UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

MÁRCIO LUÍS PETRY

CONTROLE HÍBRIDO DE UM ROBÔ AUTÔNOMO SEGUIDOR DE LINHA

TRABALHO DE CONCLUSÃO DE CURSO 2

PATO BRANCO
2016

MÁRCIO LUÍS PETRY

CONTROLE HÍBRIDO DE UM ROBÔ AUTÔNOMO SEGUIDOR DE LINHA

Trabalho de Conclusão do Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná – UTFPR, Câmpus Pato Branco, como requisito parcial para obtenção do título de Engenheiro.

Orientador: Prof. Dr. Fábio Favarim
Coorientador: Prof. Dr. César Rafael ClaureTorrico

PATO BRANCO
2016

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Às 13 horas e 30 minutos do dia 04 de julho de 2016, na sala V109, da Universidade Tecnológica Federal dão Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Fábio Favarim (orientador), Cesar Rafael Claure Torrico (coorientador), Kathya Silvia Collazos Linares e Luciene de Oliveira Marin para avaliar o trabalho de conclusão de curso com o título **Controle híbrido de um robô autônomo seguidor de linha**, do aluno **Marcio Luis Petry**, matrícula 01114999, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Fábio Favarim Orientador (UTFPR)	Cesar Rafael Claure Torrico Coorientador (UTFPR)
Kathya Silvia Collazos Linares (UTFPR)	Luciene de Oliveira Marin (UTFPR)
Beatriz Terezinha Borsoi Coordenador de TCC	Pablo Gauterio Cavalcanti Coordenador do Curso de Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Primeiramente a Deus pela força que me manteve sempre firme em todos os momentos de minha vida. Aos meus pais e familiares pelo apoio e carinho, aos professores Fábio Favarim e César Torrico pela oportunidade, contribuições e tempo em mim investidos. Aos demais professores pelo conhecimento e ensinamentos, aos meus colegas pela paciência e motivações durante todo tempo de graduação.

Obrigado.

RESUMO

PETRY, Márcio Luis. Controle híbrido de um robô autônomo seguidor de linha. 2016. 82 f. Trabalho de Conclusão de Curso - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2016.

Este trabalho descreve o desenvolvimento de um robô autônomo seguidor de linha através da modelagem e implementação do controle híbrido (de tempo contínuo e a eventos discretos). Foram estudados os controladores clássicos PID, a lógica Fuzzy e controle de sinais e eventos discretos além dos componentes e eletrônica necessária para simulação e elaboração de um protótipo. As malhas de controle de tempo contínuo do protótipo foram modeladas matematicamente e implementados em ferramenta de simulação do Matlab. O controle de eventos discretos foi modelado com autômatos de Moore e simulado com as ferramentas Deslab e Supremica. Testes práticos foram realizados em uma pista de teste seguindo as normas da RoboCore com a implementação do controle hibrido no protótipo desenvolvido. Ao final foram analisadas as diferenças entre os controles PID e Fuzzy, no qual o controlador PID obteve o melhor resultado.

Palavras-chave: Robô autônomo seguidor de linha. Controle PID. Lógica Fuzzy. Sistemas a eventos discretos.

ABSTRACT

PETRY, Márcio Luis. Hybrid control of an autonomous line follower robot. 2016. 82 f. Trabalho de Conclusão de Curso - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2016.

This work describes the development of an autonomous line follower robot through the modeling and the implementation of a hybrid control system (part time continuous – part discrete state). We studied the PID classic controllers, the Fuzzy logic control for discrete-time systems, besides the system components and the electronics knowledge required to simulate and make a prototype. The prototype's continuous controllers were mathematically modeled and implemented in a Matlab-based simulation tool. The control of discrete event systems was modeled with Moore's automaton and it was simulated with Deslab and Supremica tools. Practical tests were made at testing tracks, which followed the RoboCore's standards with the implementation of a hybrid control system in the prototype. Finally, we analyzed the differences between the Fuzzy and PID's controllers, in which the PID controller showed better results.

Keywords: Autonomous line follower robot. PID control. Fuzzy logic. Discrete event systems.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta o contexto no qual este trabalho está inserido, bem como suas principais motivações, justificativa, objetivos e organização do trabalho.

1.1 CONSIDERAÇÕES INICIAIS

A robótica é uma área de conhecimento que tem evoluído de forma muito rápida nos últimos anos, entretanto o estudo e o projeto de robôs e autômatos vêm sendo desenvolvido há vários séculos. Os primeiros robôs evoluíram de autômatos complexos passando por robôs manipuladores de base fixa, pelos dispositivos móveis guiados à distância, chegando mais recentemente aos robôs móveis semiautônomos e os totalmente autônomos (JUNG et al., 2005).

Vários centros de pesquisa têm proporcionado à robótica móvel um grande avanço no desenvolvimento de robôs cada vez mais eficientes. A ênfase dessas pesquisas está relacionada com problemas como a operação (movimentação) em ambientes que se modificam dinamicamente, compostos por obstáculos móveis e estáticos. Para operar neste tipo de ambiente é necessário que o robô tenha autonomia/inteligência e ser capaz de adquirir e utilizar conhecimento sobre o ambiente, estimar uma posição, reconhecer obstáculos e responder em tempo real (PESSIM, 2013, p. 2).

Existe uma crescente utilização de robôs autônomos em diversas aplicações como no transporte, vigilância, inspeção, limpeza de casas, exploração espacial, auxilio a deficientes físicos, entre outros. No entanto, os robôs móveis autônomos ainda não causaram muito impacto em aplicações domésticas ou industriais, principalmente devido à falta de um sistema de controle robusto, confiável e flexível que permitiria que estes robôs operassem em ambientes dinâmicos, pouco estruturados, e habitados por seres humanos (HEINEN, 2002).

Com o intuito de promover e divulgar o desenvolvimento e avanço tecnológico em robótica autônoma e incentivar estudantes de engenharia e áreas afins a desenvolver conhecimento por esse tipo de aplicações na área da robótica móvel surgiram vários grupos de pesquisa e comissões organizadoras de eventos como a RoboCore (2015) e a RoboCup (2015) que propiciam competições e desafios em diferentes categorias. Há vários desafios para os robôs de futebol,

robôs de serviço, logística, robôs de educação, robôs de resgate, combate, seguidores de linha entre outros.

Este trabalho tem por motivação o desenvolvimento de um robô autônomo seguidor de linha também conhecido como *LineFollower* para competir em eventos de competições. Normalmente os objetivos dessa categoria são diminuir o tempo para realizar um percurso sobre uma faixa de orientação e ainda reconhecer marcas (sinais de eventos discretos) indicando algumas informações do percurso, por exemplo, marca de início de trajeto, cruzamentos, curvas, retas etc. Esse sistema com variáveis contínuas e eventos discretos interagindo entre si será denotado neste trabalho como sistema híbrido (GUADAGNIN, 2014).

1.2 JUSTIFICATIVA

O desenvolvimento de um sistema de controle robusto que possibilite um seguidor de linha reconhecer marcações de curvas, início e fim de uma faixa central sobre uma superfície plana e percorrer um circuito com as restrições impostas nas competições de robôs seguidores de linha com menor tempo possível é o maior desafio neste tipo de evento.

Durante o ano de 2014, Guadagnin (2014) desenvolveu o Trabalho de Conclusão de Curso, no Curso de Engenharia Elétrica da UTFPR, Câmpus Pato Branco, intitulado “Controle Híbrido de um Robô Seguidor de Linha**”,** esse trabalho forma a base de conhecimento para o desenvolvimento deste trabalho de conclusão de curso.

No trabalho de Guadagnin (2014) foi desenvolvido a modelagem e implementação do controle híbrido (de tempo contínuo e a eventos discretos) em um robô seguidor de linha apresentado na Figura 1, implementando os métodos formais de controle contínuo (controladores PID) e também a eventos discretos (autômatos de Moore e Mealy). O robô desenvolvido tem a capacidade de reconhecer e percorrer uma linha que forma o caminho de um circuito desse tipo de competição. Durante a elaboração do referido trabalho alguns problemas foram encontrados pelo autor, como falhas no sensoriamento ao reconhecer marcações na faixa de orientação (início e fim de curvas, início e fim da pista), mas devido à falta de tempo e recursos limitados estas falhas não foram tratadas.

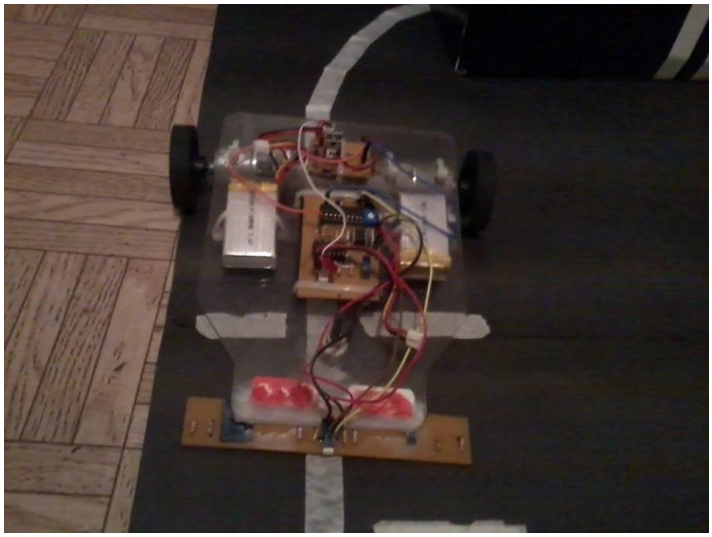

Figura 1: Robô seguidor de linha implementado por Guadagnin
Fonte: Guadagnin (2014).

A partir da experiência relatada no trabalho de Guadagnin (2014) propõe-se neste trabalho a modelagem de um protótipo novo com um sistema de sensoriamento (hardware) mais eficiente através de implantação de uma arquitetura de controle híbrido (software). O controle de posição (variável continua) foi modelado utilizando o controlador PID e a lógica *Fuzzy*. A detecção de marcas laterais (eventos discretos) será modelada através de uma metodologia formal para garantir estimativas de posição mais precisas que podem ser obtidas integrando informações cinéticas do robô através dos sensores de reconhecimento de faixa, velocidade e direção.

Para fins de teste e comparação de resultados também será implementado no protótipo a ser desenvolvido o sistema de controle proposto por Guadagnin (2014).

1.3 OBJETIVOS

A seguir são apresentados o objetivo geral e os objetivos específicos deste trabalho.

1.3.1 Objetivo Geral

Este trabalho tem como objetivo geral implementar um protótipo de robô autônomo seguidor de linha implementado com controle hibrido (de tempo contínuo e a eventos discretos) aperfeiçoando as técnicas de controle desenvolvido no trabalho de Guadagnin (2014).

1.3.2 Objetivos Específicos

- Testar diferentes estruturas e tipos de sensores fotoelétricos capazes de perceber diferentes níveis de refletância da superfície em que o robô irá percorrer;
- Projetar circuitos de condicionamento de sinais para os diferentes tipos de sensores;
- Criar um novo modelo do sistema mecânico do robô;
- Implementação do controle proposto em Guadagnin (2014) (controle PID) no protótipo desenvolvido;
- Modelar um controle *Fuzzy* aplicado a robô seguidor de linha;
- Implementar o controle *Fuzzy* no protótipo desenvolvido;
- Realizar testes do protótipo desenvolvido em um circuito de competições da Robocore aplicando os dois controles implementados.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho encontra-se estruturado da seguinte forma:

Capítulo 2: Esta seção apresenta uma introdução à robótica móvel, revisão bibliográfica, definições de sistemas de controle, sensores e atuadores para robôs móveis. Ao final de seção são apresentadas as regras de competições para robôs seguidores de linha.

Capítulo 3: São apresentados os materiais e métodos utilizados na modelagem do protótipo do robô desenvolvido.

Capítulo 4: É apresentado a implementação e simulação dos controladores.

Capítulo 5: São apresentados os experimentos e resultados.

Capítulo 6: Conclusão e considerações finais.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos que fundamentam a proposta deste trabalho.

2.1 ROBÓTICA MÓVEL

A robótica móvel é uma subárea da robótica clássica, que tem como ênfase a pesquisa e o desenvolvimento de soluções relacionadas a problemas de operação (locomoção) de robôs móveis em ambientes complexos, com obstáculos e/ou que se modificam dinamicamente.

Na robótica móvel, há dois tipos de robôs: os veículos guiados automaticamente (AGV - do acrônimo em inglês: *Automatic Guided Vehicle*) e os robôs móveis autônomos. Os primeiros operam em ambientes projetados com caminhos induzidos através de fios no chão, faróis de direção, ou usam imãs ou lasers para navegação executando tarefas de transporte ao longo de rotas fixas. Por esta razão se tornam inflexíveis e frágeis: alterar uma rota implica em aumento de custo e qualquer mudança inesperada (tais como objetos bloqueando o caminho) pode levar a falhas na execução da tarefa. Como alternativas a este tipo de robô surgiram os robôs móveis autônomos (NEHMZOW, 2000).

Robôs seguidores de linha podem ser classificados como um tipo de veículo guiado automaticamente e possuem mecanismos que lhes permitem se movimentar em um ambiente de forma autônoma ou semiautônoma (WOLF et al., 2009).

Com o atual avanço tecnológico, uma grande variedade de robôs móveis vem sendo desenvolvidos em diferentes categorias de robôs tais como os terrestres, aquáticos e aéreos. Segundo Marin (2010), apesar do hardware evoluir continuamente, resultando em diversos tipos de robôs móveis, o problema da navegação autônoma continua presente na maioria das aplicações.

Na revisão bibliográfica a seguir que subsidia os conhecimentos necessários para o desenvolvimento deste trabalho várias técnicas foram desenvolvidas para implementar os sistemas de navegação de robôs móveis autônomos para atividades específicas.

Em Heinen (2002), é descrito um sistema de controle robusto para robôs móveis e autônomos que é capaz de operar e se adaptar a diferentes ambientes e

condições. Nesse trabalho foi desenvolvida uma arquitetura de controle híbrida capaz de navegar em um ambiente dinâmico, desviando tanto de obstáculos estáticos conhecidos num mapa como de obstáculos móveis imprevistos. Heinen integrou em seu trabalho um modulo localizador capaz de localizar o robô no ambiente utilizando as informações sensoriais e um mapa. Seu sistema de localização mantém uma estimava de posição correta quando a localização inicial é conhecida, além de se localizar em um ambiente dinâmico com a utilização de um filtro de distância. Para a validação do sistema proposto, foi implementado um simulador de robôs móveis (SimRob3D) que permite a utilização de modelos de ambientes tridimensionais, bem como diversos modelos sensoriais e cinemáticos.

Marin (2010) propôs uma arquitetura de controle inteligente que dispõe de um método de aprendizado adaptativo de um mapa topológico para navegação de robôs móveis, com recursos limitados de sensores, memórias e processamento. Nesse trabalho foram integradas técnicas de inteligência artificial, tais como, redes neurais artificiais e aprendizagem por reforço. O sistema provê a habilidade de executar tarefas de navegação em labirintos do tipo *T* desconhecidos e modificáveis durante seu tempo de operação.

Outros dois trabalhos importantes são de Vuong (2006) e Nogueira (2013) que apresentam a implementação de controladores utilizando a lógica *Fuzzy* em hardware configurável. Nogueira apresenta detalhadamente o desenvolvimento de um controlador *fuzzy* para um robô móvel autônomo, desenvolvida na linguagem de descrição de hardware, *VHDL*, e a sua implementação em uma placa *DE2* do fabricante Altera, usando o software de desenvolvimento *Quartus II*, do mesmo fabricante. O objetivo do sistema de navegação proposto é um protótipo de robô capaz de navegar no ambiente sem colidir em nenhum obstáculo, usando para comunicação externa com o ambiente, três sensores de distância localizados um em cada lado do robô e um na frente, atuando através dessas informações em dois motores de passo.

Correia (2013) desenvolveu um sistema robótico móvel e autônomo de vigilância para ambientes internos. A navegação é baseada em mapas topológicos, sensores de percepção 3D e térmicos. O protótipo desenvolvido é capaz de monitorar e se deslocar sem colidir em ambientes que podem apresentar riscos à integridade física de pessoas.

Su (2010) desenvolveu um protótipo seguidor de linha que utiliza parâmetros de ganho para calibrar automaticamente sete sensores ópticos reflexivos utilizando lógica *fuzzy*. Para registrar todas as informações sobre a faixa percorrida o robô autônomo usa *encoders* nos servos motores. O firmware pode então planejar perfis para a corrida mais rápida. O robô inventado ganhou o primeiro prêmio em vários concursos nacionais da categoria seguidor de linha no Taiwan, mostrando alta eficiência e superioridade entre os demais competidores.

Neste trabalho optou-se por desenvolver um sistema de controle e navegação de um robô autônomo baseado em controle PID e uma das técnicas da inteligência artificial, a lógica *fuzzy*, também conhecida como lógica nebulosa ou difusa mesclada a sistemas de controle a eventos discretos empregando autômatos finitos. Sistemas que utilizam mais de uma técnica de controle são conhecidos na literatura como sistemas híbridos.

2.2 SISTEMAS DE CONTROLE PARA ROBÔS MÓVEIS

Para definir o que é um sistema de controle, é necessário saber quais os elementos que interagem com o sistema de controle. O sistema controlado está ligado diretamente ao sistema de controle. O ambiente não pode ser totalmente controlado pelo sistema de controle, mas interfere de maneira significativa em seu funcionamento (Figura 2) (HEINEN, 2002).

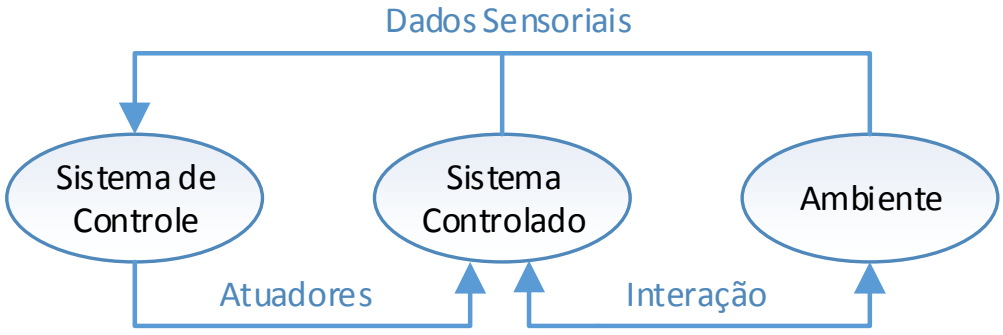

Figura 2 – Sistema de controle de robôs móveis
Fonte: Adaptado de Heinen (2002).

Os componentes que formam o sistema de controle podem ser classificados em 3 grupos (JUNIOR, 2006):

1. Percepção: Envolvem as atividades de interpretação e integração dos sensores, modelagem do mundo real, fazem o reconhecimento;
2. Planejamento: Envolve o planejamento das tarefas, sincronização, e monitoramento de toda atividade executada pelo robô;
3. Atuação: Envolve as atividades de execução dos movimentos e ações do robô e controle dos atuadores.

Sistemas de controle fornecem resposta a uma determinada entrada de acordo com sua função de transferência. Para os sistemas chamados inteligentes fornecer respostas para solucionar problemas e situações específicas, é necessário que esses sistemas tenham um comportamento único e criativo. Atualmente, existe um descompasso entre a capacidade criativa dos seres humanos e a possibilidade de solução que as máquinas computacionais proporcionam, devido ao fato de que as pessoas raciocinam de forma incerta, imprecisa, difusa, enquanto as máquinas e computadores são movidos por raciocínio preciso e binário. A eliminação dessas restrições que faz as máquinas raciocinarem da mesma maneira imprecisa como os humanos é chamada lógica *Fuzzy (*SHAW, 1999).

Este trabalho combina técnicas de controle a eventos discretos e uma função da estratégia global com lógica *fuzzy*. Essas técnicas de modelagem são apresentadas detalhadamente a seguir. Para comparação dos resultados, também será feita uma abordagem do sistema de controle clássico PID, mesclada ao controle de eventos discretos.

2.2.1 Modelos Para Sistemas a Eventos Discretos.

Segundo Cury (2011), sistemas a eventos discretos (SED) podem ser definidos como um sistema dinâmico que evolui de acordo com a ocorrência abrupta de eventos físicos, em intervalos de tempo em geral irregulares e desconhecidos.

Eventos como o início e o término de uma tarefa e a percepção de uma mudança de estado de um sensor são por natureza instantâneos, e tem um caráter discreto no tempo. Para modelar tais sistemas vários modelos foram desenvolvidos como: Redes de Petri, Cadeias de Markov, Teoria das Filas, Teoria de Linguagem e autômatos, etc. Esses modelos refletem diferentes tipos de SEDs bem como diferentes objetivos na análise dos sistemas em estudo.

Neste trabalho o controlador SED foi modelado utilizando autômatos determinísticos de estados finitos. Um autômato pode ser representado graficamente como um grafo dirigido, em que os nós representam os estados e os arcos etiquetados representam as transições entre os estados. O estado inicial é identificado através de uma seta apontando para ele e os estados finais são representados com círculos duplos ou preenchidos (CURY, 2011). No exemplo da Figura 3 é representado um autômato determinístico de uma máquina com dois eventos (Liga/Desliga) e dois estados (S0/S1).

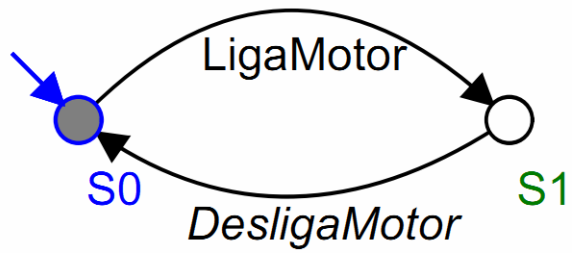

Figura 3 - Exemplo autômato determinístico

A modelagem dos componentes de um sistema é individualmente organizada dentro de um arranjo lógico, de modo que a execução conjunta desses elementos determina o comportamento global do sistema, também conhecido como planta. Modelar a planta de um SED requer a identificação do comportamento de cada subsistema. Logo, o modelo da planta pode assumir diferentes níveis de abstração e a escolha de qual deles usar é uma tarefa puramente de engenharia (TEIXEIRA, 2013).

2.2.2 Modelagem e Controle *Fuzzy*

A teoria dos conjuntos fuzzy foi introduzida em 1965, por Lotfi A. Zadeh. Seu objetivo era fornecer ferramentas matemáticas capazes de lidar com o raciocínio lógico que contemplasse aspectos imprecisos e ambíguos de uma maneira sistemática e lógica em um sistema de controle (ZADEH, 1965).

Devido a sua flexibilidade em relação à pertinência de seus elementos, os conjuntos *fuzzy* permitem a existência da pertinência parcial ao invés da classificação tradicional (um elemento pertence ou não pertence). Essa nova forma de tratar as informações disponíveis, possibilitam considerar a ambiguidade presente no mundo real, dando margem à resolução de inúmeros problemas onde se faz

necessária uma flexibilização dos conjuntos trabalhados (PEDRYCZ; GOMIDE, 2007).

Sistemas baseados em lógica fuzzy podem ser utilizados em praticamente todas as áreas de conhecimento, como engenharia, matemática, biologia, medicina, etc. Como exemplos de sua utilização prática podemos definir a quantidade de água que tem num copo. Pela definição tradicional o copo poderia pertencer ou não aos conjuntos, vazio, médio ou cheio. Com a lógica *fuzzy* existe uma maior flexibilidade na fronteira do conjunto, por exemplo um copo estaria "quase cheio" ou "quase vazio" como mostrado na Figura 4.

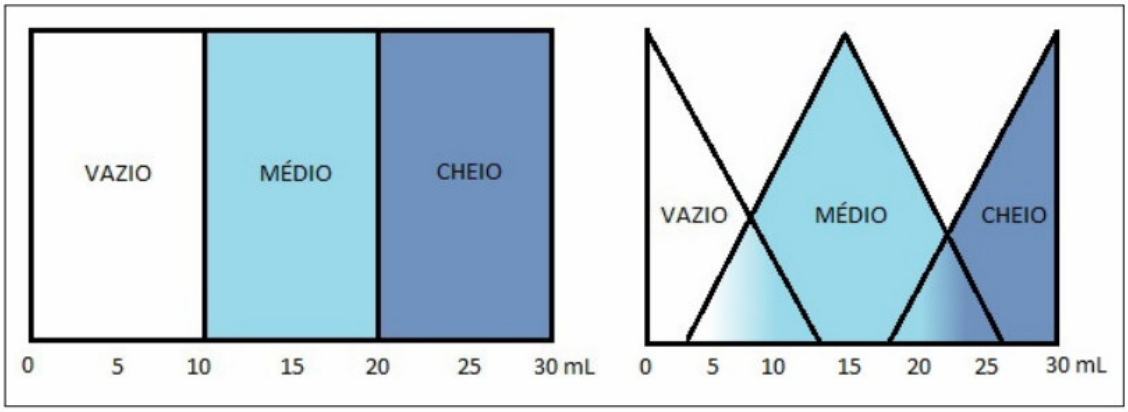

Figura 4-Representação do copo de água na forma de conjuntos com: a) lógica clássica e b) lógica nebulosa
Fonte: Adaptado de MATTIELLO (2014).

2.2.2.1 Funções de Pertinência

Para classificar um determinado dado, como a temperatura, idade, volume, etc., em um determinado grupo de forma gradual e menos abrupta, Zadeh (1965) propôs um grau de pertinência a cada um dos conjuntos *fuzzy*.

O grau de pertinência de um item é normalmente uma função que gera um número real entre 0 e 1, geralmente representado pela letra grega μ. Considerando um conjunto *A* e um elemento *α* com relação a esse conjunto, pode-se representar a tomada de decisão na teoria clássica pela expressão (1), e a da lógica *fuzzy* pela expressão (2).

$$f(x) = \begin{cases} 1, & se, e\ somente\ se, x \in \mathbf{A} \\ 0, & se, e\ somente\ se, x \notin \mathbf{A} \end{cases} \qquad \text{Eq. (1)}$$

$$\mu(x) = \begin{cases} 1, & se, e\ somente\ se, x \in A \\ 0, & se, e\ somente\ se, x \notin A \\ 0 \leq \mu(x) \leq 1, & se\ x\ pertence\ parcialmente\ a\ A \end{cases} \quad \text{Eq.(2)}$$

Assim, na lógica *fuzzy*, um determinado elemento pertence a um conjunto com certo grau de pertinência com intervalo [0,1], fazendo com que determinada sentença possa ser parcialmente verdadeira e parcialmente falsa. Já na teoria dos conjuntos clássica denominada conjuntos *crisp*, o grau de pertinência assume apenas dois estados: compatível ou incompatível.

Existem vários formatos para as funções de pertinência e a escolha entre uma delas depende do problema a ser modelado e da capacidade computacional disponível para processar o que se deseja. As formas mais comuns são as funções triangulares, trapezoidais e gaussianas, conforme ilustrado na Figura 5.

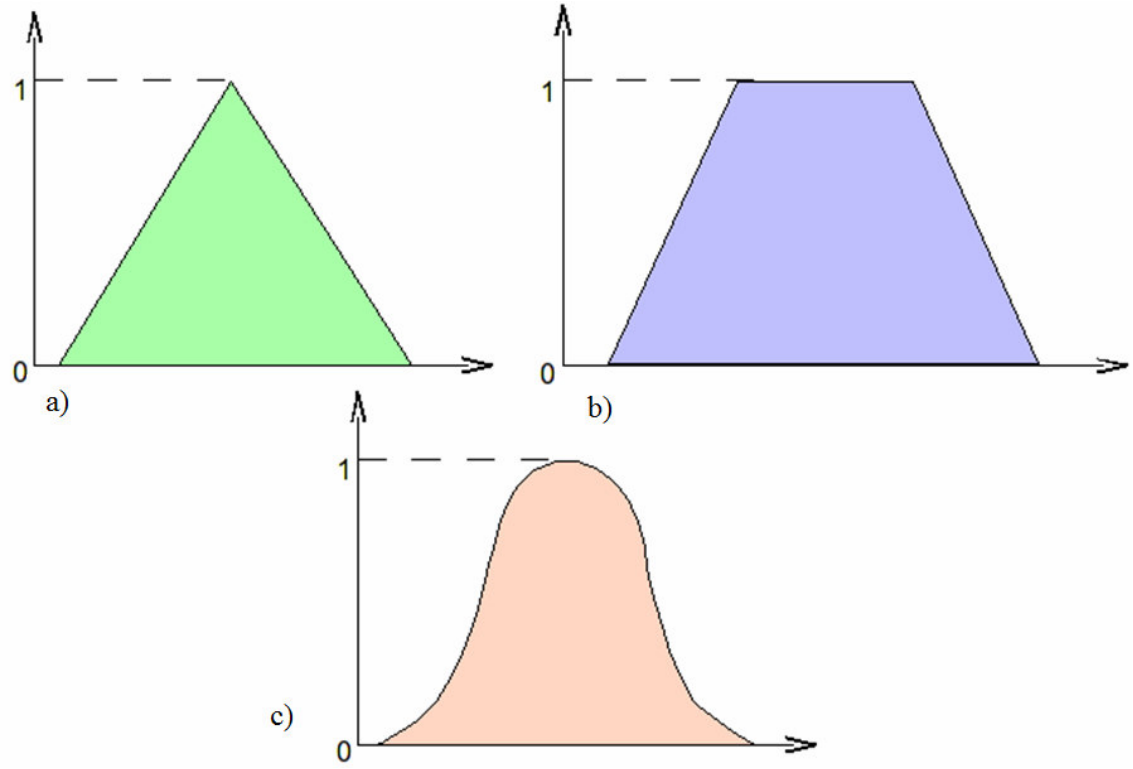

Figura 5 – Funções de pertinência a) triangular, b) trapezoidal, c) gaussiana, mais usuais da lógica *Fuzzy*

2.2.2.2 Operações e propriedades em conjuntos *Fuzzy.*

Da mesma forma como na teoria clássica, os conjuntos *fuzzy* obedecem a certas propriedades e podem ser operados de diversas maneiras. As operações

básicas dos conjuntos *fuzzy*, definidas por Zadeh, estão definidas como segue, considerando dois conjuntos *fuzzy*, A e B num universo U (MARRO et al., 2010):

$$A = \left\{\frac{x, \mu A(x)}{x} \in U\right\}, \mu A(x) \in [0,1]$$

$$B = \left\{\frac{x, \mu B(x)}{x} \in U\right\}, \mu B(x) \in [0,1]$$

- Igualdade: $A = B \leftrightarrow (\mu A(x) = \mu B(x)), \forall x \in U$
- Inclusão: $A \subseteq B \leftrightarrow (\mu A(x) \leq \mu B(x)), \forall x \in U$ onde A é, então um subconjunto *fuzzy* de B.
- Pertence: $(A \subset B) se A(X) < B(x), \forall x \in U$
- União: $\mathbf{A \cup B} = A(x) \cup B(x) = \max[A(x), B(x)]$
- Intersecção: $A \cap B = A(x) \cap B(x) = \min[A(x), B(x)]$
- Complemento: $\neg \mathbf{A(x)} = \mathbf{1 - A(x)}$

Além das operações e das relações os conjuntos *fuzzy* possuem algumas características especiais. Entre tais características encontram-se: Corte x, Conjunto de Níveis, Suporte, Altura e Normalização.

2.2.2.3 Variáveis Linguísticas

A base de regras junto com a base de dados constitui a base de conhecimento do sistema *fuzzy*. Os conjuntos *fuzzy* podem ser usados para construir conjuntos de termos linguísticos. Os conjuntos de termos representam abstrações dos valores da variável, isto é, são uma partição *fuzzy* de seus próprios valores. Em geral, a cada variável linguística, estão associadas expressões linguísticas que qualificam as variáveis em termos linguísticos, portanto, de forma imprecisa (NOGUEIRA, 2013).

Uma variável linguística pode ser representada por um conjunto fuzzy existente no universo U. Por exemplo, para uma variável linguística de temperatura são admitidos como valores expressões linguísticas do tipo: “muito frio”, “frio”, “quente”, “muito quente” como demonstrado na Figura 6.

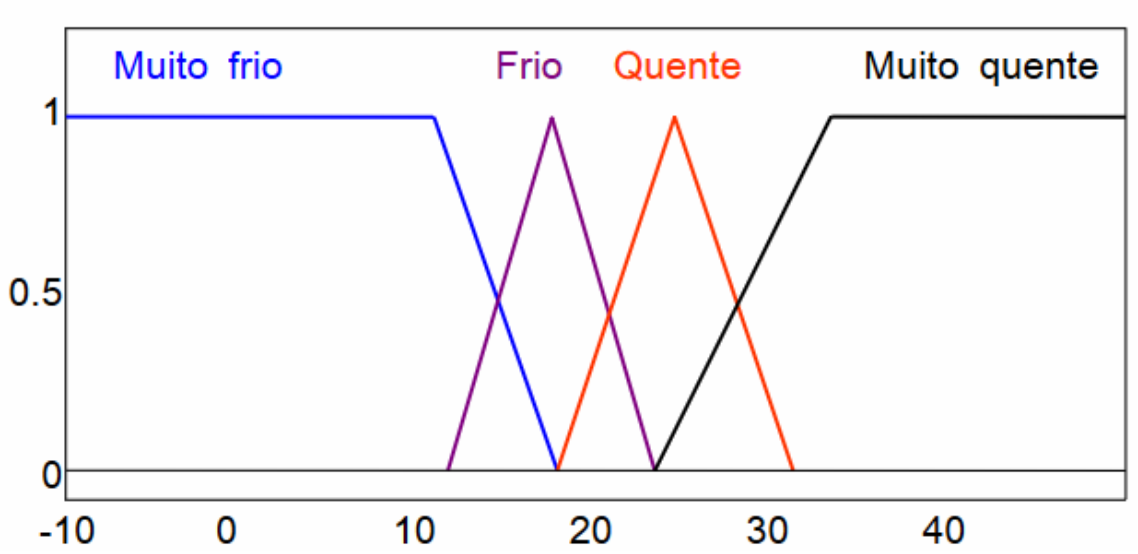

Figura 6 – Exemplo de utilização de variáveis linguísticas

2.2.2.4 Controlador *Fuzzy*

Segundo Gomide e Gudwin (1994), um controlador *fuzzy* baseado em regras é descrito por meio de variáveis linguísticas interconectadas com várias ações a serem tomadas.

A estrutura desse controlador *fuzzy* é mostrada na Figura 7, enfatizando seus componentes básicos:

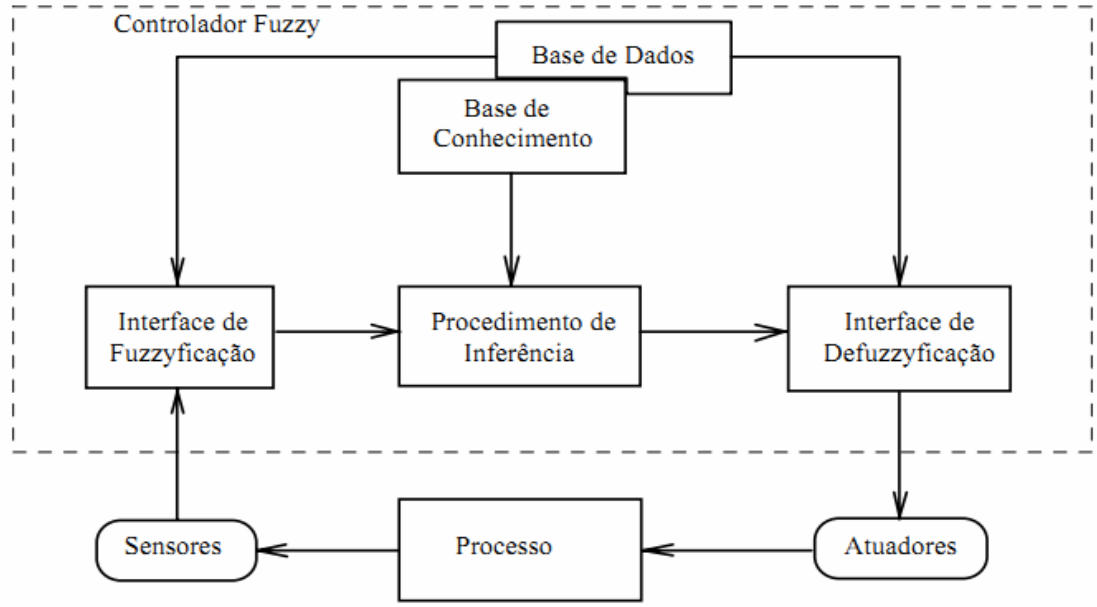

Figura 7 – Estrutura básica de um controlador *Fuzzy*

Fonte: Adaptado de GOMIDE e GUDWIN, (1994).

- Fuzzyficador converte uma entrada *crisp* em um conjunto de termos *Fuzzy;*
- Regras *fuzzy* (base de dados) baseadas na execução do sistema;
- Inferências *fuzzy* executam raciocínio associado às variáveis de entrada com regras *fuzzy*;
- Defuzzyficador converte as saídas do controlador *fuzzy* para um valor *crisp,* para a entrada do sistema real sobre o alvo.

2.2.2.4.1 Fuzzyficação

É a etapa do processo responsável pela conversão dos dados de entrada (valores *crisp)* em graus de pertinência a cada conjunto fuzzy através do conhecimento do especialista. Tomando como exemplo a Figura 8, onde a variável *fuzzy* é a altura de uma pessoa e definindo os conjuntos A1= baixo, A2= médio, A3= alto, obtém-se os seguintes graus de pertinência: μ(x=A1)=0,7, μ(x=A2)=0,2 e μ(x=A3)=0.

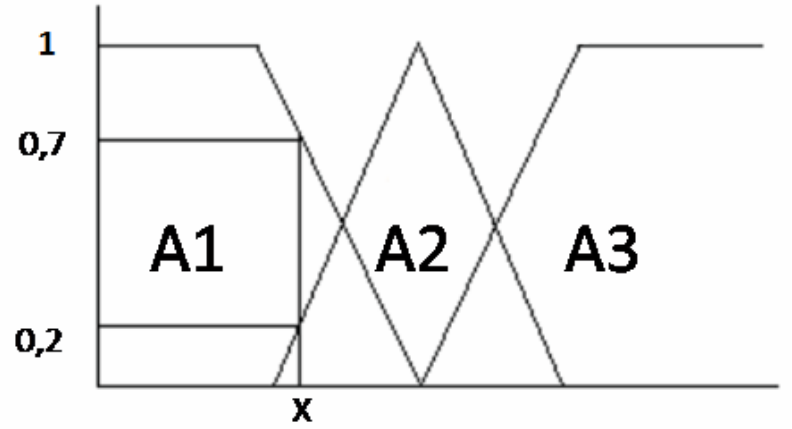

Figura 8 – Variáveis *fuzzy* para os conjuntos de altura (baixo, médio e alto) de uma pessoa

2.2.2.4.2 Regras *Fuzzy*

A construção da base de regras é o passo mais difícil no desenvolvimento de um controlador *fuzzy*, e também um fator crucial para o bom funcionamento.

As regras *fuzzy* fornecem uma descrição qualitativa do sistema em estudo. O conhecimento em um Sistema de Inferência *Fuzzy* (SIF) é armazenado em geral na forma de regras compostas pelos termos; antecedente e o consequente, SE

<antecedente> ENTÂO <consequente> e também das operações nebulosas (MINUSSI, 2009).

As regras de controle são baseadas no conhecimento e expectativa do projetista sendo que cada uma delas demanda uma ação de controle. A ordem em que as regras estão dispostas não afeta o resultado, pois elas são declarativas e não sequenciais (NOGUEIRA, 2013).

O número de regras necessárias para resolver um problema depende da quantidade de entradas e do número de conjuntos difusos de cada entrada do sistema controlado bem como da tarefa a ser realizada.

2.2.2.4.3 Motor de Inferência

O motor de inferência de um controlador é composto por duas tarefas básicas:

1. *Casamento – Matching*: determina o grau que cada regra existente na base de regras se adapta as condições de entrada do controlador, gerando valores de pertinência para as premissas de cada uma das regras da base de pertinência.

2. *Passo de Inferência*: é responsável por determinar a saída do controlador de acordo com os dados obtidos no processo do casamento. Para isso são somadas as saídas referentes à função de pertinência de cada uma das regras presentes na base de regras do controlador (SIERAKOWSKI, 2006).

2.2.2.4.4 Defuzzificação

O processo que converte a saída de um conjunto *fuzzy* obtida pelo motor de inferência para um valor escalar é conhecido como defuzzyficação.

Existem diversos métodos para defuzzyficar a saída, sendo os métodos de centro de gravidade, média da área e média dos máximos os mais conhecidos e apresentados a seguir:

1) **Centro de gravidade**: A saída *u* é calculada através do centro de gravidade do conjunto *fuzzy,* ou seja, o sinal de saída é uma média ponderada dos elementos presentes no conjunto *fuzzy* conforme equação (3).

$$u = \frac{\sum i\ u(xi) * xi}{\sum i\ u(xi)} \qquad \text{Eq. (3)}$$

2) Média da Área: Este método considera a linha vertical que divide a área sob a curva em duas áreas iguais, onde o *Min* é o ponto mais à esquerda e o *Max* é o ponto mais à direita do conjunto *fuzzy* equação (4).

$$u = \left\{ x \left| \int_{Min}^{x} u(x)dx = \int_{x}^{Max} u(x)dx \right. \right\} \qquad \text{Eq. (4)}$$

3) **Método da média dos máximos**: Gera uma ação de controle que representa o valor médio de todas as ações de controle individuais cujas funções de pertinência assumem o valor máximo.

Cada método fornece respostas diferentes, sendo necessário então, que se escolha um método mais adequado visando a aplicação. Outro fator que acarreta em diferenças na resposta do sistema é a inclinação das retas das funções e as áreas onde há superposição destas funções (NOGUEIRA, 2013).

2.2.3 Modelagem e Controle *PID*

O Sistema de Controle PID (Proporcional Integral Derivativo) é uma das técnicas mais empregadas quando se deseja realizar o controle de variáveis contínuas, sendo amplamente usado em sistemas de controle industrial devido ao seu desempenho robusto e simplicidade funcional. O controle PID consiste em um algoritmo matemático composto por três coeficientes: proporcional, integral e derivativo, que são variados para obter a resposta ideal conforme a expressão (5).

$$u(t) = VM(t) = Kpe(t) + Ki \int_{0}^{t} e(\tau)d\tau + Kd\frac{d}{dt}e(t) \qquad \text{Eq. (5)}$$

Sendo que:

u(t) é a saída do controlador em relação ao tempo conhecida também como variável manipulada (*VM(t)*);

e(t) é o sinal de erro, o qual representa a diferença entre a entrada e saída;
Kp é o ganho proporcional;
Ki é o ganho integral;
Kd é o ganho derivativo;

Na Figura 9 é representado um sistema a Malha Fechada com controle PID com uma referência *ref(t)* e uma saída *y(t).* O erro *e(t)* é gerado pela diferença entre os valores de saída e entrada, esse valor é ajustado pelos coeficientes do controlador PID e aplicado ao sistema físico controlado representado pela planta.

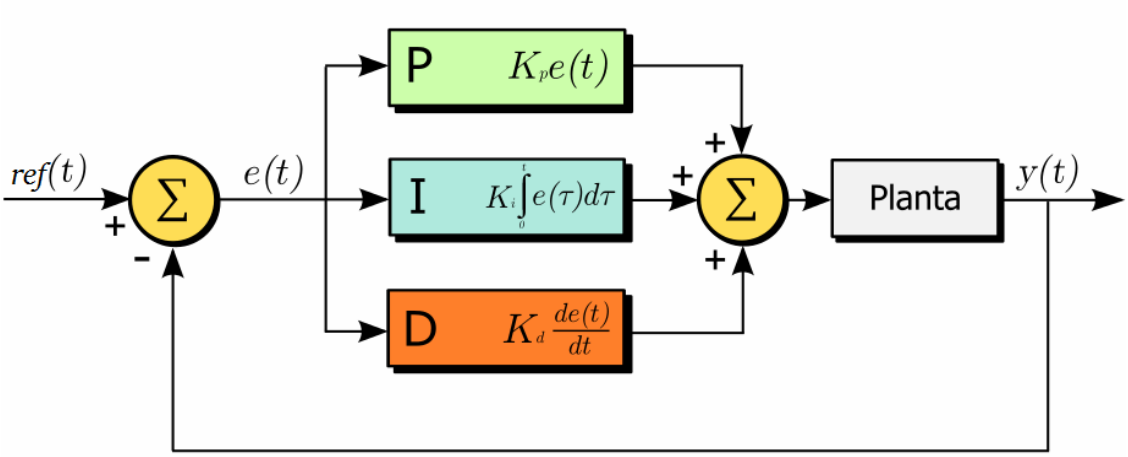

Figura 9 – Diagrama de Blocos de um sistema de controle PID
Fonte: Adaptado de (OGATA, 1998).

2.3 SENSORES E ATUADORES

Os sistemas de controle apresentados na seção anterior consistem de subsistemas e processos (ou plantas) reunidos com o propósito de controlar as saídas dos processos, em que uma entrada de referência é comparada com a saída do sistema, gerando um sinal de erro. O elemento controlador trata estes sinais que posteriormente são amplificados e enviados aos atuadores do sistema (OGATA, 1998).

Assim, a unidade de controle responde pelo gerenciamento e monitoramento dos parâmetros operacionais requeridos para realizar as tarefas do robô. Os comandos de movimentação enviados aos atuadores são originados de

controladores de movimento e baseados em informações obtidas pelos sensores (FELIZARDO; BRACARENSE, 2005).

Os sensores quando operam de forma direta, transformando uma forma de energia em outra são chamados de transdutores, ou seja, conversores de grandezas físicas em sinais elétricos correspondentes. Um robô equipado com sensores é capaz de monitorar a velocidade com que se move, a posição em que se encontra, e detectar obstáculos a sua volta, entre outras grandezas físicas (MORAES, 2003).

Os sensores são classificados pelo tipo de grandeza que avaliam. Assim, há sensores de distância (laser, ultrassom), sensores de posicionamento absoluto do robô (sistemas de GPS, *encoder*, giroscópio), sensores ambientais (que indicam temperatura, umidade), sensores inerciais (que indicam componentes diferenciais da posição do robô como, por exemplo, aceleração ou velocidade) (RIBEIRO, 2004).

O atuador assim como um sensor é um transdutor que transforma uma forma de energia em outra, porém este faz o caminho inverso, ele transforma um sinal elétrico em uma grandeza física, movimento, magnetismo, calor entre outros.

Os atuadores mais comuns na robótica são o Servo motor que tem baixo torque operando em malha fechada com um controle preciso da posição atual, e os motores de corrente contínua (CC), normalmente compactos com valor de torque constante para grandes variações de velocidade, porém necessitam de sensores de posição angular *(encode*r) ou de velocidade (tacômetro) para controle de posição.

2.4 REGRAS E COMPETIÇÕES DE ROBÔS SEGUIDORES DE LINHA

Entre as mais importantes competições de robótica do Brasil se destacam os eventos organizados pela RoboCore (2015), como a *WinterChallenge, SummerChallenge,* Arduíno*Day* SP. Entre as competições internacionais se destacam a *RobotChallenge* na Áustria, RoboCup (2015) e *RoboRave* que ocorrem em vários países.

Na Seção 2.4.1 são apresentadas as regras para competições organizadas pela RoboCore (2015) para categoria de robô seguidor de linha.

2.4.1 Especificações dos Robôs Seguidores de Linha

Os robôs seguidores de linha devem ser totalmente autônomos e com todos os componentes embarcados. Não pode ser controlado externamente por fio ou por rádio, com exceção para ser iniciado. Nenhuma adição, remoção ou alteração de hardware ou software pode ser feitas durante a competição. Porém pequenos reparos serão permitidos.

O robô não pode exceder 250 mm de comprimento, 250 mm de largura e 200 mm de altura. O Robô não poderá possuir nenhum tipo de mecanismo de sucção para aumentar a força normal em relação ao solo. Na Figura 10 é apresentado como exemplo o robô atual campeão mundial.

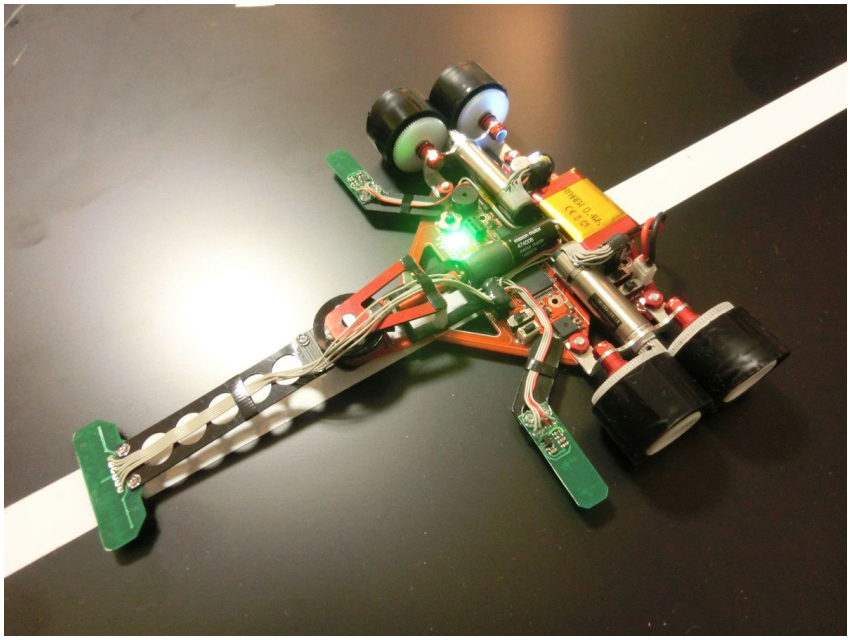

Figura 10 – Exemplo especificações robô seguidor de linha
Fonte: adaptado de (HIRAI, 2016).

2.4.2 Especificações do Percurso

O robô autônomo deve percorrer um circuito feito com uma ou mais placas de MDF revestidas com fórmica preta, tomando como referência uma linha branca de 19±1 mm de largura. O comprimento total da linha é de no máximo 60 metros. Vence o robô que finalizar o trajeto em menor tempo. O corpo do robô deve sempre ficar sobre a linha. Caso o robô saia completamente de cima da linha branca, será considerado que o robô saiu do percurso, assim sendo desclassificado. A linha

consiste em combinações de retas e arcos, podendo cruzar sobre ela mesma. Quando houver um cruzamento, o ângulo de intersecção das linhas será de 90±5°, conforme ilustrado na Figura 11. As partes das linhas que formam a intersecção serão retas 250 mm antes do cruzamento e 250 mm depois do cruzamento.

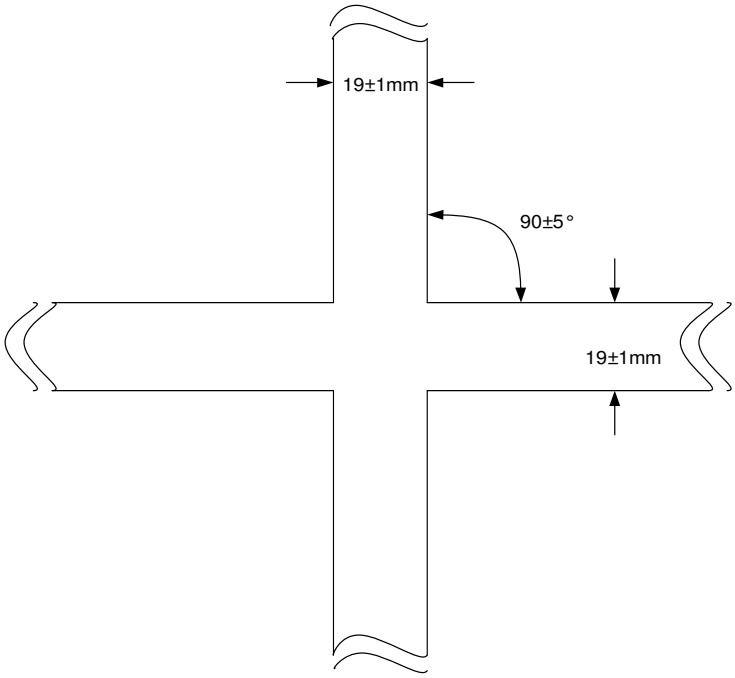

Figura 11 - Exemplo de cruzamento no percurso
Fonte: RoboCore (2015).

A área a qual se estende entre o ponto de partida e o ponto de chegada, considerando 200 mm a direta da linha e 200 mm a esquerda da linha com 1m de comprimento é denominada "área de partida-chegada", conforme ilustrado na Figura 12.

A linha de partida e a linha de chegada são localizadas em uma reta do percurso. A linha de chegada está localizada a um metro para traz da linha de partida. Há marcações no lado direito da linha (em relação ao sentido do percurso), indicando o ponto de partida e o ponto de chegada Figura 12. A linha 250 mm antes e 250 mm depois da "área de partida-chegada" será reta.

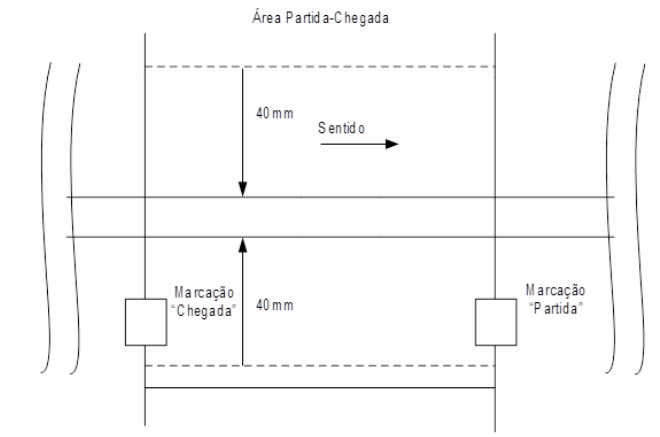

Figura 12 - Detalhes da área de partida-chegada
Fonte: RoboCore (2015).

O raio dos arcos é de pelo menos 100 mm. Cada arco possui um comprimento mínimo de 100 mm (Figura 13). Haverá uma marcação no lado esquerdo da linha (em relação ao sentido do percurso) no ponto em que houver alteração da curvatura.

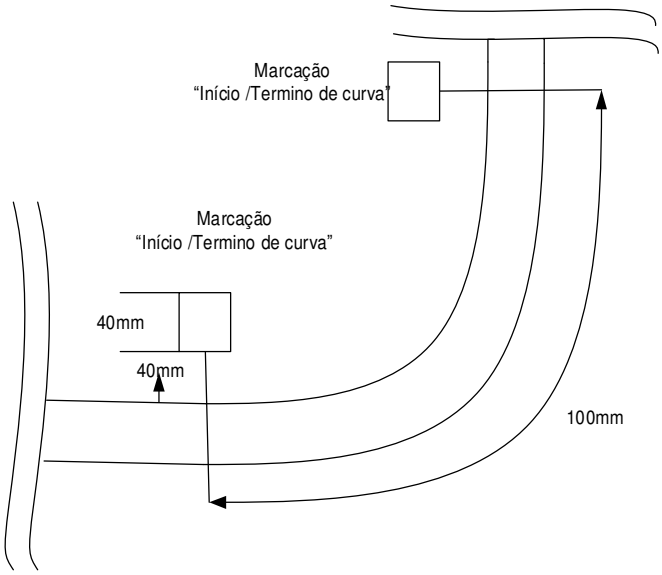

Figura 13 - Exemplo de curva no percurso
Fonte: RoboCore (2015).

3 MATERIAIS E MÉTODO

Na Seção 3.1 são apresentados os materiais e na Seção 3.2 é descrito o método utilizado no desenvolvimento deste trabalho.

3.1 MATERIAIS

Para o desenvolvimento da parte física do robô autônomo foram necessários os materiais listados nas seções 3.1.1 a 3.1.10. Na Seção 3.1.11 é apresentada uma plataforma robótica da Pololu, utilizada para testes e comparações dos sistemas de controle desenvolvidos.

3.1.1 Chassi

É a base do robô, estrutura de suporte para os demais componentes, normalmente feita de acrílico ou qualquer outro material rígido. Neste trabalho foi utilizada uma placa de fenolite com circuito impresso, assim os componentes são soldados diretamente no chassi (Figura 14).

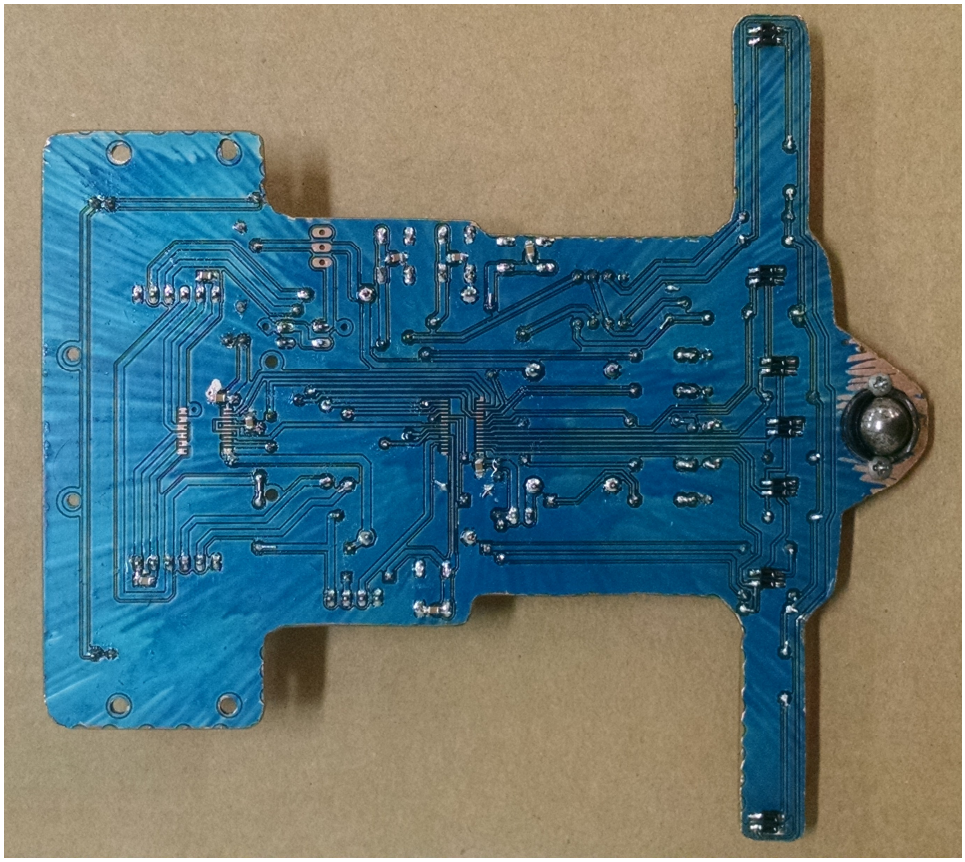

Figura 14- Placa de fenolite com circuito impresso usada como chassi do protótipo

3.1.2 Rodas

Para proporcionar uma boa tração e melhorar a estabilidade nas curvas são utilizadas rodas de poliuretano/silicone com 33mm de diâmetro e 20,10mm de largura (Figura 15).

Figura 15 - Rodas de poliuretano com silicone

3.1.3 Motor CC

Para o acionamento das rodas serão utilizados dois motores de corrente contínua (CC) com caixa de redução 30:1 - 6V- 1000rpm conforme Figura 16. As especificações do fabricante são apresentadas na Tabela 1.

Figura 16 - Motor corrente continua 6V -1000rpm

Tabela 1 - Especificações do motor de Carbono brushed da Pololu

Parâmetros	Valor
Tensão operação	6V
Corrente nominal	120mA sem carga
Corrente máxima	1,6A
Velocidade	1000rpm
Diâmetro do eixo	3 mm
Torque oz·in	9
Peso	10g

Fonte: Dados do fabricante.

Para controle de velocidade dos motores CC é necessário variar a tensão de alimentação. Este tipo de motor mantém o torque constante em todas as faixas de velocidade. No entanto a tensão de partida deve ser grande o suficiente para romper o coeficiente de atrito estático que é maior do que o coeficiente de atrito móvel. Sendo assim, é necessário um circuito elétrico de acionamento para dar a partida no motor CC em estado de repouso e manter as condições necessárias para o seu movimento.

3.1.4 Driver para motor CC TB6612FNG

Além do problema de partida é necessário controlar a velocidade e direção de rotação do motor CC. Há diversas formas de fazer esse acionamento e controle. Neste trabalho optou-se por um circuito integrado (CI), TB6612FNG, conforme ilustrado na Figura 17, que faz o acionamento de portas lógicas e duas pontes H (funcionamento da ponte H é apresentado na Tabela 3), baseadas em MOSFET muito mais eficientes que as pontes baseadas em BJT, pois permitem que mais corrente seja enviada aos motores e menos seja drenada para a alimentação lógica. As especificações do fabricante são apresentadas na Tabela 2.

A unidade de controle (microcontrolador) usa uma técnica de modulação por largura de pulso (PWM), a qual reduz o aquecimento dos componentes do driver, consequentemente também diminuem as perdas de energia, custo e tamanho.

Tabela 2 - Especificações do driver TB6612FNG

Parâmetros	Valor
Tensão motor	2,5V a 13,5V
Tesão Lógica	2,7V a 5V
Corrente máxima por canal	3A
Corrente continua por canal	1,2A
Frequência PWM	100kHz
Torque kg.cm	0.11
Peso	5g

Fonte: Dados do fabricante

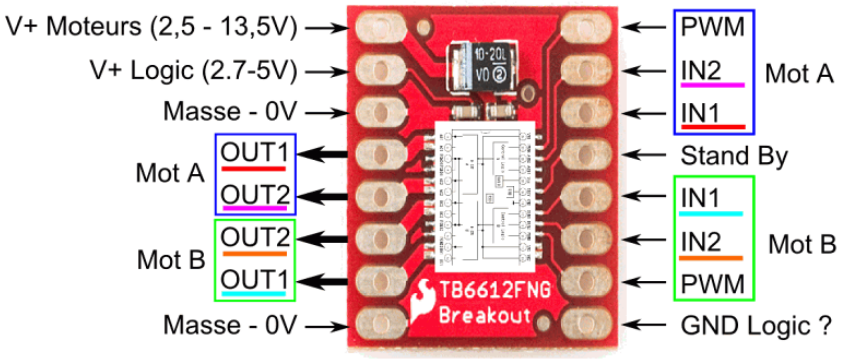

Figura 17 - CI TB6612FNG com ponte H dupla para acionamento de 2 motores CC
Fonte: Adaptado de Sparkfun, (2015).

Tabela 3 - Funcionamento da Ponte H

Motor A	IN1	IN2
Horário	VCC	GND
Anti-horário	GND	VCC
Ponto Morto	GND	GND
Freio	VCC	VCC

Fonte: Dados do fabricante.

3.1.5 Bateria Li-po

Para suprir a corrente necessária para os 2 motores, sensores, etc. é necessário uma bateria com boa relação entre carga/peso e corrente máxima de descarga. Assim optou-se por uma bateria (Figura 18) de duas células de Lítio-Polímero (Li-Po) que fornece uma tensão de 7,4V e 1300mAh de carga e 32,5A de corrente máxima de descarga.

Figura 18 - Bateria Li-po com 7,4V 1300mAh 25C

3.1.6 Módulo regulador de tensão XL6009 DC-DC *Step-Up Converter*

Este é um módulo regulador de tensão (Figura 19) de saída ajustável de alta eficiência (94%), capaz de dobrar a tensão de entrada da bateria de 7,4V fornecendo assim a tensão necessária para a potência máxima dos motores. Especificações do módulo são apresentadas na Tabela 4.

Tabela 4 - Módulo regulador de tensão *Step-up*

Tensão Entrada	5V-32V
Tensão Saída	5,5V a 35V
Tesão Lógica	2,7V a 5V
Corrente entrada máxima	4A
Corrente saída máxima	3A
Eficiência	94%
Dimensões	43mm x 21mm x 14mm
Peso	8g

Fonte: Dados do fabricante

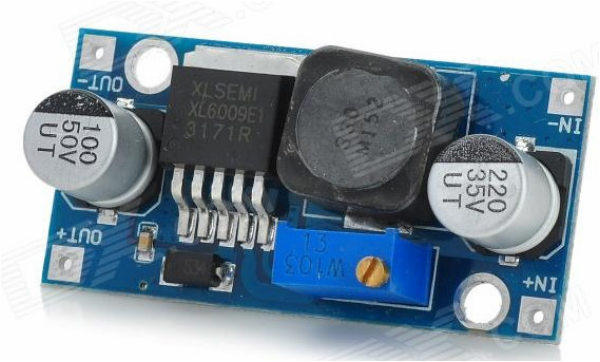

Figura 19 - Regulador de tensão Módulo XL6009 *DC-DC Step-Up*

3.1.7 Microcontrolador

A unidade de controle do sistema responsável por fazer todo o processamento é o microcontrolador MSP430G2553 28pinos fabricado pela *Texas Instrumens* (INSTRUMENTS, 2015), sua utilização é facilitada por um kit de desenvolvimento chamado *Launchpad* (Figura 20), integrando todos os componentes necessários para seu funcionamento, a um custo relativamente barato. As especificações estão na Tabela 5.

Tabela 5 - Microcontrolador MSP40G2553

Tensão Alimentação	1,8V-3.6V
Frequência operação	16MHz
Memória Flash	16KB
Memória SRAM	0,5KB
Portas entrada/saída	24
ADC	ADC10- 8ch

Fonte: Dados do fabricante.

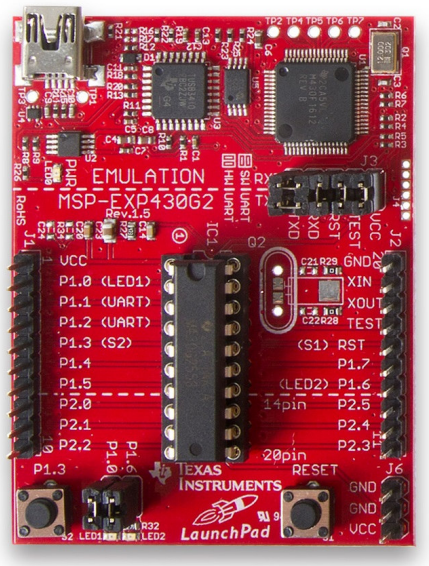

Figura 20 - Launchpad MSP430G

Fonte: Instrumens (2015).

3.1.8 Módulo regulador de tensão MP2259

Reguladores de tensão Step-down são componentes ativos os quais delimitam a tensão de saída caso a entrada passe do valor pré-estabelecido de fábrica. Para garantir alimentação segura do microcontrolador é usado o módulo regulador Xm1584 que garante uma saída de tensão de 3.6V e corrente 1A (Figura 21).

Figura 21 - Módulo regulador Step-down XM1584

3.1.9 Sensores de refletância

Um sensor de refletância consiste em um circuito eletrônico que seja capaz de detectar o quão reflexivo é a superfície através de um raio de luz incidente. Este

tipo de sensor possui um emissor que emite um raio de luz infravermelho em uma superfície perpendicular a este sendo refletido para um receptor (Figura 22).

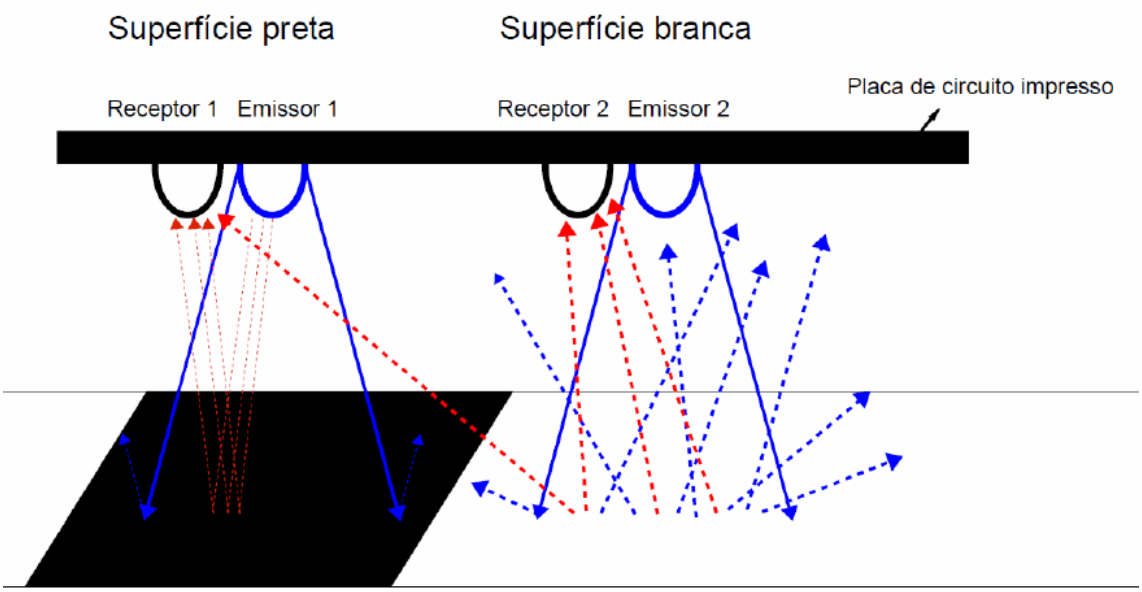

Figura 22 - Funcionamento sensores de refletância
Fonte: Meneguele, Ferreira e Arcanjo (2011, p.23).

Sabe-se que uma superfície preta absorve grande parte da luz incidente enquanto a superfície branca reflete a luz, dessa forma é fácil identificar o tipo de superfície em que o sensor se encontra.

Para a leitura da linha percorrida pelo robô foram testados diferentes tipos de estruturas e encapsulados de sensores infravermelhos (**Error! Not a valid bookmark self-reference.**) além de uma matriz de sensores de refletância QTR-8A (Figura 24), com um arranjo de 8 sensores infravermelhos (QRE1113) de alta precisão. Os pares de leds transmissores e emissores (fototransistores) estão divididos entre si por uma distância de 9,52mm. Cada fototransistor está conectado a um resistor de *pull-up* afim de formar um divisor de tensão que produz uma tensão analógica entre 0V e VIN (que, normalmente, é 3,6V). Tensão de saída baixa quer dizer grande reflexão.

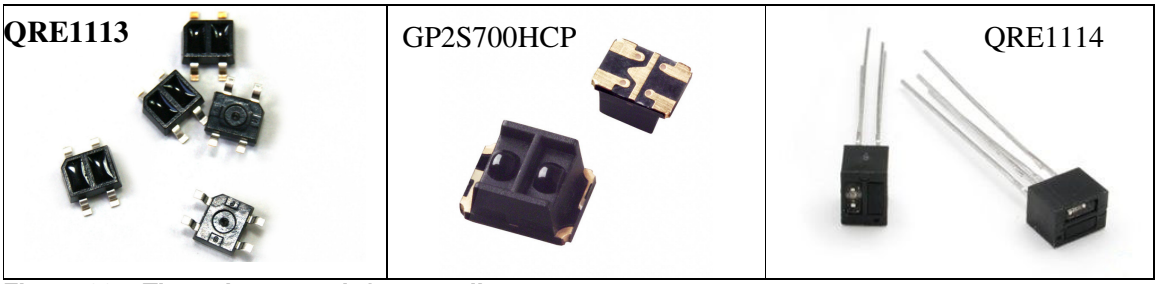

Figura 23 – Tipos de sensor infravermelho

Fonte: Adaptado de Sparkfun, (2015).

Todas as saídas analógicas são independentes, porém os LEDs estão colocados na placa em pares para diminuir pela metade o consumo de corrente. Os LEDs são controlados por um MOSFET com o *gate* normalmente colocado em nível lógico ALTO, permitindo desligar os LEDs colocando o *gate* em nível lógico BAIXO. Desligar os LEDs é vantajoso quando se quer economizar energia, ou variar o brilho efetivo dos LEDs utilizando o controle PWM.

A corrente dos LEDs é de aproximadamente 20 a 25mA, fazendo com que a placa consuma um total de 100mA.

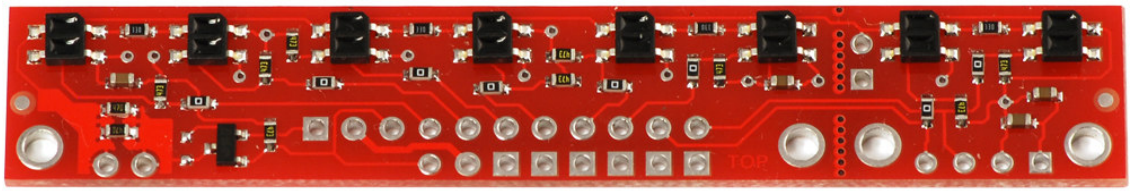

Figura 24- Matriz de sensores de refletância QTR-8ª

Tabela 6 - Especificações sensor refletância

Tensão Alimentação	3.3V-5V
Corrente drenada	100mA
Tensão de saída	Sinal Analógico
Distância leitura sensor	3mm a 6mm
Peso	3,09g

Fonte: Dados do fabricante Pololu.

Para as leituras dos eventos discretos de início e fim de curvas (do lado esquerdo) da pista e área de partida e de chegada (lado direito) são utilizados 2 módulos QTR-1A (Figura 25).

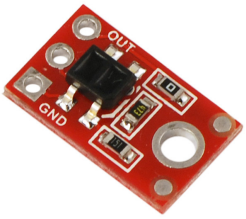

Figura 25 - Módulo QTR-1A para leitura de eventos discretos

3.1.10 Esfera Deslizante

Para sustentar a parte frontal do chassi (Figura 14) e manter os sensores de refletância a distância de 3 mm ideal para leitura da superfície foi usada uma esfera deslizante de metal como da (Figura 26).

Figura 26 - Esfera deslizante de metal

3.1.11 Robô 3pi Pololu

A plataforma robótica apresentada na Figura 27 é desenvolvida pela Pololu (2016) denominado 3pi devido seu pequeno diâmetro de 9,5 cm foi projetado para ser um hardware completo de fácil programação em C/C++ para competições de seguidor de linha e resolução de labirinto. Seu hardware é composto por 2 micromotores, 5 sensores de refletância, 1 display, 1 *buzzer*, 2 leds e 3 botões de usuário controlados pelo microcontrolador Atmega328 da Atmel.

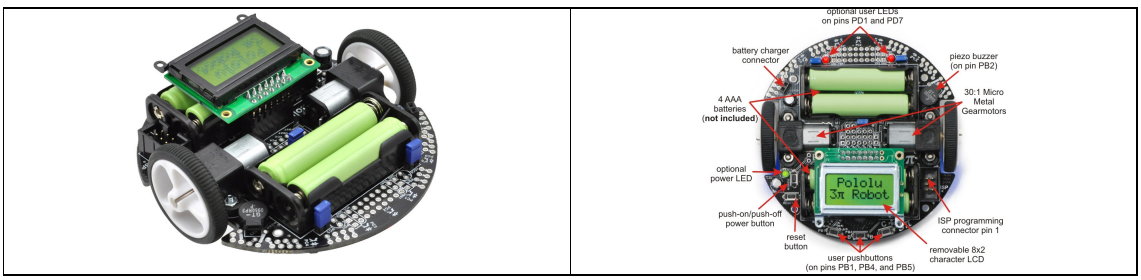

Figura 27 - Robô 3pi, desenvolvido pela Pololu

Fonte: Pololu (2016).

3.2 MÉTODO

A metodologia utilizada para realização desse trabalho iniciou com a definição do problema, seguido pela revisão bibliográfica pela qual foi definido o método de controle híbrido.

Com o sistema de controle e modelagem definida foram escolhidos e adquiridos os materiais para construção da parte física do robô autônomo.

Antes da codificação e implementação do sistema no microcontrolador uma modelagem experimental foi necessária. Os eventos discretos foram modelados através de autômatos de Moore com a utilização do software Supremica (2015). Com a utilização do software Deslab (software desenvolvido pelo professor coorientador Dr. César Rafael Claure Torrico), foi gerado o código em C sobre o autômato de Moore para implementação no microcontrolador. Para modelar o controle das variáveis continuas utilizando técnicas de lógica *fuzzy* foi utilizado o software Matlab, que dispõem de ferramentas de simulação com as quais é possível testar e ajustar o sistema antes de sua implantação no protótipo.

A etapa seguinte abrangeu a construção do protótipo do robô, modelagem do chassi, confecção de placas de circuito impresso, soldagem dos componentes, etc.

Após as simulações e confecção do hardware o sistema de controle híbrido, foi implementado no protótipo desenvolvido utilizando o controlador PID. Devido a pequena quantidade de memória RAM (0,5KB) do microcontrolador da Texas, não foi possível a implementação do controlador *Fuzzy* que necessita aproximadamente 1KB de memória RAM. Dessa forma foi necessário a utilização da plataforma robótica 3pi (figura 27) da Pololu para realização dos testes e comparações entre os controladores PID e *Fuzzy*.

4 IMPLEMENTAÇÃO E SIMULAÇÃO

A modelagem e simulação é um processo fundamental para implementação dos sistemas de controle deste trabalho, com eles é possível determinar o comportamento e desempenho de cada parte do sistema individualmente.

4.1 IMPLEMENTAÇÃO DO SISTEMA A EVENTOS DISCRETOS.

Para o controle e aquisição de sinais e eventos discretos foi modelada uma planta de acordo com as marcas da Figura 28 abaixo, circuito seguidor de linha utilizada no evento da *WinterChallenge* 2015 promovido pela RoboCore (2015).

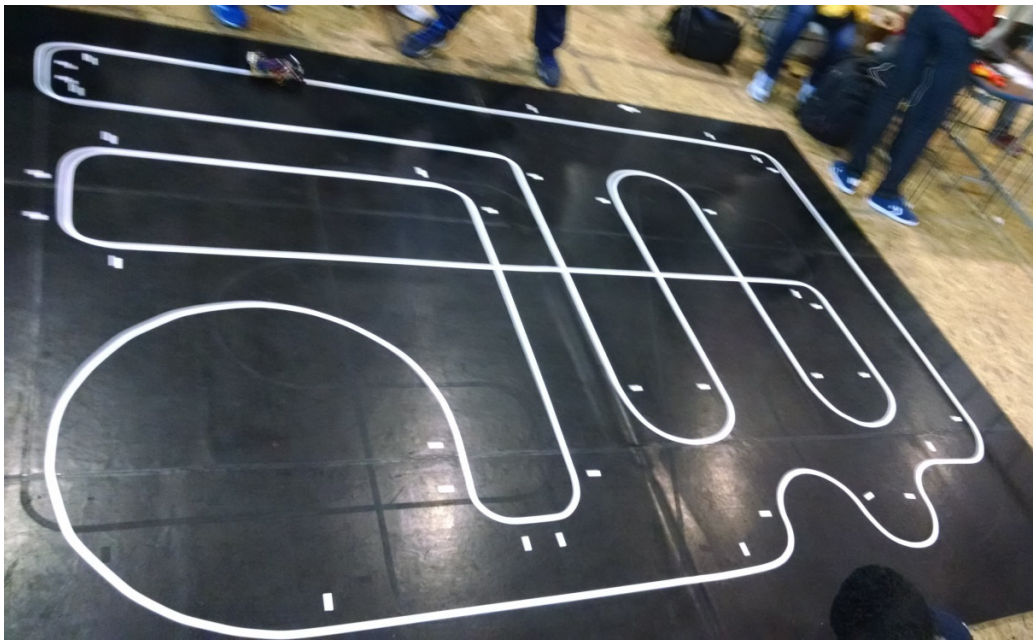

Figura 28 - Circuito seguidor de linha RoboCore 2015

O controlador SED foi modelado utilizando um autômato determinístico de estados finitos conhecido como autômato de Moore. O modelo apresentado na Figura 29 é genérico e modela qualquer tipo de planta de circuito seguidor de linha como o da Figura 28.

As variáveis a serem controladas são os dois sensores laterais (marcações do lado direito da pista são início/fim da pista, marcas do lado esquerdo são

início/fim de curvas) e um botão de partida e calibração dos sensores. Primeiramente foi realizado um levantamento do processo para depois fazer a sua implementação no software Supremica.

Este modelo do processo discreto foi projetado visando atender as seguintes especificações:

• Uma chave táctil é pressionada para calibrar os sensores e dar início à corrida (robô encontra-se parado no seu estado inicial);

• O robô fica parado esperando novo evento da chave táctil após detectar a marca de chegada;

• Nas retas o robô anda com 100% da velocidade máxima;

• Nas curvas o robô reduz para 80% da velocidade máxima;

• O número de curvas do trajeto varia de uma competição para outra, ou seja, o número de curvas não é conhecido;

• O número de cruzamentos do trajeto também varia de uma competição para outra.

Para implementação do modelo foi adotado a máquina de Moore, em que as ações de saída (valor de operação do PWM) estão nos estados. Os eventos adotados para esse processo são definidos como:

- ED: Encontra marca da direita.
- PD: Perde marca da direita.
- EE: Encontra marca da esquerda.
- PE: Perde marca da esquerda.
- Aperta_bot: Botão de calibragem sensores e iniciar corrida.

Os valores dentro dos estados correspondem à velocidade do robô, sendo nas retas (100%), curvas (80%) e robô parado (estado inicial) (0%). Este valor de velocidade deverá ser ajustado para que o robô percorra a linha o mais rápido possível sem sair do trajeto.

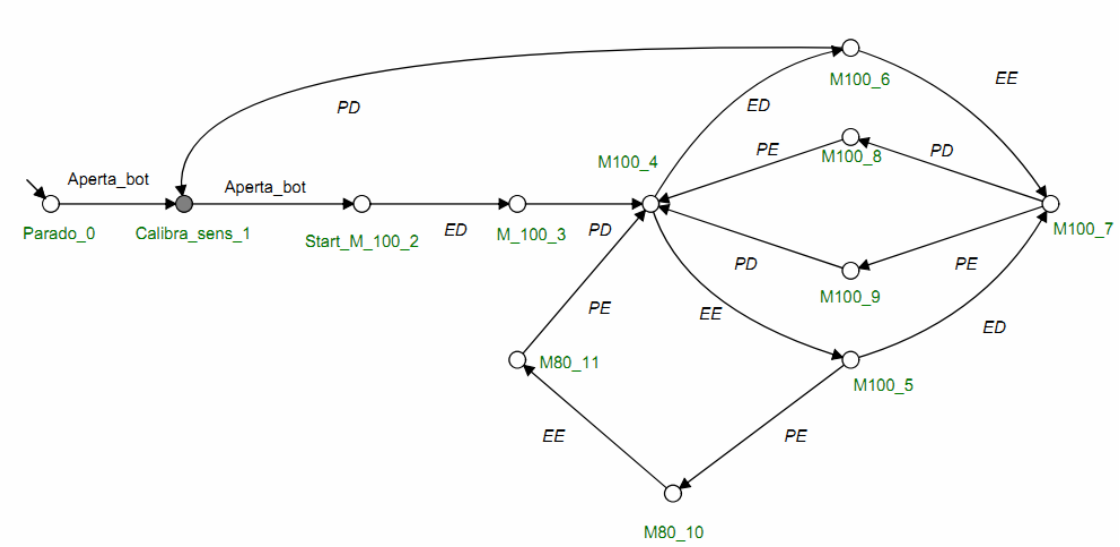

Figura 29 - Autômato de Moore para controle da planta modelada com software Supremica

Na sequência é realizada uma análise de cada estado do robô:

- Estado 0: Robô ligado, esperando o *pushbutton* ser pressionado para iniciar a calibragem e permanecer parado.
- Estado 1: Robô ligado e parado, esperando o *pushbutton* ser pressionado para iniciar o percurso.
- Estado 2: Robô está percorrendo a área de chegada e partida com os motores no ponto de operação para retas (PWM=100%), próximo evento será achar a marca da direita (ED).
- Estado 3: Encontrou a linha branca de início de percurso, agora ele irá "perder" a linha (PD).
- Estado 4: Continua na reta e espera o próximo evento que pode ser encontrar a marca da esquerda, como também achar a marca da direita.
- Estado 5: Encontrou a marca da esquerda, e espera o próximo evento, que pode ser perder a esquerda, que significa que vai entrar na curva, ou achar a direita, que significa que é um cruzamento.
- Estado 10: Perde a marca da esquerda, então ele entrou na curva, muda o ponto de operação (PWM=80%). Espera o próximo evento que é achar a marca da esquerda, que diz que vai terminar a curva.
- Estado 11: Achou a marca da esquerda, está se alinhando para entrar novamente na reta, por isso o valor do PWM não muda ainda. Só irá

mudar para o ponto de operação depois que perder a marca da esquerda e retornar para o estado 4 (PWM=100%).

- Estado 7: Achou a marca da direita após ter achado a marca da esquerda, está em um cruzamento, ponto de operação não muda, espera perder a marca da direita ou da esquerda para mudar de estado.
- Estado 8 e Estado 9: Perdeu a marca da direita ou esquerda, está deixando o cruzamento, irá perder novamente uma das duas marcas e ira retornar ao estado 4, não muda o ponto de operação.

Observação: As transições entre o estado 7- 8 ou 7- 9 e depois o 4 ocorrem da seguinte maneira, quando o robô ira deixar o cruzamento ele irá perder primeiro uma faixa e depois a outra (Exemplo: do estado 7ele perde a direita vai para o 8, em seguida perde o da esquerda, vai para o estado 4, ou seja, saiu do cruzamento).

Estado 6: Achou a marca da direita, e em seguida achar a marca da esquerda (vai para estado 7), que diz que entrou em um cruzamento, ou perder a direita, que significa que chegou a faixa de fim de percurso, retornando para o estado 1 (espera de novo evento do Botão).

4.2 IMPLEMENTAÇÃO DO CONTROLADOR FUZZY

Para implementar o controlador *fuzzy* foi necessário observar as entradas dos cinco sensores de refletância instalados na parte frontal do robô. Cada sensor retorna um valor de leitura analógico que após passar pelo conversor A/D (analógico/digital) do microcontrolador é calibrado separadamente para uma faixa de valores entre 0 e 1000. Na calibração todos os sensores são submetidos a inúmeras leituras da linha branca e superfície preta durante 5 segundos armazenando o valor máximo (max [ ]) e mínimo (min [ ]) de cada sensor. Utilizando a Eq. (6), onde A[ ] é o valor lido e B[ ] é o valor calibrado, a diferença entre os sensores é contabilizada automaticamente (Figura 34, Figura 35).

$$B[i] = (((A[i]- min[i]) *1000) / (max[i]- min[i])) \qquad \text{Eq.(6)}$$

A partir dessas variáveis contínuas obtemos a posição do robô sobre a linha que forma o circuito percorrido. Na Figura 30 são apresentadas todas as possíveis posições de distância entre o centro do robô e a linha.

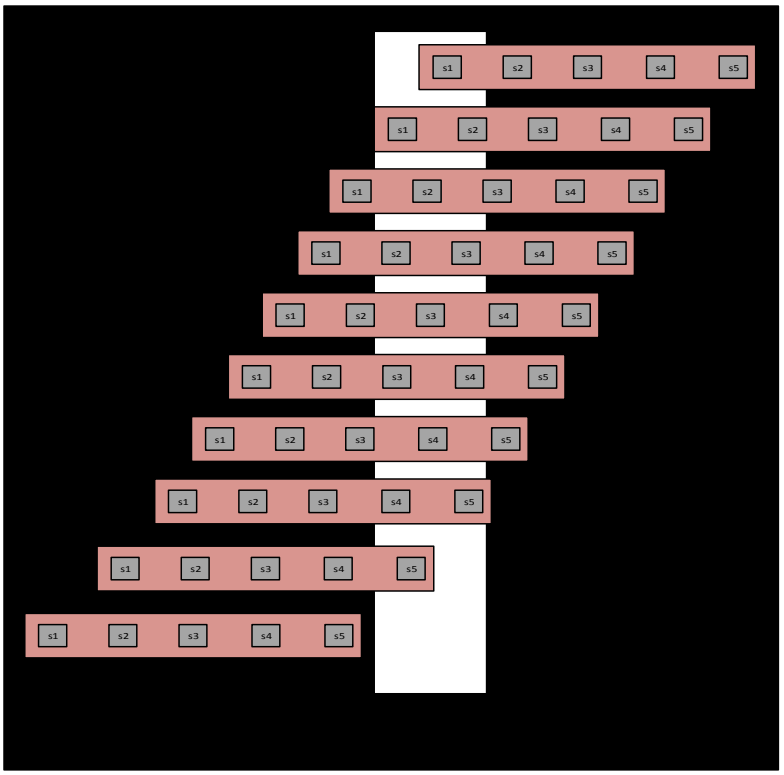

Figura 30 - Posições de distância entre o centro do robô e a linha

Os valores de leitura dos sensores entre 0 e 1000 fornecem um grau de refletância que no processo de *fuzzyficação* é associado aos estados *fuzzy* (“baixo”, “médio” e “alto”) de acordo com o grau de pertinência conforme visto na Figura 31.

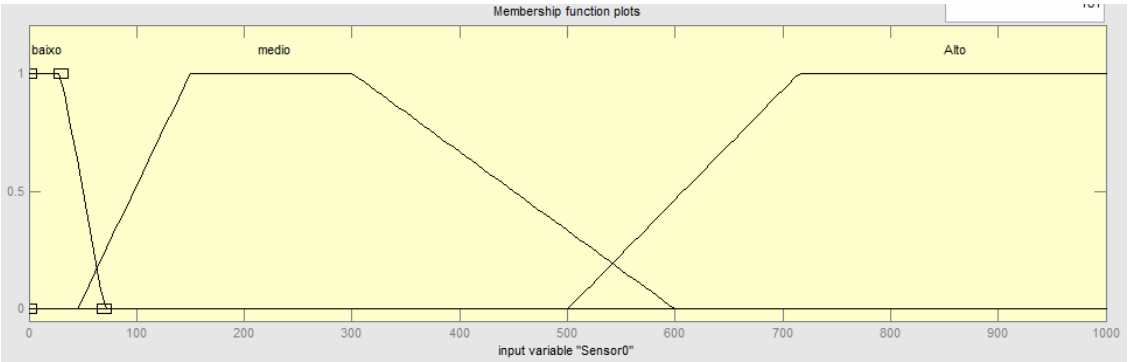

Figura 31 - Variáveis *fuzzy* utilizadas para calcular o grau de pertinência no software Matlab

O próximo passo foi a construção de um sistema de inferência na forma de regras de controle baseadas no conhecimento e expectativa do projetista sendo que cada uma delas demanda uma ação de controle da potência fornecida aos dois motores de modo a manter o centro do robô sobre a faixa na maior velocidade

possível. A Tabela 7 – Base de regras para construção do sistema de inferênciaTabela 7 a seguir apresenta as regras do controlador baseadas na posição do robô sobre a faixa conforme a Figura 30 acima.

Tabela 7 – Base de regras para construção do sistema de inferência

Posição dos sensores na faixa		Velocidade Motor Esquerdo	Velocidade motor Direito
1	Direita_4	Parado	Rápido
2	Direita_3	Devagar	Rápido
3	Direita_2	Devagar	Rápido
4	Direita_1	Médio	Rápido
5	Centro	Rápido	Rápido
6	Esquerda_1	Rápido	Médio
7	Esquerda_2	Rápido	Devagar
8	Esquerda_3	Rápido	Devagar
9	Esquerda_4	Rápido	Parado
10	Fora da faixa	Parado	Parado

Por último foi realizado o processo de *defuzzyficação* utilizando o método de centroide que gerou duas saídas (sinais PWM com valores de 0 a 255 que controlam a potência do motor) de acordo com o grau de importância para cada um dos estados de velocidade dos motores (parado, devagar, médio e rápido) apresentado na Figura 32.

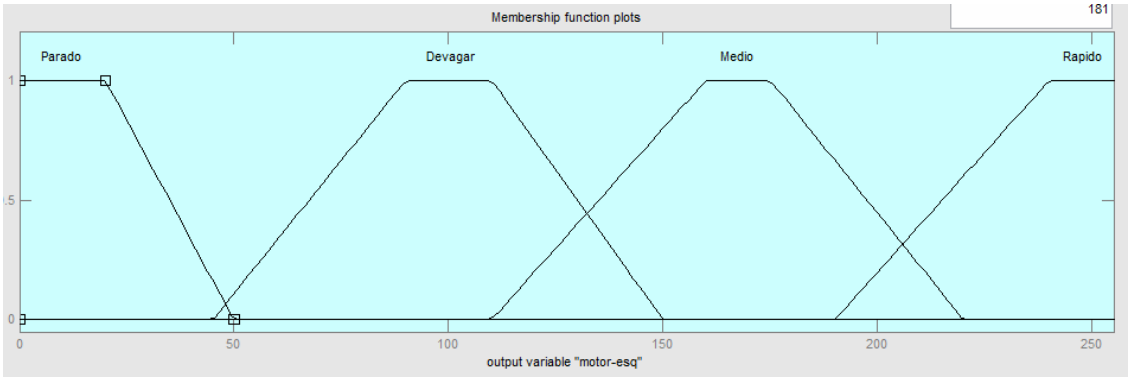

Figura 32 - Variáveis fuzzy usadas para o cálculo do sinal de saída PWM no software Matlab

De modo a facilitar a modelagem e implementação do sistema de controle *fuzzy* foi utilizado a ferramenta de Simulação do software Matlab com a qual foi gerada um código em C++ que facilitou a implementação no microcontrolador. A Figura 33 mostra a ativação das regras de inferência de acordo com o grau de

pertinência das cinco entradas dos sensores que após o processo de *defuzzyficação* retornam dois sinais PWM aplicados aos motores para controle da velocidade.

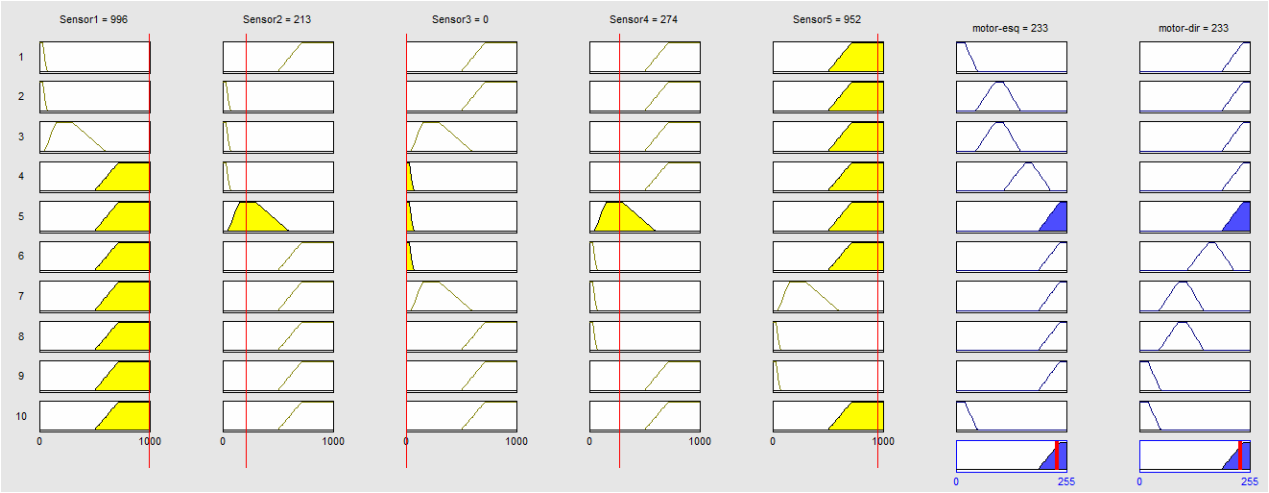

Figura 33 - Simulação do sistema de controle no Matlab

4.3 IMPLEMENTAÇÃO DO CONTROLADOR PID

Na implementação do controlador PID foi utilizado um sistema de malha fechada (o valor de saída depende do valor de entrada desejado e do valor atual de saída) conforme o modelo apresentado na Figura 9 que oferece uma saída estável com erro mínimo ajustando os 3 parâmetros:

Kp é a constante que fornece uma resposta rápida no compensador para qualquer alteração do sinal de entrada;

Ki é responsável para corrigir o erro no estado estacionário;

Kp é a constante que antecipa a correção do valor de saída melhorando o transitório e percentual de ultrapassagem (sinal de saída acima do desejado).

A Eq. (5) faz a combinação dessas três constantes de forma que o compensador PID em conjunto com a planta forneçam a resposta adequada para qualquer variação na entrada.

Analogamente ao controlador *fuzzy*, a posição do robô sobre a linha é calculada através dos sinais das leituras dos sensores após passarem pelo conversor A/D e calibração. Usando a Eq. (7) uma estimativa de posição do robô é calculada fazendo a média ponderada dos índices de sensores multiplicados por

1000, de modo que um valor de retorno de 0 indica que a linha está diretamente abaixo do sensor S0, um valor de retorno 1000 indica que a linha está diretamente abaixo do sensor S1, o valor 2000 indica que ele está abaixo do sensor S2, etc. Valores intermediários indicam que a linha está entre dois sensores.

$$\frac{0 * value0 + 1000 * value1 + 2000 * value2 + \ldots}{value0 + value1 + value2 + \ldots}$$

Eq. (7)

Para cinco sensores com o centro do robô sobre a linha temos: $\frac{0 * 0 + 500 * 1000 + 1000 * 2000 + 500 * 3000 + 0 * 4000}{0 + 500 + 1000 + 500 + 0} = 2000$

Eq. (8)

O termo proporcional é computado pela posição estimada com a Eq. (7) subtraindo a referência dada pela Eq. (8)

O termo derivativo é obtido fazendo uma subtração do valor atual do proporcional pelo seu anterior. Essa subtração é feita com espaçamentos de tempo de um ciclo do programa (leitura sensores, calibração, cálculo da posição, atuação nos motores) com frequência igual a 4,6KHz, resultando em instantes de tempo conhecidos e próximos a zero.

O termo integral é calculado pelo mesmo princípio numérico do derivativo de forma inversa, o valor integral é atualizado em cada ciclo de programa somando o proporcional passado.

O sinal de erro, que representa a diferença entre a entrada e saída é obtido fazendo o somatório dos termos do PID multiplicados pelas suas respectivas constantes conforme Eq. (9).

$$pwm = (proporcional * Kp) + (derivativo * Kd) + (integral * Ki) \qquad Eq.(9)$$

A listagem de código a seguir apresenta o controlador PID implementado no microcontrolador do protótipo desenvolvido.

```
void pid_control(void)
{
        WDTCTL = WDTPW + WDTHOLD;
        float Kp = 1.43, Ki = 1.0e-3, Kd = 44; // constantes do PID
        int proporcional = (posicao) - 2000;       // set point para 5 sensores = 2000
        integral = integral + proporcional_passado;        //obtendo a integral
        int derivativo = (proporcional - proporcional_passado); //obtendo o derivativo
        int pwm =( proporcional * Kp ) + ( derivativo * Kd )+(integral*Ki);//Calculo do erro
        if (PWM < 0)     //robô se encontra a esquerda da faixa
        {
```

```
                TA1CCR1 = velocidade+PWM; //diminui o PWM do motor direito
                TA1CCR2 = velocidade;           //motor esquerdo mantem a referencia
        }
        if (PWM > 0) //robô se encontra a direita da faixa
        {
                TA1CCR1 = velocidade;           //motor direito mantem a referencia
                TA1CCR2 = velocidade-PWM; //diminui o PWM do motor esquerdo
                }
        proporcional_passado = proporcional;
}
```

A Figura 34 e Figura 35 a seguir mostram um exemplo de leitura, calibração dos sensores utilizando as Equações (6,7) quando o centro do robô está sobre a linha.

Expression	Type	Value
max	unsigned int[5]	0x020A
[0]	unsigned int	778
[1]	unsigned int	800
[2]	unsigned int	816
[3]	unsigned int	833
[4]	unsigned int	757
min	unsigned int[5]	0x0214
[0]	unsigned int	24
[1]	unsigned int	24
[2]	unsigned int	25
[3]	unsigned int	26
[4]	unsigned int	28
posicao	unsigned long	2001

Figura 34 - Calibração armazenado valor Max e Min de cada sensor

Fonte: Ferramenta de codificação Code Composer Studio.

Expression	Type	Value
B	unsigned long[5]	0x03C6
[0]	unsigned long	19
[1]	unsigned long	27
[2]	unsigned long	983
[3]	unsigned long	27
[4]	unsigned long	12

Figura 35 - Leitura de linha com sensor[2] sobre a linha branca e demais sensores sobre a superfície preta

Fonte: Ferramenta de codificação Code Composer Studio.

5 EXPERIMENTOS E RESULTADOS

Durante o desenvolvimento deste trabalho foram confeccionados três protótipos para testar os sensores e sistemas de controle desenvolvidos. Além disso foi usada a plataforma robótica 3PI da Pololu para comparação do controle *Fuzzy* com o controlador PID.

5.1 TESTES SENSORES E PROTÓTIPOS

Devido à facilidade de implementação e uso de bibliotecas prontas para testes dos sensores a primeira versão apresentada na Figura 36 utiliza uma placa de desenvolvimento Arduíno Nano com microcontrolador Atmega328p e oscilador de 16MHz.

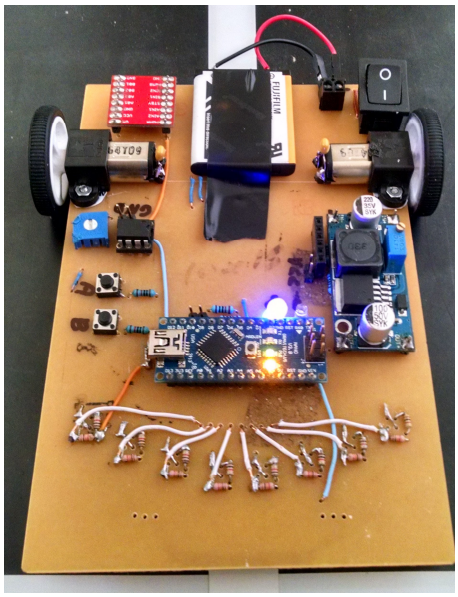

Figura 36 - Versão 1 do protótipo desenvolvido

Para guiar o robô autônomo sobre a linha é necessária uma leitura de posição muito precisa do protótipo sobre a linha. Para isto foram testados diferentes tipos de estruturas e encapsulados de sensores de refletância apresentados na Seção 3.1.9. Na primeira versão do protótipo foram utilizados 8 sensores QRE1114 conforme a Figura 37, essa configuração teve problemas ao percorrer as curvas do circuito, pois os sensores mais externos eram afetados pelas marcações de início e

final de curva, causando instabilidade e a perda da linha. Além disso um número maior de sensores afeta o tempo de processamento (leitura, calibração, cálculo de posição e atuação nos motores) que ficou em 210 ciclos de programa por segundo.

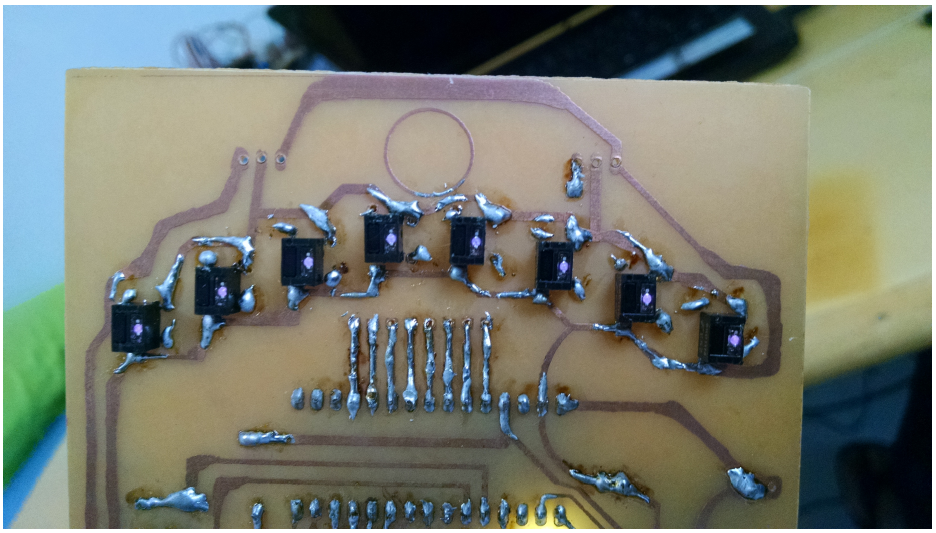

Figura 37 - Configuração de sensores QRE1114 utilizados na primeira versão do robô

Essa versão do protótipo foi testada com o controle PID em pistas oficiais da RoboCore (2015) na Feira de ciências *FACE2015* onde obteve o 3º Lugar e na *SummerChalenge* obtendo a 11ª colocação do placar geral percorrendo a linha numa velocidade média de 0,85 cm/s.

Já na segunda versão (Figura 38) foi utilizado o microcontrolador MSP430G2553 da Texas com 5 sensores SMD GP2S700HCP conforme Figura 39.

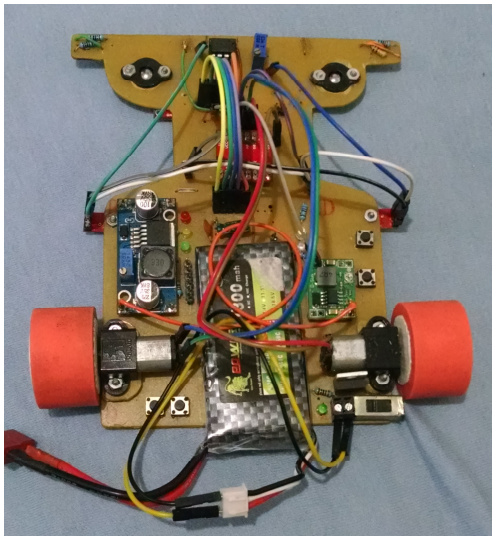

Figura 38 - Versão 2 do protótipo desenvolvido

Está configuração melhorou significativamente o rastreamento da linha, porém um erro de projeto implicou na ressolda de resistores perto dos componentes que ficaram avariados devido ao aquecimento, sendo substituídos pela matriz QTR-8A da Figura 24 que teve um ótimo desempenho usando 5 sensores.

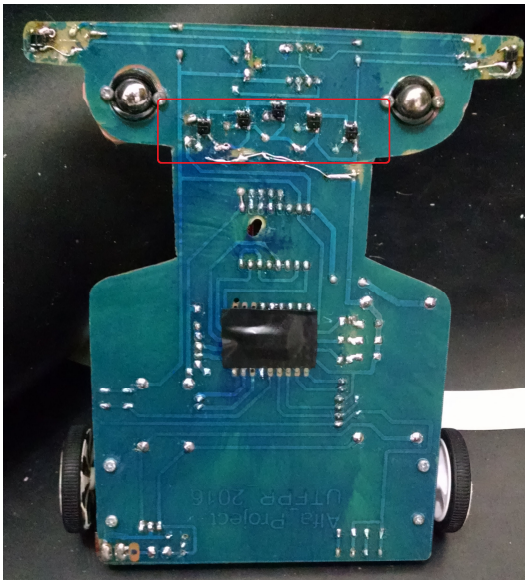

Figura 39 - Configuração sensores smd GP2S700HCP

Apesar do microcontrolador da Texas também trabalhar na frequência de 16MHz sua arquitetura permite a leitura e conversão de todos os sensores em paralelo com poucos ciclos de *clock*, obtendo a posição do robô sobre a linha 4600

vezes por segundo. Com o controle PID esse protótipo consegue seguir a linha em velocidade média 1,05m/s podendo acelerar na potência máxima nas retas.

Uma última versão do protótipo (Figura 40) foi confeccionada usando os sensores QRE1113 conforme apresentado na Figura 14, porém está versão não foi testada devido problemas no circuito, impossibilitando a codificação do microcontrolador MSP430G2553 da Texas.

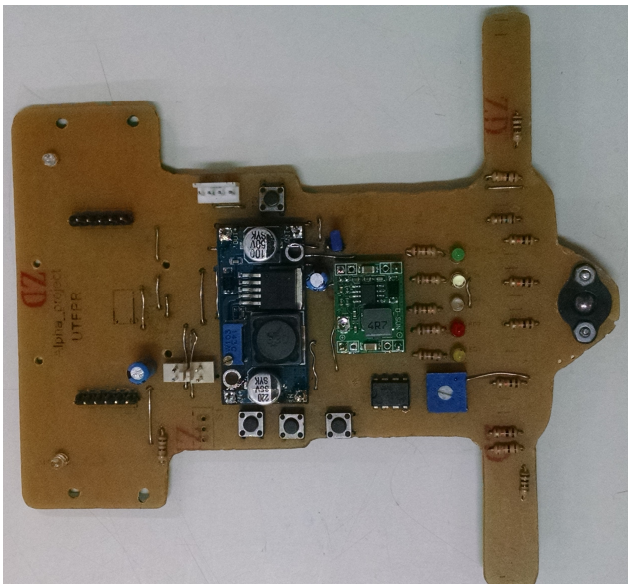

Figura 40 - Versão 3 do protótipo desenvolvido

De forma geral todos os sensores testados apresentaram ótima precisão na leitura da linha para uma distância focal de 2mm a 5mm. A escolha de um ou outro tipo de encapsulado não afetou a leitura da linha, mas a disposição e número de sensores utilizados.

5.2 TESTES DO CONTROLADOR A EVENTOS DISCRETOS

O controle do sistema a eventos discretos foi implementado na segunda versão do protótipo (Figura 38). Conforme mencionado na Seção 4.1 as variáveis discretas são provenientes de uma chave táctil e dois sensores laterais.

A chave táctil é ligada entre o Vcc (3,6V) e microcontrolador, ao ser pressionada gera uma borda de subida (nível lógico alto) dando início ao processo de calibração dos sensores, pressionado mais uma vez o robô inicia o percurso da

linha.

Ao detectar as marcas de sinalização na pista os sensores laterais apresentam o sinal em azul (3,68V no preto e 1,92V ao passar pela faixa branca) da Figura 41. Esse sinal foi tratado por um circuito comparador para gerar os eventos discretos (nível lógico alto (2,48V) e baixo 0,4V) em amarelo.

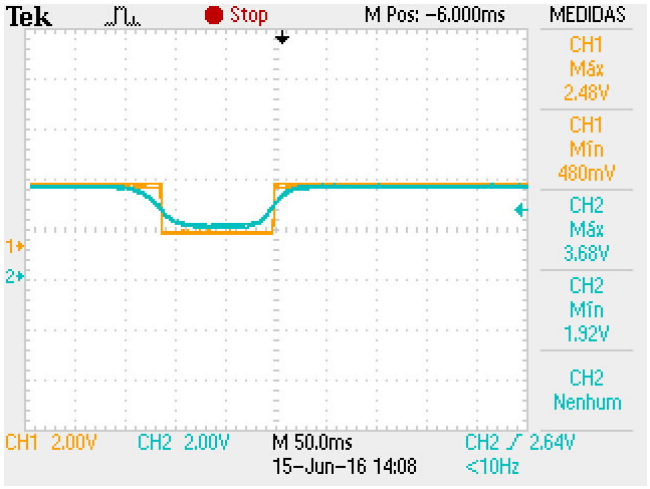

Figura 41 - Condicionamento do sinal do sensor lateral

O circuito comparador apresentado na Figura 42 foi implementado com um amplificador operacional LM358N e um potenciômetro para ajuste da tensão de referência em 3V. A saída é ligada ao microcontrolador que gera uma interrupção toda vez que uma borda (subida ou descida) é detectada.

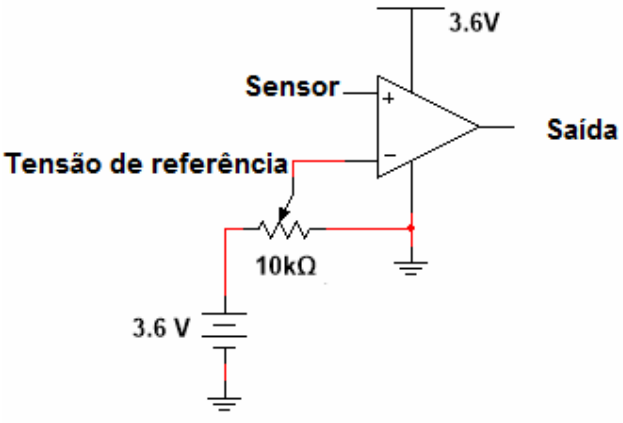

Figura 42 - Circuito comparador para tratamento de sinal dos sensores laterais

Um teste prático foi realizado na pista de testes, onde o robô conseguiu identificar todas as marcações de curvas, cruzamentos e sinal de início e fim de

percurso comprovando o perfeito funcionamento do controle discreto.

5.3 COMPARAÇÃO CONTROLADOR PID VERSUS FUZZY

Devido às restrições de memória do microcontrolador MSP430G2553 da Texas Instruments para implementação do controlador *Fuzzy*, foi utilizado a plataforma robótica 3PI da Pololu com algumas modificações (Figura 43) da versão original para testes de comparação entre os controles desenvolvidos.

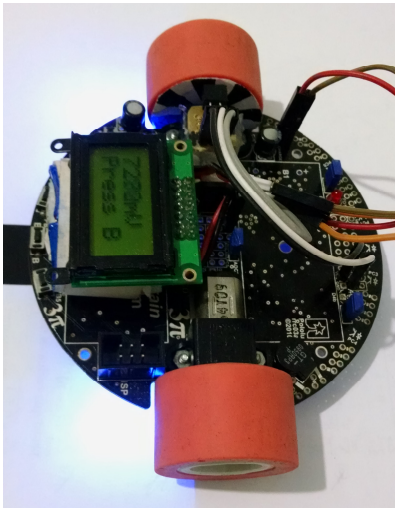

Figura 43 - Versão modificada do 3PI da Pololu

Com objetivo de aumentar a velocidade máxima os motores de baixa potência foram substituídos por outros de alta potência, que necessitam de uma tensão maior, para isto foram usadas duas baterias em série que geram 7,4V com 12A de descarga continua. Nessa configuração o robô pode alcançar uma velocidade de 2m/s, afim de reduzir a derrapagem nas curvas as rodas foram substituídas por outras mais largas de alta aderência. Devido ao formato reduzido não é possível a utilização de sensores para leitura das marcações de eventos discretos, assim foi acrescentado um *Encoder* (disco com 8 faixas pretas e 8 faixas brancas que refletem a luz de sensores infravermelhos retornando uma sequência de pulsos com os quais é possível se obter a posição e velocidade) dispositivo capaz de mapear a pista e prever as curvas diminuindo a velocidade máxima antes de chegar nelas além de parar no ponto especificado.

Para comparação dos resultados os dois sistemas de controle PID e *FUZZY*

foram implementados sem o controle à eventos discretos no protótipo 3PI da Figura 43 devido a limitações de hardware do protótipo da Pololu. Os testes são baseados no sinal PWM enviado aos motores pelo controlador que compensa o erro de acordo com o sinal de referência dada pela posição do robô sobre a linha. Os dois sistemas de controle apresentaram resultados idênticos para testes em bancada conforme apresentado a seguir.

A Figura 44 apresenta o ciclo de trabalho dos motores operando em 50% da potência quando o robô se encontra na referência, ou seja, centro do robô sobre a linha, a velocidade dos motores é igual.

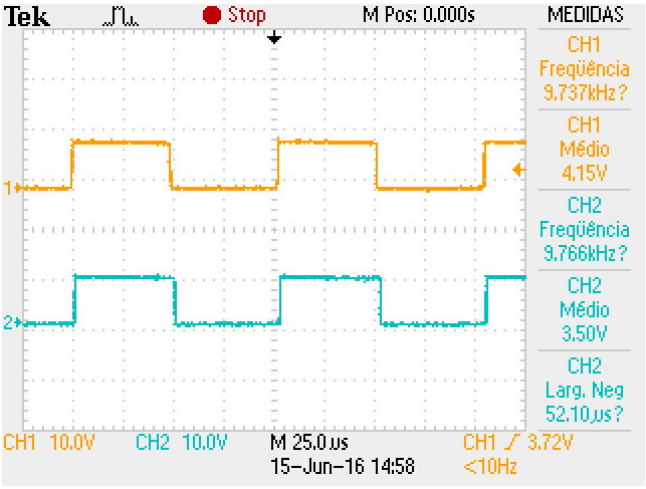

Figura 44 - Sinais PWM enviados aos motores quando o robô se encontra na referência

Quando o robô se afasta da referência para o lado esquerdo o sistema de controle tenta compensar o erro diminuindo a potência do motor direito (sinal azul) da Figura 45 mantendo velocidade máxima do motor esquerdo (sinal laranja). Quando o robô se afasta da referência pelo lado direito, acontece o inverso, ou seja, a potência do motor esquerdo é diminuída até que o robô volte para a referência.

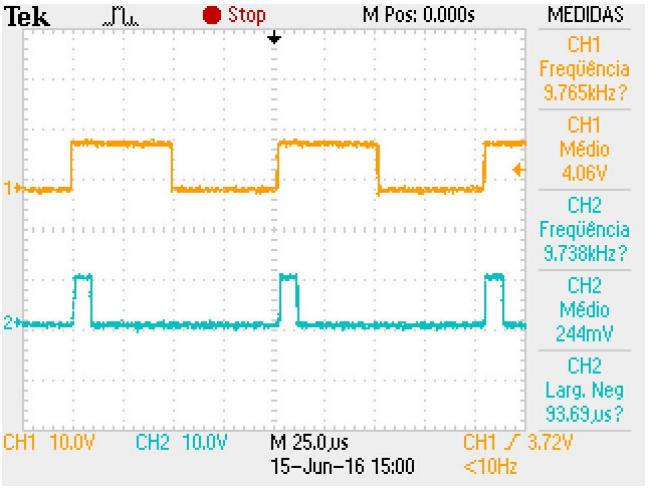

Figura 45 - Sinas PWM enviados aos motores quando o robô está fora da referência

Também foi adquirido a frequência de operação de cada controlador conforme mostrado na Figura 46. O processo de leitura e calibração dos sensores é igual nos dois sistemas de controle, mudando o cálculo da posição e atuação dos motores. Pode-se observar que o controlador PID consegue fazer aproximadamente 680 leituras e atuações nos motores, enquanto o controlador *FUZZY* realiza apenas 385 leituras e atuações nos motores por segundo.

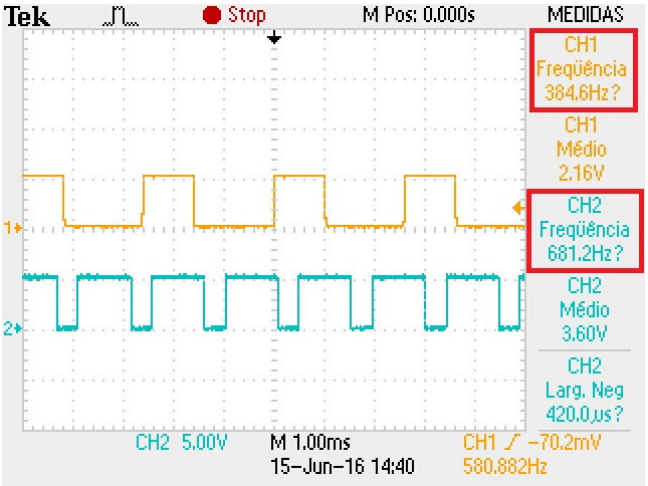

Figura 46 – Frequência de operação dos controladores: Fuzzy em laranja, PID em azul

Após os testes em bancada foram realizados testes práticos na pista seguindo as normas da RoboCore.

O sistema de controle PID foi testado exaustivamente durante semanas na busca das constantes *Kp, Kd* e *Ki* mais apropriadas da Equação 9. (Na primeira versão do protótipo foi utilizado um dispositivo de comunicação bluetooth, a partir do qual foi possível enviar os parâmetros de controle em tempo real). Como resultado o robô conseguiu seguir a linha perfeitamente com velocidade constante aproximada de 1,1m/s podendo acelerar ainda mais nas retas. A versão anterior do robô 3PI foi utilizada com o mesmo controle PID em competições da RoboCore, tendo como resultado 13º lugar na *Winter Challenge* em São Paulo, 1º Lugar na Feira de Ciências *FACE2015* em Santa Catarina e 2º Lugar na *Summer Challenge* em Minas Gerais.

O sistema de controle *Fuzzy* não foi capaz de seguir a linha durante os testes práticos, na tentativa de corrigir a instabilidade do robô durante o percurso da linha, quatro regras *Fuzzy* foram adicionadas ao motor de inferência do controlador. Assim o robô ficou mais estável, porém a frequência de operação apresentada na Figura 46 foi reduzida drasticamente, deixando o controlador muito lento para compensar o erro antes de perder a linha totalmente.

6 CONCLUSÃO

O objetivo deste trabalho foi desenvolver um sistema de controle hibrido utilizando sinais e eventos discretos para comparação de um controlador clássico usando PID e técnicas de inteligência artificial, a lógica *fuzzy*, aplicados a um robô autônomo seguidor de linha.

Na revisão bibliográfica foram vistas diferentes técnicas de controle para navegação autônoma de robôs moveis além de estudos multidisciplinares que subsidiaram os conhecimentos necessários para alcançar os objetivos propostos.

O controle PID foi implementado utilizando modelos matemáticos e formulações encontradas no levantamento bibliográfico. A possibilidade de enviar os parâmetros do controlador em tempo real com um sistema de comunicação *bluetooth* facilitaram bastante os testes. Muitos ajustes foram necessários até uma boa estabilidade do sistema.

A modelagem do sistema de controle *Fuzzy* na ferramenta de simulação do Matlab facilitou os ajustes do sistema além de gerar o código de implementação do microcontrolador. Porém as regras e parâmetro não podem ser ajustadas em tempo real como no PID tornando esse processo bastante trabalhoso. Os resultados da simulação demostram que a resposta para controle do robô está coerente, já nos testes práticos o resultado esperado não foi satisfeito.

Comparando os sistemas de controle desenvolvidos verificou se que o *Fuzzy* necessita de um processador bem mais eficiente comparado ao PID, com memória suficiente para implementar as variáveis de controle, além de maior poder computacional para processamento matemático, possibilitando o sistema de controle compensar o erro em tempo hábil.

A implementação do controle hibrido realizado a partir da modelagem dos eventos discretos representados pela teoria dos autômatos apresentou o resultado esperado, detectando todas as marcas na pista facilitando o controle da posição do robô sobre a linha.

A avaliação dos resultados obtidos nos testes práticos e em eventos de competição da RoboCore provam que os objetivos propostos nesse trabalho foram alcançados com êxito.

Para trabalhos futuros é possível sugerir a integração do controle fuzzy com

outras técnicas como redes neurais ou PID-Fuzzy. Além disso, é possível testar diferentes técnicas de defuzzyficação, e representação das funções de pertinência. A utilização de um microcontrolador de maior poder computacional com ponto flutuante pode ser crucial para se obter os resultados esperados.

REFERÊNCIAS

ÁLVAREZ'S, Daniel. Disponível em: <http://dani.foroselectronica.es/silvestre-uxbot-line-following-champions-148/>. Acesso em: 31 mar. 2015.

CORREIA, Diogo S. O. **Navegação autônoma de robôs móveis de detecção de intrusos em ambientes internos utilizando sensores 2D e 3D.** 2013. Dissertação Mestrado em Ciências de Computação e Matemática Computacional - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional. USP, São Carlos – SP.

COSTA, Eduardo. R.; GOMES, Marcel. L.; BIANCHI, Reinaldo. A. C. Um mini robô móvel seguidor de pistas guiado por visão local. In: VI SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, SBAI, 6, 2003. Bauru. **Anais**… Bauru: SBAI, 2003. p. 710-715. Disponível em: < http://fei.edu.br/~rbianchi/publications/sbai2003.pdf>. Acesso em: 17 mai. 2015.

CURY, José E. R.; Teoria de controle supervisório de sistemas a eventos discretos. In: V SIMPÓSIO BRASILEIRO DE AUTOMAÇÃO INTELIGENTE. 2011. **Anais eletrônicos**... Canela, 2011. Disponível em: <http://user.das.ufsc.br/~cury/cursos/apostila.pdf>. Acesso em: 31 mai. 2015.

FELIZARDO, Ivanilza Felizardo; BRACARENSE, Alexandre Queiroz. **Processos de Soldagem - Processos Mecanizados e Automatizados**. Apostila. 2005. Disponível em: <http://ivanilzafe.dominiotemporario.com/soldagem_20.html>. Acesso em: 12 jun. 2015.

GOMIDE, Fernando. A. C.; GUDWIN, Ricardo. R. Modelagem, controle, sistemas e lógica fuzzy. **Revista Controle & Automação**, Campinas, v. 4, n. 3, p. 97-115, 1994. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/gudwin/publications/RevSBA94.pdfl>. Acesso em: 25 mai. 2015.

GUADAGNIN, Alan Junior. **Controle híbrido de um robô seguidor de linha**. 2014. Dissertação de graduação de Engenharia Elétrica, Pato Branco.

HEINEN, Farlei Jose. **Sistema de controle híbrido para robôs moveis autônomos**. 2002. Dissertação de Mestrado, São Leopoldo.

HIRAI, Aniki. 2016 Disponível em: <http://anikinonikki.cocolog-nifty.com/blog/2014/11/cartisx04.html> Acesso em: 27 mai. 2016.

INSTRUMENTS, Texas. 2015. Disponível em: <http://www.ti.com/ww/en/launchpad/launchpads-msp430-msp-exp430g2.html >.

JUNG, Claudio Rosito et al. Computação embarcada: Projeto e implementação de veículos autônomos inteligentes. In: XXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 2005. Rio Grande do Sul. **Anais**... Rio Grande do Sul: UNISINOS, 200, p. 2.

JUNIOR, Valdir G. **Arquitetura hibrida para robôs móveis baseada em funções de navegação com interação humana**. Tese de doutorado em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo, São Paulo, 2006.

MARIN, Luciene D. O. **Arquitetura neural cognitiva para controle inteligente de robôs móveis em labirintos dinâmicos**. 2010. 239p. Tese de doutorado em Engenharia Elétrica pela Universidade Federal de Santa Catarina, Florianópolis.

MARRO, Alessandro A. et al. **Lógica Fuzzy: conceitos e aplicações.** Universidade Federal do Rio Grande do Norte. Natal, 2010. Disponível em: <files.sistele7.webnode.com/200000428-ca5bccc4fb/texto_fuzzy.pdf> Acesso em: 25 mai. 2015.

MATTIELLO, Caciano D. **Comparativo entre controlador PID e fuzzy no controle de atitude em um quadcóptero**. 2014. Dissertação de graduação de Engenharia de Computação, Pato Branco.

MENEGUELE, Bruno E. D. O.; FERREIRA, Fernando P.; ARCANJO, Vinicius D. S**. Robô explorador de labirintos 2D**. 2011. 52p.Monografia UTFPR, Curitiba, 2011.

MINUSSI, Carlos. R. **Lógica nebulosa (lógica fuzzy)**. Ilha Solteira: Unesp/FE/DEEE, 2009. 119 p.

MORAES, Airton Almeida de Moraes**. Robótica e automação industrial**, Diretoria Técnica do SENAI-SP. Serviço Nacional de Aprendizagem Industrial, 2003. Apostila. Disponível em <http://www.adororobotica.com/RBSENAI.pdf>. Acesso em: 12 jun. 2015

NEHMZOW, Ulrich. **Mobile robotics**: a practical introduction. 2. ed. Essex: Springer, 2000.

NOGUEIRA, Maycon M. **Aplicando lógica fuzzy no controle de robôs móveis usando dispositivos lógicos programáveis e a linguagem Vhdl**. 2013. Dissertação Mestrado em Engenharia Elétrica - Programa de Pós-Graduação em Engenharia Eletrica, UNESP, Ilha Solteira – SP.

PEDRYCZ, Witold.; GOMIDE, Fernando. **Fuzzy systems engineering: toward human-centric computing**. IEEE PRESS, 2007.

PESSIN, Gustavo. **Estratégias inteligentes aplicadas em robôs móveis autônomos e em coordenação de grupos de robôs**. 2013. Tese Programa de Pós-Graduação em Ciências de Computação, ICMC-USP, São Carlos.

POLOLU, Robotics & Eletronics, 2016. Disponível em: <https://www.pololu.com/product/975>. Acesso em: 30 mai. 2016.

RIBEIRO, M. Isabel. **Sensores em Robótica**, Enciclopédia Nova Activa Multimédia, Volume de Tecnologias, pags. 228-229. Portugal, 2004.

ROBOCORE, 2015. Disponível em: <https://www.robocore.net/modules.php?name=GR_Eventos>. Acesso em: 25 mar. 2015.

ROBOCUP, 2015. Disponível em: <http://www.robocup2015.org/>. Acesso em: 25 mar. 2015.

OGATA, Katsuhiko. **Engenharia de controle moderno**. Tradução de Prof. Bernardo Severo. Rio de Janeiro: LTC Editora, 1998.

SHAW, Ian S; SIMÕES, Marcelo G.. **Controle e modelagem fuzzy.** São Paulo: Edgard Blücher LTDA, 1999.

SIERAKOWSKI, Cezar A. **Inteligência coletiva aplicada a problemas de robótica móvel.** 2006. Dissertação Mestrado em Engenharia de Produção e Sistemas. PUC, Curitiba.

SPARKFUN, 2015. Disponível em: <https://www.sparkfun.com/products/9457>. Acesso em: 16 jun. 2015.

SU, Juing-Huei et al. **An intelligent line-following robot project for introductory robot courses**. In: World Transactionson Engineering and Technology Education.Vol.8, No.4, 2010. Taoyuan County, Taiwan. Disponível em: <http://www.wiete.com.au/journals/WTE&TE/Pages/Vol.8,%20No.4%20%282010%29/9-15-SU-J-H.pdf>. Acesso em: 11 mai. 2015.

SUPREMICA, 2015**. A tool for verification and synthesis of discrete supervisors**. Disponível em: <http://www.supremica.org/>. Acesso em: 16 jun. 2015.

TEIXEIRA, M. et al. **Variable abstraction and approximations in supervisory control synthesis**. In: 2013 American Control Conference (ACC'13). Washington, USA: [s.n.], 2013. p. 120-125.

VUONG, Philip. T.; MADNI, Asad. M.; VOUNG, Jim. B. **VHDL implementation for a fuzzy logic controller**. In: WORLD AUTOMATION CONGRESS – WAC, 2006, Budapest. Proceedings of the… Budapest: IEEE, 2006. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4259884>. Acesso em: 17 mai. 2015.

WOLF, Denis. F.; SIMÕES, Eduardo D. V.; OSÓRIO, Fernando S.; TRINDADE JUNIOR, Onofre. **Robótica móvel inteligente**: da simulação às aplicações no mundo real. In: JAI-Jornada de Atualização em informática 2009 (Tutorial) – Congresso da SBC, Bento Gonçalves, SP: Editora PUC do Rio, Acesso em: 27 mar. 2015. Disponível em: <http://osorio.wait4.org/palestras/texto/JAI2009_Completo_Revisado.pdf>

ZADEH, Lotfali A. **Fuzzy sets**: information and control, vol. 8, p. 338-353, 1965.

APÊNDICE A - IMPLENTAÇÃO DO CONTROLADOR HIBRIDO USANDO PID NO MICROCONTROLADOR DA TEXAS

```
#include "msp430g2553.h"
#include <intrinsics.h>  //__bic_SR_register_on_exit(CPUOFF);
//#include "io430.h"

//Dados do autômato (Não pode ser declarado dentro da função main por ser const)
#define NTRANS 16  //Número de Transições
#define NESTADOS 12 //Número de Estados
#define BUFFER 10   //Máximo Número de Eventos no Buffer
const unsigned int event[NTRANS]={2,2,4,6,8,10,8,10,10,6,6,10,4,8,6,4};
const unsigned int in_state[NTRANS]={1,2,3,4,5,10,11,4,9,4,8,4,6,7,1,7};
const unsigned int rfirst[NESTADOS] = {1,2,3,4,13,16,15,11,12,10,7,8};
const unsigned int rnext[NTRANS] = {0,0,0,0,0,0,0,0,0,9,0,5,0,14,6};

//mapeamento de eventos não controláveis como entradas
#define AB 2  //Entrada 0
#define ED 4  //Entrada 1
#define PD 6  //Entrada 2
#define EE 8  //Entrada 3
#define PE 10 //Entrada 4


//Definição de funções de inicialização
void config_clk(void);      //Configura Clock
void config_timer(void);    //Configura Timer
void config_io(void);     //Configura entradas e saídas
void leitura_adc10(void);   //Lê e armazena os valores dos sensores de posição
void pid_control(void);     //controle PID acionamento dos motores por PWM
void calibra_sensor(void);

//Declaração de variáveis globais
unsigned int buffer[BUFFER];  //Buffer para armazenar a fila de enventos externos
unsigned int n_buffer=0;    //Número de eventos no Buffer

unsigned  int A[5] = {0};
unsigned  int posicao = 0;    //posição do robô sobre a faixa (0~4000) centro= 2000
unsigned long int pos_divisor = 0;

 unsigned int max[5] = {0};
 unsigned int min[5] = {100, 100, 100, 100, 100};

int velocidade=0;

int proporcional_passado = 0; //para controle de posição PID
//int saida_pwm = 0;
int integral = 0;

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD; //Desabilita o WDT
  config_clk();       //configura o clk
  config_timer();         //configura o timer
  config_io();        //configura entradas e saídas
  P3OUT = BIT5+BIT6+BIT7;
```

```
unsigned int k;
int occur_event;       //Evento ocorrido
unsigned int current_state = 0; //Estado atual inicializado com estado inicial
int g=0;           //Flag para gerador aleatório de eventos
int gerar_evento=1;      //Flag para habilitar a temporização de eventos controláveis
int moore_output = 0;     //Inicializa saída periférica
_enable_interrupt();      //Habilita interrupção geral

while(1)
{
  WDTCTL = WDTPW + WDTHOLD;
  //P2OUT |= BIT4;    //liga led antes da leitura
  leitura_adc10();  //Lê e armazena os valores dos sensores de posição
  //P2OUT ^= BIT4;    //desliga led
          //tempo de amostragem = tempo LED ligado
  pid_control();    //controle PID acionamento dos motores por PWM



  if(n_buffer == 0) //se não existir evento no buffer então gerar um evento interno(evento controlável)
  {
    if(TACTL&TAIFG) //Se o timer estourar, habilita a geração de eventos
    {
      gerar_evento=1;
    }
    if(gerar_evento==1)
    {
      switch(g) //Aqui é implementado um gerador automático de eventos controláveis
      {
      }
    }
  }
  else       //se existir evento não controlável pegar do buffer
  {
    occur_event = buffer[0];
    n_buffer--;
    k = 0;
    while(k<n_buffer)
    {
      buffer[k] = buffer[k+1];
      k++;
    }
  }

  //Jogador de autômato
  k = rfirst[current_state];
  if(k==0)
  {
    return;     //Dead Lock!!!
  }
  else
  {
    while(k>0)
    {
      k--;
      if(event[k] == occur_event)
      {
        current_state = in_state[k];
```

```
        moore_output = 1;
        break;
      }
      k = rnext[k];
    }
  }

  if(moore_output) //Se o evento ocorrido for válido, então imprimir saída física
  {
    gerar_evento=1;
    switch(current_state)
    {
      case(0):  //Adicionar Ação para o Estado 0;(parado)
         P3OUT = BIT1+BIT4;
        break;
      case(1):  //Adicionar Ação para o Estado 1;(calibra sensor)

        __disable_interrupt();//disable all interrupts
        unsigned long int i;
        for(i=0; i<120000; i++)
        {
          P3OUT = BIT4; //Azul
          calibra_sensor();
        }
          P3OUT = BIT1; //Branco
          velocidade = 0;
           __enable_interrupt();

        break;
      case(2):  //Adicionar Ação para o Estado 2;
          //ROBO PARADA NA AREA DE START\STOP COMEÇA ANDAR COM VELOCIDADE
INICIAL
        P3OUT = BIT0  +   BIT5; //STBY ponte H + LED VERMELHO
        velocidade = 320; //vel max=399!
        //velocidade = 385;   //6,9seg        //pwm=210 motor alta potencia

        break;
      case(3):  //Adicionar Ação para o Estado 3;(saindo da Largada)
          P3OUT = BIT0  +   BIT7;  //STBY ponte H + LED VERDE

        break;
      case(4):  //Adicionar Ação para o Estado 4;
          //SAINDO DA LARGADA, CURVA OU CRUZAMENTO
          P3OUT = BIT0 +   BIT6 ;  //STBY ponte H + LED AMARELO
          velocidade = 350; //vel max=399!

        break;
      case(5):  //Adicionar Ação para o Estado 5;
           //POSIBILIDADE DE CURVA OU CRUZAMENTO
        break;
      case(6):  //Adicionar Ação para o Estado 6;
           //POSIBILIDADE DE CRUZAMENTO OU FIM DE FISTA
        break;
      case(7):  //Adicionar Ação para o Estado 7;
           //ACHOU CRUZAMENTO
        break;
      case(8):  //Adicionar Ação para o Estado 8;
           //SAINDO CRUZAMENTO
        break;
```

```
        case(9):  //Adicionar Ação para o Estado 9;
             //SAINDO CRUZAMENTO
          break;
        case(10): //Adicionar Ação para o Estado 10;
          //ENCONTROU CURVA
          P3OUT = BIT0  +   BIT5; //STBY ponte H + LED VERMELHO
          velocidade = 250; //vel max=399!
          break;
        case(11): //Adicionar Ação para o Estado 11;
          //SAINDO CURVA
          break;
      }//fim switch
      moore_output = 0;
      occur_event = -1;
    }//fim if(moore_output)
  }//fim while(1)
}//fim main

//Tratamento da Interrupção da porta P1
//-------------------------------------------------------------
//-------------------------------------------------------------
#pragma vector=PORT1_VECTOR
__interrupt void RTI_PORT1(void)

{

  if((P1IFG&BIT5) && !(P1IN&BIT5))  //verifica se P1.5 foi presionado (Encontra Direita)
  {
    P1IFG&=~BIT5;   //apaga o flag
    buffer[n_buffer]=ED;  //Atribuir evento a P1.5
    if(n_buffer<BUFFER-1) n_buffer++;
    P1IES &= ~BIT5; // habilita int. por borda de subida
    //P1IES |= BIT5; // habilita int. por borda de descida
  }
  if((P1IFG&BIT5) && (P1IN&BIT5))   //verifica se P1.5 (Perde Direita)
  {
    P1IFG&=~BIT5;   //apaga o flag
    buffer[n_buffer]=PD;  //Atribuir evento a P1.5
    if(n_buffer<BUFFER-1) n_buffer++;
    P1IES |= BIT5; // habilita int. por borda de descida
    //P1IES &= ~BIT5; // habilita int. por borda de subida
  }

  if((P1IFG&BIT6) && !(P1IN&BIT6))  //verifica se P1.6 borda descida
  {
    P1IFG&=~BIT6;   //apaga o flag
    buffer[n_buffer]=EE;  //Atribuir evento a P1.6
    if(n_buffer<BUFFER-1) n_buffer++;
    P1IES &= ~BIT6; // habilita int. por borda de subida


  }
  if((P1IFG&BIT6) && (P1IN&BIT6)) //verifica se P1.6 borda subida
  {
    P1IFG&=~BIT6;   //apaga o flag
    buffer[n_buffer]=PE;  //Atribuir evento a P1.6
    if(n_buffer<BUFFER-1) n_buffer++;
    P1IES |= BIT6; // habilita int. por borda de descida
  }
```

```
  if(P1IFG&BIT7)  //verifica se P1.7 foi presionado (Aperta Botão)
  {
    P1IFG&=~BIT7;   //apaga o flag
    buffer[n_buffer]=AB;  //Atribuir evento a P1.7
    if(n_buffer<BUFFER-1) n_buffer++;
    //P1IES &= ~BIT7; // habilita int. por borda de subida
  }
  P1IFG=0;
}

//Tratamento da Interrupção d0 ADC10
//-------------------------------------------------------------
//-------------------------------------------------------------
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  unsigned long int B[5]= {0};

  char i; //char 0-255
  for ( i=0; i<5; i++)
  {
    B[i] = (1000 - ( ((unsigned long int) (A[i]- min[i]) *1000) / (max[i]- min[i]) ) );
  }

  pos_divisor = ((B[0]*0)+(B[1]*1000)+(B[2]*2000)+(B[3]*3000)+(B[4]*4000));
  posicao = (unsigned int )(pos_divisor/(B[0]+B[1]+B[2]+B[3]+B[4]));



  __disable_interrupt();//disable all interrupts
}

//Configuração de perifericos
//-------------------------------------------------------------
//-------------------------------------------------------------
void config_clk(void)
{
    WDTCTL = WDTPW + WDTHOLD;       // Desativando o WatchDog Timer

    DCOCTL = 0x00;
    BCSCTL1 = CALBC1_16MHZ;          // Set range
    DCOCTL = CALDCO_16MHZ;             // Set DCO step + modulation
  BCSCTL1 |= XT2OFF;           //desliga oscilador XTE
  //BCSCTL2 seleciona a oriem e div dos MCLK e SMCLK
  //DIVM_3 +DIVS_3= DCO/8
  //BCSCTL2 |=  SELM_0 + DIVM_0 + DIVS_3;     //MCLK= 16MHz SMCLK/8 = 2MHz
  BCSCTL2 |=  SELM_0 + DIVM_0 + DIVS_2;     //MCLK= 16MHz SMCLK/4 = 4MHz
  //BCSCTL3  seleciona config para cristal ou resonador externo
  //BCSCTL3 |= XCAP0 +XCAP1;    //configura os capacitores para o oscilador externo ACLK = 32768
Hz
  __enable_interrupt();        //Seta o bit GIE - "chavegeral" das interrupções
}

//-------------------------------------------------------------
//-------------------------------------------------------------
void config_timer(void) //configura o timer
{
```

```
  TA1CTL = TASSEL_2 + MC_1; //10 - clock secundário (SMCLK) (símbolo TASSEL_2); 01 contagem progressiva com módulo (O até o valor de TACCRO) (símbolo MC_1);
  TA1CCTL1 = OUTMOD_7; // Modo 7: Reset/Set
  TA1CCTL2 = OUTMOD_7; //a saída é apagada na comparação e setada ao atingir o valor do TACCRO (símbolo OUTMOD_7).
  //---------------------------------------
  // TA1CCR0 para 100us para 10KHz //ou ainda 50us para 20KHz
  // 1/SMCLK = 1/4MHz = 0.25 us
  // TA1CCR0 = ( 100us / 0,25us) - 1 = 399  //ou ainda ( 50us / 0,25us) - 1 = 199 para 20KHz
  TA1CCR0 = 399;     //frequencia do timer 10KHz
  //---------------------------------------
  // TA1CCR0 para 500us para 2KHz
  // 1/SMCLK = 1/2MHz = 0.5 us
  // TA1CCR0 = ( 500us / 0,5us) - 1 = 999
  //TA1CCR0 = 999;     //frequencia do timer 2KHz
  TA1CCR1 = 0;       // variacao da razão ciclica motor Direito
  TA1CCR2 = 0;     // variacao da razão ciclica motor Esquerdo
  //Maximum PWM frequency: 100 kHz para TB6612FNG
    //veja no osciloscopio q frequencia tá, para mudar a frq mudar TACCR0
}

//-------------------------------------------------------------
//-------------------------------------------------------------
void config_io(void)  //configura entradas e saídas
{
  WDTCTL = WDTPW + WDTHOLD;
  //Configuracao porta P1
    P1DIR = 0;          //toda porta como entrada, bits 0-4 ADC e bits 5-7  como entrada (botão, sensores de faixa)
    P1REN |=BIT5+BIT6+BIT7;     //habilita resistores de pull(up/down)
    P1OUT |=BIT5+BIT6;    //pull UP

    P1OUT &= ~BIT7;     //pull down
    P1IE  |= BIT5+BIT6+BIT7;  //habilita interrupção das entradas P1
    P1IES &= ~BIT7; //a borda de subida que provoca interrupção (descida = 1 - subida = 0)
    P1IES |= BIT5+BIT6;   //descida provoca interupcao
    P1IFG &=~(BIT5+BIT6+BIT7);  //Apaga o Flag da interrupção
    //-------------------------------------------------------------
    //Configuracao porta P2
    P2SEL &= ~(BIT6 | BIT7);          //Bit 6 e 7 do P2 como IO
    P2DIR |= BIT0+BIT1+BIT2+BIT3+BIT4+BIT5+BIT6+BIT7; //toda a porta P2 como saída
    P2OUT = 0;      //todas as saídas em nível lógico 0


      //-------------------------------------------------------------
      //Configuracao porta P3
      P3DIR |= BIT0+BIT1+BIT2+BIT3+BIT4+BIT5+BIT6+BIT7; //Toda porta como saida (LEDs +STBY+ PWM)
      P3SEL = BIT2+BIT3 ;      //ativa função secundaria (TA1.1 TA1.2) = PWM
      //BIT2=PWMB, BIT3=PWMA
      P3OUT &= ~(BIT0+BIT1+ BIT4+BIT5+BIT6+BIT7); //pull down para LEDs + STBY(P3.0)


    //-------------------------------------------------------------
    //CONFIGURAÇÃO ADC10
    //freq operação do ADC deve ficar entre 450KHz e 6,3MHz!!
                        // Stop WDT
    ADC10CTL1 = INCH_4 + CONSEQ_1+ ADC10DIV_0 + ADC10SSEL_0;   // A4/A3/A2/A1/A0, sequencia de canais sem repetições
```

```
    ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;      //ADC10SHT_2 = 16 x
ADC10CLKs
    ADC10DTC1 = 0x05;                              //numero de transferencias DTC por bloco, 5
conversions
    ADC10AE0 |= 0x1F;                             // Habilita os buffers ADC na porta P1.0...P1.4
    //ADC10DTC0 |= ADC10TB + ADC10CT;                 //transferencia de um bloco modo continuo
}

//-------------------------------------------------------------------
//-------------------------------------------------------------------
void leitura_adc10(void)          //Lê e armazena os valores dos sensores de posição
{
  volatile int adc[5] = {0};       //Vetor para armazenar os valores das 5 conversões ADC
    ADC10CTL0 &= ~ENC;
    while (ADC10CTL1 & BUSY);           // Wait if ADC10 core is active
    ADC10SA = (unsigned int) &adc[0];  //ADC10 recebe o endereço de memoria RAM do vetor adc[]
    ADC10CTL0 |= ENC + ADC10SC;         //Inicializa leitura/conversao

    while (ADC10CTL1 & BUSY);
    A[0] = adc[0];
    A[1] = adc[1];
    A[2] = adc[2];
    A[3] = adc[3];
    A[4] = adc[4];

    __enable_interrupt();
}


//-------------------------------------------------------------------
//-------------------------------------------------------------------
//Calibra sensores numa faixa 0-1000

void calibra_sensor(void)
{
  WDTCTL = WDTPW + WDTHOLD;

  volatile int adc[5] = {0};       //Vetor para armazenar os valores das 5 conversões ADC
    ADC10CTL0 &= ~ENC;
    while (ADC10CTL1 & BUSY);           // Wait if ADC10 core is active
    ADC10SA = (unsigned int) &adc[0];  //ADC10 recebe o endereço de memoria RAM do vetor adc[]
    ADC10CTL0 |= ENC + ADC10SC;         //Inicializa leitura/conversao

    while (ADC10CTL1 & BUSY);


  char i;
  for ( i=0; i<5; i++)
  {
    if(adc[i] > max[i])
    {
      max[i] = adc[i];
    }
    if(adc[i] < min[i])
    {
      min[i] = adc[i];
    }
  }
```

```
}
//-------------------------------------------------------------
//-------------------------------------------------------------
//Controle dos motores usando PID
void pid_control(void)
{
  WDTCTL = WDTPW + WDTHOLD;

  float Kp = 1.43, Ki = 1.0e-3, Kd = 44; // PWM 190 HPCB 10KHz

    int proporcional = (posicao) - 2000;          // set point para 5 sensores é 2000
    integral = integral + proporcional_passado;       //obtendo a integral
    int derivativo = (proporcional - proporcional_passado); //obtendo o derivativo
    if (integral >  1000)  integral =  1000;          //limitamos a integral para nao causar problemas
    if (integral < -1000) integral = -1000;

    int saida_pwm = (int)(((float)proporcional * Kp ) + ((float)derivativo * Kd )+ ((float)integral*Ki));
    //int saida_pwm =( proporcional * Kp ) + ( derivativo * Kd )+(integral*Ki);
    int PWM = saida_pwm;


    if ( PWM > velocidade )   PWM =  velocidade; //limitamos a saida de pwm entre 0-1000
    if ( PWM < -velocidade )  PWM = -velocidade;

    if (PWM < 0)
    {
      TA1CCR1 = velocidade+PWM;
      TA1CCR2 = velocidade;
    }
    if (PWM > 0)
    {
      TA1CCR1 = velocidade;
      TA1CCR2 = velocidade-PWM;
    }

    proporcional_passado = proporcional;
}
```

APÊNDICE B - IMPLEMENTAÇÃO CONTROLE FUZZY NO ROBÔ 3PI DA POLOLU

```
#include <Pololu3pi.h>
#include <PololuQTRSensors.h>
#include <OrangutanMotors.h>
#include <OrangutanAnalog.h>
#include <OrangutanLEDs.h>
#include <OrangutanLCD.h>
#include <OrangutanPushbuttons.h>
#include <OrangutanBuzzer.h>

//--------------------------------------------------------------------
#define FIS_TYPE float
#define FIS_RESOLUSION 101
#define FIS_MIN -3.4028235E+38
#define FIS_MAX 3.4028235E+38
typedef FIS_TYPE(*_FIS_MF)(FIS_TYPE, FIS_TYPE*);
typedef FIS_TYPE(*_FIS_ARR_OP)(FIS_TYPE, FIS_TYPE);
typedef FIS_TYPE(*_FIS_ARR)(FIS_TYPE*, int, _FIS_ARR_OP);

// Number of inputs to the fuzzy inference system
const int fis_gcI = 5;
// Number of outputs to the fuzzy inference system
const int fis_gcO = 2;
// Number of rules to the fuzzy inference system
const int fis_gcR = 14;

FIS_TYPE g_fisInput[fis_gcI];
FIS_TYPE g_fisOutput[fis_gcO];
//--------------------------------------------------------------------


Pololu3pi robot;
unsigned int sensors[5]; // an array to hold sensor values

// This include file allows data to be stored in program space.  The
// ATmega168 has 16k of program space compared to 1k of RAM, so large
// pieces of static data should be stored in program space.
#include <avr/pgmspace.h>

// Introductory messages.  The "PROGMEM" identifier causes the data to
// go into program space.
const char welcome_line1[] PROGMEM = " Pololu";
const char welcome_line2[] PROGMEM = "3\xf7 Robot";
const char demo_name_line1[] PROGMEM = "Fuzzy";
const char demo_name_line2[] PROGMEM = "follower";

// A couple of simple tunes, stored in program space.
const char welcome[] PROGMEM = ">g32>>c32";
const char go[] PROGMEM = "L16 cdegreg4";

// Data for generating the characters used in load_custom_characters
// and display_readings.  By reading levels[] starting at various
// offsets, we can generate all of the 7 extra characters needed for a
```

```
// bargraph.  This is also stored in program space.
const char levels[] PROGMEM = {
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111,
  0b11111
};

// This function loads custom characters into the LCD.  Up to 8
// characters can be loaded; we use them for 7 levels of a bar graph.
void load_custom_characters()
{
  OrangutanLCD::loadCustomCharacter(levels + 0, 0); // no offset, e.g. one bar
  OrangutanLCD::loadCustomCharacter(levels + 1, 1); // two bars
  OrangutanLCD::loadCustomCharacter(levels + 2, 2); // etc...
  OrangutanLCD::loadCustomCharacter(levels + 3, 3);
  OrangutanLCD::loadCustomCharacter(levels + 4, 4);
  OrangutanLCD::loadCustomCharacter(levels + 5, 5);
  OrangutanLCD::loadCustomCharacter(levels + 6, 6);
  OrangutanLCD::clear(); // the LCD must be cleared for the characters to take effect
}

// This function displays the sensor readings using a bar graph.
void display_readings(const unsigned int *calibrated_values)
{
  unsigned char i;

  for (i=0;i<5;i++) {
    // Initialize the array of characters that we will use for the
    // graph.  Using the space, an extra copy of the one-bar
    // character, and character 255 (a full black box), we get 10
    // characters in the array.
    const char display_characters[10] = { ' ', 0, 0, 1, 2, 3, 4, 5, 6, 255 };

    // The variable c will have values from 0 to 9, since
    // calibrated values are in the range of 0 to 1000, and
    // 1000/101 is 9 with integer math.
    char c = display_characters[calibrated_values[i] / 101];

    // Display the bar graph character.
    OrangutanLCD::print(c);
  }
}

// Initializes the 3pi, displays a welcome message, calibrates, and
// plays the initial music.  This function is automatically called
// by the Arduino framework at the start of program execution.
void setup()
{
```

```
unsigned int counter; // used as a simple timer

// This must be called at the beginning of 3pi code, to set up the
// sensors.  We use a value of 2000 for the timeout, which
// corresponds to 2000*0.4 us = 0.8 ms on our 20 MHz processor.
robot.init(2000);

load_custom_characters(); // load the custom characters

// Play welcome music and display a message
OrangutanLCD::printFromProgramSpace(welcome_line1);
OrangutanLCD::gotoXY(0, 1);
OrangutanLCD::printFromProgramSpace(welcome_line2);
OrangutanBuzzer::playFromProgramSpace(welcome);
delay(1000);

OrangutanLCD::clear();
OrangutanLCD::printFromProgramSpace(demo_name_line1);
OrangutanLCD::gotoXY(0, 1);
OrangutanLCD::printFromProgramSpace(demo_name_line2);
delay(1000);

// Display battery voltage and wait for button press
while (!OrangutanPushbuttons::isPressed(BUTTON_B))
{
  int bat = OrangutanAnalog::readBatteryMillivolts();

  OrangutanLCD::clear();
  OrangutanLCD::print(bat);
  OrangutanLCD::print("mV");
  OrangutanLCD::gotoXY(0, 1);
  OrangutanLCD::print("Press B");

  delay(100);
}

// Always wait for the button to be released so that 3pi doesn't
// start moving until your hand is away from it.
OrangutanPushbuttons::waitForRelease(BUTTON_B);
delay(1000);

// Auto-calibration: turn right and left while calibrating the
// sensors.
for (counter=0; counter<80; counter++)
{
  if (counter < 20 || counter >= 60)
    OrangutanMotors::setSpeeds(34, -34);
  else
    OrangutanMotors::setSpeeds(-34, 34);

  // This function records a set of sensor readings and keeps
  // track of the minimum and maximum values encountered.  The
  // IR_EMITTERS_ON argument means that the IR LEDs will be
  // turned on during the reading, which is usually what you
  // want.
  robot.calibrateLineSensors(IR_EMITTERS_ON);

  // Since our counter runs to 80, the total delay will be
  // 80*20 = 1600 ms.
```

```
    delay(20);
  }
  OrangutanMotors::setSpeeds(0, 0);

  // Display calibrated values as a bar graph.
  while (!OrangutanPushbuttons::isPressed(BUTTON_B))
  {
    // Read the sensor values and get the position measurement.
    unsigned int position = robot.readLine(sensors, IR_EMITTERS_ON, true);

    // Display the position measurement, which will go from 0
    // (when the leftmost sensor is over the line) to 4000 (when
    // the rightmost sensor is over the line) on the 3pi, along
    // with a bar graph of the sensor readings.  This allows you
    // to make sure the robot is ready to go.
    OrangutanLCD::clear();
    OrangutanLCD::print(position);
    OrangutanLCD::gotoXY(0, 1);
    display_readings(sensors);

    delay(100);
  }
  OrangutanPushbuttons::waitForRelease(BUTTON_B);

  OrangutanLCD::clear();

  OrangutanLCD::print("Go!");

  // Play music and wait for it to finish before we start driving.
  OrangutanBuzzer::playFromProgramSpace(go);
  while(OrangutanBuzzer::isPlaying());
}

// The main function.  This function is repeatedly called by
// the Arduino framework.
void loop()
{
    robot.readLine(sensors, IR_EMITTERS_ON, true);
    // Read Input: Sensor0
    g_fisInput[0] = sensors[0];
    // Read Input: Sensor1
    g_fisInput[1] = sensors[1];
    // Read Input: Sensor2
    g_fisInput[2] = sensors[2];
    // Read Input: Sensor3
    g_fisInput[3] = sensors[3];;
    // Read Input: Sensor4
    g_fisInput[4] = sensors[4];

    g_fisOutput[0] = 0;
    g_fisOutput[1] = 0;

    fis_evaluate();

    int mot_e = (int)(g_fisOutput[0] *0.3);
    int mot_d = (int)(g_fisOutput[1] *0.3);

   OrangutanMotors::setSpeeds(mot_e, mot_d);
```

```
}


//*********************************************************************
// Support functions for Fuzzy Inference System
//*********************************************************************
// Trapezoidal Member Function
FIS_TYPE fis_trapmf(FIS_TYPE x, FIS_TYPE* p)
{
    FIS_TYPE a = p[0], b = p[1], c = p[2], d = p[3];
    FIS_TYPE t1 = ((x <= c) ? 1 : ((d < x) ? 0 : ((c != d) ? ((d - x) / (d - c)) : 0)));
    FIS_TYPE t2 = ((b <= x) ? 1 : ((x < a) ? 0 : ((a != b) ? ((x - a) / (b - a)) : 0)));
    return (FIS_TYPE) min(t1, t2);
}

FIS_TYPE fis_min(FIS_TYPE a, FIS_TYPE b)
{
    return min(a, b);
}

FIS_TYPE fis_max(FIS_TYPE a, FIS_TYPE b)
{
    return max(a, b);
}

FIS_TYPE fis_array_operation(FIS_TYPE *array, int size, _FIS_ARR_OP pfnOp)
{
    int i;
    FIS_TYPE ret = 0;

    if (size == 0) return ret;
    if (size == 1) return array[0];

    ret = array[0];
    for (i = 1; i < size; i++)
    {
        ret = (*pfnOp)(ret, array[i]);
    }

    return ret;
}


//*********************************************************************
// Data for Fuzzy Inference System
//*********************************************************************
// Pointers to the implementations of member functions
_FIS_MF fis_gMF[] =
{
    fis_trapmf
};

// Count of member function for each Input
int fis_gIMFCount[] = { 3, 3, 3, 3, 3 };

// Count of member function for each Output
int fis_gOMFCount[] = { 4, 4 };

// Coefficients for the Input Member Functions
```

```
FIS_TYPE fis_gMFI0Coeff1[] = { 0, 0, 30, 70 };
FIS_TYPE fis_gMFI0Coeff2[] = { 45, 150, 300, 600 };
FIS_TYPE fis_gMFI0Coeff3[] = { 500, 715, 1000, 1000 };
FIS_TYPE* fis_gMFI0Coeff[] = { fis_gMFI0Coeff1, fis_gMFI0Coeff2, fis_gMFI0Coeff3 };
FIS_TYPE fis_gMFI1Coeff1[] = { 0, 0, 30, 70 };
FIS_TYPE fis_gMFI1Coeff2[] = { 45, 150, 300, 600 };
FIS_TYPE fis_gMFI1Coeff3[] = { 500, 715, 1000, 1000 };
FIS_TYPE* fis_gMFI1Coeff[] = { fis_gMFI1Coeff1, fis_gMFI1Coeff2, fis_gMFI1Coeff3 };
FIS_TYPE fis_gMFI2Coeff1[] = { 0, 0, 30, 70 };
FIS_TYPE fis_gMFI2Coeff2[] = { 45, 150, 300, 600 };
FIS_TYPE fis_gMFI2Coeff3[] = { 500, 715, 1000, 1000 };
FIS_TYPE* fis_gMFI2Coeff[] = { fis_gMFI2Coeff1, fis_gMFI2Coeff2, fis_gMFI2Coeff3 };
FIS_TYPE fis_gMFI3Coeff1[] = { 0, 0, 30, 70 };
FIS_TYPE fis_gMFI3Coeff2[] = { 45, 150, 300, 600 };
FIS_TYPE fis_gMFI3Coeff3[] = { 500, 715, 1000, 1000 };
FIS_TYPE* fis_gMFI3Coeff[] = { fis_gMFI3Coeff1, fis_gMFI3Coeff2, fis_gMFI3Coeff3 };
FIS_TYPE fis_gMFI4Coeff1[] = { 0, 0, 30, 70 };
FIS_TYPE fis_gMFI4Coeff2[] = { 45, 150, 300, 600 };
FIS_TYPE fis_gMFI4Coeff3[] = { 500, 715, 1000, 1000 };
FIS_TYPE* fis_gMFI4Coeff[] = { fis_gMFI4Coeff1, fis_gMFI4Coeff2, fis_gMFI4Coeff3 };
FIS_TYPE** fis_gMFICoeff[] = { fis_gMFI0Coeff, fis_gMFI1Coeff, fis_gMFI2Coeff, fis_gMFI3Coeff,
fis_gMFI4Coeff };

// Coefficients for the Input Member Functions
FIS_TYPE fis_gMFO0Coeff1[] = { 0, 0, 20, 50 };
FIS_TYPE fis_gMFO0Coeff2[] = { 110, 160, 175, 220 };
FIS_TYPE fis_gMFO0Coeff3[] = { 190, 240, 255, 255 };
FIS_TYPE fis_gMFO0Coeff4[] = { 45, 90, 110, 150 };
FIS_TYPE* fis_gMFO0Coeff[] = { fis_gMFO0Coeff1, fis_gMFO0Coeff2, fis_gMFO0Coeff3,
fis_gMFO0Coeff4 };
FIS_TYPE fis_gMFO1Coeff1[] = { 0, 0, 20, 50 };
FIS_TYPE fis_gMFO1Coeff2[] = { 110, 160, 175, 220 };
FIS_TYPE fis_gMFO1Coeff3[] = { 190, 240, 255, 255 };
FIS_TYPE fis_gMFO1Coeff4[] = { 45, 90, 110, 150 };
FIS_TYPE* fis_gMFO1Coeff[] = { fis_gMFO1Coeff1, fis_gMFO1Coeff2, fis_gMFO1Coeff3,
fis_gMFO1Coeff4 };
FIS_TYPE** fis_gMFOCoeff[] = { fis_gMFO0Coeff, fis_gMFO1Coeff };

// Input membership function set
int fis_gMFI0[] = { 0, 0, 0 };
int fis_gMFI1[] = { 0, 0, 0 };
int fis_gMFI2[] = { 0, 0, 0 };
int fis_gMFI3[] = { 0, 0, 0 };
int fis_gMFI4[] = { 0, 0, 0 };
int* fis_gMFI[] = { fis_gMFI0, fis_gMFI1, fis_gMFI2, fis_gMFI3, fis_gMFI4};

// Output membership function set
int fis_gMFO0[] = { 0, 0, 0, 0 };
int fis_gMFO1[] = { 0, 0, 0, 0 };
int* fis_gMFO[] = { fis_gMFO0, fis_gMFO1};

// Rule Weights
FIS_TYPE fis_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Type
int fis_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

// Rule Inputs
int fis_gRI0[] = { 1, 3, 3, 3, 3 };
```

```
int fis_gRI1[] = { 1, 1, 3, 3, 3 };
int fis_gRI2[] = { 2, 1, 2, 3, 3 };
int fis_gRI3[] = { 3, 1, 1, 3, 3 };
int fis_gRI4[] = { 3, 2, 1, 2, 3 };
int fis_gRI5[] = { 3, 3, 1, 1, 3 };
int fis_gRI6[] = { 3, 3, 2, 1, 2 };
int fis_gRI7[] = { 3, 3, 3, 1, 1 };
int fis_gRI8[] = { 3, 3, 3, 3, 1 };
int fis_gRI9[] = { 3, 3, 3, 3, 3 };
int fis_gRI10[] = { 3, 1, 1, 2, 3 };
int fis_gRI11[] = { 3, 2, 1, 1, 3 };
int fis_gRI12[] = { 1, 2, 3, 3, 3 };
int fis_gRI13[] = { 3, 3, 3, 2, 1 };
int* fis_gRI[] = { fis_gRI0, fis_gRI1, fis_gRI2, fis_gRI3, fis_gRI4, fis_gRI5, fis_gRI6, fis_gRI7, fis_gRI8,
fis_gRI9, fis_gRI10, fis_gRI11, fis_gRI12, fis_gRI13 };

// Rule Outputs
int fis_gRO0[] = { 1, 4 };
int fis_gRO1[] = { 1, 2 };
int fis_gRO2[] = { 1, 2 };
int fis_gRO3[] = { 1, 3 };
int fis_gRO4[] = { 3, 3 };
int fis_gRO5[] = { 3, 1 };
int fis_gRO6[] = { 2, 1 };
int fis_gRO7[] = { 2, 1 };
int fis_gRO8[] = { 4, 1 };
int fis_gRO9[] = { 1, 1 };
int fis_gRO10[] = { 4, 3 };
int fis_gRO11[] = { 3, 4 };
int fis_gRO12[] = { 1, 2 };
int fis_gRO13[] = { 2, 1 };
int* fis_gRO[] = { fis_gRO0, fis_gRO1, fis_gRO2, fis_gRO3, fis_gRO4, fis_gRO5, fis_gRO6, fis_gRO7,
fis_gRO8, fis_gRO9, fis_gRO10, fis_gRO11, fis_gRO12, fis_gRO13 };

// Input range Min
FIS_TYPE fis_gIMin[] = { 0, 0, 0, 0, 0 };

// Input range Max
FIS_TYPE fis_gIMax[] = { 1000, 1000, 1000, 1000, 1000 };

// Output range Min
FIS_TYPE fis_gOMin[] = { 0, 0 };

// Output range Max
FIS_TYPE fis_gOMax[] = { 255, 255 };

//************************************************************************
// Data dependent support functions for Fuzzy Inference System
//************************************************************************
FIS_TYPE fis_MF_out(FIS_TYPE** fuzzyRuleSet, FIS_TYPE x, int o)
{
    FIS_TYPE mfOut;
    int r;

    for (r = 0; r < fis_gcR; ++r)
    {
        int index = fis_gRO[r][o];
        if (index > 0)
        {
```

```
            index = index - 1;
            mfOut = (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
        }
        else if (index < 0)
        {
            index = -index - 1;
            mfOut = 1 - (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
        }
        else
        {
            mfOut = 0;
        }

        fuzzyRuleSet[0][r] = fis_min(mfOut, fuzzyRuleSet[1][r]);
    }
    return fis_array_operation(fuzzyRuleSet[0], fis_gcR, fis_max);
}

FIS_TYPE fis_defuzz_centroid(FIS_TYPE** fuzzyRuleSet, int o)
{
    FIS_TYPE step = (fis_gOMax[o] - fis_gOMin[o]) / (FIS_RESOLUSION - 1);
    FIS_TYPE area = 0;
    FIS_TYPE momentum = 0;
    FIS_TYPE dist, slice;
    int i;

    // calculate the area under the curve formed by the MF outputs
    for (i = 0; i < FIS_RESOLUSION; ++i){
        dist = fis_gOMin[o] + (step * i);
        slice = step * fis_MF_out(fuzzyRuleSet, dist, o);
        area += slice;
        momentum += slice*dist;
    }

    return ((area == 0) ? ((fis_gOMax[o] + fis_gOMin[o]) / 2) : (momentum / area));
}

//************************************************************************
// Fuzzy Inference System
//************************************************************************
void fis_evaluate()
{
    FIS_TYPE fuzzyInput0[] = { 0, 0, 0 };
    FIS_TYPE fuzzyInput1[] = { 0, 0, 0 };
    FIS_TYPE fuzzyInput2[] = { 0, 0, 0 };
    FIS_TYPE fuzzyInput3[] = { 0, 0, 0 };
    FIS_TYPE fuzzyInput4[] = { 0, 0, 0 };
    FIS_TYPE* fuzzyInput[fis_gcI] = { fuzzyInput0, fuzzyInput1, fuzzyInput2, fuzzyInput3, fuzzyInput4, };
    FIS_TYPE fuzzyOutput0[] = { 0, 0, 0, 0 };
    FIS_TYPE fuzzyOutput1[] = { 0, 0, 0, 0 };
    FIS_TYPE* fuzzyOutput[fis_gcO] = { fuzzyOutput0, fuzzyOutput1, };
    FIS_TYPE fuzzyRules[fis_gcR] = { 0 };
    FIS_TYPE fuzzyFires[fis_gcR] = { 0 };
    FIS_TYPE* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };
    FIS_TYPE sW = 0;

    // Transforming input to fuzzy Input
    int i, j, r, o;
    for (i = 0; i < fis_gcI; ++i)
```

```
    {
        for (j = 0; j < fis_gIMFCount[i]; ++j)
        {
            fuzzyInput[i][j] =
                (fis_gMF[fis_gMFI[i][j]])(g_fisInput[i], fis_gMFICoeff[i][j]);
        }
    }

    int index = 0;
    for (r = 0; r < fis_gcR; ++r)
    {
        if (fis_gRType[r] == 1)
        {
            fuzzyFires[r] = FIS_MAX;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    fuzzyFires[r] = fis_min(fuzzyFires[r], fuzzyInput[i][index - 1]);
                else if (index < 0)
                    fuzzyFires[r] = fis_min(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
                else
                    fuzzyFires[r] = fis_min(fuzzyFires[r], 1);
            }
        }
        else
        {
            fuzzyFires[r] = FIS_MIN;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    fuzzyFires[r] = fis_max(fuzzyFires[r], fuzzyInput[i][index - 1]);
                else if (index < 0)
                    fuzzyFires[r] = fis_max(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
                else
                    fuzzyFires[r] = fis_max(fuzzyFires[r], 0);
            }
        }

        fuzzyFires[r] = fis_gRWeight[r] * fuzzyFires[r];
        sW += fuzzyFires[r];
    }

    if (sW == 0)
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = ((fis_gOMax[o] + fis_gOMin[o]) / 2);
        }
    }
    else
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = fis_defuzz_centroid(fuzzyRuleSet, o);
        }
    }
}
```