Desenvolvimento de uma interface multi-modal para o controlo de um eixo linear

Filipe António Coelho Alturas

Dissertação de Mestrado

Orientador: Professor Doutor Joaquim Gabriel Magalhães Mendes

Mestrado Integrado em Engenharia Mecânica

Especialização em Automação

Março de 2021

Aos meus pais e irmão,

Resumo

Com a evolução que se tem vindo a verificar nos últimos anos, o mundo atingiu um ritmo tão grande que todos os dias surgem novas tecnologias, sendo que muitas delas permitem melhorar e facilitar tarefas simples do dia a dia. Um exemplo disso, são as interfaces multi-modais que permitem realizar o controlo de um sistema através de diferentes modos. Com o aparecimento desta tecnologia, uma simples tarefa como por exemplo enviar uma mensagem, pode agora ser feita sem ter que se utilizar o ecrã do telemóvel para a escrever, podendo o utilizador ditar a mensagem em vez de a escrever.

O principal objetivo deste trabalho é controlar o movimento linear usando 3 tipos de interface (interface multi-modal): uma desenvolvida através do programa LabVIEW®, outra através de uma câmara OpenMV® Cam M7, e por fim uma que permita controlar o sistema por voz.

O sistema desenvolvido tem como principal componente um motor de passo NMB PM35S-048 com o respetivo eixo linear e sistema de transmissão. É ainda de salientar, que todo este sistema foi reaproveitado de uma impressora, o que tornou relativamente simples encontrar a solução mecânica.
Para *driver* do motor, optou-se por uma ponte H L298N, e para controlador do sistema, optou-se por um microcontrolador Arduino® DUE.
Realizou-se ainda o desenvolvimento das diversas interfaces do sistema, tendo-se desenvolvido 3 interfaces distintas, para estabelecer a comunicação entre o sistema e o utilizador. Assim, a primeira interface desenvolvida foi através do programa LabVIEW®, onde é possível escolher o sentido, a velocidade e a distância de deslocação da porta. A segunda interface desenvolvida foi através de uma câmara OpenMV® Cam M7, que possibilita realizar o controlo do sistema utilizando uma imagem, sendo que em função do sentido e da inclinação da imagem, é realizado o movimento de deslocação no eixo linear. Por fim, a última interface desenvolvida foi o controlo do sistema por voz. Para tal, utilizou-se um módulo de reconhecimento voz EasyVR Shield 3.0 que permite enviar ordens para o sistema, através do microfone, mas também receber um *feedback* do mesmo através da coluna.

Depois de se chegar à solução final, tudo estava a funcionar corretamente e de acordo com o que tinha sido idealizado, sendo possível fazer o controlo do eixo linear por qualquer uma das interfaces.

Development of a multi-modal interface for the control of a linear axis

Abstract

With the evolution that has been verified in recent years, the world has reached such a rhythm that new technologies emerge every day, many of which allow to improve and facilitate simple daily tasks. An example of this, are the multi-modal interfaces that allow to carry out the control of a system through different modes. With the appearance of this technology, a simple task such as sending a message, can now be done without having to use the mobile phone screen to write it, and the user can dictate the message instead of writing it.

The main objective of this work is to control the linear movement using 3 types of interface (multi-modal interface): one developed through the LabVIEW® program, another through an OpenMV® Cam M7 camera, and finally one that allows controlling the system by voice.

The developed system has as main component an NMB PM35S-048 stepper motor with its linear axis and transmission system. It should also be noted that this entire system was reused from a printer, which made it relatively simple to find the mechanical solution.
For the motor driver, an H L298N bridge was chosen, and for the system controller, an Arduino® DUE microcontroller was chosen.
The development of the different interfaces of the system was also carried out, having developed 3 distinct interfaces, to establish the communication between the system and the user. Thus, the first interface developed was through the LabVIEW® program, where it is possible to choose the direction, speed, and distance of travel of the door. The second interface developed was through an OpenMV® Cam M7 camera, which makes it possible to control the system using an image and depending on the direction and inclination of the image, the displacement movement is carried out on the linear axis. Finally, the last interface developed was the control of the system by voice. For this purpose, an EasyVR Shield 3.0 voice recognition module was used, which allows orders to be sent to the system through the microphone, but also to receive feedback from the microphone through the column.

After reaching the final solution, everything was working correctly and according to what had been designed, being possible to control the linear axis through any of the interfaces.

Agradecimentos

Quero deixar aqui o meu sincero agradecimento a todos que das mais diversas e variadas formas me apoiaram e ajudaram tonando possível a realização deste trabalho, nomeadamente:

Ao meu orientador, Professor Joaquim Gabriel, por toda a dedicação, paciência e conhecimento transmitido ao longo deste projeto.

Ao Engenheiro Jorge Reis, por me ter fornecido o motor e sistema de transmissão utilizado no sistema desenvolvido.

Ao Técnico Superior do Laboratório L-103, Joaquim Silva, pela ajuda prestada ao longo de todo o projeto, nomeadamente por me fornecer o material e me ter ajudado a montar todo o sistema projetado e desenvolvido.

Aos meus pais e irmão, por todo o apoio e carinho transmitido, e pelas condições que me deram durante todo o meu percurso académico.

À minha família pela ajuda, motivação e preocupação que sempre demonstraram.

Por fim, aos meus amigos por toda a ajuda e motivação prestada e pelos bons momentos que partilhámos.

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

Acrónimos

FEUP	Faculdade de Engenharia da Universidade do Porto
MIEM	Mestrado Integrado em Engenharia Mecânica
INEGI	Instituto de Ciência e Inovação em Engenharia Mecânica e Engenharia Industrial
MBUX	*Mercedes-Benz User Experience*
HMI	*Human-Machine Interface*
PLC	*Programmable Logic Controller*
SEL	Sistemas Eletromecânicos

1 Introdução

No capítulo inicial desta dissertação, apresenta-se o tema do trabalho realizado – Desenvolvimento de uma interface multi-modal para o controlo de um eixo linear – realizando o seu enquadramento, e apresentando os objetivos específicos que se pretendem alcançar. Além disso, falasse também o método seguido para realizar este projeto, e por fim, este capítulo termina com um breve resumo da estrutura da dissertação e dos assuntos que serão abordados nos capítulos seguintes.

1.1 Enquadramento do projeto e motivação

Nos últimos anos, a evolução tecnológica que se tem verificado tem sido tão grande que o aparecimento diário e constante de novas tecnologias já se tornou habitual, surgindo de forma bastante natural. Um exemplo dessa evolução são os sistemas com inteligência artificial que têm sido desenvolvidos por alguns fabricantes de automóveis. Alguns desses sistemas são capazes de reconhecer comandos de voz, gestos e até mesmo olhares, permitindo usufruir de uma operação multimodal que torna a interação entre o condutor e o seu veículo ainda mais intuitiva. Outro bom exemplo é a evolução que se tem verificado nos equipamentos domésticos, tal como nos aspiradores robots que têm vindo a surgir no mercado. Alguns deles são capazes de receber ordens do utilizador por voz, ou através de uma simples aplicação que permite colocar o aspirador em funcionamento à distância.
Neste projeto, vão desenvolver-se diferentes interfaces que permitam controlar um eixo linear através de uma interface multi-modal, isto é, interfaces com I/O múltiplos [1].

1.2 Objetivos do projeto

O tema da dissertação é o estudo e desenvolvimento de uma interface multi-modal para o controlo de um eixo linear. Desta forma, o que se pretende estudar e desenvolver, é: primeiramente o sistema mecânico (sistema e estrutura que irá suportar/fixar todos os componentes necessários), de seguida o sistema eletromecânico, isto é, qual o motor elétrico, *driver* e controlador a utilizar para se realizar o movimento linear ao longo do eixo, e por fim desenvolver 3 interfaces que permitam realizar o controlo de todo o sistema.
A primeira interface que se irá desenvolver é controlar o sistema através do programa LabVIEW®, onde através de uma HMI criada nesse mesmo programa o utilizador indique o sentido, a velocidade e a distância de deslocamento. Seguidamente irá desenvolver-se uma interface que permita controlar o sistema através de uma câmara, isto é, usando uma imagem de referência e em função não só do sentido como também dos graus de inclinação da mesma, irá realizar-se o movimento numa ou noutra direção (em função do sentido de inclinação) com maior ou menor velocidade (em função dos graus de inclinação). Por fim, irá desenvolver-se uma interface que permita controlar o sistema através de um módulo de reconhecimento de voz,

sendo que para tal, o controlo de movimento no eixo linear será realizado em função da ordem que for dada pelo utilizador através da sua voz.

1.3 Estrutura da dissertação

A dissertação está estruturada em 6 capítulos, seguidos das referências bibliográficas e dos anexos.
No primeiro capítulo, Introdução, faz-se uma introdução ao tema desta dissertação, apresentando o seu enquadramento e motivação e definindo os objetivos a atingir. No segundo capítulo, Estado da Arte, apresentam-se alguns projetos/soluções que já existem no mercado que são exemplos de sistemas controlados por uma interface multi-modal. No terceiro capítulo, Apresentação do problema, efetua-se uma breve, mas detalhada apresentação do problema apresentado. No quarto capítulo, Apresentação da solução, descreve-se a solução desenvolvida para o problema apresentado. No quinto capítulo, são ainda realizados os testes necessários para validação da solução, de forma a garantir que esta é válida e satisfaz todas os requesitos que foram inicialmente impostas. No sexto e último capítulo, Conclusões e perspetivas de trabalho futuro, apresentam-se as conclusões a retirar da elaboração da presente dissertação, fazendo-se ainda referência a possíveis trabalhos a desenvolver futuramente.

2 Estado da Arte

Neste capítulo apresenta-se a pesquisa bibliográfica elaborada sobre sistemas em que se utilizam interfaces multi-modal. Para tal, vão apresentar-se alguns sistemas já desenvolvidos que são controlados por uma ou mais interfaces, de forma a garantir a conexão entre o utilizador e o sistema a controlar.
Uma interface multi-modal é aquela em que há mais de uma opção de entrada e saída de dados (I/O) e a forma de interação pode ser alterada conforme for mais adequado ao contexto do problema [1].
De seguida, irão ser apresentados 4 sistemas: um em que o controlo do sistema é realizado pela interação do utilizador através de uma interface de voz e de uma interface com um *touchscreen* (MBUX®), outro em que é realizado através de uma interface por voz (Siri®), e por fim dois sistemas em que o seu controlo é realizado através de uma interface que utiliza várias câmaras e sensores para conseguir interpretar os gestos e olhares do utilizador e realizar o controlo em função disso (BMW Natural Interaction® e Análise de aspetos de usabilidade em interações naturais via interfaces multimodais).

2.1 Mercedes Benz User Experience ®

Apresentado em 2018, o sistema MBUX®, cujo acrónimo significa "*Mercedes-Benz User Experience*" é o sistema multimédia de inteligência artificial que equipa os automóveis Mercedes-Benz.
Este sistema adapta-se às preferências de todos os passageiros (e não apenas do condutor), e pode ser comandado por voz, com gestos, ou através do toque. Este sistema digital tem a capacidade de se adaptar constantemente às preferências e necessidades de todos os passageiros porque o veículo está equipado com o ecrã tátil do cockpit e com outros dois ecrãs táteis instalados na parte de trás dos bancos dianteiros Figura 2.1 [2].

Figura 2.1 – Ecrãs táteis instalados no veículo para interação com o passageiro [4]

O MBUX adapta-se constantemente ao recolher e tratar os dados de utilização do veículo: o sistema regista os trajetos mais percorridos, os contactos mais utilizados pelo condutor; as músicas ou estações de rádio mais ouvidas; entre outros [2]. Estes registos permitem que o sistema esteja preparado para responder aquilo que são os hábitos do condutor. Por exemplo: qualquer pessoa que, frequentemente, telefone à sua mãe às terças-feiras, durante a viagem para casa, receberá o seu número de telefone como sugestão no visor neste dia da semana, ou qualquer pessoa que, regularmente, mude para uma estação de rádio com notícias, em determinado horário, também receberá a sugestão de o fazer, e ainda se o sistema de navegação detetar uma rota com frequência, a navegação para este destino estará preparada para se iniciar, com avisos de trânsito no trajeto definido podendo alterar a rota caso detete constrangimentos no tráfego [3].
Para além disto, o sistema é também altamente interativo e permite controlar diversos equipamentos do automóvel através da voz, do toque, ou simplesmente através da realização de determinado gesto.

Figura 2.2 – Interação do condutor com o veículo através do ecrã tátil [3]

O sistema de comando de voz LINGUATRONIC® permite que interaja com o veículo simplesmente através da voz, estando este preparado para reconhecer 27 idiomas e para responder a questões de conhecimento geral ao estilo da Siri®, Google Assistant® ou Alexa® [2]. A assistência linguística inteligente é ativada através de um botão no volante ou com a frase-chave "*Hey Mercedes*" [3].
Não é o humano que se tem que adaptar à máquina, mas sim, neste caso, é a máquina que se tem que adaptar às ordens, pedidos e solicitações do utilizador. Com este sistema o utilizador pode pedir para mudar de estação de rádio, ditar uma mensagem, realizar uma chamada, pedir para que seja definido determinado trajeto, ligar o ar condicionado, alterar a posição do banco, entre muitas outras ações, tudo isto sem ser necessário utilizar comandos de voz predefinidos.
O discurso indireto e inteligente também é reconhecido. Por exemplo, o utilizador pode dizer simplesmente "estou com frio" em vez de dar ordem de voz "temperatura a 24 graus", para que o controlo climático seja ativado [3].
Se o utilizador preferir, pode ainda alterar alguns parâmetros relacionadas com a condução do automóvel, como por exemplo a escolha do modo de condução, através do toque. Para tal, basta utilizar o ecrã tátil da consola central, que por exemplo, reconhece a caligrafia do utilizador [2].
Por último, pode ainda realizar alguns gestos que serão reconhecidos pelo automóvel como comandos para determinadas ações. Uma dessas ações é, por exemplo, abrir o teto panorâmico com apenas um gesto, que se torna bastante mais divertido e prático do que utilizar botões [2].
Além de permitirem um maior conforto e comodidade ao utilizador, e valorizar o próprio veículo, este sistema oferece também maior segurança aos utilizadores: Através de sensores, o

veículo consegue prever quando um ocupante está prestes a sair do carro, e se as câmaras do veículo detetarem que se está a aproximar um carro, a iluminação ambiente piscará em vermelho para alertar o utilizador que não é seguro sair do caro naquele momento. O sistema MBUX também consegue determinar se uma cadeirinha infantil está corretamente presa ao banco em que está montada, ou se o motorista está como sinais de sonolência. Nesse caso, o MBUX avisará o motorista e, se este afirmar "estou cansado", o MBUX ativará "o *ENERGIZING COACH*" que é um modo que ajusta as condições de condução para ajudar o condutor a conduzir em segurança [4].

2.2 Siri ®

A Siri® é uma assistente virtual que atualmente incorpora todos os equipamentos desenvolvidos pela Apple®. A aplicação é capaz de compreender comandos de voz e realizar tarefas relacionadas com o sistema operativo. Uma das principais ideias desta aplicação é servir como uma amiga que responde a perguntas, conta piadas e ajuda a automatizar algumas tarefas do quotidiano, criando lembretes, ligando, ou enviando mensagens para os contatos sem que o utilizador precise de mexer ou abrir as apps no telemóvel [5].
Apesar de atualmente a Siri® pertencer à Apple®, ela não foi originalmente desenvolvida pela empresa. A assistente virtual foi um projeto fundado por Dag Kittlaus, Cheyer Adam, Tom Gruber e Norman Winarsky, e desenvolvido em 2007 pela SRI International, um instituto de pesquisa científica sem fins lucrativos [5].
A Siri® baseia-se na Inteligência Artificial e Processamento de Linguagem Natural e é composto por três componentes: uma interface conversacional, o contexto pessoal e a delegação de serviços [7].
A interface conversacional está relacionada, em primeiro lugar, sobre como a Siri® entende o que o utilizador diz. O bom funcionamento geral do reconhecimento de voz direto, isto é palavra por palavra, deve ser garantido para que a Siri® possa ouvir de forma clara o que o utilizador está a dizer [7].
Para decifrar o significado da mensagem enviada pelo utilizador, ao receber as solicitações de voz através da interface, a Siri® traduz para código. Para tal, usa o contexto pessoal, que permite identificar os padrões, frases, palavras-chave e examinar milhares de combinações para interpretar o seu significado [7].
Depois de receber e decifrar a mensagem enviada pelo utilizador, a Siri® começa a avaliar e delegar as tarefas que podem ser realizadas (delegação de serviços), usando os bancos de dados do telefone e os servidores online. Assim, ela consegue criar respostas completas e coesas para atender às suas perguntas que lhe foram colocadas [5].

Figura 2.3 – Utilização da Siri® num iPhone [6]

Para iniciar a interação com a Siri® o utilizador deve dizer apenas "*Hey Siri*" para o dispositivo Apple que estiver a utilizar e a partir daí começa então a interação entre o sistema e o utilizador. A quantidade de ações que a Siri® pode realizar é muito grande. Através desta aplicação o utilizador pode por exemplo, pedir que a Siri® leia o último e-mail recebido, sem ter que mexer uma única vez no dispositivo que está a utilizar [7].

2.3 BMW Natural Interaction ®

A tecnologia tem tornado os carros cada vez mais interativos. Pensando nisso, a BMW desenvolveu um sistema com Inteligência Artificial (BMW Natural Interaction®) capaz de reconhecer comandos de voz, gestos e até mesmo olhares [8]. Com este sistema, que combina a mais avançada tecnologia de comando de voz disponível com um controlo de gestos expandido e o reconhecimento de olhar, o utilizador pode pela primeira vez num carro BMW usufruir de uma operação multimodal que torna a interação entre o condutor e o seu veículo ainda mais intuitiva [9].
O BMW Natural Interaction® permite ao motorista usar simultaneamente, em várias combinações, a sua voz, gestos e olhar para interagir com o seu veículo, uma vez que o software desenvolvido pode detetar e executar automaticamente as instruções desejadas [9].
Com esta solução, o condutor não precisa necessariamente de utilizar o controlo por voz ou o controlo por gestos, visto que o modo de operação preferido pode ser selecionado, de acordo com a situação e o contexto apresentados. Neste sistema, o motorista decide como deseja interagir com o carro, com base nas suas preferências pessoais, hábitos ou situação atual. Assim, quando o condutor está a conversar com outro ocupante do veículo, ele provavelmente escolherá o controlo através de gestos e olhares. No entanto, quando o condutor estiver mais concentrado na sua condução e tiver os seus olhos fixados na estrada, será mais seguro recorrer ao controlo através da voz e de gestos [9].
Esta interação multimodal é constituída pelo reconhecimento de voz, pela análise de gestos e pelo interpretar de olhares do condutor. As instruções que são enviadas pelo utilizador através da sua voz são registadas e processadas usando o Natural Language Understanding®, um algoritmo de aprendizagem, que é constantemente refinado, e que combina e interpreta as informações complexas recebidas, para que o veículo possa responder às mesmas de acordo com o pretendido. A junção de todos estes fatores cria uma experiência interativa multimodal voltada para os desejos do motorista [9].

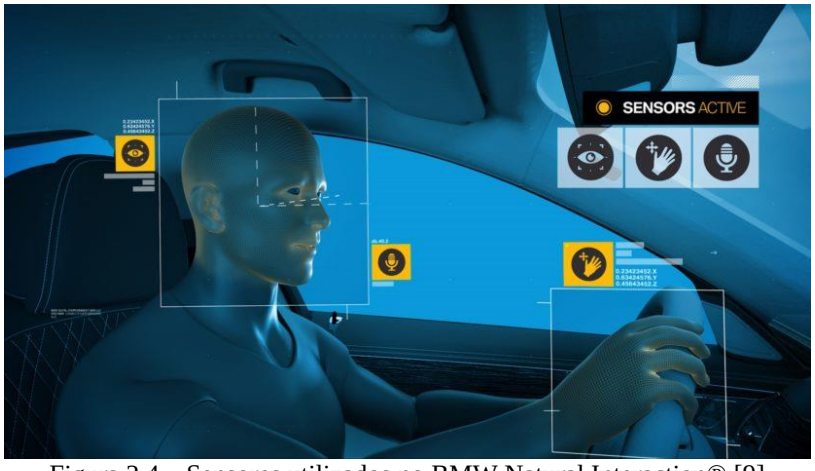


Figura 2.4 – Sensores utilizados no BMW Natural Interaction® [9]

Para reconhecer e avaliar os comandos de voz, gestos e olhares necessários para a interação natural do condutor com o veículo, este sistema usa tecnologias aprimoradas de sensores (Figura 2.4) e análise. Para capturar os movimentos das mãos e dos dedos em três dimensões em todo o ambiente operacional do condutor, esta solução usa luz infravermelhos e uma câmara

que permite determinar com precisão o vetor direcional. Isto permite, por exemplo, aumentar ou diminuir o volume da música que o condutor está a ouvir, sem ter que mexer no ecrã. Para tal, o condutor tem apenas que mover lentamente a mão com o dedo indicador esticado fazendo movimentos circulares no sentido dos ponteiros do relógio ou inverso, podendo dessa formar aumentar ou diminuir o volume (Figura 2.5).

Figura 2.5 – Demonstração de como aumentar o volume da música sem tocar no ecrã [10]

Além disso, esta solução, recorre também uma câmara de alta-definição instalada no painel de instrumentos do veículo que regista a direção da cabeça e dos olhos do condutor. Desta forma, o sistema consegue avaliar as imagens capturadas e usá-las para calcular os dados vetoriais necessários, que são processados no veículo. Assim, o motorista pode apontar o dedo para um objeto ou um lugar que esteja no seu campo de visão e dar comandos de voz relacionados, como, por exemplo, pedir informações sobre o horário de funcionamento ou avaliações de clientes, ou reservar uma mesa de um restaurante (Figura 2.6) [9].

Figura 2.6 – Demonstração de como pedir informações de um local através da interface multi-modal [9]

Para interpretar as instruções de voz de forma rápida e confiável além dos gestos, as informações transmitidas pelo motorista ao veículo de forma multi-modal são combinadas e avaliadas com o auxílio da inteligência artificial. O algoritmo responsável por interpretar os dados no carro é continuamente otimizado e refinado, de forma a permitir uma aprendizagem contínua e uma avaliação em diferentes cenários operacionais [9].

2.4 Análise de aspectos de usabilidade em interações naturais via interfaces multimodais

Nos últimos anos, dispositivos que possibilitam formas de interação consideradas mais naturais, começaram a estar amplamente disponíveis ao público e a ser utilizados com mais frequência, demonstrando um grande potencial para o seu uso [11].
Com o objetivo de estudar a aplicação dessas formas naturais de interação e para comparar os seus benefícios, foi efetuado o desenvolvimento e avaliação de uma interface multimodal para um sistema de apresentações, que permite ao utilizador utilizar o toque, gestos de corpo e a sua fala para interagir com o sistema [11].
Com este sistema, uma simples tarefa no dia-a-dia de professores e alunos da universidade, como por exemplo a apresentação de um trabalho, de uma aula ou de uma conferência, torna-se muito mais simples e intuitiva [11].
A configuração do sistema desenvolvido está representada na Figura 2.7 e a implementação prática pode ser vista na Figura 2.8.

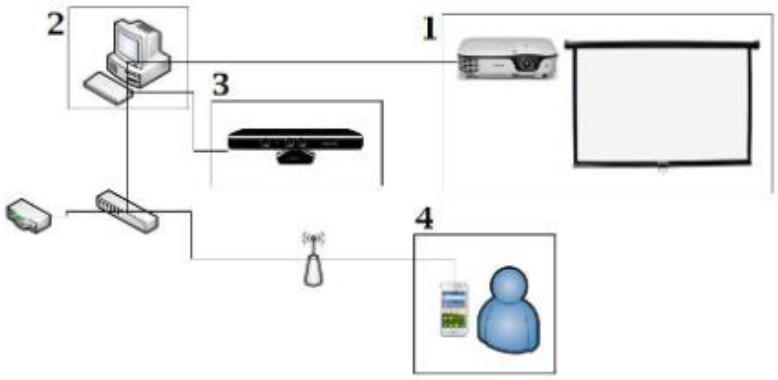


Figura 2.7 – Configuração do Sistema [11]

O sistema é composto pelos seguintes componentes:

1. Projetor de imagens e uma tela (Figura 2.8: B)
2. Computador Desktop ou Notebook, atuando como servidor (Figura 2.8: A)
3. Sensor Microsoft Kinect, para captura de gestos de corpo e fala (Figura 2.8: A)
4. Smartphone Android para captura de gestos de toque (Figura 2.8: C)

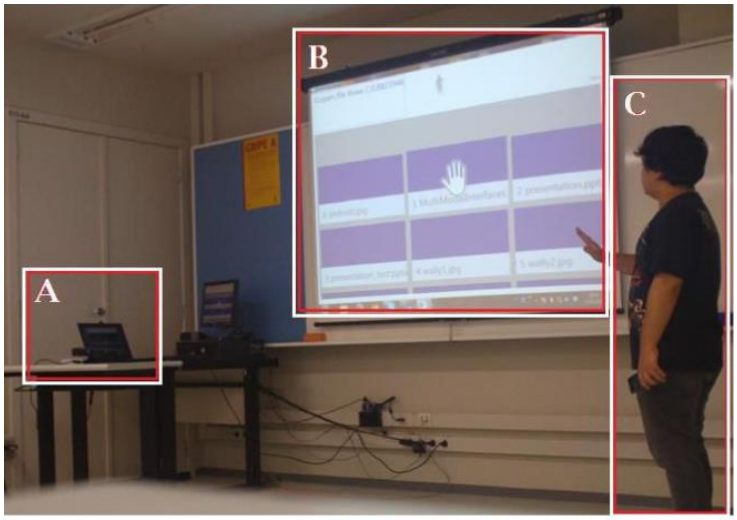


Figura 2.8 – Configuração do sistema implementado [11]

3 Solução desenvolvida

Depois de se analisar o problema apresentado, concluiu-se que se pretendia desenvolver uma bancada de demonstração em que o controlo do movimento num eixo linear pudesse ser realizado através de diferentes interfaces. Dessa forma, era necessário desenvolver-se um sistema demonstrativo em que se utilizasse uma interface multi-modal para o controlo de um eixo linear que cumprisse os seguintes requisitos:

- O primeiro requisito imposto era o desenvolvimento de uma interface multi-modal, que permitissem controlar o movimento no eixo linear de diferentes maneiras.
- O segundo requisito imposto era utilizar um eixo linear que tivesse um comprimento no mínimo 150 mm, para que se pudessem verificar as variações de velocidade e de deslocamento.
- O terceiro requisito, impunha que se pudesse controlar a velocidade, a distância e a direção do deslocamento no eixo linear. Isto é, qualquer das interfaces que fosse desenvolvida deveria permitir que o utilizador escolhesse o sentido, a distância e a velocidade de deslocamento no eixo linear.

Assim, estabeleceu-se que se iria desenvolver um sistema em que através de 3 interfaces distintas se pudesse realizar o controlo de movimento no eixo linear.
Neste capítulo será apresentada de uma forma detalhada a solução desenvolvida para chegar ao sistema pretendido. Dessa forma, irá começar por se apresentar o motor utilizado e o respetivo sistema de transmissão, a estrutura de fixação desenvolvida, os componentes escolhidos para esta solução e, por fim, por se apresentar e explicitar todas as interfaces desenvolvidas.
Na Figura 3.1, pode ver-se a arquitetura desenhada para este sistema, que permitiu desenvolver a solução para dar resposta ao problema apresentado.

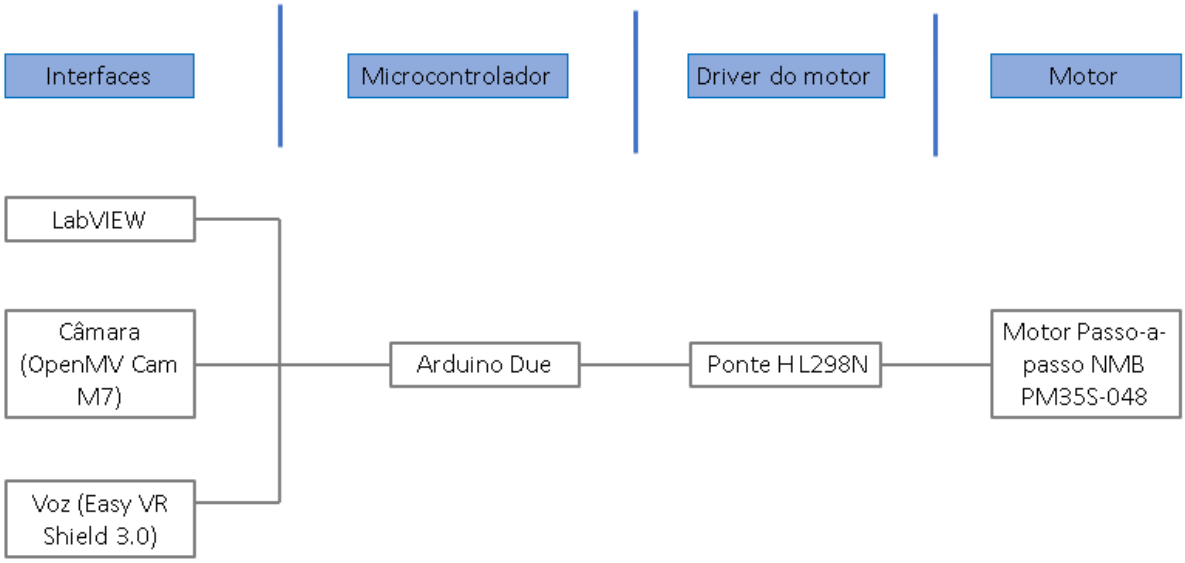


Figura 3.1 – Arquitetura do sistema

Tal como se pode verificar utilizou-se um motor de passo da marca NMB®, uma ponte H L298N como driver do motor, para que possa ser realizado o movimento do motor nas duas direções possíveis (horária e anti-horária) e um microcontrolador Arduino Due. Nessa mesma figura é ainda possível ver as diversas interfaces que foram desenvolvidas para se estabelecer a interação entre o utilizador e o sistema a controlar. De seguida explicar-se-á o porquê de todas as escolhas efetuadas e das diversas interfaces desenvolvidas.

3.1 Motor utilizado e respetivo sistema de transmissão

Perante as necessidades impostas, a primeira coisa a ter-se em consideração foi como e/ou qual seria o sistema que permitisse realizar o movimento no eixo linear.
Para tal, tiveram-se logo duas exigências em consideração. A primeira era encontrar uma solução que permitisse transformar o movimento de rotação do motor, num movimento de deslocação linear e a segunda era desenvolver um suporte que tornasse possível realizar uma fixação de forma segura e robusta de todos os componentes do sistema.
Desenvolver um sistema que para além de conter um motor transformasse o seu movimento de rotação num movimento de translação era dispendioso, visto que exigia um sistema redutor robusto e eficiente, capaz de reduzir o esforço à saída do motor. Além disso, era ainda necessário um sistema de transmissão que possibilitasse a realização do movimento linear num veio com algum comprimento.
Assim ultrapassou-se facilmente o primeiro obstáculo e o sistema reaproveitado foi aquele que se pode ver na Figura 3.2.

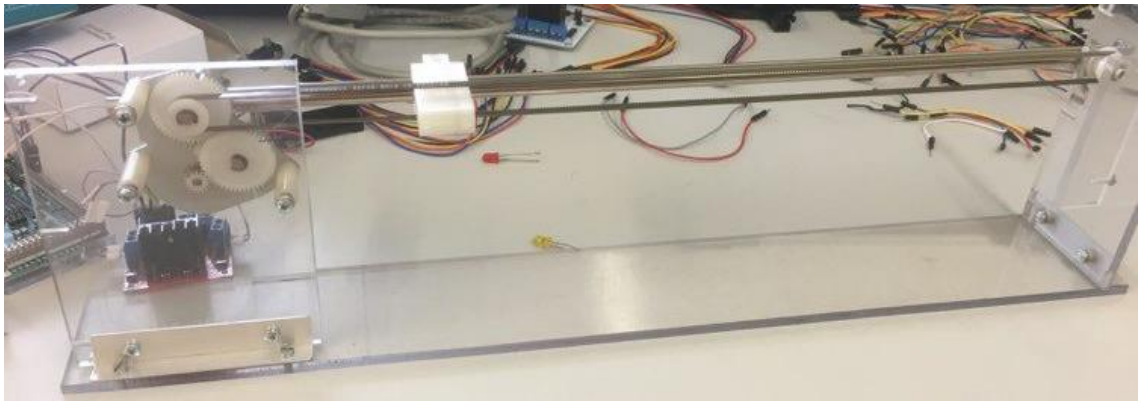


Figura 3.2 – Eixo linear com motor de passo reaproveitado de uma impressora

Esse sistema é composto pelo motor e respetivo sistema redutor, que contém diversas engrenagens (Figura 3.3), pelo veio com um comprimento de aproximadamente 200 mm e pelo sistema de transmissão (correia/polia) que permite realizar o movimento linear ao longo do eixo.

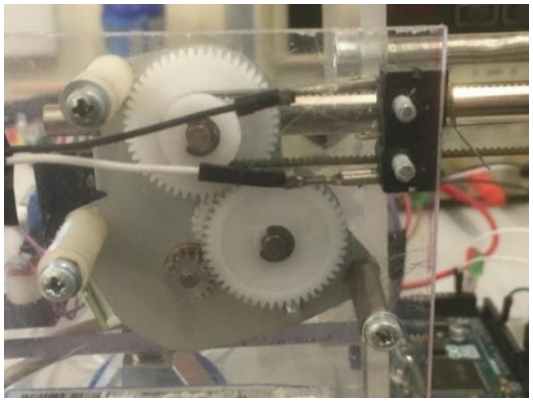


Figura 3.3 – Sistema redutor composto por diversas engrenagens

Na Figura 3.4 pode ver-se o sistema de transmissão completo, nomeadamente, o número de dentes (Z) de todas as engrenagens e o diâmetro (Ø) das duas polies responsáveis pelo movimento da correia. Com todos estes dados é possível determinar o deslocamento linear quando o motor realiza um passo, pelo que se torna possível realizar o controlo em malha aberta, ou seja, sem necessidade de utilizar um *encoder*.

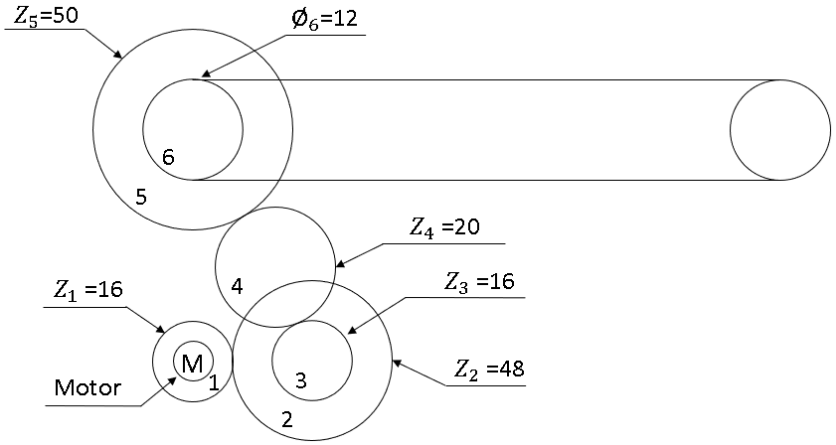


Figura 3.4 – Sistema de transmissão

Desta forma, o primeiro passo a realizar foi saber as características do motor utilizado, nomeadamente o deslocamento angular por passo. No anexo A é possível ver-se os dados técnicos do motor de passo utilizado (NMB PM35S-048), e as principais características podem ser vistas na Tabela 3.1.

Tabela 3.1– Dados do motor utilizado

Dados do motor
Marca: NMB
Modelo: PM35S-048-HPP6
$\boldsymbol{deslocamento\ angular}_{/\boldsymbol{passo}}\left(\left(^{\Delta\boldsymbol{\theta}}/_{\boldsymbol{passo}}\right)_{\boldsymbol{motor}}\right)\boldsymbol{: 7,5\ ^{\circ}/passo}$

O motor utilizado é um motor de passo bipolar de 4 fios [12], ou seja, as bobinas internas do motor são independentes, como se pode verificar na Figura 3.5.

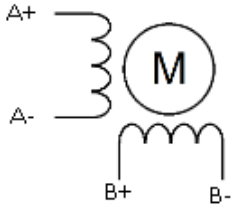


Figura 3.5- Motor de passo de 4 Fios Bipolar [12]

Convém ainda salientar que a forma como as bobinas são ligadas determina a forma de controlo do motor e o tipo de driver que terá de ser utilizado, como se irá ver no subcapítulo 4.3 quando se falar do driver utilizado [12].

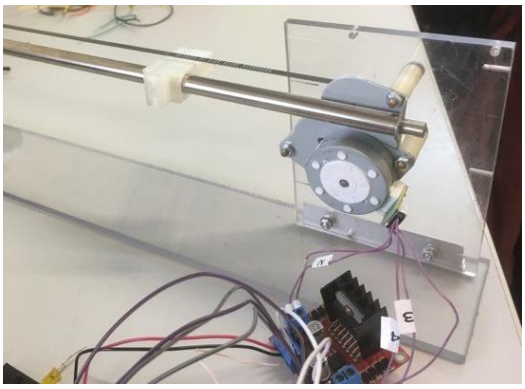


Figura 3.6- Motor de passo utilizado com os 4 fios

Após se descobrir a composição interna do motor, isto é, a arquitetura interna e a forma como ele é composto internamente como e a sequência de alimentação de cada bobina, realizaram-se então os cálculos que permitem determinar o deslocamento linear que é realizado quando o motor realiza um passo.
Aplicando a equação 1:

$$\frac{(^{\Delta\theta}/_{passo})_x}{(^{\Delta\theta}/_{passo})_y} = \frac{z_y}{z_x} \qquad (1)$$

Em que:
$(^{\Delta\theta}/_{passo})_x$ é o deslocamento angular da roda x em $^{\circ}/passo$
$(^{\Delta\theta}/_{passo})_y$ é o deslocamento angular da roda y em $^{\circ}/passo$
z_x é o nº de dentes da roda x
z_y é o nº de dentes da roda y

Assim, sabendo o deslocamento angular do motor ($(^{\Delta\theta}/_{passo})_{motor} = (^{\Delta\theta}/_{passo})_1$), e aplicando a equação (1), é possível obter o deslocamento angular em qualquer uma das engrenagens do sistema de transmissão apresentado, como se pode verificar nos cálculos seguintes:

$$\frac{(^{\Delta\theta}/_{passo})_1}{(^{\Delta\theta}/_{passo})_2} = \frac{z_2}{z_1} \Leftrightarrow (^{\Delta\theta}/_{passo})_2 = \frac{16}{48} \times 7{,}5 \Leftrightarrow (^{\Delta\theta}/_{passo})_2 = 2{,}5\,^{\circ}/_{passo}$$

Na Figura 3.4, pode verificar-se que as engrenagens 2 e 3 fazem parte da mesma roda e como tal, o seu deslocamento angular é igual, ou seja, $(^{\Delta\theta}/_{passo})_2 = (^{\Delta\theta}/_{passo})_3 = 2{,}5\,^{\circ}/_{passo}$.
Seguindo o mesmo raciocínio, e através da equação 1, é possível obter o resto dos deslocamentos angulares:

$$\frac{(^{\Delta\theta}/_{passo})_3}{(^{\Delta\theta}/_{passo})_4} = \frac{z_4}{z_3} \Leftrightarrow (^{\Delta\theta}/_{passo})_4 = \frac{16}{20} \times 2{,}5 \Leftrightarrow (^{\Delta\theta}/_{passo})_4 = 2\,^{\circ}/_{passo}$$

$$\frac{(^{\Delta\theta}/_{passo})_4}{(^{\Delta\theta}/_{passo})_5} = \frac{z_5}{z_4} \Leftrightarrow (^{\Delta\theta}/_{passo})_5 = \frac{20}{50} \times 2 \Leftrightarrow (^{\Delta\theta}/_{passo})_5 = 0{,}8\,^{\circ}/_{passo}$$

À semelhança das engrenagens 2 e 3, também as 5 e 6 fazem parte da mesma roda e como tal, $(^{\Delta\theta}/_{passo})_5 = (^{\Delta\theta}/_{passo})_6 = 0{,}8\,^{\circ}/_{passo} = 0{,}01396\ rad/passo$.

Como $180^{\circ} = \pi\, rad$,

$$0{,}8^{\circ} = \frac{\pi \times 0{,}8}{180} = 0{,}01396\, rad$$

Aplicando a equação 2:

$$\Delta x = \Delta\theta \times r \qquad (2)$$

Sendo que:
Δx é o deslocamento linear em mm
$\Delta\theta$ é o deslocamento angular rad
r é o raio em mm
Torna-se possível obter o deslocamento linear, neste caso, na engrenagem 6, o que permitirá saber qual o deslocamento linear da correia.
Assim, com base na equação 2, e como:

$$(^{\Delta\theta}/_{passo})_6 = 0{,}01396\, rad/passo, \text{ e } r_6 = \frac{\emptyset_6}{2} = \frac{12}{2} = 6\, mm:$$

Pode concluir-se que:

$$(^{\Delta x}/_{passo})_6 = 0{,}01396 \times 6 = 0{,}0838\, mm/passo$$

Desta forma, pode afirmar-se que por cada passo realizado pelo motor, o deslocamento linear é de 0,0838 mm, estabelecendo-se assim a seguinte equação (3):

$$\Delta x = \mathrm{N} \times 0{,}0838 \qquad (3)$$

Onde:
Δx é a distância de deslocamento linear no eixo em mm.
N é o número de passos que o motor de passo vai efetuar.

3.2 Estrutura de fixação desenvolvida

Posto isto, arranjou-se uma solução de fixação que permitisse fixar os componentes que pudessem vir a ser utilizados. Assim, desenvolveu-se um sistema de fixação robusto e leve, tendo-se para tal utilizado acrílico.
Para tal, começou por se projetar a estrutura no programa SOLIDWOKS® (Figura 3.7), para se proceder à fabricação da mesma.

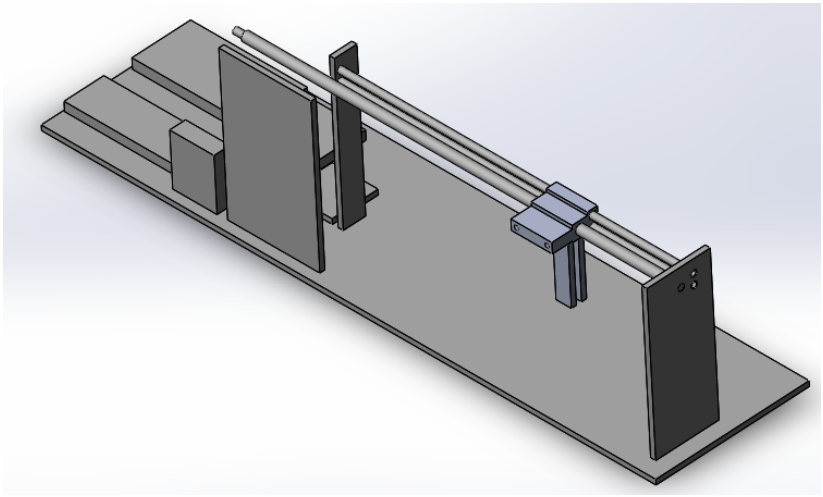


Figura 3.7 - Estrutura de fixação desenvolvida no programa SOLIDWORKS®

Na Figura 3.8, pode ver-se a solução final desenvolvida para suportar e fixar todos os componentes do sistema, e como se podem constatar é bastante semelhante ao que foi projetado no programa SOLIDWORKS ® (Figura 3.7).

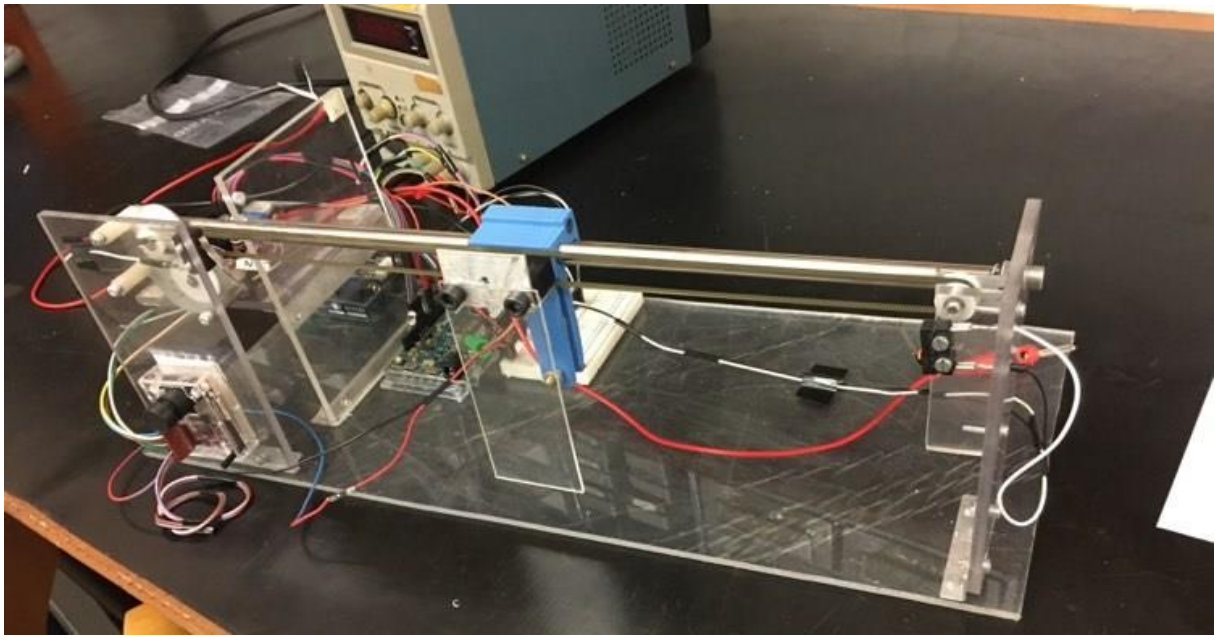


Figura 3.8 – Solução final do sistema de fixação desenvolvido em policarbonato

3.3 Componentes utilizados na solução

Tal como se pode verificar na Figura 3.8, a solução final que foi desenvolvida é composta por diversos componentes. Neste capítulo além de se identificarem todos os componentes utilizados, também irá ser descrito o modo de funcionamento dos mesmos e a respetiva função. Além do motor de passo, do sistema de transmissão e respetivo eixo linear e do sistema de fixação desenvolvido, do qual se falou no subcapítulo anterior, é necessário também falar um pouco dos seguintes componentes:

Sistema de gestão

O sistema de gestão é responsável por fazer toda a gestão de informação que é recebida e/ou emitida por um sistema. A coordenação das ordens recebidas (que são emitidas pelo utilizador através das interfaces desenvolvidas), dos sinais recebidos pelas variáveis de entrada (como por exemplo dos que são enviados pelos sensores fim de curso) e da emissão de ordens para as variáveis de saída (neste caso para o motor de passo), é realizada por este sistema.
Neste caso, para o sistema de gestão, foi utilizado um microcontrolador e não um PLC, sendo que a principal diferença entre ambos é o tamanho e a capacidade de processamento de informação, que são bem maiores num PLC ainda que o princípio de funcionamento e a função de ambos seja exatamente a mesma. Além disso, o PLC também garante mais fiabilidade, razão pela qual é muito mais utilizado em aplicações industriais. No entanto, e tendo em consideração as condições e necessidades do nosso problema, decidiu-se utilizar um microcontrolador Arduino Due (Figura 3.9) porque é uma solução mais económica e que satisfaz completamente todos os requisitos impostos.
O Arduino® Due é a primeira placa Arduino® que tem um microcontrolador ARM® de 32 bits baseado no Atmel® SAM3X8E ARM [13]. Dadas as exigências do problema apresentado e em particular o facto de se terem desenvolvidas 3 interfaces diferentes, existia a necessidade de se

utilizar um controlador que tivesse um elevado número de portas de entradas e/ou saídas digitais e que possuísse uma boa agilidade e alto poder de processamento.

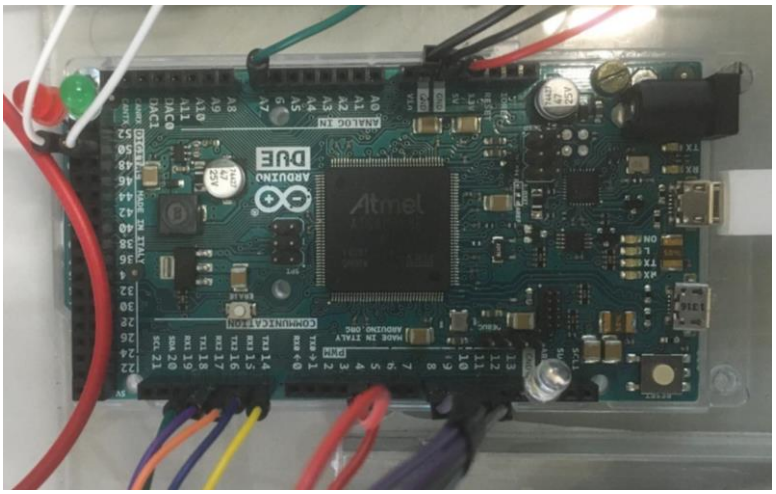


Figura 3.9 - Microcontrolador utilizado na solução desenvolvida (Arduino® DUE)

Uma vez que esta placa de Arduino® Due tem 54 pinos de entrada e saída digital, dos quais 12 podem ser utilizados como saídas PWM (*Pulse Width Modulation*), 12 entradas analógicas, 4 UARTs (*Universal asynchronous receiver/transmitter*), *clock* de 84MHz, uma conexão USB OTG (*on the go*), 2 DAC (*digital to analog converter*), 2 TWI (*two wire interface*), uma entrada de alimentação, um barramento SPI (*Serial Peripheral Interface*), um barramento JTAG (*Joint Test Action Group*), um botão de *reset*, e um botão de *erase* [13] e ainda o facto de ser uma das placas mais rápidas da família Arduino®, ideal para quem precisa de agilidade e alto poder de processamento [14], foram algumas das razões pelas quais se decidiu utilizar esta placa para controlador do sistema desenvolvido.

Por fim, mas não menos importante, o facto de todo material (*software*, bibliotecas, *hardware*) ser *open-source* é outra característica importantíssima, pois além de possuir uma biblioteca de grande versatilidade, também a utilização do programa é de extrema facilidade.

Um bom exemplo disso, são as Figura 3.10 e Figura 3.11, onde se pode ver algumas das bibliotecas e das funções que foram utilizadas para programar o sistema que foi desenvolvido.

```
#include <Stepper.h>
const int stepsPerRevolution = 48; //nº de passos realizados numa volta completa pelo motor de passo
Stepper myStepper(stepsPerRevolution,8,9,10,11); //sequência de alimentação do motor de passo
```

Figura 3.10 – Biblioteca do programa Arduino® utilizada na programação desenvolvida

Como o motor elétrico utilizado na solução desenvolvida foi um motor de passo, uma biblioteca que neste caso em particular foi muito útil foi a biblioteca "Stepper.h", como é possível verificar na Figura 3.10. Como se pode verificar nessa mesma figura, recorreram-se a duas funções que facilitaram muito o controlo do motor: a função ("stepsPerRevolution") onde se tem que indicar o nº de passos realizados pelo motor para dar uma volta completa (48) e a função ("myStepper") que serve para indicar a sequência de alimentação das bobinas do motor (8,9,10 e 11, que correspondem às entradas digitais onde foram ligados os fios 1,2,3 e 4 do motor de passo utilizado).

Isto facilitou muito e foi uma ajuda enorme na programação desenvolvida porque não só a tornou mais simples como também mais rápida e prática. Um bom exemplo disso são as funções (Figura 3.11) que foram utilizadas, que permitem selecionar a velocidade ("myStepper.setSpeed()") e a distância ("myStepper.step()") de trabalho do motor.

```
myStepper.setSpeed(velocidade); // Indicação da velocidade de trabalho do motor
myStepper.step(-distancia); // Indicação do número de passos que o motor irá realizar.
```

Figura 3.11 – Funções do programa Arduino® utilizadas na programação desenvolvida

Driver do motor

A função do *driver* é regular a entrega da excitação elétrica ao motor, ou seja, a de alimentar o motor com a sequência correta para que o mesmo possa operar de acordo com o pretendido.
Como já foi referido, o motor escolhido foi um motor de passo de 4 fios bipolar, onde as bobinas internas do motor são independentes (Figura 3.5). Por essa razão, o *driver* do motor escolhido foi uma ponte H L298N (Figura 3.12).
O uso desta ponte H como *driver* do motor permite que este se mova nos dois sentidos possíveis (horário e anti-horário). Tal facto acontece porque com o uso deste *driver* é possível inverter a polaridade de alimentação do motor sem que seja necessário utilizar uma fonte de alimentação simétrica.

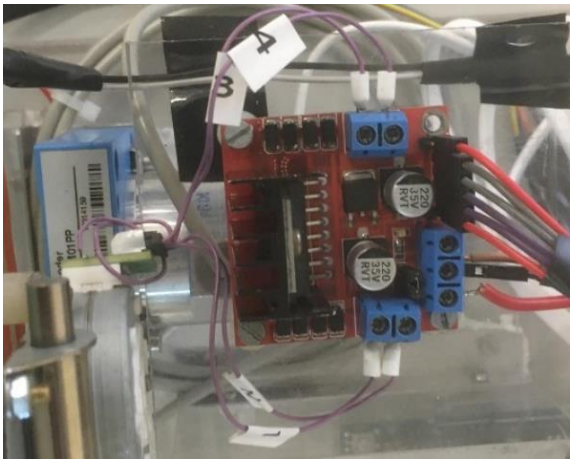


Figura 3.12 – Driver do motor utilizado na solução desenvolvida (Ponte H L298N)

Uma ponte H é um circuito idêntico ao da Figura 3.13, composto essencialmente por 4 transístores (S1, S2, S3 e S4), que são acionados 2 a 2 de forma alternada, isto é, (S1-S3) ou (S2-S4) [15].

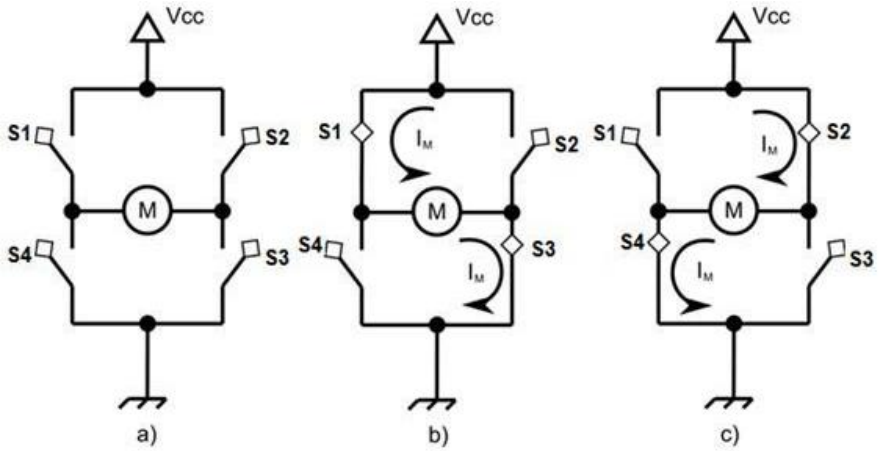


Figura 3.13 - Circuito Ponte H [15]

Depois desta breve explicação e após já se ter falado um pouco do motor, controlador e *drive* selecionados, convém fazer aqui um pequeno reparo, para demonstrar a forma como o motor, o controlador e o *driver* estão ligados (eletricamente) entre si (Figura 3.14).

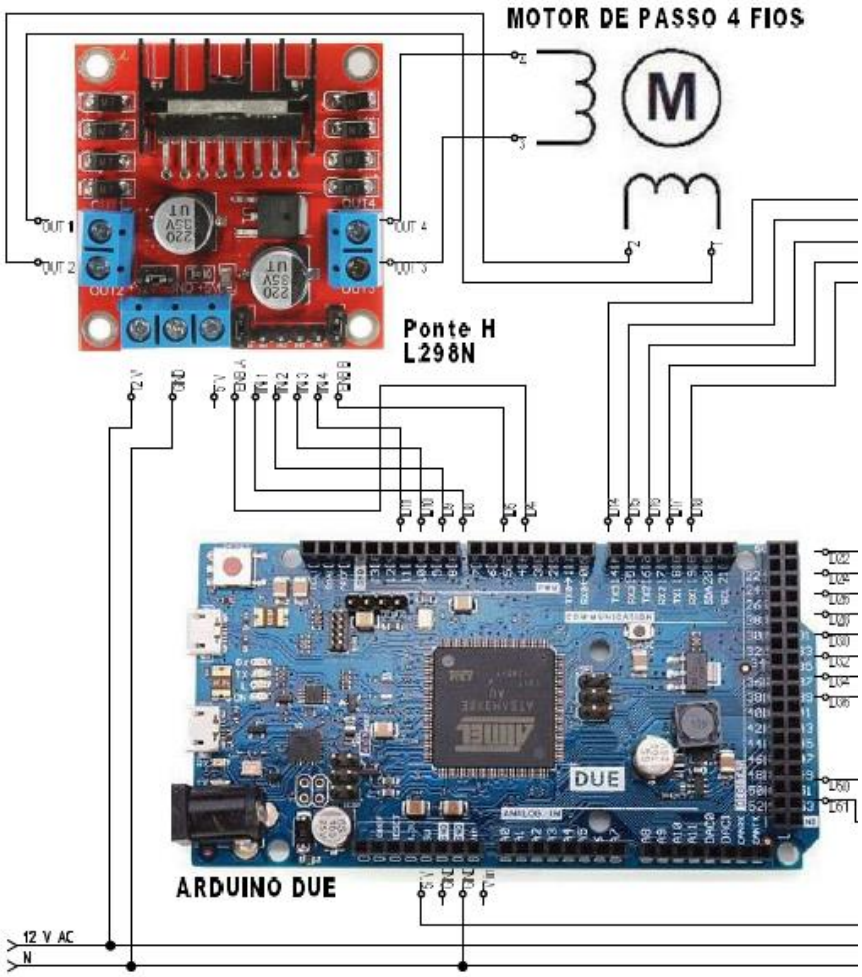


Figura 3.14 – Esquema das ligações elétricas entre o motor, o driver e o controlador

Sensores fim de curso

Neste caso em concreto, utilizaram-se 2 sensores de fim de curso mecânicos (Figura 3.15) que permitem detetar quando são atingidos os limites do eixo linear.

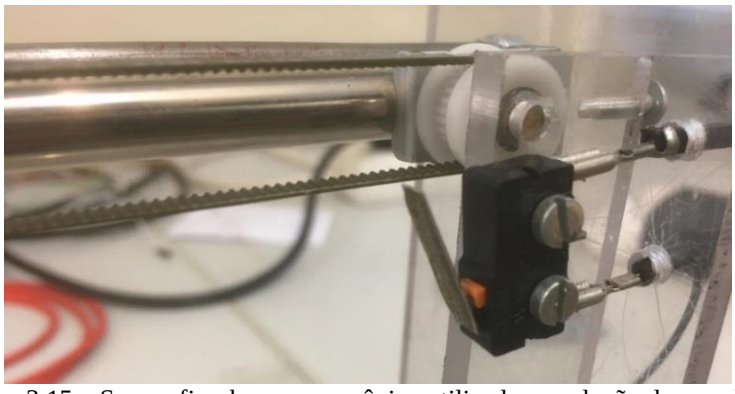


Figura 3.15 – Sensor fim de curso mecânico utilizado na solução desenvolvida

Na Figura 3.16 estão apresentadas as ligações elétricas que foram realizadas para que os fins de curso esquerdo (FDCE) e direito (FDCD) funcionassem de acordo com o pretendido.

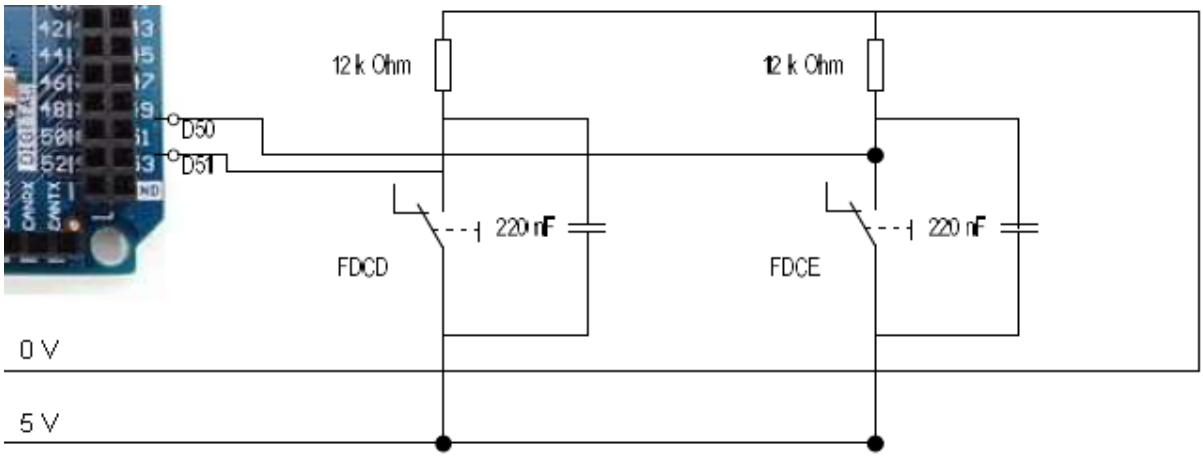


Figura 3.16 – Esquema das ligações elétricas entre os fins de curso e o controlador

À semelhança do que já foi referido anteriormente, a utilização do programa Arduino® para efetuar a programação do sistema possibilitou o uso de bibliotecas e funções que facilitaram muito o trabalho que foi desenvolvido. Como tal, para programar o funcionamento pretendido dos fins de curso decidiu-se utilizar uma função ("attachInterrupt") que quando é acionada, interrompe o programa esteja ele em que ponto estiver. No entanto, existem várias condições que podem ser escolhidas para acionar esta função. Neste caso, como se pode verificar na Figura 3.17, a condição escolhida foi ("RISING") que deteta quando ocorre uma transição do estado "LOW ou 0" para "HIGH ou 1" na entrada escolhida.

```
//Função attachInterrupt, responsável pelo devido funcionamento dos Fins de Curso do eixo linear
attachInterrupt(digitalPinToInterrupt(51),fimcursoesquerdo,RISING);  // Fim de curso Esquerdo:
//Quando o fim de curso esquerdo é acionado, a entrada digital nº 51, sofre uma transição de LOW para HIGH,
attachInterrupt(digitalPinToInterrupt(50),fimcursodireito,RISING);  //  Fim de curso Direito:
//Quando o fim de curso direito é acionado, a entrada digital nº 50, sofre uma transição de LOW para HIGH,
}
```

Figura 3.17 – Funções do programa Arduino® utilizadas para programar o funcionamento dos fins de curso

Quando, por exemplo, o fim de curso direito é acionado, existe uma transição do estado LOW para o estado HIGH na entrada digital 50 (entrada essa onde está ligado um dos fios do fim de curso direito), o programa é interrompido e irá entrar em funcionamento uma rotina com o nome de ("fimcursodireito"), como se pode ver na Figura 3.18 e que irá acender o led do lado direto e cortar a alimentação aos 2 enrolamentos do motor.

```
void fimcursoesquerdo() { //Quando o fim de curso esquerdo é accionado, o programa é interrumpido
digitalWrite(led_esq,HIGH);
digitalWrite(enrolamentoA,LOW); // Corte da alimentação do enrolamento A do motor de passo, o que
digitalWrite(enrolamentoB,LOW);} // Corte da alimentação do enrolamento B do motor de passo, o qu

void fimcursodireito() {//Quando o fim de curso direito é accionado, o programa é interrumpido de
digitalWrite(led_dir,HIGH);
digitalWrite(enrolamentoA,LOW);// Corte da alimentação do enrolamento A do motor de passo, o que
digitalWrite(enrolamentoB,LOW);}// Corte da alimentação do enrolamento B do motor de passo, o que
```

Figura 3.18 – Programa desenvolvido para o funcionamento dos fins de curso

A implementação prática deste modo de funcionamento dos fins de curso, teve alguns problemas, sendo que o principal foi que no instante em que o fim de curso era desacionado, ocorria *bouncing*.

Basicamente o que ocorria é que quando o interruptor era libertado, este ficava a vibrar, e essa vibração fazia com que este se ativasse e desativasse várias vezes, o que provocava uma variação no valor recebido na entrada digital de 1 para 0 e de 0 para 1, levando a que a função ("attachInterrupt") fosse ativada apesar do fim de curso já não estar acionado. Para se resolver esse problema, decidiu-se utilizar um condensador de 220 nF, que permitiu "absorver" essa vibração, como se pode verificar na Figura 3.19.

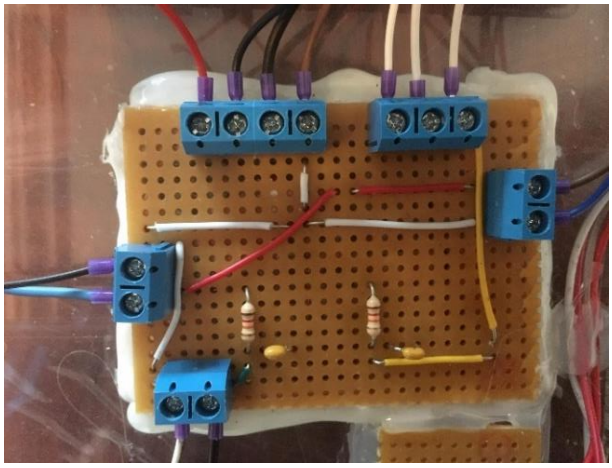


Figura 3.19 – Placa com o circuito elétrico e respetivas ligações usado na solução desenvolvida

Suporte da placa de acrílico

Após se escolher o eixo linear com o respetivo motor de passo e sistema de transmissão e de se desenvolver o sistema de fixação do mesmo, era ainda necessário projetar e desenvolver um suporte que além de ter de fixar a placa, tinha ainda de se mover em conformidade com a correia, de forma a possibilitar a transferência de movimento linear à placa. Na Figura 3.20, pode ver-se o suporte que foi desenvolvido.

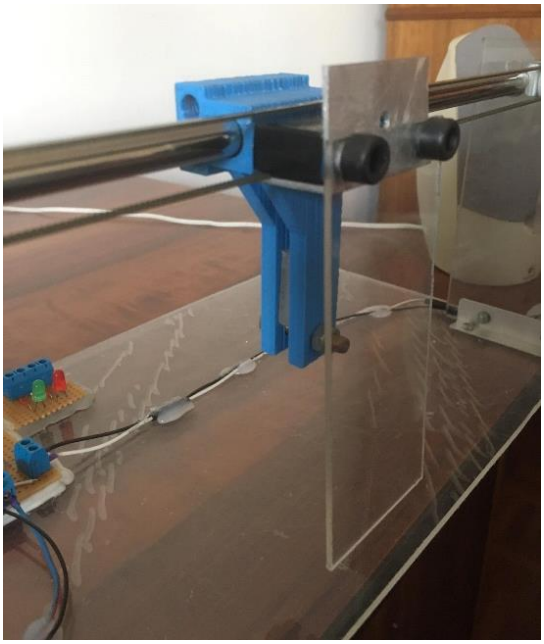


Figura 3.20 – Suporte da placa de acrílico desenvolvido

3.4 Interfaces desenvolvidas

Na Figura 3.21 é apresentado um esquema onde não só se podem ver as 3 interfaces que foram desenvolvidas, como também a maneira como os principais componentes do sistema estão ligados entre si. Além disso é ainda possível verificar-se o percurso que é realizado desde que o utilizador envia uma ordem para o sistema até que essa seja executada pelo motor de passo.

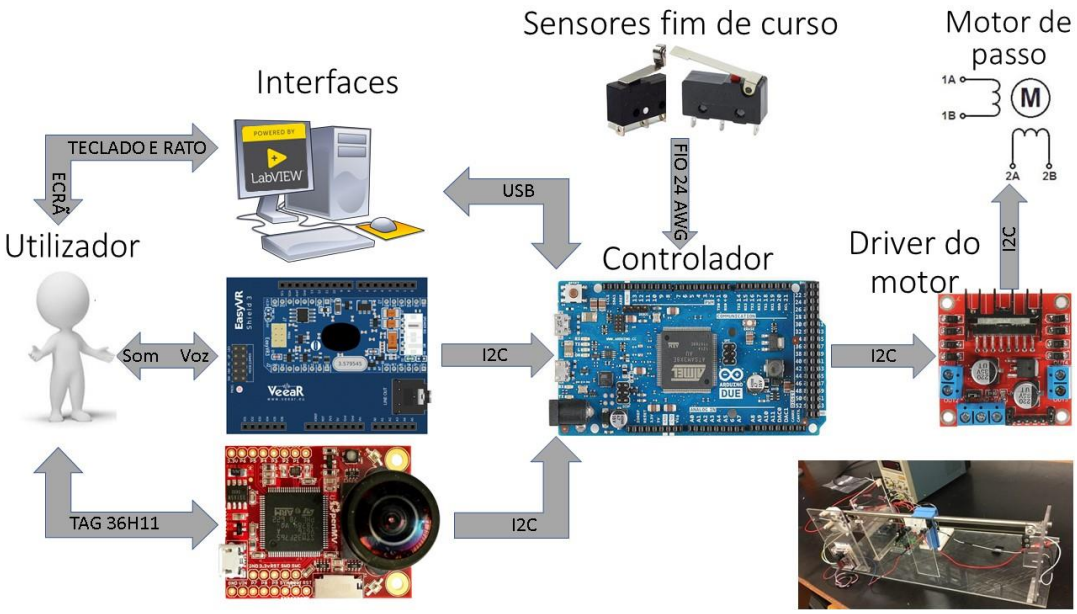


Figura 3.21 – Representação esquemática das comunicações/ligações entre os principais componentes do sistema

No entanto, e com o objetivo de deixar bem claro o modo de funcionamento do sistema, vai de seguida explicar-se detalhadamente todas as interfaces que foram desenvolvidas e a maneira como através delas é controlado o sistema.

Controlo do sistema através do programa LabVIEW®

O LabVIEW® é um software que permite controlar e fazer a aquisição de dados de um sistema, para que estes possam ser analisados e, a partir dessa análise, retirar as devidas conclusões.
Como tal neste tipo de interface, será necessário um computador para fazer toda a programação necessária.
Para a primeira interface desenvolvida neste trabalho decidiu-se que no programa LabVIEW® se iria desenvolver uma HMI, onde o utilizador pudesse ordenar ao sistema o que pretendia, mais concretamente, o sentido, a velocidade e o deslocamento para o qual a porta se iria deslocar. Além disso seria também importante receber uma indicação da mensagem que seria enviada ao controlador, para que se tivesse a certeza que a mensagem enviada correspondia àquela que o utilizador pretendia enviar. Por fim, mas não menos importante era necessário receber uma confirmação por parte do controlador que garantisse que a mensagem tinha sido recebida.
Tendo em conta todos os requisitos impostos, chegou-se então à versão final da HMI desenvolvida (Figura 3.22).

Figura 3.22 – HMI que desenvolvida no programa LabVIEW®

A programação que se desenvolveu para se chegar a esta versão final da HMI pode ser vista mais pormenorizadamente no Anexo C.

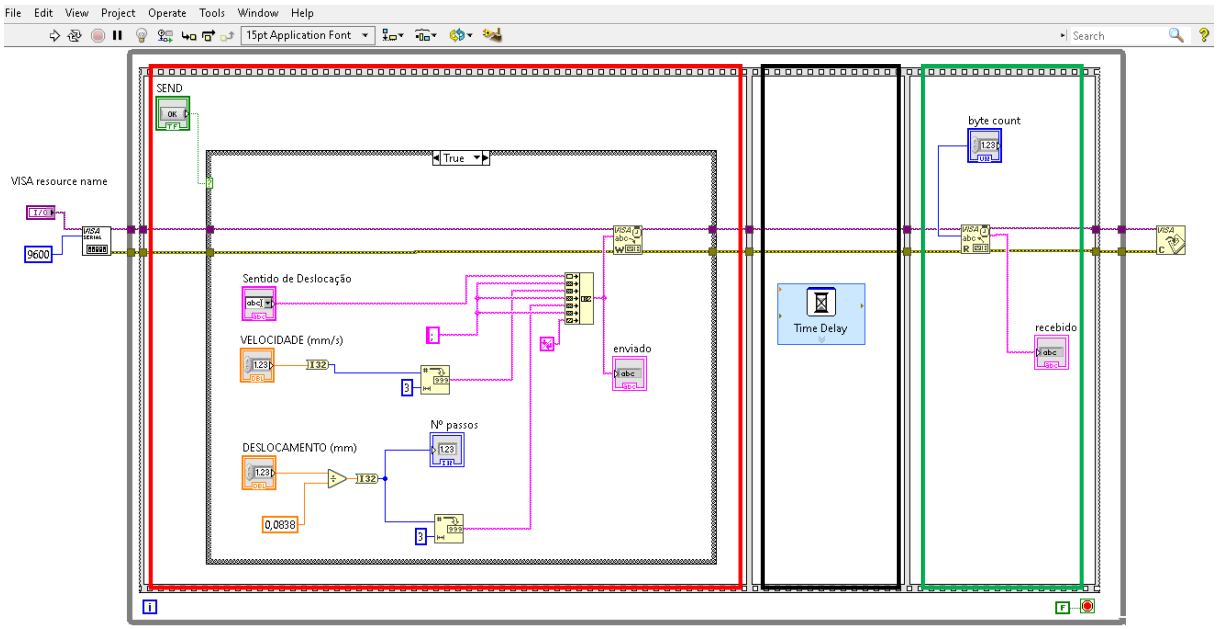


Figura 3.23 – Programação desenvolvida para se chegar à versão final da HMI

Na Figura 3.23 pode ver-se o programa desenvolvido e facilmente se pode constatar que o mesmo se encontra divido em 3 etapas, que estão preenchidas com retângulos. Na primeira etapa (retângulo vermelho) o utilizador tem que selecionar todos os dados pretendidos e clicar no botão ("SEND") para enviar a informação ao controlador. A segunda etapa (retângulo preto) é uma etapa de espera que tem como objetivo dar um tempo de espera para que se garanta que a mensagem foi não só enviada pelo LabVIEW®, como também recebida pelo controlador. Além disso é ainda de notar que esta comunicação é realizada via porta série. Por fim, chega-se à terceira etapa (retângulo verde) onde o objetivo é garantir que o controlador recebeu a mensagem que foi enviada.

É importante ainda fazer um reparo, pois na mensagem que é enviada, são enviados 3 números separados por (;), na seguinte forma (a;b;c), sendo que cada um deles tem o seguinte significado:

- O primeiro número enviado (a) é o que corresponde ao sentido de deslocação sendo que caso o sentido selecionado seja o sentido esquerdo, esse nº tomará o valor 0. Caso sentido o selecionado seja o sentido direito, esse nº tomará o valor 1.
- O segundo número enviado (b) é o que corresponde à velocidade de deslocação.
- O terceiro número enviado (c) é o que corresponde ao número de passos realizados pelo motor. Tal como foi visto anteriormente, mais concretamente no subcapítulo 4.1, esse valor está diretamente relacionado com a distância que a porta irá percorrer. Desta forma, o utilizador indica o deslocamento que a porta irá percorrer, no entanto, a informação que é transmitida ao controlador é o número de passos que o motor terá que realizar.

Posto isto, falta apenas falar da programação que foi desenvolvida para o controlador de forma a garantir a coordenação entre a mensagem que é enviada pelo utilizador e o movimento do motor que permite a porta deslocar-se na direção, velocidade e deslocamento pretendido.

No anexo D, pode ver-se toda essa programação com os devidos comentários. Além de alguns passos normais que têm de ser realizados quando se desenvolve um programa como por exemplo a definição e declaração de variáveis, ou de algumas funções que se utilizaram e que já se explicou a razão pelo o qual foram usadas, como por exemplo a função ("attachInterrupt"), convém ainda explicar e justificar a linha de pensamento que foi seguida na programação desenvolvida, recorrendo para tal ao diagrama de fluxo da Figura 3.24.

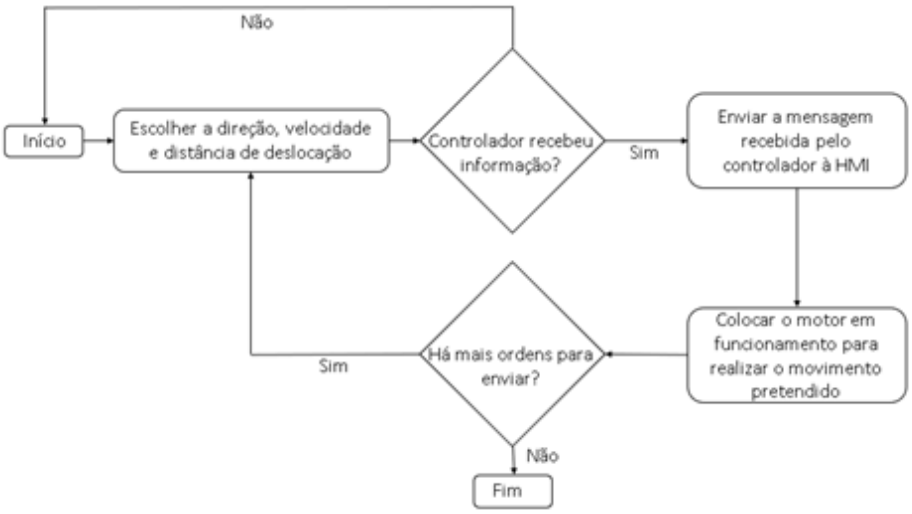


Figura 3.24 – Diagrama de fluxo para descrever a programação da interface com o LabVIEW® e da interface por voz

Controlo do sistema através de uma câmara

A segunda interface desenvolvida foi o controlo do sistema através de uma câmara. Como tal, decidiu-se recorrer à OpenMV® Cam M7 (Figura 3.25) que é uma pequena placa microcontroladora de baixa potência que permite implementar facilmente aplicações usando a visão por computador. A principal razão pela qual se decidiu optar por utilizar esta placa foi o enorme potencial que esta tem, uma vez que é uma placa microcontroladora que possui uma câmara, o que é uma enorme vantagem para ser utilizado como uma interface pois permite além de captar, processar a imagem que é captada de forma a transferir para o sistema toda a informação necessária e/ou pretendida. Outra razão que levou ao uso desta placa foi o facto do programa OpenMV IDE® possuir uma boa biblioteca e *software* ser o MicroPython®, o que permite realizar toda a programação da mesma utilizando a linguagem Python 3, escrita em C. Assim, decidiu-se implementar o seguinte modo de funcionamento: após se captar a imagem de referência é necessário medir a inclinação com que a mesma é captada e em função dessa inclinação, a porta poderá ou não movimentar-se, para a esquerda ou para a direita, com maior ou menor velocidade.

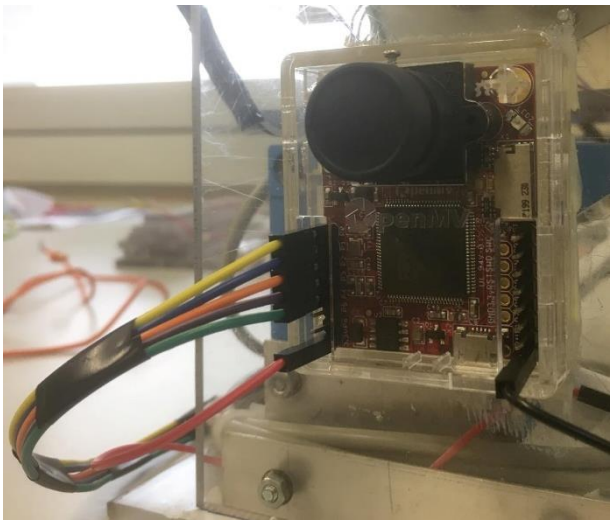


Figura 3.25 – Placa utilizada para desenvolver a segunda interface (OpenMV® Cam M7)

Na Figura 3.26, pode ver-se alguma da programação que foi desenvolvida para programar a placa OpenMV® Cam M7 e que pode ser vista na sua totalidade no anexo E.

```
while(True):
    clock.tick()
    img = sensor.snapshot() # definição da variável img, que é a captação da imagem em cada instante.

    for tag in img.find_apriltags(families=tag_families): # defaults to TAG36H11 without "families".
        img.draw_rectangle(tag.rect(), color = (255, 0, 0)) # desenho de um rectângulo em volta da imagem captada
        img.draw_cross(tag.cx(), tag.cy(), color = (0, 255, 0)) # desenho de uma cruz no centro da imagem captada
        rotacao=(180*tag.rotation())/math.pi # calculo que permite saber qual a inclinação da imagem captada
        print("Rotação:", rotacao,"Graus") # indicação da inclinação que a imagem no momento em que foi captada
```

Figura 3.26– Programação da placa OpenMV® Cam M7 desenvolvida no programa OpenMV IDE®

Nessa mesma figura, é ainda possível verificar que se utilizaram funções que permitem captar a imagem (sensor.snapshot()) de referência (TAG36H11), desenhar um retângulo em redor da imagem captada (img.draw_rectangle), desenhar uma cruz que indica o centro da imagem captada (img.draw_cross) e por fim calcular a inclinação com que a imagem é captada (rotacao). Após se ter decidido a imagem de referência a utilizar (TAG36H11), estabeleceram-se as zonas e as respetivas inclinações de forma a realizar o controlo do sistema, como se pode verificar na Figura 3.27.

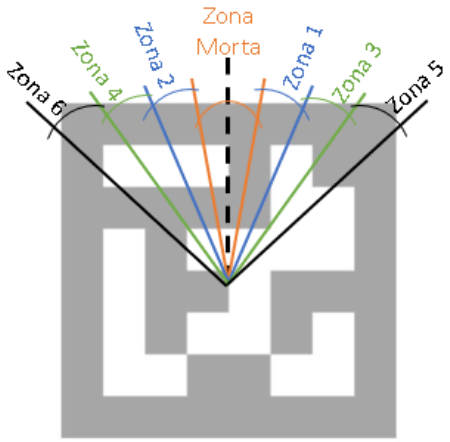


Figura 3.27 – TAG36H11 com as inclinações das zonas definidas

Na Tabela 3.2, é possível verificar-se além dos intervalos de inclinação, a ordem que está a ser transmitida para o sistema em função da zona que está a ser utilizada.

Tabela 3.2 – Zonas de trabalho em função da inclinação da imagem de referência (TAG 36H11)

Zonas	Intervalo de Graus de Inclinação	Sentido de deslocação	Velocidade de deslocação
Zona Morta	350º-10º	Nenhum	Nula
Zona 1	10º-30º	Esquerdo	Baixa
Zona 2	330º-350º	Direito	Baixa
Zona 3	30º-50º	Esquerdo	Média
Zona 4	310º-330º	Direito	Média
Zona 5	50º-70º	Esquerdo	Alta
Zona 6	290º-310º	Direito	Alta

Como se pode constatar, quando a figura que é captada está muito pouco inclinada, mais propriamente, com inclinação inferior a 10º independentemente do sentido de rotação, a ordem que é emitida para o sistema, ordena que a placa não se desloque (zona morta). Além disso, pode ainda verificar-se que caso a figura de referência seja captada inclinada no sentido de rotação horário (zonas ímpares 1,3 e 5), a ordem emitida para o sistema, ordena que a placa se desloque no sentido esquerdo e que caso a figura seja captada inclinada no sentido de rotação anti-horário (zonas pares 2,4 e 6), a ordem emitida para o sistema, ordena que a placa se

desloque no sentido direito. Por fim, mas não menos importante, é ainda de salientar que a velocidade de deslocação da placa aumenta proporcionalmente com o aumento de inclinação da imagem captada, ou seja, quanto maior for a inclinação com que a figura de referência é captada maior será a velocidade de deslocação da placa.
No anexo F pode ver-se a programação do controlador desenvolvida no programa Arduino® para a interface com a placa OpenMV® Cam M7. Como se pode verificar, cada linha de código que foi desenvolvida está devidamente comentada e explicada.
Analisando bem a programação desenvolvida para o controlador, facilmente se constata que é um pouco diferente da desenvolvida para a primeira interface. A razão pela qual a programação é diferente é o facto de que nesta interface, o modo de movimento da porta pode ser contínuo, isto é: como a imagem é constantemente captada num intervalo de tempo muito curto, a porta pode deslocar-se de forma contínua caso receba mais do que uma ordem seguida que assim o indique, ainda que a imagem não esteja a ser captada a todo o instante, mas sim num intervalo de tempo muito pequeno.
Na Figura 3.28 pode ver-se o diagrama de fluxo, para descrever o programa desenvolvido.

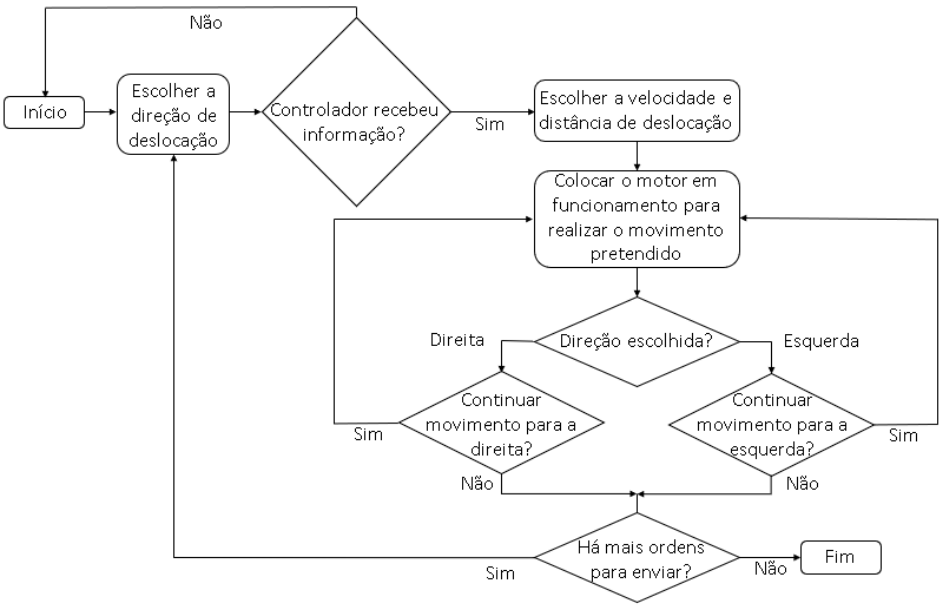


Figura 3.28 – Diagrama de fluxo para descrever a programação da interface com a câmara

Controlo do sistema por voz

Nesta 3ª interface desenvolvida o objetivo é realizar o controlo do sistema por voz, tendo-se para tal idealizado o seguinte modo para controlar o sistema: caso o utilizador queira que a porta se desloque para a direita, diz para o microfone a palavra "direita" e através do módulo de reconhecimento de voz utilizado e dos programas desenvolvidos, o controlador faz com que o motor se mova no sentido indicado para que a placa se mova sobre o eixo linear para a direita, e assim sucessivamente, caso se deseje que a placa se desloque para a esquerda ou pare. Além disso, o utilizador pode ainda decidir qual a distância e a velocidade pretendida.
Assim, implementou-se o modo de funcionamento idealizado sendo que para tal, começou por se escolher a placa de reconhecimento de voz a utilizar, tendo-se optado por usar a EasyVR® Shield 3.0.
A EasyVR® Shield 3.0 é uma placa para reconhecimento de voz para placas Arduino®, que integra um módulo EasyVR®. Como tal, inclui todas as características do módulo EasyVR® em formato de *shield* para simplificar a conexão com o Arduino® e com o PC [16].
O EasyVR® é um módulo de reconhecimento de voz de uso múltiplo desenvolvido para adicionar versatilidade e robustez a projetos que necessitam destas características [16]. É a segunda geração do módulo VRbot® e foi baseado nas suas características e funcionalidades,

sendo que tem como principal característica os 32 comandos dependentes de voz (SD) que são programados pelo utilizador [16]. Além disso, o *shield* tem conectores adicionais para a entrada de um microfone, uma saída para um altifalante de 8Ω, conector para saída de áudio, acesso aos pinos de entrada e saída do módulo EasyVR® e um LED programável que permite mostrar o feedback durante as tarefas de reconhecimento de voz [16].
Na Figura 3.29 além da placa de reconhecimento de voz utilizada, pode ainda ver-se o microfone e a coluna que permitem enviar e receber o feedback do sistema.

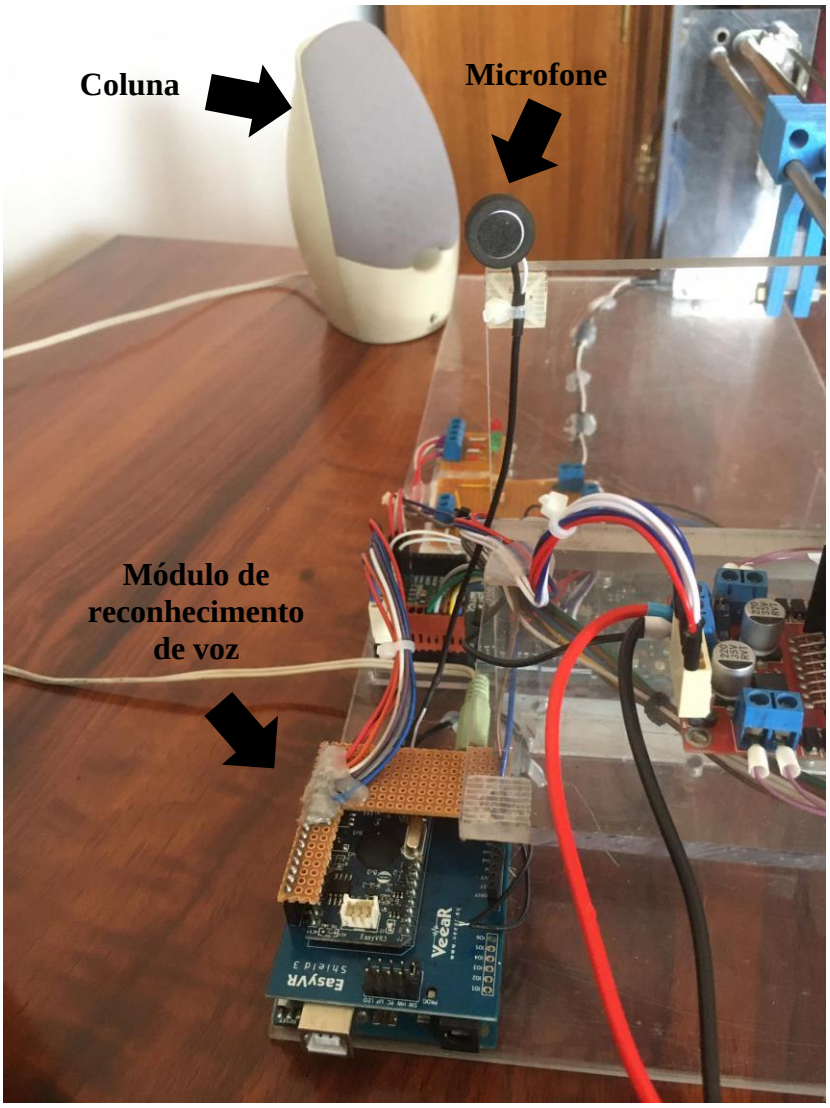


Figura 3.29– Placa de reconhecimento de voz utilizada (EasyVR® Shield 3.0)

Depois de escolhida a placa a utilizar e após se terem analisado todas as suas características, chegou-se à conclusão de que o número de comandos disponíveis não era suficiente para colocar em prática a ideia que tinha sido inicialmente idealizada.
Assim, e tendo em consideração todos os fatores, nomeadamente o número máximo de comandos programáveis (32) decidiu-se que o utilizador poderia escolher 3 valores diferentes para a velocidade de deslocamento (velocidade grande (430 mm/s), velocidade média (380 mm/s) ou velocidade pequena (330 mm/s)) e 3 valores distintos para a distância de deslocamento (distância máxima (80 mm), distância média (45 mm) ou distância mínima (20 mm)). Como tal, estabeleceram-se os comandos necessários, mais concretamente 20 comandos que irão ser responsáveis porque fazer com que a porta se mova na direção, com a velocidade e com a distância que o utilizador decida.

Na Tabela 3.3 podem ver 18 dos 20 comandos que foram programas, bem como a respetiva função e modo de acionamento de cada um.

Tabela 3.3 – Comandos de Voz programados

Comando de Voz Nº	Ordem enviada para o sistema			Palavra que o utilizador tem que emitir para acionar o comando
	Direção de deslocamento	Velocidade de deslocamento	Distância de deslocamento	
Comando nº 2	Direita	Grande	Máxima	"Dois!"
Comando nº 3	Direita	Grande	Média	"Três!"
Comando nº 4	Direita	Grande	Mínima	"Quatro!"
Comando nº 5	Direita	Média	Máxima	"Cinco!"
Comando nº 6	Direita	Média	Média	"Seis!"
Comando nº 7	Direita	Média	Mínima	"Sete!"
Comando nº 8	Direita	Pequena	Máxima	"Oito!"
Comando nº 9	Direita	Pequena	Média	"Nove!"
Comando nº 10	Direita	Pequena	Mínima	"Dez!"
Comando nº 11	Esquerda	Grande	Máxima	"Onze!"
Comando nº 12	Esquerda	Grande	Média	"Doze!"
Comando nº 13	Esquerda	Grande	Mínima	"Treze!"
Comando nº 14	Esquerda	Média	Máxima	"Catorze!"
Comando nº 15	Esquerda	Média	Média	"Quinze!"
Comando nº 16	Esquerda	Média	Mínima	"Dezasseis!"
Comando nº 17	Esquerda	Pequena	Máxima	"Dezassete!"
Comando nº 18	Esquerda	Pequena	Média	"Dezoito!"
Comando nº 19	Esquerda	Pequena	Mínima	"Dezanove!"

Na Tabela 3.4, podem ver-se esses 2 comandos que foram desenvolvidos bem como a respetiva função e a frase que utilizador tem que afirmar para acionar o comando.

Tabela 3.4 – Comandos de voz programados para acionar e desacionar o controlo

Comando de Voz Nº	Função	Frase que o utilizador tem que emitir para acionar o comando
Comando nº 1	Funciona como senha do sistema. Sem que a password correta seja afirmada pelo utilizador, o sistema não é desbloqueado, e como tal, não se terá acesso aos restantes comandos.	"Acionar controlo!"
Comando nº 20	É o comando que é responsável por fazer o bloqueio do sistema. Isto é, depois de não se querer fazer mais o controlo de movimento através do microfone, o utilizador tem que acionar este comando para bloquear o acesso a este tipo de controlo de movimento.	"Desacionar controlo!"

Para se gravarem todos estes comandos utilizou-se o programa EasyVR® Commander, como se pode ver na Figura 3.30 e na Figura 3.31.

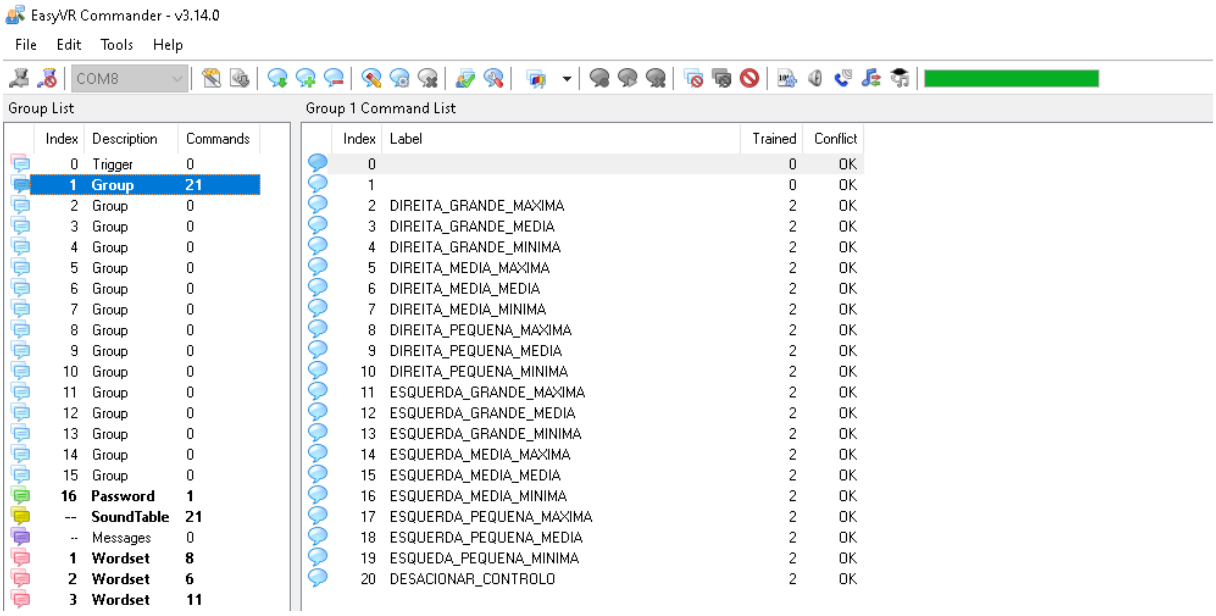


Figura 3.30 – Programação dos comandos nº 2 a 20 no programa EasyVR® Commander

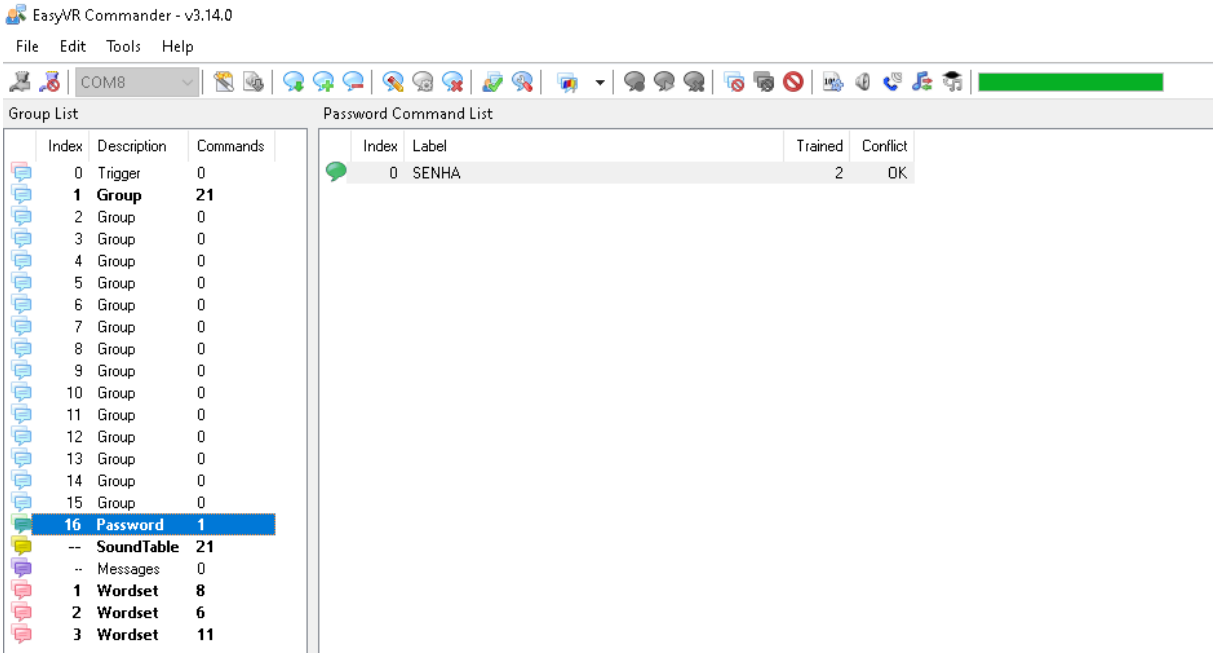


Figura 3.31 – Programação do comando nº 1(senha) no programa EasyVR® Commander

Quando se desenvolve uma interface além de se ter em consideração a interação do utilizador com o sistema, é também muito importante ter em atenção a interação do sistema com o utilizador. Dessa forma, convém sempre desenvolver uma solução que permita enviar um *feedback* ao utilizador para este ter a certeza de que as ordens por si enviadas para o sistema foram recebidas com sucesso. Tendo isto em consideração, decidiu-se que quando o utilizador enviasse uma ordem para o sistema iria receber uma mensagem através das colunas, de forma a confirmar que a mensagem foi recebida com sucesso e que iria ser executada no sistema a ordem pretendida.

Para tal, utilizou-se o programa Audacity® para gravar as mensagens de voz necessárias.

Na Tabela 3.5 podem ver todas as mensagens que foram gravadas e a que nº de comando equivale cada mensagem que foi gravada.

Tabela 3.5 – Mensagens que foram gravadas e nº de comando a que equivale cada mensagem

Comando de Voz Nº	Palavra que o utilizador tem que emitir para acionar o comando
Comando nº 1	"O controlo de movimento através do microfone será acionado!"
Comando nº 2	"Comando 2: A porta deslocar-se-á para a direita, com velocidade grande uma distância máxima!"
Comando nº 3	"Comando 3: A porta deslocar-se-á para a direita, com velocidade grande uma distância média!"
Comando nº 4	"Comando 4: A porta deslocar-se-á para a direita, com velocidade grande uma distância mínima!"
Comando nº 5	"Comando 5: A porta deslocar-se-á para a direita, com velocidade média uma distância máxima!"
Comando nº 6	"Comando 6: A porta deslocar-se-á para a direita, com velocidade média uma distância média!"
Comando nº 7	"Comando 7: A porta deslocar-se-á para a direita, com velocidade média uma distância mínima!"
Comando nº 8	"Comando 8: A porta deslocar-se-á para a direita, com velocidade pequena uma distância máxima!"
Comando nº 9	"Comando 9: A porta deslocar-se-á para a direita, com velocidade pequena uma distância média!"
Comando nº 10	"Comando 10: A porta deslocar-se-á para a direita, com velocidade pequena uma distância mínima!"
Comando nº 11	"Comando 11: A porta deslocar-se-á para a esquerda, com velocidade grande uma distância máxima!"
Comando nº 12	"Comando 12: A porta deslocar-se-á para a esquerda, com velocidade grande uma distância média!"
Comando nº 13	"Comando 13: A porta deslocar-se-á para a esquerda, com velocidade grande uma distância mínima!"
Comando nº 14	"Comando 14: A porta deslocar-se-á para a esquerda, com velocidade média uma distância máxima!"
Comando nº 15	"Comando 15: A porta deslocar-se-á para a esquerda, com velocidade média uma distância média!"
Comando nº 16	"Comando 16: A porta deslocar-se-á para a esquerda, com velocidade média uma distância mínima!"
Comando nº 17	"Comando 17: A porta deslocar-se-á para a esquerda, com velocidade pequena uma distância máxima!"
Comando nº 18	"Comando 18: A porta deslocar-se-á para a esquerda, com velocidade pequena uma distância média!"
Comando nº 19	"Comando 19: A porta deslocar-se-á para a esquerda, com velocidade pequena uma distância mínima!"
Comando nº 20	"O controlo de movimento através do microfone será desacionado!"

Depois de gravadas as mensagens, estas tiveram de ser memorizadas na placa de reconhecimento de voz. Para tal, recorreu-se novamente ao programa EasyVR® Commander, onde se realizou o *upload* dessas mesmas mensagens, como se pode verificar na Figura 3.32.

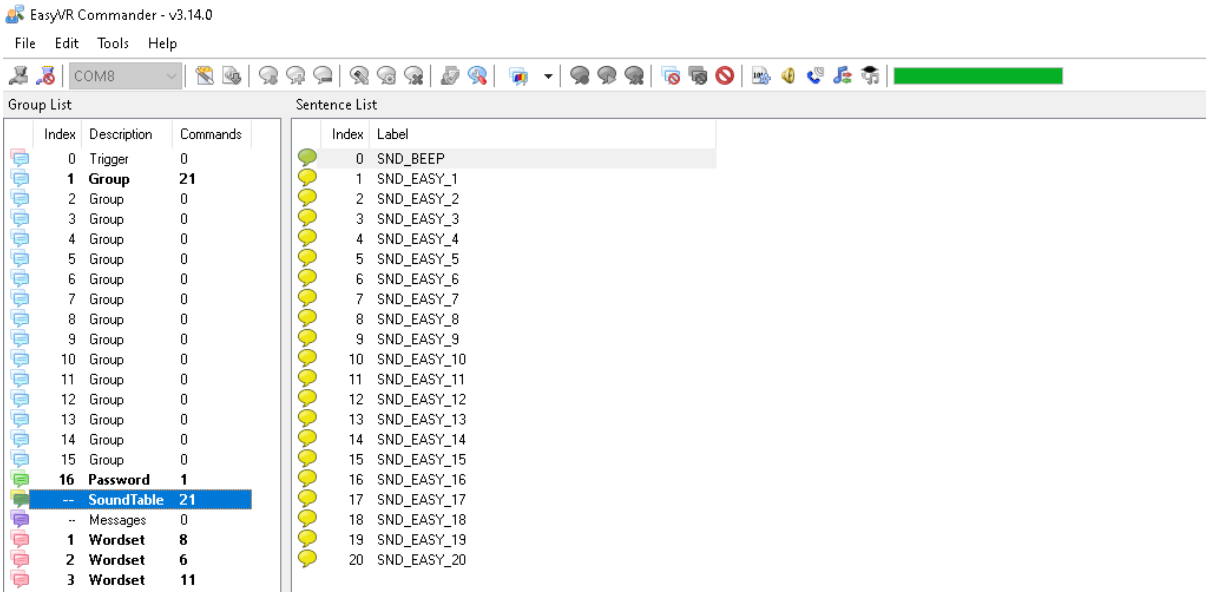


Figura 3.32 – *Upload* das mensagens gravada no programa Audacity® para a placa de reconhecimento de voz utilizada através do programa EasyVR® Commander

Após se terem programado todos os comandos necessários e de se terem gravado todas as mensagens necessárias na placa utilizada, faltava apenas gerar o programa para o Arduino® Uno, que foi utilizado para estabelecer e simplificar a ligação entre a placa EasyVR® Shield 3.0 e o PC. Para tal, recorreu-se ao programa Sensory QuickSynthesis 5® (Figura 3.33), que através da programação que foi desenvolvido no programa EasyVR® Commander permitiu gerar de forma automática todo o código de programação para programar o Arduino® UNO que estava conectado à placa de reconhecimento de voz no programa Arduino®, como se pode verificar no anexo G.

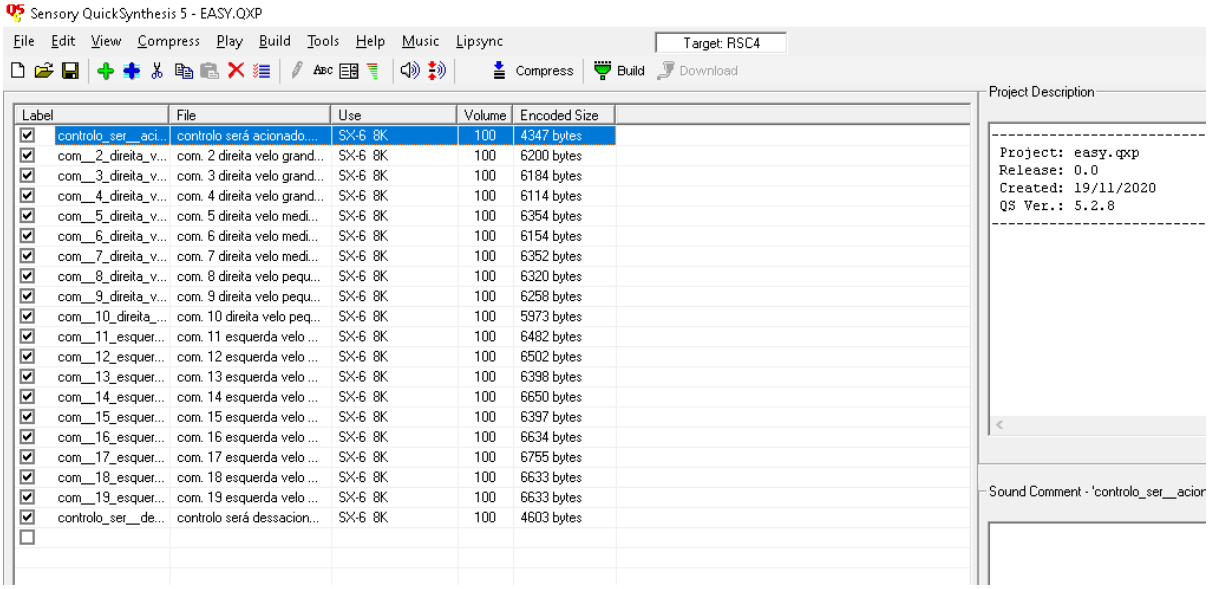


Figura 3.33 – Programação feita no programa Sensory QuickSynthesis 5® para gerar o código de programação para programar o Arduino® UNO que estava conectado à placa de reconhecimento de voz

Para se desenvolver a interface, faltava ainda fazer a programação do controlador do sistema para que esta interface funcionasse e estivesse sincronizado com o sistema desenvolvido. A ideia de programação seguida foi idêntica à da primeira interface desenvolvida e sua representação pode ser vista no diagrama de fluxo da Figura 3.24. Na Figura 3.34, pode ver-se alguma dessa programação que foi desenvolvida, e tal como nas outras duas interfaces

anteriores, também neste caso se pode ver toda programação que foi desenvolvida (anexo H) devidamente comentada e explicada.

```
// Definição do valor da variável "sentido"
if(sentido_esquerdo == HIGH)
{sentido=0;}//se o valor da variável for "0" significa que a interface está a ordenar que a porta se mova no sentido esquerdo.
if(sentido_direito == HIGH)
{sentido=1;}//se o valor da variável for "1" significa que a interface está a ordenar que a porta se mova no sentido direito.
if(sentido_direito == LOW and sentido_esquerdo == LOW)
{sentido=2;}//se o valor da variável for "2" significa que a interface está a ordenar que a porta não se mova em nenhum dos 2 sentidos.

//Definição do valor da variável "velocidade"
if(velocidade_pequena == HIGH)
{velocidade=330;} // caso o utilizador ordene que a porta se desloque com velocidade baixa, ela deslocar-se-á à velocidade de 330 mm/s.
if(velocidade_media == HIGH)
{velocidade=380;} // caso o utilizador ordene que a porta se desloque com velocidade média, ela deslocar-se-á à velocidade de 380 mm/s.
if(velocidade_grande == HIGH)
{velocidade=430;} // caso o utilizador ordene que a porta se desloque com velocidade alta, ela deslocar-se-á à velocidade de 430 mm/s.
```

Figura 3.34 – Programação do controlador desenvolvida no programa Arduino® para a interface com a a placa EasyVR® Shield 3.0

3.5 Solução final

Na Figura 3.35, pode ver-se a solução final obtida.

Figura 3.35 – Solução final obtida

Todo o sistema foi testado, de forma a garantir que a solução era válida e cumpria todos os requisitos do problema apresentado.

4 Conclusões e perspetivas de trabalho futuro

4.1 Conclusões

Graças à evolução que se tem vindo a verificar nos últimos anos, atualmente existem diversos sistemas controlados por interfaces multi-modais. Assim, neste trabalho, desenvolveu-se um sistema demonstrativo em que se utilizou uma interface multi-modal para o controlo de um eixo linear

Numa primeira fase do trabalho, foi feita uma pesquisa sobre sistemas controlados por interfaces multi-modais, dando-se especial ênfase a 4 sistemas: MBUX®, Siri®, BMW Natural Interaction® e Análise de aspetos de usabilidade em interações naturais via interfaces multimodais.

Após o estudo dos referidos sistemas reuniu-se um conjunto de informações relevantes sobre o seu funcionamento e os seus componentes. Paralelamente, foi também feito o estudo do modo de interação com o utilizador em cada um dos sistemas.

Concluída esta primeira fase de pesquisa, passou-se à análise dos requisitos imposto na apresentação do problema e definição do projeto.

Desta forma, começou por se estudar e desenvolver o sistema mecânico (sistema e estrutura que irá suportar/fixar todos os componentes necessários), de seguida o sistema eletrónico (o motor elétrico, *driver* e controlador a utilizar para se realizar o movimento linear ao longo do eixo) e por fim desenvolver algumas interfaces, mais concretamente 3, que permitam realizar o controlo de todo o sistema.

Relativamente ao primeiro passo, decidiu-se reaproveitar o sistema de uma impressora que se encontrava fora de uso. Assim, o sistema desenvolvido tem como principal componente um motor de passo NMB PM35S-048 com o respetivo eixo linear e sistema de transmissão.

Para *driver* do motor, optou-se por uma ponte H L298N, e para controlador do sistema, optou-se por um microcontrolador Arduino® DUE.

Posto isto, realizou-se o desenvolvimento das 3 interfaces do sistema. Assim, a primeira interface desenvolvida foi através do programa LabVIEW®, onde é possível escolher o sentido, a velocidade e a distância de deslocação da porta. A segunda interface desenvolvida foi através de uma câmara OpenMV® Cam M7, que possibilita realizar o controlo do sistema utilizando uma imagem, sendo que em função do sentido e da inclinação da imagem, é realizado o movimento de deslocação no eixo linear. A última interface desenvolvida foi o controlo do sistema por voz. Para tal, utilizou-se um módulo de reconhecimento voz EasyVR Shield 3.0 que permite enviar ordens para o sistema, através do microfone, mas também receber um *feedback* do mesmo através da coluna.

De forma a garantir que a solução era válida e cumpria todos os requisitos do problema apresentado, todo o sistema foi testado. Assim, rapidamente se pôde constatar que tudo estava a funcionar de acordo com o pretendido podendo-se dessa forma concluir que o objetivo foi cumprido. Devido à comodidade e facilidade de utilização que soluções deste tipo trazem para o controlo de um sistema, e tendo em consideração a evolução tecnológica que se tem vindo a

verificar, pode ainda concluir-se que soluções deste tipo tendem a ser cada vez mais utilizadas.

4.2 Trabalhos futuros

Como proposta para a continuação do estudo desenvolvido, seria interessante não só continuar a melhorar as interfaces desenvolvidas, como também a desenvolver e implementar novas interfaces que permitam controlar o sistema, como por exemplo uma interface em que através de um sensor infravermelhos o sistema pudesse realizar o movimento em função do movimento da mão do utilizador, isto é, seguindo a mesma.
Desta forma, na interface em que se utilizou a câmara, poder-se-ia tentar realizar o controlo do sistema, não através de uma imagem de referência como foi implementado, mas sim através da mão do utilizador. Outra sugestão seria, relativa à interface desenvolvida no controlo por voz. Nesta interface, em vez de se estar limitado a três níveis de velocidade e distância de deslocação, podia-se reprogramar os comandos de modo a identificar dígitos, para que se pudessem digitar a velocidade e a distância de deslocação pretendidas.
Por último, seria interessante otimizar e organizar melhor o sistema que foi desenvolvido. Poderia, por exemplo, desenvolver-se uma placa num circuito impresso, de forma a melhorar o aspeto e a robustez do circuito elétrico do sistema desenvolvido. Além disso, poder-se-ia combinar os três programas desenvolvidos (um para cada interface), num programa só, tornando a solução mais coesa e simplificada.
Ainda que a solução desenvolvida seja uma solução meramente demonstrativa e como tal, não apta para ser aplicada numa solução industrial, poder-se-ia com base no que foi desenvolvido, continuar o trabalho que foi realizado de forma a obter uma solução que dê mais garantias nomeadamente a nível de fiabilidade, para que se pudesse, por exemplo, aplicar um sistema deste tipo na abertura da porta de uma casa ou do portão de um armazém.

Referências

[1] Rodrigues, E. (2017). *Google Autodraw e as interfaces multimodais*. Acedido em 23 de Fevereiro de 2021. https://pt.linkedin.com/pulse/google-autodraw-e-interfaces-multimodais-eli-rodrigues

[2] Freitas, A. (2020). *Novo MBUX: a assistente de luxo da Mercedes*. Acedido em 24 de Fevereiro de 2021. https://www.e-konomista.pt/novo-sistema-mercedes-mbux/

[3] Soc. Com. C. Santos. (2020). *Controle De Voz: "Hey Mercedes" Obedece A Cada Ordem*. Acedido em 24 de Fevereiro de 2021. https://soccomcsantos.tumblr.com/post/171049178936/controle-de-voz-hey-mercedes-obedece-a-cada

[4] Rosolen, F. (2020). *Novos controles touchscreen da Mercedes eliminam 27 botões físicos*. Acedido em 24 de Fevereiro de 2021.https://mundoconectado.com.br/noticias/v/14476/novos-controles-touchscreen-da-mercedes-eliminam-27-botoes-fisicos

[5] Costa, M.B. (2020). *O que é a Siri? Como funciona a assistente virtual da Apple*. Acedido em 24 de Fevereiro de 2021. https://br.financas.yahoo.com/noticias/o-que-%C3%A9-siri-como-180000379.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAIzcrkQ-kVd1Hwhyma1j1s1Kz1yBKUKQa5UXKwVHX9t_uxK2d_InFsS4_NkQPDJItHc3gba1_u98aNI1XnAY-PapoTPMEKSf32ZeqtivybXkUVpardTQ-E2Xztd5Tv9Tza87ps0X5rBqHwcSMuiFswer5bJ8Y1JRj8ELV13k1Hi3

[6] Apple. (n.d.). *Siri*. Acedido em 24 de Fevereiro de 2021. https://www.apple.com/br/siri/

[7] O'Boyle, B. (2020). *O que é Siri e como funciona o Siri?* Acedido em 24 de Fevereiro de 2021. https://www.pocket-lint.com/pt-br/aplicativos/noticias/apple/112346-o-que-e-o-assistente-de-voz-pessoal-de-siri-apple-explicou

[8] Carvalho, I. (2019). *BMW anuncia sistema que reconhece voz, gestos e olhares do motorista*. Acedido em 24 de Fevereiro de 2021. https://www.startse.com/noticia/nova-economia/bmw-anuncia-sistema-que-reconhece-voz-gestos-e-olhares-do-motorista

[9] Boeriu, H. (2019). *BMW Natural Interaction introduced at the Mobile World Congress 2019*. Acedido em 24 de Fevereiro de 2021. https://www.bmwblog.com/2019/02/25/bmw-natural-interaction-introduced-at-the-mobile-world-congress-2019/

[10] Duarte, A. (2017). *Sistema de controlo por gestos: como tudo funciona*. Acedido em 24 de Fevereiro de 2021. https://automais.autosport.pt/noticias/sistema-controlo-gestos-tudo-funciona/

[11] Cossio, L. P. (2014). *Análise de aspectos de usabilidade em interações naturais via interfaces multimodais*. Dissertação de Mestrado, Universidade Católica do Rio Grande do Sul, Porto Alegre, Brasil. http://tede2.pucrs.br/tede2/bitstream/tede/5980/2/468136%20-%20Texto%20Completo.pdf

[12] Koyanagi, F. (2017). *Ligando um motor de passo de sucata com Arduino*. Acedido em 3 de Fevereiro de 2021. http://www.animobygames.com.br/download/Motor-de-passo-sucata.pdf

[13] Multilógica-Shop. (n.d.). *Arduino Due*. Acedido em 5 de Fevereiro de 2021. https://multilogica-shop.com/arduino-due

[14] Thomsen, A. (2014). *Arduino Due - o Arduino com processador ARM de 32 bits*. Acedido em 5 de Fevereiro de 2021. https://www.filipeflop.com/blog/arduino-due-atmel-arm-32bits/

[15] Cardoso, D. (2017). *Driver motor com Ponte H L298N – Controlando Motor DC com Arduino*. Acedido em 5 de Fevereiro de 2021. https://portal.vidadesilicio.com.br/driver-motor-com-ponte-h-l298n/

[16] Multilógica-Shop. (n.d.). *Shield para reconhecimento de voz EasyVR*. Acedido em 17 de Fevereiro de 2021. https://multilogica-shop.com/node/921

[17] NMB. (n.d.). *Standard PM Step Motors: PM35S-048-HHC6*. Acedido em 27 de Fevereiro de 2020. https://media.digikey.com/pdf/Data%20Sheets/NMB-MAT/PM35S-048-HHC6.pdf

ANEXO A: Especificação Técnica do Motor utilizado [17]

Standard PM Step Motors

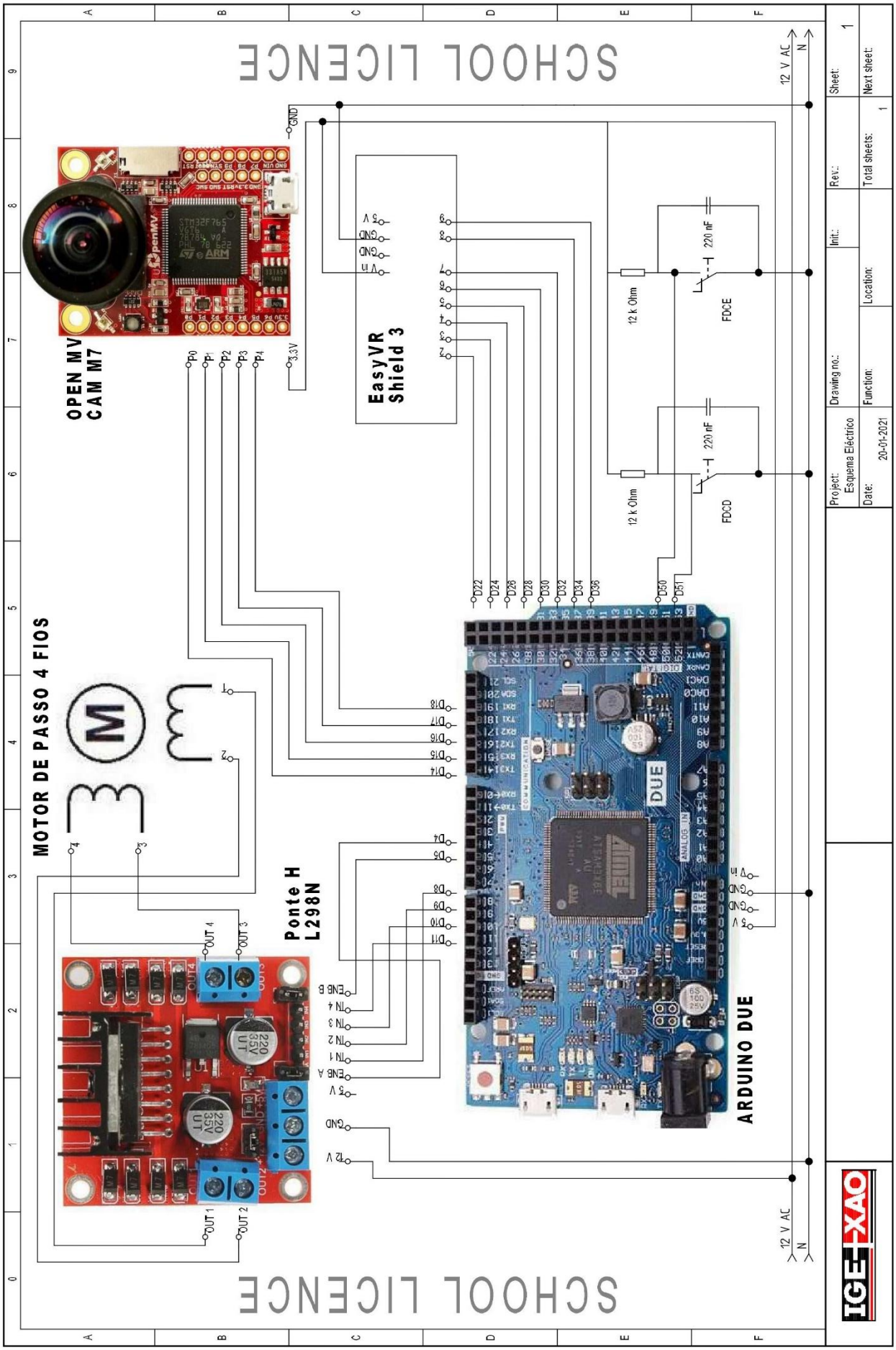
PM35S-048-HHC6

Step Angle: 7.5°

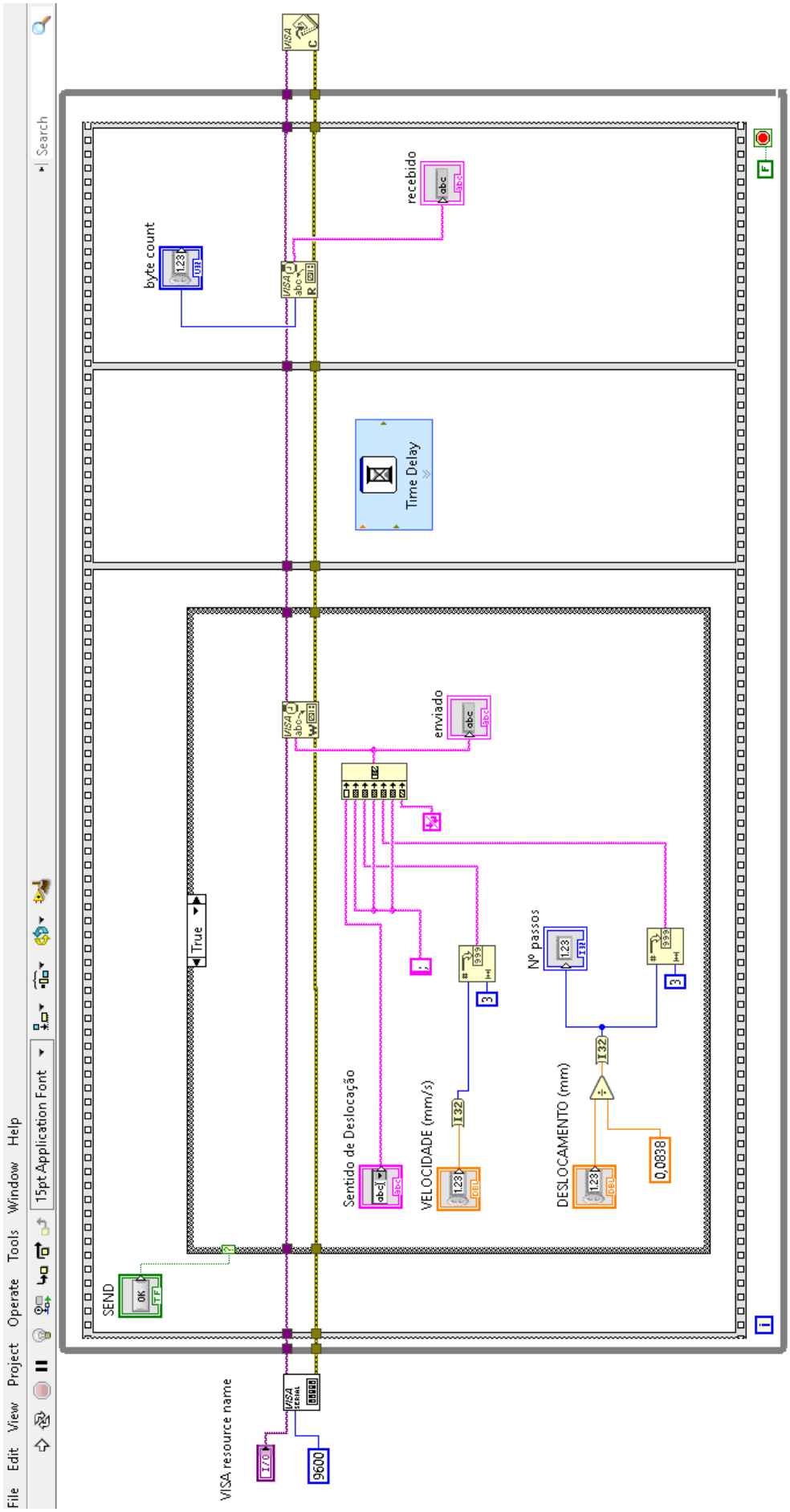
Model Specifications

Model:	PM35S-048-HHC6
Shaft Length:	10mm
Wire Length:	300mm fly lead (no connector)
Wire Holder:	90 deg. Left
Front Plate:	FPH
Electrical:	24V, 600mA
	Bipolar Constant Current
	5.5 ohms, MS70M

ANEXO B: Esquema elétrico da solução desenvolvida



ANEXO C: Programação desenvolvida no programa LabVIEW®



ANEXO D: Programação do controlador desenvolvida no programa Arduino® para a interface com o LabView®

```
#include <Stepper.h>
const int stepsPerRevolution = 48; // nº de passos realizados numa volta completa pelo motor de passo
Stepper myStepper(stepsPerRevolution,8,9,10,11); // sequência de alimentação do motor de passo

//Declaração de Variáveis
int enrolamentoA=4;
int enrolamentoB=5;
int fim_curso_esquerdo=51;
int fim_curso_direito=50;
int led_esq=53;
int led_dir=52;
int sentido;
int velocidade;
int distancia;

void setup() {
Serial.begin(9600); // Taxa de transferência em bits por segundo (baud rate) para transmissão serial

//Definição das variáveis de entrada e de saída
pinMode(enrolamentoA,OUTPUT);
pinMode(enrolamentoB,OUTPUT);
pinMode(led_esq,OUTPUT);
pinMode(led_dir,OUTPUT);
pinMode(fim_curso_esquerdo,INPUT);
pinMode(fim_curso_direito,INPUT);

// Função (“attachInterrupt”), responsável pelo devido funcionamento dos Fins de Curso do eixo linear
attachInterrupt(digitalPinToInterrupt(51),fimcursoesquerdo,RISING); // Fim de curso Esquerdo:
//Quando o fim de curso esquerdo é acionado, a entrada digital nº 51, sofre uma transição de LOW para HIGH, o que faz com que o programa que está a correr seja interrompido: função (“attachInterrupt”).
attachInterrupt(digitalPinToInterrupt(50),fimcursodireito,RISING);  //  Fim de curso Direito:
//Quando o fim de curso direito é acionado, a entrada digital nº 50, sofre uma transição de LOW para HIGH, o que faz com que o programa que está a correr seja interrompido função (“attachInterrupt”).
}

void loop() {
if (Serial.available()) // Quando existe uma receção de informação através da porta serial, o programa avança para o próximo passo.
{
fim_curso_direito=digitalRead(50); // leitura dos valores da entrada digital nº 50. Se o valor for "HIGH" ou "1", o fim de curso direito está acionado, se for "LOW" ou "0", não está acionado.
```

```
fim_curso_esquerdo=digitalRead(51); // leitura dos valores da entrada digital nº 51. Se o valor for "HIGH" ou "1", o fim de curso esquerdo está acionado, se for "LOW" ou "0", não está acionado.

//Através da porta Serial, é recebida uma mensagem, mensagem essa que é enviada pelo utilizador pelo programa LabView.
// Essa mensagem é composta por 3 números separados por ";", ou seja: (nº;nº;nº).
sentido=Serial.parseInt(); // o primeiro nº recebido na mensagem indica o sentido para o qual a porta do nosso sistema se irá deslocar.
velocidade=Serial.parseInt();// o segundo nº recebido na mensagem indica a velocidade com que a porta do nosso sistema se irá deslocar.
distancia=Serial.parseInt();// o terceiro nº recebido na mensagem indica o número de passos que o motor irá realizar, nº que está diretamente relacionado com a distância que se irá percorrer. ´
delay(1000);
Serial.print(sentido);
Serial.print(";");
Serial.print(velocidade);
Serial.print(";");
Serial.print(distancia);
Serial.print(";");
Serial.println();
Serial.parseInt();
{
 if(sentido = = 1 and fim_curso_direito = = 0) // DESLOCAMENTO PARA A DIREITA: Só acontece caso o valor do sentido seja 1 e o fim de curso direito não esteja acionado.
 {digitalWrite(led_esq,LOW); // Caso o fim de curso esquerdo tenha sido acionado o led acende e quando se faz o movimento para a direita ele terá de apagar pois deixa de estar acionado.
  digitalWrite(enrolamentoA,HIGH); // Alimentação do enrolamento A do motor de passo
  digitalWrite(enrolamentoB,HIGH); // Alimentação do enrolamento B do motor de passo
  myStepper.setSpeed(velocidade); // Indicação da velocidade de trabalho do motor
  myStepper.step(-distancia); // Indicação do número de passos que o motor irá realizar. Este nº multiplicado por 0.0838 mm indica-nos a distância que a porta do nosso sistema irá percorrer.
  //Nota: Como podem reparar, o valor introduzido leva o sinal menos (-) para que o motor se desloca no sentido que permite a porta mover-se para a direita.
  }
  if(sentido = = 0 and fim_curso_esquerdo = = 0) // DESLOCAMENTO PARA A ESQUERDA: Só acontece caso o valor do sentido seja 0 e o fim de curso esquerdo não esteja acionado.
 {digitalWrite(led_dir,LOW); // Caso o fim de curso direito tenha sido acionado o led acende e quando se faz o movimento para a esquerda ele terá de apagar pois deixa de estar acionado.
  digitalWrite(enrolamentoA,HIGH); // Alimentação do enrolamento A do motor de passo
  digitalWrite(enrolamentoB,HIGH); // Alimentação do enrolamento B do motor de passo
  myStepper.setSpeed(velocidade); // Indicação da velocidade de trabalho do motor
  myStepper.step(distancia); // Indicação do número de passos que o motor irá realizar. Este nº multiplicado por 0.0838 mm indica-nos a distância que a porta do nosso sistema irá percorrer.
  }
}
}
}
```

```
void fimcursoesquerdo() { //Quando o fim de curso esquerdo é acionado, o programa é interrompido devido à função (“attachInterrupt”), tal como já foi acima explicado, e acontece o seguinte:
digitalWrite(led_esq,HIGH); // Acende o led que indica que o fim de curso esquerdo foi acionado.
digitalWrite(enrolamentoA,LOW); // Corte da alimentação do enrolamento A do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.
digitalWrite(enrolamentoB,LOW);} // Corte da alimentação do enrolamento B do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.

void fimcursodireito() {//Quando o fim de curso direito é acionado, o programa é interrompido devido à função (“attachInterrupt”), tal como já foi acima explicado, e acontece o seguinte:
digitalWrite(led_dir,HIGH); // Acende o led que indica que o fim de curso direito foi acionado.
digitalWrite(enrolamentoA,LOW);// Corte da alimentação do enrolamento A do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.
digitalWrite(enrolamentoB,LOW);}// Corte da alimentação do enrolamento B do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.
```

ANEXO E: Programação desenvolvida no programa OpenMV IDE® para programar a placa OpenMV® Cam M7

```
#Interface através da Câmara OpenMV Cam M7
import sensor, image, time, math, pyb
from pyb import Pin

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA) # we run out of memory if the resolution
is much bigger...
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False)  # must turn this off to prevent image
washout...
sensor.set_auto_whitebal(False)  # must turn this off to prevent image
washout...

#definição das variáveis de saída
esquerda=Pin('P0', Pin.OUT_PP) # variável que ordena o deslocamento da
porta no sentido esqurdo
direita=Pin('P1', Pin.OUT_PP) # variável que ordena o deslocamento da porta
no sentido direito
velocidade_baixa=Pin('P2', Pin.OUT_PP) # variável que ordena o deslocamento
da porta a velocidade baixa
velocidade_media=Pin('P3', Pin.OUT_PP) # variável que ordena o deslocamento
da porta a velocidade média
velocidade_alta=Pin('P4', Pin.OUT_PP) # variável que ordena o deslocamento
da porta a velocidade alta

clock = time.clock()

rotacao=0 # definição do valor inicial da variável rotação
# Note! Unlike find_qrcodes the find_apriltags method does not need lens
correction on the image to work.

# The apriltag code supports up to 6 tag families which can be processed at
the same time.
# Returned tag objects will have their tag family and id within the tag
family.

tag_families = 0 # definição do valor inicial da variável tag_families

tag_families |= image.TAG36H11 # comment out to disable this family
(default family)


# What's the difference between tag families? Well, for example, the
TAG16H5 family is effectively
# a 4x4 square tag. So, this means it can be seen at a longer distance than
a TAG36H11 tag which
# is a 6x6 square tag. However, the lower H value (H5 versus H11) means
that the false positve
# rate for the 4x4 tag is much, much, much, higher than the 6x6 tag. So,
unless you have a
# reason to use the other tags families just use TAG36H11 which is the
default family.

#def family_name(tag):
  #  if(tag.family() == image.TAG36H11):
  #      return "TAG36H11"
```

```
while(True):
    clock.tick()
    img = sensor.snapshot() # definição da variável img, que é a captação
da imagem em cada instante.

    for tag in img.find_apriltags(families=tag_families): # defaults to
TAG36H11 without "families".
        img.draw_rectangle(tag.rect(), color = (255, 0, 0)) # desenho de um
rectângulo em volta da imagem captada
        img.draw_cross(tag.cx(), tag.cy(), color = (0, 255, 0)) # desenho
de uma cruz no centro da imagem captada
        rotacao=(180*tag.rotation())/math.pi # calculo que permite saber
qual a inclinação da imagem captada
        print("Rotação:", rotacao,"Graus") # indicação da inclinação que a
imagem no momento em que foi captada
#O objetivo é que em função da inclinação da imagem captada (Tag36H11),
seja ordenado através das variáveis de saída o seguinte:
#O sentido da inclinação (esquerdo ou direito), define o sentido de
deslocação da porta.
#Os graus de rotação, definem se a porta se irá descolar com maior ou menor
velocidade, sendo que quanto maior for a inclinação maior será também a
velocidade.
    #SENTIDO ESQUERDO
        #VELOCIDADE BAIXA
        if(10 < rotacao < 30):
            print("esquerda baixa")
            esquerda.value(1)
            velocidade_baixa.value(1)
            direita.value(0)
            velocidade_media.value(0)
            velocidade_alta.value(0)
        #VELOCIDADE MEDIA
        if(30 < rotacao < 50):
            print("esquerda media")
            esquerda.value(1)
            velocidade_media.value(1)
            direita.value(0)
            velocidade_baixa.value(0)
            velocidade_alta.value(0)
        #VELOCIDADE ALTA
        if(50 < rotacao < 70):
            print("esqueda alta")
            esquerda.value(1)
            velocidade_alta.value(1)
            direita.value(0)
            velocidade_baixa.value(0)
            velocidade_media.value(0)
    #SENTIDO DIREITO
        #VELOCIDADE BAIXA
        if(330 < rotacao < 350):
            print("direita baixa")
            velocidade_baixa.value(1)
            direita.value(1)
            esquerda.value(0)
            velocidade_media.value(0)
            velocidade_alta.value(0)
        #VELOCIDADE MEDIA
        if(310 < rotacao < 330):
            print("direita media")
            direita.value(1)
            velocidade_media.value(1)
            esquerda.value(0)
            velocidade_baixa.value(0)
            velocidade_alta.value(0)
```

```
    #VELOCIDADE ALTA
    if(290 < rotacao < 310):
        print("direita alta")
        direita.value(1)
        velocidade_alta.value(1)
        esquerda.value(0)
        velocidade_baixa.value(0)
        velocidade_media.value(0)
#INDICAÇÃO DE ULTRAPASSAGEM DOS LIMITES DE INCLINAÇÃO
    if(70 < rotacao < 290):
        print("fora dos limites")
        esquerda.value(0)
        direita.value(0)
        velocidade_media.value(0)
        velocidade_baixa.value(0)
        velocidade_alta.value(0)

    pyb.delay(750)
    esquerda.value(0)
    direita.value(0)
    velocidade_media.value(0)
    velocidade_baixa.value(0)
    velocidade_alta.value(0)
    #pyb.delay(2500)
#print(clock.fps())
```

ANEXO F: Programação do controlador desenvolvida no programa Arduino® para a interface com a placa OpenMV® Cam M7

```
#include <Stepper.h>
const int stepsPerRevolution = 48; // nº de passos realizados numa volta completa pelo motor de passo
Stepper myStepper(stepsPerRevolution,8,9,10,11); // sequência de alimentação do motor de passo

//Declaração de Variáveis
int enrolamentoA=4;
int enrolamentoB=5;
int fim_curso_esquerdo=51;
int fim_curso_direito=50;
int led_esq=53;
int led_dir=52;
int velocidade;
int sentido;
int distancia;

int sentido_esquerdo=14;
int sentido_direito=15;
int velocidade_baixa=16;
int velocidade_media=17;
int velocidade_alta=18;

void setup() {
Serial.begin(9600); // Taxa de transferência em bits por segundo (baud rate) para transmissão serial

//Definição das variáveis de entrada e de saída
pinMode(enrolamentoA,OUTPUT);
pinMode(enrolamentoB,OUTPUT);
pinMode(led_esq,OUTPUT);
pinMode(led_dir,OUTPUT);
pinMode(fim_curso_esquerdo,INPUT);
pinMode(fim_curso_direito,INPUT);

pinMode(sentido_esquerdo,INPUT);
pinMode(sentido_direito,INPUT);
pinMode(velocidade_baixa,INPUT);
pinMode(velocidade_media,INPUT);
pinMode(velocidade_alta,INPUT);

// Função (“attachInterrupt”), responsável pelo devido funcionamento dos Fins de Curso do eixo linear
attachInterrupt(digitalPinToInterrupt(51),fimcursoesquerdo,RISING); // Fim de curso Esquerdo:
// Quando o fim de curso esquerdo é acionado, a entrada digital nº 51, sofre uma transição de LOW para HIGH, o que faz com que o programa que está a correr seja interrompido: função (“attachInterrupt”).
attachInterrupt(digitalPinToInterrupt(50),fimcursodireito,RISING); //  Fim de curso Direito:
```

```
// Quando o fim de curso direito é acionado, a entrada digital nº 50, sofre uma transição de LOW para HIGH, o que faz com que o programa que está a correr seja interrompido: função (“attachInterrupt”).
}

void loop() {
// Neste caso, os valores são recebidos pela interface, que é a câmara (Open MV Cam M7).
sentido_esquerdo=digitalRead(14); // leitura dos valores da entrada digital nº 14. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova no sentido esquerdo.
sentido_direito=digitalRead(15); // leitura dos valores da entrada digital nº 15. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova no sentido direito.
fim_curso_esquerdo=digitalRead(51); // leitura dos valores da entrada digital nº 51. Se o valor for "HIGH" ou "1", o fim de curso esquerdo está acionado, se for "LOW" ou "0", não está acionado.
fim_curso_direito=digitalRead(50); // leitura dos valores da entrada digital nº 50. Se o valor for "HIGH" ou "1", o fim de curso direito está acionado, se for "LOW" ou "0", não está acionado.

// Definição do valor da variável "sentido"
if(sentido_esquerdo = = HIGH)
{sentido=0;} // se o valor da variável for "0" significa que a interface está a ordenar que a porta se mova no sentido esquerdo.
if(sentido_direito = = HIGH)
{sentido=1;} // se o valor da variável for "1" significa que a interface está a ordenar que a porta se mova no sentido direito.
if(sentido_direito = = LOW and sentido_esquerdo = = LOW)
{sentido=2;} // se o valor da variável for "2" significa que a interface está a ordenar que a porta não se mova em nenhum dos 2 sentidos.

 while(sentido = = 1 and fim_curso_direito == 0) // DESLOCAMENTO PARA A DIREITA:
 // Enquanto o valor da variável "sentido" for "1" e o fim de curso direto não for acionado, a porta deslocar-se-á para a direita
 {digitalWrite(led_esq,LOW); // Caso o fim de curso esquerdo tenha sido acionado o led acende e quando se faz o movimento para a direita ele terá de apagar pois deixa de estar acionado.
  digitalWrite(enrolamentoA,HIGH); // Alimentação do enrolamento A do motor de passo
  digitalWrite(enrolamentoB,HIGH); // Alimentação do enrolamento B do motor de passo

//Definição da velocidade de deslocamento da porta
  velocidade_baixa=digitalRead(16); // leitura dos valores da entrada digital nº 16. Se o valor for "HIGH" ou "1", significa que a porta deslocar-se-á com velocidade baixa.
  velocidade_media=digitalRead(17); // leitura dos valores da entrada digital nº 17. Se o valor for "HIGH" ou "1", significa que a porta deslocar-se-á com velocidade média.
  velocidade_alta=digitalRead(18); // leitura dos valores da entrada digital nº 18. Se o valor for "HIGH" ou "1", significa que a porta deslocar-se-á com velocidade alta.

// Definição do valor da variável "velocidade"
  if(velocidade_baixa = = HIGH)
  {velocidade=300;} // caso o utilizador ordene que a porta se desloque com velocidade baixa, ela deslocar-se-á à velocidade de 300 mm/s.
  if(velocidade_media = = HIGH)
```

```
  {velocidade=350;} // caso o utilizador ordene que a porta se desloque com velocidade média, ela deslocar-se-á à velocidade de 350 mm/s.
  if(velocidade_alta = = HIGH)
  {velocidade=400;} // caso o utilizador ordene que a porta se desloque com velocidade alta, ela deslocar-se-á à velocidade de 400 mm/s.
// Nota: o valor da velocidade encontra-se entre o intervalo de valores de 300 a 400 mm/s pois é o mais adequado, tal como foi concluído através de diversas tentativas/experiências.

  myStepper.setSpeed(velocidade); // Indicação da velocidade de trabalho do motor
  myStepper.step(-100); // Indicação do número de passos que o motor irá realizar. Este nº multiplicado por 0.0838 mm indica-nos a distância que a porta do nosso sistema irá percorrer.
  // Nota: Como podem reparar, o valor introduzido leva o sinal menos (-) para que o motor se desloca no sentido que permite a porta mover-se para a direita.

// Embora possa parecer estranho termos que fazer novamente a leitura dos valores de entrada que fizemos no início do ciclo (void loop), é totalmente lógico e propositado.
// Quando entramos neste ciclo while, o objetivo é que a porta se desloque sempre para a direita enquanto o utilizador desejar.
// Desta forma, antes de terminarmos este ciclo, temos que ler as variáveis que o utilizador nos envia através da interface, para saber se ele quer ou não que o ciclo se continue a realizar.
// Neste ciclo, o deslocamento da porta é para a direita, ou seja, fazemos a leitura dos valores das variáveis de entrada, sendo que os que importam são os seguintes:
// O valor da variável "sentido_direito" que tem que ter o valor de "1" caso o utilizador deseje que a porta se continue a deslocar para a direita
// E o valor da variável "fim_curso_direito", pois caso o fim de curso direito esteja acionado significa que a porta não se pode continuar a deslocar para a direita.

  sentido_esquerdo=digitalRead(14); // releitura dos valores da entrada digital nº 14.
  sentido_direito=digitalRead(15); // releitura dos valores da entrada digital nº 15.
  fim_curso_esquerdo=digitalRead(51); // releitura dos valores da entrada digital nº 51.
  fim_curso_direito=digitalRead(50); // releitura dos valores da entrada digital nº 50.

//Redefinição do valor da variável "sentido"
  if(sentido_esquerdo = = HIGH)
  {sentido=0;} // se o valor da variável for "0" significa que a interface está a ordenar que a porta se mova no sentido esquerdo, e como tal o ciclo while vai terminar.
  if(sentido_direito = = HIGH)
  {sentido=1;} // se o valor da variável for "1" significa que a interface está a ordenar que a porta se mova no sentido direito, e como tal o ciclo while poderá continuar.
  if(sentido_direito = = LOW and sentido_esquerdo = = LOW)
  {sentido=2;} // se o valor da variável for "2" significa que a interface está a ordenar que a porta não se mova em nenhum dos 2 sentidos, e como tal o ciclo while vai terminar.
 }

 while(sentido = = 0 and fim_curso_esquerdo = = 0) // DESLOCAMENTO PARA A ESQUERDA:
 // Enquanto o valor da variável "sentido" for "0" e o fim de curso esquerdo não for acionado, a porta deslocar-se-á para a esquerda
 {digitalWrite(led_dir,LOW); // Caso o fim de curso direito tenha sido acionado o led acende e quando se faz o movimento para a esquerda ele terá de apagar pois deixa de estar acionado.
  digitalWrite(enrolamentoA,HIGH); // Alimentação do enrolamento A do motor de passo
  digitalWrite(enrolamentoB,HIGH); // Alimentação do enrolamento B do motor de passo
```

```
//Definição da velocidade de deslocamento da porta
  velocidade_baixa=digitalRead(16); // leitura dos valores da entrada digital nº 16. Se o valor for "HIGH" ou "1", significa que a porta deslocar-se-á com velocidade baixa.
  velocidade_media=digitalRead(17); // leitura dos valores da entrada digital nº 17. Se o valor for "HIGH" ou "1", significa que a porta deslocar-se-á com velocidade média.
  velocidade_alta=digitalRead(18); // leitura dos valores da entrada digital nº 18. Se o valor for "HIGH" ou "1", significa que a porta deslocar-se-á com velocidade alta.

// Definição do valor da variável "velocidade"
  if(velocidade_baixa = = HIGH)
  {velocidade=300;} // caso o utilizador ordene que a porta se desloque com velocidade baixa, ela deslocar-se-á à velocidade de 300 mm/s.
  if(velocidade_media = = HIGH)
  {velocidade=350;} // caso o utilizador ordene que a porta se desloque com velocidade média, ela deslocar-se-á à velocidade de 350 mm/s.
  if(velocidade_alta = = HIGH)
  {velocidade=400;} // caso o utilizador ordene que a porta se desloque com velocidade alta, ela deslocar-se-á à velocidade de 400 mm/s.
//Nota: o valor da velocidade encontra-se entre o intervalo de valores de 300 a 400 mm/s pois é o mais adequado, tal como foi concluído através de diversas tentativas/experiências.

  myStepper.setSpeed(velocidade); // Indicação da velocidade de trabalho do motor
  myStepper.step(100); // Indicação do número de passos que o motor irá realizar. Este nº multiplicado por 0.0838 mm indica-nos a distância que a porta do nosso sistema irá percorrer.

// Embora possa parecer estranho termos que fazer novamente a leitura dos valores de entrada que fizemos no início do ciclo (void loop), é totalmente lógico e propositado.
// Quando entramos neste ciclo while, o objetivo é que a porta se desloque sempre para a esquerda enquanto o utilizador desejar.
// Desta forma, antes de terminarmos este ciclo, temos que ler as variáveis que o utilizador nos envia através da interface, para saber se ele quer ou não que o ciclo se continue a realizar.
// Neste ciclo, o deslocamento da porta é para a esquerda, ou seja, fazemos a leitura dos valores das variáveis de entrada, sendo que os que importam são os seguintes:
// O valor da variável "sentido_esquerdo" que tem que ter o valor de "1" caso o utilizador deseje que a porta se continue a deslocar para a esquerda
// E o valor da variável "fim_curso_esquerdo", pois caso o fim de curso esquerdo esteja acionado significa que a porta não se pode continuar a deslocar para a esquerda.

  sentido_esquerdo=digitalRead(14); // releitura dos valores da entrada digital nº 14.
  sentido_direito=digitalRead(15); // releitura dos valores da entrada digital nº 15.
  fim_curso_esquerdo=digitalRead(51); // releitura dos valores da entrada digital nº 51.
  fim_curso_direito=digitalRead(50); // releitura dos valores da entrada digital nº 50.

//Redefinição do valor da variável "sentido"
  if(sentido_esquerdo = = HIGH)
  {sentido=0;} // se o valor da variável for "0" significa que a interface está a ordenar que a porta se mova no sentido esquerdo, e como tal o ciclo while poderá continuar.
  if(sentido_direito = = HIGH)
  {sentido=1;} // se o valor da variável for "1" significa que a interface está a ordenar que a porta se mova no sentido direito, e como tal o ciclo while vai terminar.
  if(sentido_direito = = LOW and sentido_esquerdo = = LOW)
```

```
  {sentido=2;} // se o valor da variável for "2" significa que a interface está a ordenar que a porta não se mova em nenhum dos 2 sentidos, e como tal o ciclo while vai terminar.
 }
}
void fimcursoesquerdo() { // Quando o fim de curso esquerdo é acionado, o programa é interrumpido devido à função (attachInterrupt), tal como já foi acima explicado, e acontece o seguinte:
digitalWrite(led_esq,HIGH); // Acende o led que indica que o fim de curso esquerdo foi acionado.
digitalWrite(enrolamentoA,LOW); // Corte da alimentação do enrolamento A do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.
digitalWrite(enrolamentoB,LOW);} // Corte da alimentação do enrolamento B do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.

void fimcursodireito() { // Quando o fim de curso direito é acionado, o programa é interrumpido devido à função (attachInterrupt), tal como já foi acima explicado, e acontece o seguinte:
digitalWrite(led_dir,HIGH); // Acende o led que indica que o fim de curso direito foi acionado.
digitalWrite(enrolamentoA,LOW); // Corte da alimentação do enrolamento A do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.
digitalWrite(enrolamentoB,LOW);} // Corte da alimentação do enrolamento B do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.
```

ANEXO G: Programação gerada automaticamente pelo programa Sensory QuickSynthesis 5® no programa Arduino® para programar a placa de reconhecimento de voz EasyVR Shield 3.0

```
//A placa EasyVR Shield 3.0 foi programada no programa EasyVR Commander, sendo que foi através desse programa que a programção para o arduino foi gerada automaticamente.
//Desta forma, convém aqui salientar antres demais que o programa não foi feito por mim na totalidade, apenas foi ajustado para que realizasse as tarefas que eu pretendia.
//Como tal, apenas vou comentar a programação realizada por mim.
#include "Arduino.h"
#if !defined(SERIAL_PORT_MONITOR)
  #error "Arduino version not supported. Please update your IDE to the latest version."
#endif

#if defined(__SAMD21G18A__)
  // Shield Jumper on HW (for Zero, use Programming Port)
  #define port SERIAL_PORT_HARDWARE
  #define pcSerial SERIAL_PORT_MONITOR
#elif defined(SERIAL_PORT_USBVIRTUAL)
  // Shield Jumper on HW (for Leonardo and Due, use Native Port)
  #define port SERIAL_PORT_HARDWARE
  #define pcSerial SERIAL_PORT_USBVIRTUAL
#else
  // Shield Jumper on SW (using pins 12/13 or 8/9 as RX/TX)
  #include "SoftwareSerial.h"
  SoftwareSerial port(12, 13);
  #define pcSerial SERIAL_PORT_MONITOR
#endif

#include "EasyVR.h"

EasyVR easyvr(port);

//Declaração de Variáveis
int sentido_esquerdo=2;
int sentido_direito=3;
int velocidade_pequena=4;
int velocidade_media=5;
int velocidade_grande=6;
int distancia_minima=7;
int distancia_media=8;
int distancia_maxima=9;

//Definição dos grupos e respetivos comandos
enum Groups
{
  GROUP_1  = 1,//o grupo 1 irá conter todos os comandos que à excessão da senha. Apenas estará acessível ao utilizador caso a senha seja digitalizada.
  GROUP_16 = 16,//o grupo 16 apenas é composto por um comando que é a senha. A senha é o que fará com que ocorra a transição para o Grupo 1, onde estarão todos os outros comandos.
};
```

```
enum Group1
{
  G1_DIREITA_GRANDE_MAXIMA = 2,//Comando nº 2 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 2!".
  G1_DIREITA_GRANDE_MEDIA = 3,//Comando nº 3 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 3!".
  G1_DIREITA_GRANDE_MINIMA = 4,//Comando nº 4 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 4!".
  G1_DIREITA_MEDIA_MAXIMA = 5,//Comando nº 5 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 5!".
  G1_DIREITA_MEDIA_MEDIA = 6,//Comando nº 6 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 6!".
  G1_DIREITA_MEDIA_MINIMA = 7,//Comando nº 7 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 7!".
  G1_DIREITA_PEQUENA_MAXIMA = 8,//Comando nº 8 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 8!".
  G1_DIREITA_PEQUENA_MEDIA = 9,//Comando nº 9 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 9!".
  G1_DIREITA_PEQUENA_MINIMA = 10,//Comando nº 10 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 10!".
  G1_ESQUERDA_GRANDE_MAXIMA = 11,//Comando nº 11 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 11!".
  G1_ESQUERDA_GRANDE_MEDIA = 12,//Comando nº 12 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 12!".
  G1_ESQUERDA_GRANDE_MINIMA = 13,//Comando nº 13 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 13!".
  G1_ESQUERDA_MEDIA_MAXIMA = 14,//Comando nº 14 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 14!".
  G1_ESQUERDA_MEDIA_MEDIA = 15,//Comando nº 15 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 15!".
  G1_ESQUERDA_MEDIA_MINIMA = 16,//Comando nº 16 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 16!".
  G1_ESQUERDA_PEQUENA_MAXIMA = 17,//Comando nº 17 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 17!".
  G1_ESQUERDA_PEQUENA_MEDIA = 18,//Comando nº 18 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 18!".
  G1_ESQUEDA_PEQUENA_MINIMA = 19,//Comando nº 19 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 19!".
  G1_DESACIONAR_CONTROLO = 20,//Comando nº 20 do grupo 1. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Comando 20!".
};

enum Group16
{
  G16_SENHA = 0,//Comando nº 0 do grupo 0. Para este comando ser executado o utilizador terá que dizer no microfone a seguinte frase:"Acionar Controlo!".
};
```

```
// use negative group for wordsets
int8_t group, idx;

void setup()
{
  // setup PC serial port
  pcSerial.begin(9600);
  //Definição das variáveis de entrada e de saída
  pinMode(sentido_esquerdo,OUTPUT);
  pinMode(sentido_direito,OUTPUT);
  pinMode(velocidade_pequena,OUTPUT);
  pinMode(velocidade_media,OUTPUT);
  pinMode(velocidade_grande,OUTPUT);
  pinMode(distancia_minima,OUTPUT);
  pinMode(distancia_media,OUTPUT);
  pinMode(distancia_maxima,OUTPUT);

bridge:
  // bridge mode?
  int mode = easyvr.bridgeRequested(pcSerial);
  switch (mode)
  {
  case EasyVR::BRIDGE_NONE:
    // setup EasyVR serial port
    port.begin(9600);
    // run normally
    pcSerial.println(F("Bridge not requested, run normally"));
    pcSerial.println(F("---"));
    break;

case EasyVR::BRIDGE_NORMAL:
    // setup EasyVR serial port (low speed)
    port.begin(9600);
    // soft-connect the two serial ports (PC and EasyVR)
    easyvr.bridgeLoop(pcSerial);
    // resume normally if aborted
    pcSerial.println(F("Bridge connection aborted"));
    pcSerial.println(F("---"));
    break;

  case EasyVR::BRIDGE_BOOT:
    // setup EasyVR serial port (high speed)
    port.begin(115200);
    pcSerial.end();
    pcSerial.begin(115200);
    // soft-connect the two serial ports (PC and EasyVR)
    easyvr.bridgeLoop(pcSerial);
    // resume normally if aborted
    pcSerial.println(F("Bridge connection aborted"));
    pcSerial.println(F("---"));
    break;
  }
```

```
  // initialize EasyVR
  while (!easyvr.detect())
  {
    pcSerial.println(F("EasyVR not detected!"));
    for (int i = 0; i < 10; ++i)
    {
      if (pcSerial.read() = = '?')
        goto bridge;
      delay(100);
    }
  }

  pcSerial.print(F("EasyVR detected, version "));
  pcSerial.print(easyvr.getID());

  if (easyvr.getID() < EasyVR::EASYVR3)
    easyvr.setPinOutput(EasyVR::IO1, LOW); // Shield 2.0 LED off

  if (easyvr.getID() < EasyVR::EASYVR)
    pcSerial.print(F(" = VRbot module"));
  else if (easyvr.getID() < EasyVR::EASYVR2)
    pcSerial.print(F(" = EasyVR module"));
  else if (easyvr.getID() < EasyVR::EASYVR3)
    pcSerial.print(F(" = EasyVR 2 module"));
  else
    pcSerial.print(F(" = EasyVR 3 module"));
  pcSerial.print(F(", FW Rev."));
  pcSerial.println(easyvr.getID() & 7);

  easyvr.setDelay(0); // speed-up replies

  easyvr.setTimeout(5);
  easyvr.setLanguage(0); //<-- same language set on EasyVR Commander when code was
generated

  group = 16; //Indica o grupo em que o programa irá começar. Neste caso iniciará no grupo
16 que é o que contém a senha.
}

void loop()
{
  if (easyvr.getID() < EasyVR::EASYVR3)
    easyvr.setPinOutput(EasyVR::IO1, HIGH); // LED on (listening)

  if (group < 0) // SI wordset/grammar
  {
    pcSerial.print("Say a word in Wordset ");
    pcSerial.println(-group);
    easyvr.recognizeWord(-group);
  }
```

```
else // SD group
{
  pcSerial.print("Say a command in Group ");
  pcSerial.println(group);
  easyvr.recognizeCommand(group);
}
do
{
  // allows Commander to request bridge on Zero (may interfere with user protocol)
  if (pcSerial.read() = = '?')
  {
    setup();
    return;
  }
  // <<-- can do some processing here, while the module is busy
}
while (!easyvr.hasFinished());

if (easyvr.getID() < EasyVR::EASYVR3)
  easyvr.setPinOutput(EasyVR::IO1, LOW); // LED off

idx = easyvr.getWord();
if (idx = = 0 && group = = EasyVR::TRIGGER)
{
  // beep
  easyvr.playSound(0, EasyVR::VOL_FULL);
  // print debug message
  pcSerial.println("Word: ROBOT");
  // write your action code here
  // group = GROUP_X\SET_X; <-- jump to another group or wordset
  return;
}
else if (idx >= 0)
{
  // beep
  easyvr.playSound(0, EasyVR::VOL_FULL);
  // print debug message
  uint8_t flags = 0, num = 0;
  char name[32];
  pcSerial.print("Word: ");
  pcSerial.print(idx);
  if (easyvr.dumpGrammar(-group, flags, num))
  {
    for (uint8_t pos = 0; pos < num; ++pos)
    {
      if (!easyvr.getNextWordLabel(name))
        break;
      if (pos != idx)
        continue;
      pcSerial.print(F(" = "));
      pcSerial.println(name);
      break;
```

```
      }
    }
    // perform some action
    action();
    return;
  }
  idx = easyvr.getCommand();
  if (idx >= 0)
  {
    // beep
    easyvr.playSound(0, EasyVR::VOL_FULL);
    // print debug message
    uint8_t train = 0;
    char name[32];
    pcSerial.print("Command: ");
    pcSerial.print(idx);
    if (easyvr.dumpCommand(group, idx, name, train))
    {
      pcSerial.print(" = ");
      pcSerial.println(name);
    }
    else
      pcSerial.println();
    // perform some action
    action();
  }
  else // errors or timeout
  {
    if (easyvr.isTimeout())
      pcSerial.println("Timed out, try again...");
    int16_t err = easyvr.getError();
    if (err >= 0)
    {
      pcSerial.print("Error ");
      pcSerial.println(err, HEX);
    }
  }
}

void action()
{
  switch (group)
  {
  case GROUP_1://Caso alguns dos comandos do grupo 1 seja acionado, o que irá a acontecer em cada um deles será o seguinte:
    switch (idx)
    {
    case G1_DIREITA_GRANDE_MAXIMA://Caso seja acionado o comando nº 2 do grupo 1 ira ocorrer o seguinte:
      easyvr.playSound(2, EasyVR::VOL_FULL);//A mensagem associada ao comando nº 2 do grupo 1 (Direita_Velocidade_Grande_Distância_Máxima) será transmitida através das colunas de som.
```

```
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,HIGH);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,HIGH);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,HIGH);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_DIREITA_GRANDE_MEDIA://A mensagem associada ao comando nº 3 do grupo 1 (Direita_Velocidade_Grande_Distância_Média) será transmitida através das colunas de som.
      easyvr.playSound(3, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,HIGH);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,HIGH);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,HIGH);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_DIREITA_GRANDE_MINIMA://A mensagem associada ao comando nº 4 do grupo 1 (Direita_Velocidade_Grande_Distância_Mínima) será transmitida através das colunas de som.
      easyvr.playSound(4, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,HIGH);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,HIGH);
      digitalWrite(distancia_minima,HIGH);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
```

```
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_DIREITA_MEDIA_MAXIMA://A mensagem associada ao comando nº 5 do grupo 1 (Direita_Velocidade_Média_Distância_Máxima) será transmitida através das colunas de som.
      easyvr.playSound(5, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,HIGH);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,HIGH);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,HIGH);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_DIREITA_MEDIA_MEDIA:
      easyvr.playSound(6, EasyVR::VOL_FULL);//A mensagem associada ao comando nº 6 do grupo 1 (Direita_Velocidade_Média_Distância_Média) será transmitida através das colunas de som.
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,HIGH);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,HIGH);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,HIGH);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
```

```
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_DIREITA_MEDIA_MINIMA:
      easyvr.playSound(7, EasyVR::VOL_FULL);//A mensagem associada ao comando nº 7 do grupo 1 (Direita_Velocidade_Média_Distância_Mínima) será transmitida através das colunas de som.
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,HIGH);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,HIGH);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,HIGH);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_DIREITA_PEQUENA_MAXIMA:
      easyvr.playSound(8, EasyVR::VOL_FULL);//A mensagem associada ao comando nº 8 do grupo 1 (Direita_Velocidade_Pequena_Distância_Máxima) será transmitida através das colunas de som.
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,HIGH);
      digitalWrite(velocidade_pequena,HIGH);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,HIGH);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_DIREITA_PEQUENA_MEDIA:
      easyvr.playSound(9, EasyVR::VOL_FULL);//A mensagem associada ao comando nº 9 do grupo 1 (Direita_Velocidade_Pequena_Distância_Média) será transmitida através das colunas de som.
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,HIGH);
```

```
    digitalWrite(velocidade_pequena,HIGH);
    digitalWrite(velocidade_media,LOW);
    digitalWrite(velocidade_grande,LOW);
    digitalWrite(distancia_minima,LOW);
    digitalWrite(distancia_media,HIGH);
    digitalWrite(distancia_maxima,LOW);
    delay(1000);
    digitalWrite(sentido_esquerdo,LOW);
    digitalWrite(sentido_direito,LOW);
    digitalWrite(velocidade_pequena,LOW);
    digitalWrite(velocidade_media,LOW);
    digitalWrite(velocidade_grande,LOW);
    digitalWrite(distancia_minima,LOW);
    digitalWrite(distancia_media,LOW);
    digitalWrite(distancia_maxima,LOW);
    break;
  case G1_DIREITA_PEQUENA_MINIMA://A mensagem associada ao comando nº 10 do grupo 1 (Direita_Velocidade_Pequena_Distância_Mínima) será transmitida através das colunas de som.
    easyvr.playSound(10, EasyVR::VOL_FULL);
    digitalWrite(sentido_esquerdo,LOW);
    digitalWrite(sentido_direito,HIGH);
    digitalWrite(velocidade_pequena,HIGH);
    digitalWrite(velocidade_media,LOW);
    digitalWrite(velocidade_grande,LOW);
    digitalWrite(distancia_minima,HIGH);
    digitalWrite(distancia_media,LOW);
    digitalWrite(distancia_maxima,LOW);
    delay(1000);
    digitalWrite(sentido_esquerdo,LOW);
    digitalWrite(sentido_direito,LOW);
    digitalWrite(velocidade_pequena,LOW);
    digitalWrite(velocidade_media,LOW);
    digitalWrite(velocidade_grande,LOW);
    digitalWrite(distancia_minima,LOW);
    digitalWrite(distancia_media,LOW);
    digitalWrite(distancia_maxima,LOW);
    break;
  case G1_ESQUERDA_GRANDE_MAXIMA://A mensagem associada ao comando nº 11 do grupo 1 (Esquerda_Velocidade_Grande_Distância_Máxima) será transmitida através das colunas de som.
    easyvr.playSound(11, EasyVR::VOL_FULL);
    digitalWrite(sentido_esquerdo,HIGH);
    digitalWrite(sentido_direito,LOW);
    digitalWrite(velocidade_pequena,LOW);
    digitalWrite(velocidade_media,LOW);
    digitalWrite(velocidade_grande,HIGH);
    digitalWrite(distancia_minima,LOW);
    digitalWrite(distancia_media,LOW);
    digitalWrite(distancia_maxima,HIGH);
    delay(1000);
    digitalWrite(sentido_esquerdo,LOW);
```

```
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_ESQUERDA_GRANDE_MEDIA://A mensagem associada ao comando nº 12 do grupo 1 (Esquerda_Velocidade_Grande_Distância_Média) será transmitida através das colunas de som.
      easyvr.playSound(12, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,HIGH);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,HIGH);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,HIGH);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_ESQUERDA_GRANDE_MINIMA://A mensagem associada ao comando nº 13 do grupo 1 (Esquerda_Velocidade_Grande_Distância_Mínima) será transmitida através das colunas de som.
      easyvr.playSound(13, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,HIGH);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,HIGH);
      digitalWrite(distancia_minima,HIGH);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
```

```
    case G1_ESQUERDA_MEDIA_MAXIMA://A mensagem associada ao comando nº 14 do grupo 1 (Esquerda_Velocidade_Média_Distância_Máxima) será transmitida através das colunas de som.
      easyvr.playSound(14, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,HIGH);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,HIGH);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,HIGH);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_ESQUERDA_MEDIA_MEDIA://A mensagem associada ao comando nº 15 do grupo 1 (Esquerda_Velocidade_Média_Distância_Média) será transmitida através das colunas de som.
      easyvr.playSound(15, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,HIGH);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,HIGH);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,HIGH);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_ESQUERDA_MEDIA_MINIMA://A mensagem associada ao comando nº 16 do grupo 1 (Esquerda_Velocidade_Média_Distância_Mínima) será transmitida através das colunas de som.
      easyvr.playSound(16, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,HIGH);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,HIGH);
```

```
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,HIGH);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_ESQUERDA_PEQUENA_MAXIMA://A mensagem associada ao comando nº 17 do grupo 1 (Esquerda_Velocidade_Pequena_Distância_Máxima) será transmitida através das colunas de som.
      easyvr.playSound(17, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,HIGH);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,HIGH);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,HIGH);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_ESQUERDA_PEQUENA_MEDIA://A mensagem associada ao comando nº 18 do grupo 1 (Esquerda_Velocidade_Pequena_Distância_Média) será transmitida através das colunas de som.
      easyvr.playSound(18, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,HIGH);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,HIGH);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,HIGH);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
```

```
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_ESQUEDA_PEQUENA_MINIMA://A mensagem associada ao comando nº 19 do grupo 1 (Esquerda_Velocidade_Pequena_Distância_Mínima) será transmitida através das colunas de som.
      easyvr.playSound(19, EasyVR::VOL_FULL);
      digitalWrite(sentido_esquerdo,HIGH);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,HIGH);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,HIGH);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      delay(1000);
      digitalWrite(sentido_esquerdo,LOW);
      digitalWrite(sentido_direito,LOW);
      digitalWrite(velocidade_pequena,LOW);
      digitalWrite(velocidade_media,LOW);
      digitalWrite(velocidade_grande,LOW);
      digitalWrite(distancia_minima,LOW);
      digitalWrite(distancia_media,LOW);
      digitalWrite(distancia_maxima,LOW);
      break;
    case G1_DESACIONAR_CONTROLO://A mensagem associada ao comando nº 20 do grupo 1 (Controlo_Será_Desacionado) será transmitida através das colunas de som.
      easyvr.playSound(20, EasyVR::VOL_FULL);
      group = GROUP_16;
      break;
    }
    break;
  case GROUP_16://Caso o único comando do grupo 16 seja acionado, o que irá a acontecer será o seguinte:
    switch (idx)
    {
    case G16_SENHA:
      easyvr.playSound(1, EasyVR::VOL_FULL);//A mensagem associada ao comando nº 0 do grupo 16 (Controlo_Será_Acionado) será transmitida através das colunas de som.
      group = GROUP_1;
      break;
    }
    break;
  }
}
```

ANEXO H: Programação do controlador do sistema desenvolvida no programa Arduino® para a interface com a placa de reconhecimento de voz EasyVR Shield 3.0

```
#include <Stepper.h>
const int stepsPerRevolution = 48; // nº de passos realizados numa volta completa pelo motor de passo
Stepper myStepper(stepsPerRevolution,8,9,10,11); // sequência de alimentação do motor de passo

// Declaração de Variáveis
int enrolamentoA=4;
int enrolamentoB=5;
int fim_curso_esquerdo=51;
int fim_curso_direito=50;
int led_esq=53;
int led_dir=52;
int sentido;
int velocidade;
int distancia;

int sentido_esquerdo=22;
int sentido_direito=24;
int velocidade_pequena=26;
int velocidade_media=28;
int velocidade_grande=30;
int distancia_minima=32;
int distancia_media=34;
int distancia_maxima=36;

void setup() {
Serial.begin(9600); // Taxa de transferência em bits por segundo (baud rate) para transmissão serial

// Definição das variáveis de entrada e de saída
pinMode(enrolamentoA,OUTPUT);
pinMode(enrolamentoB,OUTPUT);
pinMode(led_esq,OUTPUT);
pinMode(led_dir,OUTPUT);
pinMode(fim_curso_esquerdo,INPUT);
pinMode(fim_curso_direito,INPUT);

pinMode(sentido_esquerdo,INPUT);
pinMode(sentido_direito,INPUT);
pinMode(velocidade_pequena,INPUT);
pinMode(velocidade_media,INPUT);
pinMode(velocidade_grande,INPUT);
pinMode(distancia_minima,INPUT);
pinMode(distancia_media,INPUT);
pinMode(distancia_maxima,INPUT);
```

```
// Função attachInterrupt, responsável pelo devido funcionamento dos Fins de Curso do eixo linear
attachInterrupt(digitalPinToInterrupt(51),fimcursoesquerdo,RISING); // Fim de curso Esquerdo:
// Quando o fim de curso esquerdo é acionado, a entrada digital nº 51, sofre uma transição de LOW para HIGH, o que faz com que o programa que está a correr seja interrompido (attachInterrupt)
attachInterrupt(digitalPinToInterrupt(50),fimcursodireito,RISING); //  Fim de curso Direito:
// Quando o fim de curso direito é acionado, a entrada digital nº 50, sofre uma transição de LOW para HIGH, o que faz com que o programa que está a correr seja interrompido (attachInterrupt)
}

void loop() {
// Neste caso, os valores são recebidos pela interface, que é a câmara (Open MV Cam M7).
sentido_esquerdo=digitalRead(22);//leitura dos valores da entrada digital nº 14. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova no sentido esquerdo.
sentido_direito=digitalRead(24);//leitura dos valores da entrada digital nº 24. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova no sentido direito.
velocidade_pequena=digitalRead(26);//leitura dos valores da entrada digital nº 26. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova com velocidade pequena.
velocidade_media=digitalRead(28);//leitura dos valores da entrada digital nº 28. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova com velocidade media.
velocidade_grande=digitalRead(30);//leitura dos valores da entrada digital nº 30. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova com velocidade grande.
distancia_minima=digitalRead(32);//leitura dos valores da entrada digital nº 32. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova uma distância mais pequena (mínima).
distancia_media=digitalRead(34);//leitura dos valores da entrada digital nº 34. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova uma distância média.
distancia_maxima=digitalRead(36);//leitura dos valores da entrada digital nº 36. Se o valor for "HIGH" ou "1", significa que a interface está a ordenar que a porta se mova uma distância maior (máxima).

//Neste programa, a interface é o microfone, e como tal, sem receber nenhuma ordem do utilizador através do microfone, o programa não avança, como se pode verificar na seguinte condição:
if ((sentido_esquerdo = = HIGH or sentido_direito = = HIGH) and (velocidade_pequena = = HIGH or velocidade_media = = HIGH or velocidade_grande = = HIGH) and (distancia_minima = = HIGH or distancia_media = = HIGH or distancia_maxima = = HIGH))
//Nota: A informação que é recebida do microfone é recebida na seguinte forma: (nº;nº;nº;nº;nº;nº;nº;nº), sendo que:
// O primeiro nº irá indicar se porta se irá deslocar no sentido esquerdo, sendo que isso acontecerá caso esse valor seja "1" ou "HIGH".
// O segundo nº irá indicar se porta se irá deslocar no sentido direito, sendo que isso acontecerá caso esse valor seja "1" ou "HIGH".
```

```
// O terceiro nº irá indicar se porta se irá deslocar com velocidade pequena, sendo que isso acontecerá caso esse valor seja "1" ou "HIGH".
// O quarto nº irá indicar se porta se irá deslocar com velocidade média, sendo que isso acontecerá caso esse valor seja "1" ou "HIGH".
// O quinto nº irá indicar se porta se irá deslocar com velocidade grande, sendo que isso acontecerá caso esse valor seja "1" ou "HIGH".
// O sexto nº irá indicar se porta se irá deslocar uma distância mínima, sendo que isso acontecerá caso esse valor seja "1" ou "HIGH".
// O sétimo nº irá indicar se porta se irá deslocar uma distância média, sendo que isso acontecerá caso esse valor seja "1" ou "HIGH".
// O oitavo nº irá indicar se porta se irá deslocar uma distância máxima, sendo que isso acontecerá caso esse valor seja "1" ou "HIGH".
{
fim_curso_direito=digitalRead(50); // leitura dos valores da entrada digital nº 50. Se o valor for "HIGH" ou "1", o fim de curso direito está acionado, se for "LOW" ou "0", não está acionado.
fim_curso_esquerdo=digitalRead(51); // leitura dos valores da entrada digital nº 51. Se o valor for "HIGH" ou "1", o fim de curso esquerdo está acionado, se for "LOW" ou "0", não está acionado.

// Definição do valor da variável "sentido"
if(sentido_esquerdo = = HIGH)
{sentido=0;} // se o valor da variável for "0" significa que a interface está a ordenar que a porta se mova no sentido esquerdo.
if(sentido_direito = = HIGH)
{sentido=1;} // se o valor da variável for "1" significa que a interface está a ordenar que a porta se mova no sentido direito.
if(sentido_direito = = LOW and sentido_esquerdo = = LOW)
{sentido=2;} // se o valor da variável for "2" significa que a interface está a ordenar que a porta não se mova em nenhum dos 2 sentidos.

//Definição do valor da variável "velocidade"
if(velocidade_pequena = = HIGH)
{velocidade=300;}// caso o utilizador ordene que a porta se desloque com velocidade baixa, ela deslocar-se-á à velocidade de 300 mm/s.
if(velocidade_media = = HIGH)
{velocidade=350;}// caso o utilizador ordene que a porta se desloque com velocidade média, ela deslocar-se-á à velocidade de 350 mm/s.
if(velocidade_grande = = HIGH)
{velocidade=400;}// caso o utilizador ordene que a porta se desloque com velocidade alta, ela deslocar-se-á à velocidade de 400 mm/s.
//Nota: o valor da velocidade encontra-se entre o intervalo de valores de 300 a 400 mm/s pois é o mais adequado, tal como foi concluído através de diversas tentativas/experiências.

//Definição do valor da variável "distância"
if(distancia_minima = = HIGH)
// caso o utilizador ordene que a porta se desloque com distância mínima, o motor realizará 239 passos, que está diretamente relacionado com a distância que a porta percorrerá.
{distancia=239;} // Por cada passo que o motor dá, a porta desloca-se 0.083 mm, logo neste caso, 239 passos do motor correspondem a um deslocamento linear da porta de 20mm.
```

```
if(distancia_media = = HIGH)
// caso o utilizador ordene que a porta se desloque com distância média, o motor realizará 537 passos, que está diretamente relacionado com a distância que a porta percorrerá.
{distancia=537;} // Por cada passo que o motor dá, a porta desloca-se 0.083 mm, logo neste caso, 537 passos do motor correspondem a um deslocamento linear da porta de 45mm.
if(distancia_maxima = = HIGH)
// caso o utilizador ordene que a porta se desloque com distância média, o motor realizará 955 passos, que está diretamente relacionado com a distância que a porta percorrerá.
{distancia=955;} // Por cada passo que o motor dá, a porta desloca-se 0.083 mm, logo neste caso, 955 passos do motor correspondem a um deslocamento linear da porta de 80mm.
delay(1000);

// Nota: Através de todas estas condições acima, é definido qual o sentido, a velocidade e a distância com que a porta se irá deslocar.

{
 if(sentido = = 1 and fim_curso_direito = = 0) // DESLOCAMENTO PARA A DIREITA: Só acontece caso o valor do sentido seja 1 e o fim de curso direito não esteja acionado.
 {digitalWrite(led_esq,LOW); // Caso o fim de curso esquerdo tenha sido acionado o led acende e quando se faz o movimento para a direita ele terá de apagar pois deixa de estar acionado.
  digitalWrite(enrolamentoA,HIGH); // Alimentação do enrolamento A do motor de passo
  digitalWrite(enrolamentoB,HIGH); // Alimentação do enrolamento B do motor de passo
  myStepper.setSpeed(velocidade); // Indicação da velocidade de trabalho do motor
  myStepper.step(-distancia); // Indicação do número de passos que o motor irá realizar. Este nº multiplicado por 0.0838 mm indica-nos a distância que a porta do nosso sistema irá percorrer.
  // Nota: Como podem reparar, o valor introduzido leva o sinal menos (-) para que o motor se desloca no sentido que permite a porta mover-se para a direita.
  }
  if(sentido = = 0 and fim_curso_esquerdo = = 0) // DESLOCAMENTO PARA A ESQUERDA: Só acontece caso o valor do sentido seja 0 e o fim de curso esquerdo não esteja acionado.
 {digitalWrite(led_dir,LOW); // Caso o fim de curso direito tenha sido acionado o led acende e quando se faz o movimento para a esquerda ele terá de apagar pois deixa de estar acionado.
  digitalWrite(enrolamentoA,HIGH); // Alimentação do enrolamento A do motor de passo
  digitalWrite(enrolamentoB,HIGH); // Alimentação do enrolamento B do motor de passo
  myStepper.setSpeed(velocidade); // Indicação da velocidade de trabalho do motor
  myStepper.step(distancia); // Indicação do número de passos que o motor irá realizar. Este nº multiplicado por 0.0838 mm indica-nos a distância que a porta do nosso sistema irá percorrer.
  }
}
}
}
void fimcursoesquerdo() { // Quando o fim de curso esquerdo é acionado, o programa é interrompido devido à função (attachInterrupt), tal como já foi acima explicado, e acontece o seguinte:
digitalWrite(led_esq,HIGH); // Acende o led que indica que o fim de curso esquerdo foi acionado.
digitalWrite(enrolamentoA,LOW); // Corte da alimentação do enrolamento A do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.
```

digitalWrite(enrolamentoB,LOW);} // Corte da alimentação do enrolamento B do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.

void fimcursodireito() { // Quando o fim de curso direito é acionado, o programa é interrompido devido à função (attachInterrupt), tal como já foi acima explicado, e acontece o seguinte:
digitalWrite(led_dir,HIGH); // Acende o led que indica que o fim de curso direito foi acionado.
digitalWrite(enrolamentoA,LOW); // Corte da alimentação do enrolamento A do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.
digitalWrite(enrolamentoB,LOW);} // Corte da alimentação do enrolamento B do motor de passo, o que não permite com que o motor de mova pois não está a ser alimentado com energia.