TRABALHO DE GRADUAÇÃO

APRENDIZADO DE MARCHAS PARA UM ROBÔ QUADRÚPEDE UTILIZANDO ALGORITMO GENÉTICO MULTIOBJETIVO

Pedro Matheus Filadelfo dos Santos

Vitor Alves Duarte

Brasília, julho de 2018

ENGENHARIA MECATRÔNICA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

APRENDIZADO DE MARCHAS PARA UM ROBÔ QUADRÚPEDE UTILIZANDO ALGORITMO GENÉTICO MULTIOBJETIVO

Pedro Matheus Filadelfo dos Santos

Vitor Alves Duarte

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Professor Alexandre R. S. Romariz, ENE/UnB _____
Orientador

Professor Geovany A. Borges, ENE/UnB _____
Coorientador

Profa. Carla C. Koike, CIC/UnB _____
Examinador interno

Prof. José Edil G. de Medeiros, ENE/UnB _____
Examinador interno

Brasília, julho de 2018

FICHA CATALOGRÁFICA

PEDRO, MATHEUS FILADELFO DOS SANTOS; VITOR, ALVES DUARTE;
Aprendizado de marchas para um robô quadrúpede utilizando algoritmo genético multiobjetivo,

[Distrito Federal] 2018.

xiii, 54p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2018). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Algoritmo Genético 2.Geração de Marchas
3. Robô Quadrúpede

I. Mecatrônica/FT/UnB II. Controle e Automação

REFERÊNCIA BIBLIOGRÁFICA

SANTOS, P. M .F d.; DUARTE, V. A., (2018). Aprendizado de marchas para um robô quadrúpede utilizando algoritmo genético multiobjetivo. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-$n°2$, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 123p.

CESSÃO DE DIREITOS

AUTORES: Pedro Matheus Filadelfo dos Santos e Vitor Alves Duarte.

TÍTULO DO TRABALHO DE GRADUAÇÃO: Aprendizado de marchas para um robô quadrúpede utilizando algoritmo genético multiobjetivo.

GRAU: Engenheiro ANO: 2018

Pedro Matheus Filadelfo dos Santos e Vitor Alves Duarte

Campus Darcy Ribeiro, SG-11, Universidade de Brasília.

70919-970 Brasília – DF – Brasil.

Dedicatórias

Dedico esse trabalho aos meus pais, Maura e José, aos meus avós e aos meus irmãos.

Vitor Alves Duarte

Dedico esse trabalho aos meus pais, Fabiana e Jurandir,à minha família e amigos..

Pedro Matheus Filadelfo dos Santos

Agradecimentos

Agradeço primeiramente aos meus pais que sempre me apoiaram, me motivaram e me entenderem durante esta trajetória. Agradeço à minha família que sempre torceu por mim e me ajudou a sempre acreditar no meu potencial. Agradeço à Equipe DROID por ter me acolhido quando ainda calouro na universidade e ter me ensinado sobre muitos assuntos acerca do curso. Agradeço à Mecajun por ter sido minha primeira experiência em uma empresa.Agradeço a Vitor Duarte, que foi meu parceiro no desenvolvimento deste e de muitos outro trabalhos realizados durante minha graduação, e um grande amigo. Agradeço à Gabriela Loureiro por tornar este ano muito mais leve, sempre me apoiando e torcendo por mim. Agradeço a meus amigos, em especial Gian Pietro, Pedro Gusmão, Marcos Vinícius, Matheus Alsan, Lucas Honda, Ygor Borgonove, Fernanda Ramos, Daniel Oliveira, Lucas Lacerda, Igor Beduin, Thiago Holanda, Jorge Simeão, Danilo Li, Rafael Pojo, Ricardo Bauchspiess e ao grupo TeaBags, que tornaram essa trajetória muito mais divertida e foram excelentes em me apoiar. Agradeço aos meus orientadores Alexandre Romariz e Geovany Borges por me guiarem no desenvolvimento desse projeto.

Pedro Matheus Filadelfo dos Santos

Agradeço primeiramente aos meus pais por sempre me motivarem a acreditar na educação como o mais importante dos agentes transformadores. Agradeço aos meus irmãos Jaqueline, Sheila, Danilo, Deivison e Camilla por sempre acreditarem no meu potencial e compreenderem minhas ausências. Agradeço aos meus avós e de certa forma a todos os meus antepassados, que por sua história de luta puderam construir todos os degraus necessários para a minha conquista. Agradeço a Flávia, por me motivar numa das fases mais difíceis da graduação, sempre me fazendo acreditar no meu potencial. Agradeço a equipe UnBall e a todos os integrantes, que me proporcionaram uma das melhores experiências da graduação. Agradeço a todos os amigos que estiveram comigo nos bons e maus momentos, em especial Fernando Rodrigues, Leonardo Lucena, Pedro Filhusi, Sara Petruce, Pedro Albuquerque, Amanda Barros, Mariana Madureira, Guilherme Rocha, Larissa Marchelly, Igor Beduin e Manoel Vieira. Além disso, agradeço ao Pedro que me acompanhou não só no trabalho de conclusão final mas também ao longo do curso, se tornando um dos meus melhores amigos. Por fim agradeço ao Alexandre Romariz e Geovany Borges que nos ofereceram todo o auxilio e orientação para a realização deste trabalho.

Vitor Alves Duarte

RESUMO

Este trabalho trata sobre o desenvolvimento de marchas factíveis para um robô quadrúpede utilizando algoritmo genético multiobjetivo. Desenvolveram-se três possíveis metodologias de otimização testadas em simulação computacional, utilizando Matlab e V-REP. A primeira consiste na busca de um conjunto de ângulos para as juntas do robô que formem uma marcha factível a ele. Isso se mostrou irrealizável, se tratando de uma otimização num espaço muito amplo de soluções. Sendo assim, a segunda abordagem envolve restringir esse espaço utilizando um conjunto de poses para as patas do robô, inspiradas em animais, com o intuito que o algoritmo encontre a melhor combinação dessas poses ao robô. A terceira é dada pela fixação de uma ordem de poses, a que se acredita ser a melhor para o quadrúpede, buscando otimizar alguns parâmetros relacionados à essas poses. Por fim, visando superar as diferenças do robô simulado com o real, faz-se a otimização, utilizando a terceira metodologia proposta, na plataforma real.

Palavras Chave: Quadrúpede, Algoritmo genético, marchas, Matlab, V-REP.

ABSTRACT

This work is about the development of feasible gaits for a quadruped robot using a multiobjective genetic algorithm. Three optimization methodologies have been developed and tested in computational simulation using Matlab and V-REP. The first one consists in a search of a set of angles for the robot's joints that results in a feasible gait to him. This has shown intractable due to the large solution space. The second one envolves reducing this space by using a set of poses, biologically inspireds, to the robot's paws, with the intent that the algorithm finds the best combination of these poses. The third one is given by fixing a order of the poses, that is believed to be the best for the quadruped, looking to optimizate some parameters related to them. And finally, looking to cross the reality gap problem, the optimization is performed, using the third methodology, on the real plataform.

Keywords: Quadruped, Genetic Algorithm, gaits, Matlab, V-REP.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

s	Posição da extremidade da pata	[m]
e	Erro entre as posições	[m]
g	Aceleração da gravidade	$[m/s^2]$
a	Aceleração	$[m/s^2]$
L	Comprimento	$[m]$

Símbolos Gregos

θ	Ângulo das juntas	[rad]
δ	Operador de derivação	
Δ	Variação	
α	Ângulo de arfagem	[rad]
ψ	Ângulo de guinada	[rad]
ϕ	Ângulo de rolagem	[rad]
π	Constante	

Grupos Adimensionais

J	Jacobiano
R	Matriz de rotação
p	Probabilidade de reprodução
n	Número de indivíduos de uma população
Z	Espaço de objetivo
R	Conjunto de número reais
S	Espaço de variáveis de decisão
d	Distância da multidão
A	Matriz de restrição
b	Matriz de restrição

Subscritos

i, j,m, k	Índices
c	Restante da população
m	mutação
eq	Igual

Sobrescritos

$\rightarrow$	Vetor
L	Inferior
U	Superior

Siglas

AG	Algoritmo Genético *Genetic Algorithm*
EO	Otimização Evolucionária - *Evoltionary Optmization*
EMO	Otimização Evolucionária multi-objetiva - *Evolutionary Multi-objective Optimization*
IDE	Ambiente de desenvolvimento integrado - *Integrated Development Environment*
MESM	*Micro Electro Mechanical System*
V-REP	*Virtual Robot Experimentation Plataform*
LARA	Laboratório de Automação e Robótica
IMU	Unidade de medida inercial - *Inertial measurement unit*
LED	Diodo emissor de luz - *Light emitter diode*
HyQ	*Hydraulic Quadruped*
USB	*Universal Serial Bus*
DC	Corrente direta - *Direct Current*
HDMI	*High Definition Multimedia Interface*
SD	*Secure Digital*
GPIO	*General Purpose Input/Output*
RAM	*Rapid Access Memory*
LPDDR	*Low Power Double Data Rate*
NSGA	*Elitist Non-dominated Sorting GA*
API	Interface de Programação de Aplicação - *Application Programming Interface*
ROS	Sistema Operacional para robôs - *Robot Operating System*
GPS	Sistema de posicionamento global - *Global Positioning System*
SSH	Secure Shell
UnB	Universidade de Brasília

Capítulo 1

Introdução

1.1 Contextualização

A implementação de robôs bio-inspirados vem sendo estudada de forma crescente ao longo do tempo. Isso ocorre com a necessidade de obtenção ou melhoria de características presentes em organismos biológicos, por parte de dispositivos robóticos. Dentre essas características pode-se citar a adaptabilidade a diferentes superfícies e alta mobilidade em terrenos irregulares.

Sendo a mobilidade um dos anseios principais para a implementação de um robô bio-inspirado, se destacam os robôs inspirados em animais quadrúpedes, uma vez que são esses animais que atingem as maiores velocidades terrestres.

A primeira aplicação possível para robôs desse tipo é no resgate de vítimas em áreas de risco. Essa arquitetura permite alcançar locais que não são possíveis de se alcançar por robôs que se movimentam por rodas. Ambientes periculosos que por muitas vezes só poderiam ser acessados por humanos, podem ser alcançados por dispositivos robóticos, permitindo realizar com segurança um resgate que antes seria de alto risco. O robô HyQ (Hydraulic Quadruped) [1], desenvolvido no Instituto Italiano de Tecnologia, foi criado com esse objetivo.

Outra aplicação interessante para uma plataforma quadrúpede é a de carregamento de cargas. Sua arquitetura permite essa aplicação devido a distribuição do peso entre as patas, proporcionando um alivio a carga existente nos motores. A implementação mais famosa de um robô quadrúpede focado no transporte de cargas é o BigDog, a plataforma que pode ser observada na Figura 1.1. Esse dispositivo foi criado pela Boston Dynamics e consegue atingir uma velocidade de 10 km/h, caminhar em superfícies com inclinação de até 35 graus e carregar até 150kg.

Desde 2015, são realizados trabalhos para a implementação e aperfeiçoamento de um robô quadrúpede no Laboratório de Automação e Robótica (LARA) da Universidade de Brasília (UnB). Varias foram as abordagens utilizando esse dispositivo, indo desde implementação de algoritmos comportamentais até a implementação de métodos de controle de estabilidade.

[1] https://youtu.be/j54gFCyWjAY

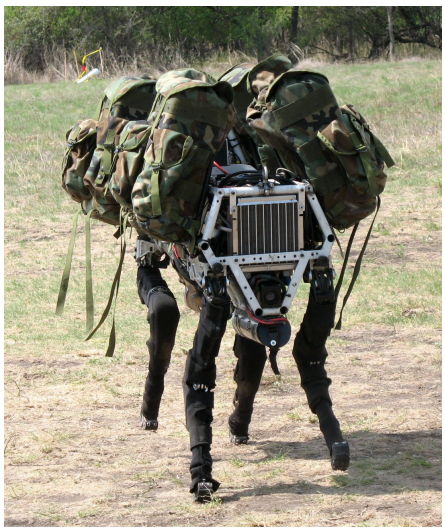


Figura 1.1: Robô BigDog desenvolvido pela Boston Dynamics (Fonte [1])

1.2 Definição do problema

Apesar de diversos trabalhos serem realizados na plataforma quadrúpede do LARA, não foi possível a realização de marchas satisfatórias no dispositivo. Isso se deu devido à necessidade de inúmeras melhorias mecânicas serem demandadas durante as execuções dos trabalhos anteriores.

Uma vez que a plataforma se mostrou suficientemente robusta para a realização das marchas, Porphirio e Santana [9] obtiveram-as via simulação e as implementaram no robô. As marchas executadas geravam resultados muito discrepantes em relação aos simulados.

Nota-se portanto, no atual estado da plataforma e das interfaces de simulação, uma diferença entre a marcha necessária para a movimentação satisfatória do robô e as simuladas.

1.3 Objetivos do projeto

O objetivo do projeto é, utilizando algoritmo genético multiobjetivo, encontrar marchas factíveis ao robô, que exploram suas próprias dinâmicas, não somente as observadas em animais. Para isso, deseja-se chegar à implementação física, ou seja, superar as diferenças entre a simulação e a vida real. Assim, ao fim desse trabalho a plataforma deverá realizar as marchas mais adequadas ao seu funcionamento.

1.4 Resultados obtidos

Os primeiros resultados adquiridos foram alguns ajustes mecânicos na plataforma. Alterou-se também o programa de execução para aumentar a taxa de amostragem dos sensores durante as marchas. Criou-se um código em python para controlar remotamente a plataforma, que facilitava a calibração e a execução de marchas na plataforma.

Ao fim das otimizações adquiriu-se algumas marchas factíveis ao robô da simulação,

1.5 Apresentação do manuscrito

No capítulo 2 faz-se uma revisão bibliográfica acerca dos conceitos teóricos utilizados neste trabalho. Nele se trata sobre algoritmo genético, otimização multiobjetiva, o software de simulação, cinemática inversa e funcionamento do robô em geral. No capítulo 3 se expõe a metodologia utilizada no desenvolvimento das marchas, tanto na simulação quanto na plataforma real. No capítulo 4 apresentam-se todos os resultados obtidos, como gráficos de posições dos motores, valores de sensores, etc. E por fim, o capítulo 5 é onde se encontra as conclusões do projeto e suas perspectivas futuras.

Capítulo 2

Fundamentos

Este capítulo trata sobre os fundamentos utilizados como referência para este trabalho. Comenta-se brevemente sobre o histórico da plataforma quadrúpede e em seguida expõe-se conceitos como cinemática inversa, funcionamento dos dispositivos embarcados (sensores, arduino e Raspberry PI), algoritmo genético, otimização multiobjetiva e vão da realidade. Ainda é dito como funcionam os software de simulação V-REP e a ferramenta de otimização utilizada, o gamultiobj.

2.1 Histórico da Plataforma

O primeiro trabalho realizado no LARA, visando o estudo de robôs multiarticulados do tipo quadrúpede foi concluído no ano de 2006. Os dois principais objetos de estudo, idealizados com a construção dessa plataforma, eram robótica terrestre e robótica comportamental. Ao longo dos anos, novos trabalhos foram realizados tendo esse dispositivo como foco. Seus objetivos foram diversos, perpassando desde melhorias mecânicas, até a implementação de modelos comportamentais para a interação do robô com agentes externos. A tabela 2.1 lista os projetos desenvolvidos, até o momento, utilizando a plataforma quadrúpede.

No ano de 2006, foram realizados dois trabalhos. Cotta e Neto [2], no projeto inaugural da plataforma, construíram a estrutura mecânica e as placas de acionamento dos servos, concebendo a primeira versão do robô, representada na Figura 2.1. Esse estudo destacou-se pela modelagem e análise das cinemática direta e inversa da plataforma. No segundo trabalho do ano, Calmon, Pinheiro e Ferreira [10] adentraram no campo da robótica comportamental. Foi desenvolvido um modelo, utilizando aprendizado por reforço, para possibilitar interação do robô com agentes externos. Esse estudo resultou grandes melhorias na plataforma, tais quais: implementação de um sistema de sensoriamento e arquitetura de comunicação.

Em 2007, Souto [5] propôs melhorias no modelo cinemático da plataforma, utilizando filtragem estocástica. Esse projeto se destacou pelo uso de filtro de Kalman estendido, como otimizador

Tabela 2.1: Projetos desenvolvidos utilizando a plataforma quadrúpede

Autores	**Ano**	**Título**
COTTA, G. H.; NETO, L. R	2006	Realização de uma plataforma para estudo de robótica comportamental baseada em quadrúpedes.
CALMON, A. d. P.; PINHEIRO, N. C.; FERREIRA, R. U.	2006	Desenvolvimento de um robô-cachorro comportamental: percepção e modelagem comportamental
SOUTO, R. F.	2007	Modelagem cinemática de um robô quadrúpede e geração de seus movimentos usando filtragem estocástica.
BATISTA, G. F.; CARDOSO, I. F.	2007	Adequação de um sistema de locomoção de um robô quadrúpede para avaliação de algoritmos de aprendizagem
NOVAIS, N. A. d; TOSCANO, R. A.	2007	Estudo de locomoção de uma plataforma quadrúpede utilizando sensoriamento inercial e geração de padrões de movimento.
RAMOS, E.G.	2008	Desenvolvimento da plataforma quadrúpede geração de software e eletrônica
PAIVA, R. C.	2012	Osciladores neurais para comando de marcha de um robô quadrúpede e robô humanoide
SANTOS, J. H. d. S.	2016	Plataforma quadrúpede Uma nova estrutura para robô quadrúpede do LARA.
PORPHIRIO, C. F; SANTANA, P. H. M.	2017	Geração de marchas para a plataforma quadrúpede utilizando algoritmo genético
FLORIANO, B. R d. O	2017	Desenvolvimento e interação de um sistema de controle de equilíbrio para um robô quadrúpede

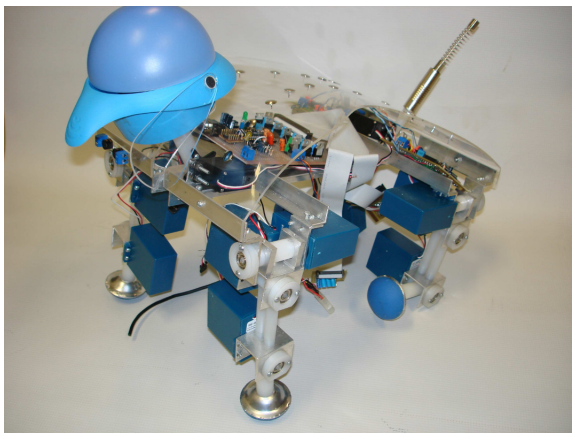


Figura 2.1: Primeira versão desenvolvida da plataforma quadrúpede do LARA. (Fonte [2])

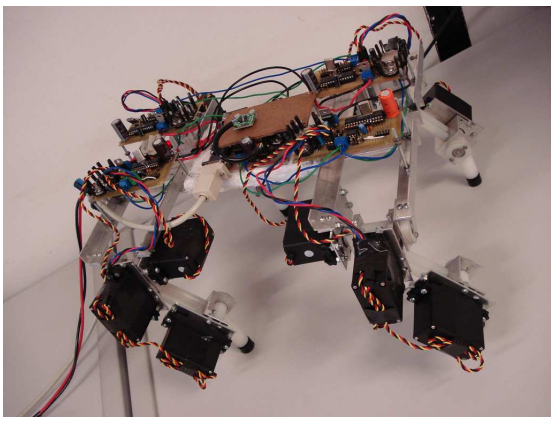


Figura 2.2: Plataforma quadrúpede utilizada no trabalho de Batista e Cardoso. (Fonte [3])

do movimento das juntas do robô. Nesse mesmo ano, Batista e Cardoso [3] alteraram os servos, introduziram acelerômetro e realizaram melhorias no protocolo de comunicação, permitindo uma melhor avaliação de algoritmos de aprendizagem. A plataforma, ao final desse trabalho, pode ser observada na Figura 2.2 Por ultimo, Novais e Toscano [11] adicionaram uma IMU (*inertial measurement unit*) e um conjunto de sensores para a detecção de contato nas patas. Essas melhorias foram implementadas visando a validação de marchas propostas utilizando autômatos de entradas/saídas temporizados.

No ano de 2008, Ramos [12] transcreveu os códigos, anteriormente implementados no MATLAB, para a linguagem C. Junto disso, foi realizada uma melhoria nas placas de acionamentos dos servos do robô.

Posteriormente, em 2012, Paiva [13] realizou um trabalho de geração de marchas bioinspiradas, utilizando osciladores neurais. O projeto não se restringiu apenas para o robô quadrúpede, a geração de marchas para robôs humanoides também foi seu objeto de estudo. Contudo, as marchas geradas não puderam ser testadas no solo, uma vez que o acréscimo de peso no robô foi suficiente para demandar um torque superior ao fornecido pelos servos.

O estudo posterior, realizado no ano de 2016 por Santos [4], foi caracterizado por uma reformulação completa na plataforma. Uma nova estrutura mecânica foi fabricada, fornecendo uma maior robustez e diminuindo o peso da plataforma. Os servos antigos foram substituídos por servos que oferecem mais torque. Além disso, o sistema embarcado foi implementado utilizando Raspberry Pi, junto com um novo sistema de leitura e controle dos servos. A Figura 2.3 mostra a plataforma no final do trabalho de Santos.

No ano de 2017, Porphirio e Santana [9] realizaram um estudo visando a geração de marchas utilizando algoritmo genético. Algoritmos foram desenvolvidos para permitir o controle do modelo robótico, presente no V-REP, pelo Matlab. Permitindo assim a execução do algoritmo genético em um ambiente de simulação. Para uma execução satisfatória das marchas geradas, algumas melhorias pontuais foram realizadas na plataforma. A primeira, foi a inserção de ponteiras de borracha nas extremidades das patas do robô, visando uma maior aderência com o solo. Foi adicionado também, um módulo de Arduino para realizar as medições do acelerômetro e dos novos sensores

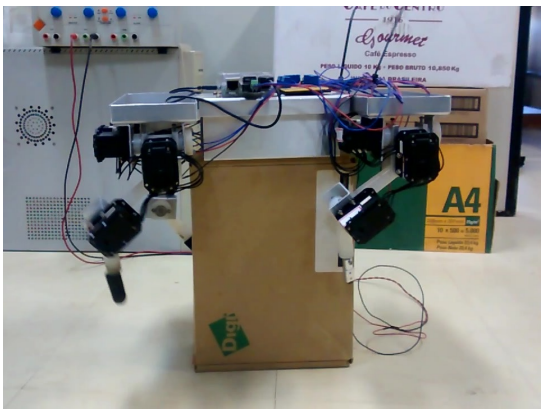


Figura 2.3: Plataforma quadrúpede ao fim do trabalho de Santos. (Fonte [4])

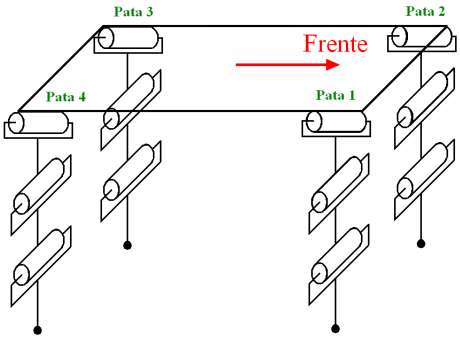


Figura 2.4: Convenção da numeração das patas do robô quadrupede (Fonte [5])

de força. Ainda nesse ano, Floriano [14] desenvolveu um trabalho focado no desenvolvimento de um sistema controle de equilíbrio da plataforma, resultando em uma arquitetura que permite que o dispositivo continue a execução de uma marcha, mesmo após um distúrbio.

2.2 Mecânica da Plataforma

A plataforma quadrúpede é um dispositivo robótico composto de quatro patas, cada uma possuindo três servos, totalizando em doze graus de liberdade. A Figura 2.4 representa a disposição dos servos na plataforma e a convenção de numeração das patas.

A estrutura mecânica atual da plataforma é muito próxima da implementada por Paiva [4], onde o robô possui uma estrutura de alumínio e patas de nylon. As diferenças se dão devido a algumas melhorias realizadas por Porphirio e Santana [9] como: fixação das conexões e adição de uma ponteira de borracha na extremidade das patas.

2.3 Cinemática Inversa

Quando se trata de manipuladores robóticos, desde um robô industrial ou o braço de um robô humanoide, é extremamente importante controlar a posição de sua extremidade (*end-effector*),

podendo ela ser uma ferramenta de usinagem ou um simples apoio, por exemplo. Uma maneira de se fazer isso é utilizando a cinemática direta, em que várias operações usando matrizes de rotação e deslocamento, dados os ângulos dos atuadores, indicam a posição no espaço, em relação ao eixo de referência, de sua extremidade. Porém, em certos casos sabe-se em que ponto do espaço se deseja colocar o ponto de interesse mas não se sabe os ângulos que os atuadores terão que girar ou a distância que terão que se deslocar para que isso acontece. Sendo assim, outra maneira de controlar a posição da extremidade é pela cinemática inversa, em que, dada a posição no espaço, em relação a um eixo coordenado de referência, determinam-se os ângulos e deslocamentos necessários dos atuadores para que se atinja tal posição. Nesse caso, é possível que o problema não apresente uma única solução ou ainda que não apresente nenhuma.

Segundo Buss [15], o sistema de um multicorpo rígido consiste em um conjunto de objetos rígidos, geralmente ligações unidas por articulações (ou juntas), sendo que estas podem ser divididas em revolucionárias e prismáticas. Sendo assim, a configuração das juntas são definidas por escalares, que podem ser ângulos (revolucionárias) ou deslocamentos (prismáticas). Dessa forma, seja $\theta_1, ..., \theta_n$ o conjunto de escalares que descrevem a configuração das articulação em um multicorpo rígido. Supondo que $s_1, ..., s_k$ são as extremidades (*end effectors*) desse multicorpo, escritas como um vetor coluna $\overrightarrow{s} = (s_1, ..., s_k)^T$. Cada extremidade é uma função dos ângulos das juntas. O multicorpo será controlado especificando posições desejadas $\overrightarrow{t} = (t_1, ..., t_k)^T$ para as extremidades, sendo assim $e_i = t_i - s_i$ a mudança de posição desejada da i-ésima extremidade. Com os ângulos das juntas escritos como o vetor coluna $\overrightarrow{\theta} = (\theta_1, ..., \theta_n)^T$ e $\overrightarrow{s} = \overrightarrow{s}(\overrightarrow{\theta})$, o problema de cinemática inversa é encontrar valores de θ_j's que satisfaçam

$$t_i = s_i(\overrightarrow{\theta}) \tag{2.1}$$

para todo i.

Como já foi comentado, esse método talvez não tenha uma única solução ou qualquer solução.

Então, pode-se utilizar métodos iterativos para aproximar uma boa solução. Um deles seria linearizar as funções s_i usando a matriz Jacobiana [15].

$$J(\overrightarrow{\theta}) = \left(\frac{\delta s_i}{\delta \theta_j}\right)_{ij}. \tag{2.2}$$

Assim, a mudança na extremidade causada pela mudança na junta pode ser estimado por

$$\Delta\vec{s} \approx J\Delta\overrightarrow{\theta} \tag{2.3}$$

A ideia é que $\Delta\overrightarrow{\theta}$ deve ser escolhido tal que $\Delta\vec{s}$ é aproximadamente igual a $\vec{e}$. Para isso, pode se utilizar os métodos do jacobiano transposto, da pseudoinversa e mínimos quadrados.

2.4 Sensoriamento

O sistema de sensoriamento da plataforma é dado com o objetivo de permitir obter informações acerca do atual estado do robô, bem como as condições nas quais ele esta submetido. O sistema de

sensoriamento do robô quadrúpede oferece medidas sobre aceleração, por meio de um acelerômetro e medidas das forças submetida nas extremidades das patas, utilizando um sensor resistivo de força.

2.4.1 Acelerômetro

O acelerômetro é um dispositivo de medição que mensura a sua aceleração própria, ou seja, a aceleração a qual ele está envolvido, em relação a um corpo em queda livre. Dessa forma, um dispositivo em repouso no planeta Terra, sem inclinações de arfagem ou rolagem, medirá uma aceleração positiva no eixo z, como pode ser observado na Figura 2.5. Essa aceleração possui o mesmo módulo e direção da aceleração da gravidade, entretanto seu sentido é invertido. O dispositivo mede exatamente a aceleração imposta ao corpo para que ele não entre em queda livre.

O acelerômetro utilizado na plataforma é o acelerômetro digital ADXL345. Seu funcionamento se dá por meio do efeito capacitivo, proporcionado por um sistema microeletrônico que consiste de uma massa móvel, acoplada a conjunto de superfícies de silício policristalino. Dessa forma, a aceleração movimenta a massa, gerando uma variação na capacitância, que é proporcional a essa aceleração. Esse funcionamento o caracteriza como um sensor MESM (*Micro Electro Mechanical Systems*). Esse acelerômetro mensura a aceleração em 3 eixos perpendiculares entre si, com uma resolução variável, que pode ser de ± 2g, ± 4g, ± 8g ou ± 16g.

Considerando um acelerômetro na Terra, que além da aceleração gravitacional, é submetido a uma aceleração a qualquer. A medida obtida pelo dispositivo, é dada por G_P

$$G_P = \begin{bmatrix} G_{Px} \\ G_{Py} \\ G_{Pz} \end{bmatrix} = R(g - a) \tag{2.4}$$

onde R é a matriz de rotação que descreve a rotação entre o sistema de coordenadas do acelerômetro e o sistema de coordenadas de referência. Considerando que a orientação inicial de referência é igual a demonstrada na Figura 2.5 e que não há aceleração linear $a \approx 0$, a medida é

$$G_P = R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2.5}$$

A matriz de rotação se dá a partir de uma sequencia ordenada de rotações em torno dos eixos. Pode-se transformar as componentes de rotação, de cada eixo, para que elas estejam em relação aos ângulos de atitude do acelerômetro: arfagem(α), guinada(ψ) e rolagem(ϕ). Assumindo os seguintes valores:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sen(\phi) \\ 0 & -sen(\phi) & cos(\phi) \end{bmatrix} \tag{2.6}$$

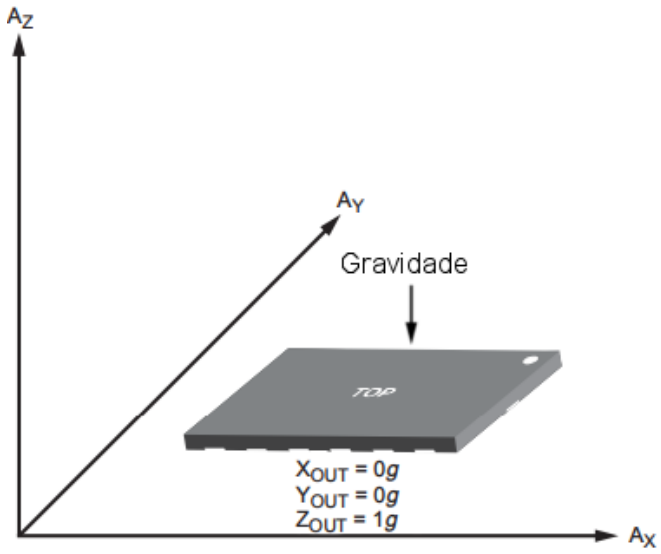


Figura 2.5: Resposta do acelerômetro sobre uma superfície de apoio. (Adaptado de [6])

$$R_y(\alpha) = \begin{bmatrix} cos(\alpha) & 0 & -sen(\alpha) \\ 0 & 1 & 0 \\ sen(\alpha) & 0 & cos(\alpha) \end{bmatrix} \tag{2.7}$$

$$R_z(\psi) = \begin{bmatrix} cos(\psi) & sen(\psi) & 0 \\ -sen(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.8}$$

A criação da matriz de rotação R se dá a partir do produto das matrizes $R_x(\phi)$, $R_y(\alpha)$ e $R_z(\psi)$. Realizando os cálculos de todas as 6 possíveis variações de sequencia de movimentos, Pedley [16] ressaltou que para cada ordem de movimentos, existia uma matriz de rotação, sendo todas igualmente válidas. Entretanto, 4 dessas matrizes não poderiam ser utilizadas, uma vez que elas são dependentes dos três ângulos de atitude. Apesar do acelerômetro possuir medidas nos três eixos, o que em um primeiro momento permitiria a solução para esses sistemas, nota-se que uma dessas medidas está sempre associada a aceleração da gravidade, fazendo o acelerômetro possuir apenas dois graus de liberdade. Impossibilitando a solução para os três ângulos de atitude. Portanto definindo a sequência de rotação como: x, y, z. Que resulta na seguinte matriz de rotação:

$$R_{xyz} = \begin{bmatrix} cos(\alpha)cos(\psi) & cos(\alpha)sen(\psi) & -sen(\alpha) \\ cos(\psi)sen(\alpha)sen(\phi) - cos(\phi)sen(\psi) & cos(\phi)cos(\psi) + sen(\alpha)sen(\phi)sen(\psi) & cos(\alpha)sen(\psi) \\ cos(\phi)cos(\psi)sen(\alpha) + sen(\phi)sen(\psi) & cos(\phi)sen(\alpha)sen(\psi) - cos(\psi)sen(\phi) & cos(\alpha)cos(\phi) \end{bmatrix}. \tag{2.9}$$

Substituindo a equação (2.9) em (2.5):

$$G_P = R_{xyz} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.10}$$

$$G_P = \begin{bmatrix} -sen(\alpha) \\ cos(\alpha)sen(\phi) \\ cos(\alpha)cos(\phi) \end{bmatrix}. \tag{2.11}$$

A equação (2.11) permite relacionar as medidas do acelerômetro aos ângulos de rolagem e arfagem. Esses ângulos são suficientes para avaliar o desempenho das marchas realizadas na plataforma.

A partir da normalização da equação (2.11),

$$\frac{G_P}{\|G_P\|} = \frac{1}{\sqrt{{G_x}^2 + {G_y}^2} + {G_z}^2} \begin{bmatrix} G_{Px} \\ G_{Py} \\ G_{Pz} \end{bmatrix}, \tag{2.12}$$

se obtêm duas equações para a obtenção dos ângulos de rolagem e arfagem, respectivamente:

$$\phi = arctan(\frac{-G_y}{G_z}), \tag{2.13}$$

$$\alpha = arctan(\frac{G_x}{\sqrt{{G_y}^2 + {G_z}^2}}). \tag{2.14}$$

2.4.2 Sensor de Força

A estimação da força aplicada nas patas do robô se faz necessária por dois motivos principais: a avaliação da carga que cada ponto de apoio do robô está submetido e análise do instante de contato entre a pata e o solo. Com o intuito de cumprir essas especificações, a plataforma dispõe de quatro sensores resistivos de força do tipo FSR 400, um em cada extremidade da pata. Segundo o manual do fabricante [17], esse sensor possui uma resolução de 0,2N e uma faixa de atuação entre 0,2 e 20N.

2.5 Sistemas Embarcados

Segundo Porphirio e Santana [9], o quadrúpede é controlado por uma integração entre Raspberry PI e o arduino, em uma relação mestre-escravo. A função do arduino é coletar dados dos sensores de força e do acelerômetro e enviá-los ao Raspberry PI, via serial, para que o microcomputador, por exemplo, tome decisões de controle como em [14]. o Raspberry PI, por sua vez é responsável por controlar os servo motores a partir de uma biblioteca da Dynamixel, em C++.

Sendo assim, programas em C++ recebem como parâmetros as marchas desenvolvidas, que são sequências de ângulos que deverão ser executadas em cada um dos servo motores e lidam com a leitura dos dados provenientes do Arduino Uno.

2.5.1 Arduino Uno

O arduino é uma placa com um chip microcontrolador, e para que seja possível acessar as suas entradas existem diversos pinos que são designados para as mais diversas tarefas. Além disso ele conta com entradas USB, entrada de tensão DC, reguladores de tensão de 5V e 3,3V, LEDs de status, um controlador ATMega328, etc.

Ele conta com 14 pinos digitais, 6 pinos analógicos, 2 pinos Serial (Tx e Rx), 1 pino de *reset*, 1 pino de entrada de 3,3V, 1 pino de entrada de 5V, 3 pinos terra, sendo um digital entre outros. A maior parte da interação da placa com o desenvolvedor é dada através desses pinos.

É uma placa com fim didático, simples de usar mas não deixa de ser flexível o suficiente para projetos com um certo nível de complexidade. Possui um ambiente de desenvolvimento integrado (IDE) próprio que já possui várias bibliotecas, como por exemplo a de adquirir dados de uma sensor de ultrassônico.

2.5.2 RaspberryPI

Raspberry PI é um microcomputador, e devido às suas pequenas dimensões é uma alternativa para se embarcar em sistemas móveis, como um robô por exemplo. Seu sistema operacional é o Raspbian, que é baseado em Debian, mas modificado para otimizar a performance do Raspberry PI.

A placa possui entradas USB, Ethernet e HDMI, espaço para um cartão Micro SD, pinos de uso geral (GPIO) e é alimentado por uma entrada micro USB. Também conta com um processador ARM Cortex-A53 de 1,2Ghz com quatro núcleos e 1GB e memória RAM LPDDR2 de 900Mhz (especificações do Raspberry PI 3).

2.6 Algoritmo Genético

Algoritmo genético (GA), um tipo de algoritmo evolucionário (EO), é um método para resolver problemas de otimização baseado em seleção natural. Inspirado no processo evolucionário biológico, o algoritmo modifica repetidamente uma população de soluções individuais ao longo de várias gerações, seja por reprodução ou mutação, à procura da solução ótima do problema, que terá sua direção, no espaço de soluções, apontada pelo indivíduo que obter a melhor pontuação, de acordo com a função objetivo. É importante se ater à algumas definições de cada componente do algoritmo em questão:

- Função objetivo: É a função que se deseja otimizar.

- Indivíduos: Um indivíduo é qualquer ponto no espaço de soluções que pode ser aplicado na função objetivo, o que gera um valor resultante, chamado comumente de pontuação do indivíduo.
- Populações e Gerações: A população é um vetor de indivíduos. Ela pode ser modificada através da reprodução e mutação dos indivíduos, e toda vez que uma nova população é gerada com sucesso, é dito que se criou uma nova geração.
- Diversidade: É a distância média entre os indivíduos de uma população. Por exemplo, é extremamente desejável se ter uma grande diversidade em alguma etapa da otimização a fim de se procurar o ótimo global em uma região maior do espaço de soluções.
- Pais e filhos: Para criar novas gerações o algoritmo seleciona pais para criar novos indivíduos (Filhos). Geralmente, escolhe-se pais que têm melhores pontuações.

Sendo assim, o algoritmo genético começa gerando uma população inicial aleatória. Pode-se definir essa população em um intervalo inicial, no qual se acredita estar a solução ótima do problema. Dessa forma, a cada etapa do algoritmo, usa-se a população atual para criar os filhos que irão compor a nova geração. Para tanto ele seleciona pais que deverão compartilhar seus genes (entradas dos seus vetores). Segundo Deb [7], um dos meios de gerar os filhos é a reprodução, em que uma probabilidade ($p \in [0, 1]$) indica a proporção de indivíduos da população que se reproduzirão. O resto da população ($1 - p_c$) é apenas copiada para próxima geração. Também é dito que cada filho é perturbado por uma mutação de probabilidade p_m, geralmente tida como $1/n$, em que n é o número de genes, garantindo que pelo menos um gene, em média, seja mutado por solução. A mutação permite que o EO procure em um espaço maior, independente da localização de outra soluções na população. Um operador elitista combina populações antigas com gerações atuais, e escolher manter as melhores soluções da população combinada. Finalmente, o desenvolvedor do EO determina critérios de parada, tais como número máximo de gerações, tempo limite, limite do objetivo, distância média entre os indivíduos.

2.6.1 Otimização multiobjetiva evolucionária

Otimização multiobjetiva consiste em otimizar vários objetivos simultaneamente. Porém, existe a possibilidade de os objetivos conflitarem entre si no sentido que otimizar um pode causar uma piora em outro, causando uma relação de troca. Resolvendo tais problemas, é gerado um conjunto de soluções ótimas conhecida como a frente de Pareto. segundo Deb [7], dada a multiplicidade das soluções, esses problemas foram propostos a serem resolvidos sucintamente utilizando algoritmos evolucionários. Como uma otimização com um único objetivo, a multiobjetivo contém uma série de constrições os quais qualquer solução, incluindo as ótimas, devem satisfazer, sendo assim, a forma geral do problema da otimização multiobjetiva é dada por

$$\left.\begin{aligned} \text{Minimize/Maximize } f_m(x),\ m = 1, 2...M; \\ \text{Sujeito a } g_j(x) \geq 0,\ j = 1, 2, ...J; \\ h_k(x) = 0,\ k = 1, 2, ...K; \\ x_i^{(L)} \leq x_i \leq x_i^{(U)},\ i = 1, 2, ...n. \end{aligned}\right\} \tag{2.15}$$

As quais $g_(x)$, $h_k(x)$ e $x_i^{(L)} \leq x_i \leq x_i^{(U)}$ são constrições.

Uma solução $\overrightarrow{x} \in R^n$ é um vetor de n variáveis de decisão $x = (x_1, x_2, ..., x_n)^T$, e as soluções que satisfazem as limitações e as fronteiras constituem um espaço de variáveis de decisão factível $S \subset R^n$. É possível notar que uma das maiores diferenças entre otimizações multiobjetivas e com um único objetivo, é que nas primeiras as funções objetivos constituem um espaço multi-dimensional, chamado de espaço de objetivoss $Z \subset R^M$. Sendo assim, para cada solução x no espaço de variável, existe um ponto $z \in R^M$ no espaço de objetivos, tido como $f(x) = z = (z_1, z_2, ..., z_M)^T$.

As soluções ótimas, nesse caso, podem ser definidas pelo conceito matemático de ordenação parcial, e assim, comumente se utiliza o termo dominação entre duas soluções, que pode ser definido a seguir:

Definição 1 *Uma solução $x^{(1)}$ é dita dominante sobre uma solução $x^{(2)}$ se ambas as seguintes condições são verdadeiras*

1. *A solução $x^{(1)}$ não é pior que $x^{(2)}$ em nenhum objetivo, sendo que as soluções são comparadas baseadas nas suas pontuações dada a função objetivo (ou localização dos pontos $z^{(1)}$ e $z^{(2)}$ correspondentes no espaço de objetivos).*
2. *A solução $x^{(1)}$ é melhor que $x^{(2)}$ em pelo menos um objetivo.*

Um conjunto de pontos não dominados é um dos objetos de interesse do algoritmo, uma vez que um ganho de um ponto para o outro acontece apenas sacrificando pelo menos um objetivo, compondo uma frente no espaço. Sendo assim, é interessante encontrar vários deles antes de se fazer a escolha final da solução ótima. Agora, pode-se definir as solução Pareto-ótimas em um problema de otimização multiobjetiva. Se um conjunto de pontos contidos no espaço de busca, residem na frente de pontos não dominados, então esses são pontos Pareto-ótimo.

Sendo assim, o papel do EO é procurar os pontos que fazem parte da frente Pareto-ótima, utilizando operadores que estimulam a evolução de soluções não dominadas. Uma vez que ele lida com uma população de soluções, ele ainda deve encontrar pontos diversos o suficientes para representar o alcance inteiro da frente Pareto-ótima. Dessa forma, tem-se um conjunto de soluções ótimas em que uma difere da outra sendo pior em um objetivo e melhor em outro, em uma relação de perde e ganha, o que causa um dilema na escolha da solução. Essa escolha é dada a partir da análise de informações de alto nível, que geralmente são não técnicas, qualitativas e baseadas em experiências.

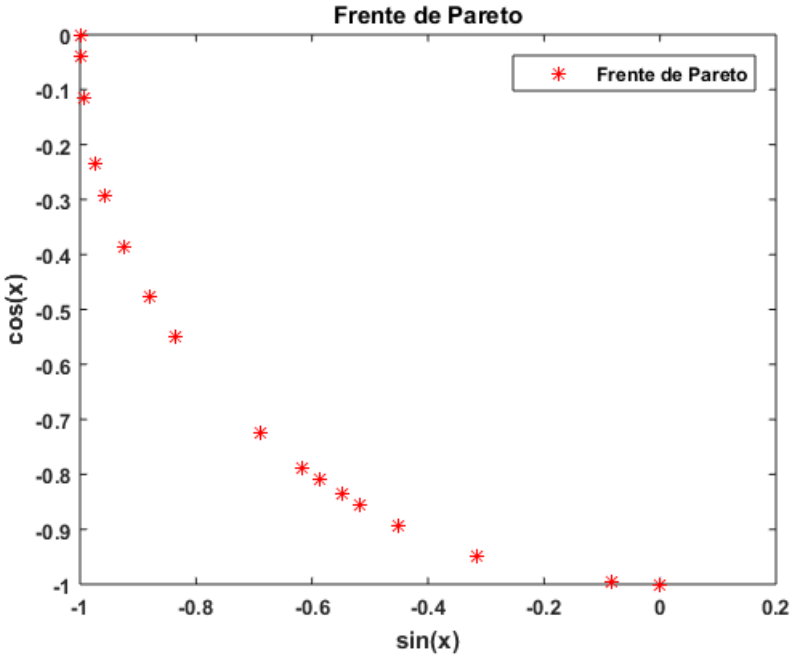


Figura 2.6: Frente Pareto-ótima encontrada por um EMO.

Uma vantagem da otimização multiobjetiva evolucionária (EMO) sobre os outros tipos é que, sabendo que o algoritmo deve muitas vezes lidar com regiões não factíveis, regiões achatadas, soluções ótimas locais para convergir para a solução ótima global, no segundo caso a busca pela frente Pareto-ótima ocorre em diversas simulações, analisando ponto a ponto, e entre uma simulação e outra, segundo Deb [7] nenhuma informação sobre o progresso é usada para acelerar o processo, demandando um grande custo computacional para obter o conjunto Pareto-ótimo. Em contrapartida uma EMO constitui uma busca paralela, sendo que uma vez que um membro de uma população supera algumas dificuldades, sua pontuação e sua combinação de genes refletem esse fato, e essa informação é considerada durante as operações de reprodução, seleção, etc.

Para fins de exemplo, suponha que se deseja utilizar um EMO para encontrar a frente Pareto-ótimo de duas funções objetivo $\sin(x)$ e $\cos(x)$ no intervalo $0 \leq x \leq 2\pi$. Dessa forma, se obtém a correspondente frente de Pareto, ilustrada pela figura 2.6.

2.6.2 AG de ordenação de não dominados elitista - NSGA-II

Segundo Deb [7] O NSGA-II é um procedimento muito popular que tenta encontrar múltiplas soluções Pareto-ótimas em um problema de otimização multiobjetiva e possui as três características

1. Usa um princípio elitista,
2. usa um mecanismo de preservação de diversidade explícito, e
3. enfatiza soluções não dominadas.

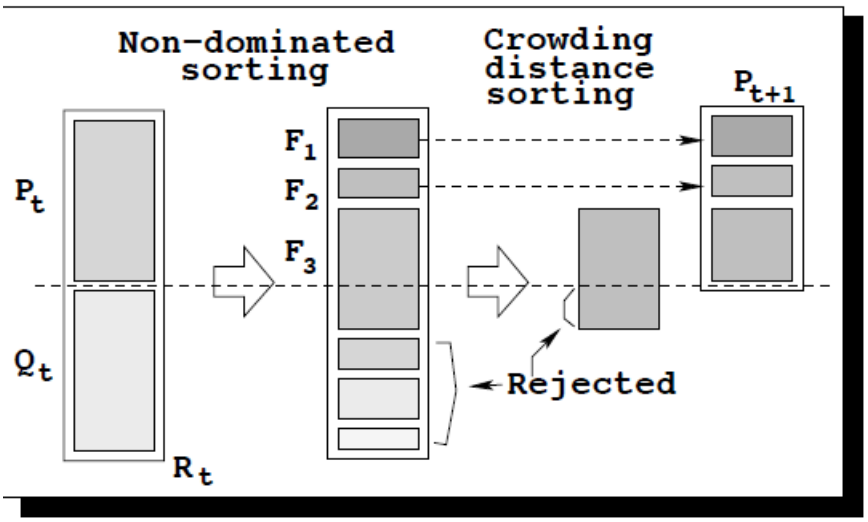


Figura 2.7: Esquema do funcionamento do NSGA-II. P_t é a população anterior. Q_t é a população nova. R_t é a combinação das duas. F_1, F_2 e F_3 são as frentes a serem alocadas em P_{t+1}. (Fonte: [7].)

Em uma geração t, uma população nova Q_t é inicialmente criada utilizando a população anterior P_t e os operadores genéticos usuais (reprodução, mutação e seleção). Em seguida as duas populações são combinadas para criar um nova população R_t de tamanho $2N$ (N é o número de indivíduos nas populações). Depois a população de R_t é classificada em diferentes classes não dominadas, ordenadamente, formando uma sequência de frentes. Sendo assim, um nova população P_{t+1} é formada preenchendo seus N espaços, começando com pontos da primeira frente não dominada, depois da segunda, e por assim por diante. Como o espaço é limitado e R_t tem tamanho $2N$, as frentes que não forem acomodadas em P_{t+1} são descartadas. E quando a última frente está sendo considerada, talvez possa existir mais pontos nela do que no espaço que está sobrando na nova população e, nesse caso, em vez de apenas descartar os outros pontos, os indivíduos que mais contribuírem para para a diversidade da população são escolhidos. A figura 2.7 explicita o que foi descrito.

A ordenação dos pontos da frente que não conseguiu ser completamente alocada na nova população é dada pelo valor da distância da multidão, de forma descendente, e os pontos do topo da lista são escolhidos. A distância da multidão d_i do ponto i é uma medida do espaço de objetivo em volta de i o qual não está ocupado por outra solução da população. Pode ser calculando utilizando o perímetro de um cuboide, por exemplo.

A situação muda um pouco ao adicionar constrições ao EMO, pois nesse caso pode-se ter soluções que são factíveis e não factíveis, sendo assim agora é necessário redefinir o princípio de dominação, considerando duas soluções $x^{(i)}$ e $x^{(j)}$.

Definição 2 *Uma solução $x^{(i)}$ é dita dominar uma solução $x^{(j)}$, dada as restrições, se qualquer uma das seguintes condições são verdadeiras*

1. *A solução $x^{(i)}$ é factível e $x^{(j)}$ não.*

2. *As soluções $x^{(i)}$ e $x^{(j)}$ são ambas não factíveis mas $x^{(i)}$ possui uma limitação menor da constrição, que pode ser computada adicionando a violação normalizada de todas as constrições:*

$$CV(x) = \sum_{j=1}^{J} \langle \overline{g}_j(x) \rangle + \sum_{k=1}^{K} abs(\overline{(h)}_k(x)) \tag{2.16}$$

em que $\langle \alpha \rangle$ é $-\alpha$, se $\alpha \leq 0$. A normalização é adquirida com a constrição das populações e mínima e máxima violadas.

3. *As soluções $x^{(i)}$ e $x^{(j)}$ são factíveis mas $x^{(i)}$ domina $x^{(j)}$ do jeito usual. Definição 1.*

2.6.3 Ferramenta de otimização

A ferramenta de otimização utilizada neste trabalho é um aplicativo do Matlab, o *gamultiobj*, que implementa um algoritmo genético multiobjetivo. Ele usa uma algoritmo genético elitista e controlado, uma variante do NSGA-II. Um AG controlado favorece indivíduos que ajudam a aumentar a diversidade da população, mesmo tendo uma pontuação reduzida.

Para indivíduos factíveis existe uma definição iterativa do *rank* de um indivíduo. Um elemento do *rank* 1 não é dominado por elementos de nenhum outro rank. Elementos do rank 2 são dominados apenas por elementos do *rank* 1. Sendo assim, pode-se dizer que indivíduos do *rank* k são dominados por indivíduos de *rank* $k-1$ ou menores. Sendo assim, quanto menor o *rank* maior a chance de ser selecionado. Todos os pontos não factíveis tem *ranks* menores que os factíveis, e entre si, o *rank* de cada um é dado pela ordenação da medida de infactibilidade, adicionado do maior *rank* entre os factíveis.

A distância da multidão (distância de um indivíduo ao seus vizinhos mais próximos) é medida entre indivíduos do mesmo *rank*, e é dada pela soma sobre as dimensões das distâncias absolutas normalizadas entre os vizinhos ordenados do indivíduo. Ou seja, para a dimensão m e indivíduo i:

$$d_i = \sum_{m} x(m, i+1) - x(m, i-1) \tag{2.17}$$

Vale dizer que o algoritmo ordena cada dimensão separadamente, então os vizinhos são necessariamente os da mesma dimensão em questão. E também, indivíduos com d_i maior tem chances maiores de seleção, a fim de encorajar a diversidade, sendo assim, esse também é um fator de desempate durante o torneio (método de seleção que escolhe o indivíduo de maior *rank*).

Assim, o primeiro do passo do algoritmo é criar uma população inicial. O usuário pode definir os primeiros indivíduos ou deixar isso a cargo do *gamultiobj*, que por padrão, cria uma população factível com respeito aos limites e às constrições lineares (não necessariamente respeitando constrições não lineares). E assim ele avalia essa população e as subsequentes nas funções objetivo, atribuindo pontuações aos indivíduos para encontrar as frente Pareto-ótima.

As iterações se dão de acordo com os seguintes passos:

1. Selecione pais para as próximas gerações utilizando o critério de seleção. O único critério nativo do algoritmo em questão é o torneio binário.
2. Crie filhos dos pais selecionados utilizando reprodução e mutação.
3. Pontue os filhos calculando seus valores nas funções objetivas e analisando sua factibilidade.
4. Combine a população atual com os filhos, criando uma população expandida.
5. Compute o *rank* e a distância de multidão para todos os indivíduos da população expandida.
6. Trunque a população para não ultrapassar o valor máximo do tamanho da população ao reter um número apropriado de indivíduos de cada *rank*.

A aplicação possui várias opções de configuração do algoritmo genético que podem ser determinadas pelo usuário de acordo com sua demanda, como

- **Fitness Function**: É onde se especifíca a função-objetivo, ou a função a ser minimizada. É requerido um número de variáveis independentes para serem avaliadas na função objetivo.
- **Constraints**: É onde se especificam as restrições do problemas, como inequações lineares $Ax \leq b$, igualdades lineares $A_{eq}x = b_{eq}$, limitações superiores e inferiores e uma função de restrição não-linear criada opcionalmente pelo usuário.
- **Population**: É onde se específicam opções para a população do algoritmo genético, como: Tipo da população (vetor de *double*, *string*, ou personalizada), tamanho da população, população inicial, pontuação da população inicial e intervalo inicial dos indivíduos da população. Pode-se também criar uma função parar gerar a população inicial.
- **Selection**: Onde se específica como a seleção dos indivíduos será feita. Nesse caso, apenas as opções torneio ou uma função desenvolvida pelo usuário estão disponíveis.
- **Reproduction**: Determina como o algoritmo genético gera um filho a cada geração. Pode-se determinar a fração da população que a reprodução cria, o restante é resultado da mutação.
- **Mutation**: Determina como será feita a mutação nos indivíduos da população. Pode-se escolher entre mutação dependente das constrições, gaussiana, uniforme, factível à adaptação e pode-se criar uma função que lida para lidar com isso.
- **Crossover**: É onde se específica a função que executa a forma de reprodução, ou seja, como os pais irão compartilhar seus genes para gerar um filho, podendo ser disperso, dependente das restrições, ponto único, dois pontos, intermediário, heurístico, aritmético e personalizado. A diferença delas
- **Migration**: É onde se determina a forma como os indivíduos vão se mover entre subpopulações. É muito comum que os melhores indivíduos de uma subpopulação substituam os piores em outra subpopulação. Pode-se controlar a migração por três parâmetros: Direção dos indivíduos, fração da subpopulação que migra e intervalo de quantas gerações passam entre as migrações.

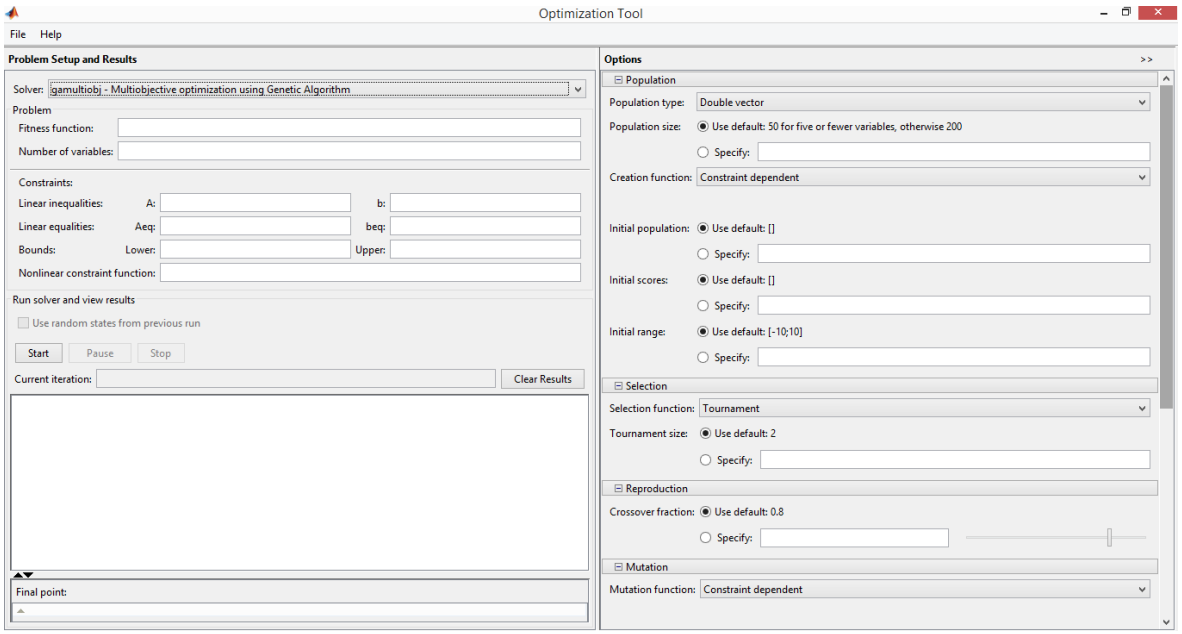


Figura 2.8: Janela da aplicação de otimização multiobjetiva *gamultiobj*.

A figura 2.8 ilustra a janela da aplicação, e nela é possível ver algumas das opções supracitadas, os campos das constrições, o campo da função objetiva, variáveis, etc.

2.7 Software de Simulação

A interação de um robô com um ambiente é definida por inúmeras variáveis, diversas condições são impostas ao dispositivo durante seu funcionamento. Torna-se difícil prever todas as possibilidades e variações possíveis na sua operação. É muito comum por exemplo, que alguns estados que não foram previstos durante a idealização de um sistema, sejam alcançados. De outro modo também é possível que estados que devem ser alcançados, não sejam atingidos. Nesses dois casos há uma discrepância entre o que se deseja que o dispositivo faça, e o que ele realmente faz.

Visando minimizar essa discrepância, é recomendável que se faça a utilização de simulações. Segundo Kelton, Sadowski e Sadowski [18], simulação diz respeito a uma grande coleção de métodos e aplicações, em geral computacionais, que replicam o comportamento de um sistema real. O principal intuito de uma simulação é validar o comportamento do dispositivo robótico, por meio de uma modelagem simplificada do mundo, fornecendo, de certa forma, testes que possam ser realizados sem prejudicar a plataforma real e os agentes envolvidos na sua operação.

Diversos programas foram projetados com o intuito de facilitar simulações robóticas, dentre eles pode-se citar: Open HRP, Gazebo, Webots e V-REP. Segundo Rohmer, Singh e Freese [8] mesmo todos competindo em quesitos de funcionalidade, o V-REP se destaca por oferecer uma grande gama de técnicas de simulação e pela portabilidade de seus modelos e controladores.

A *Virtual Robot Experimentation Plataform*, ou simplesmente V-REP, é uma plataforma de simulação computacional que teve seu lançamento público realizado em março de 2010. A arquite-

tura do V-REP foi proposta visando permitir ao usuário, a escolha do modo de execução do código de controle da simulação. Existem três formas distintas de execução do código de controle [8]:

- **Remoto**: Nesta arquitetura, o código de controle é executado em uma máquina distinta da qual está sendo realizada a simulação. Elas estão conectadas por meio de uma rede específica, seja ela via porta serial, *socket*, etc. A grande vantagem dessa implementação é a possibilidade de testar o controle realizado por um dispositivo externo, em geral tratando-se até mesmo do próprio controlador do sistema real. Além disso, tem-se uma diminuição, em relação aos outros métodos, na carga computacional no dispositivo em que se está simulando o processo. Entretanto essa arquitetura gera uma grande desvantagem, por exigir uma comunicação entre dois dispositivos, inerentemente tem-se um atraso na transmissão dos dados. Gerando uma dificuldade na sincronização com o laço da simulação.

- **Paralelo:** Nesta arquitetura, o código de controle é executado na mesma máquina do laço da simulação, porém em processos ou *threads* diferentes. Esta implementação permite um melhor balanceamento dos núcleos do processador, apesar de exigir mais carga computacional em relação a arquitetura remota. Por se tratar de uma execução paralelizada dos processo, ainda existe uma certa assincronia com o laço da simulação, menor que a configuração remota mas que deve ser considerada.

- **Serial:** Nesta arquitetura, o codigo de controle é executado na mesma maquina e na mesma *thread* do laço de simulação. A sincronia com o laço de simulação, está intrinsecamente associada a esse método. Entretanto, isso só é possível em detrimento de um aumento da carga computacional no núcleo de execução da simulação.

Buscando dar suporte para as três possíveis implementações, a arquitetura do V-REP foi definida de acordo com a Figura 2.9. No sistema implementado não existe uma funcionalidade principal, o V-REP oferece uma grande gama de formas de programação para que o usuário escolha a que lhe atenda as suas necessidades. As técnicas de programação permitidas no programa são:

- **Códigos embutidos:** São códigos em LUA, que são implementados junto com a modelagem do ambiente de simulação. Existe uma certa hierarquia entre esses códigos, uma vez que existe um código principal e seus "códigos filhos". O código principal, diz respeito as funcionalidades gerais da simulação, enquanto os "códigos filhos"são acoplados a objetos em uma cena. Por exemplo, caso seja necessário adicionar um ruido a uma medida do sensor, deve se inserir essas informações no "código filho"associado ao sensor. A execução dos códigos podem ser paralelizadas em *threads* diferentes ou não, o critério é do usuário. Essa forma de controle permite uma portabilidade e uma reutilização do sistema. É possível reutilizar objetos, junto com seus códigos, em outros projetos, além de permitir a execução desses códigos em diversas plataformas, uma vez que não é necessária a compilação explicita, basta que o V-REP esteja sendo executado.

- ***Add-ons:*** São códigos, também executados em LUA, que são executados no início do programa. Podem ser utilizados como funções, para adicionar funcionalidades ao programa, que

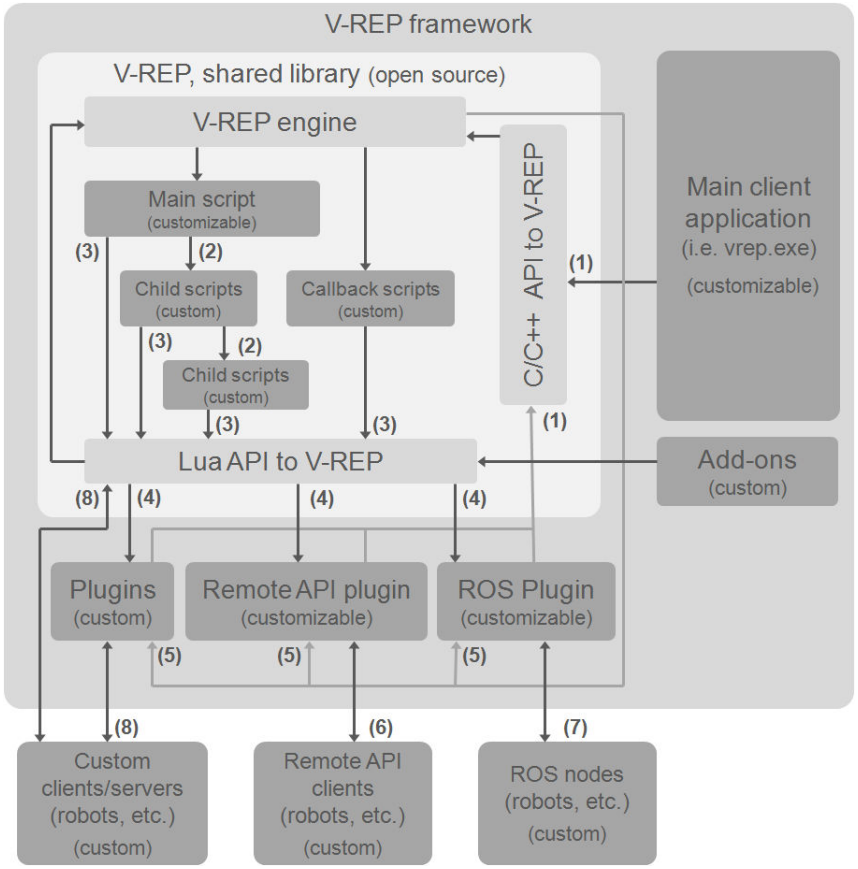


Figura 2.9: Arquitetura de controle do V-REP.(1) chamadas para a API em C/C++, (2) execução dos "códigos filhos"em cascata, (3) chamadas para a API em LUA, (4) *callbacks* personalizados para API em LUA, (5) *callbacks* de eventos do V-REP, (6) chamadas de função da API remota, (7) fluxo de dados em ROS, (8) comunicação personalizada (*socket*, serial, *pipes*, etc. (Fonte [8])

serão executadas por uma ação do usuário, bem como podem ser utilizados como códigos que são executados em um laço, durante a execução do programa.

- ***Plugins:*** Funcionam de forma semelhante aos *Add-ons*, tendo em vista que são códigos implementados com o intuito de customizar o V-REP. Os plugins são implementados em C/C++, entretanto é possível a execução de comandos em LUA, por meio de *callbacks*.

- **Clientes da API remota:** Existe uma API remota que permite a interação com o V-REP, ou com uma simulação, utilizando uma communicação via *socket*. Trata-se de uma comunicação do tipo cliente-servidor, onde o servidor é o simulador e o cliente é um programa que executa chamadas para o servidor. A API remota do V-REP dá suporte para C/C++, Python, Java, Matlab e Urbi. As chamadas para o servidor possuem dois modos de operação basicos: bloqueante ou não-bloqueante. A primeira espera a resposta do servidor para que possa executar a próxima chamada, enquanto a segunda envia as chamadas em uma transmissão, que vão sendo armazenadas em um *buffer*, até sua execução. A principal vantagem da utilização da arquitetura via API remota, é a possibilidade de interagir com o simulador utilizando diversas linguagens.

- **Nós de ROS:** Essa arquitetura consiste da implementação de *plugins* no V-REP que permitam a utilização do mesmo, utilizando o *Robot Operating System* (ROS)

2.8 Marchas Factíveis

Segundo Belter e Skrzypczyzynski [19] inspirações biológicas são comuns no desenvolvimento de padrões de marchas de robôs que caminham e suas estruturas de controle. Porém, robôs diferem significativamente dessas inspirações. Então acredita-se que o aprendizado de padrões de marchas deveria ser feito utilizando o próprio robô ou seu modelo simulado, e não apenas copiando o comportamento de um inseto, por exemplo. Mas quando se trata do aprendizado desses padrões que seriam factíveis para sistema, a solução se encontra em um espaço de busca grande e complicado, dessa forma, o que se pode fazer é, progressivamente, restringir o espaço de busca, sendo que essa restrição se dá por acrescentado a cada simulação elementos biologicamente inspirados, como coordenação de pernas, poses das patas, defasagem, etc.

Mesmo considerando esses aspectos, e desenvolvendo uma marcha factível, a simulação não corresponderá integralmente ao comportamento real do robô, devido o vão da realidade (*reality gap*). Ainda segundo Belter e Skrzypczyzynski [19] esse problema pode ser resolvido usando a otimização evolucionária de parâmetros do modelo simulado, fazendo que com o robô da simulação se comportasse o mais parecido possível com o da realidade. Dessa forma, a ideia da otimização seria encontrar um conjunto de parâmetros que juntos minimizam as discrepâncias entre as trajetórias de alguns pontos característicos do robô. Porém, a escolha desses pontos não é trivial, porque os parâmetros que devem ser encontrados se manifestam proeminente em diferentes tipos de movimentos, que são difíceis de rastrear em um robô físico. Pode-se propor alguns experimentos, que consistem essencialmente em um conjunto de trajetórias, em que o resultado de cada

um deles depende apenas de alguns parâmetros do modelo. Porém, se o número de trajetórias para os experimentos for pequena, o robô físico irá funcionar apropriadamente apenas para essas trajetórias.

Em Zagal, Solar e Vallejos [20] propõe-se que, para não haver mais diferenças entre a simulação e o robô físico, o robô e o simulador co-evoluem integrados. Ou seja, o robô aprende alternando experiências reais e virtuais, inspirado no fato que alguns organismos não conseguem distinguir entre realidade e ilusão (simulação). Dessa forma, o simulador é adaptado dada interações do robô com o ambiente, minimizando as diferenças do comportamento objetivo entre a realidade e a simulação.

A partir da fundamentação, é possível realizar a definição de metodologias para o desenvolvimento das marchas. Essas definições serão feitas no capítulo 3, que engloba desde as integrações entre as diversas plataformas computacionais até a proposição e implementação dos algoritmos para a otimização das marchas.

Capítulo 3

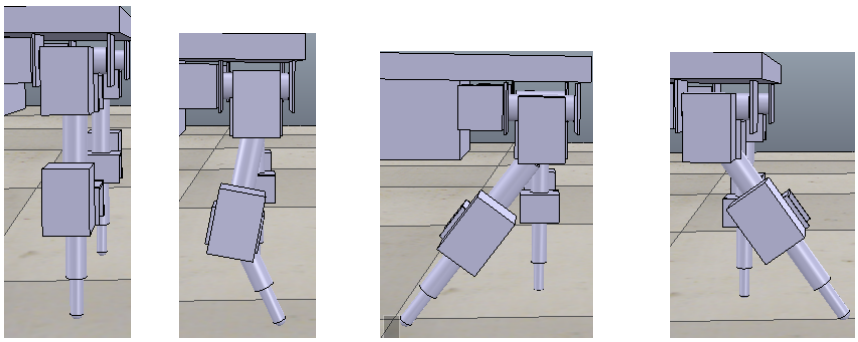
Desenvolvimento

Neste capítulo trata-se sobre as metodologias aplicadas para o desenvolvimento das marchas. Primeiramente se realizou a integração entre o Matlab e o V-REP, utilizando uma API remota. A partir disso, pôde-se iniciar as otimizações propostas para realizar a busca das marchas que compõem a frente Pareto ótima, utilizando o simulador. Para se realizar a otimização na plataforma física, fizeram-se algumas alterações no código de Porphirio e Santana [9] e criou-se um programa em python para controlar o robô remotamente.

3.1 Busca e otimização

Assim como em Belter e Skrzypczyzynski [19] deseja-se encontrar marchas factíveis para o robô quadrúpede. E para tanto usou-se do método de, progressivamente, reduzir o espaço de busca do otimizador. Em uma primeira simulação, o algoritmo genético multiobjetivo deveria encontrar uma matriz 8x12, em que cada coluna representava o ângulo em cada uma das 12 juntas, e as linhas, a evolução desses ângulos no tempo. Ou seja, esperava-se que o otimizador percorresse o espaço de soluções, sem restrição alguma, a fim de encontrar uma solução que fosse factível para o robô.

Em seguida, utilizando cinemática inversa, modelou-se quatro poses para as patas do quadrúpede, inspiradas em caminhadas de cavalos, que podem ser visualizadas na figura 3.1. Agora a tarefa do otimizador era encontrar uma ordem dessas poses no tempo, para cada uma das patas que, fizesse com que ele se deslocasse para a frente.



(a) Pose 0 (b) Pose 1 (c) Pose 2 (d) Pose 3

Figura 3.1: Poses das patas geradas por cinemática inversa.

Naturalmente, se esperava que a configuração resultante deveria ser aquela mais parecida com o andar do animal que serviu de inspiração para gerar as poses. Então, em uma última simulação, restringiu-se uma ordem das poses para cada pata de forma que agora o robô caminhasse parecido com um animal quadrúpede. Naturalmente o espaço de busca foi mais restrito ainda, e agora o algoritmo deveria encontrar parâmetros que influenciavam as extremidades (*end-effectors*) das patas, como o ângulo de abertura das poses das figuras 3.1c e 3.1d, e deveria encontrar também a posição no espaço da extremidade da pata durante a pose de transição. 3.1b.

Todos os indivíduos dos casos supracitados deveriam ser avaliados por três funções objetivo:

1. O inverso da velocidade, calculada a partir do último valor válido lido pelo GPS no eixo y de coordenadas do simulador, e do tempo transcorrido na simulação. Se escolhe o inverso pois o algoritmo em questão visa minimizar as funções objetivo, sendo assim, a velocidade seria maximizada.

2. A diferença entre o maior e o menor valor válido lido pelo GPS no eixo z de coordenadas. A ideia é minimizar, até certo ponto, o balanço da altura do robô, visando penalizar indivíduos que caem, por exemplo.

3. A diferença entre o maior e o menor valor válido lido pelo GPS no eixo x de coordenadas. Espera-se que o robô ande apenas no eixo y, então ler uma diferença grande no eixo x significa que o quadrúpede está pendendo para algum lado durante a marcha.

Em relação às opções do algoritmo de otimização, escolheram-se limites para cada uma das três simulações: Para a primeira, definiu-se o limite dos ângulos no intervalo $[-1.5, 1.5]$ rad; Para a segunda, definiu-se o limite das poses no intervalo $[1, 4]$, umas vez que só 4 poses foram modeladas; e para a segunda definiu-se os limites dos parâmetros. Os ângulos das poses das figuras 3.1c e 3.1d foram limitados no intervalo [20,60] graus, a posição x, y e z da extremidade da pata da pose da 3.1b foram limitados nos intervalos $[17, 21]$ cm, $[-4, 0]$ cm e $[0, 4]$ cm, respectivamente. Essas limitações foram impostas afim de evitar parâmetros que desconfiguravam as características de cada pose. Escolheu-se o tamanho da população de 15 indivíduos do tipo vetor de *double*. As demais opções foram:

- Probabilidade de *crossover* de 70%. Isso visava ampliar a influência da mutação no aprendizado, encorajando a diversidade e busca em outras regiões do espaço objetivo.
- Mutação factível a adaptação. Essa opção foi escolhida pois gera aleatoriamente uma direção que é adaptativa à última geração sucedida ou não-sucedida, respeitando as constrições.
- *Crossover* disperso. Isso significa que para a reprodução o algoritmo cria um vetor binário aleatório, com o mesmo número de genes dos indivíduos. E depois ele seleciona genes da mesma posição onde se tem 1 de um primeiro pai, e de onde se tem 0 de outro pai. Seja p_1 e p_2 os dois pais selecionados para a reprodução

$$p_1 = \begin{bmatrix} a & b & c & d & e & f & g & h \end{bmatrix}$$
$$p_2 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$

  E v_s o vetor de *crossover* aleatório.

$$v_s = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

  O filho resultante c é

$$\mathrm{c} = \begin{bmatrix} a & b & 3 & 4 & e & 6 & 7 & 8 \end{bmatrix}$$

  Escolheu-se essa opção por que era a que garantiria, mesmo que de forma aleatória, uma busca por soluções em regiões diferentes do espaço objetivo.

O restante das opções como migração, opções do problema multiobjetivo etc, foram deixadas no modo padrão. Mas vale a pena citar quais são os critério de parada, uma vez que eles determinam o fim da otimização, que sãO:

- Atingir o número máximo de geração, definido como 100×número de variáveis
- Tempo limite, que é definido como inf.
- Limite da função objetivo, definido como -inf.
- *Stall Generations* e tolerância da função objetivo definidos como padrão. Isso significa que se a mudança relativa no espalhamento da frente de Pareto for menor que a tolerância da função objetivo o algoritmo para. Ou seja, o algoritmo pode parar se a distância média entre os indivíduos for menor que um número.

Tempo limite e limite da função objetivo não serão casos de parada do algoritmo, uma vez que são configurados como inf e -inf, respectivamente. Escolheu-se manter esses parâmetros dessa forma pois é desejável que ele não pare ao encontrar um valor zero em uma das funções objetivo, e devido ao não conhecimento de um possível tempo de duração ótimo para este problema.

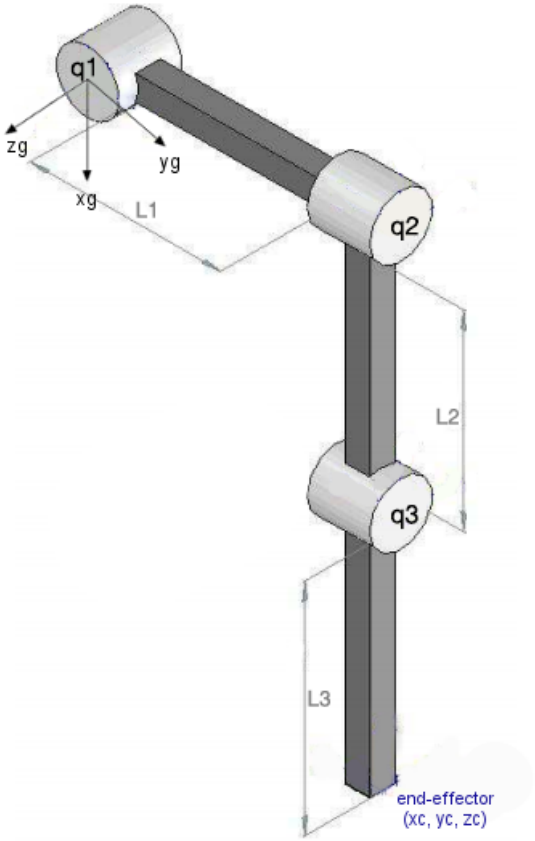


Figura 3.2: Modelo da pata. (Adaptado de: [2])

3.1.1 Modelagem de poses da pata

Como já foi dito, a partir da cinemática inversa, modelaram-se as poses das patas utilizadas nas otimizações. O que se fez foi encontrar um conjunto de equações que descrevessem o comportamento dos ângulos dos atuadores em função da extremidade da pata. Para tanto, utilizou-se o esquema representado na figura 3.2.

Dessa forma, tome como θ_1 o ângulo da junta q_1, θ_2 o ângulo da junta q_2, θ_3, o ângulo da junta q_3, (x_c, y_c, z_c) o ponto no espaço da extremidade, L_2 o tamanho da conexão entre as juntas q_2 e q_3 e L_4 o tamanho da conexão entre a junta q_3 e a extremidade. L_1, a distância entre q_1 e q_2 é desprezível no robô real. Sendo assim, dados os eixos de referência (x_g, y_g, z_g) da figura 3.2, os ângulos das juntas são dados por:

$$\begin{cases} \theta_1 = \tan^{-1}(\frac{y_c}{x_c}) \\ \theta_2 = \tan^{-1}(\frac{z_c}{x_c}) - \tan^{-1}(L_3\cos(\theta_1)\frac{\sin(\theta_3)}{L_2\cos(\theta_1)+L_3\cos(\theta_1)\cos(\theta_3)}) \\ \theta_3 = \pi - \cos^{-1}(\frac{-x_c^2-z_c^2+(L_3\cos(\theta_1))^2+(L_2\cos(\theta_1))^2}{2L_2L_3cos^2(\theta_1)}) \end{cases} \tag{3.1}$$

Com essas equações em mãos, criou-se uma função em Matlab que recebia como parâmetro o *end-effector*. Após alguns testes manuais, encontrou-se um intervalo de valores, para cada direção da posição no espaço da extremidade da pata, que se configurassem nas poses vistas na figura 3.1.

3.2 Integração Matlab e V-REP

Uma vez que o modelo do robô já fora desenvolvido no V-REP por Porphirio e Santana[9] bastaria desenvolver a integração entre o Matlab e o software via uma API remota. A integração precisava ser feita uma vez que facilitaria a otimização do algoritmo genético multiobjetivo, que é uma aplicação do Matlab.

Sendo assim, optou-se por utilizar o modo de operação síncrono, permitindo com que a simulação e o cliente remoto progridam juntos, e que o aprendizado ocorresse de forma automática. Caso fosse usado o modo assíncrono, além de a simulação progredir sem o cliente, seria necessário iniciá-la manualmente a cada novo indivíduo da otimização evolucionária. Isto garante que o otimizador receba os valores das função objetivo relativos ao indivíduo atual.

Então, a integração funciona da seguinte maneira: Recebem-se os parâmetros calculados pelo otimizador e geram-se as sequências de ângulos para cada junta do robô quadrúpede. Sendo assim, obtém-se um vetor de doze ângulos, e cada valor desse vetor deve ser passado para cada junta num mesmo instante de tempo. A API conta com uma função gatilho para realizar a sincronização. Cada chamada dessa função pelo cliente desbloqueia uma etapa na simulação. Dessa forma, assegura-se, em um laço, que cada uma das juntas cheguem à posição desejada, com um erro de $\pm 0,01$ rad, chamando a função que dispara uma etapa da simulação a cada iteração. Como existem situações, durante a simulação, de queda do robô, impedindo de algumas juntas alcançarem suas posições desejadas, esse laço também termina após um tempo máximo.

Além disso, o simulador possuiu um GPS e um acelerômetro. Faz-se necessária a aquisição da evolução dos valores desses sensores no tempo, já que eles são usados para os cálculos das funções objetivo. Então, por meio de funções da API remota, se adquirem tais informações dentro do laço supracitado, garantindo um bom número de dados.

3.3 Otimização Online

Motivados em compensar as diferenças entre a simulação e a aplicação real, propôs-se que, a partir de parâmetros adquiridos na simulação, se desse início a uma otimização da marcha utilizando a plataforma. Para tanto, utilizaram-se apenas dados da última simulação, a que o algoritmo genético multiobjetivo deveria encontrar parâmetros de uma marcha pré-estabelecida, uma vez que as buscas nos espaços maiores (com menos restrições) geravam marchas instáveis e não-cíclicas, muitas vezes fazendo o robô cair de forma abrupta, mas isso será comentado no próximo capítulo.

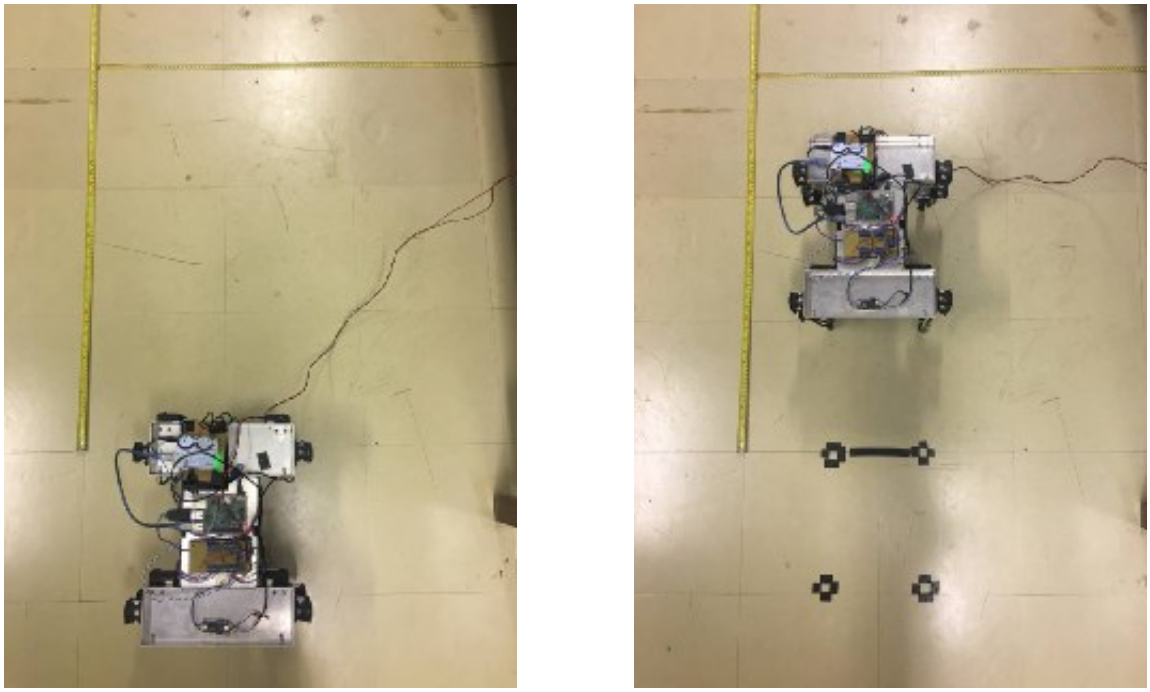
Sendo assim, desenvolveu-se um método de testes que consistia em medir a distância percorrida pelo robô com uma trena, o tempo que o robô terminasse a marcha para que a velocidade fosse calculada e utilizada como função objetivo. Outras duas funções foram criadas. Uma delas seria uma medida qualitativa da marcha do robô, avaliando o balanço que ele executava em cada marcha, ou seja, se ele esticasse demais as patas, fazendo com que ele caísse sobre os joelho, não conseguindo se levantar para executar a próxima etapa, era atribuída uma nota alta, que variava de 0 a 10,

sendo 0 muito bom e 10 muito ruim. Mas agora, se o robô não abrisse muito as patas, fazendo com que, durante a marcha, da transição da pose em 3.1b para 3.1c, ele se empurrasse para trás, uma nota ruim também lhe era atribuída, uma vez que isso desestabilizava o robô. Dessa forma, procurava-se favorecer marchas em que o robô balançasse apenas o suficiente, para uma caminhada fluída. A outra função objetivo era a distância que o quadrúpede avançava para os lados (definido como um avanço no eixo x), uma vez que, por enquanto, desejava-se apenas que o robô andasse para a frente (definida com um avanço no eixo y).

Marchas em que o robô não conseguisse completar o ciclo inteiro, recebiam infinito em todas as funções objetivo, por que eram prejudiciais à plataforma, e desejava-se desfavorecer esse indivíduos durante a otimização. No que tange as opções do algoritmo genético multiobjetivo, foram utilizadas as mesmas das simulações, com exceção ao tamanho da população, que foi reduzido para 5 indivíduos. Além disso, utilizou-se como população inicial e pontuação inicial alguns resultados adquiridos através da última simulação. Em resumo as funções objetivo utilizada foram:

1. Inverso da velocidade do robô no eixo y
2. Nota qualitativa de 0 a 10, que diz a respeito a qualidade da marcha.
3. Distância percorrida pelo robô no eixo x.

A figura 3.3 ilustra como foi montado o local de otimização online. A trena era usada para medir a distância em y e a fita métrica, em x. A posição inicial do robô era demarcada com fita isolante, como é possível na figura 3.3b, a linha demarcava a frente do robô, e os quadrados, onde as patas deveriam estar no início do teste.



(a) (b)

Figura 3.3: Local para os testes de otimização online.

3.4 Alteração nos Códigos - Programa Principal

O código utilizado como base para o treinamento online foi o resultante do trabalho de Phorphirio e Santana [9]. Ele é um código implementado em C++, responsável pela calibração do robô e execução de marchas descritas em arquivos de texto. Essa escolha foi motivada por um anseio de testar as marchas no sistema mais simples possível, progredindo com mais funcionalidades a medida que os resultados fossem satisfatórios.

A primeira mudança realizada foi a eliminação da interação com o usuário durante a execução do programa. Isso se fez necessário para que o processo de treino se dê com a minima intervenção possível. Na implementação base, era necessário informar, no meio da execução, se o processo de calibração seria executado ou não. Com a mudança implementada, essa definição é dada por meio de um parâmetro, atribuído no momento da chamada do programa.

O dispositivo só consegue ficar de pé caso os servos estejam ligados e atuando para que se mantenham em suas posições iniciais, caso contrario as articulações cedem e o robô cai. Na implementação base, após a execução da marcha, a atuação dos servos não era mais realizada. Essa dinâmica não era interessante, o anseio é que o dispositivo se mantenha fixo em sua posição inicial, esperando por um reposicionamento, para então executar uma nova marcha. Eliminando a necessidade de utilização de um suporte para cada nova iteração do treinamento. Para que esse comportamento seja alcançado, foi implementada uma lógica que, a partir de um parâmetro, defina se o sistema deve manter ou não o torque no motor após o fim da marcha.

Por fim, o código foi adaptado para adquirir e salvar todos os dados do acelerômetro, sensores de força e posições dos motores ao longo do tempo. Cada dado é salvo em um arquivo de texto separado, que pode ser lido por outros programas. Além disso, no final da execução de uma marcha, é informado o tempo gasto na execução daquela sequencia de movimentos. A forma de obtenção do tempo é utilizando a biblioteca *sys/time.h*.

3.5 Interface de Treinamento

O processo de treinamento *online* envolvia diversas etapas, que seriam repetidas para cada individuo gerado. Visando automatizar o máximo de processos possíveis, foi implementado um código em Python para realizar as tarefas exigidas pelo robô.

No inicio da execução, o programa de treinamento oferece opções para que o usuário defina o seu objetivo, que pode ser: finalizar o programa, liberar os motores após a execução das marchas, calibrar o robô ou executar a marcha mantendo os servos fixos após a execução. A escolha dessas opções definem os parâmetros que são passados na execução do programa principal.

Após a definição do modo de operação, executa-se a lógica principal do algoritmo de treinamento. Primeiramente o programa realiza a leitura de um arquivo de texto armazenado no computador do usuário. Nele estão definidos os parâmetros de uma marcha. Em posse desses dados, é criada uma conexão SSH com a plataforma, para que isso seja possível, ambos os dispositivos

devem estar conectados na mesma rede.

Uma vez estabelecida a conexão remota entre o computador e o robô, os comandos serão realizados apenas na plataforma. A primeira tarefa realizada é a geração de uma marcha a partir dos parâmetros obtidos. Para isso utiliza-se um codigo-fonte que gera um arquivo de texto, a partir dos valores dos parâmetros da marcha a ser realizada. Após isso, é executado o programa principal, utilizando as condições que foram previamente definidos na seleção de opções. No fim da execução do programa principal, os dados coletados da marcha são copiados para uma pasta nomeada pelo usuário.

A dinâmica de funcionamento do algoritmo de treinamento se deu para que seja possível deixar o Matlab responsável pela execução do AG. Realizando a tarefa de geração das marchas para cada indivíduo. Enquanto o algoritmo de treino, executado no mesmo computador do Matlab, poderia obter esses parâmetros e realizar execução da marcha de forma remota.

Ao fim de cada marcha, eram medidos e avaliados os valores referentes as funções de avaliação e informados manualmente ao Matlab. Permitindo que o programa realizasse as otimizações e gerasse novos indivíduos, reiniciando o ciclo de treinamento.

3.6 Manutenção mecânica

A estrutura mecânica do dispositivo se mostrou satisfatória em um primeiro momento, conseguindo sustentar o próprio peso e realizando os movimentos demandados. Entretanto com o decorrer do trabalho notou-se algumas folgas nas conexões entre as peças.

Visando a eliminação de folgas nas conexões, foram apertados todos os parafusos da plataforma, diante disso notou-se uma melhora satisfatória nas conexões referentes as juntas q_3.

Mesmo com o aperto nos parafusos, a junta q_1 de uma das patas possuía uma folga. Isso comprometia o funcionamento da plataforma como um todo, uma vez que mesmo que fosse definida uma posição de referência, ela não necessariamente seria atingida pela pata, haja vista que existe um deslizamento. Esse deslizamento se deu devido ao desgaste mecânico da conexão, que consistia de uma peça de nylon perfurada que se encaixava nas ranhuras da extremidade do atuador, como pode ser visto na Figura 3.4. Visando uma melhor fixação, as peças foram unidas utilizando um adesivo à base de resina epóxi.

É importante ressaltar que ao realizar a fixação, se deu a devida atenção a posição absoluta do servos. Tendo em vista que mesmo sendo realizada uma calibragem, proposta por Porphirio e Santana [9], caso a posição de calibração do servo esteja muito próxima do ponto mínimo ou do ponto máximo de operação, certas posições de referência não poderão ser atingidas, prejudicando a movimentação da plataforma.

Ao final do desenvolvimento, foram propostas quatro abordagens de otimização, sendo três realizadas via simulação e uma na plataforma. Além disso, todos os códigos computacionais necessários para a realização dessas abordagens foram implementados. Permitindo a comunicação do MATLAB com o V-REP, a execução remota de marchas no robô e a utilização de uma interface

Figura 3.4: Conexão do servo com a estrutura de nylon na junta q_1

própria para o treinamento na plataforma real. No capítulo 4, englobam-se os resultados obtidos na otimização por meio dessas abordagens.

Capítulo 4

Resultados

Aqui se expõe e se analisam os resultados obtidos das otimizações propostas: espaço aberto de soluções, restrição por poses, restrição por ordem de poses e treinamento online. O resultado é composto pro gráficos de posições dos motores, de sensores, frentes de pareto, de características da otimização e quadros de imagens que representam a execução das marchas adquiridas.

4.1 Simulação Computacional

Como já foi comentado em capítulos anteriores, se fez a otimização na simulação utilizando três tentativas: A primeira, sem restrição alguma no espaço objetivo de soluções, o algoritmo deveria encontrar um conjunto de ângulos para cada servo motor que resultasse em alguma marcha factível para o robô, não necessariamente restrita a marcha de um animal, por exemplo. Na segunda, o algoritmo deveria encontrar uma sequência de poses para cada pata, que se traduzisse também em uma marcha factível para o quadrúpede. Quatro tipos de poses foram modeladas para essa tarefa. Na terceira tentativa, restringiu-se ainda mais o espaço de busca, amarrando a sequência de poses para cada pata, de forma que a marcha resultante fosse a mais parecida com a de um animal. Então, o otimizador deveria encontrar cinco parâmetros que modificavam algumas características das poses. Os parâmetros são: ângulo de abertura das patas das poses das figuras 3.1c e 3.1d (poses 2 e 3, respectivamente), e a posição no espaço (x_c, y_c, z_c) da extremidade da pata na pose da figura 3.1b (pose 1), a pose de transição.

4.1.1 Espaço aberto de soluções

Como já foi dito, o algoritmo aqui deveria encontrar um conjunto de ângulos para cada servomotor, que variavam com tempo, a fim de tentar encontrar alguma marcha que fosse factível ao robô. Mas como o espaço objetivo é muito grande, o algoritmo não encontrou um conjunto de soluções que se parecesse com alguma marcha, ou que fizesse o robô se locomover de alguma forma estável. Os pontos que foram encontrados para compor a frente de Pareto desse caso eram

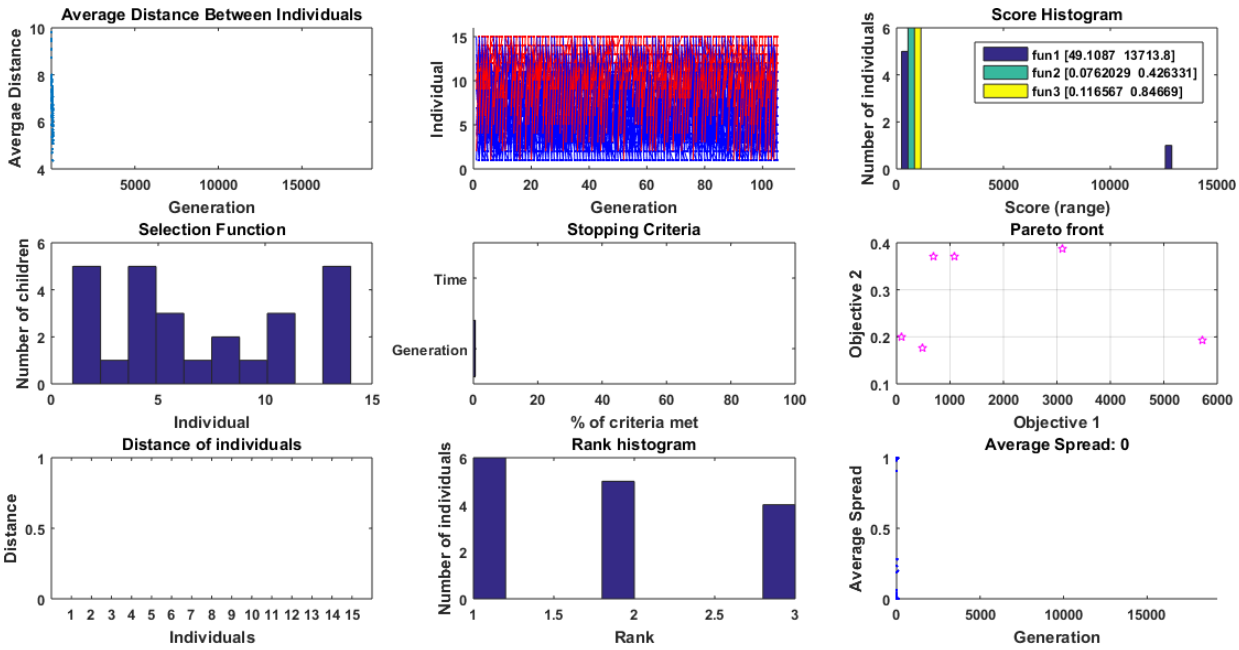


Figura 4.1: Gráficos gerados ao fim da primeira otimização: Distância média entre indivíduos, histograma de pontuação, função de seleção, critério de parada, frente de Pareto, distância entre indivíduos, histograma de *rank* e espalhamento médio.

movimentos caóticos das patas dos robôs, fazendo muitas vezes ficar de cabeça para baixo. Claramente não são resultados que podem ser aplicados na plataforma. As Figuras 4.1 e 4.2 ilustram, respectivamente: os gráficos gerados ao fim da otimização, que dizem respeito a distância média entre os indivíduos, genealogia, histograma de pontuação a cada geração, função de seleção de pais, critério de parada, frente de pareto, distância entre indivíduos, histograma de rank e espalhamento médio; E a frente de pareto em três dimensões, uma vez que se trata de um problema com três objetivos e o gráfico gerado pelo aplicativo mostra apenas uma das faces do espaço de soluções na frente de pareto. A otimização terminou por que a distância média entre os indivíduos era menor que um valor determinado pelo padrão do aplicativo.

Ao analisar a Figura 4.2 vê-se que o inverso da velocidade dos pontos não é tão grande assim, chegando próximo do 5000, sugerindo que ouve um deslocamento, mesmo que mínimo nesses pontos. Porém, a diferenças em x e z são grandes, em comparação ao tamanho do robô. Uma variação máxima de $0,3$ metros em z significa que o robô caiu.

Já as figuras 4.3 e 4.4 mostram a posição dos motores e as leituras dos sensores GPS com os ângulos de arfagem e rolagem no tempo, respectivamente, de um indivíduo na frente de pareto dessa otimização. Ao observar as posições dos motores repara-se que não existe sincronia ou simetria entre elas, que é o que se espera de uma marcha. Todas os subgráficos são diferentes uns dos outros. E essa desordenação é refletida nas leituras do sensor acelerômetro, utilizado para calcular os ângulos de arfagem e rolagem. A variação destes ângulos no tempo, para esse indivíduo específico, indicam uma empinada seguida de uma queda sobre patas laterais. Esses detalhes podem ser melhores vistos ao observar a figura 4.5, que é uma sequência de quadros do padrão de movimento executado nesse caso.

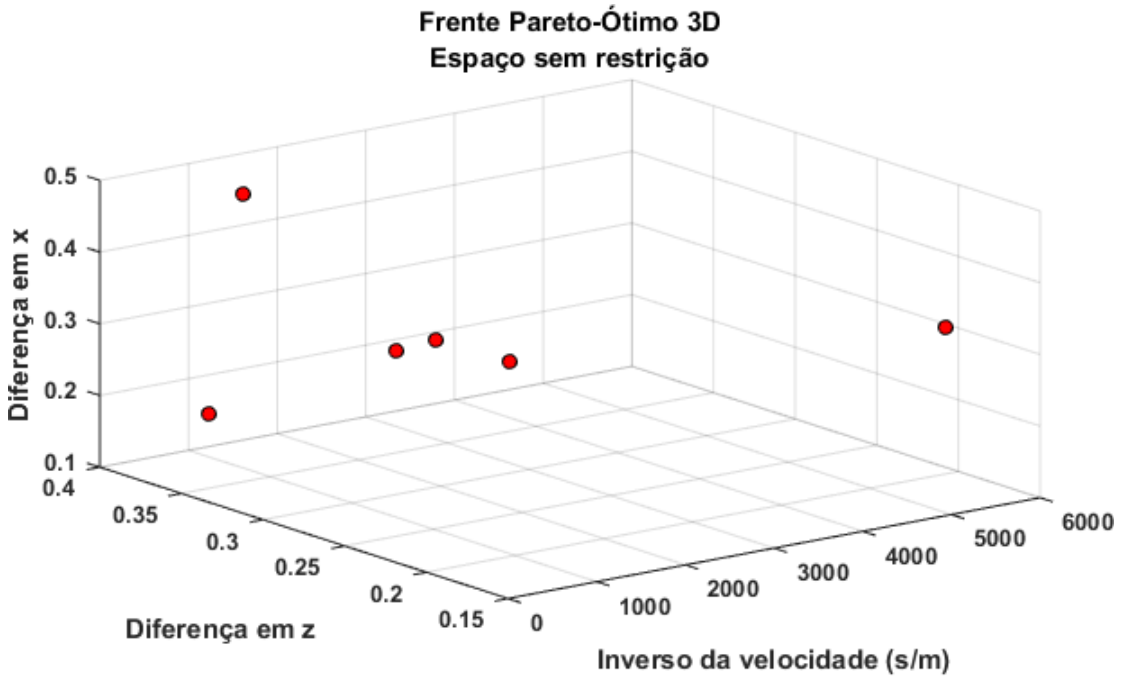


Figura 4.2: Frente de Pareto em 3D para a primeira otimização.

Já o GPS registrou uma pequena variação no deslocamento em y, várias flutuações em x e z, resultado aleatório da desordenação das patas.

Esse resultado era o esperado para esse caso, uma vez que o espaço de soluções aqui era muito grande, diminuindo as chances de o otimizador encontrar uma marcha factível ao robô, mesmo com funções objetivos que visavam maximizar a velocidade em y e minimizar deslocamentos abruptos em z, em outras palavras, minimizar as quedas.

O número de indivíduos foi pequeno para o tamanho do espaço, mas mesmo assim, acredita-se que para que se encontrasse alguma solução viável, o número de indivíduos seria tão grande que o tempo de otimização seria inviável.

4.1.2 Restrição por poses

Nessa simulação o algoritmo não aprendeu uma ordem de poses que fizesse o robô avançar. Em muitas das marchas da frente de Pareto ele movia bem pouco as patas, se empurrando bem pouco na direção y, mas se mantendo quase imóvel. Em outras, ele tentava combinações que o colocavam "sentado", mas também sem se mover na direção y. As figuras 4.6 e 4.7 mostram os gráficos gerados ao fim dessa segunda otimização e a frente Pareto-ótima em um espaço tri-dimensional, respectivamente.

A primeira característica que se repara ao analisar a figura 4.7 é a ordem de grandeza do eixo do inverso da velocidade. Isso reflete o fato que o robô andou pouco ou quase nada, nessas marchas. Como a distância considerada era muito próxima de zero, a velocidade calculada também é muito pequena. Outra análise que pode ser feita é em relação aos outros dois objetivos. Em ambos

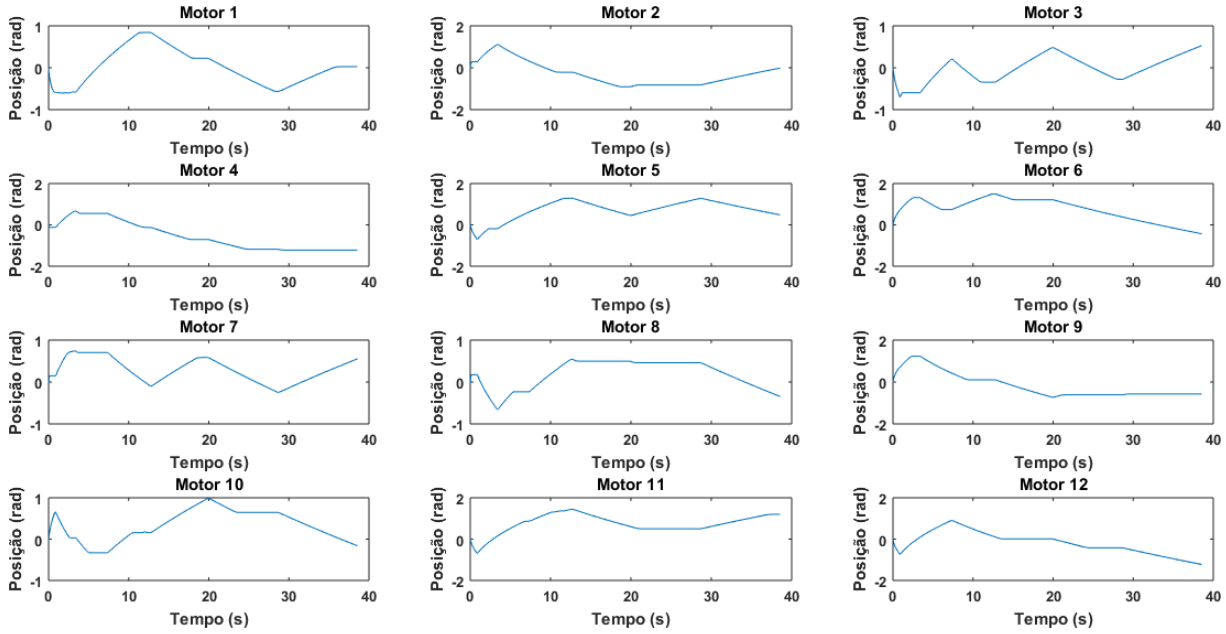


Figura 4.3: Posição dos motores para uma marcha na frente de Pareto da primeira simulação.

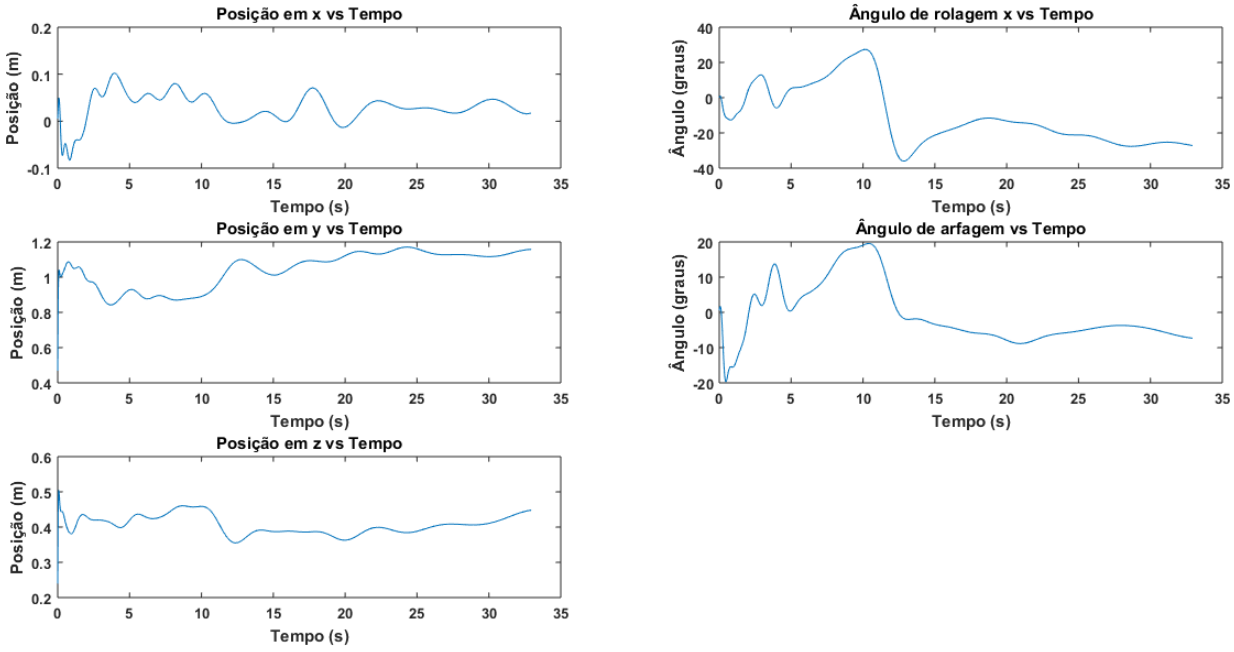


Figura 4.4: Valores dos sensores de GPS e dos ângulos de arfagem e rolagem para uma marcha na frente de Pareto da primeira simulação.

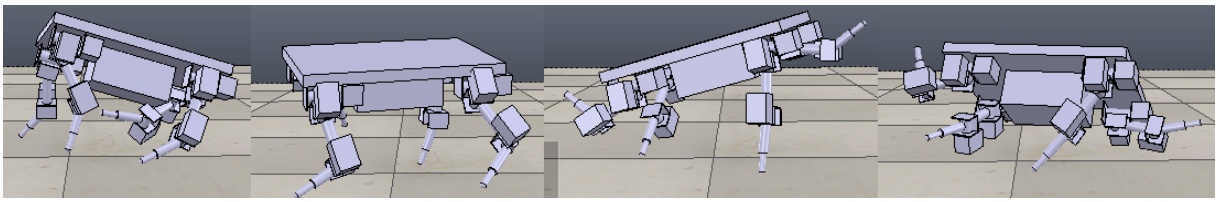


Figura 4.5: Um dos resultados da frente de Pareto encontrados na primeira otimização.

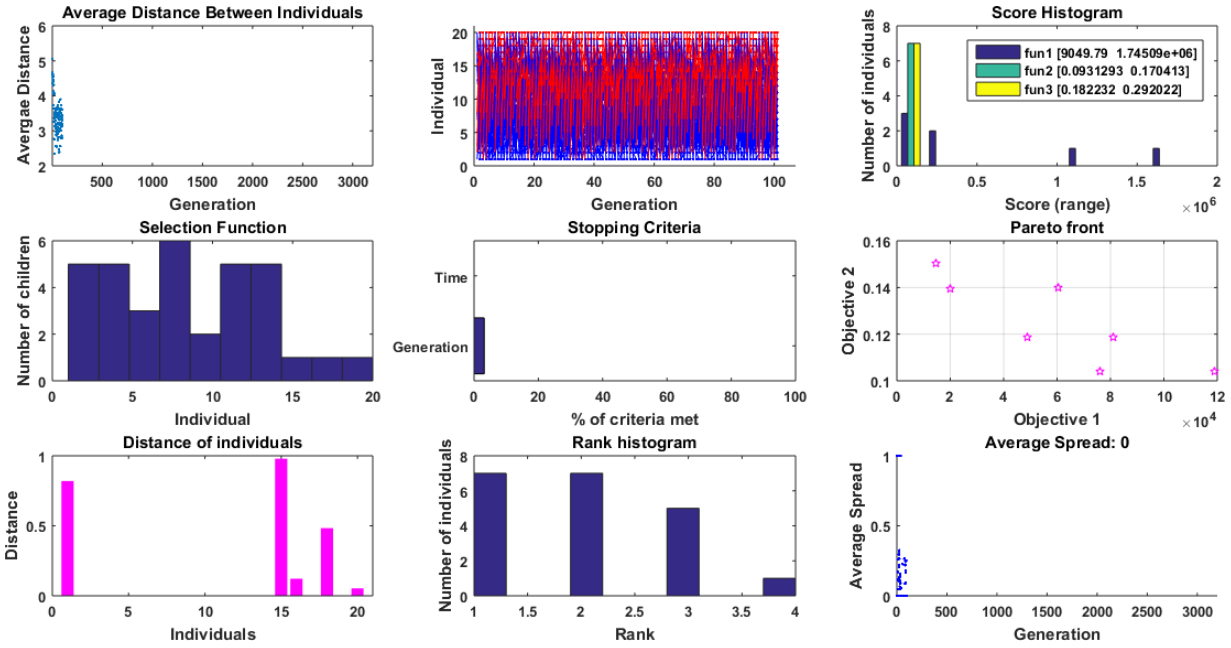


Figura 4.6: Gráficos gerados ao fim da segunda otimização: Distância média entre indivíduos, histograma de pontuação, função de seleção, critério de parada, frente de Pareto, distância entre indivíduos, histograma de *rank* e espalhamento médio.

os casos os número obtidos foram próximos de zero, o que é plausível, uma vez que o robô ficou parado e quase nenhuma variação acontece nas posições do robô nos eixos x e z.

As figuras 4.8 e 4.9 poses ilustram as posições dos motores e a variação das medidas do GPS e dos ângulos de arfagem e rolagem pra uma das marchas encontradas na frente de Pareto dessa otimização, restringindo o espaço de busca pelas poses.

Repare que os gráficos dos motores 1, 3, 10 e 12, da figura 4.8 são completamente diferentes dos demais e parecem fazer nenhum sentido. Na verdade isso é reflexo do queda que o robô sofreu na marcha em que foram adquiridos esses dados. Sendo assim, esses atuadores foram impedidos de realizar a trajetória desejada.

Já na figura 4.9, a escala dos gráficos das posições, que são as leituras do GPS nos eixos x, y e z, também revela que o robô ficou parado. Pequenas variações em y, revelam que o robô se deslocou um pouco nesse eixo. Mas ao se analisar a figura 4.10, vê se que na verdade o deslocamento, foi gerado por um queda, como sugerido pelos gráficos das posições dos motores. Essa evidência pode ser verificada também no gráfico ângulo de arfagem, que decai em função do tempo.

Em uma última análise desse caso, pode-se observar novamente na figura 4.10 que nos dois primeiros quadros existe uma coordenação entre as duas patas de trás que sugere uma marcha, mas as patas da frente não se moveram, o que fez com que o robô caísse.

Nesse caso esperava-se que o algoritmo aprendesse alguma ordem das poses que o fizesse marchar. Porém, acredita-se que o maior impedimento a isso foi o fato de que o otimizador não lida apenas com número inteiros no momento de calcular as alterações que serão feitas nos indivíduos, como por exemplo na mutação, e a ideia seria que o otimizador gera-se uma matriz 4×4, em que as

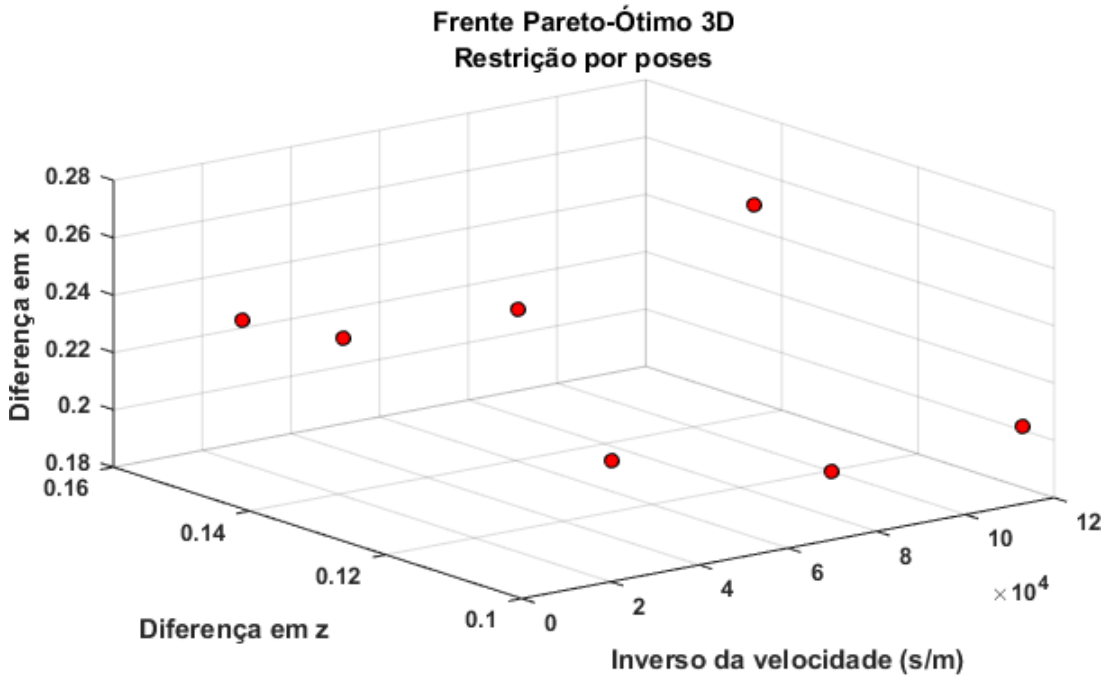


Figura 4.7: Frente de Pareto em 3D para a segunda otimização.

suas posições seriam um número inteiro no intervalo $[1, 4]$ que representasse cada uma das poses. Como as alterações, principalmente ao final da otimização, eram feitas nas casa decimais desses números, elas praticamente não importavam, uma vez que só se avaliava a sua parte inteira para a execução no robô.

4.1.3 Restrição por ordem de poses

Nesse caso, ao se iniciar os parâmetros das poses de forma arbitrária, o robô já marchava. Porém se fez necessário a otimização para que, por exemplo, evitasse o robô de abrir demais as pernas, ou de, na pose de transição, a pata não levantar o suficiente, causando um arrastamento da pata no chão, dificultando a locomoção, ou até mesmo fazendo o robô cair.

Dessa forma, a otimização demorou cerca de 6 horas para ser concluída, e as figuras 4.11 e 4.12 representam os gráficos resultantes da otimização e a frente Pareto ótima tri-dimensional, respectivamente.

No gráfico da frente de Pareto tri-dimensional, pode-se reparar também na ordem de grandeza da escala no eixo do inverso da velocidade. Isso acontece pois um único ponto da frente teve uma pontuação muito alta nesse objetivo em relação aos outros. A razão disso é por quê existe uma certa instabilidade nas medidas do GPS do V-REP, acredita-se que esse foi um dos casos,o que ocasionou uma falha na leitura do sensor no eixo y. Ela compõe um ponto da frente Pareto-ótimo pois possui valores baixos nos outros dois objetivos. Os outros pontos possuem ordem de grandeza de 10 e 10^2. A marcha mais rápida atingiu cerca de $0,01$ m/s, a troco de avançar mais no eixo x.

Nas figuras 4.13 e 4.14 é possível ver as posições de cada motor no tempo na simulação, ao

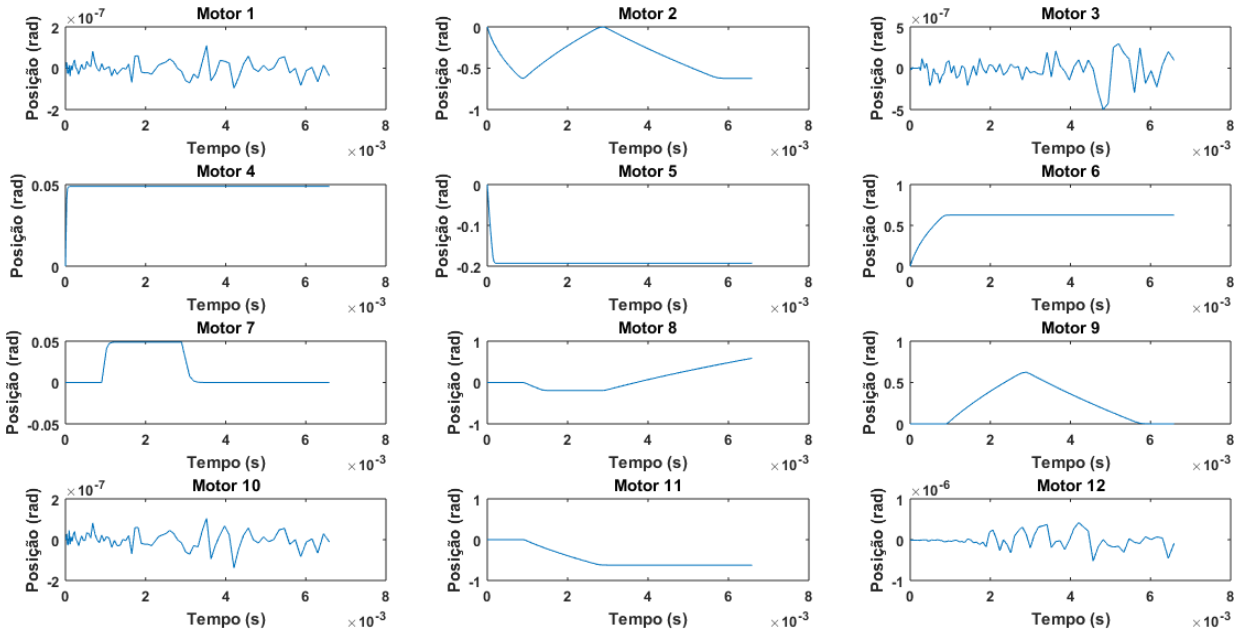


Figura 4.8: Posição dos motores para uma marcha na frente de Pareto da segunda simulação.

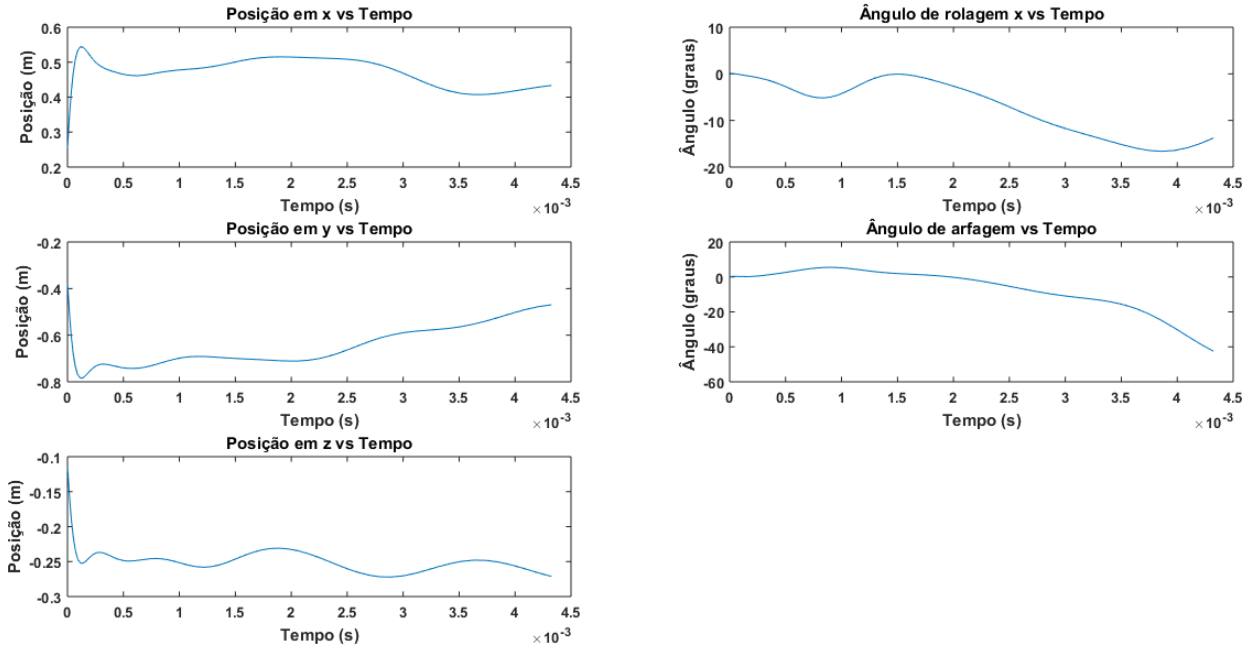


Figura 4.9: Valores dos sensores de GPS e dos ângulos de arfagem e rolagem para uma marcha na frente de Pareto da segunda simulação.

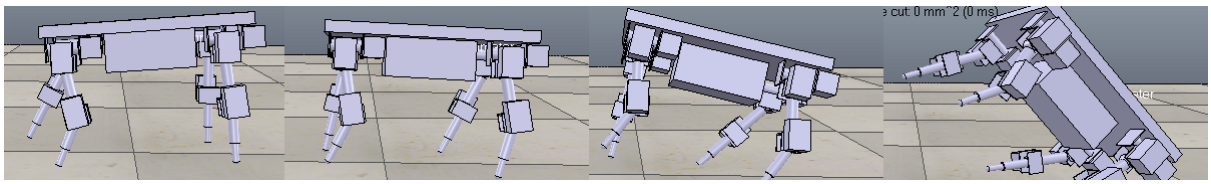


Figura 4.10: Um dos resultados da frente de Pareto encontrados na segunda otimização.

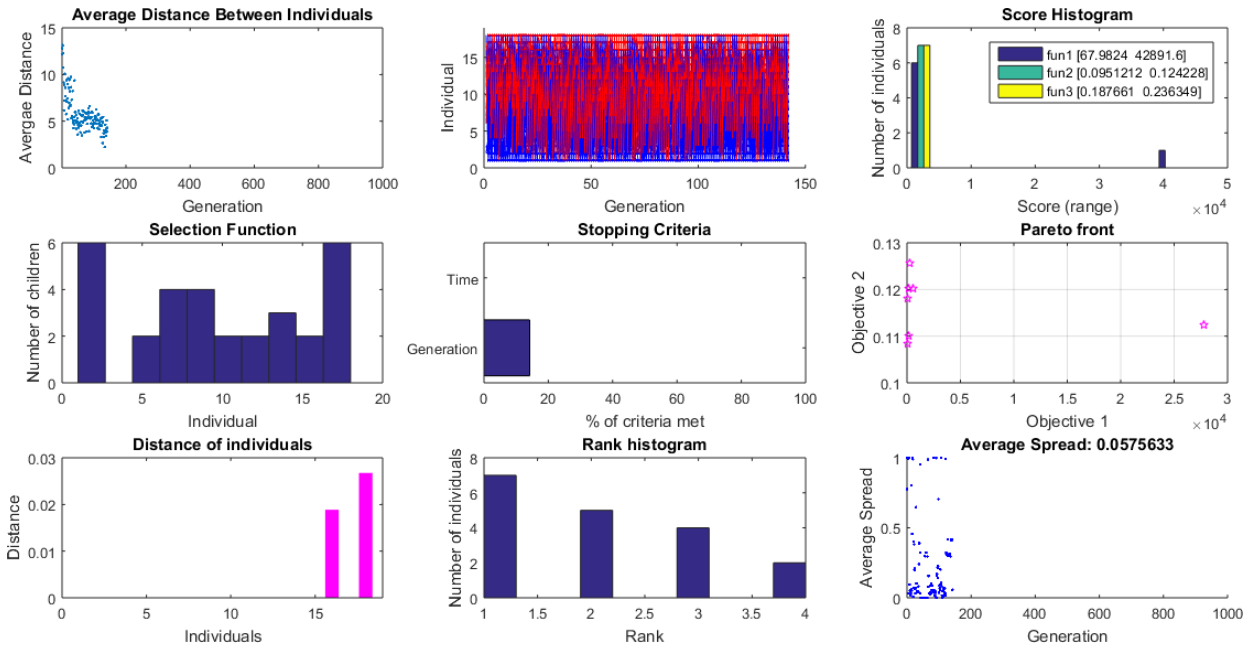


Figura 4.11: Gráficos gerados ao fim da terceira otimização: Distância média entre indivíduos, histograma de pontuação, função de seleção, critério de parada, frente de Pareto, distância entre indivíduos, histograma de *rank* e espalhamento médio.

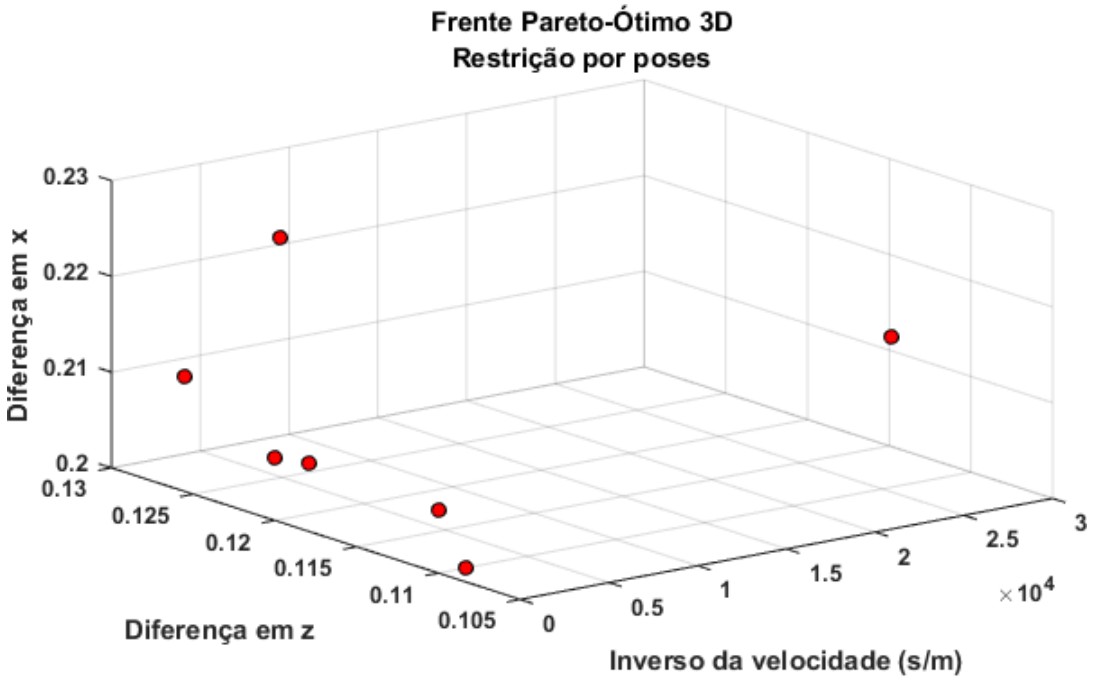


Figura 4.12: Frente de Pareto 3D para a terceira otimização.

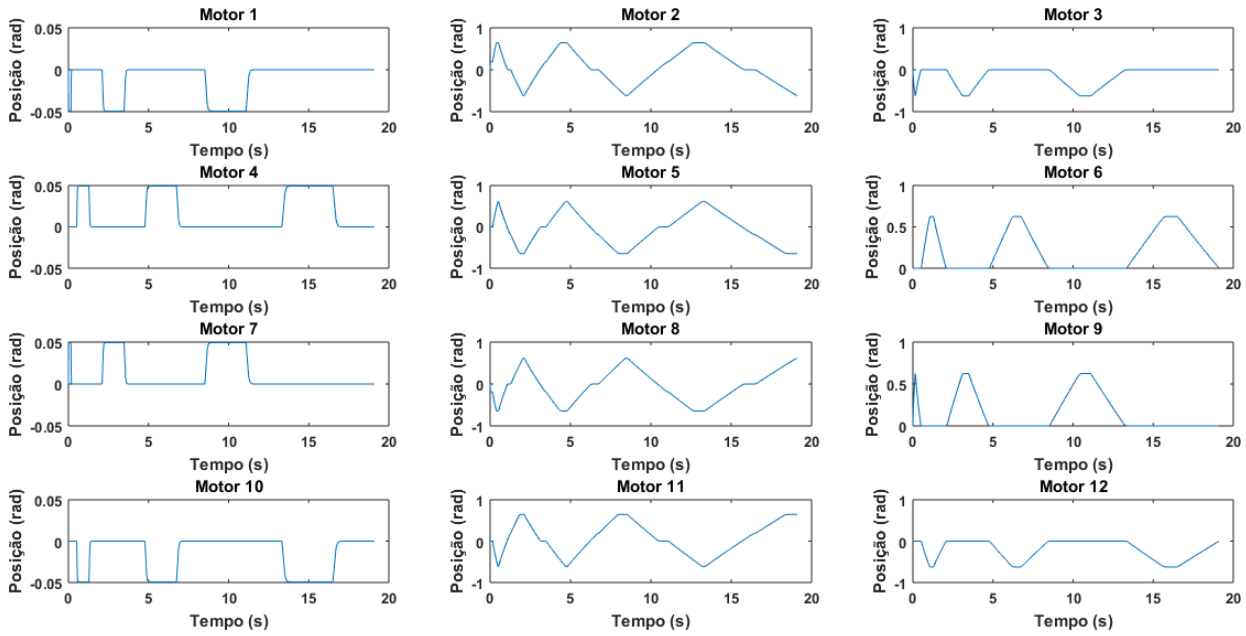


Figura 4.13: Posição dos motores para uma das marchas encontradas na frente de Pareto, para a otimização com restrição da ordem das poses.

executar duas das marchas da frente de Pareto obtidas. É possível ver que não muita diferença entre elas, uma vez que foi fixada a ordem das poses. Mas o que se pode reparar é as posições dos motores na figura 4.13 são mais "esticadas", que é o reflexo de uma marcha que demora mais tempo para ser executada, uma vez que a abertura dos ângulos das patas é maior, característica que pode ser observada nos gráficos da coluna do meio, que representam a posição dos servo motores que seriam a junta q_2 da figura 3.2, responsáveis por ampliar o diminuir a passada.

Outra característica das marchas obtidas é que o robô abre a passada na direção ortogonal ao movimento, durante a pose de posição. Isso pode ser verificado pelas posições dos motores na primeira coluna das figuras 4.13 e 4.14, em que o servomotor do ombro, ou q_1 na figura 3.2, sofre deslocamentos. Verificou-se que essa característica aumenta a estabilidade do robô durante as passadas, uma vez que ele inclina o corpo para frente durante a troca de poses.

Outro resultado obtido foi o gráfico dos sensores GPS e dos ângulos de rolagem e arfagem da simulação. É possível perceber que em ambas as figuras 4.15 e 4.16 a posição em y, direção que o robô andava, é crescente.

A troca entre as duas, se dá pelo fato que a diferença entre a posição de maior e menor valor no eixo z da segunda marcha é menor que a primeira, caracterizado pela menor abertura das patas nas poses 2 e 3. Porém, ela perde na velocidade e na estabilidade da arfagem para a primeira, podendo isso ser verificado, de modo quantitativo, comparando os gráficos da posição em y no tempo, e de maneira qualitativa, os gráficos do ângulo de arfagem no tempo das figuras 4.15 e 4.16.

A figura 4.17 representa a sequência de passos da marcha referente a figura 4.13. É possível ver que o robô propõe um balanço para executar a marcha, corroborado pelos gráficos de arfagem e

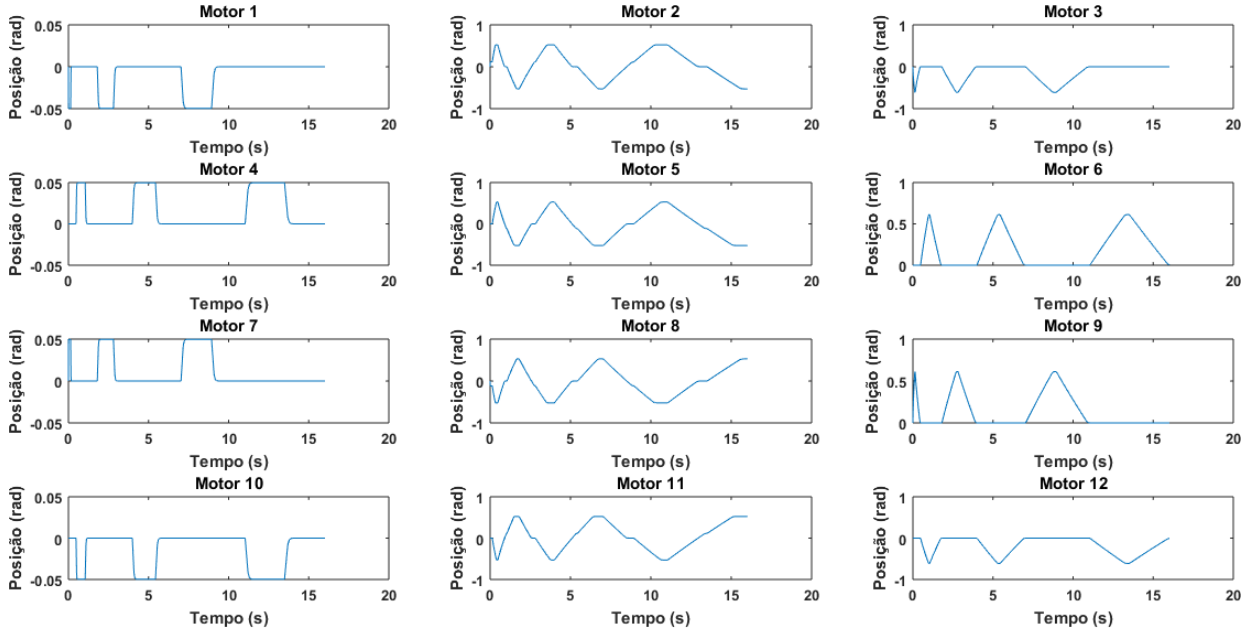


Figura 4.14: Posição dos motores para outra marcha encontrada na frente de Pareto, para a otimização com restrição da ordem das poses.

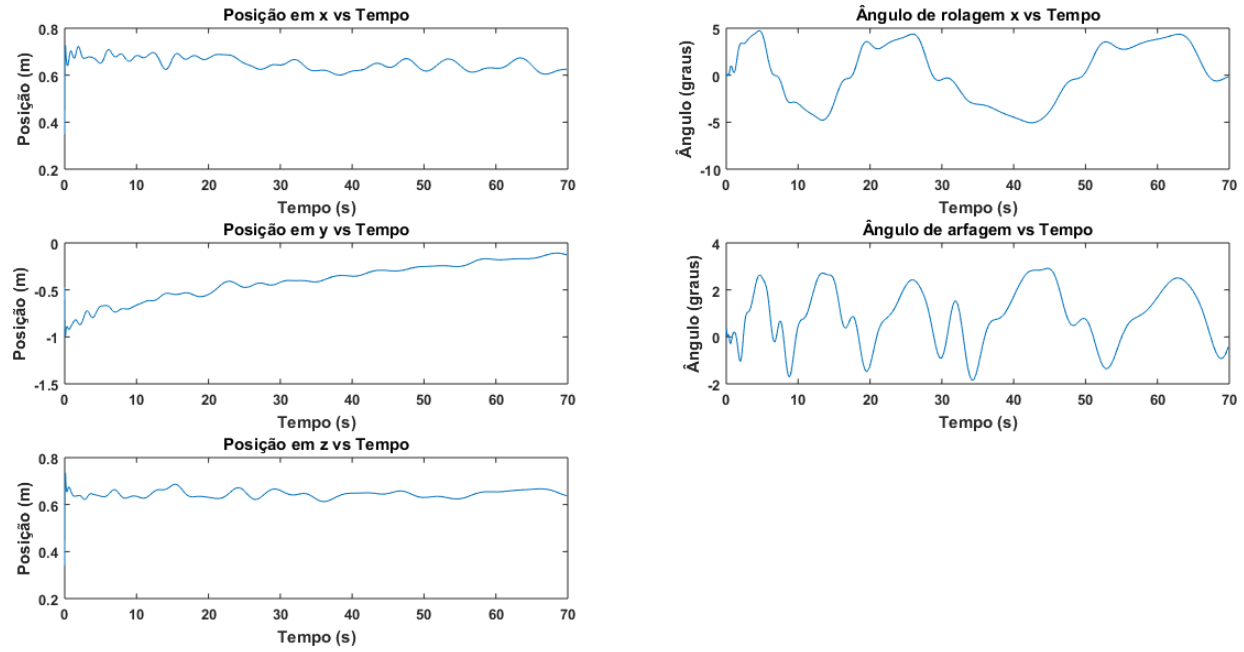


Figura 4.15: Valores dos sensores para uma das marchas encontradas na frente de Pareto, para a otimização com restrição da ordem das poses.

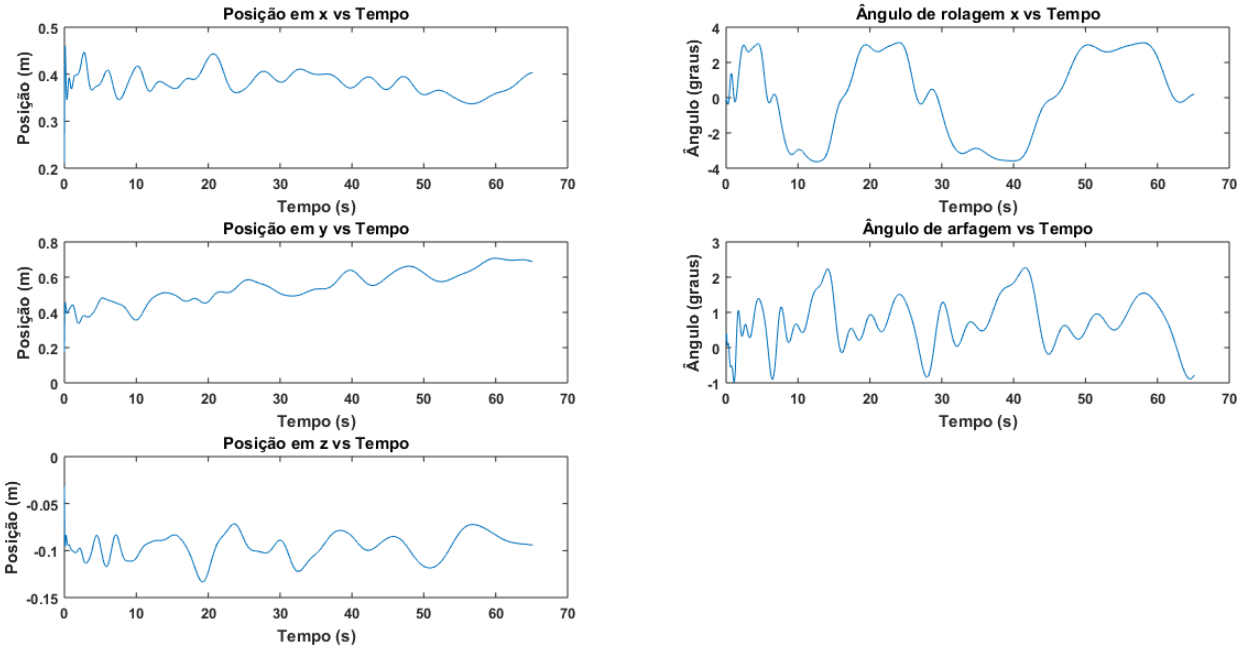


Figura 4.16: Valores dos sensores para a outra marcha encontrada na frente de Pareto, para a otimização com restrição da ordem das poses.

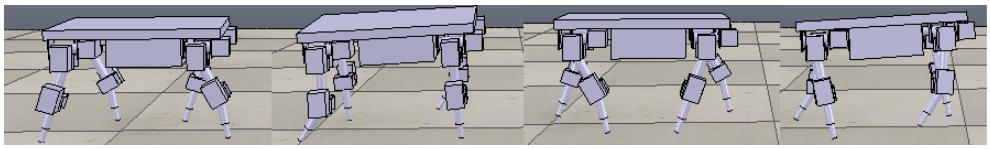


Figura 4.17: Marcha do robô adquirida na terceira simulação.

rolagem, em que claramente pode-se ver uma variação cíclica no tempo, e que existe uma sincronia entre as patas. Também é possível ver todas as poses sendo executadas.

4.2 Treinamento Online

O treinamento online foi realizado utilizando a restrição por ordem de poses, uma vez que essa restrição, quando aplicada na simulação computacional, gerou resultados satisfatórios. Promovendo uma melhor performance, menor possibilidade de queda e comportamento cíclico.

Utilizando como população inicial algumas marchas obtidas na simulação, foi realizada a otimização. Nessas condições, o treinamento na plataforma demorou 7 horas para ser concluído. As figuras 4.18 e 4.19 representam os gráficos resultantes da otimização e a frente de Pareto ótima tri-dimensional, respectivamente. Nota-se portanto que duas marchas foram obtidas ao final do processo.

A primeira marcha resultante do processo de otimização pode ser vista no vídeo [1] e na sequência de imagens da na Figura 4.20. Essa marcha obteve um valor de 0.0931 s/cm para o inverso da

[1] `https://youtu.be/S6u3aeSmM3M`

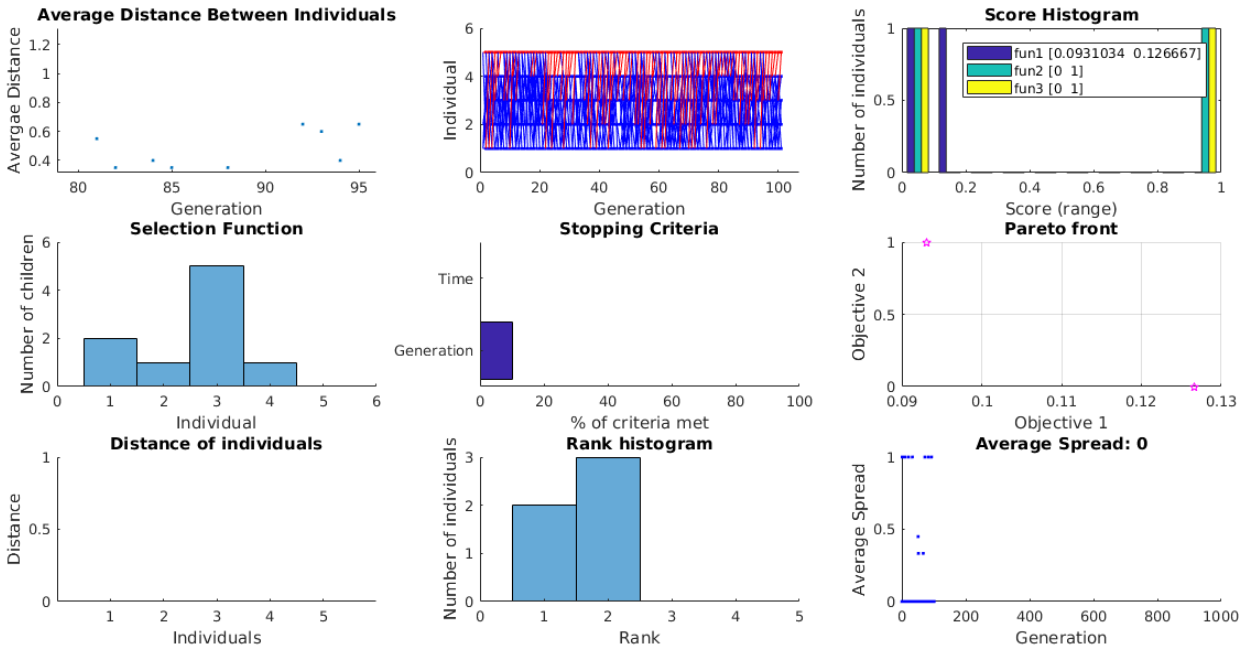


Figura 4.18: Gráficos gerados ao fim da otimização online: Distância média entre indivíduos, histograma de pontuação, função de seleção, critério de parada, frente de Pareto, distância entre indivíduos, histograma de *rank* e espalhamento médio.

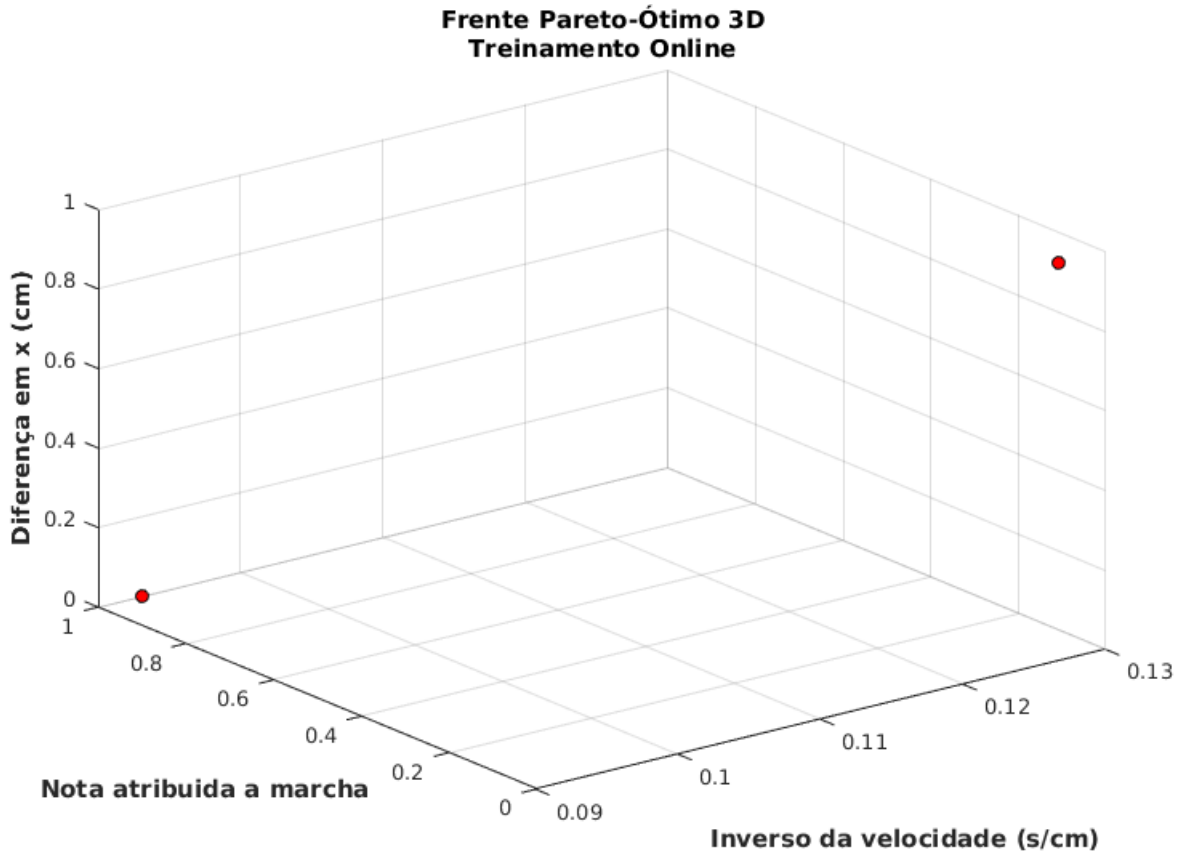


Figura 4.19: Frente de Pareto 3D para a otimização online.

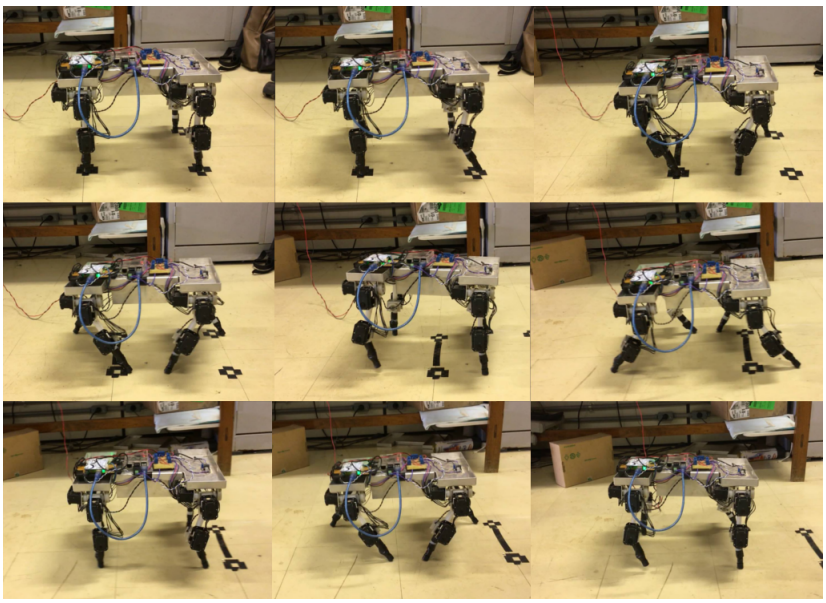


Figura 4.20: Sequencia de imagens da execução da primeira marcha resultante da otimização

velocidade. Portanto infere-se que o robô se locomoveu a uma velocidade de 0.1074 m/s. Além disso é importante destacar que não houve desvio no eixo x, um resultado muito satisfatório. As figuras 4.21 e 4.22 representam, respectivamente, os gráficos das posições dos motores e os ângulos de rolagem e arfagem, ao longo do tempo, durante a execução da marcha.

A segunda marcha resultante do processo de otimização pode ser vista no vídeo [2] e na sequência de imagens da na Figura 4.23. Essa marcha obteve um valor de 0.1267 s/cm para o inverso da velocidade. Portanto infere-se que o robô se locomoveu a uma velocidade de 0.0789 m/s. O desvio no eixo x foi de apenas 1 cm. As figuras 4.24 e 4.25 representam, respectivamente, os gráficos das posições dos motores e os ângulos de rolagem e arfagem, ao longo do tempo, durante a execução da marcha.

As duas marchas geradas são muito semelhantes entre si. Nota-se a partir da Tabela 4.1 , que apenas o parâmetro referente a posição em z do *end-effector* é diferente entre as duas marchas. A consequência física dessa diferença pode ser observada por meio dos vídeos. Na realização da segunda marcha nota-se um leve arrasto das patas traseiras no solo.

Tabela 4.1: Parâmetros das marchas resultantes do treinamento online

	Ângulo de abertura posterior	Ângulo de abertura frontal	xc	yc	zc
Primeira Marcha	27,9247	32,0582	20,6767	-3,3744	1,4311
Segunda Marcha	27,9247	32,0582	20,6767	-3,3744	0,4311

A partir da análise das Figuras 4.21 e 4.24 nota-se que as marchas possuem comportamentos muito similares, entretanto, o tempo de conclusão da primeira marcha é menor que o da segunda.

[2] `https://youtu.be/ZkV_LA6ohMM`

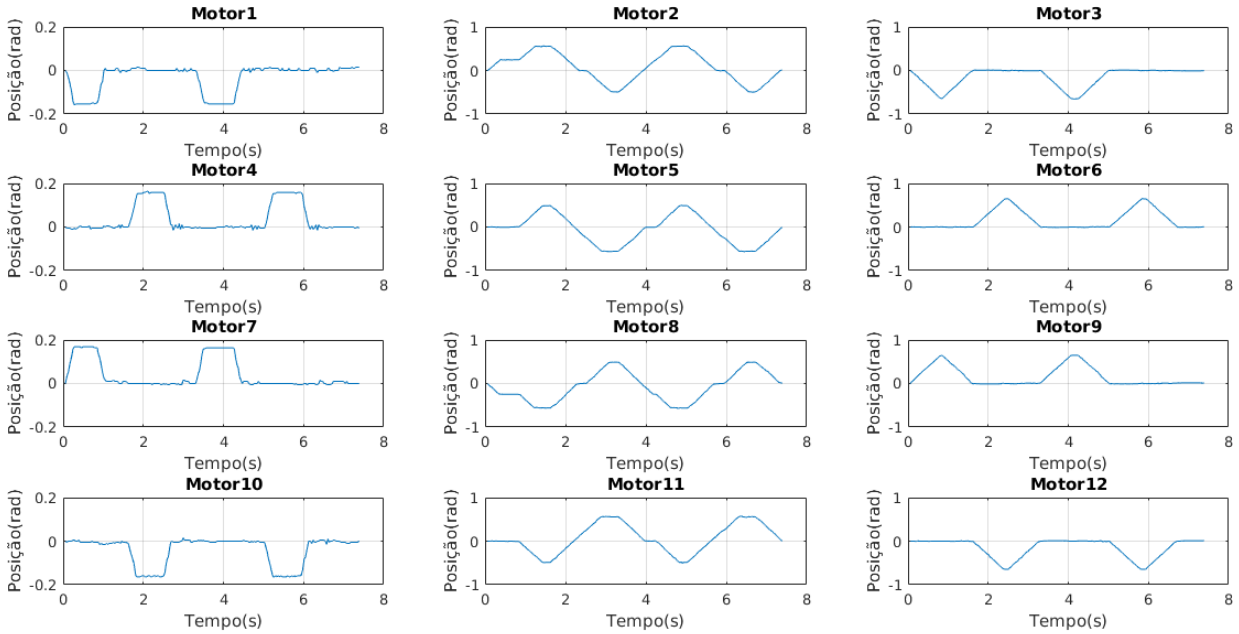


Figura 4.21: Posição dos motores para a primeira marcha encontradas na frente de Pareto, na otimização online.

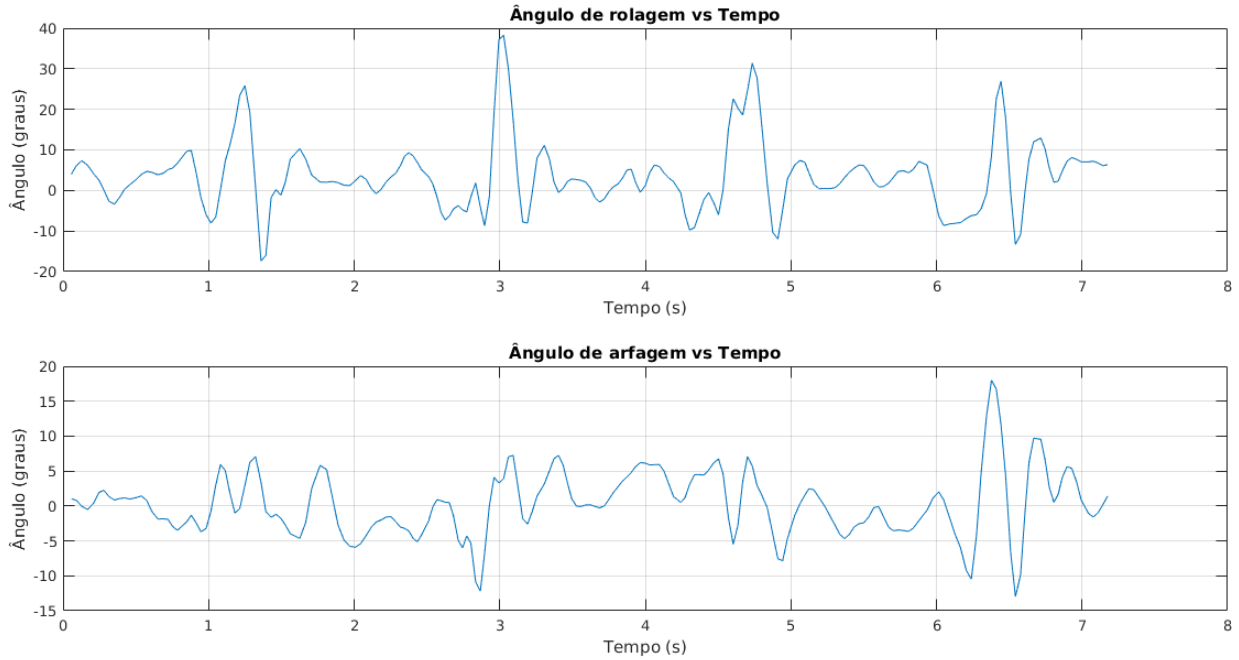


Figura 4.22: Valores dos ângulos de rolagem e arfagem ao longo do tempo para a primeira marcha encontrada na frente de Pareto, na otimização online.

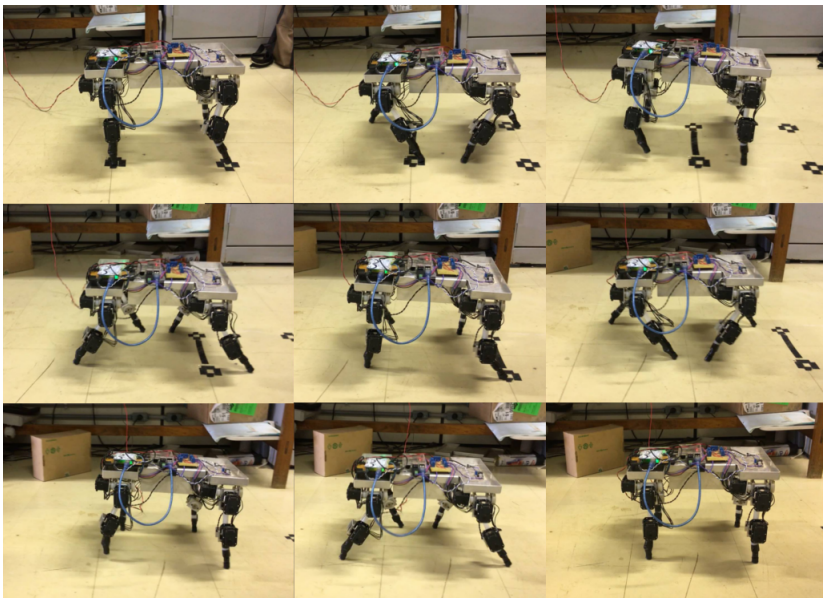


Figura 4.23: Sequencia de imagens da execução da segunda marcha resultante da otimização

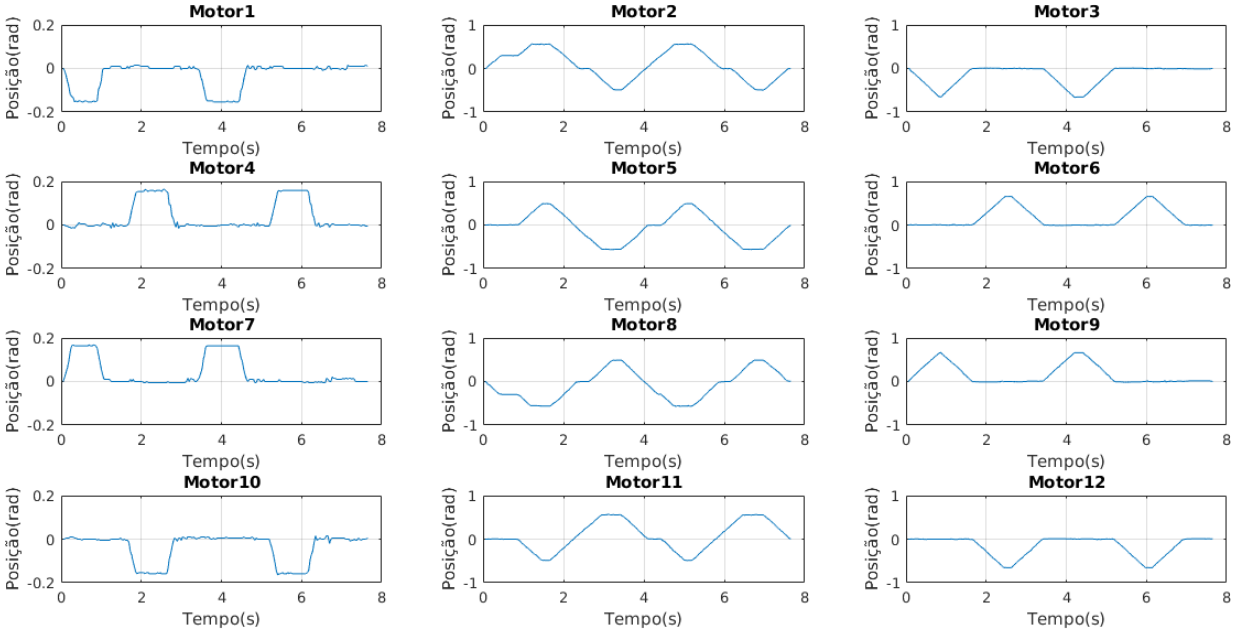


Figura 4.24: Posição dos motores para a segunda marcha encontradas na frente de Pareto, na otimização online.

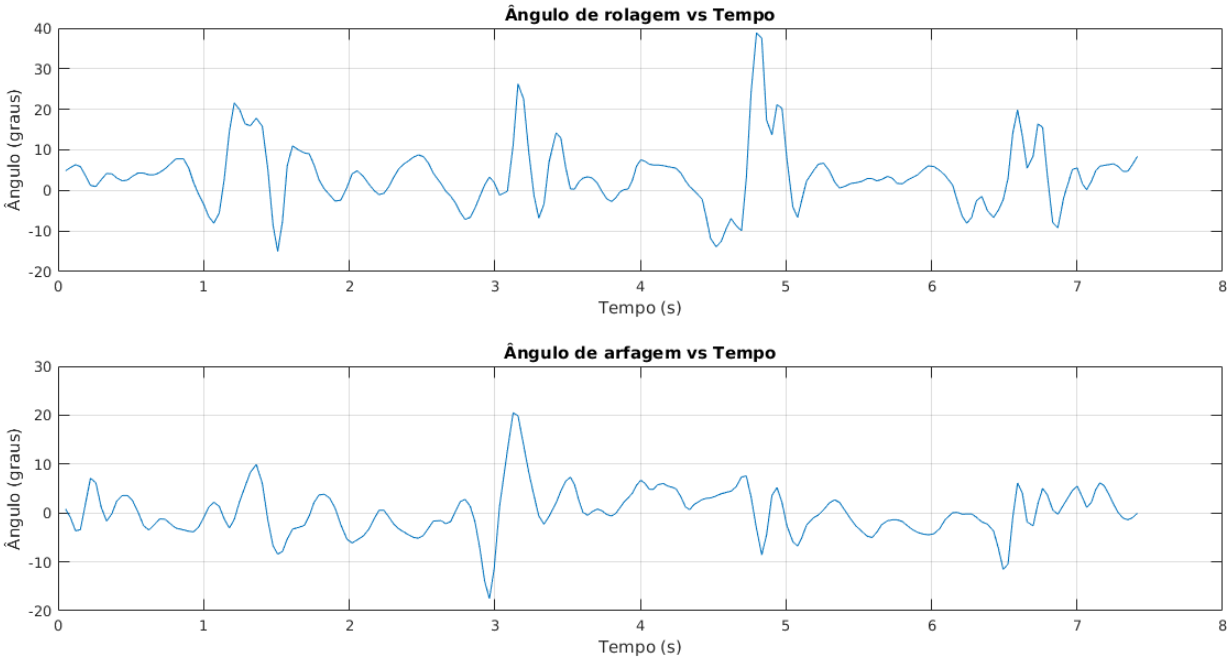


Figura 4.25: Valores dos ângulos de rolagem e arfagem ao longo do tempo para a segunda marcha encontrada na frente de Pareto, na otimização online.

Considerando que ambas as marchas percorreram distâncias semelhantes, tem-se com isso mais uma evidencia da maior velocidade da primeira marcha.

No que diz respeito ao balanço das marchas obtidas, a partir das Figuras 4.22 e 4.25 percebe-se que os ângulos máximos de rolagem das duas marchas são muito próximos. Quanto ao ângulo de arfagem, nota-se que os valor máximo na primeira marcha é menor que o da segunda. Evidenciando que a primeira marcha possui uma maior estabilidade quando comparada com a segunda marcha.

Capítulo 5

Conclusões

A começar se tratando das duas primeiras tentativas da otimização, espaço aberto de soluções e espaço restrito por poses, viu-se que o algoritmo não conseguiu encontrar uma marcha em que o robô se movesse para frente de forma estável e rápida. O que pode-se concluir é que, na primeira tentativa, talvez seja muito difícil para o algoritmo, em qualquer configuração, encontrar um conjunto de ângulos que corresponda a uma marcha para o robô, seja ela da forma que for, uma vez que o espaço de soluções é enorme. Mesmo com o acréscimo de indivíduos que sofreram mutação à população, visando uma maior diversidade, essa busca fica limitada à uma região apenas desse espaço. Já no segundo método, o maior impedimento à busca pela marcha factível foi o fato que a ferramenta utilizada não lidava com variáveis do tipo inteiro, e mudanças nos indivíduos, principalmente ao fim da otimização, aconteciam nas casas decimais, o que era irrelevante para esse caso, uma vez que os genes do indivíduo deveriam estar no intervalo $[1, 4] \in \mathbb{Z}$. Mas mesmo assim, o algoritmo encontrou ordens das poses que possuíam partes da marcha que se acreditava ser a ideal.

Mas, de um modo geral, as marchas adquiridas no final da terceira tentativa de otimização utilizando o V-REP e o Matlab, são factíveis ao robô simulado. Elas trocam entre si perdas e ganhos nas funções objetivos, no sentido que uma é mais veloz ao passo que a outra se mantém mais estável. Porém, ficou evidente a diferença entre a simulação e a vida real, uma vez que as marchas que compunham a frente de Pareto obtida via simulação muitas vezes sofriam dificuldade em serem executadas na plataforma real, seja por abrir de mais a passada, fazendo o robô simplesmente não conseguir se levantar mais, ou não levantar muito a pata, fazendo com que esta se arrastasse no chão, impedindo o movimento.

Dessa forma, surgiu a necessidade de realizar o treinamento online, visando superar as diferenças do robô simulado e do robô real. O resultado disso foi a obtenção de marchas factíveis ao robô real, que conseguiu se mover de forma relativamente simples, com eficiência e estabilidade. Naturalmente, métodos mais eficientes de se realizar essa otimização online podem ser aplicados, como em Zagal, Solar e Vallejos [20] ou em Belter e Skrzypczynski [19], uma vez que o utilizado só se tornou plausível devido ao número baixo de indivíduos por população, mas mesmo assim foram necessário mais de 500 indivíduos, ao londo de aproximadamente 100 gerações, a serem executadas

na plataforma e avaliados até o fim da otimização.

É importante dizer também que a escolha da otimização multiobjetiva para este trabalho foi satisfatória, uma vez que se desejava que o robô alcançasse boas distâncias em um curto período de tempo, mas também era desejável que ele fizesse isso de forma estável, a fim de não prejudicar sua partes mecânicas, que, como já se foi dito, são frágeis. Essas trocas são visíveis nos resultados das simulações, em que pode-se optar por uma marcha mais rápida, porém que faz com que o robô oscile com uma grande amplitude, ou por uma marcha mais lenta, diminuindo essa amplitude de oscilação, mas também aumentando o desvio da trajetória, por exemplo.

A estrutura mecânica se mostrou apta para o treinamento e execução das marchas, exceto por duas ressalvas. A primeira vem do desgaste das conexões com o servo. Notou-se que a atual forma de conexão está muito suscetível a folgas provenientes dos desgate mecânico. A segunda é o baixo atrito da extremidade da pata do robô com o solo, gerando deslizamentos que prejudicam a realização correta da marcha.

Com esse trabalho foi possível progredir com o que foi realizado por Porphirio e Santana [9]. Entretanto não foi possível implementar na arquitetura de controle proposta por Floriano [14], as marchas obtidas. Uma vez que os ajustes referentes ao tempo de amostragem do algoritmo deveriam ser feitos, para a obtenção de um resultado satisfatório na execução das marchas. A alteração no tempo de amostragem modificaria a dinâmica do sistema, gerando a necessidade de adaptação do controle de estabilidade para a frequência de amostragem requerida.

De forma geral, o trabalho realizado representa um progresso satisfatório, uma vez que deu continuidade aos trabalhos anteriores, propondo e implementando melhorias. Essas melhorias tiveram um enfoque no método de obtenção de marchas para a plataforma, utilizando treinamento online. Permitindo assim a geração de marchas que são executáveis pelo dispositivo.

5.1 Perspectivas futuras

Como perspectivas futuras tem-se melhorias na mecânica da plataforma. A mesma já possui algumas soluções paliativas implementadas em trabalhos anteriores mas que deixam de funcionar a longo prazo. Um dos primeiros desafios durante os testes na plataforma foi a folga em uma das juntas do robô, que o fazia cair em suas primeiras passadas na execução de uma marcha. Foi verificado que isso se deu ao desgaste de uma peça de nylon que se acoplava ao servomotor. Sendo assim, sugere-se utilizar outro material que suporte os torques exercidos pelo motor, ou alguma forma de conexão ajustada por meio de parafusos.

Além disso, outro problema enfrentado foi a falta de atrito entre a extremidade da pata e o chão, fazendo o robô deslizar durante a marcha. Porphirio e Santana [9] implementaram botas de borracha, mas isso não foi o suficiente. Assim, sugere-se encontrar alguma solução que sane esse problema, como por exemplo a criação de pés para o robô, ou a utilização de um material mais aderente para os calços.

Observou-se que as marchas não são suaves e possuem uma velocidade fixa, que fica mais lenta

ao se aproximar da posição desejada. Isso se dá pelo fato que é utilizado o modo posição no momento dos motores realizarem seu movimento. Ou seja, apenas se informa para qual posição o motor deve ir a cada instante da marcha. Acredita-se que se utilizado o modo velocidade, a marcha seria mais suave, e além disso, poderia variar a sua velocidade, dada alguma demanda do sistema de controle, como por exemplo, no caso em que o robô se desequilibra, e precisa se estabilizar.

Uma melhoria interessante a ser implementada é a integração das marchas obtidas com o controle de estabilidade proposto por Floriano [14]. Essa integração proporcionará uma maior robustez ao sistema, permitindo que ele reaja a distúrbios externos de forma a manter o seu equilíbrio e após isso, retomar a marcha proposta.

Como já foi dito, a otimização na plataforma teve cerca de 500 testes até a sua conclusão, o que torna isso, e certa forma, inviável. O que se pode ser feito é utilizar um método melhor, que mistura de fato a otimização na simulação e na plataforma real, seja atualizando os parâmetros do simulador, como em [20] ou rastreando pontos de interesse, como em [19]. Mas, outra alternativa, seria o desenvolvimento de uma plataforma de treinamento automática, que já realizasse as medidas desejadas, colocasse o robô na sua posição inicial e o impedisse de cair.

Um outro detalhe que não foi implementado, e é visto nas marchas de alguns animais, é a defasagem entre as patas dianteiras e traseiras. Isso pode ser implementando com um conjunto maior de posições a serem atingidos pelos servomotores ou utilizando o modo velocidade.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] BIGDOG - The First Advanced Rough-Terrain Robot. `https://www.bostondynamics.com/bigdog`. Acesso em: 25 de Junho de 2017.

[2] COTTA, G. H.; NETO, L. R. Realização de uma plataforma para estudo de robótica comportamental baseada em quadrúpedes. Trabalho de conclusão de curso de graduação em Engenharia Elétrica, 2006.

[3] BATISTA, G. F.; CARDOSO, I. F. Adequação de um sistema de locomoção de um robô quadrúpede para avaliação de algoritmos de aprendizagem. Trabalho de conclusão de curso de graduação em Engenharia Elétrica, 2007.

[4] SANTOS, J. H. d. S. Plataforma quadrúpede: uma nova estrutura para robô quadrúpede do lara. Trabalho de Graduação em Engenharia de Controle e Automação, 2016.

[5] SOUTO, R. F. Modelagem cinemática de um robô quadrúpede e geração de seus movimentos usando filtragem estocástica. Trabalho de conclusão de curso de graduação em Engenharia Elétrica, 2007.

[6] ANALOG DEVICES. Data sheet digital accelerometer. adxl345. 2009.

[7] DEB, K. Multi-objective optimization using evolutionary algorithms: an introduction. *Multi-objective evolutionary optimisation for product design and manufacturing*, p. 1–24, 2011.

[8] ROHMER, E.; SINGH, S. P.; FREESE, M. V-rep: A versatile and scalable robot simulation framework. In: IEEE. *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. [S.l.], 2013. p. 1321–1326.

[9] PORPHIRIO, C. d. F. S.; SANTANA, P. H. M. Geração de marchas para a plataforma quadrúpede utilizando algoritmo genético. Trabalho de Graduação em Engenharia de Controle e Automação, 2017.

[10] CALMON, A. d. P.; PINHEIRO, N. C.; FERREIRA, R. U. Desenvolvimento de um robô-cachorro comportamental: percepção e modelagem comportamental. Trabalho de conclusão de curso de graduação em Engenharia Elétrica, 2006.

[11] NOVAIS, N. A. d.; TOSCANO, R. A. Estudo de locomoção de uma plataforma quadrúpede utilizando sensoriamento inercial e geração de padrões de movimento. Trabalho de conclusão de curso de graduação em Engenharia Elétrica, 2007.

[12] RAMOS, E. G. Desenvolvimento da plataforma quadrúpede geração de software e eletrônica. Trabalho de Graduação em Engenharia de Controle e Automação, 2008.

[13] PAIVA, R. C. Utilizando osciladores neurais para gerar marcha de um robô quadrupede e robô humanoide. Trabalho de conclusão de curso de graduação em Engenharia Elétrica, 2012.

[14] FLORIANO, B. R. d. O. Desenvolvimento e interação de um sistema de controle de equilíbrio para um robô quadrúpede. Trabalho de conclusão de curso de graduação em Engenharia Elétrica, 2017.

[15] BUSS, S. R. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, v. 17, n. 1-19, p. 16, 2004.

[16] PEDLEY, M. Tilt sensing using a three-axis accelerometer. *Freescale semiconductor application note*, v. 1, p. 2012–2013, 2013.

[17] INTERLINK ELETRONICS. Fsr® 400 series data sheet.

[18] KELTON, W. D. et al. Simulation with arena. 2003. *Sydney: McGraw4Hill*, 2001.

[19] BELTER, D.; SKRZYPCZYŃSKI, P. A biologically inspired approach to feasible gait learning for a hexapod robot. *International Journal of Applied Mathematics and Computer Science*, Versita, v. 20, n. 1, p. 69–84, 2010.

[20] ZAGAL, J. C.; SOLAR, J. Ruiz-del; VALLEJOS, P. Back to reality: Crossing the reality gap in evolutionary robotics. *IFAC Proceedings Volumes*, Elsevier, v. 37, n. 8, p. 834–839, 2004.

6. DESCRIÇÃO DO CONTEÚDO DO CD

O CD possui os código em C++ executados na plataforma; os *scripts* em Matlab que realizam a integração síncrona com o V-REP e a avaliação dos indivíduos nas três tentativas, que geram os gráficos apresentados, que executam a otimização utilizando a plataforma física e que implementam o modelo da cinemática inversa; e o modelo do robô utilizado no V-REP para as simulações.