Reconhecimento de Linguagem Gestual

Ana Maria Sousa, Mariana Xavier

Resumo — O sinal de eletromiografia (EMG) tem sido amplamente utilizado quer na área medica para diagnostico quer para identificação de movimentos tendo em vista, por exemplo, o controlo de dispositivos como próteses. Este trabalho visa o desenvolvimento de uma aplicação que corresponde a um protótipo de um tradutor de linguagem gestual e do tradicional jogo Pedra Papel Tesoura. Para identificação dos 3 gestos em causa foram colocados 3 elétrodos de superfície no antebraço e foi desenvolvido um circuito analógico que permite a sua aquisição e processamento do mesmo de forma a obter o sinal de EMG integrado (IEMG). Este é um elemento-chave para a determinação do gesto uma vez que a cada músculo está associado um valor de *threshold* que determina se o sinal captado corresponde ao músculo no estado ativo ou inativo, que combinando os dois canais permite determinar a posição da mão. O sinal analógico processado é convertido para digital a partir do Arduíno e a informação acerca do estado de ativação é enviada para Java, onde o algoritmo de identificação de gestos e o tradutor/jogo são implementados. Os resultados demonstraram a possibilidade de recorrer a este sinal para a identificação de gestos de uma forma simples e rápida, quer para aplicação como tradutor quer para implementação do jogo em causa.

Palavras-chave – eletromiografia (EMG), antebraço, processamento de sinal fisiológico, sinal de EMG integrado (IEMG), identificação de linguagem gestual, Arduino, Java.

I. Introdução

A linguagem gestual trata-se da única forma de comunicação de pessoas que por alguma fatalidade ou por serem portadoras de alguma deficiência são incapazes de ouvir ou falar. Os surdos e mudos enfrentam diariamente desafios inerentes às suas incapacidades, apesar de já terem sido desenvolvidas ferramentas na tentativa de minimizar estes problemas, como o desenvolvimento de um dicionário de linguagem gestual portuguesa [1] e aplicações para facilitar a compreensão da mesma [2]. No entanto, continua a existir uma grande lacuna na comunicação entre pessoas portadoras destas deficiências e pessoas comuns. Desenvolver uma forma de detetar e identificar os gestos pode ser uma forma de possibilitar esta comunicação. Este projeto consiste num protótipo de uma aplicação que permite essa deteção e tradução de gestos recorrendo a sinal de EMG.

O eletromiograma corresponde um sinal mioelétrico que resulta da ativação muscular (músculos esqueléticos) através da troca de iões nas membranas musculares [3]. Os músculos estão organizados em unidades motoras que correspondem a um conjunto de fibras musculares cuja ação é comandada por um mesmo nervo motor [4]. As diferenças de potencial geradas durantes as contrações musculares podem ser medidas a partir da pele de forma não invasiva recorrendo a elétrodos de superfície. Este sinal possui amplitudes de 50 µV a 5 mV e frequências de 2-500 Hz [5].

O antebraço é formado por 20 músculos que, juntamente com as articulações, possibilita uma grande destreza de movimentos e ampla gama de posições da mão e antebraço. Neste trabalho recorreu-se a 3 elétrodos colocados em locais específicos do antebraço: músculo *extensor pollicis brevis*, músculo *extensor digitorum* e um de referência na zona óssea correspondente ao cotovelo [6]. O primeiro é o músculo extensor curto do polegar, ou seja, é responsável pela abdução do mesmo e é um músculo profundo que se liga ao cúbito do antebraço situando-se abaixo do músculo supinador. O segundo é o músculo extensor dos dedos e faz parte dos músculos superficiais do antebraço, sendo responsável pela extensão do punho e dos dedos.

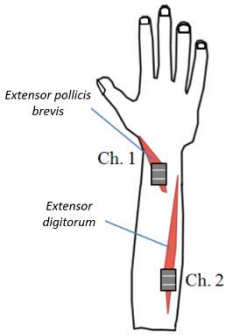

Figura 1 - Disposição dos elétrodos sob os dois músculos em causa.

Devido à falta de material e impossibilidade de recolher sinais de EMG em tempo real, o projeto foi desenvolvido integralmente por simulação recorrendo sinais EMG obtidos previamente através do software BIOPAC e utilizou-se Multisim para o design do circuito analógico para captação e processamento do sinal, Arduíno para conversão a sinal digital e avaliação do mesmo e Java para desenvolvimento da aplicação.

De forma a desenvolver simultaneamente um projeto que fosse simultaneamente ilustrativo e simples (executável recorrendo aos conhecimentos adquiridos até à data), foram escolhidos para identificação apenas 3 gestos: pedra, papel, tesoura. Estes gestos foram selecionados devido a serem bastante estudados pela literatura [3]. Por outro lado, são também significativos de letras em linguagem gestual. Por exemplo, pedra é semelhante a um "A" e tesoura a um "V".

II. Desenvolvimento

De forma a desenvolver um dispositivo capaz de detetar e interpretar sinais de natureza mioelétrica, foi desenvolvida uma componente analógica que se foca no condicionamento do sinal EMG e uma componente digital que inclui a interpretação dos sinais e a interface computador-humano.

A. Circuito Analógico em Multisim

O sinal de eletromiografia é sinal de baixa amplitude, entre 1 e 10 mV, dependendo evidentemente da atividade muscular e

de outros fatores como por exemplo o tipo de fibras musculares consideradas [7]. Para além disso o EMG é caracterizado por ser um sinal extremamente ruidoso. Daí advém a necessidade de um conjunto de etapas de processamento de sinal que compreendem a amplificação, filtragem e retificação seguida da conversão analógico-digital [6-8].

Para simulação de um sinal EMG real recorreu-se a um sinal de EMG recolhido previamente do antebraço usando o *software* BIOPAC. Recorreu-se a dois sinais de EMG: um correspondente a um músculo em contração e outro a um músculo relaxado. Estes dados encontravam-se num ficheiro em formato .txt e foram transformados em sinal fisiológico em Multisim através do componente *PWL Voltage*.

Elétrodos e Amplificação

Neste projeto foram utilizados elétrodos de superfície. Estes, embora tenham a vantagem de serem acessíveis e não invasivos, podem ser também fontes de ruído devido a deslocações ligeiras na pele ou pelo facto de captarem sinais provenientes de uma grande quantidade de fibras musculares [8].

A primeira etapa da componente analógica corresponde à etapa de aquisição na qual é utilizado o *OpAmp INA126*. Este é um amplificador de instrumentação de elevada exatidão, pequeno *offset*, baixo ruído e que apresenta coeficiente de rejeição de modo comum (CMRR) e impedância de entrada elevados na ordem dos 94dB e 109 Ω, respetivamente [9]. Desta forma, é possível garantir uma impedância de entrada segura e atenuar drasticamente o ruído de modo comum. A extração desta componente de ruído é possibilitada pela utilização de um elétrodo de referência colocado numa zona óssea [10], que como se trata de uma região sem contração muscular apenas captará sinais provenientes de outros sinais fisiológicos, músculos mais distantes ou outras fontes de ruido comum. Assim o efeito destas no sinal EMG a considerar será fortemente reduzido.

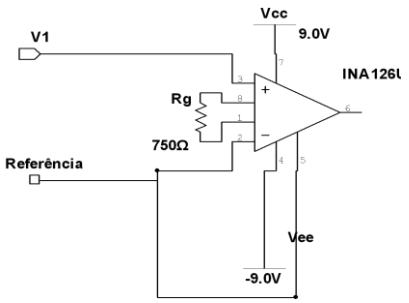

Figura 2 - Circuito amplificador de instrumentação INA126.

O ganho do circuito pode variar entre 5 e 10000 apenas alterando uma resistência R_G. Este ganho pode ser calculado através da seguinte equação:

$$G = 5 + \frac{80\ k\Omega}{R_G} \quad (1)$$

Neste caso foi utilizada uma resistência de 750 Ω (pertencente à série E24), obtendo-se um ganho aproximado de 112 V/V. A tabela 1 representa os componentes utilizados nesta etapa de condicionamento de sinal:

Tabela 1. Listagem dos componentes para aquisição e amplificação.

Aquisição e Amplificação	
Componentes	**Quantidades**
INA126	1
R = 750 Ω	1
Elétrodos de superfície	3

Amplificação e Filtro Ativo

Como referido anteriormente, o sinal de eletromiografia possui uma amplitude reduzida sendo necessário uma amplificação considerável para que possa ser detetado. Pelo que foram utilizadas várias etapas de amplificação no circuito.

Procedeu-se amplificação do sinal recorrendo ao amplificador TL084 em configuração inversora com ganho de 15. Para tal, recorreu-se a duas resistências: uma de 10 kΩ e outra de 150 kΩ. O ganho pode ser calculado através da seguinte expressão:

$$Ad = \frac{Vout}{Vin} = -\frac{Rf}{R} \quad (2)$$

Por outro lado, é importante enfatizar que embora o sinal de EMG possa oscilar entre frequências 0-500 Hz, este encontra-se predominantemente na gama de frequências entre 50-150 Hz, sendo necessário atenuar artefactos de frequências que não coincidam com a porção do sinal de EMG de interesse [4].

Assim sendo, foi adicionado um filtro passa-alto ativo de ganho unitário (duas resistências de valor igual a 150 kΩ) e frequência de corte aproximada de 106Hz que pode ser calculada pela equação abaixo. Este filtro atenua artefactos de frequências inferiores o que inclui ruído proveniente da instabilidade dos elétrodos (0-20 Hz), variações de temperatura e ruído proveniente das fontes que correspondem a frequências na banda dos 50-60 Hz [4].

$$\mathrm{fc} = \frac{1}{2\pi RC} \quad (3)$$

O condensador que incorpora o filtro passa-alto tem o valor de 0,01 µC para obter a frequência de corte desejada e permite ainda remover o *offset* DC. Note-se que, durante estas duas etapas ocorre duas inversões sucessivas do sinal, pelo que o sinal de saída corresponde a um sinal não-invertido mas com uma pequena diferença de fase relativamente ao sinal original, decorrente do *shift* provocado pelo filtro.

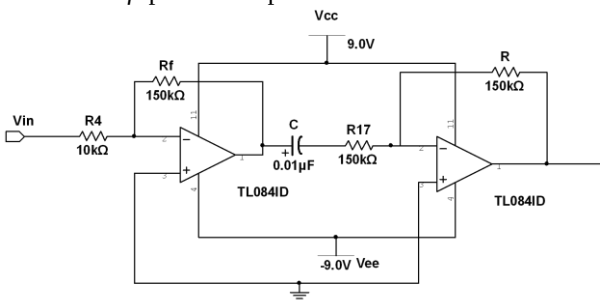

Figura 3 - Circuito amplificador e filtro passa-alto.

Tabela 2. Listagem dos componentes na etapa de amplificação e filtro passa-alto.

Amplificação		Filtro Passa-Alto	
Componentes	**Quantidades**	**Componentes**	**Quantidades**
TL084	1	TL084	1
R = 10 kΩ	1	R = 150 kΩ	2
R = 150 kΩ	1	C = 0,01 µF	1

Retificação

Após a etapa de amplificação e filtragem do sinal efetua-se uma retificação. Esta consiste na conversão da parte do sinal correspondente a tensão negativa para positiva, resultando num sinal em que toda a tensão é positiva.

A retificação do sinal de EMG é comumente utilizada quando se pretende utilizar este sinal fisiológico para controlo de dispositivos através da identificação da atividade muscular. O circuito que implementa esta fase de processamento do sinal está representado na figura 4.

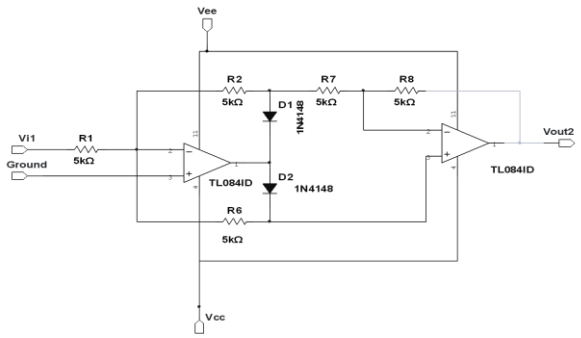

Figura 4 - Circuito retificação.

Este processo permite transformar a corrente AC em voltagem DC para ser lida pelo microcontrolador. A tabela 3 sumariza os componentes necessários para a construção da porção do circuito que permite a retificação do sinal.

Tabela 3. Listagem dos componentes na etapa de retificação.

Retificação	
Componentes	**Quantidades**
TL084	2
R = 5 kΩ	5
Díodo (1N4148)	2

Smoothing e Amplificação

Por fim, foi ainda necessário incluir uma etapa de *smoothing* através da utilização de um filtro passa-baixo ativo cuja frequência de corte pode ser igualmente calculada pela equação 3. Uma vez que este filtro é constituído por um *OpAmp* em configuração inversora o sinal de saída corresponde a um sinal invertido. Assim foi adicionado uma nova etapa de amplificação com um circuito amplificador inversor para recuperar a orientação do sinal. Neste caso a amplificação utilizada foi de ganho unitário, não tendo qualquer efeito na amplitude do sinal, que por sua vez já se encontrava amplificado; no entanto, esta etapa permite ajustar a amplificação aos sinais de EMG correspondentes aos músculos que estão a ser monitorizados. Note-se que a amplitude e frequência do eletromiograma pode variar quer pela atividade do musculo quer de musculo para musculo. O ganho do circuito pode ser calculado através da equação 2. Através desta, verificamos que a alteração de apenas 1 resistência ou a utilização de um potenciómetro permitiria ajustar o ganho a qualquer musculo.

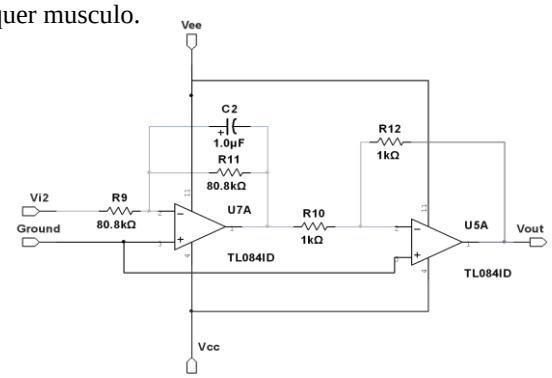

Figura 5 - Circuito de *smoothing* e amplificação.

O sinal final do circuito analógico corresponde ao sinal de EMG integrado (IEMG). Na tabela 4 estão representados os componentes utilizados nesta etapa final de condicionamento de sinal.

Tabela 4. Listagem dos componentes na etapa de *smoothing* e amplificação.

Smoothing		Amplificação	
Componentes	**Quantidades**	**Componentes**	**Quantidades**
TL084	1	TL084	1
R = 80.8 kΩ	2	R = 1 kΩ	2
C = 1 µF	1		

Uma vez que o propósito deste projeto implica a recolha de dois sinais de EMG, foram necessários 2 sensores que correspondentes a circuitos contendo todas as etapas anteriormente referidas. A figura 6 resume a implementação de um sensor dos EMG em *Multisim* e a figura 7 a junção destes sensores à leitura dos sinais EMG, permitindo simular os 3 gestos desejados.

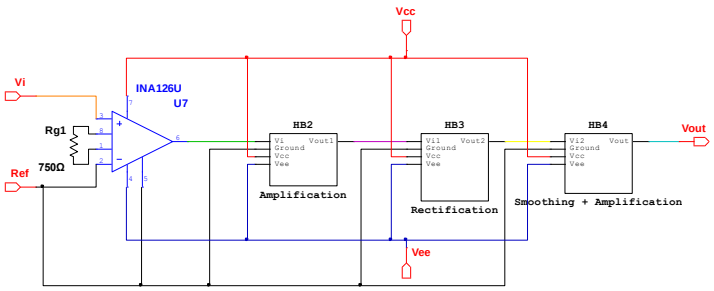

Figura 6 – Sensor de EMG.

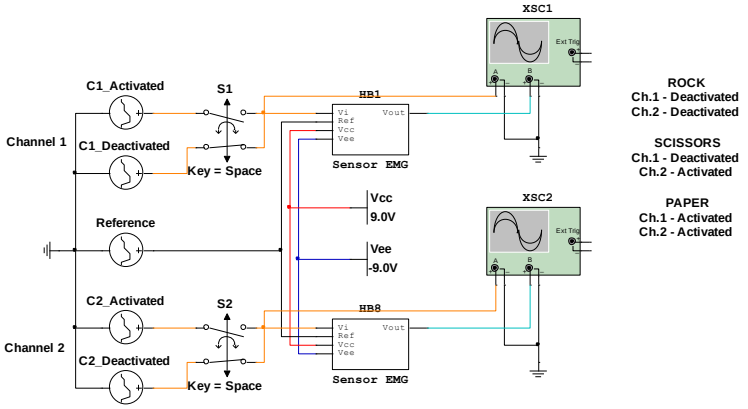

Figura 7 – Aplicação dos sensores de EMG aos sinais lidos.

Para representação de um dos sensores numa *breadboard*, tal como seria numa situação real, recorreu-se ao *Tinkercad*. Esta esquematização encontra-se no anexo 1.

B. Aquisição e Avaliação do Sinal em Arduino

Tendo obtido os dois sinais de IEMG pelo circuito analógico, torna-se necessária a sua conversão analógico-digital para que estes possam ser interpretados, sendo esta feita por Arduino.

Numa situação real, os dois sinais iriam ser lidos através de dois pins analógicos, tal como mostra a função implementada *readSample*. À medida que os valores iam sendo recolhidos, ia sendo também atualizada a média de cada canal como mostra a função *updateMean*. Estas duas funções seriam conjugadas na função *real*, que iria chamá-las em *loop* até se atingirem 200 amostras. Como isto não foi possível pela falta de material de recolha de sinal e componentes eletrónicos, recorreu-se à função *random* de Arduíno para substituir este passo, gerando um valor com 2 casas decimais para atribuir a cada uma das médias.

Para além destas implementou-se também a função *evaluateCh* que, conforme a média e um *threshold* específico, determina o estado de ativação do canal que lhe é passado como parâmetro. Isto é, se a média do canal for superior ao *threshold* este é considerado ativo, e vice-versa. Tanto o número de amostras a ler como os *thresholds* foram definidos conforme um estudo para identificação dos mesmos 3 gestos, que determinou a combinação que permitiria obter uma melhor exatidão nos resultados [6]. Assim, definiram-se os seguintes *thresholds*:

Tabela 5. *Thresholds* que determinam o estado de ativação de cada um dos canais.

	***Thresholds* (V)**
Canal 1	0,53
Canal 2	0,76

Quanto ao funcionamento geral, este tem o *setup* onde é definida a taxa de transmissão dos dados e define também a *seed* para a função *random* recorrendo a um pin analógico ao qual nada se encontra ligado.

Por fim, no *loop*, tudo isto está interligado com a comunicação bidirecional com Java. Em primeiro lugar, Arduíno espera que tenha algo disponível na porta série. Quando isso acontece lê um *byte* que recebe de Java pela função *Serial.read* e interpreta-o como correspondente ao início da aquisição do sinal. Seria neste ponto que se faria a chamada à função real. Gera, então, dois valores aleatórios entre 0 e 2 (gama do IEMG) e recorre à função *evaluateCh* para comparar cada canal com o seu *threshold*. Isto devolve dois caracteres acerca do estado de cada um dos canais, que são enviados para Java pela função *Serial.write* e faz um *reset* do *counter*. Assim, fica à espera de nova informação de Java para nova leitura.

C. Desenvolvimento da Aplicação em Java

Em Java, pretendeu-se desenvolver uma aplicação que implementa duas partes distintas:

1- Tradutor: tanto de palavra para gesto como de gesto para palavra.
2- Jogo: Pedra Papel Tesoura.

Em ambas as partes é possível selecionar um modo com dispositivo, que recorre ao sinal de EMG, ou a opção sem dispositivo permitindo utilizar a aplicação mesmo sem a captação dos sinais de EMG.

No desenvolvimento da aplicação java foram implementadas diferentes classes que vão ser de seguida abordadas.

Class Position

Esta classe pretende definir a posição da mão/gesto do jogador. Possui 3 variáveis internas: o nome da posição em formato *string* e duas médias a partir das quais os canais 1 e 2 são considerados ativos. Note se que estes valores foram previamente definidos em Arduino e que permitem avaliar o gesto feito pelo jogador; no entanto, para as jogadas do computador foram geradas posições aleatórias, daí a necessidade destas variáveis.

Quanto aos seus métodos, apresenta 4 construtores diferentes:

- o primeiro corresponde a um construtor vazio, definindo o gesto como *"NO NAME";*

- o segundo recebe uma *string* de entrada, utilizando-a como nome do gesto;

- o terceiro, a partir das médias dos dois canais, faz a avaliação e atribui o gesto respetivo, comparando com as duas médias;

- o último recebe dois *chars* relativos ao estado de ativação dos canais ('a' para ativo e 'd' para inativo – do inglês *deactivated*) e, a partir destes, define o gesto pela combinação presente na tabela abaixo. É de notar que o caso em que o canal 1 se encontra ativo e o canal 2 inativo não pertence a uma das possibilidades de sinais, sendo o *default* definido como *"UN*RECOGNISED".

Para finalizar, esta classe possui ainda o método *getName* que retorna uma *string* indicando a posição.

	Canal 1	Canal 2
Pedra	‘d’	‘d’
Papel	‘a’	‘a’
Tesoura	‘d’	‘a’

Tabela 6. Determinação dos gestos conforme o estado de ativacão dos canais.

Class Player

Esta classe pretende implementar um jogador, sendo esta uma classe abstrata, visto que um jogador pode se tratar tanto do utilizador como do computador em si, sendo estas duas classes que a estendem.

Possui 3 variáveis internas: a posição da mão do jogador, *score* que corresponde à pontuação total e o nome do jogador.

Relativamente aos métodos possui um único construtor que recebe uma *string* representativa do nome como argumento, inicializa *score* como 0 e para a posição chama o construtor *vazio da class Position.*

O método winRound é chamado quando o jogador em causa ganha uma ronda, incrementando a sua pontuação total.

Os métodos *getScore*, *getName*, *getPosition* e *getPositionName* retornam a variável que o próprio nome indica enquanto que os métodos *setName*, *setPosition* e *setScore* atualizam a variável em questão para aquela que recebem como parâmetro.

Class Computer

Esta classe estende a classe *Player*, tratando-se do adversário do utilizador no jogo. Desta forma, possui todas as variáveis e métodos da classe anterior e implementando ou definido também os seus próprios métodos.

Possui um construtor que, pelo método *super*, chama o construtor da classe *Player*.

Para além disso, possui um método denominado *definePosition* que pretende definir a posição da mão correspondente ao computador nas diversas rondas. Começa, então, por gerar dois valores aleatórios, 0 ou 1, para os dois canais. Estes valores vão corresponder a inativo e ativo, respetivamente, aquando da chamada ao método *setPosition* da classe anterior.

Class User

Esta classe estende a classe *Player* mas, desta vez, tratando-se do utilizador da aplicação, cujo sinal de EMG vai ser lido e interpretado.

Tal como a classe *Computer*, possui o construtor e o método *definePosition*. No entanto, este método recebe dois *chars* respetivos ao estado de ativação dos canais, recorrendo também ao método s*etPosition* para definir a posição.

Class Game

Esta classe implementa funções essenciais para o desenvolvimento do jogo Pedra Papel Tesoura.

Tem 3 variáveis internas: o utilizador, o computador (adversário ao utilizador) e o número de rondas total.

Começando pelo construtor por omissão, este inicializa o utilizador com *“NO NAME”* e o computador com o nome *“PC”*, colocando o número de rondas a 0. Possui ainda outro construtor que recebe tanto o nome do utilizador como o número de rondas como argumento, sendo o computador inicializado da mesma maneira.

Possui diversos métodos *get*: *getUser*, *getComputer* e *getRounds*, que retornam tal como nome sugere, o jogador correspondente ao utilizador, ao computador e o número de rondas, respetivamente. Para além disso, apresenta o método *redefineGame* que redefine um jogo previamente formado com as novas variáveis relativas ao utilizador e ao número de rondas.

Para o jogo propriamente dito, o método *roundWinner* compara as posições do jogador e do computador retornando um inteiro: 1 caso o user ganhe, -1 caso o computador ganhe e 0 em caso de empate.

O método *nextRound* implementa uma ronda, gerando as posições de ambos os jogadores, fazendo o seu *display* na consola e determinando o vencedor. De forma complementar, *allRound*s implementa um jogo inteiro, recorrendo a *nextRound* até se atingir o número de rondas final do jogo.

Tendo o jogo acabado, o método *getWinner* compara as pontuações de ambos e retorna um inteiro representante do vencedor do jogo: 1 para o *user*, 0 em caso de empate e -1 para o computador.

Por fim, *resetGame* repõe as pontuações do utilizador e do computador a 0, deixando os seus nomes e o número de rondas intacto, para fácil inicialização de um novo jogo com as mesmas condições. Esta classe possui ainda um *main* para teste.

Class CanvasUser e CanvasPC

Estas classes pretendem fazer o *display* de imagens num *Panel* e foram criadas para permitir o *display* das diferentes imagens num mesmo *Panel*. A imagem a ser mostrada depende do gesto que for identificado, sendo este indicado através de uma *string* que é a variável interna e o parâmetro do construtor das classes.

Estas classes são implementadas recorrendo às classes *Canvas e Graphics*, permitindo a delimitação da área para representação gráfica e *display* das imagens em vários formatos, suportando também *gifs.* No caso desta aplicação foram criados *gifs* correspondentes a cada gesto para *o User* (figura 8) e para o *Computer* (figura 9). A tomada de decisão é feita através de *switch…case* mediante a *string* passada por parâmetro ao método *paint*.

Ambas estas classes possuem um *main* para teste.

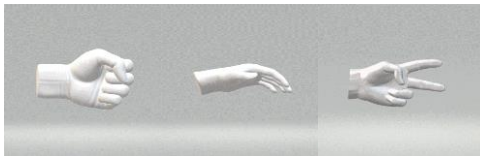

Figura 8 – Imagens ilustrativas dos gestos do *User*.

Figura 9 – Imagens ilustrativas dos gestos do *Computer*.

Class Sound

A classe *Sound* recorre à biblioteca *Applet* para permitir a reprodução de áudios associados a *JFrames*.

Possui o método *play* que faz a reprodução do áudio cujo nome recebe como *input*.

Nesta aplicação é feita a chamada a esta classe na tradução de gestos, sendo esta acompanhada de áudios dos respetivos gestos. Para além disso, no fim do jogo, é dada a indicação se o utilizador saiu ou não vitorioso, sendo esta acompanhada também de um áudio de acordo com a situação.

Class Communication

De seguida, tornou-se necessário desenvolver uma classe que fizesse a comunicação entre Java e Arduino por comunicação série, de forma a que a leitura dos sinais e a respetiva avaliação estivessem sincronizadas ao longo de toda a utilização da aplicação. Para tal, recorreu-se à biblioteca externa *jSerialComm* [11].

Esta possui uma variável interna, relativa à porta de comunicação série a ser utilizada, ou seja, um objeto do tipo *SerialPort*.

No seu construtor é feita a inicialização da comunicação série. Em primeiro lugar, é feita uma pesquisa pelas portas série disponíveis e, de seguida, é escolhida a primeira como a porta a utilizar. Para além disso são definidos os parâmetros de escrita e de leitura. O modo de leitura *TIMEOUT_READ_BLOCKING* diz-nos que caso até um determinado tempo não se consigam ler o número de *bytes* desejado, lê-se aqueles que estiverem disponíveis. Foi escolhido o modo análogo para a escrita.

Possui o método *getCommPort* apenas para retornar a porta série a ser utilizada, o que vai ser necessário para a classe *Translator*.

Posto isto, tratou-se de definir os métodos que permitem a comunicação. Por um lado, o método *recieveChar* lê da porta série o primeiro *byte* disponível e transforma-o em *char* antes de o retornar. Por outro lado, o método *sendChar* passa o *char* desejado para *byte*, enviando-o pela comunicação série para ser lido em Arduino tal como já foi especificado.

Por fim, possui um método que intercala os dois anteriores, permitindo fazer a comunicação por inteiro, a ser utilizado na classe *Play*. Este começa por fazer a abertura da comunicação, esperando dois segundos para que esta seja iniciada corretamente antes de tentar enviar ou receber objetos. De seguida, faz o envio de um *char* e, caso o envio seja bem sucedido, faz a leitura de outros dois *chars*, retornando-os dentro de um *array* denominado *recieved*.

Tendo todas as classes anteriores definidas passou-se à junção das mesmas de forma a implementar a aplicação desejada.

As classes *App*, *Translator* e *Play* permitiram a criação da interface gráfica e fazem a conjugação de todas as classes anteriormente explicadas de forma a implementar o tradutor de linguagem gestual e o jogo Pedra Papel Tesoura.

Class App

Esta classe implementa uma *Application Window* que corresponde ao menu inicial da aplicação e que permite aceder quer ao tradutor quer ao jogo mediante a escolha do utilizador, através de *buttons* cuja ação torna visível o *JFrame* correspondente ao tradutor ou ao jogo.

Class Translator

A classe *Translator* implementa toda a interface gráfica e funcionamento de um tradutor de linguagem gestual. Este é dividido (através de *layers*) em secções: tradução palavra-gesto e gesto-palavra. Sendo ainda possível em cada uma delas trocar para a outra secção ou voltar para o menu inicial pressionando o *button* "<->" ou "*Back*", respetivamente.

Relativamente à tradução *Word – Gesture*, o utilizador introduz uma *string* correspondente à palavra que pretende traduzir para linguagem gestual. Se esta *string* for corresponder à designação, em português ou em inglês, de um dos gestos na base de dados da aplicação é feito o *display* do mesmo. Caso a *string* introduzida não seja reconhecida é mostrada a imagem da figura 10.

Note-se que a aplicação reconhece as palavras independentemente de estas estarem em letra maiúscula ou minúscula, uma vez que se recorreu ao método *toLowerCase* que permite converter todas as letras para minúsculas, uniformizando as *strings* introduzidas e facilitando o reconhecimento.

Figura 10 – Imagem em caso de a palavra não ser reconhecida.

Relativamente à tradução Gesture – Word, esta é dividida numa tradução recorrendo a uma combobox para a escolha dos gestos, em que nela são apresentados os gestos disponíveis na base de dados, e uma tradução recorrendo à recolha do gesto efetuado pelo utilizador. Nesta, o utilizador faz os gestos que pretende identificar, sendo os sinais de EMG captados, processados e convertidos para DC pelo que a aplicação Java apenas recebe duas variáveis (chars) enviadas pelo Arduino e que permitem a determinação do gesto efetuado pelo utilizador.

Após identificado são mostrados o gesto e a respetiva tradução, reproduzindo um áudio e mostrando a string correspondente em inglês e português.

Class Play

A classe *Play* implementa toda a interface gráfica e funcionamento do jogo Pedra Papel Tesoura.

Em primeiro lugar, este apresenta um menu do jogo, no qual se espera que o utilizador preencha o seu número e número de rondas total que pretende jogar. Para tal, são feitos diversos testes aos *inputs* introduzidos pelo utilizador: este tem obrigatoriamente de preencher o campo do nome e, quanto ao número de rondas, este tem de estar no formato numérico (inserir "seis", por exemplo, não é permitido), e tem de ser entre 1 e 5, inclusive (visto rondas negativas não serem possíveis e o limite acima foi por nós imposto). Neste menu o utilizador tem ainda os botões "*Back*" que permite voltar ao menu inicial da aplicação e o "*Next*" que permite avançar para o jogo e é ao carregar neste que a verificação dos *inputs* vai ser efetuada.

Ao avançar, o utilizador encontra outro menu pré-jogo que lhe permite decidir se vai jogar com o dispositivo de EMG, em que os sinais relativos aos gestos realizados vão ser recolhidos em tempo real, ou sem o dispositivo de EMG, no qual o utilizador escolhe o gesto a aplicar em cada ronda a partir de uma *combobox*. Todo o resto do processamento é igual em ambas as secções.

De seguida, aparece o painel relativo a uma ronda que possui os seguintes elementos: designação do número da ronda, painéis onde vão ser mostrados os gestos do utilizador e do computador, identificados com duas *labels*, e o botão "*Play Round*". É ao carregar neste botão que o utilizador deve efetuar o gesto, sendo que este vai ser lido e processado e, após aproximadamente 2 segundos, demonstrado num dos painéis. Os gestos são mostrados através dos métodos *displayPCSign* e *displayUserSign*. É ainda chamado o método *roundWinner* da classe *Game* para comparação dos gestos e determinação do vencedor e incrementa os seus pontos. Posto isto, o botão "*Show Round Results*" torna-se ativo, levando ao painel relativo aos resultados da ronda em questão.

No painel em causa, apresentam-se os seguintes elementos: designação do número da ronda, painéis relativos ao utilizador e ao computador, em que em cada um vai ser escrito "*Winner*" ou "*Loser*" conforme o caso, explicitação do número de pontos de cada um dos jogadores até ao momento e o botão "*Next*".

A escrita das duas designações nos painéis respetivos é feita através do método *displayWinner* e, em caso de empate, faz o display de "*It's a tie*" em ambos os casos. Por fim, ao pressionar o botão "*Next*", é chamado o método *nextRound* que verifica se já se atingiu o número de rondas finais ou não. Caso ainda não se tenha atingido, volta ao painel relativo a uma ronda, limpando as informações anteriores. Caso o jogo tenha terminado, mostra o painel final.

Quanto ao painel final do jogo este mostra o número de pontos finais de ambos os jogadores e o vencedor final, reproduzindo o som de vitória no caso de ser o utilizador ou de perda no caso de ser o computador. Para além disso, pergunta ao utilizador se este deseja jogar novamente ou não. Caso deseje continuar (botão "*Yes*"), limpa apenas a informação relativo ao número de pontos e volta ao painel das rondas. Caso deseje terminar (botão "*No*"), volta ao menu inicial da aplicação.

Tendo toda a parte digital desenvolvida e implementada, recorreu-se ao programa *Visual Paradigm Project* para a criação de um diagrama de atividades UML relativo ao funcionamento da aplicação, que se encontra presente no anexo II.

III. Resultados e Discussão

A. *Circuito Analógico em Multisim*

Para teste do sinal analógico, recorreu-se ao gerador de sinais do Multisim para a geração de um sinal sinusoidal com características de frequência e amplitude semelhantes às do EMG (250 Hz e 3 mV). Sendo testadas as diferentes etapas de condicionamento de sinal de forma sequencial, analisando o efeito de cada uma delas sobre o sinal sinusoidal.

Na figura seguinte encontram-se representados os resultados das 4 etapas gerais de processamento de sinal que ocorrem no circuito explicado anteriormente.

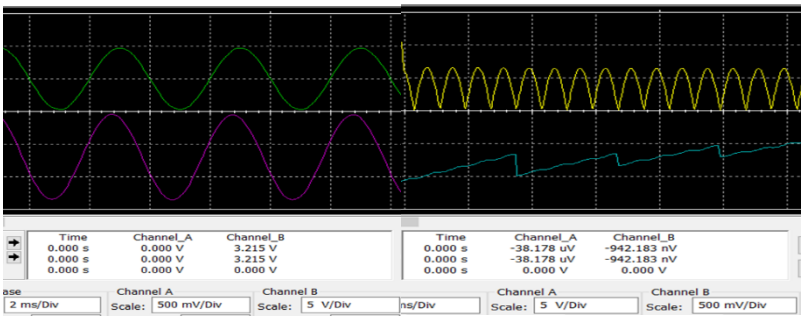

Figura 11– Sinal adquirido e amplificado (verde), após amplificação e passa-alto (violeta); sinal retificado (amarelo); após *smoothing* (azul).

Como anteriormente referido, o projeto foi integralmente desenvolvido por simulação, recorrendo a sinais de EMG previamente recolhidos. Foram utilizados 2 sinais que correspondem às situações de um musculo em atividade e em repouso. Os sinais de eletromiografia utilizados bem como o sinal de output (após processamento) encontram-se representado nas figuras 12 e 13.

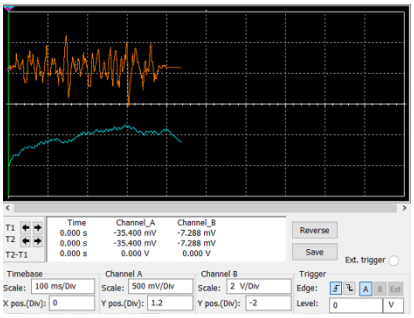

Figura 12 – EMG (laranja) e IEMG (azul) correspondentes ao músculo ativo.

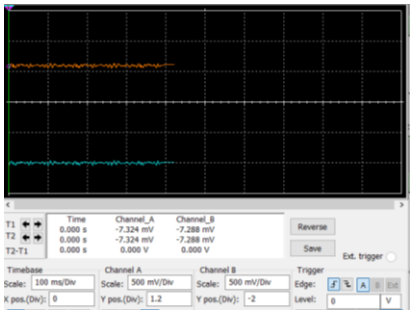

Figura 13 – EMG (laranja) e IEMG (azul) correspondentes ao músculo relaxado.

B. Aquisição e Avaliação do Sinal em Arduino

Relativamente a *Arduino*, não foi possível simular a situação real (usando EMG) embora se tenha tentado diversas abordagens incluído a conversão dos sinais para formato áudio (*.wav*) que funcionariam como *input* do *Arduino*, esta abordagem foi abandonada por falta de componentes elétricos.

No entanto, foi possível testar a interação com Java, através da geração de valores de média aleatórios para ambos os canais e avaliação dos mesmos mostraram-se a funcionar conforme o esperado.

C. Desenvolvimento da Aplicação em Java

Apesar de não ser possível utilizar os sinais fisiológicos que processamos nesta etapa, a alternativa anteriormente mencionada permitiu verificar o funcionamento da comunicação entre a aplicação Java e o Arduino e testar a totalidade da aplicação desenvolvida. A figura 14 permite visualizar a interface gráfica da aplicação que corresponde ao menu inicial.

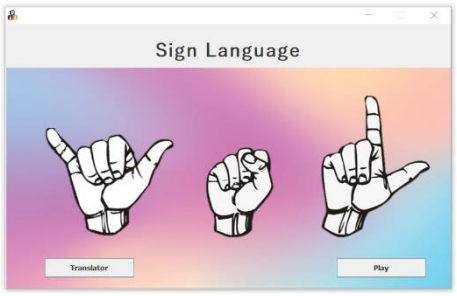

Figura 14 – Menu inicial da aplicação.

A partir deste é possível aceder ao tradutor de linguagem gestual ou ao jogo pelos botões. As figuras 15 e 16 correspondem à interface gráfica do tradutor palavra → gesto e gesto → palavra, respetivamente.

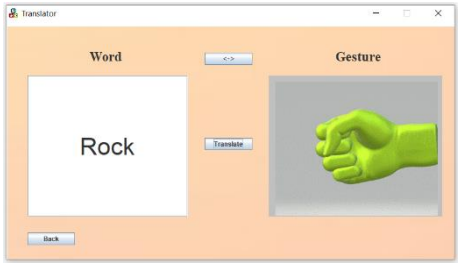

Figura 15 – Tradutor palavra-gesto de "*Rock*".

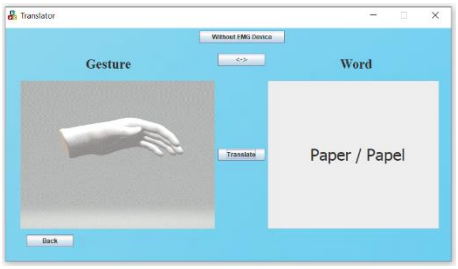

Figura 16 – Tradutor gesto-palavra de "*Paper*".

Relativamente ao jogo "*Rock Paper Scissors*", o utilizador deve inicializar o jogador através da interface da figura 17, inserindo o nome e número de rondas pretendidas.

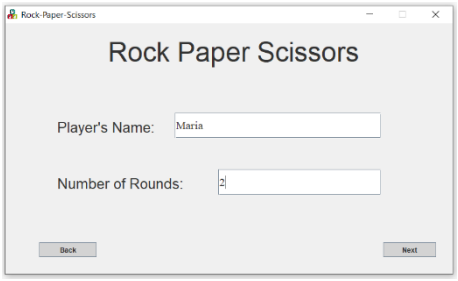

Figura 17 – Menu de inicialização do jogo.

A cada ronda o utilizador faz um dos 3 gestos possíveis para o jogo, sendo este identificado e mostrado na interface em simultâneo com a jogada do *Computer*, como demonstrado na figura 18.

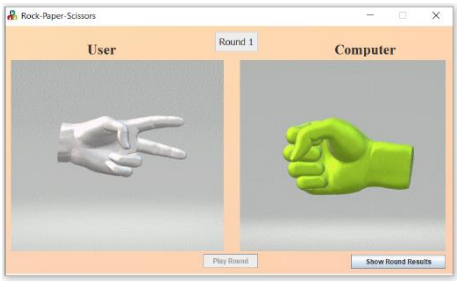

Figura 18 – Demonstração de uma ronda do jogo.

O resultado de cada ronda é mostrado e as pontuações dos jogadores são atualizadas, como se verifica se figura 19.

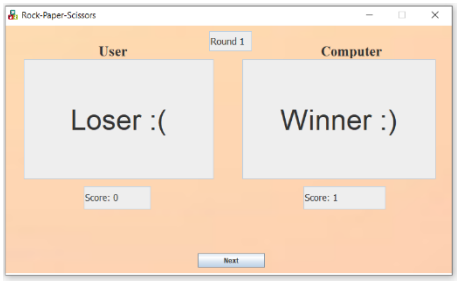

Figura 19 –Resultados de uma ronda do jogo.

No final de todas as rondas o jogo termina e são apresentados os resultados finais, sendo dada também a possibilidade de

voltar a jogar ou não e, neste caso, sendo remetido para o menu inicial.

Figura 20 – Painel final de um jogo.

É de notar que, segundo a tabela 6, há uma combinação de canais para os quais não é possível identificar um gesto: canal 1 ativo e canal 2 inativo. Apesar desta combinação ser improvável numa situação real, verificava-se a sua existência na simulação. Assim, para prevenir eventuais erros de captação dos sinais, caso o sinal não fosse reconhecido era enviada a seguinte mensagem:

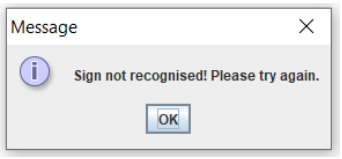

Figura 21 – Mensagem de sinal não reconhecido.

No entanto, para testar o funcionamento do tradutor e do jogo sem necessidade de estabelecer comunicação com o Arduino e de forma a generalizar mais a aplicação, foram adicionadas alternativas à utilização de um dispositivo para captação de EMG. Assim, a seleção de gestos poder ser feita através de *combobox*, como pode ser verificado nas figuras 22 e 24. A alternância entre os modos de seleção no tradutor é feito pressionando o botão "*Use EMG Device"* e no caso do jogo é selecionado o modo de jogo através da interface da figura 23.

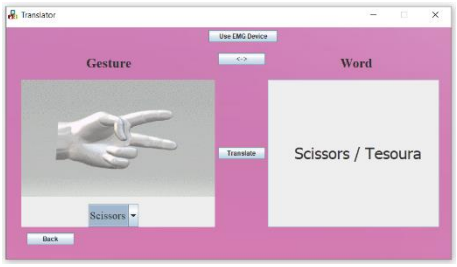

Figura 22 – Tradutor gesto-palavra de "*Scissors*" por *combobox*.

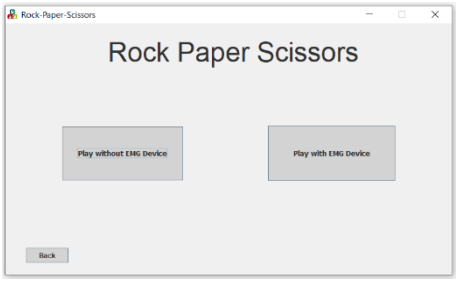

Figura 23 – Opção do jogo com ou sem dispositivo de EMG.

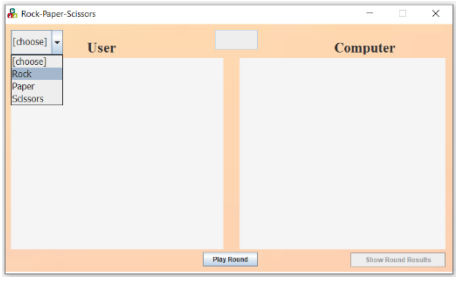

Figura 24 – Opção do jogo sem dispositivo de EMG, ou seja, por uma *combobox*.

Desta forma criou-se uma aplicação intuitiva, útil e bastante abrangente com um caracter pedagógico e lúdico. Visto que, para além de facilitar da comunicação com pessoas surdas/mudas, permite também uma aprendizagem da linguagem gestual e contém um jogo, podendo ser utilizada por qualquer um independentemente de ter ou não os sensores referidos.

IV. Conclusão

No presente trabalho, foi desenvolvido um sistema de aquisição e processamento de sinais de eletromiografia. Adicionalmente, foi realizada uma interface que permite a tradução de diferentes sinais/gestos e a implementação de um jogo Pedra Papel Tesoura, de forma a tornar a aplicação em algo mais lúdico.

Verificou-se que tanto o circuito analógico simulado como a vertente digital funcionaram conforme o esperado, permitindo uma identificação rápida e precisa dos três sinais desejados.

Em suma, o trabalho desenvolvido pode ser de grande interesse tanto para a comunidade surda como para qualquer outra pessoa com interesse na área e acreditamos que os requisitos foram cumpridos.

V. Trabalhos Futuros

Tendo em vista o melhoramento do produto desenvolvido, propõe-se um investimento num estudo que permita relacionar o uso de novos canais, abrangendo a captação de sinais correspondentes a mais músculos, e a identificação de novos gestos, definindo também *thresholds* para estes e aumentando assim a base de dados. Numa etapa posterior seria também interessante a adição de giroscópio para ser possível associar aos diferentes gestos que estão a ser efetuados a própria movimentação da mão, já que a linguagem gestual inclui não só gestos "estáticos" mas também movimentos. Assim seria possível tornar a identificação de gestos mais abrangente e exata.

Para além disso, propõe-se também o recurso a *machine learning* de forma a automatizar ainda mais o processo, o que com mais sinais será certamente necessário.

Referências

[1] Infopédia Línguagem Gestual Portuguesa. Porto Editora. Acedido em: https://www.infopedia.pt/dicionarios/lingua-gestual

[2] Hand Talk. Acedido em: https://www.youtube.com/channel/UCo23Sp0x21pezEBleh5-KYg

[3] Farah Raad Kareem, Mohamed Hassan Abd El Azeem, Mohamehgenedy, Ahmad Mohamed Sameh. “Classification of EMG Signals of Lower Arm (Forearm\ Hand) Motion Patterns Used to Control Robot Hand Movement”. 2017.

[4] Muhammad Zahak Jamal. 2012. “Signal Acquisition Using Surface EMG and Circuit Design Considerations for Robotic Prosthesis”. Em “Computational Intelligence in Electromyography Analysis” páginas 427 – 447.

[5] Semmlow, J, Signals and Systems for Bioengineers: A MATLAB-Based Introduction. Elsevier Science.

[6] Takeshi Tsujimura, Sho Yamamoto and Kiyotaka Izumi (October 17th 2012). Hand Sign Classification Employing Myoelectric Signals of Forearm, Computational Intelligence in Electromyography Analysis - A Perspective on Current Applications and Future Challenges, Ganesh R. Naik, IntechOpen, DOI: 10.5772/51080. Acedido em: https://www.intechopen.com/books/computational-intelligence-in-electromyography-analysis-a-perspective-on-current-applications-and-future-challenges/hand-sign-classification-employing-myoelectric-signals-of-forearm?fbclid=IwAR2BaR2B92ZLAJ77cdkUnuzYl2jlP-45So8dBlRKjL3nQ9e-thfzwEwxuW0

[7] EMG Circuito for a Microcontroller. Advancer Technologies. Acedido em: https://www.instructables.com/id/Muscle-EMG-Sensor-for-a-Microcontroller/

[8] JorgeR.B.Garay, Arshpreet Singh, Moacyr Martucci, Hugo D.H.Herrera, Gustavo M. Calixto, StelvioI.Barbosa, Sergio T.Kofuji, 2017. “Na Electromyography Signal Conditioning Circuit Simulation Experience”

[9] Datasheet INA126. Texas Instruments. Acedido em: https://pdf1.alldatasheet.com/datasheet-pdf/view/56682/BURR-BROWN/INA126.html

[10] Abhishek Nagar, Xu Zhy. 2015 “Gesture Control By Wrist Surface Electromyography”

[11] Biblioteca Java jSerialComm. GitHub. Acedido em: https://fazecast.github.io/jSerialComm/

ANEXOS

Anexo I – Esquematização do Circuito Analógico

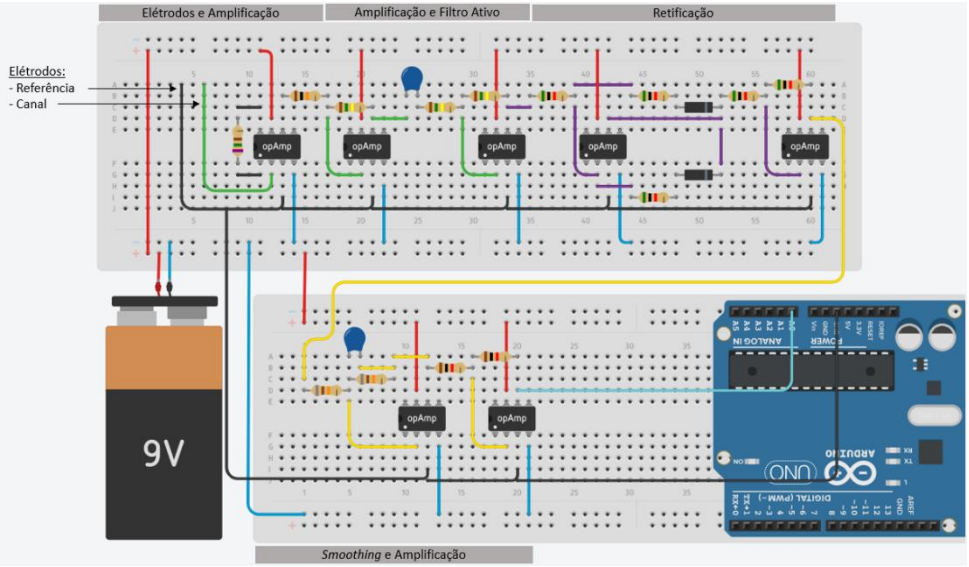

Figura 25 – Esquematização do circuito analógico em *TinkerCad*.

Anexo II – Diagrama de Atividades UML da Aplicação

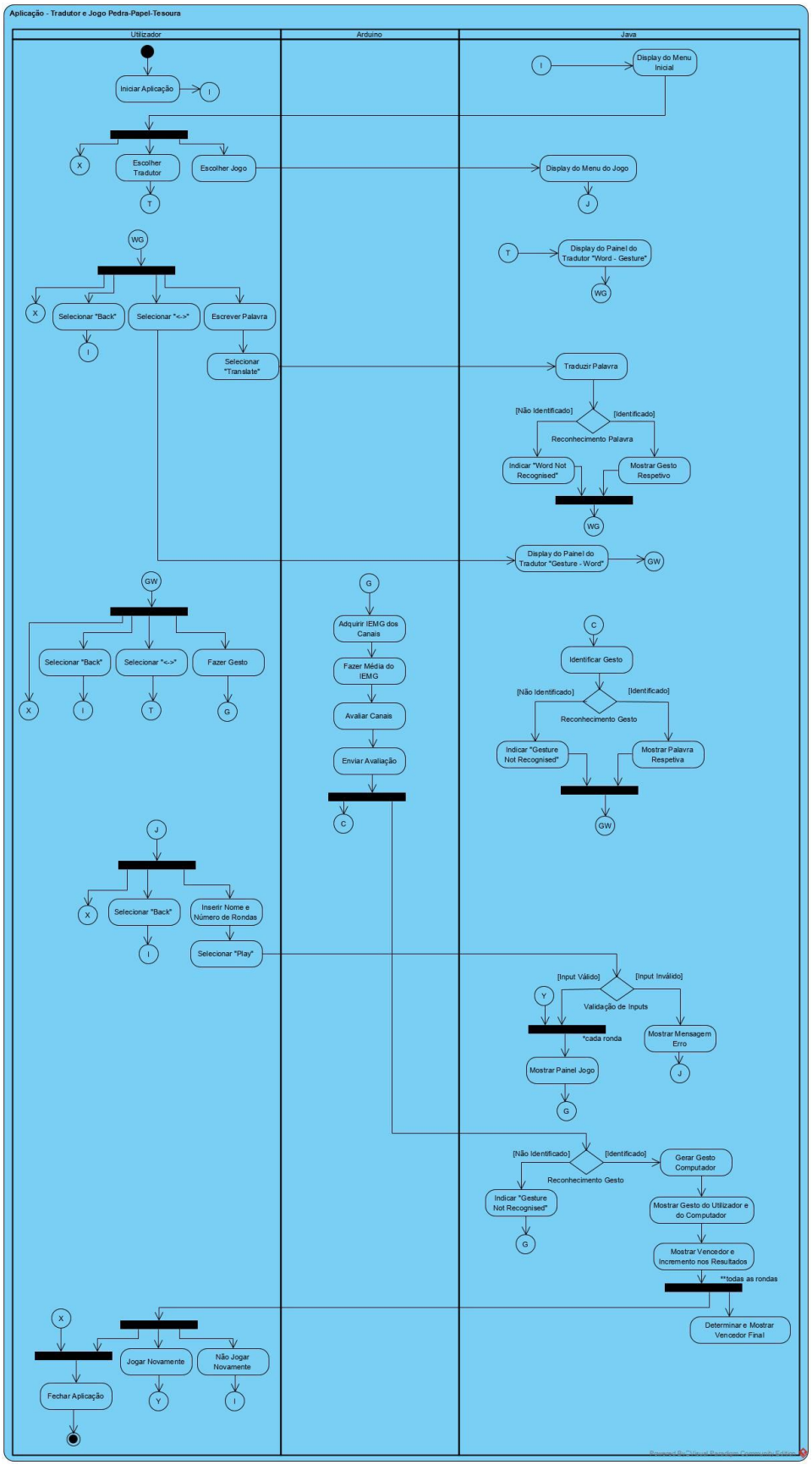

Figura 26 – Diagrama de atividades UML da aplicação desenvolvida.