UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA
DISSERTAÇÃO DE MESTRADO

OTIMIZAÇÃO DO CÓDIGO DO SISTEMA DE NAVEGAÇÃO E CONTROLE DE ROBÔS MÓVEIS BASEADO EM NMPC PARA EMBARCAR EM ARQUITETURAS DE BAIXO CUSTO

DIEGO SOUSA DE AZEVEDO

João Pessoa - Paraíba
Outubro de 2015

OTIMIZAÇÃO DO CÓDIGO DO SISTEMA DE NAVEGAÇÃO E CONTROLE DE ROBÔS MÓVEIS BASEADO EM NMPC PARA EMBARCAR EM ARQUITETURAS DE BAIXO CUSTO

DIEGO SOUSA DE AZEVEDO

Dissertação de Mestrado apresentada à banca examinadora do Programa de Pós-graduação em Informática da Universidade Federal da Paraíba como requisito para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Tiago P. Nascimento.
Coorientador: Prof. Dr. Alisson V. de Brito.

João Pessoa - Paraíba
Outubro de 2015

A994o Azevedo, Diego Sousa de.
Otimização do código do sistema de navegação e controle de robôs móveis baseado em NMPC para embarcar em arquiteturas de baixo custo / Diego Sousa de Azevedo.- João Pessoa, 2015.
98f. : il.
Orientador: Tiago P. Nascimento
Dissertação (Mestrado) - UFPB/CI
1. Informática. 2. NMPC. 3. Controle de formação. 4. Multi-Robôs. 5. Sistemas embarcados.

UFPB/BC CDU: 004(043)

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de **DIEGO SOUSA DE AZEVEDO**, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 09 de outubro de 2015.

Ao nono dia do mês de outubro do ano de dois mil e quinze, às quinze horas, no Centro de Informática - Universidade Federal da Paraíba (unidade Mangabeira), reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do **Sr. Diego Sousa de Azevedo** vinculado a esta Universidade sob a matrícula 2013116080, candidato ao grau de Mestre em Informática, na área de *"Sistemas de Computação"*, na linha de pesquisa *"Sinais, Sistemas Digitais e Gráficos"*, do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores doutores: **Tiago Pereira do Nascimento (PPGI-UFPB),** Orientador e Presidente da Banca, **Alisson Vasconcelos de Brito (PPGI-UFPB)**, Examinador Interno, **Ruy Alberto Pisani Altafim (PPGI-UFPB)**, Examinador Interno e **Luiz Marcos Garcia Golçalves (UFRN)** Examinador Externo à Instituição. Dando início aos trabalhos, o professor Presidente da Banca cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado *"Adaptação de um Controlador Preditivo de Formação Baseado em Modelo Não-Linear para Embarcar em Arquiteturas de Baixo Custo"*. Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: *"aprovado"*. Assim sendo, eu, Ruy Alberto Pisani Altafim, Vice-Coordenador do Programa de Pós Graduação em Informática - PPGI, lavrei a presente ata que vai assinada por mim e pelos membros da Banca Examinadora. João Pessoa, 09 de outubro de 2015.

Ruy Alberto Pisani Altafim

Profº Drº TIAGO PEREIRA DO NASCIMENTO
Orientador (PPGI-UFPB)

Profº Drº ALISSON VASCONCELOS DE BRITO
Examinador Interno (PPGI-UFPB)

Profº Drº RUY ALBERTO PISANI ALTAFIM
Examinador Interno (PPGI – UFPB)

Profº Drº LUIZ MARCOS GARCIA GONCALVES
Examinador Externo à Instituição (UFRN)

Dedico este trabalho à minha Mãe, Maria de Lourdes.

Fundamental em toda minha formação.

Agradecimentos

Primeiramente à Deus, por sempre guiar os meus passos durante toda essa jornada acadêmica.

À minha querida namorada Eline Raquel, pelo seu apoio e incentivo nas horas mais difíceis, sempre compreendendo minhas constantes ausências.

À minha mãe Maria de Lourdes, por nunca me deixar desistir dos meus sonhos independentes de qual seja a dificuldade.

Ao professor Dr. Tiago Nascimento, pela dedicação e pelos valiosos ensinamentos, estes essenciais para o desenvolvimento deste trabalho.

Ao professor Dr. Alisson Vasconcelos, por se dispor a ser meu coorientador, tornando-se a principal referência em Sistemas Embarcados. Sem sua presença não seria possível inserir a qualidade necessário para construção dessa dissertação.

A todos os meus amigos adquiridos durante esse desafio, dos quais lembrarei pelo resto da minha vida.

A todos os professores do PPGI que contribuíram direta ou indiretamente para minha formação.

Ao CNPQ por ter financiado esta pesquisa, disponibilizando recursos essenciais para a continuidade da mesma.

A todas as outras pessoas que não me vieram à memória, mais que contribuíram fortemente para a minha formação pessoal e profissional.

“A menos que modifiquemos à nossa maneira de pensar, não seremos capazes de resolver os problemas causados pela forma como nos acostumamos a ver o mundo”.

(Albert Einstein)

Resumo

A proposta desse trabalho é adaptar e embarcar um sistema de navegação e controle de robôs móveis, baseado em NMPC, em uma placa de baixo custo já existente no mercado, que disponibilize recursos computacionais suficientes para que o Robô seja capaz de convergir, sem perda de desempenho e utilizando os mesmos horizontes aplicados em um Laptop. Os Resultados obtidos demonstram todo o cenário proposto e de acordo com os experimentos realizados, comprovou-se que é possível o uso de placas de baixo custo, para controle de robôs móveis, baseado em NMPC, utilizando os mesmos horizontes de predição e controle aplicados em uma Laptop.

Palavras-chave: NMPC, Controle de formação, Multi-Robôs e Sistemas Embarcados.

Abstract

The purpose of this study is to adapt and embed a navigation system and control of mobile robots, based on NMPC, in a low-cost board existent on the market, to provide sufficient computational resources so that the robot is able to converge, without losing performance, using the same horizons applied in a Laptop. The obtained results demonstrate the proposed scenario according with the experiments, proving that it is possible to use low cost boards, to a navigation system and control of mobile robots, based on NMPC, using the same predictive and control horizons applied in a Laptop.

Keywords: NMPC, Training Control, Multi-Robots and Embedded Systems.

Sumário

Índice de Abreviaturas e Símbolos

NMPC *Nonlinear Model Predictive Control*

NMPFC *Nonlinear Model Predictive Formation Control*

BBB *BeagleBone Black*

Índice de Figuras

Índice de Tabelas

Índice de Gráfico

Índice de Código

Capítulo 1

Introdução

Atualmente robôs moveis estão inseridos em vários setores. A inserção de robôs móveis tem principalmente o objetivo de automatizar as atividades internas de empresas, de residências e de setores governamentais. Com o aparecimento da automação e do crescimento destes setores e da diversificação das áreas de atuação, devido à inserção destes robôs, surgiu a necessidade de que estes trabalhassem em conjunto. As vantagens da utilização de um grupo de vários robôs incluem robustez, flexibilidade e capacidade de adaptação a ambientes dinâmicos desconhecidos. Na Figura 1 é apresentado um exemplo de um sistema de multi-robôs, trabalhando em conjunto, estes são claramente importantes quando se considera aplicações, tais como missões de busca e salvamento, detecção de incêndios florestais, detecção de minas, aplicações de vigilância, detecção de intrusos ou mesmo competição de futebol de robôs [1].

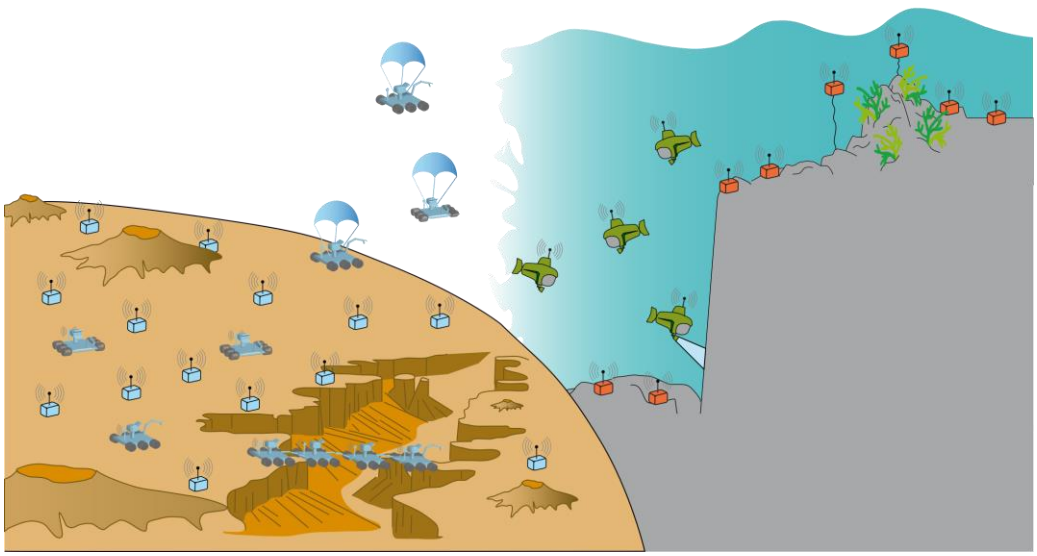


Figura 1: Exemplos de sistemas Multi-Robôs em ambientes desconhecidos.

Sistema multi-robôs é uma área implementada pela robótica móvel [2] [3] [4]. Um robô móvel é um dispositivo mecânico, montado sobre base não fixa, que age sob o controle de um sistema computacional, equipado com sensores e atuadores que o permitem interagir com o

ambiente [5]. O robô móvel deve ser capaz de mover-se através de um ambiente, evitando colisões com obstáculos estáticos ou em movimento, para alcançar o seu objetivo. Um grande desafio ao desenvolvimento de sistema desta natureza é garantir flexibilidade e a capacidade de adaptação ao comportamento dos robôs [6] [7] [8] [9]. Em sistemas multi-robôs, o controle geral dos robôs pode ser centralizado ou distribuído. Segundo Correia e Costa [6], sistemas multi-robôs podem ser descritos como um conjunto de múltiplos robôs, executando tarefas de forma coordenada, em busca de um objetivo comum.

Dessa forma, sistemas Multi-robôs desempenham um trabalho colaborativo, mas para isso, é preciso desenvolver novas soluções de controladores. Um controlador é um sistema computacional, responsável por controlar todas as funções de um robô, análogo a um cérebro, este sistema computacional deve enviar e receber informações para que o robô possa desempenhar suas funções. Um dos controladores utilizados em sistemas multi-robôs é o controle de formação. A Figura 2 representa a interface de configuração do controlador de formação utilizado neste trabalho.

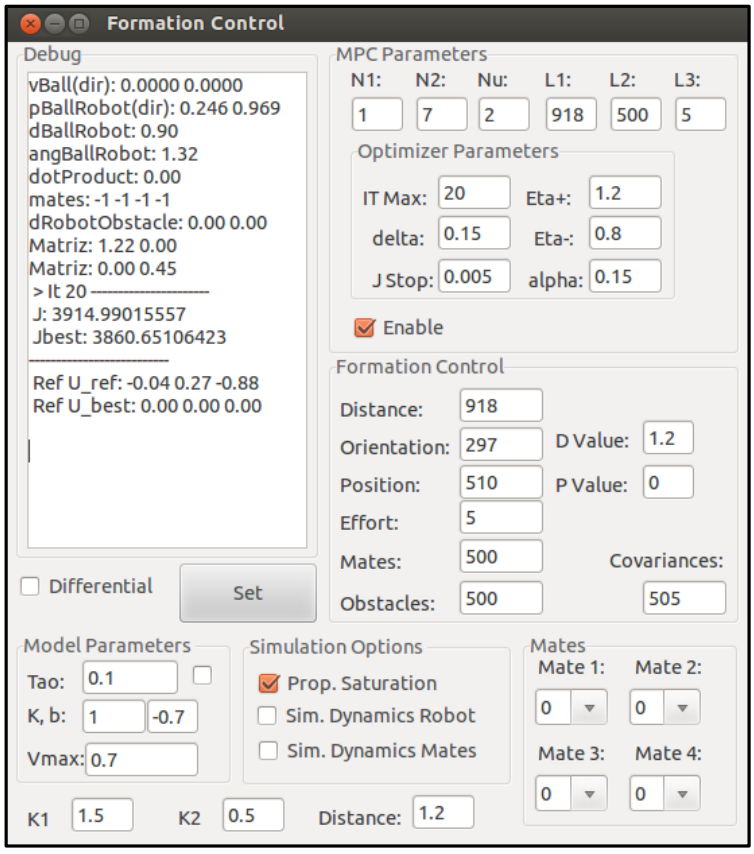


Figura 2: Interface de configuração do Controlador de Formação Preditivo (*RoC – Robot Control*).

Desta forma, controle de formação pode ser definido como a coordenação de múltiplos robôs móveis (no caso de determinadas trajetórias, para acompanhar trajetórias parametrizadas e referência de tempo), para manter uma formação espacial desejada. Atualmente manter uma formação ideal é um problema em aplicações multi-robôs já que tais sistemas necessitam de algoritmos de decisão que controle as ações de formação do robô com o menor custo computacional.

Em controle de formação são utilizados dois estimadores de tempo para identificar a autonomia do robô, o horizonte de predição (horizonte sobre o qual o agente prediz o comportamento autônomo do sistema) e o horizonte de controle (horizonte sobre o qual o agente encontra as ações). O horizonte de controle e o horizonte de predição proporcionam estimativas da eficiência da técnica aplicada, porém o resultado depende do tempo, ou seja, o resultado final é influenciado pelo desempenho do computador. O desafio então, é reduzir o poder computacional, sem reduzir os horizontes. Além disso, a formação deve ter baixo poder computacional, pouco uso de energia e alta eficiência de comunicação [1]. Diante disso, controlar a formação se tornou um tema comum de estudo em sistemas multi-robôs [1] [10] [11] [12] [13] [14].

Gradativamente, dependendo da utilização de cada sistema autônomo, torna-se necessário reduzir o tamanho, custos de fabricação, peso e energia elétrica de cada robô, porém sem impactar na autonomia de todo o sistema. Neste sentido, o uso de arquiteturas embarcadas de baixo custo é uma possível solução para esse propósito, uma vez que suas características (tamanho, custo, peso) reduzidas, possibilitam solucionar os objetivos aqui propostos. No entanto, garantir o poder de processamento necessário para que os robôs consigam convergir e manter o seu estado de convergência pode ser uma tarefa bem desafiadora, uma vez que tais arquiteturas, em relação aos Laptops, dispõem de baixo poder computacional.

1.1 Proposta

Considerando os desafios citados, a proposta geral desse trabalho *é otimizar um sistema de navegação e controle de robôs móveis, baseado em NMPFC, em seguida propor a utilização de arquiteturas de baixo custo, como plataformas embarcadas para tais aplicações, a ser utilizados em sistemas multi-robôs, assim, disponibilizando um sistema computacional mais simples.*

1.1.1 Proposta Específica

Em continuidade, os seguintes pontos foram considerados com o intuito de efetivar a proposta descrita acima, e são compreendidos como propostas específicos da pesquisa:

- Acelerar os algoritmos de controle e de predição, a fim de permitir o uso de horizontes de predição e controle iguais aos utilizados no laptop num sistema embarcado;
- Propor as características de arquiteturas de baixo custo embarcadas, necessárias para este tipo de sistema.
- Embarcar um sistema de navegação e controle de robôs móveis, baseado em NMPC, em uma placa de baixo custo que disponibilize recursos computacionais suficientes para que o Robô seja capaz de convergir, sem perda de desempenho;

Esta pesquisa é parte integrante de um projeto maior, que visa criar um sistema de controle de formação aplicado a diversas equipes de robôs móveis (sistemas multi-robôs ou SMR) terrestres e aéreos destinados a monitorar o perímetro aberto (na área de estacionamento, acessos e mata) e fechado (dentro de edificações) do Campus V da unidade e Mangabeira da Universidade Federal da Paraíba, evitando possíveis acessos de intrusos não pertencentes à comunidade acadêmica, podendo ser utilizados também para a recepção de alunos e visitantes.

1.2 Metodologia

Este trabalho foi baseado na adaptação de uma arquitetura de *software* e de *hardware* para robôs móveis aplicados a ambientes não estruturados. A metodologia deste projeto pode ser apresentada como a sucessão das seguintes atividades:

- Mapear o estado da arte de controle de formação, controle de trajetória, sistemas multi-robôs, NMPC (*Nonlinear Model Predictive Control*) e sistema embarcados;
- Analisar e verificar a viabilidade de simplificação de modelos e predição;
- Embarcar todo o código fonte de navegação no BeagleBone Black para que seja feito o estudo de viabilidade do sistema de navegação como um todo;
- Verificar a viabilidade do uso do BeagleBone Black como substituto do Laptop, no sistema de navegação do robô e seus efeitos no controlador pertencente ao sistema de

navegação.

- Caso a BeagleBone Black não seja capaz de substituir um Laptop para o uso do sistema de navegação do robô, será verificado a viabilidade do uso da UDOO como possível substituta do laptop.
- Realizar otimizações no controlador com o auxílio da técnica *Profiling*, com o objetivo de aumentar os horizontes do controlador.
- Apresentar uma solução para controle de robôs autônomos com o melhor custo benefício, focando principalmente em uso de CPU.

1.3 Organização da Dissertação

Quanto a sua estrutura, esta dissertação está organizada em quatro capítulos que discutem inicialmente a introdução, proposta, bem como as propostas específicas e metodologia deste trabalho. No segundo capítulo é apresentado o Referencial Teórico no qual são expostos os conceitos utilizados nesta pesquisa, além de alguns trabalhos relacionados que, direta ou indiretamente, possuem relação com esta pesquisa, além de disponibilizar uma fundamentação teórica com o intuito de familiarizar o leitor com alguns tópicos que são discutidos durante este trabalho. Baseado em toda a fundamentação teórica e na metodologia explanada anteriormente, o Capítulo 3 expõe os resultados atingidos, nos quais são apresentadas as dificuldades e soluções encontradas. Finalmente, no Capítulo 4, é apresentado um resumo do trabalho, expondo as conclusões e sugestões para estudos futuros. Além destes 4 Capítulos, alguns materiais foram anexados a esta dissertação. No Apêndice B, é apresentado um Tutorial de como instalar todo o ambiente utilizado nesta pesquisa.

Capítulo 2
Fundamentação teórica

Neste capítulo é apresentado uma pesquisa sobre os principais temas que, estão relacionadas ao objetivo desta pesquisa. Inicialmente, é apresentando referências sobre Sistemas Multi-Robôs, destacando os benefícios da cooperação de vários robôs executando tarefas de forma coordenada, em busca de um objetivo comum (Seção 2.1). Em seguida, é apresentado o *software* controlador do robô, chamado RoC (Robot Control) (Seção 2.2). Posteriormente, é destacado o controle de formação, utilizado para que sistemas multi-robôs possam desempenhar suas atividades em conjunto. O Controle de formação é responsável por ditar as regras de deslocamento para todo o sistema multi-robôs, normalmente baseado em uma aproximação de um líder ou de um alvo, que pode ser um robô, um objeto ou um obstáculo, mantendo uma distância segura entre cada obstáculo. Como complemento é destacado os principais conceitos do controle preditivo de formação utilizado nesse trabalho, o NMPFC. O principal objetivo deste modelo é possibilitar uma equipe de robôs convergir em formação e compartilhar suas funções custo (Seção 2.3). Por fim, é discutida as arquiteturas computacionais utilizadas, pequenas placas de baixo custo que serão utilizadas para verificar a capacidade de adaptação a controles de formação preditivo (Seção 2.4).

2.1 Sistema Multi-Robôs

Sistema multi-robôs é uma área de pesquisa da robótica móvel [2] [3] [4]. Um robô móvel é um dispositivo mecânico, montado sobre base não fixa, que age sob o controle de um sistema computacional, equipado com sensores e atuadores que o permitem interagir com o ambiente [5]. O robô móvel deve ser capaz de mover-se através de um ambiente, evitando colisões com obstáculos estáticos ou em movimento, para alcançar o seu objetivo. Um grande desafio ao desenvolvimento de sistema desta natureza é garantir flexibilidade e a capacidade de adaptação ao comportamento dos robôs [6] [7] [8] [9]. Várias tarefas robóticas exigem o benefício da cooperação de vários robôs, por exemplo, o transporte de objetos de grande porte, ampla cobertura

de área (por exemplo, para limpeza) ou de vigilância (por exemplo, para a detecção de incêndio), detecção de alvos, monitoramento e entre outros.

Em sistemas multi-robôs, o controle geral dos robôs pode ser centralizado ou distribuído. Além disso, a formação deve ter baixo poder computacional, pouco uso de energia e alta eficiência de comunicação [1]. Segundo Correia e Costa [6], sistemas multi-robôs podem ser descritos como um conjunto de múltiplos robôs, executando tarefas de forma coordenada, em busca de um objetivo comum. Algumas federações estimulam a pesquisa em robótica, em especial a *Robocup* [15], instituição criada com o intuito de promover pesquisas nas áreas de Inteligência Artificial e Robótica, estimulando estudos sobre navegação, visão computacional, fusão de sensores, estruturas mecânicas, fontes de alimentação, dentre muitas outras.

2.2 Software (RoC e WatchTower)

O controle de cada robô individualmente e da equipe como um todo é realizado por um conjunto de dois programas (RoC, e WatchTower) em permanente comunicação via protocolo UDP. Todas as máquinas envolvidas fazem parte de uma mesma rede local wireless, configurada com o propósito de permitir uma comunicação entre todos os robôs envolvidos. A cada robô é associado um IP fixo dentro da gama da rede utilizada, assim como à máquina central.

O RoC (Figura 3) é o software de controle do robô. Este é embarcado em cada robô participante e nele é armazenado os algoritmos de controle. Durante a formação, o WatchTower (Figura 4) recebe de todos os RoCs a posição do robô e da bola vista por cada robô correspondente. Dependendo destas posições e do estado de formação atual, determina a tática a utilizar e envia de volta a cada RoC o papel tático que determinou para cada um, como também, envia a posição da bola mais provável, calculada a partir da posição reportada por cada robô. Uma única instância do WatchTower é executada numa máquina central e realiza comunicação com as máquinas que correm os RoCs de cada robô.

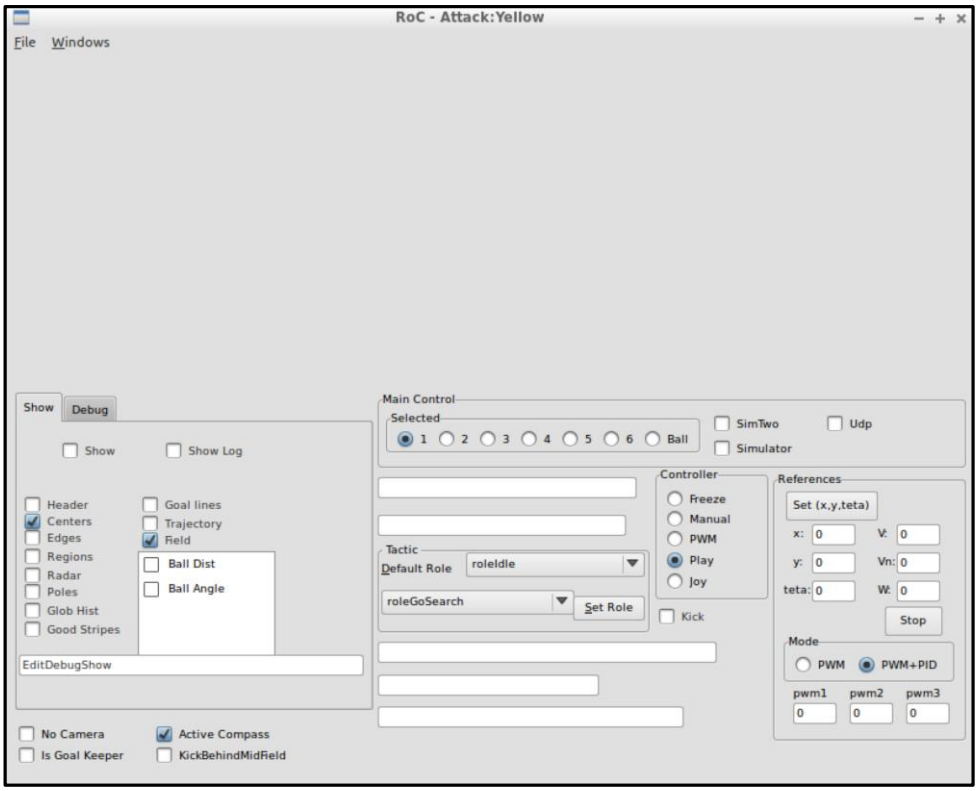


Figura 3: Tela principal do RoC (*Robot Control*)

Figura 4: Tela principal do WatchTower

2.3 Controle de formação

Controle de formação de robôs móveis tem sido alvo de estudos nos últimos anos [1] [10] [11] [12] [13] [14], tendo a aplicação em equipes de sistemas multi-robôs uma das atividades utilizadas. Um dos principais problemas enfrentados em controle de formação são os ambientes dinâmicos desconhecidos [16], como por exemplo, missões de busca e salvamento, mapeamento de fundos oceânicos, detecção de fogos florestais e detecção e remoção de minas. Basicamente controle de formação é baseado em uma aproximação de um líder ou de um alvo, que pode ser um robô, um objeto ou um obstáculo. O sistema multi-robôs segue uma trajetória desejada, mantendo uma distância segura entre cada obstáculo, seja ele um colega ou não. Controle de formação tem um papel fundamental em tarefas executadas em cooperação.

Segundo Gouvêa [17], controle de formação consiste em fazer com que cada componente, de um grupo de agentes, consiga convergir para uma posição de forma que todo o grupo alcance e mantenha uma formação específica. [18] Especificou alguns critérios que devem ser cumpridos para que tal formação seja mantida: (i) Separação: Devem ser evitadas colisões entre os componentes da formação. (ii) Alinhamento**:** Todos os componentes da formação devem ter a mesma orientação. (iii) Coesão**:** A formação deve ser mantida mesmo que todos os agentes estejam se movimentando.

Em Kanjanawanishkul e Zell [19] é apresentado um tipo de controle de formação baseado em um robô líder que segue uma trajetória fixa e transmite dados de velocidade, estado e trajetória para os demais robôs da formação, com isso, os demais robôs se tornam robôs seguidores que utilizam as informações do robô líder para tomar suas próprias decisões. Em Fontes e Caldeira [13] propõem um controlador preditivo para executar o controle de formações de veículos móveis. Lim [14] descreve também uma estratégia de controle se baseando em um líder e utilizando modelos de controladores NMPC distribuídos, com a diferença que este líder não segue uma trajetória fixa, mas é controlado manualmente por um agente humano, o que dificulta ainda mais o controle preditivo. Seguindo os critérios acima, muitos outros trabalhos foram desenvolvidos, como por exemplo, em Fatemeh Daneshfar [20] que apresenta uma pesquisa sobre *multi-agent system* (MAS) em aplicações de engenharia de controle, descrevendo os principais conceitos sobre sobre sistemas multi-agentes relacionados a controle. Em Dimos Dimarogonas [21] tenta resolver o problema de controle de formação utilizando navegação descentralizada. Por fim, em Gouvea [22] são propostas duas estratégias para controle de formação

utilizando robôs moveis não-holonômicos. A partir dessas referências é de se observar que controle de formação em robótica é um assunto bastante pesquisado.

O NMPFC (*Nonlinear Model Predictive Formation Controller*) utilizado nesse trabalho, foi implementado de forma distribuída [23], para que uma equipe de robôs pudesse convergir em formação e que suas funções custo pudessem ser compartilhadas. O acoplamento ocorre quando os estados dos companheiros de equipe são usados na função custo de controle de cada robô, assim, penalizando a geometria ou o desvio do objetivo desejado [10]. Em outras palavras, as ações realizadas por cada robô afetam todos os outros companheiros de equipe.

O NMPFC pode ser dividido em dois sub-blocos. O primeiro, o *Optimizer*, utiliza um método de minimização numérica para aperfeiçoar a função custo e gerar sinais de controle e o segundo sub-bloco, aqui chamado *Predictor*, realiza a evolução do estado do próprio robô, dos companheiros de equipe e do alvo, com base em modelos pré-definidos [24].

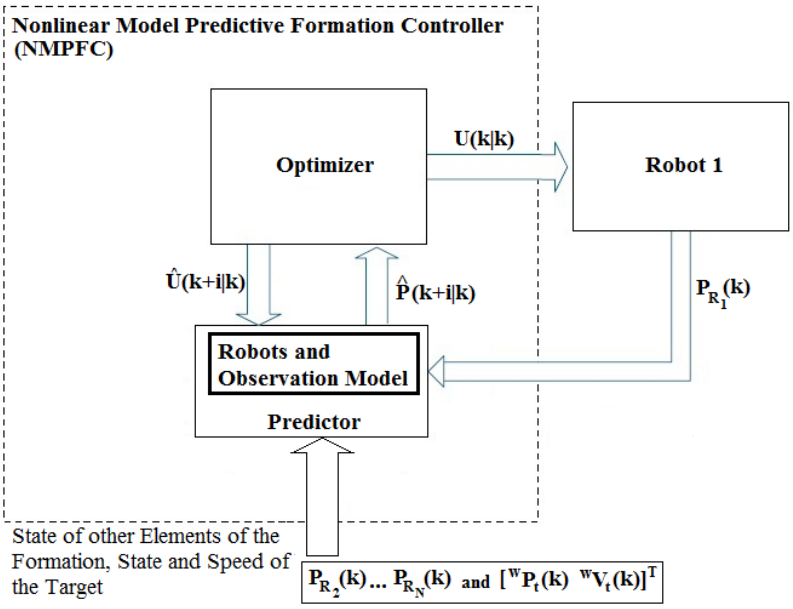


Figura 5: Estrutura do NMPFC aplicado a um robô.

Cada robô mantém um estado de formação, atualizando uns aos outros em cada fluxo de controle. Estas informações são recebidas pelo controlador de cada robô que está na formação, que por sua vez, cria a geometria de formação que irá afetar cada robô. Depois do *optimizer* receber os dados do seu robô, informações de colegas e do alvo, ele fornece a entrada de controle, que em seguida, prevê a evolução do estado de formação e fornece um valor de custo para o *Optimizer*, que por sua vez, novamente irá minimizar a função custo. Todo esse processo é

repetido diversas vezes com o intuito de atingir o menor custo computacional durante a formação. Para este trabalho, utilizamos um algoritmo de atualização em lote, chamado *Resilient Propagation* (RPROP) [25].

No controlador duas métricas são utilizadas para ditar, neste caso, o comportamento da formação entre todos os componentes participantes, ou seja, o robô, os colegas e alvo. São elas, o horizonte de predição que é o horizonte sobre o qual o agente **prever o comportamento** autônomo do sistema e o horizonte de controle que é o horizonte sobre o qual o **agente encontra as ações**. Horizonte de predição e controle representa instantes de tempo (ciclos, sendo que cada ciclo é executado a cada 40ms). De um ponto de vista de tempo computacional, quanto menor for os horizontes de predição utilizados, menor será o tempo de execução necessário para o algoritmo de controle e mais adequado será este para utilização em tempo real [26].

2.3.1 Modelo de Predição

Como mencionado anteriormente, o *predictor* é responsável pela evolução do estado do próprio robô, dos companheiros de equipe e do alvo. Esta previsão é executada com base em modelos pré-definidas de todo o sistema. Cada robô mantém um estado de formação, ou seja, mantém a posição e velocidade dos robôs em formação e de qualquer alvo que deva ser seguido, atualizando-os em cada fluxo (*loop*) de controle. Portanto, o *predictor* pode ser dividido em três partes: o modelo do robô, o modelo de formação e o modelo do alvo.

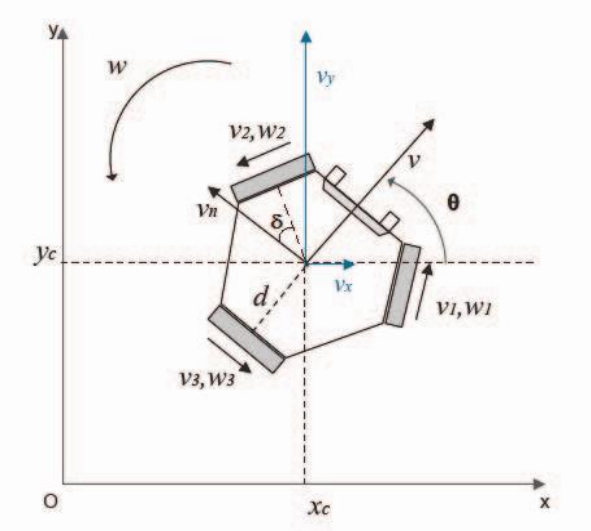


Gráfico 1: Diagrama do modelo do Robô

Neste trabalho é utilizado os modelos cinemático e dinâmico. O modelo dinâmico considera todas as forças aplicada ao robô, como as coordenadas de postura e orientação do robô e os torques desenvolvidos pelos seus atuadores, no entanto, o modelo cinemático considera apenas a geometria que coordena o sistema, desconsiderando as forças que alteram o movimento [27] [28]. No estudo de Azevedo [24], os autores mostraram que a simplificação do modelo de predição do robô e dos colegas aumenta o desempenho do algoritmo de controle melhorando também o desempenho do controlador. Para o modelo de predição do robô, utilizou-se um modelo cinemático o qual representa, o modelo do robô R_n, onde $1 \leq n \leq N$, no qual, N é o total de robôs em formação.

Neste modelo do robô, a velocidade angular da roda pode ser descrita pela função $\omega_{rRn}(k) = [\omega 1(k)\ \omega 2(k)\ \omega 3(k)]^T$ e a velocidade linear pela função $V_{rRn}(k) = [v1(k)\ v2(k)\ v3(k)]^T = r.[\omega 1(k)\ \omega 2(k)\ \omega 3(k)]^T$. A análise geométrica das velocidades do robô, pode ser encontrado usando a seguinte equação:

$$\begin{bmatrix} v_{R_n}(k) \\ vn_{R_n}(k) \\ w_{R_n}(k) \end{bmatrix} = (B)^{-1} \cdot \begin{bmatrix} v_1(k) \\ v_2(k) \\ v_3(k) \end{bmatrix} \qquad (1)$$

$$B = \begin{bmatrix} \cos(\delta) & \sin(\delta) & d \\ -\cos(\delta) & \sin(\delta) & d \\ 0 & -1 & d \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & d \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & d \\ 0 & -1 & d \end{bmatrix} \qquad (2)$$

Onde δ = 30°, r é o raio da roda, e d é a distância entre a roda e o centro do robô. Resultando:

$$(B)^{-1} = \begin{bmatrix} \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & 0 \\ \frac{1}{3} & \frac{1}{3} & -\frac{2}{3} \\ \frac{1}{3d} & \frac{1}{3d} & \frac{1}{3d} \end{bmatrix}. \qquad (3)$$

O modelo do robô móvel omnidirecional utilizado no *Predictor* é um modelo simplificado não-linear, no qual, quando devidamente parametrizado [29], é vantajosa a fim de reduzir a carga computacional de cada ciclo do algoritmo de controle. Isto traduz a capacidade de usar maiores horizontes de predição no controlador preditivo, mesmo em computadores com especificações

mais simples, mantendo o tempo do ciclo de controle dentro dos limites exigidos. O modelo é inicializado com a limitação de velocidade do motor através da detecção de saturação e proporcionalmente escalando os outros motores (que pode ser visto como uma limitação de entrada) [30]. Além disso, a predição da velocidade da roda do robô Rn segue um modelo discreto de primeira ordem, onde:

$$\begin{bmatrix} v_1(k) \\ v_2(k) \\ v_3(k) \end{bmatrix} = a \cdot \begin{bmatrix} v_1(k-1) \\ v_2(k-1) \\ v_3(k-1) \end{bmatrix} + (1-a) \cdot (B^T) \cdot \begin{bmatrix} v_{\mathrm{ref}_{R_n}}(k) \\ vn_{\mathrm{ref}_{R_n}}(k) \\ w_{\mathrm{ref}_{R_n}}(k) \end{bmatrix}.$$

Em seguida, o resultado é mais uma vez inserido na Eq. (1) a ser calculado. Portanto, o estado (pose (P_{Rn} (k)) e a velocidade (V_{Rn} (k))) são definidos como:

$$\begin{aligned} P_{R_n}(k) &= \begin{bmatrix} x_{R_n}(k) & y_{R_n}(k) & \theta_{R_n}(k) \end{bmatrix}^{\mathrm{T}} \\ V_{R_n}(k) &= \begin{bmatrix} v_{xR_n}(k) & v_{yR_n}(k) & w_{R_n}(k) \end{bmatrix}^{\mathrm{T}} \end{aligned} \quad (4)$$

A simulação da evolução do estado é dada, por:

$$\begin{bmatrix} x_{R_n}(k) \\ y_{R_n}(k) \\ \theta_{R_n}(k) \end{bmatrix} = \begin{bmatrix} x_{R_n}(k-1) \\ y_{R_n}(k-1) \\ \theta_{R_n}(k-1) \end{bmatrix} + T \cdot \begin{bmatrix} v_{xR_n}(k) \\ v_{yR_n}(k) \\ w_{R_n}(k) \end{bmatrix}$$

com T como intervalo de tempo:

$$\begin{bmatrix} v_{xR_n}(k) \\ v_{yR_n}(k) \\ w_{R_n}(k) \end{bmatrix} = \begin{bmatrix} \cos(\theta_{R_n}(k)) & -\sin(\theta_{R_n}(k)) & 0 \\ \sin(\theta_{R_n}(k)) & \cos(\theta_{R_n}(k)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{\mathrm{ref}_{R_n}}(k) \\ vn_{\mathrm{ref}_{R_n}}(k) \\ w_{\mathrm{ref}_{R_n}}(k) \end{bmatrix}.$$

Tendo em conta os elementos apresentados, a posição de P_t (k) e a velocidade de V_t (k) do alvo t (bola) no instante k é definido como:

$$P_t(k) = \begin{bmatrix} x_t(k) & y_t(k) \end{bmatrix}^{\mathrm{T}} \qquad V_t(k) = \begin{bmatrix} v_{xt}(k) & v_{yt}(k) \end{bmatrix}^{\mathrm{T}} \quad (5)$$

onde,

$$\begin{cases} x_t(k) = x_t(k-1) + T \cdot (v_{xt}(k)) \\ y_t(k) = y_t(k-1) + T \cdot (v_{y_t}(k)) \end{cases} \quad (6)$$

e

$$\begin{cases} v_{xt}(k) = v_{xt}(k-1) \cdot B_{FC} \\ v_{y_t}(k) = v_{y_t}(k-1) \cdot B_{FC} \end{cases} \quad (7)$$

onde B_{FC} é o coeficiente de atrito da bola. A unidade do vetor de velocidade do alvo é então definida como:

$$\tilde{V}_t(k) = \begin{bmatrix} \tilde{v}_x\, t(k) & \tilde{v}_y\, t(k) \end{bmatrix}^T \models \frac{V_t(k)}{\|V_t(k)\|}. \quad (8)$$

A posição do alvo relativo do robô R_n no instante k é definida como:

$$P_t^{R_n}(k) = \begin{bmatrix} x_t^{R_n}(k) & y_t^{R_n}(k) \end{bmatrix}^T \quad (9)$$

onde

$$\begin{cases} x_t^{R_n}(k) = x_t(k) - x_{R_n}(k) \\ y_t^{R_n}(k) = y_t(k) - y_{R_n}(k). \end{cases} \quad (10)$$

A unidade do vetor que indica a direção do alvo com respectivo robô é definida como:

$$\tilde{P}_t^{R_n}(k) = \begin{bmatrix} \tilde{x}_t^{R_n}(k) & \tilde{y}_t^{R_n}(k) \end{bmatrix}^T = \frac{P_t^{R_n}(k)}{\|P_t^{R_n}(k)\|}. \quad (11)$$

O alvo em relação ao robô Rn, é definido como:

$$\theta_t^{R_n}(k) = \arctan 2(y_t^{R_n}(k), x_t^{R_n}(k)). \quad (12)$$

A posição do robô Rn em relação ao seu companheiro de equipe Rj (onde $1 \leq j \leq NM$, onde NM é o número total de companheiros) são definidos como:

$$P_{\mathrm{R}_n}^{\mathrm{R}_j}(k) = \begin{cases} x_{\mathrm{R}_n}^{\mathrm{R}_j}(k) = x_{\mathrm{R}_n}(k) - x_{\mathrm{R}_j}(k) \\ y_{\mathrm{R}_n}^{\mathrm{R}_j}(k) = y_{\mathrm{R}_n}(k) - y_{\mathrm{R}_j}(k) \end{cases} \quad (13)$$

e no que diz respeito a um obstáculo O_l (onde $1 \leq l \leq \mathrm{NO}$, onde NO é o número total de obstáculos), que é definida como:

$$P_{\mathrm{R}_n}^{\mathrm{O}_l}(k) = \begin{cases} x_{\mathrm{R}_n}^{\mathrm{O}_l}(k) = x_{\mathrm{R}_n}(k) - x_{\mathrm{O}_l}(k) \\ y_{\mathrm{R}_n}^{\mathrm{O}_l}(k) = y_{\mathrm{R}_n}(k) - y_{\mathrm{O}_l}(k). \end{cases} \quad (14)$$

É importante mencionar que na evolução do estado do obstáculo, todos os obstáculos (em movimento ou estáticos) são considerados como tendo uma velocidade zero naquele instante de tempo.

Finalmente, a fim de modelar a evolução da quantidade total de incerteza no que diz respeito à posição relativa entre o robô e o alvo, um modelo de covariância foi criado. Este modelo depende do tipo de câmara usada (tais como câmera omnidirecional, câmera olho de peixe, câmeras diretas normais). No entanto, o modelo pode facilmente ser mudado usando o código do controlador, permitindo que cada robô possua um sensor distinto. O modelo de covariância de um robô em Rn em um instante k é dada por (15):

$$\Sigma_{\mathrm{R}_n}(k) = \begin{bmatrix} \sigma_{d_\mathrm{t}}^2 & \rho\sigma_{d_\mathrm{t}}\sigma_\phi \\ \rho\sigma_\phi\sigma_{d_\mathrm{t}} & \sigma_\phi^2 \end{bmatrix} \quad (15)$$

Onde σ_{dt} é a variação da distância do alvo d_t e σ_φ é a variância da medição de rolamento do alvo. ρ é o coeficiente de correlação. Supõe-se que as medições não são correlacionadas e que $\rho = 0$.

No caso dos robôs 5dpo, uma observação empírica do modelo de covariância (16) foi criada, no qual, a variação da distância do alvo é diretamente proporcional ao quadrado da distância e a variância do rolamento do alvo é inversamente proporcional à distância do alvo. Este modelo de observação foi validado com diversos experimentos, onde também foi possível encontrar os valores de K_a e K_b:

$$\Sigma_{\mathrm{R}_n}(k) = \begin{bmatrix} K_a d_\mathrm{t}^2 & 0 \\ 0 & K_b \dfrac{1}{d_\mathrm{t}} \end{bmatrix}. \quad (16)$$

Além disso, é necessário representar a observação de covariância (16) em sua forma canônica nas coordenadas cartesianas centrado em *O* devido à necessidade de amenizar as covariâncias que se unem decorrente dos companheiros de equipe. Portanto, a representação canônica do modelo de covariância na direção do alvo e na sua direção perpendicular é dada por (17):

$$\Sigma_{\mathrm{R}_n}^{\perp}(k) = \begin{bmatrix} K_1 d_{\mathrm{t}}^2 & 0 \\ 0 & K_2 d_{\mathrm{t}} \end{bmatrix} = \begin{bmatrix} {\sigma_x}^2 & 0 \\ 0 & {\sigma_y}^2 \end{bmatrix} \tag{17}$$

onde K_1 = Ka e K_2 = (Kb + KaKb) são constantes de proporcionalidade.

A junção da covariância é realizada utilizando a formulação de Gouvea [22]. Na formação, a covariância de cada companheiro do robô é $\sum_{Rj}^{\perp}(k)$ é também preditiva. A covariância de predição dos colegas de equipe também é utilizada no quadro do robô e depois se uni como pelo método apresentado por Smith [29], que estima a relação nominal e o erro esperado (covariância) entre quadros (*frames*) de coordenadas, que representam as localizações relativas dos objetos. Portanto, se for dado N robôs com a formação, então Rn é o robô que vai prever as covariâncias de formação e os robôs Rj são os companheiros de equipe com $1 \leq j \leq NM$ e $NM = N - n$ são o número de companheiro em formação. A junção da covariância pode ser descrita por:

$$\begin{aligned} \Sigma_{\mathrm{Merged}}^{\perp}(k) = {} & ([\Sigma_{\mathrm{R}_n}^{\perp}(k)]^{-1} + [\hat{\Sigma}_{\mathrm{R}_{j=1}}^{\perp}(k)]^{-1} \\ & + \cdots + [\hat{\Sigma}_{\mathrm{R}_{j=NM}}^{\perp}(k)]^{-1})^{-1} \end{aligned}$$

onde $\sum_{Rj}^{\perp}(k)$ é a covariância das matrizes dos companheiros de equipe em relação ao robô Rn.

Nenhum ruído foi introduzido durantes as simulações dos experimentos. Portanto, os meios de estimativas da observação dos companheiros de equipe são idênticos, enquanto que a incerteza em torno da observação de cada companheiro de equipe é formulada de acordo com (17) para a fusão.

Em relação ao modelo de formação, basicamente é necessário prever a posição do robô em relação aos colegas que participam da formação, já o modelo do alvo calcula o movimento do alvo, a distância entre o robô e os obstáculos, a posição do robô em relação ao alvo e a posição do robô em relação à formação [11]. Finalmente, a fim de modelar a evolução da quantidade total de incerteza, no que diz respeito à posição relativa entre o robô e o alvo, um modelo de covariância foi criado e previamente testado com resultados demonstrados no trabalho feito por Ahmad [12].

2.4 Arquiteturas computacionais utilizadas

Um conceito de arquitetura utilizado na robótica são os sistemas embarcados críticos, como é o caso dos robôs móveis autônomos em situações difíceis, que possuem requisitos específicos como heterogeneidade, adaptabilidade e confiabilidade [31]. Em alguns casos, os sistemas embarcados em robôs são formados por uma vasta quantidade de periféricos e módulos eletromecânicos, requerendo ainda mais poder computacional e maior complexidade para o controle do sistema. As aplicações também são intrinsecamente paralelas aumentando ainda mais a complexidade. Nesse contexto, a adaptabilidade é uma funcionalidade chave, já que ela permite que o sistema se autoconfigure buscando mais confiabilidade, segurança e autonomia. Um caso especial de tais sistemas são os sistemas computacionais autônomos [32], que são aqueles capazes de gerenciar automaticamente suas funcionalidades a fim de prover adaptação e ciência de todo seu funcionamento. Modelar, projetar e desenvolver tais sistemas é um desafio [33].

Os sistemas embarcados apresentam características em comum com sistemas computacionais de propósitos gerais, mas não possuem a mesma uniformidade e cada aplicação pode apresentar requisitos diferentes de desempenho, consumo de potência e área ocupada, o que vai acarretar em uma combinação distinta de módulos de *hardware* e *software* para atender estes requisitos [34]. Este tipo de sistema computacional é extremamente amplo e envolve inúmeros temas, o que torna uma área quase integral. Temas como questões de portabilidade, limite de consumo de potência sem perda de desempenho, a baixa disponibilidade de memória, a necessidade de segurança e confiabilidade, a possibilidade de funcionamento em uma rede maior e o curto tempo de projeto [35] são bastante discutidos. Segundo Giovanni [36], a arquitetura de *hardware* de um sistema embarcado pode conter um ou mais processadores, memórias, interfaces para periféricos e blocos dedicados. Os componentes são interligados por uma estrutura de comunicação que pode variar de um barramento a uma rede complexa, utilizando de vários modelos de arquitetura, como processadores, arquitetura reconfiguráveis, GPU e DSP.

Muitas vezes as arquiteturas eletrônicas para robótica utilizam de um sistema operacional de tempo real, chamados de RTOS (*real-time operating system*) que fornece serviços de comunicação e escalonamento de processos [37]. Um fator importante é a definição dos serviços que serão fornecidos pelo RTOS, como também, a escolha da forma de comunicação (*software-software*, *software-hardware* e *hardware-hardware*) e dos mecanismos (memória compartilhada, troca de mensagens, acesso direto à memória, etc.) que representa um dos aspectos mais importantes em um projeto de sistemas embarcados [34].

Os sistemas computacionais embarcados já estão presentes na vida das pessoas, seja como chips que controlam a funções de um carro, de um eletrodoméstico ou de um telefone. Com os baixos custos tecnológicos atuais, estes tipos de sistemas tendem a aumentar ainda mais sua presença no cotidiano das pessoas [38]. Atualmente é fornecida várias placas de desenvolvimento no mercado e duas delas serão apresentadas a seguir.

2.4.1 Beaglebone Black (BBB)

BeagleBone Black (Figura 6) é uma placa embarcada desenvolvida pela Circuit Corporation LLC. Possui um processador ARM Cortex-A8 de 1 GHZ, 512 MB de memória RAM, versões com 4GB e 8GB de armazenamento em flash, pesa em média 40 gramas, possui dimensões de 86x53mm e um custo inferior as demais concorrentes. Atualmente a BBB é utilizada na robótica para impressão 3D, monitoramento de ambientes e entre outros [39].

Com o intuito de fomentar a viabilidade de utilização de placas embarcadas como possível solução para tais dificuldades, foram identificados alguns trabalhos relacionados, como o de Pei-pei Ni [40] que utiliza a BBB como computador principal de um sistema de controle de voo autônomo de alto desempenho. Este sistema possui um conjunto de loops internos e externos que aumentam a capacidade do Robô de realizar tarefas complexas permitindo-lhe autonomia. A conclusão deste trabalho informa que os algoritmos embarcados de controle de altitude têm um desempenho estável.

O trabalho de Huihua Zhao [41], utiliza a BBB como sistema embarcado para um controlador de um robô bípede. Este robô é utilizado em próteses de membros inferiores de um ser humano, portanto, exigindo desenvolvimento sinérgico de sensoriamento e algoritmos de otimização em tempo real. Por fim, o artigo conclui que o uso da BBB foi satisfatório.

Também foram identificados trabalhos relacionados ao processamento em tempo real para áudio, como em Topliss [42] e McPherson [43]. No trabalho de Topliss [42], foram realizados testes destinados a avaliar a capacidade de resposta da placa BeagleBone Black em aplicações de áudio interativo de tempo real. A análise dos dados mostra que a BBB possui uma notável capacidade de resposta, segundo o autor, a maioria das configurações testadas são afetados por menos de 7 milissegundos de latência.

O Trabalho de McPherson [43], apresenta o BBB como um ambiente para processamento de latência de dados de áudio, segundos o teste apresentado foi possível atingir latências até 80 microssegundos, tornando a plataforma adequada para as mais exigentes tarefas de áudio de

baixa latência. Portanto diante das referências encontradas, foi utilizado a BeagleBone Black como possível arquitetura de baixo custo para esta pesquisa.

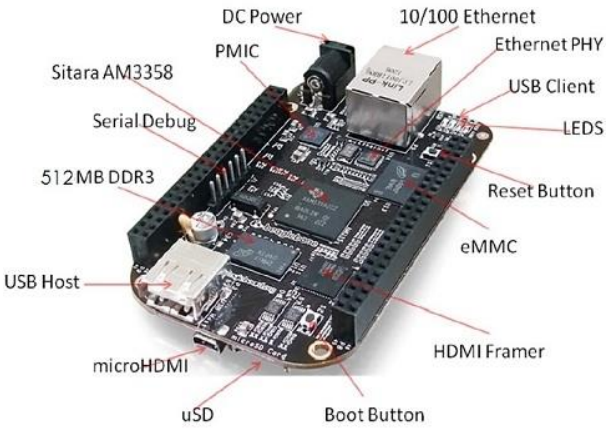


Figura 6: Placa de baixo custo BeagleBone Black.

2.4.2 UDOO

UDOO (Figura 7) é uma placa embarcada que disponibiliza várias possibilidades de aplicações, seja utilizando um sistema operacional Linux ou Android, adicionando módulos com processadores Arduino ou simplesmente para fins empresariais. A UDOO foi projetada para projetos que demandam um processamento alto, com seus quatro núcleos ARM CORTEX-A9 de 1GHz cada, sendo comumente comparado a um minicomputador.

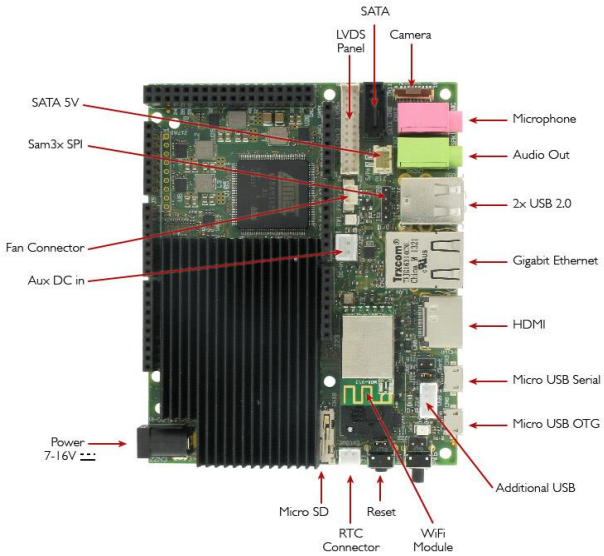


Figura 7: Placa de baixo custo UDOO

Como na BBB, também foi possível selecionar alguns trabalhos, que utilizaram e obtiveram sucesso ao uso da placa QuadCore UDOO para sistemas semelhantes ao deste trabalho. No trabalho de Gongora e Gonzalez [44] foi utilizado a UDOO como uma plataforma embarcado de um sistema de controle de voo autônomo para Multicopteros, tal sistema também possui um modulo de localização baseado em visão, utilizando de câmeras fixas externas ao robô. O objetivo é utilizar uma placa de baixo custo para o processamento de todos os cálculos do Multicoptero. Após todas os testes foi constatado que o uso da UDOO foi o suficiente para executar o *software* sem a necessidade de nenhuma adaptação.

No trabalho de Neves e Trocado [45] foi construído um robô para ser usado em tarefas de monitoramento, tais como, detecção de vegetação e mapeamento de plantações. Para que este trabalho fosse desempenhado, foi utilizado a UDOO para processamento dos dados de uma câmera RGB e de um sinal de encode que utiliza de um módulo Arduino acoplado a própria UDOO para publicar a velocidade do robô através da mensagem de encode. Em relação a BeagleBone Black, a UDOO apresenta menor indícios do uso na robótica, porém, os poucos existentes demonstram a eficiência desta placa e alinhado ao seu poder de processamento, também foi escolhida para comportar o controlador preditivo usado nesse trabalho.

Diante das constatações encontradas, percebesse que placas de baixo custo podem ser utilizadas como arquitetura embarcada de aplicações que necessitam de um alto poder de processamento.

2.5 Resumo do Capítulo

Neste capítulo foi apresentado uma sumarização sobre os principais temas que estão relacionadas a esta pesquisa. Inicialmente foi apresentado uma breve revisão sobre sistemas multi-robôs e suas características. Além disso, foi sumarizado que robôs móveis devem ser capazes de mover-se de forma eficiente, evitando colisões e atingindo o seu objetivo, com isso, várias tarefas importantes são desempenhadas por estes múltiplos robôs, como transporte de grandes objetos ou ampla cobertura de uma área. Também foi demonstrado uma descrição inicial do robô 5dpo, construído pela equipe de futebol de robô da Faculdade de Engenharia da Universidade do Porto (FEUP) e utilizado como base desta pesquisa.

No que toca os softwares utilizados para o controle e simulação do robô, foram realçados os três principais sistemas, chamados de RoC, WatchTower e SimTwo. No qual, sucintamente, o RoC é o responsável por colher os dados do ambiente e controlar cada robô individualmente;

Em seguida, o watchTower recebe os dados colhidos de cada RoC e determina orientações que devem ser adotadas por cada robô; por fim, o SimTwo, também desenvolvida pela FEUP, é uma ferramenta para construção de simulações em 3D e utilizada nessa pesquisa como substituto de robô um real e assim possibilitando todos os testes apresentados no decorrer desse trabalho.

Relativamente aos controladores de formação, foram referenciadas algumas das soluções mais encontradas e mais pertinentes sobre este assunto, como também principais problemas, função e objetivos na utilização deste controlador. Ainda na seção de controle de formação também foi discutido a divisão da estrutura do modelo preditivo do robô, chamado de NMPFC (*Nonlinear Model Predictive Formation Controller)* e conceitos como horizonte de predição e horizonte de controle, essências para o entendimento do restaste deste trabalho. Por fim, é exposto o conceito de arquitetura computacional embarcada, sua utilização nos dias atuais e diferenças entre este tipo de sistema e os sistemas de propósito geral, encerrando com a propostas de placas utilizadas nesta pesquisa, conhecidas como BBB (*BeagleBone Black*) e a UDOO.

Capítulo 3

Resultados e Discussão

Este capítulo tem o objetivo de apresentar as etapas e os resultados alcançados com a execução deste projeto. Na execução dos testes, foi utilizado modelo do robô 5dpo (Figura 8) e este foi implementado no simulador, previamente ajustado e validado por Nascimento [46]. O 5dpo foi construído pela equipe de futebol de robô da FEUP (Faculdade de Engenharia da Universidade do Porto - Portugal), com o objetivo de participar da liga média da *Robocup* [15]. Estes são robôs omnidirecionais de três rodas, com uma altura de 80 cm e um diâmetro máximo de cerca de 50 cm [26]. Atualmente, cada robô (real) possui um laptop que executa os algoritmos de controle, fazendo com que todos os robôs sejam parte de uma rede local que lhes permite trocar informações com a máquina central (WatchTower) e receber ordens da mesma. Para este trabalho, os robôs da liga média 5DPO foram utilizados como plataforma de teste, simuladas, para os controladores utilizados nesta dissertação.

Serão realizados essencialmente testes em simulação, utilizando o software desenvolvido na faculdade de Engenharia FEUP pelo Professor Paulo Costa, SimTwo [47]. (Ver Figura 9). Os algoritmos de controle serão adaptados sobre o software já desenvolvido, nomeadamente, a aplicação RoC é utilizada para o controle individual de cada robô e o WatchTower, para controle e gestão da equipe completa.

Figura 8: Robô 5dpo

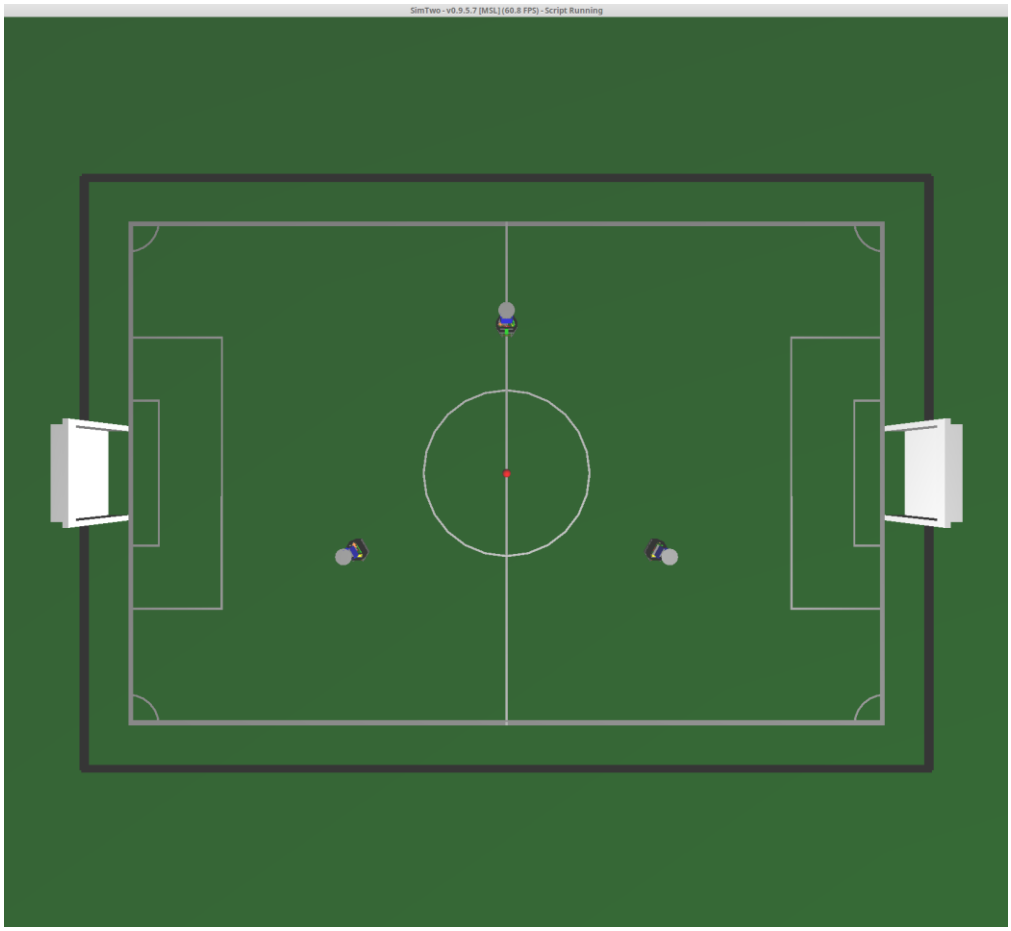


Figura 9: Tela principal do SimTwo.

Duas métricas são utilizadas para identificar o avanço das otimizações em função do controlador do robô, são elas, horizonte de predição (horizonte sobre o qual o agente prediz o comportamento autônomo do sistema) e o horizonte de controle (horizonte sobre o qual o agente encontra as ações). Estas métricas proporcionam estimativas da eficiência da técnica aplicada e são representadas por instantes de tempo (ciclos, onde cada ciclo é executado a cada 40 milissegundos).

Sendo assim, duas principais estratégias caracterizam a relação entre o horizonte de predição e controle: (I). Utilizando o modelo do sistema, as saídas do mesmo são previstas para o **horizonte de predição** definido. Estas saídas dependem do estado atual, do passado do sistema e dos sinais de controle futuros; (II). O conjunto de saídas futuras é calculado para um **horizonte de controle** através da minimização de uma função custo. Esta função custo expressa um determinado critério tal que a saída do processo seja tão próxima de uma referência quanto possível;

Dessa forma, estas métricas possuem relação com o custo computacional, pois quanto maior o custo computacional disponível, mais rápido será realizado o processamento dos algoritmos de predição e controle e com um menor índice de *overshoots* (erros) nos resultados gerados, consecutivamente, menor poderá ser seus horizontes. Em formação, isso resulta em menor tempo de convergência entre os robôs.

Portanto, o **modelo do robô** prevê a evolução do seu estado (posição e orientação no mundo) para um determinado intervalo de tempo como resposta a diversas entradas de controle. Este estado do modelo, referente ao robô, é atualizado a cada ciclo do algoritmo de controle para o estado atual do robô, então, para que isso seja possível, uma **função custo** é utilizada para comparar a evolução deste estado, prevista pelo modelo do robô, com a trajetória desejada, calculando um custo associado a cada conjunto de entradas de controle e penalizando essencialmente as diferenças entre a posição e orientação do robô previstas e as desejadas para cada instante de tempo. Da minimização desta função custo encontram-se as entradas de controle ótimas que são aplicadas ao robô. Diante disso, seria mais desejável a utilização de **horizontes de controle e predição infinitos** de modo a garantir a minimização dos critérios de penalização definidos pela função custo (J) e a obtenção dos sinais de controle ótimos. No entanto, uma vez que a otimização tem que ser realizada em tempo real recorrendo a métodos numéricos é importante que se usem horizontes temporais finitos. Com isso, percebe-se uma relação harmoniosa entre o modelo do robô, a minimização da função custo e os valores temporais dos horizontes de predição e controle [26].

Em suma, quando aplicados a robótica, horizontes de predição demasiados pequenos resultam em grandes *overshoots* (erros) e horizontes demasiados grandes resultam em curvas exageradas durante a formação. É preciso encontrar uma adequação equilibrada entre os horizontes de predição e controle, baseando-se na disponibilidade do poder computacional de cada arquitetura ou realizando otimizações nos algoritmos de controle e predição.

3.1 Análise dos modelos de predição do NMPFC.

O primeiro experimento teve como objetivo identificar qual o modelo de predição mais equilibrado para o robô, ou seja, que estimule a convergência em menos tempo e minimize a função custo do controlador, aplicado para controlar a formação de uma equipe de robôs móveis omnidirecionais. O modelo de predição calcula comportamentos de futuras formações com re-

lação a obstáculos, companheiros de equipe em formação, alvo, orientações, posição na formação e controle do esforço utilizando um modelo para prever a maioria dos termos de formação. No entanto, a previsão dos robôs em formação pode ser realizada tanto pelo modelo dinâmico quanto pelo modelo cinemático.

Duas simulações foram feitas para avaliar o modelo (NMPFC) do controlador com diferentes modelos de predição para o movimento dos robôs. Estas simulações foram realizadas utilizando o simulador SimTwo [47] e teve como objetivo visualizar a convergência durante a formação do robô. A principal diferença entre as simulações é a dificuldade na posição inicial, em que a segunda simulação (Gráfico 3) coloca os robôs em um cenário inicial pior do que a primeira simulação (Gráfico 2). Na simulação 1, os robôs possuem suas posições iniciais em (3.5, -1.8) para o robô 1, (-3.5, -1.8) para o robô 2, (0, 3.5) para o robô 3 e (0,0) para a posição da bola, entretanto, na simulação 2 os robôs possuem suas posições iniciais em (-5.5, 0) para o robô 1, (-4.5, 0) para o robô 2, (-3.5, 0) para o robô 3 e (0,0) para a posição da bola. Ambas as simulações foram realizadas três vezes cada (Gráfico 4 e
Gráfico 7), para avaliar três comportamentos de minimização diferentes.

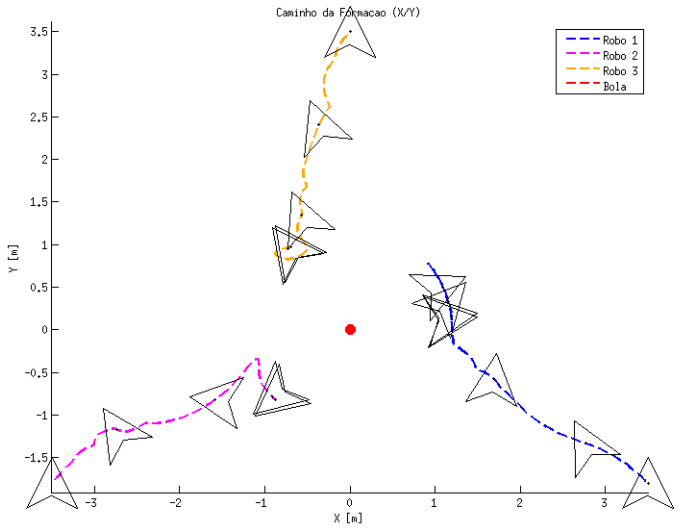


Gráfico 2: Trajetória da Simulação 1

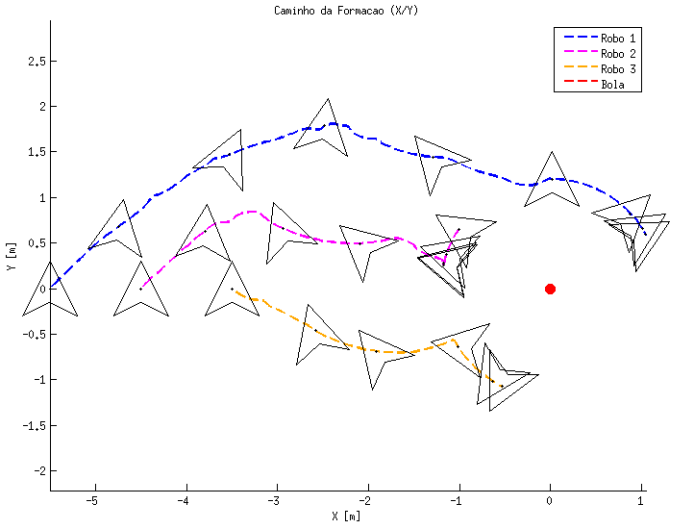


Gráfico 3: Trajetória da Simulação 2

Diante disso, o primeiro comportamento utilizou um modelo de previsão cinemático (Gráfico 4 - letra a) para prever os movimentos do robô e de seus companheiros de equipe. No segundo comportamento (Gráfico 4 - letra b) utilizou um modelo de previsão dinâmico para prever os movimentos do robô e cinemático para seus companheiros de equipe. Por fim, no terceiro comportamento (Gráfico 4 - letra c) utilizou o modelo dinâmico para prever o movimento de todos os robôs em formação. Para todos os testes foram utilizados os limites de predição inicial igual a 1 e final igual a 7 e o horizonte de controle igual a 2.

Como parte dos resultados, durante a simulação 1 (Gráfico 2), o terceiro caso (Gráfico 4 - letra c), em que é aplicado o modelo dinâmico para prever o movimento de todos os robôs, apresenta a pior minimização da função custo. Para base de entendimento, no Gráfico 4, o "J" representa o valor da função custo, que essencialmente representa a soma das saídas futuras calculada para um horizonte de controle através da otimização da função custo. Esta função custo expressa como determinado critério, que a saída do processamento seja a menor possível, assim, do resultado da minimização surge os melhores valores de controle [26].

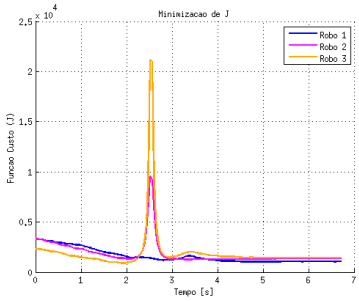
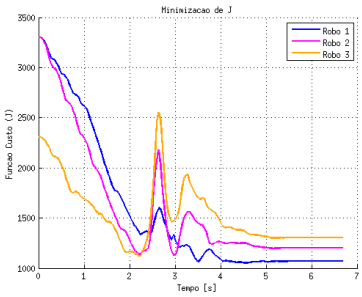
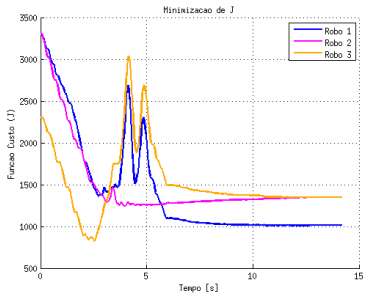
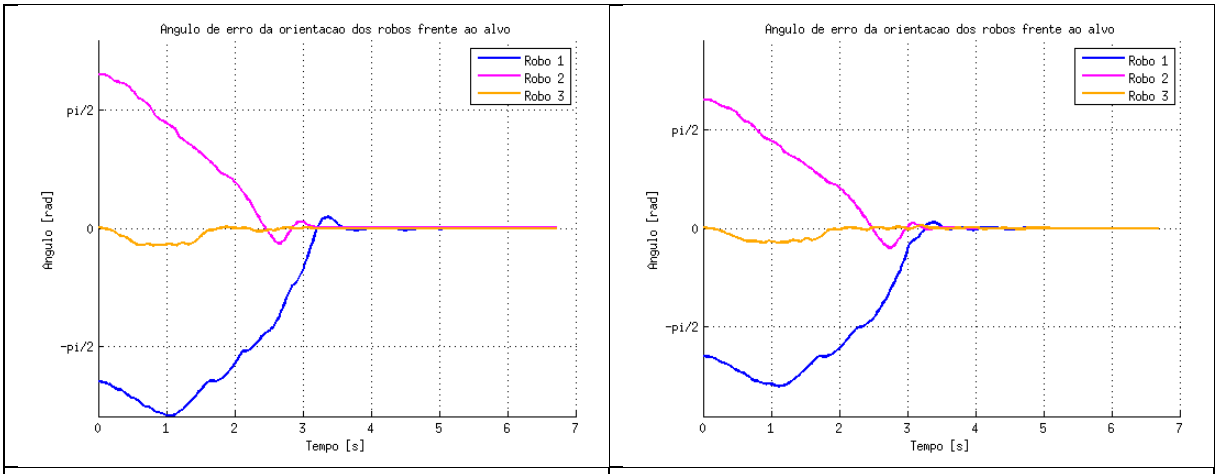
		
(a) Apenas modelo cinemático	(b) Modelo dinâmico para o robô e cinematico para os colegas.	(c) Modelo dinâmico para todos os robôs.

Gráfico 4: Minimização da função custo da Simulação 1.

Diante disso, nota-se que a minimização da função custo do robô foi realizada em 6 segundos. No entanto, nos outros dois casos, a minimização da função custo foi realizada em 5 segundos. Isso expressa à influência do custo computacional na teoria do modelo de controle preditivo. Esta influência determina que o aumento da complexidade do modelo, aumenta também o custo computacional para calcular a saída de referência. Assim, a convergência é realizada em mais tempo do que quando se utiliza um modelo simples. Além disso, com modelos mais complexos, o horizonte de predição também é afetado, necessitando ser diminuído a fim de processar os cálculos em um tempo hábil, afetando por sua vez, os resultados do controlador.

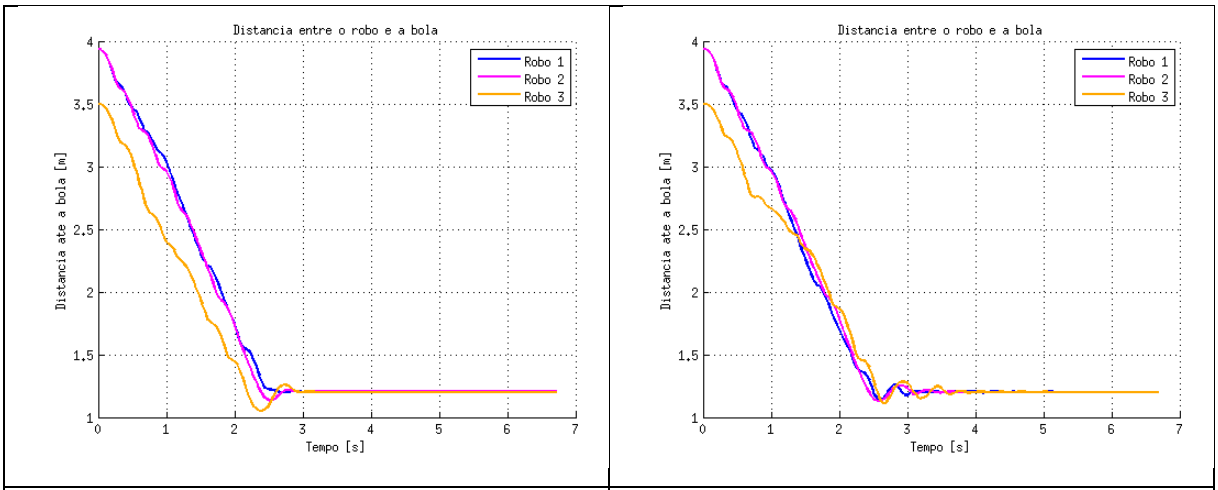
O caso em que apenas foi utilizado o modelo cinemático (Gráfico 4– letra a) converge aproximadamente no mesmo tempo que o caso em que foi utilizado o modelo dinâmico para prever o comportamento do robô e o modelo cinemático para prever o comportamento dos colegas (Gráfico 4– letra b) (cerca de 4 segundos). Finalmente, é apresentado os gráficos angulares (Gráfico 5) e os gráficos de distância entre o robô e a bola (Gráfico 6) dos dois primeiros casos (Cinemático para o robô e para os colegas e dinâmico apenas para o robô), mostrando mais uma vez que os dois modelos convergem quase que ao mesmo tempo.



(a) Apenas modelo cinemático

(b) Modelo dinâmico para o robô e cinematico para os colegas.

Gráfico 5: Ângulo de erro da orientação dos robôs frente para ao alvo da Simulação 1.



(a) Apenas modelo cinemático

(b) Modelo dinâmico para o robô e cinematico para os colegas.

Gráfico 6: Distância entre o robô e a bola da Simulação 1.

Na simulação 2 (Gráfico 3), novamente o terceiro caso (onde é aplicado o modelo dinâmico para prever o movimento de todos os robôs -

Gráfico 7 letra c) apresenta os piores resultados de minimização. A minimização da função custo dos robôs foi alcançada em 8 segundos, enquanto nos outros dois casos, a minimização da função custo foi realizada em torno de 6 segundos. Pode ser visto também que o caso em que apenas o modelo cinemático foi utilizado, converge aproximadamente ao mesmo tempo do caso em que o modelo dinâmico apenas para o robô e o modelo cinemático para os colegas.

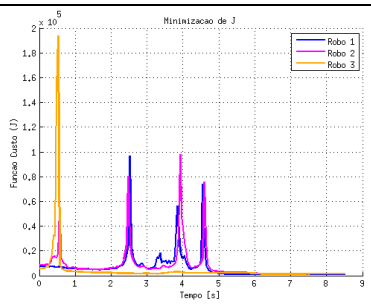
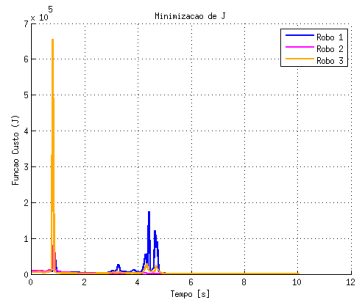
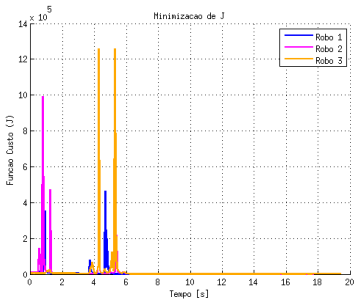
		
(a) Apenas modelo cinemático	(b) Modelo dinâmico para o robô e cinematico para os colegas.	(c) Modelo dinâmico para todos os robôs.

Gráfico 7: Minimização da função custo da Simulação 2.

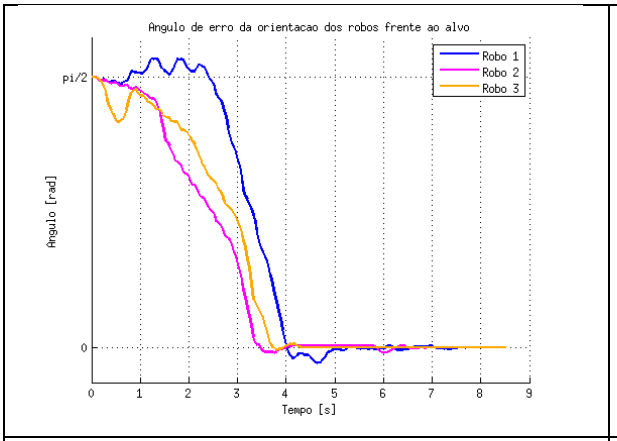
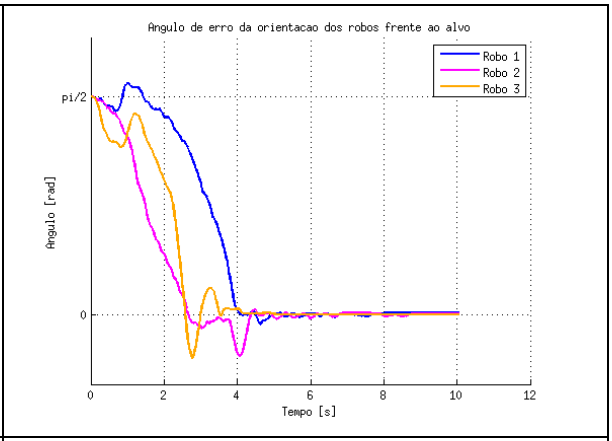
	
(a) Apenas modelo cinemático	(b) Modelo dinâmico para o robô e cinematico para os colegas.

Gráfico 8: Ângulo de erro da orientação dos robôs frente para ao alvo da Simulação 2.

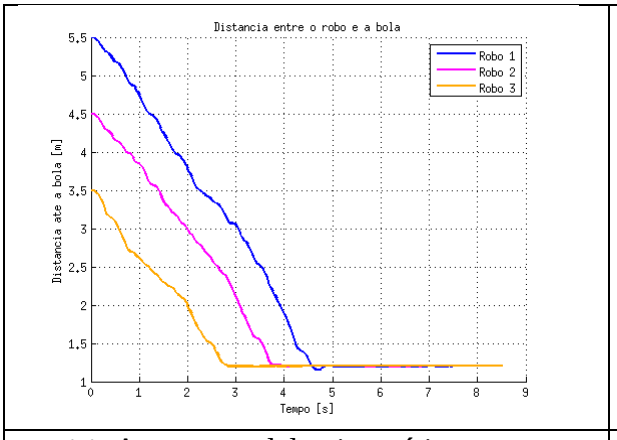
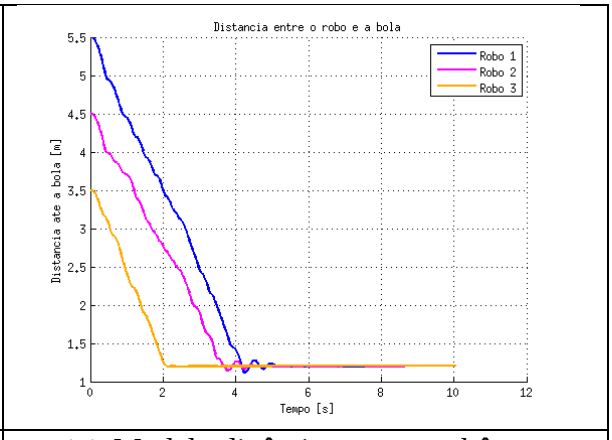
	
(a) Apenas modelo cinemático	(b) Modelo dinâmico para o robô e cinematico para os colegas.

Gráfico 9: Distância entre o robô e a bola da Simulação 2.

Como resultado final deste experimento, foi constatado que o melhor modelo para prever seu comportamento é aquele que usa apenas o modelo cinemático para o robô e para os seus colegas de equipe. Isto significa um menor custo computacional, que por sua vez, minimiza a função de custo em menos tempo. As simulações também mostraram que usando apenas o modelo cinemático não põe em risco a convergência da formação. Apesar da segunda simulação (Gráfico 3) coloca os robôs em um cenário inicial pior do que a primeira simulação (Gráfico 2), os resultados comparativos dos três comportamentos foram proporcionais aos da Simulação 1. Finalmente, estes resultados podem ser generalizados. Isto significa que outras condições iniciais, tais como diferentes cenários de convergência, diferentes funções de custo, robôs e assim por diante, vão apresentar os mesmos resultados. Isto é explicado pelo fato da complexidade do modelo em si afetar o desempenho do controlador, através do aumento ou da diminuição dos seus cálculos.

3.2 Testes com a BeagleBone Black (BBB)

Na primeira tentativa de utilizar o NMPFC em uma placa embarcada, foi utilizada a BeagleBone Black, nela foram inseridos os mesmos horizontes de predição (7) e controle (2) utilizados pelo Laptop, porém pelo baixo poder de processamento desta placa, não foi possível o robô convergir e manter seu estado de convergência. Portanto, a fim de identificar o melhor horizonte de predição e controle na BBB, foram realizadas algumas combinações. Cada horizonte de controle foi comparado a horizontes de predição em ordem decrescente, como descritas na Tabela 1.

Tabela 1: Demonstração do Horizonte de Predição e Horizonte de Controle na BBB.

Teste	**Predição**	**Controle**	**Converge?**
#1	7	2	Não
#2	6	2	Não
#3	5	2	Não
#4	4	2	Não
#5	3	2	Não
#6	2	2	Não
#7	1	2	Não
#8	7	1	Não
#9	6	1	Não
#10	5	1	Não
#11	4	1	Não
#12	**3**	**1**	**Sim**

Sem nenhuma modificação no controlador, os melhores horizontes de predição e controle encontrados na BeagleBone Black foram de 3 e 1 respectivamente. Apesar do robô convergir utilizando a BBB, o desempenho atingido não foi satisfatório, como visto em 3.2.2, onde o tempo de convergência foi maior se comparado do mesmo controlador executado em um Laptop, o que demonstra que a parametrização dos horizontes de predição e controle afeta o desempenho do controlador.

3.2.1 Analise de uso de processamento

Foram realizados dois testes de processamento, visualizados no Gráfico 10. Cada teste possui dois cenários. O primeiro cenário o *hardware* ficou executando apenas os processos padrões do sistema operacional e o segundo cenário o *hardware* ficou executando os processos padrões do sistema operacional junto com o controlador (RoC). No primeiro cenário do teste de processamento com o PC foram obtidos os dados em uma situação considerada limpa, onde a placa encontra-se ligada, sem nenhum dispositivo conectado e executando apenas os processos do sistema operacional. Após os testes, o laptop executando apenas seus processos ocupou entre 0.5% a 2.6% do poder computacional total da CPU. Ao acionar o controlador (RoC), o processamento variou entre 6.6 % a 14.9 % da CPU, um aumento de 12.3% ao executar o controlador.

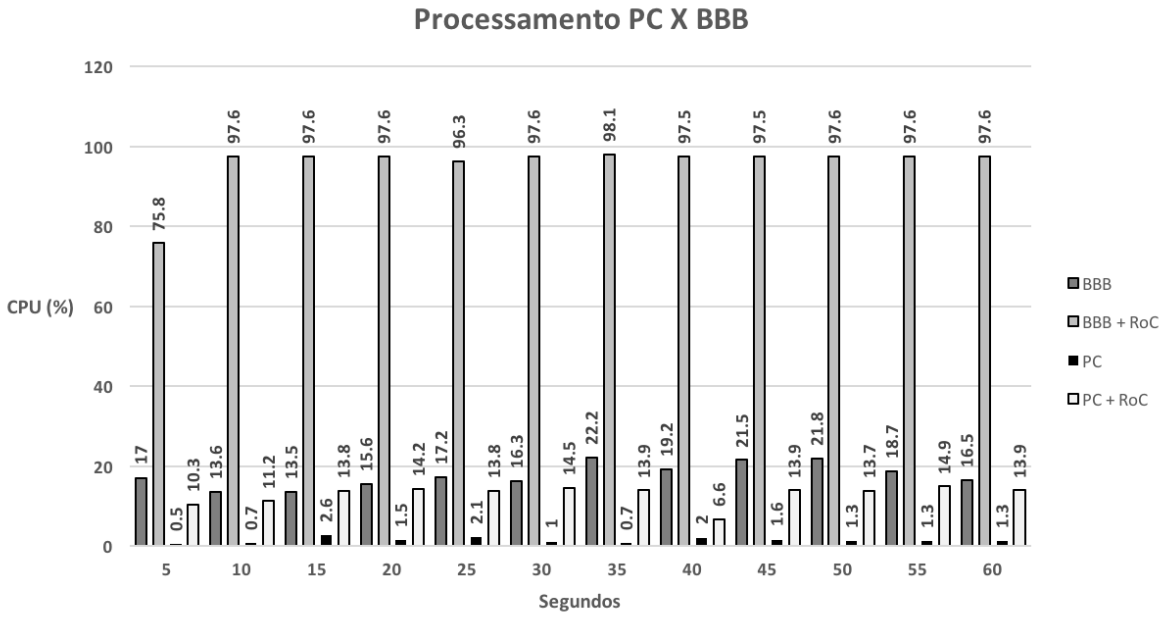


Gráfico 10: Comparação do uso de processamento do PC contra o do BBB.

No segundo teste, utilizando o BeagleBone Black conforme visto no Gráfico 10, foi obtido o gasto de processamento executando apenas os processos do próprio BBB, o que resultou em um custo entre 13.5% e 22.2%. Ao acionar o controlador (RoC), o uso de CPU variou entre 75.8% e 98.1%, um aumento de 75.9% ao executar o controlador. Para os testes aqui realizados foi utilizado um laptop (PC) com processador Intel I5-2520M, 2.50 GHz de 4 núcleos e 4GB de memória RAM.

Como resultado, o uso do controlador embarcado no BeagleBone Black apresentou um custo de processamento muito alto em comparação com o mesmo controlador executando no PC. Nos testes realizados, o PC executando o controlador possuiu um aumento de processamento de 12.3%, já o mesmo controlador no BBB teve um aumento de 75.9% ao executar o controlador, o que reforça, conforme os estudos realizados, o baixo poder de processamento deste tipo de *hardware*.

3.2.2 Analise de desempenho

Além de uma análise de processamento, devemos analisar qual o impacto do uso do controlador de formação em um sistema embarcado, ou seja, o objetivo deste teste é verificar se o baixo processamento da BBB afetaria o desempenho do NMPFC. Para isso, foi embarcado em uma placa BeagleBone Black (BBB) o controlador (RoC).

Para que fosse possível o robô convergir utilizando o controlador embarcado na BeagleBone Black, algumas modificações tiveram que ser feitas. (i) Como apresentado anteriormente, o robô que utiliza o controlador embarcado na BBB, apenas converge com os horizontes de predição e controle igual a 3 e 1, respectivamente, portanto, foi necessário diminuir os horizontes de predição e controle, para que o processamento fosse feito em um tempo hábil, possibilitando o robô a tomar as decisões corretas; (ii) Desativar da função custo o cálculo referente à predição dos colegas. Quando o robô recebe os dados dos colegas, torna a função custo mais complexa, exigindo um maior poder de processamento e impossibilitando o robô de convergir.

Duas simulações foram feitas para avaliar o uso do controlador (RoC). Uma utilizando a arquitetura embarcada (BBB) e outra utilizando a arquitetura de propósito geral (PC). Estas simulações foram realizadas utilizando o simulador SimTwo [47] que tem e teve como objetivo simular e visualizar a convergência durante a formação do robô. O Gráfico 11, Gráfico 13, Gráfico 15, Gráfico 17 e, e demonstram os resultados da simulação utilizando a arquitetura de baixo custo PC, já os representam os resultados obtidos utilizando o BBB.

Como parte dos resultados, durante a simulação com a BBB, é apresenta a pior minimização da função custo (Gráfico 14). Diante disso, nota-se que a minimização da função custo do robô foi realizada em torno de 8 segundos. No entanto, no caso do controlador utilizando o PC, a minimização da função custo foi realizada em 5 segundos. Novamente, isso expressa à influência do poder se processamento, na conclusão da convergência do robô em formação. Esta influência determina que o baixo do poder de processamento, aumenta o tempo para calcular a saída de referência. Assim, a convergência é realizada em mais tempo do que quando se utiliza um poder computacional suficiente. Com isso, o horizonte de predição também é afetado, necessitando ser diminuído a fim de processar os cálculos em um tempo hábil, afetando por sua vez, os resultados do controlador.

A simulação em que foi utilizado o Laptop (PC) converge em menos tempo que o caso em que foi utilizado a placa de baixo custo. A uma distância de 4 metros o robô levou em média 5 segundos para convergir com o controlador embarcado no PC (Gráfico 11) e a mesma distância de 4 metros o robô levou em média 8 segundos para convergir, com o controlador embarcado na BBB (Gráfico 12). Finalmente, é apresentado os gráficos angulares (Gráfico 17 e Gráfico 18), demonstrando o tempo levado para minimizar o ângulo de erro da orientação dos robôs frente para o alvo, nesse caso, a bola. Com isso, verificando mais uma vez que o controlador embarcado no Laptop, minimiza os erros em menos tempo, em torno de 5s para o PC e de 8s para a arquitetura embarcada.

Como resultado final da análise inicial de desempenho, foi constatado que a melhor arquitetura para prever o comportamento é a de propósito geral, representada pelo PC. Isto significa um menor custo computacional, que por sua vez, minimiza a função de custo em menos tempo.

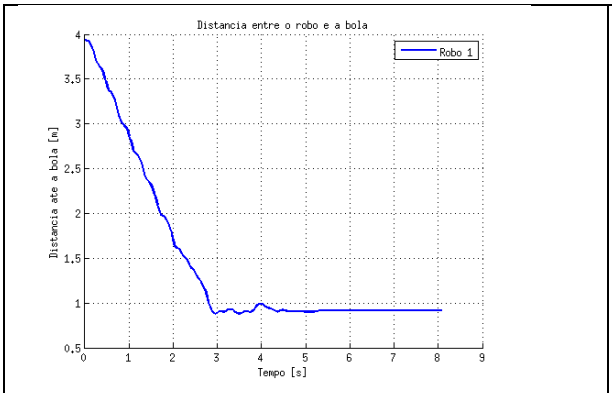


Gráfico 11: Tempo de convergência entre o robô e a bola no PC

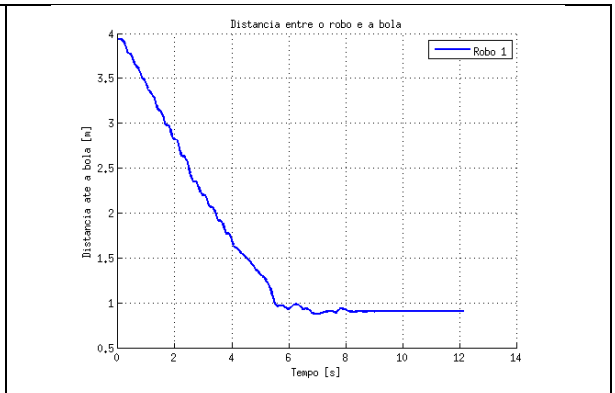


Gráfico 12: Tempo de convergência entre o robô e a bola no BBB

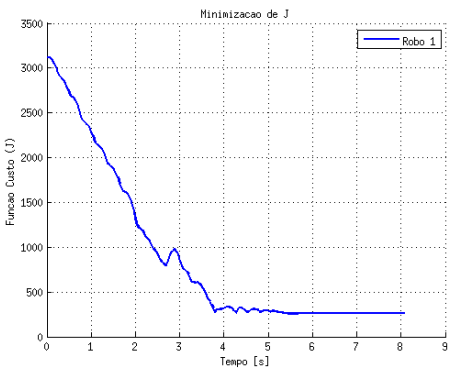


Gráfico 13: Minimização da função custo no PC

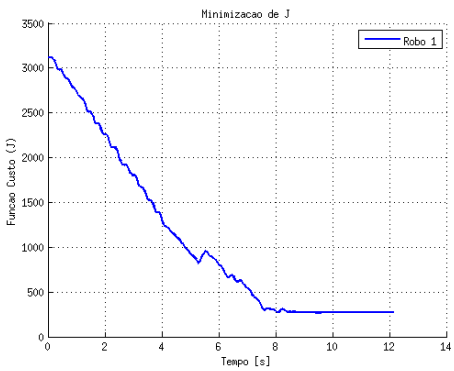


Gráfico 14: Minimização da função custo no BBB

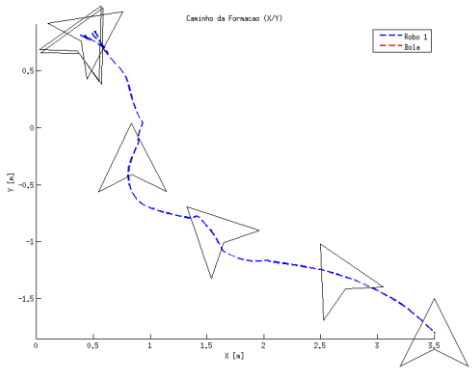


Gráfico 15: Trajetória do PC

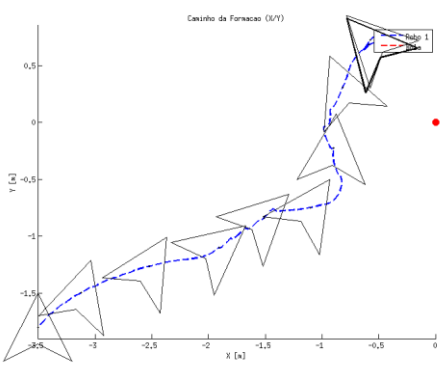


Gráfico 16: Trajetória do BBB

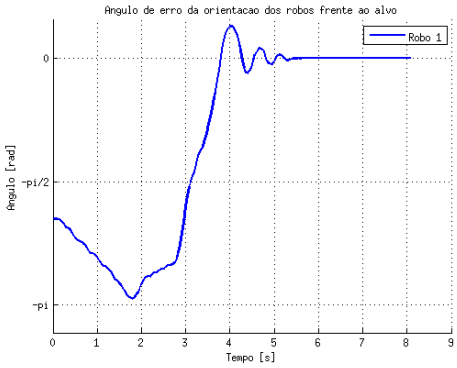


Gráfico 17: Ângulo de erro da orientação dos robôs frente para ao alvo pelo PC

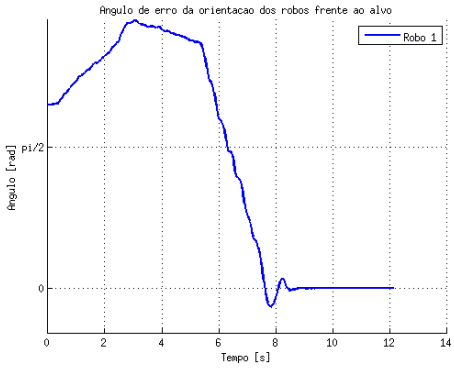


Gráfico 18: Ângulo de erro da orientação dos robôs frente para ao alvo do BBB

Para que fosse possível ter resultados comparativos imparciais, foi utilizado os mesmos parâmetros usados no teste com o BBB, ou seja, horizontes de predição e controle igual a 3 e 1, respectivamente, como também foi desconsiderado o cálculo da função custo responsável pelos colegas. A partir dos dados colhidos é percebível que o PC por ter um poder de processamento maior, faz com que o robô atinja o estado de convergência em média 60% mais rápido que quando controlado pelo BBB. Nesse teste especifico, o robô no PC convergiu em 5 segundos

enquanto no BBB em 8 segundos.

Consequentemente, percebeu-se que se chegou a um limite de processamento da BBB que só conseguiu realizar o teste com um horizonte de predição 3 e horizonte de controle 1. Em razão desses resultados, foi decidido realizar novos testes, porém em uma placa mais robusta (UDOO), que possuísse um poder computacional superior que a da BBB.

Apesar de alguns resultados apresentarem notoriedade, o objetivo dos testes feitos aqui não é identificar qual hardware possui um maior poder de processamento ou o melhor desempenho e sim o quanto distante é o uso entre estes dispositivos. A obtenção destes dados contribuiu para novos planos de testes que serão realizados nas próximas seções.

3.3 Testes com a UDOO

Diante das análises e dos resultados obtidos, foi constato que a placa embarcada BBB utilizando da mesma configuração que um Laptop (horizonte de controle igual a 7 e horizonte de predição igual a 2), não disponibilizou recursos computacionais suficientes para que o controlador pudesse realizar cálculos preditivos em um tempo hábil, sendo necessário reduzir os horizontes de predição para 3 e de controle para 1 para que fosse possível os robôs convergirem. Diante disso, os mesmos testes foram feitos, porém, utilizando de uma placa mais robusta, a UDOO.

Como resultado da viabilidade de utilização de um sistema embarcado para uso de um controlador preditivo, utilizando de uma placa Quad-Core (Multi-Core), foi possível alcançar melhores resultados em relação à placa de apenas um núcleo. Após todos os testes necessários (Tabela 2), foi possível atingir o horizonte de controle de 1 para 2 (o mesmo horizonte de controle utilizado pelo Laptop), porém ainda não foi possível ter mais um horizonte de predição, se mantendo ainda em 3. Com esse resultado, percebe-se que mesmo o controlador (RoC) não ser multi-core, ao utilizar a UDOO, de imediato, obteve-se um ganho de 1 horizonte de controle, pois ao contrário do BBB que possui apenas 1 núcleo dividindo todo seu trabalho entre as operações do Sistema Operacional e do controlador, na UDOO um dos núcleos fica dedicado ao controlador e os demais podem ser utilizados pelo Sistema Operacional e recursos gráficos.

Tabela 2: Demonstração entre Horizonte de Predição e Horizonte de Controle na UDOO.

Teste	Predição	Controle	Converge?
#1	7	2	Não

#2	6	2	Não
#3	5	2	Não
#4	4	2	Não
#5	3	2	Sim

Apesar de não ter atingido os resultados esperados, a UDOO demonstrou um ganho significativo com o aumento de 1 horizonte de controle, em comparação com a BBB, agora se equiparando com o mesmo horizonte de controle utilizado em Laptops (2). Diante desse resultado, foi mantido o uso da UDOO e iniciado outra abordagem para evolução dos horizontes. Como forma de aproximar, dessa vez, o horizonte de predição do controlador embarcado na UDOO, também ao mesmo horizonte de predição utilizado em um Laptop (7), foi adotada a utilização de uma técnica para inspeção de possíveis gargalos no código fonte, chamada de *Profiling*

3.3.1 Otimização do NMPFC

Profiling é uma técnica utilizada para identificar a frequência e duração das chamadas de funções, sendo muito utilizada para auxiliar na otimização de programas. Foi realizado um levantamento de *profiling* no código fonte do controlador (RoC), identificado gargalos e feito otimizações. Para este teste foi utilizado à ferramenta de *profiling* Valgrind.

Valgrind é um sistema para a supervisão de programas, tais como detectores de bugs e *profilers*. Ele executa programas, monitora e traduz binários, dando-lhe o controle total sobre o código-fonte monitorado e sem a necessidade de recompilação antes da execução [29].

A Figura 10 apresenta dados colhidos do *profiling* utilizando a ferramenta Valgrind, nessa figura os resultados foram ordenados por *inclusive*, ou seja, custo computacional da função selecionada incluindo o custo computacional também das funções que a chamaram. Após a execução do *profiler*, foi constatado que a maior parte do custo computacional era proveniente de várias chamadas do controlador a componentes externos (Figura 10), como funções nativas da Linguagem Pascal (utilizada na implementação do controlador RoC), o seu compilador (Free Pascal) e o componente (GTK) responsável pela criação e renderização da interface gráfica.

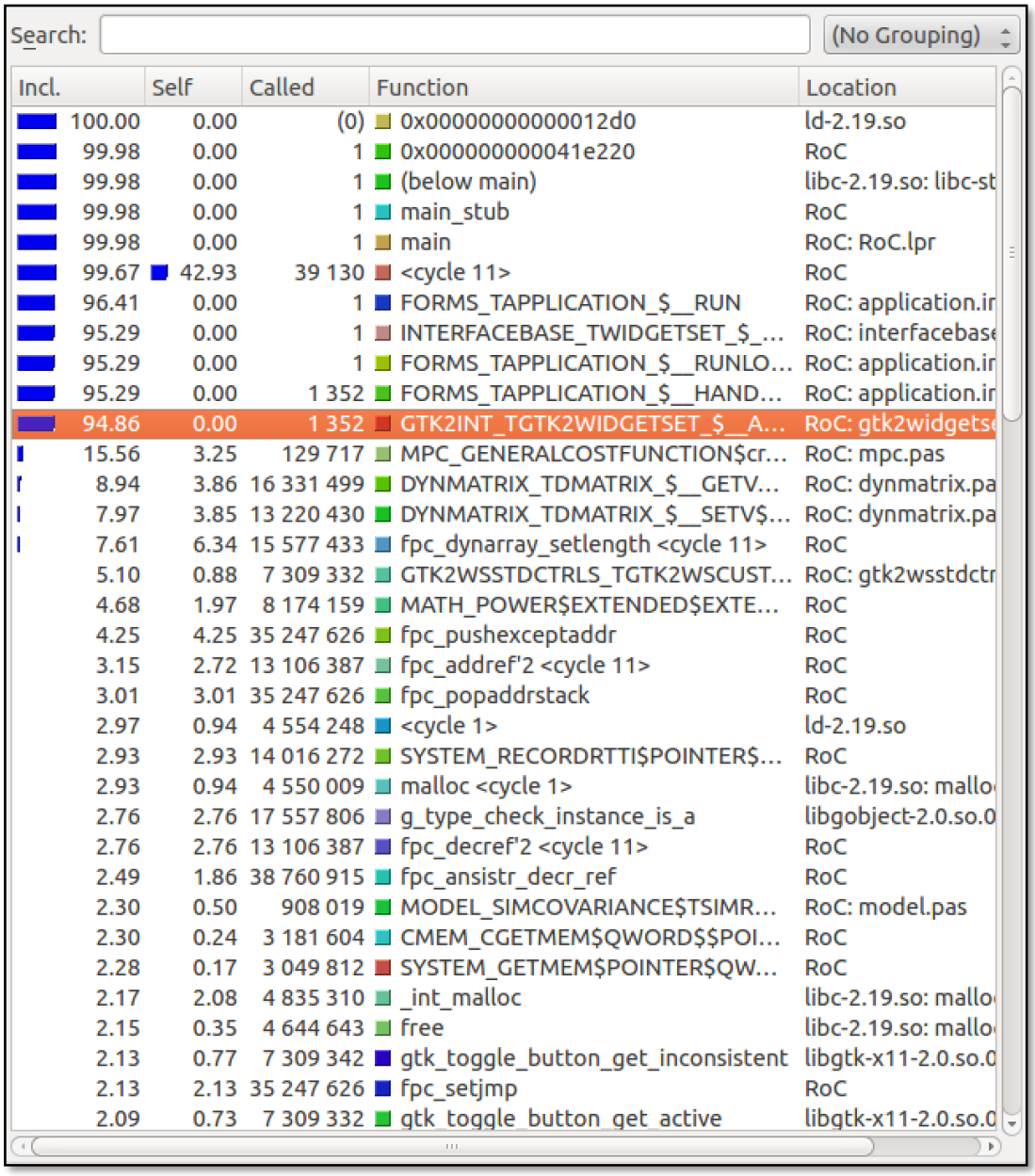
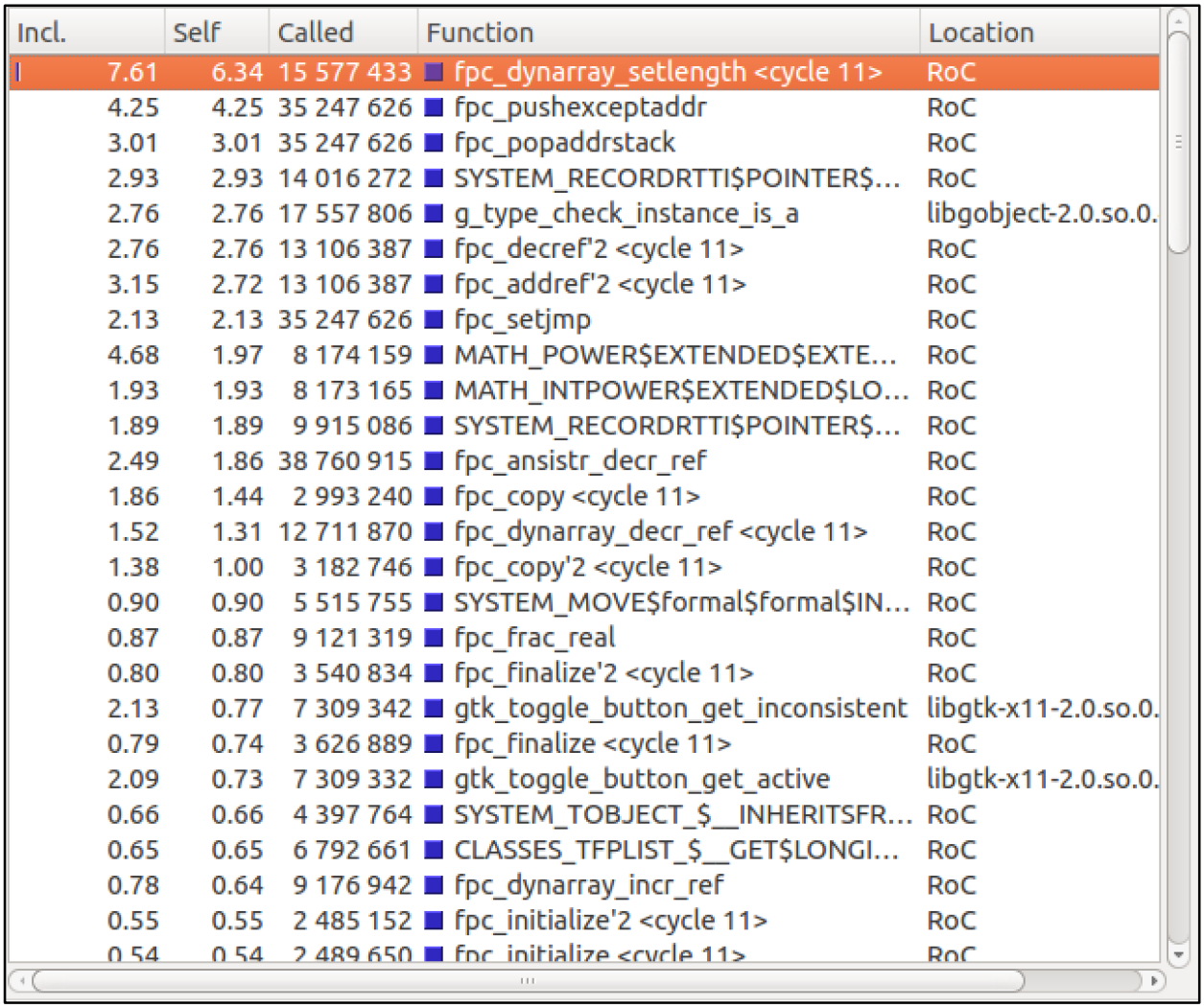


Figura 10: Lista das funções com maior recurso computacional ordenada por *inclusive*

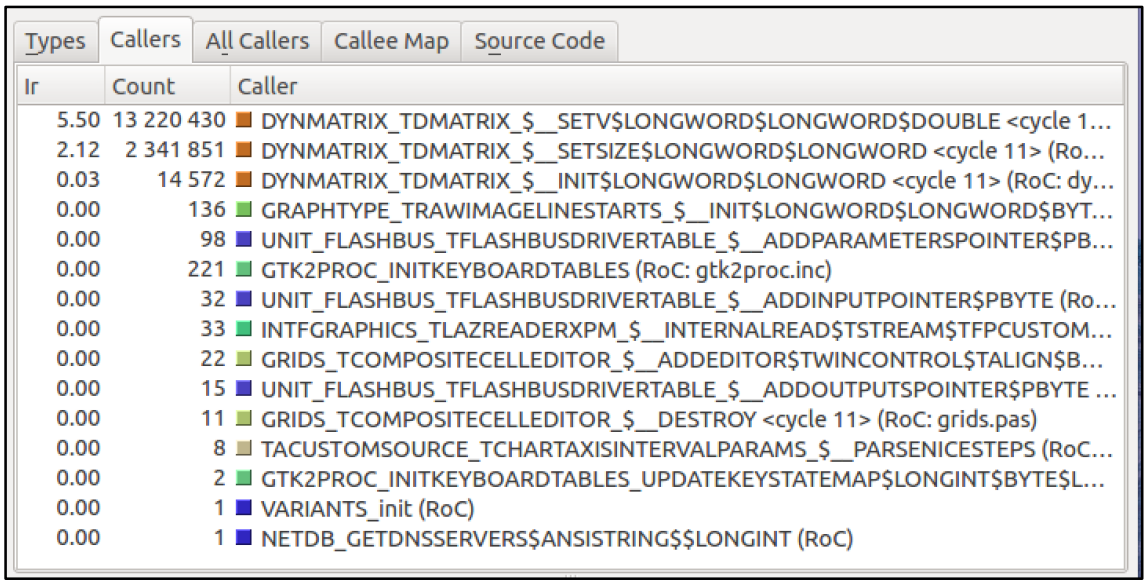
Na Figura 11 os resultados do *profiling* foram ordenados por *self*, ou seja, custo computacional da função selecionada excluindo o custo computacional das funções que a chamaram. Duas funções macros foram identificadas como as funções com o maior custo computacional no controlador. A primeira função chamada de `fpc_dynarray_setlength`, responsável por criar um Array dinâmico com um tamanho passado como parâmetro e a segunda função chamada `fpc_pushexceptaddr` responsável por lançar uma exceção em caso de um erro inesperado, ambas funções nativas da Linguagem Pascal.



Incl.	Self	Called	Function	Location
7.61	6.34	15 577 433	fpc_dynarray_setlength <cycle 11>	RoC
4.25	4.25	35 247 626	fpc_pushexceptaddr	RoC
3.01	3.01	35 247 626	fpc_popaddrstack	RoC
2.93	2.93	14 016 272	SYSTEM_RECORDRTTI$POINTER$...	RoC
2.76	2.76	17 557 806	g_type_check_instance_is_a	libgobject-2.0.so.0.
2.76	2.76	13 106 387	fpc_decref'2 <cycle 11>	RoC
3.15	2.72	13 106 387	fpc_addref'2 <cycle 11>	RoC
2.13	2.13	35 247 626	fpc_setjmp	RoC
4.68	1.97	8 174 159	MATH_POWER$EXTENDED$EXTE...	RoC
1.93	1.93	8 173 165	MATH_INTPOWER$EXTENDED$LO...	RoC
1.89	1.89	9 915 086	SYSTEM_RECORDRTTI$POINTER$...	RoC
2.49	1.86	38 760 915	fpc_ansistr_decr_ref	RoC
1.86	1.44	2 993 240	fpc_copy <cycle 11>	RoC
1.52	1.31	12 711 870	fpc_dynarray_decr_ref <cycle 11>	RoC
1.38	1.00	3 182 746	fpc_copy'2 <cycle 11>	RoC
0.90	0.90	5 515 755	SYSTEM_MOVE$formal$formal$IN...	RoC
0.87	0.87	9 121 319	fpc_frac_real	RoC
0.80	0.80	3 540 834	fpc_finalize'2 <cycle 11>	RoC
2.13	0.77	7 309 342	gtk_toggle_button_get_inconsistent	libgtk-x11-2.0.so.0.
0.79	0.74	3 626 889	fpc_finalize <cycle 11>	RoC
2.09	0.73	7 309 332	gtk_toggle_button_get_active	libgtk-x11-2.0.so.0.
0.66	0.66	4 397 764	SYSTEM_TOBJECT_$__INHERITSFR...	RoC
0.65	0.65	6 792 661	CLASSES_TFPLIST_$__GET$LONGI...	RoC
0.78	0.64	9 176 942	fpc_dynarray_incr_ref	RoC
0.55	0.55	2 485 152	fpc_initialize'2 <cycle 11>	RoC
0.54	0.54	2 489 650	fpc_initialize <cycle 11>	RoC

Figura 11: Lista das funções com maior recurso computacional ordenada por *self*

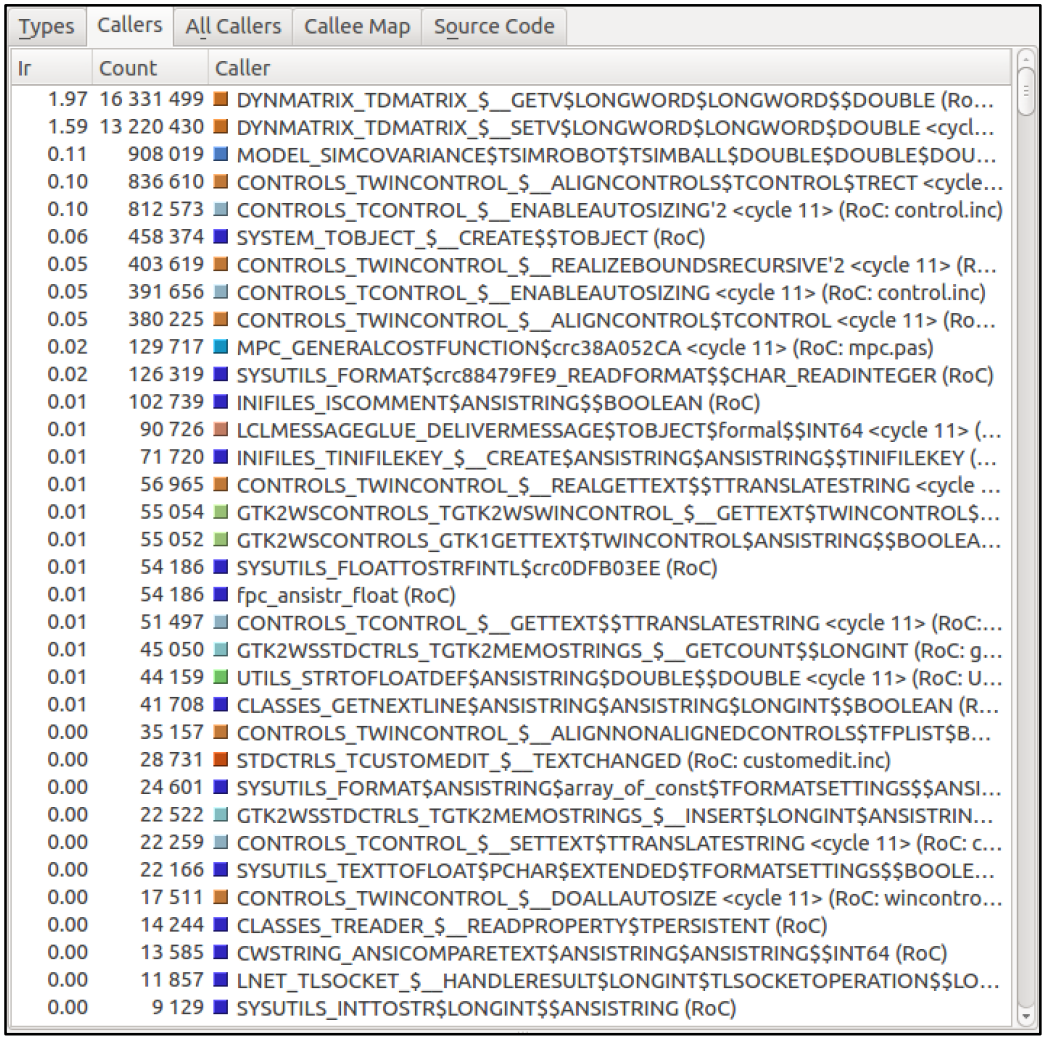
Na Figura 12 é apresentado as funções que fazem chamadas a `fpc_dynarray_setlength` ordenadas por *instructions read (IR)*. Como apresentado, as três funções mais custosas (`SetV, SetSize e Init`) são provenientes da unit `Dynmatrix` pertencente ao controlador (RoC), juntas somam 15.576.853 milhões de chamadas a `fpc_dynarray_setlength` durante toda a convergência do robô.



Types | Callers | All Callers | Callee Map | Source Code

Ir	Count	Caller
5.50	13 220 430	DYNMATRIX_TDMATRIX_$__SETV$LONGWORD$LONGWORD$DOUBLE <cycle 1…
2.12	2 341 851	DYNMATRIX_TDMATRIX_$__SETSIZE$LONGWORD$LONGWORD <cycle 11> (Ro…
0.03	14 572	DYNMATRIX_TDMATRIX_$__INIT$LONGWORD$LONGWORD <cycle 11> (RoC: dy…
0.00	136	GRAPHTYPE_TRAWIMAGELINESTARTS_$__INIT$LONGWORD$LONGWORD$BYT…
0.00	98	UNIT_FLASHBUS_TFLASHBUSDRIVERTABLE_$__ADDPARAMETERSPOINTER$PB…
0.00	221	GTK2PROC_INITKEYBOARDTABLES (RoC: gtk2proc.inc)
0.00	32	UNIT_FLASHBUS_TFLASHBUSDRIVERTABLE_$__ADDINPUTPOINTER$PBYTE (Ro…
0.00	33	INTFGRAPHICS_TLAZREADERXPM_$__INTERNALREAD$TSTREAM$TFPCUSTOM…
0.00	22	GRIDS_TCOMPOSITECELLEDITOR_$__ADDEDITOR$TWINCONTROL$TALIGN$B…
0.00	15	UNIT_FLASHBUS_TFLASHBUSDRIVERTABLE_$__ADDOUTPUTSPOINTER$PBYTE …
0.00	11	GRIDS_TCOMPOSITECELLEDITOR_$__DESTROY <cycle 11> (RoC: grids.pas)
0.00	8	TACUSTOMSOURCE_TCHARTAXISINTERVALPARAMS_$__PARSENICESTEPS (RoC…
0.00	2	GTK2PROC_INITKEYBOARDTABLES_UPDATEKEYSTATEMAP$LONGINT$BYTE$L…
0.00	1	VARIANTS_init (RoC)
0.00	1	NETDB_GETDNSSERVERS$ANSISTRING$$LONGINT (RoC)

Figura 12: Lista das chamadas a `fpc_dynarray_setlenght` ordenadas por *Ir*.



Types | Callers | All Callers | Callee Map | Source Code

Ir	Count	Caller
1.97	16 331 499	DYNMATRIX_TDMATRIX_$__GETV$LONGWORD$LONGWORD$$DOUBLE (Ro…
1.59	13 220 430	DYNMATRIX_TDMATRIX_$__SETV$LONGWORD$LONGWORD$DOUBLE <cycl…
0.11	908 019	MODEL_SIMCOVARIANCE$TSIMROBOT$TSIMBALL$DOUBLE$DOUBLE$DOU…
0.10	836 610	CONTROLS_TWINCONTROL_$__ALIGNCONTROLS$TCONTROL$TRECT <cycle…
0.10	812 573	CONTROLS_TCONTROL_$__ENABLEAUTOSIZING'2 <cycle 11> (RoC: control.inc)
0.06	458 374	SYSTEM_TOBJECT_$__CREATE$$TOBJECT (RoC)
0.05	403 619	CONTROLS_TWINCONTROL_$__REALIZEBOUNDSRECURSIVE'2 <cycle 11> (R…
0.05	391 656	CONTROLS_TCONTROL_$__ENABLEAUTOSIZING <cycle 11> (RoC: control.inc)
0.05	380 225	CONTROLS_TWINCONTROL_$__ALIGNCONTROL$TCONTROL <cycle 11> (Ro…
0.02	129 717	MPC_GENERALCOSTFUNCTION$crc38A052CA <cycle 11> (RoC: mpc.pas)
0.02	126 319	SYSUTILS_FORMAT$crc88479FE9_READFORMAT$$CHAR_READINTEGER (RoC)
0.01	102 739	INIFILES_ISCOMMENT$ANSISTRING$$BOOLEAN (RoC)
0.01	90 726	LCLMESSAGEGLUE_DELIVERMESSAGE$TOBJECT$formal$$INT64 <cycle 11> (…
0.01	71 720	INIFILES_TINIFILEKEY_$__CREATE$ANSISTRING$ANSISTRING$$TINIFILEKEY (…
0.01	56 965	CONTROLS_TWINCONTROL_$__REALGETTEXT$$TTRANSLATESTRING <cycle …
0.01	55 054	GTK2WSCONTROLS_TGTK2WSWINCONTROL_$__GETTEXT$TWINCONTROL$…
0.01	55 052	GTK2WSCONTROLS_GTK1GETTEXT$TWINCONTROL$ANSISTRING$$BOOLEA…
0.01	54 186	SYSUTILS_FLOATTOSTRFINTL$crc0DFB03EE (RoC)
0.01	54 186	fpc_ansistr_float (RoC)
0.01	51 497	CONTROLS_TCONTROL_$__GETTEXT$$TTRANSLATESTRING <cycle 11> (RoC:…
0.01	45 050	GTK2WSSTDCTRLS_TGTK2MEMOSTRINGS_$__GETCOUNT$$LONGINT (RoC: g…
0.01	44 159	UTILS_STRTOFLOATDEF$ANSISTRING$DOUBLE$$DOUBLE <cycle 11> (RoC: U…
0.01	41 708	CLASSES_GETNEXTLINE$ANSISTRING$ANSISTRING$LONGINT$$BOOLEAN (R…
0.00	35 157	CONTROLS_TWINCONTROL_$__ALIGNNONALIGNEDCONTROLS$TFPLIST$B…
0.00	28 731	STDCTRLS_TCUSTOMEDIT_$__TEXTCHANGED (RoC: customedit.inc)
0.00	24 601	SYSUTILS_FORMAT$ANSISTRING$array_of_const$TFORMATSETTINGS$$ANSI…
0.00	22 522	GTK2WSSTDCTRLS_TGTK2MEMOSTRINGS_$__INSERT$LONGINT$ANSISTRIN…
0.00	22 259	CONTROLS_TCONTROL_$__SETTEXT$TTRANSLATESTRING <cycle 11> (RoC: c…
0.00	22 166	SYSUTILS_TEXTTOFLOAT$PCHAR$EXTENDED$TFORMATSETTINGS$$BOOLE…
0.00	17 511	CONTROLS_TWINCONTROL_$__DOALLAUTOSIZE <cycle 11> (RoC: wincontro…
0.00	14 244	CLASSES_TREADER_$__READPROPERTY$TPERSISTENT (RoC)
0.00	13 585	CWSTRING_ANSICOMPARETEXT$ANSISTRING$ANSISTRING$$INT64 (RoC)
0.00	11 857	LNET_TLSOCKET_$__HANDLERESULT$LONGINT$TLSOCKETOPERATION$$LO…
0.00	9 129	SYSUTILS_INTTOSTR$LONGINT$$ANSISTRING (RoC)

Figura 13: Chamadas a `fpc_pushexceptaddr` ordenadas por *Ir*.

Na Figura 13 é apresentado as funções que fazem chamadas a `fpc_pushexceptaddr` ordenadas por instruções lidas (*Instructions Read (IR)*). Como apresentado, as duas funções mais custosas (`GetV` e `SetV`) são provenientes também da unit Dynmatrix pertencente ao controlador (RoC), onde juntas somam 29.551.929 milhões de chamadas a `fpc_pushexceptaddr` durante toda a convergência do robô.

Além da Unit `Dynmatrix`, também foi encontrado um alto processamento nas units `Model` e `MPC` (esta última responsável pelos cálculos preditivos do controlador) Figura 14, como apresentado na Figura 15, a função `GeneralCostFunction` de `MPC`, faz chamadas 6.182.568 milhões de chamadas a unit `Model`, função `SimRobotMovement`, como também, realiza inúmeras verificações a componentes de tela gerenciadas pelo GTK.

Self	Source File
61.93	(unknown)
8.63	wincontrol.inc
6.50	malloc.c
4.40	mpc.pas
4.00	control.inc
3.91	dynmatrix.pas

Incl.	Self	Called	Function	Location
53.53	3.99	220 806	MPC_GENERALCOSTFUNCTION$crc38A052CA	RoC: mpc.pas
0.13	0.13	441 612	MPC_TFORMMPC_$__PROPORTIONALSAT$DOUBLE$DOUBLE$DOUBLE$DOUBLE	RoC: mpc.pas
55.02	0.11	846	MPC_TFORMMPC_$__MPCCONTROLLER$TROBOTSTATE$TTRAJECTORY$TAVOIDSET...	RoC: mpc.pas
0.57	0.10	220 806	MPC_TFORMMPC_$__SCALEFORSATURATION$TDMATRIX	RoC: mpc.pas
0.07	0.03	16 920	MPC_TFORMMPC_$__CALCRPROPSTEP$TOPTDATA$TDMATRIX	RoC: mpc.pas
0.11	0.02	16 920	MPC_TFORMMPC_$__CALCUSTEPS$TDMATRIX$TDMATRIX	RoC: mpc.pas
0.05	0.01	16 920	MPC_TFORMMPC_$__CALCGRADIENT$TDMATRIX$TDMATRIX$TDMATRIX	RoC: mpc.pas
0.00	0.00	846	MPC_TFORMMPC_$__CALCREFTRAJ$TTRAJECTORY$TTRAJECTORY$DOUBLE	RoC: mpc.pas
0.01	0.00	7 642	MPC_EDITTOFLOATDEF$TEDIT$DOUBLE$$DOUBLE <cycle 13>	RoC: mpc.pas
1.86	0.00	846	MPC_TFORMMPC_$__SHOWDEBUG	RoC: mpc.pas
0.01	0.00	5 100	MPC_EDITTOFLOATDEF$TLABELEDEDIT$DOUBLE$$DOUBLE <cycle 13>	RoC: mpc.pas
0.00	0.00	850	MPC_TFORMMPC_$__LOADCONFIG <cycle 13>	RoC: mpc.pas
0.21	0.00	1 692	MPC_TFORMMPC_$__ADDDEBUG$ANSISTRING$DOUBLE	RoC: mpc.pas
0.00	0.00	846	MPC_TFORMMPC_$__OBSTACLEINSEGMENT$crc84C49F83	RoC: mpc.pas
0.00	0.00	1	MPC_TFORMMPC_$__FORMCREATE$TOBJECT	RoC: mpc.pas
0.00	0.00	1	MPC_TFORMMPC_$__FORMSHOW$TOBJECT <cycle 13>	RoC: mpc.pas
0.00	0.00	3	MPC_TFORMMPC_$__BUTTONSETBALLATDISTANCEINLINECLICK$TOBJECT <cycle 13>	RoC: mpc.pas
0.00	0.00	1	MPC_init	RoC: mpc.pas
0.00	0.00	1	MPC_TFORMMPC_$__CHECKBOXENABLECHANGE$TOBJECT <cycle 13>	RoC: mpc.pas

Figura 14: Lista das funções de MPC.pas com o maior custo computacional.

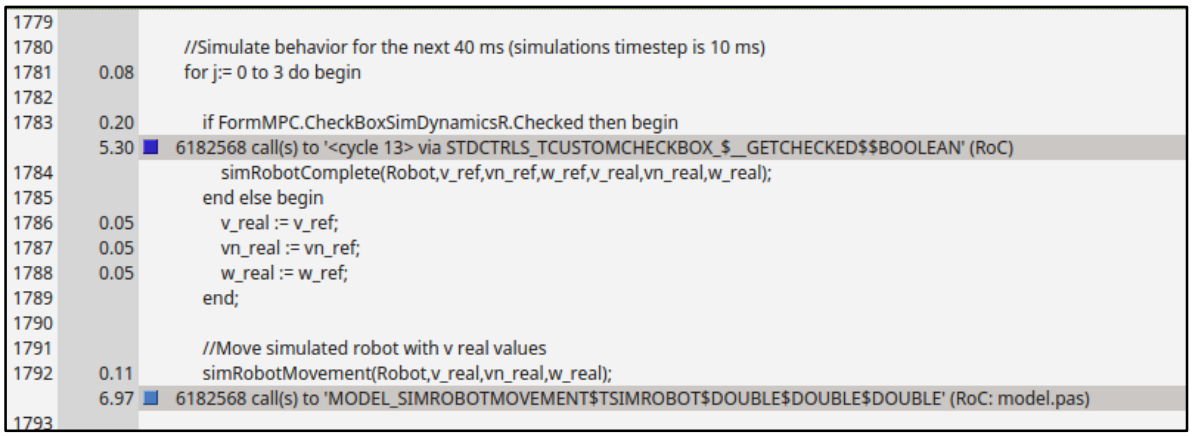


Figura 15: Resultado da análise do gargalo de MPC.pas.

A partir desta análise, foi identificado que os maiores custos computacional eram derivados da Unit DynMatrix, funções SETV, GETV e SETSIZE. Diante deste resultado alguns códigos foram otimizados (Códigos Fonte Otimizados.), evitando chamadas desnecessárias a pontos críticos do sistema, reduzindo assim o custo computacional do controlador, no qual, são melhor visualizados nos experimentos a seguir.

3.3.1.1 Experimento 1

O experimento 1 apresentado no início desta seção e melhor compreendido pela Tabela 3, tem como objetivo identificar qual o melhor horizonte de predição e controle que a placa de baixo custo UDOO suporta e qual o impacto do uso do controlador de formação em um sistema embarcado (Multi-Core), ou seja, o objetivo deste teste é verificar o quanto significativo é o uso de uma placa multi-core para o controlador e o quanto isso afetaria o desempenho do NMPFC.

Com o intuito de realizar uma análise mais criteriosa, foram obtidos dados tanto dos melhores horizontes como dos horizontes nos quais o robô não conseguiu convergir, assim identificar padrões comparativos entre os testes que convergiram e os testes que não convergiram. Como apresentado na Tabela 3, os horizontes 4 e 2 foram os horizontes de predição e controle, respectivamente, que chegaram mais próximos dos horizontes 3 e 2 e que não convergiram.

Tabela 3: Evolução do Horizonte de Predição e Horizonte de Controle na UDOO após o experimento 1.

Experimento	Predição	Controle	Converge?
#1	**3**	**2**	**Sim**
#2	4	2	Não
#3	5	2	Não
#4	6	2	Não
#5	7	2	Não

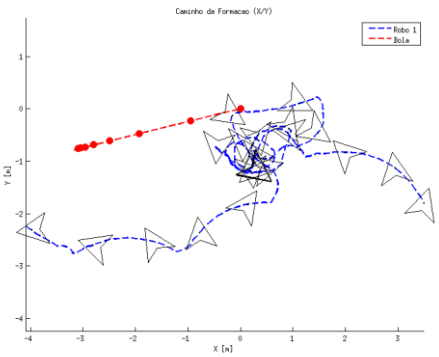


Gráfico 19: Trajetória do robô com a UDOO e H.P igual a 4 e H.C igual a 2. (Experimento 1)

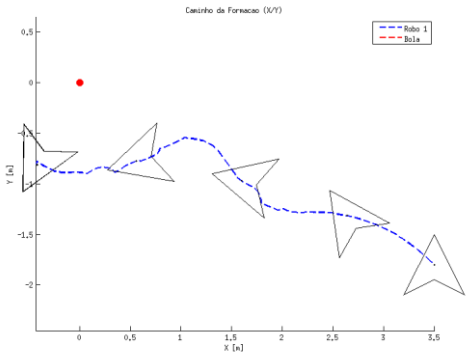


Gráfico 20: Trajetória do robô com a UDOO e H.P igual a 3 e H.C igual a 2 (Experimento 1)

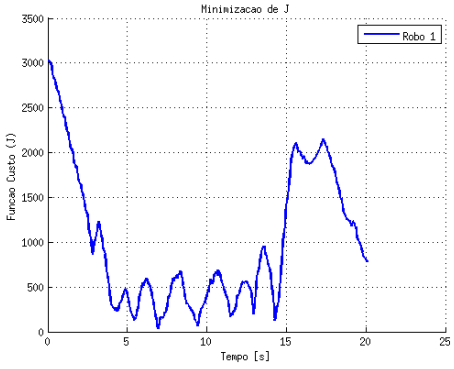


Gráfico 21: Minimização da função custo do robô com a UDOO e H.P igual a 4 e H.C igual a 2 (Experimento 1)

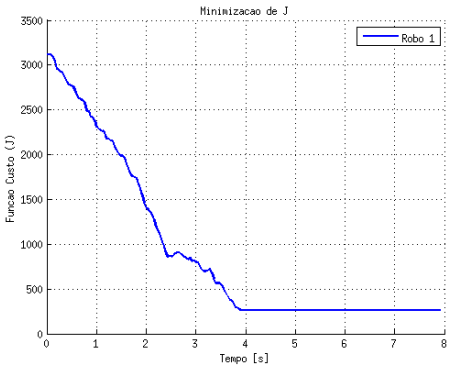


Gráfico 22: Minimização da função custo do robô com a UDOO e H.P igual a 3 e H.C igual a 2 (Experimento 1)

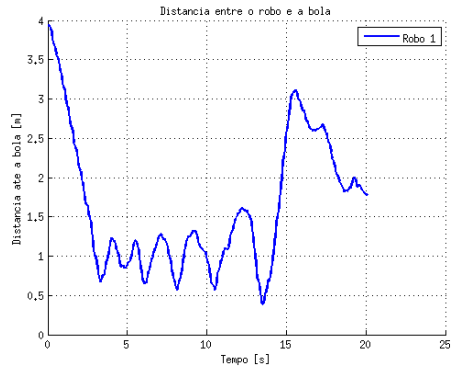


Gráfico 23: Distância entre o robô e a bola do robô com a UDOO e H.P igual a 4 e H.C igual a 2 (Experimento 1).

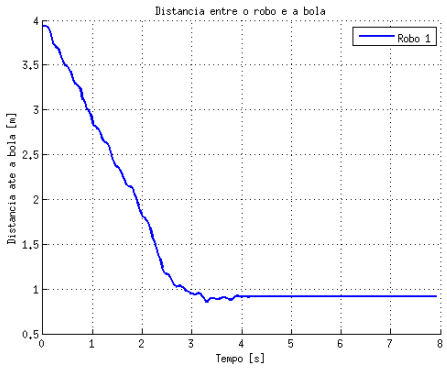


Gráfico 24: Distância entre o robô e a bola do robô com a UDOO e H.P igual a 3 e H.C igual a 2 (Experimento 1)

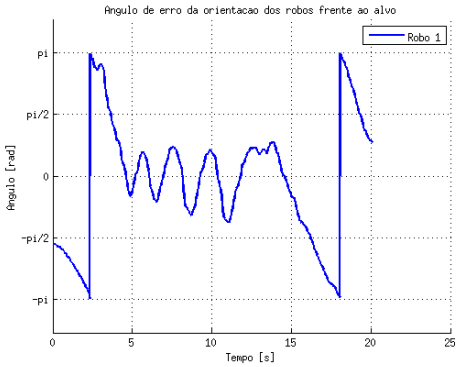


Gráfico 25: Ângulo de erro da orientação dos robôs frente para ao alvo com a UDOO e H.P igual a 4 e H.C igual a 2 (Experimento 1)

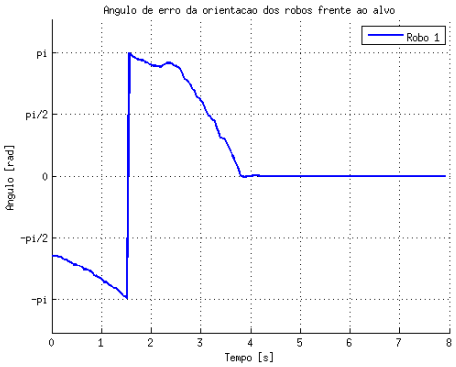


Gráfico 26: Ângulo de erro da orientação dos robôs frente para ao alvo com a UDOO e H.P igual a 3 e H.C igual a 2 (Experimento 1)

Para que fosse possível o robô convergir utilizando o controlador embarcado na UDOO, assim como na BBB, algumas modificações tiveram que ser feitas. (i) Como apresentado anteriormente, o robô que utiliza o controlador embarcado na UDOO, apenas converge com os horizontes de predição e controle igual a 3 e 2, respectivamente, portanto, foi necessário diminuir os horizontes de predição e controle, para que o processamento fosse feito em um tempo hábil, possibilitando o robô a tomar as decisões corretas; (ii) Desativar da função custo o cálculo referente à predição dos colegas.

Duas simulações foram feitas para avaliar o uso do controlador (RoC). Ambas utilizando a arquitetura embarcada (UDOO), porém na primeira simulação utilizou-se os horizontes de predição 4 e horizonte de controle 2 (Gráfico 19, Gráfico 21, Gráfico 23 e Gráfico 25), já na

segunda simulação foi utilizado os horizontes de predição 3 e horizonte de controle 2. Estas simulações foram realizadas utilizando o simulador SimTwo [47].

Como parte dos resultados, durante a primeira simulação com a UDOO, percebe-se que não foi possível realizar a minimização da função custo (Gráfico 21) uma vez que o robô não conseguiu convergir, gerando oscilações constantes. No entanto, no caso da segunda simulação, onde o robô conseguiu convergir, a minimização da função custo foi realizada em 4 segundos (Gráfico 22). Assim, percebe-se que a parametrização correta dos horizontes influência no poder se processamento e na conclusão, ou não, da convergência do robô em formação.

Consecutivamente, por não ser possível convergir, o robô utilizado na primeira simulação não conseguiu minimizar a função custo, nem reduzir os erros do ângulo de orientação dos robôs de frente para ao alvo, em ambos os resultados, os dados observados são de oscilações constantes. Em contrapartida, na simulação 2, a uma distância de 4 metros o robô levou em média 4 segundos para convergir (Gráfico 24) e ao mesmo tempo foi possível minimizar a função custo e os erros frente de orientação do robô, quando em frente ao alvo.

Como resultado, foi possível, aumentar o horizonte de controle de 1 para 2, porém não foi possível aumentar o horizonte de predição em nenhum valor, chegando à conclusão que o controlador utilizando a UDOO e sem nenhuma modificação, apenas converge com horizontes de predição e controle igual a 3 e 2, respectivamente.

3.3.1.2 Experimento 2

O experimento 2, tem como objetivo demonstrar duas otimizações feita no controlador que resultaram no acréscimo de 1 horizonte de predição.

Como primeiro passo, foram retiradas as chamadas a função `Exception.Create` (que faz a chamada direta a `fpc_pushexceptaddr)`, contidos na unit DynMatrix, função GETV (Código 1: - linhas de 4 a 10) e SETV (Código 2 - linhas entre 6 e 12), pois a sua utilização se caracteriza apenas na verificação do tamanho máximo e mínimo de linhas ou colunas, porém, após vários testes foi identificado que todas as chamadas do sistema utilizam os tamanhos corretos de linhas e colunas, ou seja, menor ou igual ao tamanho máximo aceitável. Esta modificação resultou em um ganho de 1 horizonte de predição, agora sendo possível executar o controlador com horizonte de predição igual a 4, como apresentado na Tabela 4.

Tabela 4: Evolução do Horizonte de Predição e Horizonte de Controle na UDOO após o experimento 2.

Experimento	Predição	Controle	Converge?
#1	3	2	**Sim**
#2	4	2	**Sim**
#3	5	2	Não
#4	6	2	Não
#5	7	2	Não

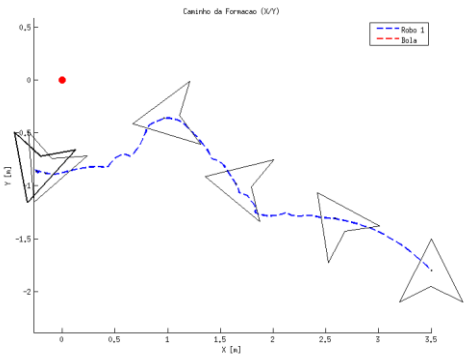


Gráfico 27: Trajetória do robô com a UDOO e H.P igual a 4 e H.C igual a 2 (Experimento 2)

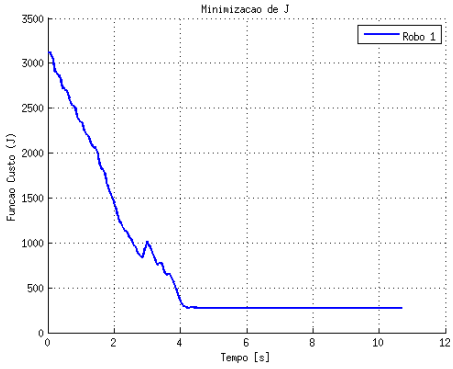


Gráfico 28: Minimização da função custo do robô com a UDOO e H.P igual a 4 e H.C igual a 2 (Experimento 2)

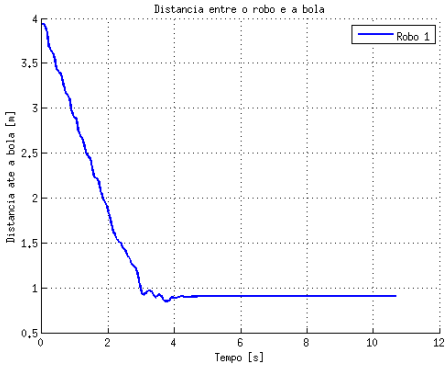


Gráfico 29: Distância entre o robô e a bola do robô com a UDOO e H.P igual a 4 e H.C igual a 2 (Experimento 2)

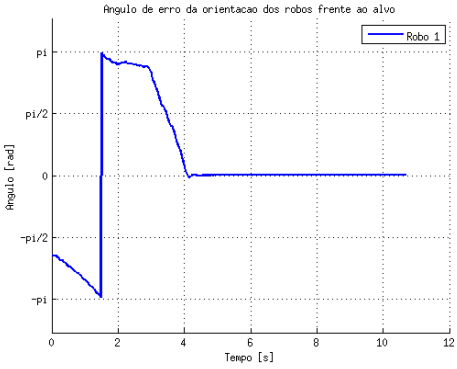


Gráfico 30: Ângulo de erro da orientação dos robôs frente para ao alvo com a UDOO e H.P igual a 4 e H.C igual a 2 (Experimento 2)

Assim como os demais testes realizados até este ponto, para que fosse possível o robô convergir utilizando o controlador embarcado na UDOO, tiveram que ser feitas duas modificações. (i) O robô que utiliza o controlador embarcado na UDOO, com as modificações feitas

nesta seção, apenas converge com os horizontes de predição e controle igual a 4 e 2, respectivamente, portanto, foi necessário diminuir os horizontes de predição e controle, para que o processamento fosse feito em um tempo hábil, possibilitando o robô a tomar as decisões corretas; (ii) Desativar da função custo o cálculo referente à predição dos colegas.

Como parte dos resultados, durante a primeira otimização com a UDOO, percebe-se que a uma distância de 4 metros o robô levou em média 4 segundos para convergir (Gráfico 29) e ao mesmo tempo foi possível minimizar a função custo (Gráfico 28) e o ângulo de erro da orientação dos robôs quando em frente ao alvo (Gráfico 30).

Realizando um comparativo com o experimento anterior (Experimento 1) ambos convergiram quase ao mesmo tempo, com diferença de poucos milissegundos.

3.3.1.3 Experimento 3

No experimento 3, tem como objetivo demonstrar uma otimização feita no controlador que resultou no acréscimo de mais um horizonte de predição.

Como forma de aumentar o horizonte de predição, foi removido à chamada a função `SETLENGHT`, contida na unit DynMatrix, função `SETV` (Código 2 – linha 4), pois sempre que uma chamada a `SETV` era feita, também era redimensionado o mesmo Array para o mesmo tamanho anterior, o que se demonstrou desnecessário e redundante, com isso aumentando em mais 1 horizonte de predição, agora sendo possível atingir um horizonte de predição igual a 5, como apresentado a evolução na Tabela 5.

Tabela 5: Evolução do Horizonte de Predição e Horizonte de Controle na UDOO após o experimento 3.

Experimento	Predição	Controle	Converge?
#1	3	2	**Sim**
#2	4	2	**Sim**
#3	5	2	**Sim**
#4	6	2	Não
#5	7	2	Não

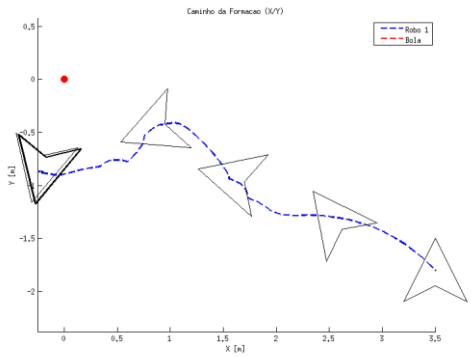


Gráfico 31: Trajetória do robô com a UDOO e H.P igual a 5 e H.C igual a 2 (Experimento 3)

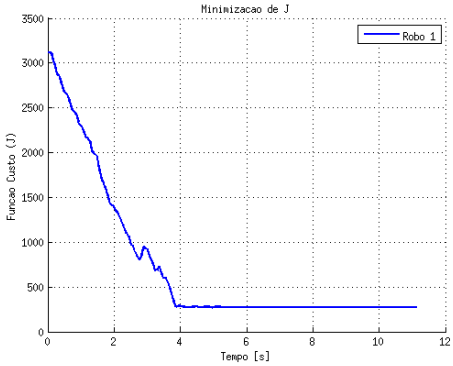


Gráfico 32: Minimização da função custo do robô com a UDOO e H.P igual a 5 e H.C igual a 2 (Experimento 3)

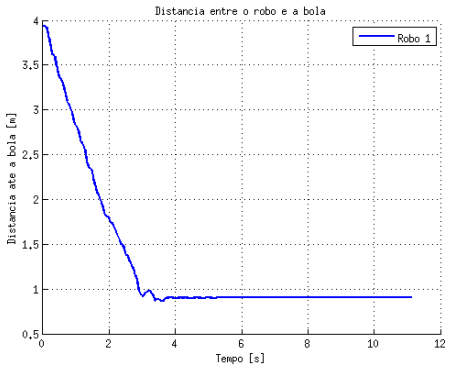


Gráfico 33: Distância entre o robô e a bola do robô com a UDOO e H.P igual a 5 e H.C igual a 2 (Experimento 3)

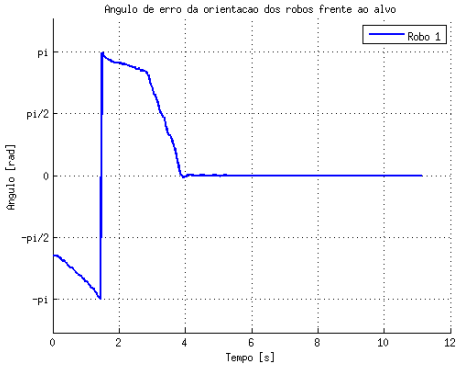


Gráfico 34: Ângulo de erro da orientação dos robôs frente para ao alvo com a UDOO e H.P igual a 5 e H.C igual a 2 (Experimento 3)

No início do teste, foi identificado, que mesmo com a modificação, ainda era necessário ser feitas as mesmas modificações no controlador. (i) O robô que utiliza o controlador embarcado na UDOO, com as modificações feitas nesta seção, apenas converge com os horizontes de predição e controle igual a 5 e 2, respectivamente, portanto, foi necessário diminuir os horizontes de predição e controle, para que o processamento fosse feito em um tempo hábil, possibilitando o robô a tomar as decisões corretas; (ii) Desativar da função custo o cálculo referente à predição dos colegas.

Como parte dos resultados, durante a segunda otimização com a UDOO, percebe-se que a uma distância de 4 metros o robô levou em média 4 segundos para convergir (Gráfico 32) e ao mesmo tempo foi possível minimizar a função custo (Gráfico 33) e o ângulo de erro da orientação dos robôs quando em frente ao alvo (Gráfico 34).

Realizando um comparativo com o experimento anterior (Experimento 3) ambos convergiram quase ao mesmo tempo, com diferença de poucos milissegundos. Com isso, é identificado que quanto maior o horizonte de predição atingido, menor é a proporção no ganho de tempo atingido.

3.3.1.4 Experimento 4

No experimento 4, foram removidas todas as chamadas a função MZEROS (Código 3) que era precedida de `SetSize`, como no exemplo do Código 4 linha 10 e 11, encontrado nas units MODEL e MPC, assim evitando a redundância de chamadas a *`fpc_dynarray_setlength`*, pois as duas possuíam o mesmo código, resultando em um ganho de mais 1 horizonte de predição, agora sendo possível utilizar o robô com horizonte de predição igual a 6, assim visto na Tabela 6.

Tabela 6: Evolução do Horizonte de Predição e Horizonte de Controle na UDOO após o experimento 4.

Experimento	Predição	Controle	Converge?
#1	3	2	**Sim**
#2	4	2	**Sim**
#3	5	2	**Sim**
#4	6	2	**Sim**
#5	7	2	Não

Como parte dos resultados, durante a quarta otimização com a UDOO, percebe-se que a uma distância de 4 metros o robô levou em média 4 segundos para convergir (Gráfico 36) e ao mesmo tempo foi possível minimizar a função custo (Gráfico 35) e o ângulo de erro da orientação dos robôs quando em frente ao alvo (Gráfico 38).

Realizando um comparativo com o experimento anterior (Experimento 4) ambos convergiram basicamente ao mesmo tempo, com diferença de poucos milissegundos. Com isso, novamente, conclui-se que quanto maior o horizonte de predição atingido, menor é a proporção no ganho de tempo atingido.

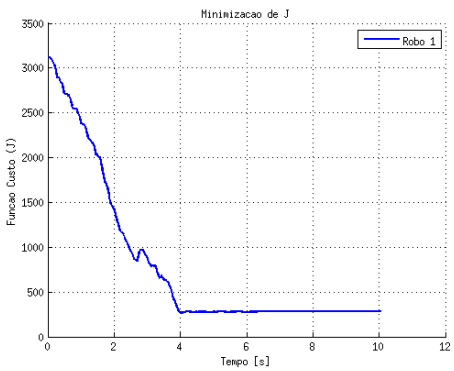


Gráfico 35: Minimização da função custo na UDOO (Experimento 4)

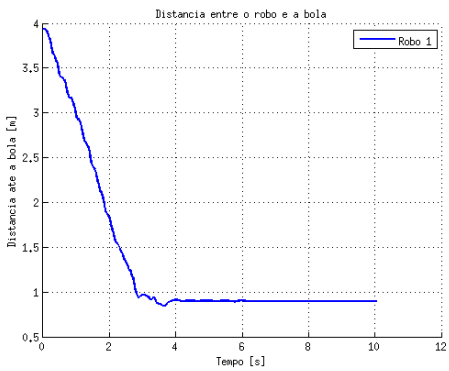


Gráfico 36: Tempo de convergência entre o robô e a bola na UDOO (Experimento 4)

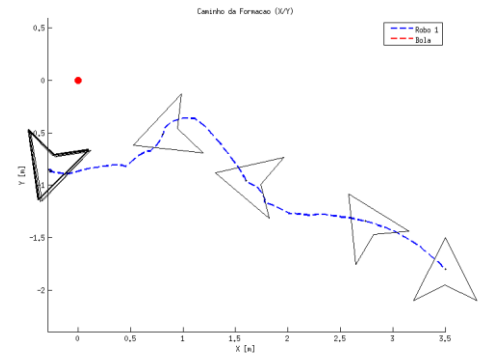


Gráfico 37: Trajetória da Simulação com a UDOO (Experimento 4)

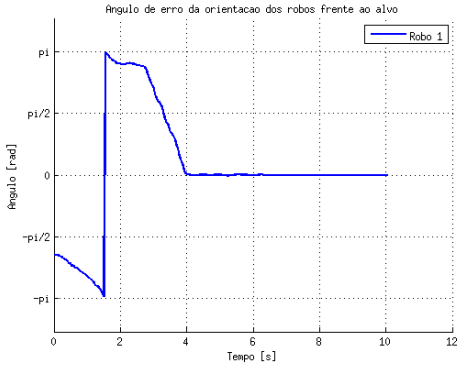


Gráfico 38: Ângulo de erro da orientação dos robôs frente para ao alvo pela UDOO (Experimento 4)

3.3.1.5 Experimento 5

No experimento 5, foram otimizadas quatro funções (Código 5, Código 6, Código 7 e Código 8) de duas units (*MPC* e *Model*). O objetivo desta otimização foi reduzir as chamadas a componentes GTK, pois quando chamados com frequência, impactam na autonomia do robô. Nas linhas em negrito dos (Código 5, Código 6, Código 7 e Código 8) se encontram todas as modificações realizadas no experimento 5.

O Código 5, representa a função `simRobotMovement`, nesta função foi adicionado mais um parâmetro (linha 3), chamado **`isCheckBoxSimDynamicsM,`** sua função é verificar se o modelo dinâmico para os robôs colegas está ativo. Este novo parâmetro complementa o código da linha 25 que por sua vez substituiu o código da linha 23. Dessa forma, o objetivo desta modificação é passar a responsabilidade de verificar qual o modelo do robô para o chamador da função simRobotMovement, assim, evita que inúmeras chamadas ao componente GTK

(**`FormMPC.CheckBoxSimDynamicsM.Checked`**) seja realizada, poupando processamento.

O Código 6, representa a função `GeneralCostFunction` pertencente a unit MPC.pas. Assim como no Código 5, o Código 6 evita que novas chamadas a componentes GTK (Interface Gráfica) seja feita. Ambos se complementam, por exemplo, a linha 65 do Código 6 se encontrava dentro de dois loops (for), ou seja, era chamada milhões de vezes. Com objetivo de otimizar a função `GeneralCostFunction` foi criado variáveis de verificação do modelo do robô, porém fora destes loops (linha 22, 23, 30, 31, 35 e 36), e utilizada na linha 65, assim apenas uma chamada aos componentes de interface gráfica será feita, poupando milhões de chamadas. Ainda no Código 6 foram utilizadas as mesmas variáveis de verificação do modelo, porém para chamadas a função `simRobotMovement` do Código 5 (linhas 81, 93, 102, 110 e 118). Os Código 7 e Código 8 tiveram as mesmas otimizações do Código 6.

Por fim, como demonstrado na Tabela 7 foi possível acrescentar 1 horizonte de predição, agora sendo possível utilizar o robô com horizonte de predição igual a 7.

Tabela 7: Evolução do Horizonte de Predição e Horizonte de Controle na UDOO após o experimento 5.

Experimento	Predição	Controle	Converge?
#1	3	2	**Sim**
#2	4	2	**Sim**
#3	5	2	**Sim**
#4	6	2	**Sim**
#5	7	2	**Sim**

Os gráficos a seguir, são resultados da simulação do controlador embarcado na UDOO. Todas as modificações de desempenho e seus novos horizontes de predição (7) e controle (2). Como apresentado pelos gráficos gerados, ambos conseguiram convergir com o alvo (bola).

No Gráfico 39 pode-se perceber a distância pelo tempo percorridos entre o robô e a bola executando o controlador na placa UDOO. Como descrito na imagem, a uma distância de 4 metros do alvo (bola) o robô embarcado na UDOO convergiu em torno de 3.5 segundos. Em paralelo o a minimização da função custo e tempo levado para sanar o erro do robô frente a

bola, foram também em torno de 3.5 segundos. Se compararmos estes resultados com os resultados obtidos antes da otimização do controlador, o robô utilizando o controlador no PC convergiu em torno de 5 segundos, ou seja, 1.5 segundo mais lento em comparação ao novo código otimizado.

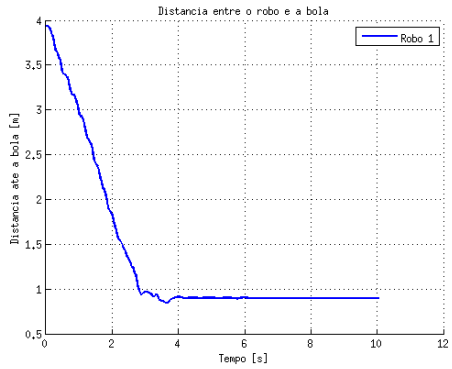


Gráfico 39: Tempo de convergência entre o robô e a bola na UDOO (Experimento 5)

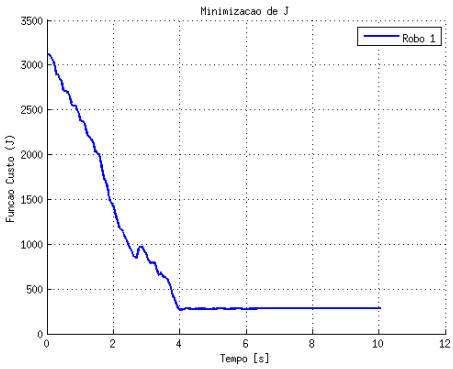


Gráfico 40: Minimização da função custo na UDOO (Experimento 5)

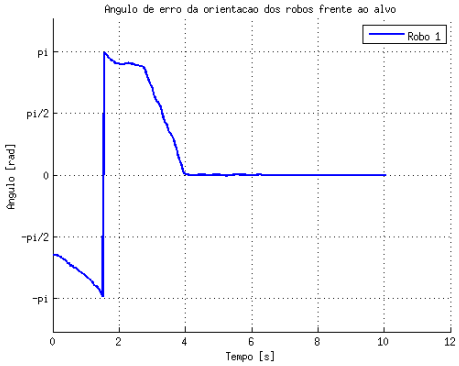


Gráfico 41: Ângulo de erro da orientação dos robôs frente para ao alvo pela UDOO (Experimento 5)

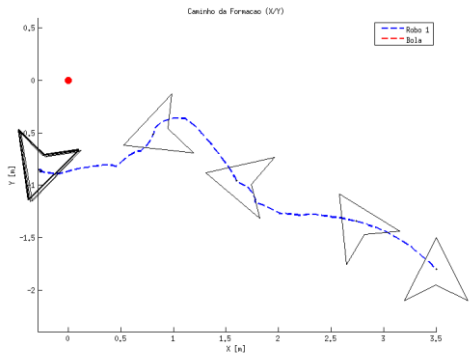


Gráfico 42: Trajetória da Simulação com a UDOO (Experimento 5)

3.4 Resumo do Capítulo

A arquitetura embarcada BeagleBone Black, utilizando do código sem otimização, convergiu em 8 segundos, contra 3.5 segundos do mesmo controlador, porém, embarcado na UDOO e com o código otimizado, resultando em um ganho superior a 50%. Sobre os ângulos de erro da orientação dos robôs frente para ao alvo, tanto o controlador executando no PC, como o controlador executando na UDOO, tiveram os mesmos resultados.

Na Tabela 8 é possível visualizar quantos horizontes de predição foram possíveis acrescentar após cada modificação no código do controlador. Após as alterações, novos testes foram executados e constatado que o controlador com todas as modificações, embarcado em uma placa Quad-Core de 1GHZ atingiu um horizonte de predição igual a 7 e um horizonte de controle igual a 2, as mesmas configurações utilizadas pelos Laptops.

Tabela 8: Experimentos do ganho de horizonte de Predição durante o *Profiling*.

Teste	N° de aumento de horizonte de predição	Horizonte de Predição	Horizonte de Controle
(I)	0	3	2
(II)	1	4	2
(III)	1	5	2
(IV)	1	6	2
(V)	1	7	2

Os resultados expostos na seção 3.1 foram transcritos em forma de artigo e publicado na "*Joint Conference on Robotics: SBR-LARS Robotics Symposium and Robocontrol, IEEE*", em outubro de 2014, com o título de "*Analysis of Prediction Models for Multi-Robot System NMPFC*" [24].

Capítulo 4

4 Conclusão

No decorrer desta dissertação foram cumpridos todos os objetivos propostos inicialmente, portanto acredita-se que o resultado final é bastante positivo. A otimização do controlador revelou-se bastante benéfico para a evolução do controlador como um todo. Este capítulo sistematiza e analisa criticamente o trabalho produzido, identificando as principais contribuições e objetivos atingidos e finalizando com sugestões de trabalho futuro.

4.1 Comentários gerais

No que diz respeito ao modelo do robô, foi constatado que o melhor modelo para prever o comportamento dos robôs é aquele que usa apenas o modelo cinemático para o robô e para os seus colegas de equipe. Isto significa um menor custo computacional, que por sua vez, minimiza a função custo em menos tempo, permitindo a utilização de horizontes de controle e predição elevados mesmo em máquinas de especificações relativamente modestas (como as placas de baixo custo). Porém requer uma adequação prévia dos horizontes dependendo da configuração da placa de baixo custo utilizada. Os resultados obtidos ao final da otimização vão de encontro aos requisitos desta dissertação.

A principal contribuição para este projeto, concentra-se na otimização do sistema de navegação e controle de robôs moveis baseado em NMPC e em todos os passos realizados, principalmente, no uso de *profilers*, uma técnica para supervisão de programas. Sem esta técnica não teria sido possível no tempo adequado, atingir todos os objetivos propostos por esta pesquisa.

Concluindo, foi constatado que a placa embarcada BeagleBone Black, uma placa com apenas 1 núcleo de 1 Ghz, demonstrou um baixo poder de processamento, assim, sendo necessário modificações nas funções do controlador, como também, redução dos horizontes de controle e predição para que fosse possível o robô convergir. Por outro lado, ao utilizar uma placa Quad-Core foi possível utilizar o mesmo horizonte de controle usado em laptops (horizonte de

controle = 2) e após algumas otimizações e adaptações foi possível aumentar de 3 para 7 os horizontes de predição, os mesmos horizontes utilizados nos laptops (7).

Dessa forma, os experimentos demonstraram uma provável possibilidade de substituição dos laptops por placas embarcadas, proporcionando recursos que garantam maior flexibilidade, menor consumo de energia e menos custo financeiro.

4.2 Objetivos atingidos

- Acelerar os algoritmos de controle e de predição, a fim de permitir o uso de horizontes de predição e controle iguais aos utilizados no laptop num sistema embarcado;
- Propor as características de arquiteturas de baixo custo embarcadas, necessárias para este tipo de sistema. Com isso, percebe-se que quanto maior os horizontes de predição e controle maior o custo computacional exigido, porém melhor a performance do robô. Além disso, foi identificado que a pesar do aumento dos horizontes e da estabilidade dos erros, o tempo evoluiu em milissegundos, o que pode apresentar um pequeno valor, mas em sistemas de tempo real, que necessita de resultados imediatos, qualquer ganho é bastante significativo.
- Embarcar um sistema de navegação e controle, baseado em NMPC, em uma placa de baixo custo que disponibilize recursos computacionais suficientes para que o Robô seja capaz de convergir, sem perda de desempenho;
- Otimizar um controlador de formação preditivo.

4.3 Trabalhos futuros

O trabalho aqui realizado apresenta potencial para ser melhorado, tanto na parte de otimização do algoritmo de controle, como no uso de placas embarcadas de baixo custo, com o maior poder de processamento. Apresenta-se algumas sugestões de continuidade deste projeto:

- Como trabalhos futuros pretende-se, realizar estudos e implementações utilizando um robô real, para que as evoluções atingidas nesse trabalho possam ser agregadas ao projeto macro desta pesquisa. Um dos robôs desejados para estes testes é o TurtleBot[1], já disponível no programa de pós-graduação da Universidade Federal da Paraíba.

- Aplicar as modificações de otimização feitas neste trabalho em um outro sistema de navegação (mais enxuto) aplicado em robôs não-holonômicos a fim de mostrar o comportamento de tais robôs.

- Dividir do o sistema de controle (RoC) em pequenos módulos que seriam embarcados em placas de baixo custo, assim paralelizando o processamento.

- A técnica de *profiler* listou vários gargalos no sistema, nessa pesquisa foi otimizado alguns deles, mas pretende-se otimizar ainda mais o controlador, reduzindo a lista de possíveis gargalos, assim, disponibilizando um controlado cada vez mais robusto, exigindo um menor custo computacional e se possível suportando o cálculo dos colegas.

[1] http://www.turtlebot.com/

Referências

[1] Tiago Perreira do Nascimento, "Coordinated Multi-robot Formation Control," Faculdade de Engenharia da Universidade do Porto, Porto, Tese de Doutorado 2012.

[2] Nakagawa S,Yoshio Kawauchi, Martin Buss Toshio Fukuda, "Structure decision for self organazing robots based on cell structures," *IEEE International Conference on Robotics and Automation*, pp. 695–700, 1989.

[3] Michael Jenkin, Evangelos Milios, David Wilkes Gregory Dudek, "A taxonomy for swarm robots.," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS93)*, pp. 441–447, 1993.

[4] Lynne Parker, "ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation," *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, vol. 14, no. 2, pp. 220-240, April 1998.

[5] Jerusa Marchi, "Navegação de robôs móveis autônomos: Estudo e implementação de abordagens.," Universidade Federal de Santa Catarina, Santa Catarina, Tese de Mestrado 2001.

[6] Leandro Coelho Correia and Augusto Loureiro da Costa, "SocialBots: Um Mecanismo de Aprendizado Multiagente para Sistemas Multi-Robos," *III Encontro de Robótica Inteligente*, pp. 230-239, julho 2006.

[7] Maja Mataric, "Interaction and Intelligent Behavior," Massachusetts Institute of Technology - MIT, Cambridge, PhD thesis 1994.

[8] George Bekey Arvin Agah, "Learning from perception, success, and failure in a team of autonomous mobile robots," *Applications of Artificial Intelligence: Expert Systems, Robots and Vision Systems, Fuzzy Logic and Neural Networks*, pp. 179–186, 1996.

[9] Claude Touzet, Fernando Fernández Lynne Parker, "Techiques for learning in multirobot teams," *Robot Teams: From Diversity to Polymorphism*, pp. 191–236, 2002.

[10] Andre G. S. Conceicao, António Paulo Moreira Tiago Pereira Nascimento, "Multi-Robot Systems Formation Control with Obstacle Avoidance," *Preprints of the 19th World Congress - The International Federation of Automatic Control*, pp. 24-29, August 2014.

[11] Tiago Pereira do Nascimento, António Paulo Moreira, and André G. Scolari Conceição, "Multi-robot nonlinear model predictive formation control: Moving target and target absence," *Robotics and Autonomous*, vol. 61, pp. 1502-1515, 2013.

[12] Aamir Ahmad, Tiago Pereira do Nascimento, António Paulo Moreira, and André G. Scolari Conceição, "Perception-Driven Multi-Robot Formation Control," *International Conference on Robotics and Automation*, pp. 1851–1856, 2013.

[13] Dalila Fontes, Fernando Fontes, and Amelia Caldeira, "Behavior-based formation control for multirobot teams," *Optimization and Cooperative Control Strategies*, pp. 371 – 384, 2009.

[14] Heonyoung Lim, Yeonsik Kang, Changwhan Kim, and Jongwon Kim, "Formation control of leader following unmanned ground vehicles using nonlinear model predictive control," *Advanced Intelligent Mechatronics*, pp. 945 –950, 14-17, 2009.

[15] Itsuk Noda, Fernando Ribeiro, Peter Stone, Oskar von Stryk, Manuela Veloso Daniele Nardi, "RoboCup Soccer Leagues," *Association for the Advancement of Artificial Intelligence MAGAZINE*, vol. 35, no. 3, 2014.

[16] REZA GHABCHELOO, ANTONIO PASCOAL, CARLOS SILVESTRE, and ISAAC KAMINER, "Coordinated path following control of multiple wheeled robots with directed communication links," *Decision and Control and European Control Conference*, pp. 7084 – 7089, 2005.

[17] Josiel Alves Gouvêa, "CONTROLE DE FORMAÇÃO DE ROBÔS NÃO-HOLONÔMICOS COM RESTRIÇÃO DE CURVATURA UTILIZANDO FUNÇÃO POTENCIAL," Universidade Federal do Rio de Janeiro, Rio de Janeiro, Tese de Doutorado 2011.

[18] Craig W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25-34, 1987.

[19] Kiattisin Kanjanawanishkul and andreas zell, "Distributed model predictive control for coordinated path following control of omnidirectional mobile robots," *Systems, Man and Cybernetics*, pp. 3120 –3125, 2008.

[20] Hassan Bevrani Fatemeh Daneshfar, "Multi-agent systems in control engineering: A survey fatemeh daneshfar," *Journal of Control Science and Engineering*, 2009.

[21] Savvas G. Loizou, Kostas J. Kyriakopoulos, Michail Zavlanos Dimos Dimarogonas, "A feedback stabilization and collision avoidance scheme for multiple independent non-point agents," *Automatica*, pp. 229–243, 2006.

[22] Fernando Lizarralde, Liu Hsu Josiel Gouvea, "Potential function formation control of nonholonomic mobile robots with curvature constraints," *IFACWorld Congress*, 2011.

[23] Muñoz de la Peña, Eduardo Camacho. João Maestre, "Distributed model predictive control based on a cooperative game," *Optimal Control Applications and Methods*, pp. 153-176, 2011.

[24] Diego Sousa de Azevedo, Luiz Felipe Silva Costa, Alisson Vasconcelos de Brito, and Tiago Pereira do Nascimento, "Analysis of Prediction Models for Multi-Robot System NMPFC," *Proceedings of the IEEE Latin American Robotics Symposium*, pp. 1-6, 2014.

[25] M. Riedmiller and H. Braun, "Direct adaptive method for faster back propagation learning: the rprop algorithm.," *IEEE International Conference on Neural Networks*, pp. 586-591, 1993.

[26] João Rodrigo Alvelos Ferreira, "Controlo Coordenado de Equipas de Robots Móveis," Faculdade de Engenharia da Universidade do Porto, Dissertação de Mestrado 2010.

[27] Tiago Pereira do Nascimento, "Controle De Trajetória De Robôs Móveis Omni-Direcionais : Uma Abordagem Multivariável," Universidade Federal da Bahia, Dissertaça de Mestrado 2009.

[28] G. and Jenkin, M. Dudek, *Computational Principles of Mobile Robotics*.: Cambridge University, 2000.

[29] P. Cheeseman C. Smith, "On the representation and estimation of spatial uncertainty," *Journal of Robotics Research*, no. 5, pp. 56-58, 1986.

[30] A.P.G.M. Moreira, P. Costa A.G.S. Conceição, "A nonlinear mobile robot modeling applied to a model predictive controller," *Proceedings of the 2009 ACM symposium on Applied Computing*, pp. 1186-1187, 2009.

[31] Paulo Gabriel Gadelha Queiroz, Uma abordagem de desenvolvimento de linha de produtos orientada a modelos para a construção de famílias de sistemas embarcados críticos., 2015.

[32] Paul Horn, "Autonomic computing," *IBM's perspective on the state of information technology*, 2001.

[33] Jeffrey O. Kephart and David M. Chess, "The vision of autonomic computing," *IEEE Computer Society*, pp. 41-50, Janeiro 2003.

[34] Edgard de F. Corrêa, Luigi Carro, Flávio R. Wagner Alexandre I. Gervini, "Avaliação de Desempenho, Área e Potência de Mecanismos de Comunicação em Sistemas Embarcados," *XXX Seminário Integrado de Software e Hardware*, 2003.

[35] W Wolf, "Computers as Componentes," *McGraw-Hill*, 2001.

[36] Luca Benini Giovanni De Micheli, "Networks-on-Chip: a New Paradigm for Systemson Chip Design," *Proceedings of the Design, Automation, and Test in Europe Conference.*, 2002.

[37] Andy Wellings Alan Burns, *Real-Time Systems and Their Programming Languages*.: Addison-Wesley, 1997.

[38] Flávio Rech Wagner Luigi Carro,.: Universidade Federal do Rio Grande do Sul - UFRGS, 2010, ch. 2.

[39] Matthew Morris, Kellen Carey, John C. Williams, Corey Randolph, Andrew B. Williams Adam B. Stroud, MU-L8: The Design Architecture and 3D Printing of a Teen-Sized Humanoid Soccer Robot, 2013, Proceedings of the 8th workshop on humanoid Soccer Robot.

[40] Qiang Lyu, Guo-sheng Wang, Kui-feng Su, Feng Guo Pei-pei Ni, Design of Quadrotor's Autonomous Flight Control System Based on BeagleBone Black, 2015, In Proceedings of the 2015 Chinese Intelligent Automation Conference Lecture Notes in Electrical Engineering.

[41] Jake Reher, Jonathan Horn, Victor Paredes, and Aaron D. Ames. Huihua Zhao, "Realization of nonlinear real-time optimization based controllers on self-contained transfemoral prosthesis.," *In Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems (ICCPS '15)*, pp. 130-138, 2015.

[42] JW Topliss, V Zappi, and A McPherson, "Latency Performance for Real-Time Audio on BeagleBone Black," *Linux Audio Conference*, 2014.

[43] Andrew McPherson and Victor Zappi, "An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black," *Audio Engineering Society Convention*, Maio 2015.

[44] Andres Gongora and Javier Gonzalez-Jimenez, "Enhancement of a commercial multicopter for research in autonomous navigation.," *23rd Mediterranean Conference on Control & Automation*, pp. 16-19, Junho 2015.

[45] Filipe Neves dos Santos and Filipe Trocado Ferreira, "AGROB V14 – Towards a low cost vineyard robot.," *Proceedings of the Field Robot Event*, 2014.

[46] Tiago Pereira Nascimento, Antonio Paulo Moreira, Pedro Costa, Paulo Costa, and Andre Scolari Conceição, "Modeling Omnidirectional Mobile Robots: An Approach Using SimTwo," *Proceedings of the 10th Portuguese Conference on Automatic Control*, pp. 117-122, 2012.

[47] Paulo José Cerqueira Gomes da Costa. (2014) SimTwo. [Online]. http://paginas.fe.up.pt/~paco/wiki/index.php?n=Main.SimTwo

[48] Byoung-Ju Lee, Sung-Oh Lee, and Gwi-Tae Park, "Trajectory Generation and Motion Tracking Control for the Robot Soccer Game," *Proceedings of the 1999 IEEE International Conference on Intelligent Robots and Systems*, pp. 1149-1154, 1999.

[49] Gedson Faria, Roseli Francelin Romero, Edson Prestes e Silva Junior, and Marco Aurelio Pires Idiart, "Comparing harmonic functions and potential fields in the trajectory control of mobile robots," *Proceedings of the 2004 IEEE Conference on Robotics, Automation and Mechatronics*, 2014.

[50] ANDRÉ GUSTAVO SCOLARI CONCEIÇÃO, "Controlo e Cooperação de Robôs Móveis Autónomos Omnidireccionais," Faculdade de Engenharia da Universidade do Porto, Tese de Doutorado 2007.

[51] Controle preditivo aplicado a poços inteligentes. [Online]. http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0812725_10_cap_03.pdf

[52] GUILHERME DE LIMA OTTONI, "PLANEJAMENTO DE TRAJETÓRIAS PARA ROBÔS MÓVEIS," Universidade Federal do Rio Grande, Trabalho de Conclusão de Curso 2000.

Apêndice A

Códigos Fonte Otimizados.

Código 1: GETV localizada na Unit DynMatrix

```
1 function TDMatrix.getv(r, c:
2 Longword): double;
3 begin
4   if (r >= Rows) or (c >= Cols)
5   then
6   raise Exception.Create(format
7   ('Invalid (row,col) value.
8   Matrix is (%d,%d),
9   element required is
10  (%d,%d)',[Rows, Cols, r,c]));
11
12 result := data[c + r*Cols];
13 end;
```

Código 2: SETV localizada na Unit DynMatrix

```
1 procedure TDMatrix.setv(r, c:
2 Longword; v: double);
3 begin
4  Setlength(data, rows * cols);
5
```

```
6  if (r >= Rows) or (c >= Cols)
7  then
8    raise Exception.Create(format
9    ('Invalid 10(row,col) value.
10   Matrix is 11(%d,%d), element
11   required is 12(%d,%d)',[Rows,
12   Cols, r,c]));
13
14   data[c + r*Cols] := v;
15 end;
```

Código 3: MZEROS localizada na Unit DynMatrix

```
1 function Mzeros(numrows, numcols:
2 LongWord): TDMatrix;
3 begin
4   result.SetSize(numrows,
5   numcols);
6 end;
```

Código 4: Exemplo de redundância encontrada na utilização de MZEROS e SETSIZE.

```
1 function SimCovariance(Robot :
2 TSimRobot; Ball: TSimBall;
3 val,k1,k2: double):TDMatrix;
4
5 var sgt,sgt2: TDMatrix;
6 v,d,sigX,sigY,tetao,sigphi:
```

```
7 double;
8
9 begin
10  sgt.SetSize(2,2);
11  sgt:=Mzeros(2,2);
12
13 d:=(sqrt(power((Robot.RobotState
14 .x Ball.BallState.x),2)+
15 power((Robot.RobotState.y-
16 Ball.BallState.y),2)));
17
18  sigX:=k1*(power(d,2));
19  sigY:=k2*d;
20
21  sgt.setv(0,0,sigX);
22  sgt.setv(0,1,0);
23  sgt.setv(1,0,0);
24  sgt.setv(1,1,sigY);
25
26  result:=sgt;
27 end;
```

Código 5: `simRobotMovement` da unit Model.pas

```
1  procedure simRobotMovement(var Robot : TSimRobot;
2  var v,vn,w : double;
3  var isCheckBoxSimDynamicsM: boolean);
```

```
 4
 5 var
 6    cteta,steta : double;
 7    v1,v2,v3 : double;
 8 begin
 9           cteta := cos(Robot.RobotState.teta);
10           steta := sin(Robot.RobotState.teta);
11
12         if Robot.RobotState.teta > pi then
13            Robot.RobotState.teta :=
14            Robot.RobotState.teta - 2*pi;
15
16           Robot.RobotState.teta :=
17           Robot.RobotState.teta + simTimeStep*w;
18           Robot.RobotState.x := Robot.RobotState.x
19           + simTimeStep*(v*cteta-vn*steta);
20           Robot.RobotState.y := Robot.RobotState.y
21           + simTimeStep*(v*steta+vn*cteta);
22
23           //if (FormMPC.CheckBoxSimDynamicsM.Checked)
24
25           if (isCheckBoxSimDynamicsM) then begin
26
27             //v,vn,w -> v1,v2,v3
28             IK(Robot.RobotState.v,
29             Robot.RobotState.vn,
30             Robot.RobotState.w,v1,v2,v3);
```

```
31
32              //simulate motors
33              simMotors(Robot,v1,v2,v3);
34
35              //v1,v2,v3 -> v,vn,w
36              DK(v1,v2,v3,v,vn,w);
37
38            end;
39  end;
```

Código 6: Função `GeneralCostFunction` da unit MPC.pas.

```
1    function GeneralCostFunction(var Robot : TSimRobot;
2    var Ball : TSimBall; var obs: array of TRObstacle;
3    num_obs: integer; RobotState : array of TRobotState;
4    var U : TDMatrix): double;
5
6    var
7       sum_cost : double;
8       deltaU : double;
9       temp, temp1: double;
10      lambtest: array[1..10] of double;
11
12      lambA,lamb0,lamb1,lamb2,lamb3,lamb4,lamb5: double;
13      vv,v1,vf: double;
14      segteta,alfa,distancia,newteta,wnew: double;
15      vnew,vnnew: double;
```

```
16
17      i,j,m,k1,k2,k3,k4,t,mate : integer;
18      v_ref,vn_ref,w_ref : double;
19      v_real,vn_real,w_real,d,dettemp:double;
20      aa,bb,cc,dd : double;
21      U_limit,sgtemp,sgtemp1: TDMatrix;
22      isCheckBoxSimDynamicsR : boolean;
23      isCheckBoxSimDynamicsM : boolean;
24  begin
25        sgtemp.SetSize(2,2);
26        sgtemp1.SetSize(2,2);
27
28        //Diego: Verifica se a simulação está usando
29        //modelo Dinamica para o Robo.
30        isCheckBoxSimDynamicsR :=
31        FormMPC.CheckBoxSimDynamicsR.Checked;
32
33        //Diego: Verifica se a simulação está usando
34        //modelo Dinamica para os colegas.
35        isCheckBoxSimDynamicsM :=
36        FormMPC.CheckBoxSimDynamicsM.Checked;
37
38        //PARAMETER DEFINITION
39        // 0 - distance from ball
40        // 1 - number of mate robot in formation
41
42        //initialize
```

```
43          sum_cost := 0;
44
45
46          //simulate robot progression for prediction
47          // instant, calculate J
48          for i:= SimParameters.N1 to SimParameters.N2
49          do begin
50
51              //change control signal for control horizon
52              if i <= SimParameters.Nu then begin
53                   v_ref := U.getv(0,i-1);
54                   vn_ref := U.getv(1,i-1);
55                   w_ref := U.getv(2,i-1);
56              end else begin
57                   v_ref := U.getv(0,SimParameters.Nu-1);
58                   vn_ref := U.getv(1,SimParameters.Nu-1);
59                   w_ref := U.getv(2,SimParameters.Nu-1);
60              end;
61
62              //Simulate behavior for the next 40 ms
63              //(simulations timestep is 10 ms)
64              for j:= 0 to 3 do begin
65               //if (FormMPC.CheckBoxSimDynamicsM.Checked)
66                  if isCheckBoxSimDynamicsR then begin
67                     simRobotComplete(
68
69                     Robot,v_ref,vn_ref,w_ref,v_real,
```

```
70                  vn_real,w_real);
71
72               end else begin
73                   v_real := v_ref;
74                   vn_real := vn_ref;
75                   w_real := w_ref;
76               end;
77
78               //Move simulated robot with v
79               //real values
80               simRobotMovement(Robot,v_real,
81               vn_real,w_real, isCheckBoxSimDynamicsM);
82
83               //Simulates the movement of the
84               //mates in the formation
85               //(kinematics equations only)
86            k1 := FormationSettings.formationMates[0];
87
88               if k1>=0 then simRobotMovement(
89               SimFormationRobots[k1],
90               SimFormationRobots[k1].RobotState.v,
91               SimFormationRobots[k1].RobotState.vn,
92               SimFormationRobots[k1].RobotState.w,
93               isCheckBoxSimDynamicsM);
94
95             k2 := FormationSettings.formationMates[1];
96
```

```
 97                if k2>=0 then simRobotMovement(
 98                SimFormationRobots[k2],
 99                SimFormationRobots[k2].RobotState.v,
100                SimFormationRobots[k2].RobotState.vn,
101                SimFormationRobots[k2].RobotState.w,
102                isCheckBoxSimDynamicsM);
103
104              k3 := FormationSettings.formationMates[2];
105                if k3>=0 then simRobotMovement(
106                SimFormationRobots[k3],
107                SimFormationRobots[k3].RobotState.v,
108                SimFormationRobots[k3].RobotState.vn,
109                SimFormationRobots[k3].RobotState.w,
110                isCheckBoxSimDynamicsM);
111
112              k4 := FormationSettings.formationMates[3];
113                if k4>=0 then simRobotMovement(
114                SimFormationRobots[k4],
115                SimFormationRobots[k4].RobotState.v,
116                SimFormationRobots[k4].RobotState.vn,
117                SimFormationRobots[k4].RobotState.w,
118                isCheckBoxSimDynamicsM);
119
120            //... Código desnecessário omitido para um
121            //melhor entendimento.
122
123        end;
```

```
124    end;
```

Código 7: Função `GeneralCostFunctionFollower` da unit MPC.pas.

```
 1    function GeneralCostFunctionFollower(
 2    var Robot : TSimRobot; var obs: array of TRObstacle;
 3    num_obs: integer; RobotState : array of TRobotState;
 4    var U : TDMatrix): double;
 5
 6    var
 7       sum_cost : double;
 8       deltaU : double;
 9       temp: double;
10       lambtest: array[1..10] of double;
11       lamb0,lamb1,lamb2,lamb3,lamb4,lamb5:double;
12       v1,vf,segteta: double;
13       alfa,distancia,newteta,wnew,vnew,vnnew: double;
14       i,j,m,k1,k2,k3,k4,t : integer;
15       v_ref,vn_ref,w_ref : double;
16       v_real,vn_real,w_real : double;
17       U_limit : TDMatrix;
18       isCheckBoxSimDynamicsM : boolean;
19    begin
20
21         //Diego: Verifica se a simulação está usando
22         //Dinâmica para os colegas.
23         isCheckBoxSimDynamicsM :=
```

```
24          FormMPC.CheckBoxSimDynamicsM.Checked;
25
26          //initialize
27          sum_cost := 0;
28
29          //simulate robot progression for prediction
30          //instant, calculate J
31          for i:= SimParameters.N1 to SimParameters.N2
32          do begin
33
34              //change control signal for control horizon
35              if i <= SimParameters.Nu then begin
36                   v_ref := U.getv(0,i-1);
37                   vn_ref := U.getv(1,i-1);
38                   w_ref := U.getv(2,i-1);
39              end else begin
40                   v_ref := U.getv(0,SimParameters.Nu-1);
41                   vn_ref := U.getv(1,SimParameters.Nu-1);
42                   w_ref := U.getv(2,SimParameters.Nu-1);
43              end;
44
45
46              //Simulate behavior for the next 40 ms
47              //(simulations timestep is 10 ms)
48              for j:= 0 to 3 do begin
49
50                  v_real := v_ref;
```

```
51                vn_real := vn_ref;
52                w_real := w_ref;
53
54                //Move simulated robot with v real values
55                simRobotMovement(
56                Robot,v_real,vn_real,w_real,
57                isCheckBoxSimDynamicsM);
58
59                //Simulates the movement of the mates in
the
60                //formation (kinematics equations only)
61              k1 := FormationSettings.formationMates[0];
62                if k1>=0 then simRobotMovement(
63                SimFormationRobots[k1],
64                SimFormationRobots[k1].RobotState.v,
65                SimFormationRobots[k1].RobotState.vn,
66                SimFormationRobots[k1].RobotState.w,
67                isCheckBoxSimDynamicsM);
68
69              k2 := FormationSettings.formationMates[1];
70                if k2>=0 then simRobotMovement(
71                SimFormationRobots[k2],
72                SimFormationRobots[k2].RobotState.v,
73                SimFormationRobots[k2].RobotState.vn,
74                SimFormationRobots[k2].RobotState.w,
75                isCheckBoxSimDynamicsM);
76
77              k3 := FormationSettings.formationMates[2];
```

```
78                if k3>=0 then simRobotMovement(
79                SimFormationRobots[k3],
80                SimFormationRobots[k3].RobotState.v,
81                SimFormationRobots[k3].RobotState.vn,
82                SimFormationRobots[k3].RobotState.w,
83                isCheckBoxSimDynamicsM);
84
85              k4 := FormationSettings.formationMates[3];
86                if k4>=0 then simRobotMovement(
87                SimFormationRobots[k4],
88                SimFormationRobots[k4].RobotState.v,
89                SimFormationRobots[k4].RobotState.vn,
90                SimFormationRobots[k4].RobotState.w,
91                isCheckBoxSimDynamicsM);
92
93            end;
94             //... Código desnecessário omitido para um
95            //melhor entendimento.
96
97        end;
```

Código 8: Função `TrajCostFunction` da unit MPC.pas.

```
1    function TrajCostFunction(var Robot : TSimRobot;
2    RobotState : array of TRobotState; var U : TDMatrix;
3     refTraj: TTrajectory) : double;
4    var
```

```
 5       sum_costraj : double;
 6       deltaU : double;
 7       i,j : integer;
 8       lamb1,lamb2,lamb3,lamb4: double;
 9       v_ref,vn_ref,w_ref : double;
10       v_real,vn_real,w_real : double;
11       U_limit : TDMatrix;
12       isCheckBoxSimDynamicsR : boolean;
13       isCheckBoxSimDynamicsM : boolean;
14    begin
15
16         //Diego: Verifica se a simulação
17         //está usando Dinamica para o Robo.
18         isCheckBoxSimDynamicsR :=
19         FormMPC.CheckBoxSimDynamicsR.Checked;
20
21         //Diego: Verifica se a simulação
22         //está usando Dinamica para os colegas.
23         isCheckBoxSimDynamicsM :=
24         FormMPC.CheckBoxSimDynamicsM.Checked;
25
26         lamb1:=SimParameters.lambda[0];
27         lamb2:=SimParameters.lambda[1];
28         //lamb3:=FormationSettings.weights[2];
29         lamb4:=SimParameters.lambda[2];
30
31
```

```
32          //initialize
33          sum_costraj := 0;
34
35          //simulate robot progression for
36          //prediction instant, calculate J
37          for i:= SimParameters.N1 to SimParameters.N2
38          do begin
39
40              //change control signal for control horizon
41              if i <= SimParameters.Nu then begin
42                   v_ref := U.getv(0,i-1);
43                   vn_ref := U.getv(1,i-1);
44                   w_ref := U.getv(2,i-1);
45              end else begin
46                   v_ref := U.getv(0,SimParameters.Nu-1);
47                   vn_ref := U.getv(1,SimParameters.Nu-1);
48                   w_ref := U.getv(2,SimParameters.Nu-1);
49              end;
50
51              //Simulate behavior for the next 40 ms
52              //(simulations timestep is 10 ms)
53              for j:= 0 to 3 do begin
54
55                  //Dynamics simulation
56                  //(v references -> v real)
57    //if (FormMPC.CheckBoxSimDynamicsM.Checked)
58                  if isCheckBoxSimDynamicsR then begin
```

```
59                  simRobotComplete(
60                  Robot,v_ref,vn_ref,w_ref,v_real,
61                  vn_real,w_real);
62               end else begin
63                   v_real := v_ref;
64                   vn_real := vn_ref;
65                   w_real := w_ref;
66               end;
67
68               //Move simulated robot with v real values
69               simRobotMovement(Robot,v_real,vn_real,
70               w_real,isCheckBoxSimDynamicsM);
71            end;
72               //... Código desnecessário omitido para um
73               //melhor entendimento.
74         end;
75
```

Apêndice B

Tutorial: Configuração do ambiente de teste.

Atualmente existem duas formas de instalar os pacotes deste tutorial. A primeira é utilizando os arquivos pré-compilados. deb ou .rpm. Sempre que possível deve escolher a instalação pelos arquivos pré-compilados, já que eles já são testados pelos fabricantes e mais fáceis de instalar. A segunda é compilando o Source manualmente, porém se possível deve ser evitada, por necessitar de um maior conhecimento em Linux, apesar que em determinados casos será a única forma, como por exemplo, a empresa Free Pascal não disponibiliza uma versão pré-compilada do Lazarus para arquiteturas ARM, nesse caso, é necessário compilar o source manualmente.

Lembrando que ao contrário do Lazarus, o FreePascal possui arquivos pré-compilados já para as principais plataformas, diante disso, apenas será demonstrado como compilar o Lazarus, para demais arquivos utilizar seus respectivos compilados disponibilizados nos links das próximas seções.

Observações Iniciais

Obs1: Sempre que possível, realizar o download das versões mais novas de cada aplicativo.
Obs2: Não instalar nada por "apt-get", pois os repositórios nem sempre estão com as novas versões, consecutivamente, não possuem novas atualizações e nem as correções de erros (bugs).
Obs 3: Este tutorial foi executado no Ubuntu versões **12.04.5 LTS** e pelo **14.04.1 LTS** com sucesso, para outras versões utilize por sua conta e risco :)
Obs 4: Dois vídeos tutoriais foram criados, seguindo exatamente os passos aqui descritos.

- Tutorial versão Ubuntu 12.04.5 LTS
  Link: https://www.youtube.com/watch?v=MCBrqWi_ToM&feature=youtu.be
- Tutorial versão Ubuntu 14.04.1 LTS
  Link: https://www.youtube.com/watch?v=b7VYdwau-IQ&feature=youtu.be

Instalação

Serão abordadas duas formas de instalação do Lazarus, utilizando os arquivos pré-compilados (recomendado) e compilando o source do Lazarus.

Obtendo os arquivos de instalação

Os links abaixo direcionam para a página de versões do Lazarus, estando lá, escolha a versão desejada (dê preferências as mais atuais) do Lazarus e em conjunto faça o download dos arquivos FPC e FPC-SRC, que se encontram na mesma pasta do arquivo Lazarus.

- **Arquivos para arquitetura de 64 bits (Pré-Compilados):**
  Link Download:
http://sourceforge.net/projects/lazarus/files/Lazarus%20Linux%20amd64%20DEB/
- **Arquivos para arquitetura de 32 bits (Pré-Compilados):**
  Link Download:
http://sourceforge.net/projects/lazarus/files/Lazarus%20Linux%20i386%20DEB/
- **Arquivo source Lazarus (Utilizar esse arquivo apenas se não tiver optado pelos pré-compilados)**
  Link Download: http://sourceforge.net/projects/lazarus/files/Lazarus%20Zip%20_%20GZip/

Instalar FPC e FPC-Source

- Instalar pela interface gráfica

Para instalar o FPC e FPC-SRC pré-compilados pela GUI, basta clicar duas vezes nos arquivos e o Ubuntu irá direcionar para central de instalação de programas.

- Instalar pelo terminal (Apenas se não utilizou a instalação pela GUI)

```
$ sudo dpkg –i fpc_<versao>_arquitetura.extensão
```

Exemplo:

```
$ sudo dpkg –i fpc-src_2.6.4-140420_amd64.deb
```

Instalar o Lazarus

- Lazarus pré-compilado

Para instalar o Lazarus pré-compilado pela GUI, basta clicar duas vezes no arquivo e o Ubuntu irá direcionar para central de instalação de programas.

- Lazarus Source (Apenas se não utilizou a instalação pelo arquivo pré-compilado)

Mova a pasta do lazarus para uma pasta de fácil acesso, de preferência em /home/*usuario/desenvolvimento*. Exemplo: */home/controle/desenvolvimento*

- **Pelo terminal, entre na pasta do Lazarus e digite:**

```
$ make clean bigide
```

Caso o comando acima não funcione, tente:

```
$ make clean all
```

Libs Lnet e SDPO (Serial, Joystick)

- Download dos arquivos 5dpo e LNet

- **Download 5dpo Component**
  Link Download: http://sourceforge.net/projects/sdpo-cl/files/
- **Download Lnet**
  Link Download: http://sourceforge.net/projects/lazarus-ccr/files/lNet/

- Preparando o ambiente

- Copie os dois arquivos (descompactados) para o interior da pasta do Lazarus.
- Abra o lazarus:

```
$ cd pastaDoLazarus
$ ./lazarus &
```

Obs1: O '&' é optativo, ele serve para desbloquear o processo no terminal.

Obs2: Geralmente o lazarus instalado pelo pré-compilado fica em **/usr/share/lazarus/<versao>/lazarus**

Instalar as Libs Sdpo Components e Lnet.

- Com o Lazarus já aberto, clique em Arquivo > Open e navegue até a pasta Sdpo que foi descompactada dentro do lazarus, em seguida, abra o arquivo SdpoJoystick/**sdpojoysticklaz.lpk**. Uma tela vai aparecer então clique em usar (*Use*) e instalar (*Install*). O lazarus será reiniciado.
- Repita o mesmo processo para SdpoSerial/**sdposeriallaz.lpk,** SdpoFreenect/**sdpofreenectlaz.lpk** e para lnet/lazaruspackage/**lnetvisual.lpk.**