Gonçalo Jorge Marques Freitas

Licenciado em Engenharia Eletrotécnica e de Computadores

Coerência Digital Neuronal

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Raul Tello Rato, Professor Auxiliar,
Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa

Co-orientador: Fernando José Vieira do Coito, Professor Associado,
Faculdade de Ciências e Tecnologia
da Universidade Nova de Lisboa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2017

Aos meus pais, Amândio Freitas e Maria Celeste Freitas, pelo esforço, dedicação e sobretudo, pela liberdade e confiança depositada nas minhas decisões!
`O que é bom, fica!´ Professor Luís Teles

Agradecimentos

Em primeiro lugar, quero agradecer à Faculdade de Ciências e Tecnologias, ao professor Raul Rato, e a todos os outros docentes que estiveram presentes ao longo deste percurso académico, com os quais, para além das aulas, tive o privilegio de travar laços de amizade e de partilha, engradecendo assim toda esta fase de crescimento intelectual. Aos meus pais, a quem dedico este trabalho, pois apesar de todas as dificuldades, sempre me deram liberdade e o apoio fundamental nas minhas decisões e permitiram que me formasse. Agradeço à minha namorada, Sónia Catarina Azevedo Matos por todas as dores de cabeça e sobretudo por não olhar ao sacrifício no momento de estar presente quando preciso dela. Um agradecimento especial aos meus colegas e amigos, João Ferreira, Ricardo Belchior, João Pires e Miguel Prego por me terem orientado e ajudado neste trabalho. Por último, mas não menos importantes, os meus amigos e colegas de curso, André Estevam, Daniela Oliveira, David Mestre, Duarte Segurado, Fábio Carmo, Fábio Silva, Joana Barruncho e Miguel Prego por todas as vezes que me trocaram as cores das canetas. A todos, o meu Obrigado!

Resumo

O trabalho realizado nesta dissertação consiste no desenvolvimento de uma máquina de estados capaz de levar um pequeno robô a solucionar um percurso. A metodologia proposta inclui a utilização de neurónios artificiais com o intuito de armazenarem estados de memória, tornando assim possível ao robô resolver o percurso por um trajecto mais curto de cada vez que o percorre. A ideia surge com a vontade de desenvolver uma nova forma de computação onde se utilizem neurónios, tornando-a o mais semelhante possível ao funcionamento do cérebro. Desta forma o trabalho desenvolvido está dividido em duas partes. A primeira consiste na análise do percurso e desenvolvimento de uma máquina de estados em Arduíno capaz de levar o robô a resolver o percurso. A segunda, no desenvolvimento dos neurónios em PCB (*Printed Circuit Board*) utilizando um circuito já existente. Estes dois processos permitiram a aquisição de novas competências a nível das máquinas de estados, microcontroladores e, sobretudo, no desenho e desenvolvimento de PCBs, tornando-se assim num trabalho bastante diversificado com focos em diferentes áreas.

Palavras-chave: Arduíno, Máquina de estados, Neurónio

Abstract

The work fulfilled in this thesis is about the development of a state machine able to take a small robot to finish a route. The methodology used contains the utilization of artificial neurons to storage states of memory, making possible to the robot to solve the course by a smaller route the next time he solves it. This idea comes with the will to create a new shape of computation where neurons are used, making it similar to the brain. The work developed is divided in two parts. The first one consists in the analysis of the route and development of the state machine able to solve it. The second one, in the development of neurons on PCBs using a circuit already existent. These two processes allowed the acquisition of new competences on micro controllers and PCB design making the work very diversified with focus on different areas of knowledge.

Keywords: Arduíno, State Machine, Neurons

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O cérebro é provavelmente a melhor ferramenta de computação até hoje existente. Os neurónios, as células que o compõe têm a capacidade de comunicar umas com as outras, transmitindo informações, o sistema de informação veiculado pelos neurónios forma um circuito eléctrico designado por influxo nevoso. A passagem do influxo de um para outro neurónio processa-se não porque exista entre eles continuidade física, mas porque se estabelece uma relação funcional chamada sinapse.

As funções cerebrais tem vindo ao longo dos tempos a ser alvo de estudo, tendo sido desenvolvidas técnicas para aceder aos sinais gerados. Através da história não faltam exemplos de tentativas de como representar e construir uma máquina que se assemelhe ao cérebro humano. Tentar representar o funcionamento cerebral através da tecnologia existente. Quando, em tempos, a máquina de rodas dentadas era a tecnologia mais avançada, pensava-se que o cérebro não passava de uma sequência de rodas encaixadas umas nas outras fazendo tudo funcionar. O mesmo acontece com a máquina a vapor, computadores digitais e atualmente a nível quântico, acreditando que todo o seu funcionamento se rege pelas leis da física quântica.

Para fazer uma máquina capaz de recriar a atividade cerebral não podemos ignorar a base da computação, as máquinas de estados.

1.1 Objectivos

O trabalho a realizar nesta dissertação pretende explicar como foi possível construir uma máquina de estados implementada num Arduíno Nano, capaz de interagir com neurónios artificiais que fazem um pequeno robô deslocar-se em linha recta controlado através de *spikes*.

Para a realização deste projecto foi necessário para além de uma máquina de estados

desenhar e produzir neurónios artificias (previamente implementados em *breadboard*) em PCB com o objectivo de reduzir o seu tamanho e facilitar a sua utilização.

1.2 Arquitectura Geral

O arquitectura do sistema foi dividida por blocos. Alguns destes blocos são constituídos por neurónios artificiais que, apesar do mesmo funcionamento geral, foram utilizados para diferentes funcionalidades. Como se pode observar na figura 1.1 a arquitectura é constituída pelo microcontrolador Arduíno, neurónios do motor, neurónios de memória e o robô.

Esta exposição vai somente evidenciar o desenvolvimento da máquina de estados em Arduíno, garantindo o funcionamento do sistema, e a capacidade de interacção com os neurónios artificiais de memória alterando o seu estado. O bloco dos neurónios do motor, que corresponde uma interface entre a máquina de estados e a ponte H, não é considerada neste documento, visto que o sistema foi desenvolvido com o Arduíno ligado directamente à ponte H e com os neurónios de memória apenas ligados ao Arduíno.

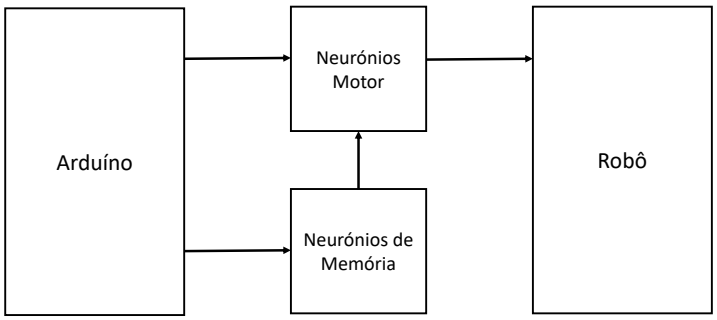

Figura 1.1: Representação da arquitectura geral do projecto

O capítulo seguinte, conta com uma fundamentação teórica da máquina de estados enquanto base da computação e com um estado da arte em torno da investigação do cérebro. A investigação neuronal é vasta e abordar todas as suas vertentes seria impossível. Assim, reduzimo-nos a alguma investigação sobre o tratamento dos sinais neuronais e a um trabalho onde também foram utilizados neurónios artificiais para levar um robô a resolver um pequeno percurso.

Capítulo 2

Fundamentos da Máquinas de Estados e Estado da Arte

2.1 Evolução da Lógica ao Encontro das Máquinas de Estados

O propósito deste trabalho é desenvolver uma máquina de estados com componentes neuronais. As máquinas de estados e todo o seu funcionamento têm como base a lógica no sentido digital. A lógica no sentido clássico foi definida [12] por Alexandre de Afrodisias com base no Organon de Aristóteles.

Logos: termo grego, que significa lógica, pensamento, discurso. Ao pensar, inferimos afirmações de outras afirmações. O que a Lógica nos permite fazer é avaliar se essas afirmações foram ou não correctamente inferidas, permite identificar as falhas dos raciocínios e garantir que partindo de premissas verdadeiras se chega a conclusões verdadeiras.

No decurso deste capítulo vão ser apresentadas as personalidades que directamente influenciaram a evolução da estrutura da Lógica formal até à actualidade. O foco, é determinar através do estudo da Lógica o desenvolvimento das máquinas de estado.

2.1.1 Aristóteles (384-322a.c.)

Aristóteles, foi um dos mais importantes filósofos gregos, dedicou-se a quase todas as áreas do saber e foi o criador da Lógica enquanto ciência.

Na lógica aristotélica, foi combinado uma afirmação geral e uma mais específica sendo possível gerar uma conclusão acerca dessas duas frases. O exemplo mais conhecido será: "Todos os homens são mortais. Sócrates é um homem, Logo, Sócrates é mortal. "

Com isto surge o conceito *syllogismo*[16]. Raciocínio dedutivo constituído por três termos e três preposições, duas premissas e uma conclusão, que decorre das premissas. A

lógica aristotélica foi uma etapa muito importante no desenvolvimento da Lógica, pois devem-se a Aristóteles noções fundamentais como a da forma lógica e a distinção entre validade e verdade. A Lógica evoluiu, sobretudo a partir do século XVII, e, principalmente durante os séculos XIX e XX, tendo sido construídos processos novos de formalização.

A lógica proposicional é a parte da Lógica formal que estuda os argumentos dedutivos, cuja validade depende exclusivamente da relação entre formas proposicionais. A combinação de afirmações - *"statements"*, necessita então de conectivas proposicionais "ou", "e", "se", "então" área desenvolvida por *Chrysippus*[8].

2.1.2 Gottfried Leibniz (1646-1716)

Vários autores surgiram com novas e variadas versões do *syllogismo*.

Leibniz surge com a ideia de tratar as frases como equações [8], tal como na álgebra. Desta forma torna-se possível a comparação e substituição de "sujeitos" entre proposições.

A introdução da igualdade entre frases permite abranger os *syllogismos* que surgiram até à altura com a adição de duas novas leis na forma: *a=not(not)a* - Se Sócrates é mortal então Sócrates não é imortal, *'a é b' = 'not-b é not-a'* - Sócrates é um homem significa que se não és um homem então não és Sócrates [21].

Com esta adição, Leibniz consegue chegar à veracidade de um encadeamento de frases que anteriormente se consideravam infinitas.

2.1.3 George Boole (1815-1864)

Em 1847 George Boole e Augustus De Morgan publicam trabalhos onde propõem formas algébricas de representar e tratar a Lógica existente até então, como dito numa carta enviada de Morgan para Boole, *"mechanical modes of making transitions, with a notation which represents our head work"* [3].

A abordagem desenvolvida por Boole tem como base os métodos algébricos já existentes, mas ainda assim, os seus resultados revelam ser um novo sistema bastante diferente da álgebra até ai conhecida.

2.1.3.1 Claude Shannon (1916-2001)

O trabalho de Boole é utilizado por Claude Shannon na sua tese de mestrado *A symbolic analysis of relay and switching circuits* [20], tendo aqui sido reconhecido e dando origem à actual teoria de síntese de circuitos Booleanos que, para além de ser uma área de estudo da Matemática, se revelou uma ferramenta extremamente importante no desenvolvimento e estudo da electrónica e arquitectura computacional.

2.1.4 George Cantor (1845-1918)

Com a evolução da Lógica e da Matemática surgiu a necessidade de agrupar todas as teorias e pensamentos desenvolvidos numa única, de onde fosse possível deduzir todas

as outras.

A teoria de conjuntos de Cantor é um passo nesse sentido. A sua teoria diz que é possível formar conjuntos de elementos sem haver a necessidade de existir algo em comum entre eles. Cada conjunto é um grupo de elementos independentes que podem ser comparados com elementos de outros conjuntos. Com eles é possível realizar as operações lógicas que já antes eram utilizadas para dados individuais como o 'e', 'ou' e até mesmo a 'negação'.

George Cantor, através da sua teoria dos conjuntos, demonstra que os mesmos podem ser infinitos e ainda assim de tamanhos diferentes.

2.1.5 Ludwing Wittgenstein (1889-1951)

Até este momento, a Lógica e o seu estudo procura fundamentar a Matemática e resolver problemas linguísticos. Ludwing Wittgenstein surge em tempos diferentes, com dois trabalhos, contidos no seu livro *Tractatus Logico-philosophicus*, que viriam a introduzir duas novas formas de tratar a Lógica.

O seu primeiro trabalho vem adicionar as formas lógicas já existentes, a lógica filosófica. Para Wittgenstein, a lógica não é nada mais do que algo que existe em comum entre a linguagem e o mundo onde no encontramos e sem ela, aquilo que dizemos e descrevemos não faria sentido.

Com isto, e depois de ter lido que nos tribunais em Paris pequenos modelos de automóveis eram utilizados para identificar a posição das viaturas em acidentes de viação, Wittgenstein substitui a linguagem pelas imagens afirmando que a única coisa que uma imagem necessita de ter em comum com a realidade por forma a ser capaz de a retratar é a sua "*form of reality*" [8]. Por exemplo, a imagem de uma bola de ténis não é efetivamente uma bola de ténis física mas é uma representação lógica da mesma.

O seu segundo trabalho foi sem dúvida o mais importante para o desenvolvimento da lógica computacional e para a engenharia: as tabelas de verdade [15]. Seguindo os trabalhos de Frege sobre as conectividades lógicas, Wittgenstein desenvolve as tabelas de verdade, demonstrando ser uma forma mais simples de as demonstrar em vez do jogo de palavras utilizado até então. Apesar do primeiro uso para as tabelas de verdade ter sido para simplificar a representação da conectividade "&"(*and*), rapidamente foram também utilizadas para as conectividades "v"(*or*) e "¬"(*not*) .

As tabelas de verdade de Wittgenstein estão na base do desenvolvimento dos sistemas lógicos e das suas portas lógicas ainda hoje utilizadas.

2.1.6 David Hilbert (1862-1943)

Desenvolvimentos matemáticos como a teoria de conjuntos de George Cantor e o cálculo proposicional de Frege foram tentativas de justificar a Matemática através da Lógica.

Uma nova tentativa surge por parte de David Hilbert com o seu trabalho "*Proof Theory*". Hilbert estava interessado no que cada ramo da Matemática tinha em comum entre

si. Cada ramo, tem como base uma série de axiomas ou regras que são tomadas como verdadeiras e pelas quais todas as seguintes se regem. Hilbert procurava encontrar uma forma de provar a consistência de qualquer lista de axiomas, assim criando uma base de onde todas as áreas da Matemática derivassem.

O seu trabalho levou a pouco mais do que resultados iniciais pois atraíram a atenção de Kurt Gödel que veio provar que o plano de Hilbert não funcionaria para todos os ramos da Matemática.

2.1.7 Kurt Gödel (1906-1978)

Kurt Gödel, no estudo da Aritmética chega à conclusão que qualquer sistema complexo o suficiente para servir de base para a Aritmética estaria incompleto [8]. No final do século XIX. Depois de diversas tentativas da sua parte, Gödel dá como impossível a tentativa de criar um ponto comum de onde derivam todos os ramos da Matemática.

O seu trabalho é reconhecido como a origem da Lógica actual. Neste momento a Lógica está dividida em três partes: filosófica, simbólica e matemática. Apesar dos descobrimentos de Gödel, o ponto de ligação entre estes três conceitos é a sua dependência na *"Prof Theory"*

2.1.8 Alan Turing (1912-1954)

Turing, em 1935, participou num curso de lógica matemática que teve como tema final os trabalhos de Gödel e Hilbert. Interessado pelas suas matérias (sobretudo com os objectivos e tentativas de Hilbert), Turing surge com o seu primeiro trabalho *"On computable numbers, with an application to the Entscheidungsproblem"* [14] onde desenvolve o conceito de uma máquina capaz de resolver cálculos, que se designa a "Máquina de Turing".

Segundo Turing, a máquina teria a capacidade de ler símbolos gravados numa fita e modificá-los com base no que leu e na sua configuração interna ou estado [1]. Teria obviamente também de ter a capacidade de parar quando chegasse a uma conclusão. Para isso, consoante o símbolo que estivesse a ler e o seu estado actual, a máquina escreveria um novo símbolo na posição em que a fita se encontraria, andava para trás e para a frente na fita de forma a alterar outros símbolos e por fim mudava para um estado final e parava.

Turing teve a oportunidade de pôr em prática os seus estudos com a necessidade de decifrar as mensagens que eram transmitidas entre os alemães durante a Segunda Guerra Mundial.

Como a codificação das mensagens era alterada todos os dias, Turing teve assim a necessidade de desenvolver uma máquina que decifrasse as mensagens mas que pudesse ser adaptável a novas codificações sempre que necessário.

Com isto, tem a ideia de que as máquinas como esta deveriam ter a capacidade de auto-aprendizagem. Esta conjectura leva a linguagens de programação como o PROLOG e aos princípios da inteligência artificial [8].

Além de responsável pelo conceito de que máquinas poderiam ser programadas para a resolução de problemas e pelo início da inteligência artificial, é também o criador dos tubos de vácuo utilizados para armazenar informação electronicamente. Nos dias de hoje são utilizados transistores no seu lugar.

2.1.9 Stephen Kleen (1909-1994)

Com Kleene chegamos finalmente às máquinas de estados, após Turing introduzir pela primeira vez a noção de estado com a sua máquina. Apesar da máquina de Turing ser mais avançada que as máquinas de estados de Kleene (pois as máquinas estados estão limitadas pela quantidade de estados nela existentes), é este último que, com o seu teorema em torno de linguagens, projeta as máquinas de estados para aquilo que são hoje [7].

2.1.9.1 Linguagens

Em qualquer linguagem são definidos três tipos de entidades: letras, palavras e frases. A junção de algumas letras formam palavras, a composição de algumas palavras formam frases, porém, nem todas as letras formam palavras válidas e o mesmo acontece com a junção de palavras para a formação de frases. Deste modo, existe um consenso em conhecer as junções que são válidas e quais não são.

Da mesma forma, foram criadas e definidas as linguagens computacionais. Estas linguagens vêm definir os conjuntos de entrada de uma máquina de estados, podendo as mesmas ser alfabetos de qualquer língua, símbolos, números ou qualquer outro caracter reconhecido por uma máquina e pelas pessoas.

2.1.9.2 Teorema de Kleene

O teorema de Kleene diz-nos que se uma linguagem poder ser definida por expressões regulares, autómatos finitos ou gráficos de transição, então, pode também ser definida por qualquer um dos outros dois métodos. Este teorema é o resultado mais importante na teoria dos autómatos finitos, tendo assim definido que eventos podem ou não ser representados em máquinas de estados [2].

Encontramo-nos desta forma, muito próximos das máquinas de estados actuais. Todas as descobertas feitas pelos nomes referidos anteriormente fazem parte de um contributo essencial, referindo que foram de tal forma inovadoras que ainda hoje algumas delas, como as tabelas de verdade e os gráficos de transição, são utilizadas. Existiram outros após Kleene que estudaram e desenvolveram as máquinas de estados mas a sua essência e base ficam desta forma definidas.

2.2 Máquinas de Estados com *Output*

Existem vários tipos de máquinas de estados, todas elas contêm um número finito de estados, um estado inicial, uma sequência de valores de entrada e uma função que define as mudanças de estados. Nesta secção vão ser referidas as máquinas que produzem um *output*.

Definição 1. *Uma máquina de estados finita* $M = (S, I, O, f, g, s0)$ *consiste num conjunto finito de S estados, um conjunto finito de entrada I, um conjunto finito de saída O, uma função de transição f que atribui a cada par de valores de entrada e estado, um novo estado, uma função de saída g que atribui a cada par de valor de entrada e estado um output correspondente, e um estado inicial s0. [18]*

Existindo uma sequência de *inputs* I que levam à evolução do funcionamento da máquina de estados, em que pode existir um *output* por cada transição, concluí-se que cada conjunto de entrada I, de tamanho definido, produz um conjunto de saída O sempre de dimensão igual ou inferior ao de entrada pois nem todas as transições têm obrigatoriamente de conter um *output*.

O funcionamento e a evolução de uma máquina de estados pode ser representado através de uma tabela de estados. Para tal precisa-se de especificar, para cada par *input* e estado, o respectivo estado seguinte e *output* resultante.

Outro modo, e provavelmente o mais utilizado, é a representação através de um gráficos de transição. Nestes gráficos os estados são representados por círculos e as transições por arcos onde é identificado o *input* que gerou a transição e o respectivo *output* separados por vírgulas.

Na tabela 2.1 e na figura 2.1 pode-se observar uma máquina de estados representada por uma tabela de estados e um gráfico de transição, que só produz o *output* 1 quando à entrada tem a sequência 111.

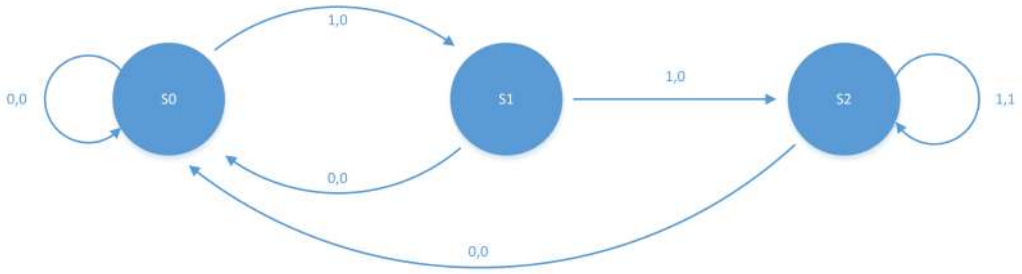

Figura 2.1: Gráfico de transição com output

Tabela 2.1: Tabela de estados referente ao exemplo da figura 2.1

	f		g	
	Input		Output	
State	0	1	0	1
s0	s0	s1	0	0
s1	s0	s2	0	0
s2	s0	s2	0	1

2.2.1 Tipos de Máquinas de Estados com *Output*

Na modelação de inúmeros sistemas, como computadores e robôs têm vindo a ser desenvolvidos vários tipos de máquinas de estados. A máquina de estados aqui mencionada, em que o seu *output* ocorre na transição entre estados, é normalmente conhecida como máquina do tipo Mealy. Existe outro tipo de máquina de estado com *output* onde o seu *output* depende unicamente do estado em que a máquina se encontra e não da sua transição. Este tipo de máquinas são normalmente chamadas de máquina do tipo Moore. Ao desenhar o gráfico de transição deste tipo de máquinas, o seu *output* aparece representado dentro dos círculos de cada estado e não nos arcos de transição.

2.3 Máquinas de Estados sem *Output*

Uma das maiores aplicações das máquinas de estados é no reconhecimento de linguagem. Esta aplicação tem um papel fundamental no funcionamento dos compiladores das linguagens de programação verificando se o código que o utilizador está a tentar compilar cumpre com todas as regras da linguagem em que está a escrever.

Na secção anterior foi dado um exemplo dessa mesma aplicação com uma máquina de estados que dava como *output* o valor 1 quando uma certa sequência de valores era colocado à sua entrada. Contudo, existe outro tipo de máquinas de estados utilizadas exatamente com o propósito de reconhecer linguagens, as máquinas de estados sem *output*. Estas, em vez de devolverem um *output*, têm um conjunto de estados considerados finais. Assim, a sequência de entrada é identificada como uma frase ou palavra, unicamente quando a máquina chega a um desses estados finais. [18].

Definição 2. *Uma máquina de estados sem output* $M = (S, I, f, s0, F)$ *consiste num conjunto de estados S finitos, uma sequência de entrada I finita, uma função de transição f que atribui um novo estado a cada par constituído por um estado e um valor de entrada, um estado incial s0, e um subconjunto F de S correspondente aos estados finais.*

As máquinas de estados sem *output* podem também ser representadas por tabelas de estados ou gráficos de transição. Ao representar estas máquinas com gráficos de transição, deixa de existir a representação do *output* nos arcos (pois este não existe) e os estados finais são representados com círculos duplos.

2.4 Aplicações das Máquinas de Estados

Alguns exemplos de aplicações das máquinas de estados com *output* são:

1. Robôs - Este é um exemplo onde os *inputs* da máquina de estados são normalmente valores numéricos vindos dos seus sensores e os *outputs* os actuadores.

2. Jogos de vídeo - Aqui os *inputs* podem ser vários, desde acontecimentos no próprio jogo até as teclas premidas pelo jogador podendo estes ser valores numéricos ou caracteres. Os seus *outputs* podem ser diversas acções no jogo em questão.

3. Máquinas de venda automática - A máquina recebe como *input* moedas do utilizador e tem como *output* o movimento do motor por forma a libertar o artigo selecionado.

4. Reconhecimento de linguagem - O reconhecimento de linguagem é uma das grandes aplicações das máquinas de estados. Neste caso o *input* são caracteres e o *output* a confirmação de que a sequência de entrada foi, ou não, reconhecida.

2.5 Estudo dos Neurónios e do seu Processamento

A segunda componente que se pretende desenvolver, são os neurónios. Com o objectivo de fazer uma máquina de estados que actua sobre neurónios artificiais faz todo o sentido que se perceba um pouco do seu funcionamento.

É fácil encontrar trabalhos em torno do funcionamento dos neurónios. Existem experiências em diversas áreas desde o estudo e análise teórica do seu funcionamento até à tentativa de fazer leituras que possam controlar robôs. Contudo é difícil encontrar trabalhos com a mesma finalidade do que se pretende. Nesta secção vão ser apresentados alguns trabalhos que, apesar do seu interesse, não têm ligação directa às máquinas de estados e ao controlo de neurónios.

Apenas um dos trabalhos descritos se aproxima daquilo que foi executado. Mas, como se poderá constatar, a abordagem na utilização dos neurónios não é a mesma.

2.5.1 Neurónio e *Spikes*

Há semelhança da máquina de estados, o cérebro, orgão coordenador, processa informação, e para tal necessita da existência de uma linguagem com a qual trate essa informação. O *output* de um neurónio consiste em pequenos impulsos eléctricos, "*spikes*".

Spike, é uma palavra inglesa porém é usada regularmente na Língua Portuguesa,tendo em conta este factor não vai ser escrita em itálico nas próximas vezes que surgir neste documento.

Um conjunto de spikes é designado por "comboio de spikes". O libertar de spike por parte de um neurónio ocorre em intervalos de tempo distintos pelo que o intervalo entre spikes de uma dada sequência não é sempre igual [17]. São estes comboios de spikes com

diferentes intervalos que são processados pelo cérebro. O cérebro não processa spikes individuais mas sim grupos dos mesmos, que devido ao tempo aleatório com que são gerados, serão diferentes uns dos outros.

Na figura 2.2 (B) pode-se observar um comboio de *spikes*. Ao dividir o comboio ao meio criando um comboio de spikes do 0 ao 1 e outro do 1 aos 2, podemos verificar que são diferentes, ou seja, "símbolos" diferentes a processar pelo cérebro.

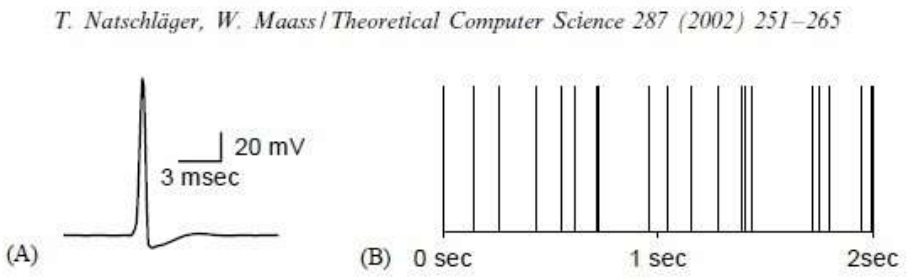

Figura 2.2: (A) Impulso de um neurónio (*spike*). (B) Comboio de *spikes* gerado por um neurónio

2.5.2 Aquisição de Sinais Neuronais

Existem várias formas de adquirir sinais neuronais, uma delas é através da pele (EEG) onde é possível medir valores de corrente, técnica de estudo não invasiva, que têm permitido uma aumento drástico na quantidade de informação recolhida nas últimas décadas. Uma outra forma de aquisição de sinais neuronais, é no contacto direito com o cérebro (invasivamente), através de electrodos na sua massa cinzenta. As técnicas de estudo têm vindo a evoluir aumentando a quantidade de neurónios possíveis de se ler simultaneamente. [22]

2.5.3 Processamento de Sinais Neuronais

O estudo do cérebro e dos neurónios tem inúmeros objectivos, e para cada objectivo existe a necessidade de tratar tipos de informação diferentes. Os tipos de informação podem ser divididos em duas categorias: espaciais e temporais [6]. A informação espacial refere-se a informação de uma zona específica do cérebro, que processe a informação que se quer tratar. A informação temporal refere-se ao tratamento de sinais dos neurónios tal como os comboios de spikes, que são tratados no tempo. Estes dois tipos de informação vão ser abordados, dando destaque à informação temporal pois é esta, que maior ligação tem ao nosso objectivo.

2.5.3.1 Estimação do campo de receptividade (*Receptive Field Estimation*)

O campo de receptividade de um neurónio define como a atividade do mesmo se altera consoante o estimulo que lhe é aplicado. É importante caracterizar o campo receptivo de um neurónio pois, dependendo da função de cada um, a sua informação terá diferentes formas de ser representada. Por exemplo, a informação recolhida de um neurónio pertencente ao sistema visual, é representada num campo receptivo espaço-temporal a duas dimensões. No caso da informação referente ao sistema auditivo já é representado como um campo receptivo espectro temporal. [6]

Um exemplo, é o mapiamento dos campos receptores da área visual V1. A area V1 é o primeiro nível do processamento visual no cérebro contento um mapa completo do campo visual observado pelos olhos [4]. O mapeamento desta área é extremamente importante para compreender como o sistema visual representa e transmite informação no seu primeiro nível.

2.5.3.2 Detecção do ponto de mudança (*Change-Point Detection*)

O objectivo da detecção do ponto de mudança é detectar uma alteração no cérebro que é dependente de um acontecimento estocástico. Não importa somente saber se a alteração ocorreu ou não mas sim o momento em que acontece. Esta técnica é utilizada por exemplo na detecção de ataques de epilepsia [5].

2.5.3.3 Interface Cérebro-Máquina

O objectivo da investigação sobre interfaces cérebro-máquina é o de criar uma ligação entre o cérebro e o sistema nervoso com mundo exterior. A necessidade desta ligação surge no auxílio a pessoas com dificuldades motoras, criando assim a possibilidade de controlar peças mecânicas através de leituras do nosso cérebro ou até mesmo forçar o funcionamento de músculos ou sistemas sensoriais através da inserção de impulsos artificiais no nosso sistema nervoso. Para tal podem ser utilizados sistemas como os comboios de spikes ou EEG [10], [9].

Um sistema de interface cérebro-máquina consiste nas seguintes etapas: aquisição de sinal, pré-processamento e melhoramento, extração de características, processamento de sinal e interface de controlo [6]. Um exemplo deste sistema é o controlo de uma garra mecânica. Neste tipo de aplicação é necessário um sistema em anel que leia a informação neuronal, a processe, faça mover a garra e por fim devolva uma resposta visual e táctil ao cérebro [19].

2.6 Trabalhos Relacionados

Nesta secção falar-se-á dum trabalho onde, utilizando neurónios artificiais, se desenvolve uma solução para levar um robô a decidir o melhor percurso possível de um ponto A a

um ponto B e percorrê-lo [13]. Este percurso ou labirinto consiste numa zona plana com diversos obstáculos onde aleatoriamente, é decidido o ponto de chegada e de início para o robô. A resolução do problema tem como base os algoritmos de propagação de *wavefronts*.

Os neurónios utilizados neste trabalho estão desenvolvidos e interligados entre si dentro dum circuito integrado. Este circuito integrado contém 100 neurónios e 30000 sinapses. Cada neurónio contem um variado número de sinapses ligadas a uma dendrite, em que, nem todas se encontram activas. A sinapses são feitas utilizando *floating gate transistors* para desta forma ser possível armazenar nelas uma carga de referência.

A resolução do problema passa por 3 etapas: Programação dos neurónios para mapear o labirinto, identificação da melhor rota a percorrer e por fim execução da mesma. Vão ser referidos os dois primeiros pontos.

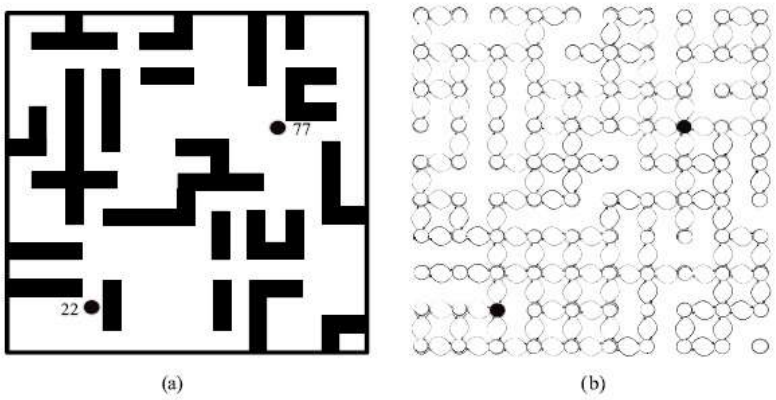

Figura 2.3: (A) Exemplo de um labirinto visto de cima sendo o ponto 22 o local de partida e o 77 o de chegada. (B) Exemplo de como o mesmo labirinto se encontra mapeado na matriz de neurónios

Para ser possível calcular o melhor caminho a percorrer pelo robô é necessário primeiro mapear o labirinto e é nesta etapa que entram os neurónios. Os neurónios estão interligados através das suas sinapses em forma de matriz. Esta matriz representa o labirinto e nas posições onde se encontram os obstáculos as ligações (sinapses) entre neurónios são desactivadas. Um exemplo deste mapeamento pode ser visto na figura 2.3. Este mapeamento é feito/programado utilizando uma bancada desenvolvida exactamente para este propósito. Assim se percebe que é necessário conhecer o espaço onde o robô vai operar antes do mesmo ser lá colocado. É necessário referir que neste mapeamento o ponto inicial e final do percurso têm também de estar já definidos.

Após programados os neurónios o robô é colocado no ponto inicial e é iniciada a fase de descoberta da melhor rota a percorrer. Para tal, o ponto (neurónio) que foi programado como o local onde o robô deve chegar, emite *wavefronts* que se propagam de neurónio em neurónio. Cada neurónio vai reter o momento em que recebeu a *wavefront* criando assim um percurso de valores crescentes até chegar ao ponto de origem (figura 2.4). Ao contrário de outros sistemas de mapeamento que também utilizam *wavefronts*, este método não

necessita de uma segunda fase para processar qual o melhor percurso a fazer pois assim que a *wavefront* chega ao ponto de partida o melhor caminho já está calculado.

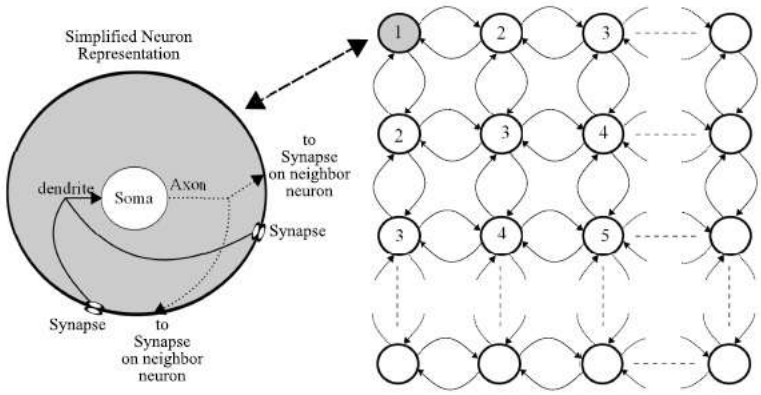

Figura 2.4: Exemplo da estrutura de um neurónio e como se o percurso é representado até chegar ao ponto inicial

Como se pode verificar, apesar da utilização de neurónios para a resolução de um percurso/labirinto, estes não são utilizados para controlo do robô mas sim para mapear o local onde o mesmo se encontra.

2.7 Conclusão

A evolução da Lógica, esclarece a forma como as máquinas de estado surgem na base da computação atual. No estudo sobre o funcionamento do cérebro há descobertas diárias sendo praticamente impossível de acompanhar todos os resultados e informações que daí advêm.

Da literatura encontrada, a investigação de interfaces cérebro-máquina é o exemplo mais próximo do que se pretende deste trabalho pois serão utilizados comboios de spikes (gerados artificialmente) para o controlo de um robô. Apesar da forte utilização de neurónios artificiais em [13], o seu uso tem como base o mapeamento do percurso e não no controlo das acções a tomar.

Capítulo 3

Desenvolvimento e Implementação da Máquina de Estados

Neste capítulo irá ser descrita a implementação de uma máquina de estados capaz de deslocar um robô ao longo de um caminho pretendido, passando por todos os obstáculos que surjam.

Para além de levar o robô a seguir o caminho sem se perder, esta máquina de estados teve também de detectar quando o percurso errado foi percorrido e posteriormente enviar um sinal de controlo para os neurónios, para que no futuro o robô saiba qual o trajecto mais adequado a percorrer.

O processo acontece em duas fases: na primeira fase o robô foi colocado num ponto do percurso e terá de chegar ao seu fim, independentemente do caminho que faça. Na segunda fase, após ter terminado o percurso, voltou a ser colocado no seu ponto inicial e desta vez, teve que chegar ao fim pelo caminho que anteriormente aprendeu. No caso de ter sido colocado num ponto de partida diferente do primeiro, nesta segunda execução deve detectar que não percorreu o caminho mais curto possível e apreender uma nova solução para o percurso que está no momento a realizar. De notar que o conhecimento de qual o percurso mais curto não é armazenado em software mas sim nos neurónios, sendo que a função do Arduíno é colocar essa informação nos neurónios responsáveis para o efeito. Para a máquina de estados, a primeira e a segunda resolução do percurso em nada diferem.

3.1 Hardware do Robô

O robô consiste numa estrutura metálica com duas rodas laterais (às quais estão ligados os motores) em borracha que promovem o movimento e um *ball caster* na zona frontal para apoio. Contém o sensor QTR-8RC da Pololu, montado na parte inferior do robô,

utilizando espaçadores, responsável por detectar o percurso. O sensor encontra-se ligado a um Arduíno nano responsável por ler os valores do sensor e por sua vez controlar uma ponte H que acciona os motores. Tanto o Arduíno como a ponte H encontram-se montados numa *breadboard* sobre o carro.

A estrutura completa conta com 1.65cm de comprimento, 1.57cm de largura e 6.5cm de altura. A altura é referente à estrutura física não tendo em conta o espaço vertical ocupado pela curvatura dos cabos utilizados nas ligações. O robô pode ser observado na figura 3.1.

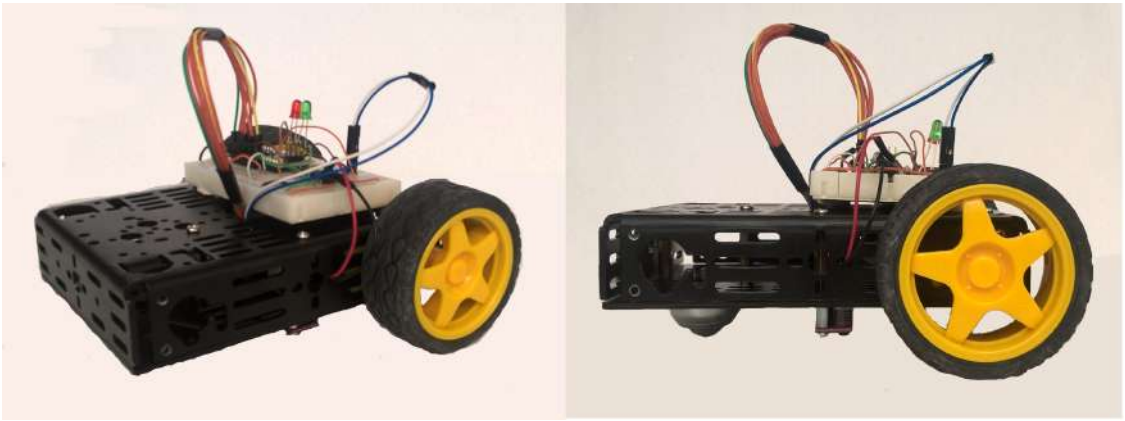

Figura 3.1: Estrutura do robô com todos os seus componentes

3.1.1 Arduíno

O Arduíno é uma plataforma *open-source* utilizada para projectos de electrónica e telecomunicações que contém um micro-controlador possível de programar através de um IDE dedicado. Esta plataforma tem-se vindo a tornar bastante popular devido à curta curva de aprendizagem necessária para ser utilizado, algo que outros micro-controladores não oferecem. Existem várias versões de placas arduíno, porém, apesar de variarem nas quantidades disponíveis, todas elas oferecem portos digitais e analógicos para utilização como *input* e *output*. O Arduíno tem também a capacidade de fornecer valores de tensão de 3.3V ou 5V. Por sua vez pode ser alimentado por USB ou por uma fonte externa entre 7V e 12V.

Das diversas placas Arduíno a mais comum é a versão Uno pois oferece tudo o que é necessário para projectos simples de iniciação. Para o desenvolvimento desta dissertação não foi o suficiente foi então utilizado um Arduíno Nano como mostrado na figura 3.2. Apesar de mais pequeno contém um maior número de portas analógicas, os quais eram necessários.

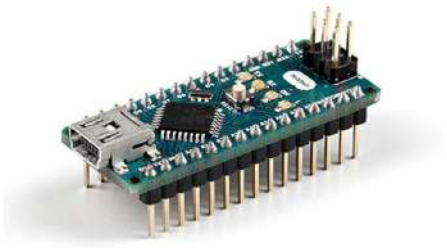

Figura 3.2: Arduino Nano

3.1.2 Sensor QTR-8RC

O sensor QTR-8RC consiste numa pequena régua com 7.5cm de comprimento e 1.2cm de largura que contém 8 emissores e receptores (fototransistores) de infravermelhos, figura 3.3. Apesar de designado como um sensor de seguimento de linha, pode também ser utilizado como sensor de proximidade. O QTR-8RC deve ser alimentado no intervalo de 3.3V a 5V fornecendo, à saída de cada sensor o valor de alimentação quando detecta a linha. Para que tenha um funcionamento correcto deve ser montado paralelo à superfície onde vai ser detectada a linha, a uma distância máxima de 9.5mm.

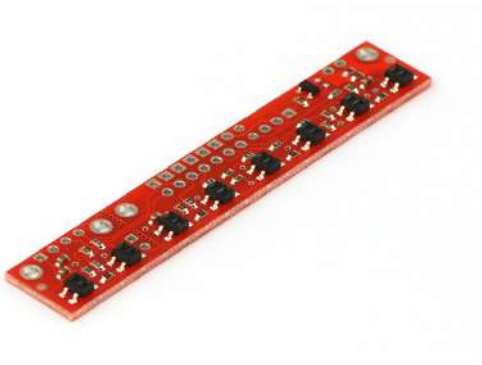

Figura 3.3: Sensor QTR-8RC

3.1.3 Ponte H

Para controlar motores de corrente contínua é necessária a utilização de uma ponte H para fazer a inversão do sentido para o qual giram. A ponte H utilizada é a L293D. Esta ponte H permite o controlo independente de dois motores, incluindo as suas tensões de alimentação. Para os controlar deve ser alimentado com 5V um de dois pinos que determinam o sentido de rotação do motor. Além do controlo de rotação existem também

os pinos de *enable* que podem ser utilizados como mecanismos de segurança, desligando os motores automaticamente ou como regulador da corrente fornecida aos mesmos.

A ponte H tem um fornecimento máximo de 600mA por motor e deve ser alimentada entre 4.5V a 3.6V. O seu *pinout* pode ser observado na figura 3.4.

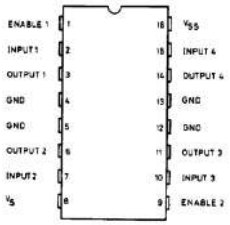

Figura 3.4: Ponte H L293D

3.1.4 Esquemático

Devido às dimensões do robô a sua estruturação teve de ser feita numa *breadboard* pequena. Contudo, o espaço na *board* revelou ser mais do que suficiente tendo em conta que o sensor se encontra montado por baixo do robô, ficando só na *board* a ponte H e o Arduíno Nano. A arquitectura geral descrita na introdução deste documento necessita de alimentação de 5V e -5V, sendo proposta a utilização de *powerbanks*. Apesar de, para alimentar todo o circuito do robô só ser necessário 5V foi utilizada também uma *powerbank* durante todo o trabalho, retirando assim qualquer dúvida de possíveis limitações a nível de corrente.

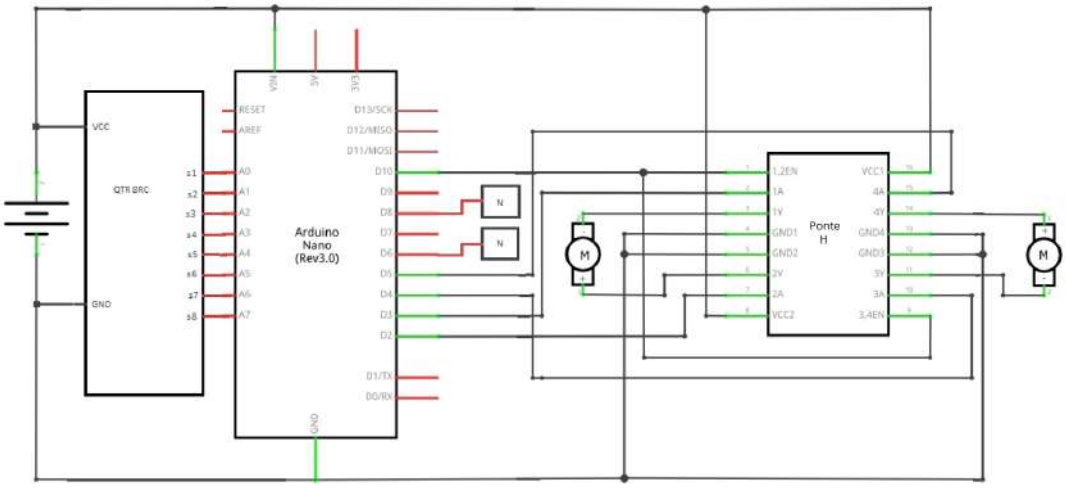

Figura 3.5: Esquemático do robô

Na figura 3.5 é possível observar o esquemático com todas as ligações feitas para o funcionamento do robô. Este encontra-se também no anexo A.

3.1.5 Ligação com Neurónios

A ligação com neurónios trata a criação de memória na resolução do percurso, tal como mencionado no início deste capítulo. Como esta dissertação só trata o desenvolvimento da máquina de estados com processo de aprendizagem e a criação dos neurónios em PCB a serem ligados ao Arduíno, a resolução inteligente do percurso não foi testada pois não se encontra dentro dos objectivos deste documento. Assim, foram programados alguns pinos para produzir os impulsos necessários para mais tarde afetarem os neurónios a serem adicionados. Os testes realizados resumem-se a verificar que os sinais gerados por estes pinos conseguem efectivamente alterar o estado dos neurónios. Este teste encontra-se no capítulo 5 deste documento.

Foi pensado ter dois neurónios de memória que ditam para que lado o robô deve virar sempre que chega a uma bifurcação. A quantidade de neurónios corresponde ao número de nós de aridade três existentes no percurso. A quantidade de nós existente no percurso é tratada na secção seguinte.

Inicialmente os neurónios encontram-se no mesmo estado, indicando ao robô para virar à direita nas bifurcações. Estes estão representados no esquemático, apresentado anteriormente, com os blocos N ligados aos pinos digitais 6 e 8.

Desconhecendo o resto da arquitectura foram adicionados também dois pinos de *enable* para estes neurónios. Estes são ativados durante a resolução de uma bifurcação com o intuito de só permitir a interferência dos neurónios de memória para com os movimentos do robô durante a execução das mesmas. Não estando esta fase do projecto desenvolvida, estes pinos não se encontram conectados no esquemático e correspondem aos pinos digitais 7 e 9.

O processo sobre a decisão de qual o melhor caminho a seguir e por sua vez, quais os neurónios a activar é descrito na secção do processo de aprendizagem. Esta solução, não tendo sido testada, fica em aberto para futuras alterações.

3.1.6 Ligação do Sensor ao Arduíno

O sensor QTR-8RC é composto por 8 sensores e estão ligados individualmente às 8 portas analógicas do Arduíno (A0 a A7). Sendo o QTR-8RC alimentado a 5V, cada um dos sensores fornece um intervalo de tensão entre 0V e 5V, consoante a detecção ou não da linha do percurso. Apesar da existência de uma biblioteca própria para usar em Arduíno de forma a ler os valores dos sensores, a sua utilização não foi necessária pois a utilização da função de leitura analógica *AnalogRead* do próprio Arduíno foi suficiente. Esta função devolve um intervalo de 0 a 1024 quando no porto analógico do Arduíno existe uma tensão no intervalo de 0V a 5V, correspondentemente. Após fazer o teste, colocando o robô sobre a linha e sobre o fundo branco, conclui-se que sobre a linha se obteve um valor fixo de 1024 e sobre o fundo branco um valor variável entre 35 e 45 dependendo da luz existente. Desta forma foi definido um *threshold* de 50.

3.1.7 Ligação do Arduíno com a Ponte H e aos Motores

Para fazer a ligação do Arduíno com os motores é necessário colocar, entre os dois, uma ponte H pois o Arduíno não tem capacidade de fornecer a corrente que os motores necessitam. Para além do problema energético, a ponte H permite também a inversão do movimento dos motores. A ponte H suporta a ligação de dois motores ligados que se encontram ligados nos pinos 3, 6, 11 e 14. Para cada motor existem dois pinos de controlo que, consoante a sua ativação, ditam a forma como são alimentados e por sua vez a sua rotação. Estes pinos são o 2, 7, 10 e 11 e estão ligados aos pinos 3, 2, 4 e 5 digitais do Arduíno, respetivamente. Os pinos 1 e 9 da ponte H correspondem aos pinos de *enable*, permitindo ou não o funcionamento do motores. Estes podem ser alimentados num intervalo de 0V a 5V, correspondendo à força mínima e máxima que a ponte H fornece e estão ligados ao pino digital 10 do Arduíno. No fim, apesar de a ponte H permitir alimentações separadas para cada motor, ambos os pinos encontram-se ligados à mesma fonte de alimentação, a *powerbank*.

3.2 Percurso

O percurso é encarado como um grafo em estrutura de arvore com raiz, convêm por isso enunciar alguns termos e definições da teoria de grafos [23] [18].

- Nó ou vértice: Um grafo é composto por pontos chamados nós, que podem ou não estar ligados entre si.
- Ramo ou aresta: Linhas utilizadas para criar as ligações entre nós.
- Aridade de um nó: A aridade de um nó corresponde à quantidade de ramos ligados a ele. Desta forma os nós podem ser designados como nós isolados ou pendentes quando as suas aridades são 0 ou 1 respectivamente.
- Circulação: Dá-se este nome a qualquer percurso que se possa fazer no grafo sem percorrer mais que uma vez o mesmo ramo e que termine no nó de partida.
- Árvore: Dá-se o nome de grafo em árvore quando não existem circulações no mesmo.
- Árvore com raiz: Um grafo construído em árvore com raiz é um grafo onde um nó é escolhido como ponto de partida. Todos os ramos são então implicitamente desenhados a partir desse nó afastando-se sempre da origem.

O percurso pode ser encontrado com e apenas 6 nós, dos quais 2 são de aridade 3 e os restantes de aridade 1. Estruturando o percurso num sistema de árvore com raiz e respeitando a regra quanto à aridade de cada nó, deparamo-nos com diversas possibilidades, figura 3.6.

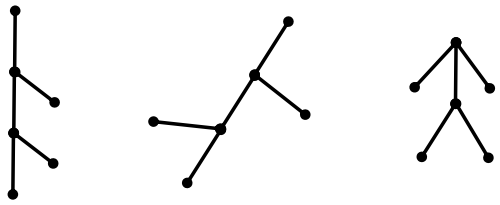

Figura 3.6: Três representações de grafo em árvore contendo 2 nós de aridade 3 e 4 nós de aridade 1

Ao estudar as diversas possibilidades de percursos inseridos nas condições anteriores, é possível notar que apesar de todas as formas abstractas diferentes que possam apresentar, todos eles são isomórficos. Todos os grafos criados contêm os 2 nós de aridade 3 no centro separados por um ramo e cada um deles possui agregado a si 2 dos restantes nós de aridade 1.

Com esta análise obtemos o percurso final a ser utilizado, o que nos permite explorar todas as possibilidades existentes a serem resolvidas pelo robô. Para tal bastou definir no mesmo percurso dois possíveis pontos de "fim"representados por uma linha transversal num dos nós de aridade 1 e três possíveis pontos de início representados pelos círculos A, B e C, figura 3.7.

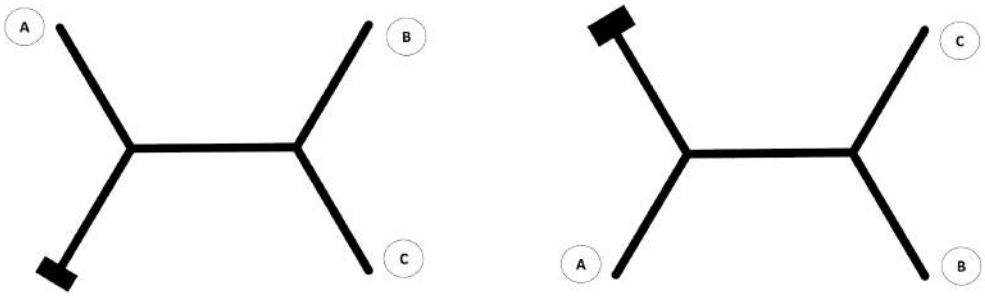

Figura 3.7: O percurso final com dois pontos de finalização adjacentes

3.2.1 Especificações

Não existe nenhum tamanho mínimo para o percurso, porém, para que o robô consiga percorrer o percurso correctamente, o mesmo deve obedecer a algumas especificações:

- O ângulo entre ramificações nos nós deve ser sempre de 120º para que desta forma a abordagem do robô às bifurcações seja idêntica, independentemente da origem da

sua aproximação.

- O comprimento de cada ramificação (incluindo a central entre os nós) deve ser pelo menos 1.5 vezes o comprimento do carro.
- O percurso deve ser pintado a preto mate sobre fundo branco para maximizar os resultados dos sensores pois a reflectância da linha negra influencia o seu desempenho. Foi utilizada uma fita adesiva preta em cartolina branca.
- A espessura da linha deve estar contida entre 1.7cm e 1.9cm possibilitando ter dois sensores sobre a linha de uma só vez, quando o robõ se encontra a andar em linha recta.

3.2.2 Resolução do Percurso e os seus Obstáculos

Para que seja possível o robô aprender qual o melhor percurso a fazer, é necessário que seja capaz de seguir a linha e por sua vez contornar os obstáculos do percurso. Estes obstáculos consistem nas bifurcações em cada nó, o final de cada ramificação onde o caminho se torna inexistente e por último, a linha transversal de "fim".

A resolução de cada um destes obstáculos, é idêntica para todas as vezes que o robô fizer o percurso, independentemente de ser ou não a primeira vez que o está a realizar.

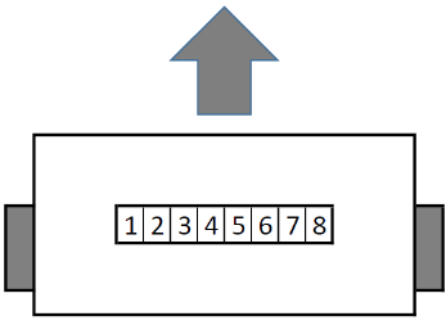

Figura 3.8: Numeração dos sensores no robô

Para facilitar a descrição dos processos, os sensores passam a ser identificados com números. Estes números indicam a ordem pelo qual os 8 sensores se apresentam da esquerda para a direita, observados no sentido em que o carro se desloca como se pode observar na figura 3.8. Um sensor é considerado activo quando detecta a linha e inactivo em qualquer outro caso.

3.2.2.1 Seguir a Linha

Como os dois motores não têm sempre exactamente o mesmo comportamento, mesmo quando o carro se encontra 100% alinhado com a linha, é necessário fazer pequenas correções. Desta forma, o processo de seguir a linha é dividido em três subprocessos:

- Andar para a frente:

  Para andar em frente ambas as rodas são activadas simultaneamente. Este processo acontece quando os sensores 4 e 5, ou pelo menos um deles, estão acionado sem que nenhum dos adjacentes esteja.

- Virar à esquerda:

  Este caso acontece quando o carro se encontra um pouco à direita da linha e é acionado quando os sensores 3 e 4 se encontram activados. Para tal, apenas a roda direita é activada.

- Virar à direita:

  É executado da mesma maneira que o movimento de virar à esquerda mas desta vez activando a roda esquerda. O movimento é activado quando os sensores 5 e 6 estão activados.

Estas três acções têm ordem de prioridade, sendo esta, a mesma pelas quais foram referidas. A prioridade foi criada para que, mesmo que por alguma razão externa como uma sombra, os sensores 4, 5 e 6 (um exemplo) estejam activos simultaneamente, o robô se mova para a frente.

3.2.2.2 Ausência de Linha ou 180º

Esta situação surge quando o robô alcança o fim de uma das ramificações que não tem a linha transversal de fim na sua extremidade. Este obstáculo é detectado quando todos os sensores se encontram inactivos. Assim, o robô deve girar sobre si mesmo até voltar a detectar a linha. A rotação acontece acionando o motor esquerdo para a frente e o motor direito para trás fazendo com que o robô gire para a direita. A rotação termina quando os sensores 4 e 5 se encontram novamente activos retomando assim o processo de seguir a linha. A movimentação pode ser observada na figura 3.9.

3.2.2.3 Bifurcação

A situação de bifurcação acontece sempre que o robô se aproxima de um dos nós. Tal como já mencionado, a separação entre cada ramificação nos nós é de 120º pelo que o processo é o mesmo independentemente da ramificação da qual o robô se aproxima do nó. A bifurcação é detectada quando os sensores 3, 4, 5, e 6 se encontram activos e a sua resolução acontece em dois processos:

- Ambas as rodas são acionadas para que o robô ande em frente. O movimento perdura até que os sensores 4 e 5 fiquem desactivados.

- Assim que os sensores 4 e 5 ficam desactivos só o motor do lado esquerdo permanece activo. Desta forma o robô vira à direita. O movimento continua até que os sensores 4 e 5 voltem a ficar activos voltando novamente ao processo de seguir a linha.

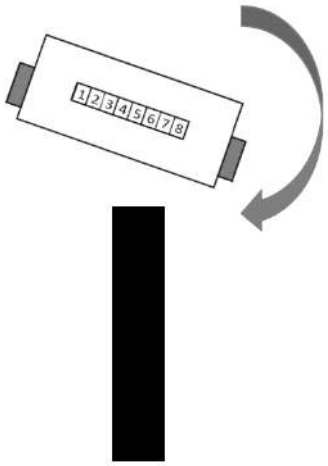

Figura 3.9: Descrição do movimento em ausência de linha

Para resolver a bifurcação o robô desloca-se para a frente antes de fazer curva. Isto acontece pois facilita a centralização do robô sobre a linha após o bifurcação. A movimentação pode ser observada na figura 3.10.

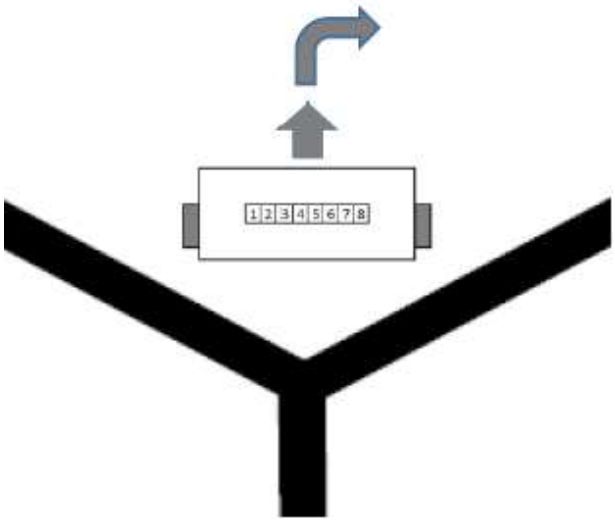

Figura 3.10: Descrição do movimento em bifurcação

3.2.2.4 Fim de Percurso ou *Stand by*

Por último, a situação de fim de percurso é determinada ao detectar a linha transversal no fim de uma ramificação. A linha transversal é detectada quando todos os sensores se encontram activos ou todos menos um dos das extremidades. Quando detectado, o robô pára, ficando em *stand by* até voltar a ser reposto num dos pontos iniciais.

A decisão da linha final ser detectada mesmo estando um dos sensores das pontas desactivado deve-se ao facto de nem sempre o robô ficar 100% sobre a linha transversal.

Assim é evitado que seja detectada erradamente a situação de bifurcação

Com estes quatro processos é possível resolver todo o percurso, independentemente do ponto onde o robô inicia a sua deslocação. O robô resolve o percurso virando em todas as bifurcações à direita até que encontre o fim. É de salientar que apenas a roda direita gira para trás. Esta decisão foi considerada para reduzir os bits de controlo enviados para o neurónios, facilitando o trabalho a desenvolver na interface de motores.

3.3 Máquina de Estados e Implementação

Nas secções anteriores foi abordado o *hardware* utilizado, o percurso a ser percorrido e como cada obstáculo do mesmo é solucionado, esta secção trata a implementação do código em Arduíno, ou seja, como é feita a leitura dos sensores, o método de activação dos motores e como se processa a alteração dos neurónios que promovem memória ao robô.

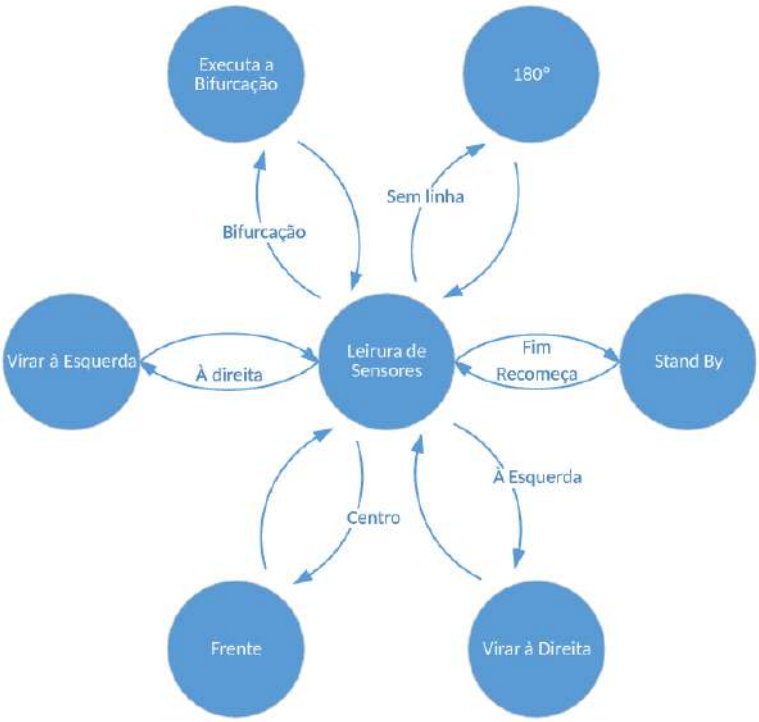

Figura 3.11: Visão geral da máquina de estados implementada

Na figura 3.11 é possível observar a máquina de estados implementada em Arduíno. O funcionamento da máquina de estados acontece em torno do estado da leitura de sensores sendo este que dita qual o próximo estado a executar. Na secção do percurso, a activação de cada um dos seis estados depende da quantidade de sensores que se encontram activos. Essa informação está disponível na tabela 3.1.

É possível notar que várias combinações de sensores activam o mesmo estado. Isto

acontece devido às dificuldades de movimentação do robô, o que obrigou a criar condições mais complexas em alguns casos. As dificuldades de movimento associadas serão descritas à posteriori.

A figura 3.11 representa uma visão de alto nível da implementação, pois alguns do estados apresentados são compostos por outros processos mais pequenos ou repetição dos mesmos. Esses estados serão tratados individualmente e explicados nas secções seguintes, deixando assim o diagrama apresentado simples e com uma visão geral do seu funcionamento.

Antes de descrever a implementação dos estados é importante tratar primeiro como é feita a leitura dos sensores e a activação dos motores para o movimento.

3.3.1 Leitura dos Sensores

Na secção do esquemático, os sensores são ligados aos oito portos analógicos do Arduíno e, para fazer a sua leitura, não é necessária a utilização de qualquer biblioteca ou ADC (conversor analógico-digital). A leitura é feita em cada sensor individualmente e a informação é guardada num vector de inteiros com nove posições. Após a leitura, cada posição do vector contém um valor no intervalo de 0 a 1024. Para facilitar outros processos, estes valores são convertidos em 0 e 1 utilizando um *threshold*. Assim, cada posição passa a conter 0 caso o valor armazenado seja igual ou inferior a 50 e 1 caso contrário.

Apesar de só existirem 8 sensores, o vector onde os valores são armazenados é de nove posições. A posição extra existe para armazenar uma cópia do primeiro sensor, facilitando caso haja necessidade de calibrar o *threshold* devido a condições de luminosidade.

O código correspondente à leitura de sensores pode ser observado na figura 3.12.

Tabela 3.1: Acções do robô para as diversas detecções dos sensores. 1-Sensor Activo; 0-Sensor desactivo; X-*don't care*.

S1	S2	S3	S4	S5	S6	S7	S8	Acção
X	X	X	1	1	X	X	X	
X	X	0	0	1	0	X	X	Frente
X	X	0	1	0	0	X	X	
X	X	1	1	X	X	X	X	Esquerda
X	X	X	X	1	1	X	X	Direita
X	X	1	1	1	1	X	X	Bifurcação
0	0	0	0	0	0	0	0	180º
0	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	Stand By
1	1	1	1	1	1	1	0	

```
void readAnalogSensors(){
  sensorValues[0]=analogRead(A0);
  sensorValues[1]=analogRead(A1);
  sensorValues[2]=analogRead(A2);
  sensorValues[3]=analogRead(A3);
  sensorValues[4]=analogRead(A4);
  sensorValues[5]=analogRead(A5);
  sensorValues[6]=analogRead(A6);
  sensorValues[7]=analogRead(A7);
  sensorValues[8]=sensorValues[0];

  for(int i=0; i<8; i++){
    if(sensorValues[i]>50) //threshold de 50
      sensorValues[i]=1;
    else
      sensorValues[i]=0;
  }
```

Figura 3.12: Código da leitura de sensores e armazenamento dos seus valores

3.3.2 Activação dos Motores

Para promover movimentação ao robô é necessário trabalhar com a ponte H inserida no circuito. O ideal seria promover ao robô uma velocidade constante e suficientemente lenta para que seja capaz de executar todos os movimentos sem se despistar. Existem duas formas de controlar a velocidade do robô:

- Alimentação: A energia passada ao robô vai depender da fonte de alimentação que alimenta a ponte H. Neste ponto não se pôde mexer pois é utilizada uma *powerbank* de 5V e 1A pelo que a velocidade do robô tem de ser controlada pela segunda hipótese.

- Pinos de *Enable*: A ponte H possui dois pinos de *enable* para activar e desactivar os motores, que não funcionam só com *on* e *off*, podendo ser alimentados num intervalo de 0V a 5V libertando gradualmente corrente para os motores, consoante maior for a tensão que lhe é fornecida. Estes pinos foram então ligados ao pino digital 10 do Arduíno que permite a utilização de PWM (*Pulse-Width Modulation*) utilizando a função analogWrite() com argumento entre 0 e 255, sendo assim possível alimentar o motor em intervalos de tempo controlados e por sua vez controlar a sua velocidade. Esta abordagem não foi suficiente pois os motores contêm uma caixa de velocidades integrada pelo que valores baixos não levavam o robô a conseguir mover-se e ao aumentar um pouco o PWM promovia uma velocidade demasiado alta.

O problema de utilizar PWM deve-se ao facto de os motores com caixa de velocidade não ganharem a sua velocidade máxima no momento em que são alimentados mas sim

de uma forma gradual. Assim sendo, quando foi utilizado um valor suficiente para que o robô se deslocasse, a sua velocidade aumentava para um valor acima do desejável.

A solução encontrada passou por fornecer um valor de PWM que promovia movimento ao robô e ao mesmo tempo ativar e desativar os pinos de direcção intermitentemente de forma cíclica. Infelizmente não se consegue um movimento contínuo como era desejado mas é possível variar a velocidade a que o robô se desloca, variando o PWM e o intervalo entre ligar e desligar o motor. Um exemplo desta implementação pode ser observado na figura 3.13.

```
analogWrite(10, 200); //200 e delay 50

digitalWrite(DF, HIGH); digitalWrite(EF, HIGH);
delay(50);
digitalWrite(DF, LOW); digitalWrite(EF, LOW);
delay(250);
```

Figura 3.13: Código da activação dos motores

Como se pode observar, entre cada ciclo de movimento é feito um *delay* de 250 milissegundos. Este compasso de espera deve-se ao escorregamento dos motores, sendo necessário tempo para que o motor pare completamente.

3.3.3 Implementação dos Estados

Será explicado a implementação de cada estado correspondendo à solução em código, de cada um dos obstáculos. Como a solução de cada obstáculo, foi descrita na secção do percurso, esta serve somente para demonstrar como foi feita em código. O código na sua totalidade encontra-se no anexo B.

3.3.3.1 Seguir a Linha

O processo de seguir a linha é composto pelos estados de andar em frente, virar à esquerda e virar à direita. É dada prioridade ao movimento de seguir em frente pelo que no código os três movimentos foram encadeados em *if* e *else*, sendo a movimentação em frente a primeira condição. O código pode ser observado na figura 3.14.

É de notar que existe a utilização da função escreve() com os argumentos "Frente", "Esquerda"e "Direita". Esta é uma função adicional criada para imprimir no ecrã o estado de cada sensor juntamente com a frase passada em argumento. Esta é passada por USB quando o Arduíno se encontra ligado ao computador, tornando assim mais fácil o *debug* do código. Esta função aparece várias vezes ao longo do código implementado.

3.3.3.2 Fim de Percurso ou *Stand By*

O objectivo do fim de percurso é que o robô parasse e só retomasse a marcha após receber informação para tal. Como ainda não havia certeza de como se iria proceder para reativar

```
if((sensorValues[3]&&sensorValues[4])
    || (!sensorValues[2]&&!sensorValues[3]&&sensorValues[4]&&!sensorValues[5])
    || (!sensorValues[2]&&sensorValues[3]&&!sensorValues[4]&&!sensorValues[5])){
  escreve("frente");
  digitalWrite(DF, HIGH); digitalWrite(EF, HIGH);
  delay(50);
  digitalWrite(DF, LOW); digitalWrite(EF, LOW);
}
else  if(sensorValues[2]&&sensorValues[3]){
        escreve("esquerda");
        digitalWrite(DF, HIGH);
        delay(50);
        digitalWrite(DF, LOW);
      }
       else if(sensorValues[4]&&sensorValues[5]){
              escreve("direita");
              digitalWrite(EF, HIGH);
              delay(50);
              digitalWrite(EF, LOW);
             }
delay(250);
```

Figura 3.14: Código correspondente aos movimentos simples para seguir a linha

o robô, este foi implementado com uma pausa de um minuto, o que promoveu tempo suficiente para recolocar o robô num ponto inicial novamente.

3.3.3.3 Ausência de Linha ou 180º

A ausência de caminho despoleta quando todos os sensores se encontram inativos. Neste caso foi criada uma função onde o Arduíno fica ciclicamente até voltar a encontrar a linha. Só depois de detectar a linha e se centrar é que o Arduíno sai da função e volta ao processo normal. De referir que este processo, apesar de um ciclo fechado, está sempre a ler os valores dos sensores. O código pode ser observado na figura 3.15.

```
void caminhoErrado(){
  do{
    digitalWrite(DT, HIGH); digitalWrite(EF, HIGH);
    delay(50);
    digitalWrite(DT, LOW); digitalWrite(EF, LOW);
    escreve("Caminho errado");
    delay(250);
    readAnalogSensors();
  } while(!(sensorValues[3]&&sensorValues[4]));
}
```

Figura 3.15: Código correspondente à ausência de linha ou 180º

3.3.3.4 Bifurcação

A bifurcação é detectada quando os sensores 3, 4, 5 e 6 se encontram activos. Isto torna-se possível devido à separação do percurso que faz com que por momentos os 4 sensores centrais se encontrem sobre a linha. O processo de resolução acontece em dois passos. Mais uma vez tudo se passa em processos cíclicos isolados sempre acedendo aos sensores.

Esta implementação não é a final pois o processo de aprendizagem acabou por ser inserido na execução da bifurcação pelo que se tornou mais complexa e a parte mais importante da resolução do percurso. O processo de aprendizagem pode ser visto na secção seguinte. Na figura 3.16 é possível observar a primeira versão do código da bifurcação.

```
do{
  escreve(Bi1);
  digitalWrite(DF, HIGH); digitalWrite(EF, HIGH);
  delay(50);
  digitalWrite(DF, LOW); digitalWrite(EF, LOW);
  delay(250);
  readAnalogSensors();
}while(!(!sensorValues[3]&&!sensorValues[4]));

do{
  escreve(Bi2);
  digitalWrite(EF, HIGH);
  delay(50);
  digitalWrite(EF, LOW);
  delay(250);
  readAnalogSensors();
}while(!(!sensorValues[2]&&sensorValues[3]&&sensorValues[4]&&!sensorValues[5]));
```

Figura 3.16: Primeira versão do código para resolver a bifurcação

3.3.4 Processo de Implementação de Memória

Mais do que percorrer o percurso e ser capaz de chegar ao fim, o robô tem também de ser capaz de apreender a melhor rota para que numa segunda resolução chegue ao fim pelo caminho mais curto. Para tal foi necessário primeiro analisar todos os possíveis trajetos tomados pelo robô, na primeira resolução, figura 3.17. É necessário relembrar que, por defeito, o robô resolve todas as bifurcações para a direita pois os neurónios de memória assim se encontraram inicializados.

Observando os trajetos é fácil perceber que sempre que o robô executa duas bifurcações no mesmo nó o robô deveria ter virado à esquerda da primeira vez que se deparou com a bifurcação. Mas, apenas está-se dependente da informação que o robô consegue obter do trajecto (nomeadamente através do sensor) para provocar uma alteração nos neurónios de memória correspondentes a cada nó. Neste caso, entenda-se nó como os nós interiores de aridade 3.

A solução passou então por conseguir contar quantas bifurcações o robô já executou e em que parte/nó do percurso se encontra. Para tal, a implementação é composta por:

- Sequênciador de bifurcações: A variável da qual vai depender a decisão em que nó o robô se encontra. O seu valor é colocado a 1 sempre que é executado uma bifurcação e a 0 quando é detectada a ausência de linha. Se no início de uma bifurcação esta variável se encontrar a 1 então significa que o robô se encontra no outro nó.

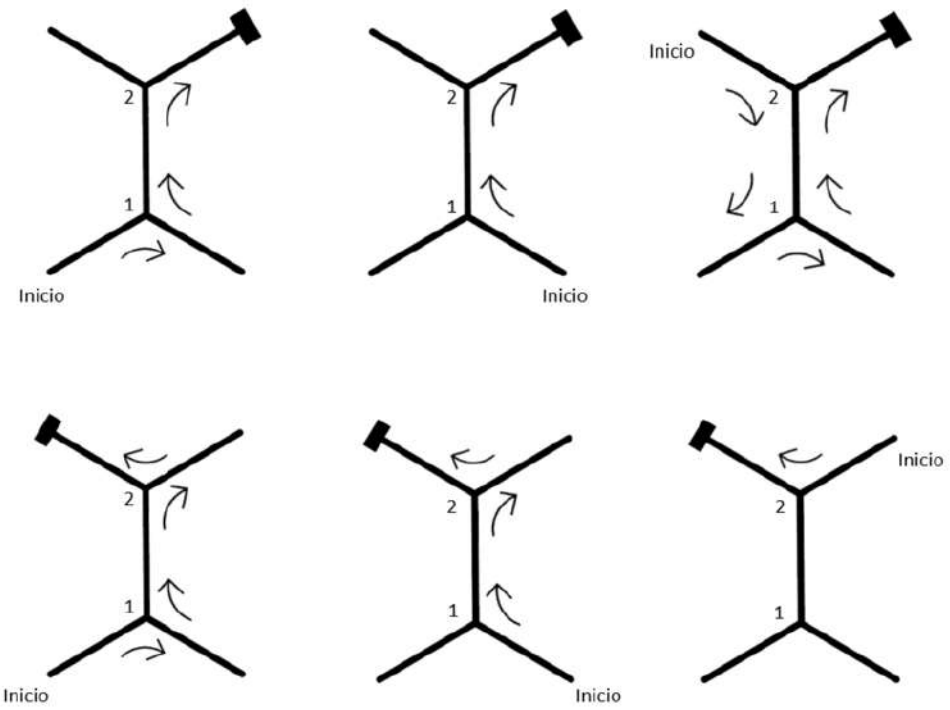

Figura 3.17: Todas as possibilidades da primeira resolução do trajecto

- Idexação do nó: Esta variável indica em que nó o robô se encontra, servindo de índice para um vector onde o contador de bifurcações se encontra implementado.
- Contador de bifurcações: Este contador é implementado como um vector de duas posições utilizando a variável anterior como índice. Sempre que uma bifurcação é executada, a posição correspondente é incrementada.

Depois de formado este sistema, sempre que o robô mudava de nó bastou verificar quantas bifurcações realizou no nó que abandonou. Se tivesse executado pelo menos duas bifurcações, um impulso era enviado para o neurónio de memória correspondente, fazendo com que na próxima execução o robô virasse à esquerda na primeira bifurcação desse nó. Na figura 3.18 é possível observar o código implementado.

Esta implementação permite também que o robô seja colocado num ponto de partida diferente do primeiro, aquando na segunda execução e detecte automaticamente que não está a fazer a melhor resolução possível.

```
//Bifurcação
if(sensorValues[2]&&sensorValues[3]&&sensorValues[4]&&sensorValues[5]){
  if(biSeq){ //Se estiver a 1 é porque não apanhou nem turn around ante
    if(noIndex==0){
      noIndex=1;
    }
    else{
      noIndex=0;
    }
  }
  if(noIndex==0)
      digitalWrite(EV1, HIGH);
  else
      digitalWrite(EV2, HIGH);
  bifurcação();

  digitalWrite(EV1, LOW);
  digitalWrite(EV2, LOW);

  nos[noIndex].biCount++;
  if(nos[noIndex].biCount==2){ //altera o virabicos sempre que encontra
    if(noIndex==0){
      digitalWrite(V1, HIGH);
      escreve("ViraBicos-1");
      delay(100);
      digitalWrite(V1, LOW);
    }
    else{
      digitalWrite(V2, HIGH);
      escreve("ViraBicos-2");
      delay(100);
      digitalWrite(V2, LOW);
    }
  }
  biSeq=1;
}
```

Figura 3.18: Código de bifurcação com alterações necessárias para resolução de percurso com memória

Capítulo 4

Desenho e Produção de Neurónio em PCB

Este capítulo retrata todo o processo de projeto e montagem de um neurónio em PCB. O objectivo de implementar neurónios em PCB passou por reduzir o tamanho da montagem original em *breadboard*, para serem aplicados sobre o robô, e ao mesmo tempo criar um fácil acesso às entradas e saídas de sinal.

O circuito do neurónio, encontrava-se desenvolvido pelo que não será analisado ao detalhe. Irá ser abordado de uma forma geral e, em algumas situações, dada alguma atenção a alguns componentes que tiveram uma maior importância, no desenvolvimento da placa.

Neste capítulo, o neurónio é referido numa perspectiva de funcionamento, passando a descrever todos os processos de desenho da placa incluindo o seu *layout* e por fim, todo o processo da montagem de componentes. Para desenhar a PCB foi utilizado o KiCad EDA, um software dedicado ao desenho de PCB.

4.1 Neurónio

Não irá ser feita qualquer análise específica ao circuito do neurónio, focando-se esta secção numa visão de alto nível dos seus blocos constituintes seguida de um estudo de *layout* e da colocação de componentes.

O funcionamento do neurónio consiste na produção de comboios de *spikes* em duas frequências diferentes (alta e baixa) consoante os sinais ao qual é sujeito. Estas diferentes frequências definem o estado do neurónio da mesma forma como na álgebra booleana é utilizado o 0 e o 1. Para alimentar o circuito, é necessário tensão positiva e negativa. A alimentação é feita com *powerbanks* interligadas pelo que se dispõe de 5V e -5V.

O neurónio é composto por quatro blocos:

- Agregador de sinais: Este bloco é o ponto de entrada do neurónio e serve para juntar

todos os sinais que lhe sejam injetados. É composto por 3 entradas positivas e 3 negativas gerando um valor de tensão entre 5V e -5V à saída.

- Módulo de decisão: Este bloco define, se a frequência de *spikes* a ser gerada é alta ou baixa dependendo do valor que vem do bloco anterior.
- Oscilador: Este é o bloco responsável por criar as frequências alta ou baixa definidas pelo módulo de decisão.
- Gerador de *spikes*: Gera à frequência definida pelo oscilador criando comboios de *spikes* à saída.

Uma representação desta estrutura pode ser observada na figura 4.1.

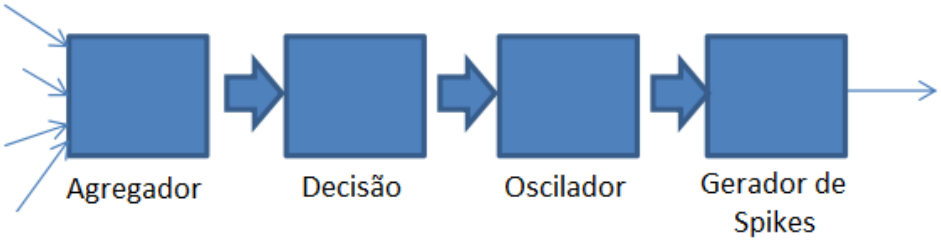

Figura 4.1: Representação por blocos da constituição de um neurónio

4.1.1 *Layout*

O objectivo foi posicionar os neurónios em cima do robô, permitindo fácil acesso aos mesmos tal como facilidade em interliga-los. Nestas condições e com o pouco espaço que o robô disponibiliza, foi decidido colocar os neurónios na vertical. Existe assim, uma interface de alimentação que é ligada à superfície do robô ficando na outra extremidade da placa a interface de entrada e saída de sinais. A figura 4.2 representa um esboço deste conceito.

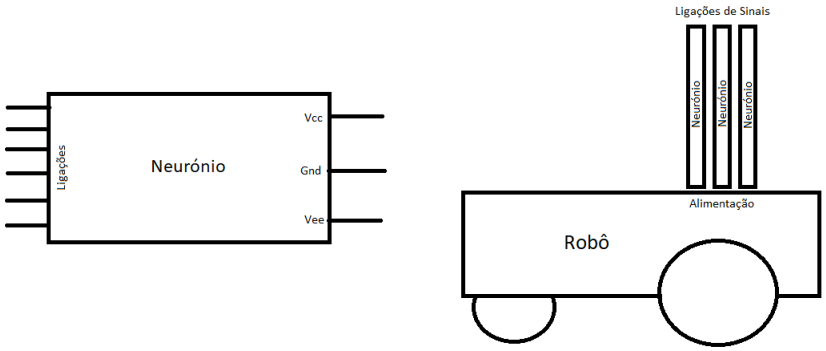

Figura 4.2: Esboço representativo da placa para produção e montagem no robô

Na superfície do robô estava uma PCB responsável por transportar energia a todos os componentes (Arduíno, ponte H, neurónios, etc), sendo composta por vários *sockets*.

4.1.1.1 Terminais de Alimentação e Sinais

No neurónio encontramos 6 entradas de sinal e uma saída. Nas primeiras montagens em *breadboard* estas ligações encontravam-se soltas quando não estavam a ser utilizadas. Porém esta abordagem não foi a melhor pois durante a execução de testes facilmente outros fios entravam em contacto com estas pontas soltas originando mau funcionamento ao circuito. Para prevenir esta situação, na colocação dos neurónios sobre o carro robô, foram adicionados pinos ligados à terra intercalados com cada terminal de entrada. Com a utilização de *jumpers* é possível assim isolar o terminal. Para a saída dos *spikes* foi adicionado um segundo terminal coexistindo lado a lado com o primeiro prevenindo assim a necessidade de transportar o sinal para mais do que um outro neurónio ou circuito. Na totalidade podemos assim contar com 14 pinos na aresta correspondente à interface de sinais de cada neurónio.

Ao nível da alimentação o neurónio contou com 5V, -5V e terra, correspondendo aos terminais Vcc, Vee e Gnd. Sendo o neurónio encaixado numa PCB na parte superior do carro com pistas que servem de *bus* de energia, surgiram duas preocupações. Em primeiro lugar, a possibilidade de uma das pistas ser afectada por algum problema corrompendo a alimentação aos neurónios. Em segundo, o facto de facilmente se poder montar o neurónio com os terminais trocados invertendo a alimentação.

De maneira a solucionar o primeiro problema, foram acrescentados dois pinos por alimentação passando a existir 3 de Vcc, Vee e Gnd. Os grupos de 3 pinos não são independentes encontrando-se interligados na placa. Desta forma mesmo que uma das pistas na PCB de alimentação seja corrompida, as outras continuam a fornecer energia aos neurónios.

Solucionado o problema da alimentação, o segundo desafio manteve-se pois a existência de pinos adicionais (nove no total) não possibilitou uma fácil detecção de quais os pinos correspondentes a cada valor de tensão. A solução passou por criar na placa furação para outros dois pinos que não são adicionados. Desta forma, a interface de energia do neurónio contém 6 pinos seguidos correspondentes ao Vcc e Gnd, seguido de um espaço antes dos três pinos de Vee. A criação deste espaço tornou a interface de alimentação assimétrica (figura 4.3) não permitindo que o neurónio fosse encaixado de outra forma que não a correta no *bus* de alimentação.

4.2 Desenvolvimento da PCB

O desenvolvimento da PCB no KiCad EDA passou por três fases: Desenho do esquemático, associação de *footprints* e desenho da PCB. Estes processos tiveram de ser sequenciais, dado que, a forma como um era feito, influenciava o seguinte.

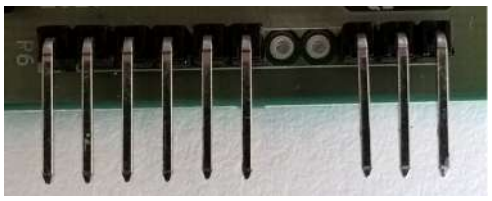

Figura 4.3: Pinos de alimentação da PCB final com dois furos livres por forma a criar assimetria

O processo de criação da placa foi moroso, tendo sido feitas várias versões experimentais antes de se chegar à versão final. Não serão expostas todas as versões conseguidas neste documento, visto que, algumas não passam de melhorias estéticas da versão anterior. Serão apenas salientadas as alterações significativas para a criação da versão final da PCB.

Ao longo deste capítulo é possível observar alguns pormenores de cada fase de desenvolvimento. Pode ser consultado o esquemático e o desenho em PCB na sua totalidade nos anexos.

4.2.1 Esquemático

A primeira fase consistiu em desenhar o esquemático do circuito a fazer na PCB. O esquemático não consiste apenas nos componentes inerentes ao circuito mas sim em tudo o que queiramos inserir na placa desde os componentes até aos conectores necessários. Não vão ser referidos, quais os componentes aplicados, porque o circuito em si só não é o foco.

No sistema de desenho de circuitos, tal como em muitos outros, é necessário colocar todos os componentes e de seguida fazer ligações entre eles utilizando a ferramenta respectiva aos fios. Todavia, não existe qualquer funcionalidade de simulação do circuito pelo que não há nenhuma necessidade prática de selecionar o valor correto dos componentes. No entanto, colocar os valores exactos serve para referência no circuito para posteriormente visualizar durante a montagem.

Não possuindo o programa funcionalidades de simulação, é necessário especificar que a alimentação é proveniente de uma fonte externa sendo necessário utilizar *power flags*. Na figura 4.4 é possível observar essas *flags* ligadas aos símbolos de alimentação. Feito isto, já se pode colocar os símbolos de alimentação no circuito sem que surjam mensagens de erro.

4.2.1.1 Etiquetas

Para a simplificação o desenho, em vez de fios podem ser colocadas etiquetas. Ao colocar em dois componentes etiquetas com o mesmo nome o programa considera que esses dois componentes se encontram ligados.

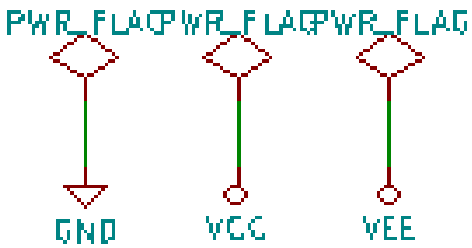

Figura 4.4: Símbolos de alimentação ligados às *powerflags* respetivas

Na figura 4.5 é possível observar uma dessas utilizações. Neste caso, o uso de fios seria extremamente complexa de se ler, pois os fios ficariam cruzados uns nos outros. O emprego de etiquetas torna todo o desenho mais limpo.

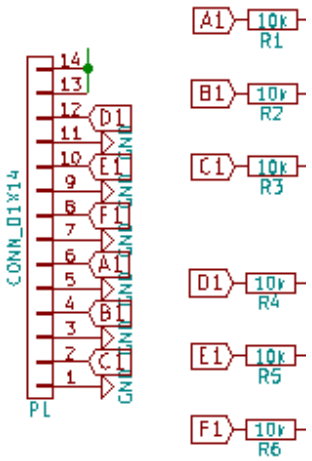

Figura 4.5: Etiquetas de ligação entre o conector de sinais e as resistências de entrada

4.2.2 Criação do Diagrama de Contactos

O diagrama de contactos de uma placa é composto por sub diagramas (*footprints*) correspondentes a cada componente do circuito.

Construído todo o esquemático foi necessário associar a cada componente uma *footprint* para colocar na placa. O KiCad, para cada componente colocado no circuito apresenta uma variada lista de *footprints* disponíveis. Neste ponto nota-se que, colocar ou não, o valor correcto do componente no esquemático não promove um efeito prático no desenho da PCB. Na figura 4.6 é possível observar um destes casos. Ao selecionar o transistor existente no circuito, aparecem várias possibilidades de *footprint*, porém, não existe qualquer associação ao modelo do transistor escolhido e sim apenas à quantidade de terminais que o mesmo contém.

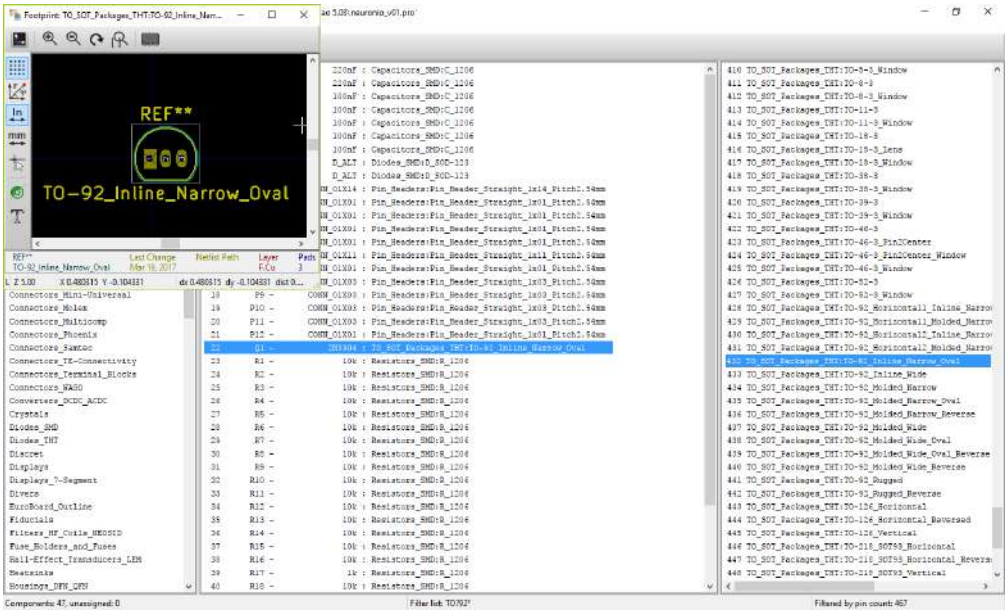

Figura 4.6: Ecrã de associação de *footprints* aos componentes do circuito onde se encontra selecionada a *footprint* para o transistor existente no mesmo

Neste caso foi escolhido uma *footprint* com as furações dos terminais o mais próximas possível entre elas, de modo a ocupar pouco espaço. Para todos os restantes componentes foram escolhidas *footprints* de SMD (*Surface Mount Tecnology*) com dimensão 1206 (ou 3216 no sistema métrico). Este tamanho de componentes é o ideal para o caso pois permite criar um neurónio de dimensão reduzida ao mesmo tempo que torna possível soldar e dessoldar componentes manualmente.

4.2.3 Desenho da PCB

O primeiro passo na disposição das *footprints* na PCB foi saber como distribuir a alimentação pela placa. Sendo esta uma PCB de duas faces decide-se que cada uma das faces seria condutora de um valor de alimentação, ficando o terceiro valor dependente de pistas para ser transportado. Dos três valores, o que requer menos ligações é o Vee ficando assim estipulado que a camada superior seria Vcc e a inferior Gnd. Desta forma, todos os componentes com terminais ligados ao Gnd encontram-se ligados à camada de baixo através de vias. O mesmo se passa com o Vee pois é na camada inferior que se encontra uma pista para alimentar todos os componentes necessários. O desenho destas camadas pode ser observado no anexo C.

Na primeira versão desenvolvida do neurónio a ideia era manter todos os componentes com a mesma orientação e alinhados entre si, obtendo uma montagem simétrica e fácil de ler. A colocação dos componentes foi dividida em quatro partes, sendo cada uma delas o conjunto de quatro terminais (face) dos circuitos integrados. Cada um destas faces corresponde a um bloco do neurónio. O resultado desta montagem pode ser observado

na figura 4.7. A numeração e a seta indicam o fluxo de funcionamento do circuito desde a entrada de sinais até à saída. A maioria dos componentes foi retirada da imagem, pois tornava a mesma demasiado confusa, não sendo necessários para o exemplo pretendido.

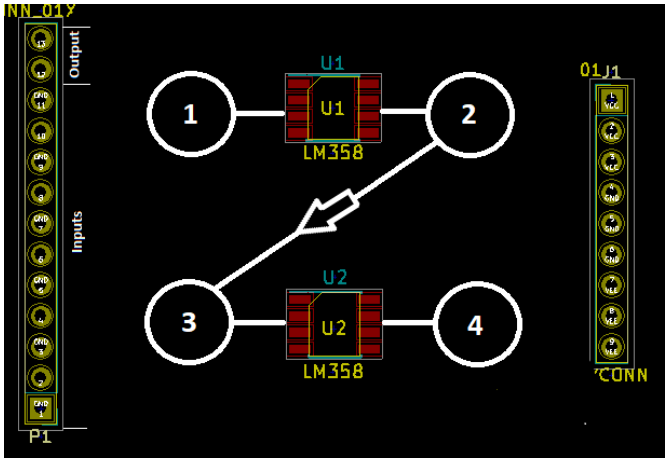

Figura 4.7: Desenho da primeira versão da PCB editado por forma a exibir a sequência de funcionamento do circuito.

Esta disposição, apesar de simétrica e visivelmente agradável, não é viável. Estando os terminais de entrada e saída de sinal localizados na mesma aresta da placa obrigava a que existisse uma pista a atravessar todo o circuito do ponto 4 para voltar ao ponto 1 por forma a trazer os *spikes* para o terminal devido. Com esta situação optei por refazer o desenho surgindo assim a PCB final.

No segundo desenho da PCB decide-se fazer a disposição numa forma circular, esquecendo a simetria da montagem, em vez da forma em Z anterior. Esta disposição resolveu o problema antes existente onde o sinal de saída de *spikes* ficava completamente afastado do seu respectivo terminal. Deste modo, o fluxo do circuito começa e acaba do mesmo lado da placa, figura 4.8.

Contudo, esta disposição trouxe um novo problema a nível da conetividade do Vcc aos componentes necessários sobre a camada superior. Ao dispor os componentes desta nova forma é extremamente fácil criar ilhas (zonas onde a alimentação não chega) ao centro do circuito. Surgindo assim um problema, pois os pinos de Vcc das dois circuitos integrados encontram-se na zona central da placa. Para resolver este problema e abrir passagem ao Vcc para essa zona, umas das ligações de uma resistência ao transistor foi feita com vias, passando pela camada de baixo da placa. Na figura 4.9 pode observar-se a camada superior onde se encontra feita esta ligação criando assim espaço livre. A imagem foi editada, adicionando uma linha tracejada correspondente à ligação na camada inferior.

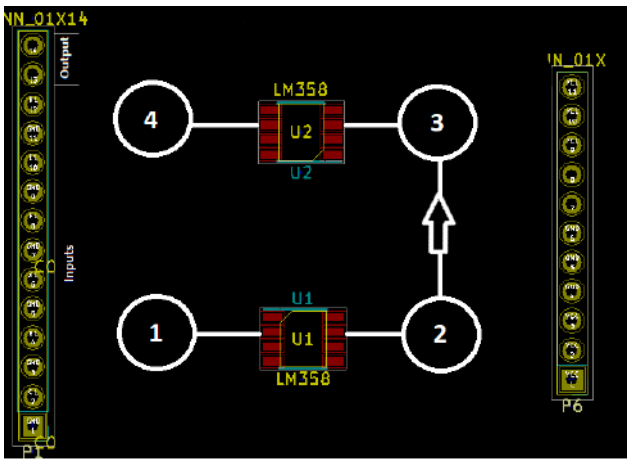

Figura 4.8: Desenho da segunda versão da PCB editado por forma a exibir a sequência de funcionamento do circuito.

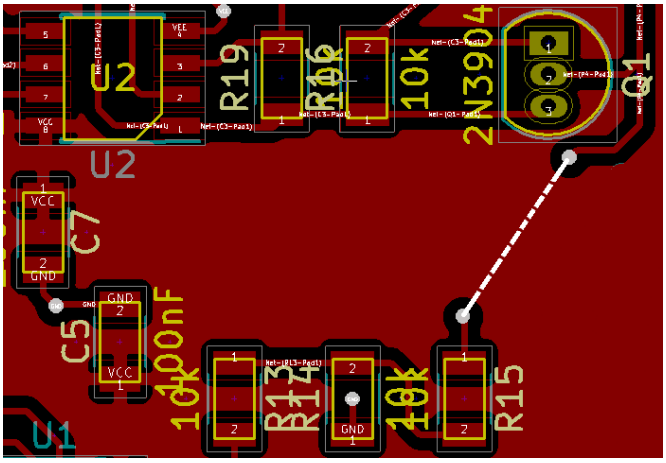

Figura 4.9: Camada superior da PCB onde é possível observar uma ligação feita por através de uma via por forma a permitir o fluxo de Vcc (zona a vermelho).

4.2.3.1 Condensadores de Desacoplamento

O valor de corrente que os circuitos integrados requerem ao longo do seu funcionamento não é constante. Estas variações podem ser bruscas não sendo as fontes de alimentação por vezes capazes de acompanhar a sua necessidade provocando assim uma diminuição da queda de tensão aos terminais do circuito integrado [11]. Para a sua prevenção foram adicionados ao circuito condensadores de desacoplamento que não existiam no circuito original.

Estes condensadores devem ser colocados o mais próximo possível dos terminais de alimentação do circuito integrado (figura 4.10) de maneira a reduzir o tempo que a corrente leva a percorrer entre eles. Neste caso o circuito contém dois circuitos integrados alimentados com Vcc (5V) e Vee (-5V),foram necessários quatro condensadores. Na primeira versão da PCB estes foram colocados na camada inferior da placa. Esta solução revelou ser inviável nas condições disponíveis para o processo de soldadura.

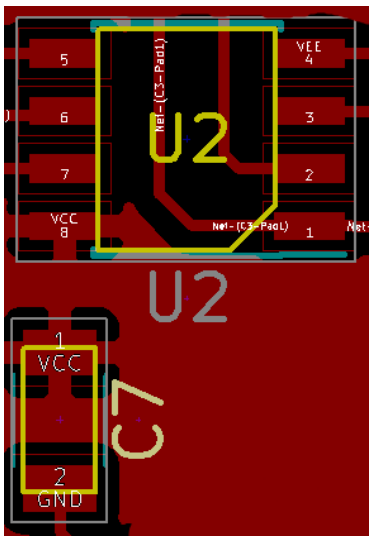

Figura 4.10: Exemplo de um dos condensadores de desacoplamento próximo do circuito integrado

A necessidade destes condensadores se encontrarem próximo dos pinos de alimentação dos circuitos integrados veio aumentar a dificuldade da disposição dos componentes sobre a placa.

4.2.3.2 Pontos de Prova

Sendo uma PCB de dimensões reduzidas tornou-se complicado utilizar ferramentas como um multímetro ou osciloscópio para fazer *debug*. Com isto, ao longo da placa foram colocados diversos pontos de prova. Estes apresentam uma forma circular, perfeita para colocação de sondas de osciloscópio, e encontram-se espalhados pelo circuito mas principalmente entre cada bloco. Na figura 4.11 é possível observar um destes pontos de prova.

Figura 4.11: Exemplo das pontas de teste colocadas ao longo do circuito

4.2.3.3 Furações Auxiliares

Observando o circuito do neurónio no anexo C é possível observar que cada entrada do neurónio se encontra ligada a uma resistência. Estas resistências definem o peso, ou, a influência de cada uma das entradas no funcionamento do circuito.

Com a possibilidade de ser necessário inserir vários sinais no mesmo neurónio e ter de lhes dar pesos diferentes, foram adicionadas furações auxiliares em torno de cada resistência (figura 4.12) simplificando a inserção de uma outra em paralelo utilizando um ferro de soldar.

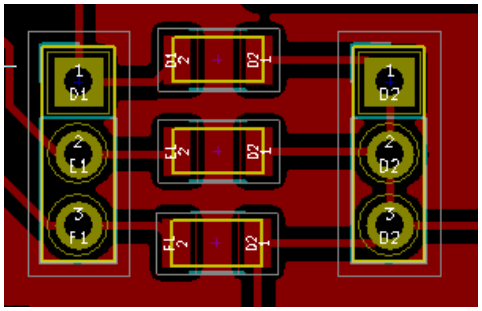

Figura 4.12: Imagem das furações colocadas em paralelo com as resistências por forma a possibilitar a inserção de outras em paralelo.

4.3 Processo de Colocação de Componentes e Soldadura

Concluído o desenho da PCB e após o projeto ter sido enviado para produção foi necessário colocar os componentes na placa. Este processo divide-se em duas partes, pois apesar de na sua maioria o neurónio ser composto por componentes SMD, contém também componentes PTH (*pin through hole*) sendo necessário utilizar dois processos de soldadura diferentes.

Para os componentes SMD o processo consiste na colocação de pasta de solda nos contactos da placa onde serão depois posicionados os componentes. Neste processo de colocação da pasta de solda é normalmente utilizada uma folha metálica chamada *stencil* com furação idêntica à localização dos contatos da placa. Sobrepõe-se o *stencil* à PCB fazendo corresponder a furação com os contatos (figura 4.13) permitindo então ser colocada pasta de solda que é "raspada"ao londo do *stencil*, retirando o excesso e deixando pasta de solda sobre os contatos da PCB.

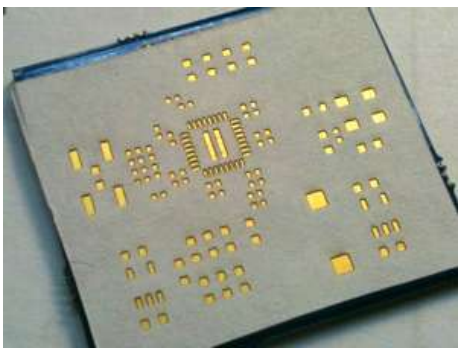

Figura 4.13: Exemplo de um *stencil* colocado sobre uma PCB deixando só os os contatos da placa visíveis

Devido à falta de experiência e de conhecimento durante o processo, no envio para produção da PCB não foi encomendado o *stencil* pelo que a pasta de solda teve de ser colocada utilizando um pequeno estilete, o que se revelou muito complicado devido à pequena dimensão dos contactos na placa. A primeira preocupação, estava sobre o controlo da quantidade de pasta a ser colocada em cada contacto, com receio de criar ligações indesejadas no circuito, devido a pasta de solda a mais. Felizmente esta situação não se verificou, a pasta de solda, depois de levada ao forno, retraiu para os contactos metálicos dos componentes e da placa pelo que seria necessário uma quantidade extremamente exagerada para que as ligações indesejadas acontecessem.

Após colocada a pasta e os componentes na placa, esta é levada ao forno. O processo de soldadura no forno consiste numa variação da temperatura ao longo do tempo levando a pasta a derreter e novamente a solidificar. Estas variações e valores de temperatura dependem inteiramente da pasta de solda que está a ser utilizada sendo necessário consultar os dados do fabricante. Neste caso a pasta utilizada é a *Easy Print Sn62Pb36Ag2* que requer um processo de soldadura apresentado na figura 4.14. Para tal, no forno utilizado, a temperatura pode ser variada manualmente ao longo do tempo ou pode ser utilizado um dos programas pré existentes que corresponda ao pretendido. Neste caso optamos pela utilização dum programa já existente.

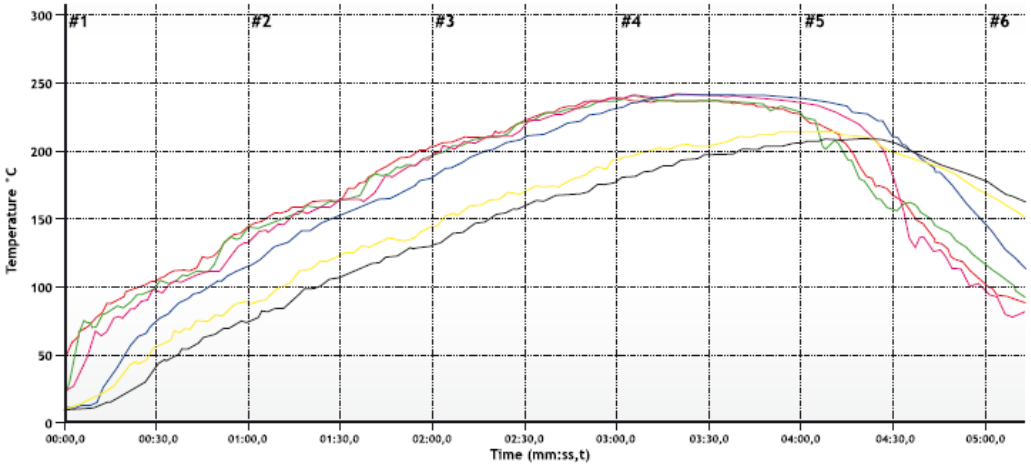

Figura 4.14: Gráfico que representa a evolução da temperatura ao longo do tempo para o processo de soldadura com a pasta *Easy Print Sn62Pb36Ag2*

Terminado processo no forno é necessário então soldar todos os componentes PTH com um ferro e estanho. Estes consistem unicamente nos pinos de ligação, pontas de prova e num transistor. Findado este processo, o neurónio encontra-se concluído e pronto a ser utilizado, figura 4.15.

Figura 4.15: PCB final após processo de soldadura

Capítulo 5

Testes e Conclusões

No último capítulo desta dissertação, são referidos os testes de resolução do percurso tal como a ligação do robô ao neurónio provando a capacidade de guardar um estado de memória no mesmo. Alguns destes testes, nomeadamente os de resolução do percurso, foram feitos ao longo do desenvolvimento do trabalho tendo levado à implementação descrita no capítulo três.

Será feita uma conclusão reflectora dos resultados obtidos e possíveis investigações futuras do mesmo.

5.1 Testes de Resolução de Percurso

Os testes de resolução de percurso correram como se previa. O algoritmo pensado leva o robô a deslocar-se na linha conseguindo encontrar o fim do trajecto.

Foram apenas encontradas dificuldades ao nível da velocidade do robô e na resolução da bifurcação. A velocidade do robô encontra-se dependente do valor de tensão empregue ao pino de *enable* da ponte H, porém, à mínima variação desse valor, o robô passava de não ter força para se mover a ter uma velocidade extremamente alta, o que promovia muito escorregamento nos motores sempre que era necessário parar uma das rodas. A solução, como mencionada anteriormente, passa por colocar o robô em movimentos espaçados. O segundo problema consiste na detecção de fim de percurso ou *stand by*. Esta detecção entrou em conflito com a detecção de bifurcação pois a primeira necessitava de todos os sensores activos e a segunda de apenas dos quatro centrais. A abordagem do robô à linha final nem sempre se deu de forma perpendicular, pelo que foi comum um dos sensores das pontas não detectar a linha transversal, despoletando o estado de bifurcação em vez de parar. A solução passou por activar também a situação de fim de percurso mesmo quando sete sensores se encontravam activos e um das pontas inactiva, tal como referido

no capitulo três.

5.2 Teste e Alteração do Estado de Memória do Neurónio

Durante os primeiros testes da resolução de percurso, para verificar que o processo de implementação de memória estava correctamente implementado, foram utilizados dois LEDs. Estes, foram ligados aos pinos do Arduíno programados para enviar o impulso aos neurónios. Assim, a alteração de estado de cada neurónio de memória correspondia ao piscar de um dos LEDs.

Verificou-se que o sistema de memória estava bem implementado sem haver necessidade de qualquer ajuste.

Estando os neurónios em PCB prontos, era necessário testá-los e verificar que é possível promover a alteração do seu estado utilizando impulsos do Arduíno.

O neurónio funcionou como esperado, contudo, as frequências de *spikes* dos dois estados eram 120Hz e 126Hz. Estes valores são muito próximos e não correspondem às frequências já conhecidas das simulações e montagens em *breadboard* anteriores a este trabalho.

Com a verificação do funcionamento do neurónio em PCB verifica-se também que a trânsição entre os estados do neurónio está dependente da variação da tensão aos seus terminais de entrada entre um valor positivo e negativo, ou seja, um dos estados do neurónio era activado quando recebia um valor superior a 3V e o outro com um valor inferior a -1V sensivelmente. Deste modo existe um problema pois o arduíno apenas consegue fornecer valores entre 0V e 5V.

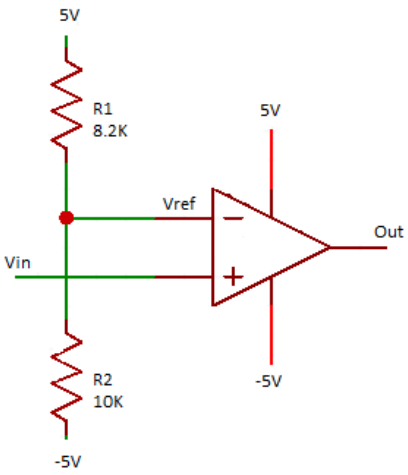

Figura 5.1: Circuito comparador de tensão

Para solucionar esta questão foi necessário colocar um circuito comparador entre o arduíno e o neurónio, figura 5.1. Com este circuito o objectivo é obter à saída um valor de 5V ou -5V quando o sinal de entrada estiver acima ou a baixo de um determinado valor

de referência. Estes valores de saída são teóricos (e equivalem aos valores de alimentação) pelo que com a utilização do circuito integrado LM741 apenas se obtém 4.60V e -2.60V estando ainda assim dentro dos valores necessários.

Este comparador não era estável pois na primeira implementação feita as resistências R1 e R2 eram iguais. Estas, determinam o valor de referência para alterar o valor de tensão à saída. Como tinham o mesmo valor, colocavam esse valor de referência em 0V. Devido à existência de ruído, mesmo quando o pino do Arduíno ligado ao comparador se encontrava inactivo, o valor à entrada do comparador era sempre ligeiramente superior a 0V, fazendo com que a saída fosse sempre alta.

Foi necessário então definir um valor de referência.

$$\triangle V = +5 - (5) = 10V$$
$$i = \frac{\triangle V}{R1 + R2}$$
$$Vref = -5 + i * R2$$

Dadas estas equações, considerando $R1 = 8.2K\Omega$ e $R2 = 10K\Omega$.

$$i = 0.55mA$$
$$Vref = -5 + i * R2 = 0.5V$$

Alterando o valor de R1 para $8.2K\Omega$ verifica-se um novo valor de referência que satisfaz a necessidade.

Alterada a resistência foi possível obter o resultado pretendido. A figura 5.2 corresponde à imagem real do osciloscópio utilizado. onde o canal 1 representa a saída do neurónio, o canal 2 a saída do comparador ligada ao neurónio e o canal 3 o pino do arduíno ligado ao comparador. Com esta informação é possível observar que quando o Arduíno coloca tensão no pino de saída, o comparador produz um valor de tensão positiva e obtemos a frequência de 120Hz no neurónio. Quando o pino do arduíno se encontra a 0V, o comparador produz uma tensão negativa passando o neurónio a ter uma frequência de 126Hz.

5.3 Conclusão e Trabalho Futuro

Como esperado, o sistema desenvolvido funciona atingindo assim a meta estabelecida. Mas, como qualquer outro sistema, pode e deve sofrer melhorias.

No robô seria interessante procurar a utilização de outros motores para desenvolver o seu movimento e tentar criar um deslocamento com uma velocidade estabilizada.

O neurónio causou problemas nos testes devido à necessidade de empregar um comparador para alterar o seu estado. Consequentemente, seria importante desenvolver uma nova versão de neurónio que tivesse o seu valor de transição entre 0V e 5V. Infelizmente este mesmo neurónio não transita de estado com a recepção de impulsos necessitando

Figura 5.2: Sinais no osciloscópio correspondentes à saída do Arduino (canal 3), saída do comparador (canal 2) e saída do neurónio (canal 1) durante o teste de transição de estados

de valores constantes nos seus terminais para manter os estados. Este não era o funcionamento esperado, obrigando a que a máquina de estados contenha também um estado de memória em vez deste existir, apenas nos neurónios, como era expectado inicialmente.

Por último e o mais importante, sendo possível promover todas as alterações já mencionadas, seria extremamente interessante passar o máximo de processamento (se não todo) para neurónios, acabando com a existência de um micro controlador no sistema.

Bibliografia

[1] S. Aaronson. *Quantum Computing Since Democritus*. Quantum Computing Since Democritus. Cambridge University Press, 2013. ISBN: 9780521199568.

[2] S. Beer. *Cybernetics and management*. Management science series. Wiley, 1959.

[3] G. Boole, G. Smith, A. De Morgan e G. Smith. *The Boole-De Morgan Correspondence, 1842-1864*. Oxford logic guides. Clarendon Press, 1982. ISBN: 9780198531838.

[4] M. Carandini. "Area V1". Em: *Scholarpedia* 7.7 (2012), p. 12105.

[5] A. M. Chan, F. T. Sun, E. H. Boto e B. M. Wingeier. "Automated seizure onset detection for accurate onset time determination in intracranial EEG". Em: *Clinical Neurophysiology* 119.12 (2008), pp. 2687–2696.

[6] Z. Chen. "A Primer on Neural Signal Processing". Em: *IEEE Circuits and Systems Magazine* 17.1 (), pp. 33–50.

[7] D. Cohen. *Introduction to computer theory*. A Wiley international edition. John Wiley & Sons Australia, Limited, 1986. ISBN: 9780471802716.

[8] D. Cryan, S. Shatil e B. Mayblin. *Introducing Logic: A Graphic Guide*. Introducing... Icon Books Limited, 2014. ISBN: 9781848317611.

[9] G. Dornhege. *Toward brain-computer interfacing*. MIT press, 2007.

[10] L. R. Hochberg, M. D. Serruya, G. M. Friehs, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn e J. P. Donoghue. "Neuronal ensemble control of prosthetic devices by a human with tetraplegia". Em: *Nature* 442.7099 (2006), pp. 164–171.

[11] C. J. Kaiser. *The capacitor handbook*. Springer Science & Business Media, 2012.

[12] W. Kneale e M. Kneale. *O Desenvolvimento da Lógica*. Fundação Calouste Gulbenkian, 1962. ISBN: 972-31-0532-2.

[13] S. Koziol, S. Brink e J. Hasler. "A neuromorphic approach to path planning using a reconfigurable neuron array IC". Em: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2724–2737.

[14] J. Largeault e R. Herken. *The Universal Turing Machine. A Half-Century Survey*. 1992.

[15] N. Malcolm, G. H. Wright e L. Wittgenstein. *Ludwig Wittgenstein: a memoir*. Oxford University Press, 2001.

[16] C. Mortari. *Introdução à lógica*. Comped, 2001. ISBN: 9788571393370.

[17] T. Natschläger e W. Maass. "Spiking neurons and the induction of finite state machines". Em: *Theoretical Computer Science* 287.1 (2002), pp. 251–265.

[18] K. Rosen. *Discrete Mathematics and Its Applications*. Discrete Mathematics and Its Applications. McGraw-Hill Higher Education, 2007. ISBN: 9780072880083.

[19] M. M. Shanechi, G. W. Wornell, Z. M. Williams e E. N. Brown. "Feedback-controlled parallel point process filter for estimation of goal-directed movements from neural signals". Em: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 21.1 (2013), pp. 129–140.

[20] C. E. Shannon. "A symbolic analysis of relay and switching circuits". Em: *Transactions of the American Institute of Electrical Engineers* 57.12 (1938), pp. 713–723.

[21] T. Smiley. "Syllogism and quantification". Em: *The Journal of Symbolic Logic* 27.01 (1962), pp. 58–72.

[22] I. H. Stevenson e K. P. Kording. "How advances in neural recording affect data analysis". Em: *Nature neuroscience* 14.2 (2011), pp. 139–142.

[23] *Teoria de Sistema e Sinais - Instituto Superior Tecnico*. 1978.

Apêndice A

Esquemático do Robô

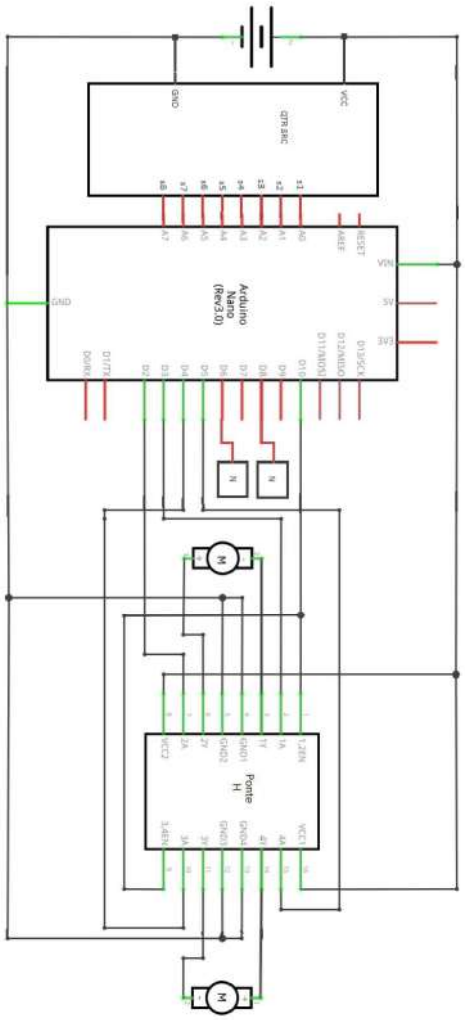

Figura A.1: Esquemático correspondente a todas as ligações do robô

APÊNDICE B

CÓDIGO ARDUÍNO

```
#define EF 2
#define ET 3
#define DF 4
#define DT 5
#define ENABLE 10
#define V1 6
#define EV1 7
#define V2 8
#define EV2 9

int sensorValues[9];

typedef struct no{
  int biCount;
} NO;

int biSeq;
int noIndex;
NO nos[2];

void bifurcacao();
void readAnalogSensors();
void escreve(char* text);
void caminhoErrado();
```

Figura B.1: Código correspondente às definições de variáveis globais e cabeçalhos das funções implementadas

```
void setup() {
  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(A2, INPUT);
  pinMode(A3, INPUT);
  pinMode(A4, INPUT);
  pinMode(A5, INPUT);
  pinMode(A6, INPUT);
  pinMode(A7, INPUT);

  pinMode(ENABLE, OUTPUT);

  pinMode(EF, OUTPUT);
  pinMode(ET, OUTPUT);
  pinMode(DF, OUTPUT);
  pinMode(DT, OUTPUT);
  pinMode(V1, OUTPUT);
  pinMode(EV1, OUTPUT);
  pinMode(V2, OUTPUT);
  pinMode(EV2, OUTPUT);

  Serial.begin(9600);

  nos[0].biCount=0;
  nos[1].biCount=0;
  biSeq = 0;
  noIndex = 0;

  analogWrite(10, 200); //200 e delay 50
}
```

Figura B.2: Código correspondente ao *setup* do Arduíno

```
void loop() {
  readAnalogSensors();

  //FIM
  if((!sensorValues[0]&&sensorValues[1]&&sensorValues[2]&&sensorValues[3]&&sensorValues[4]&&sensorValues[5]&&sensorValues[6]&&sensorValues[7])
     ||(sensorValues[0]&&sensorValues[1]&&sensorValues[2]&&sensorValues[3]&&sensorValues[4]&&sensorValues[5]&&sensorValues[6]&&!sensorValues[7])
     ||(sensorValues[0]&&sensorValues[1]&&sensorValues[2]&&sensorValues[3]&&sensorValues[4]&&sensorValues[5]&&sensorValues[6]&&sensorValues[7])){
    escreve("|FIM|");
    delay(60000);
  }

  //Caminho Errado
  if(!sensorValues[1]&&!sensorValues[2]&&!sensorValues[3]&&!sensorValues[4]&&!sensorValues[5]&&!sensorValues[6]){
    caminhoErrado();
    biSeq=0;
  }

  //Bifurcação
  if(sensorValues[2]&&sensorValues[3]&&sensorValues[4]&&sensorValues[5]){ //Ponderar detectar com os dois do meio mais 1 dos laterais
    if(biSeq){ //Se estiver a 1 é porque não apanhou nem turn around antes desta bifurcação
      if(noIndex==0){
        noIndex=1;
      }
      else{
        noIndex=0;
      }
    }
    bifurcacao();

    nos[noIndex].biCount++;
    if(nos[noIndex].biCount==2){ //altera o virabicos sempre que encontra duas bifurcações num nó
      if(noIndex==0){
        digitalWrite(V1, HIGH);
        escreve("ViraBicos-1");
        delay(100);
        digitalWrite(V1, LOW);
      }
      else{
        digitalWrite(V2, HIGH);
        escreve("ViraBicos-2");
        delay(100);
        digitalWrite(V2, LOW);
      }
    }
    biSeq=1;
  }
```

Figura B.3: Código correspondente à activação de estados

```
//Andamento Simples! Tem que ter uma grande condição devido a dificuldade de acionar 2 sensores em cima da linha
if((sensorValues[3]&&sensorValues[4])
    || (!sensorValues[2]&&!sensorValues[3]&&sensorValues[4]&&!sensorValues[5])
    || (!sensorValues[2]&&sensorValues[3]&&!sensorValues[4]&&!sensorValues[5])){
  escreve("frente");
  digitalWrite(DF, HIGH); digitalWrite(EF, HIGH);
  delay(50);
  digitalWrite(DF, LOW); digitalWrite(EF, LOW);
}
else  if(sensorValues[2]&&sensorValues[3]){
        escreve("esquerda");
        digitalWrite(DF, HIGH);
        delay(50);
        digitalWrite(DF, LOW);
      }
      else if(sensorValues[4]&&sensorValues[5]){
              escreve("direita");
              digitalWrite(EF, HIGH);
              delay(50);
              digitalWrite(EF, LOW);
            }
delay(250);
```

Figura B.4: Continuação do código correspondente à activação de estados

```
void escreve(char* text){
  Serial.print(sensorValues[0]); Serial.print(sensorValues[1]); Serial.print(sensorValues[2]);
  Serial.print(sensorValues[3]); Serial.print(sensorValues[4]); Serial.print(sensorValues[5]);
  Serial.print(sensorValues[6]); Serial.print(sensorValues[7]);
  Serial.print(" ");
  Serial.print(text);
  Serial.print("\n");
}

void bifurcacao(){
  if(noIndex==0)
    digitalWrite(EV1, HIGH);
  else
    digitalWrite(EV2, HIGH);

  do{
    escreve(Bi1);
    digitalWrite(DF, HIGH); digitalWrite(EF, HIGH);
    delay(50);
    digitalWrite(DF, LOW); digitalWrite(EF, LOW);
    delay(250);
    readAnalogSensors();
  }while(!(!sensorValues[3]&&!sensorValues[4]));

  do{
    escreve(Bi2);
    digitalWrite(EF, HIGH);
    delay(50);
    digitalWrite(EF, LOW);
    delay(250);
    readAnalogSensors();
  }while(!(!sensorValues[2]&&sensorValues[3]&&sensorValues[4]&&!sensorValues[5]));

  digitalWrite(EV1, LOW);
  digitalWrite(EV2, LOW);
}
```

Figura B.5: Funções auxiliares

```
void caminhoErrado(){
  do{
    digitalWrite(DT, HIGH); digitalWrite(EF, HIGH);
    delay(50);
    digitalWrite(DT, LOW); digitalWrite(EF, LOW);
    escreve("Caminho errado");
    delay(250);
    readAnalogSensors();
  } while(!(sensorValues[3]&&sensorValues[4]));
}

void readAnalogSensors(){
  sensorValues[0]=analogRead(A0);
  sensorValues[1]=analogRead(A1);
  sensorValues[2]=analogRead(A2);
  sensorValues[3]=analogRead(A3);
  sensorValues[4]=analogRead(A4);
  sensorValues[5]=analogRead(A5);
  sensorValues[6]=analogRead(A6);
  sensorValues[7]=analogRead(A7);
  sensorValues[8]=sensorValues[0];

  for(int i=0; i<8; i++){
    if(sensorValues[i]>50) //threshold de 50
      sensorValues[i]=1;
    else
      sensorValues[i]=0;
  }
}
```

Figura B.6: Continuação das funções auxiliares

APÊNDICE C

PCB

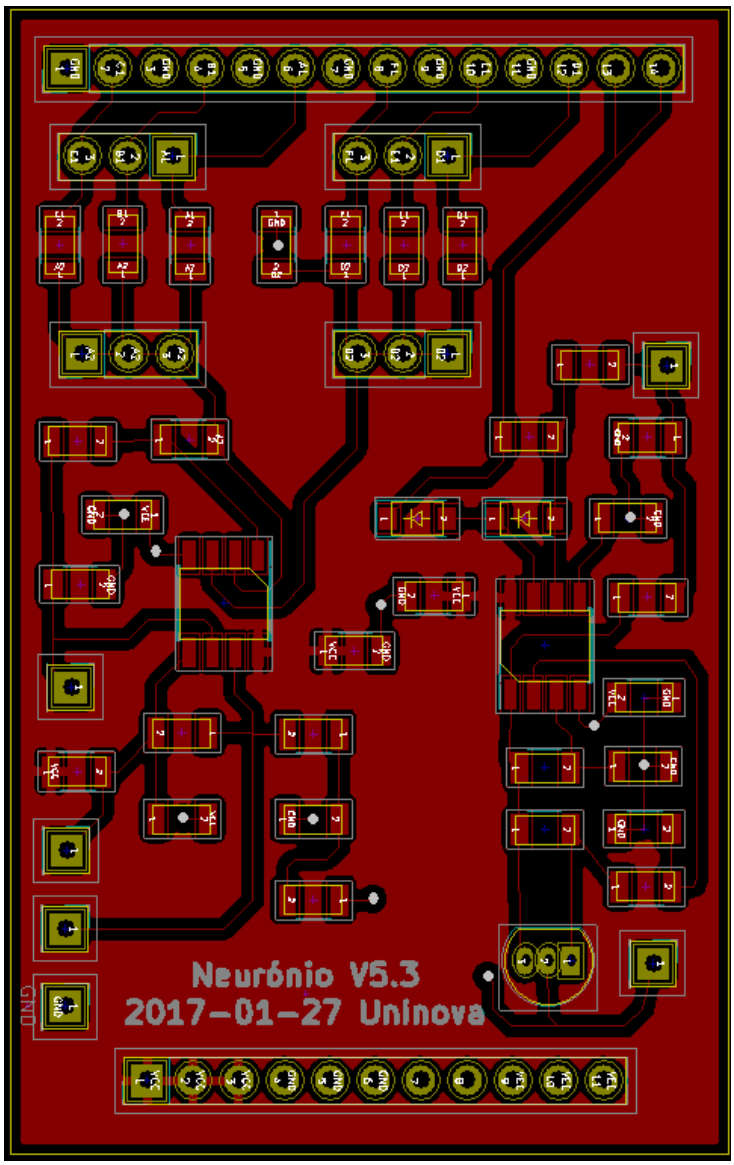

Figura C.1: Camada superior da pcb onde se pode observar a colocação dos componentes e pistas

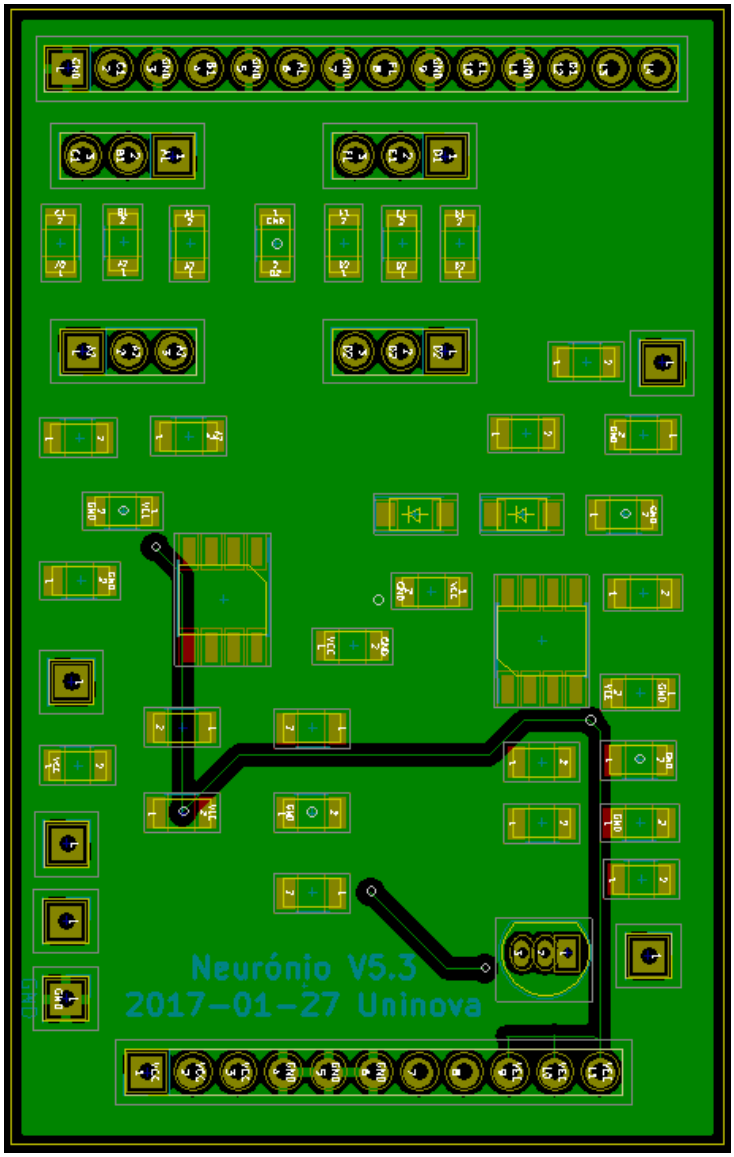

Figura C.2: Camada inferior da pcb onde se pode observar a colocação dos componentes e pistas