CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

IMPLEMENTAÇÃO DE ESTRATÉGIAS DE COORDENAÇÃO DE MÚLTIPLOS ROBÔS MÓVEIS NA PLATAFORMA LEGO MINDSTORMS

MARIANA ATHAYDE GARCIA

Orientador: Prof. Tales Argolo Jesus
Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

BELO HORIZONTE
NOVEMBRO DE 2015

MARIANA ATHAYDE GARCIA

IMPLEMENTAÇÃO DE ESTRATÉGIAS DE COORDENAÇÃO DE MÚLTIPLOS ROBÔS MÓVEIS NA PLATAFORMA LEGO MINDSTORMS

Proposta de Trabalho de Conclusão de Curso apresentadoao Curso de Engenharia de Computação do CentroFederal de Educação Tecnológica de Minas Gerais.

Orientador: Tales Argolo Jesus
Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO
BELO HORIZONTE
NOVEMBRO DE 2015

Centro Federal de Educação Tecnológica de Minas Gerais

Curso de Engenharia de Computação

Avaliação do Trabalho de Conclusão de Curso

Aluna: Mariana Athayde Garcia

Título do Trabalho: Implementação de estratégias de coordenação de múltiplos robôs móveis na plataforma LEGO Mindstorms

Data da defesa: 27/11/2015

Horário: 16:00

Local da defesa: Sala 401, Prédio 17 do CEFET-MG - Campus II

O presente Trabalho de Conclusão de Curso foi avaliado pela seguinte banca:

Professor Tales Argolo Jesus– Orientador
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Professor Ramon da Cunha Lopes – Membro da banca de avaliação
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Professor Bruno André Santos – Membro da banca de avaliação
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Dedico este trabalho à minha família, à Joicimara Araújo, aos meus amigos, a meu orientador Tales. A confiança de vocês em mim é o que me fez chegar aqui. Muito obrigada.

Agradecimentos

Quero agradecer à minha família pela dedicação, zelo, carinho, apoio e confiança, sem vocês eu nada seria. Em especial, quero agradecer à minha mãe, Andrea Athayde, por acreditar em mim e me amar incondicionalmente. Ao meu irmão, por sempre estar ao meu lado, me apoiando sempre. A minha vó, por ser essa pessoa especial na minha vida. E ao meu pai por todo o apoio.

Agradeço a Joicimara Araújo, por todo o incentivo e apoio. Obrigada por acreditar.

Agradeço a Rosinei, por tudo. Aos meus amigos, por todas as conversas e todos os momentos de apoio e de brincadeiras fúteis que me ajudaram a abstrair em momentos de tensão, vocês são a minha segunda família.

Ao meu orientador Tales Argolo, pela paciência e apoio. À professora Anolan, pela ajuda e direção. A meus colegas de trabalho, em especial para minha supervisora Paula pela compreensão e apoio. Sem essas pessoas eu não teria conseguido. Muito obrigada.

A todos os meus professores do CEFET e ao CEFET e a FAPEMIG, pela oportunidade de desenvolvimento acadêmico, profissional e pessoal.

Obrigada a todos.

"O começo de todas as ciências é o espanto de as coisas serem o que são." (Aristóteles)

Resumo

Com o avanço da tecnologia e a participação cada vez mais frequente de robôs na nossa sociedade, os estudos na área da robótica vem ganhando ênfase e vem promovendo e conquistando seu espaço no mundo contemporâneo. Do ponto de vista da robótica móvel, existe uma grande área de atuação e problemas que podem ser solucionados de forma a facilitar e proteger a vida das pessoas. Este trabalho tem como objetivo o estudo de estratégias de controle de formação, assim como, a implementação dessas estratégias em uma plataforma de relativo baixo custo e didática. O controle de formação é essencial para sistemas robóticos multiagente pois permite que cada robô esteja em seu devido lugar no momento certo. Existem diversas maneiras de se implementar um sistema como este, tanto do ponto de vista do sistema distribuído e sua rede de comunicação, quanto do ponto de vista de controle e realimentação das malhas. Foram escolhidos dois tipos de formações diferentes e uma estratégia que pode ser adaptada para ambos os problemas. Os dois problemas abordados neste trabalho consistem em sincronismo em paralelo de uma frota de robôs, em que eles devem se alinhar paralelamente e seguir se deslocando em linha reta, como em um problema de varredura em paralelo, e o outro problema é fazer com que a frota de robôs localize e circule um alvo e se distribua de forma balanceada de acordo com o número de robôs da frota. Neste trabalho não se teve como intuito a implementação de um tratamento de colisão entre os diferentes robôs, nem mesmo se pretendeu interagir com o ambiente e evitar a colisão com obstáculos externos. Para cumprir com os objetivos deste trabalho, foi projetado um sistema de controle em cascata e foram realizados diversos experimentos com diferentes controladores. Os resultados mostram que os *encoders* da própria plataforma utilizados para realizar a odometria do robô são suficientemente precisos para que sejam utilizados no sistema de localização dos robôs e que as estratégias adotadas foram eficientes para que o time de robôs não só se alinhasse paralelamente, como também localizasse e circulasse um alvo reajustando a sua configuração em função do número de robôs.

Palavras-chave: Estratégias de controle de formação. Robótica móvel. Lego Mindstorms.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

CEFET-MG	Centro Federal de Educação Tecnológica de Minas Gerais
DECOM	Departamento de Computação
IDE	*Integrated Development Environment*
NBC	Next Byte Codes
NXC	Not eXactly C
P	Controlador Proporcional
PI	Controlador Proporcional Integral
PID	Controlador Proporcional Integral Derivativo
VANT	Veículo Aéreo Não Tripulado

Lista de Símbolos

R	Raio de distância do alvo
θ	Ângulo de orientação no plano cartesiano
v	Velocidade linear
ω	Velocidade angular
(x_a,y_a)	Coordenadas do alvo no plano cartesiano
v_d	Velocidade linear desejada
ω_d	Velocidade angular desejada
(x_r,y_r)	Coordenadas do robô no plano cartesiano
(x_d,y_d)	Coordenadas desejadas do robô no plano cartesiano
e_r	Vetor de erro de posição
r_d	Vetor de distância da origem do plano cartesiano ao ponto desejado
r_r	Vetor de distância da origem do plano cartesiano ao robô
e_x	Erro de posição no eixo x
e_y	Erro de posição no eixo y
θ_d	Ângulo de orientação desejado no plano cartesiano
θ_r	Ângulo de orientação real do robô no plano cartesiano
e_θ	Erro do ângulo de orientação
T	Período de rotação ao redor do alvo
ω_c	Velocidade angular passada para primeira malha de controle
ω_r	Velocidade angular real do robô
ω_{dr}	Velocidade angular desejada da roda direita
ω_{dl}	Velocidade angular desejada da roda esquerda
ω_{rr}	Velocidade angular real da roda direita
ω_{rl}	Velocidade angular real da roda esquerda

e_{wr}	Erro de velocidade angular da roda direita
e_{wl}	Erro de velocidade angular da roda esquerda
pwm_r	Comando de giro da roda direita
pwm_l	Comando de giro da roda esquerda
R_p	Raio das rodas
L	Distância entre as rodas

Sumário

1 Introdução

Atualmente, é cada vez mais frequente a participação de robôs na nossa sociedade, desde em segmentos da indústria, onde esses vêm se mostrando uma solução tanto econômica quanto eficiente, como também em salas cirúrgicas e no nosso cotidiano, na busca de facilitar ainda mais as tarefas (CHEN et al., 2002; CARLES; HERMOSILLA, 2008). Entretanto, existem situações em que a utilização de um único robô é uma solução um tanto quanto lenta e muitas vezes inviável.

Dentro deste panorama, surge o interesse cada vez mais crescente pelo estudo, não só da robótica, mas também de um segmento mais específico da área da inteligência artificial distribuída que é o estudo de sistemas multiagentes, que consiste em agentes autônomos que percebem a ação do ambiente e agem de acordo com a percepção da rede de agentes. Ou, segundo os autores Ramchurn et al. (2004, p. 1), "[...]sistemas multiagente são sistemas compostos de agentes autônomos que interagem entre si usando determinados mecanismos e protocolos". Como exemplo de uma dessas situações, onde um único robô se mostra uma solução inviável, pode-se citar o problema de patrulhamento de fronteira (CORRÊA; JÚNIOR, 2008) : Para proteção das fronteiras de um país, manter diversas patrulhas de policiais circundando a área se torna muitas vezes caro e ineficiente. Uma alternativa é alocar um veículo aéreo não tripulado (*VANT*) vigiando essas fronteiras, entretanto, apenas um *VANT* como vigia deixará uma grande área da fronteira desprotegida por um longo período de tempo. E é por isso que a aplicação de um conjunto de robôs, cooperando entre si, se mostra muitas vezes interessante.

De acordo com Secchi (2008) , "a robótica sempre ofereceu ao setor industrial um excelente compromisso entre produtividade e flexibilidade, uma qualidade uniforme dos produtos e uma sistematização dos processos". Mas, mais importante que maximizar a lucratividade das indústrias, a qualidade dos produtos e facilitar cada vez mais as tarefas cotidianas, os robôs permitem resguardar a vida humana, substituindo seres humanos em situações de risco. Exemplos de aplicação incluem: exploração e mapeamento de áreas desconhecidas, situações de incêndio, onde grupos de pessoas precisam apagar o fogo expondo suas vidas a um risco ou até mesmo em missões de resgate em terrenos perigosos. Daí a importância de que o sistema seja tolerante à falha de um ou mais agentes. Afinal, nessas aplicações é essencial que o objetivo seja cumprido.

1.1 Relevância do tema

Hoje em dia, há tarefas que são realizadas em diversas áreas nas quais a presença ou o envolvimento direto de pessoas é algo perigoso, ou até mesmo inviável. Sendo assim, é crescente a necessidade de se estudar outros meios de acesso a essas situações de risco, sem que isso signifique um risco à vida humana. Diante dessa problemática, o estudo de estratégias de controle de robôs móveis vêm aumentando consideravelmente. Não só para problemas que colocam em risco a vida humana, mas também problemas onde a aplicação dos robôs móveis otimizaria o tempo e eficiência da resolução destes problemas. Dentre estes problemas mencionados pode-se citar (GIRARD et al., 2004; JESUS, 2013; MARJOVI et al., 2009): o patrulhamento de fronteiras, o controle de incêndio, mapeamento de áreas desconhecidas, busca de pessoas perdidas ou detecção e monitoramento de problemas em determinado alvo, dentre outros.

Como é possível perceber são inúmeras as possibilidades de aplicação dos robôs móveis. Entretanto, devido muitas vezes à urgência e/ou à extensão da cobertura do problema é necessário modular o mesmo e redistribuí-lo em um sistema multiagente de robôs móveis. Sendo assim, surge aí mais uma demanda por estudos relativos a estratégias de controle de sistemas multiagente constituídos de robôs móveis. Um dos desafios destes sistemas é não só o controle de cada agente por si só, mas também como a frota como um todo irá se comportar, para viabilizar a resolução do problema e/ou também maximizar a eficiência na resolução do mesmo. É necessário que se garanta que os robôs não colidam entre si, e trabalhem em um sistema cooperativo de fato.

Outra questão importante é a tolerância a falhas do sistema, isto é, como o sistema irá se comportar, se reestruturar e reorganizar diante da perda de um ou mais robôs, visto que além de ser um ambiente hostil (muitas vezes desconhecido ou até dinâmico), existem outros fatores críticos, dentre eles: falha de comunicação ou desligamento de um dos agentes devido ao esgotamento de bateria.

1.2 Objetivos

Este trabalho tem como objetivo geral o estudo de estratégias de controle de formação de uma frota de robôs tendo em vista o cumprimento de uma dada missão e a implementação deste sistema em uma plataforma experimental.

1.2.1 Objetivos específicos

Este trabalho tem como objetivos específicos:

- Varredura de área em paralelo

- Localizar e circular um determinado alvo de forma balanceada
- Coordenar a frota para se reajustar conforme o tamanho da frota

1.3 Definição do Problema

Visando cumprir esse objetivo, definiram-se dois problemas. O primeiro, consiste na coordenação de uma frota de robôs para que a mesma se alinhe paralelamente e eles sigam em linha reta, como em problemas de varreduras de área em paralelo. O segundo, consiste em um controle de uma frota de N robôs, para que a mesma localize um dado alvo no espaço e o circunde a uma distância R e em um período T, que se reajustará conforme o tamanho da frota.

1.4 Infraestrutura Necessária

Para a realização deste trabalho foram utilizados dois kits da plataforma da *LEGO*®: *Lego Mindstorms*®, disponível do DECOM (Departamento de Computação do Centro Federal Tecnológico de Minas Gerais). Cada kit consiste em um microcomputador NXT de 32 bits, três motores, alguns sensores e peças de lego para montagem da estrutura do robô. Além disto, também será utilizado um computador pessoal, com a seguinte configuração: processador *intel core i7*, 8GB de memória *RAM*, 1GB de mémória dedicada e 14", com o *software MATLAB*® e a *IDE Bricx Command Center* (SOURCEFORGE, 2001) que é uma plataforma de desenvolvimento para robôs *Lego Mindstorms*® que permite utilizar a linguagem *NXC (Not eXactly C)* para programar os robôs.

2 Trabalhos Relacionados

Com o interesse cada vez mais crescente na área de robótica móvel e sistemas multiagente, tem havido uma demanda cada vez maior para estudos nestas áreas. O *Lego Mindstorms*®, devido as suas limitações e devido a limitação do seu protocolo *bluetooth* com a topologia "Mestre e Escravo", não é uma plataforma muito interessante de aplicação desses conceitos , entretanto, é uma excelente plataforma a ser utilizada nos estudos dos mesmos. Isto por que, é uma plataforma acessível, existe muita documentação auxiliar, muitos trabalhos relacionados a respeito e é muito simples de ser utilizada.

Existem muitos trabalhos distintos com sistemas multiagentes utilizando-se *Lego Mindstorms*®, com as mais diversas configurações, objetivos distintos e diferentes estruturas de rede, dentre outros. Entretanto, uma dificuldade reconhecida em todos os trabalhos é a limitação da plataforma, que possui uma quantidade de conexões e tipo de comunicação muito limitada. Para que essa limitação seja superada perde-se uma característica importante da plataforma, que é a simplicidade e facilidade de implementação.

O protocolo de comunicação disponível, por exemplo, só permite a comunicação *'Master/Slave'* realizada de forma manual. Outras configurações requerem uma implementação mais complexa que afeta o custo/benefício de se utilizar essa plataforma, por perder a característica de implementação simples. Pode-se citar como um desses trabalhos, que abordam de forma mais dinâmica e independente a comunicação entre os robôs, a tese de Martinez (2009). Seu trabalho consiste em uma sociedade que se configura de forma autônoma, ou seja, tem-se uma sociedade de robôs independentes. Quando a um ou mais robôs é atribuída uma missão, eles se agrupam com outros indivíduos, coordenando-se em um time com o objetivo de cumprir a referida tarefa.

Entre outros trabalhos relacionados a este podemos citar, Benedettelli et al. (2009) que propõe uma configuração experimental para utilizar o *Lego Mindstorms*® como ferramenta de estudos de estratégias de controle de sistemas multiagente. Ele utiliza uma frota composta por quatro robôs, uma *webcam* e o software *MATLAB*®. Com o intuito de se obter uma ferramenta de baixo custo para dar aulas de laboratório de robótica, seu trabalho consiste em propor uma configuração que permita a implementação, comparação e o estudo de diversas estratégias de controle e algoritmos. Uma configuração que, além de tudo, contemple muitos dos problemas vistos em um cenário real.

Utilizando-se de uma unidade central de controle, ele primeiro propõe quatro

robôs em pontos diferentes do espaço, orientados em qualquer sentido à circular um ponto qualquer no espaço, com auxilio da *webcam* e do computador (unidade central de comando). O que se aproxima bastante deste trabalho, diferindo-se principalmente no que diz respeito à *webcam* como o principal elemento do sistema de localização.

Outro trabalho também muito interessante que pode-se citar é o de Casini et al. (2011), no qual os autores propõem um laboratório remoto utilizando o *LEGO Mindstorms*®. O trabalho se resume a um laboratório remoto para estudos de robótica móvel, em que tem-se um espaço de cerca de 13 metros quadrados que é filmado por duas câmeras, onde os robôs ficam e podem se movimentar. Foi então desenvolvida uma interface gráfica de acesso *online* ao laboratório, através da qual os usuários podem acessar e utilizar o laboratório para o estudo de robótica móvel.

Além desses trabalhos que utilizam o *Lego Mindstorms*®, existe uma tese de doutorado do Jesus (2013) que consiste em simular uma frota de *VANT* que devem primeiro rastrear um círculo paralelo ao plano xy em uma dada latitude diferente para cada robô, depois eles devem se distribuir de forma que suas projeções no plano xy estejam balanceadas e após esta etapa, devem convergir para a mesma latitude a fim de rastrear o círculo. O que é bem próximo a um dos problemas resolvidos neste trabalho.

3 Fundamentação Teórica

Para que se possa compreender o desenvolvimento deste trabalho é importante, primeiramente, conhecer alguns conceitos. Para tanto, faz-se neste capítulo uma breve contextualização teórica para auxiliar o leitor ao longo deste trabalho.

3.1 Modelos Matemáticos: Sistemas Não Holonômicos

Os robôs utilizados neste trabalho são considerados modelos não holonômicos e para entender o que é um modelo matemático não holonômico é necessário entender o que é o grau de liberdade de um sistema. Como é visto na literatura (TAYLOR et al., 2013), o grau de liberdade de um sistema é igual ao "número de coordenadas que podem variar independentemente em um pequeno deslocamento". Dito isto pode-se dizer que um sistema holonômico é um sistema onde o número de coordenadas utilizadas para descrever as configurações do sistema é igual ao grau de liberdade do sistema (TAYLOR et al., 2013). Ou seja, o sistema pode se movimentar livre e independentemente em qualquer um dos seu eixos (os eixos referentes à configuração do sistema).

Sendo assim, os sistemas não-holonômicos são sistemas em que pode-se chegar à qualquer outro ponto do espaço, entretanto, com restrições, visto que as variáveis não podem se mover independentemente. Como exemplo de um sistema não-holonômico podemos citar um veículo, que pode alcançar qualquer ponto do espaço bidimensional, entretanto, para alcançar um ponto qualquer deslocado apenas em seu eixo x é necessário um movimento não só ao longo do seu eixo x mas, também do seu eixo y, já que um veículo não pode se mover lateralmente (GOUVÊA, 2011).

3.2 Controle de Formação

Com o crescente avanço da robótica surgiu também, o interesse por sistemas robóticos cooperativos, onde muitos robôs agem em conjunto para alcançar o mesmo objetivo. Para que um sistema multiagente possa executar uma tarefa em conjunto é preciso que cada robô esteja na posição correta, para tal, é necessário o controle de formação. O controle de formação é essencial para sistemas robóticos multiagente pois permite que cada robô esteja em seu devido lugar no momento certo. Existem diversos tipos de estruturas de formação, tanto no que diz respeito à formação física da rede, como do ponto de vista lógico da rede. Como é visto na literatura (MARTINEZ, 2009), pode-se classificar uma rede multiagente de diversas formas: homogênea ou não, no que diz respeito aos tipos de unidades; centralizada ou não, no que diz respeito à estrutura

da rede como indivíduo (até mesmo se a rede é formada por indivíduos independentes ou se é o mesmo robô); quanto à estrutura organizacional, dentre outros.

Do ponto de vista físico podemos citar alguns tipos de estruturas de formação, como referenciado por Balch e Arkin (1998), tais como, estrutura em *line*, *column*, *diamond* e *wedge*, como ilustrado na Figura 1. A primeira delas consiste em uma formação em linha horizontal (*line*) como, o próprio nome revela. A segunda, em uma linha vertical (*column*), a terceira *diamond*, que consiste em uma rede em formato de um losango e a quarta, *wedge* em formato de *'V'*, o que se parece com a estrutura de um *flock*, uma revoada de pássaros.

Figura 1 – Controle de Formação: classificação quanto à estrutura física

(a) *Line*,(b) *Column*,(c) *Diamond* e (d) *Wedge*

Fonte: Balch e Arkin (1998)

Além disso tem-se a classificação quanto à estrutura da rede lógica, centralizada, descentralizada e híbrida e quanto a seu grupo de arquitetura.

3.3 Controle Proporcional Integral Derivativo e a Técnica de Controle em Cascata

O controlador Proporcional Integral Derivativo, ou simplesmente, *PID* consiste em uma técnica com ações proporcionais, integrais e derivativas. Segundo Ogata (1970) e Franklin et al. (1998), a ação proporcional (*P*), é dada por

$$p(t) = k_p e(t) \tag{1}$$

em que $p(t)$ é a saída do controlador, k_p é a constante de ganho responsável por ampliar a entrada $e(t)$, que é também o erro atuante no sistema naquele instante. A ação proporcional reduz o erro na saída do sistema, entretanto, ainda tem-se um pequeno erro estacionário. A ação integral

$$i(t) = k_i \int_0^t e(t)dt \tag{2}$$

varia proporcionalmente a taxa de erros acumulados no sistema, podendo eliminar o erro em estado estacionário. Por se tratar de uma ação proporcional ao acumulo de erros do sistema, pode também, causar saturação do atuador. Já a ação derivativa

$$d(t) = k_d \frac{de(t)}{dt} \tag{3}$$

tem sua saída proporcional a taxa de variação do erro atuante, tendo um caráter antecipatório, por outro lado, amplifica o sinal de ruído. O controlador proporcional integral derivativo tem sua equação dada por:

$$c(t) = k_p e(t) + k_i \int_0^t e(t)dt + k_d \frac{de(t)}{dt} \tag{4}$$

reunindo as três ações: proporcional, integral e derivativa, assim como suas vantagens.

Uma técnica de controle importante que também foi utilizada neste trabalho é a técnica conhecida como controle em cascata, esta técnica consiste em passar como referência para o controlador interno, a saída do controlador mais externo, que possui uma referência independente.Como será visto mais a frente, é o caso deste trabalho.

3.4 Plataformas

Para o desenvolvimento deste trabalho foram consideradas diversas plataformas de desenvolvimento e gerenciamento de robôs móveis que são compatíveis com *Lego Mindstorms*®. Uma delas é o *ROS*(*Robotic Operating System*) é um sistema operacional robótico *opensource* que dispões de ferramentas e bibliotecas desenvolvidas para criar aplicações robóticas. Esse *software* permite a comunicação entre o computador e o *Lego*, permite formar uma rede centralizada de até quatro robôs e, embora possua funções específicas para a área de robótica, ainda não dispõe de muito material publicado, por isso, optou-se por não utilizá-lo.

O *Simulink* possui um pacote compatível com o *Lego Mindstorms*® que permite desenvolver e simular algoritmos para plataformas robóticas. Entretanto, só é permitida a comunicação via *bluetooth* entre dois robôs. Portanto, o *Simulink* não foi escolhido pois, visa-se implementar futuramente as estratégias de controle de formação em um sistema com quatro robôs

O *LABVIEW* possui um módulo para programar e controlar robôs *Lego Mindstorms*®. É uma ferramenta que permite a comunicação entre o robô e o computador e entre os robôs. Entretanto, é necessário possuir uma licença para utilizar desta ferramenta, por isso, a ferramenta foi descartada como ferramenta para se utilizar neste trabalho.

O *MATLAB*® possui uma ferramenta *opensource* desenvolvida para controlar robôs *Lego Mindstorms*®*NXT*, conhecida como *RWTH Aachen NXT Toolbox*. Permite a comunicação entre robô e o computador ou entre um conjunto de até 4 robôs, no modelo de comunicação mestre/escravo.

Por fim, o ambiente de desenvolvimento integrado conhecido como *BRICX Command Center* é utilizado para o desenvolvimento de aplicações para todas as versões do *Lego Mindstorms*®, do *RCX* ao *EV3*, incluindo o modelo utilizado neste trabalho que

é o *NXT*. Suporta diversas linguagens como: *Not eXactly C* (NXC), *Next Byte Codes* (NBC) e permite o desenvolvimento em *java*, por meio do *firmware LeJos* (SOURCEFORGE, 2001). Dentre as opções, a linguagem NXC foi escolhida pela quantidade de materiais de pesquisa e simplicidade.

4 Metodologia

Para a realização deste trabalho, primeiramente fez-se um estudo das estratégias de controle de formação de robôs móveis, das possibilidades de implementação dessas estratégias na plataforma a ser utilizada (no caso, o *Lego Mindstorms*®) e da viabilidade de modelar e implementar o sistema como um sistema distribuído descentralizado. Neste tipo de topologia de comunicação não há um mestre definido, apenas uma sociedade de robôs que conforme precisam executar uma tarefa, vão se coordenando e formando um sistema multiagente.

Após realizados os estudos, concluiu-se que, apesar de viável como pode ser visto no capítulo 2, a implementação de um sistema distribuído descentralizado seria muito complexa para ser abordado no período proposto para a realização deste trabalho, que tem como objetivo principal o estudo de estratégias de controle de formação de múltiplos robôs móveis. Para tanto, foi adotada uma estrutura de rede centralizada do tipo 'Mestre/Escravo', utilizando o protocolo de comunicação *bluetooth*, o qual é suportado pela plataforma e pela linguagem *NXC*.

Existem diversas maneiras de se implementar um sistema como este, tanto do ponto de vista do sistema distribuído e sua rede de comunicação, quanto do ponto de vista de controle e realimentação das malhas. Foram escolhidos dois tipos de formações diferentes e uma estratégia que pode ser adaptada para ambos os problemas.

Para estabelecer o modelo matemático do problema, foi considerado o modelo de robô mostrado na Figura 2, que consiste em um modelo não holonômico, onde tem-se duas rodas unidirecionais e uma roda orientável. Para definição do modelo matemático levou-se em consideração que os *encoders* seriam utilizados para odometria e então, foram definidas as restrições da modelagem matemática do problema que, desconsidera problemas como: saturação do atuador, derrapagem das rodas, os erros de medição dos *encoders*, bem como as limitações da plataforma.

Após modelar o problema, foram feitas as implementações e os testes, onde seria possível observar se a odometria feita seria suficiente para localizar cada robô do sistema multiagente com precisão e tornar a solução viável. Deve-se levar em consideração que para tanto, foram utilizados os *encoders* óticos acoplados aos motores do *kit Lego Mindstorms*® os quais possuem uma imprecisão que é da ordem de $\pm$ 1 grau por rotação.

Concomitantemente, foram realizadas simulações, com o auxílio do software *MATLAB*® a fim de prever e validar a modelagem feita do problema, desconsiderando os problemas práticos como falha na comunicação e falta de sincronismo entre os robôs,

Figura 2 – Robô diferencial de duas rodas

erros dos *encoders* e problemas como saturação do motor e derrapagem das rodas. Outro objetivo dessas simulações é possibilitar que se faça uma comparação entre o sistema real e o seu modelo idealizado.

4.1 Modelo Matemático

Para introduzir a dinâmica dos robôs móveis utilizados, inicialmente o robô será considerado como um uniciclo, um elemento pontual. O comportamento dinâmico de um robô móvel não-holonômico do tipo uniciclo, desconsiderando-se as forças e os torques causadores do movimento, é comumente descrito pelas equações abaixo, como na tese do Vieira (2005):

$$\dot{x} = v\cos(\theta) \tag{5}$$

$$\dot{y} = v\sin(\theta) \tag{6}$$

$$\dot{\theta} = \omega \tag{7}$$

sendo:

- (x,y) as coordenadas da posição do robô no plano cartesiano;
- θ o ângulo de orientação do robô no plano cartesiano;
- v e ω indicam a velocidade linear e angular do robô, respectivamente, que são as entradas desse sistema dinâmico.

A discretização das equações 5, 6 e 7:

$$x_{k+1} = x_k + Tv_k \cos(\theta_k) \tag{8}$$

$$y_{k+1} = y_k + Tv_k \sin(\theta_k) \tag{9}$$

$$\theta_{k+1} = \theta_k + T\omega_k \tag{10}$$

permitem obter as equações (11a), (11b) e (11c) modeladas com base no robô real, que não é um elemento pontual no espaço e sim, um robô diferencial de duas rodas com uma restrição não holonômica. Elas serão utilizadas para gerar as trajetórias dos robôs no ambiente *MATLAB*® e verificar se elas são compatíveis com o caminho percorrido pelos robôs no mundo real.

As equações discretas que descrevem a posição do robô em função dos giros de cada roda são dadas por:

$$x_{k+1} = x_k + \left(\frac{D_r + D_l}{2}\right) \cos(\theta_k) \tag{11a}$$

$$y_{k+1} = y_k + \left(\frac{D_r + D_l}{2}\right) \sin(\theta_k) \tag{11b}$$

$$\theta_{k+1} = \theta_k + \left(\frac{D_r - D_l}{L}\right) \tag{11c}$$

sendo, y_{k+1} e y_k a coordenada y do robô no instante k e no instante $k+1$; θ_{k+1} e θ_k o sentido do robô no instante k e no instante $k+1$; D_r e D_l a distância que a roda direita e esquerda percorreram no instante de tempo entre k e $k+1$, respectivamente; E L o tamanho do eixo das rodas do robô;

Em que a Equação (11a) indica que a posição atual do robô em x é igual a posição do robô anterior em x mais a distância percorrida pelo centro de massa do robô, que é dada pela média da distância percorrida por ambas as rodas, multiplicado-se essa distância pelo cosseno do ângulo de orientação anterior do robô. Analogamente, a Equação (11b) descreve a posição em y atual do robô como a posição em y anterior do robô mais a média das distâncias percorridas pelas rodas multiplicada pelo seno do ângulo de orientação do robô no instante anterior. Já a Equação (11c) indica que o ângulo atual de orientação do robô é igual ao ângulo anterior mais a diferença entre a distância percorrida pelas rodas divido sobre a distância entre as rodas.

Tendo em vista que a odometria será feita utilizando-se os *encoders* da própria plataforma e o sistema será realimentado com essas medidas, é de extrema importância que a trajetória descrita no mundo real e a registrada pelo robô, utilizando-se os *encoders*,

sejam significativamente semelhantes. Caso contrário, a realimentação do sistema estará incorreta, comprometendo seriamente, e por que não dizer inviabilizando, o controle do sistema.

Dado que a velocidade linear e angular de um robô como o do modelo utilizado neste trabalho é dada pela velocidade angular de cada uma das suas rodas unidirecionais, tem-se nas equações (12) e (13) as funções que descrevem a velocidade linear e angular do robô a partir da velocidade angular de suas rodas. Em que a velocidade linear é dada por

$$v = \frac{(\omega_r + \omega_l)rp}{2} \tag{12}$$

Sendo a média da soma das velocidades de cada roda dadas em rad/s, multiplicado pelo raio da roda. E a velocidade angular é dada pela diferença das rodas direita e esquerda multiplicada pelo raio da roda e dividido pelo tamanho do eixo, como indicado abaixo:

$$\omega = \frac{(\omega_r - \omega_l)rp}{L} \tag{13}$$

em que:

- v é a velocidade linear do robô;
- ω é a velocidade angular do robô;
- ω_r é a velocidade angular da roda direita do robô;
- ω_l é a velocidade angular da roda esquerda do robô;
- r_p é o raio da roda do robô;
- L é a distância entre as rodas unidirecionais do robô.

4.2 Problema 1: Varredura de área em paralelo

O primeiro problema consiste em: dada uma frota de *N* robôs, esses robôs devem se alinhar horizontalmente e seguir em linha reta, andando paralelamente, utilizando a estrutura já citada na seção 3.2, denominada *"In Line"*. Primeiro, este problema será modelado considerando a estrutura de comunicação da rede e permitindo apenas velocidades lineares em que o robô pode se deslocar em linha reta e podendo imprimir velocidades negativas e positivas.

Considerando-se *N* robôs separados por uma distância Δy no eixo y, cada um em um ponto distinto no eixo x, como mostrado na Figura 3 (a), a tropa deve se alinhar paralelamente e seguir andando paralelamente com uma velocidade v_d constante.

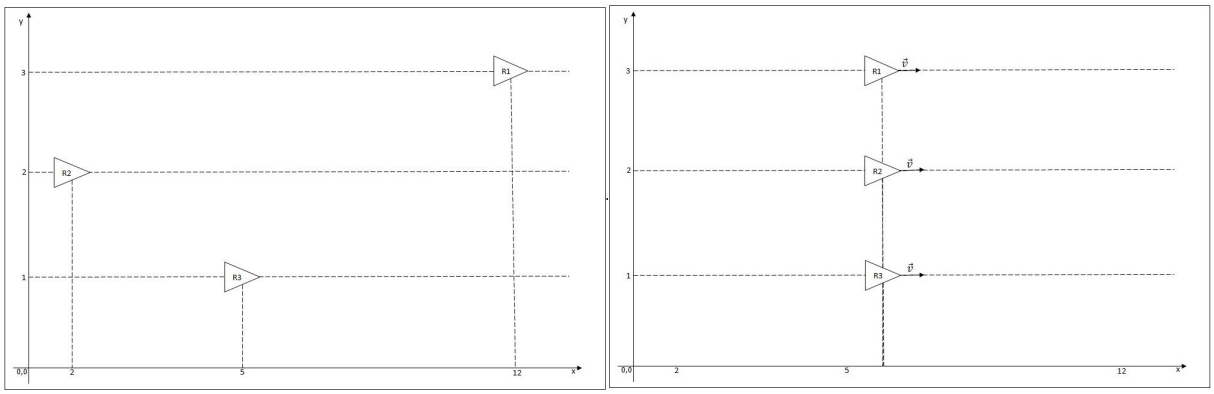


Figura 3 – Problema 1: Fig.(a): Sistema no instante t_i. Fig.(b): Sistema no instante t_2, em que $t_2 > t_1$.

Para alinhar os robôs pode-se usar diferente equações para se definir o erro de posicionamento do robô. Embora não seja a intenção deste trabalho implementar uma rede distribuída, uma forma interessante de se fazer isto é levar em consideração uma rede deste tipo, onde um robô não possui a localização de todos os outros robôs e não precisa de uma dependência com um determinado mestre. Desta forma, podemos definir os erros entre os robôs vizinhos conforme a estrutura da rede, como será explicado mais a diante.

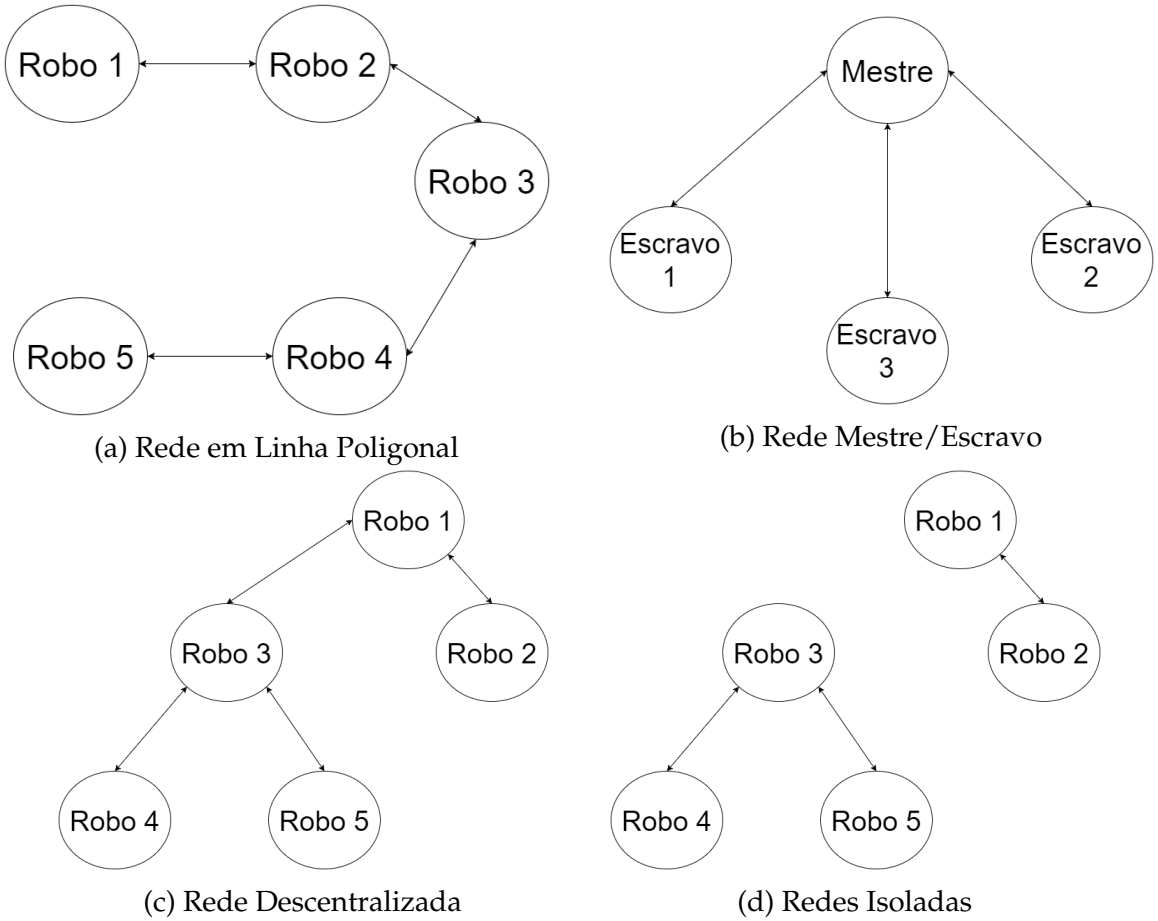
Existem diversos tipos de estruturas de rede como mostrado na Figura 4, em que cada círculo indica um robô e as setas indicam a comunicação que ocorre entre eles. Ou seja, na Figura 4a o robô 1 troca mensagens somente com o robô 2, que por sua vez, troca mensagens com o robô 1 e 3 e assim sucessivamente. Na Figura 4b a rede é como a implementada neste trabalho, em que o mestre se comunica com todos os escravos e os escravos não trocam mensagem entre sí. Já na Figura 4c a rede é descentralizada, o robô 1 troca mensagens com o robô 2 e 3. O robô 3 por sua vez, troca mensagens com o robô 4 e 5. E por fim, na Figura 4d há duas redes, sendo que uma não troca mensagem com a outra.

Sendo assim, uma forma de se alinhar a frota seria definir a velocidade de cada robô$_j$ de acordo com o erro entre a sua posição e a posição dos robôs vizinhos, aqueles com os quais o robô$_j$ troca mensagens.Assim, a velocidade do j-ésimo robô é dada por:

$$v_j = v_d + k_j \sum_{i=1}^{n_j} e_{i,j} \tag{14}$$

em que:

- j é o número do robô;
- n_j é o número de vizinhos do j-ésimo robô;
- $e_{i,j}$ é a diferença entre a posição no $eixo\ x$ do j-ésimo robô e seu i-ésimo vizinho.



(a) Rede em Linha Poligonal

(b) Rede Mestre/Escravo

(c) Rede Descentralizada

(d) Redes Isoladas

Figura 4 – Estruturas de Rede

Supondo uma tropa de três robôs utilizando a rede centralizada utilizada neste trabalho, onde o robô 2 é o mestre e todos os robôs estão orientados no mesmo sentido do eixo x, as velocidades de cada robô devem variar de acordo com o erro de posicionamento do mesmo no eixo x, conforme a distância entre os robôs, como mostrado nas equações abaixo:

$$v_1 = v_d + k_1 e_{2,1} \tag{15}$$

$$v_2 = v_d + k_2(e_{1,2} + e_{3,2}) \tag{16}$$

$$v_3 = v_d + k_3 e_{1,3} \tag{17}$$

sendo,

- v_1, v_2 e v_3 são as velocidades que cada robô deve possuir para assumir a formação *in line*;
- v_d é a velocidade que a frota deve assumir após estar alinhada;
- k_1,k_2,k_3 são as constantes de ganho proporcional do controlador;

- E as variáveis de erro são calculadas, obtendo se a diferença entre a posição dos robôs no eixo x, como demonstrado na equação abaixo:

$$erro_{i,j} = x_i - x_j \tag{18}$$

Essas equações fazem com que o robô mais adiantado da frotas e desloque com uma velocidade menor que os outros robôs, tendendo a se aproximar dos mesmos; em contrapartida os robôs que estão "atrasados" se deslocam com uma velocidade maior, até que o erro entre eles seja zero e os mesmos caminhem com velocidade constante.

É importante notar que ao abordar o problemas desta maneira elimina-se a possibilidade de colisão entre os robôs, que estão inicialmente separados no eixo y e alinhados no mesmo sentido, e assim seguem, visto que os mesmos não imprimem velocidade angular. Desta forma, tem-se um problema bem didático para ser abordado inicialmente.

Outra forma de abordar esse problema é pensar que existe uma reta perpendicular ao eixo x (supondo que deseja-se deslocar a frota no mesmo sentido que o eixo das abscissas), tal que essa reta é dada pela equação

$$x_d(t) = \frac{(max(x_{frota(t)}) + min(x_{frota(t)}))}{2}; \tag{19}$$

onde:

- x_{frota} é um vetor com todas as coordenadas do eixo x dos robôs da frota;

Em que o valor de x desejado (x_d) é a media das posições entre os robôs mais distantes entre si (considerando-se o posicionamento com relação ao eixo das abscissas). Todos os robôs devem alcançar a posição (x_d,y_j), em que o valor de y_j é a coordenada vertical do j-ésimo robô, sendo que $y_p \neq y_q$, se $p \neq q$. Sendo assim, todos os robôs seriam alinhados e a partir dai seguiriam em linha reta, cumprindo o objetivo.

4.3 Problema 2: Circulando um alvo

O segundo problema consiste em guiar um time de robôs a localizar e circular, a uma distância *R*, um alvo localizado em uma determinada posição (*x*,*y*) do plano, como mostrado na Figura 5. E tem como objetivo secundário ajustar a formação da tropa de robôs que deve se reajustar de acordo com o número de robôs N, para que continue cobrindo com eficiência a fronteira, ou seja, caso um ou mais robôs saiam da rede, a frota irá se reajustar para que cada robô tenha a mesma distância entre si e assim, não fique uma grande parte da fronteira sem cobertura, como demostrado na Figura 6, que

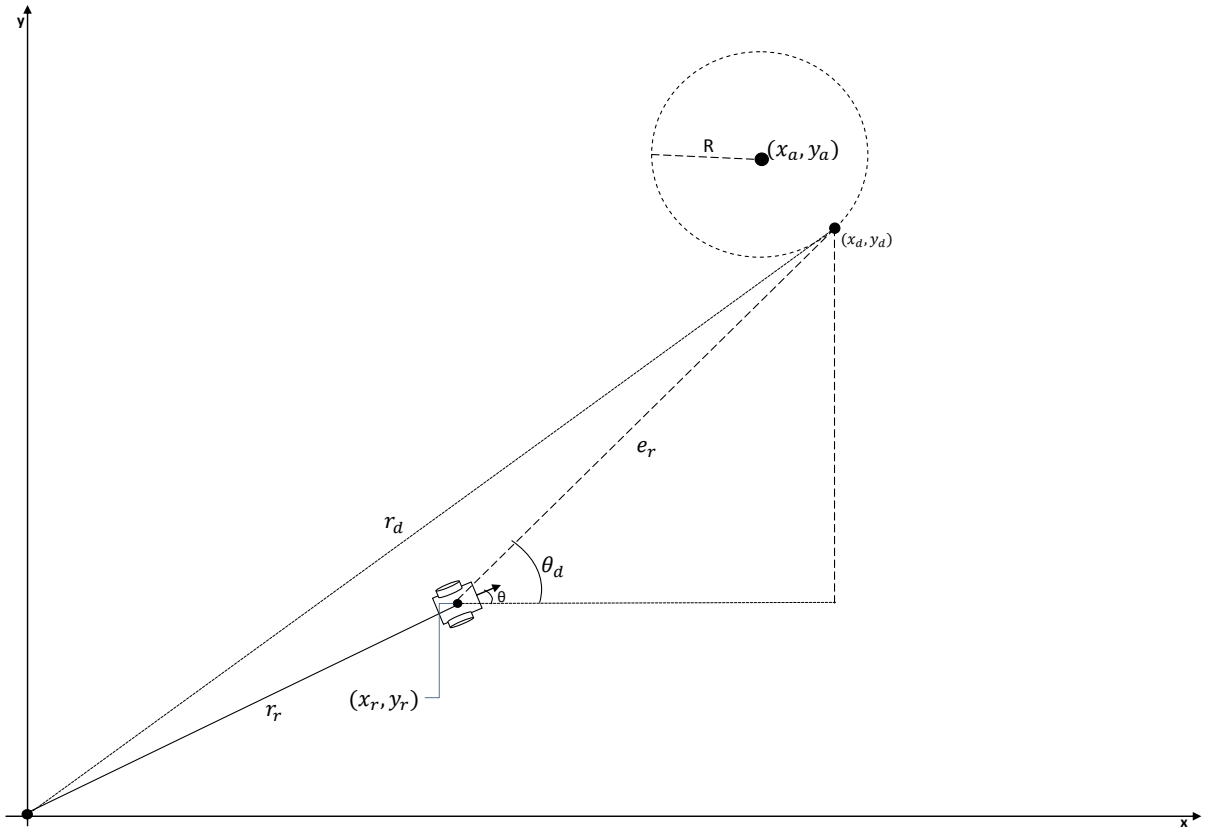


Figura 5 – Esquema do sistema do ponto de vista de apenas um agente

representa uma frota de quatro robôs andando ao redor do alvo, quando então, um dos robôs falha. O sistema, então, se reajusta para adaptar-se à rede de apenas três robôs.

Para calcular a posição desejada de cada robô ao redor do alvo, usam-se as equações (20a), (20b) e (20c) que como pode ser visto levam em consideração a posição do alvo (x_a,y_a), o raio (R) que é a distância com que se deseja circular o alvo, a velocidade angular (ω) com que se deseja percorrer o perímetro (a essa velocidade está associado um período T dado por $2\pi/\omega$) e, além disso, levam em consideração o tamanho da frota e a numeração de cada robô na mesma. O segundo termo adicionado a ωt significa exatamente a distância angular que cada par de robôs adjacentes terá entre si, que é de $\frac{2\pi}{n}$.

$$x_d = x_a + R\,cos\left(\omega t + \frac{(j-1)2\pi}{n}\right) \tag{20a}$$

$$y_d = y_a + R\,sin\left(\omega t + \frac{(j-1)2\pi}{n}\right) \tag{20b}$$

$$j = 1,...,n \tag{20c}$$

sendo:

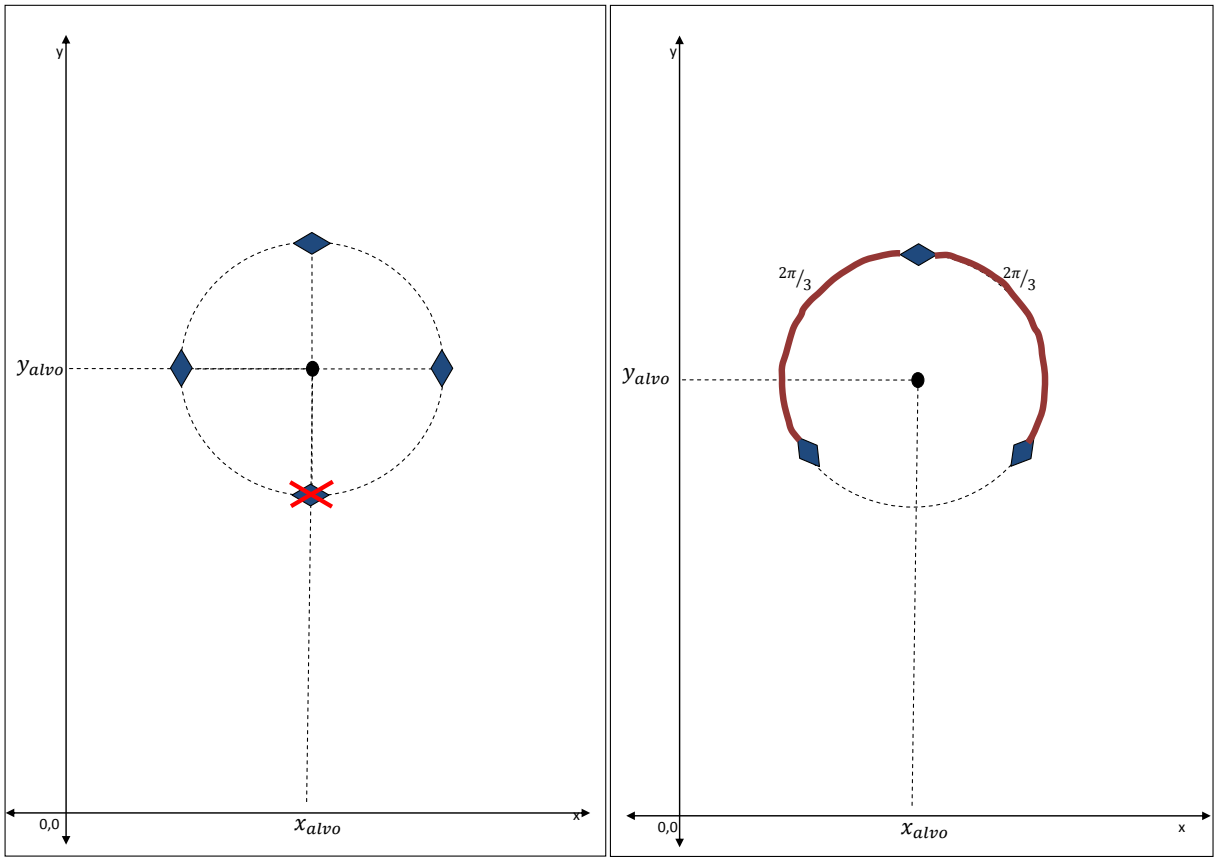


Figura 6 – Reconfiguração do sistema mediante falha de um agente

- (x_a,y_a) é a posição do alvo;
- R é o raio do círculo em torno do alvo;
- ω é a velocidade angular com que deseja-se percorrer o círculo;
- t é o tempo decorrido em segundos;
- n é o número de robôs da frota;
- j é a numeração do robô na frota.

À medida que o sistema se estabiliza, a trajetória do robô tende a convergir para um movimento circular uniforme ao redor do alvo, conforme está ilustrado na Figura 7, ou seja, a velocidade linear (v) tende a se igualar à velocidade angular (ω) vezes o raio (R) de distância do alvo.

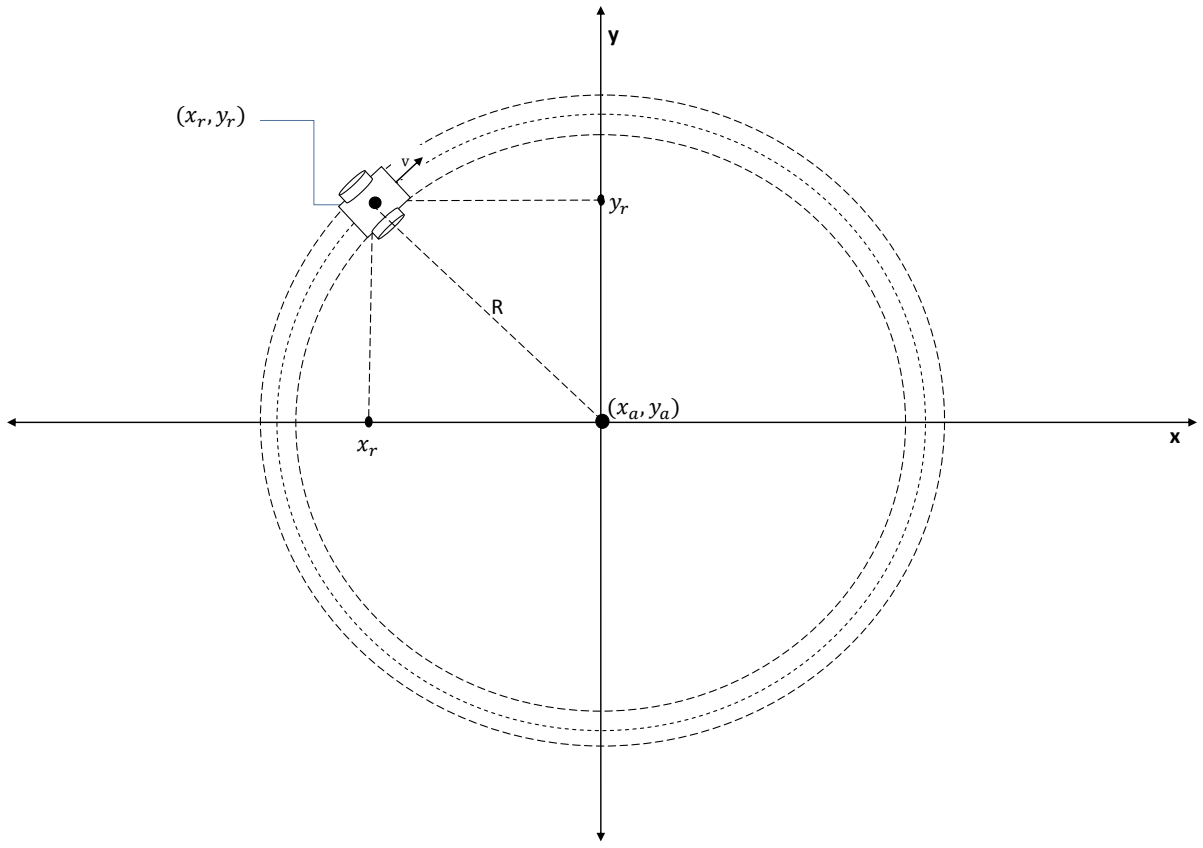


Figura 7 – Representação do sistema estabilizado com um único agente

5 Abordagem e Modelagem do Problema: Malhas de Controle

Como já dito no Capítulo 4, foi escolhida uma estratégia que atende à resolução de ambos os problemas a qual, consiste em três malhas de controle: a primeira e mais interna será responsável pelo controle da velocidade angular de cada roda, para se chegar à posição (x,y) desejada; a segunda, malha intermediária, será responsável para que cada robô chegue à um determinado ponto (x,y) no espaço, portanto, será responsável por corrigir o posicionamento do robô (A primeira e a segunda malhas serão implementadas em cada um dos robôs da frota); a terceira malha e, portanto, a mais externa, é responsável pela coordenação da frota, fornecendo a cada robô as informações necessárias para que o mesmo saiba o ponto (x,y) que deve rastrear para consolidar e manter a formação.

Faz-se então neste capítulo um detalhamento das malhas de controle utilizadas e das modificações realizadas para que a estratégia atenda a ambos os problemas, bem como uma descrição de como funcionam as malhas responsáveis pela comunicação e pelo controle de formação para a resolução de cada um dos problemas. Faz-se também um levantamento dos possíveis problemas que podem surgir na implementação do sistema.

5.1 Malha de Controle 1: Velocidade Angular das Rodas

Como já dito anteriormente, este sistema de controle consiste em um controle de três malhas, e agora será abordada a primeira malha. Ela controla os motores para atingir a velocidade angular desejada de cada roda (ω_{dr},ω_{dl}). Ou seja, a malha de controle vai receber a velocidade linear e angular que se deseja imprimir no robô e retornar a "potência"que deve ser aplicada a cada um dos motores para se atingir as velocidades desejadas, como pode ser observado na Figura 8.

É importante ressaltar que como o robô não é um elemento pontual[1], como considerado na Seção 4.1 ao descrever as equações do modelo, temos que descrever a velocidade angular (ω) e linear (v) do robô em função de cada roda, para sabermos a potência a ser aplicada em cada uma delas para que o robô obtenha a velocidade e o sentido desejados.

A partir daí o microcomputador embarcado calcula a velocidade angular desejada de cada roda, como indicado nas equações abaixo que, por sua vez, são oriundas

[1] Veja a Figura 7 para visualizá-lo como um elemento não pontual, que depende da variação de velocidade de cada roda para definir a velocidade e o sentido do robô.

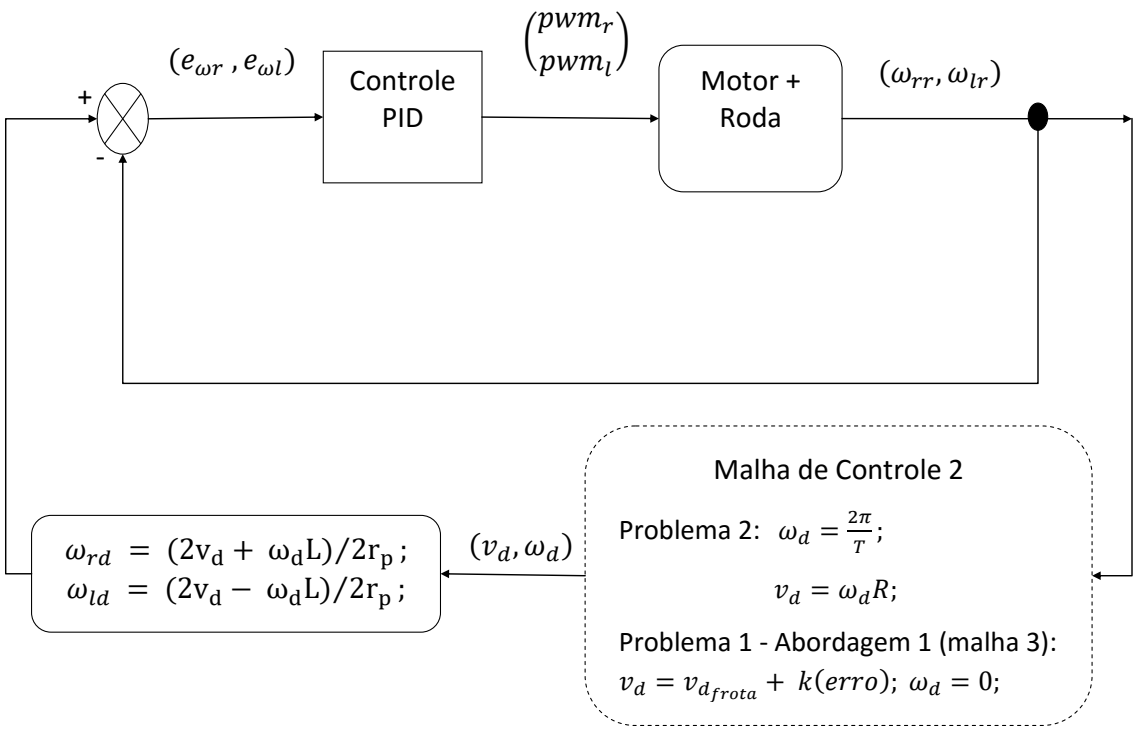


Figura 8 – Primeira malha de Controle do Sistema - Controle da velocidade angular

das equações (12) e (13):

$$\omega_{dr} = \frac{2v + \omega_d L}{2r_p} \tag{21a}$$

$$\omega_{dl} = \frac{2v - \omega_d L}{2r_p} \tag{21b}$$

onde:

- v_d é a velocidade linear desejada do robô (m/s);
- ω_{dr} é a velocidade angular desejada da roda direita (rad/s);
- ω_{dl} é a velocidade angular desejada da roda esquerda (rad/s);
- r_p o raio da roda (m);
- L o tamanho do eixo das rodas do robô (m).

Com a velocidade angular de cada roda do robô, estimada a partir dos dados fornecidos pelos *encoders*, é calculado o erro das velocidades dado por:

$$e_{wr} = \omega_{dr} - \omega_{rr} \tag{22}$$

$$e_{wl} = \omega_{dl} - \omega_{rl} \tag{23}$$

onde:

- e_{wr} e e_{wl} são os erros da velocidade angular da roda direita e esquerda, respectivamente;
- ω_{rr} e ω_{rl} é a velocidade angular real da roda direita e esquerda, respectivamente.

E o controlador os retorna as ações de controle que serão passadas como potência para cada uma das rodas.

5.2 Malha de Controle 2: Posicionamento

Para que a malha de controle 3, que consiste no controle de formação do sistema multiagente funcione, primeiramente é necessário implementar o controle de posicionamento de cada robô. Ou seja, para que o robô cumpra a missão, é necessário que a malha de controle de formação passe para cada robô os parâmetros necessários para o ajuste da estrutura, tais como, posicionamento, velocidade e número de robôs da frota. Cada robô por sua vez, deve conseguir se posicionar corretamente de acordo com a posição recebida pela malha de controle 3, o que só será possível se a malha de controle 1 conseguir controlar adequadamente os motores para que eles imprimam a velocidade correta em cada roda.

Como pode ser visto na Figura 5, o problema a ser resolvido pela segunda malha de controle consiste em, dado um sistema de coordenadas cartesianas, onde têm-se um robô móvel de posição (x_r,y_r), cuja orientação (θ) é indicada em relação ao eixo x do sistema, fazer com que o robô chegue ao ponto desejado (x_d,y_d), recebido da malha de controle 3. Para tanto, a malha 2 funciona da seguinte maneira: Recebe da malha 3 os parâmetros necessários para o cálculo da posição desejada do robô (x_d,y_d), e a partir daí é determinado o erro de posição do robô (e_x,e_y), fazendo-se a diferença entre a posição desejada e a posição real do robô (x_r,y_r), que é obtida através dos *encoders* do LEGO®, que se mostrou suficientemente precisos.

$$e_r = r_d - r_r \tag{24}$$

$$e_x = x_d - x_r \tag{25}$$

$$e_y = y_d - y_r \tag{26}$$

Através do erro de posicionamento do robô, encontra-se o ângulo de orientação desejado[2], dado por:

$$\theta_d = \arctan\left(\frac{e_y}{e_x}\right) \tag{27}$$

Então, o erro do ângulo de orientação do robô é obtido, através da equação indicada abaixo:

$$e_\theta = \theta_d - \theta_r \tag{28}$$

O erro do ângulo de orientação (e_θ) é passado para o controlador que retorna a velocidade linear e angular da ação de controle, que será passada para a malha mais interna (malha 1).

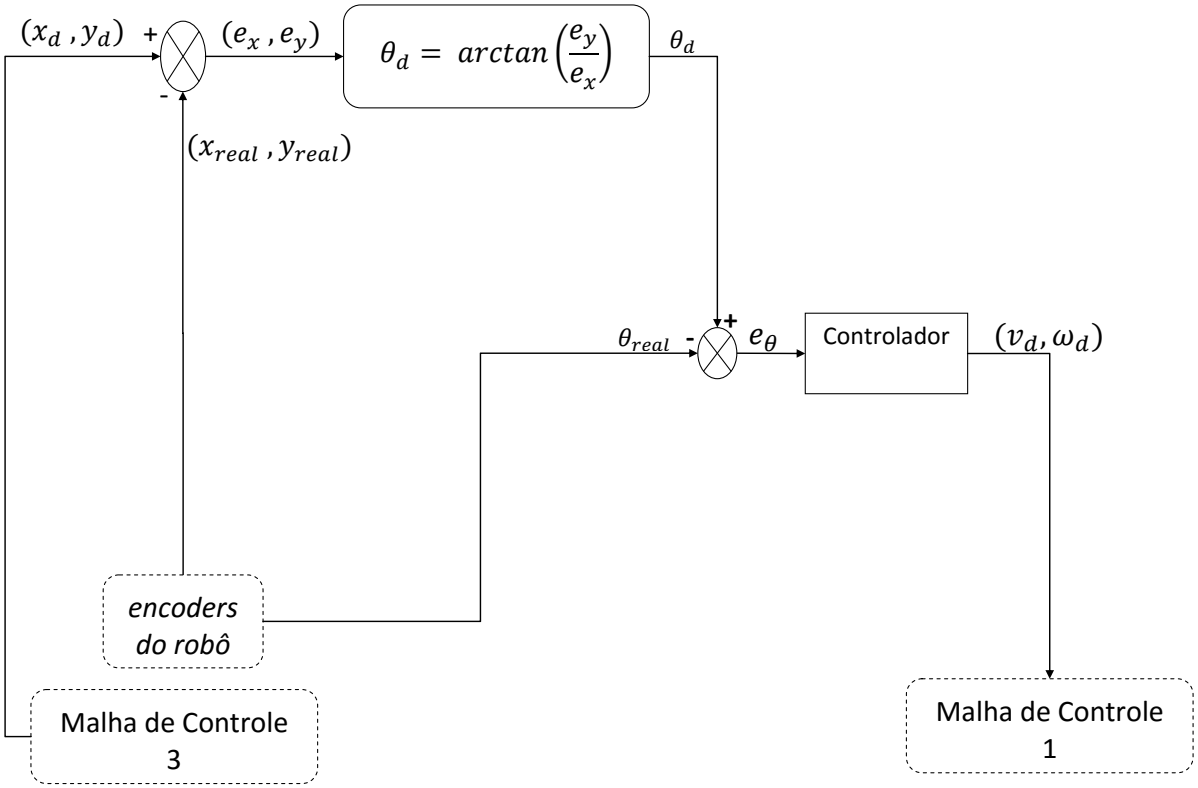


Figura 9 – Segunda malha de Controle do Sistema - Controle de Posicionamento

5.3 Malha de Controle 3: Controle de formação

A malha de controle de formação tem como função estabelecer a comunicação entre os robôs, enviando os dados necessários para que se estabeleça a formação da

[2]Para fins de implementação é altamente recomendável o uso da função *atan2* que mapeia o ângulo de orientação para o intervalo entre $-\pi$ e π, contemplando todos os quatro quadrantes do plano xy. Essa função é utilizada neste trabalho para calcular o ângulo de orientação desejado e limitar o erro do ângulo de orientação.

frota. Além disso, ela utiliza as informações recebidas para estabelecer a posição em que o robô deve assumir naquele instante. Existem duas formas de se fazer esse controle, uma delas consiste em centralizar este controle de formação no mestre e enviar para o restante da frota apenas a posição desejada já calculada; já a outra consiste na troca de informações entre os agentes do sistema sem sobrecarregar o mestre.

No caso do primeiro problema (varredura de área em paralelo), é necessário que cada robô saiba a posição no eixo das abscissas de todos os outros robôs e a partir daí calcule a posição desejada, ou seja, um determinado robô recebe a posição dos outros robôs da frota e utiliza a Equação (19) para obter a posição desejada do robô naquele instante, que é passada para a malha de controle intermediária, que por sua vez calcula o erro de posicionamento e envia para a malha de controle de velocidade interna as velocidades que devem ser assumidas pelo robô para que o mesmo possa alcançar a posição desejada e cumprir os objetivos estabelecidos. É importante ressaltar que, a depender da rede implementada, como é o caso deste trabalho em que a rede é centralizada e os robôs escravos não se comunicam entre si, essas informações devem ser passadas de um robô para o outro sempre através do mestre.

Já no caso do segundo problema (circulação de um robô), as informações que cada robô deve possuir para assumir a formação da frota são: número de robôs da frota, numeração que aquele robô representa na frota e posição do alvo. A partir daí é possível utilizar as Equação (20) para se obter a posição desejada para que aquele robô contribua positivamente para a formação da estrutura da frota. Como pode ser visto na Equação (20), a definição da posição desejada leva em consideração o tempo, o que é um problema na aplicação prática, visto que não há uma sincronia entre os relógios dos robôs. Dessa forma, para contornar este problema, será levado em consideração o relógio do mestre, sendo necessário que essa variável também seja repassada ao restante da frota.

6 Teste das Malhas de Controle

Para alcançar o objetivo final deste trabalho[1], fez-se uma modularização do problema que foi validada e integrada, módulo a módulo. Primeiramente será testada a malha 1 de controle de velocidade angular das rodas. Ou seja, ao passar um comando de velocidade angular para roda o sistema deverá estabilizar, convergindo o erro do mesmo para zero, em um determinado tempo que espera-se ser suficiente para atingir com o objetivo proposto.

O primeiro teste a ser realizado será na malha de controle mais interna com cinco controladores diferentes: P, PI, PID, Incremental e o quinto, através de uma função polinomial que converte a velocidade angular desejada na potência a ser aplicada. Para utilizar essa função polinomial é necessário recorrer a um método da linguagem *NXC* de ativação dos motores que já implementa um controlador de malha fechada. Essa função foi obtida por meio de um mapeamento da resposta do sistema às potências aplicadas. Em seguida foi realizada uma aproximação de 4º grau com os dados obtidos.

Para que a malha de controle interno cumpra seu objetivo é necessário que o erro entre a velocidade desejada e a velocidade real em cada roda tenda a zero. Ou seja, desejando-se que o robô tenha uma velocidade linear de $0.1m/s$ e uma velocidade angular de $0.6rad/s$, faz-se uma conversão desses valores utilizando da Equação (21), encontrando os valores desejados para cada uma das rodas. Feito isso, calcula-se o erro entre a velocidade desejada e a real de cada roda e passa esse erro como entrada para o controlador que deve aplicar uma potência nos motores pretendendo atingir a velocidade desejada.

Para realizar o teste com a malha interna foram aplicadas três entradas distintas e foi observado o tempo de conversão do sistema para um estado estacionário e se o sistema estava convergindo para um erro próximo de zero. Utilizando-se de um intervalo de amostragem de cerca de 25 milissegundos, aplicou-se nas primeiras 100 amostras uma velocidade linear e angular nula. Feito isso nas próximas 400 amostras aplicou-se uma velocidade linear de $0.189m/s$ e uma velocidade angular nula (velocidade angular de cada roda igual a $8.75rad/s$) e nas 500 amostras seguintes manteve-se a velocidade linear anterior e foi acrescentada uma velocidade angular de $0.628rad/s$ (sendo, $10.38rad/s$ e $7.12rad/s$, as velocidades angulares das rodas direita e esquerda, respectivamente). É importante[2] destacar que no primeiro momento o robô deve per-

[1] Todos os dados dos testes estão disponíveis em: https://github.com/MarianaAthayde/tcc/tree/master/Testes. Os códigos estão no link: https://github.com/MarianaAthayde/tcc/tree/master/Implementações/Implementações - Consolidadas.

[2] Com intuito de facilitar a visualização dos tempos de ajuste do sistema, algumas amostras de dados foram retiradas. Essas amostras não interferem no entendimento do comportamento do sistema, visto

manecer parado, adquirir uma velocidade linear e caminhar em linha reta com uma velocidade de $0.189m/s$, para então iniciar uma trajetória circular de raio de $30cm$ e período de $10s$.

6.1 Malha 1: Controlador Incremental

Primeiro a malha interna foi testada com um controlador incremental que consiste basicamente em aumentar ou diminuir a potência das rodas, de acordo com o erro. Se o erro for positivo, incrementa-se positivamente a potência, se for negativo, decrementa-se negativamente a potência. Essa potência será passada como referência para as funções de comando dos motores, que por sua vez, só aceitam valores na faixa entre -100 e 100.Foram testados os seguintes valores como incremento: 1, 3 e 6 e como podemos observar nos gráficos 10a,10b e 10c e na Tabela 1, o controlador de incremento igual a um é suave e possui oscilações aceitáveis mas, o tempo para estabilização do sistema é maior que o dobro do tempo obtido com o incremento igual a seis. Entretanto, o controlador com incremento igual a seis é muito agressivo, oscilando com amplitude dobrada se comparado ao controlador de incremento igual a um.

Fazendo-se o incremento igual a 3 tem-se um custo-benefício interessante pois, o sistema continua razoavelmente suave, com oscilações bem próximas ao do controlador de incremento igual a um e com um tempo de resposta duas vezes menor. Já, comparado ao de incremento igual a seis, ele apresenta um tempo de resposta bem próximo e com a metade da oscilação, com um controle bem mais suave. Portanto, o incremento igual a 3 parece ser o mais adequado para a malha interna se comparado ao controlador com outros valores de incremento.

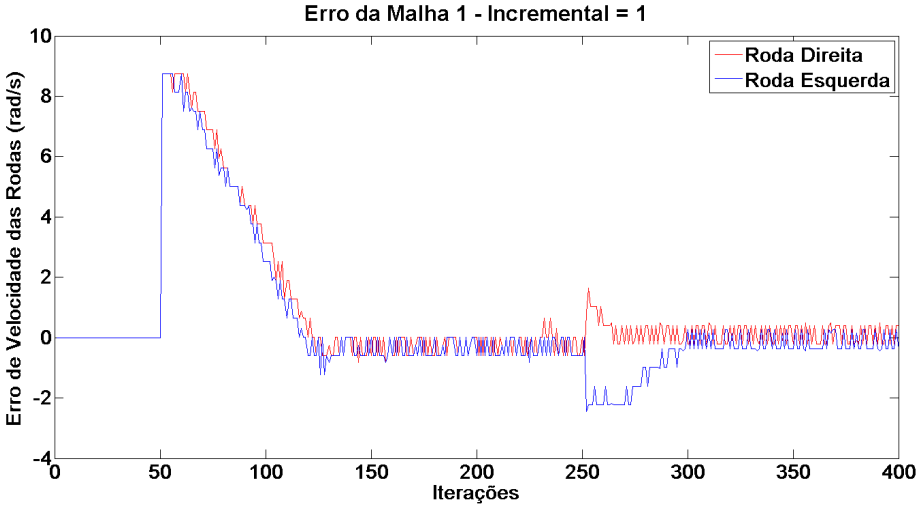
Características/ Incrementos	**Oscilação (rad/s)**	**Tempo de Assentamento (s)**	**Controle**
1	$\pm 0.6 rad/s$	$\approx 2s$	Suave
3	$\pm 0.6 rad/s$	$\approx 1s$	Suave
6	$\pm 1.2 rad/s$	$\approx 0.75s$	Agressivo

Tabela 1 – Malha1: Controlador Incremental

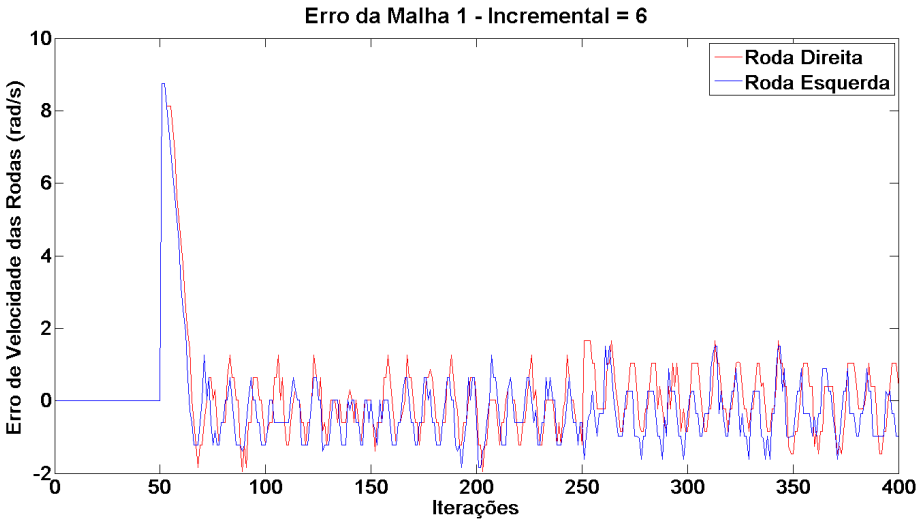
6.2 Malha 1: Controladores P, PI e PID

Foram realizados testes com controladores do tipo P, PI e PID, os ganhos foram ajustados de forma empírica e a Tabela 2, assim como as figuras 11 e 12 mostram alguns dos resultados obtidos. Pode-se observar que o melhor ganho obtido a partir dos testes realizados foi o do controlador PID com ganho proporcional igual a 10, ganho

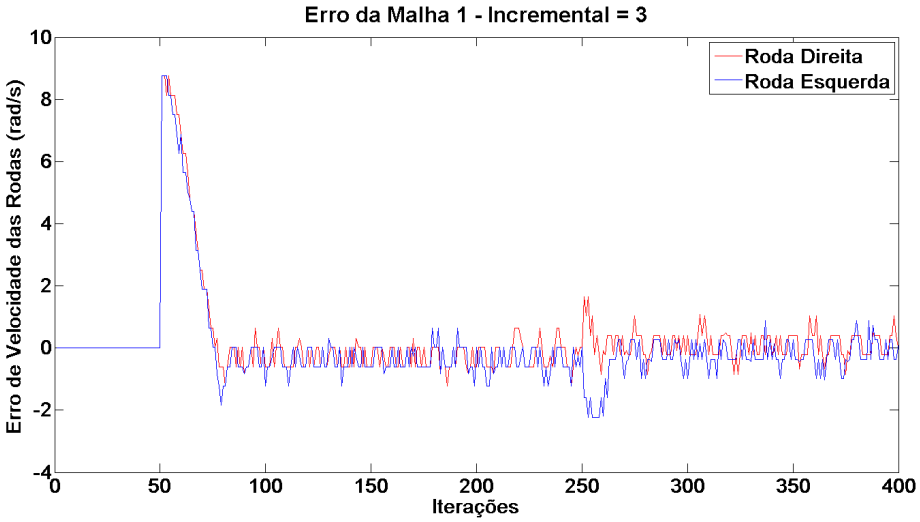
que foram retiradas faixas em que o sistema já estava estável.



(a) Erro da malha interna com incremento = 1



(b) Erro da malha interna com incremento = 6



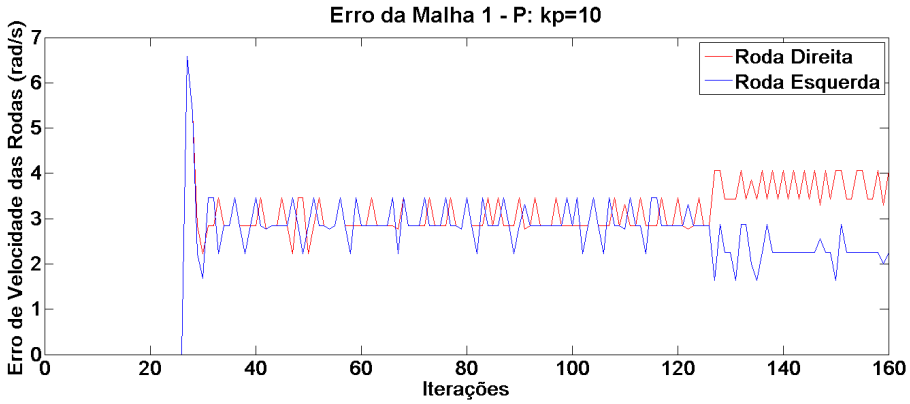
(c) Erro da malha interna com incremento = 3

Figura 10 – Experimentos com Controlador Incremental

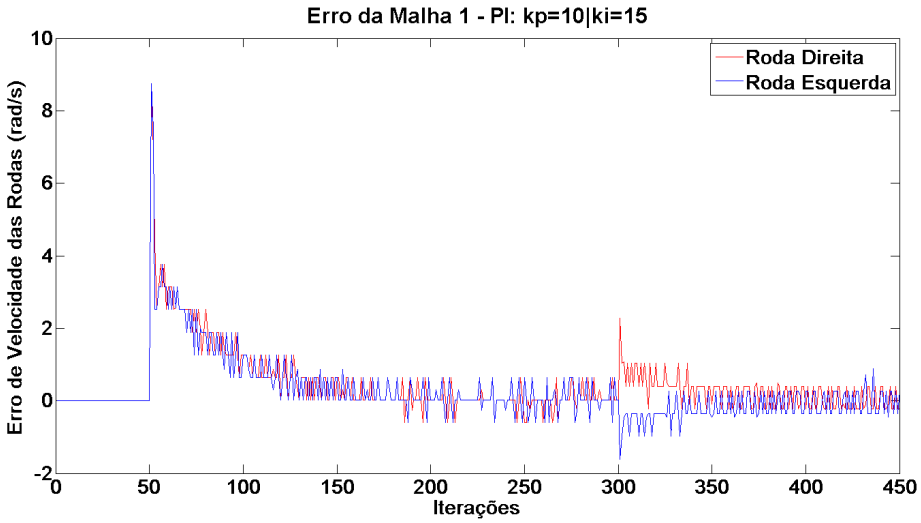
integrativo igual a 15 e ganho derivativo igual a $0{,}2$, visto que ele apresentou o menor tempo de resposta com uma oscilação razoável e inferior aos outros ganhos.

Características/ Ganhos do controlador	**Valor de Estabilização (rad/s)**	**Oscilação (rad/s)**	**Tempo de Assentamento (s)**	**Controle**
P: 10	$3\ rad/s$	$1.3 rad/s$	$0.5s$	Suave
P:10 - I:15	$0\ rad/s$	$0.65 rad/s$	$3.25s$	Suave
P:10 - I:17	0 rad/s	$0.65 rad/s$	$2.9s$	Suave
P:10 - I:15 - D:0.2	$0\ rad/s$	$0.6 rad/s$	$2.9s$	Suave
P:10 - I:15 - D:0.6	$0\ rad/s$	$1.3 rad/s$	$2s$	Agressivo

Tabela 2 – Experimentos com Controlador P/PI/PID

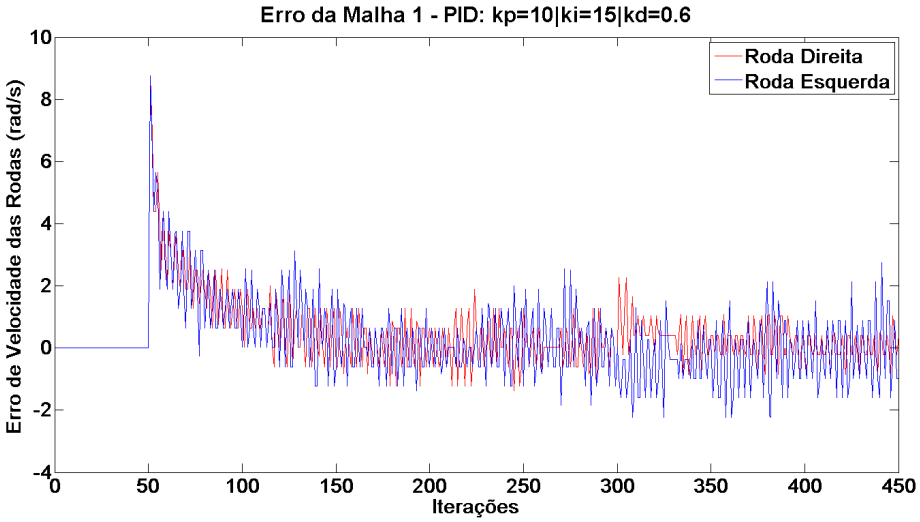


(a) Controlador Proporcional (kp = 10)

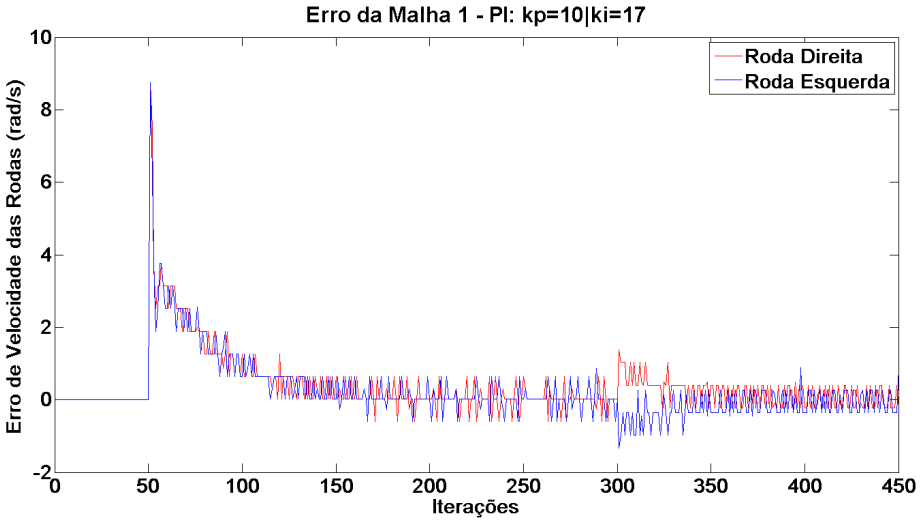


(b) Controlador Proporcional - Integral (kp = 10 e ki = 15)

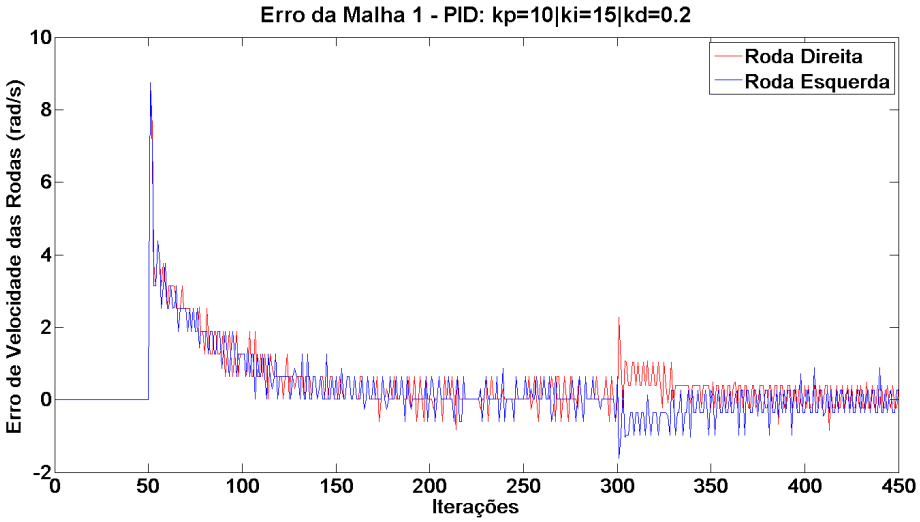
Figura 11 – Experimentos com Controlador P e PI



(a) Controlador Proporcional - Integral - Derivativo (kp = 10, ki = 15 e kd = 0.6)



(b) Controlador Proporcional - Integral - Derivativo (kp = 10 e ki = 17)



(c) Controlador Proporcional - Integral - Derivativo (kp = 10, ki = 15 e kd = 0.2)

Figura 12 – Experimentos com Controlador PID

6.3 Malha 1: Função de Ajuste Polinomial e Controlador Embutido

Como já dito anteriormente, a plataforma já implementa uma malha de controle da velocidade dos motores, um controlador *PID* já ajustado pelo fabricante. Para utilizá-la basta usar um método específico da linguagem *NXC*. Entretanto, a referência desta malha de controle é uma variável que assume valores entre -100 e 100. Surge daí, portanto, a necessidade de se fazer um mapeamento entre esta variável e a variável de velocidade angular assumida pelo motor em radianos por segundo. Para realizar este mapeamento estático, utiliza-se o método dos mínimos quadrados para se ajustar um polinômio de quarto grau aos dados medidos, obtendo-se assim a seguinte expressão:

$$f(\omega_{desejada}) = -0.00091\omega^4_{desejada} + 0.0223\omega^3_{desejada} - 0.01537\omega^2_{desejada} + 6.1864\omega_{desejada} - 0.0546 \quad (29)$$

Como pode ser visto na Figura 13, a função mapeia muito bem as saídas do motor, o sistema converge para um valor de erro nulo com um tempo de aproximadamente 0,75 segundos, com uma oscilação de $\pm 0.6 rad/s$, chegando a ser quase quatro vezes mais rápida que os resultados obtidos com o controlador PID ajustado na Seção 6.2.

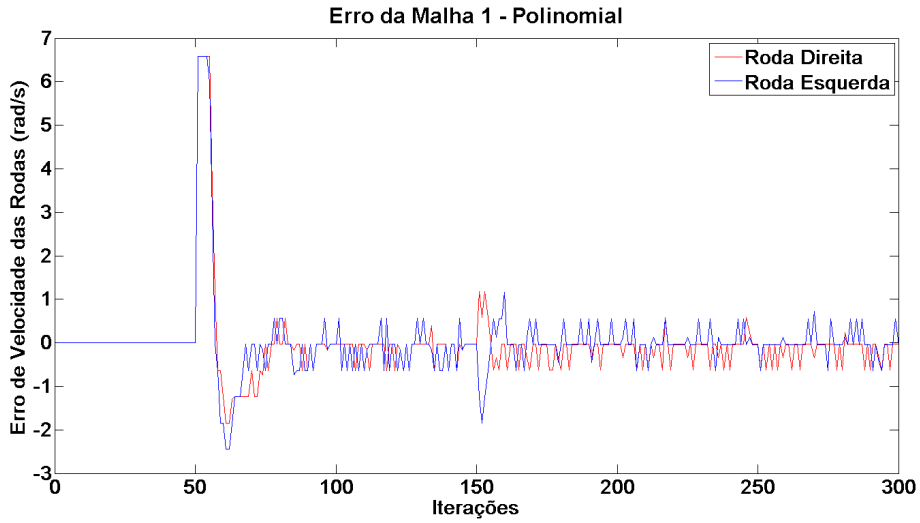


Figura 13 – Experimentos com a Função Polinomial e o Controlador PID Embutido

6.4 Malha 2: Controle de Posicionamento

Nesta parte do trabalho, serão apresentados os testes realizados com a segunda malha de controle, que é responsável pelo posicionamento do robô. Ou seja, a malha recebe como referência a posição onde deseja-se que o robô esteja, a partir daí calcula-se a velocidade angular necessária para que o robô chegue ao ponto desejado, e esta velocidade, por sua vez, é passada para a malha de controle mais interna.

Os testes foram realizados da seguinte maneira: O robô foi iniciado no ponto $(0.8,0.0)$, com orientação (θ) inicial igual a π, e foi definido um alvo na origem do sistema que esse robô teria que circundar com um raio de 30 centímetros e um período de 24 segundos. Após mil e duzentas amostras, o raio e o período do sistema mudam sendo 40 centímetros de raio em um período de 30 segundos, tornando o teste muito parecido com o segundo problema em si.

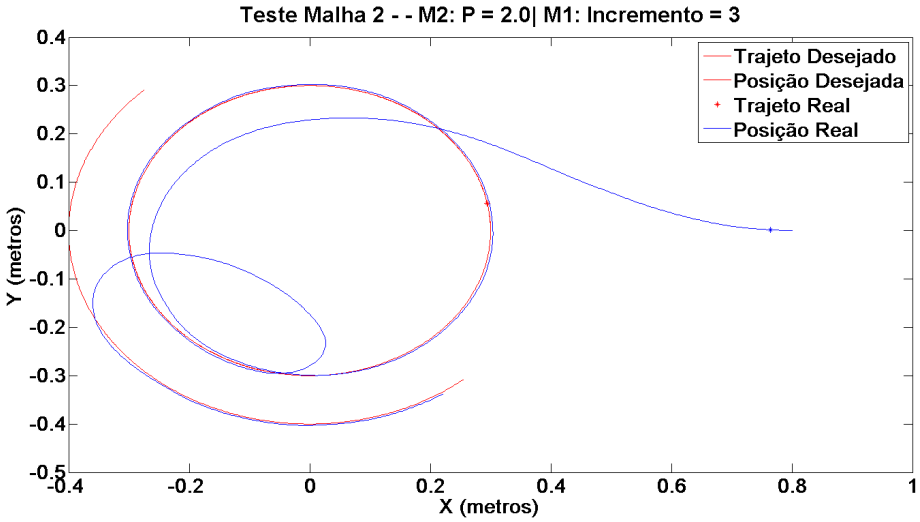
Na Tabela 3 são apresentados alguns dos resultados obtidos com os experimentos feitos com a segunda malha, utilizando-se o controlador proporcional de ganho unitário e com ganho proporcional igual a 2 na malha intermediária, a malha responsável pelo controle de posicionamento. Como já sabido, quando tem-se um controle em cascata é necessário que a malha interna responda pelo menos 5 vezes mais rápido do que a malha externa, e quanto mais rápido melhor.

Tendo isto em mente, do ponto de vista de tempo de resposta, todos os controladores da malha mais interna testados atendem ao pré-requisito, no entanto, ao observar o sistema na vida real é possível notar que o controlador PID é muito agressivo e por vezes produz um tranco no motor, o que não é desejado, visto que qualquer comando brusco pode ocasionar em derrapagem das rodas, o que poderia levar o robô a se perder por acumulo de erro odométrico e mapear uma trajetória diferente da trajetória apresentada no mundo real.

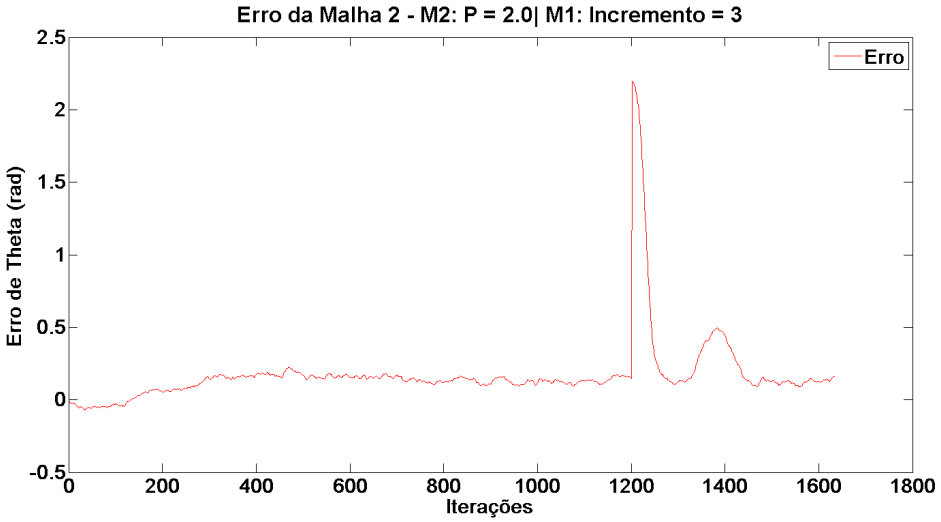
Os dois que apresentaram melhores resultados foram o controlador da malha externa proporcional com ganho de aproximadamente igual a 2 combinados com os controladores Incremental e Polinomial, o que se confirma pelas figuras 14, 15, 16 e pela Tabela 3. Como mostrado na Figura 17, os testes com controlador PI na malha intermediária não apresentaram um bom resultado, apresentando oscilações indesejadas que demoraram para se estabilizar ao mudar a ação de controle drasticamente, além de apresentar um controle mais agressivo.

Características/ Controlador Interno e Externo	**Estabiliza em (rad)**	**Termpo de Assentamento (s)**	**Controle**
Incremental (Incremento 3)/ Proporcional (kp = 1.0)	$\pm 0.3rad$	$\pm 14s$	Suave
PID (kp = 10;ki = 15;kd = 0.2)/ Proporcional (kp = 1.0)	$\pm 0.3rad$	$\pm 17.5s$	Agressivo
Polinomial/ Proporcional (kp = 1.0)	$\pm 0.3rad$	$\pm 17.5s$	Suave
Incremental (Incremento 3)/ Proporcional (kp = 2.0)	$\pm 0.18rad$	$\pm 11s$	Pouco Agressivo
Polinomial/ Proporcional (kp = 2.0)	$\pm 0.17rad$	$\pm 14s$	Suave

Tabela 3 – Comparativo dos Controladores da Malha Intermediária (malha 2)

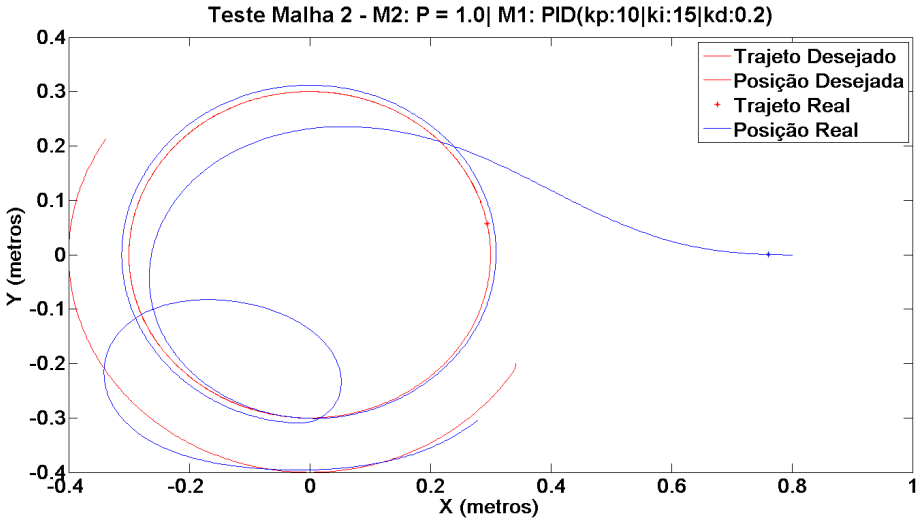


(a) Malha Intermediária: P=2.0 - Malha Interna: Incremental k = 3

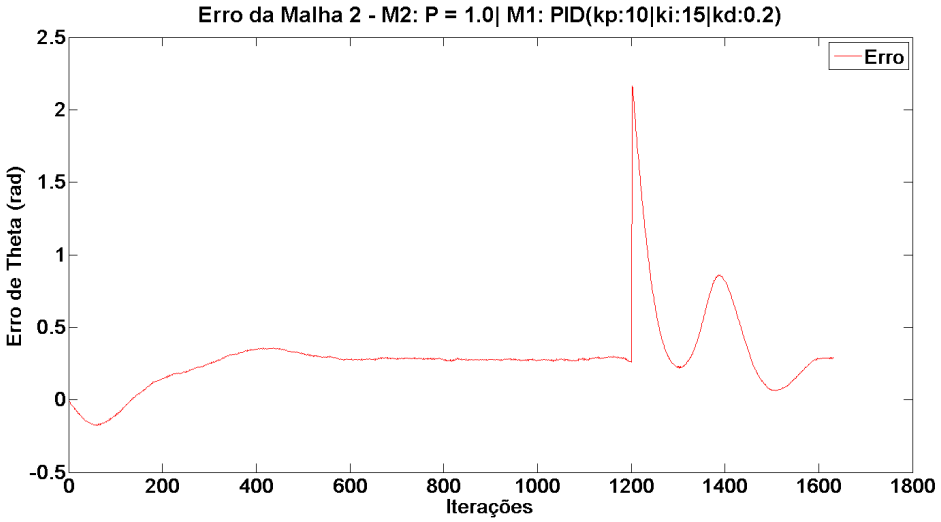


(b) Erro da Malha Intermediária: P=2.0 - Malha Interna: Incremental k = 3

Figura 14 – Experimentos com a Malha Intermediária com Controlador Proporcional(kp=2.0) e Malha Interna: Controlador Incremental (k = 3)

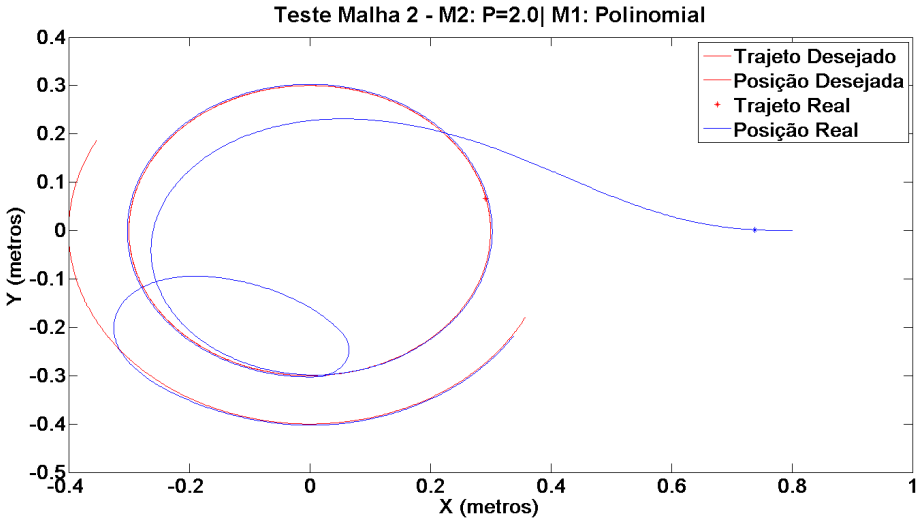


(a) Malha Intermediária: P=1.0 - Malha Interna: PID (kp = 10;ki = 15; kd = 0.2)

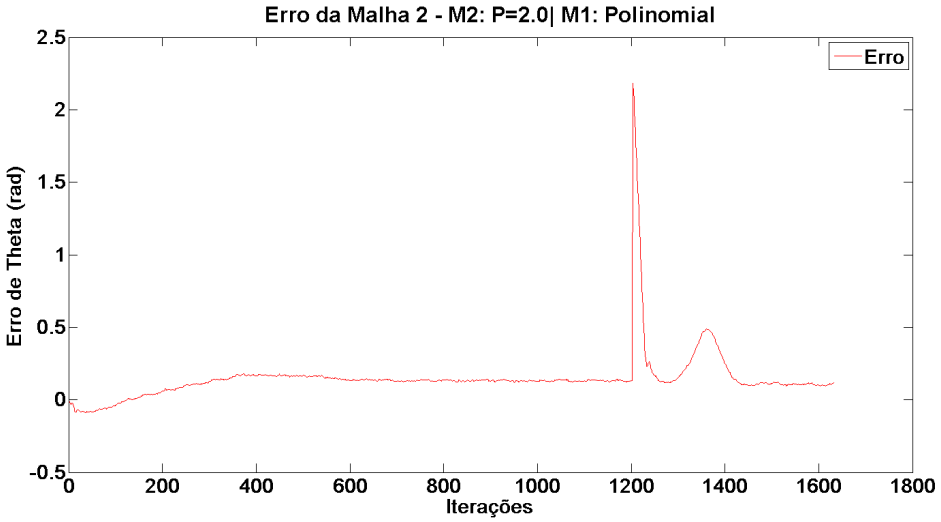


(b) Erro da Malha Intermediária: P=1.0 - Malha Interna: PID (kp = 10;ki = 15; kd = 0.2)

Figura 15 – Experimentos com a Malha Intermediária com Controlador Proporcional e Malha Interna: Controlador PID

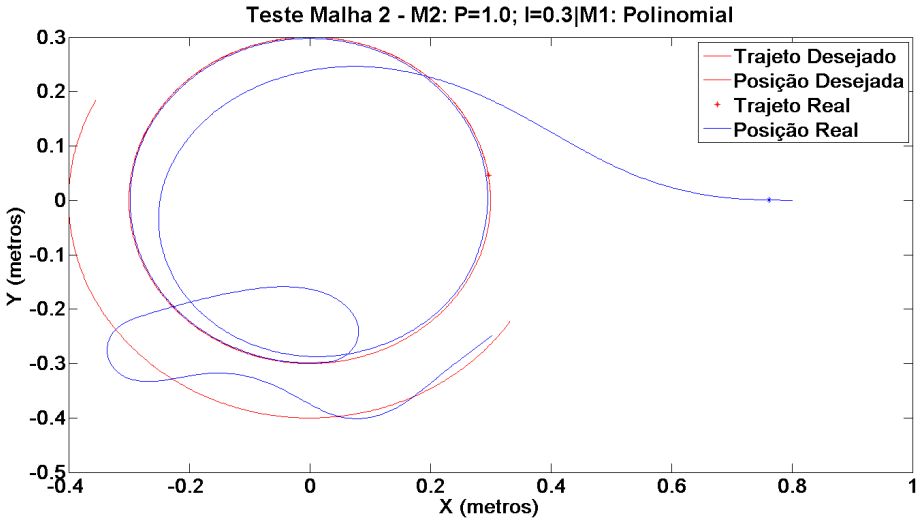


(a) Malha Intermediária: P=2.0 - Malha Interna: Polinomial

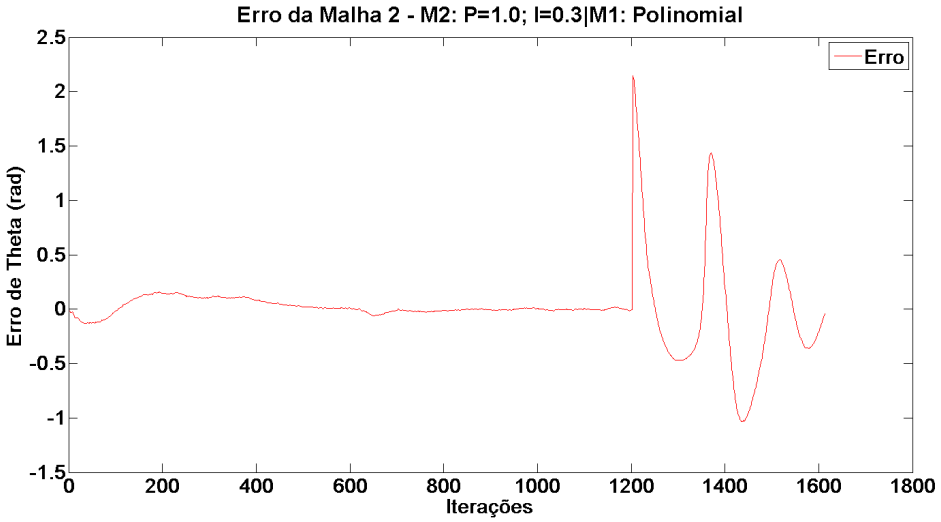


(b) Erro da Malha Intermediária: P=2.0 - Malha Interna: Polinomial

Figura 16 – Experimentos com a Malha Intermediária com Controlador Proporcional e Malha Interna: Controlador Polinomial



(a) Malha Intermediária: P=1.0; I=0.3 - Malha Interna: Polinomial



(b) Erro da Malha Intermediária: P=1.0; I=0.3 - Malha Interna: Polinomial

Figura 17 – Experimentos com a Malha Intermediária com Controlador Proporcional e Malha Interna: Controlador Polinomial

7 Simulações

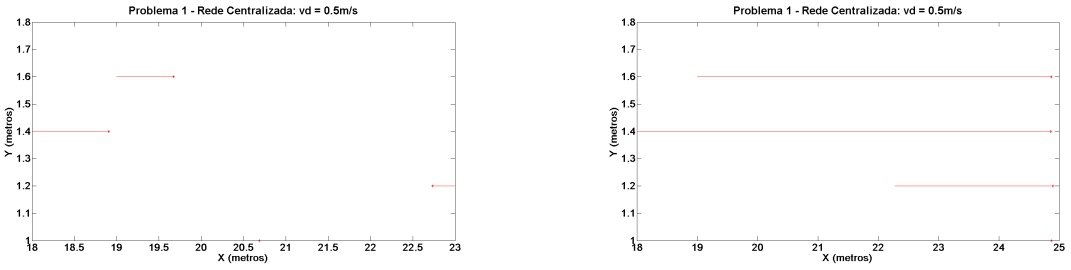
As simulações[1] realizadas neste trabalho, consideram o robô como um elemento pontual, conforme descrito nas equações 5, 6 e 7 e foram realizadas com intuito de se observar a viabilidade de implementação da estratégia de controle.

Essas implementações desconsideram não só o robô como um elemento não-pontual, mas também, desconsideram problemas como: saturação do motor, derrapagem das rodas, falhas de comunicação e imprecisão de leitura dos *encoders*.

7.1 Simulações do Primeiro Problema

Primeiramente, foi realizado a simulação da primeira estratégia de abordagem do primeiro problema, em que tem-se uma frota de N robôs separados entre si no eixo y e com uma posição randômica no eixo x, todos dispostos na mesma direção e sentido. A primeira estratégia elimina os problemas de colisão, visto que os robôs começam separados, no mesmo sentido e como são impossibilitados a imprimir uma velocidade angular pela estratégia adotada, seguem em linha reta com velocidade linear negativa ou positiva. Além disso, ela leva em consideração a rede de comunicação, utilizando-se a Equação (14).

Serão feitas simulações com duas redes, a primeira será uma rede centralizada com 4 robôs, tal qual é possível implementar com a plataforma *Lego Mindstorms*®. A segunda, será uma rede descentralizada com cinco robôs, tal como a rede ilustrada na Figura 4c. Como mostrado nas Figuras 18, 19 e 20, essa é uma estratégia viável quando considerando o robô como um elemento pontual.

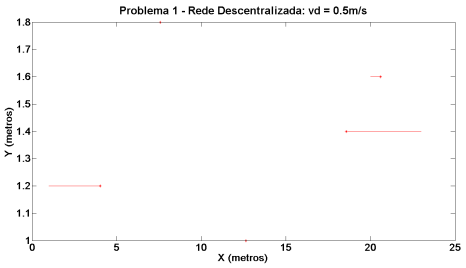


(a) Sistema com Rede Centralizada - Antes do alinhamento dos robôs

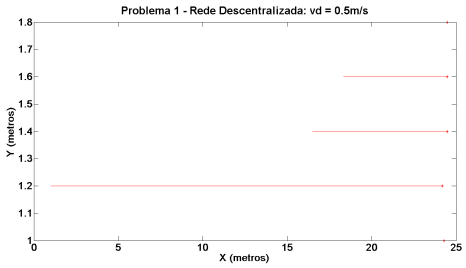
(b) Sistema com Rede Centralizada - Depois do alinhamento dos robôs

Figura 18 – Problema 1 - Primeira Abordagem com Rede Centralizada

[1] Os códigos-fonte das simulações realizadas podem ser acessador em: https://github.com/MarianaAthayde/tcc/tree/master/Implementações/Implementações - Consolidadas/Simulações

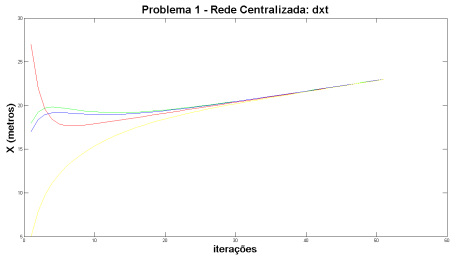


(a) Sistema com Rede Descentralizada - Antes do alinhamento dos robôs

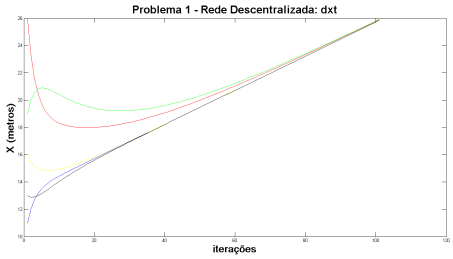


(b) Sistema com Rede Descentralizada - Depois do alinhamento dos robôs

Figura 19 – Problema 1 - Primeira Abordagem com Rede Descentralizada



(a) Sistema com Rede Centralizada - $d \times t$



(b) Sistema com Rede Descentralizada - $d \times t$

Figura 20 – Problema 1 - Gráfico de distância $\times$ tempo

Já a segunda estratégia consiste em encontrar uma reta paralela ao eixo y onde, sua localização no eixo x é a média da posição dos robôs mais distantes entre sí do sistema. Ela foi implementada de modo a permitir que o robô adquirira uma velocidade angular. Sendo assim ele pode se deslocar em qualquer direção. Essa abordagem faz uso da malha de controle intermediário modelada no Capítulo 5 e apresenta problemas de colisão, já que os robôs podem se deslocar em qualquer direção.

Não foi implementado neste trabalho um tratamento de colisão, foram adotadas algumas medidas para diminuir as colisões no sistema. A primeira medida foi estabelecer uma prioridade na rede que vai do mestre ao último escravo, aquele que possui menor prioridade para ao estar em menos de 20 centímetros de distância de outro robô. A segunda medida foi fazer com que os robôs convirjam um a um para o ponto desejado. Notou-se com essas medidas uma redução[2] no número de colisões ocorridas no sistema, como mostrado no Quadro 1.

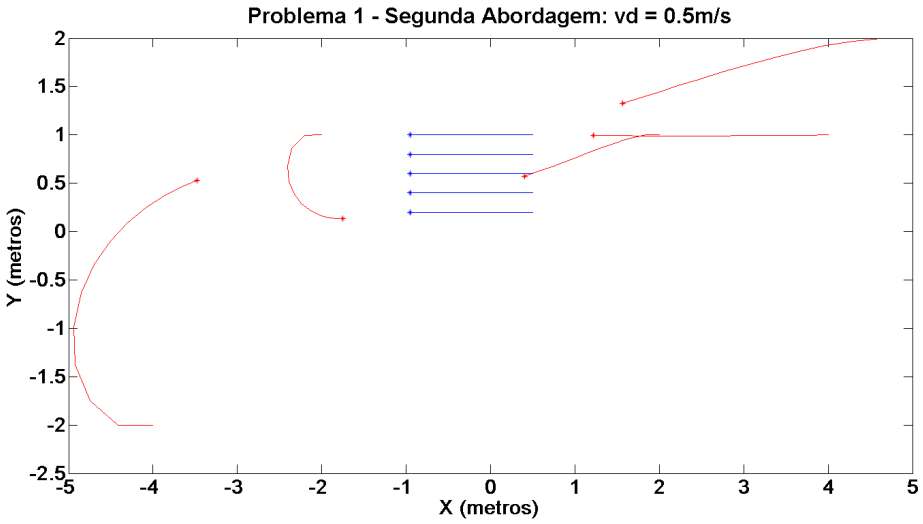
Como pode ser visto na Figura 21, essa abordagem também é viável entretanto, apesar de apresentar a vantagem de conceder mais autonomia aos robôs, deixando-os movimentar em qualquer direção, ela acarreta ao sistema problemas de colisão que

[2]Foram realizados 1000 experimentos com 300 amostras cada um, primeiro sem nenhuma medida para evitamento de colisão, depois apenas parando o robô com menor prioridade e por fim, uma combinação entre a segunda técnica e a técnica de convergir os robôs um a um para o ponto desejado. Percebeu-se com isso uma redução de 80% com aplicação da primeira técnica e de 87% com a aplicação de ambas as técnicas.

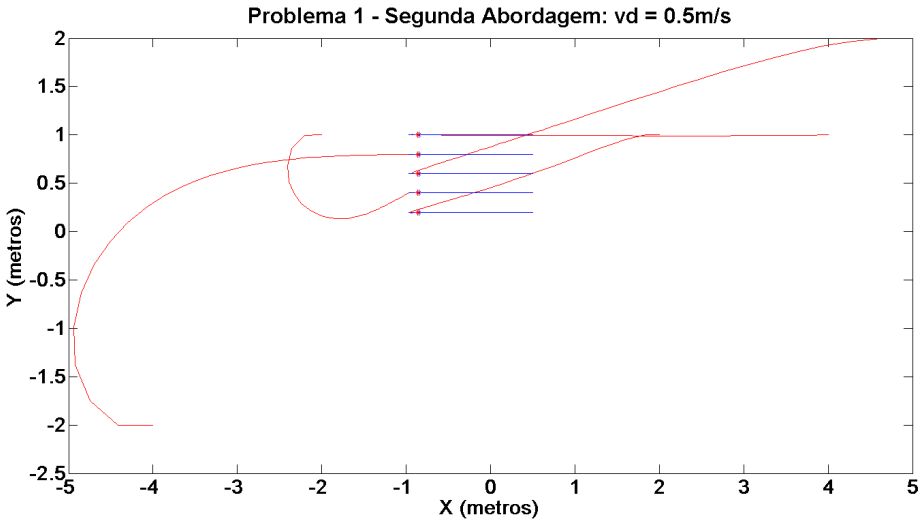
Medidas	**Redução de colisão**	**Vantagens**	**Desvantagens**
Sem técnica	0%	Não aumenta o tempo de convergência	Não evita colisões
Primeira técnica	80%	Reduz as colisões sem aumentar tanto o tempo de convergência	Não garante que as colisões nunca ocorram
Ambas técnicas	87%	Reduz as colisões	Não garante que as colisões nunca ocorram e não garante a convergência do sistema

Quadro 1 – Comparativo entre as medidas adotas para redução de colisão.

quando não contornados podem prejudicar o sistema.



(a) Segunda Abordagem - Antes do alinhamento dos robôs



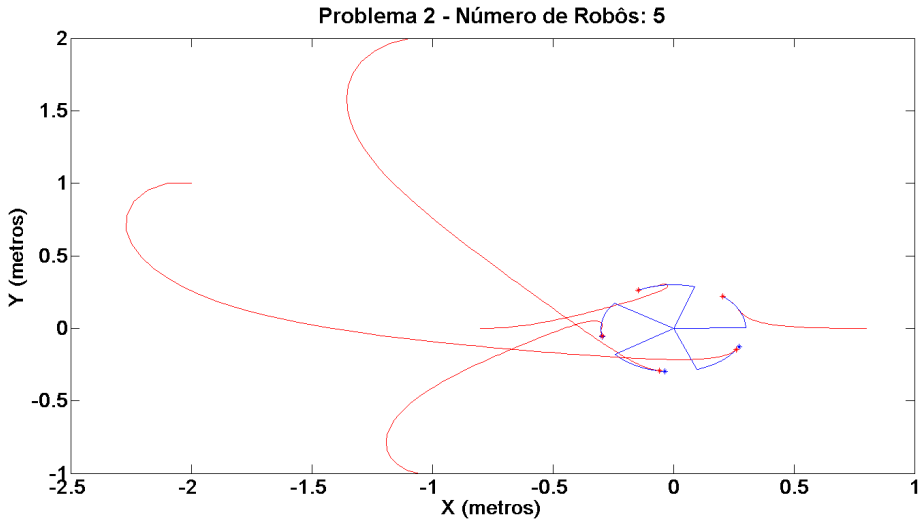
(b) Segunda Abordagem - Depois do alinhamento dos robôs

Figura 21 – Problema 1 - Segunda Abordagem

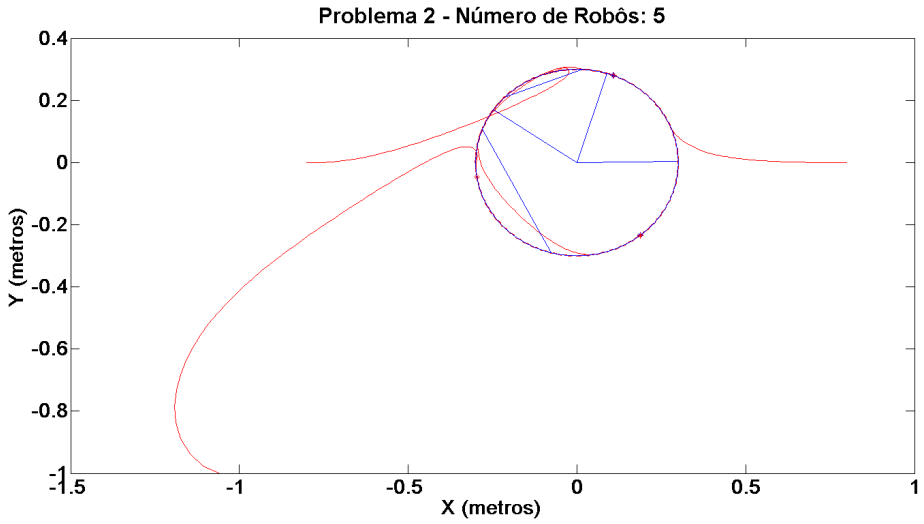
7.2 Simulações do Segundo Problema

Como já dito anteriormente neste trabalho, o segundo problema consiste em guiar uma frota de robôs a um determinado alvo e circulá-lo a uma distância R e um período T. Dito isto, fez-se algumas simulações para validar o modelo matemático e compará-lo ao modelo real. Esta simulação desconsidera um problema enfrentado no mundo real que é a falta de sincronismo entre os relógios dos motores, visto que o tempo corrido em simulação é igual para todos os robôs da frota.

A primeira simulação realizada desconsidera problemas de colisão e inicia os robôs com posições randômicas. Após mil intervalos de amostragem retiram-se dois robôs e verifica-se que o sistema se reorganiza, mantendo os robôs a mesma distância angular entre si. Já nas simulações seguintes foram tomadas as medidas citadas na Seção 7.1 para evitar que ocorra colisão entre os robôs. Como pode ser visto nas figuras 22, 23, as estratégias modeladas são viáveis dentro de um ambiente de simulação computacional.



(a) Segundo Problema - Cinco Robôs Alinhados



(b) Segundo Problema - Falha de 2 Robôs e Reajuste da Formação

Figura 22 – Segundo Problema sem Medidas para Evitamento de Colisão

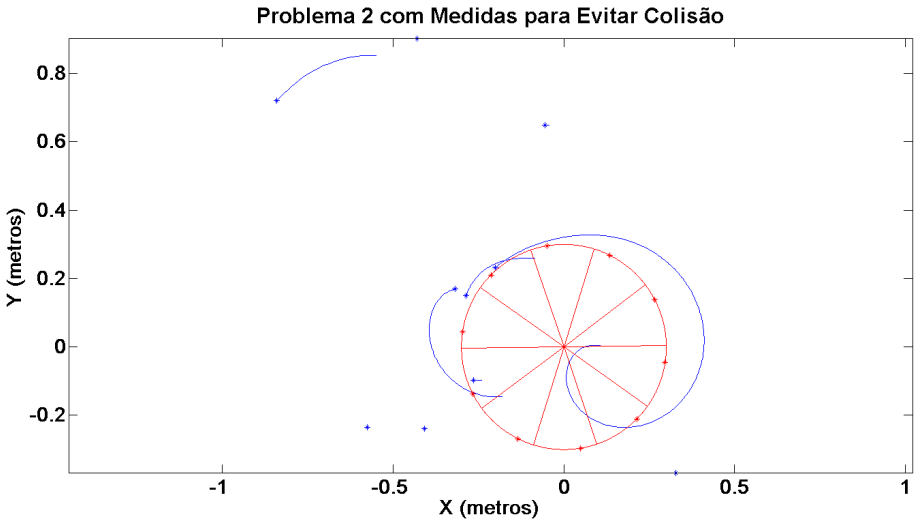


Figura 23 – Problema 2: Robôs convergindo um de cada vez, de acordo com a ordem de prioridade

8 Resultados

Após a realização de todas as simulações e testes, foram aplicadas as estratégias para resolução de ambos os problemas em uma frota de dois robôs reais. Para a resolução do primeiro problema[1], utilizou-se somente o controlador polinomial na malha interna e foi adotado a Equação (14) para estabelecer a velocidade desejada a ser aplicada nos motores. No mais, as informações trocadas entre as malhas externas foi apenas a posição dos robôs.

Como pode-se observar na Figura 24 e 25, a estratégia utilizada é suficientemente adequada para resolução do problema, visto que na Figura 25, é mostrado o erro do sistema convergindo para zero, ou seja, a distância entre os robôs reduzindo conforme eles se aproximam. Já na Figura 24a, mostra a trajetória feita pelos robôs que se alinham verticalmente e seguem paralelamente da direita para a esquerda. Além disso, observando a Figura 26, é possível notar que devido a imprecisão dos controladores, tem-se um pequeno deslocamento lateral em ambos os robôs que é de aproximadamente 2 centímetros a cada metro deslocado, o que é aceitável.

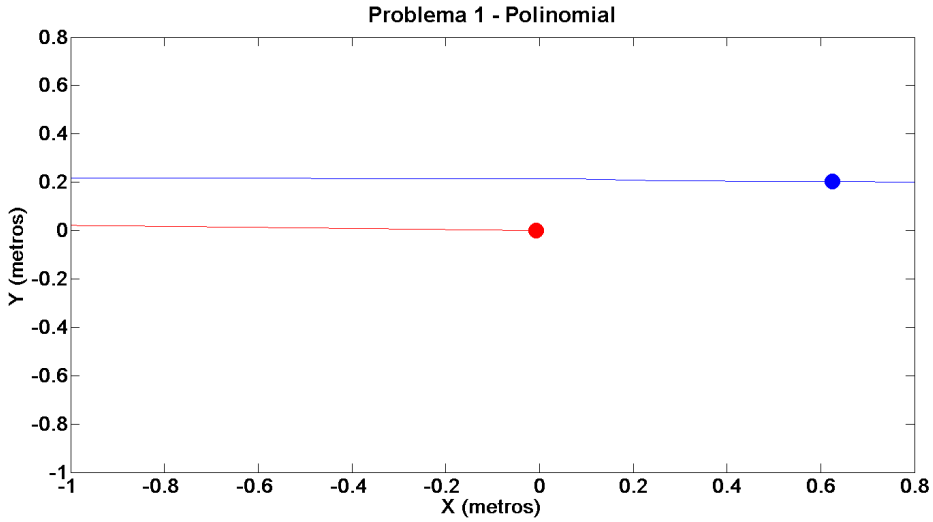
O segundo problema[2] consiste em fazer uma frota de N robôs localizar e circundar um alvo a uma distância R e em um determinado período T, que deve variar de acordo com o tamanho da frota e os robôs devem ter a mesma distância entre eles. Após realizados todos os testes e simulações, aplicou-se a estratégia em uma frota de dois robôs, colocando-os a mesma distância do alvo e em sentidos opostos, de maneira que eles não se colidissem quando o sistema ainda não estivesse estável.

Como pode ser visto na Equação (20), o cálculo da posição desejada para o segundo problema depende do tempo, devido a falta de sincronismo entre os relógios dos robôs, a posição desejada calculada pelos robôs pode apresentar erros na distribuição circular balanceada. Para contornar este problema, a solução encontrada foi passar para o outro robô o tempo do mestre. Desta forma, elimina-se o problema da falta de sincronismo entre os relógios.

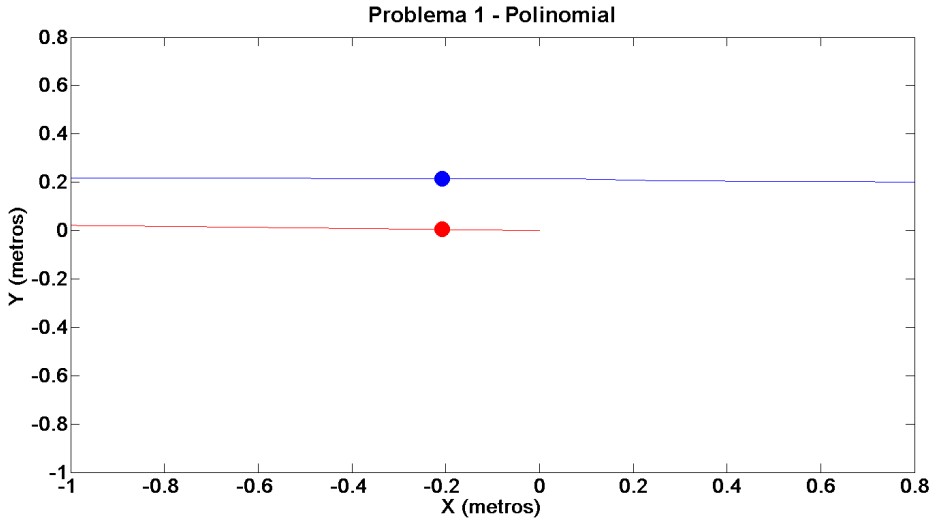
Como pode ser visto na Figura 27, a estratégia se mostrou plausível, visto que ambos os robôs convergiram para o alvo, contornando-o a uma distância de aproximadamente 30 centímetros, se mantiveram equidistantes e após a retirada de um dos robôs, dobrou-se o ω do robô, cobrindo melhor a fronteira. Para a realização deste experimento

[1] Os códigos-fonte para o primeiro problema podem ser acessados em: https://github.com/MarianaAthayde/tcc/tree/master/Implementações/Implementações - Consolidadas/Problema1

[2] Os códigos-fonte da implementação do segundo problema podem ser acessados em: https://github.com/MarianaAthayde/tcc/tree/master/Implementações/Implementações - Consolidadas/Problema2



(a) Sistema com Dois Robôs - Antes do alinhamento dos robôs



(b) Sistema com Dois Robôs - Depois do alinhamento dos robôs

Figura 24 – Problema 1 com 2 Robôs

foi utilizado o controlador da malha interna polinomial e o controlador proporcional da malha intermediária de ganho 2 que obteve um resultado melhor, como demonstrado no Capítulo 6.

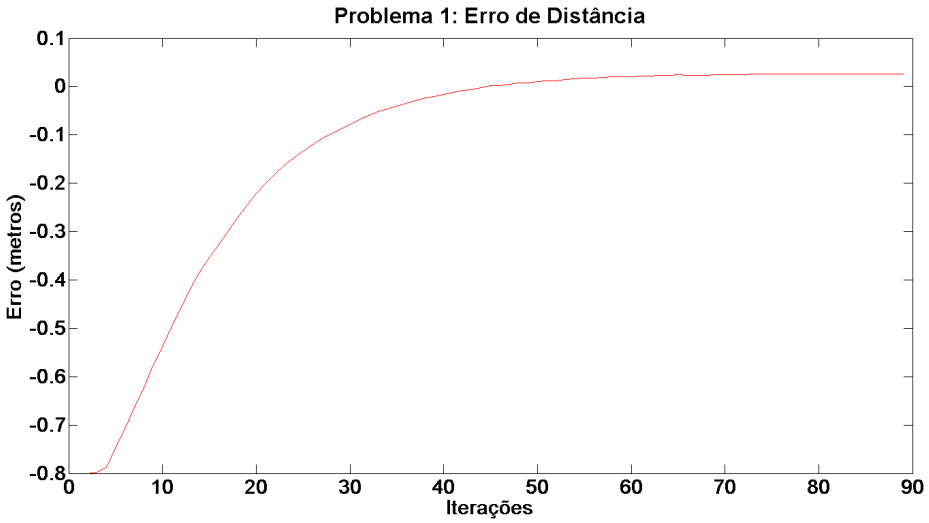


Figura 25 – Problema 1: Erro de Posição (x1 - x2)

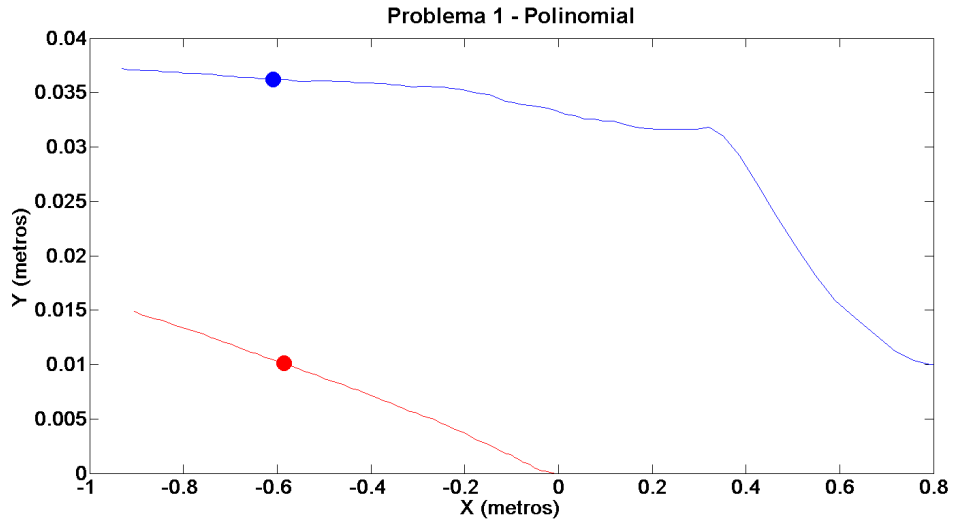
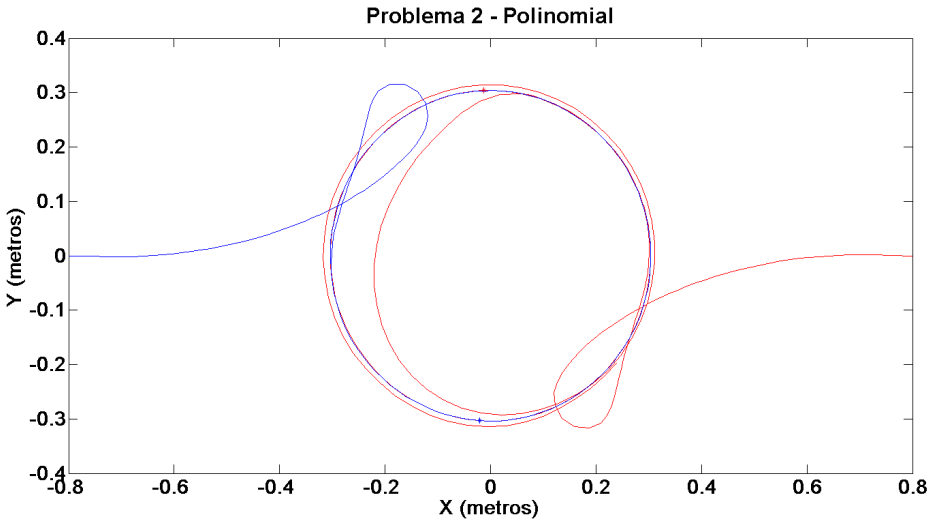
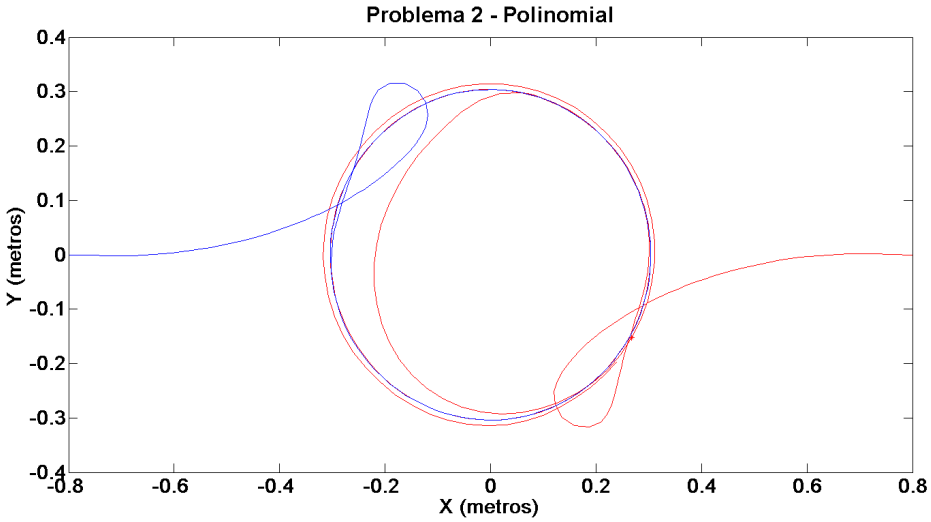


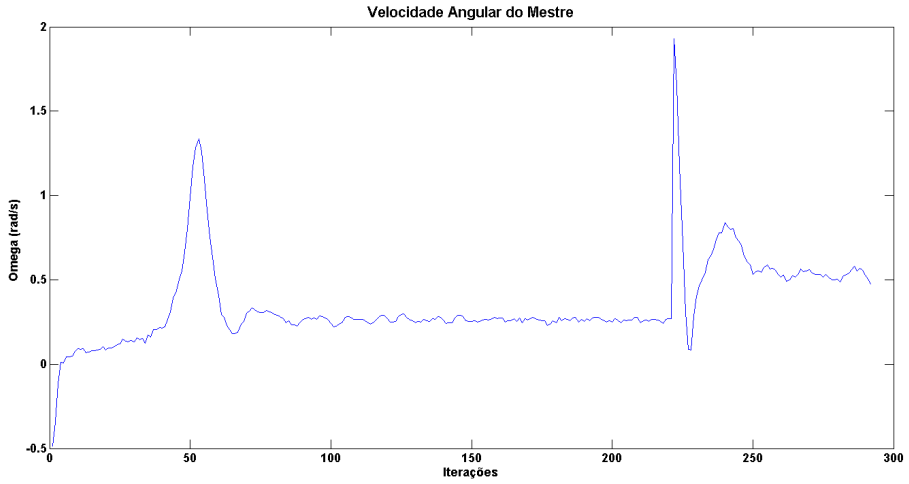
Figura 26 – Problema 1 com 2 Robôs: Deslocamento Lateral



(a) Sistema com Dois Robôs



(b) Sistema com um Robô



(c) Velocidade Angular do Mestre

Figura 27 – Segundo Problema com Falha de um dos Robôs

9 Conclusão

Neste trabalho fez-se o estudo de duas estratégias de controle de formação, tendo como intuito a implementação dessas estratégias não só por meio de simulações como também, com o intuito de desenvolver e aplicar essas estratégias em uma plataforma de custo relativamente baixo: o *Lego Mindstorms*®,

O primeiro problema visava uma espécie de varredura em paralelo por uma frota de robôs que deveriam se alinhar e seguir se deslocando em conjunto. Já o segundo problema tinha como o objetivo controlar um time de robôs para que a mesma localizasse determinado alvo e a partir daí o circundassem, mantendo uma formação circular e equidistante entre si. Visando uma melhor cobertura da fronteira, o time deveria se reajustar caso um ou mais robôs fossem retirados da formação, de modo que os robôs remanescentes se mantenham equidistantes porém, circulando o alvo em um período menor, cobrindo melhor a fronteira.

Uma parte deste trabalho foi adaptar as estratégias e soluções encontradas para se adequar a plataforma que apresenta algumas limitações, como por exemplo a da estrutura da rede de comunicação. Uma das limitações deste trabalho, é por exemplo o fato de que a localização é feita única e exclusivamente, pelos *encoders* já acoplados aos motores da plataforma. Que se mostraram, após todos os testes e experimentos realizados, suficientemente precisos para serem utilizados como ferramenta de medição e alimentação do sistema. Entretanto, o sistema fica vulnerável a deslizamentos ou a qualquer interferência externa que venha a ocorrer, podendo ocasionar uma completa desorientação do sistema sem a possibilidade de se localizar novamente.

Embora os *encoders* sejam suficientemente precisos, é necessário cautela quando se for realizar mudanças bruscas de velocidade, pois isso pode ocasionar derrapagens, fazendo com que o sistema perca sua localização. Como não há nenhum sensor externo, uma câmera ou algo para que o robô possa se localizar novamente, o erro ocorrido vai sendo acumulado até o final da execução do programa, comprometendo o cumprimento da missão do sistema multiagente.

Outra parte importante para o estudo das estratégias de controle de formação foram as simulações realizadas que, apesar de não serem simulações realísticas que levam em consideração muitas características do sistema no mundo real, foram de extrema importância para a abstração das implementações das estratégias. Embora, considerassem o robô como um elemento pontual que ele não é, as simulações foram suficientes para verificar a viabilidade da estratégia adotada e se mostrou uma maneira fácil de verificar o comportamento do sistema para diversas estratégias, como ilustrado

pela Figura 28, contribuindo significativamente para a conclusão deste trabalho.

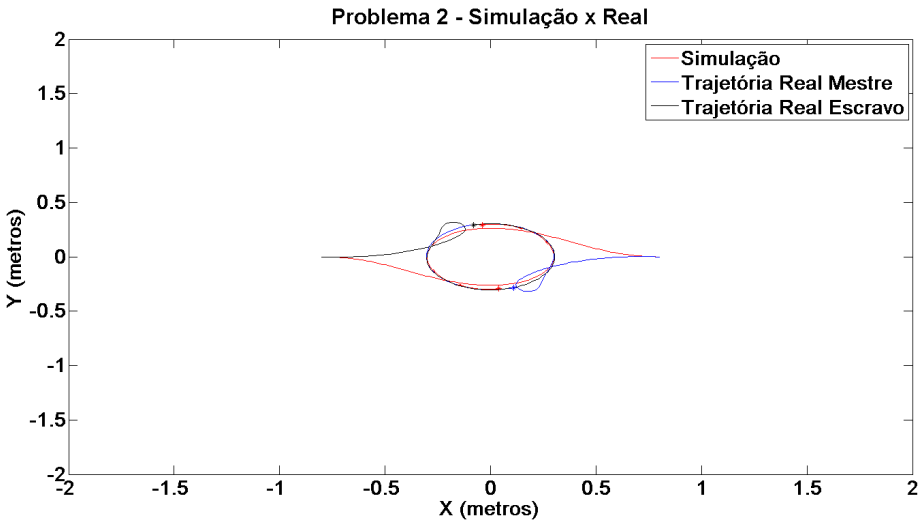


Figura 28 – Comparação entre o problema real e a simulação

No decorrer deste trabalho[1] foram testados alguns controladores distintos e ajustado empiricamente diversos ganhos para um controlador PID que atendesse aos requisitos do sistema e o que pode-se constatar é que embora alguns controladores tenha demostrado erros menores ou tempo de ajuste inferiores, como pode ser observado comparando-se as tabelas 1, 2 e os resultados obtidos na Seção 6.3, o que determinou o melhor controlador para o problema foi um custo benefício entre erro, tempo de ajuste e também, suavidade de controle.

No geral, o controlador da malha interna que melhor atendeu e cumpriu com o objetivo deste trabalho, foi o controlador aqui denominado como polinomial. Este controlador é na verdade, um controlador PID já implementado e ajustado pelo fabricante e o ajuste polinomial foi utilizado somente para se obter uma função que traduza o valor de referência (*set point*) aceito pelo controlador do fabricante em velocidade angular das rodas. Já na malha intermediária (malha 2), a malha retratada na seção 5.2, um controlador proporcional se mostrou suficiente para atingir os objetivos deste trabalho, ao passo que ao inserir um ganho integrador no sistema, o mesmo se mostrou instável.

Foram realizados neste trabalho apenas testes com implementações em uma frota de dois robôs, no entanto, seria interessante como trabalho futuro, aplicar este estudo a uma frota de quatro robôs, como o suportado pela rede de comunicação implementada pela plataforma.

[1]Todos os arquivos para simulações e a implementação deste trabalho se encontram em um repositório do *GitHub*. Para acessar o código-fonte das simulações, das malhas de controle e dos problemas, assim como acessar a alguns dados dos testes realizados neste trabalho, acesse o link: https://github.com/MarianaAthayde/tcc/tree/master.

9.1 Trabalhos Futuros

Como trabalhos futuros sugere-se a implementação de um tratamento de colisão que, embora não implementado neste trabalho, é parte essencial de um controle de formação, visto que para cumprir com um objetivo real as tropas devem ser capazes de trabalhar em conjunto sem que um robô atrapalhe a ação de outro e prejudique o desempenho da frota. A inserção de sensores e de uma malha de controle para que o robô consiga interagir com o ambiente, de modo a tornar o controle de formação mais eficiente e mais próximo aos sistemas que resolvem problemas no mundo real, também é uma perspectiva de continuidade interessante.

Outro trabalho interessante seria fazer robôs que se comportem como indivíduos inteligentes e autônomos, capazes de realizar missões individualmente mas também, capazes de se organizarem em uma sociedade que possui objetivos em comum e sejam capazes de decidir pela "melhor"estratégia de formação para cada problema.

Referências

BALCH, T.; ARKIN, R. C. Behavior-based formation control for multirobot teams. **Robotics and Automation, IEEE Transactions on**, IEEE, v. 14, n. 6, p. 926–939, 1998. Citado na página 7.

BENEDETTELLI, D.; CASINI, M.; GARULLI, A.; GIANNITRAPANI, A.; VICINO, A. A lego mindstorms experimental setup for multi-agent systems. In: **Control Applications, (CCA) Intelligent Control, (ISIC), 2009 IEEE**. [S.l.: s.n.], 2009. p. 1230–1235. Citado na página 4.

CARLES, M.; HERMOSILLA, L. O futuro da medicina: nanomedicina. **Revista Científica Eletrônica de Medicina Veterinária**, v. 6, n. 10, p. 1–7, 2008. Citado na página 1.

CASINI, M.; GARULLI, A.; GIANNITRAPANI, A.; VICINO, A. A lego mindstorms multi-robot setup in the automatic control telelab. In: **Proceedings of 18th IFAC World Congress, Milano, Italy**. [S.l.: s.n.], 2011. v. 28. Citado na página 5.

CHEN, H.; SHENG, W.; XI, N.; SONG, M.; CHEN, Y. Automated robot trajectory planning for spray painting of free-form surfaces in automotive manufacturing. In: **Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on**. [S.l.: s.n.], 2002. v. 1, p. 450–455 vol.1. Citado na página 1.

CORRÊA, M. A.; JÚNIOR, J. B. C. **Estudos de veículos aéreos não tripulados baseado em sistemas multi-agentes e sua interação no espaço aéreo controlado**. [S.l.]: Sitraer, 2008. Citado na página 1.

FRANKLIN, G. F.; POWELL, J. D.; WORKMAN, M. L. **Digital control of dynamic systems**. [S.l.]: Addison-wesley Menlo Park, 1998. v. 3. Citado na página 7.

GIRARD, A. R.; HOWELL, A. S.; HEDRICK, J. K. Border patrol and surveillance missions using multiple unmanned air vehicles. In: IEEE. **Decision and Control, 2004. CDC. 43rd IEEE Conference on**. [S.l.], 2004. v. 1, p. 620–625. Citado na página 2.

GOUVÊA, J. A. **CONTROLE DE FORMAÇÃO DE ROBÔS NÃO-HOLONÔMICOS COM RESTRIÇÃO DE CURVATURA UTILIZANDO FUNÇÃO POTENCIAL**. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2011. Citado na página 6.

JESUS, T. A. **Estratégias de Guiagem e Cooperação de Robôs Aéreos Sujeitos a Restrições nas Entradas e/ou nos Estados**. Tese (Doutorado) — Universidade Federal de Minas Gerais, 2013. Citado 2 vezes nas páginas 2 e 5.

MARJOVI, A.; NUNES, J. G.; MARQUES, L.; ALMEIDA, A. de. Multi-robot exploration and fire searching. In: IEEE. **Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on**. [S.l.], 2009. p. 1929–1934. Citado na página 2.

MARTINEZ, D. L. Reconfigurable multi robot society based on lego mindstorms. Helsinki University of Technology, 2009. Citado 2 vezes nas páginas 4 e 6.

OGATA, K. **Modern control engineering**. Englewood Cliffs (N.J.): Prentice-Hall, 1970. ISBN 0-13-590232-0. Disponível em: <http://opac.inria.fr/record=b1080640>. Citado na página 7.

RAMCHURN, S. D.; HUYNH, D.; JENNINGS, N. R. Trust in multi-agent systems. **Knowl. Eng. Rev.**, Cambridge University Press, New York, NY, USA, v. 19, n. 1, p. 1–25, mar. 2004. ISSN 0269-8889. Disponível em: <http://dx.doi.org/10.1017/S0269888904000116>. Citado na página 1.

SECCHI, H. **Uma Introdução a Robôs Móveis**. [S.l.]: Argentina, 2008. Citado na página 1.

SOURCEFORGE. **Bricx Command Center**. 2001. Disponível em: <http://bricxcc.sourceforge.net/>. Acesso em: 20 de maio de 2015. Citado 2 vezes nas páginas 3 e 9.

TAYLOR, J.; DÍAZ, J.; GRAU, A. **Mecánica clásica**. Reverté, 2013. ISBN 9788429143126. Disponível em: <http://books.google.com.br/books?id=6QJjngEACAAJ>. Citado na página 6.

VIEIRA, F. C. **Controle dinâmico de robôs móveis com acionamento diferencial**. Tese (Doutorado) — Universidade Federal do Rio Grande do Norte, 2005. Citado na página 11.

Apêndices

APÊNDICE A – Código do problema 1: Mestre

```
//Mestre P1 - Mestre
#define BT_CONN 1
#define INBOX 1
#define OUTBOX 5

#define L 0.112
#define RP 0.0216
#define motorR OUT_A
#define motorL OUT_C
float R = 0.3;
float T = 24;

//Auxiliar para gravar no arquivo txt
string write;
short bytesWritten;
byte fileHandle;
string write2;
short bytesWritten2;
byte fileHandle2;

//Converter graus em radianos 1grau = 0,0174533 rad
float convrad = 0.0174533;

int t_amostral = 25;
int amostras = 3000;

long tempo_k = 0.0;
float dt = 0.0;

float countR_k = 0;
float countL_k = 0;
float countR_kplus = 0;
float countL_kplus = 0;

float countR2_k = 0;
float countL2_k = 0;
float countR2_kplus = 0;
float countL2_kplus = 0;
```

```
float wr = 0.0;
float wl = 0.0;
float wdr = 0.0;
float wdl = 0.0;
float ewr = 0.0;
float ewl = 0.0;

float errx = 0.0;
float erry = 0.0;
float errt = 0.0;

//Variaveis do Controlador
float acum_err = 0.0;
float acum_erl = 0.0;
float prev_ewr = 0.0;
float prev_ewl = 0.0;
//Ganhos
float kp = 10;
float ki = 15;
float kd = 0;
float incremento = 3;

//Ganhos
float kp_pos = 2.0;
float ki_pos = 0;
float kd_pos = 0;
float incremento_pos = 1;

float acum_errt = 0;
float errt_past = 0;
float w_controle = 0;

//Posicionamento Inicial do Robo
float xr = 0.0;
float yr = 0.0;
float tetar_k = PI;
float tetad = PI;

float cr = 0;
float cl = 0;
```

```
float ddr = 0;
float ddl = 0;

float P[] = {-0.00091, 0.0223, -0.1537, 6.1864, -0.0546};

string out;
string in;

float vd = 0.1;

sub getWs(){
countR_kplus = MotorRotationCount(motorR);
countL_kplus = MotorRotationCount(motorL);

//Velocidade = delta_Rotacao*constConvGrau2Rad/delta_tempo em
   segundos
wr = (countR_kplus - countR_k)*convrad/dt; //wr em rad/s
wl = (countL_kplus - countL_k)*convrad/dt; //wl em rad/s
countR_k = countR_kplus;
countL_k = countL_kplus;

}

sub malha1(float vLinear,float vAng){

//Definindo as velocidades desejadas de Cada Roda
wdr = (2*vLinear + vAng*L)/(2*RP);
wdl = (2*vLinear - vAng*L)/(2*RP);

//Obtendo as velocidades reais de cada roda
getWs();

//Obtendo os erros de velocidade
ewr = wdr - wr;
ewl = wdl - wl;

//Acoes de controle
//Controle Proporcional
/*cr = kp*ewr;
cl = kp*ewl;*/
//Controle Proporcional Integrador
/*cr = kp*ewr + acum_err + ki*dt*ewr;
```

```
cl = kp*ewl + acum_erl + ki*dt*ewl;*/
//Controlador Proporcional Integrador Derivativo
/*cr = kp*ewr + acum_err + ki*dt*ewr + kd*(ewr - prev_ewr)/dt;
cl = kp*ewl + acum_erl + ki*dt*ewl + kd*(ewl - prev_ewl)/dt;
//Controlador Intedral e Derivativo
//Salvando erro atual
prev_ewr = ewr;
prev_ewl = ewl;

//Atualizando o acumulador de erro
acum_err = acum_err + ki*dt*ewr;
acum_erl = acum_erl + ki*dt*ewl; */

//Incremental
/*if(ewr > 0){
cr = cr + incremento;
}else{
if(ewr < 0 ){
cr = cr - incremento;
}
}
if(ewl > 0){
cl = cl + incremento;
}else{
if(ewr < 0 ){
cl = cl - incremento;
}
}*/

/*
if(cr > 100){
cr = 100;
}
if(cr < -100){
cr = -100;
}
if(cl > 100){
cl = 100;
}
if(cl < -100){
cl = -100;
}
```

```
OnFwd(OUT_A,cr);
OnFwd(OUT_C,cl); */


//Variaveis de Ajuste Polinomial
cr = P[0]*pow(wdr,4) + P[1]*pow(wdr,3) + P[2]*pow(wdr,2) + P[3]*wdr
   + P[4]; // Ajuste polinomial
cl = P[0]*pow(wdl,4) + P[1]*pow(wdl,3) + P[2]*pow(wdl,2) + P[3]*wdl
   + P[4]; // Pwr_{r,l} X w_{r,l}
// Comandos para os motores (Polinomial)
OnFwdReg(motorR, cr, OUT_REGMODE_SPEED);
OnFwdReg(motorL, cl, OUT_REGMODE_SPEED);


//Salvando erros em arquivo
write = StrCat(NumToStr(ewr),"|",NumToStr(ewl),"|",NumToStr(dt));
WriteLnString(fileHandle,write, bytesWritten);
}

sub malha2(/*float xd, float yd*/){
//Na verdade so atualiza a posicao real dos robos (utilizada para
   primeira abordagem)
//Obtendo distancias percorridas por cada roda
countR2_kplus = MotorRotationCount(motorR);
countL2_kplus = MotorRotationCount(motorL);

ddr = ((countR2_kplus - countR2_k) * 2 * PI * RP)/360.0;
ddl = ((countL2_kplus - countL2_k) * 2 * PI * RP)/360.0;
countR2_k = countR2_kplus;
countL2_k = countL2_kplus;

//Obtendo posicao e sentido real
xr = xr + ((ddr+ddl)* cos(tetar_k)/2.0);
yr = yr + ((ddr+ddl)* sin(tetar_k)/2.0);
tetar_k = tetar_k + (ddr - ddl)/L;

//Salvando erros em arquivo
short bytesWritten2;
string write2 = StrCat(NumToStr(xr),"|",NumToStr(yr));
WriteLnString(fileHandle2,write2, bytesWritten2);
```

```
}

sub BTCheck(int conn){
if (!BluetoothStatus(conn)==NO_ERR){
TextOut(5,LCD_LINE2,"Error");
Wait(1000);
Stop(true);
}
}

task main(){

DeleteFile("Tst_Malha1.txt");
CreateFile("Tst_Malha1.txt",8*2048, fileHandle);
DeleteFile("Tst_Malha2.txt");
CreateFile("Tst_Malha2.txt",30*2048, fileHandle2);
//write = StrCat("kp: ",NumToStr(kp),"|ki: ",NumToStr(ki),"|kd:
   ",NumToStr(kd));
//WriteLnString(fileHandle,write, bytesWritten);

BTCheck(BT_CONN); //checa a conexao com o master
float x1;
tempo_k = CurrentTick();
float tempo_inicial = tempo_k;
for(int t = 0; t < amostras; t++){
malha2();
/*******************/
out = StrCat("ok|", NumToStr(xr));
TextOut(10,LCD_LINE1,"Master Test");
TextOut(0,LCD_LINE2,"IN:");
TextOut(0,LCD_LINE3,"OUT:");
ReceiveRemoteString(INBOX, true, in);
SendRemoteString(BT_CONN,OUTBOX,out);
TextOut(10,LCD_LINE3,out);

int index = Pos("|",in);
if(strcmp(Copy(in,0,index), "ok") == 0){
//vi = 0.25;
x1 = atof(Copy(in,index+1,StrLen(in)-index+1));
TextOut(10,LCD_LINE2,in);

}
```

```
Wait(100);
//Calculando dt em segundos
long dt_aux = CurrentTick() - tempo_k;
tempo_k = tempo_k + dt_aux;
dt = dt_aux/1000.0;

//positionControl();
Wait(t_amostral);
/*********/

/*float xd = (xr+x1)/2;
float yd = 0.0;
malha2(xd,yd);*/
float v = vd + 0.2*(xr - x1);
malha1(v,0);

Wait(t_amostral);
}

CloseFile(fileHandle);

}
```

APÊNDICE B – Código do problema 1: Escravo

```
//Mestre P1 - Escravo
#define BT_CONN 0
#define INBOX 5
#define OUTBOX 1

#define L 0.112
#define RP 0.0216
#define motorR OUT_A
#define motorL OUT_C
float R = 0.3;
float T = 24;

//Auxiliar para gravar no arquivo txt
string write;
short bytesWritten;
byte fileHandle;
string write2;
short bytesWritten2;
byte fileHandle2;

//Converter graus em radianos 1grau = 0,0174533 rad
float convrad = 0.0174533;

int t_amostral = 25;
int amostras = 3000;

long tempo_k = 0.0;
float dt = 0.0;

float countR_k = 0;
float countL_k = 0;
float countR_kplus = 0;
float countL_kplus = 0;

float countR2_k = 0;
float countL2_k = 0;
float countR2_kplus = 0;
float countL2_kplus = 0;
```

```
float wr = 0.0;
float wl = 0.0;
float wdr = 0.0;
float wdl = 0.0;
float ewr = 0.0;
float ewl = 0.0;

float errx = 0.0;
float erry = 0.0;
float errt = 0.0;

//Variaveis do Controlador
float acum_err = 0.0;
float acum_erl = 0.0;
float prev_ewr = 0.0;
float prev_ewl = 0.0;
//Ganhos
float kp = 10;
float ki = 15;
float kd = 0;
float incremento = 3;

//Ganhos
float kp_pos = 2.0;
float ki_pos = 0;
float kd_pos = 0;
float incremento_pos = 1;

float acum_errt = 0;
float errt_past = 0;
float w_controle = 0;

//Posicionamento Inicial do Robo
float xr = 0.8;
float yr = 0.0;
float tetar_k = PI;
float tetad = PI;

float cr = 0;
float cl = 0;
```

```
float ddr = 0;
float ddl = 0;

float P[] = {-0.00091, 0.0223, -0.1537, 6.1864, -0.0546};

string out;
string in;

float vd = 0.1;

sub getWs(){
countR_kplus = MotorRotationCount(motorR);
countL_kplus = MotorRotationCount(motorL);

//Velocidade = delta_Rotacao*constConvGrau2Rad/delta_tempo em
   segundos
wr = (countR_kplus - countR_k)*convrad/dt; //wr em rad/s
wl = (countL_kplus - countL_k)*convrad/dt; //wl em rad/s
countR_k = countR_kplus;
countL_k = countL_kplus;

}

sub malha1(float vLinear,float vAng){

//Definindo as velocidades desejadas de Cada Roda
wdr = (2*vLinear + vAng*L)/(2*RP);
wdl = (2*vLinear - vAng*L)/(2*RP);

//Obtendo as velocidades reais de cada roda
getWs();

//Obtendo os erros de velocidade
ewr = wdr - wr;
ewl = wdl - wl;

//Acoes de controle
//Controle Proporcional
/*cr = kp*ewr;
cl = kp*ewl;*/
//Controle Proporcional Integrador
/*cr = kp*ewr + acum_err + ki*dt*ewr;
```

```
cl = kp*ewl + acum_erl + ki*dt*ewl;*/
//Controlador Proporcional Integrador Derivativo
/*cr = kp*ewr + acum_err + ki*dt*ewr + kd*(ewr - prev_ewr)/dt;
cl = kp*ewl + acum_erl + ki*dt*ewl + kd*(ewl - prev_ewl)/dt;
//Controlador Intedral e Derivativo
//Salvando erro atual
prev_ewr = ewr;
prev_ewl = ewl;

//Atualizando o acumulador de erro
acum_err = acum_err + ki*dt*ewr;
acum_erl = acum_erl + ki*dt*ewl; */

//Incremental
/*if(ewr > 0){
cr = cr + incremento;
}else{
if(ewr < 0 ){
cr = cr - incremento;
}
}
if(ewl > 0){
cl = cl + incremento;
}else{
if(ewr < 0 ){
cl = cl - incremento;
}
}*/

/*
if(cr > 100){
cr = 100;
}
if(cr < -100){
cr = -100;
}
if(cl > 100){
cl = 100;
}
if(cl < -100){
cl = -100;
}
```

```
OnFwd(OUT_A,cr);
OnFwd(OUT_C,cl); */


//Variaveis de Ajuste Polinomial
cr = P[0]*pow(wdr,4) + P[1]*pow(wdr,3) + P[2]*pow(wdr,2) + P[3]*wdr
   + P[4]; // Ajuste polinomial
cl = P[0]*pow(wdl,4) + P[1]*pow(wdl,3) + P[2]*pow(wdl,2) + P[3]*wdl
   + P[4]; // Pwr_{r,l} X w_{r,l}
// Comandos para os motores (Polinomial)
OnFwdReg(motorR, cr, OUT_REGMODE_SPEED);
OnFwdReg(motorL, cl, OUT_REGMODE_SPEED);


//Salvando erros em arquivo
write = StrCat(NumToStr(ewr),"|",NumToStr(ewl),"|",NumToStr(dt));
WriteLnString(fileHandle,write, bytesWritten);
}

sub malha2(/*float xd, float yd*/){
//Nao implementa malha 2 e sim uma adaptacao. Apenas atualiza a
   posicao real
//Obtendo distancias percorridas por cada roda
countR2_kplus = MotorRotationCount(motorR);
countL2_kplus = MotorRotationCount(motorL);

ddr = ((countR2_kplus - countR2_k) * 2 * PI * RP)/360.0;
ddl = ((countL2_kplus - countL2_k) * 2 * PI * RP)/360.0;
countR2_k = countR2_kplus;
countL2_k = countL2_kplus;

//Obtendo posicao e sentido real
xr = xr + ((ddr+ddl)* cos(tetar_k)/2.0);
yr = yr + ((ddr+ddl)* sin(tetar_k)/2.0);
tetar_k = tetar_k + (ddr - ddl)/L;

//Salvando erros em arquivo
short bytesWritten2;
string write2 = StrCat(NumToStr(xr),"|",NumToStr(yr));
WriteLnString(fileHandle2,write2, bytesWritten2);
```

```
}

sub BTCheck(int conn){
if (!BluetoothStatus(conn)==NO_ERR){
TextOut(5,LCD_LINE2,"Error");
Wait(1000);
Stop(true);
}
}

task main(){

DeleteFile("Tst_Malha1.txt");
CreateFile("Tst_Malha1.txt",8*2048, fileHandle);
DeleteFile("Tst_Malha2.txt");
CreateFile("Tst_Malha2.txt",30*2048, fileHandle2);
//write = StrCat("kp: ",NumToStr(kp),"|ki: ",NumToStr(ki),"|kd:
   ",NumToStr(kd));
//WriteLnString(fileHandle,write, bytesWritten);

BTCheck(BT_CONN); //checa a conexao com o master
float x1;
tempo_k = CurrentTick();
float tempo_inicial = tempo_k;
for(int t = 0; t < amostras; t++){
malha2();
/*******************/
out = StrCat("ok|", NumToStr(xr));
TextOut(10,LCD_LINE1,"Master Test");
TextOut(0,LCD_LINE2,"IN:");
TextOut(0,LCD_LINE3,"OUT:");
ReceiveRemoteString(INBOX, true, in);
SendResponseString(OUTBOX,out);
TextOut(10,LCD_LINE3,out);

int index = Pos("|",in);
if(strcmp(Copy(in,0,index), "ok") == 0){
//vi = 0.25;
x1 = atof(Copy(in,index+1,StrLen(in)-index+1));
TextOut(10,LCD_LINE2,in);

}
```

```
Wait(100);
//Calculando dt em segundos
long dt_aux = CurrentTick() - tempo_k;
tempo_k = tempo_k + dt_aux;
dt = dt_aux/1000.0;

//positionControl();
Wait(t_amostral);
/*********/

/*float xd = (xr+x1)/2;
float yd = 0.0;
malha2(xd,yd);*/
float v = vd + 0.2*(xr - x1);
TextOut(0,LCD_LINE4,NumToStr(v));
malha1(v,0);

Wait(t_amostral);
}

CloseFile(fileHandle);

}
```

APÊNDICE C – Código do problema 2: Mestre

```
//Mestre P2 - Mestre
#define BT_CONN 1
#define INBOX 1
#define OUTBOX 5

#define L 0.112
#define RP 0.0216
#define motorR OUT_A
#define motorL OUT_C
float R = 0.3;
float T = 12;

//Auxiliar para gravar no arquivo txt
string write;
short bytesWritten;
byte fileHandle;
string write2;
short bytesWritten2;
byte fileHandle2;

//Converter graus em radianos 1grau = 0,0174533 rad
float convrad = 0.0174533;

int t_amostral = 25;
int amostras = 3000;

long tempo_k = 0.0;
float dt = 0.0;

float countR_k = 0;
float countL_k = 0;
float countR_kplus = 0;
float countL_kplus = 0;

float countR2_k = 0;
float countL2_k = 0;
float countR2_kplus = 0;
float countL2_kplus = 0;
```

```
float wr = 0.0;
float wl = 0.0;
float wdr = 0.0;
float wdl = 0.0;
float ewr = 0.0;
float ewl = 0.0;

float errx = 0.0;
float erry = 0.0;
float errt = 0.0;

//Variaveis do Controlador
float acum_err = 0.0;
float acum_erl = 0.0;
float prev_ewr = 0.0;
float prev_ewl = 0.0;
//Ganhos
float kp = 10;
float ki = 15;
float kd = 0;
float incremento = 3;

//Ganhos
float kp_pos = 2.0;
float ki_pos = 0;
float kd_pos = 0;
float incremento_pos = 1;

float acum_errt = 0;
float errt_past = 0;
float w_controle = 0;

//Posicionamento Inicial do Robo
float xr = 0.8;
float yr = 0.0;
float tetar_k = PI;
float tetad = PI;

float cr = 0;
float cl = 0;
```

```
float ddr = 0;
float ddl = 0;

float P[] = {-0.00091, 0.0223, -0.1537, 6.1864, -0.0546};

string out;
string in;

float vd = 0.1;

int num_robos = 1;
int num = 0;
float wt = 0;

sub getWs(){
countR_kplus = MotorRotationCount(motorR);
countL_kplus = MotorRotationCount(motorL);

//Velocidade = delta_Rotacao*constConvGrau2Rad/delta_tempo em
   segundos
wr = (countR_kplus - countR_k)*convrad/dt; //wr em rad/s
wl = (countL_kplus - countL_k)*convrad/dt; //wl em rad/s
countR_k = countR_kplus;
countL_k = countL_kplus;

}

sub malha1(float vLinear,float vAng){

//Definindo as velocidades desejadas de Cada Roda
wdr = (2*vLinear + vAng*L)/(2*RP);
wdl = (2*vLinear - vAng*L)/(2*RP);

//Obtendo as velocidades reais de cada roda
getWs();

//Obtendo os erros de velocidade
ewr = wdr - wr;
ewl = wdl - wl;

//Acoes de controle
//Controle Proporcional
```

```
/*cr = kp*ewr;
cl = kp*ewl;*/
//Controle Proporcional Integrador
/*cr = kp*ewr + acum_err + ki*dt*ewr;
cl = kp*ewl + acum_erl + ki*dt*ewl;*/
//Controlador Proporcional Integrador Derivativo
/*cr = kp*ewr + acum_err + ki*dt*ewr + kd*(ewr - prev_ewr)/dt;
cl = kp*ewl + acum_erl + ki*dt*ewl + kd*(ewl - prev_ewl)/dt;
//Controlador Intedral e Derivativo
//Salvando erro atual
prev_ewr = ewr;
prev_ewl = ewl;

//Atualizando o acumulador de erro
acum_err = acum_err + ki*dt*ewr;
acum_erl = acum_erl + ki*dt*ewl; */

//Incremental
/*if(ewr > 0){
cr = cr + incremento;
}else{
if(ewr < 0 ){
cr = cr - incremento;}
}
if(ewl > 0){
cl = cl + incremento;
}else{
if(ewr < 0 ){
cl = cl - incremento;}
}

if(cr > 100){
cr = 100;
}
if(cr < -100){
cr = -100;
}
if(cl > 100){
cl = 100;
}
if(cl < -100){
cl = -100;
```

```
}

OnFwd(OUT_A,cr);
OnFwd(OUT_C,cl);*/


//Variaveis de Ajuste Polinomial
cr = P[0]*pow(wdr,4) + P[1]*pow(wdr,3) + P[2]*pow(wdr,2) + P[3]*wdr
   + P[4]; // Ajuste polinomial
cl = P[0]*pow(wdl,4) + P[1]*pow(wdl,3) + P[2]*pow(wdl,2) + P[3]*wdl
   + P[4]; // Pwr_{r,l} X w_{r,l}
// Comandos para os motores (Polinomial)
OnFwdReg(motorR, cr, OUT_REGMODE_SPEED);
OnFwdReg(motorL, cl, OUT_REGMODE_SPEED);


//Salvando erros em arquivo
write = StrCat(NumToStr(ewr),"|",NumToStr(ewl),"|",NumToStr(dt));
WriteLnString(fileHandle,write, bytesWritten);
}

sub malha2(float xd, float yd){
//Obtendo distancias percorridas por cada roda
countR2_kplus = MotorRotationCount(motorR);
countL2_kplus = MotorRotationCount(motorL);

ddr = ((countR2_kplus - countR2_k) * 2 * PI * RP)/360.0;
ddl = ((countL2_kplus - countL2_k) * 2 * PI * RP)/360.0;
countR2_k = countR2_kplus;
countL2_k = countL2_kplus;

//Obtendo posicao e sentido real
xr = xr + ((ddr+ddl)* cos(tetar_k)/2.0);
yr = yr + ((ddr+ddl)* sin(tetar_k)/2.0);
tetar_k = tetar_k + (ddr - ddl)/L;

errx = xd - xr;
erry = yd - yr;

tetad = atan2(erry,errx);

errt_past = errt;
```

```
//Diminuir de acordo com o erro
//velLinear_Desejada = 2 * sqrt((pow(errx,2)+pow(erry,2)));

errt = tetad - tetar_k;
errt = atan2(sin(errt),cos(errt));

//Controle P
//float w_controled = kp_pos*errt;

//Controle PI
//w_controled = kp_pos*errt + acum_errt + ki_pos*dt*errt;
//Controle PID
//float w_controled = kp_pos*errt + acum_errt + ki_pos*dt*errt +
   kd_pos * (errt - errt_past)/dt;
w_controle = kp_pos*errt + acum_errt + ki_pos*errt*dt + kd_pos *
   (errt - errt_past)/dt;
acum_errt = acum_errt + ki_pos*errt*dt;
acum_errt = acum_errt + ki_pos*dt*errt;

//Salvando erros em arquivo
short bytesWritten2;
string write2 = StrCat(NumToStr(w_controle));
WriteLnString(fileHandle2,write2, bytesWritten2);

float dist = sqrt((pow(errx,2)+(pow(erry,2))));
float v = (2*PI*R/T)/num_robos;
if(dist > 0.05){
v = v*1.1;
}
malha1(v, w_controle);
}


sub BTCheck(int conn){
if (!BluetoothStatus(conn)==NO_ERR){
TextOut(5,LCD_LINE2,"Error");
Wait(1000);
Stop(true);
}
}

task main(){
```

```
DeleteFile("Mestre_2.txt");
CreateFile("Mestre_2.txt",38*2048, fileHandle2);
DeleteFile("Mestre_1.txt");
//CreateFile("Mestre_1.txt",8*2048, fileHandle);
//write = StrCat("kp: ",NumToStr(kp),"|ki: ",NumToStr(ki),"|kd:
   ",NumToStr(kd));
//WriteLnString(fileHandle,write, bytesWritten);

BTCheck(BT_CONN); //checa a conexao com o master
float x1;
tempo_k = CurrentTick();
float tempo_inicial = tempo_k;
for(int t = 0; t < amostras; t++){
/*******************/
if(!BluetoothStatus(BT_CONN)==NO_ERR){
num_robos = 1;
}else{
num_robos = 2;
}
/*********************/
//Calculando dt em segundos
long dt_aux = CurrentTick() - tempo_k;
tempo_k = tempo_k + dt_aux;
dt = dt_aux/1000.0;

/*********************/

wt = ((tempo_k/1000.0)*(2*PI/T))/num_robos;

/*********************/

out = StrCat(NumToStr(num_robos),"|", NumToStr(wt));
TextOut(10,LCD_LINE1,"Master Test");
//      TextOut(0,LCD_LINE2,"IN:");
TextOut(0,LCD_LINE3,"OUT:");
//ReceiveRemoteString(INBOX, true, in);
SendRemoteString(BT_CONN,OUTBOX,out);
TextOut(10,LCD_LINE3,out);

float xd = R*cos(wt + num*(2*PI)/num_robos);
float yd = R*sin(wt + num*(2*PI)/num_robos);
```

```
malha2(xd,yd);
Wait(200);
//Wait(t_amostral);
}

CloseFile(fileHandle);

}
```

APÊNDICE D – Código do problema 2: Escravo

```
//Mestre P2 - Mestre
#define BT_CONN 0
#define INBOX 5
#define OUTBOX 1

#define L 0.112
#define RP 0.0216
#define motorR OUT_A
#define motorL OUT_C
float R = 0.3;
float T = 12;

//Auxiliar para gravar no arquivo txt
string write;
short bytesWritten;
byte fileHandle;
string write2;
short bytesWritten2;
byte fileHandle2;

//Converter graus em radianos 1grau = 0,0174533 rad
float convrad = 0.0174533;

int t_amostral = 25;
int amostras = 3000;

long tempo_k = 0.0;
float dt = 0.0;

float countR_k = 0;
float countL_k = 0;
float countR_kplus = 0;
float countL_kplus = 0;

float countR2_k = 0;
float countL2_k = 0;
float countR2_kplus = 0;
float countL2_kplus = 0;
```

```
float wr = 0.0;
float wl = 0.0;
float wdr = 0.0;
float wdl = 0.0;
float ewr = 0.0;
float ewl = 0.0;

float errx = 0.0;
float erry = 0.0;
float errt = 0.0;

//Variaveis do Controlador
float acum_err = 0.0;
float acum_erl = 0.0;
float prev_ewr = 0.0;
float prev_ewl = 0.0;
//Ganhos
float kp = 10;
float ki = 15;
float kd = 0;
float incremento = 3;

//Ganhos
float kp_pos = 2.0;
float ki_pos = 0;
float kd_pos = 0;
float incremento_pos = 1;

float acum_errt = 0;
float errt_past = 0;
float w_controle = 0;

//Posicionamento Inicial do Robo
float xr = -0.8;
float yr = 0.0;
float tetar_k = 0;
float tetad = 0;

float cr = 0;
float cl = 0;
```

```
float ddr = 0;
float ddl = 0;

float P[] = {-0.00091, 0.0223, -0.1537, 6.1864, -0.0546};

string out;
string in;

float vd = 0.1;

int num_robos = 1;
int num = 1;
float wt = 0;

sub getWs(){
countR_kplus = MotorRotationCount(motorR);
countL_kplus = MotorRotationCount(motorL);

//Velocidade = delta_Rotacao*constConvGrau2Rad/delta_tempo em
   segundos
wr = (countR_kplus - countR_k)*convrad/dt; //wr em rad/s
wl = (countL_kplus - countL_k)*convrad/dt; //wl em rad/s
countR_k = countR_kplus;
countL_k = countL_kplus;

}

sub malha1(float vLinear,float vAng){

//Definindo as velocidades desejadas de Cada Roda
wdr = (2*vLinear + vAng*L)/(2*RP);
wdl = (2*vLinear - vAng*L)/(2*RP);

//Obtendo as velocidades reais de cada roda
getWs();

//Obtendo os erros de velocidade
ewr = wdr - wr;
ewl = wdl - wl;

//Acoes de controle
//Controle Proporcional
```

```
/*cr = kp*ewr;
cl = kp*ewl;*/
//Controle Proporcional Integrador
/*cr = kp*ewr + acum_err + ki*dt*ewr;
cl = kp*ewl + acum_erl + ki*dt*ewl;*/
//Controlador Proporcional Integrador Derivativo
/*cr = kp*ewr + acum_err + ki*dt*ewr + kd*(ewr - prev_ewr)/dt;
cl = kp*ewl + acum_erl + ki*dt*ewl + kd*(ewl - prev_ewl)/dt;
//Controlador Intedral e Derivativo
//Salvando erro atual
prev_ewr = ewr;
prev_ewl = ewl;

//Atualizando o acumulador de erro
acum_err = acum_err + ki*dt*ewr;
acum_erl = acum_erl + ki*dt*ewl; */

//Incremental
/*if(ewr > 0){
cr = cr + incremento;
}else{
if(ewr < 0 ){
cr = cr - incremento;}
}
if(ewl > 0){
cl = cl + incremento;
}else{
if(ewr < 0 ){
cl = cl - incremento;}
}


if(cr > 100){
cr = 100;}
if(cr < -100){
cr = -100;}
if(cl > 100){
cl = 100;}
if(cl < -100){
cl = -100;}

OnFwd(OUT_A,cr);
```

```
OnFwd(OUT_C,cl);*/


//Variaveis de Ajuste Polinomial
cr = P[0]*pow(wdr,4) + P[1]*pow(wdr,3) + P[2]*pow(wdr,2) + P[3]*wdr
   + P[4]; // Ajuste polinomial
cl = P[0]*pow(wdl,4) + P[1]*pow(wdl,3) + P[2]*pow(wdl,2) + P[3]*wdl
   + P[4]; // Pwr_{r,l} X w_{r,l}
// Comandos para os motores (Polinomial)
OnFwdReg(motorR, cr, OUT_REGMODE_SPEED);
OnFwdReg(motorL, cl, OUT_REGMODE_SPEED);


//Salvando erros em arquivo
write = StrCat(NumToStr(ewr),"|",NumToStr(ewl),"|",NumToStr(dt));
WriteLnString(fileHandle,write, bytesWritten);
}

sub malha2(float xd, float yd){
//Obtendo distancias percorridas por cada roda
countR2_kplus = MotorRotationCount(motorR);
countL2_kplus = MotorRotationCount(motorL);

ddr = ((countR2_kplus - countR2_k) * 2 * PI * RP)/360.0;
ddl = ((countL2_kplus - countL2_k) * 2 * PI * RP)/360.0;
countR2_k = countR2_kplus;
countL2_k = countL2_kplus;

//Obtendo posicao e sentido real
xr = xr + ((ddr+ddl)* cos(tetar_k)/2.0);
yr = yr + ((ddr+ddl)* sin(tetar_k)/2.0);
tetar_k = tetar_k + (ddr - ddl)/L;

errx = xd - xr;
erry = yd - yr;

tetad = atan2(erry,errx);

errt_past = errt;
//Diminuir de acordo com o erro
//velLinear_Desejada = 2 * sqrt((pow(errx,2)+pow(erry,2)));
```

```
errt = tetad - tetar_k;
errt = atan2(sin(errt),cos(errt));

//Controle P
//float w_controled = kp_pos*errt;

//Controle PI
//w_controled = kp_pos*errt + acum_errt + ki_pos*dt*errt;
//Controle PID
//float w_controled = kp_pos*errt + acum_errt + ki_pos*dt*errt +
   kd_pos * (errt - errt_past)/dt;
w_controle = kp_pos*errt + acum_errt + ki_pos*errt*dt + kd_pos *
   (errt - errt_past)/dt;
acum_errt = acum_errt + ki_pos*errt*dt;
acum_errt = acum_errt + ki_pos*dt*errt;

//Salvando erros em arquivo
short bytesWritten2;
string write2 = StrCat(NumToStr(xr),"|",NumToStr(yr));
WriteLnString(fileHandle2,write2, bytesWritten2);

float dist = sqrt((pow(errx,2)+(pow(erry,2))));
float v = (2*PI*R/T)/num_robos;
if(dist > 0.05){
v = v*1.1;
}
//v = v + dist*0.2;
/*if(dist < 0.05){
v = 0;
w_controle = 0;
} */
malha1(v, w_controle);
}


sub BTCheck(int conn){
if (!BluetoothStatus(conn)==NO_ERR){
TextOut(5,LCD_LINE2,"Error");
Wait(1000);
Stop(true);}
}
```

```
task main(){

DeleteFile("Escravo_2.txt");
CreateFile("Escravo_2.txt",38*2048, fileHandle2);
DeleteFile("Escravo_1.txt");
//CreateFile("Escravo_1.txt",8*2048, fileHandle);
//write = StrCat("kp: ",NumToStr(kp),"|ki: ",NumToStr(ki),"|kd:
   ",NumToStr(kd));
//WriteLnString(fileHandle,write, bytesWritten);

BTCheck(BT_CONN); //checa a conexao com o master
float x1;
tempo_k = CurrentTick();
float tempo_inicial = tempo_k;
for(int t = 0; t < amostras; t++){
/*******************/
if(!BluetoothStatus(BT_CONN)==NO_ERR){
num_robos = 1;
}else{
num_robos = 2;
}
/*********************/
//Calculando dt em segundos
long dt_aux = CurrentTick() - tempo_k;
tempo_k = tempo_k + dt_aux;
dt = dt_aux/1000.0;

TextOut(10,LCD_LINE1,"Master Test");
TextOut(0,LCD_LINE2,"IN:");
TextOut(0,LCD_LINE3,"OUT:");
ReceiveRemoteString(INBOX, true, in);

int index = Pos("|",in);
num_robos = StrToNum(Copy(in,0,index));
wt = atof(Copy(in,index+1,StrLen(in)-index+1));
TextOut(10,LCD_LINE2,in);

float xd = R*cos(wt + num*(2*PI)/num_robos);
float yd = R*sin(wt + num*(2*PI)/num_robos);
malha2(xd,yd);
Wait(200);
//Wait(t_amostral);
```

```
}

CloseFile(fileHandle);
CloseFile(fileHandle2);
}
```