Universidade Federal de Juiz de Fora

Faculdade de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

Alexandre Menezes Teixeira

Coordenação Ótima de Múltiplos Robôs de Serviço em Tarefas Persistentes

Juiz de Fora

2015

Alexandre Menezes Teixeira

Coordenação Ótima de Múltiplos Robôs de Serviço em Tarefas Persistentes

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora, na área de concentração em Sistemas Eletrônicos, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: André Luís Marques Marcato

Juiz de Fora

2015

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Menezes Teixeira, Alexandre.
Coordenação Ótima de Múltiplos Robôs de Serviço em Tarefas Persistentes / Alexandre Menezes Teixeira. – 2015.
83 f. : il.

Orientador: André Luís Marques Marcato
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Faculdade de Engenharia. Programa de Pós-Graduação em Engenharia Elétrica, 2015.

1. Coordenação de Robôs. 2. Ausência de Colisão. 3. ROS. I. L.M. Marcato, André. II. Coordenaçã Ótima de Múltiplos Robôs de Serviço em Tarefas Persistentes.

Alexandre Menezes Teixeira

Coordenação Ótima de Múltiplos Robôs de Serviço em Tarefas Persistentes

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora, na área de concentração em Sistemas Eletrônicos, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. André Luís Marques Marcato - Orientador
Universidade Federal de Juiz de Fora

Professor Dr. Guilherme Augusto Silva Pereira
Universidade Federal de Minas Gerais

Professor Dr. Leonardo de Mello Honório
Universidade Federal de Juiz de Fora

Professor Dr. Leonardo de Rocha Olivi
Universidade Federal de Juiz de Fora

AGRADECIMENTOS

Primeiramente agradeço muito a Deus por ter preservado minha saúde, sendo a única condição necessária para alcançar todo o resto. Agradeço a Ele também por me iluminar, ajudar a tomar as decisões corretas e por me dar forças para superar os obstáculos que surgiram na minha vida.

Agradeço imensamente minha família e namorada por me apoiar nas minhas escolhas e tornar minha vida mais fácil, através de palavras de carinho, conselho e apoio.

Ao meu orientador, que acima de tudo conquistou minha admiração pessoal e profissional, tornando uma pessoa importante nessa fase da minha vida.

A todos os amigos do *GRIn*, que contribuíram de forma efetiva para o desenvolvimento deste trabalho, além de sempre estarem presentes nas horas difíceis.

A banca examinadora, pelo tempo dedicado a correção do trabalho e pelas sugestões.

E a todas as outras pessoas que de alguma forma torcem pelo meu sucesso, o meu muito obrigado.

"O conhecimento serve para encantar as pessoas, não para humilhá-las".

Mario Sérgio Cortella

RESUMO

Monitorar um ambiente através de múltiplos robôs autônomos em um espaço compartilhado sem que haja colisões é um grande desafio nos dias atuais. Várias pesquisas na área de robótica tem sido feitas para desenvolver essa tarefa. Sendo assim, este trabalho atua na coordenação de movimento de múltiplos robôs de serviço percorrendo de forma cíclica caminhos que se interceptam.

É desejável que os robôs sigam pelos caminhos inicialmente planejados sem mudá-los durante a missão. Para evitar possíveis colisões, um controlador central foi desenvolvido para planejar as velocidades médias que os robôs deverão efetuar em cada trecho do circuito, maximizando o menor intervalo de tempo Δt que eles cruzam por um mesmo ponto de colisão.

A solução centralizada utilizada neste trabalho foi modelada como um problema *Mixed Interger Linear Programming* (MILP) sendo usado o software *Linear Interactive and General Optimizer* (LINGO) para maximizar Δt e encontrar para cada robô os tempos de percurso para cada parte do caminho.

Foi desenvolvida uma aplicação em C++ capaz de iniciar o processo de otimização e receber os tempos otimizados de percurso de cada robô, usados para determinar as suas respectivas velocidades médias. Além disso, esta rotina de programação teve como objetivo realizar o controle dos robôs, navegando-os com o auxílio do *framework Robot Operating System* (ROS) integrado ao *LINGO*.

Para localizá-los no espaço utilizou-se um pacote de visão computacional do *ROS* denominado *ARPOSE*, com o intuito de descobrir suas respectivas posições e orientações em relação à um eixo de referência global.

Por fim, simulações e testes reais foram realizados sobre um grupo de robôs para demonstrar a eficácia da abordagem. Os resultados obtidos foram satisfatórios, o que possibilitou atingir os objetivos propostos.

Palavras-chave: Coordenação, Robôs Móveis, Ausência de Colisões, ROS, Localização.

ABSTRACT

Monitoring an environment using multi-autonomous robots operating in a shared workspace without collisions with each other is a great challenge nowadays. Many researchs have been done to develop this important task in robotics. Thus, this thesis deals with coordinating the motions of multiple working robots traversing periodic intersecting paths.

It is desired that mobile robots following the initially designed paths without change them during the mission. To avoid any collision a centralized controller was developed to planner the velocity profile of the robots over the predefined paths, maximizing the smallest time margin Δt that the robots cross over a same collision point.

The centralized solution used in this work was modeled as a Mixed Integer Linear Programming (MILP) problem using the Linear Interactive and General Optimizer (LINGO) software to maximize Δt and to find for each robot the sub-path traversal time.

It was developed a C++ code capable to start the optimization process and receive for each robot the optimized traversal times, used to calculate their respectives average speeds. Besides, this code was used to control the robots, which was done through the Robot Operating System (ROS) framework integrated with LINGO.

A computer vision ROS package named ARPOSE was used to localize the robots during the task, calculating their poses relative the global reference frame.

Finally, to demonstrate the effectiveness of the approach, simulantions and real tests was performed on mobile robots group. The results obtained were satisfactory, which allowed to achieve the goals.

Key-words: Coordinating, Mobile Robots, Collision Avoidance, ROS, Localization.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

NHTSA *National Highway Traffic Safety Administration*

VAANT Veículo Aéreo Autônomo Não-Tripulado

PLIM Programação Linear Inteira Mista

MILP *Mixed Integer Linear Programming*

ROS *Robot Operating System*

SLAM *Simultaneous Localization and Mapping*

OSRF *Open Source Robotics Foundation*

NASA *National Aeronautics and Space Administration*

DARPA *Defense Advanced Research Projects Agency*

API *Application Programming Interface*

AR *Augmented Reality*

PI Proporcional-Integral

PID Proporcional-Integral-Derivativo

LISTA DE SÍMBOLOS

i	Índice referente aos robôs.
j	Índice referente aos pontos.
k	Índice referente aos ciclos de trabalho.
P_i	Ponto de Colisão ou Inicial em Estudo
$\mathcal{R}$	Conjunto de Robôs
$\mathcal{C}_i$	Caminho percorrido pelo robô $i \in \mathcal{R}$
$\mathcal{J}$	Conjunto dos pontos de colisão j.
$\mathcal{S}_j$	Conjunto de robôs i que podem colidir no ponto j
$\mathcal{Y}_i$	Conjunto de Pontos de Colisão presentes no caminho do robô i.
τ_{min}, τ_{max}	Tempo mínimo e máximo para percorrer um dado trecho.
T	Refere-se a base de tempo.
T_i	Tempo de conclusão de um ciclo.
Δt	Menor intervalo de tempo entre a passagem dos robôs por um mesmo ponto de colisão.

NOTAÇÕES:

$t_j(i,k)$	Tempo que o robô $i \in \mathcal{R}$ chega no ponto $j \in \mathcal{J}$ no k-ésimo ciclo.
$s(j,i)$	Função que mapeia para o próximo ponto.
$a(j,i)$	Função que mapeia para o ponto anterior.
$L(j,i)$	Retorna o comprimento do ponto em questão em relação ao ponto inicial.
$l(i)$	Mapeia o robô i para o ponto inicial.
${}^{C}_{M}R$	Matriz de rotação de {M} em relação {C}.
${}^{G}_{C}T$	Matriz de transformação homogênea de {C} em relação a {G}.

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

O uso de veículos autônomos para realizar determinadas tarefas tem se tornado cada vez mais comum. Este crescimento deve-se ao desenvolvimento tecnológico e científico, principalmente em áreas da eletrônica, computação, controle e automação, impulsionado pelo interesse de governos e empresas pelo grande potencial das aplicações civis e militares desses sistemas, que podem proporcionar uma verdadeira revolução no que diz respeito à forma como são executadas determinadas tarefas, na eficiência e redução de custos.

Um exemplo de destaque conforme divulgado por [1] é o fato de grandes montadoras de veículos, como Volvo, Toyota, General Motors, Volkswagen e empresas como a Google e Intel anunciarem que a partir de 2020 carros autônomos começarão a ser comercializados. Estes veículos possuirão equipamentos como câmeras de vídeo, GPS, *scanners laser*, além de uma central de inteligência que receberá todos os dados desses sensores, realizará o processamento das informações e enviará comandos de forma a permitir que o carro seja capaz de se dirigir para um determinado local de forma completamente autônoma, respeitando a legislação de trânsito.

A Google afirma que já testou o sistema por 1,1 milhão de quilômetros em carros convencionais, sendo registrados dois acidentes, no qual em ambos o carro estava no modo manual. Umas das principais vantagens desses veículos é a promessa de uma grande queda nos números de acidente de trânsito. Segundo a matéria publicada pelo site da Organização das Nações Unidas, estima-se que morrem todos os anos 1,2 milhão de pessoas e que mais de 50 milhões se ferem por causa desse tipo de acidente em todo o planeta [2]. Dados de uma agência de segurança de tráfegos dos EUA *National Highway Traffic Safety Administration* (NHTSA), indicam que uma faixa de 70% a 90% desses acidentes acontecem por erros do motorista. Segundo Jonas Ekmark, gerente do centro de segurança da Volvo, "a expectativa é que estes carros autônomos a partir de 2020 evitem 100% dos acidentes fatais" [1].

Outra grande mudança na forma como é realizado um determinado trabalho proporcionada pelo desenvolvimento de veículos autônomos, é o caso da empresa Amazon (a maior empresa de comércio eletrônico do mundo com sede nos Estados Unidos) que pretende fazer entregas através de Veículo Aéreo Autônomo Não-Tripulado (VAANT) [3]. O VAANT denominado Prime Air poderá carregar pacotes de aproximadamente 2,3Kg, o que representa 86% das encomendas da empresa [4], [5].

Outras diversas aplicações no campo da robótica móvel podem ser citadas, como vigilância, reconhecimento de terreno [6], monitoramento de queimadas [7], inspeção de linhas aéreas de transmissão [8] e [9]. Estas aplicações exigem do sistema robótico um

alto nível de autonomia, confiança e robustez. Nestes cenários, o uso de múltiplos robôs apresenta muitas vantagens para a execução da missão, pois podem melhorar a eficiência da tarefa, uma vez que é possível reduzir significativamente o tempo para realização de um determinado trabalho.

Mas para que isso seja possível, muitas vezes é necessário que esses robôs atuem de forma cooperativa. Assim, diversos grupos de pesquisa vêm concentrando esforços para que sejam desenvolvidas técnicas de coordenação e controle de veículos autônomos aéreos ou terrestres não tripulados [10] e [11].

O presente trabalho foi inspirado no artigo intitulado *Coordination of multiple fixed-wing UAVs traversing intersecting periodic paths* [10] e trata da coordenação dos movimentos de múltiplos robôs sobre caminhos cíclicos pré-definidos que se interceptam. Deseja-se que os robôs sigam por esses caminhos de forma a nunca estarem nestes pontos de interseção ao mesmo tempo. Para isso, elaborou-se uma estratégia centralizada que define para cada robô, um perfil de velocidade ao longo do caminho visando maximizar a diferença entre os tempos de chegada dos robôs em cada ponto de colisão. Isso quer dizer que quanto maior for este intervalo de tempo, mais seguro a tarefa se torna.

Uma das condições definidas neste trabalho é que o robô não possa parar, pois deseja-se que a solução proposta possa ser aplicada em diversas bases robóticas, sejam elas terrestres ou aéreas. Em se tratando desta última, para alguns tipos de aeronaves como as de asa fixa, por exemplo, parar não é uma opção, já que ocasionaria a queda da mesma. Outra condição é que o caminho original não sofra qualquer tipo de alteração e esteja livre de obstáculos, pois pode acontecer do robô estar inserido em um ambiente muito limitado fisicamente, o que não permitiria margem para manobras.

Como se trata de um problema de Programação Linear Inteira Mista (PLIM), do inglês *Mixed Integer Linear Programming* (MILP), será utilizado um *solver* de otimização denominado LINGO para maximizar a função objetivo que é a diferença entre os tempos de chegada dos robôs a um determinado ponto de colisão. Assim, será feita uma aplicação em C++ para controlar a navegação, integrando este *solver* ao *framework Robot Operating System (ROS)*, que será utilizado para realizar a navegação do robô durante o experimento.

Uma vez que a leitura da hodometria do robô tende a integrar erros ocasionados por diversos motivos, como diferença entre o raio das rodas, derrapagens e mudanças bruscas de direção [12], será utilizado uma ferramenta do ROS para localização baseada em visão computacional denominada *ARTOOLKIT*, que surge como alternativa para estimar a posição do robô, mesmo quando acontecem os erros citados acima.

1.2 DEFINIÇÃO DO PROBLEMA E OBJETIVOS

Considere um conjunto de robôs se movimentando de forma cíclica em caminhos que se interceptam em um ambiente de trabalho compartilhado. Além disso, considere que não seja permitido aos robôs interromperem seus movimentos, como no caso de VAANT's de asa fixa, e alterar os caminhos inicialmente planejados. Tal fato é favorável à acidentes no sentido de permitir a possibilidade de colisões nos pontos de cruzamento, já que os mesmos estarão em constante movimento.

O problema de deslocamento em circuito fechado foi modelado como um problema periódico com tempos de ciclos comensuráveis. Se para o problema em questão for considerado que haja ciclos de tempos incomensuráveis, torna-se impossível garantir a segurança no que diz respeito à possíveis colisões, conforme será mostrado nos capítulos 2 e 4.

A solução proposta é centralizada, na qual uma unidade de processamento determinará para cada veículo, o tempo de início e o perfil de velocidade ao longo do caminho, a fim de que seja possível garantir seguramente que dois robôs não estarão ao mesmo tempo no mesmo lugar. Haverá então, uma velocidade média associada a cada trecho entre os pontos de colisão, de forma que essas velocidades permaneçam as mesmas em todos os ciclos de trabalho. Logo, essa abordagem permite desconsiderar os limites sobre a aceleração e velocidades instantâneas. Para a definição do perfil de velocidade a ser calculado, são consideradas as restrições de velocidades máxima e mínima que podem ser desenvolvidas pelos robôs.

Dados os intervalos de tempos mínimo e máximo permitidos para os subcaminhos (determinados pelos trechos formados entre dois pontos de colisão), o problema a ser solucionado pelo controlador de alto nível é encontrar para cada robô o tempo de travessia destes subcaminhos tal que a margem de segurança seja maximizada. Uma vantagem da solução proposta é que torna-se possível desenvolver um modelo para problemas de horizontes infinitos com número finito de variáveis e restrições.

Em resumo, o trabalho aqui apresentado tem como principais objetivos:

- Fazer com que os robôs sigam pelos caminhos inicialmente planejados de forma a evitar colisão entre si.
- Maximizar o intervalo de tempo que os robôs cruzam por um mesmo ponto de colisão a fim de aumentar a margem de segurança.
- Implementar a metodologia em bases robóticas *Pioneer P3DX* [13] utilizando o *framework* ROS para se comunicar com os robôs e realizar a sua localização no espaço de trabalho.

- Utilizar o *solver LINGO* integrado com o ROS e a uma rotina em C++ para a navegação dos robôs.

1.3 REVISÃO BIBLIOGRÁFICA

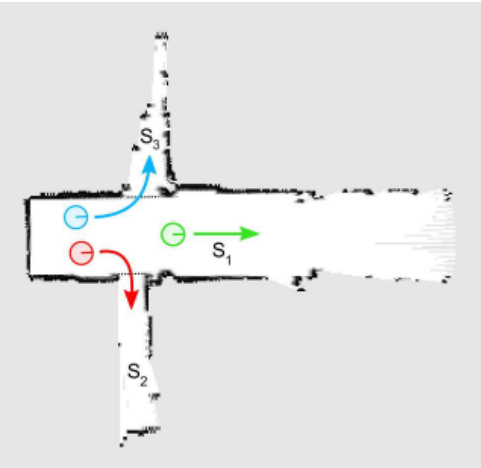
Coordenar o movimento de múltiplos robôs é uma importante tarefa, pois abre a possibilidade da implementação em diversas aplicações como monitoramento, vigilância, detecção e coleta de dados [6], [7], [10] e [14].

Neste contexto, se destacam duas grandes áreas de pesquisa na área de robótica - uma atua na coordenação de múltiplos robôs e a outra área foca na otimização de trajetória. Desta forma, diversas pesquisas tem sido realizadas com o intuito de realizar tais tarefas, porém com diferentes estratégias.

O artigo elaborado em [15] aborda o uso de múltiplos robôs na tarefa de explorar ambientes desconhecidos. O trabalho se baseia na distribuição individual dos robôs no ambiente, atribuindo para cada um deles a localização de alvos. Para isso leva-se em conta a estrutura do ambiente de forma a minimizar o tempo de conclusão da missão.

Para alcançar tal objetivo o ambiente é particionado em segmentos, que pode ser por exemplo, uma sala específica. Os autores não consideram somente o limite entre uma área conhecida e outra inexplorada para definir a localização dos alvos, mas também um determinado segmento que deve ser explorado, como mostrado na Figura 1.

Figura 1 – Robôs Explorando o Ambiente



Fonte: [15]

A Figura 2 a) mostra os robôs *Pioneer P3DX* explorando os laboratórios da Universidade de Freiburg na Alemanha. A Figura 2 b) mostra o resultado do mapeamento dos ambientes.

Figura 2 – Robôs explorando a Universidade de Freiburg

(a) Movimentação dos Pioneers:

(b) Mapa extraído através da Exploração de dois Robôs:

Fonte: [15]

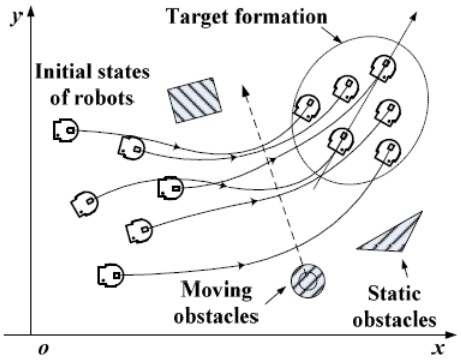
Portanto, o artigo citado acima não poderia ser usado para aplicação abordada neste trabalho, uma vez que pretende-se utilizar os robôs em caminhos conhecidos, ou seja, que são previamente especificados. Além disso, o artigo destacado é apropriado para a exploração e não para o monitoramento de ambientes, uma vez que para este último caso geralmente é exigido que os robôs percorram periodicamente diversos caminhos.

Em [16] é apresentado uma metodologia para o cálculo de caminhos percorridos por múltiplos robôs operando no mesmo espaço de trabalho de forma a evitar colisões. Ele se baseia no algoritmo $A*$, onde são calculados caminhos livres de obstáculos. Os autores propõem uma modificação no algoritmo tradicional, incluindo a capacidade de realizar replanejamento em ambientes dinâmicos sem a necessidade de começar a iteração do zero.

Da mesma forma que no artigo anterior, a abordagem utilizada em [16] não é a mais indicada para a aplicação em questão, uma vez que os robôs não podem se desviar dos caminhos pré-especificados.

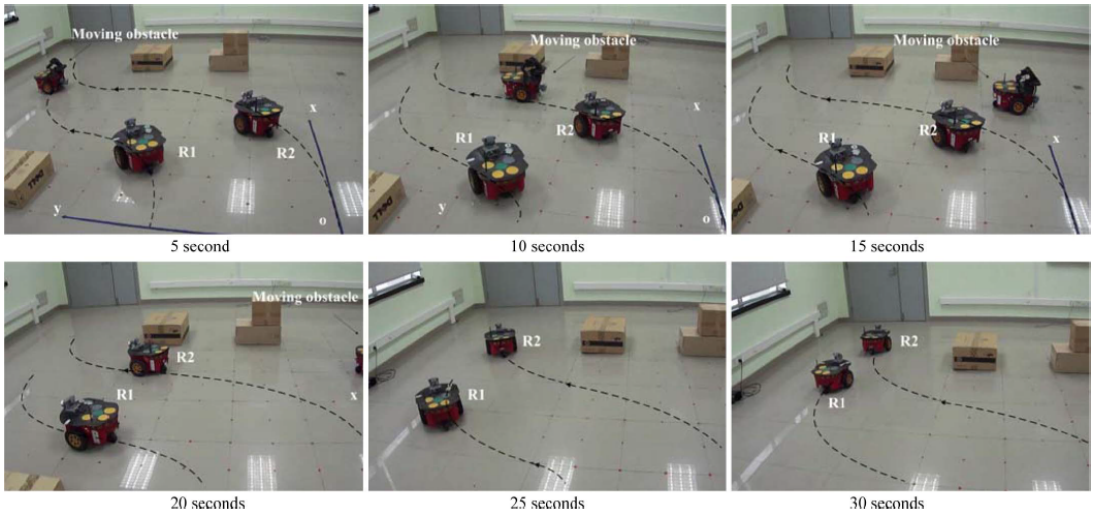
Os trabalhos apresentados em [17] e [18] também abordam a coordenação de múltiplos robôs percorrendo caminhos pré-difinidos. No entanto, se difere em vários aspectos do artigo em [10]. Primeiramente, em [17] os robôs precisam manter uma determinada formação, como mostrado nas Figuras 3 e 4. Segundo, ele apresenta restrições sobre velocidade e aceleração instantânea. E por último, ele gera perfis de velocidade para a coordenação de movimento dos robôs de modo que eles consigam manter a formação e evitar colisões entre si ou contra obstáculos dinâmicos, diferentemente da metodologia em [10] que leva em consideração diversos caminhos num mesmo espaço de trabalho percorridos pelos robôs de forma cíclica.

Figura 3 – Robôs Percorrendo Caminhos em Formação



Fonte: [17]

Figura 4 – Robôs Percorrendo Caminhos em Formação - Experimento Prático



Fonte: [18]

Para alcançar uma boa coordenação entre os robôs é de extrema importância controlá-los de forma que eles possam rastrear posição e velocidade desejadas. Para isso, os trabalhos [19], [20] e [21] focam no controle da trajetória. Em [20] é proposto uma lei de controle para rastreamento de trajetória e um método para seleção de ganhos para efetuar tal controle, respeitando as restrições dinâmicas do robô.

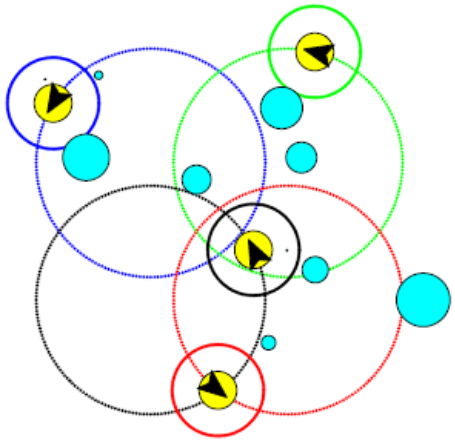
O trabalho apresentado em [21] propõe outra metodologia de controle para o rastreamento da trajetória nomeada de *"Lyapunov-based Guidance Control (LGC)"*. O controlador é projetado para guiar o robô na direção correta a todo instante, de forma que o sistema possa realizar uma predição da trajetória futura e a partir disso direcionar o robô para a trajetória desejada o mais rapidamente possível.

Em [22] é realizada a coordenação de múltiplos robôs percorrendo caminhos cíclicos. Os autores acreditam que o trabalho possa ser aplicado em tarefas de monitoramento

e limpeza, no qual o ambiente pode sofrer mudanças, além do fato que os caminhos percorridos pelos robôs podem se cruzar.

Desta forma, foi desenvolvido um algoritmo capaz de evitar colisões baseado em uma política de paradas. Simultaneamente, são realizadas análises de forma a quantificar o impacto destes tempos de paradas na estabilidade dos controladores de velocidade. A Figura 5 ilustra tal aplicação, onde quatro robôs representados pelo círculo amarelo percorrem seus respectivos caminhos, identificados pelos maiores círculos. Os círculos em volta dos robôs, chamados de *footprints* representam a região varrida pelos robôs. Os círculos coloridos em azul claro representam pontos de interesse, que podem ser entendidos como a discretização de um ambiente contínuo.

Figura 5 – Espaço de Trabalho - Robôs Executando a Tarefa



Fonte: [22]

No presente trabalho pressupõe-se que não seja possível a interrupção do movimento dos robôs, devido à intenção de aplicar a metodologia também em VAANT de asa fixa. Sendo assim, a solução proposta no artigo [22] não poderia ser aplicada uma vez que ocasionaria o *stall*.

O trabalho apresentado em [23] também tem como objetivo coordenar a trajetória de múltiplos robôs em caminhos que possuem pontos de interseção, de modo a evitar possíveis colisões. No artigo são identificadas condições necessárias e suficientes para que a tarefa esteja livre de colisões, o que é feito por meio da alteração dos tempos de início da trajetória de cada robô. Tal estratégia define o problema de otimização, que foi modelado como um problema de programação inteira mista com objetivo de encontrar soluções de tempo mínimo.

Segundo os autores, o método pode ser aplicado tanto em robôs móveis quanto em braços robóticos, em que não são restringidos os graus de liberdades dos robôs. Acredita-se também que a metodologia possa ser aplicada na indústria automotiva, nos processos de pintura e soldagem.

A metodologia desenvolvida em [10] e utilizada neste trabalho permite definir o perfil de velocidade média dos robôs ao longo do caminho e considerar apenas os pontos de colisão, não se baseando em modificar os tempos de início de cada robô. Portanto, o modelo de otimização encontrará o perfil de velocidade média que os robôs precisarão desenvolver em cada trecho do circuito para maximizar o menor intervalo de tempo entre a passagem dos robôs pelo mesmo ponto de colisão.

A solução proposta em [24] apresenta restrições sobre a cinemática e dinâmica dos robôs. Sua abordagem propõe um perfil de velocidade contínua, de forma a evitar colisões entre eles que percorrem ponto a ponto caminhos pré-determinados. Outro objetivo é minimizar o tempo de conclusão da tarefa, atuando na velocidade e aceleração dos robôs.

A metodologia que combina técnicas de controle ótimo e programação matemática, pretende identificar os segmentos que possuem e os que não possuem pontos de colisão, de modo a otimizar a velocidades sobre estes segmentos. Logo, os tempos que os robôs percorrem estes trechos (formados por pares de pontos) são calculados levando em consideração as restrições dinâmicas dos robôs.

Portanto abordagem descrita acima possui algumas características diferentes da apresentada em [10], que permite desconsiderar limites de velocidade e aceleração instantâneas, restringindo somente a velocidade média, além do fato de que em [10] os caminhos são periódicos e é proposto um planejamento de horizonte infinito com um número finito de variáveis e restrições, sendo segundo os autores, a principal contribuição do artigo.

1.4 ORGANIZAÇÃO DO TRABALHO

A dissertação está organizada da seguinte maneira:

- **Capítulo 2:** será mostrado como foi modelado o problema de coordenação de múltiplos robôs se movimentando caminhos cíclicos pré-definidos que possuem pontos de interseção e a solução utilizada. Será detalhado todo o problema explicando as equações envolvidas. Serão feitas também simulações para demonstrar a utilização da metodologia. Será apresentado o *solver* de otimização *LINGO* utilizado, suas características e vantagens.

- **Capítulo 3:** abordará a metodologia utilizada para a implementação prática do experimento, mostrando as ferramentas utilizadas, como ROS usado para a navegação e localização dos robôs. Serão dedicadas seções para apresentar os pacotes utilizados, conceitos do ROS, relação entre sistemas de coordenadas e malha de controle utilizada.

- **Capítulo 4:** irá mostrar os resultados obtidos através de simulações e de um experimento prático a fim de demonstrar a metodologia descrita no capítulo 2 e sua implementação prática. Serão feitas comparações entre os resultados de forma a destacar e entender as principais diferenças entre os dois ambientes.

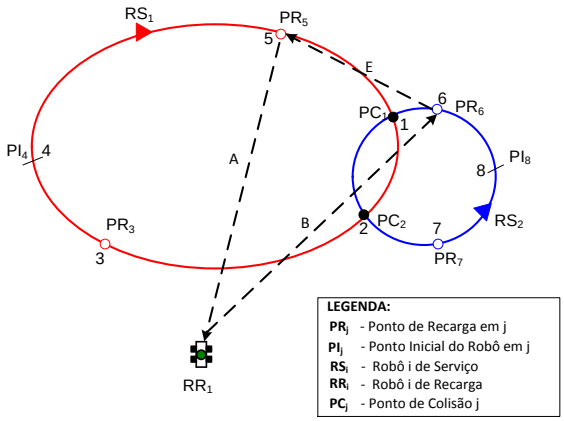
- **Capítulo 5:** serão apresentadas as conclusões obtidas através da análise dos resultados de simulações e experimento realizados para a coordenação dos robôs, suas contribuições e propostas para trabalhos futuros.

1.5 PUBLICAÇÕES DECORRENTES DO DESENVOLVIMENTO DESTE TRABALHO

Em consequência dos estudos realizados para o desenvolvimento deste trabalho, publicou-se o artigo intitulado "Coordenação Ótima de Múltiplos Robôs de Serviço e de Recarga em Tarefas Persistentes" [25], na 11° edição do *IEEE/IAS International Conference on Industry Applications - INDUSCON 2014*. Este trabalho trata do problema de coordenação de dois tipos robôs - de serviço e de recarga. Os de serviço são os robôs que executam tarefas como monitoramento, vigilância, etc e os de recarga, como o próprio nome já diz, foi introduzido para realizar o carregamento das baterias dos robôs de serviço.

Para coordenar esses dois grupos de robôs utilizou-se a estratégia que conta com dois otimizadores: o primeiro otimizador utiliza a mesma metodologia abordada nesta dissertação, atuando sobre os robôs de serviço a fim de planejar o perfil de velocidade média que eles terão ao longo de todo o caminho e desta forma evitar possíveis colisões [10]. Já o segundo, tem por objetivo encontrar a melhor rota para que aconteça o recarregamento das baterias. Desta forma, o otimizador um alimenta o segundo com informações dos tempos gastos para os robôs de serviço percorrerem cada subcaminho, permitindo que seja calculada a rota que minimiza o tempo global da tarefa de recarga.

Figura 6 – Coordenação dos Robôs de Serviço e de Recarga



Fonte: [25]

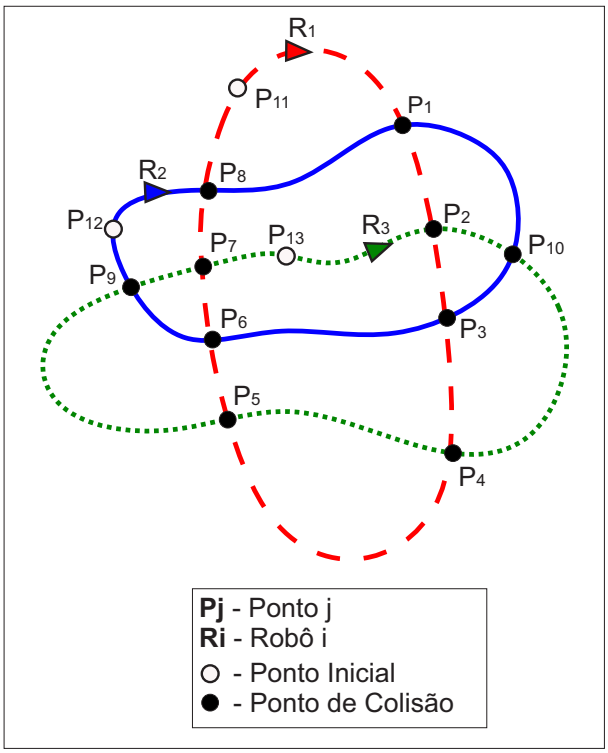
Em relação à implementação prática, no qual foram desenvolvidas rotinas em C++ para a navegação dos robôs utilizando o ROS, foi possível ministrar 3 mini-cursos em diferentes congressos que abordaram a utilização desta ferramenta:

- XI Simpósio Brasileiro de Automação Inteligente - SBAI 2013: Introdução ao ROS
- 20° Congresso Brasileiro de Automática - CBA 2014: Programação de Robôs Utilizando o ROS
- 11° *IEEE/IAS International Conference on Industry Applications - INDUSCON 2014: Robotics Applications: A Comparison between frameworks ROS and HTTPThru*

2 MODELAGEM DO PROBLEMA DE OTIMIZAÇÃO

Considere um ambiente formado por diversos caminhos cíclicos e que em cada um deles há um robô se movimentando de forma ininterrupta. Neste espaço de trabalho os caminhos se cruzam em diversos lugares, formando os chamados pontos de colisão (Figura 7). Assume-se que além de estarem sempre em movimento, os robôs não podem sair do caminho inicialmente traçado.

Figura 7 – Espaço de Trabalho



Fonte: Elaborada pelo autor

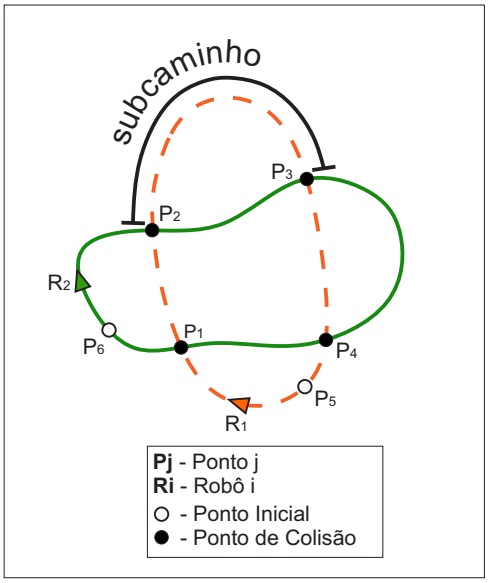
O objetivo principal do presente trabalho é evitar que haja colisão entre os robôs durante uma determinada missão. Para isso, foi adotada a metodologia desenvolvida em [10] e utilizada em [25] que propõe um controlador central de alto nível com estrutura discreta para planejar o perfil de velocidade média dos robôs ao longo de todo o percurso, de forma a garantir que eles nunca estejam no mesmo lugar em um determinado instante de tempo.

Portanto, o robô terá uma velocidade média associada em cada trecho formado entre os pontos, denominado de subcaminho (Figura 8). É importante ressaltar que essas velocidades permanecerão as mesmas em todos os ciclos de trabalho.

Dado o intervalo de tempo mínimo e máximo permitido (que são funções das velocidades máximas e mínimas e da geometria da curva) entre os subcaminhos, o controlador de alto nível definirá para cada robô o tempo que eles levarão para percorrer cada um dos trechos do circuito, de forma que o intervalo de tempo que robôs cruzam por um mesmo

ponto de colisão seja maximizado, garantindo com margem de segurança que não aconteça colisões.

Figura 8 – Trechos do Circuito



Fonte: Elaborada pelo autor

Um ponto forte da metodologia utilizada para o processo de otimização é que ela possibilita abstrair a geometria das curvas presentes no caminhos dos robôs e considerar apenas os pontos onde há interseções, ou seja, perigo de colisões. É necessário somente mapear os pontos subsequentes do caminho percorrido para cada robô [10]. Para facilitar a compreensão, foi feito o mapeamento dos pontos mostrados na Figura 7.

- Robô 1: $P_{11} \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_6 \rightarrow P_7 \rightarrow P_8 \rightarrow P_{11}$
- Robô 2: $P_{12} \rightarrow P_8 \rightarrow P_1 \rightarrow P_{10} \rightarrow P_3 \rightarrow P_6 \rightarrow P_9 \rightarrow P_{12}$
- Robô 3: $P_{13} \rightarrow P_2 \rightarrow P_{10} \rightarrow P_4 \rightarrow P_5 \rightarrow P_9 \rightarrow P_7 \rightarrow P_{13}$

De posse dessas informações e considerando os limites de tempo máximo e mínimo que os robôs conseguem percorrer cada parte do circuito, o controlador central está apto a encontrar para todos os robôs o tempo de percurso em cada subcamimho de forma que o intervalo de tempo Δt que os robôs passam por um determinado ponto de colisão seja maximizado [10].

2.1 O PROBLEMA DE OTIMIZAÇÃO

A modelagem do problema de otimização foi desenvolvida utilizando a metodologia de Programação Linear Inteira Mista (PLIM), pois como será abordado a seguir, as variáveis de decisão são lineares, tendo em vista que há variáveis reais e inteiras [10].

Inicialmente, são criados os seguintes conjuntos:

- $\mathcal{R}$ - Conjunto de Robôs $\{\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_n\}$
- $\mathcal{C}_i$ - Caminho percorrido pelo robô $i \in \mathcal{R}$
- $\mathcal{J}$ - Conjunto dos pontos de colisão j.
- $\mathcal{S}_j$ - Conjunto de robôs i que podem colidir no ponto j
- $\mathcal{Y}_i$ - Conjunto de Pontos de Colisão presentes no caminho do robô i.

Definidos os conjuntos, segue uma descrição das funções utilizadas.

- $s(j,i)$: $\mathcal{J} \times \mathcal{R} \mapsto \mathcal{J}$ - Função que mapeia o ponto atual para o próximo ponto na sequência do robô i. Exemplo: Considerando a sequência de pontos do robô 1 - $P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_1$ e que ele esteja no P_6, a função $s(6,1)$ mapeia para o ponto P_7;
- $a(j,i)$:$\mathcal{J} \times \mathcal{R} \mapsto \mathcal{J}$ - Função que mapeia o ponto atual para o ponto anterior na sequência do robô i. Análogo ao exemplo anterior. Assim, $a(6,1)$ mapeia para P_5.
- l: $\mathcal{R} \mapsto \mathcal{J}$ - Função que mapeia o robô para o seu ponto inicial. Assim, $l(1) = P_1$
- $L(j,i)$: $\mathcal{J} \times \mathcal{R} \mapsto \mathbb{R}$ - Função que mapeia o ponto, presente no conjunto $\mathcal{Y}_i$, para a distância do caminho entre o ponto inicial e o ponto em questão. Aproveitando o conjunto de pontos do robô 1: Define-se que a distância de P_5 para o ponto inicial é de 15 metros. Logo, $L(5,1)$ retorna 15.

2.1.1 **Modelagem das Restrições**

O ambiente das missões estabelecido no presente trabalho descreve caminhos que possuem pontos de interseção. Uma vez que foi definido que os robôs não podem interromper o seu movimento (para evitar *stall* caso a metodologia fosse aplicada a veículos aéreos de asa fixa) ou mudar de direção (caso dos robôs terrestres inseridos em ambientes com grandes limitações de espaço), a solução encontrada atua na diferença de tempo que os robôs cruzam por um determinado ponto de colisão. Pretende-se que essa diferença de tempo Δt seja a maior possível a fim de aumentar a margem de segurança.

Portanto, necessita-se encontrar o tempo que o robô $i \in \mathcal{R}$ chega no ponto $j \in \mathcal{J}$ no k-ésimo ciclo, $t_j[i,k]$, tal que:

$$\begin{aligned} &|t_j\left[i_1,k_1\right] - t_j\left[i_2,k_2\right]| \geq \Delta t \\ &\forall j \in \mathcal{J}, \forall i_1, i_2 \in \mathcal{S}_j, \forall k_1, k_2 \end{aligned} \tag{2.1}$$

É importante ressaltar que o resultado do problema de otimização apontará qual dos robôs cruzará primeiro pelo ponto. Assim, se o robô i_2 passar primeiro pelo ponto de colisão, a parcela será positiva. Caso contrário será negativa.

O tempo que os robôs levam para percorrer os subcaminhos também é restringido conforme mostrado nas equações (2.2) e (2.3), levando em consideração as velocidades máxima e mínima que eles podem atingir.

$$\tau_{min}[i,j] = \frac{L(s(j,i),i) - L(j,i)}{v^i_{max}} \tag{2.2}$$

$$\tau_{max}[i,j] = \frac{L(s(j,i),i) - L(j,i)}{v^i_{min}} \tag{2.3}$$

Onde $L(s(j,i),i)$ fornece o comprimento do caminho entre o ponto seguinte ao atual e o ponto inicial.

Assim, o tempo para percorrer um determinado subcaminho deve ser maior ou igual a τ_{min} e menor ou igual a τ_{max}:

$$\begin{gathered}\tau_{min}[i,j] \leq t_{s(i,j)}[i,k] - t_j[i,k] \leq \tau_{max}[i,j] \\ \forall i \in \mathcal{R}, \forall j \in \mathcal{Y}_i, \forall k\end{gathered} \tag{2.4}$$

No entanto, a inequação (2.4) não abrange o último trecho do circuito, o subcaminho formado pelo primeiro e último ponto. Logo,

$$t_{l(i)}[i,k+1] - t_{a(l(i),i)}[i,k] \leq \tau_{max}[i,a(l(i),i)] \tag{2.5}$$

$$\tau_{min}[i,a(l(i),i)] \leq t_{l(i)}[i,k+1] - t_{a(l(i),i)}[i,k] \tag{2.6}$$

$$\forall i \in \mathcal{R}, \forall k$$

Sendo que,

- $t_{l(i)}[i,k+1]$ - Tempo que o robô i chegou ao ponto inicial no ciclo $k+1$. Ou seja, o tempo completo da volta no ciclo k.
- $t_{a(l(i),i)}[i,k]$ - Tempo gasto pelo robô i para chegar no ponto anterior ao ponto inicial.

Como foi especificado anteriormente, o planejamento do perfil de velocidades médias ao longo do caminho possui horizonte infinito com um número finito de variáveis e

restrições; Logo, o robô repete a sua trajetória em cada ciclo de trabalho. Desta forma, tem-se:

$$\begin{gathered} t_j[i,k_1] - t_j[i,k_1-1] = t_j[i,k_2] - t_j[i,k_2-1] \\ \forall i \in \mathcal{R}, \forall j \in \mathcal{Y}_i, \forall k_1, k_2 \end{gathered} \tag{2.7}$$

Define-se T_i como tempo de ciclo da trajetória do robô i. Logo,

$$\begin{gathered} t_j[i,k] = t_j[i,0] + kT_i \\ \forall i \in \mathcal{R}, \forall j \in \mathcal{Y}_i, \forall k \end{gathered} \tag{2.8}$$

Uma das vantagens da metodologia utilizada é que ela permite que cada um dos robôs possa estar num determinado ciclo de trabalho independente do outro robô. Tal fato amplia a gama de casos em que esta metodologia pode ser usada, uma vez que há a possibilidade dos caminhos possuírem tamanhos distintos.

Para garantir que não aconteçam colisões as trajetórias precisam ser periódicas e os tempos de ciclo T_i comensuráveis. Dois números a e b são ditos comensuráveis se a razão entre eles é um número racional m/n tal que $m \in \mathbb{Z}$ e $n \in \mathbb{Z}^*$ [26]. Assim,

$$\frac{a}{b} = \frac{m}{n} \in \mathbb{Q} \tag{2.9}$$

Obtém-se o tempo de conclusão de um ciclo T_i para o robô i como segue:

$$T_i = \gamma_i T \tag{2.10}$$

onde T representa uma base de tempo e é comum para todos os robôs.

Apoiando-se no conceito de comensurabilidade e pelo fato que são considerados apenas tempos positivos, definiu-se γ_i como sendo um número natural sabendo pela teoria de conjuntos que o conjunto dos números naturais $\mathbb{N}$ está contido no conjunto dos números inteiros $\mathbb{Z}$.

Logo, se T_1 e T_2 representam o tempo de ciclo para os robôs 1 e 2 respectivamente, tem-se que:

$$\frac{T_1}{T_2} = \frac{m}{n} \in \mathbb{Q} \tag{2.11}$$

Portanto, baseado no exposto acima, pode-se dizer que T_1 e T_2 são grandezas comensuráveis.

Tal definição é necessária, pois se os tempos de ciclo forem diferentes e incomensuráveis, fica impossível estabelecer um planejamento para horizonte infinito, uma vez que não será possível aplicar um mesmo perfil de velocidade média a todos os ciclos, já que eles não se repetem.

Levando em consideração a equação (2.10) e sabendo que o planejamento se repetirá para todos os ciclos, pode-se definir a diferença de tempo que os robôs atingem um determinado ponto de colisão como sendo,

$$\begin{gathered} t_j[i_1,k_1] - t_j[i_2,k_2] = t_j[i_1,0] - t_j[i_2,0] + (\gamma_{i1}k_1 + \gamma_{i2}k_2)T \\ \forall j \in \mathcal{J}, \forall i_1, i_2 \in \mathcal{S}_j, \forall k_1, k_2 \end{gathered} \tag{2.12}$$

já que basta subtrair os tempos no primeiro ciclo e somar os tempos de ciclo restantes.

Diante do exposto acima, pode-se definir um modelo com número limitado de restrições, uma vez que necessita-se levar em consideração somente os dois primeiros ciclos da missão, no qual $k = 1$ é utilizado para fazer a mudança do primeiro para o segundo ciclo. Logo, tem-se que:

$$\begin{gathered} |t_j[i_1,0] - t_j[i_2,0]| \geq \Delta t \\ \forall j \in \mathcal{J}, \forall i_1, i_2 \in \mathcal{S}_j \end{gathered} \tag{2.13}$$

Assim como na inequação (2.1), a inequação (2.13) também apresenta o módulo. Dessa forma,

$$t_j[i_1,0] - t_j[i_2,0] \geq \Delta t \tag{2.14}$$

$$t_j[i_1,0] - t_j[i_2,0] \leq -\Delta t \tag{2.15}$$

$$\forall j \in \mathcal{J}, \forall i_1, i_2 \in \mathcal{S}_j$$

Uma vez que não se sabe qual dos robôs atingirá primeiro um determinado ponto de colisão j, não se pode determinar qual das inequações (2.14) ou (2.15) o modelo de otimização irá utilizar, gerando uma inconsistência no processo, pois elas são mutuamente exclusivas.

Para resolver esse problema, é necessário adotar uma estratégia que atende simultaneamente as duas inequações citadas acima. Portanto, através de um artifício matemático insere-se uma variável ξ, chamada de estimador a priori, e uma variável binária $B \in [0,1]$ que definirá qual das inequações citadas estará ativa.

$$t_j[i_1, 0] - t_j[i_2, 0] \geq \Delta t - \xi[i_1, i_2, j]B[i_1, i_2, j] \tag{2.16}$$
$$t_j[i_1, 0] - t_j[i_2, 0] \leq -\Delta t + \xi[i_1, i_2, j](1 - B[i_1, i_2, j]) \tag{2.17}$$
$$\forall j \in \mathcal{J}, \forall i_1, i_2 \in \mathcal{S}_j, B[i_1, i_2, j] \in 0, 1$$

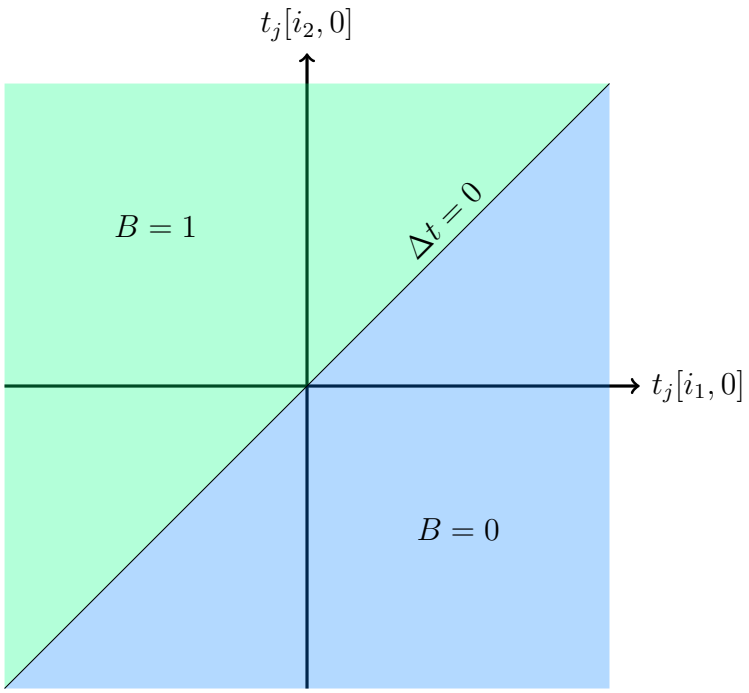
Vale ressaltar que o valor da variável ξ pode ser adotado como sendo a velocidade máxima do robô ou simplesmente um valor muito alto.

Observa-se pelas inequações (2.16) e (2.17) que se:

- $B = 0$: Logo, $\Delta t \leq t_j[i_1, 0] - t_j[i_2, 0] \leq -\Delta t + \xi[i_1, i_2, j]$
- $B = 1 : \Delta t - \xi[i_1, i_2, j] \leq t_j[i_1, 0] - t_j[i_2, 0] \leq -\Delta t$

A Figura 9 mostra a região de solução que o otimizador pode encontrar para as inequações 2.16 e 2.17. Caso $B = 0$, $t_j[i_1, 0]$ e $t_j[i_2, 0]$ podem assumir quaisquer valores dentro da região azul, lembrando que Δt e $\xi[i_1, i_2, j]$ delimitam a área do gráfico. Logo, quanto menor for $\xi[i_1, i_2, j]$ e maior for Δt, menor será a área. O mesmo pode ser dito para $B = 1$, porém a área em questão seria a verde.

Figura 9 – Região de Solução



Fonte: Elaborada pelo autor

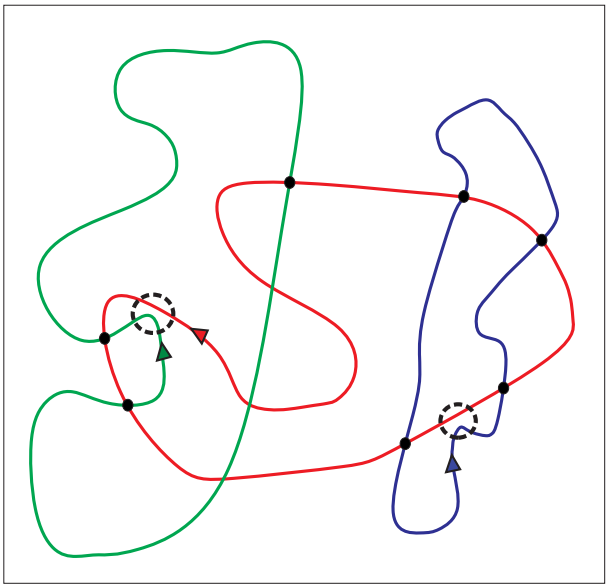
Como condição inicial do problema tem-se de acordo com a inequação (2.18) que:

$$|t_j[i_1, 0] - t_j[i_2, 0]| \leq T - \Delta t$$
$$\forall j \in \mathcal{J}, \forall i_1, i_2 \in \mathcal{S}_j \tag{2.18}$$

Foi considerado durante a modelagem do problema que quanto maior for o intervalo de tempo que dois robôs passam por um mesmo ponto de colisão, menor seria a chance de acontecer algum acidente. No entanto, em algumas situações essa consideração não nos leva de fato a garantir ausência de colisões. Isso se deve ao fato da velocidade exercer um papel importante no quesito segurança, pois como a distância de um robô para o outro é calculada como sendo $v\Delta t$ percebe-se que se a velocidade estiver muito baixa a distância pode se tornar muito pequena, o que poderia causar algum tipo de acidente. Logo, isso deve ser levado em conta durante a realização de testes experimentais.

Outro problema está associado à geometria dos caminhos em questão. Se em alguma parte destes circuitos a distância entre eles não for suficiente para passar dois robôs com segurança, poderá haver colisão. A Figura 10 ilustra essa situação. Nota-se que nas regiões destacadas pelo círculo pontilhado existe a possibilidade de ocorrer algum acidente. Logo, pode-se contornar esse problema, inserindo um ponto de colisão artificial no planejamento da missão. Desta forma, o intervalo de tempo gerado pelo o movimento dos robôs sob aquela região começará a ser considerado.

Figura 10 – Caminhos muito próximos um do outro.



Fonte: Elaborada pelo autor

2.1.1.1 Resumo das Restrições

Uma vez modelado o problema de otimização, o objetivo é encontrar um perfil de velocidade média dos robôs para cada subcaminho do circuito, de forma a maximizar o menor intervalo de tempo Δt, para que esse planejamento possa ser seguido em todos os ciclos de trabalho que constituem a missão. Sendo assim, o problema de otimização está sujeito as seguintes restrições:

- Restrição de Velocidade para pontos no mesmo ciclo:

$$\begin{gathered}\tau_{min}[i,j] \leq t_{s(i,j)}[i,0] - t_j[i,0] \leq \tau_{max}[i,j] \\ \forall i \in \mathcal{R}, \forall j \in \mathcal{Y}_i\end{gathered} \tag{2.19}$$

- Restrição de Velocidade para o último ponto do circuito:

$$t_{l(i)}[i,1] - t_{a(l(i),i)}[i,0] \leq \tau_{max}[i, a(l(i),i)] \tag{2.20}$$

$$t_{l(i)}[i,1] - t_{a(l(i),i)}[i,1] \geq \tau_{min}[i, a(l(i),i)] \tag{2.21}$$

$$\forall i \in \mathcal{R}, \forall j \in \mathcal{Y}_i$$

- Periodicidade e Comensurabilidade de Ciclos:

$$\begin{gathered}t_{l(i)}[i,1] - t_{l(i)}[i,0] = \gamma_i T \\ \forall i \in \mathcal{S}_j\end{gathered} \tag{2.22}$$

- Limite de Margem de Tempo:

$$\begin{gathered}t_j[i_1,0] - t_j[i_2,0] \geq \Delta t - \xi[i_1,i_2,j]B[i_1,i_2,j] \\ t_j[i_1,0] - t_j[i_2,0] \leq -\Delta t + \xi[i_1,i_2,j](1 - B[i_1,i_2,j]) \\ \forall j \in \mathcal{J}, \forall i_1, i_2 \in \mathcal{S}_j, B[i_1,i_2,j] \in 0,1\end{gathered}$$

- Condição Inicial: Resolvendo o módulo da inequação (2.18)

$$\begin{gathered}-T + \Delta t \leq t_j[i_1,0] - t_j[i2,0] \leq T - \Delta t \\ \forall j \in \mathcal{J}, \forall i_1, i_2 \in \mathcal{S}_j\end{gathered} \tag{2.23}$$

  Lembrando que $\Delta t \geq 0$ e que o tempo inicial do robô 1 no primeiro ciclo $k = 0$ é zero ($t_{l(i)}[1,0] = 0$).

2.2 *SOFTWARE* DE OTIMIZAÇÃO

Nesta seção será abordado de forma sucinta o *software* utilizado para modelar e resolver o problema de otimização, mostrando suas características e algumas de suas funcionalidades [27].

O *LINGO* é uma ferramenta para modelar de maneira concisa problemas lineares e não-lineares de otimização, resolvê-los e analisar a sua solução, de forma a encontrar a resposta que produz o melhor resultado. Ele foi projetado para construir e resolver modelos de otimização matemáticos de forma mais fácil e rápida, fornecendo um pacote integrado com diversos *solvers* e uma linguagem para modelagem baseada em conjuntos, permitindo expressar modelos de uma maneira intuitiva, usando somatórios e variáveis subscritas. É possível também construir modelos que capturam ou exportam informações diretamente dos bancos de dados e planilhas.

Uma das principais características do *LINGO* e que motivou o seu uso neste trabalho, é a opção de chamar o *script* de modelagem feito no *LINGO* a partir de códigos elaborados em ambientes de desenvolvimento que utilizam linguagens em C/C++, FORTRAN, Java, VB.NET, ASP.NET, Visual Basic, Delphi e Excel. Logo, essa funcionalidade foi fundamental pois, como especificado no capítulo 1, foi elaborada uma aplicação em C++ que executa o *script* de modelagem do *LINGO*, captura o valor de algumas variáveis, como Δt e os tempos em que cada robô necessita para percorrer os trechos do circuito, permitindo utilizá-las no código que realiza a navegação do robô pelo ambiente de trabalho.

Não há necessidade de especificar ou carregar um *solver* separadamente, pois o *LINGO* realiza a leitura da sua formulação de forma automática e seleciona o *solver* apropriado. Ele conta com os seguintes *solvers* para otimização linear, não-linear, inteira, etc:

- *General NonLinear Solver*
- *Global Solver*
- *Multistart Solver*
- *Barrier Solver*
- *Simplex Solver*
- *Mixed Integer Solver*

2.2.1 Trabalhando com Conjuntos

Durante a modelagem de um problema, geralmente surgem um ou mais grupos de objetos. Como exemplo de objetos, pode-se citar, no caso desta dissertação, robôs, caminhos, pontos de colisão, etc. O *software* em questão permite agrupar esses objetos em conjuntos, que constituem o fundamento da linguagem de modelagem do *LINGO*. Com isso, é possível escrever uma série de restrições similares em uma única definição, expressando longas e complexas fórmulas de forma mais clara.

Conjuntos são definidos em uma seção da modelagem do problema chamada *"sets section"*, que começa com a palavra-chave *"SETS:"* e termina com *"ENDSETS"*. Um modelo pode ter nenhuma, uma ou várias *set sections* e elas podem ser definidas em qualquer lugar, desde que antes do uso de algum conjunto. Cada membro de um conjunto pode ter uma ou mais características associadas a ele, conhecidas como atributos.

O *LINGO* reconhece dois tipos de conjuntos: primitivos e derivados. Conjunto primitivo é aquele composto apenas por objetos que não podem ser reduzidos. Já um conjunto derivado é definido usando um ou mais conjuntos, ou seja, seus membros derivam de outros conjuntos.

2.2.1.1 *Conjuntos Primitivos*

Para definir conjuntos primitivos deve-se especificar o nome do conjunto e opcionalmente seus membros, que são objetos formadores do conjunto, e seus atributos seguindo a sintaxe da Figura 11:

Figura 11 – Definição de Conjuntos

```
SETS:
        nome_do_conjunto[ / lista_de_membros / ][: lista_de_atributos];
ENDSETS
```

Fonte: Elaborada pelo autor

O uso de colchetes indica que os elementos dentro dele são opcionais. O nome do conjunto deve sempre começar com letra, porém pode ser sucedido por números. Não é feito distinção entre letras maiúsculas ou minúsculas.

Lista de membros compõe um determinado conjunto. Se esses membros são incluídos na definição do conjunto, eles podem estar listados de forma explícita ou implícita, conforme mostrado na Figura 12.

Figura 12 – Forma de Expressar a lista de Membros

```
SETS:
        ROBOTS / 1 2 3 4 /
ENDSETS
```

(a) Forma Explícita:

```
DATA:
        NR_ROBOTS = 4;
ENDDATA

SETS:
        ROBOTS / 1..NR_ROBOTS /
ENDSETS
```

(b) Forma Implícita:

Fonte: Elaborada pelo autor

Caso os membros não sejam incluídos na definição do conjunto, eles podem ser definidos posteriormente em uma seção da modelagem chamada de *Data Section*. Esta seção permite o programador isolar dados do seu modelo da formulação do problema. Ela começa e termina com as palavaras-chave *"DATA:"* e *"ENDDATA"* respectivamente, seguindo a sintaxe da Figura 12(b). Vale ressaltar que as duas formas apresentadas na Figura 12 produzem o mesmo resultado, sendo que a forma implícita é bastante útil quando a lista de membros é grande, polpando tempo durante a programação.

2.2.1.2 *Conjuntos Derivados*

A definição de um conjunto derivado é realizada de forma semelhante ao conjunto primitivo. No entanto, conforme citado no início da Seção 2.2.1, deve-se incluir outros conjuntos, como mostrado na Figura 13.

Figura 13 – Definição de Conjunto Derivado

```
SETS:
        ROBOTS : VEL_MAX, VEL_MIN, PONTO_INICIAL, FIRST_COLISION,
                LAST_COLISION, LAMBDA, LENGHT;
        POINTS   : COLISAO;
        WAYS (ROBOTS,POINTS): PROXIMO, DISTANCE;
        COLISION_POINTS (ROBOTS, ROBOTS, POINTS) | &2 #GT# &1: COLIDE,
                binario ;
ENDSETS

DATA:
        ! SET MEMBERS;
        ROBOTS = R1 R2;
        POINTS = P1 P2 P3 P4;
ENDDATA
```

Fonte: Elaborada pelo autor

Observa-se na figura 13 que o conjunto *"WAYS"* deriva de dois outros conjuntos primitivos: *"ROBOTS"* e *"POINTS"*. Note que a lista de membros não foi especificada, porém o *LINGO* constrói todas as combinações possíveis entre os conjuntos envolvidos, produzindo a lista mostrada na Tabela 1 para o conjunto *"WAYS"*, onde "R1"e "R2"são membros do conjunto "ROBOTS"e "P1","P2","P3","P4"são do conjunto "POINTS".

Tabela 1 – Lista de Membros

Índice	Membros
1	(R1,P1)
2	(R1,P2)
3	(R1,P3)
4	(R1,P4)
5	(R2,P1)
6	(R2,P2)
7	(R2,P3)
8	(R2,P4)

Fonte: Elaborada pelo autor

Outra forma interessante de gerar lista de membros é através de um filtro lógico, como o que foi usado para definir o conjunto *"COLISION_POINTS"*. Ele permite selecionar somente as combinações que satisfazem a condição exigida pelo filtro. A Tabela 2 mostra o resultado do uso do filtro e como ficaria a lista de membros sem a condição.

Tabela 2 – Lista de Membros *COLISION_POINTS*

Índice	Membros
1	R1,R1,P1
2	R1,R1,P2
3	R1,R1,P3
4	R1,R1,P4
5	R1,R2,P1
6	R1,R2,P2
7	R1,R2,P3
8	R1,R2,P4
9	R2,R1,P1
10	R2,R1,P2
11	R2,R1,P3
12	R2,R1,P4
13	R2,R2,P1
14	R2,R2,P2
15	R2,R2,P3
16	R2,R2,P4

(a) Sem Filtro:

Índice	Membros
1	R1,R2,P1
2	R1,R2,P2
3	R1,R2,P3
4	R1,R2,P4

(b) Com Filtro:

Fonte: Elaborada pelo autor

2.2.2 Funções de *Looping*

Uma das principais vantagens do *LINGO* é permitir que uma determinada operação seja realizada para todos os membros de um conjunto através de uma única definição. Existe uma série de funções que torna possível essa funcionalidade, sendo abordado nesta seção as principais. A Tabela 3 ilustra tais funções:

Tabela 3 – Funções Comumente Usadas

Índice	Membros
@FOR	Estrutura de Repetição capaz de percorrer variáveis. Usada para realizar cálculos e gerar restrições sobre os membros de um conjunto.
@SUM	Calcula o somatório de uma expressão
@PROD	Calcula o produtório de uma expressão
@MIN	Calcula o mínimo de uma expressão
@MAX	Calcula o máximo de uma expressão

Fonte: Elaborada pelo autor

Para utilizar as funções descritas acima, deve-se seguir a sintaxe mostrada na Figura 14 abaixo,

Figura 14 – Sintaxe das Funções

```
@FUNCTION (nome_do_conjunto [ ( lista_indices ) [ | condicao ] ]  : lista_expressoes) ;
```

Fonte: Elaborada pelo autor

Note que *"FUNCTION"* deve ser substituída por alguma das funções da Tabela 3. O campo "lista_indices"é usado para criar um lista de índices, em que cada um deles corresponde a um membro do conjunto "nome_do_conjunto".

O item "condicao"é utilizado para limitar o acesso ao campo "lista_expressoes", poupando processamento computacional. Já "lista_expressoes"como o próprio nome sugere, é uma lista de expressões que são aplicadas em cada membro do conjunto em questão. Se estiver sendo utilizada a função *"FOR"*, essa lista deve conter uma série de expressões separadas por ponto e vírgula. Essas expressões serão adicionadas como restrições do modelo. Quando é usada qualquer uma das funções da Tabela 3 com exceção de *"FOR"*, o item "lista_expressoes"deve conter uma única expressão. Para facilitar o entendimento a respeito dessas funções, foi criado um simples exemplo mostrado na Figura 15.

Figura 15 – Exemplo de uso das Funções

```
MODEL:
        SETS:
                ROBOTS : TRAVELED_DISTANCE,VEL;
        ENDSETS

        DATA:
                ! SET MEMBERS;
                ROBOTS = R1 R2 R3 R4;
                TRAVELED_DISTANCE = 10 20 15 30;
        ENDDATA

        MIN_DISTANCE = @MIN ( ROBOTS (J) : TRAVELED_DISTANCE (J) );
        MAX_DISTANCE = @MAX ( ROBOTS (J) : TRAVELED_DISTANCE (J) );
        MAX_DISTANCE2 = @MAX ( ROBOTS (J) | J #LE# 3  :
                TRAVELED_DISTANCE (J) );
        @FOR (ROBOTS (R) : VEL (R) <= 50);
END
```

Fonte: Elaborada pelo autor

Nota-se que o conjunto *"ROBOTS"* possui 4 membros e os atributos *"TRAVELED_DISTANCE"* informando a distância percorrida por cada robô e *"VEL"* que indica a velocidade dos robôs . Após o *LINGO* resolver este modelo, tem-se a seguinte saída para as variáveis e restrições mostrada na Tabela 4, destacando que o uso da função *"FOR"* permitiu que fossem geradas automaticamente as restrições para cada robô.

Tabela 4 – Saída do Lingo

VARIÁVEL	VALOR
MIN_DISTANCE	10
MAX_DISTANCE	30
MAX_DISTANCE2	20

(a) Valor das Variáveis:

RESTRIÇÕES
VEL(R1) <= 50
VEL(R2) <= 50
VEL(R3) <= 50
VEL(R4) <= 50

(b) Restrições Geradas:

Fonte: Elaborada pelo autor

2.2.3 **Fluxo de Controle**

Existe uma seção na estrutura de modelagem do *LINGO* que permite a manipulação de dados. É a chamada seção de cálculo, ou *"calc section"* no *LINGO*. Para utilizá-la, basta digitar as palavras-chave *"CALC:"* e *"ENDCALC"* antes e após as instruções de cálculo ou fluxo.

Na *"calc section"*, as declarações do modelo são normalmente executadas sequencialmente. Para alterar a ordem de execução dessas instruções, fluxos de controle como *"@IFC"* e *"@ELSE"* podem ser usados.

Outro fluxo de controle bastante comum, é o *"FOR"*. Dentro da seção de cálculo ele não produz restrições como abordado na Seção 2.2.2, porém executa toda instrução que estiver dentro do seu escopo através de uma estrutura de repetição.

2.2.4 **Desenvolvimento da Modelagem no Lingo**

Depois de abordar algumas funcionalidades, conceitos e sintaxe utilizadas no *LINGO*, esta seção tem como objetivo elucidar como a modelagem descrita na Seção 2.1 foi realizada no *software* de otimização. Para isso, cada trecho da modelagem será discutida, de forma a facilitar a compreensão e justificar o uso do *software* em questão.

2.2.4.1 *Data e Set Sections*

Na parte inicial da modelagem, foram definidos os dados de entrada para as variáveis *"NR_ROBOTS"* e *"NR_POINTS"*, e valores para os atributos dos conjuntos, conforme mostrado nas Figuras 16 e 17 respectivamente. Em ambos os casos, eles recebem valores oriundos de uma aplicação feita em C++ para a navegação dos robôs através da função *"POINTER"*. Os textos escritos em verde são comentários e representam os valores adotados no teste experimental realizado.

Figura 16 – *Data e Set Sections*

```
MODEL:
    DATA:
        NR_ROBOTS = @POINTER(1); ! 2;
        NR_POINTS = @POINTER(2); ! 4;
    ENDDATA

    SETS:
        ROBOTS / 1..NR_ROBOTS / : VEL_MAX, VEL_MIN, PONTO_INICIAL,
            FIRST_COLISION, LAST_COLISION, LAMBDA, LENGHT ;
        POINTS / 1..NR_POINTS / : COLISAO;
        CYCLES / 1..2 / ;
        WAYS ( ROBOTS, POINTS ): PROXIMO, DISTANCE;
        TIME_ROBOT (ROBOTS, POINTS, CYCLES): t;
        COLISION_POINTS ( ROBOTS, ROBOTS, POINTS)| &2 #GT# &1: COLIDE,
            binario;
    ENDSETS
```

Fonte: Elaborada pelo autor

Figura 17 – *Data Section* - Valores dos Atributos

```
DATA:
    ! GENERAL DATA;
    PERIODO_BASE = @POINTER(3); ! TIME IN SECONDS;

    ! ATRIBUTE VALUES;
    VEL_MAX        = @POINTER(4);  ! 0.6 m/s;
    VEL_MIN        = @POINTER(5);  ! 0.2 m/s;
    LAMBDA         = @POINTER(6);  ! 1 for all robots;
    LENGHT         = @POINTER(7);  ! lenght[0] = 10.8 m , lenght[1] = 9.02 m ;
    PONTO_INICIAL  = @POINTER(8);  ! pt_inicial[0] = 3 , pt_inicial[1] = 4;
    COLISAO        = @POINTER(9);  ! colisao[0] = 1, colisao[1] = 1, colisao[2] = 0, colisao[3] = 0;
    PROXIMO        = @POINTER(10); ! proximo[0][0] = 2, proximo[0][1] = 3, proximo[0][2] = 1,
        proximo[0][3] = -1, proximo[1][0] = 2, proximo[1][1] = 4, proximo[1][2] = -1, proximo[1][3] = 1;

     DISTANCE      = @POINTER(11); ! distance[0][0] = 5.29, distance[0][1] = 8.78, distance[0][2] = 0,
        distance[0][3] = -1, distance[1][0] = 1.95, distance[1][1] = 4.79, distance[1][2] = -1, distance[1][3] = 0;
ENDDATA
```

Fonte: Elaborada pelo autor

Na *"set section"* foram definidos os seguintes conjuntos:

- ***ROBOTS***: Esse conjunto possui dois membros (1,2). Seus atributos são listados na Tabela 5 abaixo:
- ***POINTS***: Esse conjunto possui como membros o número de pontos de colisão e pontos iniciais envolvidos no planejamento da missão. Caso seja 6 o número total de pontos, o conjunto ficaria definido como "POINTS"(1,2,3,4,5,6). Ele possui um atributo chamado "COLISAO", que receberá o valor 1 se o ponto em questão for de colisão ou 0 se não for ponto de colisão. Exemplo: Imaginando que os membros do conjunto de 1 a 4 são pontos de colisão e o restante pontos iniciais, tem-se a configuração mostrada na Tabela 6.

Tabela 5 – Atributos do Conjunto *ROBOTS*

ATRIBUTOS	ROBOTS
VEL_MAX	Velocidade Máxima que pode ser atingida pelo robô
VEL_MIN	Velocidade Mínima requerida
PONTO_INICIAL	Índice do Ponto Inicial
FIRST_COLISION	Índice do Primeiro Ponto de Colisão
LAST_COLISION	Índice do Último Ponto de Colisão
LAMBDA	Número Natural adotado para definir o Tempo de Ciclo Ti
LENGHT	Comprimento de Caminho

Fonte: Elaborada pelo autor

Tabela 6 – Valores do atributo de cada membro do conjunto *POINTS*

POINTS	
MEMBROS	**COLISAO**
1	1
2	1
3	1
4	1
5	0
6	0

Fonte: Elaborada pelo autor

- ***CYCLES***: Possui sempre dois membros referentes ao primeiro e segundo ciclos de trabalho, pois como foi abordado na Seção 2.1, o planejamento feito será adotado em todos os ciclos de trabalho, uma vez que o problema é periódico e comensurável.

- ***WAYS***: Esse é um conjunto derivado, formado pelos conjuntos *"ROBOTS"* e *"POINTS"*. Como atributos, possui "PROXIMO"e *"DISTANCE"*. Este último refere-se à distância do ponto atual ao ponto inicial. Para entender mais facilmente a função do atributo próximo, foi criada a Tabela 7 construída apenas para um robô. Para os demais, segue-se a mesma lógica. Porém, considere o mapa de pontos do caminho que o robô em questão possui (lembrando que os pontos estão em sequência): $P1 \rightarrow P2 \rightarrow P3 \rightarrow P4 \rightarrow P5 \rightarrow P1$.

Tabela 7 – Conteúdo do atributo de cada membro do Conjunto *WAYS*

WAYS	
MEMBROS (ROBOTS,POINTS)	**PRÓXIMO**
(1,1)	2
(1,2)	3
(1,3)	4
(1,4)	5
(1,5)	1
(1,6)	-1

Fonte: Elaborada pelo autor

Note então que o atributo de cada membro carrega o índice do seu próximo ponto. É importante ressaltar que o membro *"WAYS"*(1,6) recebeu -1, pois este ponto não pertence ao caminho do robô que está sendo analisado.

- ***TIME_ROBOT***: Conjunto que deriva de três outros conjuntos primitivos. Possui como atributo a variável t que carregará o tempo que os robôs levam para percorrer cada subcaminho.

- ***COLISION_POINTS***: Conjunto derivado que irá receber os pontos que serão analisados. Vale a pena ressaltar que foi utilizada a estrutura condicional para filtrar os dados, de forma que o conjunto não recebe dados repetidos. Foi utilizada a mesma estratégia mostrada na Tabela 2.

2.2.4.2 *Calc Section*

Esta seção da modelagem é responsável por realizar alguns cálculos de forma a definir valores aos atributos dos conjuntos envolvidos, como pode ser visto na Figura 18. A primeira parte atua no atributo do conjunto *"COLISION_POINTS"* e seu objetivo é preencher "COLIDE"com 1 para os índices que correspondem a pontos de colisão e 0 no restante dos pontos. Logo, a estrutura de repetição irá percorrer cada elemento permitindo que seja feito o teste condicional e a atribuição de valores.

O outro cálculo efetuado foi criado para encontrar o primeiro e último ponto presente no caminho de cada robô, conforme exigem as inequações (2.20) e (2.21). Note através da Figura 18 que a primeira instrução após o início da estrutura de repetição visa encontrar a menor distância, ou seja, o primeiro ponto de colisão enquanto que a segunda instrução acha o índice desse ponto. O mesmo raciocínio é adotado para encontrar o último ponto que é armazenado no atributo *"LAST_COLISION"*.

Figura 18 – *Calc Section*

```
CALC:
    ! COLISION POINTS EVALUATION;
    @FOR ( COLISION_POINTS (IROB1 , IROB2 , IPOINT):
        @IFC ( PROXIMO (IROB1 , IPOINT)  #GT#  0  #AND#  PROXIMO (IROB2 , IPOINT)  #GT#  0:
            COLIDE (IROB1 , IROB2 , IPOINT) = 1;
            @ELSE
                COLIDE (IROB1 , IROB2 , IPOINT) = 0;
        )
    );

    ! EVALUATE FIRST COLISION POINT FOR EACH ROBOT;
    @FOR ( ROBOTS(IROB):
        FIRST_COLISION (IROB) = @MIN ( WAYS(JROB , IPOINT) | JROB  #EQ#  IROB  #AND#
            DISTANCE (JROB , IPOINT)  #GT#  0 : DISTANCE (JROB , IPOINT) );

        FIRST_COLISION (IROB) = @SUM ( WAYS (JROB , IPOINT)  |  JROB  #EQ#  IROB  #AND#
            DISTANCE (JROB , IPOINT)  #EQ#  FIRST_COLISION (IROB) : IPOINT );

        LAST_COLISION (IROB) = @MAX ( WAYS (JROB , IPOINT)  |  JROB  #EQ#  IROB  #AND#
            DISTANCE (JROB , IPOINT)  #GT#  0 :  DISTANCE (JROB , IPOINT));

        LAST_COLISION (IROB) = @SUM ( WAYS (JROB , IPOINT)  |  JROB  #EQ#  IROB  #AND#
            DISTANCE (JROB , IPOINT)  #EQ#  LAST_COLISION (IROB) : IPOINT );
    );
ENDCALC
```

Fonte: Elaborada pelo autor

2.2.4.3 *Definindo Função Objetivo e Restrições*

Como foi dito na Seção 2.1, o objetivo é maximizar o menor intervalo de tempo Δt que os robôs passam por um determinado ponto de colisão. Logo, isso é feito no *LINGO* através do comando *"MAX"*, como pode ser visto na Figura 19.

Figura 19 – Função Objetivo e Restrições

```
! THE OBJECTIVE;
[OBJECTIVE] MAX = DELTA_T;

! TIMER MARGIN CONSTRAINT;
@FOR( POINTS(J) | COLISAO(J) #EQ# 1:
    @FOR ( COLISION_POINTS(IROB1,IROB2, IPOINT)| IPOINT #EQ# J #AND# COLIDE(IROB1,IROB2,IPOINT)
        #EQ#1:
        [MARGIN_TIME_A] t(IROB1,J,1) - t(IROB2,J,1) >= DELTA_T - 1000000*binario(IROB1,IROB2,J);
        [MARGIN_TIME_B] t(IROB1,J,1) - t(IROB2,J,1) <= -DELTA_T + 1000000*(1-binario(IROB1,IROB2,J));
    );
);

! PERIOD TIME CONSTRAINT;
@FOR ( ROBOTS(IROB):
    @FOR ( POINTS(IPOIN) | PROXIMO(IROB,IPOIN) #GT# 0:
        [CYCLE] t(IROB,IPOIN,2) - t(IROB,IPOIN,1) = LAMBDA(IROB)*PERIODO_BASE;
    );
);
```

Fonte: Elaborada pelo autor

Posteriormente, há uma série de instruções que atuam nas restrições do modelo. Na

figura 19 são mostradas duas: "Margem de Tempo", e "Periodicidade e Comensurabilidade de Ciclos". Começando pela de Margem de Tempo, pode ser visto que existe um *loop* que irá percorrer somente os pontos de colisão para que seja possível implementar devidamente as inequações (2.16) e (2.17). Depois foi implementada a restrição de Periodicidade e Comensurabilidade de Ciclos utilizando a equação (2.22).

É importante ressaltar que, como foi dito na Seção 2.2.2, o uso da função *"FOR"* permite que sejam gerada as restrições para cada membro do conjunto em questão, exatamente como foi abordado na Figura 15 e na Tabela 4(b).

As próximas restrições atuam na velocidade dos robôs, conforme pode ser visto na Figura 20. No entanto, a primeira restrição considera apenas o primeiro ciclo de trabalho, seguindo a inequação (2.19). Logo, essa restrição informa que o tempo de percurso de um dado subcaminho deve estar compreendido entre um tempo mínimo e máximo permitido, considerando as velocidades máxima e mínima.

Já a segunda foi elaborada para atender o último ponto do circuito, pois ele está na fase de transição entre os dois ciclos, de acordo com a inequação (2.20). Porém segue a mesma ideia da inequação anterior, com exceção de que o subcaminho em análise é o último.

Figura 20 – Restrições de Velocidade

```
! SPEED CONSTRAINT - POINTS IN THE SAME CYCLE;
@FOR ( ROBOTS(IROB):
    @FOR ( POINTS(J) | PROXIMO(IROB,J) #GT# 0:
        @FOR( TIME_ROBOT (JROB , JJ , ICYC) | JROB  #EQ#  IROB  #AND# JJ  #EQ#  J  #AND#  ICYC
            #EQ# 1 #AND#  DISTANCE (IROB , PROXIMO (IROB,J))  #GT#  DISTANCE(IROB,J):

            [INTERMEDIATE_A] t (IROB,PROXIMO(IROB,J),1) - t (IROB,J,1) <=
                (DISTANCE (IROB , PROXIMO (IROB,J))  -  DISTANCE(IROB,J))  /  VEL_MIN(IROB);

            [INTERMEDIATE_B]  t (IROB,PROXIMO(IROB,J),1) - t (IROB,J,1) >=
                (DISTANCE(IROB,PROXIMO(IROB,J)) - DISTANCE(IROB,J))  /  VEL_MAX(IROB);
        );
    );
);

! SPEED CONTRAINT - LAST POINT;
@FOR ( ROBOTS(IROB):
    [ENDING_A] t (IROB,PONTO_INICIAL(IROB),2) - t (IROB,LAST_COLISION(IROB),1) <= (LENGHT(IROB) -
        DISTANCE(IROB,LAST_COLISION(IROB))) / VEL_MIN(IROB);

    [ENDING_B] t (IROB,PONTO_INICIAL(IROB),2) - t (IROB,LAST_COLISION(IROB),1) >= (LENGHT(IROB) -
        DISTANCE(IROB,LAST_COLISION(IROB))) / VEL_MAX(IROB);
);

@FOR( POINTS(J) | COLISAO(J) #EQ# 1:
    @FOR ( COLISION_POINTS(IROB1,IROB2, IPOINT)  |  IPOINT  #EQ#  J  #AND#
        COLIDE(IROB1,IROB2,IPOINT)  #EQ#  1 :

            [END_A] t (IROB1,J,1)  - t (IROB2,J,1) <= PERIODO_BASE  -  DELTA_T;
            [END_B] t (IROB1,J,1)  - t (IROB2,J,1) >= - PERIODO_BASE + DELTA_T;
    );
);
```

Fonte: Elaborada pelo autor

Para finalizar o modelo, foram incorporadas as últimas três restrições, conforme abordado na Seção 2.1.1.1, ilustradas na Figura 21. Ao fim da modelagem, outra *"Data Section"* foi criada para definir as variáveis que são enviadas para a aplicação em C++ que irá processar os dados e navegar o robô.

Figura 21 – Últimas Restrições

```
DELTA_T >= 0;

t(1,PONTO_INICIAL(1),1) = 0;
t(2,PONTO_INICIAL(2),1) = 0;

! Binary variables;
@FOR ( COLISION_POINTS: @BIN(binario) );

DATA:
    @POINTER(12) = OBJECTIVE;
    @POINTER(13) = @STATUS();
    @POINTER(14) = t;
    @POINTER(15) = DELTA_T;
ENDDATA
```

Fonte: Elaborada pelo autor

2.3 ESTUDO DE CASOS DA MODELAGEM UTILIZADA

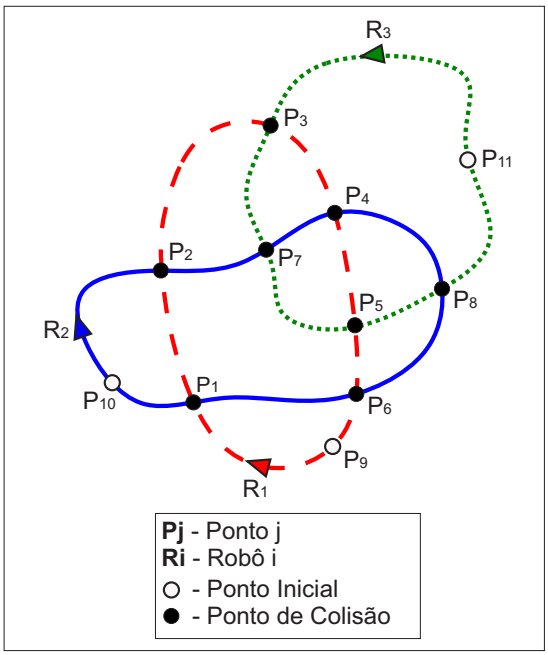
Nesta seção será feito alguns exemplos utilizando a metodologia descrita no item 2.1 e modelada no *LINGO* na subseção 2.2.4, para demonstrar sua utilização e facilitar a compreensão de tudo o que foi abordado neste capítulo e do que vai ser trabalhado no capítulo seguinte, que terá como foco principal a metodologia utilizada para desenvolver a aplicação em um caso real.

2.3.1 **Caso 1 - Três robôs e o oito pontos de colisão**

Considere o circuito mostrado na Figura 22. Os caminhos em vermelho, azul e verde possuem respectivamente 80, 75 e 83 metros de comprimento. As velocidades mínimas e máximas dos robôs são idênticas e iguais a 0,2 m/s e 2 m/s. A Tabela 8 mostra o mapa de pontos formados por cada circuito e as respectivas distâncias entre esses pontos e o ponto de partida dos robôs. Foi definido também $\gamma = 1$ para todos os robôs.

Levando-se em consideração o tamanho de cada percurso e a as velocidades máxima e mínima que os robôs podem desenvolver, encontra-se facilmente o tempo máximo e mínimo que os robôs podem completar uma volta. Sendo assim, define-se o tempo base compreendido nesse intervalo, fazendo $T = 100$. Logo, através da equação (2.10) tem-se que o tempo de ciclo $T_i = 100s$.

Figura 22 – Circuito Exemplo



Fonte: Elaborada pelo autor

Tabela 8 – Mapa de Pontos e Distâncias

ROBÔS	PONTOS							
R1	P1	P2	P3	P4	P5	P6	P9	P1
DISTÂNCIAS (m)	15	27	44	54	64	76	0	15
R2	P2	P7	P4	P8	P6	P1	P10	P2
DISTÂNCIAS (m)	12	22	30	42	54	70	0	12
R3	P3	P7	P5	P8	P11	P3		
DISTÂNCIAS (m)	30	45	60	70	0	30		

Fonte: Elaborada pelo autor

Enviando todas essas informações para o *LINGO* de modo que possa ser executado o modelo desenvolvido, obtém-se como valor da função objetivo um $\Delta t = 36,75$. A Tabela 9 mostra os instantes de tempo que cada robô cruza pelos pontos de colisão.

Tabela 9 – Tempo em que cada robô cruza pelos Pontos de Colisão

PONTOS	TEMPO (s)			INTERVALO DE TEMPO
	R1	R2	R3	
P1	38,25	75	-	36,75
P2	69,25	6	-	63,25
P3	82	-	40,25	42,25
P4	87	23,75	-	63,25
P5	92	-	55,25	36,75
P6	98	61,25	-	36,75
P7	-	11	47,75	36,75
P8	-	29,75	93	63,25

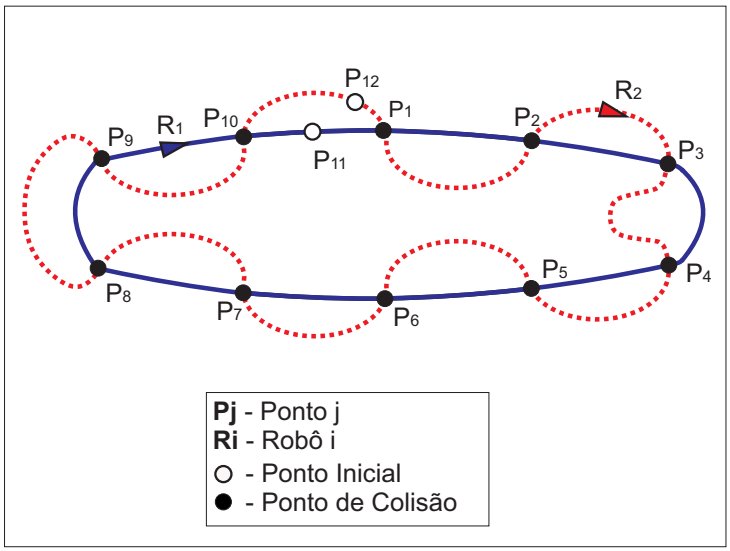
Fonte: Elaborada pelo autor

Observa-se pela Tabela 9 que não houve colisão em nenhum dos pontos descritos, uma vez que os robôs não alcançaram os pontos no mesmo instante e que todos os intervalos de tempo entre a passagens dos robôs foram iguais ou superiores à Δt.

2.3.2 Caso 2 - Dois robôs e o dez pontos de colisão

Neste exemplo, definiu-se que o caminho percorrido pelo robô 1 possua comprimento igual a 100 metros e o do robô 2 igual a 150 metros. As velocidades mínimas e máximas dos robôs são de 0,2 m/s e 1 m/s respectivamente. Utilizou-se γ igual a 1 para ambos os robôs. A tabela 10 apresenta a distância formada entre o ponto inicial e os demais pontos. Nota-se que o valor zero representa o ponto inicial para cada robô.

Figura 23 – Circuito Exemplo



Fonte: Elaborada pelo autor

Tabela 10 – Mapa de Pontos e Distâncias

ROBÔS	PONTOS											
R1	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P1
DISTÂNCIAS (m)	5	15	25	35	45	55	65	75	85	95	0	5

R2	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P12	P1
DISTÂNCIAS (m)	5	20	35	50	65	80	95	110	125	140	0	5

Fonte: Elaborada pelo autor

Após a execução do programa que realiza a otimização, obteve-se os tempos em cada robô cruza pelos pontos de colisão assim como o valor da função objetivo, em que $\Delta t = 20s$. Estes valores e o intervalo de tempo da passagem dos robôs por um mesmo ponto de colisão são mostrados na tabela 11.

Tabela 11 – Tempo em que cada robô cruza pelos Pontos de Colisão

PONTOS	TEMPO (s)		INTERVALO DE TEMPO
	R1	R2	
P1	25	5	20
P2	60	20	40
P3	70	50	20
P4	95	65	30
P5	105	80	25
P6	115	95	20
P7	145	110	35
P8	195	125	70
P9	245	200	45
P10	295	275	20

Fonte: Elaborada pelo autor

Novamente, percebe-se que os robôs não chegaram em nenhum dos pontos de colisão ao mesmo tempo, mantendo sempre um intervalo de tempo igual ou superior a Δt.

2.4 CONCLUSÃO

Esse capítulo fundamentou toda a etapa de otimização implementada no presente trabalho, começando pela Seção 2.1 onde foi descrita a modelagem matemática do problema em questão, sendo desenvolvidas as inequações que restringem o problema de forma a maximizar a função objetivo Δt referente ao menor tempo entre a passagem dos robôs por um ponto de colisão. Foi demostrando também que é possível realizar um planejamento de horizonte infinito, com um número finito de variáveis e restrições.

Posteriormente apresentou-se o *software* de otimização utilizado na modelagem, assim como suas características e principais funcionalidades. Demonstrou-se que ele

possui vantagens que foram fundamentais para o desenvolvimento do trabalho e aplicação da metodologia, como a possibilidade da troca de informações com outras aplicações envolvendo dados essenciais, como por exemplo, o perfil de velocidade que os robôs terão ao longo do percurso.

Por último, foi elaborado alguns exemplos para demonstrar a utilização do controlador central e contextualizar em quais situações a metodologia poderia ser usada. Após a execução do problema de otimização no *LINGO*, concluiu-se através da análise das Tabelas 9 e 11 que o planejamento realizado para os casos em questão cumpriu todos os requisitos, mantendo o intervalo de tempo entre a passagem dos robôs maior ou igual ao Δt encontrado.

3 METODOLOGIA PROPOSTA

Neste capítulo será descrito todo o processo de desenvolvimento prático, que consiste em realizar a navegação dos robôs em um ambiente de trabalho, seguindo as condições previamente definidas como *waypoints* e o perfil de velocidade média para cada trecho do circuito.

Para realizar esta tarefa, contou-se com as funcionalidades do *framework* ROS, que de forma geral, foi utilizado para fazer a leitura dos sensores e acionamento dos robôs. Será descrito nas seções posteriores como a implementação foi realizada, informando as ferramentas, técnicas e equipamentos utilizados.

3.1 ROBOT OPERATING SYSTEM

O *Robot Operating System* (ROS) é um *framework* de desenvolvimento de aplicações voltadas para área de robótica. É constituído por um conjunto de ferramentas, bibliotecas e convenções que auxiliam o usuário a desenvolver rotinas e comportamentos robóticos. Possui um vasto acervo de códigos *open source* que tanto atuam em baixo nível, facilitando a tarefa de estabelecer a comunicação com sensores e atuadores, quanto em alto nível fornecendo aplicações mais elaboradas como *Simultaneous Localization and Mapping* (SLAM).

Foi criado no *Stanford Artificial Intelligence Lab* e atualmente é mantido pela *Open Source Robotics Foundation* (OSRF), contando com diversos patrocinadores como *National Aeronautics and Space Administration* (NASA), *Defense Advanced Research Projects Agency* (DARPA), *BOSCH*, entre outros.

Sua concepção se deu com a ideia de integração entre as pesquisas realizadas em várias universidades e os diversos equipamentos desenvolvidos pela indústria, poupando tempo e um grande esforço que seria requerido para construção de projetos que partiriam do zero.

Com grande aceitação na academia, o ROS tornou-se uma ferramenta poderosa, pois conseguiu reunir em um só lugar, muitas técnicas e algoritmos que antes estavam restritos a grupos isolados de pesquisa e que hoje estão disponíveis para qualquer pessoa, ajudando a alavancar os estudos e fazer da robótica uma área em forte expansão.

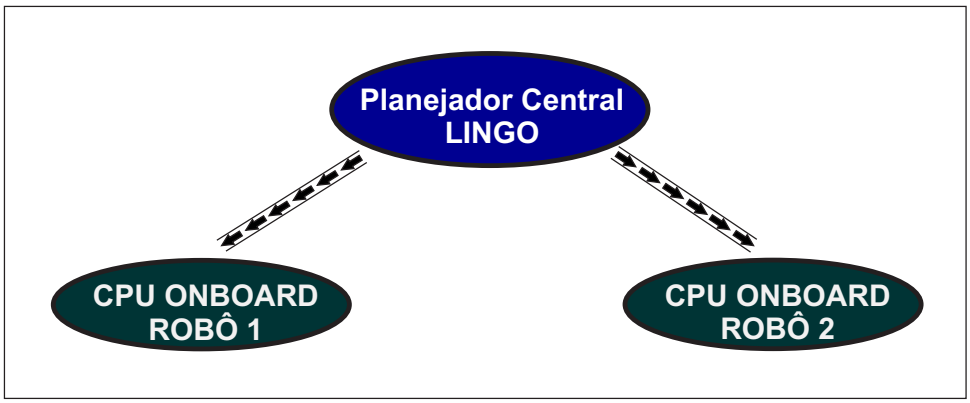
Ele fornece suporte a diversas bibliotecas e ferramentas que permitem ao programador elaborar rotinas tanto em C/C++ quanto em Python, sendo que uma das grandes vantagens é a possibilidade do usuário usufruir durante um mesmo processo de aplicações construídas em ambas as linguagens [28].

O ROS possui a filosofia de utilizar programas que realizam funções bem específicas, como fazer a leitura de um *laser scan*, câmera ou simplesmente enviar velocidades para as

rodas de um robô, por exemplo. Tais aplicações comunicam entre si através de estruturas disponibilizadas pelo *framework*, como tópicos e serviços, sendo que esses programas podem estar sendo executados em uma única unidade de processamento ou em vários hardwares, desde que estes estejam conectados a uma rede de comunicação TCP [28]. Isto é extremamente útil pois existe a possibilidade de se utilizar computadores com pouco poder de processamento embarcado nos robôs executando tarefas simples enquanto uma outra unidade realiza cálculos mais complexos e envia dados de navegação ou comandos para os computadores *onboard*.

Neste trabalho um programa desenvolvido em C++ terá a tarefa de realizar o planejamento das velocidades dos robôs chamando o *script* do *LINGO* enquanto que os computadores embarcados receberão essas informações e serão responsáveis pela navegação dos robôs, como mostrado na Figura 24.

Figura 24 – Processos Executados



Fonte: Elaborada pelo autor

3.2 CONCEITOS BÁSICOS DO ROS

Nesta seção será abordado alguns conceitos básicos do *ROS*, porém fundamentais para o entendimento de como o presente trabalho foi desenvolvido. Utilizou-se o livro *Learning ROS for Robotics Programming* [28] como fonte principal de consulta.

3.2.1 ***Node***

Node, também chamado de *Nó* são processos ou programas que executam algum tipo de processamento ou tarefa. Eles podem ser interrompidos de forma independente, ou seja, se vários *Nós* estiverem em execução há a possibilidade de encerrar algum enquanto outro está ainda em operação.

Os *Nós* comunicam entre si por meio de mensagens que são publicadas em tópicos e podem fornecer serviços para outros *Nós*, estabelecendo uma comunicação que segue o padrão cliente-servidor. Geralmente, os *Nós* controlam uma única funcionalidade como a

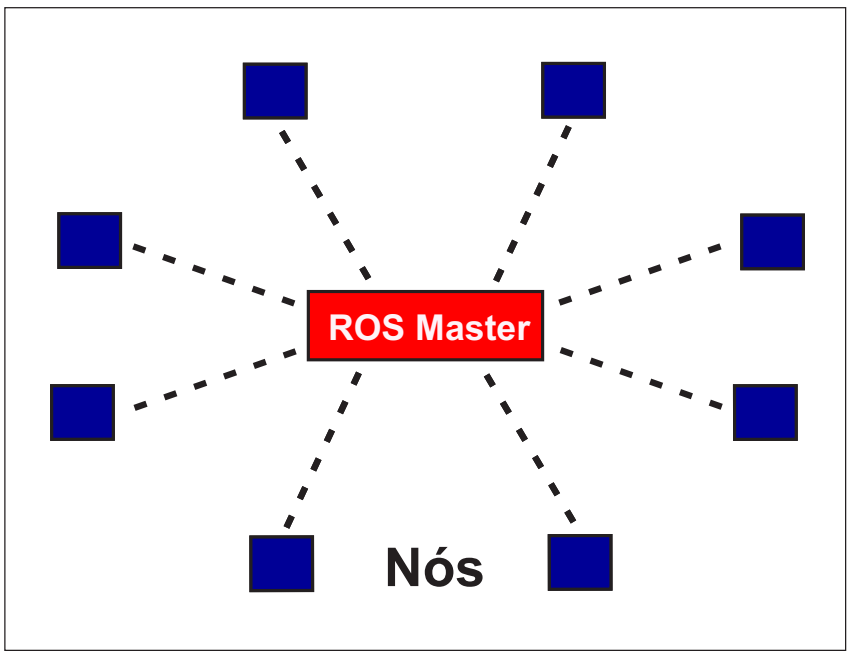
leitura de algum sensor ou executa uma única tarefa como de reconhecimento de padrões, por exemplo.

Um *Nó* que exerce papel muito importante na estrutura do *ROS* é o chamado *ROS Master*. Ele é um processo único que gerencia a comunicação entre os *Nós*, permitindo que um processo seja capaz de encontrar outros através de uma variável de ambiente chamada *ROS_MASTER_URI*. Logo, o único *Nó* obrigatório em uma rede ROS é o *ROS Master* que é executado através do comando *roscore*.

O ROS fornece várias ferramentas relacionadas aos *Nós* que auxiliam o usuário a obter informações de forma rápida, como as funções mostradas abaixo:

- **rosrun**: Comando para executar um *Nó*. É necessário passar como argumento para a função o nome do pacote onde está armazenado o código fonte e o nome do executável.
- **rosnode list**: Mostra a lista de todos os *Nós* em execução.
- **rosnode info**: Apresenta informações do *Nó* que foi passado como argumento.
- **rosnode kill:** Encerra a execução do *Nó*.

Figura 25 – ROS Master



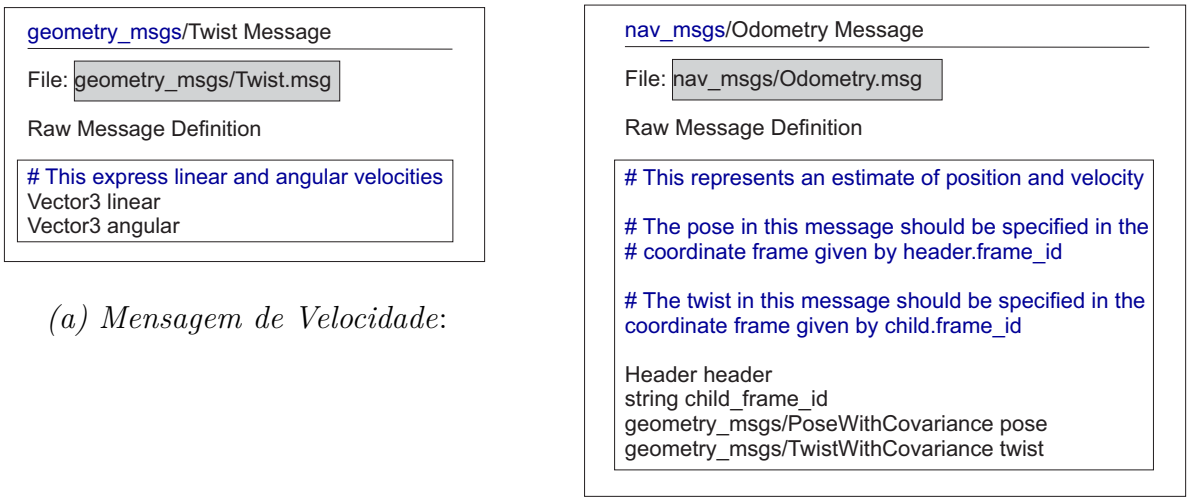
Fonte: Elaborada pelo autor

3.2.2 Mensagens

Os *Nós* comunicam entre si por meio de mensagens, que são arquivos de texto que carregam informações. Elas suportam diversos tipos de variáveis como *int, float, bool, char,*

string além de dados criados pelo ROS como streaming de vídeo. Existem vários tipos de mensagens já desenvolvidas pelo ROS, como de velocidades e de posição mostradas na Figura 26, porém é possível o usuário criar a sua própria mensagem.

Figura 26 – Exemplos de Mensagens Disponíveis no ROS



(a) Mensagem de Velocidade:

(b) Mensagem de Posição:

Fonte: [29]

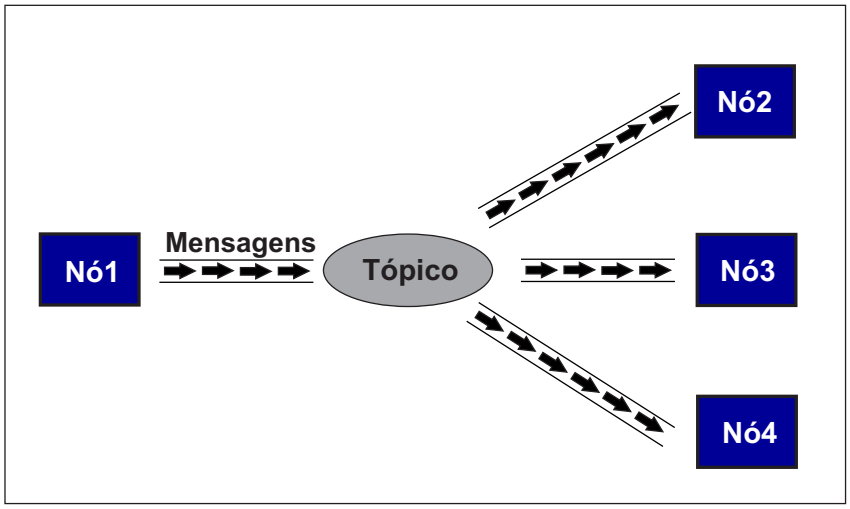
O ROS também fornece um conjunto de ferramentas para se trabalhar com as mensagens, e algumas delas são citadas abaixo:

- **rosmsg list**: lista todas as mensagens utilizadas.
- **rosmsg show**: mostra como a mensagem passada como argumento foi definida.

3.2.3 Tópicos

Tópico é o meio de comunicação desenvolvido pelo ROS para que as mensagens possam trafegar, possibilitando o envio de dados de um *Nó* para outro. É análogo a uma mensagem de *broadcast*, uma vez que o *Nó* publicador envia essa mensagem uma única vez e todos os outros *Nós* que estão inscritos neste tópico conseguem acessar essa informação, conforme mostrado na Figura 27.

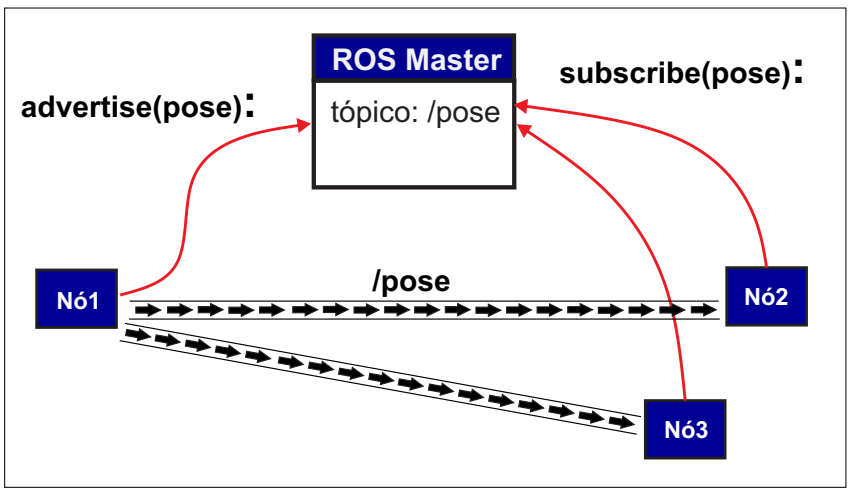
Figura 27 – Tópico



Fonte: Elaborada pelo autor

Note pela Figura 27 que a forma de envio é unidirecional e para estabelecer essa comunicação o *Nó* publicador informa para o *ROS Master* que irá publicar mensagens no tópico *pose*, por exemplo. O *ROS Master* então registra o tópico e a sua origem. A partir desse ponto as mensagens já estão sendo publicadas no tópico e disponíveis para outros *Nós*, desde que estes se inscrevam informando ao *ROS Master* que desejam receber as mensagens.

Figura 28 – Envio de Mensagens - Tópico



Fonte: Elaborada pelo autor

3.2.4 Pacotes Utilizados

O sistema do ROS é organizado em pacotes que fornecem funcionalidades bem específicas. São arquivos que contém os códigos-fonte, bibliotecas independentes, arquivos de configuração e compilação, além do próprio executável do *Nó*. O objetivo dessa

organização é proporcionar principalmente o reuso do código, caso algum usuário deseje alterá-lo.

Os pacotes desenvolvidos para o ROS devem seguir um princípio básico: fornecer funcionalidades suficientes para ele ser prático, porém leve. Um pacote é o menor arquivo que pode ser construído pelo ROS e é a forma como o software é empacotado para o lançamento.

No escopo deste trabalho, dois pacotes utilizados merecem destaque: *ROSARIA* e *ARPOSE*.

3.2.4.1 ROSARIA

O pacote que faz o acesso de baixo nível dos sensores e motores dos robôs, é denominado *ROSARIA*. Ele Permite a comunicação com a maioria dos robôs fabricados pela empresa *Mobile Robts Inc.*, como o *Pioneer P3DX* que foi utilizados para realizar os experimentos práticos [29].

O pacote faz uso da biblioteca *ARIA*, uma *Application Programming Interface* (API) que fornece diversas funções desenvolvidas para controle e leitura de velocidade linear e angular, de posição e calibração de sensores. O pacote foi implementado através do *Nó RosAria*, sendo responsável por chamar as funções do Aria. Ele se inscreve no tópico *cmd_vel* e publica dados relacionados ao robô nos seguintes tópicos: [29]

- **pose (nav_msgs/Odometry)**: publica informações de hodometria.
- **bumper_state (rosaria/BumperState)**: publica o estado (ativo ou não) do sensor localizado no para-choque.
- **sonar (sensor_msgs/PointCloud)**: publica as leituras feitas pelo sonar.
- **battery_state_of_charge (std_msgs/Float32)**: publica o nível de carga da bateria em termos de porcentagem.
- **battery_voltage(std_msgs/Float32)**: medida da tensão da bateria.
- **motors_state (std_msgs/Bool)**: indica se os motores estão ligados ou desligados.

3.2.4.2 ARPOSE

O pacote *ARPOSE* foi utilizado para fazer o reconhecimento dos marcadores que foram colocados no teto do ambiente para que o robô pudesse se localizar no espaço. Desta forma, uma câmera voltada para cima foi anexada em cada robô e com as informações fornecidas pelo *Nó*, como as distâncias e rotações da câmera em relação ao marcador, foi

possível encontrar a posição do robô em relação ao sistema de referências global. Esse assunto será tratado na seção 3.3. Foram utilizados marcadores como os mostrados na Figura 29.

Figura 29 – Exemplos de Marcadores

(a) PattHiro: *(b) PattKanji*:

Fonte: [29]

Tal estratégia foi adotada pois um dos requisitos definidos foi que o robô pudesse seguir pelos caminhos previamente planejados durante vários ciclos de trabalho. Portanto, utilizar as leituras feitas puramente pelo hodômetro tornou-se inviável uma vez que os erros inerentes são acumulativos [12], fato que colocaria em grande risco a operação, já que o controle do robô seria alimentado com informações imprecisas que se tornariam cada vez maiores com o passar tempo.

Os sensores de imagem utilizados mostrados na Figura 30 também possuem erros intrínsecos, porém eles não se acumulam e em ambiente controlado eles são bem aceitáveis, possibilitando a realização do teste prático abordado na seção 4.1.2.

Figura 30 – Câmeras Utilizadas

(a) Microsoft LifeCam Cinema: *(b) Microsoft LifeCam HD 6000*:

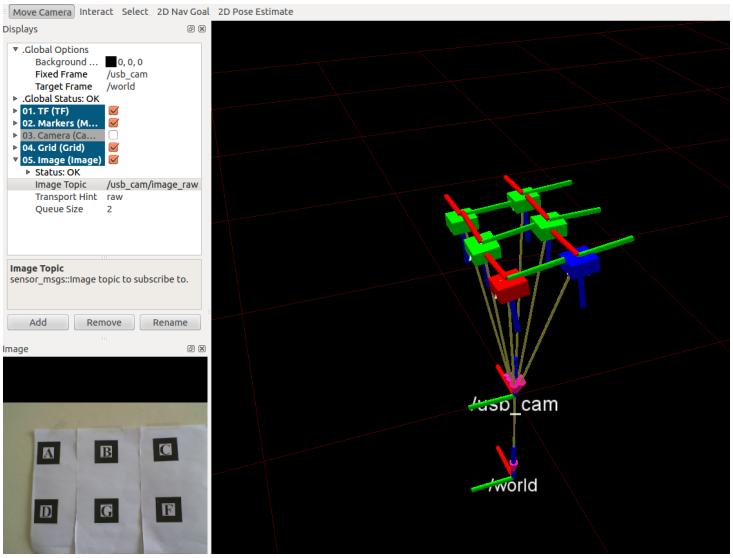
Fonte: [30]

O pacote *ARPOSE* utiliza uma ferramenta chamada *ARToolKit*, desenvolvida para construir aplicações em realidade aumentada (do inglês *Augmented Reality* (AR)).

O processo, de um modo geral, é feito capturando as imagens de uma câmera e processando-as, a fim de encontrar o local exato onde o caracter virtual deve ser construído. Para resolver esse problema, um algoritmo de visão computacional foi desenvolvido com o objetivo de realizar o *tracking* ou rastreamento do marcador e calcular a sua posição e orientação em relação à câmera. Uma das grandes dificuldades é tornar esse algoritmo rápido o suficiente para realizar o processamento em tempo real.

A Figura 31 mostra o *Nó* do *AR_Pose* em execução. Logo realiza-se o rastreamento do marcador e calcula-se sua posição e orientação em relação ao sistemas de coordenadas da câmera. Através da interface *rviz* (que é um *Nó* do ROS), é possível visualizar os sistemas de coordenadas envolvidos (da câmera e do marcador) e como eles se relacionam.

Figura 31 – *Nó AR_Pose* em Execução



Fonte: Elaborada pelo autor

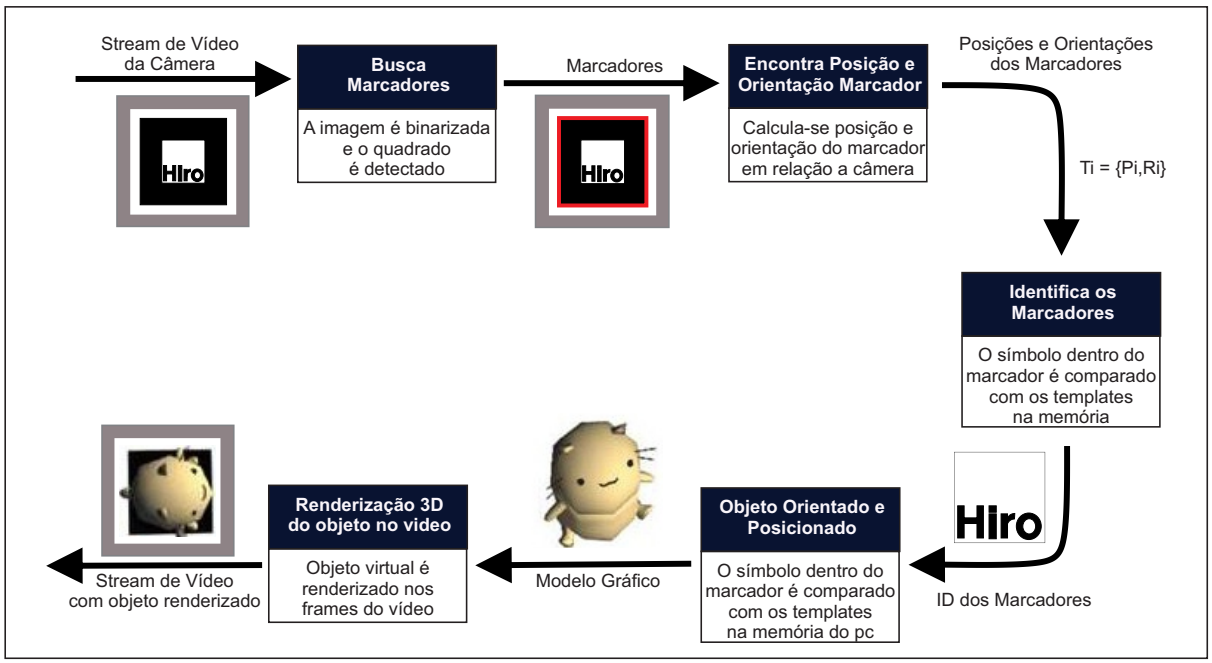
3.2.4.2.1 Funcionamento do ARToolKit

Como foi descrito anteriormente, a ferramenta *ARToolKit* permite o desenvolvimento de aplicações em realidade aumentada. Portanto, para realizar essa tarefa a câmera captura as imagens do mundo real e as envia para um computador. O algoritmo de visão computacional procura em cada *frame* o marcador, que se encontrado permite o algoritmo calcular a posição e orientação da câmera em relação ao alvo detectado [31]. O sistema de visão computacional será visto com mais detalhes na seção 3.2.4.2.2.

Posteriormente, identifica-se os marcadores utilizando a técnica *template matching* que compara o símbolo localizado no interior do marcador com os templates armazenados na memória do computador, verificando o grau de similaridade [12] e [32]. Em seguida,

o modelo gráfico é desenhado sobre o alvo permitindo que o usuário veja através de um *display* o caracter nas imagens do mundo real, conforme mostrado na Figura 32 [33].

Figura 32 – Funcionamento do ARToolKit



Fonte: [33]

O *ARToolKit* apresenta algumas limitações que restringem uma determinada aplicação. A primeira delas está relacionada ao campo de visualização da câmera, ou seja, o marcador precisa estar compreendido nessa área para que ele possa ser detectado. Dependendo do seu tamanho e de sua complexidade mesmo que ele esteja aparecendo totalmente na imagem pode acontecer dele não ser encontrado. Logo, quanto maior for o marcador, maior é a distância que ele pode estar do sensor de imagem.

A complexidade do marcador também influencia muito na sua identificação. Marcadores simples são encontrados com mais facilidade. Logo, padrões em baixa frequência, ou seja, com grandes regiões em preto e branco apresentam maior eficiência em termos de detecção [31].

O *ARToolKit* também sofre com a orientação do marcador em relação à câmera. Se ele aparecer na imagem muito inclinado, o algoritmo de visão terá problemas para detectá-lo, pois o centro do marcador poderá estar distorcido e pouco visível.

Outro problema está relacionado com as condições de luminosidade do local. Se houver pouca luz haverá menos contraste entre as regiões brancas e pretas. Logo o sistema terá mais dificuldade para identificar o marcador, pois o algoritmo de visão não possui robustez suficiente para rastrear o marcador com muitos ruídos presentes na imagem. Dependendo do tipo de material que o marcador é feito, ele pode refletir luz e produzir brilho em excesso, o que também prejudica na sua identificação.

3.2.4.2.2 Algoritmo de Visão Computacional

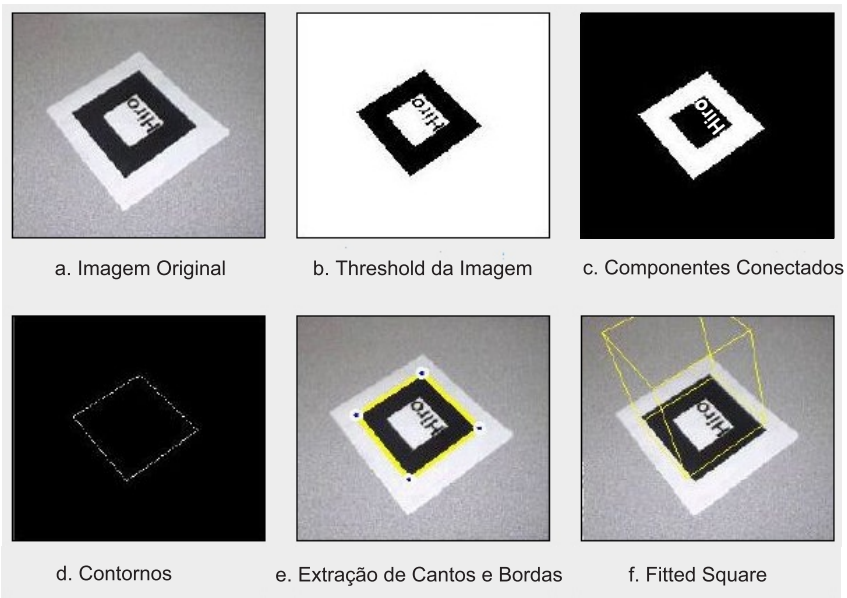
O algoritmo de visão utilizado no *ARToolKit* realiza uma série de processamentos na imagem com o intuito de deixar o processo mais rápido, com menos ruído e mais eficiente [31]. Primeiramente usa-se um filtro que aplica um *trheshold* na imagem para deixá-la binária, reduzindo a quantidade de informação presente na imagem original como na Figura 33 a) e b) [33].

Em seguida, utiliza-se a função *Labeling* que faz a leitura de toda a imagem, pixel por pixel, com o objetivo de distinguir e agrupá-los de acordo com sua conectividade (valores semelhantes de intensidade), atribuindo-os com um identificador único, o que permite a execução do *template matching* somente do símbolo localizado no interior do marcador [12], [32] e [34]. Esse processo é conhecido como *Connected Component Labeling* e seu resultado é mostrado na Figura 33 c) [33].

Posteriormente, encontra-se os contornos da imagem e estima-se as linhas que conectam os pontos externos do alvo baseando-se em detecção de *corners*, como mostrado na Figura 33 d) e e). É importante ressaltar que os índices dos pixels envolvidos são salvos nessa etapa [33].

A partir desta informação pode ser realizada a correspondência entre os pixels da imagem de entrada e dos marcadores do banco de dados. A imagem do marcador é então comparada com os padrões do banco de dados usando *template matching*. Por último, calcula-se a posição e orientação da câmera em relação ao alvo detectado, permitindo que o objeto 3D possa ser renderizado na posição correta [33].

Figura 33 – Algoritmo de Visão Computacional



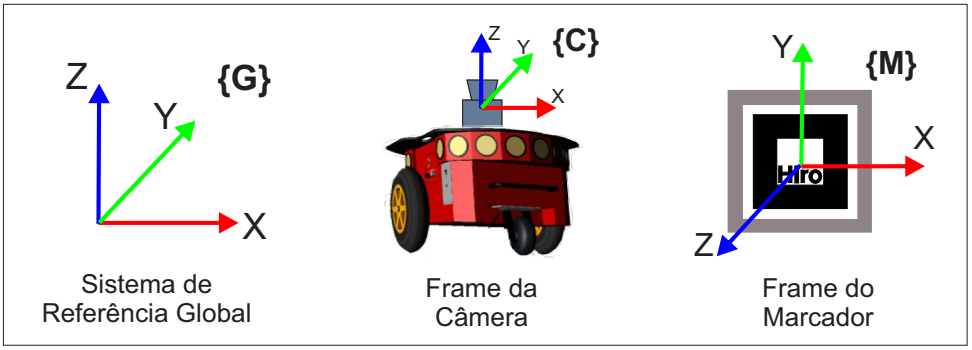
Fonte: [31]

3.3 DESCRIÇÃO DOS *FRAMES*

Como dito na seção 3.2.4.2, foi utilizada uma câmera fixada ao robô como sensor de posição. Então anexou-se marcadores no teto para que através do sistema de visão computacional descrito na seção anterior fosse possível encontrar a localização dos robôs, dado que a posição e orientação dos marcadores são conhecidas .

No entanto, para realizar tal tarefa foi necessário descrever as relações entre posições e orientações dos corpos envolvidos, sendo estabelecidos os sistemas de coordenadas mostrados na Figura 34. Vale a pena ressaltar que para a aplicação em questão foi possível considerar sem prejuízos a câmera e o robô como sendo um único corpo rígido, devido a proximidade entre eles. Logo, foi utilizado um único *frame* para os dois.

Figura 34 – Frames Utilizados



Fonte: Elaborada pelo autor

Para descrever um ponto localizado em um determinado *frame* em relação à outro sistema de coordenadas, precisa-se de informações de orientação e posição, uma fez que eles podem estar rotacionados e transladados. Logo, utiliza-se uma matriz de rotação contendo três vetores que descrevem a sua orientação, e um vetor de translação, referente à sua posição.

Desta forma, a matriz de rotação R do *frame* {C} em relação ao {M} é dado pelo produto escalar entre os vetores, como segue em (3.1) [35]. Lembrando dos conceitos de álgebra linear, que por se tratar de vetores unitários, tem-se $\hat{X}_C \cdot \hat{X}_M = ||\hat{X}_C||||\hat{X}_M||cos\theta = cos\theta$.

$$
{}^{M}_{C}R = [{}^{M}\hat{X}_C \quad {}^{M}\hat{Y}_C \quad {}^{M}\hat{Z}_C] = \begin{bmatrix} \hat{X}_C \cdot \hat{X}_M & \hat{Y}_C \cdot \hat{X}_M & \hat{Z}_C \cdot \hat{X}_M \\ \hat{X}_C \cdot \hat{Y}_M & \hat{Y}_C \cdot \hat{Y}_M & \hat{Z}_C \cdot \hat{Y}_M \\ \hat{X}_C \cdot \hat{Z}_M & \hat{Y}_C \cdot \hat{Z}_M & \hat{Z}_C \cdot \hat{Z}_M \end{bmatrix} \tag{3.1}
$$

Analisando a matriz de rotação mostrada acima, percebe-se que as linhas equivalem

aos vetores unitários de {M} expressados em {B}. Logo,

$$ {}^{M}_{C}R = [{}^{M}\hat{X}_C \quad {}^{M}\hat{Y}_C \quad {}^{M}\hat{Z}_C] = \begin{bmatrix} {}^{C}\hat{X}^T_M \\ {}^{C}\hat{Y}^T_M \\ {}^{C}\hat{Z}^T_M \end{bmatrix} \tag{3.2} $$

Percebe-se também que as colunas da matriz de rotação equivalem a vetores unitários e que se tratam de vetores ortogonais. Sendo assim, tem-se que a matriz de rotação ${}^{C}_{M}R$ de {M} em relação {C} é dado pela matriz de rotação transposta de {C} em relação a {M} ${}^{M}_{C}R^T$ [36] e [35],

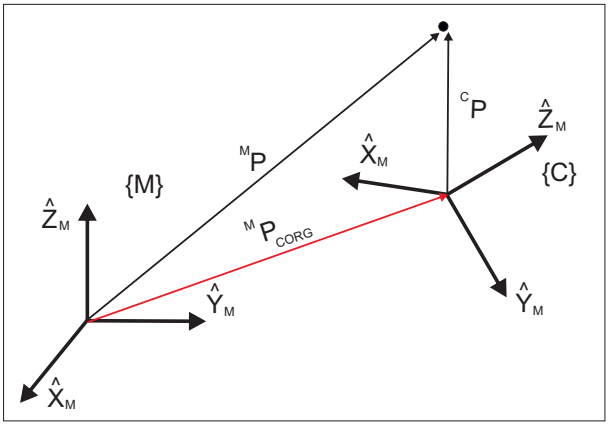
$$ {}^{M}_{C}R^T \, {}^{M}_{C}R = I_3 \tag{3.3} $$

$$ {}^{M}_{C}R = {}^{C}_{M}R^{-1} = {}^{C}_{M}R^T \tag{3.4} $$

onde I_3 é uma matriz identidade 3 x 3.

Definida a matriz de rotação, volta-se a atenção para o vetor de posição que descreve a translação de um ponto anexado na origem de um dado *frame* em relação a algum outro sistemas de coordenadas. Esse vetor nada mais é do que o deslocamento do ponto nos eixos X, Y e Z. Na Figura 35, ele é denotado por ${}^{M}P_{CORG}$.

Figura 35 – Mapeamento: Rotação + Translação



Fonte: Elaborada pelo autor

Sendo assim, descreve-se o *frame* {C} utilizando a matriz de rotação ${}^{M}_{C}R$ e o vetor de posição ${}^{M}P_{CORG}$ que fornece as coordenadas da origem do *frame* {C}:

$$ \{C\} = \{{}^{M}_{C}R, \, {}^{M}P_{CORG}\} \tag{3.5} $$

Para descrever um ponto P no referencial {M} aplica-se a equação,

$$^{M}P = {}^{M}_{C}T\ {}^{C}P \tag{3.6}$$

em que ${}^{M}_{C}T$ é formado por ${}^{M}_{C}R$ e ${}^{M}P_{CORG}$. Na forma matricial, tem-se:

$$\begin{bmatrix} {}^{M}P \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{M}R_{C} & {}^{M}P_{CORG} \\ 0\ 0\ 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{C}P \\ 1 \end{bmatrix} \tag{3.7}$$

A matriz 4x4 acima é um operador matricial chamada de transformação homogênea. Pode-se interpretá-la como sendo uma forma de descrever um dado *frame* em relação a outro, uma ferramenta para realizar o mapeamento de um ponto (${}^{M}_{C}T$ mapeia ${}^{C}P \mapsto {}^{M}P$), ou um operador de transformação (T opera em ${}^{C}P_1$ para criar ${}^{C}P_2$) [35].

Caso seja necessário encontrar a matriz inversa de uma matriz de transformação homogênea, há uma forma de realizar esta tarefa com menor esforço computacional se comparado com o método tradicional de inversão de matrizes. Considere então que deseja-se inverter a matriz ${}^{C}_{M}T$, ou seja, pretende-se encontrar a matriz ${}^{M}_{C}T$. O primeiro passo é achar a matriz de rotação inversa ${}^{M}_{C}R$. Para tal, utiliza-se a equação (3.4):

$${}^{M}_{C}R = {}^{C}_{M}R^{T}$$

Agora, é necessário descrever o vetor posição do *frame* {C} em relação ao {M} fazendo:

$$^{M}({}^{C}P_{CORG}) = {}^{M}_{C}R\ {}^{C}P_{MORG} + {}^{M}P_{CORG} \tag{3.8}$$

Uma vez que ${}^{M}({}^{C}P_{CORG})$ é igual a zero, chega-se a:

$$^{M}P_{CORG} = -{}^{M}_{C}R\ {}^{C}P_{MORG} = -{}^{C}_{M}R^{T}\ {}^{C}P_{MORG} \tag{3.9}$$

Sendo assim, monta-se a matriz de transformação homogênea inversa da seguinte maneira:

$${}^{M}_{C}T = \begin{bmatrix} {}^{C}_{M}R^{T} & -{}^{C}_{M}R^{T}\ {}^{C}P_{MORG} \\ 0\ 0\ 0 & 1 \end{bmatrix} \tag{3.10}$$

Neste trabalho a matriz de transformação homogênea será usada para descrever a posição e orientação do *frame* da câmera em relação ao sistema de referência global, fornecendo informações que alimentarão o controlador projetado. No entanto, o *frame* {M} do marcador é intermediário aos *frames* {C} e {G}. Logo, para descobrir a posição e

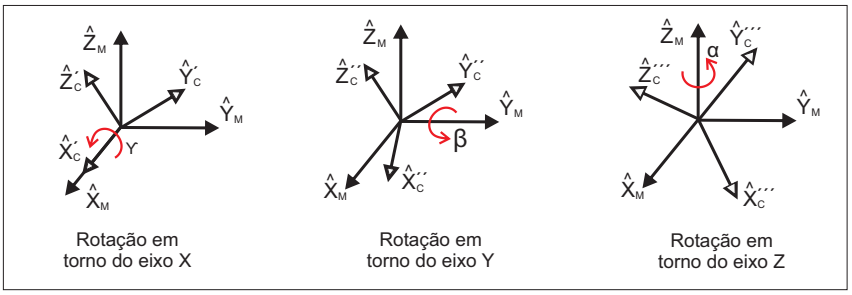
orientação global dos robôs, realiza-se a composição entre as matrizes de transformações homogêneas envolvidas, tal que:

$$ {}^{G}_{C}T = {}^{G}_{M}T\,{}^{M}_{C}T \tag{3.11} $$

3.3.1 Formas de Representar Orientações

Existem diversas formas de representar a orientação de um determinado *frame*: **ângulos fixos**, **ângulos de Euler** e todas as suas derivações, como mostrado no apêndice B da referência [35]. Porém será usado neste trabalho a descrição por **ângulos fixos X-Y-Z**, conforme mostrado na Figura 36, por ser uma forma bastante usual, sem nenhum outro motivo especial.

Figura 36 – Descrição de Rotação por ângulos fixos X-Y-Z



Fonte: Elaborada pelo autor

O processo pode ser realizado sobrepondo os dois *frames* sobre a origem e rotacionando {C} em torno de cada eixo de {M} pelos ângulos γ, β e α respectivamente. Encontra-se na literatura denominações para estes mesmos ângulos, como sendo *row, pitch e yaw*, muito conhecido nas áreas de robótica e aeronáutica, entre outras.

Realizando a multiplicação entre as matrizes de rotação nos eixos Z, Y e X respectivamente, encontra-se a matriz de rotação ${}^{M}_{C}R_{XYZ}(\gamma,\beta,\alpha)$ como mostrado na equação (3.12). Para melhorar a visualização, o cosseno foi representado por c e seno por s.

$$ {}^{M}_{C}R_{XYZ}(\gamma,\beta,\alpha) = R_Z(\alpha)R_Y(\beta)R_X(\gamma) = $$

$$ \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\gamma & -s\gamma \\ 0 & s\gamma & c\gamma \end{bmatrix} \tag{3.12} $$

Como resultado da operação em 3.12, chega-se em:

$$ {}^{M}_{C}R_{XYZ}(\gamma,\beta,\alpha) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \tag{3.13} $$

Como dito anteriormente, utiliza-se a matriz de transformação homogênea ${}^{G}_{C}T$ para encontrar a posição e orientação dos robôs em relação ao *frame* de referência global {G}. Assim, para extrair os ângulos da matriz de rotação utiliza-se os seguintes equações:

$$ \begin{aligned} \beta &= atan2(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}) \\ \alpha &= atan2(r_{21}/c\beta, r_{11}/c\beta) \\ \gamma &= atan2(r_{32}/c\beta, r_{33}/c\beta) \end{aligned} \tag{3.14} $$

Vale a pena ressaltar o uso da função *atan2(y,x)*, que calcula o arco tangente levando em consideração o sinal de x e y, sendo capaz de identificar em qual quadrante o ângulo em questão está localizado, algo de extrema importância. Note também que os índices de r equivalem às linhas e colunas respectivamente da matriz de rotação.

Observa-se também que se $\beta = \pm 90°$ torna-se impossível determinar α e γ. Uma prática adotada nestes casos é zerar α e fazer $\gamma = atan2(r_{12}, r_{22})$ para $\beta = +90°$ e $\gamma = -atan2(r_{12}, r_{22})$ para $\beta = -90°$ [35].

Pode-se também calcular os ângulos α, β e γ para a faixa de $-90° < \beta < 90°$ e $90° < \beta < 270°$, conforme mostrado em [37].

- $-90° < \beta < 90°$

$$ \begin{aligned} \alpha &= atan2(r_{21}, r_{11}) \\ \beta &= atan2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \gamma &= atan2(r_{32}, r_{33}) \end{aligned} \tag{3.15} $$

- $90° < \beta < 270°$

$$ \begin{aligned} \alpha &= atan2(-r_{21}, -r_{11}) \\ \beta &= atan2(-r_{31}, -\sqrt{r_{32}^2 + r_{33}^2}) \\ \gamma &= atan2(-r_{32}, -r_{33}) \end{aligned} \tag{3.16} $$

Outra forma de representação é conhecida como *Quaternion Unitário* e foi criada pelo matemático *sir* Willian Rowan Hamilton a mais de 150 anos, que procurava uma forma de relacionar um sistema algébrico com o espaço tridimensional. Diferentemente das representações por **Ângulos Fixos** e de **Euler** que utilizam três parâmetros, nessa forma utiliza-se quatro.

O quaternion é uma extensão dos números complexos e é constituído por duas partes: um escalar η adicionado de um vetor ϵ. É importante ressaltar que a soma dos quadrados de seus elementos sempre deve ser igual a 1, conforme a equação (3.17).

$$\mathcal{Q} = \eta + \epsilon = \eta^2 + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 = 1 \tag{3.17}$$

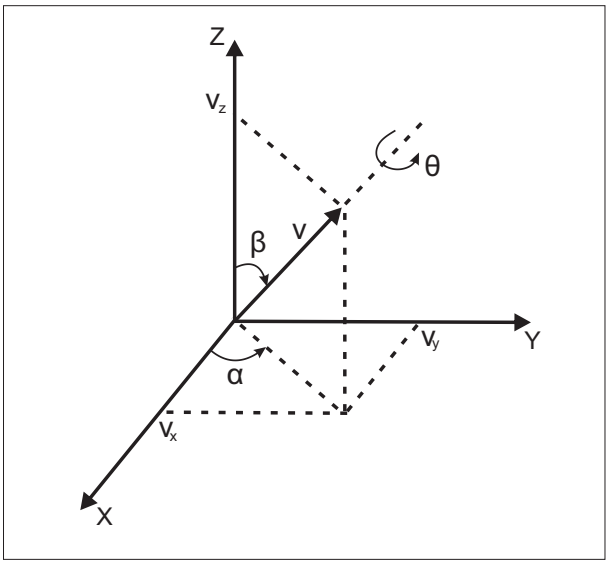
Em que,

$$\eta = cos\frac{\theta}{2} \tag{3.18}$$

$$\epsilon = sin\frac{\theta}{2}v \tag{3.19}$$

uma vez que θ é o ângulo de rotação executado sobre o vetor unitário v $= [v_x, v_y, v_z]^T$, conforme mostrado na Figura 37 abaixo.

Figura 37 – Quaternion Unitário - Rotação



Fonte: Elaborada pelo autor

Logo, para essa representação, a matriz de rotação fica definida da seguinte forma [37]:

$$\mathcal{R}(\eta, \epsilon) = \begin{bmatrix} 2(\eta^2 + \epsilon_x^2) - 1 & 2(\epsilon_x\epsilon_y - \eta\epsilon_z) & 2(\epsilon_x\epsilon_z + \eta\epsilon_y) \\ 2(\epsilon_x\epsilon_y + \eta\epsilon_z) & 2(\eta^2 + \epsilon_y^2) - 1 & 2(\epsilon_y\epsilon_z - \eta\epsilon_x) \\ 2(\epsilon_x\epsilon_z - \eta\epsilon_y) & 2(\epsilon_y\epsilon_z + \eta\epsilon_x) & 2(\eta^2 + \epsilon_z^2) - 1 \end{bmatrix} \tag{3.20}$$

Representar rotações utilizando quaternion unitário é muito comum nas áreas de robótica, visão computacional, computação gráfica, pois reduz o custo computacional se

comparada com outras formas. Isto se deve pela realização de menos operações de soma e multiplicação, sendo de grande importância para sistemas embarcados [12].

Com isso, o ROS utiliza dessas vantagens e publica no tópico *ar_pose_marker* um vetor de quatro posições contendo informações do vetor tridimensional e do escalar, conforme abordado na equação (3.17), tornando possível construir a matriz de rotação 3.20. Desta forma, pode-se extrair os ângulos de *row*, *pitch* e *yaw* com a ajuda das equações (3.14).

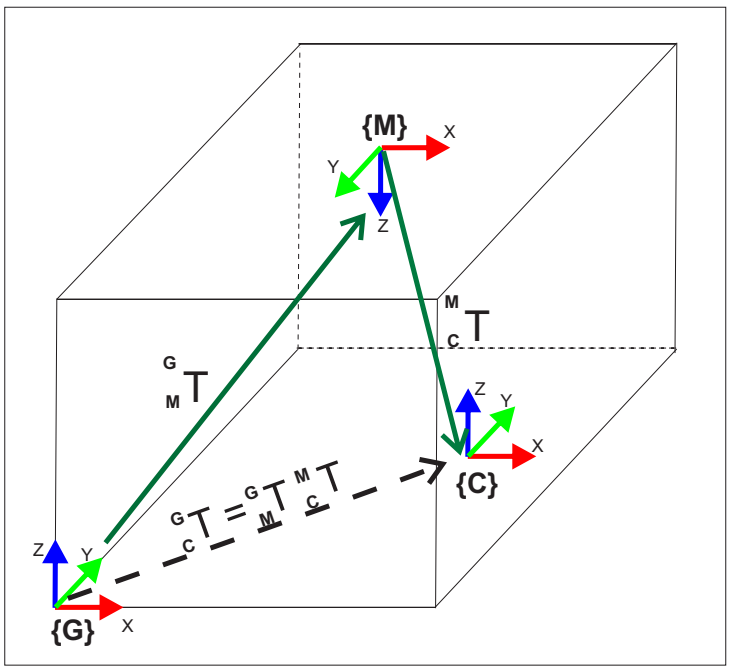
Vale a pena destacar que o conjunto formado pela câmera e robô irá apresentar rotações apenas no eixo Z, assumindo que eles são paralelos. Logo, a malha de controle irá receber apenas o ângulo α, uma vez que o robô *Pioneer P3-DX* não possui liberdade para realizar movimentos de *pitch* e *roll*.

3.3.2 **Localização do Robô**

De posse das informações abordadas nos itens anteriores e para contextualizar melhor o ambiente de trabalho, assim como a metodologia utilizada para realizar a localização dos robôs, será apresentado um exemplo de como essa tarefa foi feita.

Observe a Figura 38. Note que o sistema de referência global {G} foi fixado no canto da sala e o marcador no teto posicionado em relação ao *frame* {G} com as coordenadas $[2 \quad 2 \quad 2.5]^T$ em metros. O conjunto câmera-robô está localizado a uma distância de -0,5314 em X, -0,764 em Y e -2,1867 em Z em relação a {M}. Deseja-se encontrar as posições X,Y,Z e o ângulo que o robô está em relação ao sistema de referência global.

Figura 38 – Ambiente de Trabalho



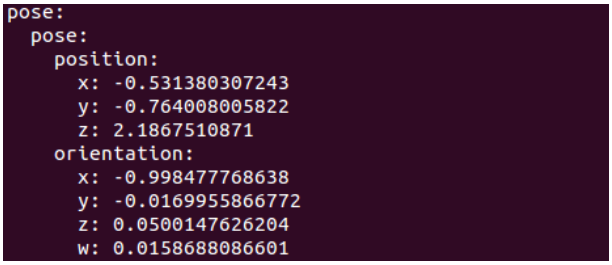
Fonte: Elaborada pelo autor

Primeiramente, encontra-se a matriz de transformação homogênea ${}^{M}_{C}T$. Para isso, utiliza-se as informações fornecidas pelo tópico *ar_pose_marker*. Porém, esses dados são relativos ao *frame* {M} para o {C}, sendo necessário posteriormente inverter a matriz.

Calcula-se então a matriz de rotação definida em (3.20) e encontra-se o vetor posição de acordo com os valores mostrados na Figura 39, uma vez que:

- $\epsilon_x = orientation.x$
- $\epsilon_y = orientation.y$
- $\epsilon_z = orientation.z$
- $\eta = orientation.w$

Figura 39 – Dados de Posição e Orientação publicados pelo Nó *ar_pose*



Fonte: Elaborada pelo autor

Utilizando a equação (3.7), encontra-se a matriz de transformação homogênea ${}^{C}_{M}T$:

$$ {}^{C}_{M}T = \begin{bmatrix} 0.9944 & 0.0324 & -0.1004 & -0.5314 \\ 0.0355 & -0.9989 & 0.03 & -0.7640 \\ -0.0993 & -0.0334 & -0.9945 & 2.1868 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.21} $$

Inverte-se a matriz acima conforme mostrado na equação (3.10). Desta forma, chega-se em:

$$ {}^{M}_{C}T = \begin{bmatrix} 0.9944 & 0.0355 & -0.0993 & 0.7728 \\ 0.0324 & -0.9989 & -0.0334 & -0.6730 \\ -0.1004 & 0.0300 & -0.9945 & 2.1443 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22} $$

O passo seguinte é encontrar a matriz de transformação homogênea do marcador em relação ao *frame* de coordenadas global {G}. Sabendo que o marcador está rotacionado

apenas no eixo X de 180°, utiliza-se somente a matriz de rotação $R\_x(\gamma)$ mostrada em (3.13). Desta forma, tem-se:

$$ {}^{G}_{M}T = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & -1 & 0 & 2 \\ 0 & 0 & -1 & 2.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.23} $$

De posse das matrizes ${}^{G}_{M}T$ e ${}^{M}_{C}T$, basta multiplicá-las conforme mostrado em (3.11) para obter a matriz de transformação homogênea da câmera {C} em relação ao *frame* de referência global {G}.

$$ {}^{G}_{C}T = {}^{G}_{M}T\,{}^{M}_{C}T = \begin{bmatrix} 0.9944 & 0.0355 & -0.0993 & 2.7728 \\ -0.0324 & 0.9989 & 0.0334 & 2.6730 \\ 0.1004 & -0.0300 & 0.9945 & 0.3557 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{3.24} $$

Aplicando as equações (3.14), (3.15) e (3.16) é possível encontrar os ângulos. Desta forma, finalmente tem-se todas as informações relativas a posição e orientação da câmera em relação ao *frame* de referência global.

$$ Pose(x, y, z) = \begin{bmatrix} 2.7728 & 2.6730 & 0.3557 \end{bmatrix} \tag{3.25} $$

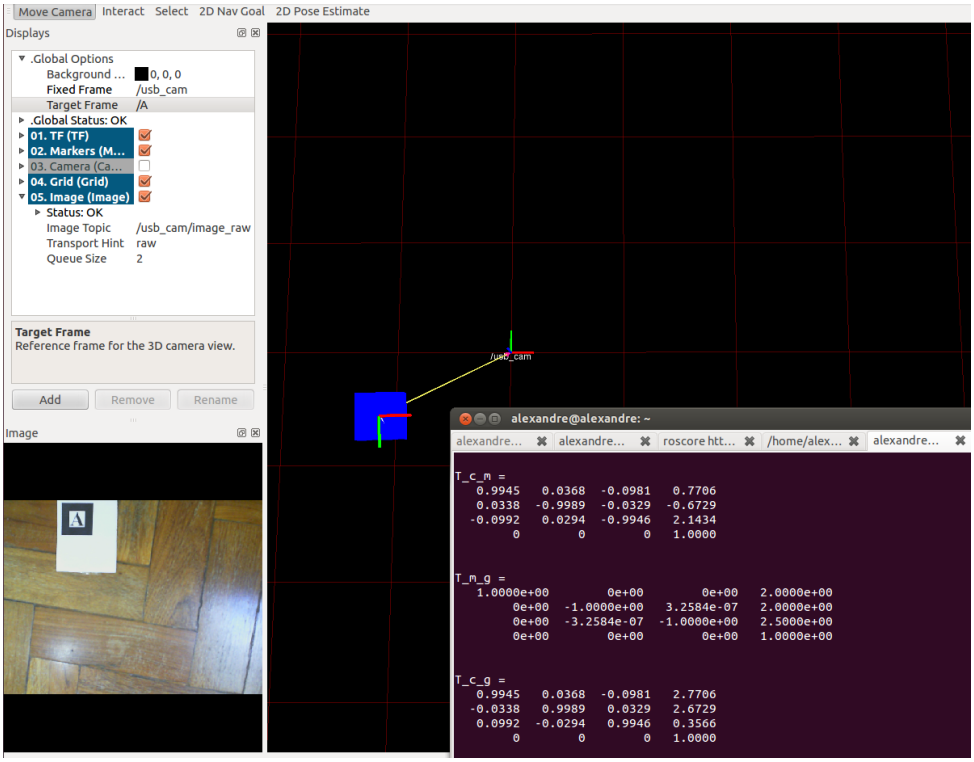
$$ Orientacao(\alpha, \beta, \gamma) = \begin{bmatrix} -1.8634° & -5.7633° & -1.7273° \end{bmatrix} \tag{3.26} $$

É importante salientar que será usado somente o ângulo α para medir o erro de direção do robô durante a navegação, uma vez que ele apresenta rotações somente no eixo Z. O motivo dos outros ângulos não estarem zerados está relacionado a erros de posicionamento da câmera. Desta forma, devido a pequenas inclinações, β e γ são diferentes de zero.

A Figura 40 mostra a saída do programa que calcula a posição e orientação do robô em relação ao *frame* de referência global assim com o sistema de visão em funcionamento. Quando um marcador é detectado, é construído no programa *rviz* um modelo tridimensional representando o objeto (mostrado em azul na figura) e o *frames* envolvidos, no caso o do marcador e da câmera assim como uma seta representando a relação entre os dois sistemas.

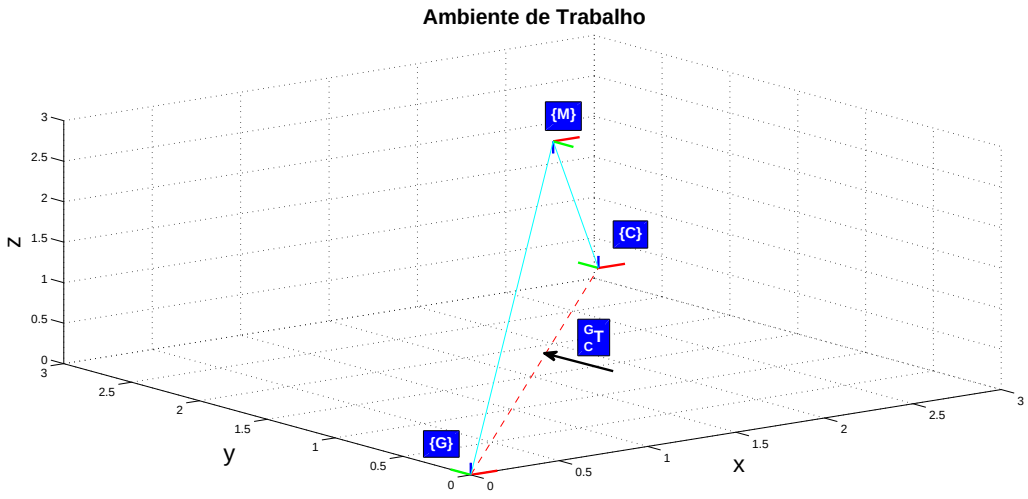
A Figura 41 ilustra o ambiente de trabalho no qual o robô realiza a tarefa. Note que a linha contínua representa as matrizes de transformação homogêneas utilizadas para obter a relação entre o robô e o *frame* de referência global destacada pela linha tracejada, possibilitando encontrar a sua posição e orientação.

Figura 40 – Sistema de Visão e cálculo das matrizes de transformação homogêneas



Fonte: Elaborada pelo autor

Figura 41 – Representação do Ambiente e relação entre os Frames



Fonte: Elaborada pelo autor

É importante destacar que a câmera anexada ao robô não consegue visualizar todo o ambiente de trabalho de uma só vez. Logo, para contornar esse problema vários marcadores foram anexados ao teto de forma que em qualquer lugar o robô possa ver no mínimo dois marcadores.

Foi utilizada essa redundância como medida de segurança, pois pode ocorrer de em algum momento um marcador não ser detectado pelo sistema de visão, o que poderia

causar algum acidente ou inviabilizar a aplicação, uma vez que os robôs devem seguir os caminhos planejados de forma precisa e contínua. Visualizando mais de um marcador a todo o tempo a possibilidade dessa falha ocorrer diminui.

3.4 NAVEGAÇÃO DO ROBÔ

Nesta seção será abordada a estratégia e a malha de controle utilizada para realizar a navegação dos robôs pelo ambiente de trabalho.

3.4.1 **O Robô Pioneer**

O Pioneer P3DX é um robô fabricado pela *Mobile Robots*. É um dos mais conhecidos e utilizados para pesquisa na área de robótica. Conta com diversos tipos de sensores, como *encoders* e sonares, que por padrão apresenta 8 deles na parte dianteira e opcionalmente pode-se inserir mais 8 na traseira. A Figura 42 mostra dois dos cinco *pioneers* que o laboratório de robótica da UFJF possui.

Figura 42 – Pioneer P3DX

Fonte: Elaborada pelo autor

O robô possui embarcado um controlador de movimento de baixo nível que realiza o controle de velocidade das rodas e estima posição x, y e o ângulo θ do robô, além de fornecer o *status* da bateria e leituras de distância realizadas pelos sonares.

Seu sistema de tração é diferencial. Logo possui dois motores que controlam de forma independente o movimento de cada uma das rodas. Ele consegue atingir velocidade linear e angular de até $1,6\,m/s$ e $300°/s$ respectivamente, além de suportar carga de até 23 Kg. Pode ser alimentado por 3 baterias de 12V, que fornecem autonomia de aproximadamente 10 horas [13].

3.4.2 Malha de Controle

A malha de controle utilizada no presente trabalho foi desenvolvida baseando-se numa estratégia simples porém efetiva denominada *Following a Path* [12], que permite o robô seguir um conjunto de pontos objetivos (coordenadas x^*,$y*$) ao longo do caminho.

A malha apresentada em [12] apresenta dois controladores: um atua na distância entre a posição atual do robô e o ponto objetivo, estabelecendo a relação de que quanto maior for esta distância, maior será a velocidade do robô; e o outro controlador age orientando-o em direção ao alvo, conforme mostrado na Figura 43.

Como o robô mantém uma distância d^* para o ponto objetivo, o erro fica definido como:

$$e = \sqrt{(x^* - x)^2 + (y^* - y)^2} - d^* \tag{3.27}$$

onde x^*,y^* são coordenadas dos pontos objetivos e x e y são coordenadas da posição atual do robô.

O erro é corrigido por um controlador Proporcional-Integral (PI) que atua na velocidade do robô.

$$v^* = K_v e + K_i \int e \mathrm{d}t \tag{3.28}$$

Utiliza-se um controlador proporcional para direcionar o robô para o ponto objetivo, atuando no ângulo:

$$\alpha = K_h(\theta^* \ominus \theta), K_h > 0 \tag{3.29}$$

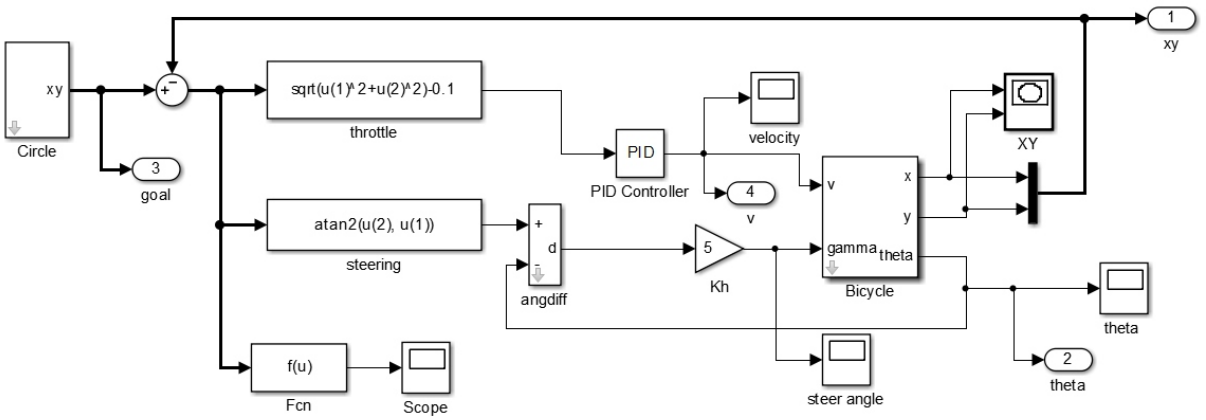
onde o ângulo objetivo é:

$$\theta^* = tan^{-1}(\tfrac{y^* - y}{x* - x}) \tag{3.30}$$

No entanto, conforme definido na seção 1.2, deseja-se que o robô mantenha uma velocidade específica para cada um dos trechos do circuito. O perfil de velocidade é definido pelo controlador central, no qual passa por processo de otimização a fim de encontrar o tempo que os robôs devem percorrer cada subcaminho do percurso, de forma a maximizar o menor intervalo de tempo Δt, conforme descrito no 2.

Portanto, efetuou-se uma modificação na malha da Figura 43 de forma que a velocidade linear do robô não estivesse mais função do erro entre distância atual do robô e o ponto objetivo, mas recebesse o valor definido pelo planejador central.

Figura 43 – Diagrama de Blocos - Malha de Controle

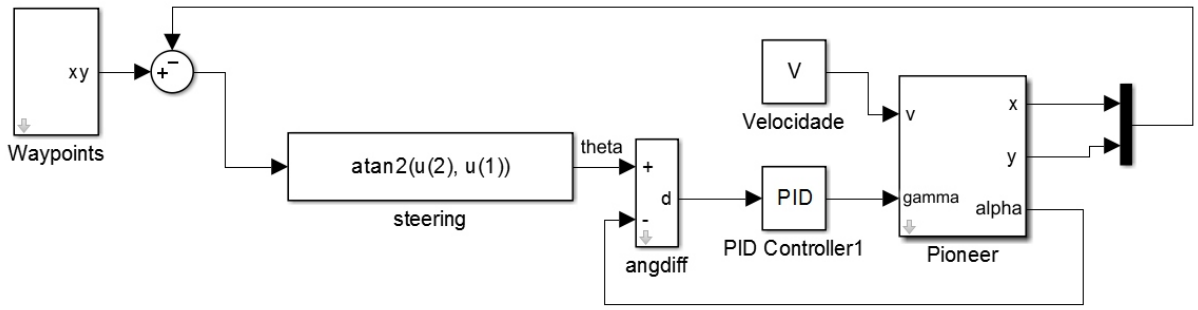


Fonte: [12]

Como a velocidade média do robô pode mudar de acordo com o trecho do circuito em que ele está percorrendo, coube ao programa desenvolvido verificar em qual posição do mapa o robô está e informar a malha de controle qual deve ser a sua velocidade. Vale ressaltar que neste trabalho não foi abordado o controlador de baixo nível que obriga o robô a atingir as velocidades linear e angular desejadas. Foi utilizado o que já vem embarcado no próprio *pioneer*.

Desta forma, a malha de controle ficou reduzida apenas em orientar o robô na direção do ponto objetivo, sendo usado somente um controlador que atua no ângulo. Outra modificação foi a substituição do controlador proporcional P presente na malha original, por um controlador Proporcional-Integral-Derivativo (PID), a fim de possibilitar a atuação na parte transitória e permanente da resposta. A Figura 44 mostra como ficou a malha de controle adotada no presente trabalho.

Figura 44 – Diagrama de Blocos - Malha de Controle



Fonte: Elaborada pelo autor

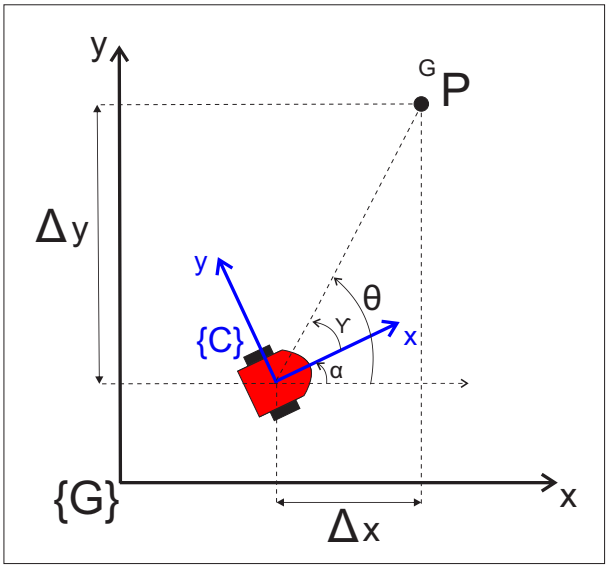
O bloco denominado *steering* calcula o ângulo θ entre as coordenadas x e y da posição atual do robô dada pela matriz de transformação homogênea ${}^{M}_{C}T$ e o ponto objetivo

${}^{G}P$.

$$\theta = tan^{-1}(\frac{\Delta y}{\Delta x}) \tag{3.31}$$

Posteriormente o bloco *angdiff* realiza a subtração entre o ângulo objetivo e o atual do robô em relação *frame* {G}, resultando efetivamente no deslocamento angular que o robô deve realizar para se orientar em direção ao ponto objetivo, conforme mostrado na Figura 45.

Figura 45 – Orientação Robô - Alvo



Fonte: Elaborada pelo autor

3.5 CONCLUSÃO

Este capítulo teve o objetivo de descrever a metodologia e as ferramentas utilizadas para realizar a tarefa de navegação dos robôs por determinados caminhos enquanto eles evitam colisões entre si, utilizando a técnica mostrada no capítulo 2.

Para tal, conclui-se que *framework* ROS é de fundamental importância no trabalho apresentado, pois apresenta ferramentas funcionais e algoritmos que facilitam o desenvolvimento de aplicações. Através do pacote *ROSARIA* realiza-se a comunicação com o robô, permitindo o envio das velocidades linear e angular e a leitura de dados, essenciais para o desenvolvimento da aplicação.

Destaca-se também o pacote *ARPOSE* que realiza a detecção dos marcadores, conforme mostrado na Figura 31 e calcula a sua posição e orientação em relação ao *frame* de coordenadas da câmera. Logo, possibilita através da composição das matrizes de transformações homogêneas ${}^{G}_{M}T$ e ${}^{M}_{C}T$ encontrar ${}^{G}_{C}T$, que carrega informações sobre posição e orientação do robô em relação ao sistema de referência global, possibilitando

extrair dados que permitem localizar o robô no espaço de trabalho, como mostrado na seção 3.3.2.

4 RESULTADOS E DISCUSSÕES

Um estudo de casos é realizado neste capítulo a fim de aplicar e validar a técnica de otimização descrita no capítulo 2. Logo, esta etapa tem como foco principal realizar análises e discutir sobre resultados obtidos através de simulações e do experimento prático.

Para realizar as simulações utilizou-se um pacote do ROS chamado *STAGEROS*. Trata-se de um simulador 2D simples, que permite a criação de mapas, obstáculos e robôs através de um arquivo *.world* que os descreve. Esse pacote fornece algumas das funcionalidades proporcionadas pela biblioteca *libstage* via ROS. É possível simular também sensores como *laser* e hodômetro, que publicam os valores de suas leituras em tópicos no ROS [29].

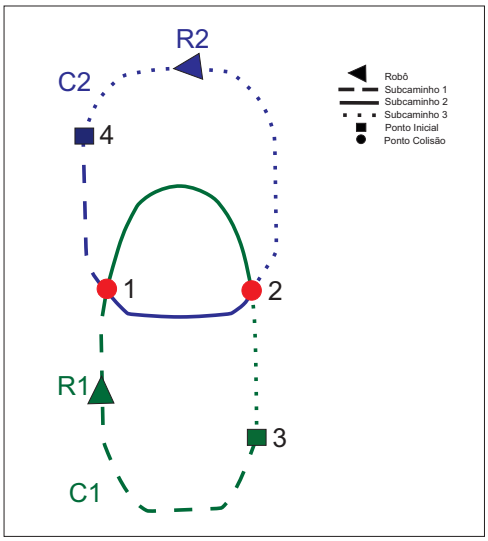
É importante destacar que para as simulações realizadas, utilizou-se as leituras de hodometria publicadas no tópico */odom* para localizar o robô, ao invés das informações de localização fornecidas pelo pacote *ARPOSE*, utilizado somente em casos reais.

4.1 CASO 1 - ROBÔS NO MESMO CICLO DE TRABALHO

Este estudo de caso foi dividido em duas partes, no qual a primeira apresenta resultados obtidos através de simulação e a segunda fornece dados experimentais. Porém em ambos, o tamanho de cada subcaminho é o mesmo, significando que a distância percorrida pelo robô 1 na simulação é a mesma que o robô 1 no teste prático. O mesmo vale para o robô 2. Portanto, foi montado apenas um problema de otimização, que forneceu o perfil de velocidade média ao longo dos caminhos para as duas partes. Na Seção 4.3, será feita a comparação entre esses resultados.

Considere um ambiente composto por dois caminhos que se interceptam nos pontos indicados por um círculo, conforme mostrado na Figura 46. Ambos os percursos são divididos em três subcaminhos, como indicado na mesma figura. Em cada um desses trechos os robôs irão se movimentar com uma determinada velocidade, planejada pelo controlador central a fim de maximizar o intervalo de tempo que eles cruzam por um mesmo ponto de colisão.

Figura 46 – Caminhos percorridos pelos Robôs



Fonte: Elaborada pelo autor

Os caminhos C1 e C2 possuem respectivamente 10,8 e 9,02 metros e as velocidades mínimas e máximas para cada robô são de $0,2m/s$ e $0,6m/s$. São utilizados γ_1 e γ_2 iguais a 1, significando que o tempo que os robôs completarão cada uma de suas voltas será o mesmo.

Levando em consideração o tamanho de cada percurso e as velocidades mínimas e máximas dos robôs, encontra-se o tempo máximo e mínimo que eles podem completar cada volta. Desta forma, define-se a base de tempo dentro deste intervalo, fazendo $T = 25$. Logo, pela equação 2.10 tem-se que cada ciclo de trabalho, definido com sendo uma volta completa pelo circuito, deve ser completado em $T_i = 25s$.

Então, de acordo como mostrado na Figura 46, tem-se o seguinte mapa de pontos para o teste prático em questão, onde P3 e P4 são pontos de partida de cada robô.

- Robô 1: $P1 \rightarrow P2 \rightarrow P3 \rightarrow P1$
- Robô 2: $P1 \rightarrow P2 \rightarrow P4 \rightarrow P1$

Enviando todas essas informações para o programa em C++ que irá executar o modelo desenvolvido no *LINGO* de acordo como abordado no item 2.2.4, obtém-se um $\Delta t = 12,5s$. Ou seja, considerando todos os pontos de colisão, o menor tempo entre a passagem de dois robôs sobre um mesmo ponto é de 12,5s. Os demais valores são mostrados na Tabela 12, em que a segunda e a terceira colunas indicam o tempo gasto pelos robôs para percorrer o caminho entre o ponto inicial e o ponto em questão e a última coluna mostra o intervalo de tempo entre a passagem dos robôs por um mesmo ponto de colisão. Logo verifica-se que que nenhum deles está abaixo de Δt, respeitando as restrições (2.16) e (2.17) de limite de margem de tempo.

Tabela 12 – Tempo em que cada robô cruza pelos pontos de colisão

PONTOS	TEMPO(s)		INTERVALO DE TEMPO
	R1	R2	
P1	15,8	3,2	12,5
P2	21,6	9,1	12,5

Fonte: Elaborada pelo autor

A Tabela 13 apresenta os valores de tempo, distância e velocidade para cada trecho do circuito. É importante notar que a soma dos tempos para os subcaminhos de cada circuito é igual a 25, que foi o valor adotado para o tempo total de conclusão de uma volta.

Tabela 13 – Velocidade, Distância e Tempo

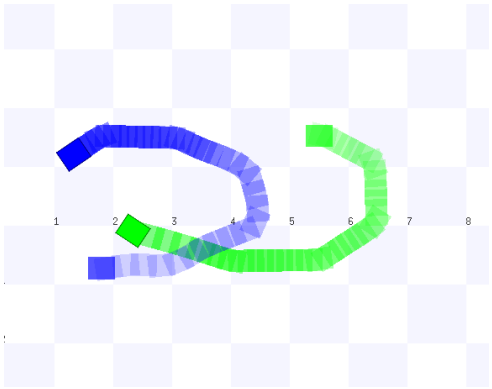
TRECHOS DOS CIRCUITOS	TEMPO(s)		DISTÂNCIA(m)		VELOCIDADES	
	R1	R2	C1	C2	R1	R2
SUBCAMINHO 1	15,8	3,2	5,29	1,95	0.33	0.60
SUBCAMINHO 2	5,8	5,9	3,5	2,84	0.60	0.48
SUBCAMINHO 3	3,4	15,9	2,01	4,23	0.59	0.27

Fonte: Elaborada pelo autor

4.1.1 Simulação do Experimento Prático - - StageROS

De posse das informações de distâncias dos subcaminhos exibidas na Tabela 13, criou-se dois circuitos com as mesmas características, de acordo com a Figura 47.

Figura 47 – Caminhos percorridos pelos Robôs - Simulação do Caso Prático

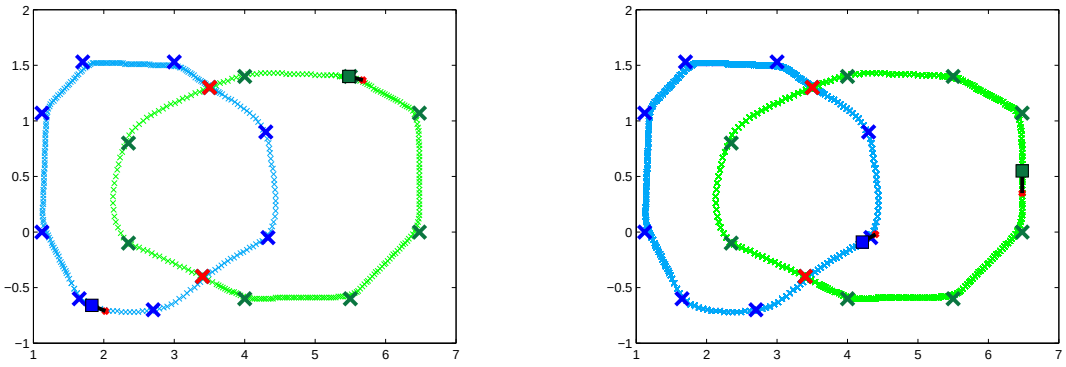


Fonte: Elaborada pelo autor

Tendo como base os tempos e velocidades em que cada robô deve atravessar os subcaminhos, executou-se o programa de navegação. Após a tarefa concluída, utilizou-se o *MATLAB* para plotar os pontos percorridos pelos robôs, mostrados na Figura 48 e os

tempos de percurso em cada trecho dos circuitos, conforme ilustrado nas Figuras 49 e 50. Para facilitar a visualização plotou-se apenas 30 ciclos.

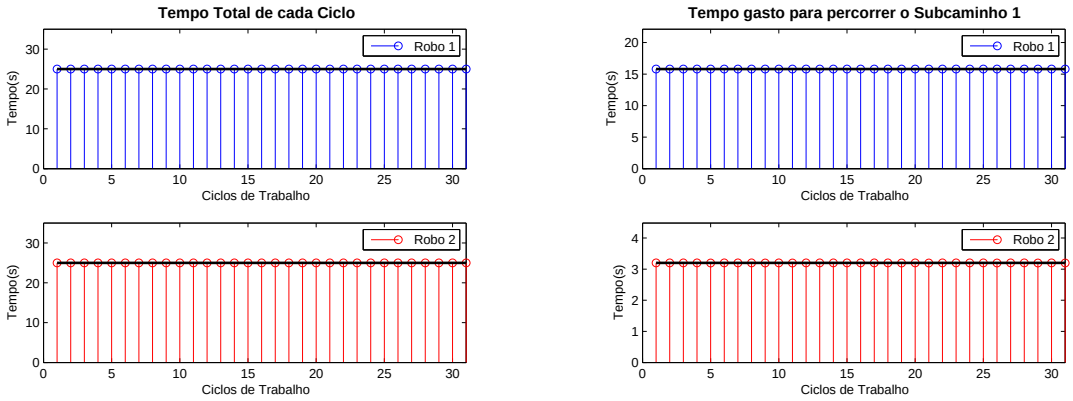
Figura 48 – Caminho Percorrido pelos Robôs



(a) Após o primeiro ciclo: *(b) Após 500 Ciclos*:

Fonte: Elaborada pelo autor

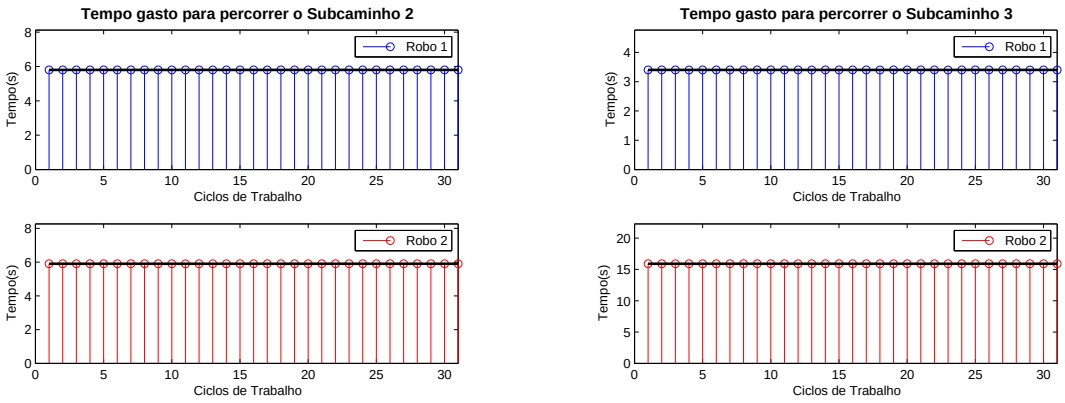
Figura 49 – Tempos envolvidos na Tarefa - Simulação do Caso Prático



(a) Tempo Total de cada Ciclo de Trabalho: *(b) Tempo de Percurso Subcaminho 1*:

Fonte: Elaborada pelo autor

Figura 50 – Tempos de Percurso



(a) Tempo de Percurso - Subcaminho 2: *(b) Tempo de Percurso - Subcaminho 3*:
Fonte: Elaborada pelo autor

Pela Tabela 14 nota-se que a média dos tempos realizada para todos os ciclos de trabalho em cada trecho do circuito é igual aos tempos mostrados na Tabela 13 definidos durante o planejamento. A última linha da Tabela 14 refere-se à média do tempo de um ciclo completo e à média de velocidade exercida durante todo o percurso.

Tabela 14 – Média dos Tempos e Velocidades - Simulação Caso Prático

TRECHOS DOS CIRCUITOS	TEMPO(s)		VELOCIDADE (m/s)	
	R1	**R2**	**R1**	**R2**
SUBCAMINHO 1	15,8	3,2	0,33	0,60
SUBCAMINHO 2	5,8	5,9	0,60	0,48
SUBCAMINHO 3	3,4	15,9	0,59	0,27
CICLO	25	25	0,432	0,361

Fonte: Elaborada pelo autor

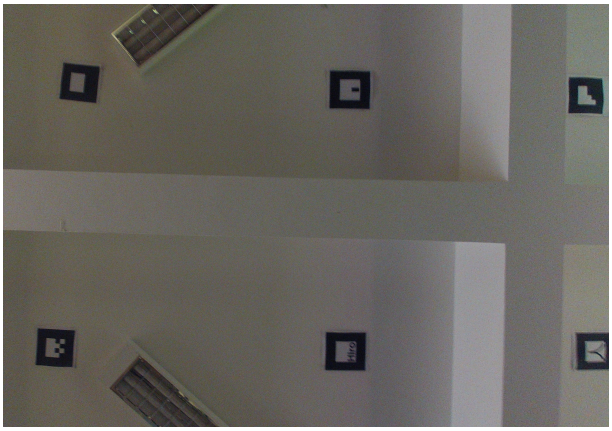
4.1.2 **Experimento Prático**

Considere um espaço conforme mostrado na Figura 51. Neste ambiente há diversos marcadores fixados no teto conforme mostrado na Figura 52, de tal forma que o campo de visão da câmera consiga visualizar no mínimo dois marcadores em qualquer lugar que o robô esteja no espaço de trabalho.

Figura 51 – Espaço de Trabalho

Fonte: Elaborada pelo autor

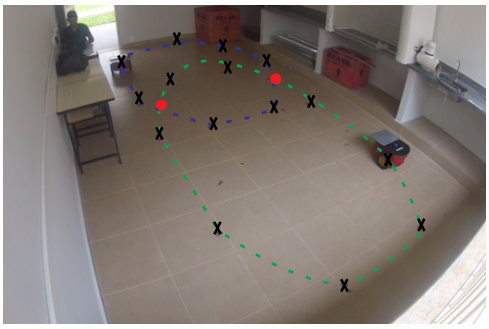
Figura 52 – Marcadores Fixados no Teto



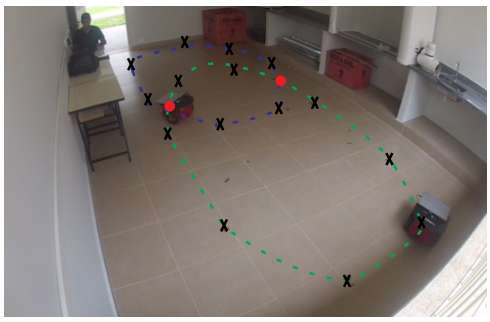
Fonte: Elaborada pelo autor

Uma vez planejado na Seção 4.1 o perfil de velocidade média que os robôs terão em cada trecho dos circuitos, inicia-se o programa que irá navegar os robôs pelos waypoints indicados por um X, conforme mostrado na Figura 53.

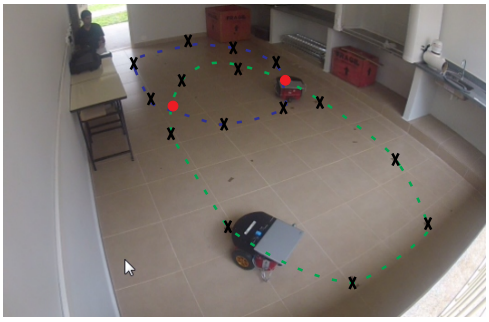
Figura 53 – Caminhos percorridos pelos Robôs



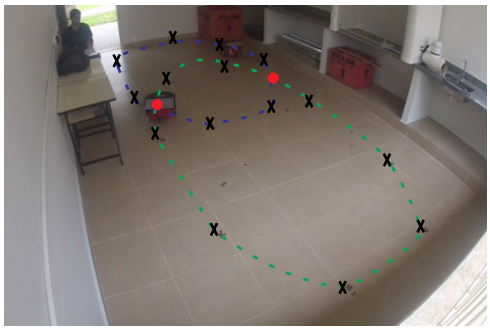
(a) Início do Ciclo de Trabalho: Posição Inicial



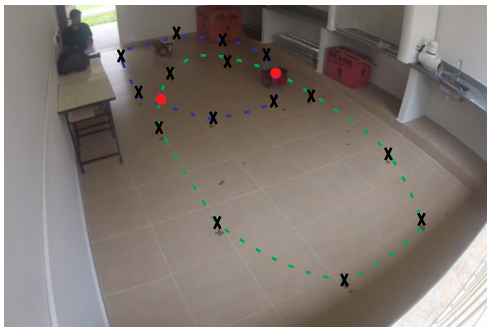
(b) Robô 2 sobre o Ponto Colisão 1: Tempo ≈ 3,338s



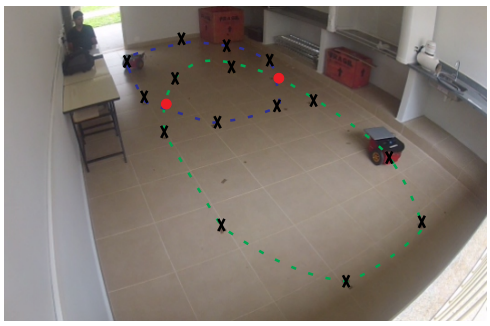
(c) Robô 2 sobre o Ponto Colisão 2: Tempo ≈ 9,18s



(d) Robô 1 sobre o Ponto Colisão 1: Tempo ≈ 15,77s



(e) Robô 1 sobre o Ponto Colisão 2: Tempo ≈ 21,61s

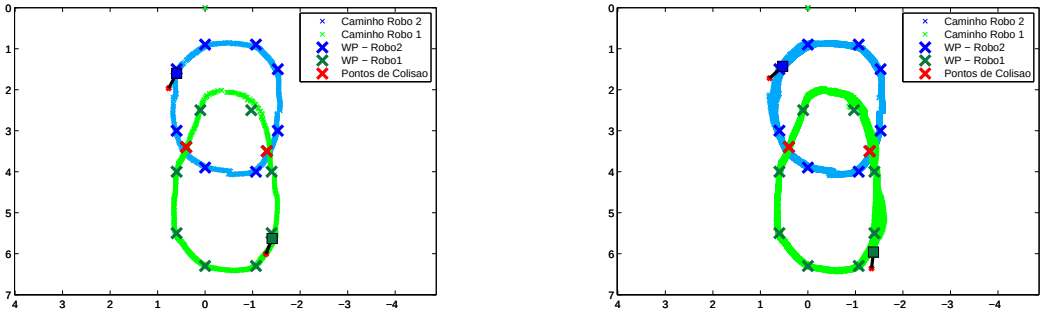


(f) Término do Ciclo de Trabalho: Tempo ≈ 24,93

Fonte: Elaborada pelo autor

Durante o percurso, alguns dados importantes foram sendo armazenados, como posição e orientação dos robôs em relação ao sistema de referência global {G}. Portanto, plota-se essas informações na Figura 54, de forma a reproduzir o caminho feito pelos robôs.

Figura 54 – Caminho Percorrido pelos Robôs

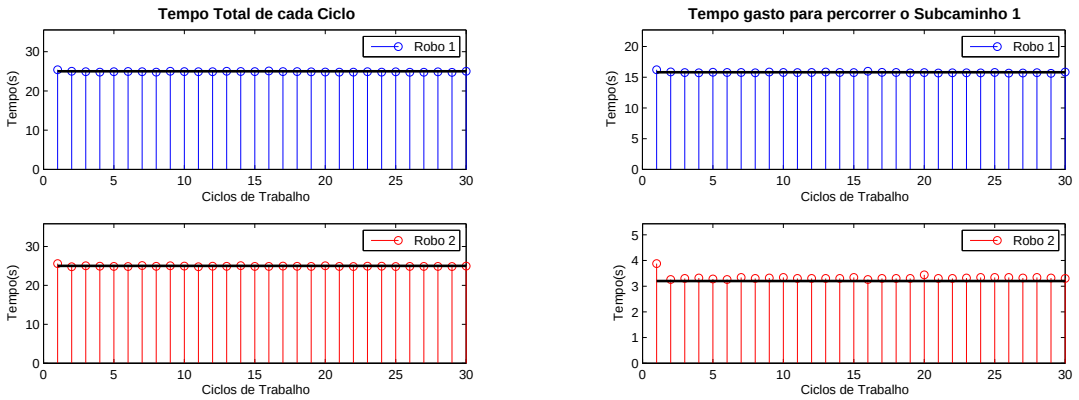


(a) Após o primeiro ciclo: *(b) Após 30 Ciclos*:

Fonte: Elaborada pelo autor

Os tempos de conclusão de cada ciclo de trabalho, assim como os tempos gastos para percorrer cada trecho do circuito também são plotados nas Figuras 55 e 56, sendo que os gráficos da parte superior das figuras são referentes ao robô 1 e os mostrados na parte inferior ao robô 2.

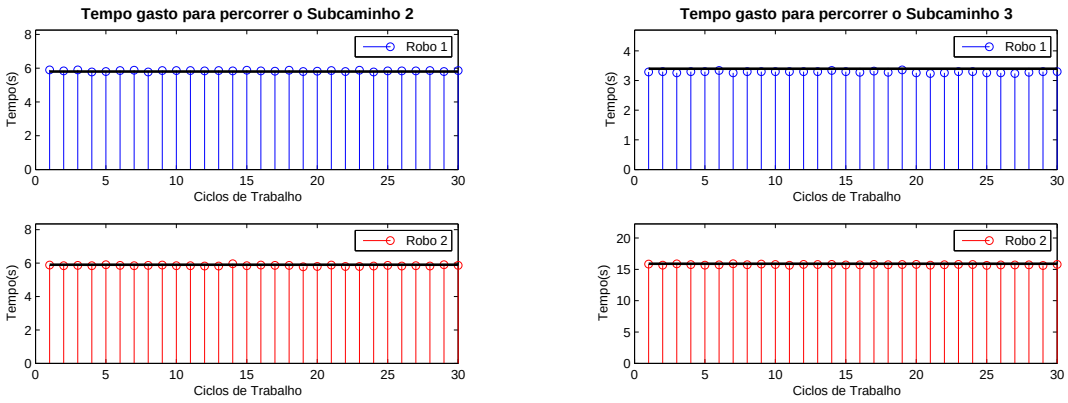
Figura 55 – Tempos envolvidos na Tarefa



(a) Tempo Total de cada Ciclo de Trabalho: *(b) Tempo de Percurso Subcaminho 1*:

Fonte: Elaborada pelo autor

Figura 56 – Tempos de Percurso



(a) Tempo de Percurso - Subcaminho 2: *(b) Tempo de Percurso - Subcaminho 3*:
Fonte: Elaborada pelo autor

A Tabela 15 mostra as informações de tempos e velocidades dos robôs para trecho do circuito. É possível perceber que a média dos tempos das leituras em relação a todos os ciclos de trabalho se aproxima muito dos valores planejados pelo controlador central.

Tabela 15 – Média dos Tempos e Velocidades - Experimento Prático

TRECHOS DOS CIRCUITOS	TEMPO(s)		VELOCIDADE (m/s)	
	R1	R2	R1	R2
SUBCAMINHO 1	15,769	3,338	0.33	0.60
SUBCAMINHO 2	5,842	5,848	0.60	0.48
SUBCAMINHO 3	3,291	15.753	0.59	0.27
CICLO	24.902	24.933	0.434	0.362

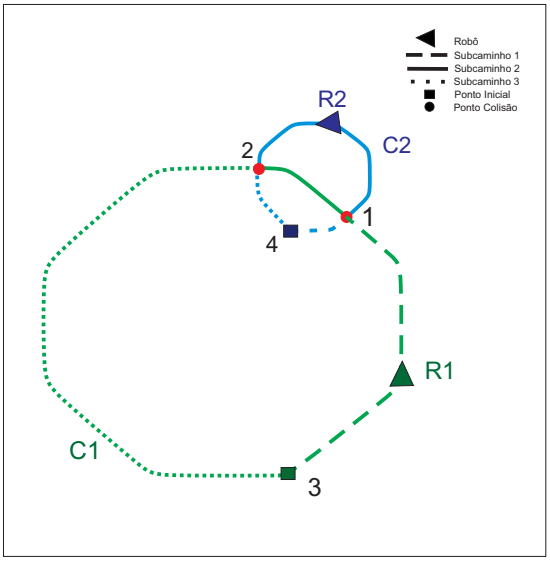
Fonte: Elaborada pelo autor

O vídeo referente ao teste prático realizado pode ser acessado através do link: https://www.dropbox.com/s/4uwk6tntpig7zqx/Experimento.mp4?dl=0

4.2 CASO 2 - ROBÔS EM DIFERENTES CICLOS DE TRABALHO

O objetivo deste estudo de caso é demonstrar o controle das velocidades médias dos robôs, adotando diferentes valores de γ para cada um deles, de forma que um robô não precise estar necessariamente no mesmo ciclo que os demais durante a realização da tarefa. Os caminhos percorridos pelos robôs seguem o esboço ilustrado na Figura 57. Ambos os percursos são divididos em três subcaminhos.

Figura 57 – Caminhos percorridos pelos Robôs - Caso 2



Fonte: Elaborada pelo autor

O comprimento do caminho $C1$ é igual a $60m$, 3 vezes maior do que o outro. Considera-se então $\gamma_1 = 3$, fazendo com que o robô 1 de acordo com a equação 2.10 complete um ciclo em 120 segundos, e $\gamma_2 = 1$ com robô 2 levando um tempo de $T_i = 40s$ por ciclo. O tempo base é $T = 40$ segundos. Foi definido também que as velocidades dos robôs mínimas e máximas são respectivamente de $0,2m/s$ e $1m/s$.

Então, de acordo como mostrado na Figura 57, tem-se o seguinte mapa de pontos para o caso em estudo, lembrando que P3 e P4 são pontos iniciais de cada robô.

- Robô 1: $P1 \rightarrow P2 \rightarrow P3 \rightarrow P1$
- Robô 2: $P1 \rightarrow P2 \rightarrow P4 \rightarrow P1$

Executando o modelo de otimização desenvolvido no *LINGO* com as informações descritas nesta seção, obtém-se como valor da função objetivo um Δt de 20s. Os demais valores são mostrados na Tabela 12, em que a segunda e a terceira colunas indicam o tempo gasto pelos robôs para percorrer o caminho entre o ponto inicial e o ponto em questão e a última coluna mostra o intervalo de tempo entre a passagem dos robôs por um mesmo ponto de colisão.

Tabela 16 – Tempo em que cada robô cruza pelos pontos

PONTOS	TEMPO(s)		INTERVALO DE TEMPO
	R1	R2	
P1	24	4	20
P2	54.4	34.4	20

Fonte: Elaborada pelo autor

Logo verifica-se que que nenhum deles está abaixo de Δt, respeitando as restrições 2.16 e 2.17 de limite de margem de tempo.

A Tabela 17 apresenta os valores de tempo, distância e velocidade para cada trecho do circuito. É importante notar que a soma dos tempos dos subcaminhos do circuito 1 é igual a 120 e para o circuito 2 é igual a 40, conforme definido no início da seção.

Tabela 17 – Tempo, Distância e Velocidade

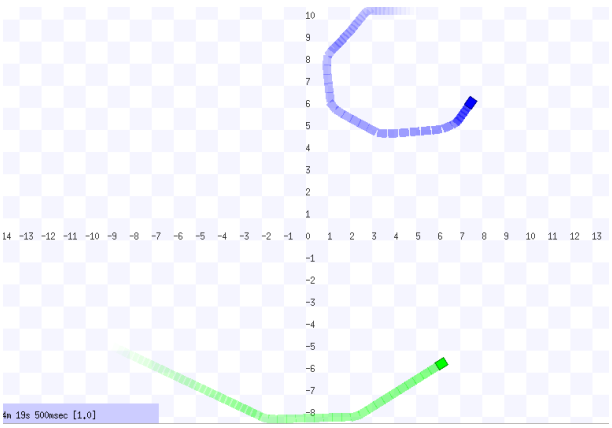
TRECHOS DOS CIRCUITOS	TEMPO(s)		DISTÂNCIA(m)		VELOCIDADES	
	R1	R2	C1	C2	R1	R2
SUBCAMINHO 1	24.0	4.0	20	3.98	0.833	0.995
SUBCAMINHO 2	30.4	30.4	6.08	11.36	0.2	0.374
SUBCAMINHO 3	65.6	5.6	33.92	4.66	0.517	0.832

Fonte: Elaborada pelo autor

4.2.1 **Simulação do Caso 2 - StageROS**

Da mesma forma que no caso anterior, simulou-se o estudo de caso no *StageROS*, tendo como base os tempos e velocidades em que cada robô deve atravessar os subcaminhos. A Figura 58 mostra os robôs executando a tarefa.

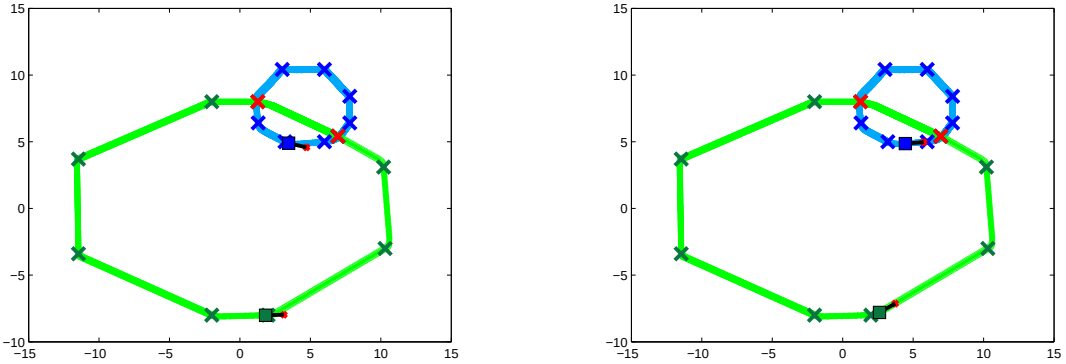
Figura 58 – Caminhos percorridos pelos Robôs - Simulação do Caso 2



Fonte: Elaborada pelo autor

Ao fim da missão, plotou-se no matlab os pontos percorridos pelos robôs, mostrados na Figura 59 e os tempos de percurso em cada trecho dos circuitos, conforme ilustrado nas Figuras 60 e 61. Vale destacar que o robô 2 conseguia percorrer 3 vezes o seu caminho ao passo de que o robô 1 percorria apenas um ciclo no mesmo intervalo de tempo.

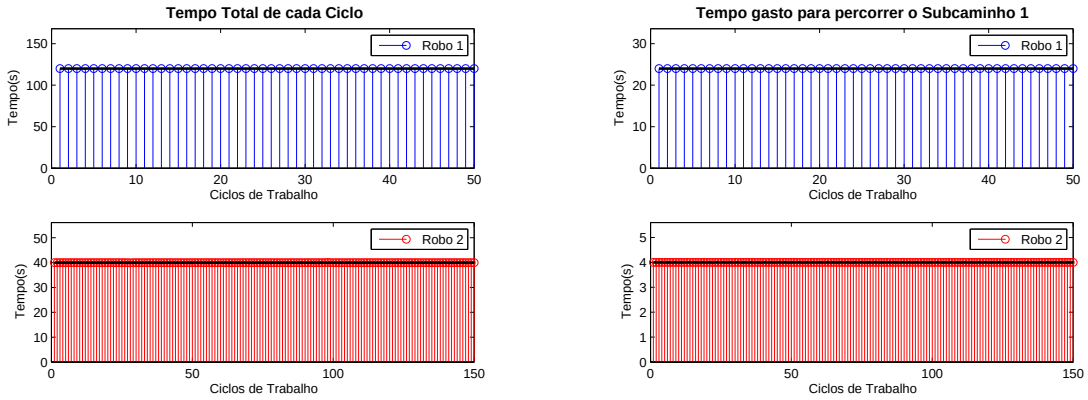
Figura 59 – Caminho Percorrido pelos Robôs - Plot Matlab



(a) Após o primeiro ciclo: *(b) Após o Último Ciclo*:

Fonte: Elaborada pelo autor

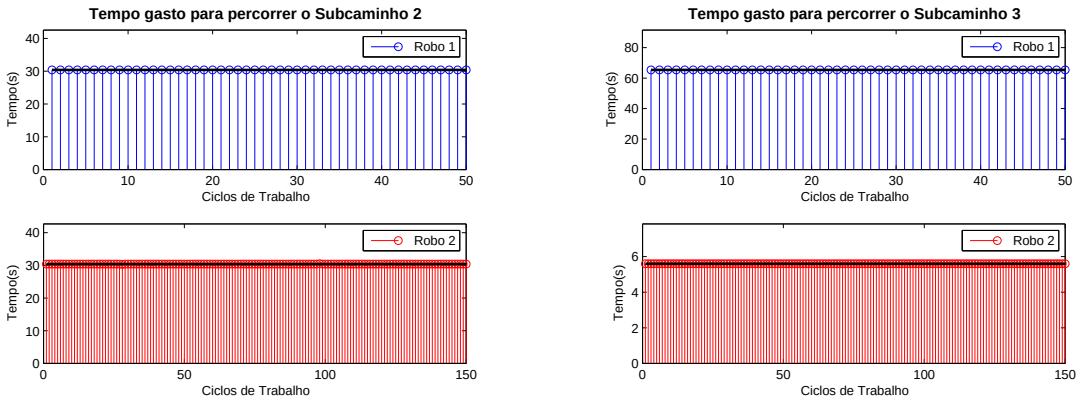
Figura 60 – Tempos Envolvidos na Tarefa - Simulação do Caso 2



(a) Tempo Total de cada Ciclo de Trabalho: *(b) Tempo de Percurso Subcaminho 1*:

Fonte: Elaborada pelo autor

Figura 61 – Tempos de Percurso



(a) Tempo de Percurso Subcaminho 2:

(b) Tempo de Percurso Subcaminho 3:

Fonte: Elaborada pelo autor

Pela Tabela 18 nota-se que a média dos tempos em cada trecho do circuito é idêntica aos tempos mostrados na Tabela 17 definidos durante o planejamento. A última linha da Tabela 18 refere-se à média do tempo de uma volta completa para todos os ciclos de trabalho efetuados e à média de velocidade exercida durante todo o percurso.

Tabela 18 – Média dos Tempos e Velocidades - Simulação Caso 2

TRECHOS DOS CIRCUITOS	TEMPO(s)		VELOCIDADE (m/s)	
	R1	R2	R1	R2
SUBCAMINHO 1	24	4	0.833	0.995
SUBCAMINHO 2	30.4	30.4	0.2	0.374
SUBCAMINHO 3	65.6	5.6	0.517	0.832
CICLO	120	40	0.5	0.5

Fonte: Elaborada pelo autor

4.3 ANÁLISE DOS RESULTADOS

Comparando os resultados obtidos no caso 1 através da simulação e do experimento prático mostrados na Seção 4.1.1, criou-se as Tabelas 19 e 20. A primeira refere-se à diferença entre os tempos advindos da simulação e do experimento prático, e a segunda aborda a comensurabilidade dos ciclos através da razão entre o tempo de ciclo e o período base. É importante destacar que os resultados mostrado em ambas as tabelas são provenientes da média realizada sobre todos os ciclos.

Tabela 19 – Diferença entre os Tempos - Simulação/Experimento

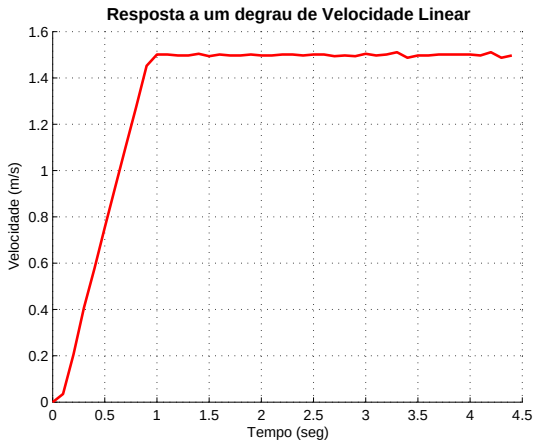
TRECHO DOS CIRCUITOS	DIFERENÇA	
	R1	R2
SUBCAMINHO 1	0,031	0,062
SUBCAMINHO 2	0,042	0,052
SUBCAMINHO 3	0,091	0,147
CICLO	0,098	0,067

Fonte: Elaborada pelo autor

A Tabela 19 mostra que existem diferenças entre os tempos computados entre a simulação e o experimento. Há diversos motivos que justificam tais diferenças, como:

- Robô Simulado/Robô Real: O robô real está sujeito a diversas forças como inércia, atrito e peso, que influenciam no seu movimento, diferentemente do robô simulado.
- Varição de Velocidade: O robô simulado consegue mudar de forma instantânea suas velocidades. Já o robô real apresenta uma curva de aceleração, conforme pode ser visto na Figura 62.

Figura 62 – Curva de Reação - Velocidade Linear



Fonte: Elaborada pelo autor

- Localização: A hodometria utilizada durante a simulação não apresenta erros. Já a usada para realizar a localização do robô revelou pequenas diferenças. Como estes erros de leitura feitos através da câmera não são determinísticos, o robô pode não conseguir repetir exatamente o mesmo percurso durante os ciclos. Portanto pode não conseguir também repetir os tempos de percurso.

A Tabela 20 mostra que há uma pequena diferença entre a razão T_{i1}/T_{i2} entre os ciclos da simulação e do experimento prático. Percebe-se que durante a simulação,

os robôs completaram suas voltas exatamente no tempo que deveriam, caracterizando periodicidade e comensurabilidade, diferentemente do teste prático.

Tabela 20 – Comensurabilidade dos Ciclos - Experimento Prático

T_{i1}/T_{i2}	
SIMULAÇÃO	**EXPERIMENTO**
1	0.9988

Fonte: Elaborada pelo autor

O caso 2 demonstrou a utilização de valores diferentes de γ, necessário quando os robôs não terminam suas voltas no mesmo ciclo. Nota-se que os resultados obtidos na simulação foram exatamente os planejados, apresentado o mesmo desempenho que o caso simulado 1.

4.4 CONCLUSÃO

Observou-se pela análise do caso 1 e 2 que os robôs percorreram o mesmo caminho no ambiente de simulação durante todas as voltas de duração da tarefa, como pode ser visto nas Figuras 48 a), b) e 59 a), b). Já no teste experimental houve uma diferença maior entre os pontos percorridos pelos robôs conforme ilustra a Figura 54 a) e b), devido a pequenos erros nas leituras de posição dos robôs e a não incorporação da modelagem dinâmica do sistema. No entanto, os resultados comprovam que o impacto no teste experimental realizado foi pequeno.

Devido à dinâmica do robô, há pequenos atrasos durante mudanças de velocidades. Tal fato contribuiu para que os robôs não atingissem os pontos de estudo exatamente nos tempos planejados, diferentemente do ocorrido nas simulações, conforme mostrado nas Figuras 49 e 50 e na Tabela 14. Mas a principal causa dessas distorções se deu pela diferença dos traçados realizados pelos robôs ao longo da tarefa.

Sobre a comensurabilidade dos ciclos conclui-se pela Tabela 20 que os resultados obtidos através de simulação e do experimento prático proporcionaram que os tempos de ciclo fossem comensuráveis, uma vez que a razão entre o tempo de ciclo T_i e o período base T é um número racional.

Porém durante o experimento prático verificou-se que os tempos de ciclos não são periódicos. Portanto, apesar deles serem praticamente iguais aos planejados conforme ilustrado nas Figuras 55 e 56, em uma situação onde tem-se muitos ciclos de trabalho os robôs podem vir a se colidir.

Em relação ao caso 4.2, provou-se que a metodologia funciona também para casos onde há grandes diferenças de tamanho entre os circuitos, de forma que um robô possa

estar em um ciclo diferente dos demais sem nenhum prejuízo a tarefa.

Contudo, apesar das peculiaridades encontradas durante a execução do teste prático, após 30 ciclos de trabalho os robôs terminaram cada volta praticamente no mesmo tempo, sendo verificado que a margem de segurança adotada através do intervalo de tempo Δt que os robôs cruzavam por um determinado ponto de colisão foi o suficiente para coordená-los sem acidentes.

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 CONCLUSÕES

O presente trabalho teve como objetivo central realizar a coordenação da navegação dos robôs por caminhos cíclicos que se interceptam em pontos de colisão enquanto eles evitam colisões entre si. Para isso, utilizou-se um controlador central que elaborou um perfil de velocidade média que os robôs precisaram exercer sobre determinados trechos dos circuitos para que pudessem completar esses percursos nos tempos desejados, com o compromisso de maximizar o menor intervalo de tempo entre a passagem dos robôs por um mesmo ponto de colisão.

Para realizar tal tarefa, foi descrito no capítulo 2 a modelagem matemática do problema, sendo demonstradas todas as equações e inequações que restringem tal situação a fim de maximizar a função objetivo Δt. Vale lembrar que a tarefa se repete por diversos ciclos que podem durar de acordo com a necessidade. Portanto, trata-se de um problema de horizonte infinito, sendo modelado com um número finito de variáveis e restrições. Desta forma, planejou-se o perfil de velocidade para cada robô uma única vez, e ele foi seguido em todos os ciclos que compõe a missão.

No capítulo 2, demonstrou-se algumas características do *LINGO*, assim como suas principais vantagens. Entre elas está a possibilidade de se comunicar com outras aplicações. No caso do presente trabalho, foi elaborado uma rotina em C++ que executou o *script* do *LINGO* que continha a modelagem do problema de otimização, permitindo que fossem armazenadas os valores da variáveis em estudo, como Δt, os tempos que os robôs deveriam percorrer cada trecho e as distâncias dos subcaminhos. Tais dados foram de fundamental importância para a execução da rotina que realizou a navegação dos robôs.

Ao final do capítulo 2, elaborou-se um exemplo com o objetivo de demonstrar a metodologia de planejamento dos perfis de velocidade. Através das respostas obtidas e mostradas na Tabela 9 concluiu-se que o planejamento realizado atingiu todos os objetivos, uma vez que cumpriu todas as restrições resumidas na seção 2.1.1.1, permitindo que o intervalo de tempo entre a passagem dos robôs fossem maximizados.

Sobre a navegação dos robôs, verificou-se que ROS foi de extrema importância no trabalho, pois através dele realizou-se a comunicação com os robôs. A utilização do pacote *ROSARIA* possibilitou o acionamento e a leitura das velocidades das rodas. Essas informações estavam disponíveis em vários tópicos produzidos pelo próprio pacote, que foram utilizadas na rotina em C++ que era responsável pela interação com os robôs.

Outro pacote que exerceu função indispensável foi *ARPOSE*. Ele permitiu detectar os marcadores, que foram estrategicamente posicionados de forma que pelo menos dois pudessem ser vistos em quaisquer pontos da sala. Assim, foram publicados continuamente

as suas posições e orientações em relação ao *frame* de coordenadas da câmera no tópico */ar_pose_marker*. Por meio dessas informações, foi possível realizar a composição das matrizes de transformações homogêneas ${}^{G}_{M}T$ e ${}^{M}_{C}T$ com o objetivo de encontrar ${}^{G}_{C}T$, que fornece informações sobre posição e orientação do robô em relação ao sistema de referência global, como foi abordado na Seção 3.3.2. Esta foi a estratégia utilizada para localizar o robô no espaço de trabalho e conclui-se que em um ambiente controlado ela é uma alternativa viável.

Os resultados do caso 1 apresentados nas seções 4.1.1 e 4.1.2 mostraram que existem diferenças entre a aplicação efetuada através de simulação e de um experimento prático, como já era esperado. Isto se deve a vários motivos, entre eles está o fato de que o robô simulado no *STAGE* representa um modelo simples de um robô, desconsiderando as forças e variáveis presentes em sua dinâmica, como torque e massa por exemplo. Logo, o robô simulado no *STAGE* consegue realizar mudanças bruscas de velocidade instantaneamente, não ocasionando atrasos durante o percurso, permitindo desta forma que os tempos de travessia inicialmente traçados fossem perfeitamente alcançados.

Outro fator a ser considerado se dá em relação às medidas de posição. Para a aplicação em questão, é fundamental que os robôs sempre passem o mais próximo pelos mesmos pontos do percurso durante todos os ciclos de trabalho. Caso exista diferenças substanciais entre o caminho percorrido em diferentes ciclos, a coordenação da tarefa fica comprometida, uma vez que podem ocorrer atrasos ou o robô chegar mais cedo no ponto objetivo.

Enquanto no simulador é possível obter leituras perfeitas da *pose* dos robôs, o mesmo não pode ser dito a respeito do método utilizado para localizar o robô na prática, que retorna uma estimativa. Quanto mais distante e complexo for o marcador e mais escuro o ambiente, mais imprecisa fica as leituras.

No entanto, foram utilizadas somente medidas que apresentavam confiabilidade superiores a 90%, permitindo os robôs percorrem seus caminhos com uma precisão aceitável durante os vários ciclos de trabalho, conforme mostrado nas Figuras 54. Vale lembrar que o valor acima é calculado pelo *Nó* do pacote *ARPOSE* e publicado em um tópico no ROS, sendo que ele foi utilizado como filtro para se utilizar ou descartar as leituras.

O controlador de ângulo abordado na Seção 3.4.2 também se mostrou eficiente, uma vez que os robôs conseguiram rastrear os *waypoints* passados como referência, como pode ser visto nas Figuras 53.

Contudo, conclui-se que a metodologia descrita no capítulo 2 é uma boa alternativa para coordenar a navegação dos robôs, desde que ela seja aplicada sob as condições definidas no presente trabalho. Os resultados obtidos foram satisfatórios, uma vez que os valores dos tempos de percurso do experimento prático foram muito semelhantes aos

tempos inicialmente planejados conforme mostrado nas Tabelas 9 e 12.

Portanto, os principais objetivos foram cumpridos tornando possível a coordenação dos robôs durante toda a tarefa. Vale destacar também que o valor do Δt representou 50% do tempo total de uma volta completa, proporcionando uma boa margem de segurança que ajudou a prevenir possíveis colisões.

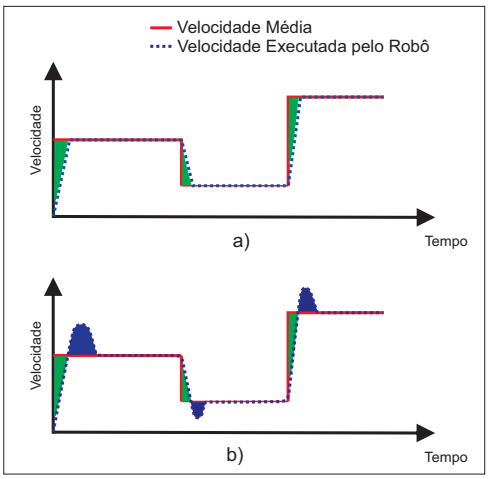
5.2 TRABALHOS FUTUROS

A continuação natural deste trabalho seria desenvolver um sistema de localização mais robusto, de maneira a minimizar os ruídos causados pela iluminação inadequada. Além disso, pretende-se utilizar filtro de *Kalman* para corrigir a hodometria realizada pela leitura dos *encoders* através das informações obtidas pelo sistema de visão.

Outra atuação seria no controle da trajetória dos robôs. No presente trabalho os cálculos foram realizados sobre a velocidade média que eles teriam que executar num dado trecho do circuito. No entanto, é sabido que na prática não se consegue atingir as velocidades necessárias instantaneamente devido à características construtivas dos robôs.

Assim sendo, atrasos provocados pela dinâmica do robô são gerados todas as vezes que as velocidades são alteradas, como indicado pelas áreas verdes na Figura 63 a). Portanto, o controlador de baixo nível precisa atuar na velocidade instantânea de forma a garantir que os robôs percorram os subcaminhos com a velocidade média planejadas, compensando os atrasos produzidos. A Figura 63 b) ilustra tal situação.

Figura 63 – Perfil de Velocidade



Fonte: Elaborada pelo autor

Como o planejamento do perfil de velocidades é feito *offline*, caso o robô altere em algum ciclo o caminho inicialmente traçado ou simplesmente ele não consiga atingir um ponto no tempo desejado, a tarefa fica comprometida. Nestas situações, pretende-se

elaborar um sistema capaz de se re-otimizar, levando em consideração a posição atual do robô que se transformaria em seu novo ponto inicial.

Pretende-se também atuar no processo de recarregamento das baterias de forma que a autonomia da missão possa ser estendida para o tempo que for necessário. Para tal, alguns estudos já foram iniciados [25] e atualmente encontra-se em fase de desenvolvimento.

REFERÊNCIAS

[1] EPOCA. *Afrouxem os cintos: o motorista sumiu.* 2014. Disponível em: <http://epoca.globo.com/ideias/noticia/2014/06/afrouxem-os-cintos-bo-motorista-sumiub.html>.

[2] ONU. *UN chief spotlights road safety as traffic accidents claim 1.2 million lives each year.* 2012. Disponível em: <http://www.un.org/apps/news/story.asp?NewsID=43528>.

[3] AMAZON. *Amazon Prime Air.* 2014. Disponível em: <http://www.amazon.com/b?node=8037720011>.

[4] G1. *Amazon testa drones para agilizar entregas.* 2013. Disponível em: <http://g1.globo.com/tecnologia/noticia/2013/12/amazon-testa-drones-para-agilizar-entregas.html>.

[5] EXAME. *Com drones, Amazon quer fazer entregas em até 30 minutos.* 2013. Disponível em: <http://exame.abril.com.br/tecnologia/noticias/com-drones-amazon-quer-fazer-entregas-em-ate-30-minutos>.

[6] ISCOLD, P.; PEREIRA, G. A.; TORRES, L. A. Development of a hand-launched small uav for ground reconnaissance. *Aerospace and Electronic Systems, IEEE Transactions on*, IEEE, v. 46, n. 1, p. 335–348, 2010.

[7] SUJIT, P.; KINGSTON, D.; BEARD, R. Cooperative forest fire monitoring using multiple uavs. In: IEEE. *Decision and Control, 2007 46th IEEE Conference on.* [S.l.], 2007. p. 4875–4880.

[8] ALVES, A. S. C. *Estudo e aplicação de técnicas de controle embarcadas para estabilização de voo de quadricópteros.* Tese (Doutorado) — UFJF, 2012.

[9] SANTOS, M. F. D. *Controle Tolerante a Falhas de um Sistema de Propulsão de Hexacópteros.* Tese (Doutorado) — UFJF, 2014.

[10] GONCALVES, V. M. et al. Coordination of multiple fixed-wing uavs traversing intersecting periodic paths. In: IEEE. *Robotics and Automation (ICRA), 2013 IEEE International Conference on.* [S.l.], 2013. p. 849–854.

[11] PAW, Y. C.; BALAS, G. J. Development and application of an integrated framework for small uav flight control development. *Mechatronics*, Elsevier, v. 21, n. 5, p. 789–802, 2011.

[12] CORKE, P. *Robotics, vision and control: fundamental algorithms in MATLAB.* [S.l.]: Springer, 2011.

[13] (ADEPT MOBILE ROBOTS. *Pioneer P3DX Datasheet.* Disponível em: <http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx>.

[14] MIRZAEI, M. et al. Cooperative multi-vehicle search and coverage problem in uncertain environments. In: *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on.* [S.l.: s.n.], 2011. p. 4140–4145. ISSN 0743-1546.

[15] WURM, K. M.; STACHNISS, C.; BURGARD, W. Coordinated multi-robot exploration using a segmentation of the environment. In: IEEE. *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. [S.l.], 2008. p. 1160–1165.

[16] LANGERWISCH, M.; WAGNER, B. Dynamic path planning for coordinated motion of multiple mobile robots. In: IEEE. *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. [S.l.], 2011. p. 1989–1994.

[17] LIU, S.; SUN, D.; ZHU, C. Coordinated motion planning of multiple mobile robots in formation. In: IEEE. *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*. [S.l.], 2010. p. 1806–1811.

[18] LIU, S.; SUN, D.; ZHU, C. Coordinated motion planning for multiple mobile robots along designed paths with formation requirement. *Mechatronics, IEEE/ASME Transactions on*, IEEE, v. 16, n. 6, p. 1021–1031, 2011.

[19] NY, J. L.; PAPPAS, G. J. On trajectory optimization for active sensing in gaussian process models. In: IEEE. *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*. [S.l.], 2009. p. 6286–6292.

[20] LOW, C. B. A trajectory tracking control scheme design for nonholonomic wheeled mobile robots with low-level control systems. In: *CDC*. [S.l.: s.n.], 2012. p. 536–543.

[21] AMOOZGAR, M.; ZHANG, Y. Trajectory tracking of wheeled mobile robots: A kinematical approach. In: IEEE. *Mechatronics and Embedded Systems and Applications (MESA), 2012 IEEE/ASME International Conference on*. [S.l.], 2012. p. 275–280.

[22] SOLTERO, D. E.; SMITH, S.; RUS, D. Collision avoidance for persistent monitoring in multi-robot systems with intersecting trajectories. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. [S.l.: s.n.], 2011. p. 3645–3652. ISSN 2153-0858.

[23] AKELLA, S.; HUTCHINSON, S. Coordinating the motions of multiple robots with specified trajectories. In: *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*. [S.l.: s.n.], 2002. v. 1, p. 624–631 vol.1.

[24] PENG, J.; AKELLA, S. Coordinating multiple robots with kinodynamic constraints along specified paths. *The International Journal of Robotics Research*, SAGE Publications, v. 24, n. 4, p. 295–310, 2005.

[25] TEIXEIRA, A. M. et al. Coordenação Ótima de múltiplos robôs de serviço e de recarga em tarefas persistentes. In: IEEE. *International Conference on Industry Applications Induscon, 2014 IEEE International Conference on*. [S.l.], 2014. p. 849–854.

[26] RIPOLL, C. C.; RIPOLL, J. B.; SANT'ANA, A. A. O mínimo múltiplo comum e o máximo divisor comum generalizados. n. 40, p. 59–74, 2006.

[27] LINGO User Manual - The Modeling Language and Optimizer. [S.l.].

[28] MARTINEZ, A.; FERNáNDEZ, E. *Learning ROS for Robotics Programming*. Packt Publishing, 2013. ISBN 1782161449. Disponível em: <`http://www.packtpub.com/learning-ros-for-robotics-programming/book`>.

[29] OPEN SOURCE ROBOTICS FOUNDATION. *Ros Documentation*. Disponível em: <http://wiki.ros.org/>.

[30] MICROSOFT - HARDWARES. *Microsoft WebCam*. Disponível em: <http://www.microsoft.com/hardware/en-us/webcams>.

[31] KATO, H. *ARToolKit Documentation - How does ARToolKit Work?* Disponível em: <http://www.hitl.washington.edu/artoolkit/documentation/index.html>.

[32] FORSYTH, D. A.; PONCE, J. *Computer Vision: A Modern Approach*. [S.l.]: Prentice Hall Professional Technical Reference, 2002. ISBN 0130851981.

[33] KATO, H. *ARToolKit Documentation - Computer Vision Algorithm*. Disponível em: <http://www.hitl.washington.edu/artoolkit/documentation/vision.htm>.

[34] WALCZYK, R.; ARMITAGE, A.; BINNIE, D. Comparative study on connected component labeling algorithms for embedded video processing systems. CSREA Press, 2010.

[35] CRAIG, J. J. *Introduction To Robotics: Mechanics And Control, 3/E*. Pearson Education, 2008. ISBN 9788131718360. Disponível em: <http://books.google.com.br/books?id=vlMl-j_cWW4C>.

[36] MASON, M. T. Compliance and force control for computer controlled manipulators. *Systems, Man and Cybernetics, IEEE Transactions on*, v. 11, n. 6, p. 418–432, June 1981. ISSN 0018-9472.

[37] SICILIANO, B. et al. *Robotics: Modelling, Planning and Control*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2008. ISBN 1846286417, 9781846286414.