ANDRÉ LUIZ SIMONAGIO GRANA

Sistema de controle de navegação embarcado para um robô móvel autônomo baseado no computador de baixo custo Raspberry PI

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em Eletrônica

ORIENTADOR: Prof. Dr. Eduardo do Valle Simões

São Carlos
2013

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Grana, André Luiz Simonagio
G748s Sistema de controle de navegação embarcado para um robô móvel autônomo baseado no computador de baixo custo Raspberry PI / André Luiz Simonagio Grana; orientador Eduardo do Valle Simões. São Carlos, 2013.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2013.

1. Robótica Móvel. 2. Controle Embarcado. 3. Navegação Autônoma. 4. Raspberry Pi. I. Título.

FOLHA DE APROVAÇÃO

Nome: André Luiz Simonagio Grana

Título: "Sistema de controle de navegação embarcado para um robô móvel autônomo baseado no computador de baixo custo Raspberry PI"

Trabalho de Conclusão de Curso defendido e aprovado em 26/11/2013,

com NOTA 9,0 (Nove, zero), **pela Comissão Julgadora:**

Prof. Dr. Eduardo do Valle Simões - (Orientador - SSC/ICMC/USP)

Prof. Associado Evandro Luís Linhari Rodrigues - (SEL/EESC/USP)

Prof. Dr. José Roberto Boffino de Almeida Monteiro - (SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Para Adriano, Rosa e
Letícia, com amor.

Agradecimentos

Ao Simões, por ser um verdadeiro norte durante o desenvolvimento desse trabalho e durante minha graduação.

À minha família, por oferecer muito mais do que o suporte financeiro necessário para realizar este curso, oferecer um porto seguro nos momentos de incerteza.

Ao meu amigo André Aureliano, por fornecer a fagulha de onde surgiu a paixão por dispositivos embarcados.

E, em especial, à minha namorada, Letícia, que me deu força e inspiração para concluir esse projeto e com quem pude sempre contar nos momentos mais difíceis desses últimos 5 anos nos quais se passaram o curso.

Sumário

Lista de Figuras

Resumo

Este projeto descreve o desenvolvimento de um sistema de navegação para um robô móvel autônomo de pequeno porte que utiliza a placa *Raspberry Pi*. Essa placa é um computador de dimensões reduzidas com um processador *ARM (Advanced RISC Machine)*, que foi projetada para ser utilizada no ensino da ciência da computação, e por isso possui baixo custo. O sistema de navegação é responsável por conduzir o robô com segurança em um ambiente contendo vários obstáculos de diversos formatos. Este sistema deve perceber a situação que o robô se encontra a cada iteração, com base na leitura dos sensores de proximidade dos obstáculos, e determinar a manobra mais adequada para evitar colisões. É proposto um módulo de sensoriamento de obstáculos baseado em sonares, que alimentam uma rede neural do tipo *RAM (Random Access Memory)* ou *n-tuple classifier*. Os pesos, ou os conteúdos dos neurônios, dessa rede neural são tratados como os cromossomos dos indivíduos de uma "população" de soluções e são otimizados por um algoritmo evolutivo. A utilização da *Raspberry Pi* como controladora de hardware exigiu o desenvolvimento de uma placa adicional para proteção e interface com os motores e sonares. O robô utilizado nesse projeto possui duas esteiras laterais e dimensões de 13 cm de largura por 18 cm de comprimento, e o controle implementado é do tipo reativo puro, que processa as entradas dos sensores e toma decisões, tendo como resultado esperado o deslocamento do robô pelo ambiente sem que ocorram colisões.

Palavras-chave: Robótica móvel, Controle Embarcado, Navegação Autônoma, Raspberry Pi.

Abstract

This Project describes the development of a navigation system for a small-size autonomous mobile robot that uses the board Raspberry Pi. This credit-card-sized board is a single-board computer that includes an ARM processor, and it was developed with the intention of promoting the teaching of computer science, and therefore has a low cost. The navigation system it is responsible for driving the robot safely in an environment with obstacles that have various sizes and shapes. This system should notice, at each iteration, the situation in which the robot is with the information provided from the sonar sensor, and decide the best move for avoiding collision. It is proposed a sonar-based obstacles sensing module, that feeds a RAM (or n-tuple classifier) neural network. The weights, or the neurons content, of this neural network are treated as the individual chromosomes of a population of solutions, and they are optimized using an evolutionary algorithm.The using of the Raspberry Pi as the hardware controller required the development of an additional board for protection and interface with the motors and the sonar sensors. In this project, it is used a 13 cm width and 18 cm length robot that has two lateral treads. It is implemented a pure-reactive controller, that proccess the inputs from the sensors and make decisions of how to activate the motors. The robot is expected to moving in the environment without colliding.

Keywords: Mobile Robotics, Embedded Control, Autonomous Navigation, Raspberry Pi.

1. Introdução

A busca por um sistema robótico de exploração em ambientes desconhecidos e dinamicamente mutáveis tem utilizado cada vez mais sistemas multirrobóticos (VOLKOV, 2013) (WANG, DANG e PAN, 2013) ao invés de um robô centralizando todas as funcionalidades. Uma das vantagens de um sistema com um enxame de robôs que cooperem para a solução de um problema em relação a um sistema com um único robô é a maior tolerância a falhas, uma vez que existe redundância, ou seja, se um robô falhar ou parar de funcionar, outros poderão assumir suas tarefas e o sistema multirrobótico pode cumprir seu objetivo com êxito. Outro ponto favorável à utilização de sistemas multirrobóticos é a possibilidade de construção de robôs heterogêneos, podendo, por exemplo, um robô ser especializado em exploração e, portanto, possuir a característica de ser mais veloz e mais leve, e outro robô especializado em transporte, contendo sistema de coleta e possuindo mais potência para locomoção, sendo o segundo acionado quando o primeiro encontrar o objetivo em questão. Essa heterogeneidade leva a maior eficiência energética, pois como no exemplo descrito, o sistema de coleta e armazenamento é transportado apenas no momento da aquisição do objetivo e não durante a busca.

Esta solução, entretanto, também tem suas limitações, pois construir um grande número de robôs impõe normalmente maiores custos com hardware e, para que seja viável desenvolver um time de pequenos robôs ao invés de um maior contendo muitos recursos, os que compõem o time devem possuir baixo custo de fabricação. Torna-se interessante, portanto, o uso de um circuito de controle embarcado que exija poucos recursos financeiros com processadores de menor custo possível. Por isso, é comum em trabalhos que envolvam um número grande de robôs, a utilização de microcontroladores de baixo custo como *PIC, MSP430* ou *ATMega* (AMSTUTZ, CORRELL e MARTINOLI, 2008), (BERGBREITER, 2003) e (SASMAL, NAYAK e PATNAIK, 2012).

A necessidade de processamento embarcado vem crescendo com o aumento da complexidade das aplicações, e os microcontroladores de baixo custo muitas vezes não conseguem atender aos requisitos como processamento de vídeo, acesso à rede, entre outros (GERKEY, VAUGHAN e HOWARD, 2003) (HATA, SHINZATO e WOLF, 2010) (SE, DAVID e LITTLE, 2001). Soma-se a essa limitação o fato de que os algoritmos de inteligência artificial que controlam a navegação, o mapeamento e o planejamento de trajetória dos robôs são cada vez mais complexos, como na utilização de uma *RNA (Rede Neural Artificial)* em conjunto com um *AE (Algoritmo Evolutivo)* adaptando constantemente o comportamento do robô às variações no ambiente (HOFFMANN, 2001). Isso dificulta, portanto, a utilização de microcontroladores simples, como os descritos anteriormente, pois normalmente possuem baixa velocidade de processamento e pouca memória para o software de controle do robô.

É preciso então utilizar um hardware de maior capacidade computacional, tornando viável a utilização de algoritmos mais custosos em relação à memória e ao processamento, e que possa executar concorrentemente sistemas operacionais que tornam transparente a interface com periféricos como câmera, teclado, monitor, além de gerenciar a rede e o armazenamento permanente. Isso implica em um maior custo no hardware de controle, o que limita a quantidade de robôs que um time pode conter, portanto, se torna importante a investigação de hardware de baixo custo, com poder de processamento e que podem executar tais algoritmos mais complexos, como as placas *BeagleBone*, fabricada pela BeagleBoard; a *Mini210s*, da FriendlyARM; e outras que podem ser encontradas na lista disponível em (List of single-board computers, 2013).

A placa microprocessada *Raspberry Pi* foi desenvolvida com objetivo de estimular o ensino de ciência da computação básica em escolas, porém, por ser um hardware de baixo custo, com dimensões pequenas (8,6 cm de comprimento por 5,4 cm de largura) baixo consumo de energia (aproximadamente 3,5 W) e possuir todos os componentes soldados na mesma placa, com exceção do cartão de memória responsável pelo armazenamento permanente, se tornou bastante comum entre hobistas, apresentando vários projetos de hardware e software livres na Internet (About us | Raspberry Pi, 2012). Ela foi introduzida no mercado em fevereiro de 2012 e rapidamente se popularizou pela infinidade de aplicações possíveis. Possui um processador baseado em *ARM* de 700 MHz, 512 MB de memória *RAM*, além de uma *GPU (Graphics Processing Unit)* integrada ao processador, construído com a tecnologia *System on Chip*, ou seja, traz *CPU (Central Processing Unit)*, *RAM* e *GPU* em um mesmo circuito integrado (FAQs | Raspberry Pi, 2013), o que maximiza o desempenho e minimiza o consumo. O armazenamento não-volátil é feito em um cartão *SD*, e possui portas *USB*, *Ethernet* e *HDMI*, entre outras, o que facilita a integração com outros hardwares. Esta placa é um computador de baixo custo baseado em um processador *RISC*, e possui grande respaldo da comunidade envolvida com software livre. Existem distribuições compiladas para o hardware descrito com finalidades específicas, que vão de centrais de mídia a servidores *WEB* (RPi Distributions - eLinux.org, 2013). Nesse projeto será utilizado um *SO* de uso geral: o Raspbian (Raspbian About, 2012).

O sistema de navegação autônoma de robôs proposto será responsável por executar o sensoriamento do ambiente para que colisões sejam evitadas. Assim, o processamento dos dados coletados irá informar ao software de controle de navegação a situação atual em que o robô se encontra e um controlador reativo (WOLF, SIMÕES, *et al.*, 2009) deve encontrar a manobra adequada para evitar colisões com obstáculos. Para viabilizar uma navegação adaptativa em ambientes dinâmicos e ruidosos, como ambientes externos ou internos com presença de pessoas, será avaliada a utilização de redes neurais artificiais e algoritmos evolutivos embarcados no hardware do robô.

A utilização da *Raspberry Pi* nesse projeto facilita a programação por possuir um Linux embarcado, tornando transparente uma programação de baixo nível que pode dificultar a compreensão de um algoritmo que trate estruturas de dados mais complexas, como matrizes tridimensionais, utilizadas na rede neural a ser adotada. As conexões presentes na placa possibilitam ainda fácil adaptação de hardware de terceiros, e como exemplo disso, será utilizado um módulo *Wi-Fi* conectado a uma das duas portas *USB* disponíveis na placa. O gerenciamento de conexões de rede também é facilitado, pois o mesmo é implementado no sistema operacional utilizado.

O custo dessa facilidade é o consumo maior de energia, se comparado com um microcontrolador com menor poder de processamento e que não tenha que executar um sistema operacional de uso geral, também é perdido o controle de tempo real, quando não utilizado um sistema operacional próprio. Outro obstáculo a ser transposto é a menor disponibilidade de portas para acionamento de hardware de baixo nível, como por exemplo, os sonares e motores a serem utilizados no robô. Apesar de possuir dezessete portas digitais para entrada e saída de uso geral, o número de periféricos é limitado: uma porta serial, uma porta I^2C e um *PWM (Pulse-Width Modulation)* no barramento de pinos disponíveis para conexão, enquanto em um microcontrolador, com custo muito menor, é normalmente encontrado quatro ou mais portas com PWM, ao menos um periférico serial e I^2C, além de conversores *AD*, não encontrados na *Raspberry Pi*.

Uma abordagem que vem sendo utilizada para solucionar a limitação de periféricos é a instalação de hardware auxiliar dedicado para geração de sinal com *PWM* através da porta I^2C, como feito em (TOWNSEND, 2013). Com essa abordagem, o objetivo principal de controle de motores com a *Raspberry Pi* foi atingido, porém adicionando custo ao projeto, incluindo uma placa auxiliar e necessitando de tratamento extra para controle desse hardware.

Outro método para desviar dessa limitação do número portas com *PWM* utilizado foi a implementação de *PWM* por software (KAR, 2012), o que não exige hardware adicional e pode ser feito utilizando *threads* na linguagem C. Essa técnica pode apresentar problemas, uma vez que a *Raspberry Pi* não possui *RTC* (Real-Time Clock, ou relógio de tempo real, um circuito integrado responsável por manter um controle preciso do tempo) e sua temporização depende exclusivamente de *ticks* do *clock*. Como normalmente é utilizado um sistema operacional com múltiplos processos sendo executados de forma concorrente, a frequência e o *duty cycle* do pulso gerado são imprecisos, sendo desaconselháveis para utilização com servo-motores, como testado e discutido em (Software PWM Library, 2012), e podendo gerar instabilidade na velocidade quando utilizados motores *DC*, como será o caso nesse projeto.

O controle de robôs móveis deve capacitar o robô a se deslocar em um ambiente e reagir a situações encontradas. Essa interação é feita através de sensores e atuadores que podem ser controlados de maneira reativa, deliberativa, ou híbrida (WOLF, SIMÕES, *et al.*, 2009). No primeiro tipo de controle, as informações dos sensores são processadas instantaneamente, gerando comandos para os atuadores. No controle deliberativo existe um plano prévio, como um mapa, para que haja a tomada de decisões. Porém o controle puramente deliberativo pode ser inadequado quando houver eventos imprevistos. Como solução frequentemente adotada, é utilizado um controle híbrido.

Existem soluções já implementadas para o controle reativo de robôs móveis utilizando controlador *Fuzzy*, redes neurais baseadas em *MLP* (*Multi-Layer Perceptron*), *GSN* (*Goal-Seek Neuron*), e mesmo *RAM*, apresentando resultados suficientes para ações como desviar de obstáculos ou seguir uma linha. Porém, resultados em (UEBEL, BOTELHO, *et al.*, 1995) indicam que um controle em sistemas com pouca inércia e onde velocidade de reação é exigido, é mais recomendada a utilização de uma rede neural *RAM*, que é mais rápida, por ser constituída de operações básicas com números inteiros.

O aprendizado de redes neurais pode ser feito de maneiras distintas, contando ou não com um tutor, e também existem modelos que utilizam *AE* para treinamento, ajustando não apenas o conteúdo dos neurônios, mas também para dimensionar a arquitetura da rede (SIMÕES, 2000). O treinamento utilizando um *AE* se diferencia dos demais por não ser necessário o conhecimento de como a rede será treinada nem fornecer qualquer tipo de exemplos, pois o aprendizado se torna um processo indireto. As soluções são testadas e seleciona-se a mais adequada, recombinando e gerando novas soluções que também serão testadas, repetindo iterativamente o processo até uma solução suficiente ser encontrada.

Uma característica fundamental desses sistemas é o processo não precisar necessariamente ser interrompido quando uma solução é encontrada, e as iterações podem se manter de forma que as características do controle se ajustem continuadamente às variações no ambiente de trabalho do robô. Utilizar regras ou exemplos para ajustar o controle do robô pode limitar seu desempenho em ambientes mutáveis, podendo o *AE* modificar inclusive a arquitetura da rede e o conteúdo dos neurônios simultaneamente.

Este projeto deve prover um sistema de controle de robôs móveis embarcado em uma placa de baixo custo como o *Raspberry Pi*. Isto possibilita a sua utilização em diversas áreas de interesse, como por exemplo, em ambientes internos no transporte de mercadorias em chão de fábrica, ou na segurança de um prédio; ou, em ambientes externos, na exploração de regiões de difícil acesso e na busca e coleta de objetos.

1.1. Objetivo

O objetivo deste trabalho é avaliar a aplicação da *Raspberry Pi* para implementar um sistema de navegação autônoma para um robô de pequeno porte, com dois motores que acionam esteiras laterais. Sendo viável essa implementação, serão aplicados sensores para detecção de obstáculos e um módulo de comunicação *wireless*. Esse módulo de comunicação deve permitir o monitoramento remoto do comportamento e da situação em que o robô se encontra e o envio de comandos de navegação.

O robô (com dimensões de 18 cm de comprimento e 13 cm de largura) no qual o controle de navegação será embarcado possui dois motores *DC* independentes que movem esteiras laterais, permitindo manobras como girar no próprio eixo, o que pode ser desejado em ambientes com pouco espaço aberto.

Serão utilizados sensores de distância baseados em ultrassom para estimar a distância entre o robô e os obstáculos que deverão ser evitados. Estes sonares são módulos comerciais e utilizam cerca de 70 mW durante a operação, o que é suficiente para uma aplicação embarcada com uma fonte de energia limitada.

Para fornecer potência de maneira controlada que acione os motores, um *driver* comercial será montado em uma placa. Juntamente com esse circuito será feito um que adapte o sinal de acionamento dos sonares e outro que forneça isolação entre os sinais da *Raspberry Pi* e o *driver* de potência.

A alimentação dos motores, da placa de controle e de todos os periféricos embarcados no robô será feita através de três baterias de íon de lítio associadas em série, garantindo aproximadamente 9 Wh, dependendo do conjunto de baterias utilizado.

Para habilitar a comunicação com o robô, será utilizada uma placa de rede sem fio com padrão 802.11 conectada em uma das portas *USB*. Assim, será possível controlar e monitorar o estado do robô em outro computador da rede.

O algoritmo de navegação autônoma deverá ser capaz de avaliar os dados dos sensores e atuar sobre a direção e velocidade dos motores: primeiramente utilizando uma rede neural sem peso do tipo *RAM* e em seguida utilizando um algoritmo genético.

1.2. Organização do documento

Será abordado no capítulo 2: “Desenvolvimento de Hardware e Software embarcado para robôs móveis”, o estudo dos módulos necessários ao projeto, descrevendo as ferramentas utilizadas e seu funcionamento teórico.

A implementação de cada um desses módulos estudados está apresentado no capítulo 3: “Implementação do sistema de controle de navegação”. São descritos os métodos utilizados para o funcionamento de cada módulo independentemente.

Os testes realizados para comprovar o funcionamento dos módulos, tanto de forma independente como interligados, são abordados no capítulo 4: “Resultados”. Também é feita uma breve análise sobre as informações obtidas.

O capítulo 5: “Conclusões”, trás uma discussão sobre o resultado final do trabalho, tal como as possibilidades de desenvolvimento futuro e os principais obstáculos encontrados.

2. Desenvolvimento de Hardware e Software embarcado para robôs móveis

Neste capítulo serão apresentados os temas estudados para a realização do projeto. Será apresentado um breve histórico e as características da placa utilizada, os princípios de funcionamento dos sensores de distâncias escolhidos, conceitos sobre modulação por largura de pulso (*PWM*). Também serão apresentados o *buffer* e o *driver* utilizados para o acoplamento da placa de controle com os motores. Nas últimas seções serão revisados a rede neural e o algoritmo evolutivo implementados no robô.

2.1. *Raspberry Pi* e Linux embarcado

Em 2012 foi lançado o primeiro modelo desse computador de dimensões reduzidas. Os criadores desse hardware tinham como objetivo inicial estimular o ensino de ciência da computação em escolas, buscando recuperar o interesse dos jovens pelo estudo de computação básica (About us | Raspberry Pi, 2012).

Para que fosse amplamente usado e incentivasse realmente o estudo de funções de baixo nível, foi mantido o baixo custo nesse hardware, o que chamou atenção de toda a comunidade de desenvolvedores, fazendo a *Raspberry Pi* ser rapidamente popularizada, o que causou uma fila de espera nos primeiros meses de vendas. Uma fotografia da placa utilizada nesse projeto está reproduzida na Figura 2.1.

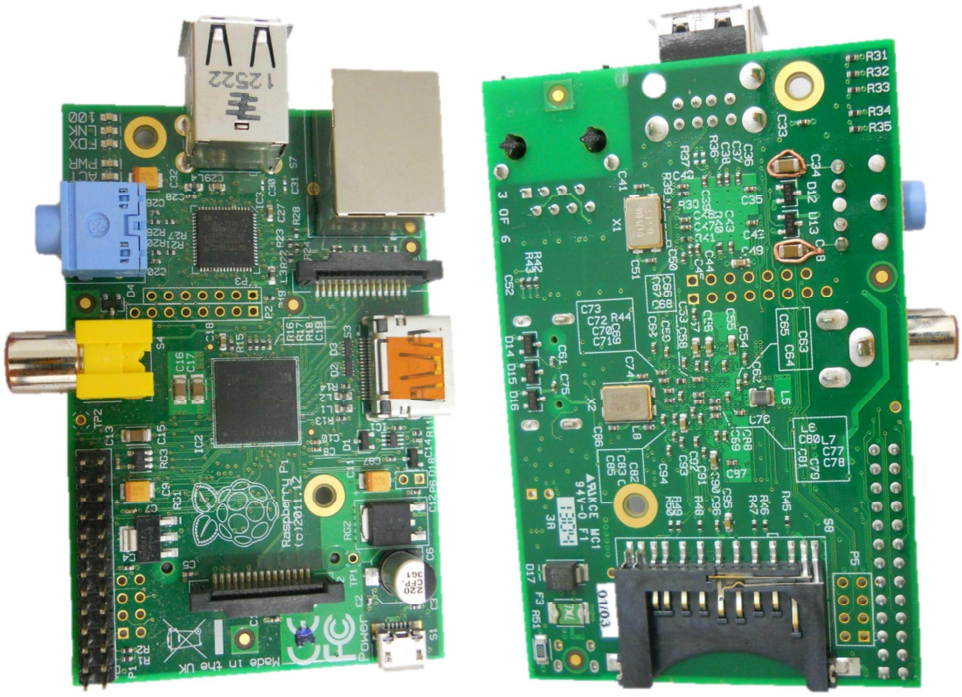


Figura 2.1. Placa *Raspberry Pi* utilizada no projeto.

Como informado em (FAQs | Raspberry Pi, 2013), esse pequeno computador porta um *system on a chip* (SoC) Broadcom BCM2835, que inclui um processador *ARM* 11 de 700 MHz, *GPU* dedicada, e 512 MB de memória *RAM*. Mas o que realmente atraiu olhares da comunidade em geral foi a existência de uma saída *HDMI*, conexão *Ethernet*, duas portas *USB*, entre outras, o que implica em um grande número de periféricos possíveis.

Foi esta relação de custo/benefício que levou a popularização dessa placa. Rapidamente surgiram inúmeras aplicações e projetos, o que contribuiu para o surgimento de uma infinidade de material de referência para outros projetos e ampliou o suporte oferecido pela comunidade, principalmente a envolvida com software livre.

Apesar de não ser vinculado ao fabricante da placa, o sistema operacional que vem sendo mais utilizado é o *Raspbian*, uma variante não oficial do Debian Wheezy armhf, otimizada para o conjunto de instruções *ARMv6* presente na *Raspberry Pi* (RPi Distributions - eLinux.org, 2013). Essa distribuição oferece mais de 35.000 pacotes pré-compilados, abrindo um grande leque de aplicações possíveis para a *Raspberry Pi*, desde centrais de mídia até controle de hardware (Raspbian About, 2012). Não é o mais enxuto possível para a aplicação em um projeto específico, porém possui instalação facilitada e grande suporte da comunidade, levando a uma frequência de atualizações com correções de bugs maior e suporte a possíveis obstáculos na utilização desse *SO*.

Por ter essas características, a *Raspberry Pi* foi escolhida para a implementação do sistema de navegação robótico deste projeto. Mais especificamente, por ser um processador de baixo custo e possuir capacidade de processamento e memória suficientes para executar algoritmos de navegação, planejamento e tomada de decisão mais complexos do que usualmente se obtém com microcontroladores como o *Arduino, PIC, MSP430*, entre outros.

2.2. Sensor de distância por som

Uma alternativa encontrada para detectar obstáculos e assim alimentar o sistema de navegação foi a utilização de sonares, como o reproduzido na Figura 2.2. Esse tipo de sensor utiliza a reflexão de ondas sonoras para aferir a distância em uma direção. Um pulso ultrassônico modulado é emitido e um circuito contendo um sensor piezoelétrico compara os sinais recebidos até que o pulso possua frequência igual à emitida e a mesma sequência de pulsos (HC-SR04 Datasheet, 2012).

A saída do módulo de sensores é um pulso com largura proporcional ao tempo total entre a emissão do pulso e a recepção do mesmo refletido por algum objeto. Conhecendo a velocidade do som aproximada para a temperatura da atmosfera em que o sensor está aferindo as distâncias, é possível determinar a distância de um obstáculo com uma superfície que reflita o som, bastando contar o tempo entre emissão e recepção.

É importante atentar à abertura especificada na folha de dados do sensor que é de 15°, não devendo haver intersecção entre os sonares, ou então o pulso emitido por um sonar pode ser reconhecido por outro sensor, causando um valor discrepante (HC-SR04 Datasheet, 2012).

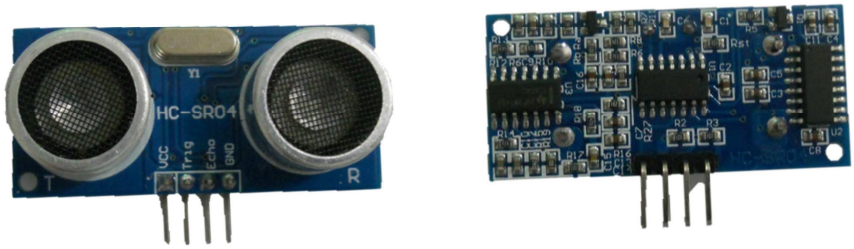


Figura 2.2. Sensor de distância baseado em pulso ultrassônico.

2.3. PWM

O controle das velocidades dos motores foi feito com um chaveamento em frequência constante, uma vez que se dispunha apenas de saídas digitais. Para isso foi utilizado o conceito de pulsos modulados por largura (*Pulse-Width Modulation*), assim, com uma onda quadrada com frequência constante e razão cíclica (*duty cycle*) ajustável é possível transferir uma quantidade de potência desejada através do valor médio de tensão do sinal (AHMED, 2000). Observe na Figura 2.3 uma representação de um pulso modulado com diferentes *duty cycles*, o valor de tensão médio de saída é proporcional a esse valor.

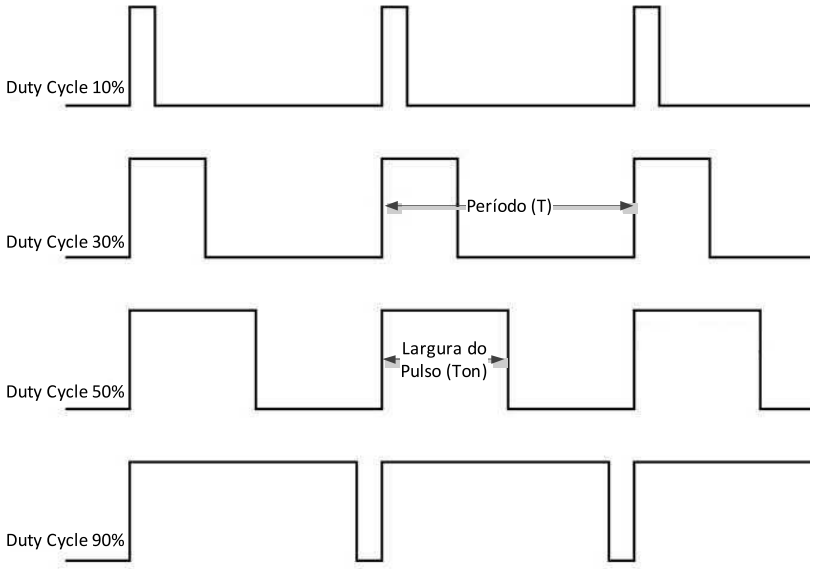


Figura 2.3. Pulso modulado em largura com diferentes razões cíclicas (Duty Cycles).

Em (AHMED, 2000), é dado que a tensão média de saída V_O é

$$V_O = \frac{T_{ON}}{T * V_I}$$

E que a potência de saída pode ser descrita por $P = V_O * I_O$, onde $V_O = V_I * d, d = {T_{ON}}/{T}$ e $I_O = {d * V_I}/{R}$. Assim, a potência de saída é dada por

$$P_O = \frac{(d * V_I)^2}{R}$$

Considerando uma carga completamente resistiva, podemos controlar a potência entregue de maneira proporcional ao quadrado da largura de pulso que é configurada no periférico.

É importante observar a frequência de trabalho para respeitar os limites do hardware de potência utilizado e, em casos onde há carga indutiva, como em motores, é necessário pulsar em uma frequência que faça a corrente se manter estável em uma mesma largura de pulso, fazendo o motor girar suavemente, sem trancos ou ruído audível na frequência do pulso modulado em largura.

2.4. Buffer de acoplamento dos sensores e *driver* dos motores

O acoplamento dos comandos da *Raspberry Pi* com o módulo de potência do robô é feito utilizando opto acopladores, que são compostos de *LEDs* e foto-transistores integrados em um encapsulamento, permitindo o acionamento desacoplado entre as partes (FAIRCHILD, 2010). Porém, para acionar o *LED* de maneira que o foto-transistor entre em região de saturação, que é desejável quando se está trabalhando de forma digital, é necessária uma corrente maior que 20mA (D217 Datasheet, 2013). Essa corrente não pode ser fornecida diretamente pelos pinos do microprocessador da *Raspberry Pi*, então foram utilizados buffers para os sinais que são transmitidos para a parte de potência.

Um *buffer* pode ser feito com um amplificador operacional, com configuração feita como na Figura 2.4 e é obtido uma impedância de entrada muito baixa, além de conseguir fornecer a corrente necessária para o acionamento do *LED*, por exemplo (CLAYTON e WINDER, 2003). A corrente de saída fica limitada apenas pela corrente máxima de saída do amplificador operacional, valor que geralmente pode ser encontrado na folha de dados do componente. Além dessa situação onde a fonte não consegue fornecer a corrente necessária para a aplicação, uma impedância de entrada alta é utilizada também quando a tensão cai com uma carga alta, como o caso de um divisor resistivo, que sofre alteração na tensão quando uma corrente grande é exigida para leitura. Dessa forma, a utilização de *buffers* é uma maneira fácil e que exige um número mínimo de componentes para resolver problemas como os expostos acima. Uma solução de menor custo pode ser obtida utilizando um transistor em região de saturação, porém, por disponibilidade dos componentes, foi utilizada a solução com amplificador operacional.

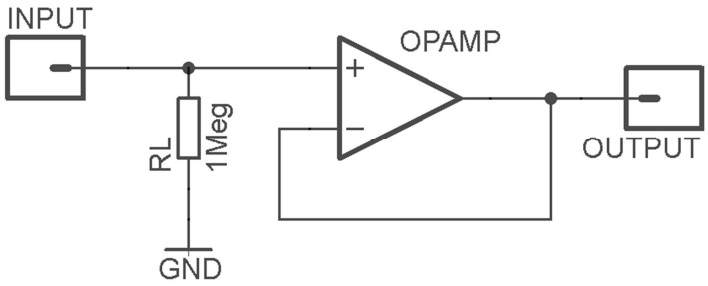


Figura 2.4. Amplificador operacional utilizado como *Buffer*.

2.5. Driver

Para que se possa controlar a velocidade e direção dos motores, é necessário que uma corrente possa fluir em ambas as direções dentro de sua bobina, gerando campos com intensidade e sentidos diferentes. A configuração utilizada para controlar a corrente nesse projeto é um *driver* em ponte, com configuração de ponte H, como pode ser observado na Figura 2.5 (INOUE e OSUKA, 2004).

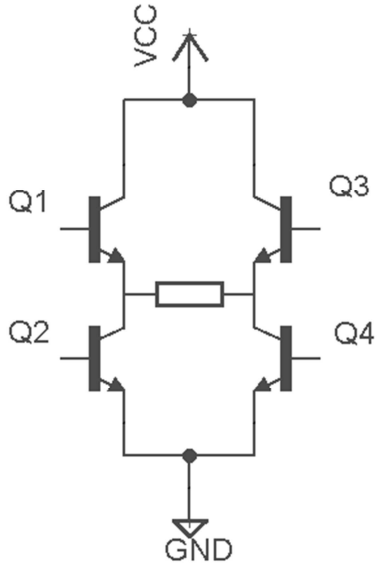


Figura 2.5. Ponte H com uma carga.

São apresentados aqui transistores de junção bipolar, porém também podem ser utilizados MOSFETS ou mesmo chaves mecânicas como Relés. O que difere cada chave é o princípio de funcionamento e o tipo de acionamento, que podem possuir particularidades, principalmente quando são utilizados transistores de efeito de campo e é necessária uma referência entre *Gate* e *Source* para acioná-los, apresentando um circuito específico para as chaves de cima da ponte H.

Chavear uma corrente elevada com uma ponte H exige um cuidado especial para evitar que ocorra um curto circuito em um dos braços da ponte, o que ocorreria se Q1 e Q2, na Figura 2.5, fossem acionados simultaneamente. Pode ser utilizado então um circuito lógico em cada um dos braços da ponte para evitar o estado de curto circuito (STMICROELECTRONICS, 2000). O circuito apresentado na folha de dados do *driver* utilizado no projeto está reproduzido na Figura 2.6.

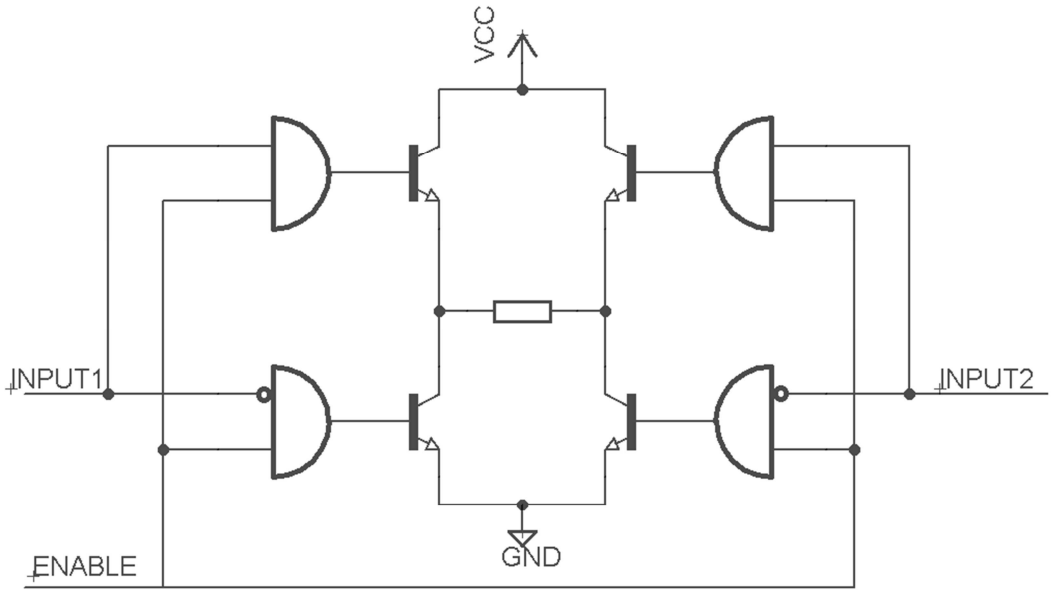


Figura 2.6. Circuito lógico para prevenção de curto circuito em uma ponte H.

Nesse caso, o controle de cada lado da ponte funciona de forma binária, não sendo permitido o curto circuito, além de possuir liberdade para acionar os dois transistores inferiores ou superiores simultaneamente, o que pode ser desejado quando se deseja liberar o fluxo de corrente armazenado nas bobinas de um motor.

Outra técnica utilizada é a ligação de diodos de retorno em paralelo com os transistores e com polarização reversa, para que uma corrente induzida no motor possa fluir mesmo quando ambos os lados da ponte estiverem no estado desligado, ou seja, bloqueando a passagem de corrente. A utilização desses diodos também evita que uma corrente recirculante no motor danifique um transistor que está “cortado” (BARBI, 2006).

Na Figura 2.7. Ponte H com diodos de retorno., pode-se ver a ponte com os diodos de retorno. Quando os transistores Q1 e Q4 estão acionados, a corrente flui de A para B, pois o potencial em A é maior; quando ambos são cortados, ainda existe uma corrente armazenada na carga; essa corrente faz com que o potencial em A seja menor do que zero e em B maior do que *VCC*, fazendo os diodos D2 e D3 conduzirem por um curto período de tempo suficiente para dissipar essa corrente residual, pois após a corrente ser dissipada a tensão de A volta a ser maior do que terra e B menor do que *VCC*.

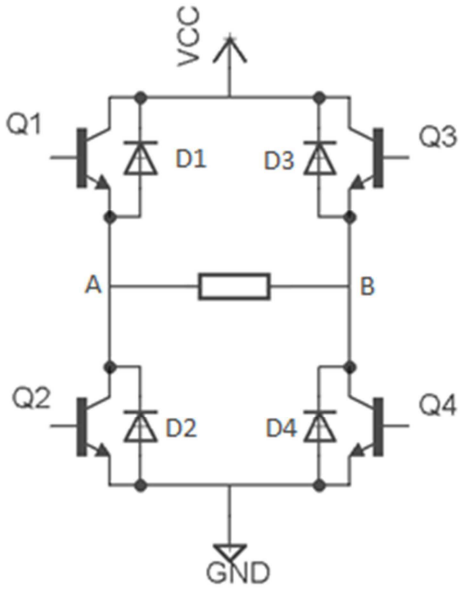


Figura 2.7. Ponte H com diodos de retorno.

2.6. Rede neural artificial do tipo RAM

Redes Neurais Artificiais são técnicas computacionais que apresentam estruturas inspiradas na estrutura neural de organismos para obter conhecimento através de aprendizado e tomar decisões baseando-se em conexões efetuadas. Na biologia, os neurônios se comunicam através de sinapses, transmitindo impulsos nervosos. Um neurônio pode disparar ou não um impulso a outro neurônio dependendo de suas entradas e da estrutura de conexão entre esses neurônios (SILVA, SPATTI e FLAUZINO, 2010).

Em uma rede neural artificial, um neurônio é uma estrutura simples que dispara um sinal dado um determinado somatório na entrada, e são conectados entre si com determinados pesos, que devem ser ajustados para que determinada entrada corresponda a uma saída esperada. O comportamento inteligente vem dessas interações entre as unidades de processamento (SIMA, 1998). A operação de cada unidade de processamento (neurônio) é basicamente composta da apresentação de sinais à entrada, multiplicação de cada um por pesos, indicando sua influência na saída, soma dos resultados e comparação da soma com um limiar, produzindo determinada resposta na saída.

Na maioria dos modelos de redes neurais, os pesos de suas conexões são ajustados de acordo com os padrões apresentados, aprendendo através de exemplos. Esse processo de aprendizagem é feito de forma iterativa e, por exigir que os pesos sejam ajustados conforme o erro obtido na saída, normalmente exige grande poder de processamento e envolvem o cálculo de expressões com ponto flutuante, o que também é requisitado no teste da rede, o que usa recursos de processamento não desejados em um ambiente embarcado.

Para sanar essa alta requisição de recurso de hardware, foi utilizada uma rede neural sem peso baseada em células de memória. Observadas como forma de um sistema fechado, as configurações com e sem peso apresentam comportamento parecido. A rede neural do tipo *RAM* ou *n-tuple classifier* possui treinamento rápido (*one-shot learning*), e o processamento a partir da modificação de uma entrada até o cálculo da saída depende de poucas operações de endereçamento, soma e comparação (AUSTIN, 1998).

O neurônio da rede neural *RAM* é uma célula de memória, que possui n entradas binárias que são responsáveis pelo endereçamento da célula, e a saída dessa célula é acumulada com outras saídas que representam a mesma classe. Na Figura 2.8 está reproduzido um exemplo de uma rede neural do tipo RAM onde os neurônios, que são endereçáveis de forma binária, são representados como círculos azuis. Assim, a classe com maior soma (*winner takes all*) é escolhida como a saída da rede (UEBEL, BOTELHO, *et al.*, 1995). O aprendizado é rápido e o conjunto de treinamento precisa ser apresentado à rede somente uma vez, pois uma vez conhecida a entrada, o endereçamento da classe correspondente pode ser efetuado, e os conteúdos de seus neurônios são alterados para que um novo padrão possa ser aprendido.

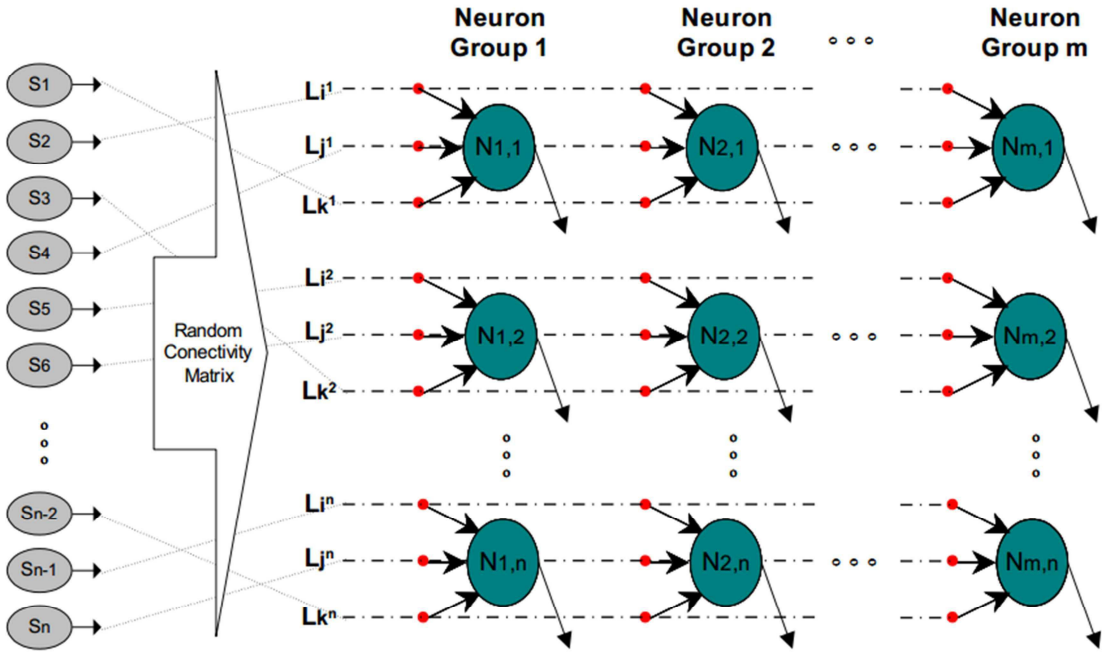


Figura 2.8. Exemplo de rede neural do tipo RAM contendo neurônios de 3 entradas e *m* classes com *n* neurônios cada, e conectados aleatoriamente a entradas binárias (SIMÕES, 2000).

2.7. Algoritmos evolutivos

Algoritmos Evolutivos são baseados em mecanismos inspirados na evolução biológica das espécies (HOLLAND e GOLDBERG, 1988). A motivação para a utilização dessas técnicas computacionais surgiu da observação da natureza, que resolveu problemas complexos utilizando

tais recursos, ou seja, tenta-se construir de maneira computacional situações que possam se comparar com mecanismos que conhecidamente influenciam na evolução. Uma comparação entre as técnicas computacionais e a evolução ocorrida na natureza é mais explorada em (EIBEN e SMITH, 2003).

É possível então aplicar técnicas evolutivas a problemas de otimização, mesmo quando o modelo do problema não possa ser definido completamente, bastando que seja possível avaliar uma resposta possível.

As possíveis soluções são então tratadas como indivíduos, e seus conjuntos como populações, que interagem buscando a evolução através de operadores genéticos. A literatura clássica (GOLDBERG, 1989), trás o mecanismo padrão de evolução dividido nas etapas representadas no fluxograma presente na Figura 2.9.

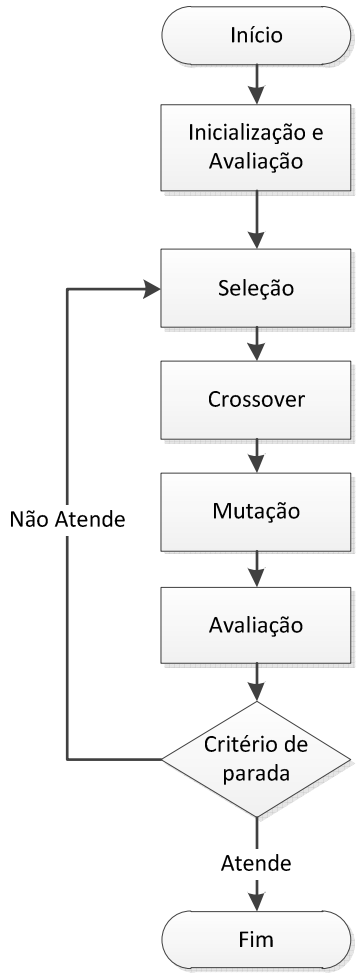


Figura 2.9. Fluxograma básico de um algoritmo evolutivo.

Porém, em robôs reais, a função *fitness* possui um ruído, pois a pontuação obtida na avaliação depende da leitura de sensores e pode sofrer interferência de agentes externos, como escorregamento e irregularidade de terreno. Com essas variações não é possível garantir que o indivíduo avaliado possuirá sempre o mesmo *fitness*, sendo necessário continuar a evolução para

cada nova situação encontrada pelo robô. Dessa maneira, o controle reativo a ser implementado nesse projeto deverá ser capaz de se adaptar a variações do ambiente, como por exemplo a quantidade de obstáculos.

3. Implementação do sistema de controle de navegação

O hardware utilizado nesse projeto possui alguns módulos que permitem o controle e sensoriamento do robô, além de comunicação sem fio. O bom funcionamento desses módulos é fundamental para que um algoritmo de navegação autônoma possa ser testado nesse hardware, evitando que problemas indiretos interfiram nos resultados dos testes.

3.1. Sonares

No módulo de sensoriamento, foi necessário adaptar a tensão de operação dos sonares com a tensão de operação da *Raspberry Pi*. Sabendo que o microprocessador opera com 3,3V e os sonares com 5V, primeiramente foi verificado na folha de dados do módulo utilizado a tensão mínima para que um pulso de *trigger* pudesse ser compreendido como nível alto. Após a verificação teórica onde foi constatado que sinais maiores que 3V seriam suficientes, foi feito um ensaio que comprovou o funcionamento do disparo do sonar com um pulso de 3,3V.

A resposta do sonar, chamada de eco possui, entretanto, um pulso de 5V, o que pode danificar as entradas digitais da Raspberry se conectado diretamente. Foi feito então um divisor resistivo simples, visto que as entradas digitais são de alta impedância e, portanto não drenam um valor de corrente significativo, o que poderia alterar o valor de tensão lido. Com o circuito apresentado na Figura 3.1 os pulsos foram conformados e pode-se implementar um algoritmo para aferição de distância.

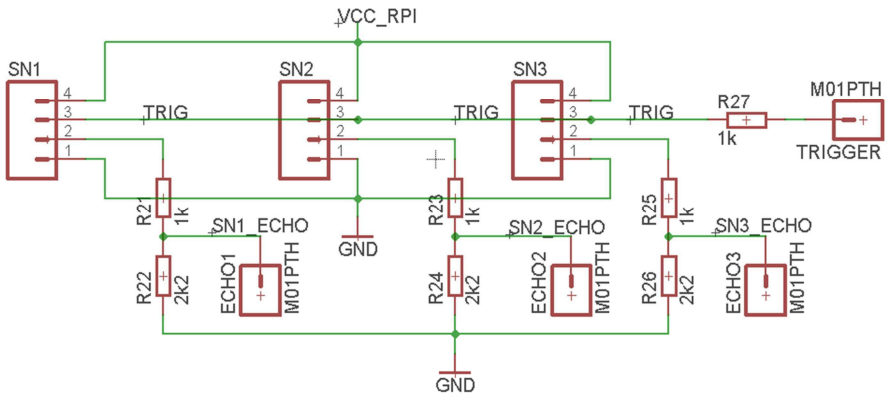


Figura 3.1. Esquemático do circuito para conversão de tensão dos pulsos do sonar (Conectores SN1-3 para ligar os sonares e pinos ECHO1-3 e TRIGGER para conectar a *Raspberry Pi*).

O funcionamento especificado para o módulo HC-SR04, utilizado nesse projeto, consiste em manter um pulso de pelo menos 10 microssegundos no pino de *trigger* do sonar e, após o pulso ser levado para nível baixo, 8 pulsos de ultrassom com frequência de 40 kHz são emitidos pelo sensor. Após a recepção dos pulsos refletidos, a saída ECHO do módulo responde com um pulso de duração igual ao tempo decorrido entre a emissão e a recepção do pulso ultrassônico, bastando então contar esse tempo e multiplicá-lo pela velocidade do som para obter a distância

total percorrida pelo sinal ultrassônico, ou seja, é obtido o dobro da distância entre o objeto e o sensor, portanto, a distância entre obstáculo e robô é a metade da distância percorrida pelo sinal. A Figura 3.2 apresenta um diagrama de tempo do funcionamento do sonar.

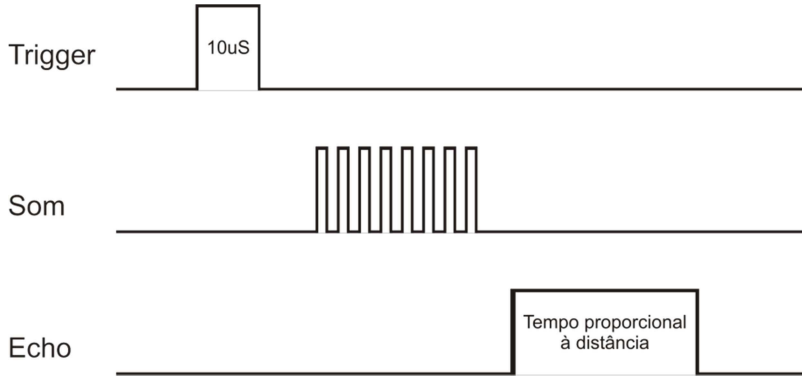


Figura 3.2. Diagrama de tempo apresentando o funcionamento do sonar.

3.2. Acionamento dos motores

Para efetuar o acionamento foi utilizado o circuito integrado L298 que possui duas pontes que podem ser controladas de maneira independente. Esse *CI* apresenta internamente a configuração de ponte H, muito utilizada para controle de motores *DC* e de passo. A corrente máxima fornecida por esse *driver* é de 2A por canal, o que é suficiente para acionar os motores contidos no chassi utilizado.

O *CI* utilizado recebe como entrada dois pulsos por ponte H, um sinal para cada lado da ponte, permitindo assim que os dois transistores inferiores da ponte fiquem habilitados, possibilitando uma recirculação de corrente no caso de uma inversão. Essa funcionalidade não foi explorada nesse projeto e foram adicionados diodos de roda livre, para que uma corrente reversa não cause danos ao *driver*. Foi utilizado apenas um sinal de controle de direção binário por motor, para isso, o sinal de um lado da ponte vem diretamente do opto-acoplador e outro sinal é obtido com a inversão lógica utilizando um transistor de junção bipolar em coletor aberto. Dessa maneira, o nível lógico baixo faz o motor girar em um sentido e o nível lógico alto no sentido inverso.

A velocidade é controlada através da entrada do CI L298 chamada *Enable* que permite desabilitar a saída de uma ponte H, de maneira independente. Assim, aplicando um sinal *PWM* nessa porta, pode-se alterar a potência entregue a cada motor independentemente.

3.3. WiringPi

O acionamento de pinos do processador quando é utilizado um sistema operacional de uso geral não é direto como em microcontroladores. Para facilitar essa interface entre software e hardware foi utilizada uma biblioteca de uso gratuito criada por (HENDERSON, 2013), chamada de *WiringPi*. O uso dessa biblioteca tornou a programação mais natural e clara, uma vez que a configuração dos registradores com função especial do microprocessador fica abstrato.

3.4. Sistema de navegação

Portando todas as ferramentas descritas anteriormente, foi possível desenvolver um sistema de navegação inteligente, com funcionalidades como adaptação em relação à mudanças na disposição de obstáculos em relação ao robô baseando-se em padrões pré-configurados e a melhora das soluções utilizando um algoritmo evolutivo.

A Figura 3.3 apresenta um diagrama de blocos do sistema de navegação implementado, que caracteriza um controle reativo como descrito em (WOLF, SIMÕES, *et al.*, 2009), por sua estrutura de leitura de sensores, processamento imediato dos dados e geração de comando para os atuadores.

Uma rede neural do tipo *RAM* será responsável por acionar o módulo de controle dos motores durante o funcionamento normal do robô e essa rede neural será avaliada periodicamente por um algoritmo supervisor capaz de detectar e reagir a colisões, pontuando a configuração programada na rede neural. A pontuação obtida pela configuração será utilizada por um algoritmo evolutivo, que configura de novas formas com novos indivíduos a rede neural, buscando as melhores decisões para determinada leitura dos sensores. Os itens serão expostos independentemente a seguir.

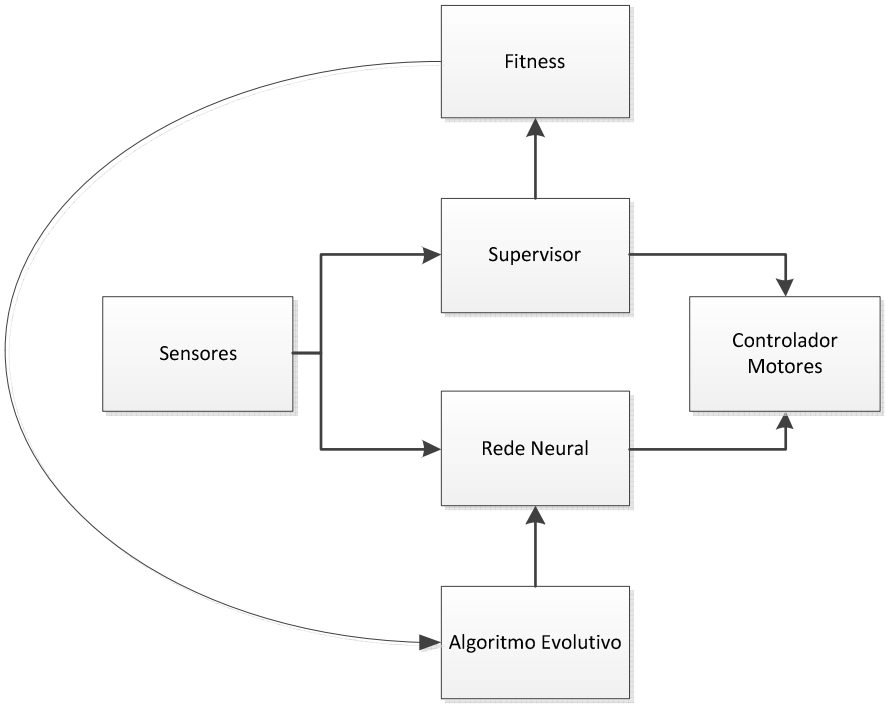


Figura 3.3. Diagrama do sistema de navegação autônoma.

3.4.1. Sensores

Foram utilizados nesse projeto três sonares que respondem a distância, em centímetros, a que o obstáculo se encontra do sensor. Porém, para utilização da rede neural desejada, que possui entradas binárias, foi realizada uma discretização das distâncias lidas pelos sonares.

Essa discretização foi feita em quatro regiões, resultando em doze regiões de observação para serem tratadas de forma binária pela rede neural. A Figura 3.4 ilustra a divisão adotada. Assim, cada região responde sobre a presença ou não de obstáculos, sendo que, quando há um obstáculo na região 0, por exemplo, faz com que as regiões que representam distâncias maiores respondam afirmativamente em relação à presença de obstáculos.

Foi convencionado que a presença de obstáculo na região observada é representada por 1, e a ausência por 0.

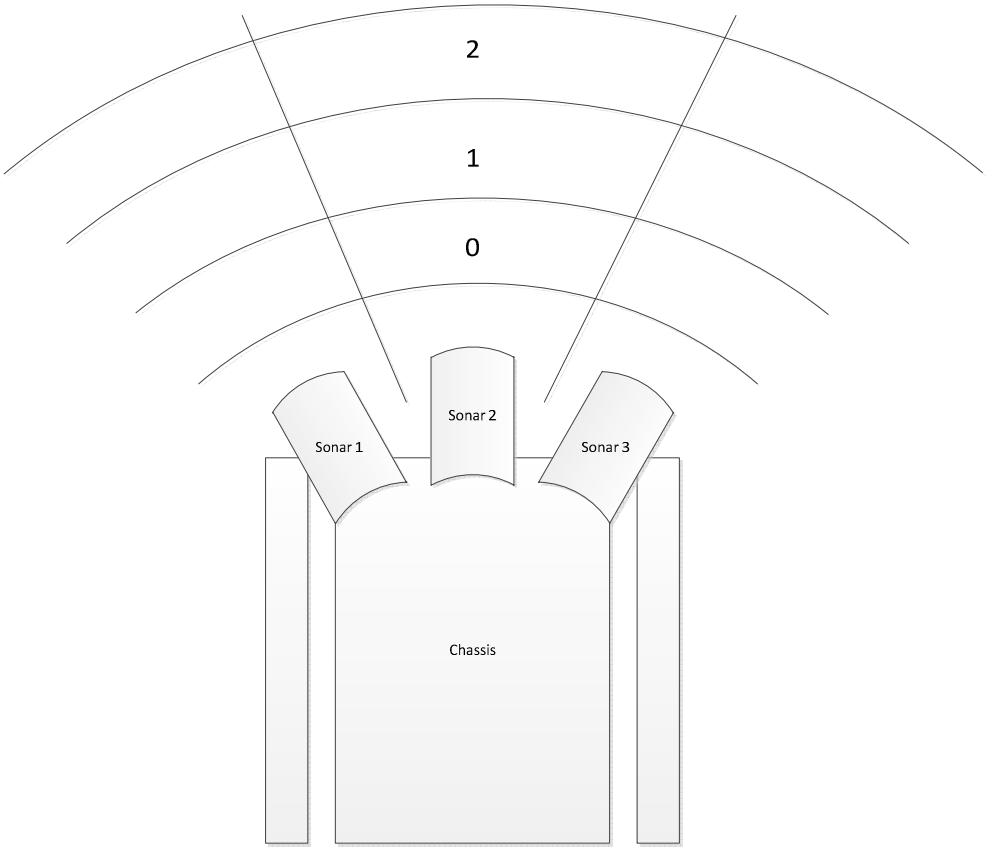


Figura 3.4. Discretização das distâncias dos obstáculos.

3.4.2. A rede neural

É utilizada nesse projeto uma rede neural baseada em uma memória *RAM*, que possui seus neurônios com formato de vetores simples. Uma entrada com um número predefinido de bits endereça uma posição do vetor, como em uma célula de memória, que deve retornar zero ou um. Esta resposta é somada com outras dos demais neurônios da mesma classe, sendo que cada classe é uma possível resposta da rede neural, e a que obtiver a maior soma é a que possui a resposta que corresponde com o treinamento.

O treinamento baseado na informação de um especialista consiste em endereçar os neurônios e gravar o algarismo 1 na posição daqueles que forem da classe conhecida. A Figura 3.5 exemplifica o endereçamento para o caso no qual as regiões dos sensores conectadas nos neurônios da linha em questão são 0 para o primeiro, a mais próxima do robô, 3 para o segundo, e 1 para o terceiro. Observa-se que há um obstáculo na região 1 do terceiro sensor, assim, a entrada para as células de memória para essa linha específica será o número binário 001, ou simplesmente a segunda posição do vetor. Considerando que a rede estava zerada, sem nenhum treinamento, e a ação desejada para esse caso é “virar para esquerda”, tem-se o valor 1 gravado no endereço obtido na classe “esquerda”. As classes adotadas a princípio foram respectivas à movimentação do robô para frente, para direita e para esquerda, sendo, a princípio, três classes.

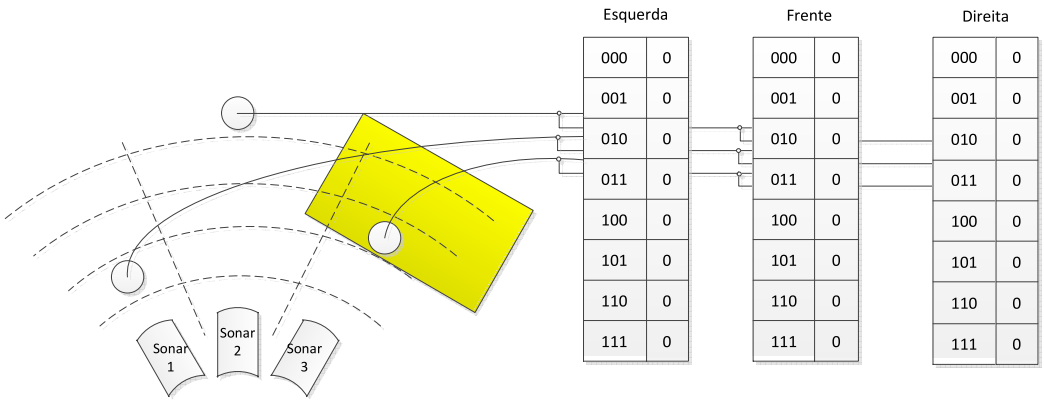


Figura 3.5. Exemplo de endereçamento e treinamento da rede neural. As entradas que endereçam os neurônios são os bits relativos à presença (1) ou não (0) de obstáculos nas regiões observadas (marcadas com círculos na figura).

Tendo em vista que é desejável que todas as doze regiões de observação sejam utilizadas para endereçar pelo menos uma linha de neurônios, é feito no início do programa um sorteio, sendo que cada região pode ser utilizada apenas uma vez. Dessa maneira, existirão quatro neurônios por classe, utilizando todas as regiões de observação.

Para efetuar uma decisão baseada nessa rede neural, é feito o endereçamento dos neurônios presentes em cada classe, tendo suas respostas somadas. Como no treinamento os neurônios que são acionados são gravados com 1, a região na qual apresenta maior soma é a escolhida como resposta mais provável.

3.4.3. Algoritmo supervisor

Um algoritmo supervisor é executado após cada leitura do sensor, sendo responsável por reagir a situações em que a rede neural não conseguiu evitar os obstáculos, levando o robô a colidir. Essa parte do programa também é encarregada de incrementar ou decrementar a pontuação da configuração atual da rede neural, de acordo com a Tabela 3.1. Essa avaliação é utilizada como *fitness* do algoritmo evolutivo.

A entrada do algoritmo supervisor será a leitura dos sensores, e se a distância for menor que o limiar estipulado, o que foi caracterizado como colisão, o robô é reposicionado rapidamente com instruções predefinidas e a pontuação do robô é decrementada de um valor fixo. Após esse reposicionamento, a rede neural é acionada novamente, para continuar sendo avaliada.

Caso a distância for maior que o limiar, a pontuação do robô é acrescida e o controle é passado novamente para a rede neural. Esse ciclo deve permanecer por um tempo predefinido, de maneira que todas as configurações sejam avaliadas por um tempo predeterminado, garantindo que uma pontuação maior represente realmente um indivíduo melhor.

Tabela 3.1. Pontuação atribuída ao robô com relação à sua movimentação.

Decisão tomada	Pontuação
Frente	+2
Direita	+1
Esquerda	+1
Colisão / Reposicionamento	-10

3.4.4. Algoritmo evolutivo

A cada população avaliada o algoritmo evolutivo é acionado, realizando as operações de seleção, *crossover* e mutação. O indivíduo do algoritmo foi estabelecido como sendo o vetor com as saídas de todos os neurônios da rede neural. Como a rede é composta por três classes com quatro neurônios, e cada um possui oito saídas possíveis, tem-se um total de 96 bits compondo cada indivíduo. Portanto, existem 2^{96} possíveis indivíduos, tornando aplicável um algoritmo evolutivo para encontrar a melhor configuração para o robô.

O algoritmo evolutivo foi desenvolvido com foco em manter o número de aplicações da função *fitness* baixo, pois uma avaliação de um indivíduo no mundo real leva ao menos 30 segundos. O tempo de avaliação ideal é maior, porém o tempo máximo que viabiliza os testes com o robô foi o de meio minuto.

Para validar o algoritmo e verificar se sua aplicação no robô seria possível, foi utilizada uma função *fitness* alternativa, comparando cada entrada possível nos sensores com uma saída considerada ideal, e adicionando um ponto ao indivíduo caso sua saída fosse igual à ideal. Assim, cada teste do algoritmo evolutivo leva apenas alguns segundos, então as taxas utilizadas no algoritmo e possíveis falhas podem ser ajustadas e corrigidas rapidamente. O fluxograma dessa simulação está representado na Figura 3.6.

Devido também ao longo tempo que dura o teste de cada indivíduo, uma tabela registrando o indivíduo e sua avaliação é mantida, evitando que uma mesma configuração seja testada mais que uma vez em um curto período de tempo.

Foi utilizado um algoritmo evolutivo baseado em um algoritmo genético, com as etapas de seleção, *crossover* e mutação. Para efetuar uma busca refinada do resultado foi implementada uma mutação variável dependente da velocidade com a qual o algoritmo progride, ou seja, quantas gerações se passam até que o melhor indivíduo mude. A Figura 4.8 apresenta o

planejamento inicial do algoritmo de navegação, note que após a avaliação de todos os indivíduos da população, as funções pertinentes ao algoritmo genético são atingidas.

A mutação variável é útil para buscar soluções em todo o espaço de busca e refinar o resultado quando se aproximando de um máximo local, com uma taxa fixa de mutação, o *fitness* pode não atingir o máximo se ela for muito alta, ou pode-se ficar preso em um máximo local e não percorrer todas as soluções se a taxa for baixa.

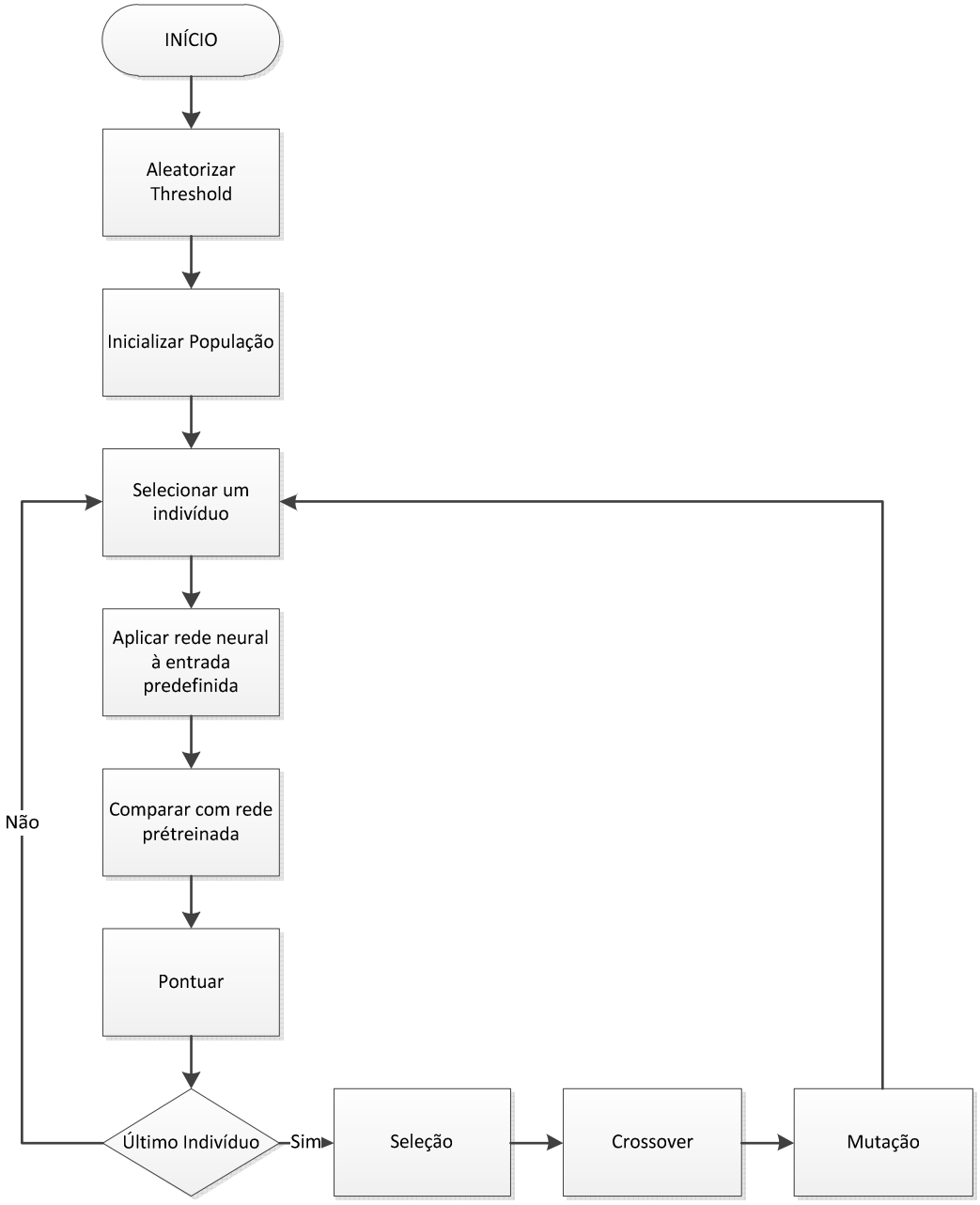


Figura 3.6. Fluxograma da simulação computacional do algoritmo genético com evolução contínua.

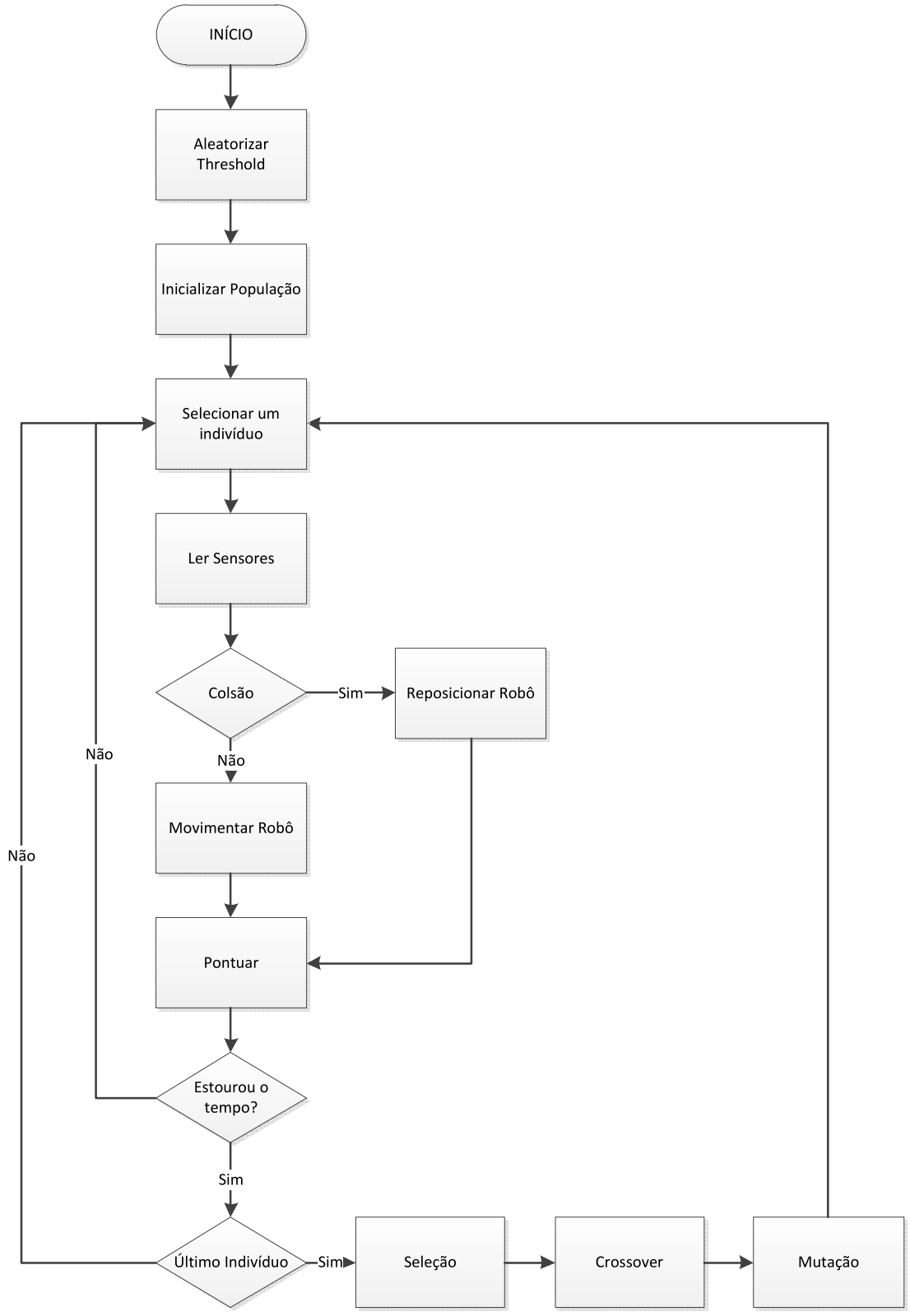


Figura 3.7. Fluxograma do sistema de navegação com evolução contínua, buscando sempre a adaptação à novas condições do ambiente.

4. Resultados

4.1. Experimentos

Uma placa unificando o condicionamento dos sinais dos sonares, o controle do robô e o *driver*, foi construída e está reproduzida na.Figura 4.1. Os resultados dos módulos estão expostos a seguir.

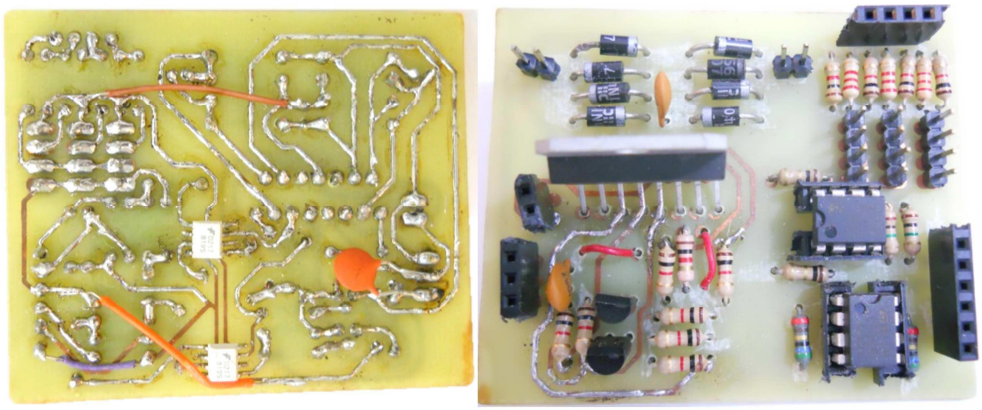


Figura 4.1. Placa construída contendo o *driver* e o circuito de condicionamento dos sonares. (Vista inferior à esquerda e superior à direita).

4.1.1. Sonares

Os primeiros testes foram feitos com o conjunto software e hardware para avaliar a qualidade dos sensores de distância baseados em som.

A *Raspberry Pi*, por ter em seu projeto como objetivo primordial o baixo custo de construção, não possui um *RTC*, sigla que vem do inglês para relógio de tempo real, por isso, a referência de tempo depende exclusivamente de uma contagem do processador. Porém, com um sistema operacional de uso geral e com o escalonamento de processos, um simples contador para determinar o tempo transcorrido pode apresentar uma variação muito dependente da disponibilidade da CPU. Foi feito um ensaio para comprovar esse efeito na leitura dos sonares, no qual uma distância foi fixada e a contagem foi feita periodicamente, o resultado pode ser observado no gráfico abaixo. Para um funcionamento ideal do sonar, é desejável que a leitura permaneça constante, porém, o resultado variou para até 50% do valor máximo.

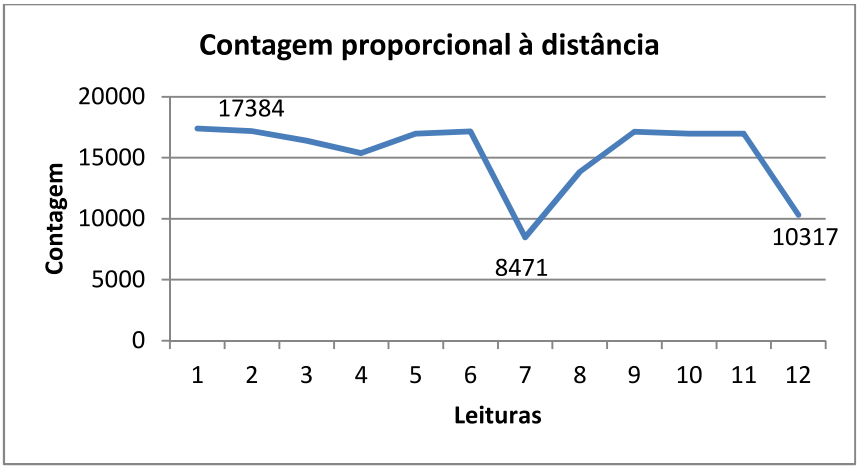


Figura 4.2. Gráfico das contagens obtidas para uma distância fixa.

Para solucionar este problema foi utilizado um módulo do kernel que mantém um relógio funcionando através de interrupções no processador, o que aumenta a precisão da contagem de tempo. O tempo do SO pode ser acessado através da função *gettimeofday*, e com a utilização da biblioteca *time.h*, foram obtidas as leituras da Tabela 4.1 e da Figura 4.3. O código do programa implementado para esse teste pode ser encontrado no Apêndice A.

Tabela 4.1. Comparação da distância real com a obtida pelo sonar.

Real	Lida
100	97,7
90	87,9
80	77,7
70	68,1
60	59,1
50	48,8
40	39,6
30	30,7
20	19,4
10	9,7

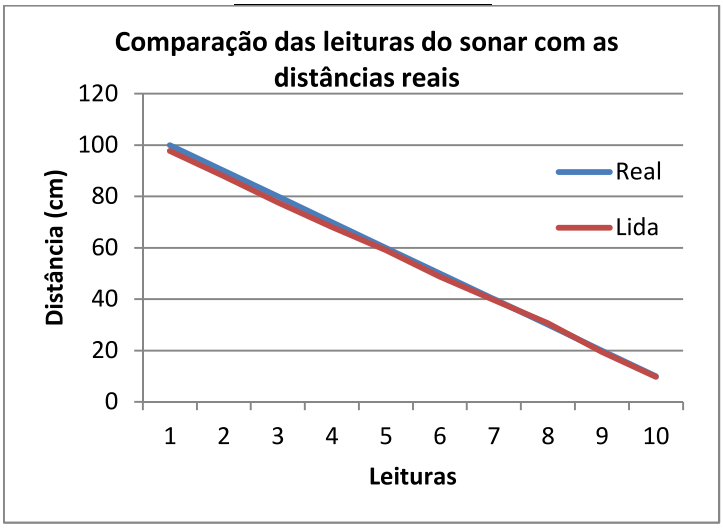


Figura 4.3. Gráfico da distância obtida nas leituras para efeito de validação das leituras obtidas com o sonar.

4.1.2. Controle de velocidade e direção dos motores

Foi necessário gerar dois pulsos *PWM* e dois digitais para controlar as velocidades e direções dos motores, respectivamente. Os pulsos digitais foram gerados utilizando as portas de entrada e saída de uso geral, que são disponibilizadas na *Raspberry Pi*. O acionamento em software dessas portas foi feito utilizando a biblioteca *Wiring Pi*, já apresentada.

O *ARM* presente na placa utilizada possui duas saídas que podem gerar sinais *PWM* de maneira independente da *CPU*. Esses dois periféricos podem ser conectados a diversos pinos do chip através da configuração dos registradores responsáveis pelas funções alternativas, porém apenas um desses foi conectado pelo fabricante diretamente com o barramento disponível para o usuário. Portanto, em um primeiro teste foi utilizado o pino de *PWM* dedicado para controlar a velocidade de um motor e o outro pulso modulado foi criado em software, exigindo algum processamento da *CPU* quando o estado da saída fosse alterado.

Assim, se uma frequência mais alta de *PWM* for utilizada, mais vezes o *thread* criado para gerenciar a criação do pulso é acionado. Uma captura do programa *top* com o programa que gera os pulsos em software em execução foi feita e está reproduzida na Figura 4.4, nota-se que a utilização do processador pelo processo *contrRoboSoft*, que é o processo que está gerando o pulso, é de 12,1%, e essa utilização é constante e para apenas um pino.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2450	root	20	0	10072	624	528	S	12.1	0.2	0:02.26	contrRoboSoft
2383	root	20	0	4880	1300	976	R	1.0	0.3	0:05.33	top
2397	root	20	0	0	0	0	S	0.7	0.0	0:02.90	kworker/u:1
6	root	20	0	0	0	0	S	0.3	0.0	0:06.57	kworker/u:0
2013	nobody	20	0	2012	636	516	S	0.3	0.2	0:00.02	thd
1	root	20	0	2140	712	608	S	0.0	0.2	0:01.63	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
7	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/u:0H

Figura 4.4. Captura do programa TOP apresentando o uso do processador.

Como uma tentativa de obter um resultado melhor foi feito um estudo no esquemático da placa *Raspberry Pi*. Pôde-se notar que os GPIO 40 e 45 do *ARM* estão conectados com a saída de áudio, então foi feita uma tentativa de utilizar essa saída através de um conector P2, o mesmo encontrado em fones de ouvido. Porém, as ligações entre os pinos 40 e 45 e a saída de áudio possuem filtros para a faixa de áudio, como pode ser observado na Figura 4.5.

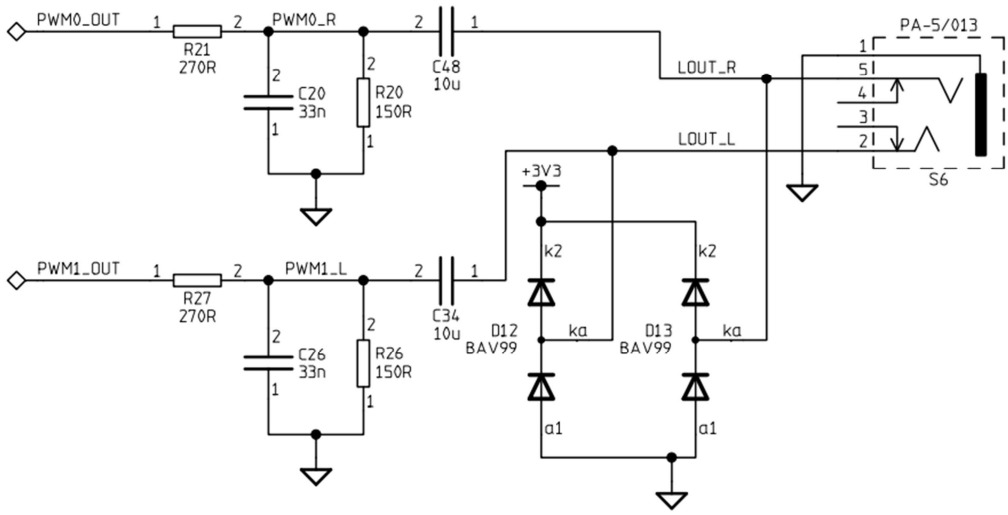


Figura 4.5. Esquemático da saída de áudio da *Raspberry Pi* (PBL, 2012).

O capacitor de acoplamento, no caso, faz com que o nível DC do sinal seja levado a zero, não sendo possível manter um sinal PWM variando entre 0V e 3,3V, ou seja, nível lógico baixo e alto. Foi necessário então realizar uma modificação na placa, para desabilitar os capacitores do filtro passa-alta, C34 e C48, no esquemático da Figura 4.5 e no detalhe da Figura 4.6. Para evitar maiores modificações, a frequência do PWM foi mantida na faixa de passagem do filtro passa-baixa presente na montagem original da placa, mas, caso fosse necessário uma frequência maior, bastaria efetuar a remoção do filtro passa-baixa presente na placa. A tensão obtida da onda quadrada gerada foi uma variação entre +1V e 0V, o que é suficiente para acionar o foto-transistor do opto-acoplador, que está polarizado na região de saturação.

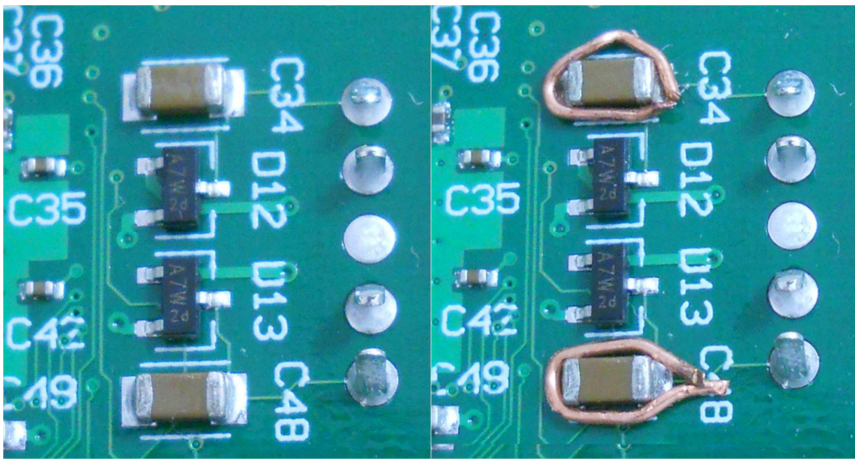


Figura 4.6. Capacitor de acoplamento presente na saída de áudio da *Raspberry Pi* (Placa original à esquerda e adaptação feita na imagem à direita).

A captura do programa TOP reproduzida na Figura 4.7 foi feita na mesma situação em que a captura da Figura 4.4, com a diferença da geração do PWM exclusivamente por Hardware. Nota-se que o processo *contrRobo*, que é responsável por configurar o módulo gerador de PWM não utiliza de maneira significativa o processador, ficando clara a vantagem em se utilizar o módulo do *Timer* dedicado à geração de pulsos.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5574	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:1H
8940	root	20	0	0	0	0	S	0.0	0.0	0:05.40	kworker/u:2
8959	root	20	0	0	0	0	S	0.0	0.0	0:00.26	kworker/0:0
8962	root	20	0	0	0	0	S	0.0	0.0	0:00.00	flush-179:0
8964	root	20	0	5360	1600	1284	S	0.0	0.4	0:00.04	sudo
8965	root	20	0	1880	592	504	S	0.0	0.2	0:00.01	contrRobo
8966	root	20	0	0	0	0	S	0.0	0.0	0:00.22	kworker/0:1

Figura 4.7. Captura do programa top. Destaque para o processo contrRobo utilizando 0.0% da CPU.

4.1.3. Simulação do algoritmo evolutivo

Antes de programar o robô e testar o algoritmo evolutivo no mundo real, foi feito um teste computacional comparando uma saída considerada ideal com a resposta de cada indivíduo.

Assim, foram corrigidas rapidamente diversas falhas que faziam o algoritmo perder eficiência, como, por exemplo, a situação que pode ser observada na Figura 4.8, onde por aproximada 300 gerações o *fitness* médio é igual ao melhor, ou seja, todos os indivíduos da população são iguais e a população não evolui, esse fator é causado pelo valor da mutação que foi levado a zero pela função que a ajusta. Esse problema foi corrigido aumentando a mutação quando a mesma atinge um valor nulo, assim o espaço de testes é abrido novamente.

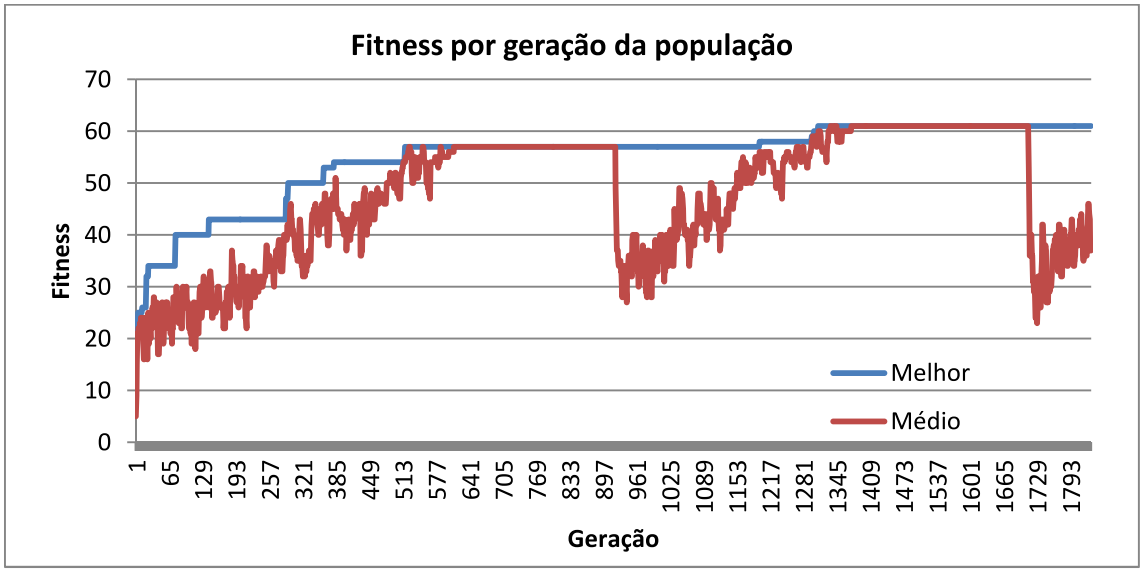


Figura 4.8. Gráfico do *fitness* do melhor indivíduo e da média dos *fitness* da população.

A Figura 4.9 apresenta o resultado do algoritmo com a correção. Como se trata de uma simulação, o valor máximo do *fitness* é conhecido, e foi atingido por volta da geração 900, mas o comportamento da população mantém o mesmo ciclo, onde o valor médio diminui quando a taxa de mutação aumenta e sobe gradativamente com a diminuição dessa taxa repetidamente. A utilização dessa busca constante visa manter a evolução da população em ambientes dinâmicos, onde o *fitness* pode mudar e uma adaptação da população pode ser necessária.

Parte do software desenvolvido está reproduzido no Apêndice B. O algoritmo evolutivo foi desenvolvido de forma modular, assim, as mesmas funções utilizadas na simulação da evolução foi a utilizada no sistema de navegação, além de ser possível substituir facilmente os mecanismos evolutivos.

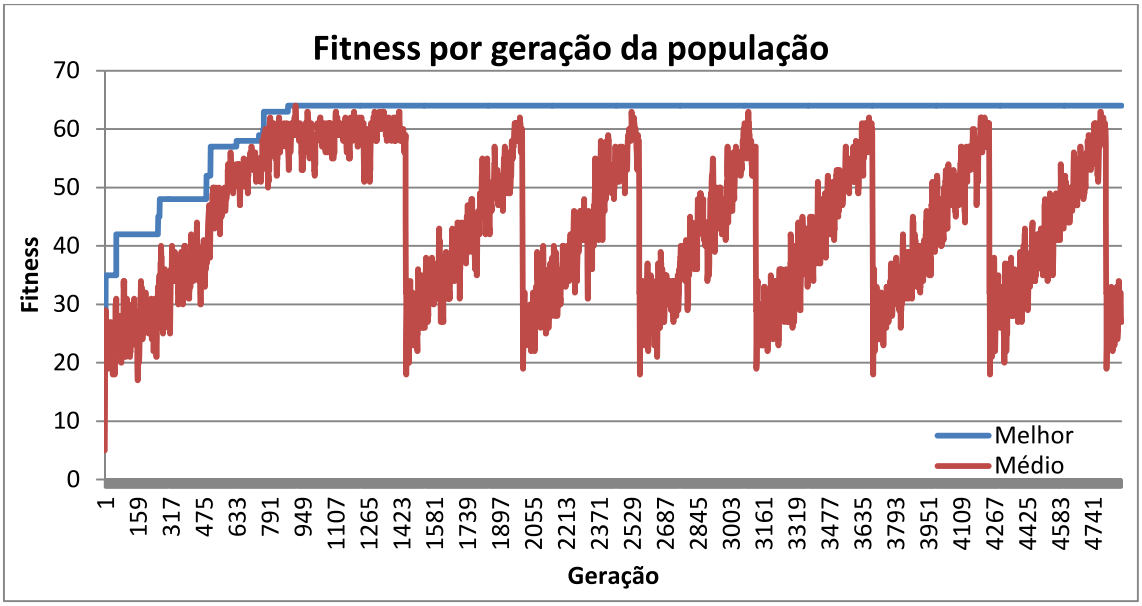


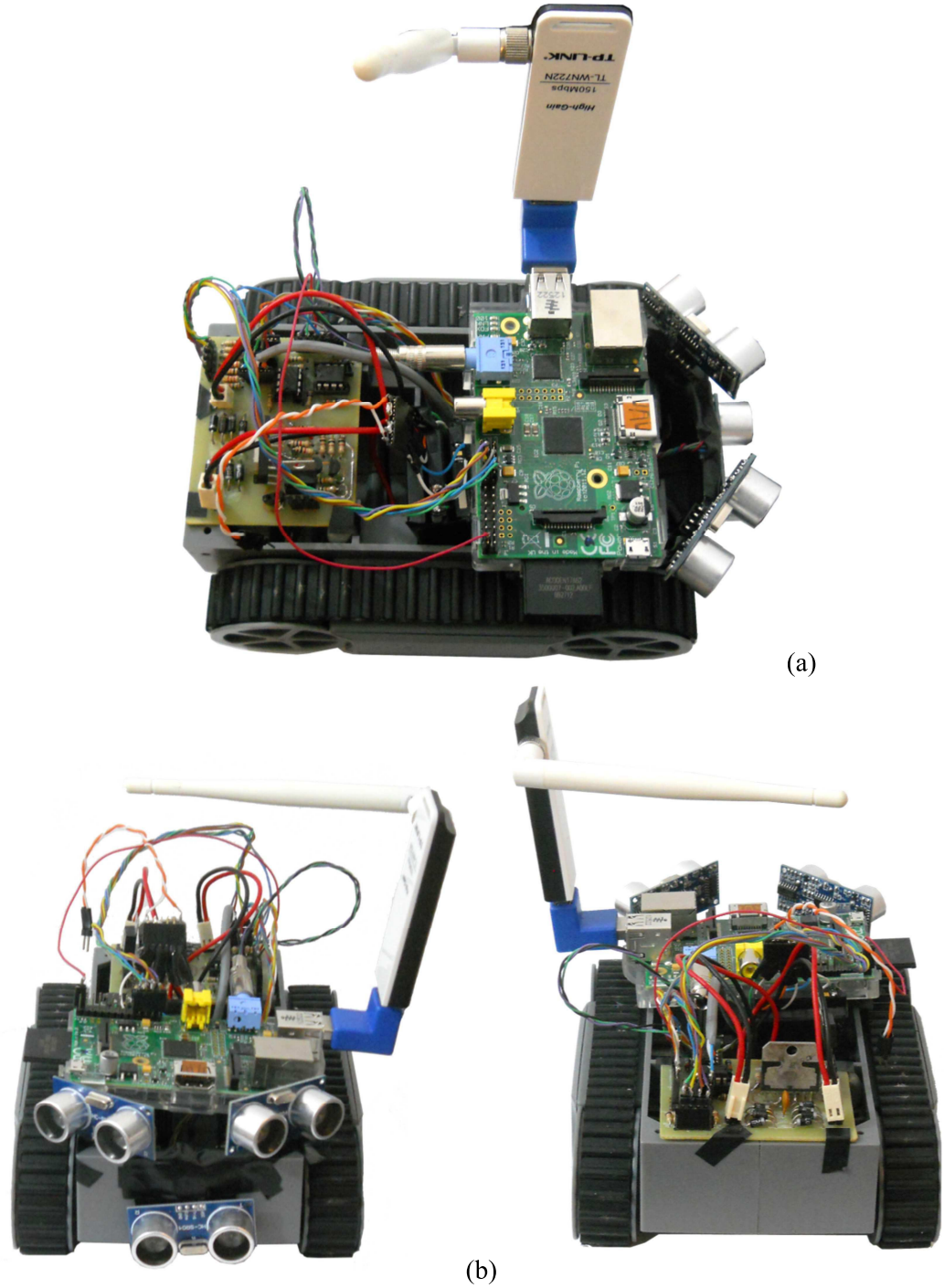
Figura 4.9. Resultado da simulação do algoritmo evolutivo.

4.1.4. Teste do sistema de navegação

O robô pronto para ser testado pode ser observado na Figura 4.10. O primeiro teste consistiu em suspender o robô e simular algumas situações que podem ser encontradas no mundo real, colocando obstáculos a distâncias fixadas.

Observando o comportamento do algoritmo, pôde-se notar que em muitas situações o robô acionava a rotina para reposicionar após colisão erroneamente, o que estava sendo causado por leituras equivocadas dos sonares. Após uma análise detalhada do procedimento de acionamento e comparação com o software utilizado para a validação do sonar, concluiu-se que um tempo de espera entre as leituras é necessário. Devido à reflexão dos pulsos nos obstáculos, é possível saber a distância a que se encontram. Porém uma leitura precoce após outra pode apresentar erros causados por ecos, ou seja, reflexões indiretas dos pulsos. Adicionando um *delay* de 50 milissegundos antes de cada leitura, obteve-se um resultado com menos leituras erradas.

Outra medida tomada a fim de minimizar a detecção incorreta de colisão foi a redundância na leitura dos sonares. Assim, quando é detectada uma colisão, a leitura é feita novamente. Essa redundância não foi adotada em situações de movimentação livre, pois uma variação pequena dura apenas um ciclo de avaliação, não trazendo prejuízos ao *fitness* final atribuído ao indivíduo.



(a) (b) (c)

Figura 4.10. Robô com os módulos conectados pronto para ser testado (a: vista lateral, b: vista frontal, c: vista traseira).

Para validar o mecanismo de avaliação utilizado pelo AE, o robô foi colocado em um espaço fechado com dois obstáculos distribuídos (Figura 4.11) e uma configuração para a rede neural (um indivíduo) foi avaliada. O teste consistiu em executar a função de *fitness* sucessivamente no mesmo indivíduo, zerando sua pontuação a cada 30 segundos. Com esse experimento, desejava-se a mesma pontuação em todos os intervalos de tempo, comprovando a estabilidade do mecanismo de avaliação, porém o comportamento apresentado no gráfico da

Figura 4.12 foi obtido a princípio. Nesse gráfico pode ser observada uma grande variação na pontuação atribuída.

Para se obter um resultado mais estável foram feitos alguns ajustes: mudança dos limiares utilizados para a discretização das distâncias, diminuição do tempo utilizado para reposicionar o robô e adaptação do tempo de avaliação. O resultado obtido com essas modificações está ilustrado na Figura 4.13. É possível notar que ainda existe ruído, que é causado principalmente pela variação na densidade de obstáculos que o robô encontra. Uma vez testada a função de *fitness*, foram realizados os testes com o algoritmo evolutivo.

Figura 4.11. Ambiente real de testes improvisado com caixas e dois obstáculos retangulares para testes do sistema de navegação (vista superior e lateral).

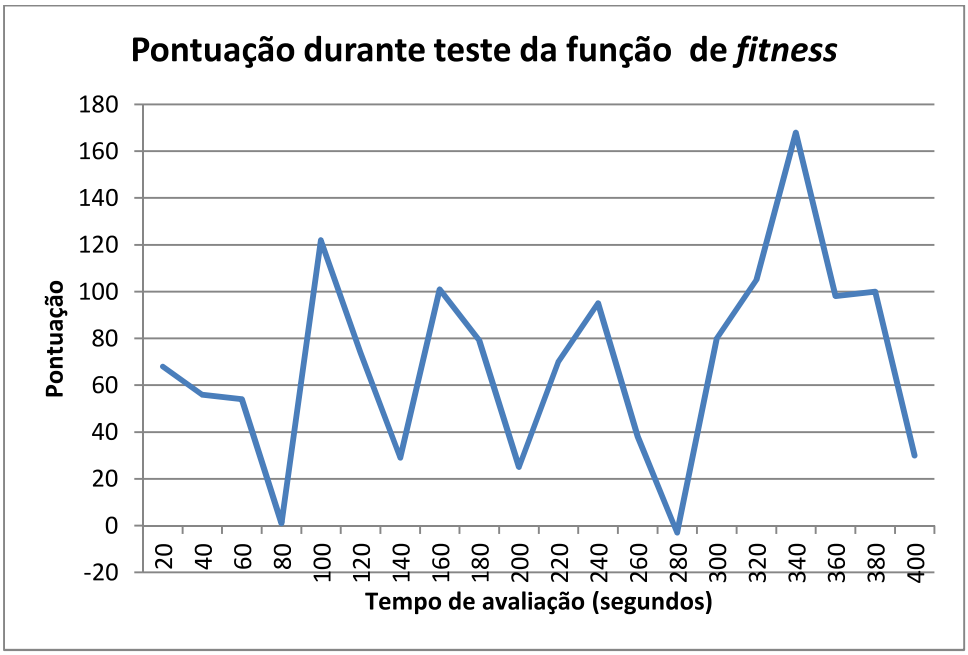


Figura 4.12. Gráfico das pontuações obtidas no primeiro teste da função de avaliação com o robô sendo controlado por uma rede neural fixa e pré-treinada com exemplos dados por um especialista.

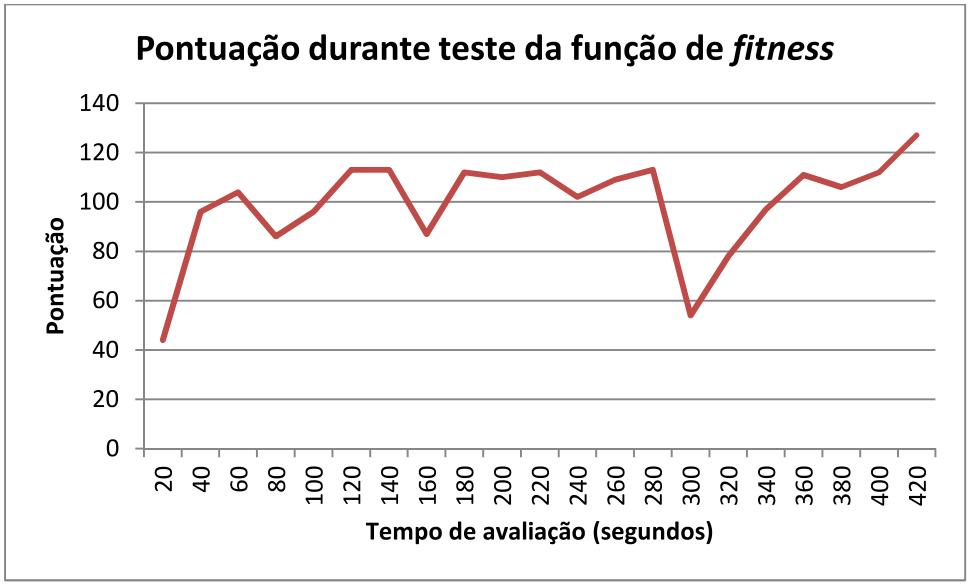


Figura 4.13. Gráfico das pontuações obtidas com as melhorias feitas na função de avaliação com o robô sendo controlado pela mesma rede neural fixada na figura anterior.

Utilizando o mesmo ambiente de testes da Figura 4.11, com uma população inicial aleatória e com o tempo limitado disponível para a evolução, não foi possível obter um resultado suficientemente bom para dirigir o robô evitando colisões. No gráfico da Figura 4.14 está reproduzido o resultado obtido onde pôde ser observada uma tendência de melhora do fitness, em contrapartida há um ruído muito grande causado pelas diferentes posições do robô no momento da avaliação. Foi necessário utilizar uma população inicial treinada com os exemplos fornecidos, e assim o robô se adaptou aos obstáculos e ao espaço disponível, o gráfico do *fitness* pode está apresentado na Figura 4.15. Algumas gerações receberam *fitness* baixo devido à região com maior densidade de obstáculos na qual robô se encontrava.

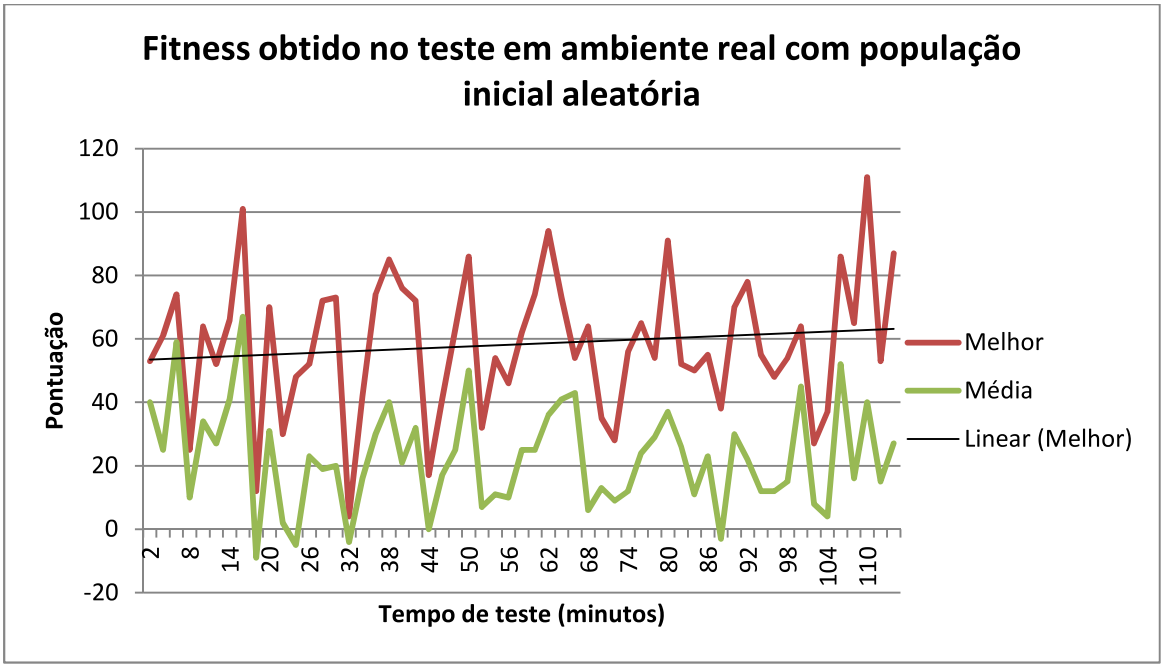


Figura 4.14. Gráfico dos resultados obtidos com o robô sendo testado em um ambiente real. Linhas do melhor indivíduo e média por geração, e tendência de crescimento da pontuação do melhor indivíduo.

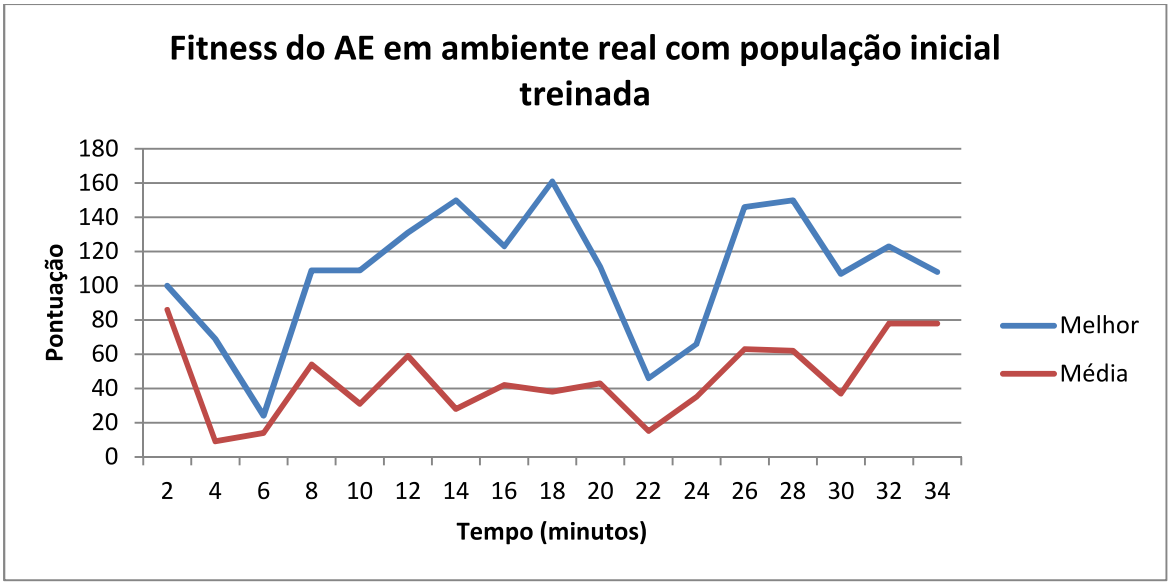


Figura 4.15. Gráfico do robô sendo testado em ambiente real com população inicial treinada.

4.2. Análise dos resultados

Como esperado, as distâncias lidas independentemente com os sonares apresentaram precisão de variação de apenas 3 cm, sendo suficiente para alimentar o sistema de navegação. Os sensores utilizados em conjunto apresentaram alguns problemas com relação à reflexão do som, abrindo espaço para uma melhoria no projeto: o acionamento independente de cada um dos sensores.

Os motores puderam ser acionados com sucesso através do uso do *PWM* gerado pelo periférico presente no *ARM*. Porém, utilizar uma frequência audível para chaveamento dos

motores, introduz um ruído desagradável para pessoas próximas ao robô. Como o *driver* utilizado suporta um pulso modulado com uma frequência de pelo menos 25kHz, seria interessante gerar um pulso com tal frequência, se fosse desejado um desempenho mais silencioso dos motores. Outra possível melhoria poderia ser obtida com a instalação de *encoders*, pois a estabilidade da velocidade dos motores se mostrou suscetível a variações de carga nas baterias.

A rede neural programada apresentou uma solução básica para o problema de navegação do robô. Porém a utilização de um simulador seria mais apropriada para fornecer uma solução inicial para o algoritmo evolutivo, devido ao tempo necessário para executar a função de *fitness* de cada indivíduo. O resultado obtido com o algoritmo evolutivo foi prejudicado pela limitação das situações que o robô se depara durante um período de avaliação. Essa limitação se deve ao fato de que as situações encontradas pelo robô dependem das decisões atuais da rede. Por exemplo, se uma configuração aplicada faz com que o robô gire ao encontrar um espaço livre, uma situação variada de obstáculos com distâncias distintas não será encontrada pelo sistema de navegação.

5. Conclusões

Este trabalho demonstrou a viabilidade de utilização da placa *Raspberry Pi* para implementar o sistema de controle de navegação de um robô móvel autônomo de baixo custo, possibilitando a sua replicação para a construção de um sistema multirrobótico no qual um grande número de robôs com grande capacidade de processamento podem ser construídos. A capacidade do processador embarcado foi testada com a implementação de um algoritmo de controle de complexidade razoável, envolvendo uma rede neural artificial do tipo *RAM* para detectar a situação que o robô se encontra e selecionar uma manobra adequada para exploração do ambiente e desvio de obstáculos. Também foi embarcado um algoritmo evolutivo capaz de otimizar os conteúdos dos neurônios da rede, ajustando continuamente o comportamento do robô às variações no ambiente de trabalho. A capacidade de memória e velocidade de processamento da *Raspberry Pi* mostrou-se mais que satisfatória para executar com eficiência este algoritmo de controle de navegação, calculando o comportamento adequado do robô para cada situação ao mesmo tempo que lia e processava os dados dos sensores e comandava os motores do robô para a realização das manobras.

O processador *ARM* empregado na *Raspberry Pi* é bastante antigo, uma vez que o projeto da Raspberry Pi foi feito em 2011, mas mostrou-se suficiente para executar os algoritmos propostos, que foram escolhidos e adaptados para facilitar sua execução pelo hardware utilizado. A estratégia proposta neste trabalho de utilizar os dois canais de saída de som para geração dos pulsos de *PWM* para os motores possibilitou utilizar os osciladores periféricos embarcados no chip do processador para o controle e a geração dos pulsos, eliminando o gasto computacional da geração de *PWM* por software, como era proposta na literatura.

O acoplamento óptico dos sensores e *driver* dos motores possibilitou isolar o circuito do processador e a placa *Raspberry Pi* de qualquer ruído elétrico que possa ser produzido nos motores ou até mesmo de falhas nos circuitos do *driver* e de alimentação que pudessem causar danos na placa. Esse hardware foi proposto neste trabalho e foi desenvolvido especificamente para o acoplamento da *Raspberry Pi* com os sensores e motores do robô utilizado. Não obstante, este mesmo circuito pode ser utilizado com pequenas modificações para acoplar qualquer outra placa controladora ao hardware do robô, como placas de *Arduino, FPGAs* (*Field-Programmable Gate Arrays*) ou até mesmo dispositivos móveis como telefones celulares e *tablets*.

A navegação de robôs móveis é um desafio que pode ser solucionado com muitos níveis de confiabilidade e precisão. Para o caso estudado neste trabalho, a placa *Raspberry Pi* foi utilizada com sucesso para implementar um sistema de navegação autônoma para um robô de pequeno porte, com dois motores *DC* que acionam esteiras laterais e sonares para detecção de

obstáculos. O robô conseguiu aprender a desviar de obstáculos e pôde se adaptar a algumas variações do ambiente de trabalho, executando algoritmos de redes neurais artificiais e computação evolutiva. Desta maneira, este trabalho vem demonstrar a viabilidade de utilização da placa *Raspberry Pi* como um sistema de baixo custo e alta capacidade de processamento e controle para sistemas baseados em times de robôs móveis autônomos.

Também foi possível implementar um controle de hardware de baixo nível diretamente com a *Raspberry Pi*, sendo necessário apenas garantir a isolação de ruído para proteção do processador presente na placa. Porém, as limitações ficaram evidentes com relação à disponibilidade de periféricos de baixo nível presentes na placa, como *Timers* com *PWM*, conversores *AD* e mesmo pinos digitais de uso geral. Portanto, utilizar um microcontrolador para o interfaceamento com dispositivos periféricos como motores, servos e vários tipos de sensores é aconselhável, principalmente quando um número maior de sensores é utilizado ou quando medidas devem ser realizadas em tempo real.

A instalação e programação do software utilizado na placa se mostraram muito eficientes, visto que o Linux embarcado na placa trás um suporte completo à rede, mesmo com a placa de *wi-fi* utilizada. O compilador *GCC* (GNU Compiler Collection) pré-instalado foi utilizado, e a biblioteca W*iringPi* foi essencial para o desenvolvimento facilitado do interfaceamento com o hardware do robô. As informações presentes na literatura disponibilizada em (Raspberry Pi Forum) foram muito precisas e suficientes para sanar as dúvidas que foram surgindo durante a implementação deste projeto.

A rede neural utilizada no projeto foi simulada com um treinamento baseado em exemplos e seu funcionamento foi comprovado com as decisões corretas que foram tomadas para situações não fornecidas no exemplo. A execução do algoritmo evolutivo foi simulada comparando as soluções da população em evolução com a rede neural treinada, sua convergência pode ser observada com esse teste. Dessa maneira, a inteligência artificial utilizada no projeto foi simulada em blocos independentes e apresentou resultados dentro do esperado. A simulação não foi realizada em um software que considere as propriedades físicas do robô, como atrito e escorregamento das esteiras, o que causou a necessidade de ajustar o algoritmo de navegação no mundo real, tomando mais tempo.

Foram realizados experimentos em uma área delimitada e com obstáculos retangulares para validar o funcionamento da inteligência artificial aplicada no robô. Verificou-se que uma rede neural treinada com exemplos fornecidos por um especialista foi suficiente para dirigir o robô. Porém, quando utilizando o algoritmo evolutivo para treinar um navegador neural a partir de uma população inicial com indivíduos criados aleatoriamente, notou-se que o tempo levado para encontrar uma configuração que desvie com sucesso dos obstáculos é muito longo, ressaltando a

necessidade de futuras melhorias nos mecanismos de evolução. Como há muito ruído no mecanismo de pontuação do robô, sugere-se a implementação do mecanismo de herança proposto em (SIMÕES, 2000), reduzindo a perda de material genético bom causado por uma avaliação incorreta. Quando utilizada uma população inicial pré-treinada, o sistema também apresentou grande quantidade de ruído, porém pode-se notar melhoria na movimentação do robô.

5.1. Trabalhos futuros

Os sensores utilizados nesse projeto trazem um resultado confiável, porém sensíveis a reflexões acústicas. Assim, uma possível melhoria seria aliar outro tipo de sensor aos sonares, como sensores de proximidades infravermelhos. Futuramente podem ser aplicados outros mecanismos de evolução para que um resultado suficiente, ou seja, que dirija o robô sem que ocorram colisões, seja atingido com menos gerações. Outro trabalho interessante para uma evolução mais rápida da população é a utilização de um time de robôs reais evoluindo em conjunto e simultaneamente, dessa maneira, a avaliação dos indivíduos é feita de forma paralela.

5.2. Considerações sobre o curso de graduação

O desenvolvimento desse projeto envolveu uma abordagem abrangente dos conteúdos estudados no curso de engenharia elétrica, em especial os abordados nas disciplinas específicas, como Circuitos Eletrônicos, Eletrônica de Potência, Aplicação de Microprocessadores, e outras que envolvem o estudo da eletrônica e da computação embarcada. Conceitos estudados nas disciplinas básicas do curso foram fundamentais para que houvesse maior agilidade na construção e utilização das partes que compõe o hardware. A ênfase em computação, cursada no ICMC (Instituto de Ciências Matemáticas e de Computação), contribuiu muito para o sucesso do trabalho. As disciplinas cursadas durante a ênfase, como Introdução à Ciência da Computação II, Redes de Computadores, Sistemas Evolutivos e Aplicados à Robótica, entre outras, ofereceram bagagem técnica para implementação de todas as etapas do desenvolvimento do software do projeto. A maior colaboração do curso para o desenvolvimento do trabalho e para a formação em si foi a grande evolução da capacidade de encontrar soluções e alternativas para os problemas encontrados. A robótica exige a fusão de conhecimentos de eletrônica e computação, o que levou a opção pela ênfase em computação, dessa maneira, objetivou-se uma maior completude da formação para atuação na promissora área da robótica.

6. Referências bibliográficas

ABOUT us | Raspberry Pi. **Raspberry Pi**, 2012. Disponivel em: <http://www.raspberrypi.org/about>. Acesso em: 21 Outubro 2013.

AHMED, A. Modulação por largura de pulso (PWM). In: AHMED, A. **Eletrônica de Potência**. São Paulo: Pearson Prentice Hall, 2000. p. 365-368.

AMSTUTZ, P.; CORRELL, N.; MARTINOLI, A. Distributed boundary coverage with a team of networked miniature robots using a robust market-based algorithm. **Annals of Mathematics and Artificial Intelligence, vol. 52**, Abril 2008. 307-333.

AUSTIN, J. **RAM-based Neural Networks**. 1ª. ed. [S.l.]: World Scientific, 1998.

BARBI, I. **Eletrônica de potência**. 6ª. ed. [S.l.]: Edição do Autor, 2006.

BERGBREITER, S. **CotsBots:** An Off-the-Shelf Platform for Distributed Robotics. Department of Electrical Engineering and Computer Sciences: University of California at Berkeley, 2003.

CLAYTON, G.; WINDER, S. **Operational Amplifiers**. 5ª. ed. Burlington: Newnes, 2003. 82-85 p.

D217 Datasheet. **Dual Channel Phototransistor Small Outline Surface Mount Optocouplers**, Abril 2013. Disponivel em: <http://www.fairchildsemi.com/ds/MO/MOCD217M.pdf>. Acesso em: 20 Setembro 2013.

EIBEN, A. E.; SMITH, J. E. **Introduction to Evolutionary Computing**. Holanda: Springer, 2003. Disponível em http://91.216.243.21/fachbuch/leseprobe/9783540401841_Excerpt_001.pdf.

FAIRCHILD, S. Optocoupler Solutions. **FairchildSemi**, 2010. Disponivel em: <http://www.fairchildsemi.com/collateral/solutionguides/Optocoupler-Solutions-Guide.pdf>. Acesso em: Setembro 2013.

FAQS | Raspberry Pi. **Raspberry Pi**, 2013. Disponivel em: <http://www.raspberrypi.org/faqs>. Acesso em: 21 Outubro 2013.

GERKEY, B. P.; VAUGHAN, R. T.; HOWARD, A. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. **Proceedings of the International Conference on Advanced Robotics (ICAR 2003)**, Coimbra, Portugal, 2003. p. 317-323.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**. EUA: Addison-Wesley, 1989.

HATA, A. Y.; SHINZATO, P. Y.; WOLF, D. F. Mapeamento e Classificação de Terrenos Utilizando Aprendizado Supervisionado. **XVIII Congresso Brasileiro de Automática**, Setembro 2010. p. 122-129.

HC-SR04 Datasheet. **Ultrasonic Ranging Module**, 2012. Disponivel em: <http://elecfreaks.com/store/download/HC-SR04.pdf>. Acesso em: 20 Setembro 2013.

HENDERSON, G. Gordons Projects. **Raspberry Pi | Wiring | Gordon Projects**, 2013. Disponivel em: <https://projects.drogon.net/raspberry-pi/wiringpi/>. Acesso em: 2013 Julho 14.

HOFFMANN, F. Evolutionary Algorithms for Fuzzy Control. **Proceedings of the IEEE**, v. 89, n. 9, Setembro 2001.

HOLLAND, J. H.; GOLDBERG, D. E. Genetic Algorithms and Machine Learning. **Machine Learning**, v. 3, p. 95-99, Outubro 1988.

INOUE, K.; OSUKA, T. **H-bridge driver**. 6753717, 22 Junho 2004.

KAR, R. PWM on Raspberry Pi. **Raspberry Pi Blog**, Novembro 2012. Disponivel em: <http://www.rpiblog.com/2012/11/pwm-on-raspberry-pi.html>. Acesso em: 18 Setembro 2013.

LIST of single-board computers. **Wikipedia**, 2013. Disponivel em: <http://en.wikipedia.org/wiki/List_of_single-board_computers>. Acesso em: 30 Setembro 2013.

PBL. Raspberry Pi Schematics, p. 2 de 5, 17 Abril 2012. Disponivel em: <http://www.raspberrypi.org/archives/1090>. Acesso em: Outubro 2013.

RASPBERRY Pi Forum. **RaspberryPi.org**. Disponivel em: <http://www.raspberrypi.org/phpBB3/>. Acesso em: Outubro 2013.

RASPBIAN About. **Raspbian**, 2012. Disponivel em: <http://www.raspbian.org/RaspbianAbout>. Acesso em: 20 Setembro 2013.

RPI Distributions - eLinux.org. **eLinux.org**, 14 Outubro 2013. Disponivel em: <http://elinux.org/RPi_Distributions/>. Acesso em: 18 Outubro 2013.

SASMAL, H. S.; NAYAK, T.; PATNAIK, S. Obstacle detection and navigation of autonomous robot. **International Journal of Computer & Communication Technology**, Institute of Technical Education & Research, Bhubaneswar, Odisha, India, v. 3, 2012. ISSN 2231 - 0371.

SE, S.; DAVID, L.; LITTLE, J. **Vision-based Mobile Robot Localization And Mapping using Scale-Invariant Features**. Department of Computer Science: University of British Columbia, 2001.

SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. A. **Redes Neurais Artificiais para Engenharia e Cieências Aplicadas:** Curso Prático. 1ª. ed. [S.l.]: Artliber, 2010.

SIMA, J. **Introduction to Neural Networks**. Institute of Computer Science, Academy of Sciences of the Czech Republic. [S.l.]. 1998.

SIMÕES, E. V. **Development of an Embedded Evolutionary Controller to Enable Collision-Free Navigation of a Population of Autonomous Mobile Robots**. The University of Kent at Canterbury. (Tese de Doutorado). 2000.

SOFTWARE PWM Library. **Gordons Projects**, 2012. Disponivel em: <https://projects.drogon.net/raspberry-pi/wiringpi/software-pwm-library/>. Acesso em: Outubro 2013.

STMICROELECTRONICS. L298 Datasheet. **Dual Full-Bridge Driver**, 2000. Disponivel em: <http://www.tech.dmu.ac.uk/~mgongora/Resources/L298N.pdf>. Acesso em: 22 Outubro 2013.

TOWNSEND, K. Adafruit 16 Channel Servo Driver with Raspberry Pi. **Adafruit Learning System**, 16 Setembro 2013. Disponivel em: <http://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/>. Acesso em: 30 Setembro 2013.

UEBEL, L. F. et al. Controle Inteligente de Robôs Móveis Autônomos: RAM x Fuzzy x GSN. **II Simpósio Brasileiro de Redes Neurais**, 1995. 43-48.

VOLKOV, M. **Deployment algorithms for multi-agent exploration and patrolling**. Department of Electrical Engineering and Computer Science: Massachusetts Institute of Technology, 2013.

WANG, T.; DANG, Q.; PAN, P. A Multi-Robot System Based on A Hybrid Communication Approach. **Faculty of Computing, London Metropolitan University, London N7 8DB, UK**, 4 Março 2013. Disponivel em: <http://dx.doi.org/10.11114/smc.v1i1.124>.

WOLF, D. F. et al. Robótica Móvel Inteligente: Da Simulação às Aplicações no Mundo Real. In: _____ **XXVIII Jornadas de Atualização em Informática**. [S.l.]: [s.n.], 2009.

I. Apêndice A

Nesse apêndice está apresentado o código utilizado para validação da leitura de distâncias com os sonares.

```
#include <stdio.h>
#include <wiringPi.h>
#include <sys/time.h>

/* Definição dos pinos a serem conectados na Raspberry Pi*/
#define TRIGGER_PIN 10
#define ECHO_PIN1 25
#define ECHO_PIN2 8
#define ECHO_PIN3 7

/* Contagem máxima 10300us ~= 3,5m */
#define TIMEOUT 10300

/* Função para esperar estado no pino Pin (conta o tempo até atingir o estado)*/
int waitState(unsigned char Pin, unsigned char State){
    struct timeval now, start; //Estrutura da <time.h>
    long micros;
    gettimeofday(&start, NULL); //Início da contagem
    micros = 0;
    while(digitalRead(Pin) != State){
        gettimeofday(&now, NULL);
        micros = now.tv_usec - start.tv_usec;
        if(micros < 0) micros = 1000000L + now.tv_usec; /*Correçao s->us*/
        if(micros > TIMEOUT) return(0); /* Testa tempo máximo */
    }
    return micros;
}

int main (){
    int i = 0;
    long width = 0;
    float dist = 0, dist2 = 0, dist3 = 0;
    wiringPiSetupGpio(); /*Seta a numeração dos pinos*/
    piHiPri(99); /*Alta prioridade:aumentar a precisão da contagem*/
    /*Configura os pinos para entrada ou saída. A biblioteca wiringPi implementa o acesso aos registradores de configuração.*/
    pinMode(TRIGGER_PIN, OUTPUT);
    pinMode(ECHO_PIN1, INPUT);
    pinMode(ECHO_PIN2, INPUT);
    pinMode(ECHO_PIN3, INPUT);

    for (i = 0;; i++){
        /* rotina de acionamento do sonar */
        delay(50);
        digitalWrite(TRIGGER_PIN, LOW);
        delayMicroseconds(50);
        digitalWrite(TRIGGER_PIN, HIGH);
        delayMicroseconds(50);
        digitalWrite(TRIGGER_PIN, LOW);
        waitState(ECHO_PIN1,HIGH);
        width = waitState(ECHO_PIN1, LOW);
        dist = width * 340.0 / 1000000.0;
/*        repete acionamento para os outros 2 sonares
        [...]    */
        /*Impressão para debug*/
        printf("Dist= %f | Dist2 = %f | Dist3 = %f\n",dist/2,dist2/2,dist3/2);
    }
    return(0);
}
```