Sistema Embarcado Aplicado em Controle *Joystick* para Computador

Lamartine A. P. de Queiroga[1], Jessé P. Coutinho[1], Filipe A. Lira[1], Denis da Silva[1], Francisco Laurindo Costa Junior[1], Sandro C. S. Jucá[1]

[1] Eixo da Computacão - Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) - Av. Parque Central - CEP 61919-140 - Maracanaú - CE - Brasil

lapq2006@gmail.com, jessecdm@gmail.com, filipe.al2015@gmail.com
denispsilvace@gmail.com, laucostajr@gmail.com, sandro.juca@gmail.com

Abstract. *This paper describes the process of development of an arcade-style videogame controller, with a USB jack, to be used in fighting games for personal computer. The general aim of the project was to produce a low cost controller, with acceptable quality for competition and possibility of future expansion of its design, compatibility and functionalities. The controller follows the USB HID specifications and uses a microcontroller to execute its logic. As a result, an efficient equipment is presented, in which a specialized board was replaced by a more customizable and low-cost option.*

Resumo. *Este artigo descreve o processo de desenvolvimento de um controle de videogame no estilo arcade, com conexão USB, adequado para jogos de luta para computador pessoal. O objetivo geral do projeto foi produzir um controle de baixo custo, com qualidade aceitável para competição e possibilidade de futura expansão de seu design, compatibilidade e funcionalidades. O controle segue as especificacões USB HID e utiliza um microcontrolador para executar sua lógica. Como resultado, apresenta-se um equipamento eficiente, no qual foi substituída uma placa especializada por uma opção mais customizável e de baixo custo.*

1. Introdução

A escolha de um bom controle é essencial para os jogadores entusiastas de jogos de *videogames* competitivos do gênero 'luta'. Para tanto, os comandos pressionados devem ser precisos em tempo e em execução, mas o controle também deve ser confortável, já que será usado por horas seguidas de treino ou de jogo.

Os controles no estilo *arcade* são uma escolha popular entre esses jogadores, pois historicamente o gênero 'luta' foi desenvolvido em gabinetes de *arcade*. Em versões de uso doméstico, esses controles são muito confortáveis. O tamanho e o espaçamento dos botões favorecem o pressionamento com vários dedos, e não apenas com os polegares (como no caso dos modelos *joypad*), o que permite rapidez e precisão nos comandos e facilita o acionamento simultâneo de vários botões. Além disso, adequam-se a diferentes tamanhos de mãos e dedos. Já o seu direcional *joystick* facilita anatomicamente vários tipos de comandos bastante usados em 'jogos de luta'. Esses controles são produzidos em diversos *layouts*, podendo a escolha ser orientada por critérios individuais.

Entretanto, exemplares de alta qualidade desses controles são muito caros. Os preços variam de acordo com marca, escolha de padrão de peças e qualidade do material; mas um controle que se encaixe nos critérios de uso 'viável para competição' custa no

mínimo trezentos e cinquenta reais, sendo que alguns ultrapassam o valor de um mil reais (pesquisa de preço feita pelos autores em sites de compras *online* em dezembro de 2019).

Portanto, esse projeto teve como objetivo geral produzir um modelo de controle de *videogame* no estilo *arcade* com qualidade aceitável para competição, preço acessível e possibilidade de futura expansão de seu *design*, compatibilidade e funcionalidades. Nesse sentido, as perguntas que norteiam a pesquisa envolvem os procedimentos de construção desse controle, as ferramentas e os componentes necessários para sua construção, além dos valores para a aquisição desses componentes.

Esse projeto foi desenvolvido na disciplina de Microcontroladores do curso de Ciências da Computação, durante o semestre letivo de 2019.2, no IFCE Maracanaú. Neste artigo, relata-se o processo de desenvolvimento do controle em cinco seções, incluindo esta introdutória. A segunda apresenta Pressupostos Teóricos; a terceira descreve os Materiais e Métodos; a quarta, os Resultados e Conclusão e a quinta, as Considerações Finais.

2. Pressupostos Teóricos

A nível de dispositivos físicos, o termo *HID* (*Human Interface Device*) referencia um instrumento eletrônico periférico cujo propósito é possibilitar uma interação de *Input/Output* entre um usuário humano e um computador. Também se refere a um conjunto de especificações técnicas a respeito desses dispositivos, das quais pode-se destacar as que dizem respeito a dispositivos *USB HID*, ou seja, os que transmitem dados via porta *USB* (*Universal Serial Bus*). Essas especificações são utilizadas em *API*s (*Application Programming Interface*) e *drivers* para aplicações diversas, como o emulador de *mouse* controlado por língua, com foco em acessibilidade [Horne et al. 2013]. Tais especificações constam no *website* da organização *USB Implementers' Forum* [USB 2001].

Um microcontrolador (doravante *MCU*) é um sistema de *hardware* formado por um processador, uma memória e uma variedade de outros componentes, sendo todos integrados em um *chip*. Dessa forma, é possível reduzir o tamanho da placa de circuito impresso usada, se comparado ao de uma placa com funcionalidades semelhantes, mas construída com componentes dedicados. Assim, pode-se economizar tempo, espaço e dinheiro [Gridling and Weiss 2007]. Aplicações diversas são possíveis com o uso de um *MCU*, como o sensor lingual conectado a um *Atmel ATSAM3XBE* [Horne et al. 2013] e um pequeno veículo autônomo, para o qual foi utilizado um *PIC16F877* [Drugărin et al. 2014].

Quanto à linguagem, para o desenvolvimento do sensor lingual, foi utilizado *Assembly*, enquanto a lógica do pequeno veículo foi desenvolvida em C. Os autores [Drugărin et al. 2014] defendem que a linguagem C é mais rápida e fácil de escrever e, portanto, adequada a projetos como o ora relatado. Como *IDE* (*Integrated Development Environment*), eles usaram o MPLAB da Microchip, o que também foi feito no projeto a que se refere este artigo.

Para a gravação do código para o *MCU*, há diversas ferramentas possíveis, mas neste projeto, foi feito uso da ferramenta didática *SanUSB* [Jucá et al. 2009]. Esta é composta por duas partes: o *software* de gravação e um circuito básico de um *MCU PIC18F2550* com conexão *USB*, que pode ser montado seguindo o modelo dos autores. Sua vantagem é a simplicidade de uso e o baixo custo comparado com outras opções usando equipamentos profissionais.

Os dois projetos descritos a seguir tratam do desenvolvimento de controles de *videogame*. [Goya et al. 2012] apresentam um projeto de um controlador de jogos eletrônicos focado em princípios ergonômicos, que seja viável em termos tecnológicos e de custos e, ainda, acessível a pessoas com necessidades especiais. Eles enfatizam também os problemas que acometem os usuários de *videogames* devido ao longo tempo de utilização e a posturas inadequadas, à necessidade de ajuste dos botões dos controles e a outros fatores. Já [Gomes et al. 2015] tratam da construção de um volante para jogos eletrônicos de corrida e simulação de veículos (realizado com alunos de ensino fundamental e médio do IFPI), visando evidenciar como a tarefa de desenvolvimento de controladores de *games* possibilita o ensino/aprendizagem de conteúdos diversos.

[Goya et al. 2012] compartilham nossa preocupação com a construção de um controle que proporcione 'conforto', o que tem relação com aspectos ergonômicos como tamanho dos botões e espaçamento que se adeque ao tamanho das mãos do jogador (conforme o tipo descrito na Seção 1). O projeto do protótipo desenvolvido por [Gomes et al. 2015], embora tendo objetivo distinto do projeto do controle descrito neste artigo, também explora a construção de controladores de jogos, assemelhando-se nos componentes usados e no enfrentamento de dificuldades nas configurações desses artefatos (conforme relato na Subseção 3.4).

3. Materiais e Métodos

A seguir, os procedimentos de desenvolvimento do controle serão apresentados em quatro subseções. A primeira discorre sobre as Ferramentas do *Hardware*; a segunda, sobre os *Softwares* Usados; a terceira, sobre as Configurações e a quarta, sobre a Montagem.

3.1 Ferramentas do *Hardware*

A seleção dos materiais para montagem do circuito elétrico foi feita tendo como guia o uso do microcontrolador *PIC18F2550* (em detrimento de outros também populares como o *Arduino*), visto que ele tem funcionalidades suficientes para o projeto e baixo custo. Foi escolhida uma placa com componentes periféricos e porta *USB* organizados de forma adequada à utilização dos recursos desse microcontrolador. Para expansão externa do circuito, foi usada uma *protoboard* para testes e, para o circuito final, uma placa fenolite perfurada. Por fim, utilizou-se componentes diversos (fios, resistores, potenciômetros e diodos).

Os botões e a alavanca são réplicas do modelo da marca japonesa *Sanwa*. Devido à premissa de utilizar um microcontrolador de propósito geral, não foi utilizada a placa do chamado *kit* de controle *arcade*. Para a caixa externa, foram usadas chapas de madeira.

3.2 *Softwares* Usados

Para a programação interna do controle, após testes com diversas *firmwares*, foi escolhida a *Joystic nUSB*, que utiliza especificacões *HID USB*. Nesta, ocorre processamento de uma multiplexação de botões por matriz de *inputs*. Isso se deve ao fato de que a *firmware* em questão tem suporte para mais botões e eixos do que seria possível por meio de associação direta de cada botão e comando direcional com os contatos livres da placa microcontroladora, a qual dispõe de apenas 28 pinos. As alterações no código foram realizadas usando a *IDE MPLAB* com compilador C18 e, logo após, o código foi gravado no *PIC* usando a ferramenta *SanUSB*. Nos testes, foi utilizado o *driver* padrão do Windows 10 para controladores de jogos.

3.3 Configurações

Um controle de *arcade* típico possui um único direcional de 2 eixos e de 5 a 10 botões de função, e a *firmware* utilizada tem funcionalidades excessivas para o propósito do projeto. Dessa forma, foi analisada a possibilidade de fazer uso apenas de parte das funções disponíveis, sem que estas sofressem interferência dos outros recursos não utilizados. Para tal, foram apagados trechos de código referentes a esses recursos. A *firmware* original tratava os 24 botões suportados como uma matriz de *inputs* 3x8. Foi mantida a lógica de matriz, mas não os códigos relativos a recebimento de *hid_report*'s. Funções ou definições a respeito da terceira linha da matriz foram apagadas, tornando a matriz original em uma de dimensão 2x8. Da mesma forma, foram eliminados trechos referentes a três dos cinco eixos, além de serem modificadas definições de *driver* sobre estes no arquivo *usb_descriptors.c*. Por fim, foi modificado o valor hexadecimal de *USAGE_MAXIMUM*, nesse mesmo arquivo, referente à quantidade de botões a ser interpretada pelo *driver*, eliminando seis dos oito botões da segunda linha da matriz de *inputs*. Isso efetivamente reduziu a quantidade de botões do *joystick* para 10. Por conseguinte, foram utilizados 8 botões de funções principais associados às 8 colunas da Linha 1 da matriz e 2 botões de funcões secundárias (*Start* + *Select* ou *Start* + *Coin*) associados às duas únicas colunas reconhecidas da Linha 2, e um único direcional de dois eixos.

Figura 1. Diagrama de botões e eixos utilizados e botões desabilitados

A Figura 1 ilustra a definição final dos eixos analógicos e da matriz de botões. Apenas os elementos marcados como "Habilitado" existem e são exibidos pelo *driver* do *Windows*. Os "Desabilitados" estavam presentes apenas na versão original da *firmware*.

3.4 Montagem

Nessa etapa, a maior dificuldade veio com a descoberta de que a *firmware* utilizava apenas eixos analógicos, tendo em vista que um controle de *arcade* é quase sempre totalmente digital. A própria implementação física da alavanca direcional permite apenas comandos digitais, sendo essa, a nível de circuito, equivalente a quatro botões (um para cada uma das quatro direções: cima, baixo, direita e esquerda). Já as direções intermediárias, como diagonal cima + esquerda, são usadas com acionamento simultâneo de 2 direções perpendiculares. Por outro lado, o acionamento de cima + baixo ou esquerda + direita são fisicamente impossíveis com o uso da alavanca; então, não foram tratados neste projeto.

Já um direcional analógico permitiria comandos mais continuamente descritos, como 30% do alcance total de "cima", somado a 15% do alcance total de "direita". Num jogo com suporte a esse direcional, isso permite ações como fazer um personagem se mover vagarosamente ou rapidamente (dependendo de quanto do alcance total do direcional está sendo acionado) ou se mover para direções ainda mais intermediárias às oito citadas.

Foi decidido adaptar fisicamente o circuito para interpretar os *inputs* dos botões da alavanca como sendo 4 direções de intensidade máxima numa alavanca analógica de 2 eixos. Dessa forma, foi possível a eliminação dos *inputs* intermediários do *joystick*.

A forma como o *driver* de *joystick* interpreta um espaço analógico é por meio de duas entradas (uma para cada eixo). O valor que está sendo acionado para cada um desses eixos depende da intensidade de corrente elétrica transmitida para o pino correspondente. Seria intuitivo pensar que uma intensidade nula de corrente passando por um pino receptor do direcional analógico é interpretada como zero numa direção desse eixo, mas como para isso seriam necessários 4 pinos direcionais, essa não é a implementação verdadeira. Ao invés disso, uma corrente nula transmitida para um pino é interpretada como acionamento total para uma direção desse eixo e corrente máxima como acionamento total na direção contrária. Qualquer corrente intermediária equivale a um acionamento parcial em um dos dois sentidos desse eixo. Mas, numa adaptação para direcional digital, a possibilidade deste acionamento deve ser eliminada, sendo a única exceção o valor intermediário que equivale à metade da corrente total, correspondendo a intensidade zero nesse eixo (que neste caso, estaria na posição central/neutra).

Essa discrepância foi resolvida revisando o planejamento do circuito físico, adaptando-o por meio de resistores. Como a posição neutra do direcional implica em passagem de alguma corrente elétrica não nula para os pinos dos eixos, não faria sentido ligar esses pinos numa porta *ground* (*GND*). Ao invés disso, foi definido que, quando os pinos estivessem ligados no *GND*, seriam acionadas as direções extremas do direcional equivalentes a 100% esquerda e 100% cima. Logo, acionar a alavanca para a "esquerda" ou para "cima" fará um curto circuito entre o *GND* e o pino correspondente do eixo.

Já os valores da corrente do acionamento neutro do direcional deveriam ser equivalentes à metade do valor da corrente total de alimentação. Mas a corrente de alimentação era excessivamente grande para ser interpretada pelo *driver* como o valor correto. Para encontrar o valor exato da "meia corrente" aceito pelo *driver*, foi feita uma conexão entre o pino do eixo e o de alimentação intermediada por um potenciômetro, o qual foi ajustado manualmente até mostrar a posição de eixo como centralizada nas configurações do *driver*. Como também relatado em [Gomes et al. 2015], oscilações mínimas ainda ocorrem nessa posição, mas neste projeto não houve necessidade de tratá-las, visto que são pequenas demais para passar da região de tolerância da *deadzone* (área que o *driver* interpreta como região neutra do direcional).

Para os acionamentos de "direita" e "baixo", foi testada a ligação dos pinos de eixos diretamente na corrente de alimentação, mas o valor excessivo de um eixo estava sendo somado ao outro. Por exemplo: dado um *input* para baixo, o *driver* acusava que estava sendo pressionado baixo + direita, ou seja, houve a soma do excedente do valor do *input* para baixo com o valor do eixo horizontal. Para contornar esse problema, foi escolhido um par de resistores com a metade da resistência medida no potenciômetro.

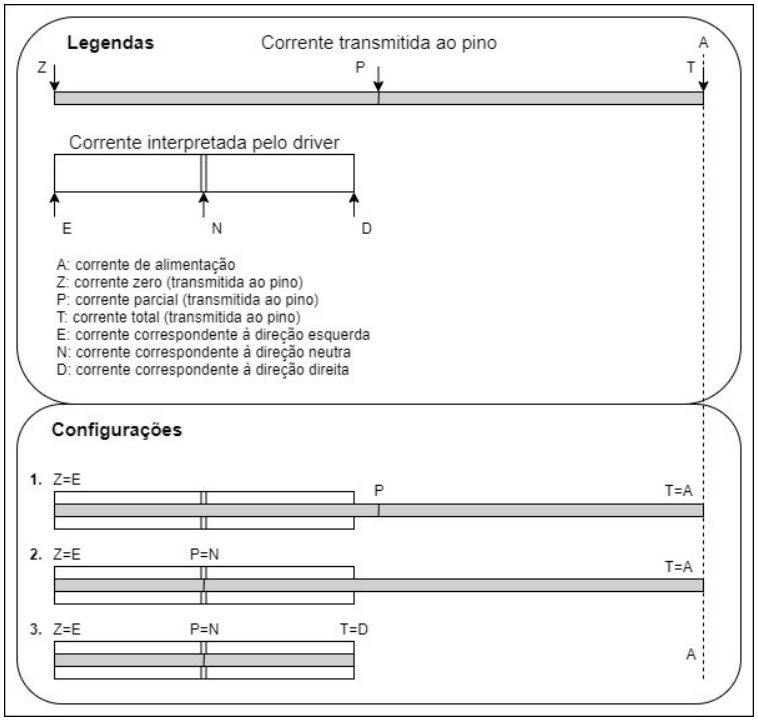

Figura 2. Diagrama da calibragem de um eixo analógico

A Figura 2 ilustra (sem obedecer uma escala) o processo de 'moldar' a corrente elétrica de acordo com as intensidades aceitas pela *firmware* e pelo *driver*. Foi usado como exemplo o eixo horizontal. O espaço de '**Legendas**' mostra individualmente cada elemento, a faixa 'Corrente transmitida ao pino' mostra os níveis de corrente elétrica que o pino do eixo está recebendo de acordo com a configuração de cada etapa, e o retângulo 'Corrente interpretada pelo *driver*' mostra, evidentemente, os níveis de corrente interpretados pelo *driver*. A linha pontilhada 'A' representa a intensidade máxima de corrente possível neste sistema: a corrente de alimentação. Os níveis 'Z', 'P' e 'T' representam, respectivamente, a intensidade zero, parcial e total que está sendo enviada para o pino. O retângulo 'Corrente interpretada pelo *driver*' mostra os níveis 'E', 'N' e 'D', que representam os níveis de corrente que o *driver* interpreta como direções esquerda, neutra e direita, respectivamente.

O espaço das '**Configurações**' representa três diferentes etapas no processo de calibragem das correntes transmitidas ao pino do eixo. Configuração 1: nesta etapa, ainda não havia sido feito redução de corrente, ou sequer comparação entre as correntes transmitidas ao pino e as correntes aceitas pelo *driver*. Inicialmente, o nível 'P' foi definido como metade de 'A', visto que esse nível foi planejado para ser interpretado como 'meia corrente' e, portanto, posição neutra no eixo. Porém, o valor de 'P' já era maior do que o valor máximo interpretado pelo *driver*. Daí a necessidade de uma nova configuração. Configuração 2: para solucionar o problema anterior, foi feito uso de um potenciômetro, por meio do qual foi reduzida a corrente parcial até que 'P' fosse igual a 'N'. Porém, devido ao excedente de 'T', ocorreu o 'vazamento de corrente' para o outro eixo. Configuração 3: foi escolhido um resistor com metade da resistência calibrada no potenciômetro para permitir a passagem do dobro da corrente de 'P' (que deve ser o valor exato da corrente de 'D'). Na 'Configuração 3', as correntes zero (nula), parcial e total transmitidas para o pino coincidem com os valores que o *driver* considera como 'esquerda', 'neutra' e 'direita', respectivamente. A Figura 3 ilustra essa configuração.

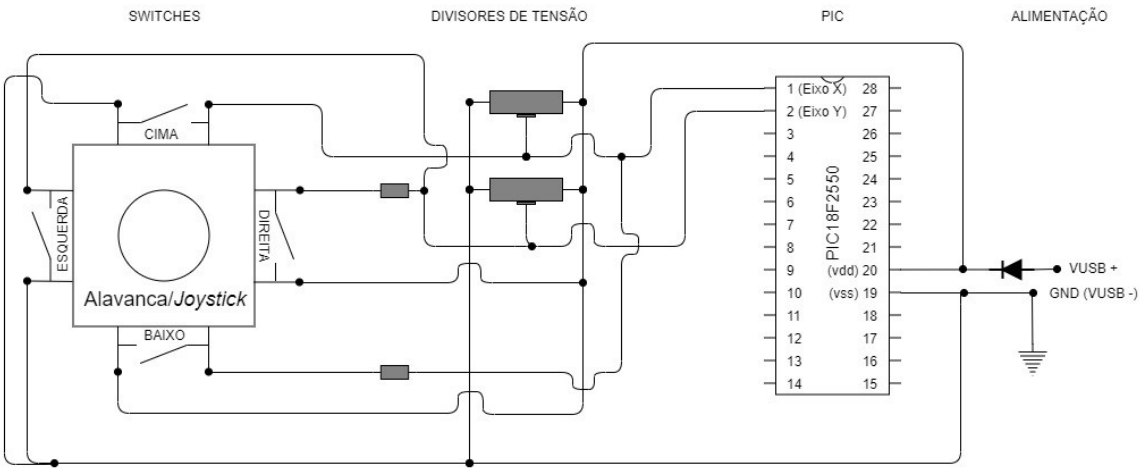

Figura 3. Diagrama do circuito do direcional em sua configuracão final

4. Resultados e Conclusão

As ferramentas virtuais de desenvolvimento utilizadas foram adequadas para esse projeto. O uso de uma *firmware* desenvolvida em linguagem C tornou o código razoavelmente legível e, portanto, mais fácil de modificar do que um código em *Assembly*. A escolha da *IDE* MPLAB e do compilador C18 foi uma consequência natural dessa decisão. A *firmware* escolhida foi importante por sua implementação que permitiu uma quantidade de comandos que supriu as necessidades do projeto.

No entanto, ao longo do processo de produção e testes do controle, ocorreram alguns problemas, que exigiram novas implementações. A interferência de eixos direcionais não utilizados pôde ser solucionada com a análise e modificação/eliminação de trechos de código que sequer viriam a ser aproveitados nas funcionalidades almejadas. Além disso, foram feitas sucessivas modificações na conexão da alavanca, a fim de transformar os dois eixos analógicos restantes em eixos digitais e alcançar apenas as correntes elétricas interpretadas pelo *driver* de controle. Assim, foi obtido um direcional que simula eficientemente a experiência de *arcades*, com *inputs* simples para 8 direções.

Ao final do processo, o controle apresentou um bom desempenho, mostrando facilidade em executar comandos complexos (conforme teste prático em laboratório). Devido à flexibilidade de *design* do espaçamento e da organização dos botões, o controle pode ser ergonomicamente adaptado para qualquer jogador.

O custo da construção do controle totalizou cento e setenta e oito reais (placa de fenolite, circuito impresso e componentes: R$20,00; *PIC18F2550*: R$30,00; fios: R$3,00; 10 botões de *arcade*: R$50,00; alavanca *joystick* completa: R$40,00; caixa de madeira: R$35,00). A Figura 4 mostra o controle pronto e em teste.

Figura 4. Foto do controle construído

5. Considerações Finais

A experimentação resultou num produto final que atigiu o objetivo almejado - um controle com qualidade satisfatória, baixo custo e possibilidade de expansão. A escolha de uma placa com um *MCU* de propósito geral e conector *USB* possibilitou não apenas a customização do *design* do controle, mas de suas funcionalidades e compatibilidade. Entretanto, por falta de recursos em laboratório, não foi possível testar precisamente o tempo de resposta de comandos.

O projeto nos proporcionou conhecimentos e habilidades de solução de problemas. Essa experiência poderá levar a projetos futuros com novas funcionalidades ou, ainda, ao desenvol-vimento de uma linha voltada para acessibilidade. Pode-se também desenvolver implementações em diferentes *API*'s (*XInput* ou *DirectInput*) para construir controles compatíveis com outros dispositivos, como *Android*, sistemas *RetroArch* embarcados em *RaspBerry Pi* e outros.

Referências

Drugărin C. V. A. et al. (2014). Method for programming an autonomous vehicle using pic 16f877 microcontroller. *Proceedings in Conference of ICTIC 2014*.

Gomes, F. R. et al. (2015). A construção de um volante para *racing games* aplicado no ensino de automação. In: *Mostra Nacional de Robótica (MNR)*, pages 18–22.

Goya, J. Y. L. et al. (2012). Criação e desenvolvimento de um controlador de jogos eletrônicos: um projeto inclusivo. In: *Proceedings of SBGames*, pages 247–250.

Gridling, G. and Weiss, B. (2007). Introduction to Microcontrollers (Courses 182.064 & 182.074). *Vienna University of Technology, Institute of Computer Engineering, Embedded Computing Systems Group.*

Horne, R. et al. (2013). A framework for mouse emulation that uses a minimally invasive tongue palate control device utilizing resistopalatography. In: *Kent Academic Repository*.

Jucá, S. C. S. et al. (2009). SanUSB: software educacional para o ensino da tecnologia de microcontroladores. In: *Ciências & Cognição*, volume 14 (3), pages 134–144.

USB Implementers' Forum (2001). Device class definition for human interface devices (hid). https://usb.org.10-1-108-210.causewaynow.com/sites/default/files/documents/hid1_11.pdf. Acesso: 05 jan. 2020.