Juan Ramón Belchior de França Silva

Desenvolvimento de um Sistema de Controle para Caminhada Autônoma de um Robô Humanoide

Goiânia, Brasil

2018

Juan Ramón Belchior de França Silva

Desenvolvimento de um Sistema de Controle para Caminhada Autônoma de um Robô Humanoide

Monografia submetida a graduação da Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás, como requisito parcial necessário para de obtenção de título de Engenheiro de Computação

Universidade Federal de Goiás – UFG

Escola de Engenharia Eletrica, Mecanica e de Computação

Orientador: Prof. Dr. Marco Antonio Assfalk de Oliveira

Goiânia, Brasil

2018

Juan Ramón Belchior de França Silva

Desenvolvimento de um Sistema de Controle para Caminhada Autônoma de um Robô Humanoide

Monografia submetida a graduação da Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás, como requisito parcial necessário para de obtenção de título de Engenheiro de Computação

Trabalho aprovado. Goiânia, Brasil, 19 de julho de 2018:

Marco Antonio Assfalk de Oliveira, Dr.
Orientador

Sérgio Pires Pimentel, Dr.
Convidado 1

Kléber Macedo Cabral, Ms.
Convidado 2

Goiânia, Brasil
2018

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Dedico este trabalho, primeiramente a Deus e a minha família que sempre me apoiou em toda a minha vida, a minha mãe Maria Helena por suas palavras de incentivo, conforto, esperança e, sobretudo, por suas orações que me deram força para seguir adiante. Ao meu irmão e melhor amigo Hugo Eduardo, por todo apoio em toda a minha. Agradeço as minhas amigas Bruna Michelly e Ana Gabriella, compartilhando os mesmos sofrimentos nas aulas, nas disciplinas, nos testes, nos trabalhos difíceis e especialmente as muitas risadas no dia a dia e todos os cafés. Agradeço a todos os meus amigos que, de uma maneira ou outra, me ajudaram com as suas palavras de incentivo, de apoio, que nunca deixaram de acreditar em mim, mesmo com todas as dificuldades que passei ao longo do curso de engenharia. Sem essas pessoas provavelmente eu não teria conseguido. Agradeço de uma forma especial à Raquel Carneiro pela sua amizade, pelos cafés, por sempre ter palavras de incentivo, de apoio e que sobretudo nunca deixou de me lembrar de que Deus existe e sempre está connosco dia a dia. Dedico também o trabalho ao Núcleo de Robótica Pequi Mecânico e a todos os membros, por todo esse tempo de trabalho e de amizade, em especial a equipe do Humanoide por todo esforço e empenho. Agradeço por fim, o meu orientador, Prof. Dr. Marco Antonio Assfalk de Oliveira pela orientação e paciência, com quem aprendi muito e que me deu todo o suporte necessário ao desenvolvimento deste projeto.

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

Este trabalho descreve o desenvolvimento de um sistema de controle de malha fechada para caminhada e orientação autônoma de um robô humanoide. O robô Hubo foi escolhido como objeto de estudo para este trabalho, devido sua semelhança com a maioria dos robôs humanoides utilizados em pesquisa. Para sua modelagem, foi aplicada a álgebra dos quatérnios duais. Além disso, é utilizado o modelo dinâmico *3D DUAL SLIP* para gerar os padrões de caminhada para este robô. Como foco do trabalho, é desenvolvido um controlador LQR no espaço dos quatérnios duais, para regular o estado do robô, tendo o objetivo de garantir que o robô humanoide descreva as trajetórias obtidas pelo modelo *3D DUAL-SLIP*, sendo mostrado que o controlador LQR pode ser usado para o rastreamento de trajetórias.

Abstract

This work describes the development of a closed loop control system for walking and autonomous orientation of a humanoid robot. The Hubo robot was chosen as the object of study for this work, due to its similarity with most humanoid robots used in research. The dual quaternion algebra was applied at the modeling. In addition, the dynamic model 3D DUAL SLIP was used to generate the walking patterns for this robot. Further, a LQR controller is developed in the dual quaternion space, to regulate the state of the robot, aiming to ensure that the humanoid robot described the trajectories obtained by the 3D DUAL-SLIP model, showing that the LQR controller can be used for trajectory tracing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CoM	Centro de Massa (Center of Mass)
CP	Ponto de Captura (Capture Point)
DCM	Componente Divergente do Movimento (Divergent-Component of Motion Point)
DOF	Graus de liberdade (Degrees of Freedom)
D-H	Denavit-Hartenberg
FKM	cinemática direta (Forward Kinematic Model)
K, K+FW	controlador proporcional sem e com um *feed-forward*, respectivamente
LQR	Regulador Linear Quadrático (Linear–Quadratic Regulator)
LIPM	Modelo Linear de Pendulo Invertido (Linear Inverted Pendulum Model)
DUAL SLIP	Modelo 2D de Pendulo Invertido com massa e duas molas (Dual Spring Loaded Inverted Pendulum)
SLIP	Modelo de Pendulo Invertido com Massa Mola (Spring Loaded Inverted Pendulum)
3D DUAL SLIP	Modelo 3D de Pendulo Invertido com Massa e Duas Molas (3D Dual Spring Loaded Inverted Pendulum)
HUBO	Robô Humanoide
MS	Midstance
TD	Touchdown
LH	Lowest Height
LO	Lift Off
SS_A	Suporte simples da perna A (Single Suporte)
DS	Suporte Duplo (Double Suport)
$vGRF$	Força de reação vertical (vertical Ground Reaction Force)
ZMP	Ponto de Momento Zero (Zero Moment Point)

Lista de símbolos

a, b, c, ...	escalares são representados por letras minúsculas
a, b, c, ...	vetores são representados por letras em negrito minúsculas
A, B, C, ...	matrizes são representadas por letras maiúsculas em negrito
$\mathbf{I}$	matriz de identidade com dimensões apropriadas
$\|\|\mathbf{f}\|\|$	norma da função $\mathbf{f}$
$\dot{\mathbf{f}}$	derivada temporal da função $\mathbf{f}$
$\mathbf{q}^*$	conjugado de $\mathbf{q}$
$\mathbf{J}^+$ *ou* $\mathbf{J}^\dagger$	pseudo-inversa de Moore-Penrose de $\mathbf{J}$
$\mathbf{J}^{-1}$ *ou* $inv(\mathbf{J})$	operação inversa da matriz $\mathbf{J}$
$\mathbf{a}^{-1}$ *ou* $inv(\mathbf{a})$	operação inversa do vetor $\mathbf{a}$
$\in$	Pertence
$\mathbf{g}$	vetor gravidade
k	constante de mola
m	massa
$\mathbf{J}$, $\mathcal{J}$	Jacobiano Analítico
θ, ϕ, α	ângulos
$\mathcal{F}$	sistema de referência
$\mathbb{R}$	conjunto dos números reais
$\mathbb{H}$	espaço dos quatérnios
$\mathbb{C}$	conjunto do números complexos
$\mathcal{H}$	espaço dos quatérnios duais
$\hat{i}, \hat{j}, \hat{k}$	vetores unitários
ε	unidade dual

vec_4, vec_8	operador vec para quatérnios e quatérnios duais respectivamente
$\overset{+}{\mathbf{H}}_4(\cdot),\ \overset{-}{\mathbf{H}}_4(\cdot)$	Operadores de Hamilton para quatérnios
$\overset{+}{\mathbf{H}}_8(\cdot),\ \overset{-}{\mathbf{H}}_8(\cdot)$	Operadores de Hamilton para quatérnios duais

Sumário

1 Introdução

1.1 Objetivos geral e específicos

Este trabalho tem como objetivo a modelagem do robô humanoide HUBO, o planejamento de trajetórias do CoM utilizando o modelo 3D Dual SLIP e o desenvolvimento de um sistema de controle no espaço dos quatérnios duais. Tende em vista que, os controladores implementados tem como objetivo de garantir que o robô humanoide modelado, descreva a trajetória obtida pelo modelo *3D DUAL-SLIP*, sendo este modelo, capaz de gerar padrões similares à caminhada humana. Como proposta um controlador LQR é avaliado para descrever as trajetórias do CoM e o controlador proporcional com um *feed-forward* utilizado para descrever o percurso dos pés.

1.2 Motivação

O Núcleo de Robótica Pequi Mecânico é uma organização estudantil sem fins lucrativos que reúne, incentiva e oferece suporte às iniciativas de pesquisa e desenvolvimento em robótica e automação na Universidade Federal de Goiás. O núcleo é composto e totalmente gerido por alunos da universidade, tendo como panorama gerencial a estrutura de uma empresa, com cargos administrativos e técnicos. O grupo foi fundado em 2010 por alunos do curso de Engenharia Elétrica da Universidade Federal de Goiás, com o foco de participar do 1º Torneio Universitário de Robótica, em Uberlândia-MG. Desde então o grupo não parou de evoluir, sendo que o grupo já esteve presente em mais de oito competições nacionais, em ao menos 3 categorias diferentes, além da organização de eventos acadêmicos, palestras e minicursos para dentro e fora da universidade.

Em 2015 o núcleo iniciou pesquisas na área de robôs humanoides, tendo como objetivo participar da Competição Latino-Americana Robótica (LARC), na categoria IEEE Humanoid Robot Racing Child Size http://www.cbrobotica.org/?page_id=91. Nesta categoria, um robô humanoide deve ser capaz de mover-se, em uma linha reta, em uma distância superior a 4 metros em menos de 3 minutos.

Em 2018 o núcleo de robótica começou a pesquisa para a participação na RoboCup http://www.cbrobotica.org/?page_id=122, iniciando uma nova equipe de competição e uma nova área. O novo objetivo teve a finalidade de alcançar e inovar outras áreas de pesquisa até então não exploradas pelo núcleo. A RoboCup é uma competição de robótica anual internacional fundada em 1997. A RoboCup Tem como objetivo desenvolver uma equipe de robôs humanoides capazes de derrotar a equipe campeã mundial de futebol

humana, desta forma a RoboCup tem como objetivo fomentar pesquisas na área de robôs humanoides. Sendo assim, este trabalho tem com o objetivo de contribuir com o Núcleo de Robótica e com as pesquisas na área de robôs humanoides.

Na seção de anexos é descrito o trabalho desenvolvido pelo Núcleo de Robótica Pequi Mecânico na área de humanoides, sendo que este trabalho esteve vigente até Outubro de 2017. Tendo em vista a constante frequência de utilização do hardware no decorrer deste tempo, os atuadores do robô (os servomotores) se danificaram, desta forma, o projeto permaneceu sem robô, por um período de quase 6 meses, até que a aquisição de novos componentes.

Contudo, a pesquisa dentro do núcleo passou por mudanças no escopo do projeto, inserindo-se na categoria de Futebol de Robôs Humanoides, a RoboCup Humanoid Soccer Kid Size. Visto que, o projeto foi completamente reformulado, havendo mudanças em todas as áreas sendo estas: estrutura mecânica, hardware, controle de baixo nível (nível de hardware), controle de alto nível, detecção e odometria visual.

Inicialmente, a proposta aqui descrita, seria a implementação deste trabalho no robô, no entanto, optou-se por uma mudança de escopo, devido a ausência de um robô durante período do desenvolvimento deste trabalho. Visto que, posteriormente, este trabalho poderá ser implementado no robô que será construído pelo núcleo.

1.3 Metodologia

Será utilizado o modelo 3D Dual-SLIP para a gerar trajetórias dinamicamente estáveis do CoM. A modelagem do robô foi feita utilizando a álgebra dos quatérnios duais utilizando os parâmetros de denavit-hartenberg, para representear o robô humanoide no espaço. Como o modelo 3D Dual-SLIP é estável apenas para 2 passos é demostrado o controlador LQR proposto por Liu et al. (2015), no entanto este controlador não foi implementado neste trabalho, sendo utilizado o modelo para gerar um trajetória estável para 2 passos e este percurso replicado no tempo. Foi desenvolvido um controlador LQR no espaço dos quatérnios duais para regular o estado do modelo, tendo o objetivo de garantir que o robô humanoide descreva as trajetórias obtidas pelo modelo 3D DUAL-SLIP. As trajetórias de pernas em movimento foram geradas utilizando uma curva de Bézier de 4ª ordem relativa ao CoM. O controlador proporcional com um *feedfoward* é utilizado para descrever as trajetórias dos pés.

A Fig. 1.1 demostra esta proposta, com o modelo *3D Dual SLIP* são geradas as trajetórias do centro de massa e dos pés, sendo $\underline{\mathbf{h}}_{ref,CoM}$ e $\underline{\mathbf{h}}_{CoM}$ posição e orientação desejada e atual do robô para o centro de massa. Calculado o erro representado por $\underline{\mathbf{e}}$, é computado o controlador LQR que retorna as velocidades das variáveis de junta para a perna que está em contato com o solo, demostrado pela cor azul, as velocidades $\dot{\vec{\theta}}$ são

integradas e o resultado é o incremento, $\vec{\theta}_d$, que deve ser adicionado a configuração atual do robô. Desta forma é simulado o robô para a nova posição e computado a cinemática direta para obter a nova posição e orientação do CoM. Semelhantemente, é implementado o controlador proporcional com um *feedfoward* para a perna em movimento, demostrado pela cor vermelha, onde $\underline{\mathbf{h}}_{ref,f}$ e $\underline{\mathbf{h}}_f$ representam posição e orientação de referência e atual do robô.

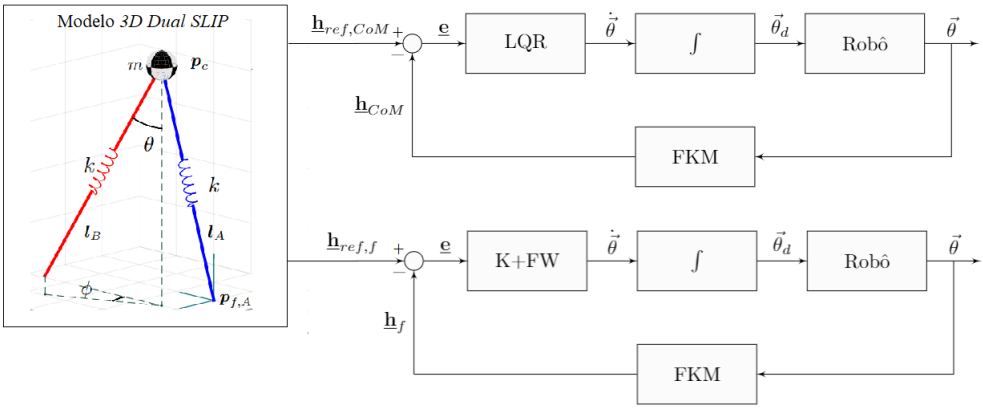


Figura 1.1: Estrutura do trabalho

1.4 Revisão bibliográfica

Dentro do âmbito da robótica, robôs humanoides são alvos de inúmeras pesquisas. Devido à sua semelhança com os seres humanos, estes robôs apresentam um potencial para realizar tarefas humanas, tais como auxiliem ou substituam em trabalhos realizados em casas, escritórios, espaços abertos ou ambientes perigosos para a presença de um ser humano, sendo que, os robôs humanoides geralmente requer menos modificações em equipamentos ou instalações e no ambiente, tendo em vista que estes ambientes originalmente foram projetadas para humanos. Um dos principais objetivos ao estudar os robôs humanoides, refere-se a necessidade de locomoção para exploração em terrenos irregulares e com obstáculos, pois possuem vantagens nessas áreas sobre os robôs com rodas. O estudo da locomoção de robôs, apresenta-se relevante, mediante as diversas aplicações as quais robôs são requisitados. A capacidade de locomoção de robôs humanoides, torna-se um problema complexo, devido ao alto número de graus de liberdade e suas características dinâmicas e, por este motivo, contribui com diversos desafios a serem solucionados, nesta vasta área de pesquisa. Como apresentado em Liu et al. (2015) e Full e Koditschek (1999) várias áreas de pesquisa tem concentrado esforços no uso de modelos simplificados para gerar padrões de locomoção. Estes modelos objetivam focar nas principais características de um movimento complexo de corpo inteiro, como a caminhada dinâmica, e a aplicação de modelos simples podem resolver muitos dos desafios de controle de caminhada. O Mo-

delo de pêndulo invertido linear, LIPM (Kajita, Matsumoto e Saigo (2001)), é um modelo simples e bem conhecido, no qual foi desenvolvido para reproduzir a caminhada humana. As equações de movimento para o LIPM são lineares e desacopladas, tornando possível o uso de métodos analíticos. Desde a introdução e aplicação do LIPM, muitas estratégias de controle foram desenvolvidas. Em Liu et al. (2015) cita uma abordagem utilizando o LIPM, na qual utiliza a técnica de controle *Zero Moment Point* (Kajita et al. (2003)), sendo esta estratégia usada para gerar uma marcha quase estática. O ponto de momento zero é um conceito relacionado com a dinâmica e controle de locomoção para robôs com pernas, por exemplo, para robôs humanoides específica o ponto em relação ao qual a força de reação dinâmica no contato do pé com o solo não produz nenhum momento na direção horizontal, isto é, o ponto no qual o total de forças de inércia horizontais e gravidade é igual a zero. O conceito assume que a área de contato é plana e tem atrito suficientemente alto para impedir que os pés deslizem. Outros trabalhos relacionados, como em Pratt et al. (2006), mostra o conceito de um "ponto de captura" (CP), como o ponto no chão, no qual o robô deve colocar seu ZMP, a fim de assintoticamente parar o movimento do centro de massa , sendo sua computação mostrada com o LIPM. Inicialmente, o ponto de captura foi proposto como sendo um ponto fixo no chão, no qual o humanoide poderia se recuperar de uma queda próxima. Posteriormente, foi proposto em outros estudos o ponto de captura é mudado dinamicamente (chamando de Componente Divergente do Movimento, ou DCM) e este tem sido usado para desenvolver caminhada dinâmica. Nestes trabalhos, foi possível verificar que o CoM sempre converge para o DCM, desta forma, o controle da dinâmica do CoM pode ser transformado no controle da dinâmica do DCM. Dados estes estudos, em trabalhos mais recentes, algumas extensões do conceito do DCM foram aplicadas, sendo agora, o controle de movimento baseado no DCM, com objetivo de adaptar os valores não constantes de CoM e altura do chão, como mostrado em Englsberger, Ott e Albu-Schaffer (2013). Liu et al. (2015) cita em seu trabalho que, outro modelo menos estudado para descrever uma caminhada é o Pêndulo invertido de dupla mola (Dual-SLIP). Este modelo é uma variação do modelo SLIP (*Spring Loaded Inverted Pendulum*), o qual, tem sido aplicado em robôs humanoides com o objetivo de modelar a corrida (Wensing e Orin (2013)) e o salto deste (Wensing e Orin (2014)). Geyer, Seyfarth e Blickhan (2006) apresenta em seu trabalho que, o modelo Dual-SLIP pode reproduzir a dinâmica da marcha humana com melhor fidelidade, em comparação ao LIPM. O modelo Dual-SLIP pode produzir oscilações verticais do CoM e padrões da força de reação vertical do solo semelhante aos encontrados em humanos. Este é o recurso mais útil do modelo Dual-SLIP para robótica, sendo que, o modelo também aborda o período de suporte duplo para caminhada . A Caminhada baseada no LIPM ignora completamente o suporte duplo ou é designado manualmente um período de suporte duplo. Pesquisas mais recentes (Liu et al. (2015)), apresenta uma maneira de encontrar trajetórias periódicas de caminhada utilizando o modelo *3D Dual-SLIP*, sendo este, uma extensão do modelo DUAL-SLIP.

O modelo *3D Dual-SLIP* consegue localizar a posição para os dois pés, sendo capaz de gerar uma trajetória dinâmica para o CoM semelhante ao observado durante a caminhada humana, assim como, o modelo DUAL-SLIP, incluindo oscilações verticais e a oscilação lateral, sendo esta, o diferencial ao modelo DUAL-SLIP. O modelo 3D Dual-SLIP foi utilizado no desenvolvimento deste trabalho para encontrar os padrões de caminhada para um robô humanoide. O robô escolhido é o robô Hubo, devido sua semelhança com a maioria dos robôs humanoides utilizados em pesquisa, sendo que, estes conceitos podem ser aplicados ao demais robôs humanoides.

A representação de um corpo no espaço é dado pela posição e orientação representada por um sistema de coordenada anexado ao corpo, e são representados em relação a um sistema base (sistema inercial). A posição e orientação podem ser parametrizadas de diferentes maneiras, mais comumente são aplicadas as matriz de transformação homogêneas, ângulos de Euler (Paul (1982)), quatérnios unitários (Xian et al. (2004)) ou quatérnios duais (Wu et al. (2005)). Em uma matriz de transformação homogênea, doze parâmetros são usados para representar a posição e orientação de um corpo, sendo que, ainda deve ser um quatérnio unitário ou ângulos Euler para a representação da orientação. Além disso, todos os tipos de ângulos de Euler têm uma natureza de rotação em sequência, sendo assim, o método do ângulo de Euler é adequado para representar a orientação de único frame. Outro problema para os ângulos de Euler se refere as singularidades de representação ocorridas (*guimbal lock*). A utilização de quatérnios para representar a orientação apresenta a vantagem de não possuir singularidades, mas dificulta o projeto do controlador, dado que os erros de posição e orientação são calculados separadamente (Xian et al. (2004)). Em algumas técnicas de controle, geralmente, são utilizados dois controladores, um para a rotação e outro para translação. Alguns autores como Aspragathos e Dimitros (1998), Funda e Paul (1990), Kenwright (2006) afirmam que o quatérnio dual é a forma mais compacta e eficiente para expressar o movimento de um corpo no espaço. Kenwright (2006) apresenta que o modelo utilizando quatérnios duais é computacionalmente eficiente, robusto e flexível para representação de transformações rígidas. Enquanto as matrizes apresentam o problema de *guimbal lock*, este não é apresentado quando são utilizados quatérnios duais. Ao lidar com transformações de corpos rígidos, os métodos utilizando quatérnios apresentam mais vantagens.

1.5 Estrutura do documento

O trabalho proposto é dividido nas seguintes etapas: No capítulo 2 é demostrado os fundamentos matemáticos, no qual, é descrito a álgebra dos quatérnios duais e sua utilização para modelagem de cadeias cinemáticas. No capítulo 3 é apresentado o modelo dinâmico *3D DUAL SLIP*, a possibilidade de utilizá-lo com o propósito de gerar trajetórias para caminhadas. Além disso, é demonstrado o controlador proposto por Liu et al.

(2015). No capítulo 4 é mostrado como podemos implementar controladores cinemáticos, sendo estes o controlador proporcional, controlador proporcional com um termo de *feed-foward* e o controlador LQR utilizando a álgebra dos quatérnios duais. No capítulo 5 é descrito a modelagem referente do robô HUBO, sua cinemática direta e seus parâmetros a serem utilizados no modelo *3D DUAL SLIP*. E por último são mostrados os resultados deste trabalho, utilizando o modelo *3D DUAL SLIP* para gerar uma trajetória para o robô humanoide HUBO, sendo implementado um controlador LQR no espaço dos quatérnios duais para realizar o rastreamento da trajetória obtida para o CoM e o controlador proporcional com um *feedfoward* para descrever o percurso dos pés.

1.6 Conclusões Parciais

O modelo *3D Dual-SLIP* será utilizado no desenvolvimento deste trabalho para encontrar os padrões de caminhada para um robô humanoide. Este modelo é capaz de gerar uma trajetória dinâmica para o CoM semelhante ao observado durante a caminhada humana. O robô escolhido é o robô Hubo, devido sua semelhança com a maioria dos robôs humanoides utilizados em pesquisa, sendo que, estes conceitos podem ser aplicados ao demais robôs humanoides, sendo utilizado para a sua representação a álgebra dos quatérnios duais. O uso dos quatérnios duais tem como objetivo simplificar a modelagem do robô, bem como, no projeto dos controladores cinemáticos. Como proposta um controlador LQR é avaliado para descrever as trajetórias do CoM e o controlador proporcional com um *feedforward* é utilizado para descrever o percurso dos pés, sendo as trajetórias dos pés geradas utilizando uma função de Bézier de 4ª ordem entre os pontos de contato gerados pelo modelo *3D Dual SLIP*.

2 Fundamentos Matemáticos

Este capítulo tem como objetivo apresentar os fundamentos matemáticos para representar transformações entre sistemas de coordenadas e movimentos de corpos rígidos utilizando os quatérnios duais unitários. Também é apresentado a convenção de Denavit-Hartenberg, reescrita por meio de quatérnios duais.

2.1 Quatérnios

Os quatérnios foram introduzidos por Hamilton no século XIX e são uma extensão do conjunto dos números complexos que pertence ao conjunto $\mathbb{H}$ (Adorno (2011)). São compostos por uma parte real e três partes imaginárias definidas por $\hat{i}, \hat{j}, \hat{k}$ que possuem as seguintes propriedades:

$$\hat{i}^2 = \hat{j}^2 = \hat{k}^2 = \hat{i}\hat{j}\hat{k} = -1 \tag{2.1}$$

Pode-se definir um quatérnio $\mathbf{h}$ como o número hipercomplexo (Adorno (2011)):

$$\begin{aligned} \mathbf{h} &= h_1 + \hat{i}h_2 + \hat{j}h_3 + \hat{k}h_4, \qquad h_i \in \mathbb{R}, i = 1, ..., 4 && (2.2)\\ &= Re(\mathbf{h}) + \mathbf{i_m} \cdot Im(\mathbf{h}) && (2.3) \end{aligned}$$

no qual, $Re(h) = h_1$ e $Im(h) = [\ h_2\ h_3\ h_4\ ]^T$ representam respectivamente a parte real e a parte imaginária do quatérnio. O termo $\mathbf{i_m} = [\ \hat{i}\ \hat{j}\ \hat{k}\ ]^T$ é vetor unitário imaginário , sendo análogo a unidade imaginária de um número complexo do conjunto $\mathbb{C}$ e o operador $\cdot$ consiste no produto escalar entre dois vetores. As operações matemáticas com quatérnios são descritas abaixo (Adorno (2011))

Definição 1 *Seja* $\mathbf{h}$ *e* $\mathbf{h}' \in \mathbb{H}$ *, a adição/subtração de quatérnios é definida como,*

$$\begin{aligned} \mathbf{h} + \mathbf{h}' &= (\mathbf{h}_1 + \mathbf{h}_1') + \hat{i}(\mathbf{h}_2 + \mathbf{h}_2') + \hat{j}(\mathbf{h}_3 + \mathbf{h}_3') + \hat{k}(\mathbf{h}_4 + \mathbf{h}_4')\\ &= (Re(\mathbf{h}) + Re(\mathbf{h}')) + \mathbf{i_m} \cdot (Im(\mathbf{h}) + Im(\mathbf{h}')) && (2.4) \end{aligned}$$

$$\begin{aligned} \mathbf{h} - \mathbf{h}' &= (\mathbf{h}_1 - \mathbf{h}_1') + \hat{i}(\mathbf{h}_2 - \mathbf{h}_2') + \hat{j}(\mathbf{h}_3 - \mathbf{h}_3') + \hat{k}(\mathbf{h}_4 - \mathbf{h}_4')\\ &= (Re(\mathbf{h}) - Re(\mathbf{h}')) + \mathbf{i_m} \cdot (Im(\mathbf{h}) - Im(\mathbf{h}')) && (2.5) \end{aligned}$$

sendo o resultado outro quatérnio.

Definição 2 *Seja* $\mathbf{h}$ *e* $\mathbf{h}' \in \mathbb{H}$ *, a multiplicação de quatérnios é definida como,*

$$\mathbf{h}\mathbf{h}' = (h_1 + \hat{i}h_2 + \hat{j}h_3 + \hat{k}h_4)(h_1' + \hat{i}h_2' + \hat{j}h_3' + \hat{k}h_4')$$

$$
\begin{aligned}
&= (h_1h_1^{'} - h_2h_2^{'} - h_3h_3^{'} - h_4h_4^{'}) + \\
&= \hat{i}(h_1h_2^{'} + h_2h_1^{'} + h_3h_4^{'} - h_4h_3^{'}) + \\
&= \hat{j}(h_1h_3^{'} - h_2h_4^{'} + h_3h_1^{'} + h_4h_2^{'}) + \\
&= \hat{k}(h_1h_4^{'} + h_2h_3^{'} - h_3h_2^{'} + h_4h_1^{'}), \qquad (2.6)
\end{aligned}
$$

sendo o resultado outro quatérnio. A multiplicação entre dois quatérnios não é comutativa , ou seja,

$$\mathbf{h}\mathbf{h}^{'} \neq \mathbf{h}^{'}\mathbf{h} \qquad (2.7)$$

Definição 3 *Seja* $\boldsymbol{h} \in \mathbb{H}$*, o conjugado do quatérnio é dado por*

$$
\begin{aligned}
\mathbf{h}^{*} &= h_1 - \hat{i}h_2 - \hat{j}h_3 - \hat{k}h_4 \\
&= Re(\mathbf{h}) - \mathbf{i_m} \cdot Im(\mathbf{h}) \qquad (2.8) \\
& \qquad (2.9)
\end{aligned}
$$

Definição 4 *Seja* $\boldsymbol{h} \in \mathbb{H}$*, a norma de um quatérnio é dada por*

$$
\begin{aligned}
||\mathbf{h}|| &= \sqrt{Re(\mathbf{h})^2 + Im(\mathbf{h}) \cdot Im(\mathbf{h})} \\
&= \sqrt{\mathbf{h}\mathbf{h}^{*}} = \sqrt{\mathbf{h}^{*}\mathbf{h}} \qquad (2.10)
\end{aligned}
$$

Além das definições de quatérnios, são definidos o operador vec_4 e operadores de $Hamilton$ ($\overset{+}{\mathbf{H}}_{\mathbf{4}}$ e $\overset{-}{\mathbf{H}}_{\mathbf{4}}$).

Definição 5 *Seja o quatérnio* $\mathbf{h} = h_1 + \hat{i}h_2 + \hat{j}h_3 + \hat{k}h_4$*, operador* vec_4 *faz um mapeamento bijetor* $\mathbb{H} \rightarrow \mathbb{R}^4$

$$vec_4(\mathbf{h}) \triangleq [\ h_1\ h_2\ h_3\ h_4\ ]^T \qquad (2.11)$$

desta forma o quatérnio é mapeado em uma forma vetorial. A operação inversa $vec4^{-1}$ *mapeia* $\mathbb{R}^4 \rightarrow \mathbb{H}$*.*

Definição 6 *Seja o quatérnio* $\mathbf{h} = h_1 + \hat{i}h_2 + \hat{j}h_3 + \hat{k}h_4$*, os operadores de Hamilton* $\overset{+}{\mathbf{H}}_{\mathbf{4}}(\cdot)$ *e* $\overset{-}{\mathbf{H}}_{\mathbf{4}}(\cdot)$ *são definidos:*

$$
\overset{+}{\mathbf{H}}_4(\mathbf{h}) = \begin{bmatrix} h_1 & -h_2 & -h_3 & -h_4 \\ h_2 & h_1 & -h_4 & h_3 \\ h_3 & h_4 & h_1 & -h_2 \\ h_4 & -h_3 & h_2 & h_1 \end{bmatrix}_{4\times4}, \ \overset{-}{\mathbf{H}}_4(\mathbf{h}^{'}) = \begin{bmatrix} h_1 & -h_2 & -h_3 & -h_4 \\ h_2 & h_1 & h_4 & -h_3 \\ h_3 & -h_4 & h_1 & h_2 \\ h_4 & h_3 & -h_2 & h_1 \end{bmatrix}_{4\times4} \qquad (2.12)
$$

desta forma o quatérnio é mapeado em uma forma matricial.

A definição destes operadores são de grande utilidade para realização de algumas operações, pela afinidade dos recursos de software atuais em manipulação de matrizes e vetores. Além disso, a operação de multiplicação de quatérnios torna-se algebricamente comutativa. Desta forma, é possível demonstrar que:

$$vec_4(\mathbf{h}\mathbf{h}^{'}) = \overset{+}{\mathbf{H}}_4(\mathbf{h})vec_4(\mathbf{h}^{'}) = \bar{\mathbf{H}}_4(\mathbf{h}^{'})vec_4(\mathbf{h}) \tag{2.13}$$

sendo $\mathbf{h}$ e $\mathbf{h}^{'}$ dois quatérnios.

2.2 Números Duais

Os números duais são formados por duas partes, ligadas por um operador de adição ou subtração. Uma das partes é multiplicada por uma unidade dual ε, $\underline{a} = a + \varepsilon a^{'}$, a é a parte primária e $a^{'}$ a parte dual do número. A unidade dual ε é definida como:

$$\varepsilon \neq 0 \; e \; \varepsilon^2 = 0 \tag{2.14}$$

Definindo os operadores $\mathcal{P}(\cdot)$ e $\mathcal{D}(\cdot)$, é possível obter as duas partes do numero dual separadamente, ou seja, $\mathcal{P}(\underline{a})$ e $\mathcal{D}(\underline{a})$ retornam a parte primária e dual respectivamente. As propriedades dos números duais são descritas abaixo (Adorno (2011))

$$\underline{a} = \mathcal{P}(\underline{a}) + \varepsilon\mathcal{D}(\underline{a}) \tag{2.15}$$

Definição 7 *Sejam dois números duais $\underline{a_1}$ e $\underline{a_2}$ a soma/subtração desses números é dada por:*

$$\underline{a_1} + \underline{a_2} = (\ \mathcal{P}(\underline{a_1}) + \mathcal{P}(\underline{a_2})\ ) + \varepsilon(\ \mathcal{D}(\underline{a_1}) + \mathcal{D}(\underline{a_2})\ ) \tag{2.16}$$

$$\underline{a_1} - \underline{a_2} = (\ \mathcal{P}(\underline{a_1}) - \mathcal{P}(\underline{a_2})\ ) + \varepsilon(\ \mathcal{D}(\underline{a_1}) - \mathcal{D}(\underline{a_2})\ ) \tag{2.17}$$

Definição 8 *Sejam dois números duais $\underline{a_1}$ e $\underline{a_2}$ a multiplicação desses números é dada por:*

$$\begin{aligned} \underline{a_1 a_2} &= (\ \mathcal{P}(\underline{a_1}) + \varepsilon\mathcal{D}(\underline{a_1})\ )(\ \mathcal{P}(\underline{a_2}) + \varepsilon\mathcal{D}(\underline{a_2})\ ) \\ &= \mathcal{P}(\underline{a_1})\mathcal{P}(\underline{a_2}) + \varepsilon(\ \mathcal{P}(\underline{a_1})\mathcal{D}(\underline{a_2}) + \mathcal{D}(\underline{a_1})\mathcal{P}(\underline{a_2})\ ) \end{aligned} \tag{2.18}$$

Definição 9 *O inverso de um escalar dual $\underline{a}$ é*

$$\underline{a}^{-1} = \frac{1}{\mathcal{P}(\underline{a})} - \varepsilon\frac{\mathcal{D}(\underline{a})}{\mathcal{P}(\underline{a})^2}, \qquad \mathcal{P}(\underline{a}) \neq 0 \tag{2.19}$$

As duas partes $\mathcal{P}(\cdot)$ e $\mathcal{D}(\cdot)$ são do mesmo tipo e podem ser escalares, vetores, matrizes e até quatérnios.

2.3 Quatérnios Duais

Os quatérnios duais são números duais cujas partes primária e dual são quatérnios (Adorno (2011)),

$$\underline{\mathbf{h}} = \mathcal{P}(\underline{\mathbf{h}}) + \varepsilon\mathcal{D}(\underline{\mathbf{h}}) \tag{2.20}$$

As partes real e imaginária do quatérnio dual $\underline{\mathbf{h}}$ são obtidas da seguinte maneira

$$Re(\underline{\mathbf{h}}) \triangleq Re(\mathcal{P}(\underline{\mathbf{h}})) + \varepsilon Re(\mathcal{D}(\underline{\mathbf{h}})) \tag{2.21}$$

$$Im(\underline{\mathbf{h}}) \triangleq Im(\mathcal{P}(\underline{\mathbf{h}})) + \varepsilon Im(\mathcal{D}(\underline{\mathbf{h}})) \tag{2.22}$$

As propriedades para os quatérnios duais são descritas abaixo

Definição 10 *Seja* $\underline{\mathbf{h}}$ *e* $\underline{\mathbf{h}}^{'} \in \mathcal{H}$*, a adição/subtração entre dois quatérnions duais é definida como,*

$$\underline{\mathbf{h}} + \underline{\mathbf{h}}^{'} = (\mathcal{P}(\underline{\mathbf{h}}) + \mathcal{P}(\underline{\mathbf{h}}^{'})) + \varepsilon(\mathcal{D}(\underline{\mathbf{h}}) + \mathcal{D}(\underline{\mathbf{h}}^{'})) \tag{2.23}$$

$$\underline{\mathbf{h}} - \underline{\mathbf{h}}^{'} = (\mathcal{P}(\underline{\mathbf{h}}) - \mathcal{P}(\underline{\mathbf{h}}^{'})) + \varepsilon(\mathcal{D}(\underline{\mathbf{h}}) - \mathcal{D}(\underline{\mathbf{h}}^{'})) \tag{2.24}$$

Definição 11 *Seja* $\underline{\mathbf{h}}$ *e* $\underline{\mathbf{h}}^{'} \in \mathcal{H}$*, a multiplicação entre dois quatérnios duais é definida como,*

$$\begin{aligned} \underline{\mathbf{h}}\underline{\mathbf{h}}^{'} &= (\mathcal{P}(\underline{\mathbf{h}}) + \varepsilon\mathcal{D}(\underline{\mathbf{h}}))(\mathcal{P}(\underline{\mathbf{h}}^{'}) + \varepsilon\mathcal{D}(\underline{\mathbf{h}}^{'})) \\ &= \mathcal{P}(\underline{\mathbf{h}})\mathcal{P}(\underline{\mathbf{h}}^{'}) + \varepsilon(\mathcal{P}(\underline{\mathbf{h}})\mathcal{D}(\underline{\mathbf{h}}^{'}) + \mathcal{D}(\underline{\mathbf{h}})\mathcal{P}(\underline{\mathbf{h}}^{'})) \end{aligned} \tag{2.25}$$

Definição 12 *Seja* $\underline{\mathbf{h}} \in \mathcal{H}$*, a norma de um quatérnio dual é dada por*

$$||\underline{\mathbf{h}}|| = \sqrt{\underline{\mathbf{h}}\underline{\mathbf{h}}^{*}} = \sqrt{\underline{\mathbf{h}}^{*}\underline{\mathbf{h}}} \tag{2.26}$$

$$= \sqrt{||\mathcal{P}(\underline{\mathbf{h}})||^2 + 2\varepsilon vec_4(\mathcal{P}(\underline{\mathbf{h}})) \cdot vec_4(\mathcal{D}(\underline{\mathbf{h}}))} \tag{2.27}$$

Definição 13 *O conjugado de um quatérnio dual* $\underline{\mathbf{h}} \in \mathcal{H}$ *é dado por*

$$\underline{\mathbf{h}}^{*} = \mathcal{P}(\underline{\mathbf{h}}^{*} + \varepsilon\mathcal{D}(\underline{\mathbf{h}}^{*}) \tag{2.28}$$

Além das definições de quatérnios duais são definidos o operador vec_8 e os operadores de *Hamilton* ($\overset{+}{\mathbf{H}}_{\mathbf{8}}$ e $\overset{-}{\mathbf{H}}_{\mathbf{8}}$).

Definição 14 *Seja o quatérnio dual* $\underline{\mathbf{h}} = h_1 + \hat{i}h_2 + \hat{j}h_3 + \hat{k}h_4 + \varepsilon(h_5 + \hat{i}h_6 + \hat{j}h_7 + \hat{k}h_8) \in \mathcal{H}$*, operador* vec_8 *faz um mapeamento de* $\mathcal{H} \rightarrow \mathbb{R}^8$

$$vec_8(\underline{\mathbf{h}}) \triangleq [\ h_1\ h_2\ h_3\ h_4\ h_5\ h_6\ h_7\ h_8\ ]^T \tag{2.29}$$

desta forma o quatérnio dual é mapeado em uma forma vetorial. A operação inversa vec_8^{-1} *mapeia* $\mathbb{R}^8 \rightarrow \mathcal{H}$*.*

Definição 15 *Seja o quatérnio dual* $\underline{\mathbf{h}} = h_1 + \hat{i}h_2 + \hat{j}h_3 + \hat{k}h_4 + \varepsilon(h_5 + \hat{i}h_6 + \hat{j}h_7 + \hat{k}h_8)$, *os operadores de Hamilton* $\overset{+}{\mathbf{H}}_{\mathbf{8}}(\cdot)$ *e* $\overset{-}{\mathbf{H}}_{\mathbf{8}}(\cdot)$ *são definidos como:*

$$\overset{+}{\mathbf{H}}_8(\underline{\mathbf{h}}) = \begin{bmatrix} \overset{+}{\mathbf{H}}_4(\mathcal{P}(\underline{\mathbf{h}})) & \mathbf{0}_4 \\ \overset{+}{\mathbf{H}}_4(\mathcal{D}(\underline{\mathbf{h}})) & \overset{+}{\mathbf{H}}_4(\mathcal{P}(\underline{\mathbf{h}})) \end{bmatrix}_{8\times 8}, \quad \overset{-}{\mathbf{H}}_8(\underline{\mathbf{h}}) = \begin{bmatrix} \overset{-}{\mathbf{H}}_4(\mathcal{P}(\underline{\mathbf{h}})) & \mathbf{0}_4 \\ \overset{-}{\mathbf{H}}_4(\mathcal{D}(\underline{\mathbf{h}})) & \overset{-}{\mathbf{H}}_4(\mathcal{P}(\underline{\mathbf{h}})) \end{bmatrix}_{8\times 8} \tag{2.30}$$

com $\underline{\mathbf{h}}$ e $\underline{\mathbf{h}}'$ dois quatérnios duais, desta forma o quatérnio dual é mapeado em uma forma matricial. É possível demonstrar que:

$$vec_8(\mathbf{h}\mathbf{h}') = \overset{+}{\mathbf{H}}_8(\mathbf{h})vec_8(\mathbf{h}') = \overset{-}{\mathbf{H}}_8(\mathbf{h}')vec_8(\mathbf{h}) \tag{2.31}$$

Podemos definir uma outra relação para o conjugado de um quatérnio dual. Seja a matriz,

$$\mathbf{C}_8 = diag(1, -1, -1, -1, 1, -1, -1, -1) \tag{2.32}$$

utilizando 2.32 temos que:

$$vec_8(\underline{\mathbf{h}}^*) = \mathbf{C}_8 vec_8(\underline{\mathbf{h}}) \tag{2.33}$$

Definida a norma de um quatérnio dual em (2.26), um quatérnio dual unitário é aquele que possui norma igual a 1 e pode ser representado como:

$$\underline{\mathbf{h}} = \mathbf{r} + \varepsilon\frac{1}{2}\mathbf{p}\mathbf{r} \tag{2.34}$$

no qual $\mathbf{r} = cos(\frac{\phi}{2}) + sin(\frac{\phi}{2})\mathbf{n}$, $\mathbf{n} = \hat{i}n_x + \hat{j}n_y + \hat{k}n_z$ e $\mathbf{p} = 0 + \hat{i}p_x + \hat{j}p_y + \hat{k}p_z$. Se $\mathbf{n}$ for unitário o quatérnio $\mathbf{r}$ é unitário, por definição. O conceito de quatérnios unitários será muito útil para a representação de movimentos rígidos e possui definições importantes relacionadas.

2.4 Rotações e Translações usando Quatérnios

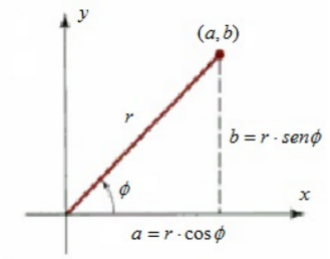


Figura 2.1: Representação de um número complexo

Seja $z \in \mathbb{C}$ um número complexo não nulo ,tal que , $z = a + ib$, representado em um sistema de coordenadas $\mathcal{F}$, o qual pode ser descrito na forma polar

$$z = r(cos(\phi) + isen(\phi)) \tag{2.35}$$

sendo que, z representa uma rotação de um ângulo ϕ, em torno do eixo z do sistema de coordenadas $\mathcal{F}$, como mostra a Figura 2.1, $r = ||z|| = \sqrt{a^2 + b^2}$ é o comprimento do vetor z e $\phi = tan^{-1}(\frac{b}{a})$ é o ângulo de rotação em relação ao eixo de coordenadas. Esse conceito pode ser expandido para uma representação tridimensional usando quatérnios de norma unitária. Seja um quatérnio $\mathbf{r}$ tal que

$$\mathbf{r} = cos(\frac{\phi}{2}) + sen(\frac{\phi}{2})\mathbf{n} \tag{2.36}$$

Se $\mathbf{n} = \hat{i}n_x + \hat{j}n_y + \hat{k}n_z$ for unitário, o quatérnio em (2.36) possui norma unitária e, consequentemente, representa uma rotação. Pontos podem ser representados por quatérnios puros (de parte real nula) como o quatérnio

$$\mathbf{p} = 0 + \hat{i}p_x + \hat{j}p_y + \hat{k}p_z \tag{2.37}$$

A operação de rotação utilizando quatérnios é dado por:

$$\mathbf{p_1} = \mathbf{rpr}^* \tag{2.38}$$

com $\mathbf{r}$ dado por 2.36, e o ponto $\mathbf{p}_1$ é o ponto depois de realizada a rotação em torno do quatérnio unitário $\mathbf{r}$.

2.5 Movimento Rígido Completo Usando Quatérnios Duais

Utilizando quatérnios duais unitários é possível representar rotação e translação em uma única estrutura que representa todo o movimento rígido no espaço (Adorno (2011)).

$$\underline{\mathbf{h}}_1^0 = \mathbf{r}_1^0 + \varepsilon\frac{1}{2}\mathbf{p}_1^0\mathbf{r}_1^0 \tag{2.39}$$

O quatérnio dual unitário $\underline{\mathbf{h}}_1^0$, é análogo à matriz de transformação homogênea, e também representa um movimento no espaço, sendo este quatérnio dual referido como posição dual. Na parte primária, $\mathcal{P}(\underline{\mathbf{h}}_1^0)$, está a rotação do sistema de coordenadas $\mathcal{F}_0$ para o sistemas de coordenadas $\mathcal{F}_1$. Na parte dual $\mathcal{D}(\underline{\mathbf{h}}_1^0)$, contém a translação de $\mathcal{F}_1$ em relação à $\mathcal{F}_0$, sendo esta dada pela operação:

$$\mathbf{p}_1^0 = 2\mathcal{D}(\underline{\mathbf{h}}_1^0)\mathcal{P}(\underline{\mathbf{h}}_1^0)^* \tag{2.40}$$

onde, $\mathcal{P}(\underline{\mathbf{h}}_1^0)^*$ é o conjugado da parte primária. Dadas essas definições, a sequência de movimentos de $\mathcal{F}_0$ para $\mathcal{F}_1$, de $\mathcal{F}_1$ para $\mathcal{F}_2$ pode ser representada, Fig. 2.2, por uma multiplicação de quatérnios duais unitários.

$$\underline{\mathbf{h}}_2^0 = \underline{\mathbf{h}}_1^0\underline{\mathbf{h}}_2^1 \tag{2.41}$$

Se há um número n de transformações a equação (2.39) passa a ser:

$$\underline{\mathbf{h}}_n^0 = \underline{\mathbf{h}}_1^0\underline{\mathbf{h}}_2^1...\underline{\mathbf{h}}_n^{n-1} \tag{2.42}$$

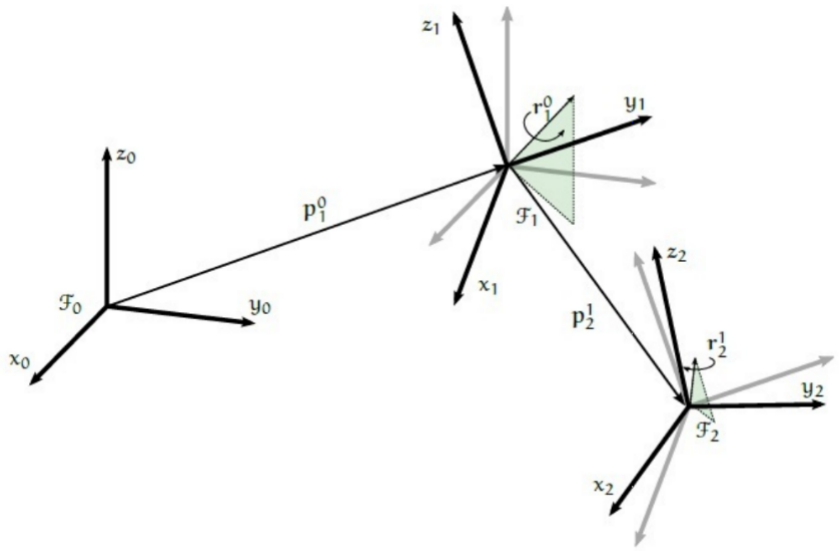


Figura 2.2: Sequências de Movimentos Rígidos (adaptado de Adorno (2011)))

2.6 Modelo Cinemático Direto usando Quatérnios Duais

Esta seção apresenta a metodologia proposta por Adorno (2011) para obter o modelo cinemático direto usando a álgebra dos quatérnios duais. A equação (2.42) mostra uma sequência de quatérnios duais multiplicados , o qual define a cinemática direta de um robô. Esta cinemática pode ser utilizada em conjunto com os parâmetros de *Denavit-Hartenberg*. A notação de *Denavit-Hartenberg* padrão utilizando quatérnios duais fica definida como

$$\mathbf{h}_{DH} = \mathbf{r}_{z,\theta}\mathbf{p}_{z,d}\mathbf{x}_{x,a}\mathbf{p}_{x,\alpha} \tag{2.43}$$

$\mathbf{r}_{z,\theta}$ e $\mathbf{r}_{x,\alpha}$ representam rotações puras de θ e α em torno de dos eixos z e x, respectivamente, e $\mathbf{p}_{z,d}$ e $\mathbf{p}_{x,a}$ representam translações puras de d e a ao longo de z e x, respectivamente. Resolvendo a equação 2.43, pode se obter a seguinte expressão:

$$\underline{\mathbf{h}}_{\mathbf{DH}} = \mathcal{P}(\underline{\mathbf{h}}_{DH}) + \varepsilon\mathcal{D}(\underline{\mathbf{h}}_{DH}) \tag{2.44}$$

onde,

$$\mathcal{P}(\underline{\mathbf{h}}_{DH}) = h_{dh1} + \hat{i}h_{dh2} + \hat{j}h_{dh3} + \hat{k}h_{dh4} \tag{2.45}$$

$$\mathcal{D}(\underline{\mathbf{h}}_{DH}) = \frac{-(dh_{dh4} + ah_{dh2})}{2} + \hat{i}\frac{(-dh_{dh3} + ah_{dh1})}{2} + \hat{j}\frac{(dh_{dh2} + ah_{dh4})}{2} + \hat{k}\frac{(dh_{dh1} - ah_{dh3})}{2} \tag{2.46}$$

sendo,

$$h_{dh1} = cos(\theta/2)cos(\alpha/2) \tag{2.47}$$

$$h_{dh2} = cos(\theta/2)sen(\alpha/2) \tag{2.48}$$

$$h_{dh3} = sen(\theta/2)sen(\alpha/2) \tag{2.49}$$

$$h_{dh4} = sen(\theta/2)cos(\alpha/2) \tag{2.50}$$

A notação de Denavit-Hartenberg modificada também pode ser escrita usando quatérnios duais , sendo ela demostrada em Adorno (2011).

2.7 Jacobiano Analítico usando Quatérnios Duais

Seja $\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{0}}$ a posição dual de um robô serial de n juntas, em relação a uma base fixa, $\mathcal{F}_0$. Pela cinemática diferencial temos (Adorno (2011))

$$\dot{\underline{\mathbf{h}}}_n^0 = J_{\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{0}}}\dot{\theta} \tag{2.51}$$

onde $\dot{\theta}$ é o vetor de velocidade das juntas e $J_{\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{0}}}$ é o Jacobiano analítico. Se $\underline{\mathbf{h}}_n^0$ é uma sequência de transformações representadas por quatérnios duais, sua derivada $\dot{\underline{\mathbf{h}}}_n^0$ pode ser representada por

$$\dot{\underline{\mathbf{h}}}_n^0 = \sum_{i=0}^{n-1} \underline{\mathbf{h}}_{\mathbf{i}}^{\mathbf{0}}\dot{\underline{\mathbf{h}}}_{i+1}^{i}\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{i+1}} \tag{2.52}$$

Seja um quatérnio dual $\underline{\mathbf{w}}_{i+1}^{i}$, que satisfaz a seguinte relação, $\dot{\underline{\mathbf{h}}}_{i+1}^{i} = \frac{1}{2}\underline{\mathbf{w}}_{i+1}^{i}\underline{\mathbf{h}}_{i+1}^{i}$, logo:

$$\begin{aligned}
\dot{\underline{\mathbf{h}}}_n^0 &= \sum_{i=0}^{n-1} \underline{\mathbf{h}}_{\mathbf{i}}^{\mathbf{0}}(\frac{1}{2}\underline{\mathbf{w}}_{i+1}^{i}\underline{\mathbf{h}}_{i+1}^{i})\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{i+1}} \qquad (2.53)\\
&= \frac{1}{2}\sum_{i=0}^{n-1} \underline{\mathbf{h}}_{\mathbf{i}}^{\mathbf{0}}\underline{\mathbf{w}}_{i+1}^{i}\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{i}}\\
&= \frac{1}{2}\sum_{i=0}^{n-1} \underline{\mathbf{h}}_{\mathbf{i}}^{\mathbf{0}}\underline{\mathbf{w}}_{i+1}^{i}(\underline{\mathbf{h}}_{\mathbf{i}}^{\mathbf{0}})^{*}\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{0}} \qquad (2.54)
\end{aligned}$$

$\dot{\underline{\mathbf{h}}}_{i+1}^{i}$ está em função de θ_{i+1}, consequentemente:

$$\begin{aligned}
\underline{\mathbf{w}}_{i+1}^{i} &= 2\dot{\underline{\mathbf{h}}}_{i+1}^{i}(\underline{\mathbf{h}}_{i+1}^{i})^{*}\\
&= 2\frac{\partial\dot{\underline{\mathbf{h}}}_{i+1}^{i}}{\partial\dot{\theta}_{i+1}}(\underline{\mathbf{h}}_{i+1}^{i})^{*}\dot{\theta}_{i+1}\\
&= \underline{\mathbf{w}}_{i+1}^{i}\dot{\theta}_{i+1} \qquad (2.55)
\end{aligned}$$

seja

$$\underline{\mathbf{z}}_i\dot{\theta}_{i+1} = \frac{1}{2}\underline{\mathbf{h}}_i^0\underline{\mathbf{w}}_{i+1}^{i}(\underline{\mathbf{h}}_i^0)^{*} \tag{2.56}$$

então

$$\dot{\underline{\mathbf{h}}}_n^0 = \sum_{i=0}^{n-1} \underline{\mathbf{z}}_{\mathbf{i}}\underline{\mathbf{h}}_n^0\dot{\theta}_{i+1} \tag{2.57}$$

Sendo $vec_8(\dot{\underline{\mathbf{h}}}_n^0) = J_{\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{0}}}\dot{\theta}_{i+1}$, e $J_{\underline{\mathbf{h}}_{\mathbf{n}}^{\mathbf{0}}} = [j_1...j_n]^T$, onde j_i é um vetor coluna. De 2.57:

$$j_{i+1} = vec(\underline{\mathbf{z}}_{\mathbf{i}}\underline{\mathbf{h}}_n^0),\ i = 0, ..., n-1 \tag{2.58}$$

onde $\underline{\mathbf{h}}_n^0$ é obtido usando a cinemática direta e representa a posição dual final do atuador. As colunas da matriz de $J_{\underline{\mathbf{h}}_\mathbf{n}^\mathbf{0}}$ dependem de $\underline{\mathbf{z}}_\mathbf{i}$, que é uma variável dependente da parametrização utilizada para representar o modelo cinemático do robô. Usando a notação de Denavit-Hartenberg padrão , Eq. (2.43), obtêm-se (Adorno (2011)):

$$\begin{aligned}\mathcal{P}(\underline{\mathbf{z}}_\mathbf{i}) &= \hat{i}(h_{i2}h_{i4} + h_{i1}h_{i3}) \\ &\quad +\hat{j}(h_{i3}h_{i4} - h_{i1}h_{i2}) \\ &\quad +\hat{k}(\frac{h_{i4}^2 - h_{i3}^2 - h_{i2}^2 + h_{i1}^2)}{2}\end{aligned} \tag{2.59}$$

$$\begin{aligned}\mathcal{D}(\underline{\mathbf{z}}_\mathbf{i}) &= \hat{i}(h_{i2}h_{i8} + h_{i6}h_{i4} + h_{i1}h_{i7} + h_{i5}h_{i3}) \\ &\quad +\hat{j}(h_{i3}h_{i8} + h_{i7}h_{i4} + h_{i1}h_{i6} + h_{i5}h_{i2}) \\ &\quad +\hat{k}(h_{i4}h_{i8} + h_{i3}h_{i7} + h_{i2}h_{i6} + h_{i1}h_{i5})\end{aligned} \tag{2.60}$$

onde $h_{i1}...h_{i8}$ são coeficientes de $\underline{\mathbf{h}}_n^0$. Resolvendo a equação (2.58) obtém-se o Jacobiano analítico de posição dual $J_{\underline{\mathbf{h}}_\mathbf{n}^\mathbf{0}}$. Um pseudo código é mostrado abaixo para calcular o Jacobiano de um quatérnio dual.

Algorithm 1 Jacobiano de um quatérnio dual

1: *Calcular* $\underline{\mathbf{h}}_n^0$
2: $\underline{\mathbf{h}} \leftarrow 1$
3: **for** $i = 0$ **to** $n - 1$ **do**
4: *Calcular* $\underline{\mathbf{z}}$
5: $j_{i+1} \leftarrow vec_8(\underline{\mathbf{z}}\underline{\mathbf{h}}_n^0)$
6: $\underline{\mathbf{h}} \leftarrow vec(\underline{\mathbf{h}}\underline{\mathbf{h}}_{i+1}^i)$
7: **end for**

2.8 Conclusões Parciais

Este capítulo mostrou as principais definições dos quatérnios, números duais e quatérnios duais. Foi observado como representar rotações e translações usando quatérnios duais, bem como a cinemático direta de um robô serial. Também foi demostrado como obter o Jacobiano Analítico que será utilizado para implementar os controladores cinemáticos.

3 Modelo 3D Dual-SLIP

3.1 Introdução

O modelo 3D de pêndulo invertido com massa e mola dupla (*3D Dual-SLIP, Dual Spring-Load Inverted Pendulum*) pode ser usado, com a parametrização adequada, para reproduzir as características principais da caminhada dinâmica de uma pessoa sem limitações físicas. Este modelo apresenta um melhor comportamento do que o modelo de pêndulo invertido (*LIPM, Linear Inverted Pendulum Model*), como abordado em Liu et al. (2015). Desta forma, o modelo *3D Dual-SLIP* pode ser usado como um modelo alternativo simplificado para geração de padrões de caminhadas dinamicamente estáveis. O modelo permite capturar diferentes comportamentos da dinâmica, quando uma ou duas pernas estão em contato com o chão, fazendo o suporte para a caminhada. Neste capítulo é mostrado como usar o modelo *3D Dual-SLIP* para gerar padrões de caminhadas sobre um terreno plano, utilizando uma otimização de um quarto de passo para encontrar os parâmetros adequados para o modelo, como mostrado em Liu et al. (2015).

3.2 3D Dual-SLIP

O modelo *3D Dual-SLIP* e seus parâmetros são mostrados na Fig 3.1. O modelo consite em um pêndulo invertido com massa e mola dupla. Durante a fase de suporte de uma perna a posição do pé de contato é dado por $\mathbf{p_f} = [x_f, y_f, z_f]^T \in \mathbb{R}$, e a dinâmica do centro de massa (CoM) é dada pela equação abaixo:

$$m\ddot{\mathbf{p}}_\mathbf{c} = m\mathbf{g} + k(l - ||\mathbf{l}||)\hat{\mathbf{l}}, \tag{3.1}$$

no qual, m representa a massa $\in \mathbb{R}$, $\mathbf{p_c} = [x_c, y_c, z_c]^T \in \mathbb{R}^3$, é a posição do centro de massa (CoM), k $\in \mathbb{R}$ é a constante de mola, $\mathbf{l} = \mathbf{p_c} - \mathbf{p_f}$ é o vetor sobre a perna, $\hat{\mathbf{l}} \in \mathbb{R}^3$ é o correspondente vetor unitário e $l \in \mathbb{R}$ é o tamanho da perna em descanso. Durante o suporte com as duas pernas, as equações dinâmicas são:

$$m\ddot{\mathbf{p}}_\mathbf{c} = m\mathbf{g} + \sum_{i\in\{A,B\}}^{n} k_i(l_i - ||\mathbf{l_i}||)\hat{\mathbf{l}}_\mathbf{i}, \tag{3.2}$$

no qual, {A,B} é usado para identificar a perna que está em contato. A Fig. 3.1 mostra um frame em um determinado instante de tempo do modelo movimentando na direção +x, m representa uma massa pontual, com a posição de centro de massa dada pelas coordenadas de $\mathbf{p_c} = [x_c, y_c, z_c]^T$. A posição da perna de balanço é determinado pelos ângulos θ, ângulo entre a perna em movimento e o eixo vertical z, e o ângulo ϕ, ângulo entre a projeção

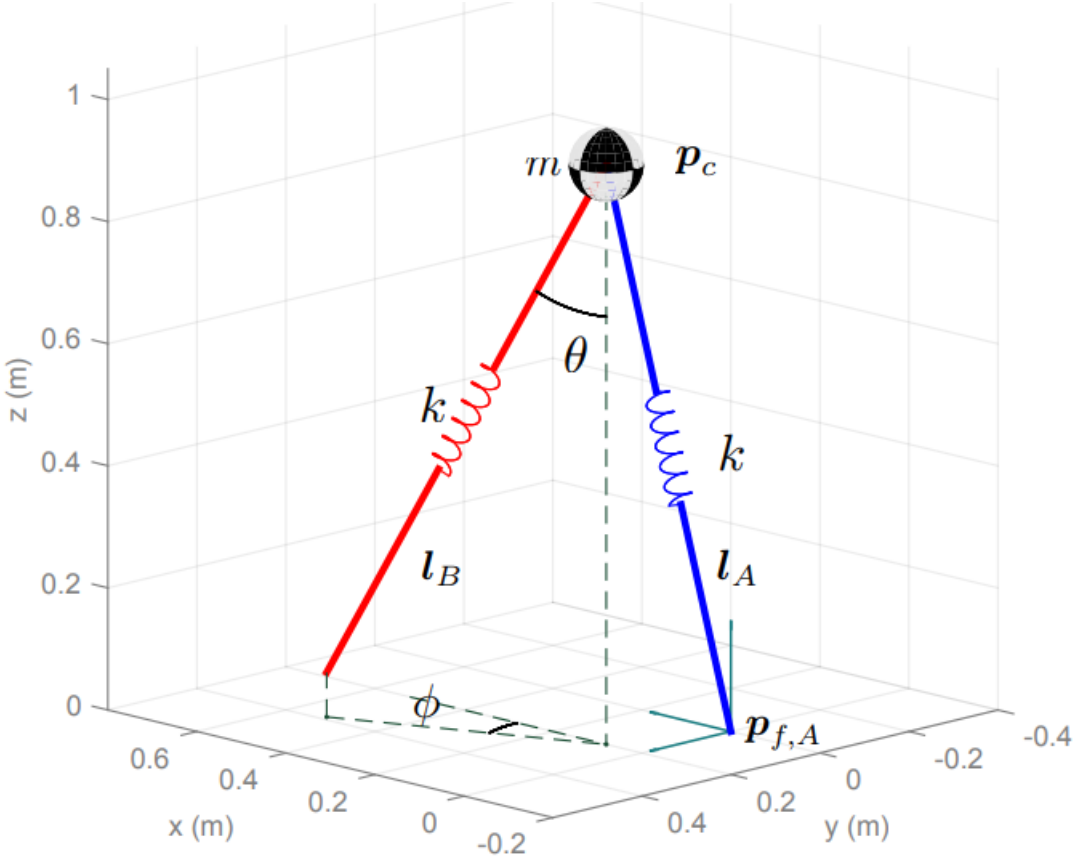


Figura 3.1: Modelo *3D DUAL-SLIP* e seus parâmetros.

da perna em movimento no plano xy e o eixo x (na direção positiva). A posição do pé é denotada por p_i^f, $i = \{A, B\}$. O tamanho da perna quando está em descanso é l_0. O tamanho da perna i quando está em movimento é $||l_i|| = l_0$, e $||l_i|| < l_0$ quando a perna i está no suporte (em contato no chão). A Fig. 3.2 mostra a trajetória do CoM de um ciclo completo (dois passos consecutivos) de uma caminhada. Um passo é definido como dois estados intermediários subsequentes, sendo este estado intermediário definido como *midstance* (MS). O estado intermediário ocorre durante a fase de contato único de uma perna quando $\dot{z} = 0$. Considerando um passo , começando em MS, o centro de massa do modelo *3D Dual-SLIP* terá outros três eventos: primeiro a perna em movimento encosta no chão, definido como *touchdown* (TD), depois o centro de massa chega a posição mais baixa de sua trajetória, definido como *lowest height* (LH) e por último a perna que estava em suporte tira o contato com o chão, definido como *lifts off* (LO). Seguindo LO , o modelo continua até chegar a posição de MS terminando um passo completo. Para esta caminhada, a força de reação vertical do solo ($vGRF$) tem a forma com dois picos, como mostrado na Fig. 3.3. Esta dinâmica de caminhada pode ser entendida como:

$$S_{TD} = \{(\mathbf{p_c}, \dot{\mathbf{p}}_\mathbf{c}) |\ \mathbf{e}_z^T \mathbf{l_A} = L_{TD} cos\theta\} \tag{3.3}$$

$$S_{LO} = \{(\mathbf{p_c}, \dot{\mathbf{p}}_\mathbf{c}) |\ ||\mathbf{l_A}|| = L_{LO}\} \tag{3.4}$$

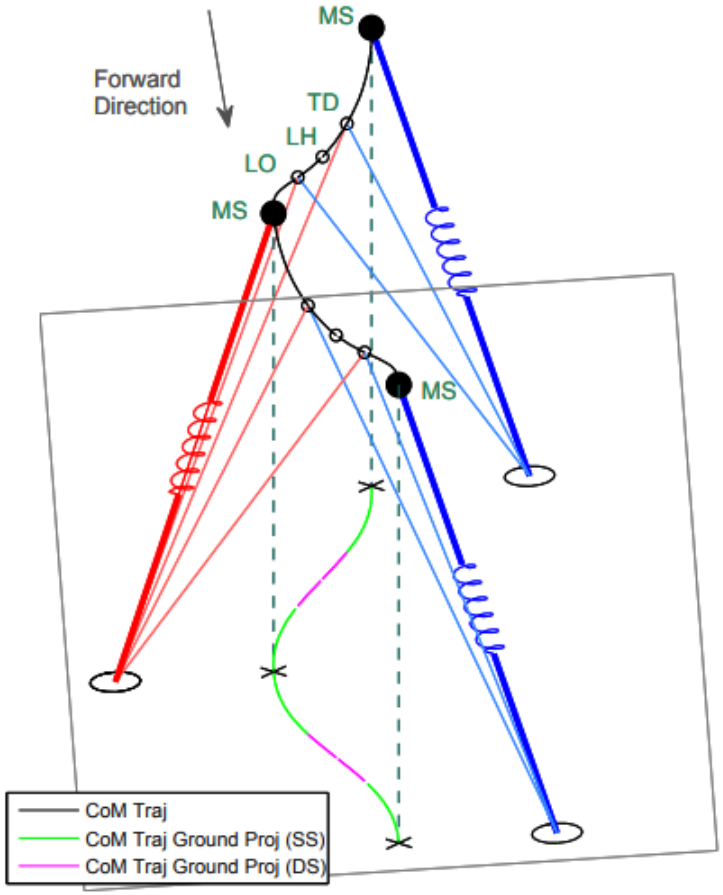


Figura 3.2: Trajetória do centro de massa em um ciclo completo (2 passos) do *modelo 3D Dual-SLIP*. A projeção da trajetória do centro de massa no solo é mostrada nas fases da caminhada, suporte simples (SS) e no suporte duplo (DS).

no qual , $\mathbf{e}_z = [0, 0, 1]^T$ é o vetor unitário na direção vertical. $\theta \in \mathbb{R}$ é o ângulo da posição de *touchdown*. L_{TD} e L_{LO} são o tamanho da perna na posição de *touchdown* e *liftoff* respectivamente, onde,

$$L_{TD} = L_{LO} = l_A = l_B = constante \tag{3.5}$$

A posição da perna em movimento (perna B neste caso) pode ser controlada com os ângulos frontal e lateral ($\theta, \phi \in \mathbb{R}$ como mostrado na Fig 3.1):

$$\mathbf{p}_{f,B} = \mathbf{p}_c + L_{TD} \begin{bmatrix} sin\theta cos\phi \\ sin\theta sin\phi \\ -cos\theta \end{bmatrix} \tag{3.6}$$

Supondo que a perna A começa com o suporte inicial, o estado de MS ocorre quando:

$$S_{MS} = \{(\mathbf{p}_c, \dot{\mathbf{p}}_c) | \dot{z}_c = 0, z_c > z_{TH}, ||\mathbf{l}_A|| < \mathbf{l}_0\}, \tag{3.7}$$

onde $z_{TH} = l_0 cos\theta$, no qual é um valor limite para a altura do CoM, e depende do ângulo frontal θ. Os estados subsequentes são descritos abaixo:

$$S_{TD} = \{(\mathbf{p}_c, \dot{\mathbf{p}}_c) | \dot{z}_c < 0, z_c = z_{TH}\}, \tag{3.8}$$

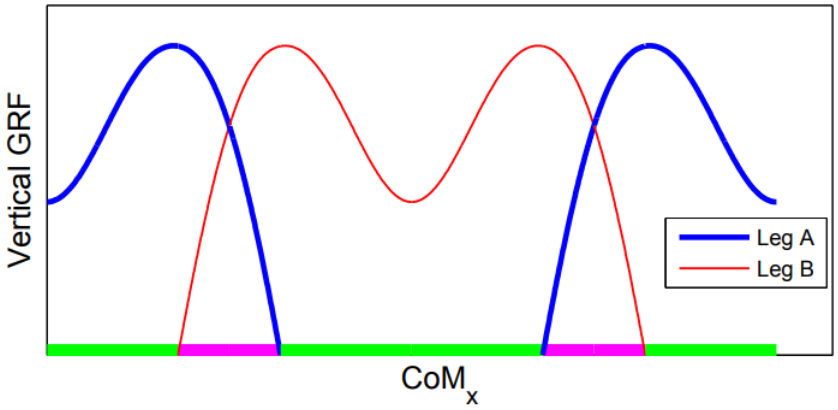


Figura 3.3: O padrão nominal da força de reação vertical do solo (vGRF) para cada perna de uma caminhada semelhante à humana, como mostrado na Fig 3.2, as fases de suporte simples (SS) e duplo (DS) são marcadas com o mesmo esquema de cores da Fig 3.2.

$$S_{LH} = \{(\mathbf{p}_c, \dot{\mathbf{p}}_c) | \dot{z}_c = 0, z_c < z_{TH}\}, \tag{3.9}$$

$$S_{LO} = \{(\mathbf{p}_c, \dot{\mathbf{p}}_c) | \dot{z}_c > 0, ||\mathbf{l}_A|| = \mathbf{l}_0\}, \tag{3.10}$$

3.3 Trajetórias utilizando o modelo *3D Dual-SLIP*

Esta seção apresenta a metodologia proposta por Liu et al. (2015) para encontrar trajetórias periódicas utilizando o modelo *3D Dual-SLIP*. Devido ao maior espaço de estados e parâmetros, não é viável uma abordagem de mapeamento exaustivo utilizando o modelo *3D Dual-SLIP*. Sendo assim, foi proposto encontrar trajetórias periódicas através de uma abordagem de otimização não linear, que inclui simulações em uma etapa na rotina de avaliação. O estado MS discreto $\mathbf{x}$, é escolhido como:

$$\mathbf{x} = [\ (x_c - x_f)\ \ (y_c - y_f)\ \ (z_c - z_f)\ \ \dot{x}_c\ \ \dot{y}_c\ ]^T \ \in\ \mathbb{R}^5 \tag{3.11}$$

Sendo esta uma parte do estado do modelo em relação à posição atual . Um vetor de controle discreto, denotado como $\mathbf{u}$, também descrito,

$$\mathbf{u} = [\phi, \theta, k]^T \tag{3.12}$$

no qual, $\mathbf{u}$ é um vetor que contem os ângulos de *Touchdown* (TD) e a constante de rigidez k, Fig. 3.1. Liu et al. (2015) cita que uma marcha típica humana apresenta um pico duplo de vGRF padrão, Fig. 3.3, sendo este, um dos muitos modos periódicos possíveis a ser capturado pelo modelo *3D Dual-SLIP*. Muitos padrões de caminhadas periódicas resultantes de 1 passo de otimização, ou seja, de MS para MS, exibem características

não humanas, com 3 ou mais picos de $vGRF$. Devido a existência de um grande número de mínimos locais, a otimização não-linear de um passo, é muito sensível as entradas. A natureza híbrida da dinâmica também aumenta a falta de suavidade do problema de otimização e, em alguns casos, o processo de otimização não consegue convergir. A Fig. 3.4 mostra a projeção do plano sagital de uma trajetória de CoM referente a meio passo com o CoM posição de LH para ser exatamente acima do ponto médio entre os dois pés de apoio. Sendo assim para melhorar o comportamento do otimizador, um método mais simples é proposto (meio-passo, do MS para o LH, como mostrado na Fig. 3.4). Dada uma velocidade desejada em MS de x_0, um problema de otimização de meio-passo pode ser formulado para encontrar uma trajetória de CoM de MS para LH, para que a projeção do solo do CoM em LH seja diretamente entre os dois pés de apoio.

$$(\mathbf{p_c} - \frac{\mathbf{p_{f,A}} - \mathbf{p_{f,B}}}{2})^T.\dot{\mathbf{p}}_\mathbf{c}(t_{LH}) = 0 \tag{3.13}$$

Supondo que a perna A está em suporte inicial, na posição de MS, a otimização para o problema pode ser escrito como:

$$\begin{aligned}\min_{\phi,\theta,k,z_0} \|\frac{1}{2}(x_A^{\mathrm{f}} + x_B^{\mathrm{f}}(\mathbf{x}_0,\ \mathbf{u}_0)) - x_c(\mathbf{x}_0,\ \mathbf{u}_0,\ t_{LH})\|^2 \\ +\|\frac{1}{2}(y_A^{\mathrm{f}} + y_B^{\mathrm{f}}(\mathbf{x}_0,\ \mathbf{u}_0)) - y_c(\mathbf{x}_0,\ \mathbf{u}_0,\ t_{LH})\|^2\end{aligned} \tag{3.14}$$

com :

$$\begin{aligned}\mathbf{x}_0 &= [x_{0,d} \quad y_{0,d} \quad z_0 \quad \dot{x}_{0,d} \quad \dot{y}_{0,d}]^T, \\ \mathbf{u}_0 &= [\phi\ \theta\ k]^T, \\ \mathbf{x}_{0,d} &= 0, \dot{y}_{0,d} = 0,\end{aligned}$$

no qual, (x_A^f, y_A^f) é a posição inicial do pé de apoio em MS ; (x_B^f, y_B^f) é a posição do pé em movimento em TD, que é parametrizada pela posição inicial, par $(\mathbf{x}_0, \mathbf{u}_0)$; $(x_c(t), y_c(t))$ é a projeção do solo do CoM no tempo t, que também é parametrizado pelo par $(\mathbf{x}_0, \mathbf{u}_0)$. Sendo que, ambos (x_B^f, y_B^f) e $(x_c(t), y_c(t))$, são avaliados através de simulação dinâmica. Para o estado inicial de MS, é necessário que o deslocamento na direção para frente (relativo ao pé de apoio) e a velocidade na direção lateral ambos sejam igual a zero. O deslocamento lateral, $y_{0,d}$, é pré-selecionado para induzir uma largura do passo semelhante à humana. Para uma perna de 1 m comprimento, $y_{0,d} = 0,05\ m$ é um valor adequado. O deslocamento vertical no MS inicial , $z0$, e o vetor de controle $\mathbf{u}_0$, são deixados a para serem determinados pelo otimizador. Embora a Eq. 3.14 otimiza apenas para meio passo, o par $(\mathbf{x}_0, \mathbf{u}_0)$, que resulta em um erro residual zero na função de custo é uma condição suficiente para obter trajetórias para o CoM, com o modelo *3D Dual-SLIP*, que são simétricas sobre a posição LH, para mais de um passo. Essa simetria esquerda-direita, por sua vez, garante uma marcha periódica de 2 dois passos. Outros parâmetros podem ser adicionados a função objetivo para refinar a marcha, como o comprimento do passo desejado ou pode adicionar ainda uma razão para a fase de suporte duplo desejada.

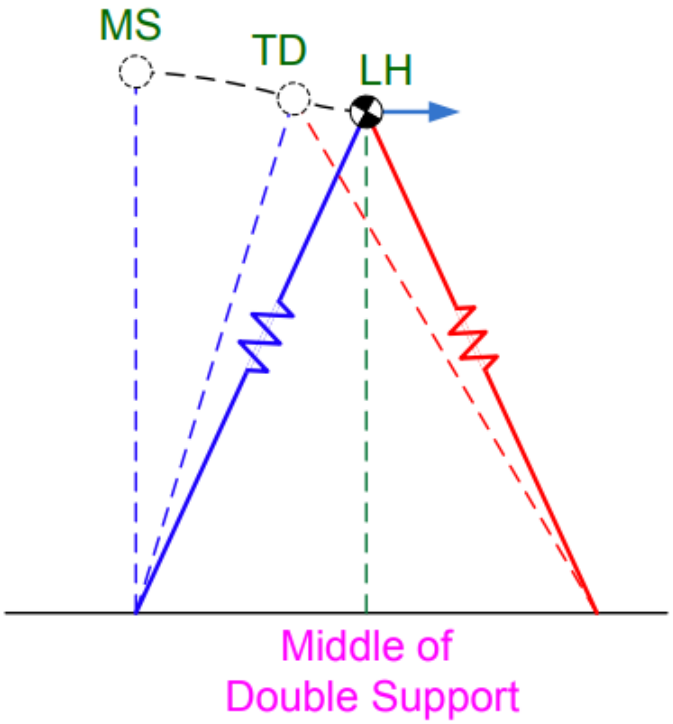


Figura 3.4: (Fig 3 em Liu et al. (2015)) Projeção do plano sagital de uma trajetória de CoM em meio passo com o CoM posição em LH para ser exatamente acima do ponto médio entre os dois pés de apoio.

3.4 Modelo *3D Dual-SLIP* Melhorado

A otimização proposta na seção 3.3 foi testada em um modelo *3D Dual-SLIP* com uma massa corporal de $m = 80\ kg$ e comprimento da perna de apoio $l = 1,0\ m$. Esses números aproximadamente coincidem com a massa e altura do CoM de um adulto humano. Com estes parâmetros a otimização do problema, descrito pela Eq. 3.14, é capaz de encontrar movimentos de caminhada periódica com velocidades desejadas, entre aproximadamente $0,7\ m/s$ e $1,3\ m/s$, como mostrado em Liu et al. (2015). De modo a melhorar o modelo, é adicionado uma estrategia de atuação durante as fases de suporte simples e duplo do modelo. Desta forma, uma nova variável de controle β_{SS}, é adicionada no método de otimização. β_{SS} determina a taxa na qual o comprimento da perna de apoio aumenta ou diminui durante a fase de suporte único e suporte duplo, como mostrado na Fig. 3.4. Durante a primeira fase de suporte único a partir do MS, o comprimento da perna A aumenta linearmente com uma taxa β_{SS} até a perna B alcançar a posição de *touchdown*. O comprimento da perna B na posição de TD é o comprimento da perna em repouso, e ambas as pernas durante a fase de apoio duplo são mantidos constantes até que a perna A levante. Na segunda fase de apoio individual, o comprimento da perna B diminui linearmente com taxa β_{SS} até o próximo MS. A primeira equação dinâmica para

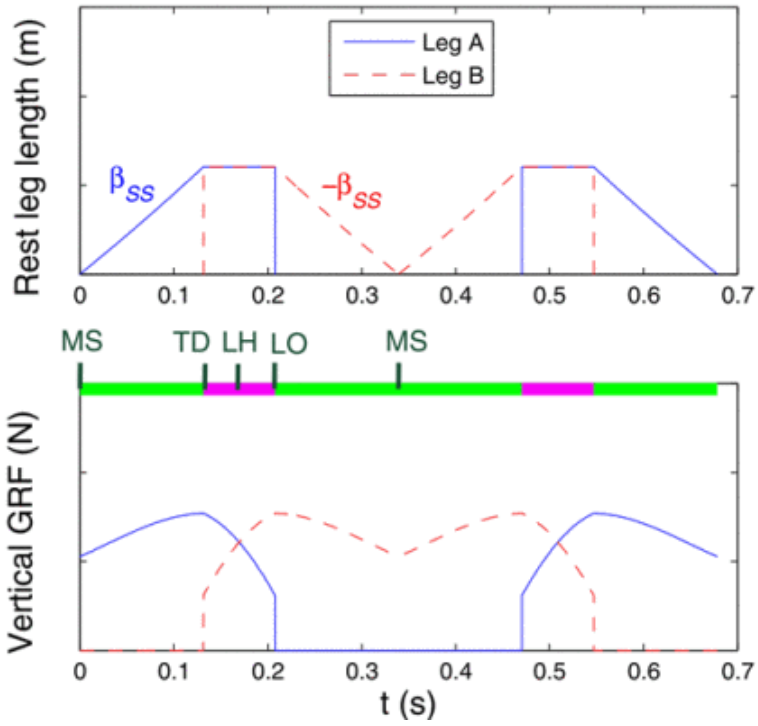


Figura 3.5: (Fig 5 em Liu et al. (2015) adaptada) A evolução do comprimento da perna ao longo do tempo para 2 passos e o padrão GRF vertical correspondente. As curvas são geradas para um velocidade de caminhada de 1,5 m /s.

o suporte único adicionando a variável β_{SS}:

$$m\ddot{\mathbf{p}}_\mathbf{c} = m\mathbf{g} + k(l + \beta_{SS}.t - ||\mathbf{l}||)\hat{\mathbf{l}}, \tag{3.15}$$

a dinâmica da fase de suporte duplo:

$$m\ddot{\mathbf{p}}_\mathbf{c} = m\mathbf{g} + \sum_{i\in\{A,B\}}^{n} k_i(l_i + \beta_{SS}.t_{TD} - ||\mathbf{l_i}||)\hat{\mathbf{l}}_\mathbf{i}, \tag{3.16}$$

e a segunda dinâmica de fase de suporte única é:

$$m\ddot{\mathbf{p}}_\mathbf{c} = m\mathbf{g} + \sum_{i\in\{A,B\}}^{n} k_i(l_i + \beta_{SS}.t_{TD} - \beta(t - t_{LO}) - ||\mathbf{l_i}||)\hat{\mathbf{l}}_\mathbf{i}, \tag{3.17}$$

no qual, t é redefinido para zero no próximo estado de MS . Supondo que a perna A está em suporte no MS inicial, a otimização problema pode ser escrito como:

$$\begin{aligned}\min_{\phi,\theta,k,\beta_{SS}} \|\frac{1}{2}(x_A^\mathrm{f} + x_B^\mathrm{f}(\mathbf{x}_0,\, \mathbf{u}_0)) - x_c(\mathbf{x}_0,\, \mathbf{u}_0,\, t_{LH})\|^2 \\ +\|\frac{1}{2}(y_A^\mathrm{f} + y_B^\mathrm{f}(\mathbf{x}_0,\, \mathbf{u}_0)) - y_c(\mathbf{x}_0,\, \mathbf{u}_0,\, t_{LH})\|^2\end{aligned} \tag{3.18}$$

com :

$$\begin{aligned}\mathbf{x}_0 &= [x_{0,d} \quad y_{0,d} \quad z_0 \quad \dot{x}_{0,d} \quad \dot{y}_{0,d}]^T, \\ \mathbf{u}_0 &= [\phi \;\; \theta \;\; k \;\; \beta_{SS}]^T, \\ \mathbf{x}_{0,d} &= 0, \\ \dot{y}_{0,d} &= 0\end{aligned}$$

A altura inicial z_0 não é mais uma variável de otimização, mas em vez disso, é fixado em um valor $z_{0,d}$. Para o comprimento da perna de descanso = 1 m, $z_{0,d}$; é definido como $0,96m$. Isso ocorre porque o intervalo possível para z_0 é muito pequeno (apenas um poucos centímetros) e como mostrado em Liu et al. (2015), se z_0 é deixado como uma variável livre, o otimizador tende a empurrá-lo para o limite superior a medida que a velocidade aumenta, o que resulta em uma marcha de andar indesejável.

3.5 Variando os parâmetros do modelo 3D Dual-SLIP

A Fig. 3.6 mostra a coleta de resultados para a otimização (Eq. 3.18) variando a velocidade de caminhada desejada, a qual, varia entre 1 m/s e 2 m/s. Estas curvas podem ser utilizadas para obter diferentes trajetórias para uma dada velocidade desejada e pode ser usada para ajustar a velocidade de deslocamento do modelo 3D Dual SLIP de forma dinâmica. É mostrado na próxima seção, como pode ser implementado um controlador para o modelo *3D Dual-SLIP*. Com isso, podemos utilizar o modelo, alterando sua velocidade com o tempo, e estender para uma quantidade maior de passos para o modelo, incorporando a dinâmica do modelo em toda a trajetória, conforme mostrado em Liu et al. (2015).

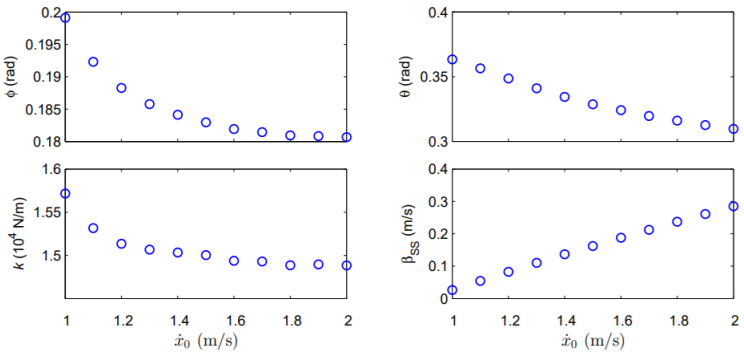


Figura 3.6: Os resultados de otimização para caminhada usando o método de otimização Eq. 3.18 e variando a velocidade de avanço desejada.

3.6 Controlador LQR para o modelo 3D Dual-SLIP

Nesta seção é demostrado a metodologia proposta por Liu et al. (2015), para implementar um controlador LQR para o modelo *3D Dual-SLIP*. Como o modelo *3D Dual-SLIP* não é estável para mais de 2 passos, é necessário implementar um controlador para regular o estado do modelo, deixando-o mais robusto, e desta forma, sendo possível utilizar este durante toda a caminhada, incorporando sua dinâmica. Supondo que identifiquemos

um par $(\mathbf{x}_0^*, \mathbf{u}_0^*)$, que consegue produzir uma caminhada periódica, utilizando o modelo *3D Dual-SLIP* convencional. É possível verificar a simetria esquerda-direita, para uma marcha periódica de 2 passos, o estado MS qualquer é dado por :

$$\mathbf{x}_n^* = \mathbf{A}^n \mathbf{x}_0^* \tag{3.19}$$

sendo $\mathbf{A} = diag(1, -1, 1, 1, -1)$. O vetor de controle é dado por

$$\mathbf{u}_n^* = \mathbf{B}^n \mathbf{u}_0^* \tag{3.20}$$

sendo $\mathbf{B} = diag(-1, 1, 1)$, sendo usado para fornecer o ângulo correto da perna em movimento em cada passo. Devido a perturbações, o estado real do MS $\mathbf{x}_n \neq \mathbf{x}_n^*$. O controlador LQR foi proposto por Liu et al. (2015) para regular o estado de MS. Definindo $\Delta\mathbf{x}_n = (\mathbf{x}_n - \mathbf{x}_n^*)$ e $\Delta\mathbf{u}_n = (\mathbf{u}_n - \mathbf{u}_n^*)$, fazendo uma mudança de variáveis, $\Delta\tilde{\mathbf{x}}_n \approx \mathbf{A}^n \Delta\mathbf{x}_n$ e $\Delta\tilde{\mathbf{u}}_n \approx \mathbf{B}^n \Delta\mathbf{u}_n$. Esta mudança de variáveis permite analisar a estabilidade da marcha periódica para dois passos (dois estados de MS), usando o mapa de retorno de 1 passo, Poincaré map (Morris e Grizzle (2015)). A identidade do mapa de retorno

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{A}\mathbf{f}(\mathbf{A}\mathbf{x}, \mathbf{B}\mathbf{u}) \tag{3.21}$$

é utilizada para obter erros da dinâmica para o par $(\mathbf{x}_n, \mathbf{u}_n)$:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n \tag{3.22}$$

Essa identidade pode ser entendida alterando a convenção de sinais do eixo y para a simulação de um passo e retornando o resultado à convenção original. Com este resultado, uma análise de primeira ordem do mapa de retorno em torno da trajetória periódica fornece:

$$\Delta\tilde{\mathbf{x}}_{n+1} \approx \mathbf{J}_x \Delta\tilde{\mathbf{x}} + \mathbf{J}_u \Delta\tilde{\mathbf{u}} \tag{3.23}$$

sendo, $\mathbf{J}_x = \mathbf{A}\frac{\delta \mathbf{f}}{\delta \mathbf{x}}$ e $\mathbf{J}_u = \mathbf{A}\frac{\delta \mathbf{f}}{\delta \mathbf{u}}$ avaliado em $(\mathbf{x}_0^*, \mathbf{u}_0^*)$. Na prática, os jacobianos são aproximados através de diferenças finitas. Então considere o custo quadrático:

$$\begin{aligned} \min_{\Delta\tilde{\mathbf{u}}} \sum_{n=0}^{\infty} \Delta\tilde{\mathbf{x}}_n^T \mathcal{Q} \Delta\tilde{\mathbf{x}}_n + \Delta\tilde{\mathbf{u}}_n^T \mathcal{R} \Delta\tilde{\mathbf{u}}_n \\ s.t \Delta\tilde{\mathbf{x}}_{n+1} = \mathbf{J}_x \Delta\tilde{\mathbf{x}}_n + \mathbf{J}_u \Delta\tilde{\mathbf{u}}_n \end{aligned} \tag{3.24}$$

Se $\mathcal{Q}$ e $\mathcal{R}$ são positivos definidos, e o par $(\mathbf{J}_x, \mathbf{J}_u)$ é controlável, então um ganho de *feedback* estável é dado por

$$\mathbf{K} = -(\mathbf{J}_u^T \mathbf{P} \mathbf{J}_u + \mathcal{R})^{-1} \mathbf{J}_u^T \mathbf{P} \mathbf{J}_x \tag{3.25}$$

onde $\mathbf{P}$ é a solução única da Equação de algébrica de Riccati no tempo Discreto (DARE). Podemos calcular a lei de controle como:

$$\mathbf{u}_n = \mathbf{u}_n^* + \mathbf{B}^n \mathbf{K} \mathbf{A}^n (\mathbf{x}_n - \mathbf{x}_n^*) \tag{3.26}$$

Note que para o modelo 3D Dual-SLIP, o erro de estado MS está em $\in \mathbb{R}^5$

$$\Delta \mathbf{x} = [\ \Delta x\ \Delta y\ \Delta z\ \Delta \dot{x}\ \Delta \dot{y}\ ]^T \tag{3.27}$$

enquanto o erro de controle está em $\in \mathbb{R}^3$. Para o modelo convencional

$$\Delta \mathbf{u} = [\ \Delta \phi\ \Delta \theta\ \Delta k\ ]^T \tag{3.28}$$

para o modelo melhorado temos:

$$\Delta \mathbf{u} = [\ \Delta \phi\ \Delta \theta\ \Delta B_{SS}\ ]^T \tag{3.29}$$

Para o modelo *3D Dual-SLIP* melhorado, a constante de rigidez da mola k não é ajustada pelo controlador LQR. Liu et al. (2015) cita que testes numéricos mostraram que a alteração da constante de rigidez da mola, por vezes, resulta em um jacobiano mal-condicionado. O controlador LQR tem a vantagem de nos permitir usar menos variáveis de controle e ainda ter um sistema (numericamente) controlável. O controlador LQR proposto é capaz de estabilizar o CoM de trajetórias periódicas epode ser usado para ajustar a velocidade da marcha on-line. Na seção anterior, identificamos uma coleção de passos periódicos em várias velocidades. Pode-se calcular uma matriz de ganho de realimentação constante K para cada velocidade de caminhada. Então, durante a caminhada, quando a velocidade desejada se altera, calcula o erro de estado do MS em relação ao ponto fixo (estado MS desejado) associado à velocidade desejada e usa o ganho K pré-calculado dessa velocidade para conduzir o estado MS para o ponto fixo desejado no mapa de retorno do MS.

3.7 Conclusões Parciais

Este capítulo mostrou como é possível utilizar o modelo *3D Dual-SLIP* para gerar trajetórias do CoM semelhantes a caminhada humana. Estas trajetórias foram geradas através das equações dinâmicas do modelo e estas foram avaliadas com uma equação de otimização não linear, o qual foi utilizado uma condição de simetria na dinâmica do modelo para 1 passo e a otimização foi simplificada para meio passo. Utilizando um modelo modificado ativo, foi possível obter trajetórias semelhantes com o caminhar humano, comparando a força de reação vertical observada durante o caminhar de uma pessoa. Como a dinâmica do modelo não é estável durante mais de 2 passos, um controlador LQR de tempo discreto foi proposto por Liu et al. (2015), sendo este computado apenas no MS compensando o modelo *3D DUAL SLIP* durante toda a trajetória, desta forma o modelo 3D dual SLIP torna estável para vários passos. No entanto, não foi implementado este controlador sobre o modelo neste trabalho, sendo que as trajetórias são derivadas do modelo *3D dual SLIP*, e estas replicadas no tempo para a simulação de vários passos do robô humanoide.

4 Controladores Cinemáticos

4.1 Função de erro invariante

Definindo $\underline{\mathbf{h}}_d$ e $\underline{\mathbf{h}}$ como dois quatérnios duais unitários que representam, respectivamente, a configuração desejada e a configuração atual de um robô, ou seja, a posição e orientação desejada e atual. A distância euclidiana no espaço dos quatérnios duais é representada como $\underline{\mathbf{h}}_e = \underline{\mathbf{h}}^* \underline{\mathbf{h}}_d$. A função de erro invariante é definida por Marinho, Figueredo e Adorno (2015):

$$\underline{\mathbf{e}} = 1 - \underline{\mathbf{h}}_e \tag{4.1}$$

$$\underline{\mathbf{e}} = 1 - \underline{\mathbf{h}}^* \underline{\mathbf{h}}_d \tag{4.2}$$

Quando $\underline{\mathbf{h}}$ converge para $\underline{\mathbf{h}}_d$, a distância espacial $\underline{\mathbf{h}}_e$ tende a 1 e, consequentemente, a função de erro invariante $\underline{\mathbf{e}} \rightarrow 0$. As propriedades de convergência de $\underline{\mathbf{h}}$ para $\underline{\mathbf{h}}_d$ estão relacionadas com a definição do erro, que é invariável, mesmo com alterações de coordenadas. Por exemplo, se considerar que a configuração desejada $\underline{\mathbf{h}}_d$ e a configuração atual $\underline{\mathbf{h}}$ forem transformadas por uma mudança de coordenadas, representado pelo quatérnio dual unitário $\underline{\mathbf{y}}$, desta forma:

$$\underline{\mathbf{h}}^{'} = \underline{\mathbf{y}}\underline{\mathbf{h}} \tag{4.3}$$

$$\underline{\mathbf{h}}_d^{'} = \underline{\mathbf{y}}\underline{\mathbf{h}}_d \tag{4.4}$$

$$\underline{\mathbf{h}}_e^{'} = \underline{\mathbf{h}}^{'} \underline{\mathbf{h}}_d^{'} \tag{4.5}$$

O erro invariante no novo sistema de coordenadas é dado por

$$\begin{aligned} \underline{\mathbf{e}}^{'} &= 1 - \underline{\mathbf{h}}_e^{'} = 1 - \underline{\mathbf{h}}^{'*} \underline{\mathbf{h}}_d^{'} \\ &= 1 - \underline{\mathbf{h}}^* \underline{\mathbf{y}}^* \underline{\mathbf{y}} \underline{\mathbf{h}}_d \\ &= 1 - \underline{\mathbf{h}}^* \underline{\mathbf{h}}_d \\ &= \underline{\mathbf{e}} \end{aligned} \tag{4.6}$$

Neste contexto, $\underline{\mathbf{e}}$ é invariante, independente da escolha do sistema de coordenadas de base do robô e de mudanças de coordenadas. Do ponto de vista do controle, os controladores são projetados para estabilizar assintoticamente o sistema, com o objetivo de $\underline{\mathbf{h}}(\mathrm{t}) \rightarrow \underline{\mathbf{h}}_d$ (t) quanto $\mathrm{t} \rightarrow \infty$. Nesse sentido, a norma do erro invariante resulta na convergência da configuração atual para a configuração desejada, sem levar em conta o sistema de coordenadas de base do robô e de mudanças de coordenadas. Desta forma, a função de erro invariante será usada no controlador cinemático implementado neste trabalho, no espaço dos quatérnios duais.

4.2 Controlador Proporcional

O controlador proporcional utilizando a álgebra dos quatérnios duais é demostrado em (Pereira (2016)). Supondo que, a configuração atual $\underline{\mathbf{h}}$ esteja variando no tempo , e a configuração desejada $\underline{\mathbf{h}}_d$ possui uma velocidade constante , é possível diferenciar a função do erro invariante (4.1)

$$\dot{\underline{\mathbf{e}}} = -\dot{\underline{\mathbf{h}}}^* \underline{\mathbf{h}}_d - \underline{\mathbf{h}}^* \dot{\underline{\mathbf{h}}}_d = -\dot{\underline{\mathbf{h}}}^* \underline{\mathbf{h}}_d \tag{4.7}$$

Para $\underline{\mathbf{h}}_d$ constante, $\dot{\underline{\mathbf{h}}}_d$ é igual a zero. Aplicando o operador *vec* (2.29) em (4.7), considerando que $\underline{\mathbf{e}} = vec(\underline{\mathbf{e}})$ e $\dot{\underline{\mathbf{e}}} = vec(\dot{\underline{\mathbf{e}}})$, temos que:

$$\dot{\underline{\mathbf{e}}} = vec(-\dot{\underline{\mathbf{h}}}^* \underline{\mathbf{h}}_d) = \bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d) vec(-\dot{\underline{\mathbf{h}}}^*) \tag{4.8}$$

$$\dot{\underline{\mathbf{e}}} = -\bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d)\mathbf{C}_8 vec(\dot{\underline{\mathbf{h}}}^*) \tag{4.9}$$

pela cinemática diferencial temos que:

$$\dot{\underline{\mathbf{h}}} = \mathbf{J}\dot{\vec{\theta}} \tag{4.10}$$

utilizando 4.10 é possível obter

$$\dot{\underline{\mathbf{e}}} = -\bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d)\mathbf{C}_8\mathbf{J}\dot{\vec{\theta}} \tag{4.11}$$

Definindo

$$\mathbf{N} = \bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d)\mathbf{C}_8\mathbf{J} \tag{4.12}$$

substituindo em (4.9) e resolvendo a equação para $\dot{\vec{\theta}}$

$$\dot{\underline{\mathbf{e}}} = -N\dot{\vec{\theta}} = vec(\dot{\underline{\mathbf{e}}}) \tag{4.13}$$

$$\dot{\vec{\theta}} = -N^{\dagger} vec(\dot{\underline{\mathbf{e}}}) \tag{4.14}$$

$N^{\dagger}$ é a pseudo-inversa da matriz N. A matriz pseudo-inversa é calculada utilizando o método *damped least-squares* proposto em (Wampler (1986)). Para garantir a convergência, um ganho k é adicionado. Assim, o controlador proporcional baseado em quatérnios duais (controlador K) é dado por

$$\mathbf{K} = k\mathbf{I}, k \in \mathrm{R}^+ \tag{4.15}$$

$$\dot{\vec{\theta}} = -N^{\dagger}\mathbf{K} vec(\dot{\underline{\mathbf{e}}}) \tag{4.16}$$

A matriz de ganho $\mathbf{K}$ arbitrário (4.15) controla a taxa de convergência. É mostrado em (SPONG, HUTCHINSON e VIDYASAGAR (2005)) que o sistema é estável para $\mathbf{K} > 0$. Abaixo é mostrado um pseudocódigo de como pode ser computado o controlador proporcional no espaço dos quatérnios duais

Algorithm 2 Controlador Proporcional

1: *Calcular* $\underline{\mathbf{h}}_d$
2: **while** $i \ < it$ **do** ▷ it - número máximo de iterações
3: *Calcular* $\mathbf{J}$
4: *Calcular* $\mathbf{N}$
5: *Calcular* $\mathbf{N}^\dagger$
6: *Calcular* $\mathbf{e}$
7: $\dot{\underline{\theta}} \leftarrow \mathbf{N}^\dagger(\mathbf{Ke})$
8: $\underline{\theta} \leftarrow \dot{\underline{\theta}}\Delta t$
9: $\underline{\mathbf{h}} \leftarrow \underline{\mathbf{h}}(\underline{\theta})$
10: $i \leftarrow i + 1$
11: **end while**

4.3 Controlador Proporcional com um termo de *Feed-forward*

O controlador proporcional não considera a influência de uma referência em movimento. Para melhorar o controlador, é possível incluir um termo de avanço (K + FF) ao controlador proporcional(4.16). Considerando uma referência variando no tempo $\underline{\mathbf{h}}_d = \underline{\mathbf{h}}_d(t)$, então a derivada de 4.1 é dada por

$$\dot{\underline{\mathbf{e}}} = -\dot{\underline{\mathbf{h}}}^* \underline{\mathbf{h}}_d - \underline{\mathbf{h}}^* \dot{\underline{\mathbf{h}}}_d \tag{4.17}$$

Aplicando o operador *vec* em 4.17, considerando que $\underline{\mathbf{e}} = vec(\underline{\mathbf{e}})$ e $\dot{\underline{\mathbf{e}}} = vec(\dot{\underline{\mathbf{e}}})$, temos que:

$$\dot{\underline{\mathbf{e}}} = vec(-\dot{\underline{\mathbf{h}}}^* \underline{\mathbf{h}}_d) - vec(\underline{\mathbf{h}}^* \dot{\underline{\mathbf{h}}}_d) \tag{4.18}$$

usando 2.31, 2.32, 4.10 e 4.12 em 4.18

$$\dot{\underline{\mathbf{e}}} = -\underline{\mathbf{H}}_8^-(\underline{\mathbf{h}}_d)\mathbf{C}_8 vec_8(\dot{\underline{\mathbf{h}}}^*) - vec_8(\underline{\mathbf{h}}^* \underline{\dot{\mathbf{h}}_\mathbf{d}}) \tag{4.19}$$

$$\dot{\underline{\mathbf{e}}} = -\underline{\bar{\mathbf{H}}}_8(\underline{\mathbf{h}}_d)\mathbf{C}_8\mathbf{J}\dot{\vec{\theta}} - vec_8(\underline{\mathbf{h}}^* \underline{\dot{\mathbf{h}}_\mathbf{d}}) \tag{4.20}$$

$$\dot{\underline{\mathbf{e}}} = -N\dot{\vec{\theta}} - vec_8(\underline{\mathbf{h}}^* \underline{\dot{\mathbf{h}}_\mathbf{d}}) \tag{4.21}$$

resolvendo 4.21

$$\mathbf{K} = k\mathbf{I}, com \ k \ \in \ \mathbb{R}^+ \tag{4.22}$$

$$\dot{\vec{\theta}} = -N^\dagger(\mathbf{K}\underline{\mathbf{e}} - vec_8(\underline{\mathbf{h}}^* \underline{\dot{\mathbf{h}}_\mathbf{d}}) \ ) \tag{4.23}$$

$\dot{\underline{\mathbf{e}}} = k\underline{\mathbf{e}}$, e k controla a taxa de convergência. O termo de *feedfoward* funciona como um *feedback* que compensa a variação da trajetória em decorrer do tempo. Abaixo é mostrado um pseudocódigo de como pode ser computado o controlador proporcional com o termo de *feedfoward* no espaço dos quatérnios duais.

Algorithm 3 Controlador Proporcional + FF

1: *Calcular* $\underline{\mathbf{h}}_d(t)$
2: **while** $t < t_f$ **do**
3: *Calcular* $\mathbf{J}$
4: *Calcular* $\mathbf{N}$
5: *Calcular* $\mathbf{N}^\dagger$
6: *Calcular* $\mathbf{e}$
7: $\dot{\underline{\theta}} \leftarrow \mathbf{N}^\dagger(\mathbf{K}\mathbf{e})$
8: $\underline{\theta} \leftarrow \dot{\underline{\theta}}\Delta t$
9: $\underline{\mathbf{h}} \leftarrow \underline{\mathbf{h}}(\underline{\theta})$
10: **end while**

4.4 Controlador Ótimo Baseado em Quatérnios Duais

Em Marinho, Figueredo e Adorno (2015) é demonstrado como pode ser implementado um controlador LQR no espaço dos quatérnios duais. O LQR explora o conhecimento futuro da trajetória desejada, portanto, calcula os ganhos de controle para uma determinada trajetória desejada. Com $\underline{\mathbf{h}}$ e $\underline{\mathbf{h}}_d$ variando no tempo, a derivada do erro é dada por 4.16. Temos também que:

$$\begin{aligned} \underline{\mathbf{e}} &= 1 - \underline{\mathbf{h}}^*\underline{\mathbf{h}}_d \\ \underline{\mathbf{e}}\underline{\mathbf{h}}_d^* &= \underline{\mathbf{h}}_d^* - \underline{\mathbf{h}}^* \\ \underline{\mathbf{h}}^* &= \underline{\mathbf{h}}_d^* - \underline{\mathbf{e}}\underline{\mathbf{h}}_d^* \end{aligned} \tag{4.24}$$

Usando o operador vec_8 em ambos os lados de 4.16 , e aplicando 4.16 a 4.24

$$\vec{e} = -\bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d)vec(\dot{\underline{\mathbf{h}}}^*) + \bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d^*\dot{\underline{\mathbf{h}}}_d)\underline{\mathbf{e}} - vec(\underline{\mathbf{h}}_d^*\dot{\underline{\mathbf{h}}}) \tag{4.25}$$

usando a cinemática diferencial 4.10 é possível obter

$$\vec{e} = -\bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d)\mathbf{C}_8\mathbf{J}\dot{\vec{\theta}} + \bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d^*\dot{\underline{\mathbf{h}}}_d)\underline{\mathbf{e}} - vec(\underline{\mathbf{h}}_d^*\dot{\underline{\mathbf{h}}}) \tag{4.26}$$

definindo $\mathbf{A} = \bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d^*\dot{\underline{\mathbf{h}}}_d)$, $\mathbf{N} = \bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{h}}_d)\mathbf{C}_8\mathbf{J}$ e $\vec{c} = -vec(\underline{\mathbf{h}}_d^*\dot{\underline{\mathbf{h}}})$

$$\vec{e} = \mathbf{A}\underline{\mathbf{e}} - \mathbf{N}\dot{\vec{\theta}} + \vec{c} \tag{4.27}$$

Então, o objetivo é encontrar o controlador ideal para o sistema variante no tempo.

$$\vec{e}(t) = \mathbf{A}\underline{\mathbf{e}}(t) + \vec{u}(t) + \vec{c} \tag{4.28}$$

onde $\vec{u}(t) = -\mathbf{N}\dot{\vec{\theta}}$.

Do ponto de vista do erro, é possível resolver o problema de rastreamento para uma trajetória usando um LQR aplicado a um sistema perturbado. A perturbação de erro é causada pela trajetória variável no tempo $\vec{c}(t)$. Outras perturbações contínuas podem também ser modeladas e agrupadas em $\vec{c}(t)$ e usadas na mesma solução. Considere que

pretendemos rastrear uma trajetória durante $t \in [0; t_f]$. Então, é desejado minimizar a seguinte função de custo.

$$\mathbf{F} = \frac{1}{2}\vec{e}(t_f)^T\mathbf{S}\vec{e}(t_f) + \frac{1}{2}\int_0^{t_f}(\vec{e}^T\mathbf{Q}\vec{e} + \vec{u}^T\mathbf{R}\vec{u})dt \tag{4.29}$$

dadas as matrizes $\mathbf{S}, \mathbf{Q} \geq 0$ *e* $\mathbf{R} > 0$ com $\mathbf{S}, \mathbf{Q}, \mathrm{R}^{8x8}$. A matriz $\mathbf{S}$ é o peso do erro no final da trajetória, a matriz $\mathbf{Q}$ é o custo do erro ao longo da trajetória, e a matriz $\mathbf{R}$ é o ganho do controlador. Se com o tempo $\mathbf{N}$ está bem condicionado, um aumento em $\mathbf{R}$ também causará uma diminuição geral nas velocidades das juntas. A otimização de 4.29 leva a um *feedback* ótimo sem gasto excessivo de energia no controle, mantendo o erro $e(t)$ perto de zero (Marinho, Figueredo e Adorno (2015)). Para resolver o problema de otimização, é introduzida a variável de custo p, que atua como um Multiplicador de Lagrange para as equações de estado. Então, usando a restrição de igualdade definida em 4.28, a função 4.29 pode ser reescrita como

$$\mathbf{H} = \mathbf{F} + \int_0^{t_f} p^T(\mathbf{A}\vec{e} + \vec{u} + \vec{c} - \dot{\vec{e}})dt \tag{4.30}$$

temos que $\partial H/\partial\vec{u} = 0$ e $\partial H/\partial\vec{e} = 0$ como condições necessárias para a trajetória ótima, então,

$$\begin{aligned} \partial H/\partial\vec{u} = 0 \Longrightarrow \mathbf{R}\vec{u} + \vec{p} = 0 \Longrightarrow \vec{u} = -\mathbf{R}^{-1}\vec{p} \\ \partial H/\partial\vec{e} = 0 \Longrightarrow \mathbf{Q}\vec{e} + \mathbf{A}^T\vec{p} + \dot{\vec{p}} = 0 \\ \Longrightarrow \dot{\vec{p}} = -(\mathbf{Q}\vec{e} + \mathbf{A}^T\vec{p}) \end{aligned} \tag{4.31}$$

O termo $\partial^2 H/\partial^2\vec{u} = 0$ deve ser positivo para minimizar 4.30, o que requer $\mathbf{R} > 0$. O sistema e a função de custo proposta, permite o uso da função de custo (Marinho, Figueredo e Adorno (2015)):

$$\dot{\vec{p}} = \mathbf{P}\vec{e} + \dot{\vec{\xi}} \tag{4.32}$$

onde $\mathbf{P}$ é um ganho proporcional variável no tempo e $\dot{\vec{\xi}}$ é um termo de avanço ponderado. A derivada de 4.32 é dado por:

$$\ddot{\vec{p}} = \dot{\mathbf{P}}\vec{e} + \mathbf{P}\dot{\vec{e}} + \ddot{\vec{\xi}} \tag{4.33}$$

Aplicando 4.32 em 4.31 e usando o resultado em 4.28, e também substituindo 4.32 em 4.31, estes resultados são aplicados a 4.33. Considera-se também que 4.33 deve ser válido para qualquer escolha do estado inicial de $\vec{e}$, e tanto $\mathbf{P}$ como $\dot{\vec{\xi}}$, não dependem do erro inicial, logo:

$$\dot{\mathbf{P}} = -\mathbf{P}\mathbf{A} - \mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{R}^{-1}\mathbf{P} - \mathbf{Q} \tag{4.34}$$

$$\ddot{\vec{\xi}} = -\mathbf{A}^T\dot{\vec{\xi}} + \mathbf{P}\mathbf{R}^{-1}\dot{\vec{\xi}} - \mathbf{P}\vec{c} \tag{4.35}$$

As equações 4.34 e 4.35 são resolvidas encontrando as condições de contorno, usando o tempo final t_f da trajetória e assumindo $\dot{\vec{\xi}}(t_f) = 0$ para encontrar a primeira condição de contorno. De 4.32, com $\dot{\vec{\xi}}(t_f) = 0$, é obtido $\partial H/\partial\vec{e}(t_f) = 0$, que produz $\mathbf{P}(t_f) = \mathbf{S}$.

É importante notar, que $\mathbf{A} = \bar{\underline{\mathbf{H}}}_8(\underline{\mathbf{x}}_d^* \dot{\underline{\mathbf{x}}}_d)$ para todo tempo. Assim, a equação diferencial Ricatti $\dot{\mathbf{P}}(t)$ pode ser resolvida numericamente de forma regressiva no tempo. Como $\vec{c} = \underline{\mathbf{x}}_d^* \dot{\underline{\mathbf{x}}}$ é também conhecido para todo tempo , e assim, como a solução de $\dot{\mathbf{P}}(t)$, $\dot{\vec{\xi}}$ pode ser encontrada resolvendo numericamente de forma regressiva no tempo. Portanto, de 4.31 e 4.32 o controle ótimo é dado por $\vec{u}(t) = -\mathbf{R}^{-1}(\mathbf{P}\vec{e} + \vec{\xi})$. As velocidades das juntas são computadas como:

$$\dot{\vec{\theta}} = \mathbf{N}^{\dagger}\mathbf{R}^{-1}(\mathbf{P}\vec{\mathbf{e}} + \vec{\xi}) \tag{4.36}$$

Abaixo é mostrado um pseudocódigo para computar o controlador LQR no espaço dos quatérnios duais

Algorithm 4 Controlador LQR

Definir $\mathbf{R}$
Definir $\mathbf{Q}$
Definir $\mathbf{S}$
$\underline{\mathbf{h}} \leftarrow \underline{\mathbf{h}}(\underline{\theta})$
$\underline{\mathbf{h_d}} \leftarrow \underline{\mathbf{h}}_d(t)$
while $t \ < t_f$ **do**
    Calcular $\mathbf{A}$
    Calcular $\mathbf{c}$
    Calcular $\mathbf{J}$
    Calcular $\mathbf{N}$
    Calcular $\mathbf{N}^{\dagger}$
    Calcular $\mathbf{e}$
    $\dot{\underline{\theta}} \leftarrow \mathbf{N}^{\dagger}\mathbf{R}^{-1}(\mathbf{Pe} + \xi)$
    $\underline{\theta} \leftarrow \dot{\underline{\theta}}\Delta t$
    $\underline{\mathbf{h}} \leftarrow \underline{\mathbf{h}}(\underline{\theta})$
end while

4.5 Conclusões Parciais

Neste capítulo o controlador proporcional foi derivado da função de erro invariante no tempo. No entanto, este controlador não leva em consideração uma trajetória variante no tempo. Desta forma, um termo de *feedforward* foi adicionado ao controlador. Entretanto, ambos os controladores não levam em consideração incertezas e distúrbios que são presente na maioria das aplicações. Desta forma, foi mostrado o controlador LQR, um controlador ótimo, sendo este capaz de incorporar distúrbios e variações da trajetória, sendo possível alcançar um equilíbrio entre as velocidades de juntas e o erro de trajetória ao definir dois ganhos, as matrizes $\mathbf{R}$ e $\mathbf{Q}$. Um dos objetivos deste trabalho é implementar um controlador cinemático em robô humanoide, sendo o robô detalhado melhor no próximo capitulo.

5 Robô Humanoide

O modelo do robô escolhido é baseado no humanoide HUBO, mostrado na Fig. 5.1. Este robô apresenta um peso total de 38,1 kg e a distribuição de massa é resumida na Tabela 5.1. A massa de uma única perna é responsável por uma grande porcentagem da massa total do corpo (cerca de 26%), enquanto o tronco apenas detém 20% . Como a maioria da massa está concentrada nas pernas, o CoM se encontra entre ambas, aproximadamente na mesma altura das articulações do quadril, todos os dados são referentes ao artigo Liu et al. (2015). Quando o modelo está em pé, na posição vertical com as pernas

Corpo	Parte da massa (kg)	Porcentagem
Tronco	7,96	20,5%
Pélvis	3,42	8,8%
Cada braço	3,36	8,6%
Cada perna	10,23	26,3%
Cabeça	0,38	1%

Tabela 5.1: Distribuição da massa do modelo HUBO

totalmente esticadas, a distância do solo até o quadril é de $66,63\ cm$ e a distância entre os dois pés é aproximadamente 17 cm. Na seção 3.2 foram apresentados os resultados empregando o modelo 3D Dual-SLIP, com um comprimento de perna de 1 m. Como o modelo do HUBO tem um comprimento de perna menor, as velocidades podem ser escaladas através do número não-dimensional de Froude (Vaughan e O'Malley (2005)). O número de Froude é definido como: $Fr = v^2/gl$. Por exemplo, andando $1/m/s$ com $1/m$ de comprimento da perna , uma caminhada de $0,75/m/s$ com $0,6/m$ de comprimento da perna apresenta aproximadamente o mesmo número de Froude. Esta é uma forma de estimar a velocidade de um modelo em relação a outro. Sendo assim, a massa e o comprimento da perna em repouso do modelo *3D Dual-SLIP* são atualizada, respectivamente, em $m = 38,1\ kg$ e $l_0 = 0,63/m$. Na equação de otimização Eq. 3.18, sendo a altura desejada do MS também é atualizada para $z_{0,d} = 0.6\ m$. Usando os novos parâmetros para o modelo 3D Dual-SLIP, é executada a otimização na Eq. 3.18 para encontrar as trajetórias periódicas compatíveis com este modelo do humanoide. O balanço lateral do CoM não varia mais de 5 cm durante a caminhada regular. As trajetórias das pernas em movimento foram geradas utilizando uma curva de Bezier de 4ª ordem relativa ao CoM. Na tabela 5.2 é mostrado o comprimento dos elos das pernas, e na tabela 5.3 mostra o comprimento dos elos dos braços. Em Ali, Park e Lee (2010) é mostrado os parâmetros de DH para o robô HUBO, Fig.5.2 e Fig. 5.3. A cinemática direta do robô é obtida usando matrizes de transformação, no qual é mostrado uma forma de obter a cinemática

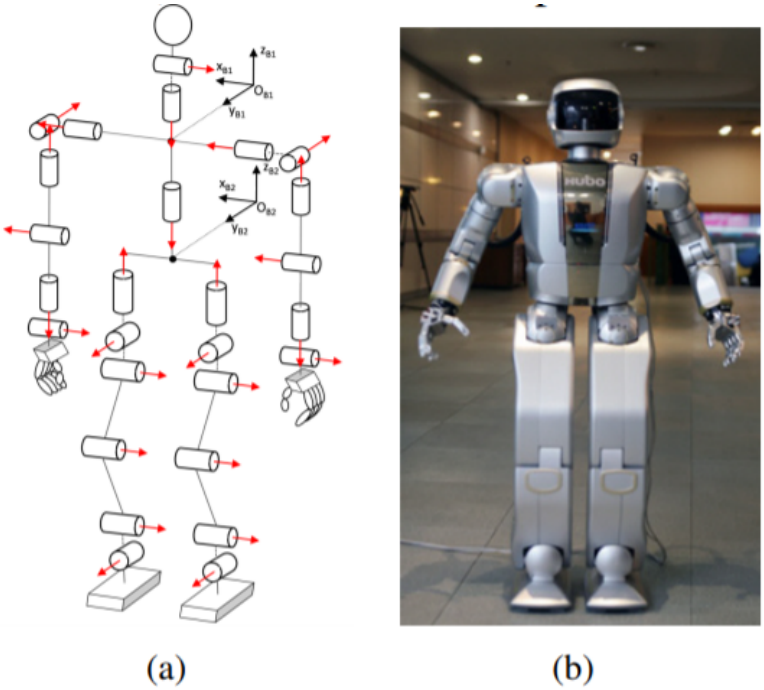
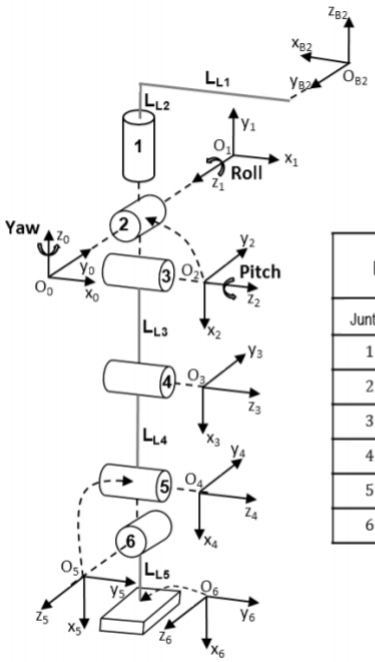


Figura 5.1: (Fig 1 em Ali, Park e Lee (2010)) Robô Humanoide Hubo.



Parâmetros de D-H perna direita				
Junta *i*	θ_i	α_i	a_i	d_i
1	0	90°	0	0
2	−90°	−90°	0	0
3	0	0	L_{L3}	0
4	0	0	L_{L4}	0
5	0	90°	0	0
6	0	0	L_{L5}	0

Figura 5.2: (Fig 3 em Ali, Park e Lee (2010)) Perna direita do Robô Hubo e seus parâmetros D-H.

inversa para uma configuração específica de robôs humanoides, como mostrada na Fig. 5.4. A cadeia cinemática de todos esses robôs humanoides é quase a mesma comparada com o robô da Hubo. Para encontrar a cinemática inversa Ali, Park e Lee (2010) utiliza a solução de Pieper (Pieper (1968)), na qual é demostrada para manipuladores com 6 (ou 5 ou 4) graus de liberdade, quando as 3 (ou 2 ou 1) últimas juntas forem rotacionais e os

Link	Comprimento (mm)
L_{L1}	85
L_{L2}	182
L_{L3}	300
L_{L4}	300
L_{L5}	95

Tabela 5.2: Comprimento dos elos da perna do robô HUBO

seus eixos interceptam-se em um único ponto, é possível obter-se uma solução geral para o problema cinemático inverso. Neste caso, é possível desacoplar o problema cinemático inverso em dois problemas mais simples, o problema de posicionamento inverso e o problema de orientação inverso, sendo estendido para o caso do robô humanoide. A partir dessa abordagem, é possível encontrar uma solução geométrica para robôs humanoides com uma cadeia cinemática parecida com a encontrada para o robô Hubo. Sendo assim, foi escolhido o robô HUBO para o estudo neste trabalho, devido sua característica comum a vários outros robôs humanoides. No entanto, neste trabalho não foi feita esta abordagem de Pieper, no qual, foi considerado que todas as juntas são utilizadas para a contribuição do movimento do robô e sua modelagem utilizando os parâmetros de DH mostrados em Fig.5.2 e Fig.5.3, com a cinemática direta estendida para os quatérnios duais.

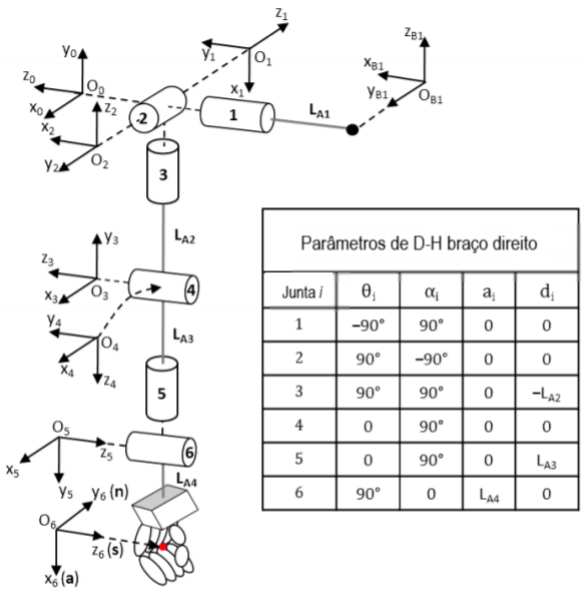


Figura 5.3: (Fig. 2 em Ali, Park e Lee (2010)) Braço direito de um robô Hubo e seus parâmetros D-H.

Link	Comprimento (mm)
L_{A1}	215
L_{A2}	179
L_{A3}	182
L_{A4}	121

Tabela 5.3: Comprimento dos elos do braço do robô HUBO

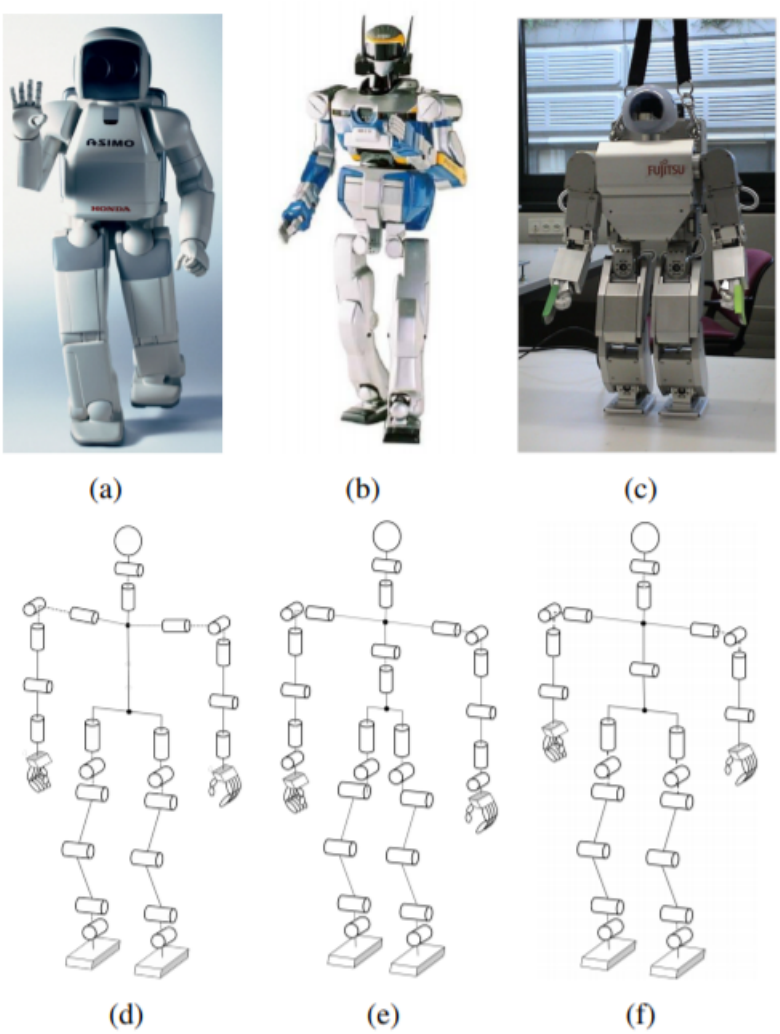


Figura 5.4: (Fig. 4 em Ali, Park e Lee (2010)) (a) Robô HONDA ASIMO e o seu diagrama cinemático associado em (d), (b) Robô AIST HRP-2 e seu diagrama cinemático associado em (e), e (c) Robô Fujitsu HOAP-2 e seu diagrama cinemático associado em (f).

5.1 Conclusões Parciais

Neste capítulo foi descrito o robô humanoide escolhido, sendo este o robô HUBO, devido sua característica comum a vários outros robôs humanoides, bem como sua modela-

gem utilizando os parâmetros de D-H para sua representação no espaço. Os controladores cinemáticos foram implementados para que o robô descreva as trajetórias geradas do modelo *3D DUAL SLIP*, assim descrevendo uma caminhada, sendo descrito no próximo capítulo.

6 Resultados

No capítulo 3 foi mostrado como obter trajetórias dinamicamente estáveis com o modelo 3D Dual-SLIP. Os parâmetros para o modelo usado no robô Hubo são descritos no capítulo 5. As trajetórias foram obtidas pela otimização da equação 3.18. Para resolver esta equação optou-se por implementar o método do gradiente descendente, devido sua ampla utilização em problemas de otimização.

Em Ruder (2017) é mostrado sobre o método do gradiente descendente e suas variações. Neste trabalho foi implementado para solução da equação de otimização Eq. 3.18, o método do gradiente descendente, gradiente com momento, NAG (*Nesterov accelerated gradient*) e Adagrad. Todos estes métodos são descritos em Ruder (2017), assim como mais algumas outras variações, sendo estas, não utilizadas neste trabalho. Inicialmente é utilizado o método do gradiente descendente, no entanto é deixado o parâmetro de aprendizado do método ajustado manualmente. O cálculo dinâmico, pode causar a não convergência, bem como levar a soluções não desejadas. Este parâmetro de aprendizado nos testes variava entre 0,1 a 0,0001. Quando o método do gradiente não obtinha êxito em convergir utilizou-se as variações (gradiente com momento, NAG ou Adagrad). Os testes realizados conseguiram uma convergência da ordem de 10^{-10}, sendo este valor considerado zero para a otimização, ou seja, o parâmetro de parada do algorítimo. O cálculo do gradiente foi realizado numericamente, utilizando diferenças finitas centradas (Eq. 6.1). As diferenças finitas centradas produzem fórmulas mais acuradas.

$$O(h^2) : \frac{-f(x_{i+2}) + 8f(x_{i+1} - 8f(x_{i-1}) + f(x_{i-2})}{12h} \tag{6.1}$$

Um pseudocódigo para implementação do método de otimização é descrito no algoritmo 6.

Algorithm 5 Método do Gradiente

1: $X \leftarrow \mathbf{x}_{0,d}$ ▷ x recebe o valor inicial
2: $U \leftarrow \mathbf{u}_0$ ▷ u recebe o valor do vetor de controle
3: **while** $i < max \;||\; fo > 10^{-10}$ **do** ▷ enquanto i é menor que o número máximo de iterações ou o valor da função objetivo é maior que o valor de parada
4: $U \leftarrow SGD(U, X)$ ▷ calcula o valor do gradiente descendente estocástico
5: $[pa, pb, pc] \leftarrow trajetoria(U, X)$ ▷ calcula os parâmetros da otimização Eq. 3.18
6: $fo \leftarrow funcaoObjetivo(pa, pb, pc)$ ▷ calcula o valor da função objetivo
7: $i \leftarrow i + 1$ ▷ incrementa i
8: **end while**

A Fig. 6.1 mostra a trajetória obtida pela otimização. Em azul, descreve-se a fase de suporte único, ou seja, quando uma perna está em contato com o solo, e na cor magenta

a fase de suporte duplo, quando as duas pernas estão em contato com o chão. No plano XY é mostrado a projeção da trajetória do CoM, sendo na cor verde, a fase de suporte único, e em magenta a fase de suporte duplo. O centro da trajetória é mostrado pelo círculo azul. A curva tracejada na cor preta, mostra o eixo de simetria da projeção do CoM.

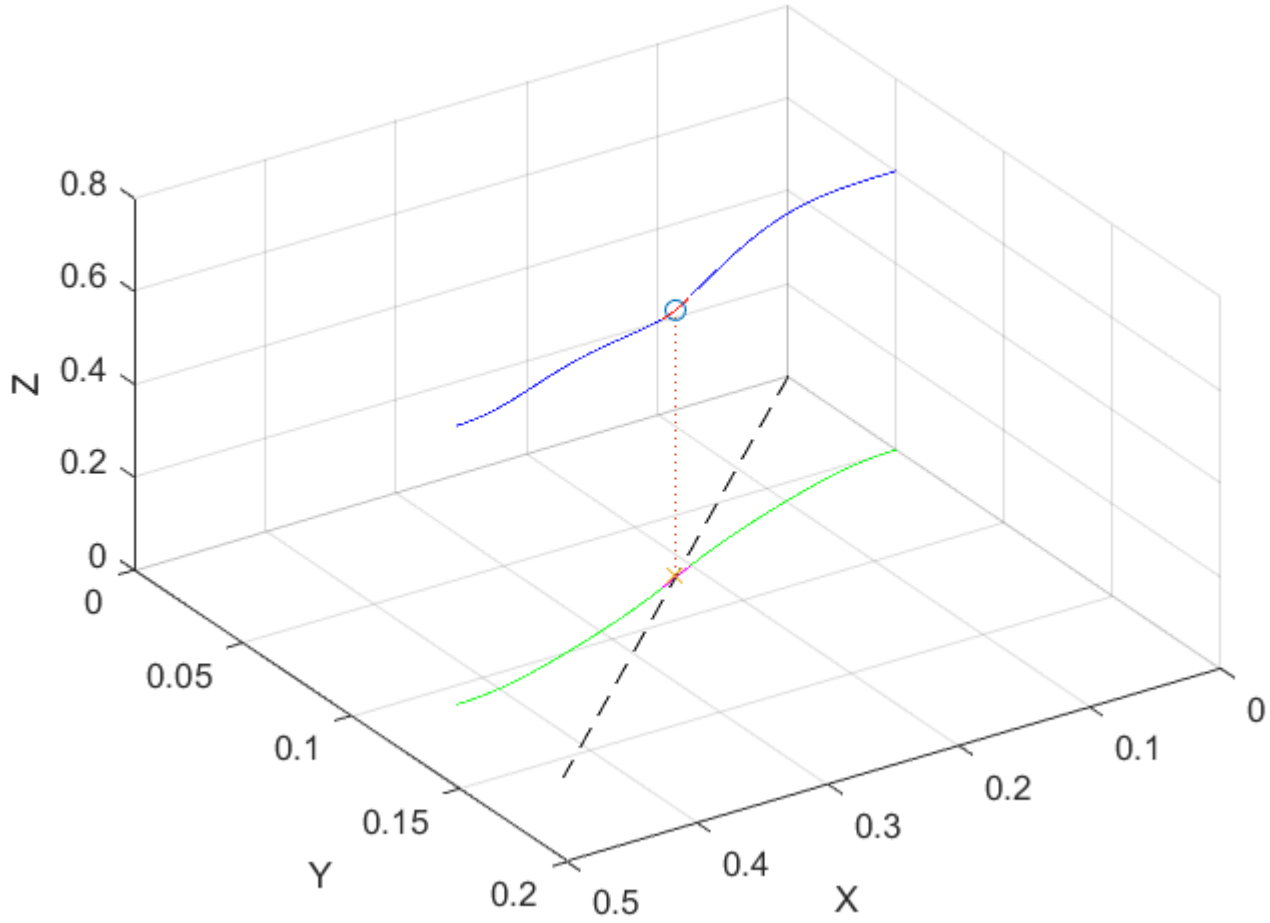


Figura 6.1: Trajetória do centro de massa do Hubo pelo modelo 3D Dual-SLIP.

A Fig. 6.2 mostra a vista no plano XY da trajetória do CoM, retirado da Fig. 6.1. É importante observar a simetria da trajetória, sendo este, um dos parâmetros da otimização, mostrando que visualmente o algorítimo de otimização conseguiu computar a trajetória. A linha preta tracejada descreve o eixo de simetria, no qual, nos extremos se encontra a posição dos pés. Em azul está a fase de suporte único e na cor magenta a fase de contato duplo do movimento.

A Fig. 6.3 descreve a vista do plano ZX da trajetória do CoM. Como citado no

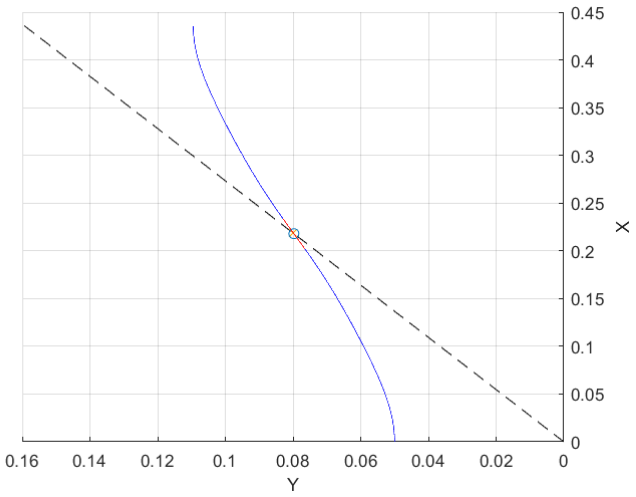


Figura 6.2: Trajetória do centro de massa do Hubo pelo modelo 3D Dual-SLIP, plano XY.

capítulo 2, a variação no eixo z é pequena. A condição de LH é mostrada nesta figura, sendo a metade da trajetória, sendo o momento que o centro de massa se encontra na sua menor altura.

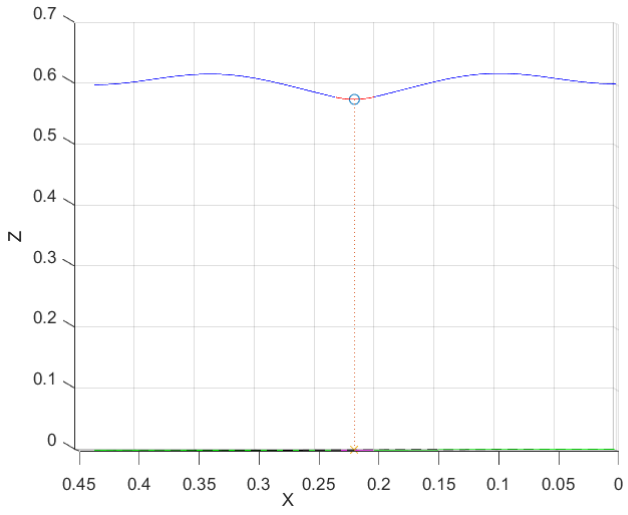


Figura 6.3: Trajetória do centro de massa do Hubo pelo modelo 3D Dual-SLIP plano ZX.

A Fig. 6.4 mostra uma vista do plano ZY da trajetória do CoM. Podemos observar que esta trajetória é simétrica com uma variação pequena no eixo Y. Essa trajetória é o balanco lateral periódico da caminhada.

A Fig. 6.5 mostra as curvas dos parâmetros do modelo 3D DUAL SLIP, para uma dada velocidade. Estas curvas podem ser armazenadas, e utilizadas para gerar trajetórias

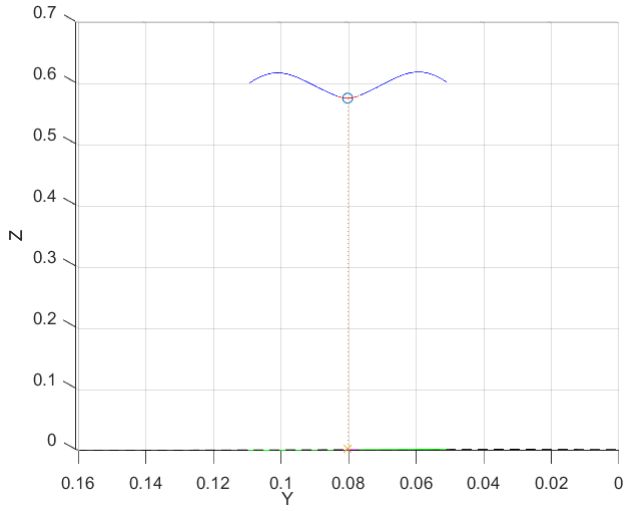


Figura 6.4: Trajetória do centro de massa do Hubo pelo modelo 3D Dual-SLIP plano ZY.

para varias velocidades. Como descrito na seção 3.6, é possível implementar um controlador para o modelo e incorporar a dinâmica dele para varias velocidades. No entanto, devido o tempo para desenvolvimento deste trabalho, este controlador não foi implementado. Desta forma, é utilizado as funções apenas para obter uma trajetória para caminhada e esta é replicada com o tempo para o robô.

A Fig. 6.6 mostra a abordagem descrita. Foi utilizado o modelo *3D DUAL SLIP* para gerar a trajetória com o tempo, demostrado pela curva na cor preta e sua projeção na cor verde. O eixo de simetria entre os pés é demostrado com a curva pontilhada no plano XY. Em azul, descreve-se a trajetória devido a perna direita e em vermelho a trajetória devido a perna esquerda. O movimento dos pés foram descritas pelas curvas de Bézier de 4º ordem. A Fig. 6.6 mostra a evolução para vários passos.

Como descrito, para a realização deste trabalho, o modelo 3D DUAL SLIP, foi usado para encontrar uma trajetória para o CoM, semelhante a caminhada humana, sendo esta replicada com o tempo para o robô. Sendo assim, o robô sempre descreve uma trajetória com velocidade constante. Neste trabalho não foi implementado a variação de velocidades no decorrer da trajetória incorporando o modelo 3D DUAL SLIP dinamicamente na caminhada, no qual este modelo deveria ser resolvido em tempo real computando os parâmetros com o controlador LQR para o modelo, assim como descrito na seção 3.6. Com o modelo do robô e as trajetórias, obtidas através do modelo dinâmico 3D DUAL-SLIP, queremos desenvolver um controlador que consiga fazer o robô, simulado, descrever trajetórias obtidas pelo modelo dinâmico. Pelo fato de ter duas trajetórias, do centro de massa e dos pés, inicialmente como proposta foi implementado o controlador LQR para descrever as trajetórias do centro de massa e o controlador pro-

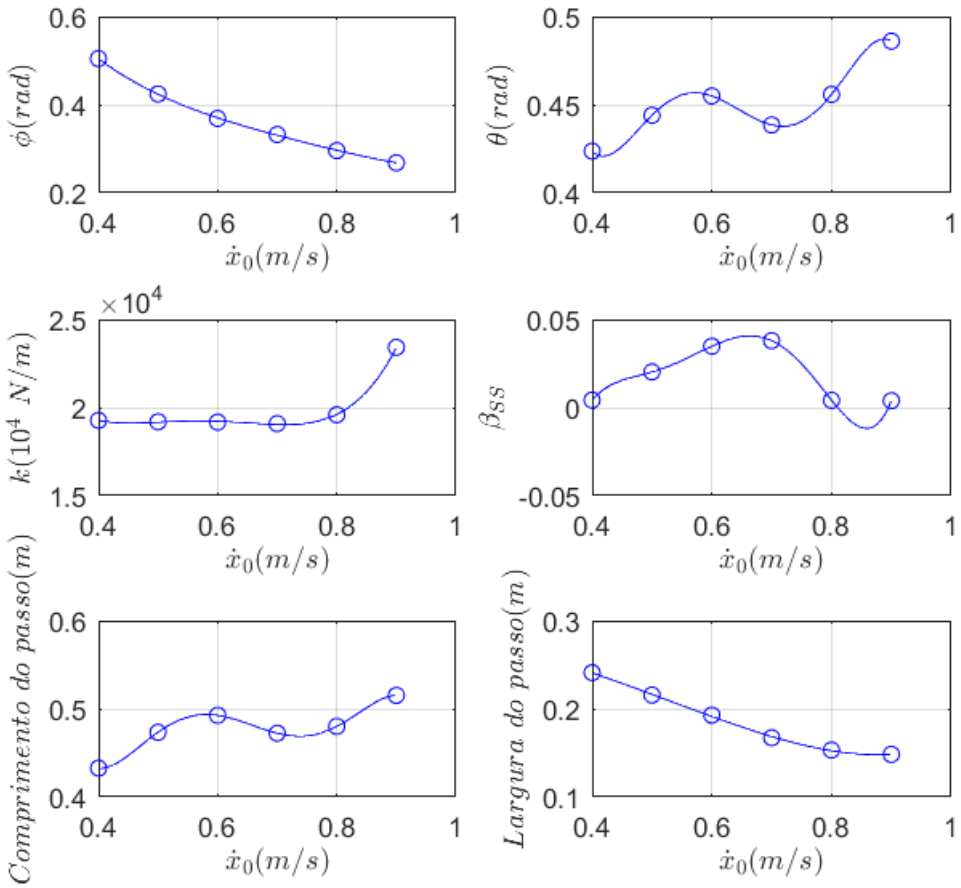


Figura 6.5: Parâmetros obtidos a partir do modelo 3D Dual-SLIP para diversas velocidades sobre o robô Hubo

porcional com um *feedfoward* para o percurso dos pés que estão em movimento durante a caminhada e avaliar seus resultados. Ressaltando, as trajetórias dos pés em movimento, estas foram geradas utilizando uma curva de Bézier de 4ª ordem, relativa ao CoM. Os controladores cinemáticos foram implementados assim como descritos no capitulo 4. Para implementar o controlador LQR é necessário resolver as equações 4.34 e 4.35, sendo estas, contendo a equação de Ricatti. Estas equações são não lineares, e uma solução através de uma equação específica pode não ser encontrada em muitos casos. Como consequência, aproximações numéricas são utilizadas usando diferenças finitas , onde temos que:

$$\frac{\partial \mathbf{P}(t)}{\partial t} \approx \frac{\mathbf{P}(t) - \mathbf{P}(t-\tau)}{\tau} \tag{6.2}$$

$$\frac{\partial \xi(t)}{\partial t} \approx \frac{\xi(t) - \xi(t-\tau)}{\tau} \tag{6.3}$$

O intervalo de amostragem τ afeta a precisão da aproximação. Como queremos obter as matrizes de ganho, $\mathbf{P}(t)$, para todo intervalo de tempo t. Começamos com o valor

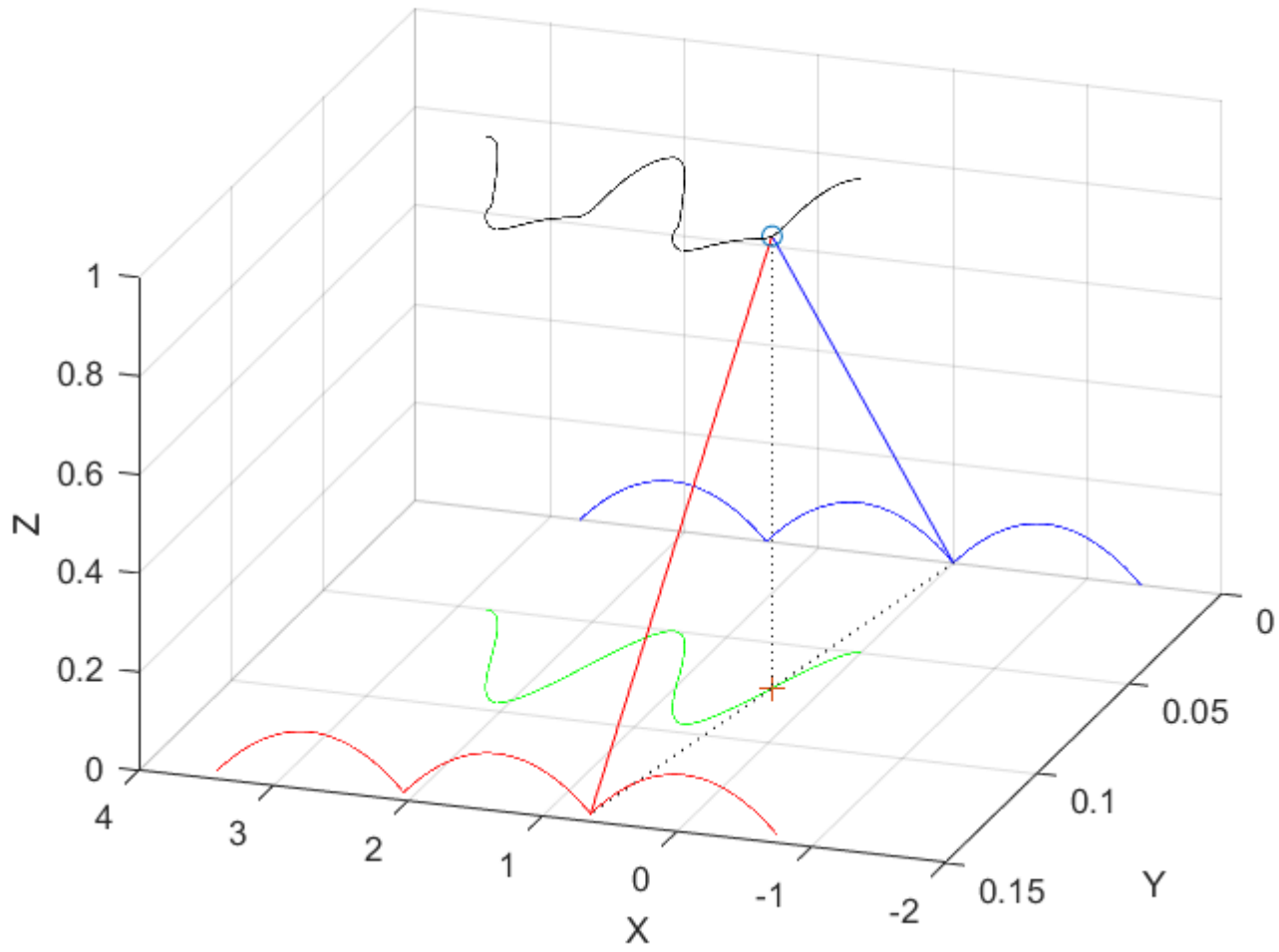


Figura 6.6: trajetórias geradas pelo modelo 3D-Dual SLIP para 2 MS.

$\mathbf{P}(t_f)$ e usamos a derivação regressiva na trajetória até $\mathbf{P}(0)$. Os valores de $\mathbf{A}(t)$, $\mathbf{A}(t) = \bar{\mathbf{H}}_8(\underline{\mathbf{h_d}}^*\dot{\underline{\mathbf{h}}}_d)$, é conhecido para todo o intervalo de tempo, sendo dependente apenas da trajetória desejada. A mesma abordagem pode ser aplicada para encontrar uma solução aproximada para $\xi(t)$, sendo que $c(t) = -\underline{\mathbf{h_d}}^*\dot{\underline{\mathbf{h}}}_d$, é conhecido para todo intervalo de tempo. Todos estes cálculos podem ser realizados antes do sistema iniciar a operação e as matrizes de ganho P e o vetores ξ são armazenados. Os parâmetros de projeto do controlador são as matrizes $\mathbf{Q}$, $\mathbf{S}$, $\mathbf{R}$. As matrizes $\mathbf{Q}$ e $\mathbf{S}$, estão relacionados com a norma do erro da trajetória. A fim de simplificar a escolha de parâmetros, foram definidos as variáveis s, q, $r \in \mathrm{R}$, tal que

$$\mathbf{S} = s\mathbf{I} \quad \mathbf{Q} = q\mathbf{I} \quad \mathbf{R} = r\mathbf{I} \tag{6.4}$$

onde $\mathbf{I} \in \mathrm{R}^{8x8}$. As condições de contorno para resolver as equações 4.34 e 4.35 são obtidas como descritas na seção 4.4.

Um passo é composto de dois estados intermediários subsequentes, dois MS (*Midstance*). O estado intermediário ocorre durante a fase de contato único de um perna quando $\dot{z} = 0$. Considerando que o passo comece em MS, o centro de massa do modelo *3D Dual-SLIP* terá outros três eventos: primeiro a perna em movimento encosta no chão, definido como *touchdown* (TD), depois o centro de massa chega a posição mais baixa de sua trajetória, definido como *lowest height* (LH) e por último a perna que estava em suporte tira o contato com o chão, definido como *lift off* (LO). Seguindo LO, o modelo continua até chegar a posição de MS terminando um passo completo. Para testar o controlador, inicialmente será implementado para 1 passo, como descrito acima e seus resultados discutidos. Espera-se que o controlador faça o rastreamento da trajetória mostrada na Fig. 6.1. A trajetória do CoM foi dividida em duas partes sendo a primeira descrita do estado inicial de MS para o estado LH, e a segunda parte do LH até o próximo MS. Supondo a perna direita em contato inicial com o solo, e a perna esquerda sendo a perna em movimento, a primeira trajetória é descrita com um controlador LQR para a perna direita e um controlador proporcional para a perna esquerda até que a condição de LH seja satisfeita. Quando o modelo chegar na posição de LH, a perna esquerda assume o contato com o chão e a perna direita assume como a perna em movimento, sendo assim, utiliza-se um controlador LQR para a perna esquerda e um controlador proporcional com um *feedforward* para a perna direita. Para a trajetória foi considerado que a orientação do CoM seria a mesma durante toda a trajetória, sendo a mesma na sua configuração de repouso. A orientação dos pés foram consideradas sendo paralela ao chão durante todo o movimento.

A Fig. 6.7, descreve o comportamento do controlador LQR implementado para a perna direita quando está com o suporte no solo. Podemos observar que o controlador LQR conseguiu fazer o rastreamento para a trajetória para os eixos x, y e z com um erro de ordem de 10^{-2} m, e sua orientação , descrita pelo ângulo do quatérnio de rotação, apresenta um erro na ordem de 10^{-4} rad. A Fig. 6.8, descreve o comportamento do controlador proporcional implementado para a perna esquerda em movimento. Podemos observar que o controlador proporcional com um *feedfoward* conseguiu fazer o rastreamento para a trajetória para os eixos x, y e z com um erro de ordem de 10^{-3} m. Sua orientação, demostrada pelo ângulo do quatérnio de rotação, apresenta um pequeno desvio no início da trajetória de ordem de 10^{-5} rad, estabilizando no decorrer da trajetória.

A Fig. 6.9, descreve o comportamento do controlador LQR implementado para a perna esquerda quando está com o suporte no solo. Podemos observar que o controlador LQR conseguiu fazer o rastreamento para a trajetória para os eixos x, y e z apresentando um erro de ordem de 10^{-2} m, e sua orientação , demonstrada pelo ângulo do quatérnio de rotação, apresentando um erro na ordem de $10^{-7} rad$. A Fig. 6.10, descreve o comportamento do controlador proporcional implementado para a perna direita, quando está em movimento. Podemos observar que o controlador proporcional com um *feedfoward* conse-

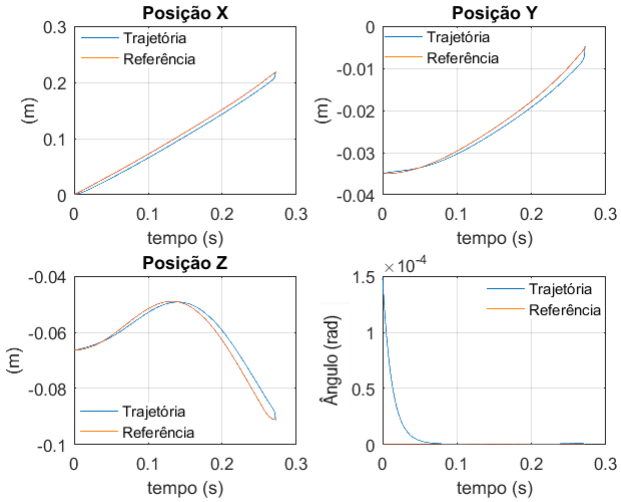


Figura 6.7: Controlador LQR para a perna de suporte.

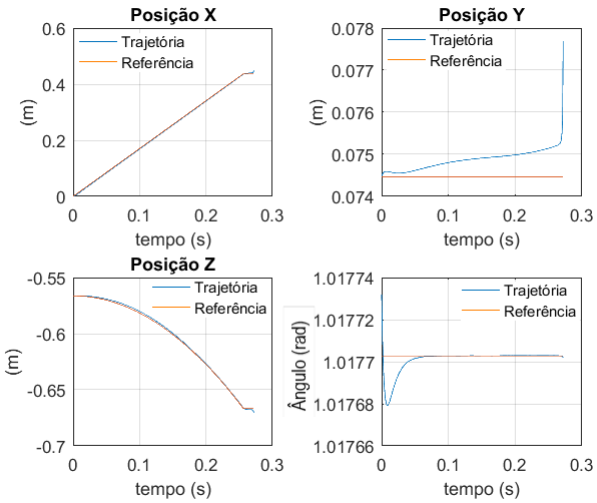


Figura 6.8: Controlador Proporcional com um *feedfoward* para a perna em movimento.

guiu realizar o rastreamento da trajetória para os eixos x, y e z apresentando de ordem de 10^{-3} m. Sua orientação, demostrada pelo ângulo do quatérnio de rotação, apresenta um erro na ordem de 10^{-4} rad.

Em alguns testes foi possível observar um comportamento indesejado do controlador LQR, apresentando picos em alguns pontos da trajetória. Dependendo da trajetória escolhida, para que a cadeia cinemática execute, em alguns pontos a matriz $\mathbf{N}$ apresenta-se mal condicionada, aproximando-se de zero. Nesses pontos, a pseudo-inversa assume valores muito altos, o que satura as juntas do robô. Considerando apenas para análise, sabendo que não seria possível um retorno do robô para tal comportamento , bem como apenas para discutir a respeito do controlador, podemos observar que a medida que esse evento ocorre, o controlador consegue estabilizar novamente quando está longe dos pontos de singularidade. No entanto, quando tal evento ocorre, o controlador apenas consegue

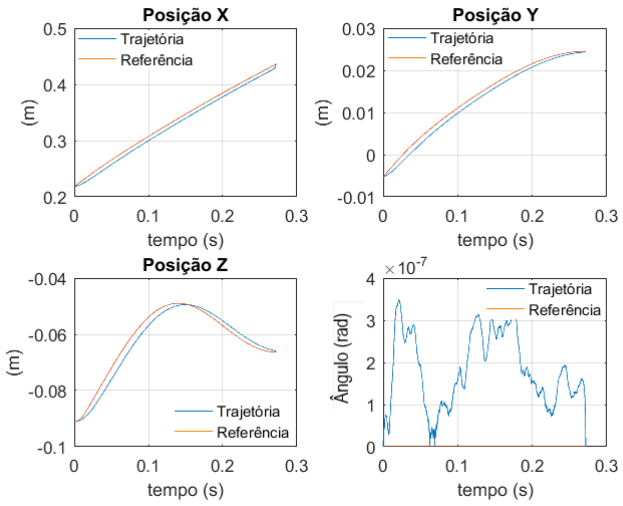


Figura 6.9: Controlador LQR para a perna de suporte.

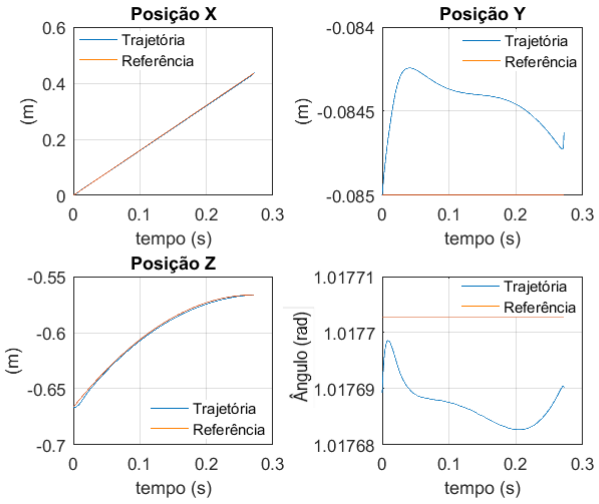


Figura 6.10: Controlador Proporcional com um *feedfoward* para a perna em movimento.

convergir se este for devidamente parametrizado, $\mathbf{R}$, $\mathbf{Q}$, $\mathbf{S}$. Em Vargas, Leite e Costa (2013), cita-se uma possível solução para o comportamento mencionado anteriormente, com a aplicação da inversa filtrada, sendo esta não discutida neste trabalho. Neste trabalho mostra uma solução para contornar singularidades, comparando-a com técnicas mais comumente utilizadas (Wampler (1986)), no entanto sua aplicação é sujeita a algumas restrições, discutidas em Vargas, Leite e Costa (2013). Na Fig. 6.11 é mostrado o caso ocorrido para uma trajetória de MS para LH, ou seja, metade de um passo completo. Os picos mostrados na Fig. 6.11, ocorre por que a matriz Jacobiana possui uma linha de zeros nesta configuração do robô, o que leva a valores da matriz pseudo inversa de $\mathbf{N}$ para valores muito altos, da ordem de 10^{12}, no entanto o controlador consegue estabilizar no decorrer do tempo. Dado a configuração inicial, cada junta pode movimentar no intervalo de $[-\pi/2, \pi/2]$. Como mostrado na Fig. 6.11 o mal condicionamento de N levou a um erro

inicial de cerca de 50 cm na direção x, -1 cm em y e cerca de 60 cm em z. A orientação teve um pico de $0,8$ rad , no entanto, com o passar do tempo, saindo da região inicial o controlador converge para a trajetória, realizando seu rastreamento.

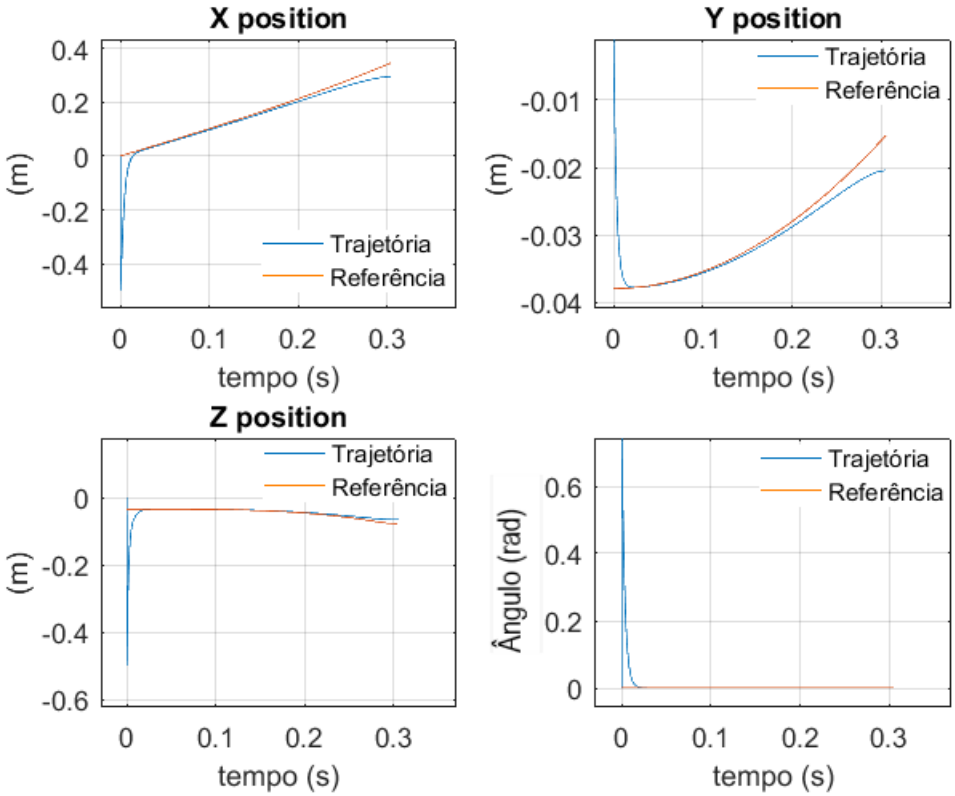


Figura 6.11: Erro na trajetória, picos observados e convergência

Com base nesses resultados, observa-se que o controlador LQR no espaço dos quatérnios duais, consegue realizar o rastreamento da trajetória do CoM do robô humanoide para um passo. As Figuras 6.12 e 6.13 mostram o comportamento dos controladores para vários passos.

A Fig. 6.12 mostra a translação do CoM no tempo. Em azul, está a representação do deslocamento do CoM devido a perna direita, quando esta se encontra em contato com o chão. Em vermelho, está a representação do movimento do CoM, quando a perna esquerda esta em contato com o chão. A curva de referência está representada pela cor preta. A análise do gráfico, torna possível observar, que o controlador LQR para ambas pernas, apresenta um comportamento satisfatório para o rastreamento de trajetórias, apresentando erros na ordem de 10^{-3} para x, y e z e para a orientação representada pelo ângulo do quatérnio de rotação um erro da ordem de $10^{-5}/rad$.

A Fig. 6.13 mostra a evolução dos pés no tempo. Em azul, está representado o deslocamento do pé direito, e em vermelho está representado o movimento do pé esquerdo. A curva de referência está representada pela cor preta. A análise do gráfico, torna possível

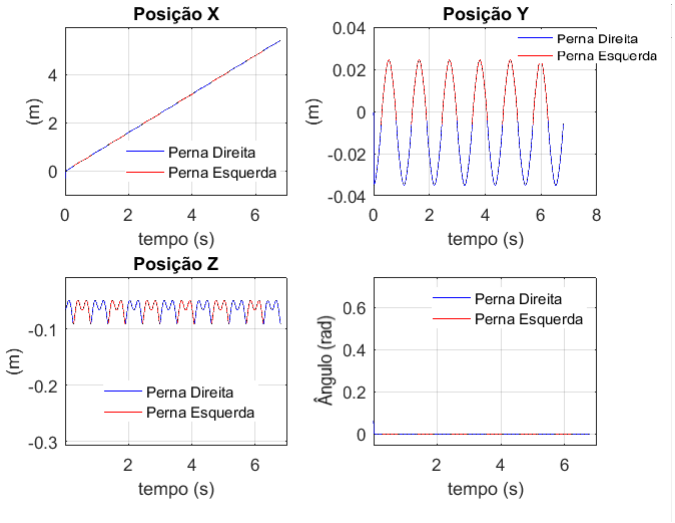


Figura 6.12: Trajetória do CoM para vários passos.

observar que o controlador Proporcional com um *feedfoward* para ambas pernas, apresenta um comportamento satisfatório para o rastreamento de trajetórias dos pés, apresentando erros da ordem de 10^{-3} m para a trajetória no decorrer do tempo. No eixo y apresenta pequenos distúrbios na trajetórias da ordem de 10^{-1} m, no entanto, apresenta convergência no decorrer do tempo. Observa-se que este controlador apresenta, uma convergência com o tempo para a orientação, representada pelo ângulo do quatérnio de rotação. Na Fig. 6.13 da orientação é possível observar um pico no inicio, este pico ocorre devido um alto ganho das variáveis de junta naquele determinado momento. Por hipótese, uma possível causa deste comportamento, deve-se ao mal condicionamento da matriz $\mathbf{N}$, bem como ao cálculo da pseudo inversa desta, quando apresenta um ganho alto para as variáveis de junta. No entanto, este erro é corrigido no decorrer da trajetória.

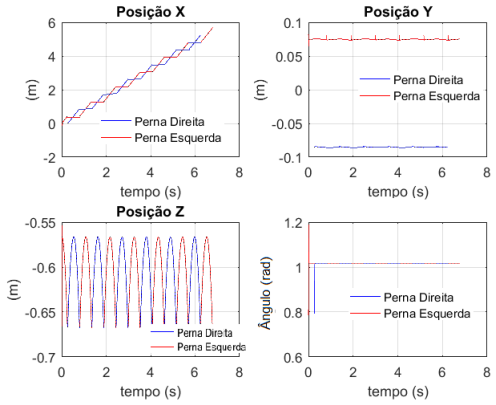


Figura 6.13: Trajetória dos pés para vários passos.

6.1 Conclusões Parciais

Neste capítulo foi demostrado como obter as trajetórias utilizando o modelo *3D DUAL SLIP*, adaptando aos parâmetros do robô HUBO. O controlador LQR no espaço dos quatérnios duais proposto para descrever as trajetórias do CoM conseguiu realizar o rastreamento desta, apresentando um erro da ordem de 10^{-2} m e o erro de orientação perto de zero. A utilização do controle LQR é uma vantagem, pois este explora o conhecimento futuro da trajetória desejada. Para a trajetórias dos pés o controlador proporcional com um *feedfoward* foi avaliado, o qual apresentou comportamento satisfatório para o rastreamento das trajetórias dos pés, apresentando um erro da ordem de 10^{-3} m, sendo o erro de orientação perto de zero. Dado o tempo para o desenvolvimento do trabalho, não foi possível realizar uma comparação entre os dois controladores, sendo neste trabalho avaliado o comportamento do controlador LQR para descrever a trajetória do CoM e o controlador proporcional com um *feedfoward* para o percurso dos pés.

Conclusões Finais

Este trabalho abordou a modelagem do robô humanoide HUBO, o planejamento de trajetórias do CoM utilizando o modelo *3D Dual SLIP*, o desenvolvimento de um controlador LQR e o controlador proporcional com um *feedforward*, sendo os controladores implementados no espaço dos quatérnios duais. Tende em vista que, os controladores implementados tiveram o objetivo de garantir que o robô humanoide modelado, descrevesse a trajetória obtida pelo modelo *3D DUAL-SLIP*, sendo este modelo, capaz de gerar padrões similares à caminhada humana, assim como descrito em Liu et al. (2015). Os resultados expostos, mostraram que a aplicação do LQR apresenta-se viável para o rastreamento do percurso descrito pelo CoM do robô. Visto que, a modelagem, utilizando a álgebra de quatérnios duais, apresentou vantagens para a modelagem do robô, bem como, no projeto dos controladores cinemáticos utilizando a função de erro invariante. Como proposta inicial foi implementado o controlador LQR para descrever as trajetórias do centro de massa e o controlador proporcional com um *feedfoward* para o percurso dos pé que está em movimento durante a caminhada. Devido o prazo para o desenvolvimento deste trabalho, não foi possível realizar uma comparação entre os controladores, sendo apenas estudados os resultados destes controlados para sua aplicação da proposta inicial. O controlador LQR foi implementado para realizar o rastreamento das trajetórias do CoM, o qual apresentou um erro da ordem de 10^{-2} m e erro de orientação próximo de zero, ordem de 10^{-5} rad. A utilização do controle LQR é uma vantagem, pois este explora o conhecimento futuro da trajetória desejada e computa os ganhos para cada ponto da trajetória, bem como é capaz de incorporar distúrbios e variações da trajetória. O controlador proporcional com um *feedfoward*, foi responsável por descrever o percurso descrito pelo pé em movimento, sendo este controlador capaz de realizar esta tarefa, apresentando um erro da ordem de 10^{-3} e erro de orientação próximo de zero, ordem de 10^{-5} rad. Apesar do controlador proporcional levar em consideração as trajetórias variantes no tempo e de ser um controlador de simples implementação, este não considera distúrbios podendo ser um problema em uma aplicação. Como contribuição este trabalho disponibiliza toda a codificação utilizada, além de abordar a implementação dos controladores cinemáticos aplicados em um robô humanoide, sendo que grande parte dos estudos foram utilizados apenas em manipuladores robóticos. Contudo, espera-se de trabalhos futuros, implementar o objeto de estudo deste trabalho em um robô humanoide, como forma de validar a pesquisa, bem como, estudos mais aprofundados sobre o modelo *3D DUAL SLIP* e sua utilização para modelagem de robôs humanoides.

Referências

ADORNO, B. V. *Two-arm Manipulation: From Manipulators to Enhanced Human-Robot Collaboration*. 2011. Université Montpellier II - Sciences et Techniques du Langue-doc, 2011. English <tel-00641678>. Citado 8 vezes nas páginas 13, 31, 33, 34, 36, 37, 38 e 39.

ALI, M. A.; PARK, H. A.; LEE, C. S. G. Closed-form inverse kinematic joint solution for humanoid robots. *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 704–709, 2010. Citado 5 vezes nas páginas 13, 57, 58, 59 e 60.

ASPRAGATHOS, N.; DIMITROS, J. A comparative study of three methods for robot kinematics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, p. 135–145, 1998. Citado na página 29.

ENGLSBERGER, J.; OTT, C.; ALBU-SCHAFFER, A. Three-dimensional bipedal walking control using divergent component of motion. *IEEE/RSJ Int. Conf. on Intelligent Rob. and Sys.*, p. 2600–2607, 2013. Citado na página 28.

FULL, R. J.; KODITSCHEK, D. E. Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *J. Exp. Biol., vol. 202, no. 23*, p. 3325–3332, 1999. Citado na página 27.

FUNDA, R. T. J.; PAUL, R. On homogeneous transforms, quaternions, and computational efficiency. *Robotics and Automation, IEEE Transactions*, p. 382–388, 1990. Citado na página 29.

GEYER, H.; SEYFARTH, A.; BLICKHAN, R. Compliant leg behaviour explains basic dynamics of walking and running. *Proc. of the Royal Society B: Bio. Sciences*, p. 2861–2867, 2006. Citado na página 28.

KAJITA, S. et al. Biped walking pattern generation by using preview control of zero-moment point. *IEEE Int. Conf. on Robotics and Automation*, p. 1620–1626, 2003. Citado na página 28.

KAJITA, S.; MATSUMOTO, O.; SAIGO, M. Real-time 3d walking pattern generation for a biped robot with telescopic legs. *IEEE Int. Conf. on Robotics and Automation*, p. 2299–2306, 2001. Citado na página 28.

KENWRIGHT, B. A beginners guide to dual-quaternions what they are, how they work, and how to use them for 3d character hierarchies. *???*, p. ?, 2006. Citado na página 29.

LIU, Y. et al. Dynamic walking in a humanoid robot based on a 3d actuated dual-slip model. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, p. 5710 – 5717, 2015. Citado 14 vezes nas páginas 13, 26, 27, 28, 30, 41, 44, 46, 47, 48, 49, 50, 57 e 75.

MARINHO, M. M.; FIGUEREDO, L. F. C.; ADORNO, B. V. A dual quaternion linear-quadratic optimal controller for trajectory tracking. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 4047–4052, 2015. Citado 3 vezes nas páginas 51, 54 e 55.

MORRIS, B.; GRIZZLE, J. A restricted poincaré map for determining exponentially stable periodic orbits in systems with impulse effects: Application to bipedal robots. *In 44th IEEE Conference on Decision and Control and Euro- pean Control Conference (CDC-ECC*, p. 4199–4206, 2015. Citado na página 49.

PAUL, R. P. Robot manipulators: Mathematics, programming, and control. *MIT Press, Cambridge, MA, USA,*, 1982. Citado na página 29.

PEREIRA, M. S. *Trajectory Control of Anthropomorphic Compliant Manipulator with Dual Quaternion Based Kinematic Controllers*. 2016. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-n019, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 140p. Citado na página 52.

PIEPER, D. L. *The kinematics of manipulators under computer control*. 1968. Ph.D. dissertation, Stanford Univ. Citado na página 58.

PRATT, J. et al. Capture point: A step toward humanoid push recovery. *IEEE/RAS Int. Conf. on Humanoid Robots*, p. 200–207, 2006. Citado na página 28.

RUDER, S. An overview of gradient descent optimization algorithms. 2017. Citado na página 63.

SPONG, M.; HUTCHINSON, S.; VIDYASAGAR, M. *Robot Modeling and Control*. [S.l.: s.n.], 2005. Citado na página 52.

VARGAS, L. V.; LEITE, A. C.; COSTA, R. R. Kinematic control of robot manipulators using filtered inverse. *Control & Automation (MED), 2013 21st Mediterranean Conference on*, 2013. Citado na página 71.

VAUGHAN, C. L.; O'MALLEY, M. J. Froude and the contribution of naval architecture to our understanding of bipedal locomotion. *Gait & Posture*, p. 350–362, 2005. Citado na página 57.

WAMPLER, C. W. Manipulator inverse kinematic solutions based on vector formulations and damped lesat-squares methods. *IEEE Transactions on Systems, Man, and Cybernetics (Volume: 16, Issue: 1, Jan. 1986)*, p. 93 – 101, 1986. Citado 2 vezes nas páginas 52 e 71.

WENSING, P. M.; ORIN, D. E. High-speed humanoid running through control with a 3d-slip model. *IEEE/RSJ Int. Conf. on Intelligent Rob. and Sys.*, p. 5134–5140, 2013. Citado na página 28.

WENSING, P. M.; ORIN, D. E. Development of high-span running long jumps for humanoids. *IEEE/RSJ Int. Conf. on Intelligent Rob. and Sys.*, p. 222–227, 2014. Citado na página 28.

WU, Y. et al. Strapdown inertial navigation system algorithms based on dual quaternions. *IEEE transactions on aerospace and electronic systems*, p. 110–132, 2005. Citado na página 29.

XIAN, B. et al. Task-space tracking control of robot manipulators via quaternion feedback. *Robotics and Automation, IEEE Transactions*, p. 160–167, 2004. Citado na página 29.

Anexos

ANEXO A – Trabalhos Anteriores

Esta seção descreve o trabalho desenvolvido pela equipe PMec Humanoide do Núcleo de Robótica Pequi Mecânico. O núcleo desenvolveu o projeto com a intenção em participar da Competição Latino-Americana Robótica (LARC) em 2015 na categoria IEEE Humanoid Robot Racing Child Size http://www.cbrobotica.org/?page_id=91, contudo, o projeto sofreu constantes evoluções ao longo deste tempo. Nesta categoria, um robô humanoide deve ser capaz de mover-se, em uma linha reta , em uma distância superior a 4 metros em menos de 3 minutos. Tendo em vista resolver este problema, um pequeno robô humanoide , que pode se orientar e caminhar de forma autônoma, foi exigido, sendo que, a estabilidade do robô se torna um grande desafio, enquanto este se movimenta. Neste sentido foi desenvolvidos conceitos de coordenação e equilíbrio de robôs humanóides ao projeto. Um robô humanóide é um robô multitarefa que detém características dos seres humanos, tais como cabeça, tronco braços e pernas. Sendo assim, podem andar sobre duas pernas, usar as suas mãos para manipular objetos do ambiente no qual se encontra. O robô construído para o projeto, detém técnica computacional de análise visual, não tendo qualquer tipo de comunicação ou intervenção humana caracterizando-se assim como um dispositivo móvel autônomo com capacidade de análise visual do ambiente. O sistema de processamento embarcado das imagens do robô detém capacidade de reconhecimento do ambiente, obtendo dados para ajustar sua trajetória.

O IEEE Humanoid Robot Racing ocorre em uma arena de $1.0m \times 5.0m$ de madeira MDF com espessura de 15 mm, na cor branca, conforme Fig. A.1. O robô deve ser capaz de se locomover, superando o percurso em um tempo de até 3 minutos.

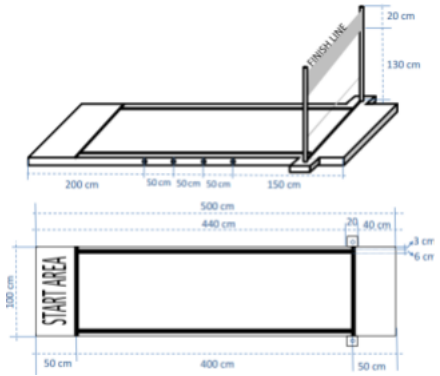


Figura A.1: Arena.

Para a construção da estrutura mecânica do robô foram adquiridas algumas peças

do kit comercial Biped Robot 17 DOF (Fig. A.2), ao qual foi adaptado ao projeto. A princípio, o hardware foi projetado contendo, um Raspberry Pi, microcontroladores atmega 168, servos motores MG996r, mini servos motores 9g, uma webcam, baterias Li-Po e uma placa de circuito integrado (projetada pela equipe). O robô projetado é mostrado na Fig. A.3, sendo este robô com 13 graus de liberdade, uma altura de $40cm$, largura entre os braços de $47,5cm$ e o tamanho da perna de $17cm$.

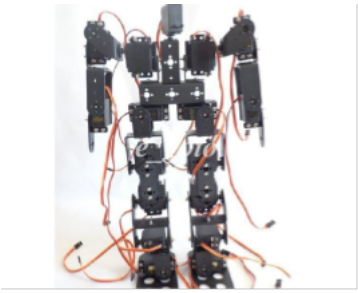


Figura A.2: Biped Robot 17 DOF

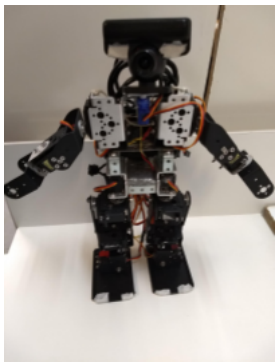


Figura A.3: Projeto Humanóide do Núcleo de Robótica Pequi Mecânico 2017

Sob a necessidade de obter maior controle dos motores, optou-se por desenvolver novos servos motores utilizando o motor e a caixa de redução do servo motor MG996r comercialmente vendido e uma placa controladora desenvolvida pela equipe. A placa controladora consiste em uma ponte H, um acionador para a ponte H e um microcontrolador. A ponte H possui 2 mosfets canal P e 2 mosfets canal N, que operava o motor de corrente contínua a 31250 Hz, assim controlando a velocidade do motor sem grandes perdas. Além dos mosfets foram utilizados 4 diodos schottky para suprimir a corrente reversa gerada no motor e dois capacitores de bypass para eliminar ruídos gerados devido a alta frequência de acionamento dos motores. O acionador para a ponte H tinha o intuito fazer a ligação correta e segura entre o microcontrolador e a ponte H, já que ambos trabalham em níveis lógicos diferentes, a ponte H entre $0V$ e $8,4V$ e o microcontrolador entre $0v$ e $3,3v$. Além disso esse acionador permitia controlar a velocidade e a direção do motor utilizando apenas uma saída PWM do microcontrolador. O microcontrolador consiste em um STM32F103 que possui uma arquitetura ARM de 32-bit com 72 MHz. Esse microcontrolador é capaz de fazer uma conversão de analógico para digital em um microsegundo

com 12-bit de resolução, assim efetuando uma rápida e precisa leitura das posições dos 12 servos motores do projeto. A bateria usada era de polímero de lítio com peso de 138 gramas e $2200mAH$. O sistema de visão foi implementado com uma web cam e conectado via porta USB do Raspberry Pi, onde foram extraídas e pré-processadas imagens usadas como entrada de um algoritmo de fluxo óptico, que por sua vez, tem a função de detectar vetores de movimentação em pontos pré-determinados(cantos e/ou bordas), se um objeto se encontra a uma distância maior, os vetores gerados a partir desta malha tem módulos menores em comparação com vetores gerados a partir de objetos mais próximos. O conjunto dos vetores formam o que chamamos de fluxo óptico, um mapa que diz o sentido que uma superfície se movimenta entre os frames, com base nesse mapa podemos determinar onde não ir, assim, calculamos uma trajetória de maior confiabilidade. O controle do robô, iniciou-se com testes baseados em uma máquina de estados e seu desenvolvimento foi em malha aberta (sem leitura dos acelerômetros e giroscópios), baseando-se apenas nos posicionamentos dos servo motores, tendo em vista que, continuamente a posição dos servos eram monitoradas até alcançarem os valores de terminação para a mudança de estado. Os valores para os terminadores dos estados foram ajustados empiricamente, bem como as funções que atuam nos servo-motores a fim de realizar os movimentos desejados. Posteriormente, utilizando o sistema de visão, utilizando o resultado do processamento de imagem para corrigir a orientação da trajetória.

ANEXO B – Códigos

B.1 mainControle.m

```
%-------------------------------------------------------------
%Método principal para o controle LQR
%-------------------------------------------------------------
%-------------------------------------------------------------
%iniciar, limpar memória, fechar todas as janelas
%-------------------------------------------------------------
clc
close all
clear all
format short
%-------------------------------------------------------------
%adicionar as libs necessárias
%-------------------------------------------------------------
addpath('debug')
addpath('derivadas')
addpath('edoSolve')
addpath('graphics')
addpath('modelos')
addpath('otimizacaoTrajetoria')
addpath('trajetoriaCoM')
addpath('trajetoriaPes')
addpath('quaternion_library');
addpath('dual_quaternion_library');
addpath('kinematic_dualquartenion_library');
addpath('transformation');
addpath('dif_Kinematic');
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
global  m L g  h hEdo
m    = 38.1;    %massa
```

```
L    = 0.63;    %tamanho da perna
g    = 9.8;     %gravidade
h    = 10^-4;   %passo para o calculo das derivadas
hEdo = 10^-3.85;%passo para calculo - EDO
pfa = [0;0;0];  %posição do pé de suporte em MS

%-------------------------------------------------------------
%condição inicial para MS
%-------------------------------------------------------------
xod  = 0.00;%x inicial
yod  = 0.05;%y inicial
zod  = 0.6;%z inicial
dxod = 0.7;%velocidade desejada no MS
dyod = 0.00;%condição de balanço
dzod = 0.00;%velocidade em z (igual a zero condição necessaria)

expK   = 10000;%ordem de grandeza da constante massa-mola

%-------------------------------------------------------------
%variável de controle inicial
%modificar os valores obtidos aqui
%-------------------------------------------------------------

%1m/s
phi = 0.3408040779;
theta = 0.4052906627;
k = 19196.8680322965;
Bss = 0.0415640571;

%Ganhos LQR (perna direita) e proporcional (perna esquerda)
ganhoS1 = 0;
ganhoQ1 = 1;
ganhoR1 = 0.000001;
ganhoK1 = 1000;

%Ganhos LQR (perna esquerda) e proporcional (perna direita)
ganhoS2 = 0;
ganhoQ2 = 1;
ganhoR2 = 0.000001;
```

```
ganhoK2 = 1000;

%quantidade de passos
quatidadePassos = 3;
%-------------------------------------------------------------
%Não modificar aqui
%-------------------------------------------------------------

k = k/expK;%tratar o valor
U(1,1) = theta;
U(2,1) = phi;
U(3,1) = k;
U(4,1) = Bss;
U(5,1) = expK;

X  = [xod;yod;zod;dxod;dyod;dzod];%condição inicial
vecGanho1 = [ganhoS1 ganhoQ1 ganhoR1 ganhoK1]';
vecGanho2 = [ganhoS2 ganhoQ2 ganhoR2 ganhoK2]';

caminhada(U,X,quatidadePassos,vecGanho1,vecGanho2)
```

B.2 mainOtimizacao.m

```
%-------------------------------------------------------------
%Método principal para a otimização do modelo
%retornando no console os valores para as variáveis de controle
%Modelo implementado:
%3D Dual-SLIP
%Otimizações implementadas:
%metodo : 0 estocástico  gradiente descendente SGD
%metodo : 1 Nesterov accelerated gradient
%metodo : 2 momento
%metodo : 3 Adagrad
%Variável de Controle:
%U = [theta,phi,k,BSS];
%theta - ângulo de abertura frontal da perna
%phi   - ângulo de abertura lateral da perna
%k     - constante da massa mola do modelo
%BSS   - taxa de crescimento linear da perna durante o passo
```

```
%-------------------------------------------------------------


%-------------------------------------------------------------
%iniciar, limpar memória, fechar todas as janelas
%-------------------------------------------------------------
clc
close all
clear all
%-------------------------------------------------------------
%adicionar as libs necessárias
%-------------------------------------------------------------
addpath('debug')
addpath('derivadas')
addpath('edoSolve')
addpath('graphics')
addpath('modelos')
addpath('otimizacaoTrajetoria')
addpath('trajetoriaCoM')
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
global  m L g lstep pfa thetaM phiM KM expK BSSM
%massa do corpo
m = 38.1;
%tamanho da perna
L = 0.63;
%gravidade
g = 9.8;
%posição do pé de suporte em MS
pfa = [0;0;0];
%-------------------------------------------------------------
%Numero maximo de iterações para o metodo do gradiente
%-------------------------------------------------------------
global maxNGrad ganhoAlpha  gamma h hEdo
maxNGrad   = 10^6;%número máximo de iterações método
ganhoAlpha = 10^-2;%ganho do fator de ganho para cada passo
gamma      = 0.2 ;%ganho para os método gradiente(momento)
h          = 10^-4;%passo para o calculo das derivadas
%calculado (comparado com a função do matlab)
```

```
hEdo       = 10^-3.85;%passo para o calculo das derivadas
%-------------------------------------------------------------
%condição inicial para MS
%-------------------------------------------------------------
%hubo 2 + velocidade maxima 0.4 m/s

xod  = 0.00;%x inicial (d - desejado)
yod  = 0.05;%y inicial
zod  = 0.6;%z inicial (muito pequeno apens alguns centimetros)
dxod = 0.4;%velocidade desejada no MS
dyod = 0.00;%condição de balanço
dzod = 0.00;%velocidade em z (igual a zero condição necessaria)

%-------------------------------------------------------------
%Tamanho do passo (não sendo usado ainda estudar como incorporar
%esse dado)
%-------------------------------------------------------------
lstep = 0.5 + 0.1*(dxod - 1);
%-------------------------------------------------------------
%valores máximos das variáveis de controle
%reduzir o espaço de busca
%-------------------------------------------------------------
thetaM = 0.5;
phiM   = 0.5;
KM     = 2;
expK   = 10000;%ordem de grandeza da constante massa-mola
BSSM   = 0.2;
%params = [thetaM;phiM;KM;expK;BSSM];
%-------------------------------------------------------------
%variavel de controle inicial
%modificar os valores obtidos aqui
%Geralmente na literatura são usados x para variável de estado
%e u para a variável de controle sendo usados essas letras
%em maiusculo para representação
%-------------------------------------------------------------

phi = 0.5000000000;
theta = 0.3801423352;
k = 19611.4821640244;
```

```
Bss = 0.0275951012;

k = k/expK;%tratar o valor
U(1,1) = theta;
U(2,1) = phi;
U(3,1) = k;
U(4,1) = Bss;
U(5,1) = expK;
%-------------------------------------------------------------
%condição inicial constante
%-------------------------------------------------------------
X  = [xod;yod;zod;dxod;dyod;dzod];
%-------------------------------------------------------------
%Executa o metodo de otimização
%passando o vetor de controle U
%e o vetor da condição inicial X
%tipo: 1 para executar a otimização e plotar a trajetoria dos
%dos valores obtidos pela otimização
%tipo:0 apenas plotar a trajetoria dos valores de U ja estipulados
%-------------------------------------------------------------
tipo = 1;%recebe 0 ou 1 (alterar para executar a otimização ou não)
%metodo : 0 estocástico  gradiente descendente SGD
%metodo : 1 SGD com momento
%metodo : 2 Nesterov accelerated gradient
%metodo : 3 Adgrad
metodo = 3;
otimizacao(U,X,tipo,metodo);
```

B.3 imprimirConsole.m

```
%-------------------------------------------------------------
%Imprime no console os valores - usado para verificar o código
%Parâmetros
%iteracao - número da iteração que deseja imprimir no console
%vec      - vetor com os valores a serem imprimidos
%theta - vec(1,1)
%phi   - vec(2,1)
%K     - vec(3,1)
```

```
%BSS   - vec(4,1)
%fo    - vec(5,1)
%-------------------------------------------------------------
function imprimirConsole(iteracao,vec)
%-------------------------------------------------------------
%separar os valores do vetor
%-------------------------------------------------------------
     theta = vec(1,1);
     phi   = vec(2,1);
     K     = vec(3,1);
     BSS   = vec(4,1);
     expK  = vec(5,1);
     fo    = vec(6,1);
%-------------------------------------------------------------
%imprimir no console
%-------------------------------------------------------------
     disp('-----------------------------------------------------------------')
     msg = sprintf('iteracao : %d:',iteracao);
     disp(msg);
     msg = sprintf('phi = %0.10f;',phi);
     disp(msg);
     msg = sprintf('theta = %0.10f;',theta);
     disp(msg);
     msg = sprintf('k = %0.10f;',K*expK);
     disp(msg);
     msg = sprintf('Bss = %0.10f;',BSS);
     disp(msg);
     msg = sprintf('FO : %0.10f',fo);
     disp(msg);
end
```

B.4 gradienteFuncao.m

```
%-------------------------------------------------------------
%Calcula o gradiente da função objetivo
%Parâmetros :
%X0 - Vetor com a condição inicial do sistema X (Constante)
%U0 - Vetor com a condição inicial das variáveis de controle u
%Retorno:
```

```
%grad -  vetor gradiente das variáveis de controle
%-------------------------------------------------------------
function grad = gradienteFuncao(X0,U0)
%-------------------------------------------------------------
%cálculo do gradiente para cada variável de controle
%-------------------------------------------------------------
    grad(1,1) = primeiraDerivadaGradiente(X0,U0,1);% linha referente a theta
    grad(2,1) = primeiraDerivadaGradiente(X0,U0,2);% linha referente a phi
    grad(3,1) = primeiraDerivadaGradiente(X0,U0,3);% linha referente a K
    grad(4,1) = primeiraDerivadaGradiente(X0,U0,4);% linha refetente a BSS
end
```

B.5 primeiraDerivadaGradiente.m

```
%-------------------------------------------------------------
%Cálculo da derivada da função objetivo dado a variação de
%um parâmetro
%derivada calculada pelo método das diferenças centradas
%utilizando 4 pontos, este método tem uma incerteza
%da O(h^4)
%Parâmetros :
%X0 - Vetor com a condição inicial do sistema (Constante)
%U0 - Vetor com a condição inicial das variáveis de controle u
%Retorno:
%d1 -  derivada
%-------------------------------------------------------------
function d1 = primeiraDerivadaGradiente(X0,U0,ind)
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
    global h
%-------------------------------------------------------------
%cálculo do primeiro ponto para 2h
%-------------------------------------------------------------
    y = U0(ind,1)+2*h;
    Y = U0(:,1);%copiando o vetor de controle original
    Y(ind,1) = y;%substituindo o valor com a variação na posição desejada
    [pa,pb,pc] = trajetoria(Y,X0);%cálculo dos pontos da trajetória
    a = funcaoObjetivo(pa,pb,pc);%calculo da função objetivo
```

```
%-------------------------------------------------------------
%cálculo do segundo ponto para h
%-------------------------------------------------------------
    y = U0(ind,1)+h;
    Y = U0(:,1);
    Y(ind,1) = y;
    [pa,pb,pc] = trajetoria(Y,X0);
    b = funcaoObjetivo(pa,pb,pc);
%-------------------------------------------------------------
%cálculo do terceiro ponto para -h
%-------------------------------------------------------------
    y = U0(ind,1)-h;
    Y = U0(:,1);
    Y(ind,1) = y;
    [pa,pb,pc] = trajetoria(Y,X0);
    c = funcaoObjetivo(pa,pb,pc);
%-------------------------------------------------------------
%cálculo do quarto ponto para -2h
%-------------------------------------------------------------
    y = U0(ind,1)-2*h;
    Y = U0(:,1);
    Y(ind,1) = y;
    [pa,pb,pc] = trajetoria(Y,X0);
    d = funcaoObjetivo(pa,pb,pc);
%-------------------------------------------------------------
%cálculo da derivada
%-------------------------------------------------------------
    d1 = (-a + 8*b  -8*c + d) / (12 * h)  + h^4;
end
```

B.6 DerivadaJacobiano.m

```
%-------------------------------------------------------------
%Cálculo da derivada do Jacobiano
%-------------------------------------------------------------
function q1 = DerivadaJacobiano(theta,hOrg,hP,tipo,CoM,ind)
%-------------------------------------------------------------
%variaveis globais
%-------------------------------------------------------------
```

```
    global h MDH
    thetar = theta(:,1);
    thetal = theta(:,2);

    or = thetar;
    ol = thetal;
%-------------------------------------------------------------
%cálculo do primeiro ponto para 2h
%-------------------------------------------------------------
    if tipo
        if CoM
            or(ind,1) = or(ind,1) + 2*h;
        else
            ol(ind,1) = ol(ind,1) + 2*h;
        end
    else
        if CoM
            ol(ind,1) = ol(ind,1) + 2*h;
        else
            or(ind,1) = or(ind,1) + 2*h;
        end
    end

    a = KinematicRobo([or ol],hOrg,hP,tipo,CoM);

    or = thetar;
    ol = thetal;
%-------------------------------------------------------------
%cálculo do segundo ponto para h
%-------------------------------------------------------------
    if tipo
        if CoM
            or(ind,1) = or(ind,1) + h;
        else
            ol(ind,1) = ol(ind,1) + h;
        end
    else
        if CoM
            ol(ind,1) = ol(ind,1) + h;
```

```
        else
            or(ind,1) = or(ind,1) + h;
        end
    end

    b = KinematicRobo([or ol],hOrg,hP,tipo,CoM);

    or = thetar;
    ol = thetal;
%-------------------------------------------------------------
%cálculo do terceiro ponto para -h
%-------------------------------------------------------------
    if tipo
        if CoM
            or(ind,1) = or(ind,1) - h;
        else
            ol(ind,1) = ol(ind,1) - h;
        end
    else
        if CoM
            ol(ind,1) = ol(ind,1) - h;
        else
            or(ind,1) = or(ind,1) - h;
        end
    end

    c = KinematicRobo([or ol],hOrg,hP,tipo,CoM);

    or = thetar;
    ol = thetal;
%-------------------------------------------------------------
%cálculo do quarto ponto para -2h
%-------------------------------------------------------------
     if tipo
        if CoM
            or(ind,1) = or(ind,1) - 2*h;
        else
            ol(ind,1) = ol(ind,1) - 2*h;
        end
```

```
    else
        if CoM
            ol(ind,1) = ol(ind,1) - 2*h;
        else
            or(ind,1) = or(ind,1) - 2*h;
        end
    end

    d = KinematicRobo([or ol],hOrg,hP,tipo,CoM);
%-------------------------------------------------------------
%cálculo da derivada
%-------------------------------------------------------------
    q1 = (-a + 8*b  -8*c + d)* (1/12 * h);

    r = getRotationDualQuat(q1);
    normq = norm(r);

    if normq ~=0
       q1 = q1*(1/normq);
    end
end
```

B.7 jacobianoCinematica.m

```
%-------------------------------------------------------------
%Cálculo do Jacobiano Analítico
%Parâmetros:
%theta - vetor com as variáveis de juntas
%hOrg  - quatérnio dual com a origem do sistema de referência
%hP    - quatérnio dual com a posição do pé de apoio
%tipo  - flag que indica se a cinemática é da perna direita ou esquerda
%CoM   - flag para indicar se é para computar a cinemática até o centro de
%  massa ou até o outro pé.
%-------------------------------------------------------------
function J = jacobianoCinematica(theta,hOrg,hP,tipo,CoM)
%-------------------------------------------------------------
%cálculo das derivadas para cada variável de controle
%-------------------------------------------------------------
    J(:,1) = DerivadaJacobiano(theta,hOrg,hP,tipo,CoM,1);
```

```
    J(:,2) = DerivadaJacobiano(theta,hOrg,hP,tipo,CoM,2);
    J(:,3) = DerivadaJacobiano(theta,hOrg,hP,tipo,CoM,3);
    J(:,4) = DerivadaJacobiano(theta,hOrg,hP,tipo,CoM,4);
    J(:,5) = DerivadaJacobiano(theta,hOrg,hP,tipo,CoM,5);
    J(:,6) = DerivadaJacobiano(theta,hOrg,hP,tipo,CoM,6);
end
```

B.8 getRotationDualQuat.m

```
%--------------------------------------
%Método para obter a rotação de um quatérnio dual
%Parâmetros:
%q: quatérnio dual
%Retorno:
%r: quatérnio de rotação (orientação)
%--------------------------------------
function r = getRotationDualQuat(q)
     r = [q(1,1) q(2,1) q(3,1) q(4,1)]';
end
```

B.9 getPositionDualQuat.m

```
%--------------------------------------
%Método para obter a posição x y z
%de um quatérnio dual
%Parâmetros:
%q: quatérnio dual
%Retorno:
%p: posição x y z
%--------------------------------------
function p = getPositionDualQuat(q)
     r = getRotationDualQuat(q);
     qd = [q(5,1) q(6,1) q(7,1) q(8,1)]';
     p = 2*quatMult(qd,quatConj(r));
     p = p(2:4,1);
end
```

B.10 dualQuatMult.m

```
%-------------------------------------
%Método para calcular o produto entre dois quatérnios duais
%Parâmetros:
%q:  quatérnio dual
%p:  quatérnio dual
%Retorno:
%qr: quatérnio dual resultante
%-------------------------------------
function qr = dualQuatMult(p,q)
     %separando p em parte primaria e dual
     pp = [p(1,1) p(2,1) p(3,1) p(4,1)]';
     pd = [p(5,1) p(6,1) p(7,1) p(8,1)]';
    %separando q em parte primaria e dual
     qp = [q(1,1) q(2,1) q(3,1) q(4,1)]';
     qd = [q(5,1) q(6,1) q(7,1) q(8,1)]';

     aux1 = quatMult(pp,qp);
     aux2 = quatAdd(quatMult(pp,qd),quatMult(pd,qp));
     %quatérnio dual resultante
     qr = [aux1;aux2];
end
```

B.11 dualQuatDot.m

```
%-------------------------------------
%Método para calcular o produto escalar
%entre dois quatérnio dual
%Parâmetros:
%q: dual quatérnio
%p: dual quatérnio
%Retorno:
%mag: produto escalar entre os quatérnios
%-------------------------------------
function mag = dualQuatDot(q,p)
    mag = q(1,1)*p(1,1) + q(2,1)*p(2,1) + q(3,1)*p(3,1) + q(4,1)*p(4,1);
end
```

B.12 dualQuatDH.m

```
%--------------------------------------
%Método para calcular o quatérnio dual que representa
%o frame atual
%Parâmetros:
%o - rotação de theta em z
%d - distancia d em z
%a - deslocamento de a em x
%s - rotação de alpha em x
%Retorno:
%xe: quatérnio de transformação
%--------------------------------------
function xe = dualQuatDH(o,d,a,s,k)
    %calculo dos parâmetros
     h1 = cos(o/2)*cos(s/2);
     h2 = cos(o/2)*sin(s/2);
     h3 = sin(o/2)*sin(s/2);
     h4 = sin(o/2)*cos(s/2);
    %calculo do quatérnio dual
     xe(1,1) = h1;
     xe(2,1) = h2;
     xe(3,1) = h3;
     xe(4,1) = h4;
     xe(5,1) = -0.5*(d*h4 + a*h2);
     xe(6,1) = 0.5*(-d*h3 + a*h1);
     xe(7,1) = 0.5*(d*h2 + a*h4);
     xe(8,1) = 0.5*(d*h1 - a*h3);
     %retorna o conjugado do quatérnio dual caso k
     %seja igual a 1
     if k == 1
        xe = dualQuatConj(xe);
     end
end
```

B.13 dualQuatConj.m

```
%--------------------------------------
%Método para calcular o quatérnio dual conjugado
```

```
%Parâmetros:
%q: quatérnio dual
%Retorno:
%qr: quatérnio dual conjugado
%-------------------------------------
function qr = dualQuatConj(q)

    qp =  quatConj([q(1,1) q(2,1) q(3,1) q(4,1)]');
    qd =  quatConj([q(5,1) q(6,1) q(7,1) q(8,1)]');
    qr = [qp;qd];
end
```

B.14 dualHamiltonOp.m

```
%-------------------------------------
%Método para calcular operador de Halmiton
% de um quatérnio dual
%Parâmetros:
%h: quatérnio dual
%op: tipo de operador + ou -
%Retorno:
%T: matriz resultante
%-------------------------------------
function T = dualHamiltonOp(h,op)
    %separa elementos do dual quatérnio
    h1 = h(1,1);
    h2 = h(2,1);
    h3 = h(3,1);
    h4 = h(4,1);
    h5 = h(5,1);
    h6 = h(6,1);
    h7 = h(7,1);
    h8 = h(8,1);

   %calculo de H
   Hp = HamiltonOp([h1 h2 h3 h4]',op);
   Hd = HamiltonOp([h5 h6 h7 h8]',op);
   %matriz de zeros
   O = zeros(4,4);
```

```
   %operador de Halmiton para quatérnios duais
   T = [Hp 0;Hd Hp];
end
```

B.15 rungeKutta42.m

```
%-------------------------------------------------------------
%método de Runge-Kutta de 4ª ordem
%para a resolução numérica (aproximação)
%Soluções de equações diferenciais ordinárias.
%Parâmetros:
%var - vetor com parâmetros para o método
%params - vetor com os parâmetros passados para o modelo dinâmico
%Retorno:
%yk - nova condição inicial , solução para a EDO para a iteração
%sh - novo passo ótimo - (nesse caso esta passando o passo fixo, sempre h)
%-------------------------------------------------------------
function [yk sh] = rungeKutta42(var,y,params)
%-------------------------------------------------------------
%parâmetros para o método de rungeKutta
%-------------------------------------------------------------
    t   = var(1,1);
    h   = var(2,1);
    fun = var(3,1);
%-------------------------------------------------------------
%verifica qual o modelo será resolvido
% 1 - modelo 1
% ~= 1 - modelo 2
%-------------------------------------------------------------
    if fun
%-------------------------------------------------------------
%cálculo dos parâmetros do método de rungeKutta
%-------------------------------------------------------------
        K1 = model1(t,y,params);
        K2 = model1(t + h/2,y + (h/2)*K1,params);
        K3 = model1(t + h/2,y + (h/2)*K2,params);
        K4 = model1(t + h,y + h*K3,params);
%-------------------------------------------------------------
%novo valor de yk e o novo passo
```

```
%-------------------------------------------------------------
        yk = y + (h/6)*(K1 + 2*K2 + 2*K3 + K4);
        sh = h;
    else
%-------------------------------------------------------------
%cálculo dos parâmetros do método de rungeKutta
%-------------------------------------------------------------
        K1 = model2(t,y,params);
        K2 = model2(t + h/2,y + (h/2)*K1,params);
        K3 = model2(t + h/2,y + (h/2)*K2,params);
        K4 = model2(t + h,y + h*K3,params);
%-------------------------------------------------------------
%novo valor de yk e o novo passo
%-------------------------------------------------------------
        yk = y + (h/6)*(K1 + 2*K2 + 2*K3 + K4);
        sh = h;
    end
end
```

B.16 plotTraj1.m

```
%--------------------------------
%Plotar gráficos do controle
%--------------------------------
function plotTraj1(t,Pos,Posd,angle,angled,Hplot,Hdplot,
Mtheta,Pos2,Posd2,angle2,angled2,Hplot2,Hdplot2,Mtheta2)

    figure(1)
    subplot(2,2,1)
    plot(t(1,:),Pos(1,:));
    hold on
    plot(t(1,:),Posd(1,:));
    xlabel('iteration')
    ylabel('(cm)')
    title('X position')
    grid on

    subplot(2,2,2)
    plot(t(1,:),Pos(2,:));
```

```
hold on
plot(t(1,:),Posd(2,:));
xlabel('iteration')
ylabel('(cm)')
title('Y position')
grid on

subplot(2,2,3)
plot(t(1,:),Pos(3,:));
hold on
plot(t(1,:),Posd(3,:));
xlabel('iteration')
ylabel('(cm)')
title('Z position')
grid on

subplot(2,2,4)
plot(t(1,:),angle(1,:));
hold on
plot(t(1,:),angled(1,:));
xlabel('iteration')
ylabel('angle (rad)')
title('angle of rotation')
grid on


%plotar o quartenion e sua covergencia
figure(2)
subplot(3,3,1)
plot(t(1,:),Hplot(1,:));
hold on
plot(t(1,:),Hdplot(1,:));
xlabel('iteration')
ylabel('')
title('q1')
grid on

subplot(3,3,2)
plot(t(1,:),Hplot(2,:));
```

```
hold on
plot(t(1,:),Hdplot(2,:));
xlabel('iteration')
ylabel('')
title('q2')
grid on

subplot(3,3,3)
plot(t(1,:),Hplot(3,:));
hold on
plot(t(1,:),Hdplot(3,:));
xlabel('iteration')
ylabel('')
title('q3')
grid on

subplot(3,3,4)
plot(t(1,:),Hplot(4,:));
hold on
plot(t(1,:),Hdplot(4,:));
xlabel('iteration')
ylabel('')
title('q4')
grid on

subplot(3,3,5)
plot(t(1,:),Hplot(5,:));
hold on
plot(t(1,:),Hdplot(5,:));
xlabel('iteration')
ylabel('')
title('q5')
grid on

subplot(3,3,6)
plot(t(1,:),Hplot(6,:));
hold on
plot(t(1,:),Hdplot(6,:));
xlabel('iteration')
```

```
ylabel('')
title('q6')
grid on

subplot(3,3,7)
plot(t(1,:),Hplot(7,:));
hold on
plot(t(1,:),Hdplot(7,:));
xlabel('iteration')
ylabel('')
title('q7')
grid on

subplot(3,3,8)
plot(t(1,:),Hplot(8,:));
hold on
plot(t(1,:),Hdplot(8,:));
xlabel('iteration')
ylabel('')
title('q8')
grid on

figure(3)
subplot(3,2,1)
plot(t(1,:),Mtheta(1,:));
xlabel('iteration')
ylabel('')
title('o1')
grid on

subplot(3,2,2)
plot(t(1,:),Mtheta(2,:));
xlabel('iteration')
ylabel('')
title('o2')
grid on

subplot(3,2,3)
plot(t(1,:),Mtheta(3,:));
```

```
hold on
xlabel('iteration')
ylabel('')
title('o3')
grid on

subplot(3,2,4)
plot(t(1,:),Mtheta(4,:));
hold on
xlabel('iteration')
ylabel('')
title('o4')
grid on

subplot(3,2,5)
plot(t(1,:),Mtheta(5,:));
hold on
xlabel('iteration')
ylabel('')
title('o5')
grid on

subplot(3,2,6)
plot(t(1,:),Mtheta(6,:));
hold on
xlabel('iteration')
ylabel('')
title('o6')
grid on

%plotar para os pé
figure(4)
subplot(2,2,1)
plot(t(1,:),Pos2(1,:));
hold on
plot(t(1,:),Posd2(1,:));
xlabel('iteration')
ylabel('(cm)')
title('X position')
```

```
grid on

subplot(2,2,2)
plot(t(1,:),Pos2(2,:));
hold on
plot(t(1,:),Posd2(2,:));
xlabel('iteration')
ylabel('(cm)')
title('Y position')
grid on

subplot(2,2,3)
plot(t(1,:),Pos2(3,:));
hold on
plot(t(1,:),Posd2(3,:));
xlabel('iteration')
ylabel('(cm)')
title('Z position')
grid on

subplot(2,2,4)
plot(t(1,:),angle2(1,:));
hold on
plot(t(1,:),angled2(1,:));
xlabel('iteration')
ylabel('angle (rad)')
title('angle of rotation')
grid on


%plotar o quartenion e sua covergencia

figure(5)
subplot(3,3,1)
plot(t(1,:),Hplot2(1,:));
hold on
plot(t(1,:),Hdplot2(1,:));
xlabel('iteration')
ylabel('')
```

```
title('q1')
grid on

subplot(3,3,2)
plot(t(1,:),Hplot2(2,:));
hold on
plot(t(1,:),Hdplot2(2,:));
xlabel('iteration')
ylabel('')
title('q2')
grid on

subplot(3,3,3)
plot(t(1,:),Hplot2(3,:));
hold on
plot(t(1,:),Hdplot2(3,:));
xlabel('iteration')
ylabel('')
title('q3')
grid on

subplot(3,3,4)
plot(t(1,:),Hplot2(4,:));
hold on
plot(t(1,:),Hdplot2(4,:));
xlabel('iteration')
ylabel('')
title('q4')
grid on

subplot(3,3,5)
plot(t(1,:),Hplot2(5,:));
hold on
plot(t(1,:),Hdplot2(5,:));
xlabel('iteration')
ylabel('')
title('q5')
grid on
```

```
subplot(3,3,6)
plot(t(1,:),Hplot2(6,:));
hold on
plot(t(1,:),Hdplot2(6,:));
xlabel('iteration')
ylabel('')
title('q6')
grid on

subplot(3,3,7)
plot(t(1,:),Hplot2(7,:));
hold on
plot(t(1,:),Hdplot2(7,:));
xlabel('iteration')
ylabel('')
title('q7')
grid on

subplot(3,3,8)
plot(t(1,:),Hplot2(8,:));
hold on
plot(t(1,:),Hdplot2(8,:));
xlabel('iteration')
ylabel('')
title('q8')
grid on

figure(6)
subplot(3,2,1)
plot(t(1,:),Mtheta2(1,:));
xlabel('iteration')
ylabel('')
title('o1')
grid on

subplot(3,2,2)
plot(t(1,:),Mtheta2(2,:));
xlabel('iteration')
ylabel('')
```

```
    title('o2')
    grid on

    subplot(3,2,3)
    plot(t(1,:),Mtheta2(3,:));
    hold on
    xlabel('iteration')
    ylabel('')
    title('o3')
    grid on

    subplot(3,2,4)
    plot(t(1,:),Mtheta2(4,:));
    hold on
    xlabel('iteration')
    ylabel('')
    title('o4')
    grid on

    subplot(3,2,5)
    plot(t(1,:),Mtheta2(5,:));
    hold on
    xlabel('iteration')
    ylabel('')
    title('o5')
    grid on

    subplot(3,2,6)
    plot(t(1,:),Mtheta2(6,:));
    hold on
    xlabel('iteration')
    ylabel('')
    title('o6')
    grid on
end
```

B.17 plotGraficosControle.m

```
%--------------------------------
```

```
%Plotar gráficos do controle
%-------------------------------
function plotGraficosControle(t1,dt,T,Pos,Posd,angle,angled,Mha,Mhd,
Mtheta,Pos2,Posd2,angle2,angled2,Mha2,Mhd2,Mtheta2,color1,color2)

    trajD = 'k:';
    t = (0+t1):dt:(T-1)*dt+t1;
    figure(1)
    subplot(2,2,1)
    plot(t(1,:),Pos(1,:),color1);
    hold on
    plot(t(1,:),Posd(1,:),trajD);
    xlabel('tempo (s)')
    ylabel('(m)')
    title('Posição X')
    grid on

    subplot(2,2,2)
    plot(t(1,:),Pos(2,:),color1);
    hold on
    plot(t(1,:),Posd(2,:),trajD);
    xlabel('tempo (s)')
    ylabel('(m)')
    title('Posição Y')
    grid on

    subplot(2,2,3)
    plot(t(1,:),Pos(3,:),color1);
    hold on
    plot(t(1,:),Posd(3,:),trajD);
    xlabel('tempo (s)')
    ylabel('(m)')
    title('Posição Z')
    grid on

    subplot(2,2,4)
    plot(t(1,:),angle(1,:),color1);
    hold on
    plot(t(1,:),angled(1,:),trajD);
```

```
xlabel('tempo (s)')
ylabel('Angulo (rad)')
%title('angle of rotation')
grid on


%plotar o quartenion e sua covergencia
figure(2)
subplot(3,3,1)
plot(t(1,:),Mha(1,:),color1);
hold on
plot(t(1,:),Mhd(1,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q1')
grid on

subplot(3,3,2)
plot(t(1,:),Mha(2,:),color1);
hold on
plot(t(1,:),Mhd(2,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q2')
grid on

subplot(3,3,3)
plot(t(1,:),Mha(3,:),color1);
hold on
plot(t(1,:),Mhd(3,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q3')
grid on

subplot(3,3,4)
plot(t(1,:),Mha(4,:),color1);
hold on
plot(t(1,:),Mhd(4,:),trajD);
```

```
xlabel('tempo (s)')
ylabel('')
title('q4')
grid on

subplot(3,3,5)
plot(t(1,:),Mha(5,:),color1);
hold on
plot(t(1,:),Mhd(5,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q5')
grid on

subplot(3,3,6)
plot(t(1,:),Mha(6,:),color1);
hold on
plot(t(1,:),Mhd(6,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q6')
grid on

subplot(3,3,7)
plot(t(1,:),Mha(7,:),color1);
hold on
plot(t(1,:),Mhd(7,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q7')
grid on

subplot(3,3,8)
plot(t(1,:),Mha(8,:),color1);
hold on
plot(t(1,:),Mhd(8,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q8')
```

```
grid on

figure(3)
subplot(3,2,1)
plot(t(1,:),Mtheta(1,:),color1);
xlabel('tempo (s)')
ylabel('')
title('o1')
grid on

subplot(3,2,2)
plot(t(1,:),Mtheta(2,:),color1);
xlabel('tempo (s)')
ylabel('')
title('o2')
grid on

subplot(3,2,3)
plot(t(1,:),Mtheta(3,:),color1);
hold on
xlabel('tempo (s)')
ylabel('')
title('o3')
grid on

subplot(3,2,4)
plot(t(1,:),Mtheta(4,:),color1);
hold on
xlabel('tempo (s)')
ylabel('')
title('o4')
grid on

subplot(3,2,5)
plot(t(1,:),Mtheta(5,:),color1);
hold on
xlabel('tempo (s)')
ylabel('')
title('o5')
```

```
grid on

subplot(3,2,6)
plot(t(1,:),Mtheta(6,:),color1);
hold on
xlabel('tempo (s)')
ylabel('')
title('o6')
grid on


figure(4)
subplot(2,2,1)
plot(t(1,:),Pos2(1,:),color2);
hold on
plot(t(1,:),Posd2(1,:),trajD);
xlabel('tempo (s)')
ylabel('(m)')
title('Posição X')
grid on

subplot(2,2,2)
plot(t(1,:),Pos2(2,:),color2);
hold on
plot(t(1,:),Posd2(2,:),trajD);
xlabel('tempo (s)')
ylabel('(m)')
title('Posição Y')
grid on

subplot(2,2,3)
plot(t(1,:),Pos2(3,:),color2);
hold on
plot(t(1,:),Posd2(3,:),trajD);
xlabel('tempo (s)')
ylabel('(m)')
title('Posição Z')
grid on
```

```
subplot(2,2,4)
plot(t(1,:),angle2(1,:),color2);
hold on
plot(t(1,:),angled2(1,:),trajD);
xlabel('tempo (s)')
ylabel('angle (rad)')
%title('angle of rotation')
grid on


%plotar o quartenion e sua covergencia
figure(5)
subplot(3,3,1)
plot(t(1,:),Mha2(1,:),color2);
hold on
plot(t(1,:),Mhd2(1,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q1')
grid on

subplot(3,3,2)
plot(t(1,:),Mha2(2,:),color2);
hold on
plot(t(1,:),Mhd2(2,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q2')
grid on

subplot(3,3,3)
plot(t(1,:),Mha2(3,:),color2);
hold on
plot(t(1,:),Mhd2(3,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q3')
grid on
```

```
subplot(3,3,4)
plot(t(1,:),Mha2(4,:),color2);
hold on
plot(t(1,:),Mhd2(4,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q4')
grid on

subplot(3,3,5)
plot(t(1,:),Mha2(5,:),color2);
hold on
plot(t(1,:),Mhd2(5,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q5')
grid on

subplot(3,3,6)
plot(t(1,:),Mha2(6,:),color2);
hold on
plot(t(1,:),Mhd2(6,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q6')
grid on

subplot(3,3,7)
plot(t(1,:),Mha2(7,:),color2);
hold on
plot(t(1,:),Mhd2(7,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q7')
grid on

subplot(3,3,8)
plot(t(1,:),Mha2(8,:),color2);
hold on
```

```
plot(t(1,:),Mhd2(8,:),trajD);
xlabel('tempo (s)')
ylabel('')
title('q8')
grid on

figure(6)
subplot(3,2,1)
plot(t(1,:),Mtheta2(1,:),color2);
xlabel('tempo (s)')
ylabel('')
title('o1')
grid on

subplot(3,2,2)
plot(t(1,:),Mtheta2(2,:),color2);
xlabel('tempo (s)')
ylabel('')
title('o2')
grid on

subplot(3,2,3)
plot(t(1,:),Mtheta2(3,:),color2);
hold on
xlabel('tempo (s)')
ylabel('')
title('o3')
grid on

subplot(3,2,4)
plot(t(1,:),Mtheta2(4,:),color2);
hold on
xlabel('tempo (s)')
ylabel('')
title('o4')
grid on

subplot(3,2,5)
plot(t(1,:),Mtheta2(5,:),color2);
```

```
    hold on
    xlabel('tempo (s)')
    ylabel('')
    title('o5')
    grid on

    subplot(3,2,6)
    plot(t(1,:),Mtheta2(6,:),color2);
    hold on
    xlabel('tempo (s)')
    ylabel('')
    title('o6')
    grid on
end
```

B.18 plotCoord.m

```
%---------------------------------------
%Método para plotar o sistema de coordenadas
%---------------------------------------
function plotCoord(hOrg)
    Org = getPositionDualQuat(hOrg);
    hX = dualQuatMult(hOrg,transformacao([0;0.1;0;0],[1;0;0;0]));
    hY = dualQuatMult(hOrg,transformacao([0;0;0.1;0],[1;0;0;0]));
    hZ = dualQuatMult(hOrg,transformacao([0;0;0;0.1],[1;0;0;0]));

    plotVector(Org,getPositionDualQuat(hX),'b');
    plotVector(Org,getPositionDualQuat(hY),'r');
    plotVector(Org,getPositionDualQuat(hZ),'g');
end
```

B.19 plotarTrajetoria.m

```
%-----------------------------------------------------------
%Método para mostrar graficamente a trajetória do modelo dinâmico
%Parâmetros:
%U - variáveis de controle
%X - vetor com as condições iniciais
```

```
%Retorno:
%gráfico do modelo dinâmico
%-------------------------------------------------------------
function plotarTrajetoria(U,X)
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
global m L g pfa expK hEdo
%-------------------------------------------------------------
%condições iniciais para MS
%-------------------------------------------------------------
    xod  = X(1,1);
    yod  = X(2,1);
    zod  = X(3,1);
    dxod = X(4,1);
    dyod = X(5,1);
    dzod = X(6,1);
%-------------------------------------------------------------
%valores para a otimização - valores de u
%-------------------------------------------------------------
    %u = [phi theta k Bss]
    theta = U(1,1);
    phi   = U(2,1);
    k     = U(3,1);
    Bss   = U(4,1);
    expK  = U(5,1);
%-------------------------------------------------------------
%vetor com as condições iniciais MS
%-------------------------------------------------------------
    y0 = [xod;dxod;yod;dyod;zod;dzod];
%-------------------------------------------------------------
%vetor com os parâmetros constantes
%-------------------------------------------------------------
    params = [m;L;g;k;Bss;expK];
%-------------------------------------------------------------
%Parâmetros para os métodos
%-------------------------------------------------------------
    t = 0;%inicio do tempo  t = 0
    h = hEdo;%passo do método rungeKutta42 inicial
```

```
    N = 10000;%número máximo de iterações

    %primeiro método
    sh = h;%tamanho do passo para o método rungeKutta42
    %atualizando durante a execução do método
    ind = 1;%contador
%-------------------------------------------------------------
%vetores auxiliares para guardar a trajetória
%-------------------------------------------------------------
    px(1,1) = y0(1,1);
    py(1,1) = y0(3,1);
    pz(1,1) = y0(5,1);
%-------------------------------------------------------------
%inicio do método primeiro MS para TD
%-------------------------------------------------------------
    for x = 0:h:N*h
%-------------------------------------------------------------
%vetor de parâmetros
%-------------------------------------------------------------
        var = [t;h;1];
%-------------------------------------------------------------
%método numérico para solucionar as equações diferenciais
%passo a passo
%-------------------------------------------------------------
        [y sh] = rungeKutta42(var,y0,params);
%-------------------------------------------------------------
%atualizando a condição inicial
%-------------------------------------------------------------
        y0  = y;
%-------------------------------------------------------------
%atualizando o instante t
%-------------------------------------------------------------
        t = t+sh;
%-------------------------------------------------------------
%verificando a condição de parada posição Z < que Z de touchdown
%Z de touchdown = L*cos(theta)
%-------------------------------------------------------------
        if y0(5,1) < L*cos(theta)
         break
```

```
        end
%-------------------------------------------------------------
%colocando os valores nos vetores auxiliares
%-------------------------------------------------------------
        px(1,ind) = y0(1,1);
        py(1,ind) = y0(3,1);
        pz(1,ind) = y0(5,1);
%-------------------------------------------------------------
%atualizando o contador
%-------------------------------------------------------------
        ind = ind + 1;
    end
%-------------------------------------------------------------
%atualizando o contador - tratando o valor
%-------------------------------------------------------------
    if ind > 1
        ind = ind -1;
    end
%-------------------------------------------------------------
%Posição do centro de massa no momento de  Touchdown (TD)
%-------------------------------------------------------------
    pc = [px(1,ind);py(1,ind);pz(1,ind)];%centro de massa
%-------------------------------------------------------------
%posição do pé de balaço quando toca o chão
%-------------------------------------------------------------
    pfb = pc + L*[sin(theta)*cos(phi);sin(theta)*sin(phi);-cos(theta)]
%-------------------------------------------------------------
%tempo em que acontece a codição de touchdown
%-------------------------------------------------------------
    TD = t;%tempo de TD
%-------------------------------------------------------------
%parâmetros constante para o segundo método
%-------------------------------------------------------------
    params = [m;L;g;k;Bss;t;pfb(1,1);pfb(2,1);pfb(3,1);expK];
%-------------------------------------------------------------
%iniciando o segundo contador
%-------------------------------------------------------------
    ind2 = 1;
    sh = h;%tamanho do passo para o método rungeKutta42
```

```
    %atualizando durante a execução do método
%-------------------------------------------------------------
%vetores auxiliares para guardar a trajetória
%-------------------------------------------------------------
    px2(1,1) = y0(1,1);
    py2(1,1) = y0(3,1);
    pz2(1,1) = y0(5,1);
%-------------------------------------------------------------
%inicio do método 2 - TD para LH
%-------------------------------------------------------------
    for x = 0:h:N*h
%-------------------------------------------------------------
%vetor de parâmetros
%-------------------------------------------------------------
        var = [t;h;0];
%-------------------------------------------------------------
%método numérico para solucionar as equações diferenciais
%passo a passo
%-------------------------------------------------------------
        [y sh] = rungeKutta42(var,y0,params);
%-------------------------------------------------------------
%atualizando nova condição inicial
%-------------------------------------------------------------
        y0  = y;
%-------------------------------------------------------------
%atualizando o instante t
%-------------------------------------------------------------
        t = t+sh;
%-------------------------------------------------------------
%verificando a condição de parada posição dZ > 0
%-------------------------------------------------------------
        if y0(6,1) > 0
            break
        end
%-------------------------------------------------------------
%atualizando os vetores auxiliares da trajetória
%-------------------------------------------------------------
        px2(1,ind2) = y0(1,1);
        py2(1,ind2) = y0(3,1);
```

```
        pz2(1,ind2) = y0(5,1);
%-------------------------------------------------------------
%atualizando o contador
%-------------------------------------------------------------
        ind2 = ind2+1;
    end
%-------------------------------------------------------------
%atualizando o contador
%-------------------------------------------------------------
    if ind2 > 1
        ind2 = ind2 -1;
    end
%-------------------------------------------------------------
%atualizando a posição do centro de massa
%-------------------------------------------------------------
    pc = [px2(1,ind2);py2(1,ind2);pz2(1,ind2)];%centro de massa
%-------------------------------------------------------------
% a trajetoria é simetrica ao ponto de middle suport
% desta forma fazemos um espelho da função
%deslocado ela para a origem
%-------------------------------------------------------------
    plot3(px(1,1:ind),py(1,1:ind),pz(1,1:ind),'b')
    hold on
    plot3(px2(1,1:ind2),py2(1,1:ind2),pz2(1,1:ind2),'r')
    hold on
%-------------------------------------------------------------
%espelho da função
%-------------------------------------------------------------
    offsetx = px2(1,ind2);
    offsety = py2(1,ind2);
    plot3(-px2(1,ind2:-1:1) + 2*offsetx,-py2(1,ind2:-1:1)
    +2*offsety,pz2(1,ind2:-1:1),'r')
    hold on
    offsetx = -px2(1,ind2) + 2*offsetx;
    offsety = -py2(1,ind2) + 2*offsety;
    plot3(-px(1,ind:-1:1) + 2*offsetx,-py(1,ind:-1:1)+2*offsety,pz(1,ind:-1:1),'b')
    hold on
%-------------------------------------------------------------
%projeção
```

```
%-------------------------------------------------------------
    plot3(px(1,1:ind),py(1,1:ind),0*pz(1,1:ind),'g')
    hold on
    plot3(px2(1,1:ind2),py2(1,1:ind2),0*pz2(1,1:ind2),'m')
    hold on
%-------------------------------------------------------------
%espelho da função
%-------------------------------------------------------------
    offsetx = px2(1,ind2);
    offsety = py2(1,ind2);
    plot3(-px2(1,ind2:-1:1) + 2*offsetx,-py2(1,ind2:-1:1)
    +2*offsety,0*pz2(1,ind2:-1:1),'m')
    hold on
    offsetx = -px2(1,ind2) + 2*offsetx;
    offsety = -py2(1,ind2) + 2*offsety;
    plot3(-px(1,ind:-1:1) + 2*offsetx,-py(1,ind:-1:1)
    +2*offsety,0*pz(1,ind:-1:1),'g')
    hold on
%-------------------------------------------------------------
%plotar posição dos pés
%-------------------------------------------------------------
    plot3([pfa(1,1) pfb(1,1)],[pfa(2,1) pfb(2,1)],[0 0],'--k');
    hold on
%-------------------------------------------------------------
%projeção centro de massa e projeção centro de massa
%-------------------------------------------------------------
    plot3(pc(1,1),pc(2,1),pc(3,1),'o')
    hold on
    plot3([pc(1,1) pc(1,1)],[pc(2,1) pc(2,1)],[pc(3,1) 0],':');
    hold on
    plot3(pc(1,1),pc(2,1),0,'x');
%-------------------------------------------------------------
%configuração do grafico
%-------------------------------------------------------------
    grid on
    xlabel('X')
    ylabel('Y')
    zlabel('Z')
end
```

B.20 configGrafico.m

```
%--------------------------
%Configurar Gráficos
%--------------------------
function configGrafico()
    grid on
    axis([-1 1 -0.5 0.5 -1 0.2])
    view(60,35)
    xlabel('X')
    ylabel('Y')
    zlabel('Z')
end
```

B.21 KinematicRobo.m

```
%-------------------------------------
%Método para calcular o a cinemática direta
%do robô utilizando quatérnio dual
%-------------------------------------
function hr = KinematicRobo(theta,hOrg,hP,tipo,CoM)
 global   hpi L1 L2 height MDH

 thetar = theta(:,1);
 thetal = theta(:,2);

 %transformações da origem para a origem
 %da configuração inicial das 2 pernas
 hCoM_O0_rightLeg  = dualQuatDH(hpi,-L2,-L1,0,0);
 hCoM_O0_leftLeg   = dualQuatDH(hpi,-L2, L1,0,0);


    hB_O6a = dualQuatMult(hOrg,hP);

    if tipo
        hO6_O0 = KinematicModel(MDH,thetar,1,0);
     else
        hO6_O0 = KinematicModel(MDH,thetal,1,0);
    end
```

```
    hB_O0 = dualQuatMult(hB_O6a,hO6_O0);

    if tipo
        hB_CoM = dualQuatMult(hB_O0,dualQuatConj(hCoM_O0_rightLeg));
    else
        hB_CoM = dualQuatMult(hB_O0,dualQuatConj(hCoM_O0_leftLeg));
    end

    hr = hB_CoM;

    if CoM == 0
        if tipo
            hB_O0 = dualQuatMult(hB_CoM,hCoM_O0_leftLeg);
            hO0_O6 =  KinematicModel(MDH,thetal,6,1);
            hr = dualQuatMult(hB_O0,hO0_O6);
        else
            hB_O0 = dualQuatMult(hB_CoM,hCoM_O0_rightLeg);
            hO0_O6 =  KinematicModel(MDH,thetar,6,1);
            hr = dualQuatMult(hB_O0,hO0_O6);
        end
    end
end
```

B.22 KinematicModel.m

```
%--------------------------------------
%Método para calcular o a cinemática direta
%da cadeia cinemática
%Parâmetros:
%MDH: parâmetros de Denavit-Hartenberg [thetai di ai alphai]
%N: numero de linhas da tabela de MDH
%foward: indica se a cinemática vai ser foward ou backward
%Retorno:
%xe: cinemática direta
%--------------------------------------
function xe = KinematicModel(MDH,theta,N,foward)
    %vetor de retorno
    xe = [1 0 0 0 0 0 0 0]'; %quatérnio dual unitário
```

```
    %foward
    if foward == 1
        for i = 1:1:N
         %separar parâmetros
         o = MDH(i,1) + theta(i,1);
         d = MDH(i,2);
         a = MDH(i,3);
         s = MDH(i,4);
         %cálculo de xe atual
         x = dualQuatDH(o,d,a,s,0);
         xe = dualQuatMult(xe,x);
        end
    else
        %backward
        for i = 6:-1:N
         %separar parâmetros
         o = MDH(i,1) + theta(i,1);
         d = MDH(i,2);
         a = MDH(i,3);
         s = MDH(i,4);
         %cálculo de xe atual
         x  = dualQuatDH(o,d,a,s,1); %conjugado
         xe = dualQuatMult(xe,x);
        end
    end
end
```

B.23 model1.m

```
%-------------------------------------------------------------
%Calcula o valor das derivadas do modelo dinâmico 1
%Parâmetros
%t   - tempo
%Y0  - Vetor com a condição inicial do sistema
%params: parâmetros do sistema
%m   - massa
%L   - comprimento da perna do modelo
%g   - gravidade
%k   - constante da mola
```

```
%Bss - coeficiente angular do modelo
%Retorno:
%dydt - vetor com primeira e segunda derivada do sistema
%-------------------------------------------------------------
function dydt = model1(t,Y0,params)
%-------------------------------------------------------------
%constantes
%-------------------------------------------------------------
    m    = params(1,1);
    L    = params(2,1);
    g    = params(3,1);
    k    = params(4,1);
    Bss  = params(5,1);
    expK = params(6,1);
%-------------------------------------------------------------
%condição inicial
%-------------------------------------------------------------
    x  = Y0(1,1);
    dx = Y0(2,1);
    y  = Y0(3,1);
    dy = Y0(4,1);
    z  = Y0(5,1);
    dz = Y0(6,1);
%-------------------------------------------------------------
%norma de L
%-------------------------------------------------------------
    norma = (x*x + y*y + z*z )^0.5;
%-------------------------------------------------------------
%derivadas
%-------------------------------------------------------------
    f1 = dx;
    f2 = (k*expK/m)*( L +Bss*t - norma)*(x/norma);
    f3 = dy;
    f4 = (k*expK/m)*( L +Bss*t - norma)*(y/norma);
    f5 = dz;
    f6 = (k*expK/m)*( L +Bss*t - norma)*(z/norma) - g;
%-------------------------------------------------------------
%solução
%-------------------------------------------------------------
```

```
    dydt = [f1;f2;f3;f4;f5;f6];
end
```

B.24 model2.m

```
%-------------------------------------------------------------
%Calcula o valor das derivadas do modelo dinâmico 2
%Parâmetros
%t   - tempo
%Y0  - Vetor com a condição inicial do sistema
%params: parâmetros do sistema
%m   - massa
%L   - comprimento da perna do modelo
%g   - gravidade
%k   - constante da mola
%Bss - coeficiente angular do modelo
%TD  - valor do tempo de touchdown
%bx  - valor de x do pé B
%by  - valor de y do pé B
%bz  - valor de z do pé B
%expK- valor do ganho da constante da mola (ordem de grandeza)
%Retorno:
%dydt - vetor com primeira e segunda derivada do sistema
%-------------------------------------------------------------
function dydt = model2(t,Y0,params)
%-------------------------------------------------------------
%constantes
%-------------------------------------------------------------
    m   = params(1,1);
    L   = params(2,1);
    g   = params(3,1);
    k   = params(4,1);
    Bss = params(5,1);
    TD  = params(6,1);
    bx  = params(7,1);
    by  = params(8,1);
    bz  = params(9,1);
    expK = params(10,1);
%-------------------------------------------------------------
```

```
%condição inicial
%-------------------------------------------------------------
    x  = Y0(1,1);
    dx = Y0(2,1);
    y  = Y0(3,1);
    dy = Y0(4,1);
    z  = Y0(5,1);
    dz = Y0(6,1);
%-------------------------------------------------------------
%norma de L
%-------------------------------------------------------------
    norma1 = (x*x + y*y + z*z )^0.5;
    norma2 = ((x - bx)^2 + (y - by)^2  + (z - bz)^2  )^0.5;
%-------------------------------------------------------------
%derivadas de y
%-------------------------------------------------------------
    f1 = dx;
    f2 = (k*expK/m)*( ( L + Bss*TD - norma1)*(x/norma1) +
    ( L + Bss*TD - norma2)*( (x - bx)/norma2) );
    f3 = dy;
    f4 = (k*expK/m)*( ( L + Bss*TD - norma1)*(y/norma1) +
    ( L + Bss*TD - norma2)*( (y - by)/norma2) );
    f5 = dz;
    f6 = (k*expK/m)*( ( L + Bss*TD - norma1)*(z/norma1) +
    ( L + Bss*TD - norma2)*( (z - bz)/norma2) ) + g;
%-------------------------------------------------------------
%sistema de equações
%-------------------------------------------------------------
    dydt = [f1;f2;f3;f4;f5;f6];
end
```

B.25 SGDMomento.m

```
%-------------------------------------------------------------
%Método do SGD com momento utilizado para
%otimizar a trajetória do centro de massa do modelo 3D dual SLIP
%-------------------------------------------------------------
function [U,vt] = SGDMomento(U0,X0,vt)
%-------------------------------------------------------------
```

```
%variáveis globais
%-------------------------------------------------------------
    global ganhoAlpha gamma
%-------------------------------------------------------------
%cálculo do gradiente da função
%-------------------------------------------------------------
    G = gradienteFuncao(X0,U0);
    g1 = G(1,1);
    g2 = G(2,1);
    g3 = G(3,1);
    g4 = G(4,1);
%-------------------------------------------------------------
%cálculo do momento da função
%-------------------------------------------------------------
    vt(1,1) = gamma*vt(1,1) + ganhoAlpha*g1;
    vt(2,1) = gamma*vt(2,1) + ganhoAlpha*g2;
    vt(3,1) = gamma*vt(3,1) + ganhoAlpha*g3;
    vt(4,1) = gamma*vt(4,1) + ganhoAlpha*g4;
%-------------------------------------------------------------
%atualizando o valor do vetor de controle - U
%-------------------------------------------------------------
    u1 = U0(1,1) - vt(1,1);
    u2 = U0(2,1) - vt(2,1);
    u3 = U0(3,1) - vt(3,1);
    u4 = U0(4,1) - vt(4,1);
    u5 = U0(5,1);
%-------------------------------------------------------------
%Valor atualizado
%-------------------------------------------------------------
    U = [u1;u2;u3;u4;u5];
end
```

B.26 SGD.m

```
%-------------------------------------------------------------
%Método do gradiente descendente estocástico utilizado para
%otimizar a trajetória do centro de massa do modelo 3D dual SLIP
%-------------------------------------------------------------
function U = SGD(U0,X0)
```

```
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
    global ganhoAlpha
%-------------------------------------------------------------
%cálculo do gradiente da função
%-------------------------------------------------------------
    G = gradienteFuncao(X0,U0);
    g1 = G(1,1);
    g2 = G(2,1);
    g3 = G(3,1);
    g4 = G(4,1);
%-------------------------------------------------------------
%atualizando o valor do vetor de controle - U SGD
%-------------------------------------------------------------
    u1 = U0(1,1) - ganhoAlpha*g1;
    u2 = U0(2,1) - ganhoAlpha*g2;
    u3 = U0(3,1) - ganhoAlpha*g3;
    u4 = U0(4,1) - ganhoAlpha*g4;
    u5 = U0(5,1);
%-------------------------------------------------------------
%Valor atualizado
%-------------------------------------------------------------
    U = [u1;u2;u3;u4;u5];
end
```

B.27 setLimites.m

```
%-------------------------------------------------------------
%Método para verificar  se a variável de controle
%está dentro de determinados limites
%caso não esteja atualiza o valor desta colocando
%o valor mínimo ou máximo
%Parâmetro:
%U0 - variáveis de controle
%Retorno:
%U - variável de controle corrigida
%-------------------------------------------------------------
function U = setLimites(U0)
```

```
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
global thetaM phiM KM BSSM
%-------------------------------------------------------------
%Separar as variaveis de controle
%-------------------------------------------------------------
    u1 = U0(1,1);
    u2 = U0(2,1);
    u3 = U0(3,1);
    u4 = U0(4,1);
    u5 = U0(5,1);
%-------------------------------------------------------------
%tratar os dados da variável de controle - theta
%-------------------------------------------------------------
        if u1 < 0
           u1 = 0;
        end

        if u1 > thetaM
           u1 = thetaM;
        end
%-------------------------------------------------------------
%tratar os dados da variável de controle - phi
%-------------------------------------------------------------
        if u2 < 0
           u2 = 0;
        end

        if u2 > phiM
           u2 = phiM;
        end
%-------------------------------------------------------------
%tratar os dados da variável de controle - K
%-------------------------------------------------------------
        if u3 < 0
           u3 = 0;
        end
```

```
        if u3 > KM
           u3 = KM;
        end
%-------------------------------------------------------------
%tratar os dados da variável de controle - BSS
%-------------------------------------------------------------
        if u4 < 0
           u4 = 0;
        end

        if u4 > BSSM
           u4 = BSSM;
        end
%-------------------------------------------------------------
%retorno da função
%vetor corrigido
%-------------------------------------------------------------
        U = [u1;u2;u3;u4;u5];
end
```

B.28 otimizacao.m

```
%-------------------------------------------------------------
%método para calcular os melhores parâmetros para a variável de controle U
%minimizando uma função objetivo.
%Otimizações implementadas:
%metodo : 0 estocástico  gradiente descendente SGD
%metodo : 1 SGD com momento
%metodo : 2 Nesterov accelerated gradient
%metodo : 3 Adagrad
%Parâmetros:
%tipo: 1 para executar a otimização e plotar a trajetória
%dos dos valores obtidos pela otimização
%tipo:0 apenas plotar a trajetória dos valores de U já estipulados
%U - variável de controle inicial
%X - condição inicial constante
%Retorno:
%mostrado no console o melhor resultado e mostrado graficamente a
%trajetória obtida
```

```
%-------------------------------------------------------------
function otimizacao(U,X,tipo,metodo)
if tipo ==1
    %-------------------------------------------------------------
    %variáveis globais
    %-------------------------------------------------------------
    global maxNGrad ganhoAlpha gamma
    %-------------------------------------------------------------
    %iniciar variáveis de controle inicial
    %-------------------------------------------------------------
    u1 = U(1,1);
    u2 = U(2,1);
    u3 = U(3,1);
    u4 = U(4,1);
    u5 = U(5,1);
    %-------------------------------------------------------------
    %inicio do método
    %-------------------------------------------------------------
    fo = 1;%condição de parada
    %-------------------------------------------------------------
    %melhores valores
    %-------------------------------------------------------------
    fm = fo;
    UM = U(:,1);
    %-------------------------------------------------------------
    %vetores axiliares para os métodos de otimização
    %-------------------------------------------------------------
    vt = zeros(4,1); %usado como auxiliar NAG
    Grad = zeros(4,1);%usado como auxiliar adagrad

    for j = 1:1:maxNGrad

    %-------------------------------------------------------------
    % gradiente descendente estocástico SGD
    %-------------------------------------------------------------
    if metodo == 0
        U = SGD(U,X);
    end
    %-------------------------------------------------------------
```

```
%SGD com momento
%-------------------------------------------------------------
if metodo == 1
    [U,vt] = SGDMomento(U,X,vt);
end
%-------------------------------------------------------------
%Nesterov accelerated gradient
%-------------------------------------------------------------
if metodo == 2
    [U,vt] = NAG(U,X,vt);
end
%-------------------------------------------------------------
%Adagrad
%-------------------------------------------------------------
if metodo == 3
    [U,Grad] = adagrad(U,X,Grad);
end

%-------------------------------------------------------------
%Setar os limites inferiores e superiores em U
%-------------------------------------------------------------
    U0 = setLimites(U);
    u1 = U0(1,1);
    u2 = U0(2,1);
    u3 = U0(3,1);
    u4 = U0(4,1);
    u5 = U0(5,1);
%-------------------------------------------------------------
%atualizar o vetor U  (variáveis de controle)
%-------------------------------------------------------------
    U  = [u1;u2;u3;u4;u5];
%-------------------------------------------------------------
%Cálculo do valor da função objetivo
%-------------------------------------------------------------
   [pa,pb,pc] = trajetoria(U,X);
   fo = funcaoObjetivo(pa,pb,pc);
%-------------------------------------------------------------
%verificar melhor resultado
%-------------------------------------------------------------
```

```
        if fo<fm
            fm = fo;
            UM = U;
        end
    %-------------------------------------------------------------
    %verificar condição de parada
    %-------------------------------------------------------------
        if fo < 1*10^-10
            break
        end
    %-------------------------------------------------------------
    %imprimir resultado no console
    %-------------------------------------------------------------
        imprimirConsole(j,[U;fo]);
    end
    %-------------------------------------------------------------
    %imprimir resultado final no console
    %-------------------------------------------------------------
    disp('************************************************************')
    disp('Melhor Solução: ')
    imprimirConsole(0,[UM;fm]);
    disp('************************************************************')
    %-------------------------------------------------------------
    %mostrar a trajetória
    %-------------------------------------------------------------
    plotarTrajetoria(UM,X)
else
    %-------------------------------------------------------------
    %mostrar a trajetória
    %-------------------------------------------------------------
    plotarTrajetoria(U,X)
end

end
```

B.29 NAG.m

```
%-------------------------------------------------------------
```

```
%Método do Nesterov accelerated gradient utilizado para
%otimizar a trajetória do centro de massa do modelo 3D dual SLIP
%-------------------------------------------------------------
function [U,vt] = NAG(U0,X0,vt)
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
    global ganhoAlpha gamma
%-------------------------------------------------------------
%cálculando o valor com o ganho anterior
%-------------------------------------------------------------
    y1 = U0(1,1) - gamma*vt(1,1);
    y2 = U0(2,1) - gamma*vt(2,1);
    y3 = U0(3,1) - gamma*vt(3,1);
    y4 = U0(4,1) - gamma*vt(4,1);
    y5 = U0(5,1);
    Y0 = [y1;y2;y3;y4;y5];
%-------------------------------------------------------------
%cálculo do gradiente da função
%-------------------------------------------------------------
    G = gradienteFuncao(X0,Y0);
    g1 = G(1,1);
    g2 = G(2,1);
    g3 = G(3,1);
    g4 = G(4,1);
%-------------------------------------------------------------
%cálculando o valor com o ganho atual
%-------------------------------------------------------------
    vt(1,1) = gamma*vt(1,1) + ganhoAlpha*g1;
    vt(2,1) = gamma*vt(2,1) + ganhoAlpha*g2;
    vt(3,1) = gamma*vt(3,1) + ganhoAlpha*g3;
    vt(4,1) = gamma*vt(4,1) + ganhoAlpha*g4;

    u1 = U0(1,1) - vt(1,1);
    u2 = U0(2,1) - vt(2,1);
    u3 = U0(3,1) - vt(3,1);
    u4 = U0(4,1) - vt(4,1);
    u5 = U0(5,1);
%-------------------------------------------------------------
```

```
%Valor atualizado
%-------------------------------------------------------------
    U = [u1;u2;u3;u4;u5];
end
```

B.30 funcaoObjetivo.m

```
%-------------------------------------------------------------
%Calcula o valor da função objetivo
%Parâmetros:
%pa - Vetor com posição espacial do pé A
%pb - Vetor com posição espacial do pé B
%pc - Vetor com posição espacial do centro de massa
%Retorno:
%fo -  valor da função objetivo
%-------------------------------------------------------------
function fo = funcaoObjetivo(pa,pb,pc)
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
    global lstep
%-------------------------------------------------------------
%Pegar os valores de x e y do pé A e B e do CoM
%CoM - centro de massa
%-------------------------------------------------------------
    xa = pa(1,1);%posição x do pé A
    xb = pb(1,1);%posição x do pé B
    ya = pa(2,1);%posição y do pé A
    yb = pb(2,1);%posição x do pé B
    xc = pc(1,1);%posição x do CoM
    yc = pc(2,1);%posição x do CoM
%-------------------------------------------------------------
%cálcula a norma euclidiano do espaço (função objetivo - fo)
%-------------------------------------------------------------
    s(1,1) = (0.5*(xa + xb) - xc);
    s(2,1) = (0.5*(ya + yb) - yc);
    fo = norm(s).^2;
end
```

B.31 adagrad.m

```
%-------------------------------------------------------------
%Método do adagrad utilizado para
%otimizar a trajetória do centro de massa do modelo 3D dual SLIP
%-------------------------------------------------------------
function [U,G0] = adagrad(U0,X0,G0)
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
    global ganhoAlpha

    h = 10^-8;
%-------------------------------------------------------------
%valor do ganho anterior
%-------------------------------------------------------------
    G1 = G0(1,1);
    G2 = G0(2,1);
    G3 = G0(3,1);
    G4 = G0(4,1);
%-------------------------------------------------------------
%cálculo do gradiente da função
%-------------------------------------------------------------
    G = gradienteFuncao(X0,U0);
    g1 = G(1,1);
    g2 = G(2,1);
    g3 = G(3,1);
    g4 = G(4,1);
%-------------------------------------------------------------
%cálculo do novo ganho
%-------------------------------------------------------------
    G1 = G1 + g1*g1;
    G2 = G2 + g2*g2;
    G3 = G3 + g3*g3;
    G4 = G4 + g4*g4;
    G0 = [G1;G2;G3;G4];
%-------------------------------------------------------------
%atualizando o valor do vetor de controle - U
%-------------------------------------------------------------
```

```
    u1 = U0(1,1) - (ganhoAlpha/(sqrt(G1+h)))*g1;
    u2 = U0(2,1) - (ganhoAlpha/(sqrt(G2+h)))*g2;
    u3 = U0(3,1) - (ganhoAlpha/(sqrt(G3+h)))*g3;
    u4 = U0(4,1) - (ganhoAlpha/(sqrt(G4+h)))*g4;
    u5 = U0(5,1);
%-------------------------------------------------------------
%Valor atualizado
%-------------------------------------------------------------
    U = [u1;u2;u3;u4;u5];
end
```

B.32 rot2quat.m

```
%--------------------------------------
%Método para calcular o quatérnio equivalente
%dado uma matriz de rotação
%Parâmetros:
%T: Matriz de Rotação
%Retorno:
%q: quatérnio resultante
%--------------------------------------
function q = rot2quat(T)

    %iniciar o quarténio
    q = [0 0 0 0]';

    %Elementos da matriz de Rotação
    D1 = T(1,1);
    D2 = T(2,2);
    D3 = T(3,3);

    Au = T(1,2);
    Bu = T(1,3);
    Cu = T(2,3);

    Ad = T(2,1);
    Bd = T(3,1);
    Cd = T(3,2);
```

```
%solução baseada em w
bw = 1 + D1 + D2 + D3;
if bw > 0
    disp('w')
    w = sqrt(bw)/2;
    x = (Cd - Cu)/(4*w);
    y = (Bu - Bd)/(4*w);
    z = (Ad - Au)/(4*w);

    q = [w x y z ]';
    return
end

%solução baseada em x
if D1 > D2 & D1 > D3
    disp('x')
    bx = 1 + D1 - D2 -D3;
    x = sqrt(bx)/2;
    y = (Au+Ad)/(4*x);
    z = (Bu+Bd)/(4*x);
    w = (Cd - Cu)/(4*x);

    q = [w x y z ]';
    return
end

%solução baseada em y
if D2 > D1 & D2 > D3
    disp('y')
    by = 1 - D1 + D2 -D3;
    y = sqrt(by)/2;
    x = (Au+Ad)/(4*y);
    z = (Cd+Cu)/(4*y);
    w = (Bu - Bd)/(4*y);

    q = [ w x y z]';
    return
end
```

```
    %solução baseada em z
    if D3 > D1 & D3 > D2
        disp('z')
        bz = 1 - D1 - D2 +D3;
        z = sqrt(bz)/2;
        x = (Bu+Bd)/(4*z);
        y = (Cd+Cu)/(4*z);
        w = (Ad - Au)/(4*z);

        q = [w x y z]';
        return
    end
end
```

B.33 quatSub.m

```
%-------------------------------------
%Método para calcular a subtração de
%dois  quatérnios
%Parâmetros:
%q1: primeiro quatérnio
%q2: segundo quatérnio
%Retorno:
%qr: quatérnio resultante
%-------------------------------------
function qr = quatSub(q1,q2)
    %q = a + bi + cj + dk
    qr(1,1) = q1(1,1)- q2(1,1);%parte real
    qr(2,1) = q1(2,1)- q2(2,1);%imaginário i
    qr(3,1) = q1(3,1)- q2(3,1);%imaginário j
    qr(4,1) = q1(4,1)- q2(4,1);%imaginário k
end
```

B.34 quatScale.m

```
%-------------------------------------
%Método para calcular o produto escalar de um
%quatérnio
```

```
%Parâmetros:
%s: escalar
%q: quatérnio
%Retorno:
%qr: quatérnio resultante
%--------------------------------------
function qr = quatScale(s,q1)
    %q = a + bi + cj + dk
    qr(1,1) = s*q1(1,1);%parte real
    qr(2,1) = s*q1(2,1);%imaginário i
    qr(3,1) = s*q1(3,1);%imaginário j
    qr(4,1) = s*q1(4,1);%imaginário k
end
```

B.35 quatRot.m

```
%--------------------------------------
%Método para calcular a rotação dada por
%um quatérnio
%Parâmetros:
%p: ponto
%r: quatérnio de rotação
%Retorno:
%pr: ponto rotacionado
%--------------------------------------
function pr = quatRot(p,r)
%--------------------------------------
%cálcula a rotação pr = rpr*
%--------------------------------------
    pr = quatMult(r,quatMult(p,quatConj(r)));
end
```

B.36 quatNormalize.m

```
%--------------------------------------
%Método para normalizar um
%um quatérnio
%Parâmetros:
```

```
%q: quatérnio
%Retorno:
%qr: quatérnio resultante
%--------------------------------------
function qr = quatNormalize(q)
%--------------------------------------
%cálculo da norma do quatérnio
%--------------------------------------
    normq = quatNorm(q);
%--------------------------------------
%se a norma for igual a zero retorna o
%proprio quatérnio
%--------------------------------------
    if normq == 0
        qr = q;
    else
        qr = quatScale(1.0 / normq, q);
    end
end
```

B.37 quatNorm.m

```
%--------------------------------------
%Método para calcular a norma de
%um quatérnio
%Parâmetros:
%q: quatérnio
%Retorno:
%qr: quatérnio resultante
%--------------------------------------
function normq=quatNorm(q)
    normq = sqrt(q(1,1)*q(1,1) + q(2,1)*q(2,1) + q(3,1)*q(3,1) + q(4,1)*q(4,1));
end
```

B.38 quatMult.m

```
%--------------------------------------
%Método para calcular a multiplicação
```

```
%entre dois quatérnios
%Parâmetros:
%q1: primeiro quatérnio
%q2: segundo quatérnio
%Retorno:
%qr: quatérnio resultante
%--------------------------------------
function qr = quatMult(q1,q2)
    %q = a + bi + cj + dk
    qr(1,1) = q1(1,1)*q2(1,1) - q1(2,1)*q2(2,1) -
    q1(3,1)*q2(3,1) - q1(4,1)*q2(4,1);%parte real
    qr(2,1) = q1(1,1)*q2(2,1) + q1(2,1)*q2(1,1) +
    q1(3,1)*q2(4,1) - q1(4,1)*q2(3,1);%imaginário i
    qr(3,1) = q1(1,1)*q2(3,1) - q1(2,1)*q2(4,1) +
    q1(3,1)*q2(1,1) + q1(4,1)*q2(2,1);%imaginário j
    qr(4,1) = q1(1,1)*q2(4,1) + q1(2,1)*q2(3,1) -
    q1(3,1)*q2(2,1) + q1(4,1)*q2(1,1);%imaginário k
end
```

B.39 quatConj.m

```
%--------------------------------------
%Método para calcular o conjugado de
%um quatérnio
%Parâmetros:
%q: quatérnio
%Retorno:
%qr: quatérnio resultante
%--------------------------------------
function qr = quatConj(q)
    %q = a + bi + cj + dk
    qr(1,1) =  q(1,1);%parte real
    qr(2,1) = -q(2,1);%imaginário i
    qr(3,1) = -q(3,1);%imaginário j
    qr(4,1) = -q(4,1);%imaginário k
end
```

B.40 quatAdd.m

```
%-------------------------------------
%Método para calcular a soma de dois
%quatérnios
%Parâmetros:
%q1: primeiro quatérnio
%q2: segundo quatérnio
%Retorno:
%qr: quatérnio resultante
%-------------------------------------
function qr = quatAdd(q1,q2)
    %q = a + bi + cj + dk
    qr(1,1) = q1(1,1)+ q2(1,1);%parte real
    qr(2,1) = q1(2,1)+ q2(2,1);%imaginário i
    qr(3,1) = q1(3,1)+ q2(3,1);%imaginário j
    qr(4,1) = q1(4,1)+ q2(4,1);%imaginário k
end
```

B.41 quat2rot.m

```
%-------------------------------------
%Método para calcular a matriz de rotação
%equivalente dado um quatérnio
%Parâmetros:
%q: quatérnio
%Retorno:
%R: matriz de rotação
%-------------------------------------
function R = quat2rot(q)
    %inicializar variáveis
    w = q(1,1);
    x = q(2,1);
    y = q(3,1);
    z = q(4,1);
    %montar matriz de rotação
    R(1,1) = w*w + x*x - y*y - z*z;
    R(2,1) = 2*x*y + 2*z*w;
    R(3,1) = 2*x*z - 2*y*w;
```

```
    R(4,1) = 0;
    R(1,2) = 2*x*y - 2*z*w;
    R(2,2) = w*w - x*x + y*y - z*z;
    R(3,2) = 2*y*z + 2*x*w;
    R(4,2) = 0;
    R(1,3) = 2*x*z + 2*y*w;
    R(2,3) = 2*y*z - 2*x*w;
    R(3,3) = w*w - x*x - y*y + z*z;
    R(4,3) = 0;
    R(1,4) = 0;
    R(2,4) = 0;
    R(3,4) = 0;
    R(4,4) = 1;
end
```

B.42 HamiltonOp.m

```
%--------------------------------------
%Método para calcular o operador de Halmilton
%quatérnios
%Parâmetros:
%op: tipo do operador + ou -
%h - quatérnio
%Retorno:
%T: Matriz resultante
%--------------------------------------
function T = HamiltonOp(h,op)
    %quatérnio
    h1 = h(1,1);
    h2 = h(2,1);
    h3 = h(3,1);
    h4 = h(4,1);
    %Operador -
    H_ = [h1 -h2 -h3 -h4;
          h2  h1  h4 -h3;
          h3 -h4  h1  h2;
          h4  h3 -h2  h1];
    %operador +
    H = [h1 -h2 -h3 -h4;
```

```
         h2  h1 -h4  h3;
         h3  h4  h1 -h2;
         h4 -h3  h2 h1];
    %tipo do retorno
    if op == 1
        T = H;
    else
        T =H_;
    end
end
```

B.43 getAngleQuaternion.m

```
%----------------------------------------
%Método para retornar o valor do angulo de rotação de
%um quatérnio
%----------------------------------------
function theta = getAngleQuaternion(q)
    theta = acos(q(1,1));

    n1 = q(2:4,1)*(1/sin(theta));
    n2 = q(2:4,1)*(1/sin(-theta));

    q1 = [q(1,1);n1];
    if norm(q-q1) ~= 0
        theta = -theta
    end
end
```

B.44 trajetoria.m

```
%------------------------------------------------------------
%Método para calcular a trajetória do modelo dinâmico
%retornando os pontos dos pés, CoM e a trajetória
%Parâmetros:
%U - variáveis de controle
%X - vetor com as condições iniciais
%Retorno:
```

```
%pa - ponto do pé A
%pb - ponto do pé B
%pc - ponto do centro de massa
%M  - trajetória completa vetores x,y e z
%-------------------------------------------------------------
function [pa,pb,pc,M,ponto] = trajetoria(U,X,ind)
%-------------------------------------------------------------
%variáveis globais
%-------------------------------------------------------------
    global m L g pfa expK hEdo
%-------------------------------------------------------------
%condições iniciais para MS
%-------------------------------------------------------------
    xod  = X(1,1);
    yod  = X(2,1);
    zod  = X(3,1);
    dxod = X(4,1);
    dyod = X(5,1);
    dzod = X(6,1);
%-------------------------------------------------------------
%valores para a otimização valores de u
%-------------------------------------------------------------
    %u = [theta phi k Bss]
    theta = U(1,1);
    phi   = U(2,1);
    k     = U(3,1);
    Bss   = U(4,1);
    expK  = U(5,1);
%-------------------------------------------------------------
%vetor com as condições iniciais MS
%-------------------------------------------------------------
    y0 = [xod;dxod;yod;dyod;zod;dzod];
%-------------------------------------------------------------
%vetor com os parâmetros constantes
%-------------------------------------------------------------
    params = [m;L;g;k;Bss;expK];
%-------------------------------------------------------------
%Parâmetros para os métodos
%-------------------------------------------------------------
```

```
    t = 0;   %inicio do tempo  t = 0
    h = hEdo;%passo do método rungeKutta42 inicial
    N = 10000;%número máximo de iterações
    %primeiro metodo
    sh = h;%tamanho do passo para o método
    %rungeKutta42 atualizando durante a execução do método

    ind = 1;%contador
%-------------------------------------------------------------
%vetores auxiliares para guardar a trajetória
%-------------------------------------------------------------
    px(1,1) = y0(1,1);
    py(1,1) = y0(3,1);
    pz(1,1) = y0(5,1);
%-------------------------------------------------------------
%inicio do método 1 MS para TD
%-------------------------------------------------------------
    for x = 0:h:N*h
%-------------------------------------------------------------
%vetor de parâmetros
%-------------------------------------------------------------
        var = [t;h;1];
%-------------------------------------------------------------
%método numérico para solucionar as equações diferenciais
%passo a passo
%-------------------------------------------------------------
        [y sh] = rungeKutta42(var,y0,params);
%-------------------------------------------------------------
%atualizando a condição inicial
%-------------------------------------------------------------
        y0  = y;
%-------------------------------------------------------------
%atualizando o instante t
%-------------------------------------------------------------
        t = t+sh;
%-------------------------------------------------------------
%verificando a condição de parada posição Z < que Z de touchdown
%Z de touchdown = L*cos(theta)
%-------------------------------------------------------------
```

```
        if y0(5,1) < L*cos(theta)
            break
        end
%-------------------------------------------------------------
%colocando os valores nos vetores auxiliares
%-------------------------------------------------------------
        px(1,ind) = y0(1,1);
        py(1,ind) = y0(3,1);
        pz(1,ind) = y0(5,1);
%-------------------------------------------------------------
%atualizando o contador
%-------------------------------------------------------------
        ind = ind + 1;
    end
%-------------------------------------------------------------
%atualizando o contador - tratando o valor
%-------------------------------------------------------------
    if ind > 1
        ind = ind -1;
    end
    ponto = ind;
%-------------------------------------------------------------
%Posição do centro de massa no momento de  Touchdown (TD)
%-------------------------------------------------------------
    pc = [px(1,ind);py(1,ind);pz(1,ind)];%centro de massa
%-------------------------------------------------------------
%posição do pé de balaço quando toca o chão
%-------------------------------------------------------------
    pfb = pc + L*[sin(theta)*cos(phi);sin(theta)*sin(phi);-cos(theta)];
%-------------------------------------------------------------
%tempo em que acontece a codição de touchdown
%-------------------------------------------------------------
    TD = t;%tempo de TD
%-------------------------------------------------------------
%parametros constante para o segundo método
%-------------------------------------------------------------
    params = [m;L;g;k;Bss;t;pfb(1,1);pfb(2,1);pfb(3,1);expK];

%-------------------------------------------------------------
```

```
%iniciando o segundo contador
%-------------------------------------------------------------
    ind2 = 1;
    sh = h;%tamanho do passo para o método
    %rungeKutta42 atualizando durante a execução do método
%-------------------------------------------------------------
%vetores auxiliares para guardar a trajetória
%-------------------------------------------------------------
    px2(1,1) = y0(1,1);
    py2(1,1) = y0(3,1);
    pz2(1,1) = y0(5,1);
%-------------------------------------------------------------
%inicio do método 2  TD para LH
%-------------------------------------------------------------
    for x = 0:h:N*h

%-------------------------------------------------------------
%vetor de parâmetros
%-------------------------------------------------------------
        var = [t;h;0];
%-------------------------------------------------------------
%método numérico para solucionar as equações diferenciais
%passo a passo
%-------------------------------------------------------------
        [y sh] = rungeKutta42(var,y0,params);
%-------------------------------------------------------------
%atualizando nova condição inicial
%-------------------------------------------------------------
        y0  = y;
%-------------------------------------------------------------
%atualizando o instante t
%-------------------------------------------------------------
        t = t+sh;
%-------------------------------------------------------------
%verificando a condição de parada posição dZ > 0
%-------------------------------------------------------------
        if y0(6,1) > 0

            break
```

```
        end
%-------------------------------------------------------------
%atualizando os vetores auxiliares da trajetória
%-------------------------------------------------------------
        px2(1,ind2) = y0(1,1);
        py2(1,ind2) = y0(3,1);
        pz2(1,ind2) = y0(5,1);
%-------------------------------------------------------------
%atualizando o contador
%-------------------------------------------------------------
        ind2 = ind2+1;
    end
%-------------------------------------------------------------
%atualizando o contador - tratando o valor
%-------------------------------------------------------------
    if ind2 > 1
        ind2 = ind2 -1;
    end
%-------------------------------------------------------------
%trajetória do centro de massa CoM M = [x y z]
%-------------------------------------------------------------
    M = [px' py' pz';px2' py2' pz2'];
%-------------------------------------------------------------
%atualizando a posição do centro de massa
%-------------------------------------------------------------
    pc = [px2(1,ind2);py2(1,ind2);pz2(1,ind2)];%centro de massa
%-------------------------------------------------------------
%atualizando a posição dos pés e do centro de massa
%para o retorno da função
%-------------------------------------------------------------
    pa  = [0;0;0];
    pb  = [pfb(1,1);pfb(2,1);0];
    pc  = [pc(1,1);pc(2,1);pc(3,1)] ;
end
```

B.45 fase1.m

```
%-----------------------------------
%Método para executar o passo inicial
```

```
%-----------------------------------
function [ha,ha2,theta,tempo] = fase1(trajCoM1,ind,trajPB1,theta,vecGanho)
    global hpi L1 L2 L3 L4 L5 height MDH hEdo

    hOrg = [1 0 0 0 0 0 0 0]';%posição da base
    %calcular posição atual do pé
    n = [0 1 0];
    thetab = hpi;
    rb = [cos(thetab/2) sin(thetab/2)*n]';
    pb = [0 0 -L1 -height]';

    %base B para a base O6
    hB_O6 = transformacao(pb,rb);
    hB_O6 = dualQuatMult(hOrg,hB_O6);
    hP = hB_O6;

    dt = hEdo;%dt é o tempo da solução da equação Edo
    T = size(trajCoM1,1);
    %T = 100;
    tempo = (T-1)*dt;
    t = 1:1:T;

    %matrizes auxiliares
    Mhd = zeros(8,T);
    Mha = zeros(8,T);
    Mdhd = zeros(8,T);
    Mtheta = zeros(6,T);

    Mhd2 = zeros(8,T);
    Mha2 = zeros(8,T);
    Mdhd2= zeros(8,T);
    Mtheta2 = zeros(6,T);

    %calculo de Mhd - matriz de hd
    r = [1 0 0 0]';
    p = [0 0 -L1 -height]';
    hB1 = transformacao(p,r);%transformação base robô
    for i = 1:1:T
        p = [0 trajCoM1(i,:)]';
```

```
    r = [1 0 0 0]';
    hd = transformacao(p,r);
    hd = dualQuatMult(hB1,hd);
    Mhd(:,i) = hd;

    if i <=ind

        p = [0 trajPB1(i,:)]';
        n = [0 1 0];
        angulo = 90;
        r = [cos(angulo/2) sin(angulo/2)*n]';
        hd = transformacao(p,r);
        hd = dualQuatMult(hB1,hd);
        Mhd2(:,i) = hd;
    else
        Mhd2(:,i) = Mhd2(:,ind);
    end
end

ha = KinematicRobo(theta,hOrg,hP,1,1);
ha2 = KinematicRobo(theta,hOrg,hP,1,0);

Mdhd(:,1)  = (Mhd(:,1)  - Mhd(:,1))*(1/dt);
Mdhd2(:,1)  = (Mhd2(:,1)  - Mhd2(:,1))*(1/dt);

for i = 2:1:T
    dhd =  (Mhd(:,i) - Mhd(:,i-1))*(1/dt);
    Mdhd(:,i) = dhd;


    dhd2 =  (Mhd2(:,i) - Mhd2(:,i-1))*(1/dt);
    Mdhd2(:,i) = dhd2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%inicio do codigo
%LQR
ganhoS = vecGanho(1,1);
ganhoQ = vecGanho(2,1);
```

```
ganhoR = vecGanho(3,1);
%controlador proporcional
ganhoK2 = vecGanho(4,1);
K2 = ganhoK2*eye(8);

S = ganhoS*eye(8);
Q = ganhoQ*eye(8);
R = ganhoR*eye(8);
Rinv = inv(R);
C8 = diag([1 -1 -1 -1 1 -1 -1 -1]);

%iniciar condições finais esperadas para P e E
Pf = S;
Ef = [0 0 0 0 0 0 0 0]';

P = Pf;
MP2(:,T) = P(:);
E = Ef;
ME2(:,T) =E;

%calcular matrizes de ganho
for i=T-1:-1:1
    aux = dualQuatMult(dualQuatConj(Mhd(:,i+1)),Mdhd(:,i+1));
    A  = dualHamiltonOp(aux,0);
    c = -aux;
    P = P -(-P*A -A'*P + P*Rinv*P - Q)*dt;
    MP2(:,i) = P(:);
    E = E - (-A'*E + P*Rinv*E - P*c)*dt;
    ME2(:,i) = E;
end

for i=1:1:T
    tic
    %Controlador LQR para O CoM
    %calculo de A e c
    aux = dualQuatMult(dualQuatConj(Mhd(:,i)),Mdhd(:,i));
    A  = dualHamiltonOp(aux,0);
    c = -aux;
    %inicio do controlador
```

```
Ja = jacobianoCinematica(theta,hOrg,hP,1,1);
%calculo de P e E
%calculo de N
Hd  = dualHamiltonOp(Mhd(:,i),0);
N  = Hd*C8*Ja;
%pseudo inversa de N
Np  = pinv(N);

%calculo do erro
e  = [1 0 0 0 0 0 0 0]' - dualQuatMult(dualQuatConj(ha),Mhd(:,i));
%calculo de P e E
E = ME2(:,i);
P = reshape(MP2(:,i),size(A));
do = Np*Rinv*(P*e + E);
%calculo do o deseja
od  = (do*dt)/2;
theta(:,1) = theta(:,1) + od;

for j=1:1:6
    if abs(theta(j,1)) > hpi
        theta(j,1) = sign(theta(j,1))*hpi;
    end
end

ha  = KinematicRobo(theta,hOrg,hP,1,1);

%plotar os dados
Mha(:,i)   = ha;
%posição
Pos(:,i)  = getPositionDualQuat(ha);
Posd(:,i) = getPositionDualQuat(Mhd(:,i));
%orientação
ra = getRotationDualQuat(ha);
rd = getRotationDualQuat(Mhd(:,i));
co = acos(ra(1,1));
angle(1,i) = co;
co = acos(rd(1,1));
angled(1,i) = co;
Mtheta(:,i)       = theta(:,1);
```

```
%controlador 2
%calculo de A e c
aux2 = dualQuatMult(dualQuatConj(Mhd2(:,i)),Mdhd2(:,i));
A2  = dualHamiltonOp(aux2,0);
c = -aux2;
%inicio do controlador
Ja2 = jacobianoCinematica(theta,hOrg,hP,1,0);
%calculo de P e E
%calculo de N
Hd2  = dualHamiltonOp(Mhd2(:,i),0);
N2  = Hd2*C8*Ja2;
%pseudo inversa de N
Np2  = pinv(N2);

%calculo do erro
e2  = [1 0 0 0 0 0 0 0]'  -  dualQuatMult(dualQuatConj(ha2),Mhd2(:,i));

vec2 = dualQuatMult(dualQuatConj(ha2),Mdhd2(:,i));
%
do2 = Np2*(K2*e2-vec2);
od2  = (do2*dt)/2;

theta(:,2) = theta(:,2) + od2;
for j=1:1:6
    if abs(theta(j,2)) > hpi
        theta(j,2) = sign(theta(j,2))*hpi;
    end
end

ha2  = KinematicRobo(theta,hOrg,hP,1,0);

%plotar os dados
Mha2(:,i)   = ha2;
%posição
Pos2(:,i)  = getPositionDualQuat(ha2);
Posd2(:,i) = getPositionDualQuat(Mhd2(:,i));
%orientação
ra = getRotationDualQuat(ha2);
```

```
        rd = getRotationDualQuat(Mhd2(:,i));
        co = acos(ra(1,1));
        angle2(1,i) = co;
        co = acos(rd(1,1));
        angled2(1,i) = co;
        Mtheta2(:,i)       = theta(:,2);

        %mostrar no console o andamento do metódo
        msg = sprintf('%d de  %d | tempo (s): %f',i,T,toc);
        disp(msg);
    end
    t1 = 0;
    plotGraficosControle(t1,dt,T,Pos,Posd,angle,angled,Mha,Mhd,Mtheta,
    Pos2,Posd2,angle2,angled2,Mha2,Mhd2,Mtheta2,'b','r')
end
```

B.46 fase2.m

```
%-----------------------------------
%Método para executar o  passo com a perna esquerda como suporte da
%caminhada e a perna direita em movimento
%-----------------------------------
function [ha,ha2,theta,tempo] = fase2(ha,ha2,trajCoM,ind,trajPA,theta,t1,vecGanho)
    global hpi L1 L2 L3 L4 L5 height MDH hEdo
    dt = hEdo;%dt é o tempo da solução da equação Edo

    hOrg = [1 0 0 0 0 0 0 0]';%posição da base
    T = size(trajCoM,1);
    t = 1:1:T;
    tempo = (T-1)*dt;
    %matrizes auxiliares
    Mhd = zeros(8,T);
    Mha = zeros(8,T);
    Mdhd = zeros(8,T);
    Mtheta = zeros(6,T);

    Mhd2 = zeros(8,T);
    Mha2 = zeros(8,T);
    Mdhd2= zeros(8,T);
```

```
Mtheta2 = zeros(6,T);

%calculo de Mhd - matriz de hd
r = [1 0 0 0]';
%p = [0 0 0 0]';
p = [0 0 -L1 -height]';
hB1 = transformacao(p,r);%transformação base robô
for i = 1:1:T
    p = [0 trajCoM(i,:)]';
    r = [1 0 0 0]';
    hd = transformacao(p,r);
    hd = dualQuatMult(hB1,hd);
    Mhd(:,i) = hd;

    if i <=ind
        p = [0 trajPA(i,:)]';
        n = [0 1 0];
        angulo = 90;
        r = [cos(angulo/2) sin(angulo/2)*n]';
        hd = transformacao(p,r);
        hd = dualQuatMult(hB1,hd);
        Mhd2(:,i) = hd;
    else
        Mhd2(:,i) = Mhd2(:,ind);
    end
end

hP = ha2;
ha2 = KinematicRobo(theta,hOrg,hP,0,0);

Mdhd(:,1)  = (Mhd(:,1)  - Mhd(:,1))*(1/dt);
Mdhd2(:,1)  = (Mhd2(:,1)  - Mhd2(:,1))*(1/dt);

for i = 2:1:T
    dhd =  (Mhd(:,i) - Mhd(:,i-1))*(1/dt);
    Mdhd(:,i) = dhd;

    dhd2 =  (Mhd2(:,i) - Mhd2(:,i-1))*(1/dt);
    Mdhd2(:,i) = dhd2;
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%inicio do codigo
%LQR
ganhoS = vecGanho(1,1);
ganhoQ = vecGanho(2,1);
ganhoR = vecGanho(3,1);
%controlador proporcional

ganhoK2 = vecGanho(4,1);
K2 = ganhoK2*eye(8);

%ganho P-FF
S = ganhoS*eye(8);
Q = ganhoQ*eye(8);
R = ganhoR*eye(8);
Rinv = inv(R);
C8 = diag([1 -1 -1 -1 1 -1 -1 -1]);
%iniciar condições finais esperadas para P e E
Pf = S;
Ef = [0 0 0 0 0 0 0 0]';

P = Pf;
MP2(:,T) = P(:);
E = Ef;
ME2(:,T) =E;

for i=T-1:-1:1
    aux = dualQuatMult(dualQuatConj(Mhd(:,i+1)),Mdhd(:,i+1));
    A  = dualHamiltonOp(aux,0);
    c = -aux;
    P = P -(-P*A -A'*P + P*Rinv*P - Q)*dt;
    MP2(:,i) = P(:);
    E = E - (-A'*E + P*Rinv*E - P*c)*dt;
    ME2(:,i) = E;
end

for i=1:1:T
```

```
tic
%Controlador LQR para O CoM
%calculo de A e c
aux = dualQuatMult(dualQuatConj(Mhd(:,i)),Mdhd(:,i));
A  = dualHamiltonOp(aux,0);
c = -aux;
%inicio do controlador
Ja = jacobianoCinematica(theta,hOrg,hP,0,1);
%calculo de P e E
%calculo de N
Hd  = dualHamiltonOp(Mhd(:,i),0);
N  = Hd*C8*Ja;
%pseudo inversa de N
Np  = pinv(N);

%calculo do erro
e  = [1 0 0 0 0 0 0 0]' - dualQuatMult(dualQuatConj(ha),Mhd(:,i));
%calculo de P e E
E = ME2(:,i);
P = reshape(MP2(:,i),size(A));
do = Np*Rinv*(P*e + E);
%calculo do o deseja
od  = (do*dt)/2;

theta(:,2) = theta(:,2) + od;

for j=1:1:6
    if abs(theta(j,1)) > hpi
        theta(j,1) = sign(theta(j,1))*hpi;
    end
end

ha  = KinematicRobo(theta,hOrg,hP,0,1);

%controlador 2
%calculo de A e c
aux2 = dualQuatMult(dualQuatConj(Mhd2(:,i)),Mdhd2(:,i));
A2  = dualHamiltonOp(aux2,0);
c = -aux2;
```

```
%inicio do controlador
Ja2 = jacobianoCinematica(theta,hOrg,hP,0,0);
%calculo de P e E
%calculo de N
Hd2  = dualHamiltonOp(Mhd2(:,i),0);
N2  = Hd2*C8*Ja2;
%pseudo inversa de N
Np2  = pinv(N2);

%calculo do erro
e2  = [1 0 0 0 0 0 0 0]'  -  dualQuatMult(dualQuatConj(ha2),Mhd2(:,i));

vec2 = dualQuatMult(dualQuatConj(ha2),Mdhd2(:,i));
do2 = Np2*(K2*e2-vec2);
od2  = (do2*dt)/2;

theta(:,1) = theta(:,1) + od2;
ha2  = KinematicRobo(theta,hOrg,hP,0,0);

%plotar os dados
Mha(:,i)   = ha;
%posição
Pos(:,i)  = getPositionDualQuat(ha);
Posd(:,i) = getPositionDualQuat(Mhd(:,i));
%orientação
ra = getRotationDualQuat(ha);
rd = getRotationDualQuat(Mhd(:,i));
co = acos(ra(1,1));
angle(1,i) = co;
co = acos(rd(1,1));
angled(1,i) = co;
Mtheta(:,i)       = theta(:,1);

%plotar os dados
Mha2(:,i)   = ha2;
%posição
Pos2(:,i)  = getPositionDualQuat(ha2);
Posd2(:,i) = getPositionDualQuat(Mhd2(:,i));
%orientação
```

```
        ra = getRotationDualQuat(ha2);
        rd = getRotationDualQuat(Mhd2(:,i));
        co = acos(ra(1,1));
        angle2(1,i) = co;
        co = acos(rd(1,1));
        angled2(1,i) = co;
        Mtheta2(:,i)       = theta(:,2);

        %mostrar no console o andamento do metódo
        msg = sprintf('%d de  %d | tempo: %f',i,T,toc);
        disp(msg);
    end
    hold on
    plotGraficosControle(t1,dt,T,Pos,Posd,angle,angled,Mha,Mhd,Mtheta,
    Pos2,Posd2,angle2,angled2,Mha2,Mhd2,Mtheta2,'r','b')
end
```

B.47 fase3.m

```
%-------------------------------------------
%Método para executar o  passo com a perna direita como suporte da
%caminhada e a perna esquerda em movimento
%-------------------------------------------
function [ha,ha2,theta,tempo] = fase3(ha,ha2,trajCoM,ind,trajPB,theta,t1,vecGanho)
    global hpi L1 L2 L3 L4 L5 height MDH hEdo
    dt = hEdo;%dt é o tempo da solução da equação Edo
    hOrg = [1 0 0 0 0 0 0 0]';%posição da base
    T = size(trajCoM,1);
    %T = 500;
    t = 1:1:T;
    tempo = (T-1)*dt;
    %matrizes auxiliares
    Mhd = zeros(8,T);
    Mha = zeros(8,T);
    Mdhd = zeros(8,T);
    Mtheta = zeros(6,T);

    Mhd2 = zeros(8,T);
    Mha2 = zeros(8,T);
```

```
Mdhd2= zeros(8,T);
Mtheta2 = zeros(6,T);

%calculo de Mhd - matriz de hd
r = [1 0 0 0]';
%p = [0 0 0 0]';
p = [0 0 -L1 -height]';
hB1 = transformacao(p,r);%transformação base robô
for i = 1:1:T
    p = [0 trajCoM(i,:)]';
    r = [1 0 0 0]';
    hd = transformacao(p,r);
    hd = dualQuatMult(hB1,hd);
    Mhd(:,i) = hd;

    if i <=ind
        p = [0 trajPB(i,:)]';
        n = [0 1 0];
        angulo = 90;
        r = [cos(angulo/2) sin(angulo/2)*n]';
        hd = transformacao(p,r);
        hd = dualQuatMult(hB1,hd);
        Mhd2(:,i) = hd;
    else
        Mhd2(:,i) = Mhd2(:,ind);
    end
end

hP = ha2;

Mdhd(:,1)  = (Mhd(:,1)  - Mhd(:,1))*(1/dt);
Mdhd2(:,1)  = (Mhd2(:,1)  - Mhd2(:,1))*(1/dt);

for i = 2:1:T
    dhd =  (Mhd(:,i) - Mhd(:,i-1))*(1/dt);
    Mdhd(:,i) = dhd;

    dhd2 =  (Mhd2(:,i) - Mhd2(:,i-1))*(1/dt);
    Mdhd2(:,i) = dhd2;
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%inicio do codigo
%LQR
ganhoS = vecGanho(1,1);
ganhoQ = vecGanho(2,1);
ganhoR = vecGanho(3,1);
%controlador proporcional

ganhoK2 = vecGanho(4,1);
K2 = ganhoK2*eye(8);

%ganho P-FF
S = ganhoS*eye(8);
Q = ganhoQ*eye(8);
R = ganhoR*eye(8);
Rinv = inv(R);
C8 = diag([1 -1 -1 -1 1 -1 -1 -1]);
%iniciar condições finais esperadas para P e E
Pf = S;
Ef = [0 0 0 0 0 0 0 0]';

P = Pf;
MP2(:,T) = P(:);
E = Ef;
ME2(:,T) =E;


for i=T-1:-1:1
    aux = dualQuatMult(dualQuatConj(Mhd(:,i+1)),Mdhd(:,i+1));
    A  = dualHamiltonOp(aux,0);
    c = -aux;
    P = P -(-P*A -A'*P + P*Rinv*P - Q)*dt;
    MP2(:,i) = P(:);
    E = E - (-A'*E + P*Rinv*E - P*c)*dt;
    ME2(:,i) = E;
end
```

```
for i=1:1:T
    tic
    %Controlador LQR para O CoM
    %calculo de A e c
    aux = dualQuatMult(dualQuatConj(Mhd(:,i)),Mdhd(:,i));
    A  = dualHamiltonOp(aux,0);
    c = -aux;
    %inicio do controlador
    Ja = jacobianoCinematica(theta,hOrg,hP,1,1);
    %calculo de P e E
    %calculo de N
    Hd  = dualHamiltonOp(Mhd(:,i),0);
    N  = Hd*C8*Ja;
    %pseudo inversa de N
    Np  = pinv(N);

    %calculo do erro
    e  = [1 0 0 0 0 0 0 0]' - dualQuatMult(dualQuatConj(ha),Mhd(:,i));
    %calculo de P e E
    E = ME2(:,i);
    P = reshape(MP2(:,i),size(A));
    do = Np*Rinv*(P*e + E);
    %calculo do o deseja
    od  = (do*dt)/2;

    theta(:,1) = theta(:,1) + od;

    for j=1:1:6
        if abs(theta(j,1)) > hpi
            theta(j,1) = sign(theta(j,1))*hpi;
        end
    end

    ha  = KinematicRobo(theta,hOrg,hP,1,1);

    %plotar os dados
    Mha(:,i)   = ha;
    %posição
    Pos(:,i)  = getPositionDualQuat(ha);
```

```
Posd(:,i) = getPositionDualQuat(Mhd(:,i));
%orientação
ra = getRotationDualQuat(ha);
rd = getRotationDualQuat(Mhd(:,i));
co = acos(ra(1,1));
angle(1,i) = co;
co = acos(rd(1,1));
angled(1,i) = co;
Mtheta(:,i)       = theta(:,1);

%controlador 2
%calculo de A e c
aux2 = dualQuatMult(dualQuatConj(Mhd2(:,i)),Mdhd2(:,i));
A2  = dualHamiltonOp(aux2,0);
c = -aux2;
%inicio do controlador
Ja2 = jacobianoCinematica(theta,hOrg,hP,1,0);
%calculo de P e E
%calculo de N
Hd2  = dualHamiltonOp(Mhd2(:,i),0);
N2  = Hd2*C8*Ja2;
%pseudo inversa de N
Np2  = pinv(N2);

%calculo do erro
e2  = [1 0 0 0 0 0 0 0]'  -  dualQuatMult(dualQuatConj(ha2),Mhd2(:,i));

vec2 = dualQuatMult(dualQuatConj(ha2),Mdhd2(:,i));
%
do2 = Np2*(K2*e2-vec2);
od2  = (do2*dt)/2;

theta(:,2) = theta(:,2) + od2;

ha2  = KinematicRobo(theta,hOrg,hP,1,0);

%plotar os dados
Mha2(:,i)   = ha2;
%posição
```

```
        Pos2(:,i)  = getPositionDualQuat(ha2);
        Posd2(:,i) = getPositionDualQuat(Mhd2(:,i));
        %orientação
        ra = getRotationDualQuat(ha2);
        rd = getRotationDualQuat(Mhd2(:,i));
        co = acos(ra(1,1));
        angle2(1,i) = co;
        co = acos(rd(1,1));
        angled2(1,i) = co;
        Mtheta2(:,i)       = theta(:,2);

        %mostrar no console o andamento do metódo
        msg = sprintf('%d de  %d | tempo: %f',i,T,toc);
        disp(msg);
    end
    hold on
    plotGraficosControle(t1,dt,T,Pos,Posd,angle,angled,Mha,Mhd,Mtheta,
    Pos2,Posd2,angle2,angled2,Mha2,Mhd2,Mtheta2,'b','r')
end
```

B.48 caminhada.m

```
%----------------------------------------
%Executar a caminhada
%Parâmetros:
%U:Vetor com as variáveis de controle
%X:Vetor com a condição inicial
%vecGanho1:vetor com os ganhos do controlador para a condição da perna
%          direita em contato com chão
%vecGanho2:vetor com os ganhos do controlador para a condição da perna
%          esquerda em contato com o chão
%----------------------------------------
function caminhada(U,X,tam,vecGanho1,vecGanho2)
    %-------------------------------------------------------------
    %Obter todas as trajeotrias do CoM
    %-------------------------------------------------------------
    %trajetória para o CoM
    [PA,PB,PC,trajCoM1,indContadoPe] = trajetoria(U,X,1);
```

```
%trajetoria 2
CoM = trajCoM1;
ind = size(CoM,1);%pegar a última posição do vetor de pontos
offsetx = CoM(ind,1);%cálcular o offset em x
offsety = CoM(ind,2);%calcular o offset em y
trajCoM2(:,1) = -CoM(ind:-1:1,1) + 2*offsetx;
trajCoM2(:,2) = -CoM(ind:-1:1,2) + 2*offsety;
trajCoM2(:,3) =  CoM(ind:-1:1,3);%em z não muda

offsetx = trajCoM2(ind,1);%cálcular o offset em x
offsety = trajCoM2(ind,2);%calcular o offset em y
trajCoM3(:,1) = -trajCoM2(ind:-1:1,1) + 2*offsetx;
trajCoM3(:,2) = trajCoM2(ind:-1:1,2) ;
trajCoM3(:,3) =  trajCoM2(ind:-1:1,3);%em z não muda

trajCoM2 = [trajCoM2;trajCoM3];

passoTrajCoM = trajCoM2(size(trajCoM2,1),1) - trajCoM2(1,1);


%trajetoria3
ind = size(trajCoM2,1);%pegar a última posição do vetor de pontos
offsetx = trajCoM2(ind,1);%cálcular o offset em x
offsety = trajCoM2(ind,2);%calcular o offset em y
clear trajCoM3
trajCoM3(:,1) = -trajCoM2(ind:-1:1,1) + 2*offsetx;
trajCoM3(:,2) = -trajCoM2(ind:-1:1,2) + 2*offsety;
trajCoM3(:,3) =  trajCoM2(ind:-1:1,3);%em z não muda

%-------------------------------------------------------------
%Trajetória dos pés
%-------------------------------------------------------------
passoComprimento = PB(1,1);%tamanho do passo
passoLargura     = PB(2,1);%Largura do passo
passoAltura      = 0.2;    %altura de cada passo

%trajetoria pé B inicial
tamTrajPeB1 = indContadoPe;
trajPB1 = trajetoriaPes([passoComprimento;passoLargura;0],
```

```
passoComprimento,passoAltura,1,tamTrajPeB1);


passoComprimento2 = PB(1,1);%tamanho do passo
passoLargura2     = 0;%Largura do passo
passoAltura2      = 0.2;    %altura de cada passo

%trajetoria pé A
tamTrajPa= (size(trajCoM1,1)+indContadoPe)/2;
trajPA = trajetoriaPes([passoComprimento2+passoComprimento2
;passoLargura2;0],passoComprimento2,passoAltura2,0,tamTrajPa);
%trajtoria pé B
trajPB = trajPA;
trajPB(:,1) = trajPB(:,1) + passoComprimento;
trajPB(:,2) = passoLargura;

%-----------------------------------------------
%Modelo do robô
%-----------------------------------------------
%parâmetros necessarios
global hpi L1 L2 L3 L4 L5 height MDH
hpi = pi/2;
N = 6;%Numero de Juntas
%Comprimento das Juntas
%parametros hubo
L1 = 0.085;
L2 = 0.0;
L3 = 0.3;
L4 = 0.3;
L5 = 0.0663;
%altura inicial total do robô pé para o centro de massa
height = L2+L3+L4+L5;
%Parametros de D.H. Perna- tabela do artigo
oi = [ 0 -hpi 0 0 0 0]';
di = [ 0 0 0 0 0 0]';
ai = [ 0 0 L3 L4 0 L5]';
si = [ hpi -hpi 0 0 hpi 0]';
or = [0 0 0 0 0 0 ]';%perna direita
ol = [0 0 0 0 0 0 ]';%perna esquerda
```

```
%matriz dos parametros de D.H
MDH  = [oi di ai si];%igual para as duas pernas
%matriz com os parametros variaveis
theta = [or ol];% parametros variaveis
tempo = 0;

%primeira parte da caminhdada
passos = 1;
[ha,ha2,theta,tempo1] = fase1(trajCoM1,indContadoPe,trajPB1,theta,vecGanho1);

if passos >=tam
    return
end

tempo = tempo+tempo1;
i = 0;
while 1

    trajCoM = trajCoM2;
    trajCoM(:,1) = trajCoM(:,1) + i*2*passoTrajCoM;
    trajP = trajPA;
    trajP(:,1) = trajP(:,1)+ i*2*passoComprimento;

    passos = passos + 1;
    [ha,ha2,theta,tempo2] = fase2(ha,ha2,trajCoM,size(trajPA,1)
    ,trajP,theta,tempo,vecGanho2);

    if passos >=tam
        return
    end

    tempo=tempo+tempo2;

    trajCoM = trajCoM3;
    trajCoM(:,1) = trajCoM(:,1) + i*2*passoTrajCoM;
    trajP = trajPB;
    trajP(:,1) = trajP(:,1)+ i*2*passoComprimento;
    passos = passos + 1;
    [ha,ha2,theta,tempo3] = fase3(ha,ha2,trajCoM,size(trajPB,1),
```

```
        trajP,theta,tempo,vecGanho1);

        if passos >=tam
            return
        end

        tempo=tempo+tempo3;
        i = i+1;
    end

end
```

B.49 trajetoriaPes.m

```
%---------------------------------------
%Calcular a trajetória dos pés
%Função de Bézier de 4ª ordem
%---------------------------------------
function trajPes = trajetoriaPes(posP,passo,altura,metade,tam)

    p0 = [posP(1,1);posP(3,1)] + [-2*passo;0];
    p1 = [posP(1,1);posP(3,1)] + [-passo;altura];
    p2 = [posP(1,1);posP(3,1)];

    dt = 1/(2*tam-1);
    t = 0:dt:1;
    ind = size(t,2);

    for i = 1:1:ind
        B(i,1) = (1 - t(1,i))^2 * p0(1,1) +
        2*t(1,i)*(1 - t(1,i))*p1(1,1) + t(1,i)*t(1,i)*p2(1,1);
        B(i,2) = (1 - t(1,i))^2 * p0(2,1) +
        2*t(1,i)*(1 - t(1,i))*p1(2,1) + t(1,i)*t(1,i)*p2(2,1);
    end

    if metade
        trajPes = [B(tam+1:ind,1),posP(2,1)*ones(tam,1),B(tam+1:ind,2)];
    else
        trajPes = [B(:,1),posP(2,1)*ones(ind,1),B(:,2)];
```

```
    end
end
```

B.50 transformacao.m

```
%-----------------------------------------
%Método para retornar a transformação
%dado a posição e orientação
%Parâmetros:
%p - vetor de translação do sistema base
%r - orientação
%Retorno:
%h0 - quatérnio dual de transformação
%-----------------------------------------
function h0 = transformacao(p,r)
    qp = r;%parte primaria
    %0.5pr
    qd = 0.5*quatMult(p,r);%parte dual
    %quatérnio dual de posição e orientação atual
    h0 = [qp;qd];
end
```