Universidade Federal de Uberlândia
Faculdade de Engenharia Elétrica
Curso de Engenharia Biomédica

**THAÍLA FERREIRA ZARUZ**

# APLICAÇÃO DA LEI DE FITTS NA AVALIAÇÃO DE UM SISTEMA ROBÓTICO CONTROLADO POR JOYSTICK

Uberlândia
2020

**THAÍLA FERREIRA ZARUZ**

# APLICAÇÃO DA LEI DE FITTS NA AVALIAÇÃO DE UM SISTEMA ROBÓTICO CONTROLADO POR JOYSTICK

Trabalho apresentado como requisito parcial de avaliação na disciplina Trabalho de Conclusão de Curso do Curso de Engenharia Biomédica da Universidade Federal de Uberlândia.

Orientador: Prof. Dr. Adriano de Oliveira Andrade

_____

Assinatura do Orientador

Uberlândia

2020

Dedico este trabalho a minha família. O amor, carinho, confiança e incentivo de vocês me fez chegar aqui. Sem vocês eu nada seria.

## AGRADECIMENTOS

Ao Prof. Adriano de Oliveira Andrade pela orientação, motivação e oportunidade em realizar este trabalho e durante todo o período na universidade.

À minha família, Maria José, Sali, Thael, Thassio, avós, tios e tias por todo amor, apoio, carinho, paciência e diversas opiniões ao longo de toda minha trajetória. Agradecimento especial aos meus irmãos Thael e Thassio, pelas ideias e auxilio no desenvolvimento de diversas etapas deste trabalho.

Aos meus amigos Mariana, Ludimila, Amanda, Natan e Wilker pelas risadas, puxões de orelha e conselhos para a vida.

Ao meu namorado, João, pelo carinho, incentivo e paciência por me ouvir falar sobre o mesmo assunto diversas vezes.

Ao Núcleo de Inovação e Avaliação Tecnológica em Saúde – NIATS, pelo conhecimento, amizades e oportunidade de crescimento.

A todos que me apoiaram e contribuíram de alguma maneira para o desenvolvimento deste trabalho.

À Sociedade Brasileira de Engenharia Biomédica, pela doação do braço robótico utilizado neste trabalho.

À Universidade Federal de Uberlândia, por propiciar tamanha oportunidade de aprendizado e desenvolvimento.

## RESUMO

Este trabalho apresenta o uso de um sistema de braço robótico, composto por braço robótico, fonte alimentadora e caixa controladora, em que o usuário realiza o controle com movimentos de *joystick* programados utilizando um arduino. A fim de apresentar uma aplicação para o sistema, tarefas relacionadas à Lei de Fitts foram executadas. Para isso foi implementada uma interface gráfica em C#, com uma página para preenchimento de informações do pesquisador e do voluntário e outra página com informações sobre a execução das tarefas, fornecendo feedback ao voluntário. Dados adquiridos em testes realizados pela presente autora, utilizando a interface gráfica, foram salvos em um banco de dados SQL local para facilitar o seu acesso posterior. As informações coletadas foram analisadas, comparando os diferentes objetos e índices de dificuldade de acordo com os critérios da Lei de Fitts. Dessa forma, a usabilidade do sistema foi avaliada, juntamente com sua ampla variedade de aplicações, podendo auxiliar os alunos do curso de Engenharia Biomédica da Universidade Federal de Uberlândia.

## ABSTRACT

This work presents the development of robotic arm system, composed of a robotic arm, power supply and control box, in which the user performs the control with joystick movements programmed using an Arduino. In order to illustrate the application of the system, tasks related to Fitts' Law were performed. For this, a graphical interface was implemented in C#, with a page to fill in information from the researcher and the volunteer and another page with information about the execution of the tasks, providing feedback to the volunteer. Data acquired by the author during tests with the graphical interface was saved to a local SQL database for easing later access. The information collected was analyzed, comparing the different objects and difficulty indexes according to the criteria of Fitts' Law. Thus, the usability of the system was assessed, together with its wide variety of applications, and can assist students of the Biomedical Engineering course of the Federal University of Uberlândia.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| *a* | Coeficiente angular |
| *b* | Coeficiente linear |
| CBEB | Congresso Brasileiro de Engenharia Biomédica |
| DETEB | Desafio Temático em Engenharia Biomédica |
| GUI | *Graphical User Interface* |
| I2C | *Inter-Integrated Circuit* |
| ID | Índice de dificuldade |
| LF | Lei de Fitts |
| $r^2$ | Coeficiente de determinação |
| SBEB | Sociedade Brasileira de Engenharia Biomédica |
| SDA | *Serial Data* |
| SCL | Serial Clock |
| SQL | *Standard Query Language* |
| TM | Tempo de movimento |
| UFU | Universidade Federal de Uberlândia |
| WIMP | *Windows,* ícone, menus, ponteiro |

# SUMÁRIO

# 1. INTRODUÇÃO

## 1.1. Motivação

O estudo do corpo humano, seja ela físico ou psicológico, é amplamente requisitado para o entendimento do seu funcionamento. O estabelecimento de modelos para explicar e sistematizar alguns comportamentos é o objetivo de diversas pesquisas, uma delas tendo sido a de Paul Fitts. Ele formulou uma equação para modelar o movimento humano, relacionando o tempo de execução de uma tarefa com a dificuldade em executá-la (WU; YANG; HONDA, 2010).

Além disso, a evolução da tecnologia faz com que estudos acerca da interação entre humanos e robôs sejam explorados. Com isso, diversos aspectos dessa interação podem ser abordados, como o método de controle ou a replicação de comportamentos humanos em robôs.

Observando esses fatos, e visando aplicar conceitos aprendidos durante a graduação em Engenharia Biomédica, o presente trabalho propõe a elaboração de um sistema de um braço robótico controlado por *joystick* e sua aplicação na execução de tarefas avaliadas pela Lei de Fitts.

## 1.2. Objetivo geral

Avaliação de um sistema composto por um braço robótico, caixa controladora com joystick e interface gráfica por meio da aplicação da Lei de Fitts, analisando os resultados quando aplicados objetos e índices de dificuldade diferentes.

## 1.3. Objetivos específicos

De forma a alcançar o objetivo citado, foram definidos os seguintes objetivos específicos:

1) Montar e controlar o braço robótico.
2) Desenvolver interface gráfica para facilitar a interação com o usuário.
3) Criar um banco de dados para armazenar informações relacionadas ao experimento.
4) Calcular os parâmetros da Lei de Fitts com os dados adquiridos.

5) Analisar os parâmetros de acordo com os índices de dificuldade e dimensões de diferentes objetos.

# 2. REVISÃO BIBLIOGRÁFICA

## 2.1 Manipuladores robóticos

O constante avanço tecnológico faz com que a definição de robô seja frequentemente atualizada. Apesar disso, uma definição amplamente aceita advém da Associação de Indústrias de Robótica (*Robotic Industries Association* – RIA) (1979) citada por Barrientos et al. (2014, p. 17), em que “Um robô industrial é um manipulador reprogramável multifuncional, capaz de mover materiais, peças, ferramentas ou dispositivos especiais, conforme trajetórias variáveis, projetadas para executar diferentes tarefas.”

Um robô é constituído de um corpo rígido com elos e juntas, onde as juntas são os pontos móveis. Os atuadores são os responsáveis por movimentar as juntas, podendo ser elétricos, pneumáticos ou hidráulicos, sendo os servo-motores um tipo comum de atuador. A sua ponta ou *end-effector* é o componente que geralmente determina a aplicação do dispositivo, visto que está conectado à extremidade do braço e possui funções específicas como pegar, soldar, pulverizar tinta, entre outras. A sua movimentação é medida em graus de liberdade, que representa a quantidade de movimentos independentes que o dispositivo pode realizar (SANTOS, 2003).

A variação destes componentes permite a construção de robôs com diferentes aplicações, assim eles são classificados de acordo com sua funcionalidade, podendo ser robôs humanoides, domésticos, manipuladores, exploradores, médicos, entre outros (BARRIENTOS et al., 2014). Os robôs podem ser usados em laboratórios, hospitais, residências, universidades, indústrias (automotiva, construção, militar, espacial, energia, mineração, transporte, entretenimento), na manipulação de materiais radioativos, em segurança, serviços de comida ou na interação direta com o ser humano, reproduzindo seu comportamento (INC. STAFF, 2020).

Além dos componentes, o método de controle do manipulador também varia, necessitando de controle direto de um ser humano ou sendo controlado via inteligência artificial de acordo com sua programação (INC. STAFF, 2020).

Quando o controle é realizado por um operador, dispositivos como joysticks, sensores flexíveis, acelerômetros e eletromiografia são comumente utilizados (BAJERSKI; ABELLA, 2010; BARBOSA; SILVA, [s.d.]; COSTA et al., 2014). Já quando a operação do manipulador robótico é independente, o uso de sensores para

a percepção do ambiente auxilia na execução adequada da tarefa. Aqueles mais usados são sensores de força, fim de curso, proximidade, visuais, de posição, velocidade e torque (INC. STAFF, 2020; SANTOS, 2003).

Para o desenvolvimento de inovações na área da robótica, os robôs com fins acadêmicos são essenciais. Estudos envolvendo novos métodos de controle, comparativos entre diferentes tipos de atuadores e montagens ou para a realização de experimentos envolvendo a interação homem-máquina são ótimas áreas para que os estudantes desenvolvam novas técnicas e explorem mais esse tópico. Logo, a presença de robôs no dia a dia se torna cada vez mais frequente, visando sempre a melhoria de processos e o auxílio ao ser humano na execução de tarefas.

## 2.2 Lei de Fitts

Elaborada por Paul Fitts em 1954, a Lei de Fitts (LF) relaciona o tempo de movimento para execução de uma tarefa com a dificuldade em realizá-la, sendo um modelo de movimento humano. Em seu experimento, voluntários deveriam mover as mãos entre dois alvos, com tamanho e amplitudes diferentes, sendo estes os parâmetros usados para definir o índice de dificuldade (ID) (PAUL M. FITTS, 1954). A equação (1) demonstra uma das formas de calcular o ID (WU; YANG; HONDA, 2010), onde A é a amplitude do movimento, ou seja, a distância entre os centros dos alvos e W é a largura do alvo.

$$ID = log_2 \frac{2A}{W} \tag{1}$$

Observando o ID e o tempo de execução da tarefa ou tempo de movimento (TM), Fitts percebeu uma correlação linear entre eles, sendo então equacionado com base em uma função de primeiro grau, demonstrada na equação (2) (ALIMI, 1997).

$$TM = a * ID + b \tag{2}$$

Nela temos as constantes *a* e *b*, sendo *a* o coeficiente angular e *b* o coeficiente linear. Ambos são definidos empiricamente, uma vez que TM é medido durante o experimento e ID é calculado para a realização do experimento.

Com os experimentos, notou-se que conforme o ID aumentava, o TM também aumentava, visto a sua correlação linear. Além disso, quanto maior a necessidade de acurácia que a tarefa exige, menor é a velocidade para realizar o movimento, sendo que a diminuição do tamanho do alvo ocasiona em maior necessidade de acurácia (OKAZAKI et al., 2011). Essa relação é chamada de troca velocidade-acurácia.

Ao longo dos anos, a Lei de Fitts foi aplicada em diversos tipos de experimentos, sendo comprovada a sua aplicabilidade e robustez em modelar o movimento humano. Diferentemente do experimento original, a LF já foi aplicada em testes utilizando movimentos das mãos, pulsos, pés, cabeça e dedos, ou possuindo dispositivos de manipulação como mouse, *joystick*, pedais, teclado, *touchpad* e caneta (ALIMI, 1997; ANDRES; HARTUNG, 1989; CARD; ENGLISH; BURR, 2007).

Além dessas aplicações, com o avanço da tecnologia, a LF passou a ser aplicada em contextos digitais, sendo usada em estudos de experiência do usuário e design de interface do usuário. Desse modo ela auxilia programadores a criarem interfaces mais voltadas aos usuários, com menus e botões posicionados estrategicamente de acordo com a sua função e grau de importância, suprindo os interesses do usuário e tornando a interação mais intuitiva e ágil (IXDF COURSE INSTRUCTOR, 2019).

## 2.3 Interface Gráfica

Nos primórdios da computação, as interfaces de comunicação entre o usuário e o computador eram o teclado, o cartão perfurado e o *teletype*, sendo que eram voltados apenas para a execução da tarefa, sem preocupação com a facilidade de uso e entendimento do usuário, limitando o seu acesso a programadores e pesquisadores (SILVEIRA; RIBEIRO, 2013).

Com o aumento de pesquisas na área e o enfoque sendo dado também ao usuário, visto a nova possibilidade de tornar o computador um dispositivo comercial, novas interfaces surgiram, como o mouse, telas *touchscreen*, *touchpad*, canetas ópticas. As informações apresentadas no computador também evoluíram, com a criação de símbolos e ícones para programas, menus *popup*, cabeçalhos de janelas,

botões, caixas de diálogo, programas para edição de texto e criação de imagens (SILVEIRA; RIBEIRO, 2013).

Um exemplo de interface gráfica do usuário (GUI, do inglês *graphical user interface*) é o que foi utilizado no computador Star, da empresa Xerox, no início da década de 1980. Ele seguia o paradigma *Windows,* ícones, menus, ponteiros (WIMP), onde a tela simulava itens de um escritório e o usuário fazia associações facilmente entre os ícones e funções do computador com os itens do escritório, tornando o uso intuitivo (SILVEIRA; RIBEIRO, 2013; SILVEIRA; RIBEIRO; FOLLE, 2014).

Sobre o paradigma WIMP, temos:

- *Windows:* O uso de janelas independentes, organizadas pelo usuário conforme preferência;
- Ícones: Símbolos que representam documentos ou ferramentas;
- Menus: Listas de comandos organizadas, onde a seleção de um item pode apresentar submenus;
- Ponteiros: Indica na tela o ponto de ação do usuário, permitindo editar ou mover objetos (SILVEIRA; RIBEIRO, 2013).

Esse paradigma obedece aos princípios básicos para o design de uma GUI, sendo 10 heurísticas estabelecidas por Jakob Nielsen, citadas abaixo (LEMES, 2018; MACEDO, 2017):

1) Visibilidade do status do sistema: A interface deve sempre mostrar as informações do que está acontecendo;
2) Compatibilidade com o mundo real: A linguagem apresentada deve ser familiar ao usuário;
3) Controle do usuário e liberdade: O usuário deve poder voltar e/ou cancelar opções que o conduzem a um erro;
4) Consistência e padronização: Não variar em termos e símbolos para significados idênticos, mantendo uniformidade;
5) Prevenção de erros: Avisar quando alguma interação pode ocasionar algum erro;
6) Reconhecimento ao invés de memorização: Diminuir a utilização da memória, mantendo ícones e outros objetos sempre visíveis;

7) Flexibilidade e eficiência do uso: Permitir aos usuários adaptar ações frequentes, ou seja, o uso de atalhos;
8) Estética e design minimalista: Evitar o uso de informações desnecessárias, deixando assim a quantidade de informação em tela menor;
9) Ajudar os usuários a reconhecerem, diagnosticarem e recuperarem-se de erros;
10) Ajuda e documentação: Mesmo quando não necessária, sempre é bom manter material de ajuda em lugar acessível e que permita rápida e fácil compreensão.

O uso das heurísticas por desenvolvedores auxilia na criação de GUI, uma vez que ela é voltada para a sua aplicação, seu público e suas funcionalidades, minimizando assim a sua rejeição por parte dos usuários.

Com o aumento do acesso à tecnologia, interagimos com diferentes GUIs por meio dos smartphones, notebooks, *smart* TVs, games e até mesmo caixas eletrônicos. Assim, o seu uso em pesquisas também se tornou comum, visando tanto estudar a interação entre usuário e interface para entender melhor o comportamento humano e aprimorar as interfaces, quanto a aplicação em pesquisas como meio de interagir com o voluntário para cumprir as finalidades do estudo, variando desde o uso em reabilitação e área da saúde até facilitar a execução de cálculos (ALLOUCHE, 2012; CARDOSO; GONÇALVES; OLIVEIRA, 2013; MACHADO; MORAES; NUNES, 2009; MAGEZI, 2015).

# 3. METODOLOGIA

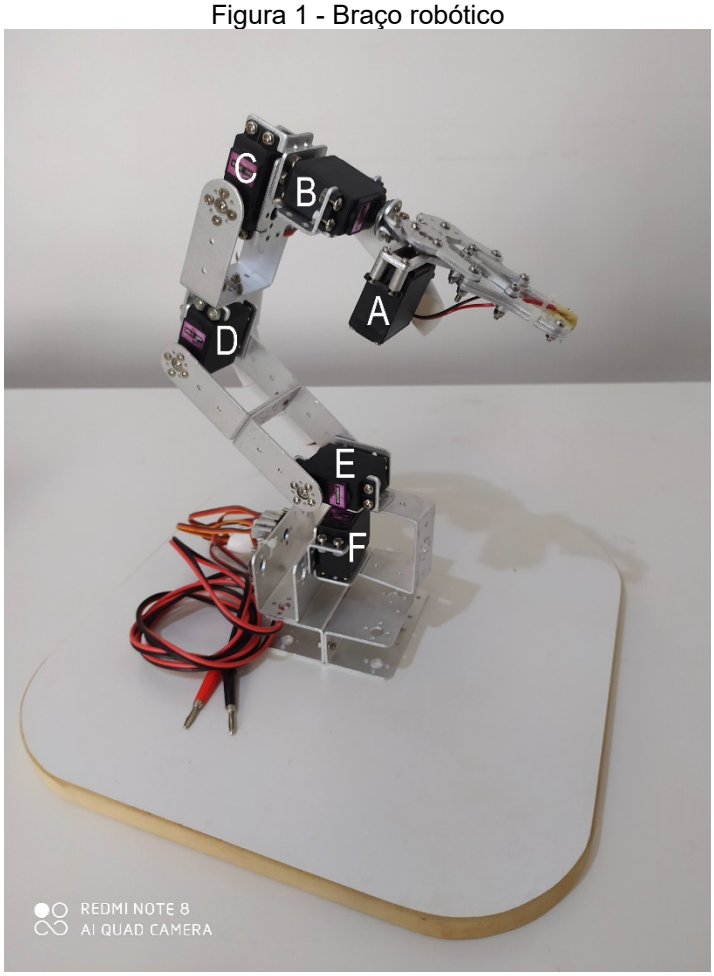
Para a realização deste trabalho, foi utilizado um braço robótico como instrumento na execução do experimento da Lei de Fitts. Foram implementados também um banco de dados e uma interface gráfica para o usuário.

## 3.1 Manipulador robótico

O braço robótico ROT2U 6DOF foi fornecido pela Sociedade Brasileira de Engenharia Biomédica (SBEB) durante o Congresso Brasileiro de Engenharia Biomédica em 2018 (CBEB 2018) para participação no primeiro Desafio Temático em Engenharia Biomédica (DETEB). Após a conclusão do congresso, o braço foi doado ao Núcleo de Inovação e Avaliação Tecnológica em Saúde da Universidade Federal de Uberlândia (UFU). O dispositivo possui seis servo-motores MG 996R Tower Pro, energizados por uma fonte de alimentação chaveada de 5 V 10 A, que são fixados à estrutura de alumínio do manipulador, permitindo sua movimentação. Para sua identificação, os motores foram nomeados de A à F, sendo A o motor da garra e F se referindo à base.

A montagem do braço foi realizada conforme guia de montagem fornecido pelo CBEB, observando qual configuração apresentaria maior estabilidade durante movimentos complexos. Uma base de madeira foi confeccionada para sustentar e garantir que o braço permanecesse com a base fixa durante a execução de qualquer tipo de movimento, mesmo com o deslocamento de seu centro de massa. A Figura 1 mostra a montagem final do braço robótico.

Figura 1 - Braço robótico



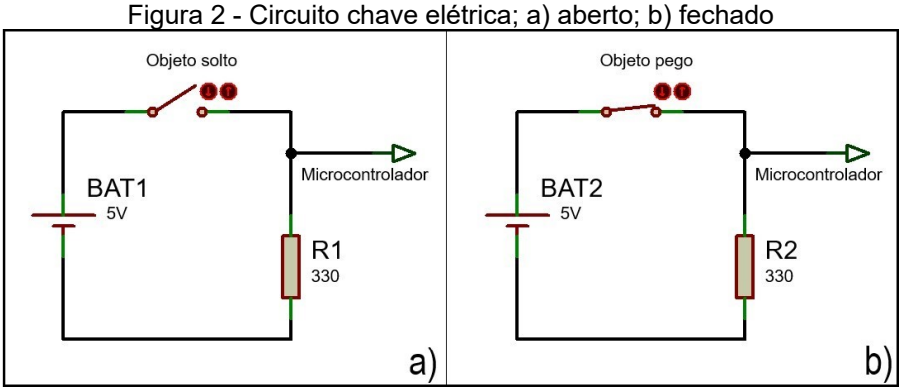

Fonte: Autoria própria (2020).

Foi acrescentado à estrutura da garra do braço um sistema simples de chave elétrica, que serve para identificar se um objeto foi pego pela garra ou não. Nesse sistema o objeto funciona como chave, ou seja, quando ele é pego, o circuito fecha, permitindo a passagem de corrente elétrica para o microcontrolador, e quando o objeto é solto, o circuito é aberto e o sinal para de ser enviado. A Figura 2 mostra o funcionamento do sistema, onde em a) temos a chave elétrica em aberto, ou seja, a garra não está com o objeto e então não está enviando sinal para o microcontrolador; e em b) onde a garra está com o objeto e o sinal está sendo enviado ao microcontrolador, uma vez que o circuito está fechado permitindo a passagem de corrente elétrica.

Para o seu funcionamento, é essencial que o objeto utilizado seja condutor de eletricidade. Devido à aplicação da Lei de Fitts para avaliar o sistema, a identificação do momento em que um objeto é pego ou solto é essencial para a aquisição do tempo

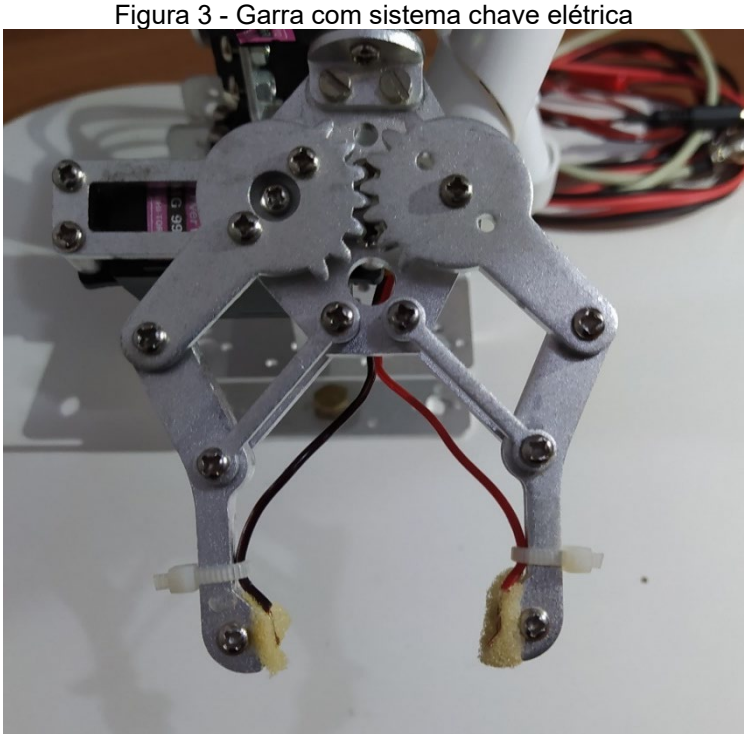
de execução de uma tarefa. A Figura 3 mostra em detalhe a garra com o sistema de chave elétrica implementado.

Figura 2 - Circuito chave elétrica; a) aberto; b) fechado




Fonte: Autoria própria (2020)

Figura 3 - Garra com sistema chave elétrica

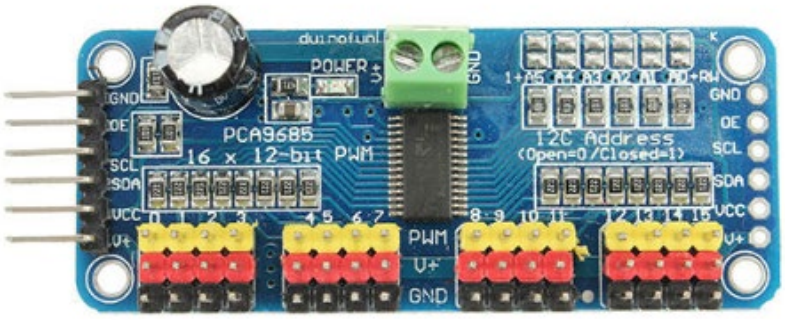


Fonte: Autoria própria (2020)

Com o objetivo de reduzir o número de entradas digitais dos motores, o módulo PWM PCA9685 foi adicionado conforme Figura 4, auxiliando na alimentação e controle dos servos. Ele possui 16 entradas PWM para servo-motor, com interface de comunicação *Inter-Integrated Circuit* (I2C) para comunicação com microcontroladores, de forma que são necessários apenas dois fios, sendo *Serial Data*

*(*SDA) e *Serial Clock (*SCL), para transmitir e receber dados e controlar todos os motores do braço (ELETRONICA, [s.d.]).

Figura 4 - Módulo PWM PCA9685




Fonte: Eletrogate.com

### 3.2 Controle

Para realizar o envio de comandos para o braço robótico e receber dados acerca do objeto, foi utilizado a plataforma arduino, cuja características estão descritas na Tabela 1.

Tabela 1 - Especificações técnicas da plataforma arduino uno

| ESPECIFICAÇÕES TÉCNICAS | |
|---|---|
| Microcontrolador | Atmega328P |
| Tensão de operação | 5 V |
| Tensão de alimentação | 7 – 12 V |
| Pinos de entrada/saída digitais | 14 pinos (sendo 6 deles PWM) |
| Pinos de entrada analógica | 6 pinos |
| Comunicação I2C | SDA pino A4/ SCL pino A5 |
| Memória flash | 32 KB |
| Velocidade do clock | 16 MHz |
| Comprimento | 68,6 mm |
| Largura | 53,4 mm |
| Peso | 25 g |

Fonte: arduino.cc

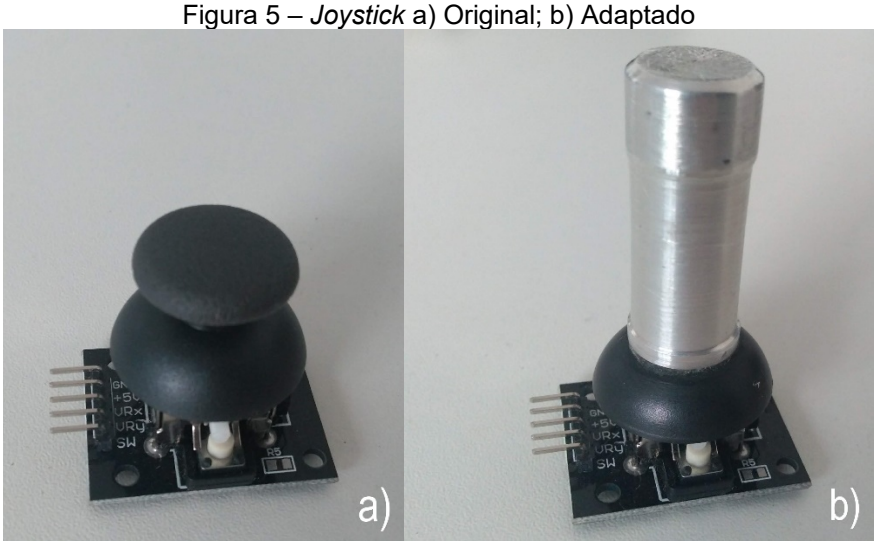
Foi usado também o joystick KY-023 para arduino para que o usuário pudesse controlar os movimentos a serem realizados pelo braço. Nele, os controles referentes aos eixos X e Y são potenciômetros, cujo valor varia de acordo com a posição e

direção que o usuário manipula. Além disso, possui um botão representado como eixo Z, que é acionado ao ser pressionado.

Nessa aplicação apenas o eixo X do *joystick* é acionado, uma vez que, durante os testes, o eixo Y não apresentou a estabilidade correta para ser utilizado. O botão do joystick também é utilizado para realizar o controle do braço robótico.

Visto que o manipulador do *joystick* é pequeno, com pouca superfície de contato para o usuário, foi realizada uma adaptação, onde um tubo de alumínio foi utilizado para substituir uma parte do manipulador e facilitar o seu manuseio pelo usuário. O *joystick* adaptado pode ser comparado com o original na Figura 5

Figura 5 – *Joystick* a) Original; b) Adaptado

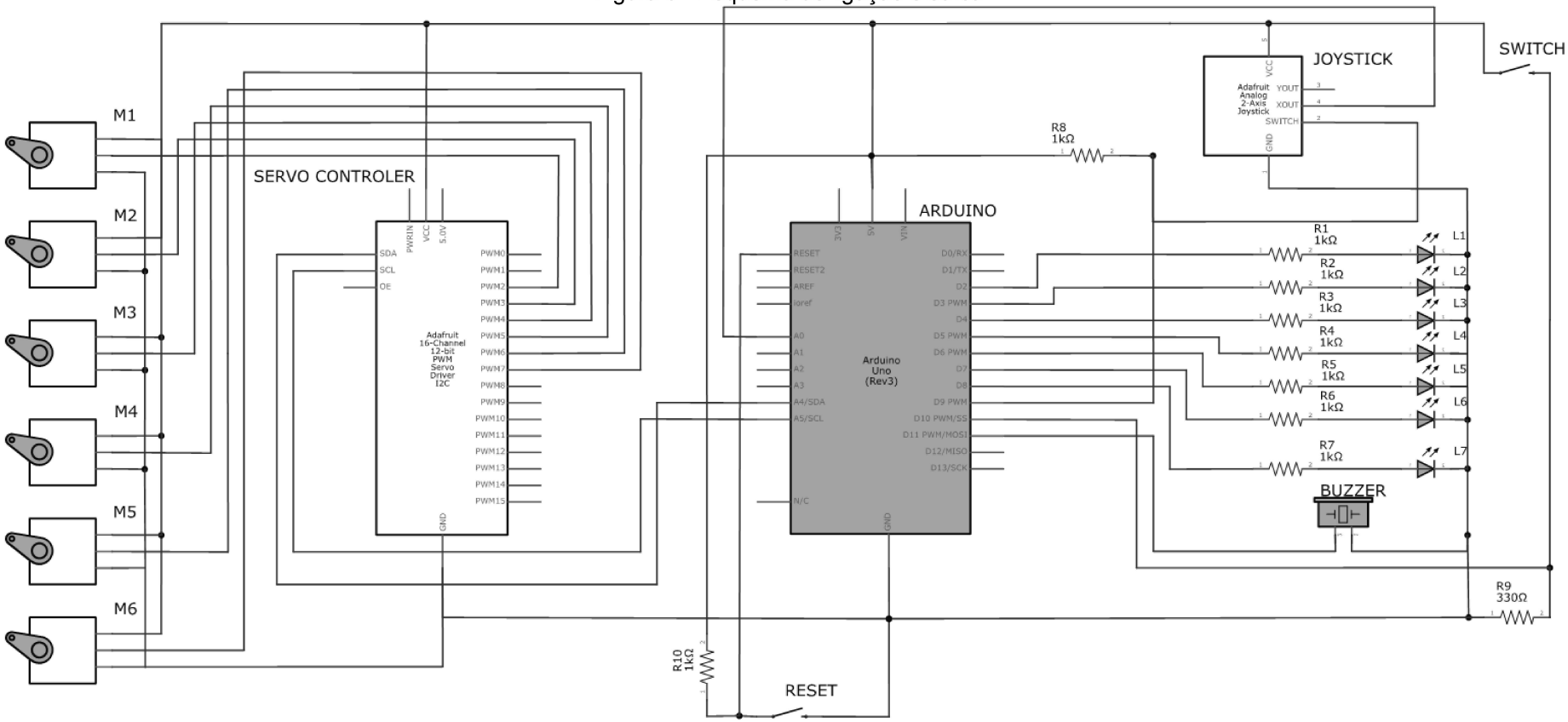



Fonte: Autoria própria (2020)

Foi feita uma caixa de plástico para armazenar os componentes do circuito de forma apropriada. De forma que o usuário tivesse *feedback* independente de uma interface gráfica, também foram adicionados leds para indicar cada motor e se ele está em movimento. Um *buzzer* também faz parte do sistema. O esquema de ligação elétrica dos componentes é representado na

Figura 6.

Figura 6 - Esquema de ligação elétrica




Fonte: Autoria própria (2020).

### 3.2.1 Programação arduino

A programação foi feita no IDE do Arduino v. 1.8.10. Foram utilizadas as bibliotecas *Wire* (ARDUINO, 2016) para uso do protocolo I2C e a biblioteca *Adafruit_PWMServoDriver* (ADAFRUIT, 2020) para efetuar o controle dos motores servo por meio da placa PCA9685. O código comentado pode ser consultado no Apêndice A (ZARUZ, 2020a).

O sistema de controle funciona dividido em dois modos: modo escolha e modo movimento. O programa inicia no modo escolha, onde será escolhido o motor a ser movimentado. Para isso o usuário utiliza movimentos do *joystick* para a direita e esquerda, no eixo X, para pré-selecionar o motor que deseja movimentar. Uma vez que o motor seja o adequado, basta pressionar o *joystick* para apertar o botão e confirmar que aquele será o motor a realizar os movimentos, entrando assim no modo movimento. Após, o usuário utiliza os mesmos movimentos para as laterais para movimentar o motor escolhido, com a direção de giro, horário ou anti-horário, sendo definidos pela direção de movimentação do *joystick*. Uma vez na posição requerida, o usuário precisa pressionar novamente o botão para sair do modo de movimento e retornar ao modo de escolha.

Leds são usados para indicar qual motor está pré-selecionado, de A a F, e se ele está em modo de movimento. Além disso, um sinal sonoro indica quando entra e sai do modo de movimento.

A fim de realizar o experimento de Fitts, quando o objeto é pego, o programa inicia a contagem do tempo para a realização da tarefa, e para quando identifica que o objeto foi solto. Esses dados são enviados à interface gráfica por meio da porta serial e salvos em um banco de dados (BD) posteriormente.

## 3.3 Interface Gráfica

Para fornecer outra alternativa de feedback para o usuário e tornar o experimento mais lúdico, foi implementada uma interface gráfica do usuário em programação C#, utilizando o programa Visual Studio 2017. Todas as informações adquiridas pela interface são salvas em um banco de dados, que será detalhado na Seção 3.4.

A primeira página da interface é um formulário, onde o pesquisador preenche as informações sobre si mesmo e sobre o voluntário a realizar o experimento, facilitando o rastreamento de dados mesmo com vários voluntários e pesquisadores. Dessa forma, as informações requeridas são padronizadas e não dependem de meios menos confiáveis para serem salvas ou consultadas. A Tabela 2 demonstra quais são essas informações.

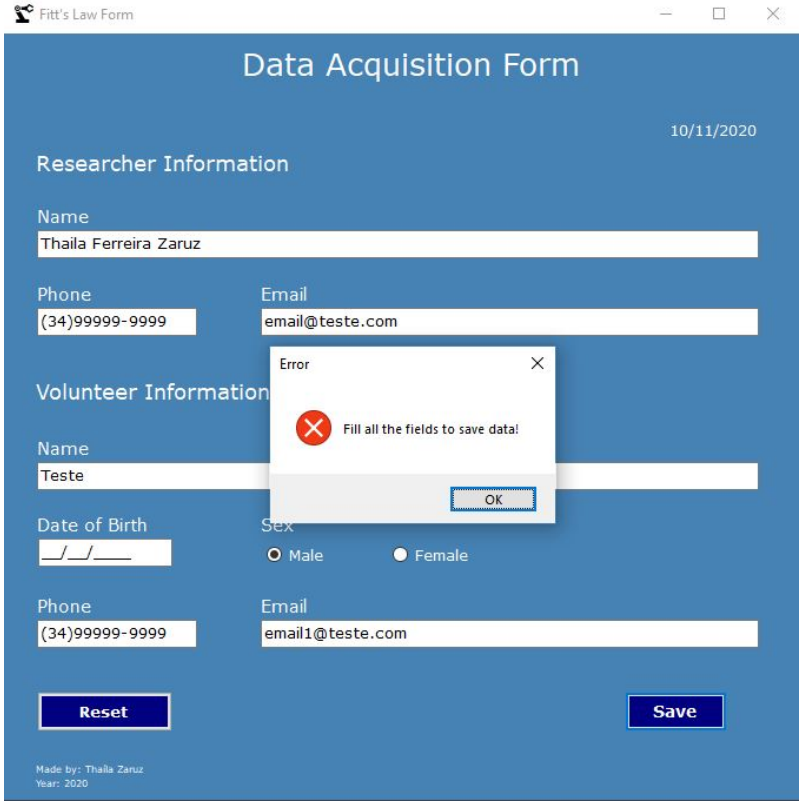
Tabela 2 - Informações requeridas no formulário GUI

| **Formulário IGU** | |
|---|---|
| **Pessoa** | **Dados** |
| Pesquisador | Nome |
| | Telefone |
| | E-mail |
| Voluntário | Nome |
| | Data de nascimento |
| | Sexo |
| | Telefone |
| | E-mail |

Fonte: Autoria própria (2020)

Caso o pesquisador esqueça de preencher algum campo ou ocorra algum problema para salvar as informações, mensagens de erro alertam o usuário sobre o problema. Um exemplo é o da Figura 7, em que o usuário não preencheu todas as informações e a mensagem de erro informa a necessidade de retornar e preencher os campos faltantes adequadamente. Neste caso, a informação faltosa é a data de nascimento do voluntário.

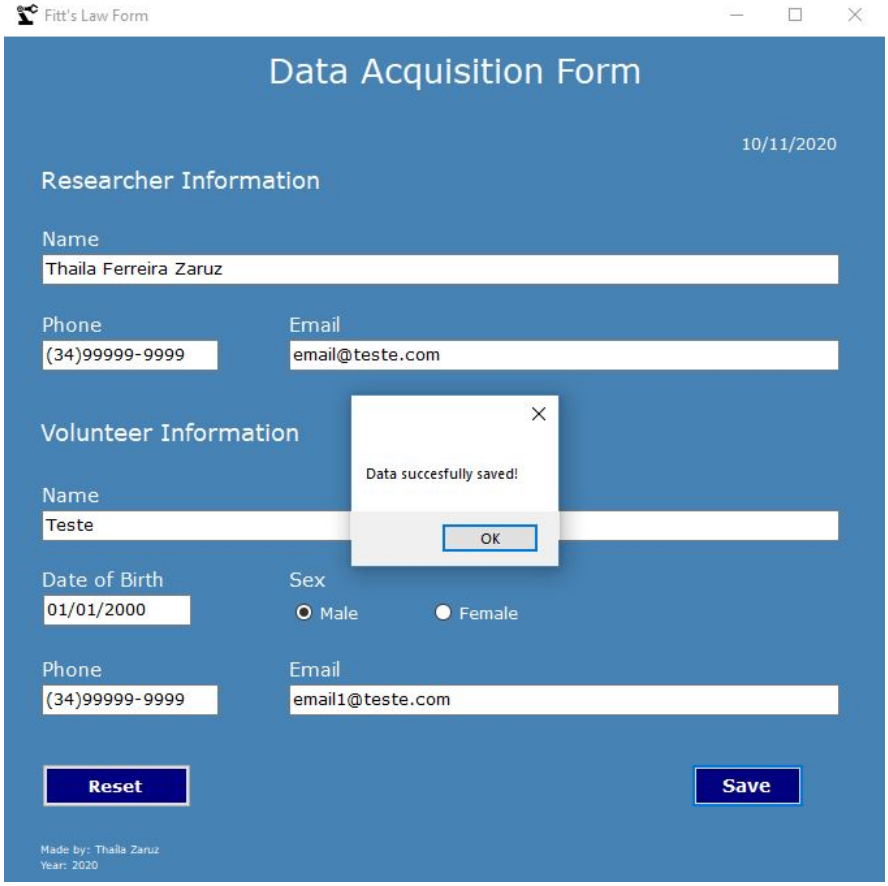
Figura 7 - Mensagem de erro do formulário




Fonte: Autoria própria (2020)

Ainda para garantir que o programa funcionou adequadamente, uma mensagem informa se os dados foram salvos corretamente, para, então, fechar a página do formulário e prosseguir para a próxima etapa da GUI. A mensagem está demonstrada na Figura 8.

Figura 8 - Mensagem de sucesso ao salvar os dados no formulário




Fonte: Autoria própria (2020)

A segunda página da interface é a página do usuário, com itens para a realização do experimento da Lei de Fitts. Nela, as informações processadas pelo arduino são transmitidas via porta serial para a aplicação e mostradas na página.

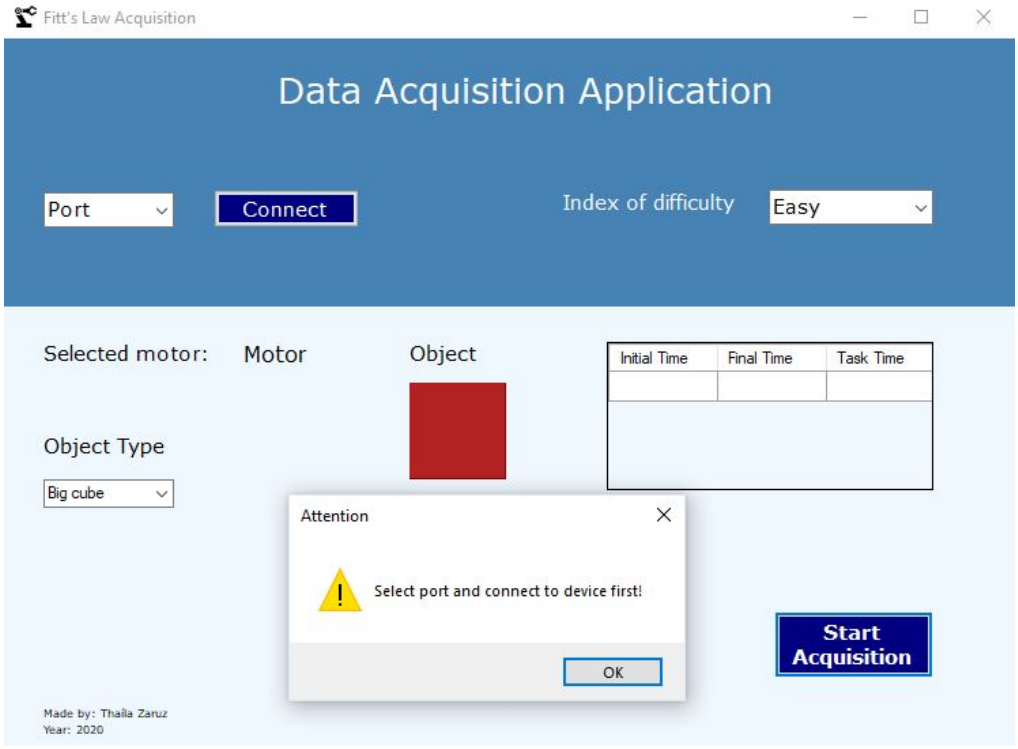
Para sua operação, o usuário deve selecionar o índice de dificuldade de execução da tarefa e o objeto utilizado conforme valores já pré-selecionados mostrados na página. Além disso, deve escolher em qual porta o arduino está conectado e iniciar a sua comunicação clicando no botão *Connect.* Após esses procedimentos, o usuário pode então iniciar a aquisição de dados clicando no botão *Start Acquisition.*

A partir desse momento, as informações sobre o motor selecionado, movimentação do motor e se o objeto foi pego ou não são mostradas conforme o voluntário movimenta o braço robótico. Para auxiliar o pesquisador a visualizar os dados durante o procedimento de coleta, os valores de tempo inicial, tempo final e tempo de execução da tarefa também são exibidos na interface, garantindo que em

caso de erro com a informação a coleta possa ser reiniciada ou descartada sem prejuízos futuros.

Assim como na página anterior, mensagens de erro aparecem quando o usuário não preenche todas as informações necessárias para o funcionamento do programa ou quando ocorre algum erro ao salvar os dados no banco de dados. Na Figura 9 temos um exemplo quando o usuário tenta iniciar a aquisição, mas não selecionou a porta de conexão com o arduino.

Figura 9 - Mensagem de erro na página do usuário




Fonte: Autoria própria (2020)

### 3.3.1 Programação GUI

Para o desenvolvimento de GUI em C#, o Visual Studio exibe uma aba de design, onde é possível montar visualmente a interface gráfica antes de realizar a programação dos seus elementos. Assim, primeiro foram montadas as duas páginas de interação com o usuário, para em seguida programar o funcionamento dos seus elementos. O código para a página de formulário pode ser consultado no Apêndice B, enquanto que o código para a página do usuário está no Apêndice C.

Como todas as informações preenchidas na interface são salvas em um banco de dados, foi preciso programar também a interação e conexão entre os dois programas. Para isso e visando a organização dos códigos, foram criadas três classes dentro do aplicativo da GUI que realizam toda essa interação.

A classe *Connection* realiza a conexão e desconexão com o banco de dados criado localmente. Cada vez que há a necessidade de envio de informações, a conexão é aberta, o comando executado e então a conexão é fechada. A interrupção da conexão com o BD quando o programa é encerrado ou após o envio das informações é importante para impedir o seu travamento e o recebimento de dados errôneos. Essa classe pode ser consultada no Apêndice D.

Na página de formulário da GUI, onde as informações acerca do pesquisador e usuário são preenchidas, é utilizada a classe *Register.* Caso a tabela correta não exista, a classe cria uma tabela no BD para salvar as informações adquiridas pelo formulário. Na hipótese da ocorrência de erros na conexão com o bando de dados ou para acessar a tabela correta, mensagens de erro são mostradas ao usuário com a descrição do problema, facilitando a sua solução. Essa classe pode ser consultada no Apêndice E.

Por último, para acessar e salvar os dados adquiridos durante o experimento e executados na página do usuário da GUI, é utilizada a classe *StartAcquisition,* demonstrada no Apêndice F. Seu modo de funcionamento é bem semelhante à classe *Register,* criando uma tabela para os dados da coleta, caso ainda não exista, e enviando mensagens de erro ao usuário quando ocorre algum problema.

Vale destacar que a classe *Connection* não é usada diretamente nos códigos das páginas de formulário e do usuário, e sim nas outras duas classes. A programação em camadas desse jeito evita que o programa se torne pesado e execute lentamente.

O código completo, comentado pode ser encontrado em (ZARUZ, 2020b). Em adicional foi implementado um instalador para a interface, permitindo sua instalação e rápido acesso em outros computadores.

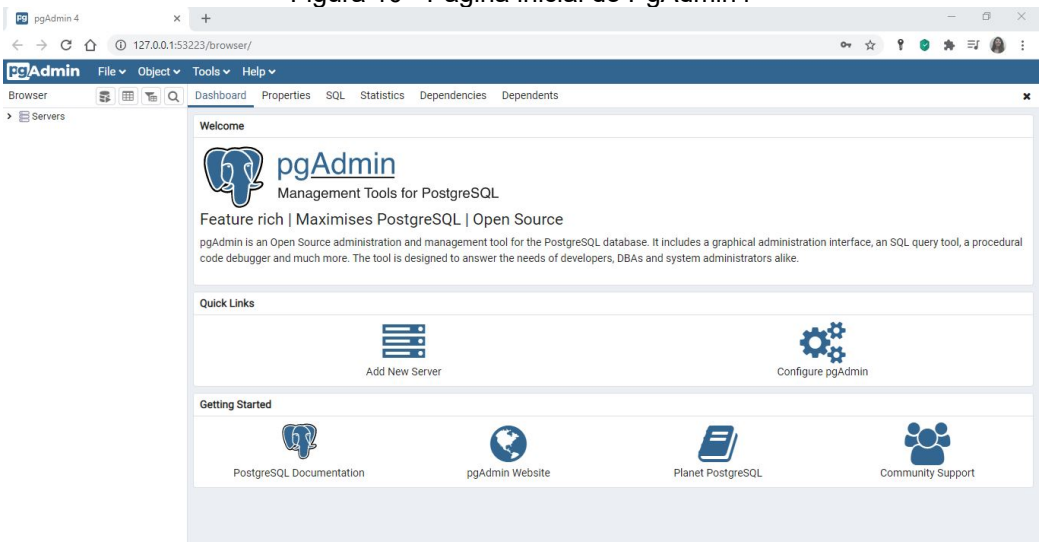
### 3.4 Banco de dados

Com o propósito de armazenar os dados adquiridos pela interface gráfica e pelo arduino, o uso de um banco de dados se torna essencial. Para esta aplicação foi escolhido o programa Postgresql, um banco de dados relacional com base em

linguagem *standard query language* (SQL) com código aberto (POSTGRESQL, 2020). Sua instalação é simples, possuindo versões para diversos sistemas operacionais e interface com o usuário de fácil entendimento.

Após o download do programa, a interação com o usuário é feita por meio da ferramenta de gerenciamento PgAdmin4. É por meio dela que as tabelas e informações são criadas, acessadas e atualizadas. A Figura 10 mostra a página inicial da ferramenta.

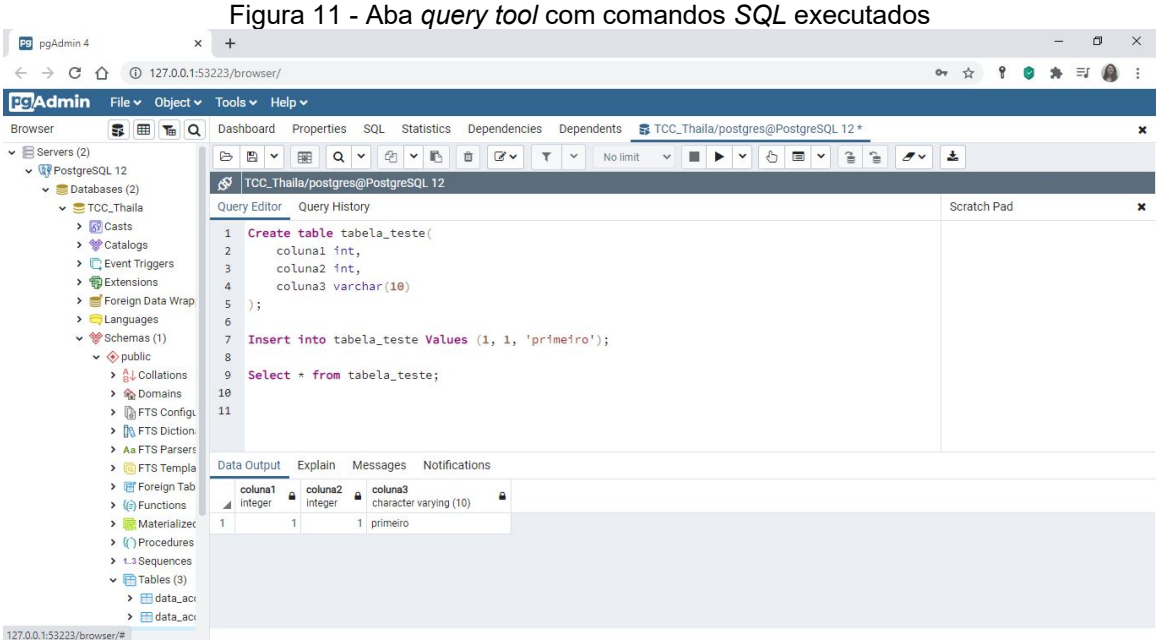
Figura 10 - Página inicial do PgAdmin4




Fonte: Autoria própria (2020)

A aba *query tool* é onde os comandos SQL são executados. Aqui os principais comandos são para criar tabelas, inserir dados e visualizá-las, sendo eles *CREATE TABLE, INSERT INTO* nome_da_tabela *VALUES( )* e *SELECT * FROM* nome_da_tabela. A Figura 11 demonstra um exemplo de uso dessas funções, criando uma tabela de nome tabela_teste com três colunas, inserindo valores iniciais e então exibindo as informações armazenadas nela.

Figura 11 - Aba *query tool* com comandos *SQL* executados




Fonte: Autoria própria (2020)

Neste projeto foram criadas duas tabelas para armazenar as informações, a primeira sendo a *Volunteer_information*, com dados do pesquisador e usuário/voluntário preenchidos na página de formulário da aplicação, e a segunda sendo *Data_acquisition*, com dados do experimento da Lei de Fitts obtidos na página do usuário. A Tabela 3 relaciona as informações salvas e a respectiva tabela referente no banco de dados.

Tabela 3 - Dados salvos no banco de dados

| Banco de dados | |
|---|---|
| **Tabela** | **Dados** |
| Volunteer_information | ID do voluntário |
| Volunteer_information | Data do experimento |
| Volunteer_information | Nome do pesquisador |
| Volunteer_information | Telefone do pesquisador |
| Volunteer_information | E-mail do pesquisador |
| Volunteer_information | Nome do voluntário |
| Volunteer_information | Data de nascimento do voluntário |
| Volunteer_information | Sexo do voluntário |
| Volunteer_information | Telefone do voluntário |
| Volunteer_information | E-mail do voluntário |
| Data_acquisition | Número da aquisição |
| Data_acquisition | ID do voluntário |
| Data_acquisition | Objeto |
| Data_acquisition | Número da tarefa |
| Data_acquisition | Índice de dificuldade |
| Data_acquisition | Tempo inicial |
| Data_acquisition | Tempo final |
| Data_acquisition | Tempo de execução |

Fonte: Autoria própria (2020).

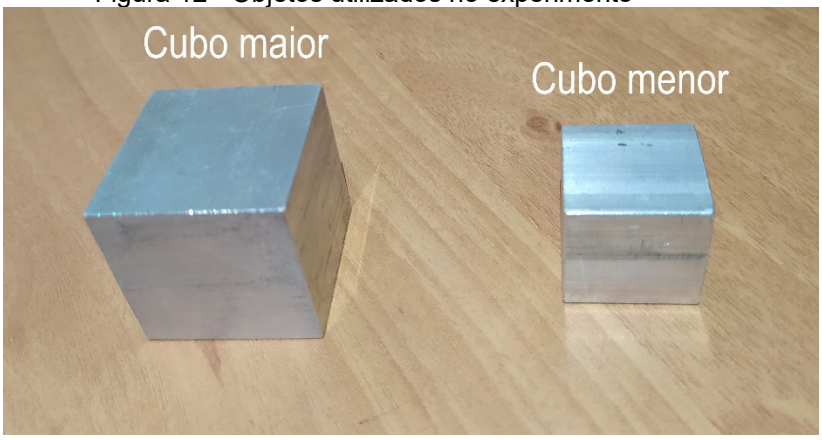
Foram criadas duas tabelas de forma a mantê-las organizadas e somente com informações relevantes. Para isso, na tabela *Data_acquisition* existe uma chave estrangeira, ou seja, uma coluna que se relaciona com outra tabela. A chave estrangeira utilizada é a coluna ID do voluntário, que é preenchida primeiramente em *Volunteer_information* e utilizada posteriormente na segunda tabela para relacionar os dados do experimento ao voluntário que os executou, garantindo que mesmo sem as informações do voluntário presentes na segunda tabela seja possível identificar o voluntário que realizou cada atividade.

### 3.5 Lei de Fitts

Para o cálculo e realização de testes, foram escolhidos dois diferentes objetos, sendo: um cubo com 2,2 cm de lado, chamado aqui de cubo menor, e um cubo com 3,1 cm de lado, chamado de cubo maior, ilustrados na Figura 12. Para cada objeto

foram definidos ID diferentes de acordo com o tamanho e a distância entre os alvos. As informações são exibidas na Tabela 4.

Figura 12 - Objetos utilizados no experimento




Fonte: Autoria própria (2020)

Tabela 4 - Relação entre índice de dificuldade e objetos

| Relação objeto – índice de dificuldade | | | | |
|---|---|---|---|---|
| **Objeto** | **Índice de dificuldade (ID)** | **Valor ID** | **Tamanho alvo (cm)** | **Distância entre alvos (cm)** |
| Cubo menor | Fácil | 2,58 | 6 | 18 |
| | Médio | 3,13 | 5 | 22 |
| | Difícil | 3,70 | 4 | 26 |
| Cubo maior | Fácil | 2,58 | 6,5 | 19,5 |
| | Médio | 3,13 | 6 | 26,4 |
| | Difícil | 3,70 | 5 | 32,5 |

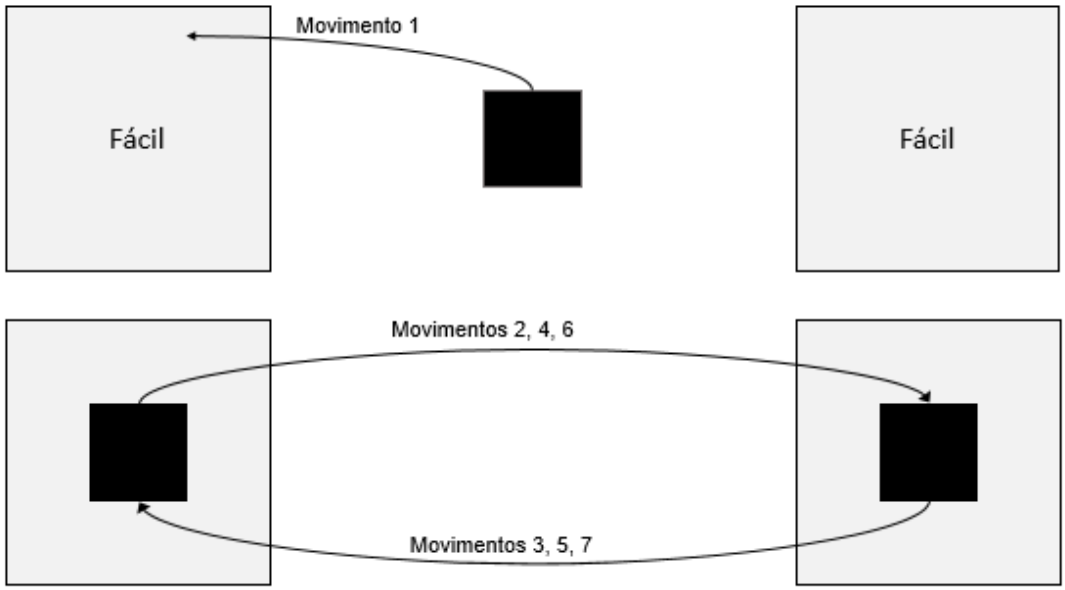
Fonte: Autoria própria (2020).

A tarefa, seguindo os princípios da Lei de Fitts, se inicia com o cubo localizado entre dois alvos e a base do braço robótico posicionada próxima aos alvos. O participante deve, primeiramente, levar o cubo até um dos alvos, e a partir disso movimentar o cubo entre os alvos seis vezes. Toda movimentação é realizada utilizando o braço robótico como manipulador.

O tempo de movimentação do cubo entre um alvo e outro é cronometrado e utilizado para o cálculo dos coeficientes da Lei de Fitts, uma vez que o ID já foi definido. O movimento inicial usado para posicionar o cubo no primeiro alvo não é considerado para a Lei de Fitts, uma vez que não cumpre os seus critérios. A tarefa é repetida para cada índice de dificuldade, com ambos os objetos.

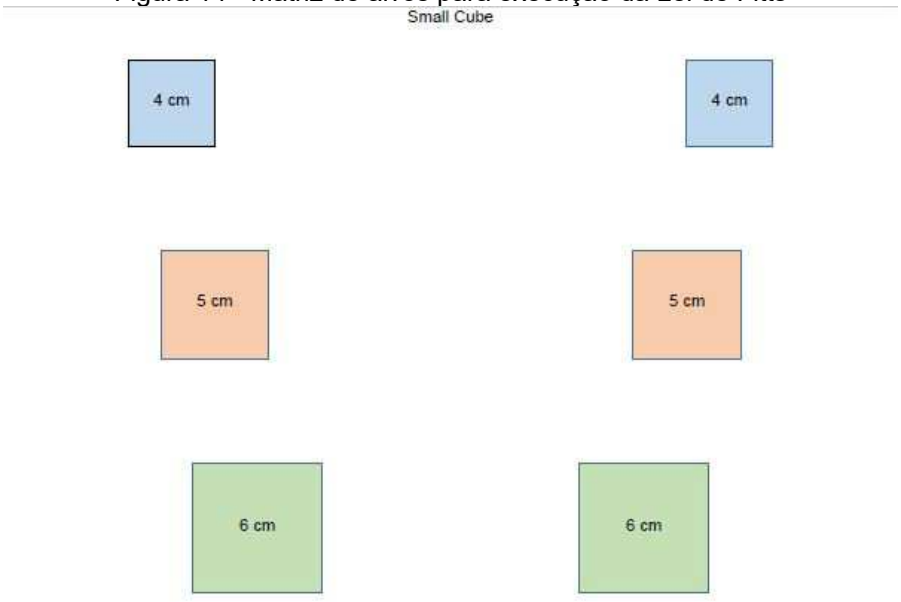
Como cada ID é independente na execução de sua tarefa e devido ao formato que a matriz de alvos foi criada, reposicionamentos do braço robótico para a execução da tarefa em cada ID são necessários. A Figura 13 ilustra o modo como os movimentos são realizados. Já a Figura 14 mostra a matriz de alvos, que é impressa em folha A3 e semelhante para os dois objetos.

Figura 13 - Demonstração dos movimentos executados




Fonte: Autoria própria (2020)

Figura 14 - Matriz de alvos para execução da Lei de Fitts




Fonte: Autoria própria (2020)

# 4. RESULTADOS

Para alcançar os objetivos definidos, o projeto foi dividido em etapas, explicadas na metodologia do presente trabalho. Aqui são apresentados os resultados de cada etapa.

## 4.1 Montagem e controle do braço robótico

O braço robótico foi montado conforme explicado na metodologia e é controlado por meio de *joystick*, possuindo a funcionalidade necessária para o projeto. O conjunto é constituído pelo braço robótico, fonte de alimentação e caixa controladora do braço. Cada um desses três itens pode ser desmontado e armazenado separadamente pois sua união é feita por meio de cabos de fácil desconexão do circuito. A Figura 15 mostra a fonte de alimentação, onde é possível visualizar como é realizada a conexão dos cabos para energização dos motores do braço robótico.

Figura 15 - Fonte de alimentação com cabo de energia

Fonte: Autoria própria (2020)

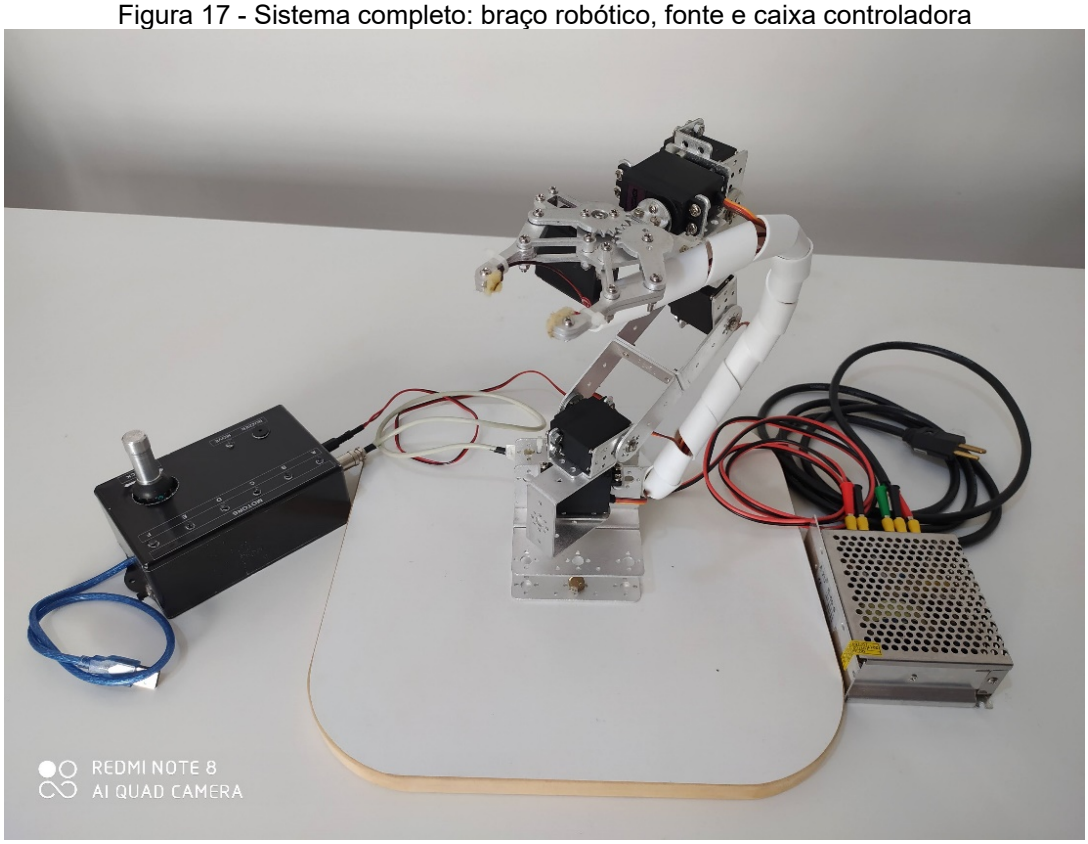
Na Figura 16, os detalhes da caixa controladora são mostrados com sua vista superior, destacando os elementos pelos quais o usuário realiza o controle e recebe feedback dos movimentos executados, vista lateral direita, com o cabo de conexão USB e botão *reset*, e a vista lateral esquerda, onde são feitas as conexões da placa controladora do braço e do sistema de chave elétrica

Figura 16 - Caixa controladora; a) Vista superior; b) Vista lateral direita; e c) Vista lateral esquerda


Fonte: Autoria própria (2020)

A Figura 17 mostra o sistema completo, composto pelo braço robótico, fonte de alimentação e caixa controladora. Esse sistema pode ser usado para diferentes aplicações e, visto o meio acadêmico em que o projeto foi executado, o seu uso em outros estudos ou para fins educacionais se torna uma excelente ferramenta.

Figura 17 - Sistema completo: braço robótico, fonte e caixa controladora

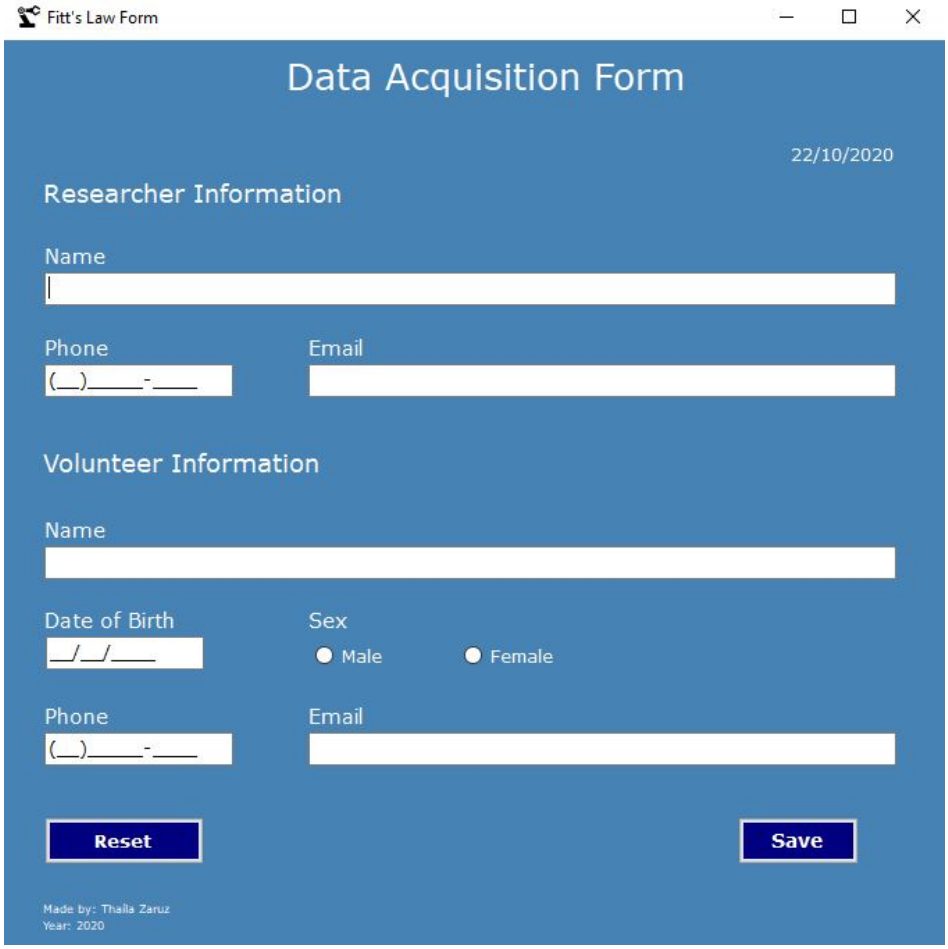



Fonte: Autoria própria (2020)

## 4.2 Desenvolvimento da interface gráfica

A interface gráfica foi implementada com duas páginas para interação do usuário. A primeira página é demonstrada pela Figura 18, onde é visualizado o formulário com as informações que devem ser preenchidas sobre o pesquisador e sobre o voluntário.

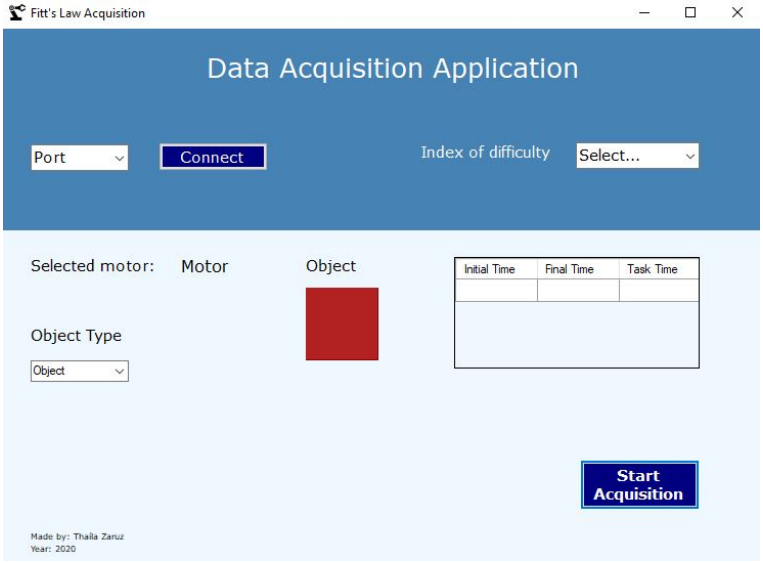
Figura 18 - Primeira página da interface gráfica: formulário




Fonte: Autoria própria (2020)

Já a Figura 19 demonstra a segunda página, em que as informações necessárias para a execução das tarefas da Lei de Fitts são apresentadas para o voluntário enquanto ele realiza o experimento. A escolha em manter o texto de ambas as páginas em inglês se deve ao fato de que, assim, o sistema pode ser utilizado amplamente em outras pesquisas, visto a dominância do idioma no meio acadêmico.

Figura 19 - Segunda página da interface gráfica: página do usuário

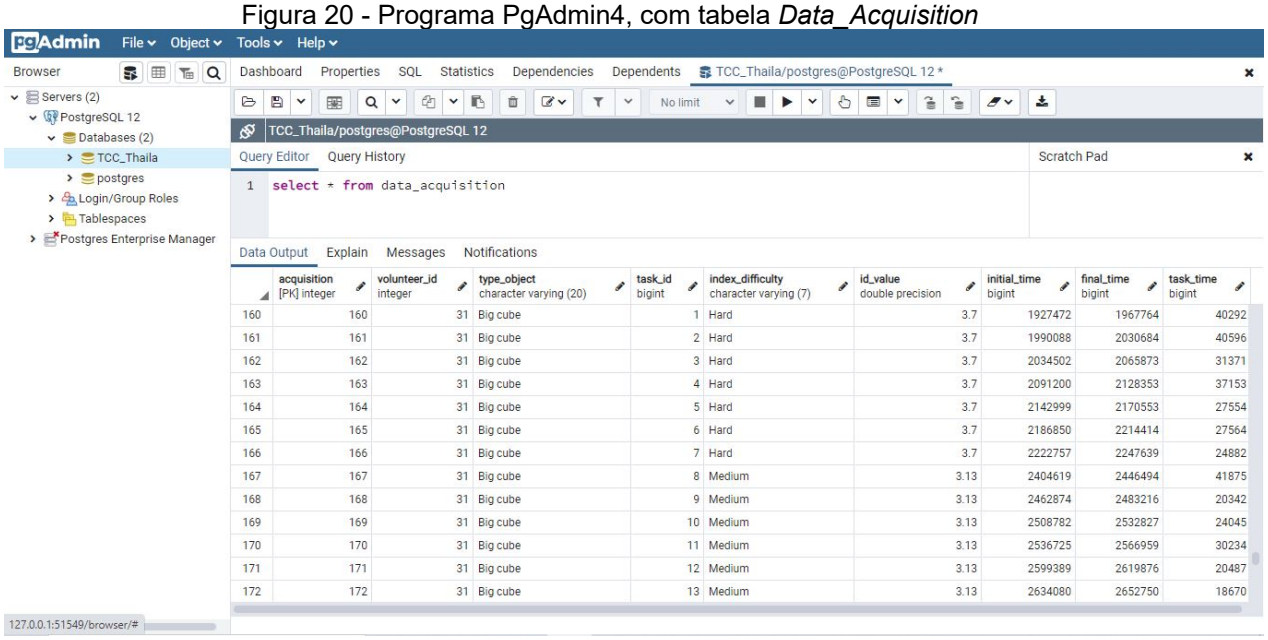



Fonte: Autoria própria (2020).

## 4.3 Criação de banco de dados

Complementando a GUI, o banco de dados criado localmente armazena as informações adquiridas durante a execução da aplicação gráfica, sendo acessadas por meio do programa PgAdmin4 após a execução de alguns comandos SQL. A Figura 20 mostra algumas informações da tabela *Data_Acquisition*, preenchidas durante a execução de testes.

Figura 20 - Programa PgAdmin4, com tabela *Data_Acquisition*




| | acquisition [PK] integer | volunteer_id integer | type_object character varying (20) | task_id bigint | index_difficulty character varying (7) | id_value double precision | initial_time bigint | final_time bigint | task_time bigint |
|---|---|---|---|---|---|---|---|---|---|
| 160 | 160 | 31 | Big cube | 1 | Hard | 3.7 | 1927472 | 1967764 | 40292 |
| 161 | 161 | 31 | Big cube | 2 | Hard | 3.7 | 1990088 | 2030684 | 40596 |
| 162 | 162 | 31 | Big cube | 3 | Hard | 3.7 | 2034502 | 2065873 | 31371 |
| 163 | 163 | 31 | Big cube | 4 | Hard | 3.7 | 2091200 | 2128353 | 37153 |
| 164 | 164 | 31 | Big cube | 5 | Hard | 3.7 | 2142999 | 2170553 | 27554 |
| 165 | 165 | 31 | Big cube | 6 | Hard | 3.7 | 2186850 | 2214414 | 27564 |
| 166 | 166 | 31 | Big cube | 7 | Hard | 3.7 | 2222757 | 2247639 | 24882 |
| 167 | 167 | 31 | Big cube | 8 | Medium | 3.13 | 2404619 | 2446494 | 41875 |
| 168 | 168 | 31 | Big cube | 9 | Medium | 3.13 | 2462874 | 2483216 | 20342 |
| 169 | 169 | 31 | Big cube | 10 | Medium | 3.13 | 2508782 | 2532827 | 24045 |
| 170 | 170 | 31 | Big cube | 11 | Medium | 3.13 | 2536725 | 2566959 | 30234 |
| 171 | 171 | 31 | Big cube | 12 | Medium | 3.13 | 2599389 | 2619876 | 20487 |
| 172 | 172 | 31 | Big cube | 13 | Medium | 3.13 | 2634080 | 2652750 | 18670 |

Fonte: Autoria própria (2020).

A escolha de um banco de dados cujo funcionamento fosse baseado em SQL torna a sua manipulação simples e, caso o usuário não se recorde dos comandos, uma rápida busca na internet permite a sua operação.

### 4.4 Calculo dos parâmetros da Lei de Fitts

Dessa forma, todas as etapas de montagem e implementação foram finalizadas para, então, iniciar a aquisição de dados. Foram realizados os testes necessários no sistema a fim de validar o projeto e realizar os cálculos da LF. As informações coletadas serviram ao propósito de demonstrar um exemplo de aplicação e uso do braço robótico juntamente com a interface gráfica, assim como do funcionamento da Lei de Fitts. A presente autora realizou todos os testes, sem a necessidade de voluntários do público geral, uma vez que o objetivo é ilustrar o uso do sistema, sem a realização de análises estatísticas acerca da aplicabilidade da Lei de Fitts.

Para cada objeto em cada ID foram realizadas sete coletas de dados. A primeira é realizada apenas para posicionar o objeto e não é considerada nas análises, uma vez que o movimento é realizado para iniciar o procedimento. Todas as tabelas e gráficos foram gerados utilizando o software Excel 2016.

A Tabela 5 mostra os dados coletados utilizando o cubo menor, com movimentos realizados nos três índices de dificuldade diferentes. Essas informações ficam salvas no banco de dados para consulta posterior à coleta. Na tabela, os dados em cor cinza são aqueles desconsiderados na análise da Lei de Fitts.

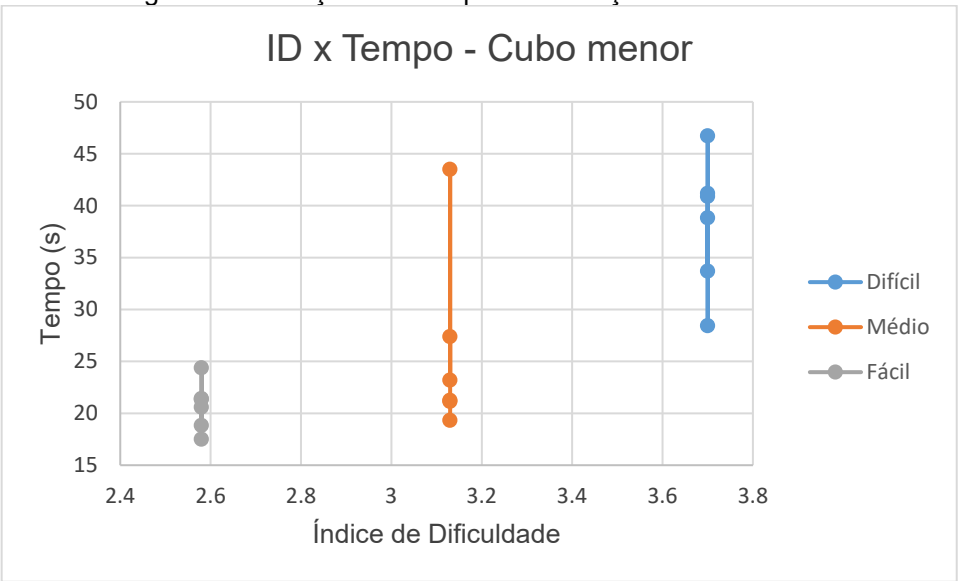
Tabela 5 - Dados coletados com o cubo menor

| Número da tarefa | Índice de dificuldade | Valor do ID | Tempo inicial (s) | Tempo final (s) | Tempo de execução (s) | Tempo médio de execução (s) |
|---|---|---|---|---|---|---|
| 1 | Difícil | 3,7 | 212,656 | 236,611 | 23,955 | |
| **2** | Difícil | 3,7 | 238,955 | 277,780 | 38,825 | |
| **3** | Difícil | 3,7 | 280,690 | 309,118 | 28,428 | |
| **4** | Difícil | 3,7 | 311,866 | 353,057 | 41,191 | 38,291 |
| **5** | Difícil | 3,7 | 355,644 | 389,340 | 33,696 | |
| **6** | Difícil | 3,7 | 391,845 | 438,567 | 46,722 | |
| **7** | Difícil | 3,7 | 441,316 | 482,202 | 40,886 | |
| 8 | Médio | 3,13 | 535,195 | 550,653 | 15,458 | |
| **9** | Médio | 3,13 | 553,159 | 580,542 | 27,383 | |
| **10** | Médio | 3,13 | 585,795 | 608,992 | 23,197 | |
| **11** | Médio | 3,13 | 611,175 | 654,678 | 43,503 | 25,967 |
| **12** | Médio | 3,13 | 657,829 | 678,976 | 21,147 | |
| **13** | Médio | 3,13 | 681,481 | 702,724 | 21,243 | |
| **14** | Médio | 3,13 | 704,744 | 724,073 | 19,329 | |
| 15 | Fácil | 2,58 | 774,347 | 793,119 | 18,772 | |
| **16** | Fácil | 2,58 | 832,627 | 851,457 | 18,830 | |
| **17** | Fácil | 2,58 | 880,020 | 901,440 | 21,420 | |
| **18** | Fácil | 2,58 | 920,153 | 940,735 | 20,582 | 20,674 |
| **19** | Fácil | 2,58 | 943,160 | 960,657 | 17,497 | |
| **20** | Fácil | 2,58 | 971,901 | 996,280 | 24,379 | |
| **21** | Fácil | 2,58 | 1024,555 | 1045,891 | 21,336 | |

Fonte: Autoria própria (2020)

Usando algumas das informações adquiridas, temos na Figura 21 a sua representação gráfica, em que o ID é relacionado com o tempo de execução de cada tarefa.

Figura 21 - Relação ID x tempo de execução do cubo menor




Fonte: Autoria própria (2020)

Realizando as mesmas tarefas, utilizando o cubo maior, temos os dados mostrados na Tabela 6.

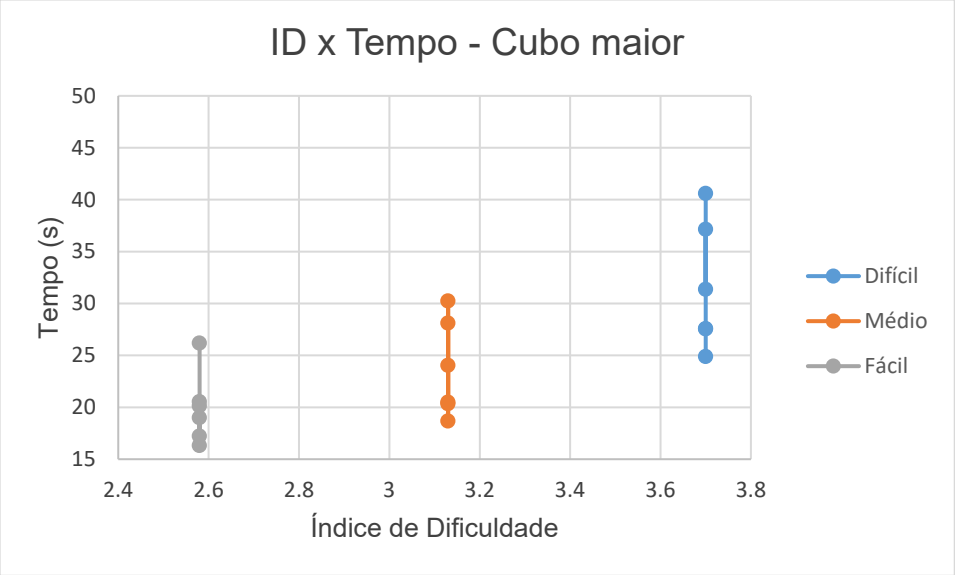
Tabela 6 - Dados coletados com o cubo maior

| Índice da tarefa | Índice de dificuldade | Valor do ID | Tempo inicial (s) | Tempo final (s) | Tempo de execução (s) | Tempo médio de execução (s) |
|---|---|---|---|---|---|---|
| 1 | Difícil | 3,7 | 1927,472 | 1967,764 | 40,292 | |
| 2 | Difícil | 3,7 | 1990,088 | 2030,684 | 40,596 | |
| 3 | Difícil | 3,7 | 2034,502 | 2065,873 | 31,371 | |
| 4 | Difícil | 3,7 | 2091,200 | 2128,353 | 37,153 | 31,520 |
| 5 | Difícil | 3,7 | 2142,999 | 2170,553 | 27,554 | |
| 6 | Difícil | 3,7 | 2186,850 | 2214,414 | 27,564 | |
| 7 | Difícil | 3,7 | 2222,757 | 2247,639 | 24,882 | |
| 8 | Médio | 3,13 | 2404,619 | 2446,494 | 41,875 | |
| 9 | Médio | 3,13 | 2462,874 | 2483,216 | 20,342 | |
| 10 | Médio | 3,13 | 2508,782 | 2532,827 | 24,045 | |
| 11 | Médio | 3,13 | 2536,725 | 2566,959 | 30,234 | 23,648 |
| 12 | Médio | 3,13 | 2599,389 | 2619,876 | 20,487 | |
| 13 | Médio | 3,13 | 2634,080 | 2652,750 | 18,670 | |
| 14 | Médio | 3,13 | 2660,286 | 2688,397 | 28,111 | |
| 15 | Fácil | 2,58 | 2779,870 | 2799,638 | 19,768 | |
| 16 | Fácil | 2,58 | 2803,294 | 2829,484 | 26,190 | |
| 17 | Fácil | 2,58 | 2832,493 | 2852,616 | 20,123 | |
| 18 | Fácil | 2,58 | 2875,952 | 2896,486 | 20,534 | 19,896 |
| 19 | Fácil | 2,58 | 2919,189 | 2936,413 | 17,224 | |
| 20 | Fácil | 2,58 | 2959,360 | 2978,351 | 18,991 | |
| 21 | Fácil | 2,58 | 2983,138 | 2999,457 | 16,319 | |

Fonte: Autoria própria (2020)

Relacionando o tempo de execução da tarefa com o índice de dificuldade, temos a Figura 22. Assim como na Figura 21, é notável que o tempo de execução aumenta juntamente com o índice de dificuldade.

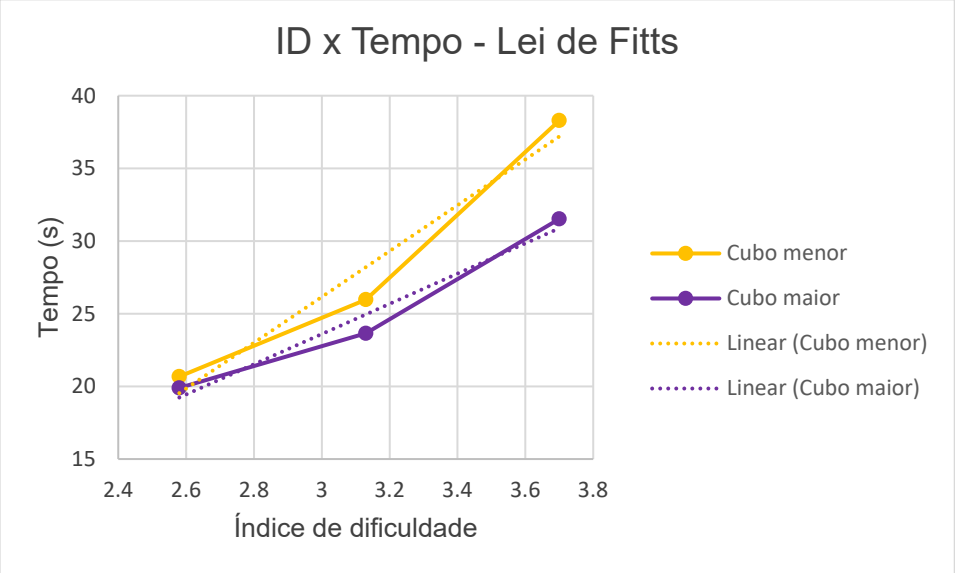
Figura 22 - Relação ID x tempo de execução do cubo maior




Fonte: Autoria própria (2020)

A fim de comparar melhor as informações adquiridas dos dois objetos, foi calculado o tempo médio de execução da tarefa para cada ID, mostrado nas Tabela 5 e Tabela 6 e representado na Figura 23 pelas retas contínuas. Nela temos duas retas para cada objeto, na qual a cor amarela se refere ao cubo menor e a cor roxa se refere ao cubo maior.

Figura 23 - Relação ID x tempo de execução médio de ambos objetos




Fonte: Autoria própria (2020)

Com base nos tempos médios de execução das tarefas observados na Figura 23, foi realizada a regressão linear da reta de ambos os objetos, sendo representada pelas retas tracejadas. Novamente a cor amarela se refere ao cubo menor e a cor roxa se refere ao cubo maior. Segue a equação da reta para a série do cubo menor (3) e do cubo maior (4), respectivamente:

$$y = 15765x - 21140 \quad (3)$$

$$y = 10399x - 7595{,}3 \quad (4)$$

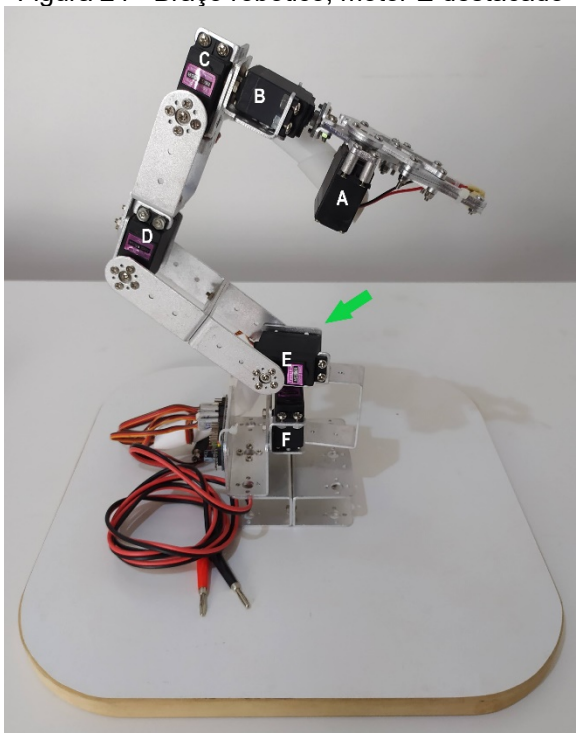
# 5. DISCUSSÃO

## 5.1 Montagem e controle do braço robótico

A montagem do braço robótico apresentou robustez na execução dos movimentos requeridos, com a base de madeira sendo suficiente para manter o braço no local adequado e auxiliando o seu transporte.

Apesar disso e da estrutura do braço robótico ter sido entregue com todos os componentes já selecionados, um fato observado durante a realização de testes longos foi o mau desempenho do motor E, indicado na Figura 24.

Figura 24 - Braço robótico, motor E destacado




Fonte: Autoria própria (2020)

Nesses testes percebia-se que o motor esquentava bastante e, após um tempo de funcionamento de cerca de 30 minutos não respondia mais aos comandos enviados. Quando acontecia, era necessário desligar a fonte de alimentação dos motores, desligar o sistema da caixa controladora e aguardar o resfriamento do motor, cerca de 1 hora, para que então voltasse a funcionar.

Uma das causas para este problema pode ser o mau dimensionamento do motor para as tarefas realizadas pelo braço, onde o torque do motor não é suficiente

para sustentar o peso do braço quando está em determinadas posições da articulação.

O tamanho da base de madeira também influenciou na região de atuação do braço, fazendo com que a articulação do motor E fique mais estendida para alcançar os objetos.

Como o servo-motor MG996R foi o modelo fornecido no kit, optou-se por não trocar o motor por outro com especificações diferentes, visto que ele poderia não encaixar na estrutura de alumínio.

A caixa controladora apresentou funcionamento adequado, enviando comandos ao sistema e fornecendo feedback ao usuário por meio dos *leds* e *buzzer*. De acordo com o uso do braço robótico, a conexão do cabo do sistema de chave elétrica na caixa controladora pode não ser necessária, uma vez que ela identifica quando o braço agarrou ou soltou um objeto metálico. Aplicações diferentes podem não utilizar objetos metálicos ou necessitar da confirmação que o objeto foi pego, sendo a confirmação visual do usuário suficiente para os fins determinados.

A implementação de outros métodos de controle em substituição ao joystick, como eletromiografia, sensores inerciais, *flex* sensor ou até mesmo um *smartphone* com transmissão de dados sem fio são maneiras de utilizar parte do sistema do braço robótico construído neste projeto.

O estudo COSTA et al., (2014) é um exemplo de uso do mesmo kit de braço robótico, porém com controladores diferentes, onde foi utilizado sensores flexíveis e um acelerômetro para controlar os movimentos do braço. Por ele também é demonstrado a variedade de maneiras em montar a estrutura do braço robótico, já que fez uso apenas de três motores, garantindo três graus de liberdade para o manipulador robótico.

Já o trabalho BAJERSKI; ABELLA, (2010) demonstra ainda o controle de um braço robótico por meio de um controle remoto do videogame Nintendo Wii, chamado *wiimote.* Além da forma de controle diferente, a transmissão da informação é feita via *bluetooth*. Apesar das melhorias no método de controle quando comparado a este trabalho, a estrutura do braço robótico aparenta ser mais rústica e com menos graus de liberdade, uma vez que possui cinco motores.

A possibilidade de utilização braço robótico apresentado neste trabalho ou de partes dele em trabalhos futuros foi um item considerado durante a escolha do projeto, uma vez que o braço robótico foi doado pela SBEB e tinha-se a intenção de tornar o

sistema acessível a outros alunos de graduação do curso de Engenharia Biomédica da UFU.

### 5.2 Desenvolvimento de interface gráfica e banco de dados

A GUI implementada apresentou bom funcionamento. Mesmo que o propósito de uso do sistema robótico seja modificado, as informações requeridas na primeira página foram escolhidas devido à sua relevância para qualquer pesquisa com voluntários. Uma vez que este trabalho foi elaborado visando sua continuidade, a implementação da interface já visando este fim auxilia os próximos pesquisadores.

A segunda página é específica para a execução das tarefas da Lei de Fitts, possibilitando ao usuário escolher o índice de dificuldade e o tipo de objeto escolhidos para a tarefa, visualizando também os motores selecionados e o *status* do objeto.

Ainda que a interface implementada não seja utilizada para os fins estabelecidos neste trabalho, ela possui elementos que podem ser utilizados em outros trabalhos. A primeira página contém informações geralmente requeridas por pesquisadores. Na segunda página os conceitos implementados de atualização e escolha de porta USB, botões, *combobox* e *datagrid* podem ser usados na maioria das interfaces criadas em C#.

A criação de um arquivo executável para a interface permite que ela possa ser executada de qualquer computador, não ficando restrita aos laboratórios de pesquisa e podendo ser usada para fins didáticos.

Considerando que o pesquisador sempre explicará o protocolo experimental, uma página ou vídeo explicando a tarefa a ser realizada não foi acrescentado, mas pode ser tratado como uma melhoria ao projeto.

A criação e conexão com banco de dados local é essencial para a segurança dos dados coletados nessa aplicação, garantindo que estejam salvos e acessíveis após o cumprimento das tarefas. A facilidade de criação e manipulação das tabelas também é relevante, visto a quantidade de informações que podem ser salvas. Uma desvantagem do uso deste BD é a restrição de acesso aos dados salvos apenas ao computador onde a aplicação foi instalada, podendo um problema no computador tornar os dados inacessíveis.

Após a elaboração deste projeto, a implementação de um modelo computadorizado 3D do braço robótico, reproduzindo os movimentos realizados pelo

braço real foi considerada como uma melhoria do projeto. A ideia seria que o modelo reproduzisse todos os movimentos do braço robótico, de acordo com o controle executado pelo usuário. Dessa forma, o controle do braço poderia ser realizado de um local onde o acesso não é possível ou viável, mas o usuário ainda teria conhecimento sobre as movimentações realizadas.

Nos trabalhos VENANCIO TEIXEIRA; DA SILVA HOUNSELL, (2018); VIANA et al., (2018) foram criados modelos 3D de braços robóticos para diferentes fins, porém auxiliam na exemplificação de como seria um braço robótico em uma simulação computadorizada.

### 5.3 Cálculo e análise dos parâmetros da Lei de Fitts

Analisando a Figura 21, que representa a relação entre o índice de dificuldade e o tempo de execução das tarefas utilizando o cubo menor, é possível notar que para o ID fácil o tempo é menor que nas séries dos IDs médio e difícil. Da mesma forma, o tempo de execução para o ID médio é menor do que para o ID difícil. Apenas com essa análise, já podemos comprovar a teoria da Lei de Fitts, uma vez que, à medida que o índice de dificuldade aumenta, o tempo de execução da tarefa também aumenta.

Da mesma forma, após a execução das tarefas da LF com o cubo maior, os dados de índice de dificuldade foram relacionados ao tempo de execução de cada tarefa, podendo ser observados na Figura 22. Seguindo as observações feitas à Figura 21, também podemos verificar que conforme o ID aumenta, o tempo de execução também aumenta.

Para facilitar a análise e comparação das tarefas executadas com os diferentes objetos, podemos consultar a Figura 23. Nela, as retas contínuas representam a relação entre o ID e o tempo médio de execução das tarefas, a cor amarela representa o cubo menor e a cor roxa representa o cubo maior.

Ainda que as tarefas foram executadas com objetos diferentes, o valor do índice de dificuldade foi mantido, conforme visto na Tabela 4. Apesar disso, é perceptível a diferença nos valores de tempo entre os objetos utilizados, sendo ela mais destacada no nível difícil. Isto posto, podemos indicar que não só o índice de dificuldade influencia no tempo de execução da tarefa, mas o tamanho do objeto também é um elemento a ser considerado.

A partir das retas tracejadas, onde foi feita a regressão linear das retas contínuas, foram extraídas as equações (3) e (4).

Analisando primeiro os dados do cubo menor, temos a equação (3) que, quando comparada à equação (2), podemos identificar os valores dos coeficientes angular e linear, sendo eles respectivamente, $a = 15765$ e $b = -21140$. Relembrando que a Lei de Fitts é representada por uma equação de reta, e que o índice de dificuldade já foi calculado, a equação (3) é uma equação genérica, nos moldes da Lei de Fitts, para explicar o tempo de movimento do cubo menor na realização desta tarefa para este voluntário.

Uma das formas de verificar se uma regressão linear representa adequadamente os dados coletados é por meio do coeficiente de determinação r². Ele indica o quanto as informações coletadas estão próximas da reta de regressão linear. Seu valor varia de zero a um, sendo muitas vezes representado utilizando porcentagem. Em geral, quanto maior o valor, mais o modelo se aproxima da realidade dos dados (MINITAB, 2019).

Para o cubo menor o valor de r² obtido na análise dos dados e da reta é de 0,954. Isso indica que 95,4% da informação coletada é explicada pela equação (3). Ela também pode ser usada para predizer qual será o tempo gasto, em média, na execução da tarefa quando variado o índice de dificuldade, mesmo que o usuário não tenha executado a tarefa em si.

A mesma análise pode ser feita para o cubo maior. Comparando as equações (4) e (2), obtém-se os coeficientes angular e linear $a = 10399$ e $b = -7595{,}3$, respectivamente. Para os dados coletados com este objeto, o valor de r² é de 0,9637. Isso indica que 96,37% dos dados é explicado por meio da equação (4), uma correlação ainda maior do que aquela observada com o cubo menor.

Vale ressaltar que os dados e equações apresentados aqui se aplicam apenas ao usuário que realizou as tarefas. Para cada novo voluntário que completar o procedimento estabelecido de coletas, serão gerados novos dados que serão analisados pela mesma metodologia aqui demonstrado. Como resultado, tem-se uma nova equação para cada objeto, juntamente com os coeficientes angular, linear e de determinação para explicar, por meio da Lei de Fitts, os movimentos realizados.

Testes com o usuário, usando ID diferentes daqueles usados nos testes iniciais podem ser realizados a posteriori para a aplicação das equações (3) e (4), de forma a verificar na prática o seu uso.

# 6. CONCLUSÃO

A partir dos resultados aqui demonstrados, conclui-se que o projeto alcançou o objetivo proposto de criar um sistema de braço robótico, com feedback para o usuário utilizando uma caixa controladora. Com o programa Visual Studio 2017 foi desenvolvida a interface gráfica que funciona juntamente com o banco de dados local, criado com o programa PostgreSQL. Esse conjunto de ferramentas se provou eficiente na coleta de dados das tarefas da Lei de Fitts e, posteriormente, permitiu os cálculos para análise dos seus parâmetros, que comprovaram a robustez da lei.

Dessa forma, o sistema robótico apresentado se provou eficaz e possui funcionalidade para aplicação em diferentes estudos. O uso da LF demonstrou uma das maneiras de utilizar o sistema, explorando a interação do usuário com uma interface gráfica, que também pode ser utilizada em projetos com objetivos além do cálculo dos parâmetros da Lei de Fitts.

A criação e implementação de todo o projeto visou o seu uso em outras aplicações, permitindo que outros estudantes do curso de Engenharia Biomédica também pudessem desenvolver outras vertentes de estudo tendo como base o mesmo sistema.

## REFERÊNCIAS

ADAFRUIT. **Adafruit PWM Servo Driver Library**. Disponível em: <https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library>.

ALIMI, A. M. Speed / accuracy trade-offs in target-directed movements. **Behavioral and brain sciences**, v. 20, p. 279–349, 1997.

ALLOUCHE, A. Software News and Updates Gabedit — A Graphical User Interface for Computational Chemistry Softwares. **Journal of computational chemistry**, v. 32, p. 174–182, 2012.

ANDRES, R. O.; HARTUNG, K. J. Prediction of head movement time using Fitts' law. **Human Factors**, v. 31, n. 6, p. 703–713, 1989.

ARDUINO. **Wire Library**. Disponível em: <https://www.arduino.cc/en/reference/wire>. Acesso em: 4 jun. 2020.

BAJERSKI, I.; ABELLA, V. D. B. **Braço Robótico Com Controle Remoto Bluetooth**. [s.l.] Pontifícia Universidade Católica do Rio Grande do Sul, 2010.

BARBOSA, W. H. A.; SILVA, R. DE O. **Braço articulado comandado via bluetooth por um aplicativo desenvolvido na plataforma android**. [s.l.] Universidade de Rio Verde, UNIRV, [s.d.].

BARRIENTOS, A. et al. **Fundamentos de Robótica | 2da Edicion | Antonio Barrientos, Luis Felipe Peñin, Carlos Balaguer, Rafael Aracil Santoja**. 2. ed. Madrid: McGraw-Hill/Interamericana de de España, S.A.U., 2014.

CARD, S. K.; ENGLISH, W. K.; BURR, B. J. Evaluation of Mouse , Rate-Controlled Isometric Joystick , Step Keys , and Text Keys for Text Selection on a CRT. **English**, v. 21, n. March 2012, p. 37–41, 2007.

CARDOSO, M. C.; GONÇALVES, B. S.; OLIVEIRA, S. R. R. E. Avaliação de ícones para interface de um sistema médico on-line. **Revista Brasileira de Design da Informação**, v. 10, p. 14, 2013.

COSTA, L. C. et al. **CONTROLE DE UM BRAÇO MECÂNICO POR MEIO DE MOVIMENTOS REAIS DE UM BRAÇO HUMANO**. XXIV Congresso Brasileiro de Engenharia Biomédica - CBEB 2014. **Anais**...Uberlândia - MG: 2014

ELETRONICA, M. **I2c – protocolo de comunicação mcu**. Disponível em: <http://www.meccomeletronica.com/site/data/uploads/i2c-protocolo-de-comunicacao.pdf>. Acesso em: 17 set. 2020.

INC. STAFF. **Robotics**. Disponível em: <https://www.inc.com/encyclopedia/robotics.html#:~:text=The Robotic Industries Association (RIA,however%2C the industry's current working>. Acesso em: 17 nov. 2020.

IXDF COURSE INSTRUCTOR. **Fitts ' s Law : The Importance of Size and Distance in UI Design Applying Fitts ' s Law to User Interface Design**. Disponível em: <https://www.interaction-design.org/literature/article/fitts-s-law-the-importance-of-size-and-distance-in-ui-design>. Acesso em: 10 nov. 2020.

LEMES, D. DE O. Aspectos gerais de uso das interfaces gráficas de usuário. **Revista Digital de Tecnologias Cogntivas**, v. 18, p. 37–46, 2018.

MACEDO, G. M. **10 heur í sticas de Nielsen para o design de interface**. Disponível em: <https://brasil.uxdesign.cc/10-heurísticas-de-nielsen-para-o-design-de-interface-58d782821840>. Acesso em: 18 nov. 2020.

MACHADO, L. S.; MORAES, R. M. DE; NUNES, F. L. S. Serious Games para Saúde e Treinamento Imersivo. In: **Abordagens práticas de realidade virtual e aumentada**. Porto Alegre: SBC, 2009. p. 32.

MAGEZI, D. A. Linear mixed-effects models for within-participant psychology experiments: An introductory tutorial and free, graphical user interface (LMMgui). **Frontiers in Psychology**, v. 6, n. JAN, p. 1–7, 2015.

MINITAB. **Análise de regressão: Como interpretar o R-quadrado e avaliar a qualidade de ajuste?** Disponível em: <https://blog.minitab.com/pt/analise-de-regressao-como-interpretar-o-r-quadrado-e-avaliar-a-qualidade-de-ajuste>. Acesso em: 5 nov. 2020.

OKAZAKI, V. H. A. et al. Speed-accuracy trade-off in task of drawing geometric figures. **Revista Brasileira de Ciências do Esporte**, v. 33, n. 1, p. 249–264, 2011.

PAUL M. FITTS. the Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. **Journal of Experimental Psychology**, v. 47, n. 6, p. 381–391, 1954.

POSTGRESQL. **About Postgresql**. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 27 ago. 2020.

SANTOS, V. M. F. **Robótica Industrial**. Aveiro: [s.n.].

SILVEIRA, A. L. M. DA; RIBEIRO, V. G. Os princípios básicos para o design das interfaces gráficas de usuário: uma revisão de literatura histórica. **Revista Educação Grafica**, v. 17, n. 3, p. 25, 2013.

SILVEIRA, A. L. M. DA; RIBEIRO, V. G.; FOLLE, L. F. AS TEORIAS QUE FUNDAMENTAM OS PRINCÍPIOS BÁSICOS PARA O DESIGN DAS INTERFACES GRÁFICAS DE USUÁRIO. **Revista Educação Gráfica**, v. 18, n. 3, p. 18, 2014.

VENANCIO TEIXEIRA, J.; DA SILVA HOUNSELL, M. Desenvolvimento de um Simulador 3D com Modos de Treinamento. **Revista Principia - Divulgação Científica e Tecnológica do IFPB**, v. 1, n. 39, p. 79, 17 abr. 2018.

VIANA, C. P. et al. GRAPHIC USER INTERFACE FOR A 3D COORDINATE ARM. **Congresso de Engenharias da UFSJ**, p. 11, 2018.

WU, J.; YANG, J.; HONDA, T. Fitts' law holds for pointing movements under conditions of restricted visual feedback. **Human Movement Science**, v. 29, n. 6, p. 882–892, 2010.

ZARUZ, T. F. **Software to control robotic arm system.** Zenodo, , 22 nov. 2020a. Disponível em: <https://doi.org/10.5281/zenodo.4284953#.X8B3RUDAZ0A.mendeley>. Acesso em: 22 nov. 2020

ZARUZ, T. F. **Graphical user interface in C# to execute Fitts'Law tasks with a robotic system.** Zenodo, , 29 nov. 2020b. Disponível em: <https://doi.org/10.5281/zenodo.4296994#.X8QV0tSyHdc.mendeley>. Acesso em: 29 nov. 2020

# APÊNDICE A – SOFTWARE PARA CONTROLE DO BRAÇO ROBÓTICO EM ARDUINO

```
/* Software to control robotic arm    by Thaíla Zaruz

  ---------- Notes ----------------------------------
  1) Joystick button is HIGH. When pressed is LOW
  2) User interface in visual studio (C#)
  3) Database in POSTGRESQL
*/

#include <Adafruit_PWMServoDriver.h>    // servo motors library
#include <Wire.h>         // allows communication via I2C

/*  JOYSTICK  VARIABLES  */
const byte J_X_PIN = A0;    //x axis pin
const byte Z_AXIS_PIN = 9;  //z axis pin (button)

/*  SELECTING MOTORS  */
int command = 6;    //value of selected motor
int dir_dof = 0;    //direction of joystick when selecting motor
int dof = 0;      //motor selected to move
int select = -1;    //digital value of joystick button
bool confirmation = false;    //true when a motor is selected to move
int dir_motor = 0;    //joystick direction when moving a motor

/*  DEBOUNCE TIME BUTTON  */
unsigned long debounceDelay = 100;     //debounce time to the button
unsigned long lastDebounceTime = 0;   //last time the button was pressed

/*  OBJECT  AND BUZZER*/
const byte OBJ_PIN = 10;    //pin to identify if claw grabs the object
bool state_obj = HIGH;    //current state of object, HIGH when claw has the object
bool last_state_obj = LOW;    //last state of object

const byte BUZZER_PIN = 11;   //buzzer pin

const byte LED_PINS[] = {2, 3, 4, 5, 6, 7};   //led pins, represents wich motor is selected
const byte MOV_LED_PIN = 8;   //led pin, HIGH when a motor is moving

/* SERVO MOTORS */
const byte NUMBER_OF_SERVOS = 6;    //amount of motors in robotic arm
const uint16_t MIN[] = {360, 120, 140, 160, 180, 200};   //minimum pwm value of wich
motor    //claw to base
const uint16_t MAX[] = {640, 530, 490, 540, 600, 630};    //maximum pwm value of wich
motor    //claw to base
const uint16_t START[] = {640, 344, 450, 500, 600, 440};   //pwm value to start wich motor
//claw starts closed
const byte SERVOS_SPEED[] = {2, 1, 1, 1, 1, 1};   //moving velocity of wich motor
```

```
uint16_t lastPositions[] = {640, 344, 450, 500, 600, 440};    //last position of wich motor
//claw to base

uint8_t servonum = 12;    //driver pin with the first motor
byte moving = 6;        //number of motor currently moving
byte lastMoving = 6;      //number of last motor moving

/*  TIME VARIABLES */
unsigned long initial_time;   //saves initial movement time
unsigned long final_time;   //saves final movement time
unsigned long task_time;   //saves execution task time
bool task_running = false;    //verify if a task is in execution
String IT, FT, TT;      //variables to save initial time, final time and task time in string format
char it[60], ft[60], tt[60];      //variables initial time, final time and task time in char format
int Lit, Lft, Ltt;    //length of string variables of initial time, final time and task time

/*******************************/

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();      //library to use pca9685

/*******************************/

void moveToHome() {       //move the robotic arm to its initial position
  for (byte i = 0; i < NUMBER_OF_SERVOS; i++) {
    pwm.setPWM(servonum - i, 0, START[i]);
    delay(50);
  }
}

void start_time() {     //start time counting to fitts law
  initial_time = millis();
  task_running = true;

  //transforms initial_time to char, in order to send the information to user interface
  IT = String(initial_time, DEC);
  Lit = (IT.length()) + 1;
  IT.toCharArray(it, Lit);
  for (int j = 0; j <= Lit; j++) {
    Serial.write(it[j]);
  }
  Serial.write("i");
}

void stop_time() {    //stop time counting to fitts law
  final_time = millis();
  task_time = final_time - initial_time;
  task_running = false;

  if (task_time < 1000) {       //eliminates accidental touchs or when the object falls from the
claw
```

```
    initial_time = 0;
    final_time = 0;
    task_time = 0;
  }
  else {      //transforms final_time and task_time to char, in order to send the information to
user interface

    FT = String(final_time, DEC);
    Lft = (FT.length()) + 1;
    FT.toCharArray(ft, Lft);
    for (int j = 0; j <= Lft; j++) {
      Serial.write(ft[j]);
    }
    Serial.write("f");

    TT = String(task_time, DEC);
    Ltt = (TT.length()) + 1;
    TT.toCharArray(tt, Ltt);
    for (int j = 0; j <= Ltt; j++) {
      Serial.write(tt[j]);
    }
    Serial.write("t");
    Serial.write("Z");
  }
}

void moveObject() {   //function called when claw is moving, to identify when the object is
moving

  state_obj = digitalRead(OBJ_PIN);       //read object pin

  if ((state_obj == HIGH) && (state_obj != last_state_obj) && (task_running == false)) {
//grabed object
    Serial.write("G");      //send to user interface that the object was grabed
    digitalWrite(MOV_LED_PIN, LOW);   //blink led to indicate the object was grabed
    start_time();
    last_state_obj = state_obj;
    delay(100);
    digitalWrite(MOV_LED_PIN, HIGH);
  }
  else if ((state_obj == LOW) && (state_obj != last_state_obj) && (task_running == true)) {
//released object
    Serial.write("R");      //send to user interface that the object was released
    digitalWrite(MOV_LED_PIN, LOW);   //blink led to indicate the object was released
    stop_time();
    last_state_obj = state_obj;
    delay(100);
    digitalWrite(MOV_LED_PIN, HIGH);
  }
}
```

```
//controls the movement of selected motor
void moveServo() {

  while (confirmation) {      //stop condition of function moveServo()
    int stop_it;    //reads button value
    if ((millis() - lastDebounceTime) > debounceDelay) {              //if enough time has passed between clicks
      stop_it = digitalRead(Z_AXIS_PIN);
      if (stop_it == LOW)  {              //if button was pressed during servo moving, stop moveServo function
        moving = NUMBER_OF_SERVOS;
        confirmation = false;

        tone(BUZZER_PIN, 247, 200);       //plays a sound to indicates it is no longer moving a motor
        delay(20);
        tone(BUZZER_PIN, 247, 200);
        lastDebounceTime = millis();      //updates time of last button click
        digitalWrite(MOV_LED_PIN, LOW);
        break;
      }
    }     //stop condition

    if ((moving != NUMBER_OF_SERVOS) && (confirmation == true)) {     //certifies if a motor is selected
      digitalWrite(MOV_LED_PIN, HIGH);
      uint16_t newPosition;
      dir_motor = analogRead(J_X_PIN);        //reads joystick
      dir_motor = map(dir_motor, 0, 1023, -15, 16);     //sets direction and velocity to move the motor
      newPosition = lastPositions[moving] + SERVOS_SPEED[moving] * (dir_motor / 2);     //calculates new position
      newPosition = constrain(newPosition, MIN[moving], MAX[moving]);       //restrain the new position to servo motor limits
      pwm.setPWM(servonum - moving, 0, newPosition);
      delay(80);

      lastPositions[moving] = newPosition;      //saves last position of each motor

      if (moving == 0) {      //claw is moving
        moveObject();         //function to identifie if object was caught or not
      }
    }
  }
}

void setup() {

  Serial.begin(9600);
```

```
  pinMode(BUZZER_PIN, OUTPUT);      //Initiates buzzer pin
  pinMode(Z_AXIS_PIN, INPUT);       //Initiates button
  pinMode(OBJ_PIN, INPUT);      //Initiates object pin
  pinMode(LED_PIN, OUTPUT);     //Initiates led indicator
  pinMode(MOV_LED_PIN, OUTPUT);

  Wire.begin();         //Open I2C communication (SDA/SCL) to control servo motors
  pwm.begin();          //Initiates servo motors
  pwm.setPWMFreq(60);   // Analog servos run at ~60 Hz updates
  moveToHome();       //Move motors to initial position

  for (int k = 0; k < NUMBER_OF_SERVOS; k++) {        //Initiates motors leds indicators
    pinMode(LED_PINS[k], OUTPUT);
    digitalWrite(LED_PINS[k], HIGH);
    delay(500);
    digitalWrite(LED_PINS[k], LOW);
  }
}

void loop() {

  //Selects the motor to move
  confirmation = false;
  dir_dof = analogRead(J_X_PIN);     //read x axis from joystick
  dir_dof = map(dir_dof, 0, 1020, -9, 10);     //limits joystick values from -1 to 1
  dir_dof = constrain(dir_dof, -1, 1);        //limits the value from -1 to 1
  dof += dir_dof;       //selects servo motor, from 0 to 5
  dof = constrain(dof, 0, 5);     //limits the value to the number of motors
  delay(250);


  switch (dof) {      //show through leds wich motor is pre selected

    case 0:                           //motor A
      for (int j = 0; j < NUMBER_OF_SERVOS; j++) {
        digitalWrite(LED_PINS[j], LOW);
      }
      digitalWrite(LED_PINS[dof], HIGH);
      Serial.write("A");
      break;

    case 1:                           //motor B
      for (int j = 0; j < NUMBER_OF_SERVOS; j++) {
        digitalWrite(LED_PINS[j], LOW);
      }
      digitalWrite(LED_PINS[dof], HIGH);
      Serial.write("B");
      break;
```

```
    case 2:                         //motor C
      for (int j = 0; j < NUMBER_OF_SERVOS; j++) {
        digitalWrite(LED_PINS[j], LOW);
      }
      digitalWrite(LED_PINS[dof], HIGH);
      Serial.write("C");
      break;

    case 3:                         //motor D
      for (int j = 0; j < NUMBER_OF_SERVOS; j++) {
        digitalWrite(LED_PINS[j], LOW);
      }
      digitalWrite(LED_PINS[dof], HIGH);
      Serial.write("D");
      break;

    case 4:                         //motor E
      for (int j = 0; j < NUMBER_OF_SERVOS; j++) {
        digitalWrite(LED_PINS[j], LOW);
      }
      digitalWrite(LED_PINS[dof], HIGH);
      Serial.write("E");
      break;

    case 5:                         //motor F
      for (int j = 0; j < NUMBER_OF_SERVOS; j++) {
        digitalWrite(LED_PINS[j], LOW);
      }
      digitalWrite(LED_PINS[dof], HIGH);
      Serial.write("F");
      break;
  }

  select = digitalRead(Z_AXIS_PIN);     //Read button state

  if ((select == LOW) && (confirmation == false)) {       //if button was pressed
    if ((millis() - lastDebounceTime) > debounceDelay) {      //if has passed enough time
between button clicks
      command = dof;        //confirms the selected motor
      lastMoving = moving;       //saves wich motor was last mooved
      moving = dof;              //saves wich motor is curently mooving
      confirmation = true;       //ensure a motor was selected
      lastDebounceTime = millis();    //saves when the button was clicked
    }
  }

  if (confirmation == true) {     //a motor was selected

    switch (command) {
      case 0:       //motor A selected
```

```
      Serial.write("M");    //tells user interface there is a motor currently moving
      moving = 0;     //saves number of wich motor is moving
      tone(BUZZER_PIN, 2620, 400);     //plays Dó sound
      break;

    case 1:       //motor B selected
      Serial.write("M");    //tells user interface there is a motor currently moving
      moving = 1;
      tone(BUZZER_PIN, 2940, 400);     //plays Ré sound
      break;

    case 2:       //motor C selected
      Serial.write("M");    //tells user interface there is a motor currently moving
      moving = 2;
      tone(BUZZER_PIN, 3300, 400);     //plays Mi sound
      break;

    case 3:       //motor D selected
      Serial.write("M");    //tells user interface there is a motor currently moving
      moving = 3;
      tone(BUZZER_PIN, 3490, 400);     //plays Fá sound
      break;

    case 4:       //motor E selected
      Serial.write("M");    //tells user interface there is a motor currently moving
      moving = 4;
      tone(BUZZER_PIN, 3920, 400);     //plays Sol sound
      break;

    case 5:       //motor F selected
      Serial.write("M");    //tells user interface there is a motor currently moving
      moving = 5;
      tone(BUZZER_PIN, 4400, 400);     //plays Lá sound
      break;

    case 6:       //waiting comand
      moving = 6;
      break;

   }
   while (confirmation) {      //if a motor was selected to move
    moveServo();      //properly moves the selected motor
   }
  }
}
```

# APÊNDICE B – CÓDIGO DA PÁGINA DE FORMULÁRIO DA INTERFACE GRÁFICA EM C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Npgsql;
using System.Threading;

namespace teste15_tcc
{
    public partial class Form1 : Form
    {

        Thread fittsapp;            //initiates a new thread to initiate second page

        public Form1()
        {
            InitializeComponent();      //initiate form
            label10.Visible = true;
            label10.Text = DateTime.Now.ToShortDateString();        //takes current
date time
        }

        DateTime currently_date = DateTime.Today;
        string R_name;          //researcher name
        string R_phone;         //researcher phone
        string R_email;         //researcher email
        string V_name;          //volunteer name
        DateTime V_birth;       //volunteer birth
        string V_sex;           //volunteer sex
        string V_phone;         //volunteer phone
        string V_email;         //volunteer email
        Int32 id_volunteer;     //volunteer id from database

        private void btSave_Click(object sender, EventArgs e)
        {
            if (IsEmpty())          //check if there is an empty field before saving
data
            {
                MessageBox.Show("Fill all the fields to save data!",
                                "Error",
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Error);
            }
            else if (!IsEmpty())
            {
                //saves information filled in form into variables
                R_name = txbrname.Text;
                R_phone = txbrphone.Text;
                R_email = txbremail.Text;
                V_name = txbvname.Text;
                V_birth = Convert.ToDateTime(txbvbirth.Text);
                GetSex();
                V_phone = txbvphone.Text;
```

```
                V_email = txbvemail.Text;
                Register form_info = new Register(currently_date, R_name, R_phone,
R_email, V_name, V_birth, V_sex, V_phone, V_email);      //call class register to
register new volunteer
                MessageBox.Show(form_info.message);         //show status from
database update
                if (!form_info.save)        //shows error code in case of error
updating database
                {
                    MessageBox.Show(form_info.error);
                }
                id_volunteer = form_info.GetVolunteer();        //get number of
volunteer id



                if (form_info.save)             //case when update database was
successfull
                {
                    ResetInfo();        //clear fields
                    this.Close();       //close form
                    fittsapp = new Thread(newform);     //creates new form
                    fittsapp.SetApartmentState(ApartmentState.STA);
                    fittsapp.Start();       //starts new form
                }

            }
        }

        private void GetSex()       //check wich sex was selected
        {
            if (rbtmale.Checked)
            {
                this.V_sex = "Male";
            }
            else if (rbtfemale.Checked)
            {
                this.V_sex = "Female";
            }
        }

        private bool IsEmpty()      //check if there is an empty field
        {
            if ((txbrname.Text == "") || (txbremail.Text == "") || (txbvname.Text ==
"") || (txbvbirth.Text == "") || (txbvphone.Text == "") || (txbvemail.Text == ""))
            {
                return true;
            }
            else if ((rbtmale.Checked == false) && (rbtfemale.Checked == false))
            {
                return true;
            }
            else if ((!txbrphone.MaskFull) || (!txbvbirth.MaskFull) ||
(!txbvphone.MaskFull))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
```

```
        private void ResetInfo()            //erase fields after saving or when click
in reset button
        {
            txbrname.Text = "";
            txbrphone.Text = "";
            txbremail.Text = "";
            txbvname.Text = "";
            txbvbirth.Text = "";
            rbtmale.Checked = false;
            rbtfemale.Checked = false;
            txbvphone.Text = "";
            txbvemail.Text = "";
        }

        private void btReset_Click(object sender, EventArgs e)      //reset button
        {
            ResetInfo();
        }

        private void newform()      //run new form, sending variable id_volunteer
        {
            Application.Run(new Form2(id_volunteer));
        }

        private void txbvbirth_Click(object sender, EventArgs e)        //position
cursor at beginning in volunteer birth field
        {
            txbvbirth.SelectionStart = 0;
        }

        private void txbvphone_Click(object sender, EventArgs e)        //position
cursor at beginnig in volunteer phone field
        {
            txbvphone.SelectionStart = 0;
        }

        private void txbrphone_Click(object sender, EventArgs e)        //position
cursor at beginning in researcher phone field
        {
            txbrphone.SelectionStart = 0;
        }

    }
}
```

# APÊNDICE C – CÓDIGO DA PÁGINA DE USUÁRIO DA INTERFACE GRÁFICA EM C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;
using Npgsql;
using System.Threading;


namespace teste15_tcc
{
    public partial class Form2 : Form
    {

        string RcString;
        int A = 65;         //letter A
        int B = 66;         //letter B
        int C = 67;         //letter C
        int D = 68;         //letter D
        int E = 69;         //letter E
        int F = 70;         //letter F
        int G = 71;         //letter G
        int R = 82;         //letter R
        int M = 77;         //letter M
        int it = 105;       //letter i
        int ft = 102;       //letter f
        int tt = 116;       //letter t
        int end = 90;       //letter Z
        Int64 task_id = 1;
        Int32 volunteer;
        string difficulty = "";
        string type_obj = "";
        Int64 initial_time = 0;
        Int64 final_time = 0;
        Int64 task_time = 0;
        string I_time;
        string F_time;
        string T_time;
        char time;
        string time_to_fill = "";
        bool acquisition = false;
        bool begin = false;
        string[] row;
        bool first;
        double d_value;
        double[] id = { 2.58, 3.13, 3.7 };      //difficulty index


        delegate void SetTextCallback(Int64 text);

        private BackgroundWorker hardWorker;        //background work for new thread

        private Thread readThread = null;       //initiates thread
```

```
        public Form2(Int32 vol)         //initializes form
        {
            InitializeComponent();
            timerCOM.Enabled = true;        //starts clock
            hardWorker = new BackgroundWorker();
            volunteer = vol;        //volunteer id
            first = true;
        }

        Pen red = new Pen(Color.DarkRed);       //rectangle design and colors
        Pen green = new Pen(Color.LimeGreen);

        SolidBrush fillred = new SolidBrush(Color.Firebrick);
        SolidBrush fillgreen = new SolidBrush(Color.Lime);

        Rectangle rect = new Rectangle(298, 253, 70, 70);


        private void Form2_Paint(object sender, PaintEventArgs e)       //draw
rectangle
        {
            Graphics g = e.Graphics;
            g.DrawRectangle(red, rect);
            g.FillRectangle(fillred, rect);
        }

        private void updateCOMs()           //check available ports to communicate
        {
            int i;
            bool diff;
            i = 0;
            diff = false;

            if (cbPort.Items.Count == SerialPort.GetPortNames().Length)
            {
                foreach (string s in SerialPort.GetPortNames())
                {
                    if (cbPort.Items[i++].Equals(s) == false)
                    {
                        diff = true;
                    }
                }
            }
            else
            {
                diff = true;
            }
            if (diff == false)
            {
                return;
            }
            cbPort.Items.Clear();
            foreach (string s in SerialPort.GetPortNames())
            {
                cbPort.Items.Add(s);
            }
            cbPort.SelectedIndex = 0;
        }
```

```
        private void timerCOM_Tick(object sender, EventArgs e)      //update COM list
at 1 sec
        {
            updateCOMs();

        }
        private void btConnect_Click(object sender, EventArgs e)        //connect with
usb port
        {
            if (serialPort1.IsOpen == false)
            {
                serialPort1.NewLine = Convert.ToString(10);
                try
                {
                    if (cbPort.Text == "Port")
                    {
                        MessageBox.Show("Select port to connect",
                                        "Attention",
                                        MessageBoxButtons.OK,
                                        MessageBoxIcon.Exclamation);
                    }
                    else
                    {
                        serialPort1.PortName =
cbPort.Items[cbPort.SelectedIndex].ToString();
                        serialPort1.Open();
                        timerCOM.Enabled = false;           //stop timer while arduino
is connected
                    }
                }
                catch
                {
                    return;
                }
                if (serialPort1.IsOpen)
                {
                    readThread = new Thread(new ThreadStart(Read));        //thread to
read income data
                    readThread.Start();
                    hardWorker.RunWorkerAsync();
                    btConnect.Text = "Disconnect";
                    cbPort.Enabled = false;

                }
            }
            else           //when it's already connected, to disconnect
            {
                try
                {
                    if (acquisition)
                    {
                        MessageBox.Show("Finish Data Acquisition before disconnect!",
                                        "Error",
                                        MessageBoxButtons.OK,
                                        MessageBoxIcon.Exclamation);
                    }
                    else
                    {
                        serialPort1.Close();
                        cbPort.Enabled = true;
                        btConnect.Text = "Connect";
                        timerCOM.Enabled = true;
```

```
                }
            }
            catch
            {
                return;
            }
        }
    }

    private void Form2_FormClosed(object sender, FormClosedEventArgs e)
//finishes thread, usb connection and form when closed
    {
        try
        {
            if (!(readThread == null))
                readThread.Abort();
        }
        catch (NullReferenceException) { }
        try
        {
            if (serialPort1.IsOpen == true)
                serialPort1.Close();
        }
        catch (NullReferenceException) { }
    }

    private void PrintText(Int64 text)          //compares data received from
arduino
    {
        if (this.textBox1.InvokeRequired)
        {
            SetTextCallback d = new SetTextCallback(PrintText);
            this.Invoke(d, new object[] { text });
        }
        else
        {
            Graphics g_object = CreateGraphics();
            labelMoving.Visible = false;

            if (acquisition)
            {
                if (text.Equals(A))           //motor A
                {
                    labelMotor.Text = "A";
                }
                else if (text.Equals(B))        //motor B
                {
                    labelMotor.Text = "B";

                }
                else if (text.Equals(C))        //motor C
                {
                    labelMotor.Text = "C";

                }
                else if (text.Equals(D))        //motor D
                {
                    labelMotor.Text = "D";

                }
                else if (text.Equals(E))        //motor E
                {
```

```
                    labelMotor.Text = "E";

                }
                else if (text.Equals(F))        //motor F
                {
                    labelMotor.Text = "F";

                }
                else if (text.Equals(M))        //if motor is moving
                {
                    labelMoving.Visible = true;
                    labelMoving.Text = "Moving...";
                }
                else if (text.Equals(G))        //if object is grabbed
                {
                    g_object.DrawRectangle(green, rect);
                    g_object.FillRectangle(fillgreen, rect);
                }
                else if (text.Equals(R))        //ifi object is released
                {
                    g_object.DrawRectangle(red, rect);
                    g_object.FillRectangle(fillred, rect);

                }
                else if (text.Equals(it))       //saves initial time from
fitts'task
                {
                    if (!begin)
                    {
                        initial_time = Convert.ToInt64(time_to_fill);
                        time_to_fill = "";
                        begin = true;
                        I_time = Convert.ToString(initial_time);
                    }
                }
                else if (text.Equals(ft))       //saves final time from fitts'task
                {
                    if (begin)
                    {
                        final_time = Convert.ToInt64(time_to_fill);
                        time_to_fill = "";
                        F_time = Convert.ToString(final_time);
                    }
                }
                else if (text.Equals(tt))       //saves execution time from
fitts'task
                {
                    if (begin)
                    {
                        task_time = Convert.ToInt64(time_to_fill);
                        time_to_fill = "";
                        T_time = Convert.ToString(task_time);
                        begin = false;
                    }
                }
                else if (text.Equals(end))      //save data in database
                {
                    StartAcquisition acq = new StartAcquisition(first, volunteer,
type_obj, task_id, difficulty, d_value, initial_time, final_time, task_time);

                    if (!acq.save)      //show error messages
                    {
```

```
                            MessageBox.Show(Convert.ToString(first));
                            MessageBox.Show(acq.message);
                            MessageBox.Show(acq.error);
                        }
                        else
                        {       //show initial time, final time and execution time in
data grid
                            row = new string[] { I_time, F_time, T_time };
                            table.Rows.Add(row);
                            task_id++;
                            first = false;
                        }
                    }
                    else if ((text < 58) && (text > 47))        //excludes different
characters
                    {
                        time = Convert.ToChar(text);
                        time_to_fill += time;
                    }
                }
            }

        }

        public void Read()      //reads data received from arduino when available
        {
            while (serialPort1.IsOpen)
            {
                try
                {
                    if (serialPort1.BytesToRead > 0)
                    {
                        Int64 message = serialPort1.ReadChar();
                        if ((message != 0) && (message != 10) && (message != 13))
                        {
                            this.PrintText(message);
                        }
                    }
                }
                catch (TimeoutException) { }
            }
        }


        private void btStart_Click(object sender, EventArgs e)      //starts
acquisition of fitts'tasks
        {
            if (cbID.Text == "Select...")       //check if a difficulty index was
selected
            {
                MessageBox.Show("Select difficulty index to start acquisition",
                                "Attention",
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Exclamation);
            }
            if (!serialPort1.IsOpen)        //check if a usb connection was selected
            {
                MessageBox.Show("Select port and connect to device first!",
                                "Attention",
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Warning);
            }
```

```
            if (cbObject.Text == "Object")      //check if an object was selected
            {
                MessageBox.Show("Select the object type to start acquisition",
                                "Attention",
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Exclamation);
            }
            else
            {
                if (!acquisition)       //updates form elements during fitts'tasks
                {
                    table.Rows.Clear();
                    difficulty = cbID.Items[cbID.SelectedIndex].ToString();
                    type_obj = cbObject.Items[cbObject.SelectedIndex].ToString();
                    d_value = id[cbID.SelectedIndex];
                    cbID.Enabled = false;
                    cbObject.Enabled = false;
                    acquisition = true;
                    btStart.Text = "Stop Acquisition";
                    btStart.BackColor = SystemColors.ControlLightLight;
                    btStart.ForeColor = Color.Navy;
                }
                else            //updates form elements when fitts'tasks are finished
                {
                    cbID.Enabled = true;
                    cbObject.Enabled = true;
                    btStart.Enabled = true;
                    acquisition = false;
                    btStart.Text = "Start Acquisition";
                    btStart.BackColor = Color.Navy;
                    btStart.ForeColor = SystemColors.ControlLightLight;
                }
            }
        }
    }
}
```

# APÊNDICE D – CÓDIGO DA CLASSE *CONNECTION* DA INTERFACE GRÁFICA EM C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;

namespace teste15_tcc
{
    public class Connection         //class used to connect/disconnect to database
    {
        NpgsqlConnection con = new NpgsqlConnection();          //creation of new
connection with database

        public Connection()
        {
            con.ConnectionString = @"Server=localhost;Port=5432;User
Id=postgres;Password=4321lleD;Database=TCC_Thaila";        //string specifying
information from database
            //con.ConnectionString = @"Server=localhost;Port=5432;User
Id=postgres;Password=niats123;Database=postgres";          //string specifying
information from database
        }

        public NpgsqlConnection Connect()           //function to connect to database
        {
            if (con.State == System.Data.ConnectionState.Closed)        //if
connection is closed, open it
            {
                con.Open();
            }
            return con;         //returns a connection string
        }

        public void Disconnect()            //function to disconnect from database
        {
            if (con.State == System.Data.ConnectionState.Open)      //if connection is
opened, close it
            {
                con.Close();
            }
        }

    }
}
```

# APÊNDICE E – CÓDIGO DA CLASSE *REGISTER* DA INTERFACE GRÁFICA EM C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;

namespace teste15_tcc
{
    public class Register           //class used to save researcher/volunteer
information in database
    {
        NpgsqlCommand cmd = new NpgsqlCommand();        //stabilish new variable to
command string
        Connection cnt = new Connection();      //estabilish new connection
        public string message = "";         //string to inform user of saving status
        public bool save;           //variable to ensure if data was properly saved
        public string error;        //in case of erros to save data, displays the
error code
        Int32 volunteer;        //number of volunteer in database

        public Register(DateTime date, String Rname, String Rphone, String Remail,
String Vname, DateTime Vbirth, String Vsex, String Vphone, String Vemail)
        {
            //creates a table with needed collums in case it does not exists
            cmd.CommandText = @"CREATE TABLE IF NOT EXISTS
public.volunteer_information(" +
                "volunteer_id serial primary key, acquisition_date date not null," +
                " researcher_name varchar(200) not null, researcher_phone char(15) not
null," +
                "researcher_email varchar(100) not null, volunteer_name varchar(200)
not null," +
                "volunteer_birth date not null, volunteer_sex varchar(8) not null," +
                "    volunteer_phone char(15) not null, volunteer_email varchar(100)
not null)";
            try
            {
                cmd.Connection = cnt.Connect();
                cmd.ExecuteNonQuery();
                cnt.Disconnect();
            }
            catch (NpgsqlException e) { }

            //saves data inserted into first page of application
            cmd.CommandText = "insert into
public.volunteer_information(acquisition_date, researcher_name, researcher_phone,
researcher_email, volunteer_name, volunteer_birth, volunteer_sex, volunteer_phone,
volunteer_email)" +
                "values(@acquisition_date,@researcher_name, @researcher_phone,
@researcher_email, @volunteer_name,@volunteer_birth,
@volunteer_sex,@volunteer_phone,@volunteer_email)";

            //insert variable values inserted into form page
            cmd.Parameters.AddWithValue("acquisition_date", date);
            cmd.Parameters.AddWithValue("researcher_name", Rname);
            cmd.Parameters.AddWithValue("researcher_phone", Rphone);
            cmd.Parameters.AddWithValue("researcher_email", Remail);
            cmd.Parameters.AddWithValue("volunteer_name", Vname);
            cmd.Parameters.AddWithValue("volunteer_birth", Vbirth);
```

```
            cmd.Parameters.AddWithValue("volunteer_sex", Vsex);
            cmd.Parameters.AddWithValue("volunteer_phone", Vphone);
            cmd.Parameters.AddWithValue("volunteer_email", Vemail);
            try
            {
                cmd.Connection = cnt.Connect();
                cmd.ExecuteNonQuery();
                cnt.Disconnect();
                save = updateMessage(true);
            }
            catch (NpgsqlException e)
            {
                error = e.Message;
                save = updateMessage(false);
            }
        }

        public Int32 GetVolunteer()         //reads the number of the volunteer
registered in database
        {
            cmd.Connection = cnt.Connect();
            cmd.CommandText = "SELECT * FROM volunteer_information";
            NpgsqlDataReader rdr = cmd.ExecuteReader();
            while (rdr.Read())
            {
                volunteer = rdr.GetInt32(0);
            }
            cnt.Disconnect();
            return volunteer;
        }

        public bool updateMessage(bool ok)      //updates message shown to user
        {
            if (ok)
            {
                this.message = "Data succesfully saved!";       //in case data was
saved
                return true;
            }
            else
            {
                this.message = "Error trying to connect to database!";      //in case
there was trouble saving data
                return false;
            }
        }
    }
}
```

# APÊNDICE F – CÓDIGO DA CLASSE *START_ACQUISITION* DA INTERFACE GRÁFICA EM C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;

namespace teste15_tcc
{
    public class StartAcquisition       //class used to save data from fitts' law
tasks in database
    {
        NpgsqlCommand cmd = new NpgsqlCommand();        //stabilish new variable to
command string
        Connection cnt = new Connection();      //estabilish new connection
        public string message = "first";        //indicates a failure to update
database
        public string error = "unknown error";         //error value caught from
database
        public bool save;       //indicates success or failure in saving data to
database

        public StartAcquisition(Boolean first, Int32 v_id, String obj, Int64 task_id,
String dif_index, Double id_value,Int64 init_time, Int64 fin_time, Int64 task_time)
        {
            //creates a new table in case it does not exists
            if (first)
            {
                Console.WriteLine("Criar tabela");
                cmd.CommandText = @"CREATE TABLE IF NOT EXISTS
public.data_acquisition1(" +
                                    "acquisition serial primary key, volunteer_id
integer not null," +
                                    "type_object varchar(20) not null, task_id bigint
not null," +
                                    "index_difficulty varchar(7) not null, ID_value
float not null," +
                                    "initial_time bigint not null," +
                                    "final_time bigint not null, task_time bigint not
null," +
                                    "FOREIGN KEY(volunteer_id) REFERENCES
public.volunteer_information (volunteer_id))";
                try
                {
                    cmd.Connection = cnt.Connect();
                    cmd.ExecuteNonQuery();
                    cnt.Disconnect();
                    first = false;
                }
                catch (NpgsqlException e)
                {
                    //message = "Pegou erro";
                    error = e.Message;
                    save = updateMessage(false);
                }
            }
            //updates table with data from fitts'law tasks
```

```
                cmd.CommandText = "insert into public.data_acquisition1(volunteer_id,
type_object, task_id, index_difficulty, ID_value, initial_time, final_time,
task_time)" +
                    "values(@v_id, @obj, @t_id,@dif_index, @id_val, @i_time,
@f_time,@t_time)";      //comand line to database

                cmd.Parameters.AddWithValue("v_id", v_id);
                cmd.Parameters.AddWithValue("obj", obj);
                cmd.Parameters.AddWithValue("t_id", task_id);
                cmd.Parameters.AddWithValue("dif_index", dif_index);
                cmd.Parameters.AddWithValue("id_val", id_value);
                cmd.Parameters.AddWithValue("i_time", init_time);
                cmd.Parameters.AddWithValue("f_time", fin_time);
                cmd.Parameters.AddWithValue("t_time", task_time);
                try
                {
                    cmd.Connection = cnt.Connect();
                    cmd.ExecuteNonQuery();
                    cnt.Disconnect();
                    save = updateMessage(true);
                }
                catch (NpgsqlException e)
                {
                    error = e.Message;
                    save = updateMessage(false);
                }

            bool updateMessage(bool ok)
            {
                if (!ok)
                {
                    this.message = "Error to update database.";
                    return false;
                }
                else
                {
                    return true;
                }
            }
        }
    }
}
```