UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

JORGE AUGUSTO VASCONCELOS
ALVES

CONTROLE EM TEMPO REAL DE ROBÔS MÓVEIS NÃO-HOLONÔMICOS

Porto Alegre
2007

JORGE AUGUSTO VASCONCELOS
ALVES

CONTROLE EM TEMPO REAL DE ROBÔS MÓVEIS NÃO-HOLONÔMICOS

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.
Área de concentração: Automação e Instrumentação Eletro-Eletrônica

ORIENTADOR: Prof. Dr. Walter Fetter Lages

Porto Alegre
2007

JORGE AUGUSTO VASCONCELOS
ALVES

CONTROLE EM TEMPO REAL DE ROBÔS MÓVEIS NÃO-HOLONÔMICOS

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Walter Fetter Lages, UFRGS
Doutor pelo Instituto Tecnológico de Aeronáutica – São José dos Campos, Brasil

Banca Examinadora:

Prof. Dr. Edson Roberto De Pieri, UFSC
Doutor pela Université Pierre et Marie Curie – Paris, França

Prof. Dr. João Manoel Gomes da Silva Jr., UFRGS
Doutor pela Université Paul Sabatier de Toulouse – Toulouse, França)

Prof. Dr. Renato Ventura Bayan Henriques, UFRGS
Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil)

Coordenador do PPGEE: _____
Prof. Dr. Marcelo Soares Lubaszewski

Porto Alegre, julho de 2007.

DEDICATÓRIA

À minha família, especialmente aos meus pais.

AGRADECIMENTOS

Agradeço profundamente ao meu orientador, prof. Dr. Walter Fetter Lages, pela disponibilidade e inspiração, tendo sido a pessoa com quem mais aprendi durante toda a minha vida acadêmica.

Ao Programa de Pós-Graduação em Engenharia Elétrica, PPGEE, pela oportunidade de aprender com professores de excelente nível e de realizar trabalhos em minha área de interesse; à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, CAPES, pela provisão de bolsa de mestrado e aos professores do Departamento de Engenharia Elétrica, DELET, pela confiança e pelo profissionalismo e qualidade no desempenho de suas atividades de ensino e pesquisa.

Aos diversos colegas com quem convivi durante o curso. Agradecimentos especiais a Lucíola Camprestrini, Ângelo Zerbetto Neto, Moisés Beck, Hermes José Gonçalves Júnior, Sérgio Ricardo Suess, Germán Cláudio Tarnowski, Eduardo Luís Rhod, Vinícius Menezes de Oliveira, Vinícius Gubiani Ferreira, Fernando Augusto Bender, Marcos Rosendo Dalte, Lorenzo Taddei, Diogo de Oliveira Fialho Pereira e Diego Caberlon Santini.

Finalmente, a minha família e, principalmente, aos meus pais, pelo incentivo e confiança.

RESUMO

Este documento trata da implementação em tempo real de controladores preditivos para robôs móveis não-holonômicos. É sabido que, conforme o Teorema de Brockett, não é possível se obter uma lei de controle do tipo realimentação de estados suave e invariante no tempo que estabilize um robô móvel não-holonômico em um ponto qualquer. Pode-se contornar o problema utilizando controle preditivo baseado em modelo (MPC). Entretanto, ao se empregar uma função-custo quadrática em relação ao estado do sistema conforme modelo comumente utilizado não se obtém convergência para o ponto de equilíbrio desejado. Para tal, funções-custo específicas podem ser consideradas na formulação do MPC. São utilizados controladores MPC também para o problema de rastreamento de trajetória. São revisadas estas estratégias de controle preditivo baseado em modelo aplicadas a robôs móveis, apresentadas novas estratégias e então implementados os diversos controladores em tempo real. Medições de tempo são feitas no sistema em funcionamento para comprovar a aplicabilidade dos controladores propostos.

Palavras-chave: Controle preditivo, controle por horizonte deslizante, robôs móveis não-holonômicos.

ABSTRACT

This document deals with the real-time implementation of predictive controllers for non-holonomic mobile robots. According to Brockett's Theorem, it is known that for such a system a smooth, time invariant state feedback can not be used in order to obtain asymptotic point stability. This issue may be overcome by using model predictive control (MPC). However, by considering a cost function that is quadratic with respect to the system state, according to the usual model considered for the system, one does not obtain convergence to the desired point. With the objective of overcoming this problem, particular cost functions can be used. MPC controlers can also be used with mobile robots for the tracking problem. In this work different MPC strategies are revised, new strategies are presented and the controlers are then implemented in real-time. Time measures are taken on the running controllers so as to assure their applicability.

Keywords: predictive control, receding horizon control, control of mobile robots.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ANSI	*American National Standards Institute*
DIAPM	*Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano*
FIFO	*First in - First out*
LASCAR	Laboratório de Sistemas de Controle, Automação e Robótica
LMPC	*Linearized Model Predictive Control*
LXRT	*Linux Real Time Module*
MPC	*Model Predictive Control*
NMPC	*Non-Linear Model Predictive Control*
OOQP	*Object-Oriented Software for Quadratic Programming*
RAM	*Random Access Memory*
RHC	*Receding Horizon Control*
RTAI	*Real-Time Application Interface*
RTHAL	*Real-Time Hardware Abstraction Layer*

LISTA DE SÍMBOLOS

$\mathbb{N}$	Conjunto dos números naturais
$\mathbb{R}$	Conjunto dos números reais
$\mathbf{x}$	Vetor de estado
$\mathbf{u}$	Vetor de entrada
$\mathbf{A}_{n_1 \times n_2}$	Matriz com n_1 linhas e n_2 colunas cujos elementos na linha i e coluna j são dados por a_{ij}
$\text{diag}(a_1, \ldots, a_n)$	matriz diagonal de ordem n cujos elementos da diagonal principal são dados por a_i, $i \in [1, n]$, $i \in \mathbb{N}$
$\mathbf{x}^{\mathrm{T}}$	Transposto de $\mathbf{x}$
T	Período de amostragem
k	Instante de amostragem
$\mathbf{x}(k)$	Valor de $\mathbf{x}$ no instante k
$\mathbf{x}(k_2\|k_1)$	Predição de $\mathbf{x}$ no instante k_2 feita no instante k_1
$\mathbf{x}_r$	Valor de referência de $\mathbf{x}$
$\tilde{\mathbf{x}}$	Erro da variável $\mathbf{x}$ em relação a referência, i.e. $\mathbf{x} - \mathbf{x}_r$
$\mathbf{x}^\star$	Valor ótimo de $\mathbf{x}$
$\mathbf{0}$	Vetor de dimensão apropriada composto de zeros: $[0, \ldots, 0]$
$\mathbf{I}$	Matriz identidade de dimensão apropriada
$\mathbf{Q}$	Matriz de ponderação do erro de estado
$\mathbf{R}$	Matriz de ponderação do esforço de controle ou erro em relação a entrada de referência
Ω	Matriz de ponderação do custo terminal
${}^{o}\mathbf{R}_c$	Matriz mudança de base do sistema de coordenadas do robô móvel para o sistema de coordenadas inercial
$\mathbf{P}$	Matriz mudança de base de velocidades linear e angular para velocidades das rodas
$\sum$	Somatório
$\min\{f\}$	valor mínimo da função f

$f_{\mathbf{x}_i}$	Matriz jacobiana da função $f(\mathbf{x})$ no ponto $\mathbf{x}_i$
Φ	Função-custo considerada pelo MPC
ε	Erro médio quadrático entre o estado real e de referência

1 INTRODUÇÃO

A palavra robô se popularizou a partir da palavra *robota*, da obra Rossum's Universal Robots, do autor tcheco Karel Capek. A palavra tcheca *robota* significa "trabalho forçado." Na obra, robôs foram fabricados com objetivo de dar lucro ao substituir trabalhadores humanos, eventualmente se revoltando contra seus criadores e aniquilando a raça humana (FU; GONZALES; LEE, 1987). Definições da palavra "robô" utilizadas no século passado, especialmente antes da década de 1980 e em textos não científicos, associavam a palavra a um dispositivo com formas humanas, ou capaz de desempenhar atividades essencialmente humanas (WEBSTER, 1983; OXFORD SENIOR DICTIONARY, 6.ED., 1978; HOLANDA FERREIRA, 1986). Tais definições não incluíam robôs capazes de realizar tarefas perigosas ou em ambientes hostis, que poderiam ser entendidas como não-humanas; e poderiam incluir máquinas relativamente simples e capazes de desempenhar tarefas específicas, como uma máquina de lavar.

A atual noção do que seja um robô, relacionada a um dispositivo com partes mecânicas, eletro-mecânicas e eletrônicas e que pode ser reprogramado para realizar tarefas distintas, se desenvolveu após a Segunda Guerra Mundial. Inicialmente o uso de manipuladores na indústria se restringia a tarefas repetitivas que necessitassem de maior precisão ou velocidade. Robôs móveis eram empregados essencialmente para transporte seguindo trajetórias pré-definidas. Com o avanço técnico-científico o uso destes tipos de robôs tem se diversificado. Além disso já existem robôs comerciais capazes de realizar tarefas domésticas, além de robôs já estarem sendo disponibilizados como brinquedos. Nas últimas décadas a robótica tem se enfocado em veículos autônomos ou controlados remotamente, sensoriamento, inteligência artificial e outras áreas, além dos conhecidos manipuladores robóticos e robôs humanóides.

De acordo com (LEONARD; DURRANT-WHITE, 1991), o problema de navegação é resumido em três perguntas: "onde estou?", "aonde vou?" e "como chegar lá?". O primeiro problema trata de determinar a posição do robô no espaço onde está operando. O segundo e o terceiro problemas são relacionados a especificação de um objetivo e de como alcançá-lo, tratando de detectar e desviar de possíveis obstáculos. O problema de navegação é inerente ao uso de robôs móveis autônomos, assunto em que este trabalho se inclui. Entretanto, o objetivo de um robô não pode ser resumido em atingir um determinado ponto ou região. Pode ser necessário percorrer um conjunto de pontos da melhor maneira possível, sendo que o ponto final da trajetória é menos importante que a capacidade do robô móvel de percorrer os pontos anteriores. Controversalmente, pode ser o objetivo de um robô móvel explorar um ambiente e obter informações a respeito do mesmo, não havendo ponto ou trajétória específicas que devam ser alcançadas ou percorridas.

Este trabalho trata de controle de robôs móveis não-holonômicos sujeitos a restrições na amplitude das entradas de controle e, eventualmente, nos valores dos estados; e da implementação em tempo real de algoritmos de controle para robôs deste tipo. Especialmente, serão abordados algoritmos de controle preditivo, conforme (KÜHNE, 2005), que tem como objetivo estabilizar o robô em um determinado ponto ou fazer com que este rastreie uma trajetória pré-determinada. Há diversos tipos de robôs móveis, dentre os quais destacam-se três: os *dotados de rodas*, os robôs do *tipo lagarta* e os *acionados por pernas* (LAGES, 1998). Particularmente, em (KÜHNE, 2005) e neste trabalho, são abordados apenas robôs dotados de rodas. Tais robôs, dependendo de sua configuração, podem ter restrições cinemáticas não-holonômicas (CAMPION; BASTIN; D'ANDRÉA-NOVEL, 1996), ou seja, as equações que representam estas restrições não podem ser integradas. Para este tipo de robô não existe uma lei de controle contínua e invariante no tempo que torne o sistema assintoticamente estável para qualquer ponto de equilíbrio i.e. posição no plano e orientação (BROCKETT, 1982). Em (KÜHNE, 2005) são revisadas diversas estratégias clássicas de controle de robôs móveis não-holonômicos e apresentadas estratégias utilizando controle preditivo baseado em modelo (MPC).

Os métodos clássicos de controle para robôs não-holonômicos se baseiam em contornar as condições de Brockett (BROCKETT, 1982), empregando ou controle *variante no tempo* (POMET et al., 1992; TEEL; MURRAY; WALSH, 1995) ou controle *não-suave* (SØRDALEN, 1993; LAGES, 1998). Os métodos de controle utilizando MPC levam em consideração algum modelo do sistema e uma *função-custo* a ser minimizada em um intervalo de tempo (KÜHNE, 2005; GU; HU, 2005) ou após o mesmo.

1.1 Organização do documento

No Capítulo 2 são apresentados os modelos de robôs móveis que serão considerados neste trabalho. Suas características serão brevemente analisadas e problemas relacionados ao controle do sistema são discutidos. O Capítulo 3 trata de controle preditivo baseado em modelo. A formulação básica, considerando um horizonte de predição finito, é apresentada para problemas de seguir uma trajetória de referência e para o problema estabilizar o robô em um determinado ponto. O problema de *rastreamento de trajetória* utilizando um modelo linearizado do sistema é tratado no Capítulo 4. No Capítulo 5 é utilizado o modelo não-linear e também é tratado o problema de *estabilização em um ponto*. No Capítulo 6 são feitas considerações finais acerca do trabalho desenvolvido e sugestões para trabalhos futuros.

2 MODELAGEM DO ROBÔ

2.1 O Robô Twil

Neste documento será utilizado o robô Twil (LAGES, 1998; LAGES; HEMERLY, 1998). O robô Twil é dotado de duas rodas convencionais fixas paralelas acopladas a motores de corrente contínua e mais uma roda passiva que tem a função de dar sustentação e equilíbrio ao robô. Com o uso de um único equipamento a comparação de diferentes métodos de controle é facilitada, além do fato de ser o robô que se tem disponível para testes (Figura 1).

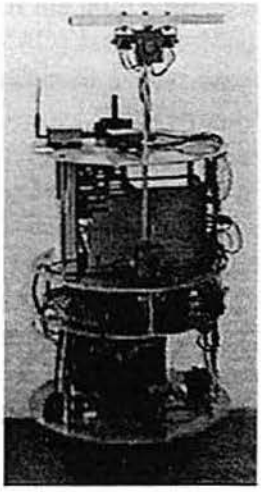

Figura 1: O robô móvel Twil.

2.2 Restrições Holonômicas e Não-Holonômicas

O termo "restrição" pode estar relacionado a sistemas dinâmicos de diversas formas, notadamente com controle ou modelagem. Relacionado a controle normalmente denota limites nos estados e nas entradas que devem ser satisfeitos ou para o bom funcionamento do mesmo ou devido a peculiaridades do sistema real que não constam na modelagem. Isto será discutido no Capítulo 3. Pode também ser de interesse que seja obtida uma lei de controle que garanta determinadas medidas de desempenho e possivelmente estabilidade ou rejeição a distúrbios. Uma lei de controle pode se obtida considerando conjuntos invariantes de tal modo que tem-se certeza de que os requisitos de performance e as restrições serão atendidos para instantes de tempo

positivos (MESQUINE; TADEO; BENZAOUIA, 2004; BITSORIS, 1988). Tal área é comumente chamada *controle sob restrições*. A obtenção de leis de controle com base em conjuntos invariantes não é assunto deste trabalho, porém. Com relação a modelagem, assunto desta seção, o termo denota a relação entre coordenadas que o descrevem. Dividem-se em restrições *holonômicas* e *não-holonômicas*.

As restrições holonômicas são equações expressas em termos das coordenadas que descrevem o sistema em um dado momento e, possivelmente, do tempo. Para sistemas com restrições holonômicas – ditos então sistemas holonômicos – pode-se sempre encontrar um conjunto de coordenadas generalizadas *independentes*. Tal número é igual ao número de graus de liberdade do sistema (VU; ESFANDIARI, 1997).

Restrições não-holonômicas não são expressas apenas em termos das coordenadas e tempo diretamente, mas incluem diferenciais de tais variáveis. São tais que não podem ser integradas de modo a se obter um conjunto de coordenadas generalizadas independentes. Tem-se que o número de graus de liberdade para sistemas que contém restrições não-holonômicas no seu modelo – ditos então sistemas não-holonômicos – é inferior ao número de coordenadas generalizadas *dependentes* (VU; ESFANDIARI, 1997). Um exemplo clássico de sistema não-holonômico é o modelo simplificado de uma moeda rolando sem deslizamento no plano (VU; ESFANDIARI, 1997; LATOMBE, 1991). Tal modelo, por ignorar a ação da gravidade e o ângulo entre o plano da moeda e o plano sobre o qual ela rola, é o mesmo que descreve uma roda que gira em torno de um eixo horizontal (CAMPION; BASTIN; D'ANDRÉA-NOVEL, 1996), o qual serve de base para a modelagem de robôs móveis dotados de rodas.

Considere uma moeda, que rola sobre um plano horizontal sem deslizamento, conforme a Figura 2. O ponto de contato entre a moeda, cujo raio é r, e o plano horizontal por $[x \;\; y]$, o ângulo entre o plano da moeda e o eixo Ox é dado por θ e o ângulo de rotação da moeda em torno do eixo que passa pelo centro da mesma e é normal ao plano que a contém é dado por γ. Tem-se, então, o seguinte conjunto de restrições que descrevem o movimento da mesma sobre o plano (VU; ESFANDIARI, 1997; LATOMBE, 1991):

$$dx - r\,\mathrm{sen}\,\theta d\gamma = 0$$
$$dy - r\cos\theta d\gamma = 0.$$

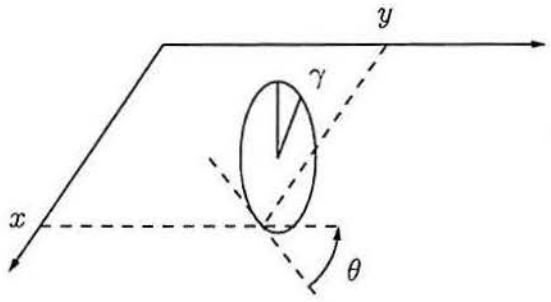

Figura 2: Moeda rolando sobre um plano horizontal.

Tal exemplo apresenta quatro coordenadas generalizadas dependentes (x, y, θ e γ), mas apenas dois graus de liberdade. Pode-se modificar θ e γ independentemente, mas isto acarretará em mudanças em x e y (VU; ESFANDIARI, 1997).

A obtenção de leis de controle para sistemas não-holonômicos é mais complexa que para sistemas holonômicos. A explicação é o reduzido número de graus de liberdade em relação ao número de estados deste tipo de sistema. Mais especificamente, a possibilidade de se estabilizar assintoticamente um dado ponto de equilíbrio qualquer é explicada pelas condições de Brockett (BROCKETT, 1982). Isto será tratado na Seção 5.4.

2.3 Modelos do Robô Móvel

É atribuído ao robô um sistema de coordenadas $\{C,X_c,Y_c\}$ que se move junto com o mesmo em relação a um sistema de coordenadas inercial $\{O,X,Y\}$, sendo que as coordenadas de C neste último sistema de coordenadas são dadas por (x,y) e o ângulo entre os eixos OX_c e OX é dado por θ (vide Figura 3). Assume-se que as duas rodas ativas não são deformáveis, estão presas a uma estrutura rígida e o contato das mesmas com o plano horizontal é puntual. Assume-se também que o plano das rodas permanece vertical durante o movimento e a rotação das rodas fixas se dá em torno de um eixo cuja orientação em relação ao sistema de coordenadas do robô é constante. Finalmente, assume-se que não há deslizamento das rodas, portanto as componentes de velocidade paralela e ortogonal ao plano das rodas são nulas. Tais restrições determinam a não-holonomicidade dos modelos que descrevem o comportamento do robô (CAMPION; BASTIN; D'ANDRÉA-NOVEL, 1996).

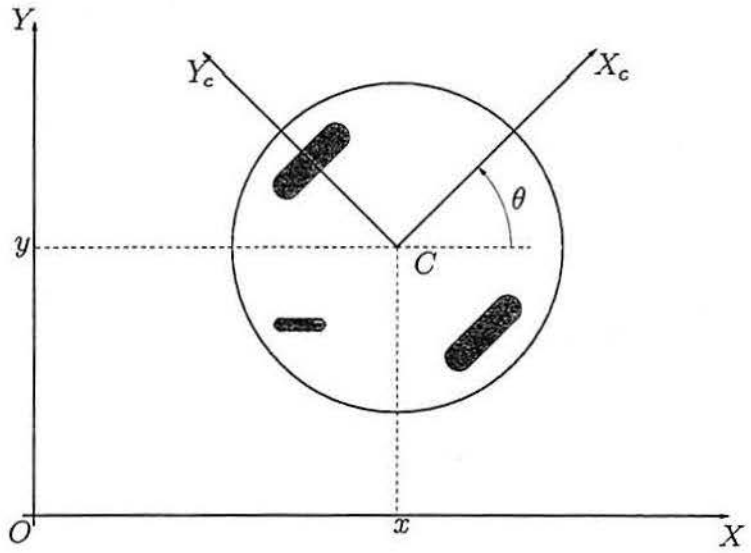

Figura 3: Sistema de coordenadas do robô móvel.

Os ângulos de rotação das rodas fixas são dados por φ_1 e φ_2 (vide Figura 4) enquanto que o ângulo de rotação da roda orientável não-centrada é dado por φ_3 (vide Figura 5). A distância entre o centro das rodas fixas e o centro do robô é dada por l_c, a distancia entre o eixo vertical de rotação da roda orientável não-centrada e o centro da mesma roda é dada por d e a distância entre o eixo vertical de rotação e o centro do robô é dada por l_{oc}. O ângulo entre o plano da roda e o eixo Y_c é dado por β (vide Figura 6).

O acionamento do robô se dá por aplicação da tensão de armadura nos motores acoplados às rodas fixas do mesmo. Tal acionamento pode ser considerado

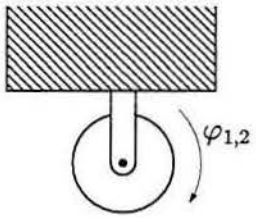

Figura 4: Ângulo de rotação das rodas fixas.

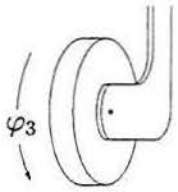

Figura 5: Ângulo de rotação da roda orientável não-centrada.

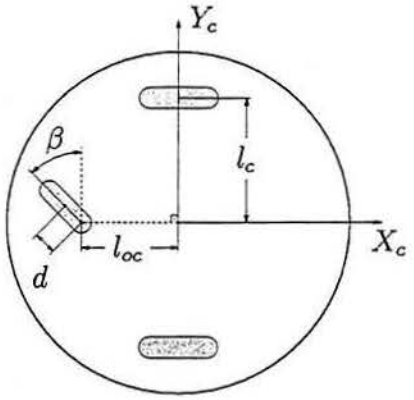

Figura 6: Parâmetros do robô móvel.

como aplicação de torque nas rodas levando-se em consideração que as variações na corrente de armadura dos motores têm um comportamento muito mais rápido que variações na velocidade angular dos eixos dos motores (e conseqüentemente das rodas) e portanto podem ser desprezadas.

Existem dois tipos principais de modelos utilizados para descrever o comportamento de robôs móveis. Os modelos chamados *cinemáticos* são assim chamados por não levar em consideração as forças e torques que causam o movimento. Os segundos, que consideram as forças e torques causadores do movimento, a massa do robô e sua distribuição, são chamados *dinâmicos*. Os modelos cinemáticos podem ser obtidos considerando-se as restrições relativas ao movimento das rodas. Distinguem-se dois tipos de modelos cinemáticos: de *postura*, que leva em consideração apenas a posição do robô no plano e sua orientação, e de *configuração*, que inclui o modelo cinemático de postura e, adicionalmente, leva em consideração os ângulos de rotação e orientação das rodas. O modelo cinemático de postura do robô é dado por (CAMPION; BASTIN; D'ANDRÉA-NOVEL, 1996)

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\,\mathrm{sen}\,\theta \\ w \end{bmatrix}, \tag{1}$$

onde a dependência temporal de x, y, θ, u e v foi omitida; ou, de maneira mais compacta:

$$\dot{\mathbf{x}} = \mathbf{E}(\mathbf{x})\mathbf{u} \tag{2}$$

onde v e w são as velocidades linear e angular do robô, respectivamente; $\mathbf{x} \triangleq$

$[x \;\; y \;\; \theta]^{\mathrm{T}}$; $\mathbf{u} \triangleq [v \;\; w]^{\mathrm{T}}$ e

$$\mathbf{E}(\mathbf{x}) \triangleq \begin{bmatrix} \cos\theta & 0 \\ \operatorname{sen}\theta & 0 \\ 0 & 1 \end{bmatrix}.$$

Ele descreve a posição e a orientação do sistema de coordenadas do robô em relação ao sistema de coordenadas inercial. Observa-se que neste modelo os ângulos de orientação e rotação da roda orientável não-centrada (roda livre) não são considerados. Aspectos de interesse, como controlabilidade, estabilidade e linearização em torno do estado e por realimentação de estados relativos a robôs descritos por este modelo podem ser encontrados em (KÜHNE, 2005, sec. 2.3.1). Especificamente, o sistema é estável no sentido de Lyapunov e controlável, entretanto um ponto qualquer não pode ser estabilizado por uma realimentação de estados suave e invariante no tempo (BROCKETT, 1982). O modelo obtido da sua linearização em torno do estado não é controlável e o sistema não pode ser linearizado por realimentação de estados. É possível, entretanto, obter uma linearização entrada-saída se a saída $\mathbf{y}$ considerada for $[x \;\; y]^{\mathrm{T}}$, apenas.

O modelo cinemático de configuração é dado por

$$\dot{\mathbf{q}} = \mathbf{S}(\mathbf{q})\mathbf{u}, \tag{3}$$

onde

$$\mathbf{q} \triangleq [x \;\; y \;\; \theta \;\; \beta \;\; \varphi_1 \;\; \varphi_2 \;\; \varphi_3]^{\mathrm{T}}$$

e

$$\mathbf{S}(\mathbf{q}) \triangleq \begin{bmatrix} & \mathbf{E}(\mathbf{x}) & \\ \frac{1}{d}\cos(\beta) & -1 - \frac{l_{oc}}{d}\operatorname{sen}(\beta) \\ \frac{1}{r} & -\frac{l_f}{r} \\ \frac{1}{r} & \frac{l_f}{r} \\ -\frac{\operatorname{sen}(\beta)}{r} & -\frac{l_{oc}}{r}\cos(\beta) \end{bmatrix}$$

que inclui o modelo cinemático de postura e ainda considera os ângulos das rodas.

O modelo dinâmico do robô tem a forma

$$\begin{cases} \dot{\mathbf{q}} = \mathbf{S}(\mathbf{q})\mathbf{u} \\ \mathbf{H}(\mathbf{q})\dot{\mathbf{u}} + \mathbf{f}(\beta,\mathbf{u}) = \mathbf{F}(\beta)\tau \end{cases} \tag{4}$$

onde $\mathbf{H}(\mathbf{q})$ é uma matriz inversível relacionada à inércia do robô, $\mathbf{f}(\beta,\mathbf{u})$ é relacionada a forças centrífugas e de Coriolis, $\mathbf{F}(\beta)$ depende da configuração do acionamento e τ representa os torques não-conservativos (aplicados pelos motores e resultantes de atrito viscoso e de Coulomb) nos eixos do robô (CAMPION; BASTIN; D'ANDRÉA-NOVEL, 1996).

Os controladores propostos neste trabalho consideram apenas o modelo cinemático de postura do robô móvel. Além disso, consideram um sistema em tempo discreto na sua formulação. Conseqüentemente é necessária uma discretização de (1). Por simplicidade, optou-se por utilizar um modelo obtido por aproximação de Euler do modelo em tempo contínuo descrito por (1), dado por

$$\begin{cases} x(k+1) = x(k) + v(k)T\cos\theta(k) \\ y(k+1) = y(k) + v(k)T\operatorname{sen}\theta(k) \\ \theta(k+1) = \theta(k) + w(k)T, \end{cases} \tag{5}$$

onde T é o período de amostragem.

2.3.1 Modelo Cinemático de Postura em Coordenadas Polares

Considera-se o modelo cinemático de postura, expresso por (1), que tem como estados as coordenadas cartesianas no plano, ou seja, x e y, e o ângulo de orientação θ. Conjuntos alternativos de coordenadas podem ser obtidos para descrever a posição e orientação do robô. Em (ASTOLFI, 1994; KÜHNE, 2005; CHWA, 2004) é utilizada a seguinte transformação de coordenadas (vide Figura 7):

$$\begin{aligned} e &= \sqrt{x^2 + y^2} \\ \psi &= \text{atan2}(y, x) \\ \alpha &= \theta - \psi. \end{aligned}$$

que pode ser utilizada para se obter o seguinte modelo cinemático:

$$\begin{cases} \dot{e} = v \cos \alpha \\ \dot{\psi} = v \dfrac{\text{sen}\, \alpha}{e} \\ \dot{\alpha} = -v \dfrac{\text{sen}\, \alpha}{e} + w. \end{cases} \tag{6}$$

Observa-se que se $e = 0$ tanto $\dot{\psi}$ quanto $\dot{\alpha}$ não estão definidas, já que a variável e se encontra no denominador em (6). Embora $\dot{\psi}$ e $\dot{\alpha}$ tendam a $\pm\infty$ ao redor da origem é possível se encontrar uma lei de controle que estabiliza o sistema neste ponto com base em uma função de Lyapunov quadrática em relação ao estado do sistema conforme considerado em (6), tornando o sistema em malha fechada estável (LAGES, 1998). No Capítulo 5 será avaliada a utilidade desta transformação para o projeto de um controlador MPC, conforme apresentado em (KÜHNE, 2005). Um outro controlador empregando esta e outra transformação de coordenadas será apresentado, e seu desempenho será avaliado tanto com relação ao sistema realimentado quanto à sua aplicabilidade em tempo real.

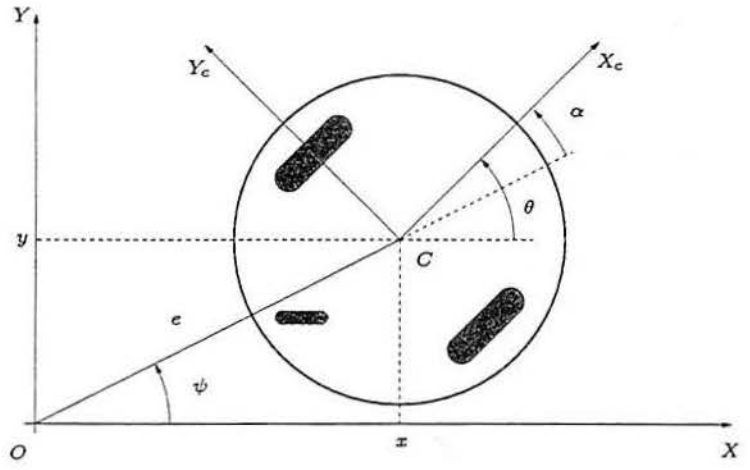

Figura 7: Relação entre coordenadas polares e cartesianas para um robô móvel com acionamento diferencial.

3 CONTROLE PREDITIVO BASEADO EM MODELO

3.1 Introdução

O termo controle preditivo baseado em modelo (MPC) na realidade não se refere a uma única estratégia de controle mas sim dá nome a um conjunto de métodos de controle que utilizam explícitamente um modelo do sistema para obter um sinal de controle ao minimizar uma função-custo. Idéias relacionadas a controle preditivo são (CAMACHO; BORDONS, 1999):

- Uso explícito de um modelo do sistema para se obter uma predição da saída do mesmo para instantes de tempo futuros. O intervalo de tempo transcorrido entre o instante de tempo em que é feita a medição da saída do sistema e o instante de tempo da última predição da saída ou do estado do sistema é chamado *horizonte de predição* (vide Figura 8);
- Cálculo de uma seqüência de controle que minimiza uma função-custo ao ser aplicada no sistema por um determinado período, chamado *horizonte de controle*;
- Estratégia recorrente: uma vez obtida a seqüência de controle, o primeiro valor da mesma é aplicado no sistema e no próximo instante de amostragem deslocam-se os horizontes de predição e controle, repetindo-se o procedimento a partir da nova leitura da saída. Por este motivo MPC é também conhecido como *controle por horizonte deslizante* (RHC).

A função-custo deve ser tal que possui valor mínimo para o comportamento ideal do sistema. Este comportamento tipicamente está relacionado a atingir um determinado valor da saída (referência ou *set point*) ou seguir um padrão pré-determinado (trajetória). Os problemas relacionados a estes objetivos são chamados problema de estabilização em um ponto e de rastreamento de trajetória. Convém ressaltar que a obtenção de uma função-custo que possua valor mínimo quando o sistema se comporta da maneira desejada não significa que tal situação será atingida. A relação entre diferentes funções-custo para os problemas em estudo neste trabalho e o comportamento do sistema realimentado será tratada nos próximos capítulos.

3.2 Perspectiva Histórica

O MPC começou a ser utilizado no final da década de 1970, inicialmente na indústria. Seguem fatores que impulsionaram seu desenvolvimento (CAMACHO; BORDONS, 1999):

- O problema de controle é abordado de maneira bastante direta no domínio do tempo;
- Pode ser expandido facilmente para o caso multivariável;
- Intrinsecamente pode levar em consideração a existência de atrasos de transporte;
- Pode ser empregado quando referências futuras são conhecidas.
- Diferentes modelos podem ser utilizados (entrada-saída, representação interna).

Inicialmente foi empregado para sistemas lineares ou para sistemas linearizados. Para este tipo de aplicação, considerando uma função-custo do tipo

$$\Phi(k) = \sum_{i=1}^{n} a_i|x_i(k+1)| + \sum_{i=1}^{m} b_i|u_i(k+1)|,$$

onde n é o número de estados e m é o número de entradas, e restrições lineares, pode-se resolver o problema através de um algoritmo de programação linear (CAMACHO; BORDONS, 1999). Considerando uma função custo quadrática, a inexistência de restrições e o problema de estabilização em um ponto, tem-se um controlador que implementa uma realimentação linear de estados (DORATO; ABDALLAH; CERONE, 1995). Também para uma função-custo quadrática e o problema de estabilização em um ponto, mas considerando a existência de um distúrbio multiplicativo gaussiano com estrutura de covariância conhecida também pode ser encontrado um controlador por realimentação linear de estados (ATHANS, 1971; DORATO; ABDALLAH; CERONE, 1995). Em ambos os casos a solução do problema se dá a partir da solução de equações de Riccati. Porém, consideração de restrições normalmente implica inexistência de solução analítica (BEMPORAD et al., 2002). Ainda considerando-se uma função-custo quadrática e restrições lineares pode-se reduzir o problema de otimização a um problema de programação quadrática, para o qual algoritmos numericamente robustos, levando a soluções ótimas globais, estão disponíveis (KÜHNE; LAGES; GOMES DA SILVA JR., 2004). Finalmente, para problemas onde há restrições não lineares e diferentes funções-custo o problema de otimização é não-linear, não existindo métodos capazes de resolvê-lo após um número conhecido e limitado de iterações para um problema qualquer. A possibilidade de se implementar um controlador MPC nestas condições que seja factível em tempo real é dependente do sistema dinâmico em questão, do algoritmo utilizado e de parâmetros do mesmo.

Na Seção 3.3 será abordado o problema de estabilização em um ponto, que é resolvido através de um controlador MPC. Na Seção 3.4, considerando-se o problema de rastreamento de trajetória, será obtido um controlador MPC mais simples, que considerará uma função-custo que pode ser reescrita na forma quadrática padrão.

3.3 MPC Não Linear

Seja um sistema dinâmico descrito por

$$\dot{\mathbf{x}} = f_c(\mathbf{x}(t), \mathbf{u}(t)), \tag{7}$$

onde $\mathbf{x} \in \mathbb{R}^n$ é o vetor de estados, $\mathbf{u} \in \mathbb{R}^m$ é o vetor de entrada de controle e t é o tempo. Pode-se obter uma discretização de (7) dada por

$$\mathbf{x}(k+1) = f_d(\mathbf{x}(k), \mathbf{u}(k)), \tag{8}$$

onde $k \in \mathbb{N}$ é o instante de amostragem.

Deve-se obter uma seqüência de controle que minimiza uma função-custo relacionada ao objetivo de controle. Normalmente é utilizada uma função quadrática em relação aos estados e entradas. Considere o problema de estabilização em um ponto – mais especificamente na origem – e a função-custo $\Phi(k)$ dada por

$$\Phi(k) = \sum_{j=N_1}^{N_2} \mathbf{x}^{\mathrm{T}}(k+j|k)\mathbf{Q}\mathbf{x}(k+j|k) + \sum_{j=1}^{N_u} \mathbf{u}^{\mathrm{T}}(k+j|k)\mathbf{R}\mathbf{u}(k+j|k), \tag{9}$$

onde $N = N_2 - N_1 + 1$ é o horizonte de predição, N_u é o horizonte de controle, $Q \geq 0$ e $R > 0$ são matrizes de ponderação penalizando o erro de estado e as entradas de controle, respectivamente. Observa-se que não há perda de generalidade ao se considerar a origem como ponto de equilíbrio do sistema já que sempre existe uma mudança de coordenadas tal que o sistema esteja em equilíbrio em um dado ponto qualquer do espaço de estados. Tem-se então que, ao se minimizar $\Phi(k)$, o estado do sistema descrito por (7) tende a zero se $\Phi(k)$ tende a zero e $f_c(\mathbf{0}, \mathbf{0}) = \mathbf{0}$.

Para uma aplicação prática sempre há restrições a que está submetido o sistema. Tais restrições são, principalmente, de três tipos (CAMACHO; BORDONS, 1999):

limites físicos Limites relacionados a barreiras físicas, resultado da própria construção do equipamento. Como são inerentes ao sistema, não podem ser violados. Exemplos são fim do curso de partes móveis sobre guias e vazão máxima de tubulações.

limites de segurança Limites que jamais devem ser violados para se preservar o equipamento ou pessoas próximas ao mesmo, evitar danos ao meio ambiente ou evitar o custoso desligamento do processo.

limites operacionais Limites relacionados ao desempenho do sistema e à qualidade do produto final. Eventualmente podem ser ultrapassados para se preservar limites físicos e de segurança.

Tais limites se traduzem em restrições nas variáveis do sistema, ou seja, nas entradas ou nos estados. Obstáculos físicos, como paredes ou objetos massivos, devem ser considerados como restrições nos estados. Podem ainda ser mapeados como restrições nos estados limites de segurança e operacionais, relativos a valores dos estados que devem ser evitados. Também como limites físicos são consideradas as restrições nas entradas, já que os atuadores estão naturalmente sujeitos a limitações de amplitude, como velocidade e potência. Limites operacionais relativos às entradas podem também ser considerados. Prevenindo variações bruscas dos valores das entradas pode-se aumentar a vida útil das peças que compõem o sistema. Podem então ser definidas expressões de restrição de uma forma geral:

$$\begin{aligned} \mathbf{x}(k+j|k) &\in \mathbb{X}, \quad j \in [N_1, N_2] \\ \mathbf{u}(k+j|k) &\in \mathbb{U}, \quad j \in [0, m] \end{aligned}$$

onde $\mathbb{X}$ e $\mathbb{U}$ são conjuntos fechados e contém os possíveis valores de $\mathbf{x}$ e $\mathbf{u}$, respectivamente. Tais restrições podem ser expressas na forma de desigualdades como:

$$g_x(\mathbf{x}(k+j|k)) \leq \mathbf{a}, \quad j \in [N_1, N_2] \tag{10}$$
$$g_u(\mathbf{u}(k+j|k)) \leq \mathbf{b}, \quad j \in [0, N_u] \tag{11}$$

com $\mathbf{a} \in \mathbb{R}^{l_x}$ e $\mathbf{b} \in \mathbb{R}^{l_u}$. O número de desigualdades em $\mathbf{x}$ e $\mathbf{u}$ é dado l_x e l_u, respectivamente.

Definindo-se $\bar{\mathbf{x}}$, $\bar{\mathbf{x}}^\star$, $\bar{\mathbf{u}}$ e $\bar{\mathbf{u}}^\star$ como

$$\bar{\mathbf{x}} \triangleq \begin{bmatrix} \mathbf{x}(k|k) \\ \mathbf{x}(k+1|k) \\ \vdots \\ \mathbf{x}(k+N_2|k) \end{bmatrix} \quad \bar{\mathbf{x}}^\star \triangleq \begin{bmatrix} \mathbf{x}^\star(k|k) \\ \mathbf{x}^\star(k+1|k) \\ \vdots \\ \mathbf{x}^\star(k+N_2|k) \end{bmatrix}$$

$$\bar{\mathbf{u}} \triangleq \begin{bmatrix} \mathbf{u}(k|k) \\ \mathbf{u}(k+1|k) \\ \vdots \\ \mathbf{u}(k+N-1|k) \end{bmatrix} \quad \bar{\mathbf{u}}^\star \triangleq \begin{bmatrix} \mathbf{u}^\star(k|k) \\ \mathbf{u}^\star(k+1|k) \\ \vdots \\ \mathbf{u}^\star(k+N-1|k) \end{bmatrix}$$

pode-se então formalizar o problema do otimização como encontrar uma seqüência de controle $\bar{\mathbf{u}}^\star$ e uma seqüência de estados $\bar{\mathbf{x}}^\star$ que minimizam a função-custo $\Phi(k)$ e satisfazem as restrições (10), (11), ou seja:

$$\bar{\mathbf{u}}^\star, \bar{\mathbf{x}}^\star = \arg\min_{\bar{\mathbf{u}}, \bar{\mathbf{x}}} \Phi(k) \tag{12}$$

sujeito a

$$\begin{aligned} \mathbf{x}(k|k) &= x(k) \\ \mathbf{x}(k+j|k) &= f_d(\mathbf{x}(k+j-1|k), \mathbf{u}(k+j-1|k)), \quad j \in [1, N_2] \\ g_x(\mathbf{x}(k+j|k)) &\leq \mathbf{a}, \quad j \in [N_1, N_2] \\ g_u(\mathbf{u}(k+j|k)) &\leq \mathbf{b}, \quad j \in [0, N_u]. \end{aligned}$$

O problema expresso por (12) é então resolvido a cada instante de amostragem sendo que o sinal de controle é dado por $\mathbf{u}^\star(k|k)$, sendo a lei de controle em tempo contínuo dada por:

$$\mathbf{u}(t) = \mathbf{u}^\star(k|k), \text{para } t \in [kT, (k+1)T) \tag{13}$$

onde T é o período de amostragem.

No próximo instante de amostragem $k+1$ obtém-se novamente a medida do estado $(x(k+1))$, desloca-se o horizonte de predição para a frente e repete-se o procedimento, substituindo-se $x(k)$ por $x(k+1)$ no problema (12). A Figura 8 ilustra o funcionamento do MPC para $N_1 = 0$.

Observa-se que as restrições definidas por (10) e (11) não necessariamente definem um conjunto convexo, o que pode tornar a minimização significativamente mais custosa.

3.4 MPC Linearizado

Considere um sistema dinâmico descrito por (7) cuja discretização é dada por (8). Um modelo linear pode ser obtido considerando-se o erro com relação a uma

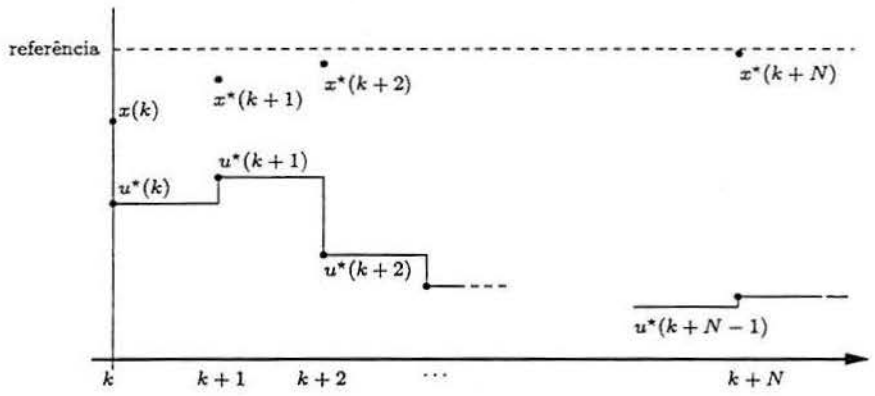

Figura 8: Esquema do MPC.

trajetória de referência também descrita por (7). A trajetória de referência $\mathbf{x}_r$ e a entrada de referência $\mathbf{u}_r$ estão relacionadas da seguinte maneira:

$$\dot{\mathbf{x}}_r = f_c(\mathbf{x}_r, \mathbf{u}_r). \tag{14}$$

Expandindo o lado direito de (7) em série de Taylor em torno do ponto $(\mathbf{x}_r, \mathbf{u}_r)$ e descartando os termos de ordem 2 e superior segue que

$$\dot{\mathbf{x}} = f_c(\mathbf{x}_r, \mathbf{u}_r) + \left.\frac{\partial f_c(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\mathbf{x}_r \\ \mathbf{u}=\mathbf{u}_r}} (\mathbf{x} - \mathbf{x}_r) + \left.\frac{\partial f_c(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}\right|_{\substack{\mathbf{x}=\mathbf{x}_r \\ \mathbf{u}=\mathbf{u}_r}} (\mathbf{u} - \mathbf{u}_r), \tag{15}$$

ou

$$\dot{\mathbf{x}} = f_c(\mathbf{x}_r, \mathbf{u}_r) + f_{\mathbf{x}_r}(\mathbf{x} - \mathbf{x}_r) + f_{\mathbf{u}_r}(\mathbf{u} - \mathbf{u}_r), \tag{16}$$

onde $f_{\mathbf{x}_r}$ e $f_{\mathbf{u}_r}$ são os jacobianos de f_c com respeito a $\mathbf{x}$ e $\mathbf{u}$, respectivamente, calculados em torno do ponto de referência $(\mathbf{x}_r, \mathbf{u}_r)$.

Subtraindo-se então (14) de (16) resulta em:

$$\dot{\tilde{\mathbf{x}}} = f_{\mathbf{x}_r}\tilde{\mathbf{x}} + f_{\mathbf{u}_r}\tilde{\mathbf{u}} \tag{17}$$

Desse modo, $\tilde{\mathbf{x}} \triangleq \mathbf{x} - \mathbf{x}_r$ representa o erro com relação a trajetória de referência e $\tilde{\mathbf{u}} \triangleq \mathbf{u} - \mathbf{u}_r$ é a perturbação da entrada de controle associada a mesma (vide Figura 9 (KÜHNE, 2005)).

A aproximação de $\dot{\tilde{\mathbf{x}}}$ utilizando-se diferenças progressivas levam ao seguinte modelo em tempo discreto:

$$\tilde{\mathbf{x}}(k+1) = \mathbf{A}(k)\tilde{\mathbf{x}}(k) + \mathbf{B}(k)\tilde{\mathbf{u}}(k). \tag{18}$$

onde

$$\mathbf{A}(k) \triangleq \mathbf{I} + f_{\mathbf{x}_r}T, \quad \mathbf{B}(k) \triangleq f_{\mathbf{u}_r}T.$$

Considera-se uma função-custo quadrática em relação a $\tilde{\mathbf{x}}$ e $\tilde{\mathbf{u}}$, ou seja:

$$\Phi(k) = \sum_{j=1}^{N} \tilde{\mathbf{x}}^{\mathrm{T}}(k+j|k)\mathbf{Q}\tilde{\mathbf{x}}(k+j|k) + \tilde{\mathbf{u}}^{\mathrm{T}}(k+j-1|k)\mathbf{R}\tilde{\mathbf{u}}(k+j-1|k). \tag{19}$$

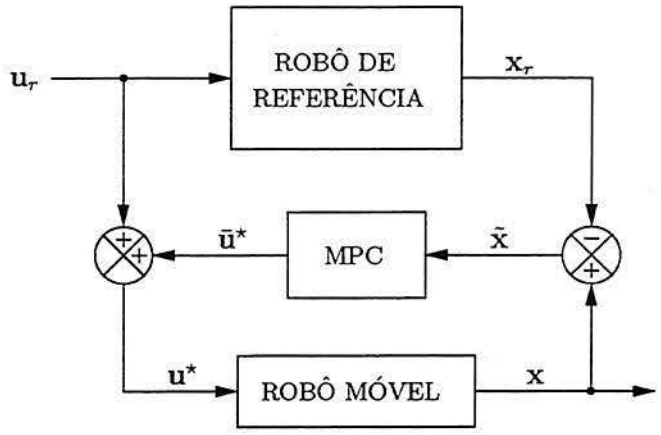

Figura 9: Diagrama de blocos do LMPC.

Com o objetivo de reescrever o problema de otimização na forma quadrática usual introduz-se os seguintes vetores:

$$\bar{\mathbf{x}}(k+1) \triangleq \begin{bmatrix} \tilde{\mathbf{x}}(k+1|k) \\ \tilde{\mathbf{x}}(k+2|k) \\ \vdots \\ \tilde{\mathbf{x}}(k+N|k) \end{bmatrix} \quad \bar{\mathbf{u}}(k) \triangleq \begin{bmatrix} \tilde{\mathbf{u}}(k|k) \\ \tilde{\mathbf{u}}(k+1|k) \\ \vdots \\ \tilde{\mathbf{u}}(k+N-1|k) \end{bmatrix}$$

Desse modo, (19) pode ser reescrita como:

$$\Phi(k) = \bar{\mathbf{x}}^{\mathrm{T}}(k+1)\bar{\mathbf{Q}}\bar{\mathbf{x}}(k+1) + \bar{\mathbf{u}}^{\mathrm{T}}(k)\bar{\mathbf{R}}\bar{\mathbf{u}}(k), \tag{20}$$

com

$$\bar{\mathbf{Q}} \triangleq \begin{bmatrix} \mathbf{Q} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{Q} \end{bmatrix} \quad \bar{\mathbf{R}} \triangleq \begin{bmatrix} \mathbf{R} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{R} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{R} \end{bmatrix}$$

Conseqüentemente, é possível, com base em (18) escrever $\bar{\mathbf{x}}(k+1)$ como:

$$\bar{\mathbf{x}}(k+1) = \bar{\mathbf{A}}(k)\tilde{\mathbf{x}}(k|k) + \bar{\mathbf{B}}(k)\bar{\mathbf{u}}(k), \tag{21}$$

com

$$\bar{\mathbf{A}}(k) \triangleq \begin{bmatrix} \alpha(k+1,k) \\ \alpha(k+2,k) \\ \vdots \\ \alpha(k+N,k) \end{bmatrix}$$

e

$$\bar{\mathbf{B}}(k) \triangleq \begin{bmatrix} \beta_{11}(k) & \mathbf{0} & \cdots & \mathbf{0} \\ \beta_{21}(k) & \beta_{22}(k) & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{N1}(k) & \beta_{N2}(k) & \cdots & \beta_{NN}(k) \end{bmatrix}$$

onde β_{ij} é definida como

$$\beta_{ij}(k) \triangleq \alpha(k+i,k+j)\mathbf{B}(k+j-1)$$

e $\alpha(k_1,k_0)$ como

$$\alpha(k_1,k_0) \triangleq \begin{cases} \mathbf{I} & \text{para } k_1 = k_0, \\ \prod_{i=0}^{n-1} \mathbf{A}(k+i) & \text{para } k_1 > k_0. \end{cases}$$

De (20) e (21), pode-se reescrever a função-custo (19) na forma quadrática padrão

$$\Phi(k) = \frac{1}{2}\bar{\mathbf{u}}^{\mathrm{T}}(k)\mathbf{H}(k)\bar{\mathbf{u}}(k) + \mathbf{f}^{\mathrm{T}}(k)\bar{\mathbf{u}}(k) + \mathbf{d}(k) \tag{22}$$

com

$$\begin{aligned} \mathbf{H}(k) &\triangleq 2\left(\bar{\mathbf{B}}^{\mathrm{T}}(k)\bar{\mathbf{Q}}\bar{\mathbf{B}}(k) + \bar{\mathbf{R}}\right) \\ \mathbf{f}(k) &\triangleq 2\bar{\mathbf{B}}^{\mathrm{T}}(k)\bar{\mathbf{Q}}\bar{\mathbf{A}}(k)\tilde{\mathbf{x}}(k|k) \\ \mathbf{d}(k) &\triangleq \tilde{\mathbf{x}}^{\mathrm{T}}(k|k)\bar{\mathbf{A}}^{\mathrm{T}}(k)\bar{\mathbf{Q}}\bar{\mathbf{A}}(k)\tilde{\mathbf{x}}(k|k). \end{aligned}$$

A matriz $\mathbf{H}$ é uma matriz *hessiana*, e deve ser positiva definida. Ela descreve a parte quadrática da função-custo, enquanto que o vetor $\mathbf{f}$ descreve a parte linear. $\mathbf{d}$ é independente de $\tilde{\mathbf{u}}$ e não tem influência na determinação de $\mathbf{u}^\star$.

Para se minimizar (22) pode-se utilizar um algoritmo de programação quadrática. Entretanto, tais algoritmos consideram restrições lineares em sua formulação. Restrições lineares consideram uma região resultante da interseção de semi-espaços. Como semi-espaços são regiões convexas e a intereseção entre dois conjuntos convexos é também um conjunto convexo, tem-se que a região definida por restrições lineares é sempre convexa. As restrições definidas por (10) e (11) podem expressar regiões não-convexas e com formatos quaisquer, portanto a consideração de restrições lineares, apenas, implica perda de generalidade. Isto é justificado, pois, normalmente, algoritmos de programação quadrática conseguem resolver problemas de otimização mais rapidamente que algoritmos de programação não-linear e são capazes de resolver o problema com determinismo temporal. A maior rapidez na obtenção do sinal de controle, por sua vez, é interessante para se reduzir o atraso entre a leitura dos sensores e a aplicação da entrada no sistema. Assim, o problema de otimização é reformulado como encontrar $\bar{\mathbf{u}}^\star$, dado por

$$\bar{\mathbf{u}}^\star = \arg\min_{\bar{\mathbf{u}}} \Phi(k) \tag{23}$$

s. a.

$$\mathbf{d}_1 \leq \mathbf{D}\bar{\mathbf{u}} \leq \mathbf{d}_2 \tag{24}$$

sendo a entrada de controle a ser aplicada no sistema $\mathbf{u}^\star(k)$ no instante k dada por $\mathbf{u}^\star(k) = \tilde{\mathbf{u}}^\star(k) + \mathbf{u}_r(k)$. O funcionamento básico do LMPC pode ser observado na Figura 9. Detalhes quanto à utilização de algoritmos de programação quadrática em um controlador MPC para um robô móvel serão apresentados no Capítulo 4.

4 RASTREAMENTO DE TRAJETÓRIA COM MPC LINEAR

4.1 Formulação do Problema para o robô Twil

Conforme (MORARI; LEE, 1999) e detalhado na Seção 3.4 é possível reescrever a função-custo quadrática em $\tilde{\mathbf{x}}$ e $\tilde{\mathbf{u}}$ na forma quadrática padrão em relação a seqüência de entrada definida na forma de $\bar{u}$ (equações (19) e (22)) se o sistema é linear ou se for considerada um linearização do mesmo. No caso do robô Twil tem-se de (2) que

$$\dot{\mathbf{x}} = f_c(\mathbf{x}, \mathbf{u}) \triangleq \mathbf{E}(\mathbf{x})\mathbf{u}. \tag{25}$$

A vantagem de expressar o problema de minimização na forma quadrática padrão é que torna possível a utilização de algoritmos de programação quadrática. O uso de tais algoritmos é interessante devido ao fato de serem bastante eficientes. Adicionalmente, estão disponíveis diversos pacotes de software para solução deste tipo de problema.

Observa-se em (25) a utilização do modelo cinemático de postura. Tal modelo ignora a distribuição da massa que compõe as partes do robô. Ele descreve o movimento do robô no plano cartesiano a partir das velocidades linear e angular do mesmo, sendo que o comportamento do sistema é tal que há movimento sempre que há uma entrada $\mathbf{u}$ não-nula. Similarmente, o movimento cessa instantâneamente se a entrada assume valor nulo. Tal modelo dinâmico é chamado *sem deriva*. Apesar de não considerar aspectos importantes do robô móvel é sabido que (2) constitui uma aproximação razoável se for considerado o limite que as velocidades linear e angular podem alcançar a partir do torque dos motores. Tal aproximação torna-se ainda mais aceitável quando não há variações bruscas freqüentes na velocidade do robô e portanto pode ser utilizada para se tratar do problema de rastreamento de uma trajetória contínua (KÜHNE, 2005).

O modelo obtido por uma linearização em torno de um ponto estacionário qualquer não é controlável para um robô móvel descrito por (1). Adicionalmente, para $\mathbf{u} = \mathbf{0}$ tem-se $f_c(\mathbf{x},\mathbf{0}) = \mathbf{0}$. Entretanto, a linearização se torna controlável se o vetor de entrada $\mathbf{u}$ é não-nulo (SAMSON; AIT-ABDERRAHIM, 1991a). Isto implica rastreamento de trajetória ser possível com MPC linear (ESSEN; NIJMEIJER, 2001).

A partir de (25) e da formulação na Seção 3.4 pode-se transformar o problema de rastreamento de trajetória para o robô Twil em um problema de programação quadrática. A partir de (18) e definindo-se $\mathbf{x}_r = [x_r \;\; y_r \;\; \theta_r]^{\mathrm{T}}$ e $\mathbf{u}_r = [v_r \;\; w_r]^{\mathrm{T}}$ tem-se

$$\tilde{\mathbf{x}}(k+1) = \mathbf{A}(k)\tilde{\mathbf{x}}(k) + \mathbf{B}(k)\tilde{\mathbf{u}}(k), \tag{26}$$

onde

$$\mathbf{A}(k) \triangleq \begin{bmatrix} 1 & 0 & -v_r(k)\operatorname{sen}\theta_r(k)T \\ 0 & 1 & v_r(k)\cos\theta_r(k)T \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{B}(k) \triangleq \begin{bmatrix} \cos\theta_r(k)T & 0 \\ \operatorname{sen}\theta_r(k)T & 0 \\ 0 & T \end{bmatrix}$$

Normalmente algoritmos de programação quadrática consideram apenas restrições do tipo

$$\mathbf{M_{eq}}\xi = \mathbf{m}_{eq} \tag{27}$$

ou

$$\mathbf{M}_{in}\xi \leq \mathbf{m}_{in} \tag{28}$$

onde ξ é representa variável de decisão do problema quadrático. Tais restrições são chamadas lineares devido a ξ sofrer uma transformação linear no lado esquerdo de (27) e (28). Porém, a restrição descrita por (28) equivale a uma saturação, que é uma operação não linear.

As restrições reais relativas à velocidade do robô não são diretamente os valores de velocidade linear e angular, mas sim as velocidades das rodas. Em (KÜHNE, 2005) foram utilizados limites em v e w, o que leva a uma região maior do que a fisicamente possível, como mostra a Figura 10. Restrições expressas nestas variáveis não expressam a realidade uma vez que para velocidade linear máxima as duas rodas estão na velocidade angular máxima $\dot{\varphi}_{max_i}$, portando a velocidade angular w tem que ser nula. Isto significa que a capacidade de um robô móvel com acionamento diferencial de fazer curvas diminui conforme sua velocidade aumenta, o que não se podia observar nas trajetórias percorridas pelo robô da partida até se aproximar da referência obtidas anteriormente. As velocidades das rodas esquerda e direita

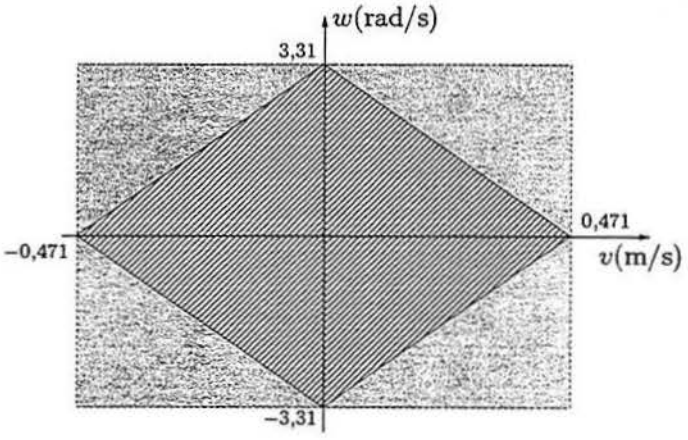

Figura 10: Restrições nas rodas: regiões que satisfazem as restrições nas entradas considerando-se limitações em $\dot{\varphi}_{\min}$ e $\dot{\varphi}_{\max}$ (região hachurada) e limitações em $v_{\max}$ e $w_{\max}$ (região em cinza).

$\dot{\varphi}_1$ e $\dot{\varphi}_2$ podem ser expressas como uma combinação linear das velocidades linear e angular:

$$\mathbf{u}(k) = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ -\frac{r}{l_f} & \frac{r}{l_f} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_1(k) \\ \dot{\varphi}_2(k) \end{bmatrix} = \mathbf{P}\dot{\varphi}(k)$$

O limite relativo às entradas de controle pode então ser expresso definido-se limites inferior e superior para as velocidades das rodas. O problema de otimização deve então ser resolvido de modo a assegurar que o controle permanecerá dentro destes limites. Considerando-se que as duas rodas são iguais e tem os mesmos limites de velocidade máximo e mínimo $\dot{\varphi}_{min_i}$ e $\dot{\varphi}_{max_i}$ pode-se descrever as restrições relativas às entradas como

$$\dot{\varphi}_{min}(k) \leq \dot{\varphi}(k) \leq \dot{\varphi}_{max}(k). \tag{29}$$

A restrição de desigualdade em (28) pode ser reescrita como

$$\mathbf{d}_{min} \leq \mathbf{D}\bar{\mathbf{u}} \leq \mathbf{d}_{max} \tag{30}$$

considerando-se

$$\mathbf{M}_{in} \triangleq \begin{bmatrix} \mathbf{D} \\ -\mathbf{D} \end{bmatrix} \quad \xi \triangleq \bar{\mathbf{u}} \quad \mathbf{m}_{in} \triangleq \begin{bmatrix} \mathbf{d}_{max} \\ -\mathbf{d}_{min} \end{bmatrix}$$

Entretanto, não é possível considerar-se $\mathbf{D} = \mathbf{I}$ em (30) e $\mathbf{P}\dot{\varphi}_{min}$, $\mathbf{P}\dot{\varphi}_{max}$ como limites mínimo e máximo de $\mathbf{u}(k)$ por que a operação de saturação não é linear. Multiplicando-se (29) por $\mathbf{P}^{-1}\mathbf{P}$ e substituindo-se $\mathbf{u}$ por $\tilde{\mathbf{u}} + \mathbf{u}_r$ pode-se reescrever (29) como

$$\dot{\varphi}_{min}(k) - \mathbf{P}^{-1}\mathbf{u}_r(k) \leq \mathbf{P}^{-1}\tilde{\mathbf{u}}(k) \leq \dot{\varphi}_{max}(k) - \mathbf{P}^{-1}\mathbf{u}_r(k), \tag{31}$$

que é uma expressão válida para as restrições relativas à entrada, e pode então ser reescrita conforme (28) com

$$\mathbf{M}_{in} \triangleq \begin{bmatrix} \text{diag}(\mathbf{P}^{-1}, \ldots, \mathbf{P}^{-1}) \\ -\text{diag}(\mathbf{P}^{-1}, \ldots, \mathbf{P}^{-1}) \end{bmatrix} \quad \xi \triangleq \bar{\mathbf{u}} \quad \mathbf{m}_{in} \triangleq \begin{bmatrix} \dot{\varphi}_{max} \\ \vdots \\ \dot{\varphi}_{max} \\ -\dot{\varphi}_{min} \\ \vdots \\ -\dot{\varphi}_{min} \end{bmatrix}$$

4.2 Implementação em Tempo Real

4.2.1 A biblioteca OOQP

Para a resolução do problema de programação quadrática foi utilizada a biblioteca OOQP (GERTZ; WRIGHT, 2003) (*Object Oriented Software for Quadratic Programming*). A biblioteca está disponível em (GERTZ; WRIGHT, 2007) É utilizado por ela o algorimo de ponto interno corretor-preditor de Mehrotra (MEHROTRA, 1992) com correções múltiplas de Gondzio (GONDZIO, 1996).

A formulação de um problema de programação quadrática utilizada pela biblioteca OOQP é (GERTZ; WRIGHT, 2004):

$$\min \frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathbf{Q}\mathbf{u} + \mathbf{f}^{\mathrm{T}}\mathbf{u} \tag{32}$$

sujeito a

$$\mathbf{A}\mathbf{u} = \mathbf{b}, \quad \mathbf{c}_{min} \leq \mathbf{C}\mathbf{u} \leq \mathbf{c}_{max}, \quad \mathbf{u}_{min} \leq \mathbf{u} \leq \mathbf{u}_{max},$$

onde $\mathbf{u}$ é um vetor coluna de incógnitas (variáveis de decisão), $\mathbf{Q}$ é uma matriz quadrada positiva semidefinida, $\mathbf{A}$ e $\mathbf{C}$ matrizes relacionadas a restrições de igualdade e desigualdade, respectivamente, e o vetores $\mathbf{f}$, $\mathbf{b}$, $\mathbf{c}_{min}$, $\mathbf{c}_{max}$, $\mathbf{u}_{min}$ e $\mathbf{u}_{max}$ de

tamanho apropriado. Alguns dos elementos de $\mathbf{b}$, $\mathbf{c}_{min}$, $\mathbf{c}_{max}$, $\mathbf{u}_{min}$ e $\mathbf{u}_{max}$ podem ter magnitude infinita, ou seja, podem não haver limites mínimos ou máximos em relação a alguns elementos de $\mathbf{Au}$, $\mathbf{Cu}$ ou $\mathbf{u}$.

O uso da biblioteca OOQP utilizando-se a linguagem C++ se dá basicamente através da seqüência

```
QpGenSparseMa27 qp(nu,my,mz,nnzH,nnzA,nnzC);
QpGenData *prob = (QpGenData *) qp.copyDataFromSparseTriple(
    f,      irowQ,  nnzQ,   jcolQ,  dQ,
    ulow,   iulow,  uupp,   iuupp,
    irowA,  nnzA,   jcolA,  dA,     b,
    irowC,  nnzC,   jcolC,  dC,
    cmin,   icmin,  cmax,   icmax );
QpGenVars *vars = (QpGenVars *) qp.makeVariables(prob);
QpGenResiduals *resid = (QpGenResiduals *) qp.makeResiduals(prob);
GondzioSolver s(&qp,prob);
int ierr = s.solve(prob,vars,resid);
```

onde `irowQ`, `nnzQ`, `jcolQ`, `dQ` e `irowA`, `nnzA`, `jcolA`, `dA` são variáveis relacionadas a definição de $\mathbf{Q}$ e $\mathbf{A}$, respectivamente; `ulow` e `iulow`, `uupp` e `iuupp`, `cmin` e `icmin`, `cmax` e `icmax` são relacionadas a definição dos vetores $\mathbf{u}_{min}$, $\mathbf{u}_{max}$, $\mathbf{c}_{min}$ e $\mathbf{c}_{max}$, respectivamente; e as variáveis `f` e `b` contém os valores de $\mathbf{f}$ e $\mathbf{b}$, respectivamente. Os tipos `QpGenSparseMa27`, `QpGenData` e `QpGenVars` estão relacionados à dimensão dos parâmetros do problema, aos parâmetros do mesmo e às variáveis, respectivamente. O tipo `GondzioSolver` contem funções para a otimização em si, sendo que a função-membro relacionada a solução do problema de minimização é `solve`. A solução e outros dados obtidos a minimização são armazenados na variável do tipo `QpGenVars`.

As matrizes são armazenadas linha a linha de forma esparsa, isto é, somente as posições da matriz onde há valores não-nulos precisam ser armazenadas pelas variáveis do problema. No caso da matriz Q, que é hessiana, somente a diagonal principal e as posições acima precisam ser armazenadas, caso não sejam nulas.

4.2.2 RTAI

O uso de sistema microprocessados se difundiu a partir de 1970, sendo, hoje em dia, utilizados nas mais diversas aplicações. Encontram-se microprocessadores e microcontroladores não apenas em computadores pessoais, mas em toda a sorte de eletrodomésticos hoje fabricados. Em aplicações industriais e em pesquisa a situação não é diferente: diversos dispositivos são acionados, controlados e supervisionados através de microprocessadores. Não obstante, tais atividades costumam estar sujeitas a restrições mais rígidas em relaçao ao seu funcionamento, notadamente com relação a confiabilidade e tempo de resposta a estímulos.

Aplicações de uso geral são normalmente sujeitas a requisitos relativamente pouco estritos quanto ao funcionamento. Eventuais erros ou demoras podem ser aceitáveis para alguma operação realizada pelo dispositivo. Por outro lado, aplicações de uso específico podem estar sujeitas a requisitos estritos de tempo e necessitar de respostas adequadas. A não satisfação dos requisitos de tempo ou erro no funcionamento podem ser inaceitáveis. Normalmente refere-se a estas últimas aplicações como *sistemas de tempo real.*

É considerado neste trabalho o uso de computadores pessoais (PCs) em sistemas de controle. Sistemas operacionais comuns utilizados em computadores pessoais,

como Windows e (GNU/)Linux, são projetados de maneira a disponibilizar recursos da máquina de maneira apropriada ou aceitável para diversas tarefas executadas pelo computador do ponto de vista do usuário. Estas tarefas podem estar relacionadas a uma enorme variedade de aplicações, como utilizar um navegador, editor de texto ou tocador de música. Os recursos da máquina, como processador e memória RAM, são normalmente distribuídos entre estas várias tarefas de maneira que o desempenho de cada uma delas não fica significativamente prejudicado, ou seja, de modo que não possa ser percebido pelo usuário. Assim, todas as tarefas devem ter acesso ao processador e outros recursos de tempos em tempos para que, aparentemente, todos os programas sejam executados "ao mesmo tempo." Não há, entretanto, garantias quanto à capacidade de uma dada tarefa de ter acesso aos recursos de que necessita do instante em que os requer até um determinado tempo limite (*deadline*). Similarmente, não se tem ferramentas confiáveis para se garantir que, uma vez obtidos os recursos, determinada tarefa tenha acesso aos mesmos por tempo suficiente para realizar determinada função.

Aplicações de engenharia, como sistemas de controle e automação digitais ou de aquisição de dados, podem necessitar de acesso a recursos em um intervalo finito de tempo. Não ter acesso a esses recursos pode causar danos a equipamentos, por em risco a segurança de funcionários ou de alguma maneira comprometer a atividade a que o equipamento se destina. Enfim, garantias temporais podem ser intrínsecas ao bom funcionamento do sistema como um todo. Isto significa que é necessário se poder garantir estes recursos para determinadas tarefas dentro do tempo especificado, mesmo que outras de menor importância quase não tenham acesso aos mesmos. Tais garantias não podem ser fornecidas por sistemas operacionais comuns, sistemas ditos *não de tempo real*. Similarmente, sistemas operacionais capacitados a fornecer tais garantias são ditos *de tempo real*.

O escalonamento de tarefas e a alocação de recursos em sistemas operacionais não de tempo real não é feita de maneira que se possa garantir determinismo temporal. Tipicamente, o escalonamento das tarefas em tais sistemas operacionais é processado disponibilizando uma parcela de tempo de processamento para cada uma (vide Figura 11). A parcela de tempo é relacionada a prioridade da tarefa, a demanda de processamento e algum outro fator que venha a ser considerado no projeto do sistema operacional. A disponibilidade de processamento para cada tarefa faz com que, aos olhos do usuário, todas sejam executadas "ao mesmo tempo." A alocação de algum outro recurso, como memória ou um dispositivo de entrada-saída, é feita pela tarefa que estiver sendo executada. Existe a possibilidade, embora remota, de uma tarefa com menor prioridade alocar um recurso que também é necessário para outra tarefa, com prioridade maior, simplesmente porque isto foi feito na pequena parcela do tempo em que estava sendo executada. Sistemas operacionais não de tempo real não são projetados de modo a garantir o cumprimento de requisitos de tempo ou disponibilizar acesso a hardware entre diferentes tarefas de um modo previsível (TANENBAUM, 2001).

Em sistemas operacionais de tempo real as tarefas são escalonadas de acordo com a prioridade, apenas (vide Figura 12), e com os recursos que foram alocados pelas mesmas. Uma tarefa de maior prioridade só deixará de ter acesso ao processador, em favor de uma tarefa de menor prioridade, caso esta nececessite de um recurso já alocado. Não há compartilhamento de tempo de acesso a recursos de modo a dar a impressão de que diversos programas rodam simultaneamente. Tarefas de menor

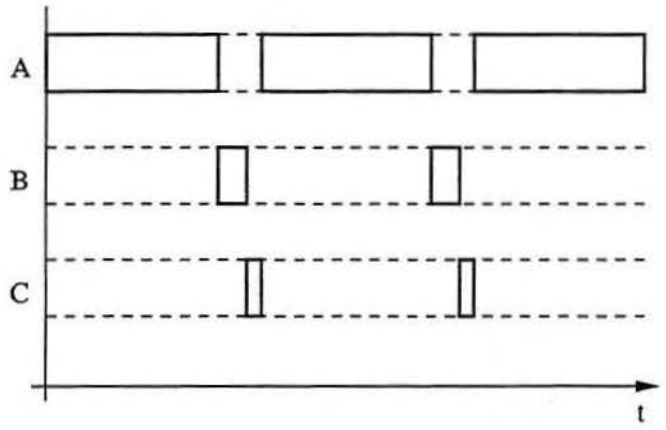

Figura 11: Escalonamento de tarefas com tempos de execução ilimitados em sistemas operacionais não de tempo real. Tarefa A: maior prioridade. Tarefa C: menor prioridade.

prioridade poderão até mesmo não ser executadas por tempo indeterminado, estando o processador e outros recursos alocados por tarefas de maior prioridade (TANENBAUM, 2001).

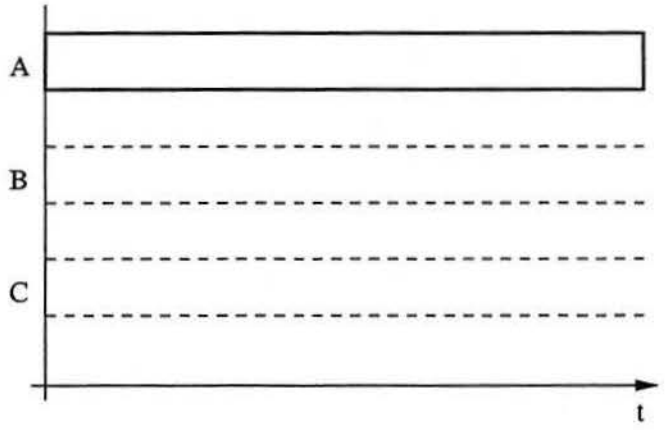

Figura 12: Escalonamento de tarefas com tempo de execução ilimitado em sistemas operacionais de tempo real. Tarefa A: maior prioridade. Tarefa C: menor prioridade.

O escalonamento de tarefas com tempos de execução limitados pode ser verificado nas Figuras 13 e 14. São consideradas três tarefas, com diferentes prioridades e tempos de execução. A tarefa A tem prioridade mais alta e um tempo de execução de três unidades de tempo. A tarefa B tem prioridade intermediária, tendo um tempo de computação de uma unidade de tempo. Já a tarefa C tem a menor prioridade entre as três, sendo o seu tempo de execução igual a duas unidades de tempo. É considerado que todas as tarefas foram iniciadas no mesmo instante de tempo. A Figura 13 ilustra que em sistemas operacionais não de tempo real é possível que uma tarefa de maior prioridade tenha sua execução interrompida para que outra tenha acesso ao processador. Isto pode acontecer simplesmente para evitar *livelock*, ou seja, que parte das tarefas sejam constantemente suspensas em função de outras, com prioridade superior. A Figura 14 ilustra o comportamento esperado caso as tarefas sejam iniciadas ao mesmo tempo em um sistema operacional de tempo real.

Observa-se que as tarefas de menor prioridade só tem acesso ao processador após as tarefas de prioridade superior terminarem de executar (FARINES; SILVA FRAGA; OLIVEIRA, 2000).

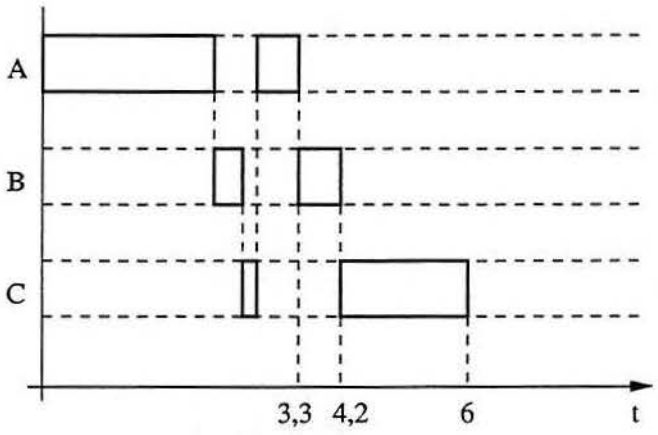

Figura 13: Escalonamento de tarefas com tempos de execução limitados em sistemas operacionais não de tempo real. Tarefa A: maior prioridade. Tarefa C: menor prioridade.

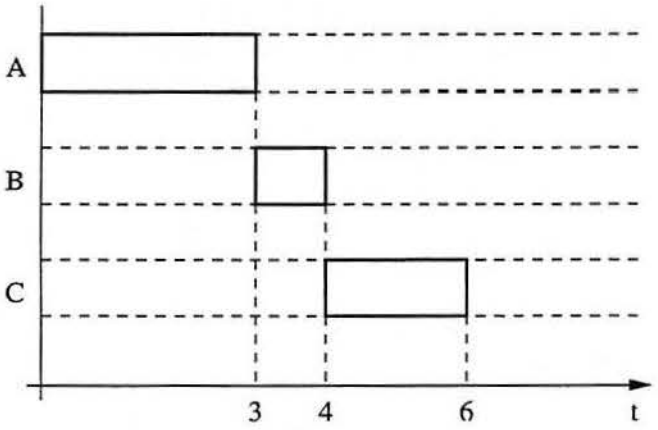

Figura 14: Escalonamento de tarefas com tempo de execução limitado em sistemas operacionais de tempo real. Tarefa A: maior prioridade. Tarefa C: menor prioridade.

Um sistema operacional orientado a prioridades, cujo funcionamento é retratado pelas Figuras 12 e 14, é adequado quando o único recurso a ser compartilhado entre as diferentes tarefas que compõem o sistema de tempo real tem como único recurso compartilhado o processador. Entretanto, é comum a existência de outros recursos que devem ser compatilhados entre as tarefas. O acesso a tais recursos pode ser tal que apenas uma tarefa pode utilizá-lo durante um intervalo de tempo, denotando *compartilhamento em exclusão mútua*. É considerada uma *relação de exclusão* entre as tarefas, sendo que a seção da tarefa que acessa o recurso compartilhado nestes termos chamada de *seção crítica*. Um exemplo é a escrita e leitura em uma região de memória. Considere um sistema em que um dado necessita de mais de um endereço de memória para ser armazenado, sendo que há uma tarefa responsável pela atualização (escrita) deste dado e outra que o utiliza para tomar alguma ação. Sendo

o dado constituído de mais de um endereço de memória é importante que a leitura seja realizada em regime de exclusão mútua com a escrita, caso contrário poderá ser obtido um dado corrompido pela tarefa que o utiliza (FARINES; SILVA FRAGA; OLIVEIRA, 2000).

É considerada neste trabalho a implementação de um controlador para um sistema dinâmico, o qual deve fornecer o sinal de controle em tempo inferior ao período de amostragem utilizado. À tarefa relacionada ao cálculo do sinal de controle é atribuída a máxima prioridade, e o deadline relacionado à mesma é, portanto, o período de amostragem do sistema. Esta restrição foi considerada do tipo *hard*, ou seja, é intrínseco para o funcionamento do sistema como um todo que ela seja satisfeita. Este tipo de restrição contrasta com *soft*, quando é aceitável que *deadlines* sejam perdidos ocasionalmente, havendo tratamento de exceção para avaliar a ação que o sistema deve tomar quando o limite de tempo para a execução da tarefa não pode ser respeitado.

O sistema RTAI (Linux Realtime Application Interface) (DOZIO; MANTEGAZZA, 2003) é uma extensão do sistema operacional Linux com a finalidade de oferecer funcionalidades de tempo real. Foi desenvolvido no DIAPM (*Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano*). O sistema consiste de um *patch* ao kernel do Linux, fornecendo uma camada de abstração de hardware, e uma série de utilidades direcionadas a programadores que precisem trabalhar com sistemas de tempo real.

Através da *Real Time Hardware Abstraction Layer* do RTAI (RTHAL-RTAI) é implementada uma camada de abstração relacionada ao hardware. Ela realiza 3 (três) funções principais (MANTEGAZZA, 2006):

- agrupa ponteiros relacionados a dados internos e funções em uma única estrutura, `rthal`, permitindo *trapeamento*[1] de funcionalidades importantes para tempo real, podendo ser dinamicamente direcionadas para funções de emulação do RTAI se *hard real time* for necessário;
- disponibiliza substitutas das funções agrupadas pela estrutura `rthal` e direciona ponteiros para as mesmas;
- substitui as chamadas de função originais por ponteiros da estrutura `rthal` para todas as funções do kernel que as utilizam.

Estas modificações tornam possível a preempção do kernel e do escalonador de tarefas do Linux e a implementação de parte de um sistema operacional (ou até mesmo um sistema operacional por completo) e um escalonador de tempo real. Assim, *threads* de programas de tempo real podem ter prioridade superior a não apenas todas as tarefas escalonadas pelo Linux, mas ao próprio kernel, e serem controladas por um escalonador determinístico.

Além da RTHAL-RTAI o sistema disponibiliza uma série de funcionalidades importantes para o projeto de sistemas de tempo real. Algumas destas funcionalidades, como memória compartilhada, já estavam disponíveis, porém sem fornecer garantias temporais. Incluem-se entre elas:

- funções para utilização de semáforos;

[1] o termo trapeamento denota o tratamento de interrupções geradas por *software*

- funções para utilização de *mailboxes*;
- funções para ajuste do escalonador de tempo real;
- funções relativas a tarefas de tempo real;
- funções para envio de mensagens entre diferentes tarefas;
- drivers de tempo real para portas serial e paralela e placas de comunicação e aquisição e dados;
- módulo LXRT para utilização de tempo real no espaço do usuário.

Tipos de tarefas executadas em uma máquina com o sistema RTAI incluem os seguintes (ALT, 2003):

tarefas de tempo real no kernel Na implementação original do RTAI tarefas de tempo real deveriam ser implementadas como módulos do kernel. Ao invés de serem executados arquivos executáveis, eram introduzidos módulos no kernel relacionados às tarefas de tempo real.

tarefas normais do Linux Não utilizam funcionalidades de tempo real e não constam no registro do RTAI. São escalonadas pelo escalonador do Linux.

tarefas LXRT *soft real-time* Utilizam funcionalidades de tempo real disponibilizadas pelo RTAI e constam no registro, porém utilizam o escalonador do Linux.

tarefas LXRT *hard real-time* Utilizam funcionalidades de tempo real, constam no registro do RTAI e são escalonadas pelo escalonador, também, do RTAI, o qual é capaz de fornecer garantias temporais. O escalonador pode interromper o próprio kernel do Linux e dar acesso ao processador para uma tarefa de *hard real-time*.

O módulo LXRT (Linux Real Time) permite o acesso as funcionalidades de tempo real do RTAI dentro do espaço do usuário, em comparação com a implementação original, que apenas permitia o acesso a recursos específicos de tempo real ao se rodar programas como um módulo do kernel. Funções relacionadas a funcionalidades do RTAI são disponibilizadas através das mesmas chamadas de funções. O uso do módulo LXRT traz vantagens, como definir *threads* de tempo real e não de tempo real no mesmo programa, mas também requer comprometimento do usuário para garantir o determinismo temporal. *Threads* de tempo real não podem ter a memória paginada nem podem gerar chamadas de sistema. A paginação de memória pode ser explicitamente impedida no programa, enquanto que chamadas de sistema devem ser evitadas não utilizando-se funcionalidades que possam gerá-las. São consideradas chamadas de sistema funcionalidades disponibilizadas pelo kernel do Linux, e.g. acesso a arquivos, gerenciamento de contas de usuários, acesso ao *display* (monitor), alocação dinâmica de memória ou criação e encerramento de processos. A utilização de serviços do Linux por parte de uma tarefa de *hard real-time* causará um chaveamento para o modo *soft real-time* até que o serviço relacionado seja completado.

Tarefas de tempo real que utilizem o modo LXRT podem ser de *soft* ou *hard real-time*. A diferença entre as duas é que tarefas de *hard real-time* utilizam o escalonador do RTAI. Quando uma tarefa é registrada no LXRT (chamando a função `rt_task_init`) é criada pelo RTAI uma tarefa de tempo real que será um servidor para o programa. O papel do servidor é executar serviços de tempo real para a tarefa-cliente (CLOUTIERS, 2007). Em contrapartida, apesar de tais serviços serem executados em tempo real pelo servidor, tarefas de *soft real-time* são escalonadas pelo Linux, portanto o programa não executa em *hard real-time* porque pode ter sua execução suspensa por aquele sistema operacional. Segue um esquema passo-a-passo simplificado do uso do módulo LXRT em um programa (SOETENS, 2006):

1. Iniciar o escalonador do Linux com o método FIFO (`SCHED_FIFO`), mais adequado a aplicações com requisitos de tempo em relação ao método de repartição de tempo padrão do Linux (`SCHED_OTHER`);

```
struct sched_param mainsched;

mainsched.sched_priority =
    sched_get_priority_max(SCHED_FIFO) - 1;
if(sched_setscheduler(0, SCHED_FIFO, &mainsched ) == -1 ) {
    cerr << "Error setting scheduler: " << strerror(errno)
                                                << "\n";
    return -1;
}
```

2. Tornar todas as *threads* do programa tarefas do RTAI (`RT_TASK`), permitindo o uso de diversas funcionalidades do RTAI em *soft real time*:

```
unsigned long maintsk_name = nam2num(argv[0]);
if(!(maintsk = rt_task_init(maintsk_name, 1, 0, 0))) {
    cerr << "Can't init master task\n";
    return -1;
}
```

   Observa-se o uso de `nam2num` para a obtenção de uma chave capaz de ser referenciada tanto do kernel quanto do espaço do usuário.

3. Fazer uso da função `rt_allow_nonroot_hrt()` ou rodar o programa como superusuário (root).

4. Travar a memória do processo para impedir paginação:

```
mlockall(MCL_CURRENT | MCL_FUTURE);
```

5. Inicializar o *timer* de tempo real e tornar cada *thread* de *hard real time*, utilizando o escalonador do RTAI:

```
rt_set_oneshot_mode();
int period = (int) nano2count((RTIME)(ST*1e9));
start_rt_timer(period);
rt_make_hard_real_time();
rt_task_make_periodic(maintsk,rt_get_time(),period);
```

Nota-se que, independentemente do número de *threads* ou processos que devam interagir entre si, o *timer* deve ser inicializado apenas uma vez, pois somente um *timer* é utilizado pelo sistema RTAI ao escalonar as tarefas.

6. Esperar pelo próximo período ao final do *loop* relacionado ao mesmo:

```
rt_task_wait_period();
```

7. Utilizar novamente o escalonador do Linux quando não mais forem necessárias operações em tempo real pela *thread*:

```
rt_make_soft_real_time();
```

8. Parar o escalonador de tempo real depois que todas as *threads* de tempo real não necessitem de escalonamento, e.g. jamais antes de uma *thread* chamar `rt_task_wait_period()`, e retirar as *threads* do registro do RTAI:

```
stop_rt_timer();
rt_task_delete(maintsk);
```

Ao se utilizar o escalonador do RTAI, tem-se as tarefas de tempo real sendo escalonadas de modo preemptivo de acordo com a prioridade. A atribuição de prioridades às tarefas é de responsabilidade do projetista do sistema a ser implementado, devendo ser feita com base na importância de cada tarefa e no respectivo *deadline*. Caso existam recursos compartilhados que não o processador, devendo ser utilizados em regime de exclusão mútua, pode-se utilizar um semáforo específico para que seja empregado o algoritmo de *herança de prioridade* (SHA; RAJKUMAR; LEHOCZKY, 1990) por parte do escalonador.

4.2.3 Implementação do controlador LMPC

Uma explicação resumida dos problemas relacionados ao controle de robôs móveis foi dada no Capítulo 3. Dentre eles encontra-se o problema de rastreamento de trajetória, o qual consiste de fazer o robô se movimentar sobre ou o mais próximo possível a uma trajetória previamente conhecida. Considera-se para este problema um robô virtual que se move sobre a mesma ao longo do tempo (KÜHNE; LAGES; GOMES DA SILVA JR., 2004), comportamento que o robô real deve reproduzir da melhor maneira possível.

Pode-se encontrar referências de trabalhos anteriores que tratam do rastreamento de trajetória por parte de robôs móveis em (KÜHNE, 2005). O método é vantajoso pois se pode determinar uma trajetória que contorne obstáculos conhecidos e contemple quaisquer objetivos específicos da aplicação como evitar choques com outros objetos ou seres humanos que se movem no mesmo recinto ou manter o robô afastado de zonas onde possa causar ou sofrer danos. Métodos baseados em MPC são muito vantajosos em casos como este, onde referências são conhecidas *a priori*, podendo o sistema reagir com antecedência às futuras mudanças de referências (CAMACHO; BORDONS, 1999).

O *software* implementando o controlador MPC linearizado foi escrito em C++, utilizando-se o sistema RTAI para garantir a temporização e a biblioteca OOQP para resolver o problema de otimização. Nesta seção são apresentados gráficos mostrando a evolução dos estados conforme simulação a partir do controlador em tempo

real. O comportamento do robô é descrito através da aproximação de (1) pelo método Runge-Kutta de 4ª ordem. A trajetória de referência foi obtida *online* através da aproximação de Euler de (1) e uma seqüência de entrada de referência para o período de amostragem em questão. Observa-se que o modelo do sistema utilizado pelo controlador MPC é também a aproximação de Euler de (1) ou seja, (5). Isto é recomendado, pois a utilização de um modelo composto por equações mais extensas aumentaria grandemente a complexidade do algoritmo. A utilização de uma aproximação por Runge-Kutta de 4ª ordem para se obter o comportamento do robô tem como objetivo obter um comportamento mais próximo do descrito por (1). Para a obtenção da trajetória pode-se utilizar qualquer modelo, pois não é necessário que seja obtida *online* e não é preciso que seja factível.

A configuração inicial do robô e do carro de referência são $\mathbf{x}(0) = [0 \quad -1 \quad \pi/2]^{\mathrm{T}}$ e $\mathbf{x}_r(0) = [0 \ 0 \ 0]^{\mathrm{T}}$, respectivamente. As matrizes de ponderação foram arbitradas como sendo $\mathbf{Q} = \mathrm{diag}(4, 4, 0{,}5)$ e $\mathbf{R} = 0{,}1\mathbf{I}_{2\times 2}$. O horizonte de predição é $N = 5$. Restrições relacionadas à amplitude das variáveis de controle são: $\varphi_{max} = [2\pi \quad 2\pi]^{\mathrm{T}}$ rad/s e $\varphi_{min} = -\varphi_{max}$. O período de amostragem utilizado foi de 50 ms.

Pode ser claramente visto na Figura 15 que o estado assintoticamente converge para a referência. Convém notar que, devido a orientação original, o robô móvel tem que inicialmente afastar-se da referência para lidar com as restrições não-holonômicas. Na Figura 17 pode-se ver que as entradas de controle estão dentro dos limites impostos pelas restrições. A Figura 18 mostra o tempo necessário para

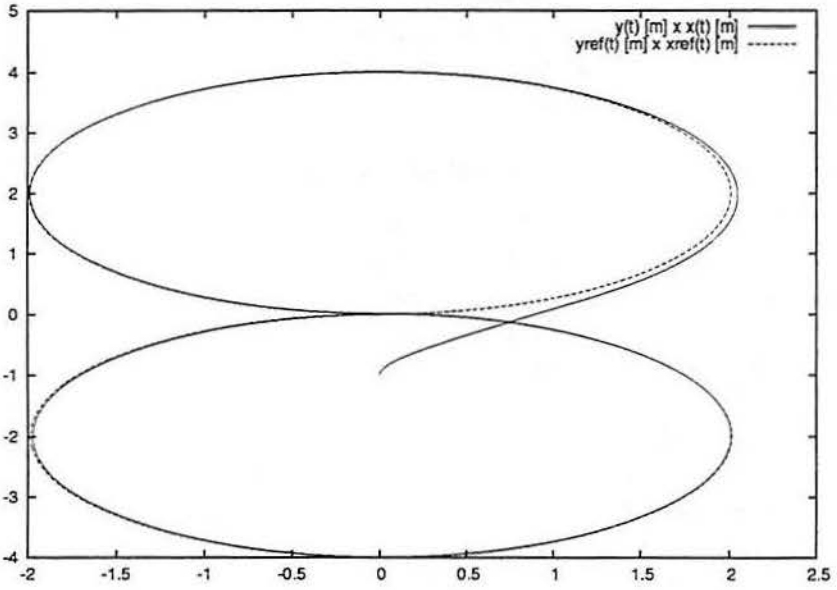

Figura 15: Trajetórias real e de referência para LMPC seguindo uma trajetória em "oito".

calcular a lei de controle como função do tempo para três diferentes processadores. Pode-se ver que até para um processador obsoleto como um Pentium II 300 MHz a lei de controle pode ser calculada dentro de um período de amostragem. As variações no tempo de computação se devem em parte ao processo de otimização, mas principalmente ao *jitter* no escalonamento da tarefa que calcula o controle. As causas do *jitter* são outras tarefas (não de tempo real) executadas pelo processador.

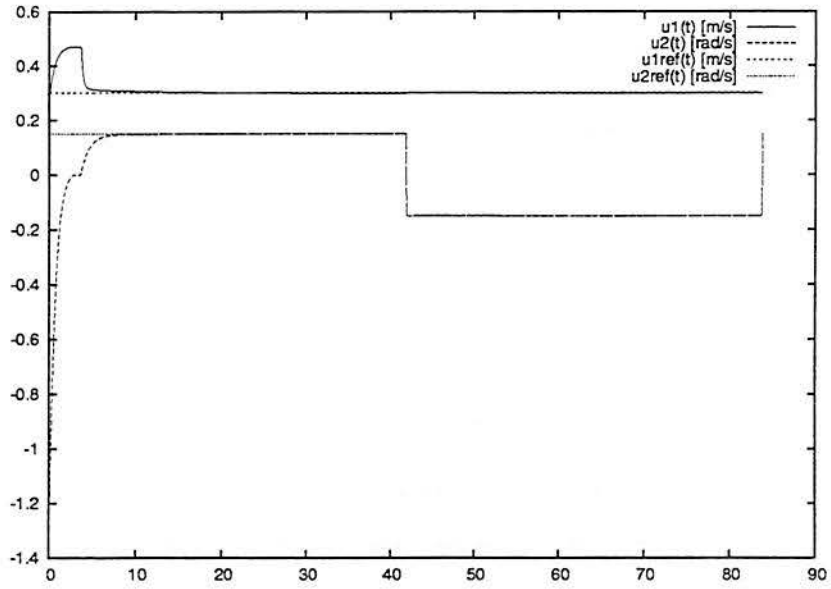

Figura 16: Entradas de controle para LMPC seguindo uma trajetória em "oito".

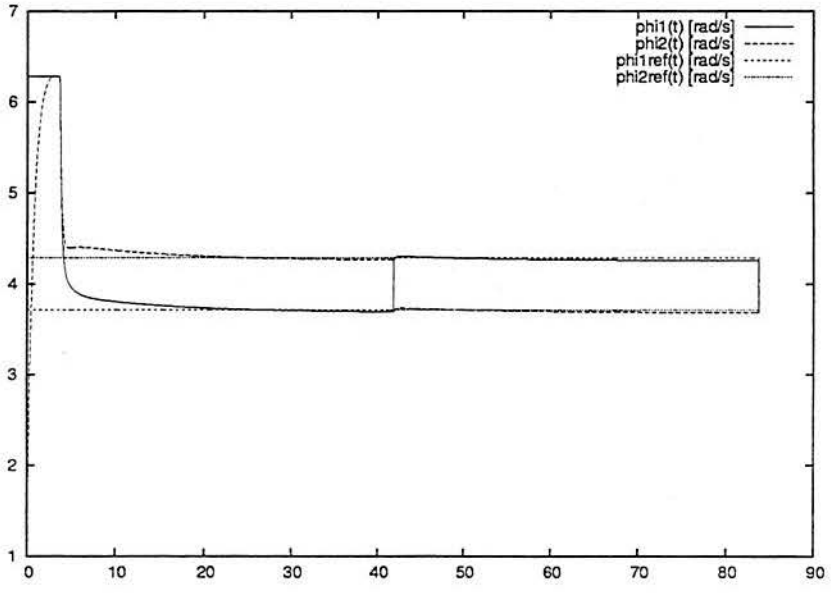

Figura 17: Velocidades das rodas para LMPC seguindo uma trajetória em "oito".

Uma maneira usual de se medir convergência é o erro médio quadrático (ESSEN;

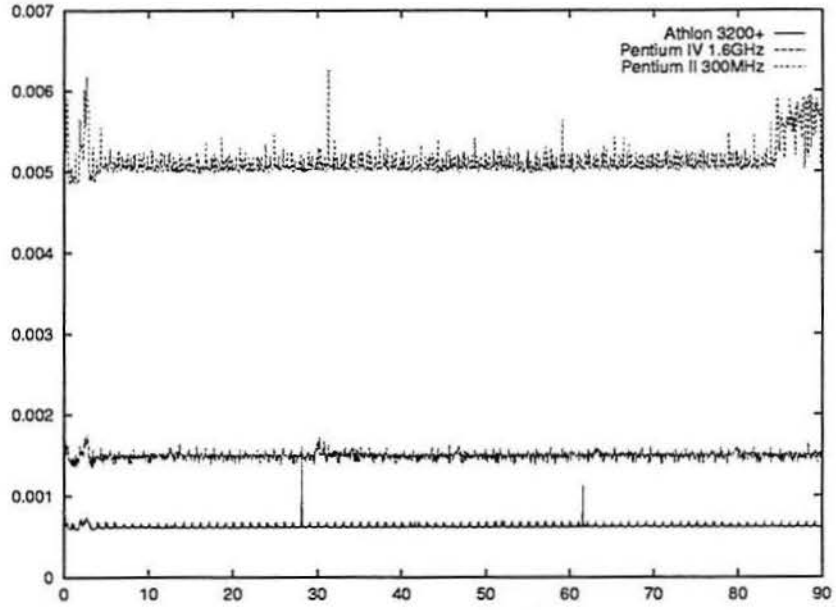

Figura 18: Tempo de computação do sinal de controle para o LMPC.

NIJMEIJER, 2001),

$$\varepsilon \triangleq \frac{1}{K_f} \sum_{k=0}^{K_f} \|\tilde{\mathbf{x}}\|^2,$$

onde K_f é o número de passos necessários para percorrer toda a trajetória e $\|\cdot\|$ denota a norma euclidiana.

A Tabela 1 mostra alguns resultados relativos ao custo computacional, além do erro médio quadrático ε em função do horizonte de predição N: o número médio de operações em ponto flutuante utilizado para calcular a lei de controle a cada instante de amostragem, seu tempo de computação estimado e real e o desempenho em Mflops obtido pelo processador. O tempo de computação estimado foi obtido considerando a performance de 282.5 Mflops para um processador Pentium IV 1.6 GHz, de acordo com (ABURTO, 1992). O verdadeiro tempo de computação foi medido utilizando-se a função `rt_get_time_ns()` do sistema RTAI.

Tabela 1: Horizonte de predição × tempo de computação para LMPC.

N	flo	Tempo de Computação (s)		Mflops	ε
		Estimado	Real		
1	4.343	$15{,}3 \times 10^{-6}$	0,0005861	8,934	1,21994
3	9.529	$33{,}7 \times 10^{-6}$	0,0009833	9,690	0,054278
5	25.643	$90{,}7 \times 10^{-6}$	0,0015	17,095	0,035891
10	160.180	0,000567	0,0040	40,045	0,035358
15	528.570	0,0019	0,0094	56,230	0,038740
20	1.269.500	0,0045	0,0188	67,526	0,047251
30	4.949.000	0,0175	0,0536	92,332	0,073223

De fato, os dados da Tabela 1 contém evidências suficientes de que um computador recente consegue rodar um controlador MPC para um robô móvel. O algoritmo MPC proposto poderia ser calculado para $N = 20$ em cerca de 20 ms enquanto que a dinâmica do robô móvel é tal que períodos da amostragem da ordem de 50 ms são adequados, mostrando que uma implementação em tempo real é bastante factível.

Observa-se ainda que, assim como em (KÜHNE; LAGES; GOMES DA SILVA JR., 2004), o erro ε não apresenta melhorias significativas para $N > 5$. Mais que isso, para valores mais elevados de N o erro ε tende a aumentar. Isto se dá devido a utilização de um modelo aproximado para o erro sistema expresso por (18). Para pontos afastados da referência e conforme o horizonte de predição aumenta o modelo torna-se pouco exato.

Obviamente o horizonte de predição deve ser escolhido de maneira que o tempo de computação seja menor que o período de amostragem. Aqui, $T = 50$ ms, então para $N = 30$ ou maior MPC não é factível. Entretanto, conforme a Tabela 1 e (KÜHNE; LAGES; GOMES DA SILVA JR., 2004), é sabido que para o presente problema não há melhoras perceptíveis na taxa de conversão dos estados para $N > 5$. Adicionalmente, já que para $N = 5$ o tempo de computação é aproximadamente 33 vezes menor que o período de amostragem, trata-se de uma boa escolha para o horizonte de predição.

Analisando-se a Tabela 1 também fica claro que a partir do número de operações em ponto flutuante necessárias e do desempenho do processador não se obtém uma boa estimativa do tempo de processamento já que os tempos de processamento efetivos foram muito maiores que os estimados. Uma razão é que desempenhos de processadores publicados ou medidos são tipicamente valores de pico que não são mantidos em aplicações reais onde operações de ponto flutuante estão intercaladas com operações com inteiros em configurações variadas. Outra razão é que parte do tempo de computação é gasta com operações de manutenção e gerência: alocar e desalocar memória para variáveis, desreferenciar ponteiros, acessar registradores, escalonamento de tempo real, etc. Estas operações não são de ponto flutuante mas devem ser adicionadas ao tempo de execução. Por outro lado, estão relacionadas com o *setup* do problema e não com o número de passos necessário para solução do problema de otimização. Devido a isto elas tomam um tempo aproximadamente constante a cada período de amostragem. Isto explica porque para valores maiores de N a desempenho medido para as operações em ponto flutuante se aproxima do desempenho real: o tempo gasto em gerenciamento do programa permanece constante enquanto a carga computacional associada a operações de ponto flutuante aumenta, uma maior parcela de tempo de execução é gasta com processamento de ponto flutuante, portanto.

5 MPC NÃO LINEAR

5.1 Introdução

O problema de rastreamento de trajetória foi tratado no Capítulo 4, onde foi obtido um controlador MPC baseado em um modelo linearizado do robô móvel. A utilização de um modelo linear permitiu que a função custo fosse reescrita na forma quadrática padrão, e em conjunto com a consideração de restrições lineares nas entradas pôde-se, da mesma maneira que em (KÜHNE, 2005), utilizar um algoritmo de programação quadrática para resolver o problema mais rapidamente que com algoritmos de programação não linear. Entretanto, caso as restrições a que o robô esteja submetido sejam não lineares não mais de pode utilizar um algoritmo de programação quadrática. Além disso, pode ser necessário o conhecimento de expressões analíticas para os gradientes das funções que definem o problema, com o objetivo de alcançar maiores taxas de convergência. Naturalmente, o problema de estabilização em um ponto, tratado no Capítulo 4, pode também ser resolvido através de MPC utilizando um modelo que não tenha sido linearizado. Em (KÜHNE, 2005) são feitas comparações entre o desempenho de controladores NMPC para o problema de rastreamento de trajetória utilizando uma função custo normal e mais outra, previamente utilizada por (ESSEN; NIJMEIJER, 2001) também para controle de robôs móveis. Uma estimativa do tempo de computação necessário para calcular a lei de controle feita a partir do número de operações em ponto flutuante sugeria ser possível a implementação em tempo real dos controladores.

O problema de estabilização em um ponto traz maiores complicações que o problema de rastreamento de trajetória. A utilização de um modelo linearizado com um controlador MPC não é possível pois os modelos resultantes das linearizações tanto em torno da referência quanto em torno do estado não são controláveis. Além disso, para este problema é necessário se levar em consideração as restrições de Brockett, segundo as quais para um sistema sem deriva com restrições não-holonômicas não é possível estabilizar assintoticamente um dado ponto de equilíbrio qualquer através de uma lei de controle contínua e invariante no tempo (BROCKETT, 1982). Todos os modelos que descrevem o robô contém restrições não-holonômicas. Particularmente, os modelos cinemáticos são sem deriva, isto é, $f_c(\mathbf{x},\mathbf{u}) = \mathbf{0}$ para $\mathbf{u} = \mathbf{0}$.

Diversas leis de controle clássicas foram desenvolvidas para se contornar as restrições de Brockett. Tais leis de controle podiam ser *variantes no tempo* ou *não-suaves*. Leis de controle variantes no tempo normalmente consideravam explicitamente funções trigonométricas na variável de tempo em sua formulação e eram tais que as trajetórias percorridas até a referência apresentavam um comportamento oscilatório (TEEL; MURRAY; WALSH, 1995; SAMSON; AIT-ABDERRAHIM, 1991b) ou

aparentemente errático (POMET et al., 1992). Além disso, apresentavam baixas taxas de convergência. Já as leis de controle não-suaves (SØRDALEN, 1993; LAGES, 1998) podiam superar essas desvantagens. Entretanto, a consideração de restrições não podia ser feita diretamente e dependia da condição inicial e ajuste de parâmetros do controlador.

O MPC tem diversas vantagens em relação a estratégias de controle clássicas. A consideração de restrições é feita de maneira direta. A consideração de uma seqüência de controle que minimize a função-custo e satisfaça as restrições relativas aos estados e entradas fica a cargo do algoritmo de otimização. Além disso, é possível operar o sistema próximo destes limites sem extrapolá-los, o que se traduz em maiores taxas de convergência e uma maior região de operação. A grande desvantagem está no fato de que algoritmos de otimização para funções-custo ou restrições não lineares geralmente não oferecem garantias temporais. Especialmente, sequer tem-se garantias de que seja encontrada uma solução. Uma vez encontrada uma solução, também não se tem garantias de que tal ponto seja um mínimo ou máximo global. Enfim, a possibilidade de se aplicar MPC para sistemas não-lineares é profundamente dependente do sistema em sí e do algoritmo de minimização utilizado.

Simulações e implementações reais se fazem necessárias para se avaliar a possibilidade de aplicação do MPC em sistemas não lineares. É um fato que para sistemas não lineares há a necessidade de se considerar restrições não-lineares para a minimização da função-custo, pois diferentemente de sistemas lineares não é fácil se incorporar a relação entre os estados e as entradas naquela função (como na matriz $\mathbf{H}$ na formulação do MPC linearizado). O comportamento do sistema é mapeado no problema de otimização como uma restrição. Não é possível, então, a utilização de um algoritmo de programação quadrática ou linear para resolução do problema. Como a obtenção de uma prova analítica da capacidade de um algoritmo de programação não-linear encontrar a solução do problema e de fazê-lo em um determinado intervalo de tempo pode ser extremamente complexa ou até mesmo impossível, são necessárias simulações e implementações dos controladores e do sistema para se verificar se há convergência para o ponto de equilíbrio. Naturalmente, tendo-se um controlador MPC implementado, pode-se medir o tempo que este necessita para retornar o sinal de controle para o sistema e avaliar se o desempenho é adequado para o sistema real (vide Seções 5.3.2 e 5.4.2).

Na Seção 5.3 serão tratados o problema de rastreamento de trajetória, considerando um controlador MPC e um modelo discreto do sistema, e a implementação em tempo real dos controladores NMPC para rastreamento de trajetória previamente propostos em (KÜHNE, 2005). Na Seção 5.4 será formalizado o problema de estabilização em um ponto conforme (12) para o robô Twil e serão apresentados resultados de experimentos relativos a implementação em tempo real de um controlador MPC para o problema. Será utilizada para resolução dos problemas de otimização não-linear a biblioteca donlp2 (SPELLUCCI, 1998a), disponível *online* em (MITTELMANN, 2007).

5.2 A Biblioteca donlp2

Para a resolução *online* dos problemas de minimização relativos aos controladores NMPC implementados neste trabalho foi utilizado o pacote donlp2 (SPELLUCCI, 1998a). O pacote foi originalmente escrito em Fortran, sendo convertido para ANSI

C por Serge Schoeffert. A partir desta versão foi feita uma conversão para C++ no laboratório LASCAR, sendo então utilizado no controlador na forma de uma biblioteca.

O pacote donlp2 considera o problema de minimizar uma função f sujeita a restrições de igualdade e desigualdade, sendo estas, em geral, não lineares, conforme:

$$f(\mathbf{x}^*) = \min f(\mathbf{x})$$

sujeito a

$$h(\mathbf{x}) = \mathbf{0}$$
$$g(\mathbf{x}) \geq \mathbf{0}$$

onde $\mathbf{x} \in \mathbb{R}^n$ é a variável de decisão, $\mathbf{x}^*$ é o valor da variável de decisão relativo ao mínimo de f, $h : \mathbb{R}^n \rightarrow \mathbb{R}^{n_h}$ é a função relativa às restrições de igualdade, $g : \mathbb{R}^n \rightarrow \mathbb{R}^{n_g}$ é a função relativa às restrições de desigualdade e n é a dimensão do problema (SPELLUCCI, 1999). Detalhes do algoritmo utilizado podem ser encontrados em (SPELLUCCI, 1998b) e (SPELLUCCI, 1998a).

O uso do pacote se dá basicamente através das funções `setup0`, `ef`, `eh`, `eg` e `donlp2`. As quatro primeiras devem ser definidas pelo usuário conforme interface, e a última apenas chamada para que a solução do problema seja encontrada. A função `setup0` contém definições de parâmetros do problema, como a dimensão do problema e número de restrições, e parâmetros específicos do algoritmo, como número máximo de iterações e método e avaliação dos gradientes das diferentes funções relacionadas ao problema (f, h e g). As funções `ef`, `eh` e `eg` definem as funções objetivo, de restrição de igualdade e de restrição de desigualdade, nesta ordem, ou seja, f, h e g. Ainda, outras funções podem ser definidas caso estejam disponíveis expressões analíticas para os gradientes das três funções matemáticas relacionadas ao problema. Além disso, há funções que podem ser definidas pelo usuário em casos especiais e não são necessárias para a resolução do problema.

Com o objetivo de melhor organizar o conjunto de funções que deve ser utilizado, contido no pacote donlp2, estas funções foram encapsuladas em um objeto, sendo criada uma biblioteca que pode então ser utilizada por qualquer programa escrito em C++. O pacote original continha definições de diversos vetores com tamanhos iguais aos máximos que poderiam ser utilizados. Caso houvesse necessidade de resolver problemas de dimensões diferentes era necessário ou se duplicar os arquivos ou recompilá-los, redefinindo o valor de constantes relacionadas ao tamanho destes vetores. Ao organizar as funções em um objeto foi possível utilizar vetores com tamanho igual ao necessário para armazenar os dados do problema, o que também requer menos memória para a execução do programa.

O uso da biblioteca, conforme a implementação em C++, define as seguintes funções-membro:

```
DONLP_PROB(int nx, int neq, int nineq);
virtual ~DONLP_PROB(void);
virtual cvector donlp2(void);
virtual void setup0(void)=0;
virtual void setup(void);
virtual void solchk(void);
virtual void ef(const cvector &x, DOUBLE *fx)=0;
```

```
virtual void egradf(const cvector &x,cvector &gradf);
virtual void eh(INTEGER i,const cvector &x,DOUBLE *hxi);
virtual void egradh(INTEGER i,const cvector &x,cvector &gradhi);
virtual void eg(INTEGER i,const cvector &x, DOUBLE *gxi);
virtual void egradg(INTEGER i,const cvector &x, cvector &gradgi);
```

onde `DONLP_PROB` é o construtor da classe, requerendo o tamanho do problema e a dimensão das restrições; `~DONLP_PROB` é o destrutor da classe, `donlp2` é a função que resolve o problema de minimização, retornando a solução; `setup0` e `setup` são funções que permitem se definir parâmetros do algoritmo; `solchk` é chamada a cada iteração, sendo seu uso opcional; `ef`, `eh` e `eg` definem as função-objetivo $f(\mathbf{x})$, a função de restrição de igualdade $h(\mathbf{x})$ e a função de restrição de desigualdade $g(\mathbf{x})$, nesta ordem, e `egradf`, `egradh` e `egradg`, de uso opcional, definem os gradientes das funções $f(\mathbf{x})$, $h(\mathbf{x})$ e $g(\mathbf{x})$, respectivamente. O uso da biblioteca é feito da seguinte maneira:

1. Cria-se uma classe derivada de `DONLP_PROB`;
2. Define-se, necessariamente, as funções `setup0`, `ef`, `eh` e `eg`.
3. Em caso de se dispor de funções analíticas para os gradientes de $f(\mathbf{x})$, $h(\mathbf{x})$ e $g(\mathbf{x})$ define-se também `egradf`, `egradh` e `egradg`;
4. Utiliza-se o construtor para criar o objeto da classe derivada de `DONLP_PROB`, sendo as variáveis relacionadas ao tamanho do problema e à dimensão das funções de restrição passadas como parâmetros;
5. Utilizasse a função-membro `donlp2` para resolver o problema.

O pacote donlp2 utiliza aproximações numéricas dos gradientes das funções do problema caso estes não sejam disponíveis. Há a possibilidade de se utilizar aproximação for diferenças progressivas, diferenças centrais ou extrapolação de Richardson, requerindo estas opções n, $2n$ ou $6n$ avaliações da função relacionada, respectivamente (SPELLUCCI, 1999).

5.3 Rastreamento de trajetória

5.3.1 Formulação do NMPC para rastreamento de trajetória

Considera-se o robô móvel descrito pelo modelo cinemático de postura (2):

$$\dot{\mathbf{x}} = \mathbf{E}(\mathbf{x})\mathbf{u} = f_c(\mathbf{x},\mathbf{u}), \tag{33}$$

Será implementado um controlador digital para o robô, e para tanto será considerada uma discretização de (33). Especificamente, foi considerada para este trabalho uma discretização do modelo obtida por aproximação de Euler, dada por (5), que pode ser reescrita como:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{x}(k) + \mathbf{E}_d(\mathbf{x}(k))\mathbf{u}(k) \\ &= f_d(\mathbf{x}(k),\mathbf{u}(k)), \end{aligned} \tag{34}$$

com

$$\mathbf{E}_d(\mathbf{x}(k)) = \begin{bmatrix} T\cos x_3(k)\cos x_3(k) & 0 \\ T\cos x_3(k) & 0 \\ 0 & T \end{bmatrix}.$$

Considera-se uma trajetória de referência também descrita por uma discretização de (33), não necessariamente a mesma expressa por (34), ou seja:

$$\mathbf{x}_r(k+1) = f_{d_r}(\mathbf{x}_r(k),\mathbf{u}_r(k)), \tag{35}$$

onde $\mathbf{x}_r(k)$ é o estado de referência e $\mathbf{u}_r(k)$ a entrada de referência no instante amostragem k.

Similarmente ao que foi feito na Seção 3.4, são definidos os erros de trajetória e entrada $\tilde{\mathbf{x}}(k)$ e $\tilde{\mathbf{u}}(k)$ como sendo

$$\tilde{\mathbf{x}}(k) \triangleq \mathbf{x}(k) - \mathbf{x}_r(k) \quad \tilde{\mathbf{u}}(k) \triangleq \mathbf{u}(k) - \mathbf{u}_r(k)$$

É então considerada uma função-custo quadrática em relação aos erros de trajetória e entrada dada por

$$\Phi(k) = \sum_{j=1}^{N} \tilde{\mathbf{x}}^{\mathrm{T}}(k+j|k)\mathbf{Q}\tilde{\mathbf{x}}(k+j|k) + \tilde{\mathbf{u}}^{\mathrm{T}}(k+j-1|k)\mathbf{R}\tilde{\mathbf{u}}(k+j-1|k). \tag{36}$$

Assim, pode-se formular o problema de minimização da mesma maneira que em (37), porém com a função-custo $\Phi(k)$ definida conforme (36), ou seja:

$$\bar{\mathbf{u}}^*, \bar{\mathbf{x}}^* = \arg\min_{\bar{\mathbf{u}},\bar{\mathbf{x}}} \Phi(k) \tag{37}$$

sujeito a

$$\begin{aligned}
\mathbf{x}(k|k) &= x(k) \\
\mathbf{x}(k+j|k) &= f_d(\mathbf{x}(k+j-1|k), \mathbf{u}(k+j-1|k)), && j \in [1, N_2] \\
g_x(\mathbf{x}(k+j|k)) &\leq \mathbf{a}, && j \in [N_1, N_2] \\
g_u(\mathbf{u}(k+j|k)) &\leq \mathbf{b}, && j \in [0, N_u].
\end{aligned}$$

Para se resolver (37) foi utilizado o pacote donlp2 (SPELLUCCI, 1998a). O controlador foi executado em tempo real no sistema RTAI (DOZIO; MANTEGAZZA, 2003)

5.3.2 Implementação do NMPC: rastreamento de trajetória

Foi implementado um controlador em tempo real a partir da formulação da Seção 5.3.1. Este controlador utiliza a função custo normal, quadrática em relação a $\tilde{\mathbf{x}}$ e $\tilde{\mathbf{u}}$, com $\mathbf{Q}$ constante em relação ao instante de predição. No entanto foi considerado na formulação deste controlador restrições nas rodas, ou seja,

$$g_u(\mathbf{u}(k+j|k)) = \begin{bmatrix} \mathbf{P}^{-1} \\ -\mathbf{P}^{-1} \end{bmatrix} \mathbf{u}(k+j|k) \quad \mathbf{b} = \begin{bmatrix} \varphi_{max} \\ -\varphi_{min} \end{bmatrix}$$

e nenhuma restrição nos estados.

Os parâmetros utilizados no controlador NMPC para rastreamento de trajetória foram $\mathbf{Q} = \mathrm{diag}(4,4,0{,}5)$, $\mathbf{R} = \mathrm{diag}(0{,}01, 0{,}01)$ e $\varphi_{max_i} = -\varphi_{min_i} = 2\pi$ rad/s. Pode-se observar a trajetória do robô no plano cartesiano na Figura 19. A entrada relacionada e o tempo de computação necessário constam nas Figuras 20 e 21.

Foi observado, ao se ajustar os parâmetros $\mathbf{Q}$ e $\mathbf{R}$, que o desempenho do sistema realimentado é bastante suscetível a variações nestes parâmetros, especialmente à

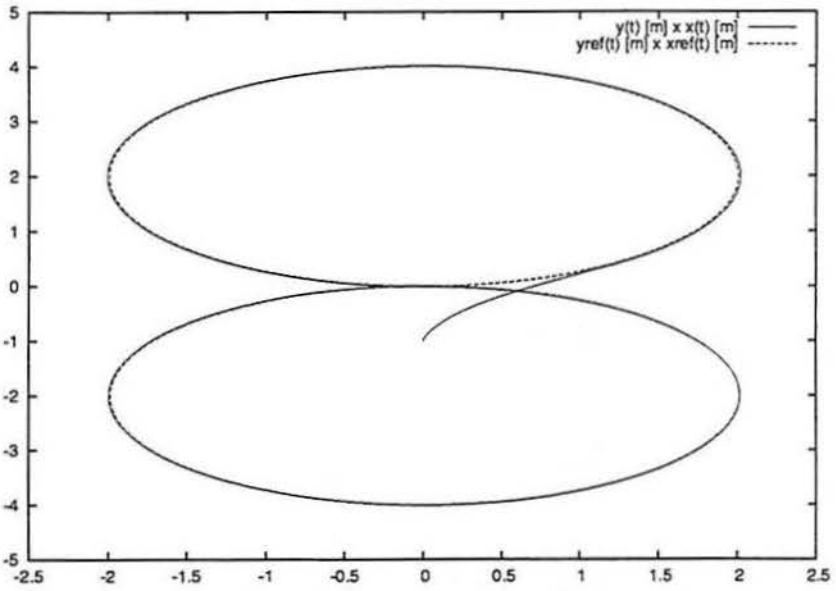

Figura 19: Trajetórias real e de referência para NMPC com função custo de (KÜHNE, 2005) seguindo uma trajetória em "oito".

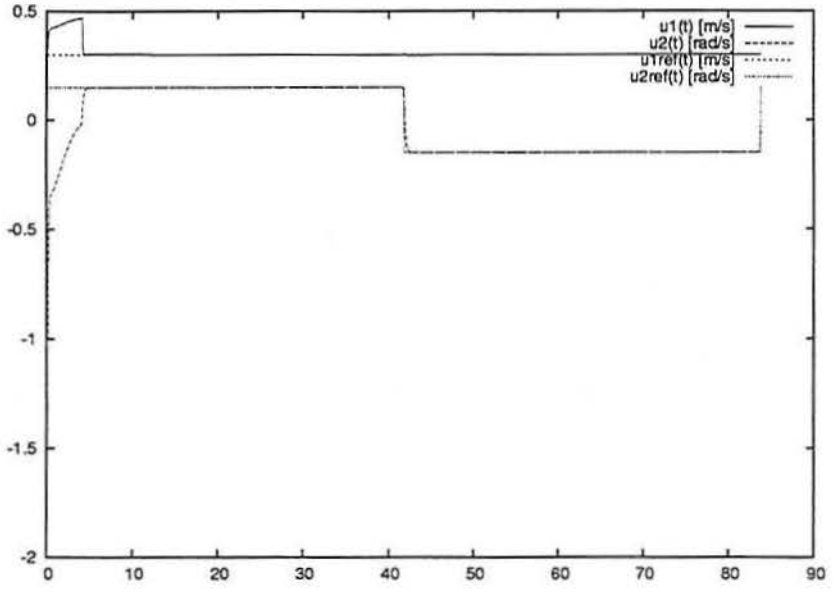

Figura 20: Sinal de controle para rastreamento de uma trajetória em "oito" com NMPC com função custo de (KÜHNE, 2005).

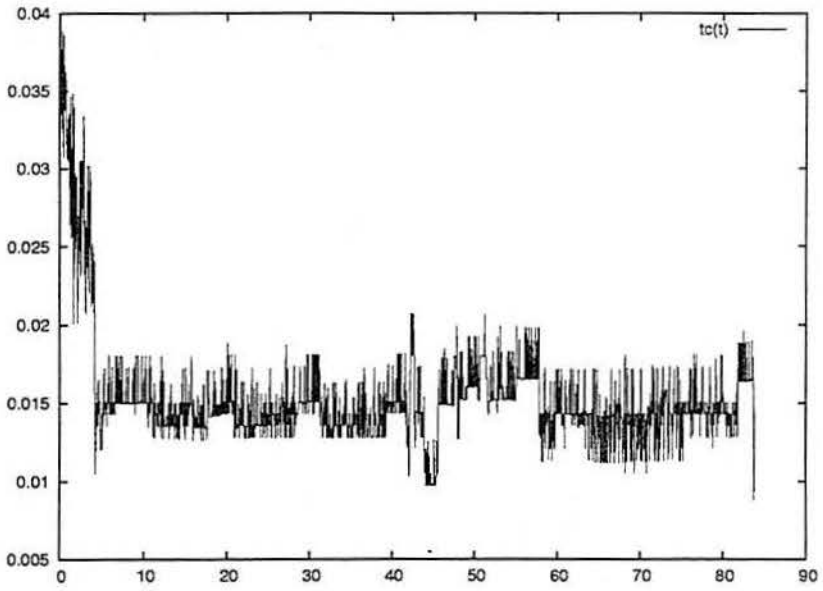

Figura 21: Tempo de computação do sinal de controle para rastreamento de uma trajetória em "oito" com NMPC com função custo de (KÜHNE, 2005).

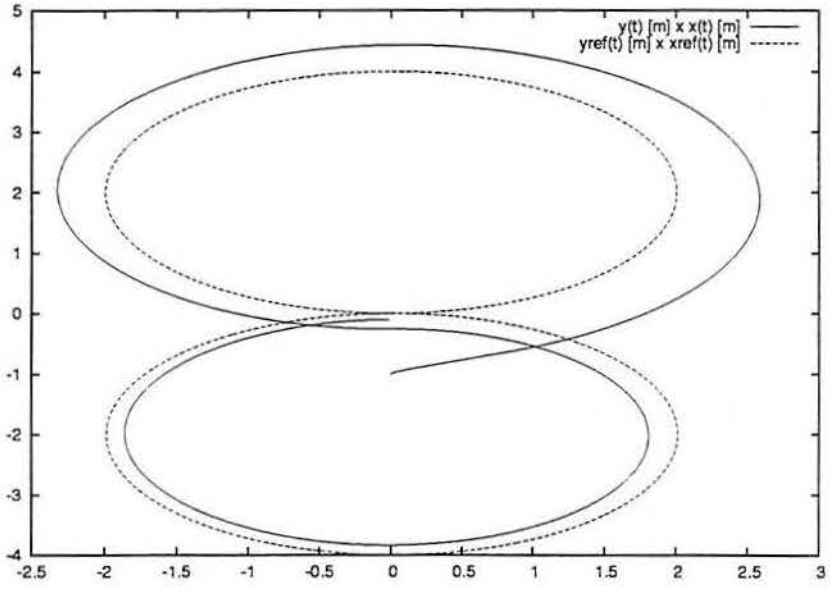

Figura 22: Trajetórias real e de referência para NMPC com função custo de (KÜHNE, 2005) seguindo uma trajetória em "oito" para $\mathbf{Q} = \text{diag}(1, 1, 0{,}5)$.

relação entre x, y e θ. Escolhendo-se $\mathbf{Q} = \text{diag}(1,1,0{,}5)$ obtém-se a trajetória no plano mostrada na Figura 22. O erro entre a referência e o estado real é visível ao longo de toda a trajetória.

Em (KÜHNE, 2005) é feita a comparação entre o desempenho do controlador NMPC para rastreamento de trajetória apresentado anteriormente, porém considerando restrições nas entradas u e v, e um controlador considerando a função custo utilizada em (ESSEN; NIJMEIJER, 2001), a saber:

$$\Phi(k) = \sum_{j=1}^{N-1} 2^{j-1}\tilde{\mathbf{x}}^{\mathrm{T}}(k+j|k)\mathbf{Q}\tilde{\mathbf{x}}(k+j|k) + \sum_{j=0}^{N-1} \tilde{\mathbf{u}}^{\mathrm{T}}(k+j|k)\mathbf{R}\tilde{\mathbf{u}}(k+j|k) + \tilde{\mathbf{x}}^{\mathrm{T}}(k+N|k)\Omega\tilde{\mathbf{x}}(k+N|k) \quad (38)$$

O controlador NMPC foi então alterado para considerar (38) como função-custo. Os parâmetros utilizados no controlador NMPC para rastreamento de trajetória foram $\mathbf{Q} = \text{diag}(4,4,0{,}5)$, $\mathbf{R} = \text{diag}(0{,}01, 0{,}01)$, $\Omega = 2^{N-1}\,\text{diag}(4,4,0{,}5)$ e $\varphi_{max_i} = -\varphi_{min_i} = 2\pi$ rad/s. Pode-se observar a trajetória do robô no plano cartesiano utilizando a função-custo de (ESSEN; NIJMEIJER, 2001) na Figura 23, o sinal de controle na Figura 24 e o tempo de computação necessário para o cálculo do mesmo na Figura 25.

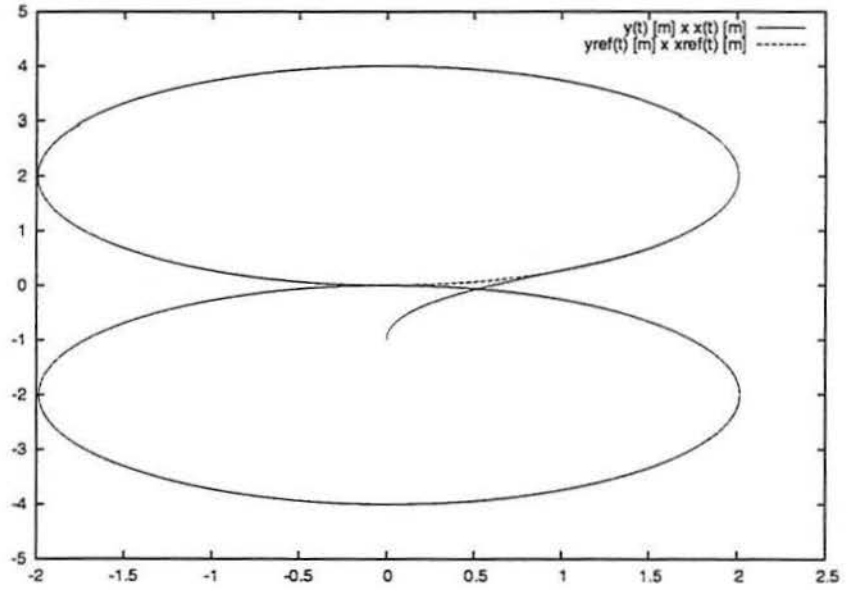

Figura 23: Trajetórias real e de referência para NMPC com função-custo de (ESSEN; NIJMEIJER, 2001) seguindo uma trajetória em "oito".

Foi observado que alterações nos parâmetros $\mathbf{Q}$ e $\mathbf{R}$ do controlador utilizando a função-custo de (ESSEN; NIJMEIJER, 2001) afetavam o desempenho do sistema em malha fechada de maneira muito menos pronunciada que no caso do controlador com função-custo de (KÜHNE, 2005). Isto se deve exatamente devido à função-custo considerada na formulação do controlador. Ao se penalizar pouco o erro dos estados x e y em relação a θ e à entrada tem-se pouca convergência para a referência uma vez que limitando-se a entrada e o erro em θ restringe-se o movimento do robô. A função-objetivo considerada por (ESSEN; NIJMEIJER, 2001) é menos suscetível a variações

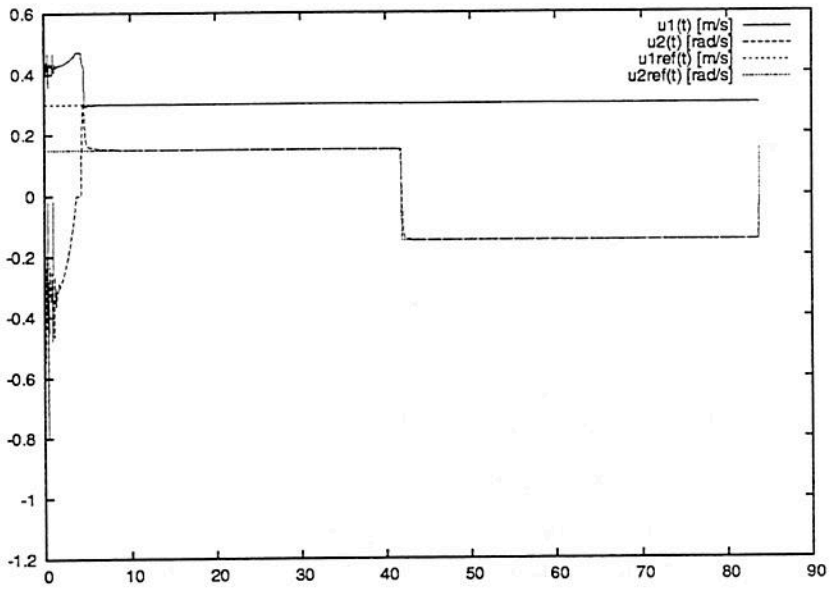

Figura 24: Sinal de controle para rastreamento de uma trajetória em "oito" com NMPC com função custo de (ESSEN; NIJMEIJER, 2001).

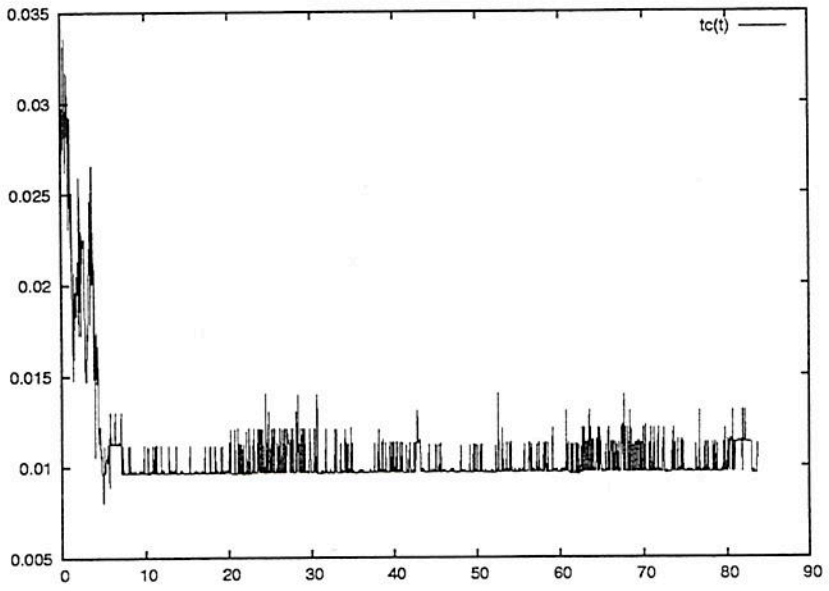

Figura 25: Tempo de computação do sinal de controle para rastreamento de uma trajetória em "oito" com NMPC com função custo de (ESSEN; NIJMEIJER, 2001).

paramétricas porque o erro nos estados futuros são penalizados sobremaneira em relação ao erro de estado para as primeiras predições. A importância da função-custo considerada no MPC é mais evidente e melhor explicada considerando-se o problema de estabilização em um ponto, notadamente mais complexo que o problema de rastreamento de trajetória para sistemas não-holonômicos e sem deriva. Isto será tratado na Seção 5.4.

Observa-se nas Figuras 21 e 25 que os controladores implementados foram capazes de resolver o problema de minimização em tempo menor que o período de amostragem, que é o *deadline* para a aplicação do sinal de controle. Em particular, o tempo necessário para a resolução do problema de minimização foi inferior ao se utilizar a função-custo de (KÜHNE, 2005), o que já havia sido observado em (KÜHNE, 2005).

5.4 Estabilização em um Ponto

5.4.1 Formulação do Problema

A formulação do problema de estabilização em um ponto para um robô móvel com acionamento diferencial é feita com base na seção 3.3. É considerado que o robô deve se mover em um plano horizontal com o objetivo de atingir um ponto qualquer no plano. Sem perda de generalidade, pode-se considerar que este ponto equivale a origem do espaço de estados, ou seja, $\mathbf{x} = [0 \;\; 0 \;\; 0]^{\mathrm{T}}$. Considera-se o robô móvel descrito pelo modelo cinemático de postura dado por (33). Para a implementação do controlador digital foi considerada uma discretização de (33) dada por (34). O controlador digital foi testado em um sistema simulado. Com o objetivo de se obter uma melhor aproximação do comportamento do robô foi utilizada uma aproximação através do método Runge-Kutta de 4ª ordem.

A função-custo para a minimização é dada por (9), com $N_1 = 0$:

$$\Phi(k) = \sum_{j=1}^{N} \mathbf{x}^{\mathrm{T}}(k+j|k)\mathbf{Q}\mathbf{x}(k+j|k) + \mathbf{u}^{\mathrm{T}}(k+j-1|k)\mathbf{R}\mathbf{u}(k+j-1|k), \qquad (39)$$

Então tem-se que (39) deve ser minimizada a cada instante de amostragem. Pode-se utilizar uma biblioteca de minimização para tal, existindo duas maneiras de se fazer isto quanto à função-custo. Uma é considerar a função (39) exatamente como descrita, com os estados e entradas como variáveis de decisão, e considerar o comportamento do sistema como uma restrição. Este método tem a vantagem de facilitar a descrição de restrições nos estados, pois para o algoritmo o estado do sistema está descrito diretamente na variável de decisão. A outra maneira é considerar apenas as entradas como variáveis de decisão, incluindo o comportamento de $\mathbf{x}(k+j|k)$ na função-custo a partir do estado atual $\mathbf{x}(k|k)$ e das entradas $\mathbf{u}(k+j-1|k)$. Assim não é necessário incluir restrições que não estejam relacionas as restrições reais relativas a entradas e estados. Entretanto, a descrição de restrições nos estados implica recalcular os mesmos, aumentando a carga computacional caso sejam consideradas restrições deste tipo. Os controladores implementados utilizaram a primeira formulação, com o estado contido na variável de decisão conforme (12). Nota-se que na realidade a variável de decisão é apenas a entrada: uma vez selecionada uma seqüência de entrada e como o estado atual não varia a predição dos estados futuros está absolutamente determinada.

A formulação do problema relativa à descrição dos estados como restrições consiste em encontrar uma seqüência de entrada ótima $\mathbf{u}^\star$ e uma seqüência de estados ótimos $\mathbf{x}^\star$ que minimizam

$$\mathbf{u}^\star, \mathbf{x}^\star = \arg\min_{\mathbf{u},\mathbf{x}} \Phi(k) \tag{40}$$

sujeito a

$$\begin{aligned}
\mathbf{x}(k|k) &= x(k) \\
\mathbf{x}(k+j|k) &= f_d(\mathbf{x}(k+j-1|k), \mathbf{u}(k+j-1|k)), \quad j \in [1, N] \\
g_x(\mathbf{x}(k+j|k)) &\leq \mathbf{a}, \quad j \in [1, N] \\
g_u(\mathbf{u}(k+j|k)) &\leq \mathbf{b}, \quad j \in [0, N-1].
\end{aligned}$$

Conforme visto na Seção 4.1, restrições relativas às velocidades das rodas devem ser consideradas. Pode-se obter uma expressão para as restrições nas entradas considerando-se o problema de estabilização em um ponto como um caso particular de rastreamento de trajetória. Fazendo-se $\mathbf{x}_r(k) = \mathbf{0}$ e $\mathbf{u}_r(k) = \mathbf{0}$ em (31) tem-se

$$\dot{\varphi}_{min}(k) \leq \mathbf{P}^{-1}\mathbf{u}(k) \leq \dot{\varphi}_{max}(k), \tag{41}$$

que pode ser reescrita na forma de (11) definindo-se $g_u(\mathbf{u}(k+j|k))$ e $\mathbf{b}$ como:

$$g_u(\mathbf{u}(k+j|k)) \triangleq \begin{bmatrix} \mathbf{P}^{-1} \\ -\mathbf{P}^{-1} \end{bmatrix} \quad \mathbf{b} \triangleq \begin{bmatrix} \dot{\varphi}_{max}(k) \\ -\dot{\varphi}_{min}(k) \end{bmatrix}$$

5.4.2 Implementação do NMPC: estabilização em um ponto

A formulação do problema de estabilização em um ponto, conforme Seção 5.4.1, leva em consideração uma função custo com base em coordenadas cartesianas. Conforme (KÜHNE, 2005), o controlador empregando uma função custo deste tipo não é capaz de tornar a origem do sistema assintoticamente estável. Entretanto, naquele trabalho foram consideradas restrições nas entradas em v e w separadamente. Conforme visto na Seção 4.1, deve-se considerar os limites em φ_1 e φ_2. O controlador foi então implementado com restrições nas entradas de acordo com a formulação apresentada na Seção 5.4.1:

$$\begin{bmatrix} \mathbf{P}^{-1} \\ -\mathbf{P}^{-1} \end{bmatrix} \mathbf{u}(k) \geq \begin{bmatrix} \dot{\varphi}_{max}(k) \\ -\dot{\varphi}_{min}(k) \end{bmatrix}, \quad k \in [0, N-1] \tag{42}$$

O controlador NMPC então foi implementado com a seguinte formulação:

$$\mathbf{u}^\star, \mathbf{x}^\star = \arg\min_{\mathbf{u},\mathbf{x}} \Phi(k)$$

sujeito a

$$\begin{aligned}
\mathbf{x}(k|k) &= x(k) \\
\mathbf{x}(k+j|k) &= f_d(\mathbf{x}(k+j-1|k), \mathbf{u}(k+j-1|k)), \quad j \in [1, N] \\
\begin{bmatrix} \mathbf{P}^{-1} \\ -\mathbf{P}^{-1} \end{bmatrix} \mathbf{u}(k) &\geq \begin{bmatrix} \dot{\varphi}_{max}(k) \\ -\dot{\varphi}_{min}(k) \end{bmatrix}, \quad j \in [0, N-1].
\end{aligned}$$

Assim como para o controlador LMPC, o código foi escrito em C++, desta vez tendo sido utilizada a biblioteca donlp2 (vide Seção 5.2) para solução do problema

de minimização. O programa foi executado com o sistema RTAI (vide Seção 4.2.2), utilizando-se o módulo LXRT. Seguem resultados do controlador executado em tempo real, sendo o sinal de controle aplicado a um simulador que considera uma discretização do modelo do robô expresso em (1) através de Runge-Kutta de 4ª ordem. Foram utilizados os seguintes parâmetros no controlador: $\mathbf{Q} = \text{diag}(1, 1, 0{,}5)$, $\mathbf{R} = \text{diag}(0{,}05, 0{,}05)$, $\varphi_{max_i} = -\varphi_{min_i} = 2\pi$ rad/s e $N = 5$.

Pode-se observar nas Figuras 26, 27 e 28 o comportamento do sistema utilizando o controlador NMPC em coordenadas polares. A Figura 26 mostra a trajetória do robô no plano. O valor do estado no fim da trajetória é $[94{,}628 \times 10^{-6} \quad -1.770 \quad 109{,}32 \times 10^{-6}]$, ou seja, o estado não tende para a origem. Pode-se observar as velocidades linear e angular do robô na Figura 27, assim como as velocidades das rodas na Figura 28. Verifica-se que a entrada tende a zero, o que significa que efetivamente o robô pára.

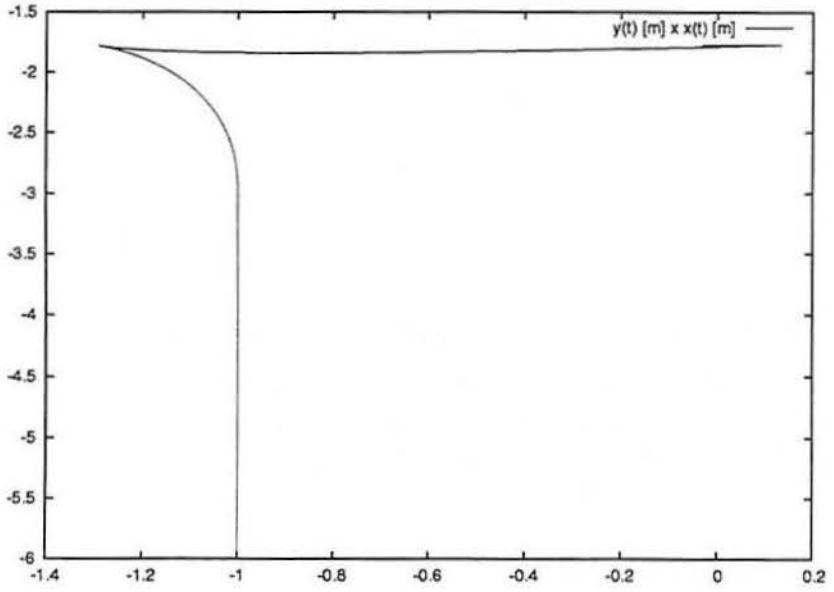

Figura 26: Trajetória do robô móvel no plano XY para NMPC com coordenadas cartesianas e $\mathbf{x}(0) = [-1 \quad -6 \quad \pi/2]$.

O sistema realimentado utilizando-se MPC em coordenadas cartesianas não é assintoticamente estável, o que já havia sido observado em (KÜHNE, 2005) utilizando-se um controlador simulado e restrições nas velocidades u e v. O desempenho do sistema está relacionado a minimização da função custo utilizada, expressa por (19). Considere-se uma condição inicial da forma $\mathbf{x}(0) = [0 \quad y(0) \quad 0]^{\mathrm{T}}$, com $y(0)$ relativamente pequeno. Fica claro que, devido às restrições não-holonômicas, o robô precisa fazer curvas para atingir a origem com a orientação especificada, i.e. $\theta(k) = 0$. A Figura 29 mostra uma trajetória possível que levaria um robô móvel do eixo OY até a origem com $\theta(k) = 0$ em torno dela. Pode-se verificar que o robô precisa ou se afastar do eixo OY, o que faz com que x aumente, e também fazer curvas tanto para um lado quanto para o outro, o que significa que o valor absoluto de θ também deve ser diferente de zero. Entretanto, todas estas ações são penalizadas pela função-custo, e portanto o robô tende a ficar parado. A função-custo empregada na formulação do controlador NMPC tem grande importância no desempenho do sis-

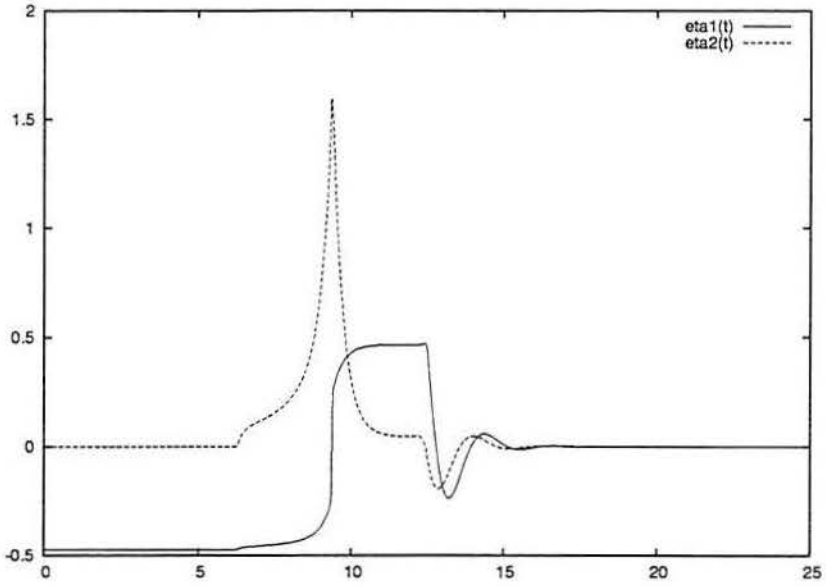

Figura 27: Sinal de controle para NMPC com coordenadas cartesianas e $\mathbf{x}(0) = [-1 \quad -6 \quad \pi/2]$.

tema realimentado, especialmente quando há restrições não-holonômicas no modelo do sistema.

Em (KÜHNE; LAGES; GOMES DA SILVA JR., 2005) e, mais detalhadamente, em (KÜHNE, 2005) propõe-se um controlador que utiliza um conjunto de coordenadas polares na função-custo (vide Seção 2.3.1). O conjunto de coordenadas $\{x \;\; y \;\; \theta\}$ é transformado para $\{e \;\; \psi \;\; \alpha\}$, com:

$$\begin{aligned} e &\triangleq \sqrt{x+y} \\ \psi &\triangleq \text{atan2}(y,x) \\ \alpha &\triangleq \theta - \psi. \end{aligned} \tag{43}$$

Pode-se modificar (19) para

$$\Phi(k) = \sum_{j=1}^{N} \mathbf{x}_p^{\mathrm{T}}(k+j|k)\mathbf{Q}\mathbf{x}_p(k+j|k) + \mathbf{u}^{\mathrm{T}}(k+j-1|k)\mathbf{R}\mathbf{u}(k+j-1|k), \tag{44}$$

com $\mathbf{x}_p \triangleq [e \;\; \psi \;\; \alpha]^{\mathrm{T}}$.

A transformação de coordenadas é ilustrada pela Figura 7. Ao contrário do caso do controlador em coordenadas cartesianas, obtém-se um sistema realimentado sem erro em regime permanente. Não obstante, o robô somente se aproxima da origem pelo semiplano direito, ou seja, com $x > 0$ e, conseqüentemente, $v < 0$. Pode-se verificar isto melhor na Figura 30. O robô está inicialmente a uma distância de 3 m da origem com $\theta(0) = 0$ para todas as trajetórias. Observa-se que o robô entra no semiplano direito antes de se aproximar da origem para qualquer condição inicial. É desejável que o robô seja também capaz de atingir o ponto de equilíbrio pelo semiplano esquerdo, com velocidade linear positiva.

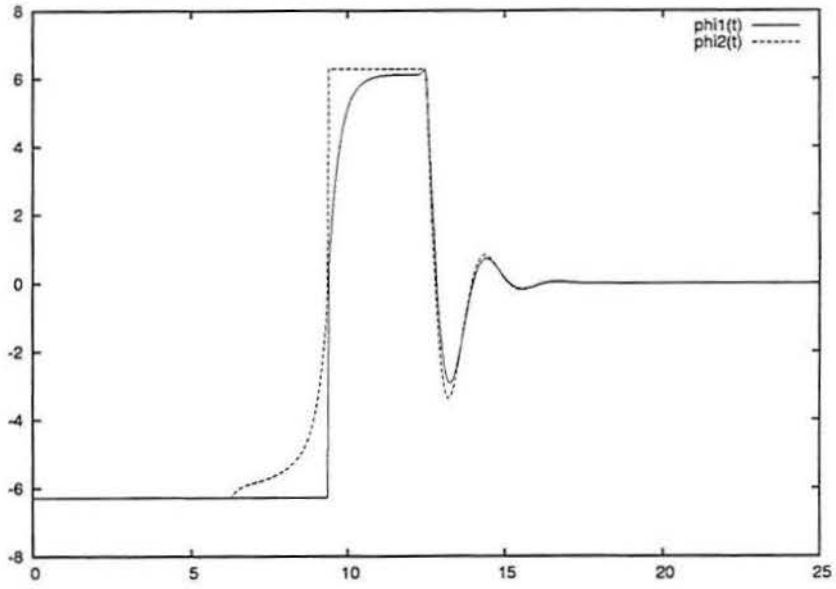

Figura 28: Velocidades das rodas para NMPC com coordenadas cartesianas e $\mathbf{x}(0) = [-1 \quad -6 \quad \pi/2]$.

O comportamento do sistema realimentado utilizando NMPC e coordenadas polares é também resultado da minimização da função-custo utilizada na sua formulação. Ao minimizar ψ e α faz-se com que o robô tenda a adotar uma posição sobre o lado positivo do eixo OX, com $\theta = 0$. Apesar de se aproximar do ponto $[0 \quad 0]^{\mathrm{T}}$ mais rapidamente que outros controladores factíveis quando localizado inicialmente no semiplano direito, o robô percorrerá uma trajetória mais longa que a necessária se posicionado no outro semiplano no início da trajetória. Devido a este problema outro controlador será proposto.

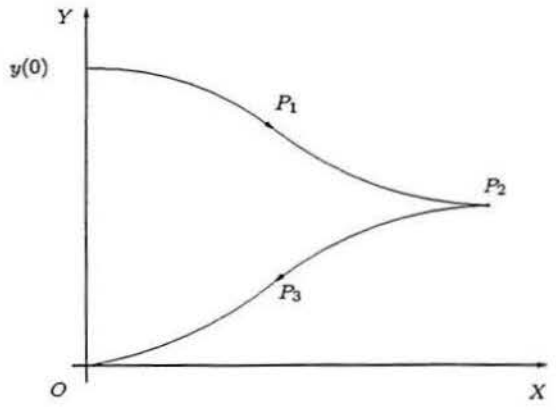

Figura 29: Possível trajetória para um robô móvel atingir a origem com condições iniciais do tipo $[0 \quad y(0) \quad 0]^{\mathrm{T}}$.

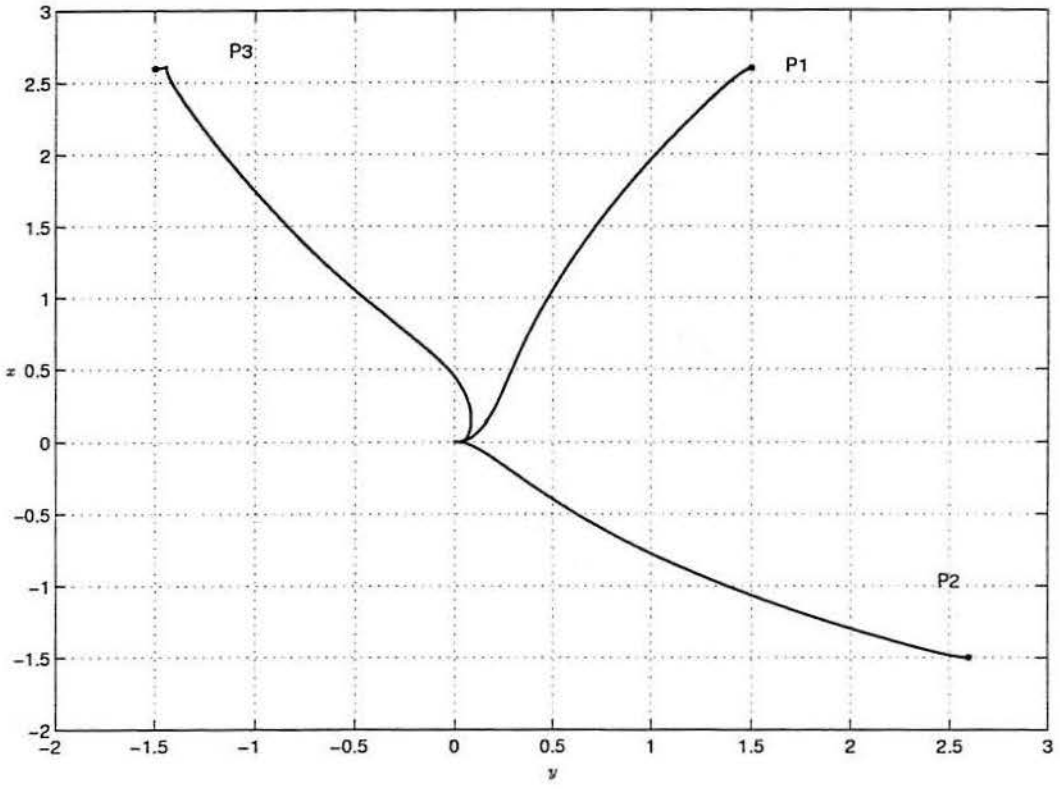

Figura 30: Trajetórias para diferentes condições iniciais, NMPC em coordenadas polares de (KÜHNE, 2005).

A transformação de coordenadas alternativa $\{e \ \psi \ \alpha\}$, definida como

$$
\begin{aligned}
e &\triangleq \sqrt{x+y} \\
\psi &\triangleq \begin{cases} \text{atan2}(y,x) & \text{se } x(0) \geq 0, \\ \text{atan2}(-y,-x) & \text{se } x(0) < 0, \end{cases} \\
\alpha &\triangleq \theta - \psi,
\end{aligned}
$$

é exatamente igual a expressa por (43) quando o robô está no semiplano direito no tempo inicial. Por outro lado, se o robô está no semiplano esquerdo uma transformação similar é utilizada, porém simétrica com relação à origem. Pontos em torno do lado negativo do eixo OX têm valores baixos de ψ enquanto que o máximo valor absoluto desta variável estará relacionado a pontos ao redor do lado negativo do mesmo eixo. A performance difere do sistema realimentado utilizando esta transformação de coordenadas difere da do sistema utilizando o NMPC em coordenadas polares conforme (KÜHNE; LAGES; GOMES DA SILVA JR., 2005) para $x(0) < 0$. Resultados obtidos com o novo controlador podem ser verificados na Figura 32. Pode-se observar em maior detalhe a região próxima à origem na Figura 33.

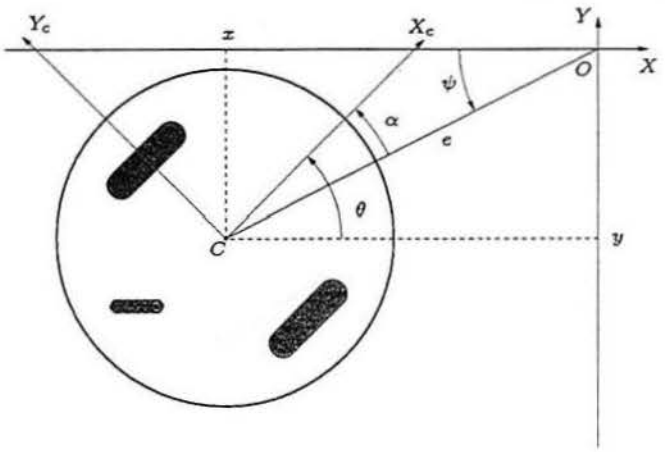

Figura 31: Conjunto de coordenadas alternativo para um robô móvel com acionamento diferencial.

Pode ser visto claramente nas Figuras 32 e 33 que o estado converge assintoticamente para a origem. A evolução dos estados em relação ao tempo é mostrada nas Figuras 34, 35 e 36. Na Figura 37 pode ser visto que as entradas satisfazem as restrições. A entrada que é efetivamente aplicada no sistema é mostrada na Figura 38.

A Figura 39 mostra o tempo necessário para se calcular a lei de controle como função do tempo para um processador AMD Athlon 64 4000+. Pode ser visto que a lei de controle pode ser calculada em tempo menor que um período de amostragem. Ao contrário do MPC linearizado (LAGES; ALVES, 2006), as variações no tempo de computação se dão devido ao processo de otimização, tendo o escalonamento pouca influência. A solução do problema de otimização pode tomar mais ou menos tempo dependendo da condição inicial, sendo que há um limite superior que foi ajustado limitando-se o número de vezes que a função-custo pode ser calculada pelo algoritmo.

As Tabelas 2 e 3 mostram resultados relacionando o erro e a carga computacional em função do horizonte de predição, similarmente ao que foi feito para LMPC

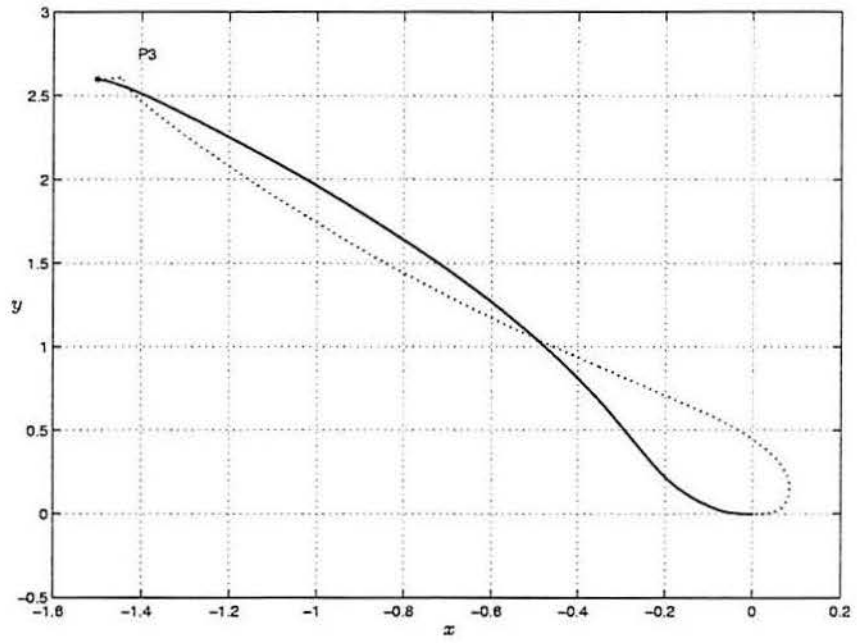

Figura 32: Comparação entre NMPC em coordenadas polares de (KÜHNE, 2005) (linha tracejada) e NMPC em coordenadas polares alteradas (linha contínua).

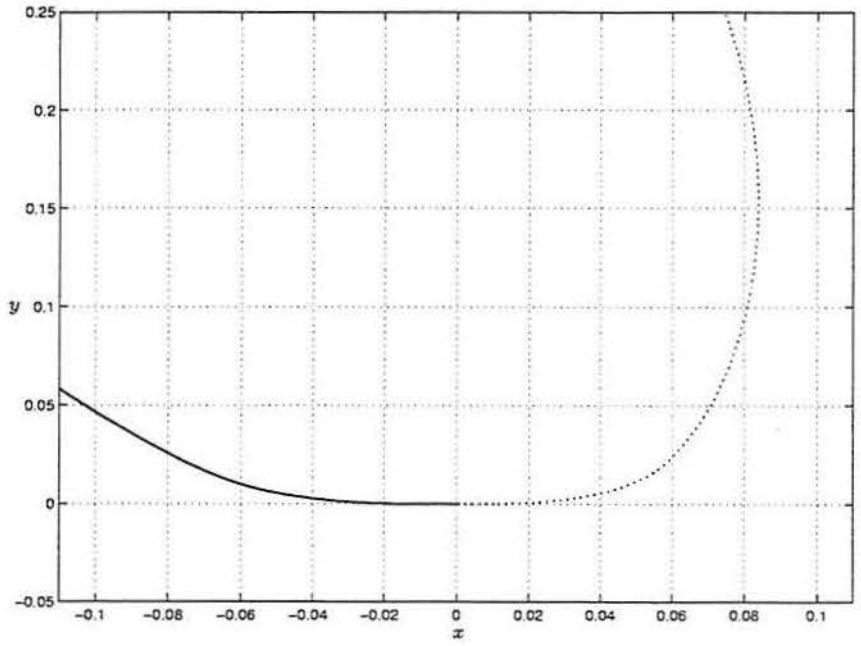

Figura 33: Detalhe da Figura 32 em torno da origem. Linha tracejada: NMPC em coordenadas polares de (KÜHNE, 2005). Linha contínua: NMPC em coordenadas polares alteradas.

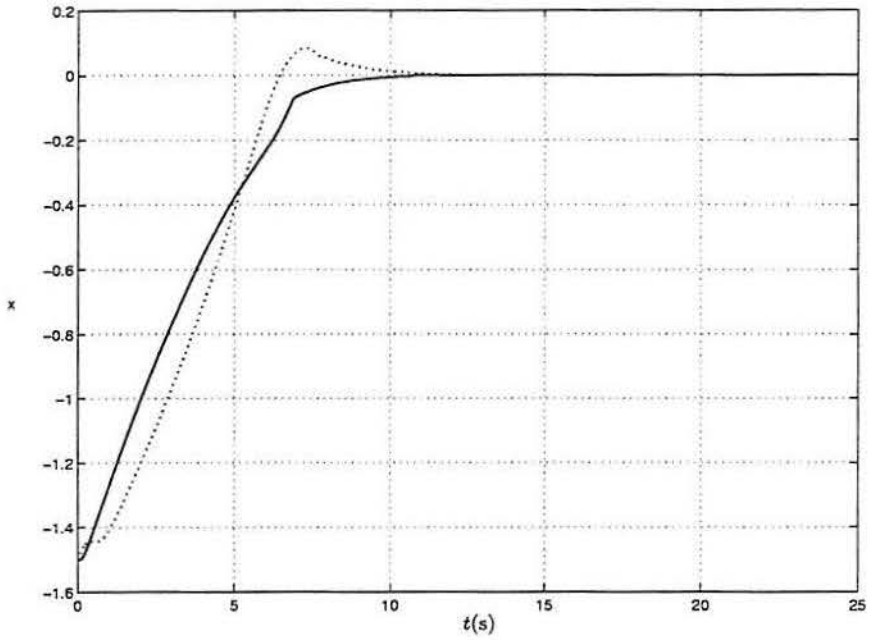

Figura 34: x em função do tempo. Linha tracejada: NMPC em coordenadas polares de (KÜHNE, 2005). Linha contínua: NMPC em coordenadas polares alteradas.

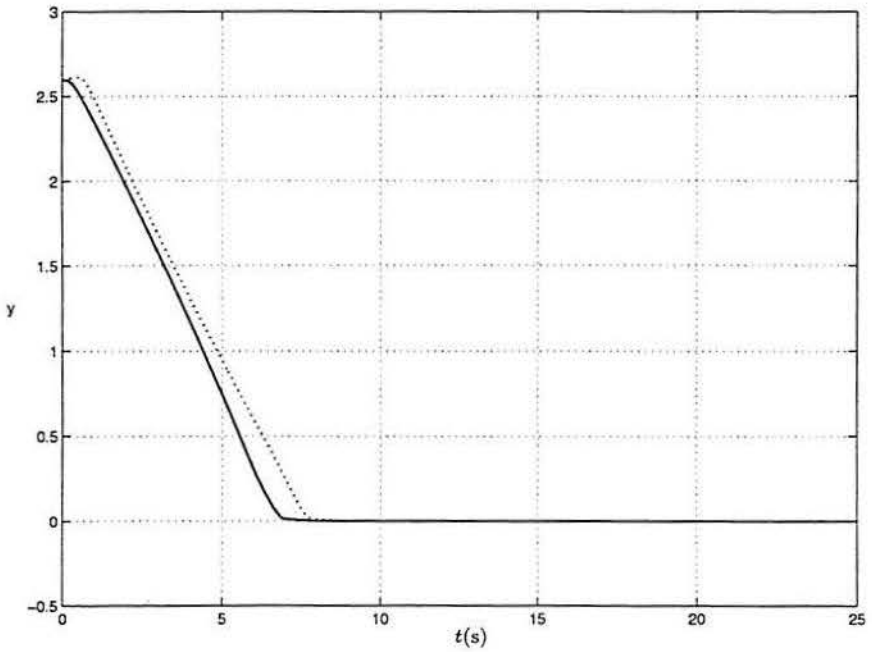

Figura 35: y em função do tempo. Linha tracejada: NMPC em coordenadas polares de (KÜHNE, 2005). Linha contínua: NMPC em coordenadas polares alteradas.

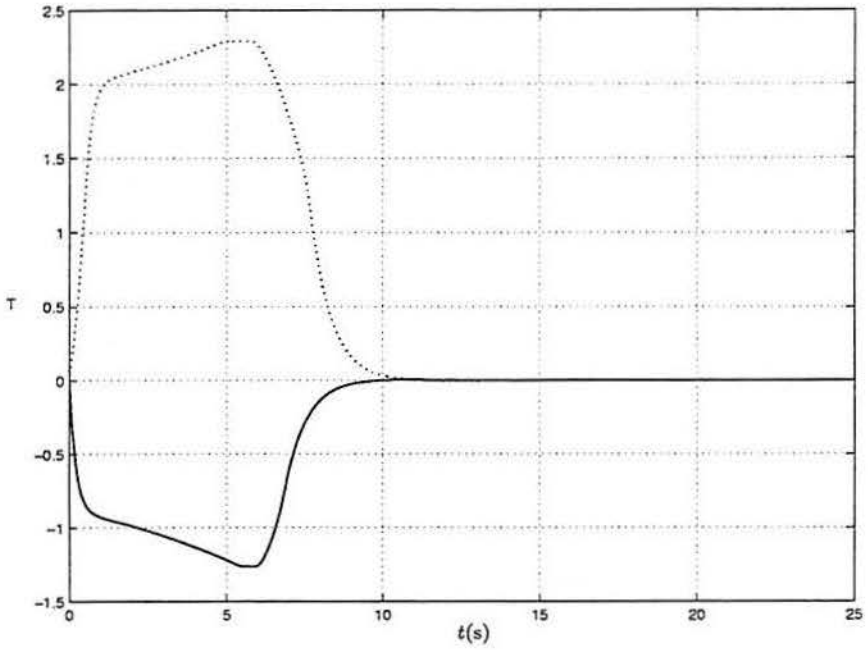

Figura 36: θ em função do tempo. Linha tracejada: NMPC em coordenadas polares de (KÜHNE, 2005). Linha contínua: NMPC em coordenadas polares alteradas.

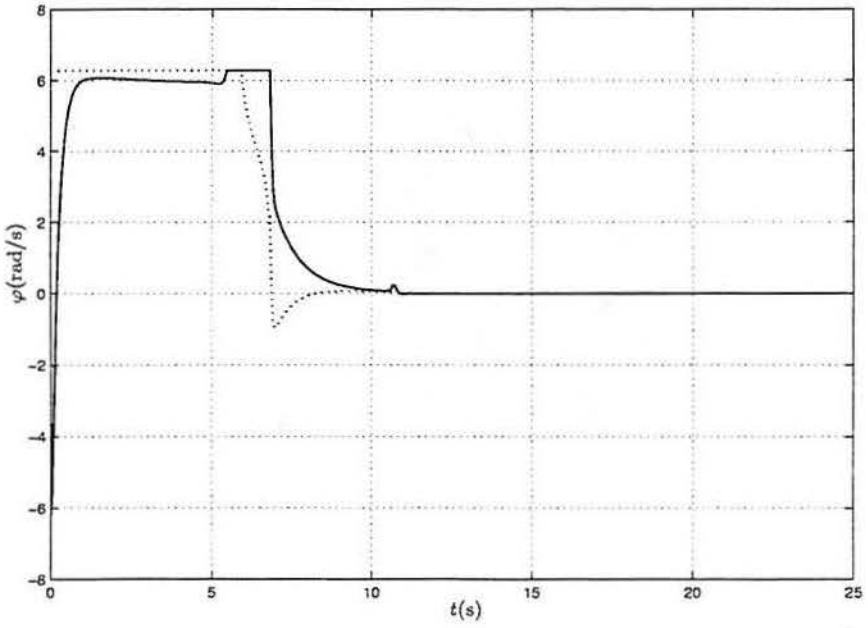

Figura 37: Velocidades das rodas para NMPC em coordenadas polares alteradas. Linha contínua: $\varphi_r(t)$. Linha tracejada: $\varphi_l(t)$.

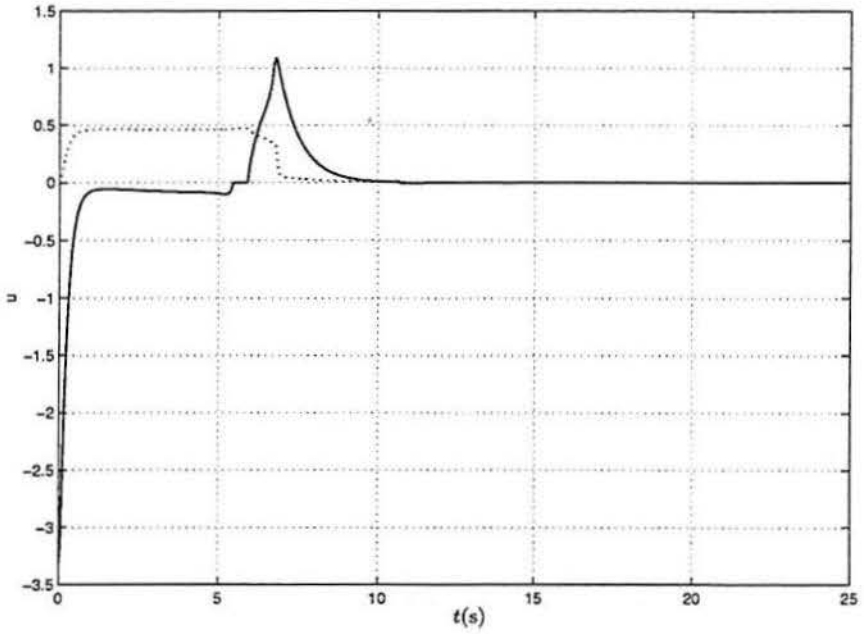

Figura 38: Entradas do sistema para NMPC em coordenadas polares alteradas. Linha contínua: $v(t)$. Linha tracejada: $w(t)$.

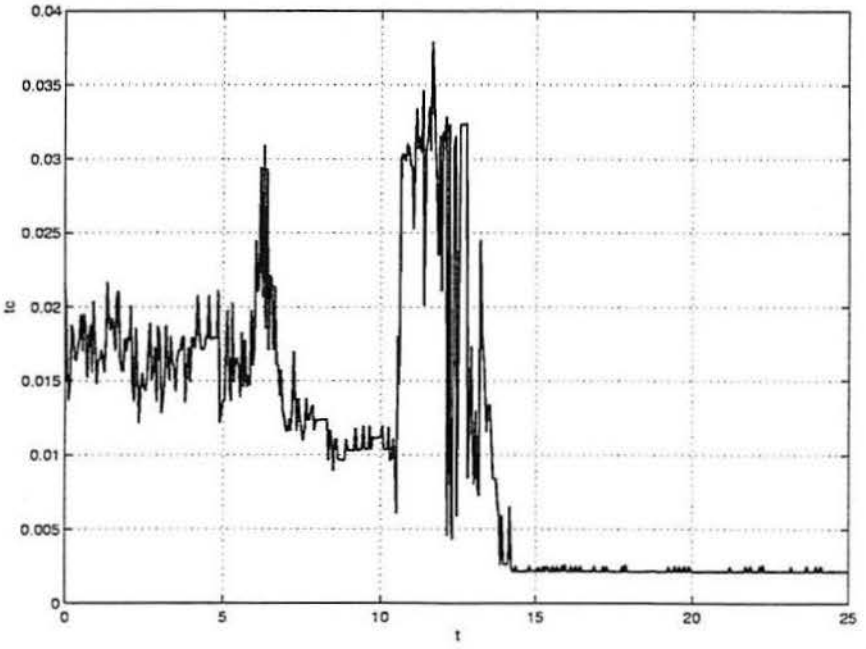

Figura 39: Tempo de computação do sinal de controle para NMPC em coordenadas polares alteradas.

na Tabela 1, na Seção 4.2.3. Mais especificamente, constam: a) numero de operações em ponto flutuante necessários para calcular a lei de controle a cada passo de amostragem, conforme Matlab, b) tempo estimado de computação, c) tempo real de computação e d) erro médio quadrático. A Tabela 2 mostra os valores médios relacionados ao tempos de computação enquanto que a Tabelas 3 mostra os valores máximos observados. Os tempos estimados de computação foram obtidos considerando uma performance de 895,8203 Mflops para um processador AMD Athlon 64 4000+ índice este obtido através do programa Flops (ABURTO, 1992).

Tabela 2: Horizonte de predição × tempo médio de computação para NMPC.

N	flo	Tempo de computação (s)		ε
		Estimado	Real	
2	57.824	$64{,}548 \times 10^{-6}$	$5{,}1 \times 10^{-3}$	1,15827
3	199.065	$222{,}22 \times 10^{-6}$	$2{,}2 \times 10^{-3}$	0,905772
4	499.108	$577{,}15 \times 10^{-6}$	$7{,}8 \times 10^{-3}$	0,904453
5	1.143.769	$1{,}2768 \times 10^{-3}$	$9{,}3 \times 10^{-3}$	0,861873
10	$14{,}947 \times 10^{6}$	$16{,}685 \times 10^{-3}$	$36{,}3 \times 10^{-3}$	0,857001
15	—	—	$86{,}4 \times 10^{-3}$	0,88336

Tabela 3: Horizonte de predição × tempo máximo de computação para NMPC.

N	flo	Tempo de computação (s)		ε
		Estimado	Real	
2	1.450.781	$1{,}620 \times 10^{-3}$	$9{,}1 \times 10^{-3}$	1,15827
3	4.227.858	$4{,}719 \times 10^{-3}$	$10{,}3 \times 10^{-3}$	0,905772
4	7.539.557	$8{,}416 \times 10^{-3}$	$31{,}7 \times 10^{-3}$	0,904453
5	18.080.403	$20{,}183 \times 10^{-3}$	$41{,}0 \times 10^{-3}$	0,861873
10	143.361.338	$160{,}033 \times 10^{-3}$	$164{,}4 \times 10^{-3}$	0,857001
15	—	—	$336{,}1 \times 10^{-3}$	0,88336

Conforme mencionado anteriormente, foi ajustado o número máximo de iterações permitidas pela biblioteca donlp2 com o objetivo de limitar o tempo de computação. Foi verificado que um limite máximo de 40 iterações tem pouca influência sobre a resposta do sistema realimentado para $N = 5$. A alteração foi importante, porém, uma vez que o cálculo da lei de controle muitas vezes demorava tempo superior ao período de amostragem, que é o maior *deadline* possível para a aplicação do sinal de controle.

Os dados contidos na Tabela 3 mostram que um computador recente é capaz de rodar um controlador NMPC para um robô móvel. De acordo com a Seção 4.2.3, períodos de amostragem da ordem de 50 ms são adequados, e o algoritmo MPC proposto pode ser calculado para $N = 5$ em cerca de 40 ms. Entretanto, o cálculo toma 80% do período de amostragem, o que é significativo. Ainda assim, não foram feitas tentativas de otimizar os cálculos relacionados a minimização, como ajustar os parâmetros do algoritmo ou eliminar possíveis cálculos desnecessários presentes nas funções relacionadas às restrições e ao custo. Além disso, um processador de 64 bits foi utilizado, porém operando no modo 32 bits.

Similarmente ao que acontece com MPC Linearizado (vide Seção 4.2.3), uma estimativa baseada no número de operações em ponto flutuante mostra-se pouco confiável. Os tempos de computação foram muito mais longos que o estimado, como pode ser verificado nas Tabelas 2 e 3. Por outro lado, o tempo de computação apresenta grandes variações dependendo do valor do estado no instante de amostragem em questão, o que pode ser melhor observado comparando-se as Figuras 18 e 39. Quando o robô se aproxima da origem há inicialmente um aumento do tempo de computação para o NMPC. De cerca de 20 ms o tempo chega até cerca de 40 ms, sendo então limitado pelo número de iterações. Quando o sistema entra em regime permanente o tempo de computação cai para valores da ordem de 2,1 ms. Para o LMPC um processador Athlon 3200+ resolve o problema de minimização em média em menos de 1,0 ms, sendo que há poucas variações no tempo de computação.

5.5 Biblioteca MPC desenvolvida

O código dos controladores implementados em tempo real neste trabalho foi escrito em C++, sendo os diversos tipos de controladores organizados em classes. Os arquivos foram compilados com o GCC 3.3.3 instalado em uma máquina (GNU/)Linux 2.6.16. O uso da biblioteca se dá através das classes `DIFFKIN_LMPC`, `DIFFKIN_NLMPC`, `DIFFKIN_LMPC_TRK` e `DIFFKIN_ESSEN_TRK`. O diagrama de classes da biblioteca pode ser visto na Figura 40.

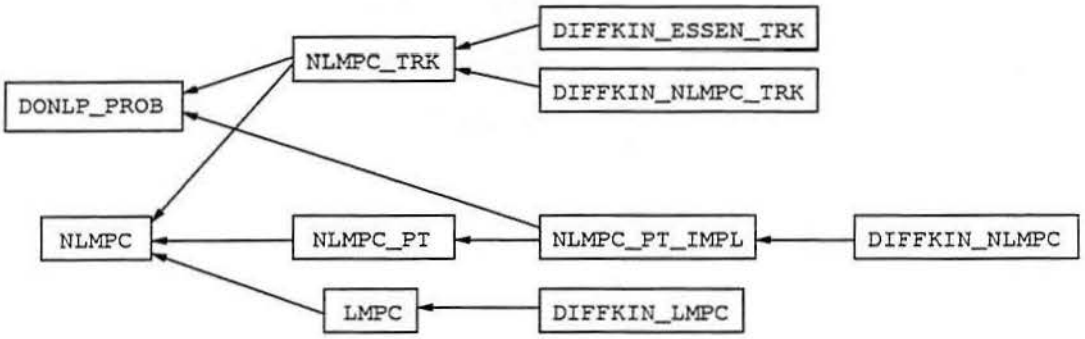

Figura 40: Diagrama de classes dos controladores implementados.

A classe `NLMPC` contém campos de dados necessários para se armazenar parâmetros comuns a diversos tipos de controladores MPC, como as matrizes de ponderação de erro nos estados e entradas $\mathbf{Q}$ e $\mathbf{R}$, respectivamente. As classes `NLMPC_PT` e `NLMPC_TRK` contém dados e funções-membro específicas para os problemas de estabilização em um ponto e rastreamento de trajetória, nesta ordem, quando se considera o modelo não-linear do sistema. No caso do LMPC é definida a classe de mesmo nome, a qual é extendida por `DIFFKIN_LMPC` representando o controlador MPC para um robô móvel de acionamento diferencial. Outras classes considerando o mesmo tipo de robô móvel, porém para o modelo não linear do sistema, são `DIFFKIN_NLMPC_TRK`, `DIFFKIN_ESSEN_TRK` e `DIFFKIN_NLMPC`, relativas aos problemas de rastreamento de trajetória com função-custo de (KÜHNE, 2005), rastreamento de trajetória com função-custo de (ESSEN; NIJMEIJER, 2001) e estabilização em um ponto, respectivamente. Especificamente, as classes citadas implementam as seguintes funcionalidades:

`NLMPC` Classe virtual que define os dados a serem utilizados em controladores MPC gerais: as matrizes de ponderação $\mathbf{Q}$ e $\mathbf{R}$, as dimensões do estado e da entrada, e as referências a serem seguidas. É definida a função-membro `out` como interface para obtenção do sinal de controle.

`LMPC` Classe virtual que contém dados e funções-membro a serem utilizados por um controlador MPC que utilize um modelo linear (possivelmente variante no tempo) para o problema de rastreamento de trajetória, conforme formulação apresentada na Seção 3.4 Ao se extendê-la para se obter um controlador para um problema específico é necessário o conhecimento das matrizes $\mathbf{A}(k)$ e $\mathbf{B}(k)$.

`DIFFKIN_LMPC` Classe derivada de `LMPC` implementando o controlador LMPC para um robô de acionamento diferencial. Considera as matrizes $\mathbf{A}(k)$ e $\mathbf{B}(k)$ conforme modelo linearizado do erro em relação à trajetória expresso por (26). Utiliza a biblioteca OOQP (GERTZ; WRIGHT, 2003) para resolução do problema quadrático e consideração de restrições.

`NLMPC_PT` Classe derivada de `NLMPC` que define a interface específica para a obtenção do sinal de controle no caso do problema de estabilização em um ponto, quando referências de estado e entrada são desnecessárias.

`NLMPC_TRK` Classe derivada de `NLMPC` que utiliza a biblioteca donlp2 (SPELLUCCI, 1998a) para solução do problema de programação não-linear, definindo portanto a interface para implementação de controladores MPC para o problema de rastreamento de trajetória que considerem modelos não-lineares.

`NLMPC_PT_IMPL` Classe derivada de `NLMPC_PT` que utiliza a biblioteca donlp2 (SPELLUCCI, 1998a) para solução do problema de programação não-linear, definindo portanto a interface para implementação de controladores MPC para o problema de estabilização em um ponto que considerem modelos não-lineares.

`DIFFKIN_NLMPC_PT` Classe derivada de `NLMPC_PT_IMPL`, implementa controladores MPC para robôs móveis de acionamento diferencial considerando funções-custo com base em coordenadas cartesianas, coordenadas polares conforme (KÜHNE, 2005) ou coordenadas polares alteradas.

`DIFFKIN_NLMPC_TRK` Classe derivada de `NLMPC_TRK` que implementa um controlador MPC considerando uma função-custo quadrática em relação ao estado do sistema expresso em coordenadas cartesianas, com matriz $\mathbf{Q}$ constante em relação ao instante de predição. Define o modelo do sistema como restrição de igualdade e as velocidades das rodas como restrições de desigualdade.

`DIFFKIN_ESSEN_TRK` Classe derivada de `NLMPC_TRK` que implementa um controlador MPC considerando uma função-custo quadrática em relação ao estado do sistema expresso em coordenadas cartesianas, com matriz $\mathbf{Q}$ variável em relação ao instante de predição e considerando um custo terminal, conforme (ESSEN; NIJMEIJER, 2001). Define o modelo do sistema como restrição de igualdade e as velocidades das rodas como restrições de desigualdade.

A implementação da biblioteca MPC permite que os controladores sejam utilizados de maneira bastante direta em um programa escrito em linguagem C++. Segue, a

título de exemplo, um esquema passo-a-passo do uso da biblioteca para o problema de rastreamento de trajetória utilizando a função-custo de (ESSEN; NIJMEIJER, 2001):

1. cria-se um objeto da classe `DIFFKIN_ESSEN_TRK` informando os limites mínimo e máximo relativos às entradas, a matriz mudança de base relativa a restrição nas entradas, as matrizes de ponderação $\mathbf{Q}$ e $\mathbf{R}$, o período de amostragem utilizado pelo controlador e o horizonte de predição:

```
DIFFKIN_ESSEN_TRK nmpc(phimin, phimax, D, Q, R, ST, HORIZ);
```

2. uma vez tendo-se a medida do estado utiliza-se a função-membro `out` para resolver o problema de minimização, informando o estado atual e as referências de estado e entrada, retornando a função-membro diretamente a entrada de controle $\mathbf{u}^*(k)$ (neste caso na variável `eta`):

```
eta = nmpc.out(x, xref, uref);
```

3. Espera-se o novo instante de amostragem, obtém-se novamente uma medida do estado do sistema e utiliza-se novamente a função `out` conforme item 2;

4. repete-se o procedimento dos itens 2 e 3 até que seja desligado o controlador ou que a trajetória tenha chego ao final;

O uso da biblioteca é semelhante para outros tipos de controlador, bastando substituir `DIFFKIN_ESSEN_TRK` pela classe correspondente e, no caso do problema de estabilização em um ponto, eliminando os parâmetros relativos às referências de estado e entrada.

Finalmente, observa-se que todas as classes consideram a existência de restrições relativas às entradas. Controladores que não estejam sujeitos a restrições podem ser obtidos utilizando-se limites infinitos em `phimin` e `phimax`.

6 CONCLUSÃO

Foi apresentada neste trabalho a implementação em tempo real de controladores preditivos baseados em modelo para uma classe de robôs móveis não-holonômicos. Foram implementados controladores levando em consideração diferentes formulações de MPC, tanto para rastreamento de trajetória quanto para estabilização em um ponto. Foram ainda consideradas restrições nas entradas como sendo as velocidades das rodas, o que permite a obtenção de sinais de controle que podem efetivamente ser aplicados em um robô. Além disso, para o problema de estabilização em um ponto, foi desenvolvida uma nova formulação, a qual permite se obter um controlador com desempenho superior aos anteriores sendo ainda factível para a implementação em tempo real.

Em (KÜHNE, 2005) foram revisadas leis de controle clássicas para controle de robôs móveis não-holonômicos. Em especial, deu-se grande importância para as restrições a que pode estar submetido o robô, tanto nas entradas quanto nos estados. Em particular, foi percebido que tais leis de controle tipicamente ultrapassam os limites que um robô pode suportar. Embora fosse considerado o ajuste de parâmetros de alguns dos algoritmos de controle apresentados, era claro que, nos casos em que isso era possível, não havia um método sistemático para se levais tais restrições em consideração.

O MPC aparece como alternativa às leis de controle clássicas devido a diversas vantagens, sendo as principais a facilidade de se considerar restrições relativas às entradas de controle e a capacidade de se gerar a lei de controle implicitamente, respeitando as condições de Brockett (BROCKETT, 1982) relativas a estabilidade de sistemas não-holonômicos sem deriva. O MPC ainda leva em consideração a minimização de uma função-objetivo, sendo a lei de controle ótima com relação a este critério.

Foi analisada, para o MPC, a relação entre o desempenho do sistema realimentado e a função-custo considerada. Para o problema de estabilização em um ponto, para o qual não existe uma lei de controle do tipo realimentação de estados suave e invariante no tempo que torne um ponto qualquer assintoticamente estável (BROCKETT, 1982), não basta que a função-custo tenha valor mínimo para o comportamento ideal do sistema, ou seja, no ponto de equilíbrio. A partir de funções-custo que utilizam transformações de coordenadas pode-se obter controladores MPC que tornam qualquer ponto assintoticamente estável, o que não acontece quando se consideram coordenadas cartesianas, que por sua vez permitem caracterizar o estado do robô de maneira mais simples (KÜHNE, 2005). Diferenças no desempenho dos diferentes controladores foram relacionadas às funções-custo consideradas.

Os controladores implementados foram executados em tempo real utilizando-se o

sistema RTAI (DOZIO; MANTEGAZZA, 2003). Os tempos de computação da lei de controle foram medidos, podendo-se então verificar que a implementação das diversas formulações de MPC em tempo real satisfizeram as restrições de tempo do robô móvel disponível no laboratório. Foi verificado que o controlador LMPC pode ser executado até mesmo por processadores ultrapassados, caso do Pentium II 300 MHz (vide Figura 18). Os controladores NMPC requerem mais tempo de computação que o controlador LMPC, sendo este tempo variável, dependente do estado e da referência. Já o tempo de computação do LMPC é aproximadamente constante e independente do estado, havendo variações apenas devido ao escalonamento do sistema de tempo real.

Para trabalhos futuros deseja-se implementar controladores MPC considerando o modelo dinâmico do sistema. Estima-se que, devido a ordem superior ao modelo cinemático de postura, será necessária maior capacidade de processamento. A existência de funções-custo que, utilizadas nos controladores MPC, levem a melhores desempenhos por parte do sistema realimentado pode ser investigada. O esquema de classes utilizado para os controladores pode ser alterado de maneira a considerar outros tipos de controlador, como por exemplo não sujeitos a restrições nas entradas, mais facilmente. A estabilidade do sistema em malha fechada e funções-custo específicas para que se possa garantir estabilidade podem ser investigadas.

REFERÊNCIAS

ABURTO, A. **FLOPS C Program (Double Precision) V2.0 18 Dec 1992.** 1992.

ALT, G. H. **Controle em Tempo Real de Robôs através de Redes IP**. 2003. Tese (Mestrado Profissional), Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2003. [Orientador: Marcelo Soares Lubaszewski e Walter Fetter Lages].

ASTOLFI, A. On The Stabilization of Nonholonomic Systems. In: IEEE AMERICAN CONFERENCE ON DECISION AND CONTROL, 33., 1994, Lake Buena Vista, FL. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 1994. p.3481–3486.

ATHANS, M. The Role and Use of the Stochastic Linear-Quadratic-Gaussian Problem in Control System Design. **IEEE Transactions on Automatic Control**, Piscataway, NJ, USA, v.16, n.6, p.529– 552, Dec. 1971.

BEMPORAD, A.; MORARI, M.; DUA, V.; PISTIKOPOULOS, E. N. The Explicit Linear Quadratic Regulator for Constrained Systems. **Automatica**, Oxford, UK, v.38, n.1, p.3–20, Jan. 2002.

BITSORIS, G. Positively Invariant Polyhedral Sets of Discrete-Time Linear Systems. **International Journal of Control**, London, UK, v.47, n.6, p.1713–1726, 1988.

BROCKETT, R. W. **New Directions in Applied Mathematics**. New York: Springer-Verlag, 1982.

CAMACHO, E. F.; BORDONS, C. **Model Predictive Control**. London: Springer-Verlag, 1999. (Advanced Textbooks in Control and Signal Processing).

CAMPION, G.; BASTIN, G.; D'ANDRÉA-NOVEL, B. Structural Properties and Classification of Kinematic and Dynamical Models of Wheeled Mobile Robots. **IEEE Transactions on Robotics and Automation**, New York, USA, v.12, n.1, p.47–62, Feb 1996.

CHWA, D. Sliding-Mode Tracking Control of Nonholonomic Wheeled Mobile Robots in Polar Coordinates. **IEEE Transactions on Control Systems Technology**, New York, v.12, n.4, p.637–644, Jul. 2004.

CLOUTIERS, P. **LXRT-Informed FAQs**. Parte da documentação da RTAI API, Disponível em: <http://www.rtai.org> [Online]. Acesso em 01/06/2006.

DORATO, P.; ABDALLAH, C.; CERONE, V. **Linear Quadratic Control**: an introduction. Englewood Cliffs, Nova Jersey, USA: Prentice Hall, 1995.

DOZIO, L.; MANTEGAZZA, P. Linux Real Time Application Interface (RTAI) in Low Cost High Performance Motion Control. In: MOTION CONTROL 2003 CONFERENCE, 2003, Milano, Italy. **Proceedings...** Milano, Italy: ANIPLA, 2003.

ESSEN, H. V.; NIJMEIJER, H. Non-Linear Model Predictive Control of Constrained Mobile Robots. In: EUROPEAN CONTROL CONFERENCE, 2001, Porto, Portugal. **Proceedings...** Porto, Portugal: EUCA, 2001. p.1157–1162.

FARINES, J.-M.; SILVA FRAGA, J. da; OLIVEIRA, R. S. de. **Sistemas de Tempo Real**. Florianópolis: Universidade Federal de Santa Catarina, 2000. Disponível em: <http://www.lcmi.ufsc.br/gtr/livro/principal.htm> [Online]. Acesso em 23/05/2007.

FU, K. S.; GONZALES, R. C.; LEE, C. S. G. **Robotics Control, Sensing, Vision and Intelligence**. New York: McGraw-Hill, 1987. (Industrial Engineering Series).

GERTZ, E. M.; WRIGHT, S. **OOQP User Guide**. Madison, USA: University of Wisconsin-Madison, 2004.

GERTZ, E. M.; WRIGHT, S. J. Object-Oriented Software for Quadratic Programming. **ACM Transactions on Mathematical Software**, New York, USA, v.29, n.1, p.58–81, Mar 2003.

GERTZ, M.; WRIGHT, S. **OOQP**: object-oriented software for quadratic programming. Disponível em: <http://pages.cs.wisc.edu/~swright/ooqp/> [Online]. Acesso em 04/07/2007.

GONDZIO, J. Multiple Centrality Corrections in a Primal-Dual Method for Linear Programming. **Computational Optimization and Applications**, New York, NY, USA, v.6, n.2, p.137–156, Sep. 1996.

GU, D.; HU, H. A Stabilizing Receding Horizon Regulator for Nonholonomic Mobile Robots. **IEEE Transactions on Robotics**, Piscataway, NJ, USA, v.21, n.5, p.1022–1028, Oct. 2005.

HOLANDA FERREIRA, A. B. de. **Novo Dicionário Aurélio da Língua Portuguesa**. 2.ed. Rio de Janeiro: Nova Fronteira, 1986.

KÜHNE, F. **Controle Preditivo de Robôs Móveis Não-Holonômicos**. 2005. Tese (Mestrado), Escola de Engenharia, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2005. [Orientador: João Manoel Gomes da Silva Jr. e Walter Fetter Lages].

KÜHNE, F.; LAGES, W. F.; GOMES DA SILVA JR., J. M. Model Predictive Control of a Mobile Robot Using Linearization. In: MECHATRONICS AND ROBOTICS 2004, 2004, Aachen, Germany. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2004. p.525–530.

KÜHNE, F.; LAGES, W. F.; GOMES DA SILVA JR., J. M. Point Stabilization of Mobile Robots With Nonlinear Model Predictive Control. In: IEEE INTERNATIONAL CONFERENCE ON MECHATRONICS & AUTOMATION, 2005, Niagara Falls, Canada. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 2005. p.1163–1168.

LAGES, W. F. **Controle e Estimação de Posição e Orientação de Robôs Móveis**. 1998. Tese (Doutorado), Instituto Tecnológico de Aeronáutica, São José dos Campos, SP, 1998. [Orientador: Elder M. Hemerly].

LAGES, W. F.; ALVES, J. A. V. Real-Time Control of a Mobile Robot Using Linearized Model Predictive Control. In: IFAC SYMPOSIUM ON MECHATRONIC SYSTEMS, 4., 2006, Heidelberg, Germany. **Anais...** Amsterdam: Elsevier, 2006. p.968–973.

LAGES, W. F.; HEMERLY, E. M. Smooth Time-invariant Control of Wheeled Mobile Robots. In: INTERNATIONAL CONFERENCE ON SYSTEMS SCIENCE, 13., 1998, Wrocław, Poland. **Proceedings...** Wrocław, Poland: Ofycina Wydawnicza Politechniki Wrocławskiej, 1998.

LATOMBE, J. C. **Robot Motion Planning**. Boston, USA: Kluwer Academic Publishers, 1991. n.124. (Kluwer International Series in Engineering and Computer Science).

LEONARD, J. J.; DURRANT-WHITE, H. F. Mobile Robot Localization by Tracking Geometric Beacons. **IEEE Transactions on Robotics and Automation**, Piscataway, NJ, USA, v.7, n.3, p.376–382, Jun. 1991.

MANTEGAZZA, P. **RTAI — Dissecting RTAI**. Disponível em: <`http://www.aero.polimi.it/~rtai/documentation/articles/paolo-dissectin%g.html`> [Online]. Acesso em 01/06/2006.

MEHROTRA, S. On the Implementation of a Primal-Dual Interior Point Method. **SIAM Journal on Optimization**, Philadelphia, USA, v.2, n.4, p.575–601, 1992.

MESQUINE, F.; TADEO, F.; BENZAOUIA, A. Regulator Problem for Linear Systems with Constraints on Control and its Increment or Rate. **Automatica**, Oxford, UK, v.40, n.8, p.1387–1395, Aug 2004.

MITTELMANN, H. D. **DONLP2 and Related Software for Nonlinear Programming**. Disponível em: <`http://plato.la.asu.edu/donlp2.html`> [Online]. Acesso em 04/07/2007.

MORARI, M.; LEE, J. H. Model Predictive Control: past, present and future. **Computers & Chemical Engineering**, New York, USA, v.23, n.4, p.667–682, May 1999.

OXFORD SENIOR DICTIONARY, 6.ed. Oxford, UK: Oxford University Press, 1978.

POMET, J. B.; THUILOT, B.; BASTIN, G.; CAMPION, G. A Hybrid Strategy for the Feedback Stabilization of Nonholonomic Mobile Robots. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1992, Nice, France. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 1992. p.129–134.

SAMSON, C.; AIT-ABDERRAHIM, K. Feedback Control of a Nonholonomic Wheeled Cart in Cartesian Space. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 1991, Sacramento, CA. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 1991. p.1136–1141.

SAMSON, C.; AIT-ABDERRAHIM, K. Feedback Stabilization of a Nonholonomic Wheeled Mobile Robot. In: IEEE/RSJ INTERNATIONAL WORKSHOP ON INTELLIGENT ROBOTS AND SYSTEMS, 1991, Osaka, Japan. **Proceedings...** Piscataway, NJ, USA: IEEE Press, 1991. p.1242–1247.

SHA, L.; RAJKUMAR, R.; LEHOCZKY, J. P. Priority Inheritance Protocols: an approach to real-time synchronization. **IEEE Transactions Computers**, New York, v.39, n.9, p.1175–1185, Sep. 1990.

SOETENS, P. **Porting your C++ GNU/Linux Application to RTAI/LXRT**. Disponível em: <`http://people.mech.kuleuven.be/~psoetens/portingtolxrt.html`> [Online]. Acesso em 01/06/2006.

SØRDALEN, O. J. **Feedback Control of Nonholonomic Mobile Robots**. 1993. Tese (Dr. ing.), The Norwegian Institute of Technology, Trondheim, Norway, 1993.

SPELLUCCI, P. A New Technique for Inconsistent QP Problems in the SQP Method. **Mathematical Methods of Operations Research**, New York, v.47, n.3, p.355–400, Oct 1998.

SPELLUCCI, P. An SQP Method for General Nonlinear Programs Using Only Equality Constrained Subproblems. **Mathematical Programming**, New York, v.82, n.3, p.413–448, Aug 1998.

SPELLUCCI, P. **Donlp2 Users Guide**. Darmstadt, Germany: Technical University at Darmstadt, Department of Mathematics, 1999.

TANENBAUM, A. S. **Modern Operating Systems**. Upper Saddle River, NJ: Prentice-Hall, 2001.

TEEL, A. R.; MURRAY, R. M.; WALSH, G. C. Non-holonomic Control Systems: from steering to stabilization with sinusoids. **International Journal of Control**, London, UK, v.62, n.4, p.849–870, 1995.

VU, H. V.; ESFANDIARI, R. S. **Dynamic Systems**: modeling and analisys. New York: McGraw-Hill, 1997.

WEBSTER, N. **Webster's New Twentieth Century Dictionary of the English Language**. 2.ed. New York: Simon and Schuster, 1983.