso que a máquina assistiva está, perante o teste que está sendo realizado. Por exemplo, durante o Teste 3, *Progress* exibirá o tempo já decorrido da máquina em funcionamento. O segundo mostra o objetivo final do teste, de forma que no exemplo do Teste 3 aparecerá 2 min, pois o objetivo do Teste 3 é manter a máquina ligada por 2 min.

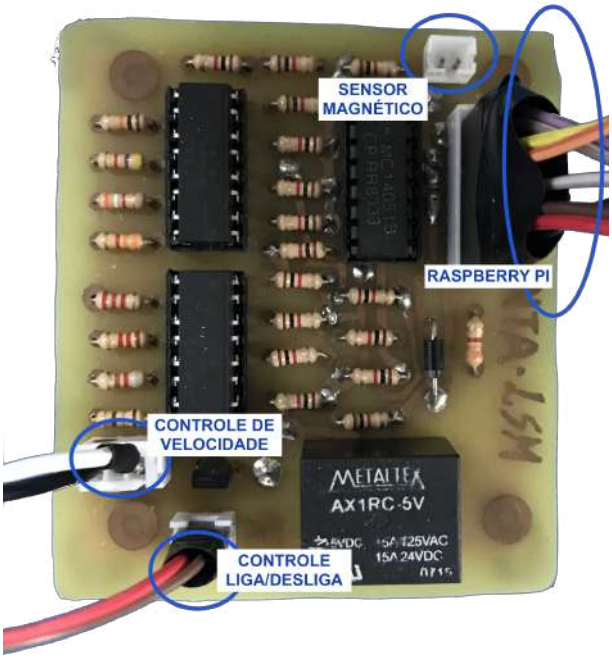
26 O quadro *Next Rev* é responsável por exibir a próxima quantidade de voltas que a máquina realizará após sua pausa, sendo aplicada somente aos Testes 1 e 2, visto que o terceiro teste liga a máquina durante 2 min.

Novamente, utiliza-se a mesma estrutura das duas últimas abas porém com destaque da cor verde para este modo. Pode-se observar o tamanho utilizado para indicar a próxima quantidade de revoluções em *Next Rev*. A ideia utilizada, junto com as cores de destaque, é para facilitar a visualização do usuário na máquina assistiva, e não somente quem está conduzindo a interface. Isso se deve à necessidade da pessoa que está nos pedais da máquina visualizar qual a próxima ação da máquina no Teste 1.

5.2 *Hardware*

O circuito explicado anteriormente, e representado nas Figuras 12 e 13, foi produzido em uma placa de fenolite que pode ser vista na Figura 29.

Figura 29 – *Hardware* desenvolvido na placa de fenolite.



Fonte: Produção do próprio autor.

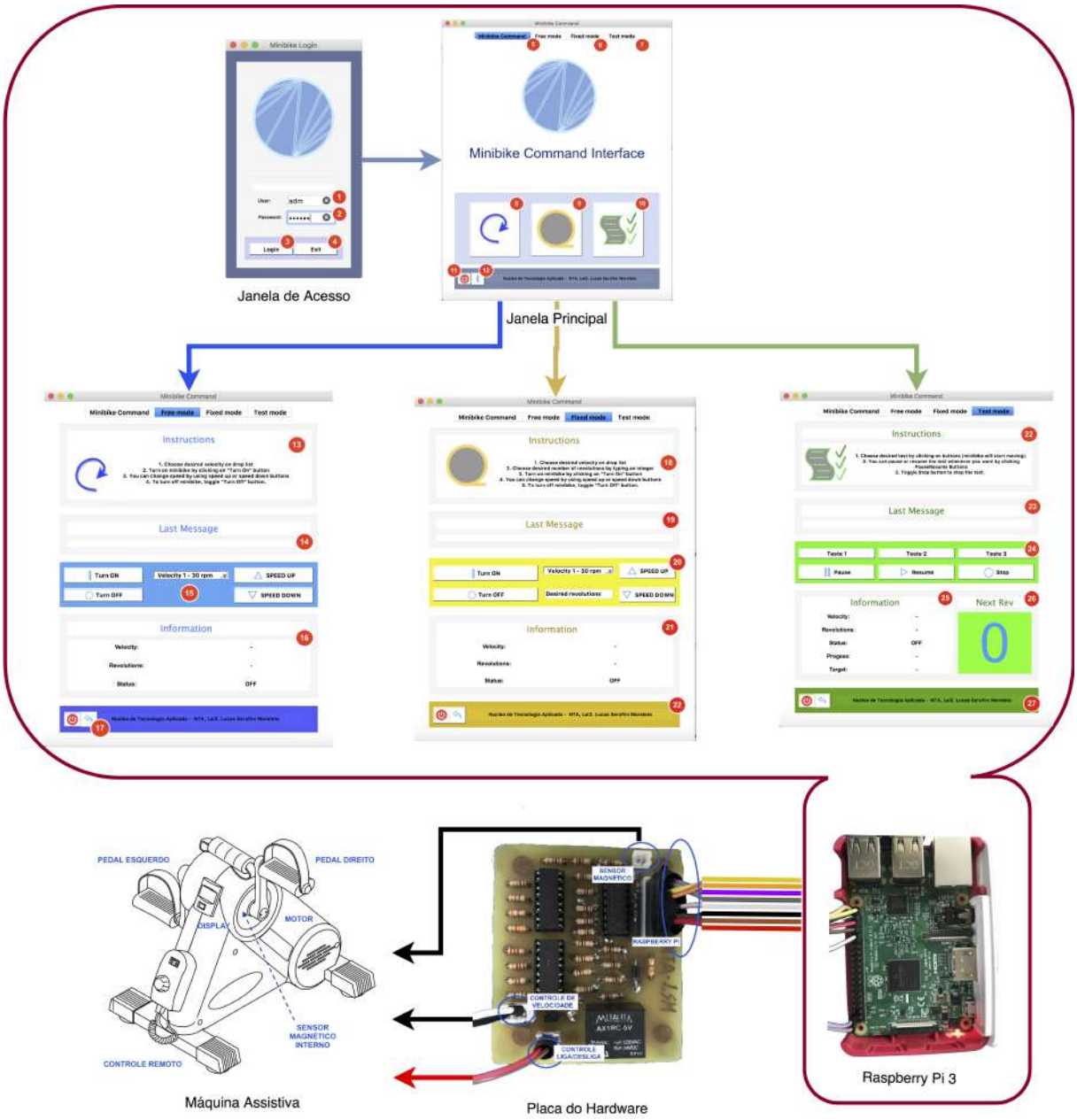
5.3 Diagrama interface - Raspberry Pi 3 - *hardware* - máquina assistiva

O diagrama representativo do sub-sistema final, com as interfaces produzidas e o *hardware*, pode ser visto na Figura 30.

5.4 Utilização do sub-sistema para leitura de dados

Um dos resultados deste projeto de graduação foi sua utilização para a leitura e plotagem da velocidade e posição angular relacionados ao movimento da máquina assistiva captados por um dispositivo IMU (Unidade de Medida Inercial). A IMU possui um giroscópio que foi utilizado para medir a velocidade angular (graus por segundo) do pedal, fixando a placa no pedal direito da máquina assistiva, sendo que a posição angular foi obtida integrando os valores da velocidade no tempo (ROMERO LAISECA et al., 2018).

Figura 30 – Diagrama final do projeto: interface - RPi - *hardware* - máquina assistiva.



Fonte: Produção do próprio autor.

5.5 Apuração da velocidade real e número de revoluções

No intuito de verificar a velocidade da máquina assistiva, foram realizados testes para confrontar com os valores de velocidades em rpm enviados para serem executados na máquina assistiva pelo *software* desenvolvido, de acordo com o quadro de resistores mostrado no Quadro 4. O teste consistiu na medição do tempo que a máquina leva para sair da inércia e atingir cada velocidade desejada, e na medição dessa. A medição foi realizada por meio do *software* desenvolvido junto à IMU, conforme (ROMERO LAISECA

Quadro 7 – Medição das velocidades e tempo de atingimento.

Velocidade Selecionada	Tempo atingimento	Velocidade Real
30 rpm	1.29s	29.5rpm
35 rpm	1.49	36.1 rpm
40 rpm	1.58	56.1 rpm
45 rpm	1.75	45.4 rpm
50 rpm	1.87	50.6 rpm
55 rpm	1.94	55.8 rpm
60 rpm	2.01	61.0 rpm
70 rpm	2.15	70.7 rpm

Fonte: Produção do próprio autor.

et al., 2018). A leitura direta dos dados do giroscópio da IMU enviados via *Bluetooth* para o *software* permite medir a velocidade angular em graus por minuto e converter para rotações por minuto (rpm) diretamente.

Os dados podem ser vistos no Quadro 7. Observe que os dados foram medidos sem carga, ou seja, com pedais livres. O tempo para alcançar a velocidade decorre da inércia que os pedais apresentam ao movimento, e as velocidades dependem diretamente das resistências escolhidas nas medições.

Pode-se observar que as velocidades medidas divergem das enviadas, e isso se dá principalmente pela diferença entre a resistência que deveria ser colocada comparada com a utilizada, devido aos seus valores comerciais, conforme o Quadro 3. Uma das soluções seria colocar um resistor variável em cada valor de resistência, ajustar para a resistência desejada, e colocar na placa. O problema seria o aumento no tamanho da placa do *hardware*, visto que o objetivo foi construir uma placa com o menor tamanho possível, suficiente para anexá-la junto à Raspberry Pi, e acoplá-la à máquina assistiva.

6 CONCLUSÃO

Neste projeto de graduação, uma máquina estática assistiva para terapias de neuro-reabilitação teve sua aplicação incrementada com a aplicação da Raspberry Pi e da interface desenvolvida. A Raspberry Pi permite um comando maior da máquina assistiva, permitindo ao usuário ter um maior poder na utilização, com os modos de operação desenvolvidos. A interface gráfica permite ainda um controle mais dinâmico e interativo da máquina, deixando a usabilidade mais atraente, com informações de número revolução, velocidade e atual status de operação.

No decorrer do projeto foram encontradas algumas dificuldades, como por exemplo entregar os exatos 360° da última volta que é executada pelo pedal antes de parar. A inércia para realizar a parada dos pedais no momento exato que contabiliza a última volta acarretou erros de até 30°, provocando, portanto, que a última volta atingisse até 390°, ao invés de 360°. Algumas abordagens foram realizadas, como gravar o tempo de execução de uma volta para cada usuário cadastrado e para cada velocidade, e utilizar este tempo para definir o número de voltas, ao invés de utilizar a medida do sensor. Porém, o erro continuou alto, sem sucesso.

Algumas melhorias de trabalhos futuros podem ser realizadas no projeto para aprimorar sua funcionalidade, tais como o incremento da interface com geração automática de relatórios. Como o objetivo é atuar na terapia de pacientes, o *software* pode ser incrementado para gerar e salvar relatórios de cada execução realizada, para cada paciente cadastrado no banco de dados já existente. Assim, seria possível desenvolver um sistema de geração de relatório em um padrão pré-definido para entregar ao paciente após o fim de cada utilização da máquina. Quanto à realização de testes, o trabalho pode ser aprimorado, permitindo ao terapeuta criar sua própria versão de teste utilizando, um formulário que cria uma aba para o teste personalizado.

Por fim, o projeto de graduação desenvolvido cumpriu com as necessidades de controle da máquina assistiva para terapias de neuro-reabilitação para sua utilização no sistema em desenvolvimento, com a finalidade de melhorar as habilidades motoras e sensoriais de pacientes com dificuldades motoras nos membros inferiores. Além disso, usou-se uma interface amigável, limpa e concisa para controlar a máquina assistiva.

Vale ressaltar que os resultados obtidos neste Projeto de Graduação foram utilizados para a publicação de um artigo no Congresso Brasileiro de Engenharia Biomédica (CBEB 2018): (ROMERO LAISECA et al., 2018).

BIBLIOGRAFIA

BORGES, Luiz Eduardo. **Python para Desenvolvedores**. 2. ed. [S.l.]: Edição do Autor, 2010. ISBN 9788590945116.

DITUNNO JR, J. F. et al. The Walking Index for Spinal Cord Injury (WISCI/WISCI II): nature, metric properties, use and misuse. **Spinal Cord**, v. 51, n. 6, 2013. DOI: 10.1038/sc.2013.9.

DONATI, Ana R.C. et al. Long-term Training With a Brain-Machine Interface-Based Gait Protocol Induces Partial Neurological Recovery in Paraplegic Patients. **Nature**, v. 79, n. 6, p. 1–16, ago. 2016. DOI: 10.1038/srep30383.

FOURMENT, Mathieu; GILLINGS, Michael R. A comparison of common programming languages used in bioinformatics. **BMC Bioinformatics**, v. 82, n. 9, fev. 2008. DOI: 10.1186/1471-2105-9-82.

GUTIERREZ MIRANDA, M. **The importance of graphic users interface, analysis of graphical user interface design in the context of human-computer interaction**. Barcelona, Spain: IATED, abr. 2011. p. 7137–7144. (3rd International Conference on Education and New Learning Technologies). ISBN 978-84-615-0441-1.

LEE, Edward A. **The Problem with Threads**. [S.l.], jan. 2006. The published version of this paper is in IEEE Computer 39(5):33-42, May 2006. Disponível em: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.html>.

MACHADO, Angelo. **Neuroanatomia Funcional**. 2. ed. [S.l.]: Atheneu Rio, 1993. ISBN 8573790695.

NATIONAL HEART, LUNG, AND BLOOD INSTITUTE. **Stroke, Also known as Cerebrovascular accident**. 2018. Disponível em: <https://www.nhlbi.nih.gov/health-topics/stroke>. Acesso em: 11 dez. 2018.

ORACLE CORPORATION. **MySQL 10.5 cursor.MySQLCursor Class**. 2018. Disponível em: <https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor.html>. Acesso em: 18 nov. 2018.

_____. **MySQL 5.7 Reference Manual: Chapter 1 General Information**. 2017. Disponível em: <https://dev.mysql.com/doc/refman/5.7/en/introduction.html>. Acesso em: 2 dez. 2017.

_____. **MySQL Connector/Python Developer Guide: Chapter 1 Introduction to MySQL Connector/Python**. 2017. Disponível em: <https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html>. Acesso em: 2 dez. 2017.

PYTHON SOFTWARE FOUNDATION. **General Information - What is Python?** 2017. Disponível em: <https://docs.python.org/3/faq/general.html%5C#what-is-python>. Acesso em: 2 dez. 2017.

RIVERBANK COMPUTING. **What is PyQt?** 2016. Disponível em: <https://www.riverbankcomputing.com/software/pyqt/intro>. Acesso em: 1 dez. 2017.

ROMERO LAISECA, M. A. et al. Design and Development of Hardware and Software to Command a Motorized Exercise Static Bike. **Congresso Brasileiro de Engenharia Biomédica**, p. 1–10, out. 2018.

ROUSE, Margaret. **What is database (DB)?** 2017. Disponível em: <http://searchsqlserver.techtarget.com/definition/database>. Acesso em: 2 dez. 2017.

THE LINUX INFORMATION PROJECT. **GUI Definition**. 2004. Disponível em: <http://www.linfo.org/gui.html>. Acesso em: 2 dez. 2017.

WELLNESS, Paradigm Health. **ACTIVcycle Owner's Manual**. [S.l.: s.n.], 2005.

WORLD HEALTH ORGANIZATION, The International Spinal Cord Injury. International Perspectives on Spinal Cord Injury. **World Health Organization**, p. 68, 2013.

A PYTHON

A.1 Variáveis

Python utiliza a tipagem dinâmica, o que significa que para a declaração de variáveis o tipo é inferido pelo interpretador em tempo de execução (BORGES, 2010). Isto é, o tipo da variável não é explicitamente definido na sintaxe do código, e sim de acordo com o dado atribuído na execução. Como pode ser visto no Código A.1, não é necessário informar se a variável que acabou de ser declarada será um texto ou inteiro, por exemplo. Contudo, as conversões entre tipos que não são compatíveis de variáveis não é feita automaticamente, sendo assim necessário explicitar a conversão no código.

Código A.1 – Inicialização de variáveis em Python.

```
variavelum = 10
nome = "Lucas Morelato"
```

A.2 Blocos

Em Python, a estrutura do código é realizada em blocos que são delimitados por meio de endentação (BORGES, 2010). Assim, cada bloco está indentado do outro, sendo o próximo bloco indentado mais a direita que o bloco anterior. Essa organização tem como privilégio o menor uso de símbolos, como por exemplo as chaves usadas na linguagem C que delimitam os blocos, o que torna o código visualmente mais organizado e limpo.

A.3 Funções

Funções são os blocos de código identificados por um nome (BORGES, 2010). As funções podem receber parâmetros pré-determinados e podem retornar ou não alguma informação. São definidas no código pelo termo *def*, que antecede seu nome, passando os parâmetros que receberá entre parênteses (), conforme o Código A.2. Note que o retorno é realizado pelo método *return()*.

Código A.2 – Definição de função em Python.

```
def multiplicar(num1, num2):
```

```
        return(num1*num2)
```

A função poderia realizar a mesma operação sem retornar nada e salvar em uma variável. Porém, a variável deverá ser declarada como *global* dentro da função para que sua alteração seja visível para todas as funções e estruturas do programa, conforme o Código A.3.

Código A.3 – Variável global em Python.

```
def multiplicar(num1, num2):
        global multiplicacao
        multiplicacao = num1*num2
```

A.4 Objetos

Sendo uma linguagem orientada a objeto, as estruturas de dados (objetos - abstrações computacionais) no Python apresentam atributos e métodos que são acessados por meio do ponto (.). A orientação a objeto é bastante utilizada na Interface deste trabalho para acessar atributos das telas como, por exemplo, botões e realizar ações nos mesmos (métodos). No Código A.4 pode ser visto um exemplo de uso de um atributo e como executar um método.

Código A.4 – Atributo de um objeto em Python.

```
print objeto.atributo #utilizando um atributo
objeto.metodo(argumentos) #utilizando um metodo
```

A.5 Classe

Classe é uma estrutura básica das linguagens orientadas a objetos (BORGES, 2010), representando o tipo de objeto e definindo um modelo representativo dos objetos que serão dela. Por exemplo, criar uma classe para representar motos (objetos) e definir atributos como cor, modelo, ano, marca e outros. A classe é definida pelo termo *class* logo antes do nome que a identificará. Para criar um objeto da classe, basta declarar o objeto chamando a classe com seu nome e os parâmetros a serem enviados. Note que toda vez que um objeto é criado uma função de inicialização (`__init__()`) da classe será executada. No Código A.5 a estrutura de uma classe é exibida. Observe que *self* é uma variável que representa o objeto na classe. Apesar de fortemente usado, *self* pode ser trocado por qualquer outro nome.

Código A.5 – Estrutura de uma Classe em Python.

```
class Moto():
        #Classe das motos
        def __init__(self, argumentos):
        #inicializador da classe
        <codigo>

        def cor(self, parametros):
        #metodo do objeto

SuzukiGSX = Moto() #definir um objeto da classe
SuzukiGSX.cor() #utilizar o metodo da classe
```

A.6 Módulos

No Python, módulos são arquivos fontes que podem conter qualquer estrutura e que são importados para um programa (BORGES, 2010). Os módulos ajudam na execução de tarefas e tornam o programa principal mais limpo e organizado, incrementando a produtividade do código. Para importar um módulo para o programa basta utilizar o termo *import* seguido do nome do módulo conforme no Código A.6

Código A.6 – Importação de módulos em Python.

```
import time
import os
import threading
```

Um módulo pode ser facilmente criado a partir de um código Python em um arquivo *.py* separado; o Código A.7 é um exemplo de arquivo chamado *media.py*.

Código A.7 – Código que calcula a média de dois números.

```
def media(a,b):
        return (a+b)/2.0
```

Desta forma, em um outro código pode-se utilizar o módulo "media" conforme o Código A.8.

Código A.8 – Utilização do módulo média.py em outro código.

```
from media import media # importando a funcao media do modulo media.py
print(media(10,8))
```

A.7 *Threading*

Em Python, o módulo da biblioteca padrão de *thread* é a *threading.py*. Essa biblioteca provê classes de alto nível de abstração e usa o módulo *thread*. Cada *thread* deve ser construída em uma classe composta pelas funções de inicialização, e a função *run* que representa a atividade da *thread*, e é executada quando o método *start* da classe é chamado. A biblioteca utilizada no projeto é a *QThread*, dos mesmos desenvolvedores da biblioteca utilizada para desenvolver a interface do projeto. A vantagem do uso da *QThread* são os mecanismos de uso de sinais para conectar o funcionamento do código da *thread* com os elementos da interface, que utiliza a biblioteca *PyQT5*. O código que é executado pela *thread* fica dentro da função *run*, conforme o Código A.9. Note que a *thread QThread* é uma classe.

Código A.9 – Estrutura de uma classe com *threading*.

```
class maquinaAssistiva(QThread):
        def __init__(self):
                # Construtor da thread
        def run(self):
                # Codigo que sera executado pela thread
```

Uma *QThread* pode ser inicializada utilizando o método *start()* que a *QThread* apresenta. Por outro lado, ela pode ser parada ou finalizada utilizando *stop()* ou *quit()*, e seu status pode ser verificado utilizando os atributos *isFinished()* e *isRunning()*.

A comunicação entre *QThreads* pode ser realizada de forma segura utilizando sinais que são emitidos pela *thread*. Esses sinais são declarados utilizando a função *pyqtSignal()*, passando como parâmetro o tipo da variável que será transmitida no sinal conforme o Código A.10.

Código A.10 – Declaração de sinais em Python com pyqt.

```
#declaracao do sinal que sera realizado utilizando texto (string) - str
numeroDeVoltas = pyqtSignal(str)
```

Dentro da função *run()*, a qualquer momento o sinal pode ser emitido utilizando o método do sinal criado *emit()*, passando como parâmetro a variável que será emitida, conforme o Código A.11

Código A.11 – Utilizando um sinal para transferência de dados em Python com *threading*.

```
#emitir a mensagem "5" pelo sinal numeroDeVoltas
numeroDeVoltas.emit("5")
```

Para receber a mensagem fora da *thread* deve-se utilizar um método que consiste na identificação automática quando a mensagem é recebida e posterior execução de uma função declarada para tratar esta mensagem. Para isso, é necessário vincular o sinal a uma função específica para ela. Assim, toda vez que uma mensagem for emitida, conforme o Código A.11, a função associada é executada. Este método é muito utilizado com a biblioteca *PyQT5* para atualização da interface utilizando sinais (mensagens) recebidos, por exemplo, da execução de uma máquina que está sendo controlada por uma *thread*.

Para associar uma função a ser executada pelo sinal recebido de uma *thread*, o método *connect()* é utilizado, passando como parâmetro a função que será executada automaticamente ao receber o sinal, conforme o Código A.12

Código A.12 – Associando função a um sinal.

```
#conectar a funcao atualizarNumVoltas ao sinal numeroDeVoltas
numeroDeVoltas.connect(atualizarNumVoltas)
```

A.8 Banco de Dados

No projeto, o banco de dados MySQL foi programado utilizando a linguagem SQL por código em Python intermediado por um conector chamado MySQL Conector/Python (ORACLE CORPORATION, 2017b). O uso do conector permite o código Python enviar comandos em linguagem SQL para acessar, criar e atualizar as tabelas no MySQL. O pacote já existente responsável por isso no Python é o *pymysql*, o que apresenta a função *connect()* que permite criar o conector passando como parâmetros o *host*, o usuário do banco de dados, a senha do banco de dados e o nome do banco de dados criado, conforme o Código A.13 de (BORGES, 2010).

Código A.13 – Inicialização e conexão entre Python e MySQL utilizando pymysql.

```
import pymysql
# Cria uma conexao
con = pymysql.connect(db='test', user='root', passwd='')

# Cria um cursor
cur = con.cursor()

# Executa um comando SQL
cur.execute('show databases')

# Recupera o resultado
recordset = cur.fetchall()
```

```
# Mostra o resultado
for record in recordset:
        print record

# Fecha a conexao
con.close()
```

Note que é preciso de um cursor para executar operações que utilizam a linguagem SQL dentro do código Python, pois é o cursor que interage com o servidor MySQL por meio do conector (ORACLE CORPORATION, 2018).

A.9 Interface

Como exemplo do uso do módulo PyQT com o *software* QtDesigner, abaixo temos o exemplo realizado por (BORGES, 2010) para a criação de uma simples janela com dois botões, conforme a Figura 31.

Figura 31 – Janela exemplo com dois botões no QtDesigner.

Fonte: (BORGES, 2010).

Chamando este módulo de Dialog.py, têm-se dois botões que são objetos da classe *QPushButton* chamados de "msg" e "fim", respectivamente, e a janela, que é um objeto da classe *QDialog*. Essas classes já são definidas para cada tipo de elemento que o PyQT suporta, sendo assim, cada uma apresenta seus atributos característicos que devem ser analisados para levantar suas ações possíveis.

Com a criação dos objetos dessas classes, funções *callback* são associadas para realizar tratativas de eventos, conforme o Código A.14 (BORGES, 2010).

Código A.14 – Exemplo de uma classe de interface e sua inicialização em Python utilizando PyQT.

```
import sys
```

```
import time
from PyQt4 import QtCore, QtGui

# Modulo gerado pelo pyuic
from dialog import Ui_Dialog

class Main(QtGui.QMainWindow):
# janela principal
        def __init__(self, parent=None):
        # Inicializacao da janela
        QtGui.QWidget.__init__(self, parent)

        # Cria um objeto a partir da interface gerada pelo pyuic
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)

        #Conecta o metodo ao botao que foi definido atraves do Qt Designer
        self.connect(self.ui.msg, QtCore.SIGNAL('clicked()'), self.show_msg)

        def show_msg(self):
        #metodo para abrir uma caixa de mensagem
                reply = QtGui.QMessageBox.question(self, 'Messagem', 'Hora: '
                + time.asctime().split()[3],            QtGui.QMessageBox.Ok)

if __name__ == "__main__":
app = QtGui.QApplication(sys.argv)
myapp = Main()
myapp.show()
sys.exit(app.exec_())
```

Observe que neste exemplo é preciso inicialmente importar o módulo em Python gerado do XML (.ui) do QtDesigner que, por padrão, é nomeado do mesmo nome da janela precedido de "Ui_". Na classe principal (*Main*) é preciso inicializar a janela QtGui e criar o objeto que vai representar o módulo criado pelo QtDesigner. Com o objeto criado, no caso self.ui, os atributos deste objeto, que são os botões e as janelas, podem ser acessados e associados a funções *callback*, o que é feito associando a função *show_msg()* ao botão "msg" (Mensagem) quando ele sofre o evento de ser clicado (*'clicked()'*). Observe na Figura 32 o retorno da interface quando este botão é clicado.

Figura 32 – Janela exemplo e retorno da função *callback* criada.

Fonte: (BORGES, 2010).