

INSTITUTO TECNOLÓGICO DE AERONÁUTICA



Luiz Filipe Martins Ramos
Bernardo Monteiro Rufino

Desenvolvimento de um classificador de postagens em rede social utilizando Processamento de Linguagem Natural

Trabalho de Graduação
2015

Computação

Número da CDU (tamanho 10)

Luiz Filipe Martins Ramos
Bernardo Monteiro Rufino

**Desenvolvimento de um classificador de postagens em rede social
utilizando Processamento de Linguagem Natural**

Orientador
Prof. Dr. Paulo André Lima de Castro

Engenharia de Computação

SÃO JOSÉ DOS CAMPOS
INSTITUTO TECNOLÓGICO DE AERONÁUTICA

2015

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Ramos, Luiz Filipe Martins
Rufino, Bernardo Monteiro

Desenvolvimento de um classificador de postagens em rede social utilizando Processamento de Linguagem Natural / Luiz Filipe Martins Ramos, Bernardo Monteiro Rufino

São José dos Campos, 2015

Número de folhas no formato 999f.

Trabalho de Graduação – Divisão de Ciência da Computação –

Instituto Tecnológico de Aeronáutica, 2015. Orientador: Prof. Dr. Paulo André Lima de Castro

1. Inteligência Artificial. 2. Processamento de Linguagem Natural. 3. Redes Bayesianas. I. Nome do segundo autor, se houver. II. Instituto Tecnológico de Aeronáutica. III. Desenvolvimento de um classificador de postagens em rede social utilizando Processamento de Linguagem Natural

REFERÊNCIA BIBLIOGRÁFICA

RAMOS, Luiz Filipe Martins; RUFINO, Bernardo Monteiro. Desenvolvimento de um classificador de postagens em rede social utilizando Processamento de Linguagem Natural. 2015. Total de folhas. Trabalho de Conclusão de Curso. (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS – (NEGRITO, TAMANHO 12)

NOME DO AUTOR: Luiz Filipe Martins Ramos; Bernardo Monteiro Rufino

TÍTULO DO TRABALHO: Desenvolvimento de um classificador de postagens em rede social utilizando Processamento de Linguagem Natural

TIPO DO TRABALHO/ANO: Graduação / 2015

É concedida ao Instituto Tecnológico de Aeronáutica permissão para reproduzir cópias deste trabalho de graduação e para emprestar ou vender cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desta monografia de graduação pode ser reproduzida sem a autorização do autor.

Luiz Filipe Martins Ramos
Pça Mal-do-Ar Eduardo Gomes, 50
12228-900 – São José dos Campos – SP

Bernardo Monteiro Rufino
Pça Mal-do-Ar Eduardo Gomes, 50
12228-900 – São José dos Campos - SP

**DESENVOLVIMENTO DE UM CLASSIFICADOR DE POSTAGENS EM REDE SOCIAL
UTILIZANDO PROCESSAMENTO DE LINGUAGEM NATURAL**

Essa publicação foi aceita como Relatório Final de Trabalho de Graduação

Luiz Filipe Martins Ramos
Autor

Bernardo Monteiro Rufino
Autor

Prof. Dr. Paulo André Lima de Castro (ITA)
Orientador

Prof. Dr. Carlos Henrique Costa Ribeiro (ITA)
Coordenador do Curso de Engenharia de Computação

São José dos Campos, _____ de _____ de _____

Agradecimentos

4.0141 There is a general rule by means of which the musician can obtain the symphony from the score, and which makes it possible to derive the symphony from the groove on the gramophone record, and, using the first rule, to derive the score again. That is what constitutes the inner similarity between these things which seem to be constructed in such entirely different ways. And that rule is the law of projection which projects the symphony into the language of musical notation. It is the rule for translating this language into the language of gramophone records.

Ludwig Wittgenstein. Tractatus Logico-Philosophicus.

Resumo

Abstract

Sumário

Agradecimentos	p. 6
Resumo	p. 8
Abstract	p. 9
Lista de Figuras	p. 14
Lista de Tabelas	p. 17
Lista de Algoritmos	p. 18
Lista de Abreviaturas	p. 20
1 Introdução	p. 21
1.1 Motivação	p. 21
1.2 Objetivos	p. 21
1.3 Organização do Texto	p. 21
2 Fundamentação Teórica	p. 22
3 Proposta de Classificador	p. 23
4 Instrumentos analisados	p. 24
4.1 Características de interesse	p. 24
4.1.1 Mecanismo de produção de som	p. 24
4.1.2 Timbre	p. 25

4.1.3	Tom	p. 25
4.1.4	Popularidade	p. 25
4.2	Instrumentos escolhidos	p. 25
4.3	Flauta doce	p. 26
4.3.1	Características	p. 26
4.3.2	Análise espectral	p. 27
4.3.3	Conclusões	p. 29
4.4	<i>Tin Whistle</i>	p. 29
4.4.1	Características	p. 29
4.4.2	Análise espectral	p. 30
4.4.3	Conclusões	p. 31
4.5	Pífano	p. 31
4.5.1	Características	p. 32
4.5.2	Análise espectral	p. 32
4.5.3	Conclusões	p. 32
4.6	Flauta transversal ocidental	p. 33
4.6.1	Características	p. 33
4.6.2	Análise espectral	p. 34
4.6.3	Conclusões	p. 35
4.7	Flauta transversal chinesa (<i>dizi</i>)	p. 36
4.7.1	Características	p. 36
4.7.2	Análise espectral	p. 36
4.7.3	Conclusões	p. 36
4.8	Ocarina	p. 37
4.8.1	Características	p. 37
4.8.2	Análise espectral	p. 37

4.8.3 Conclusões	p. 37
5 Algoritmos de Otimização	p. 39
5.1 Algoritmo Genético	p. 39
Cromossomo	p. 39
Função de <i>Fitness</i>	p. 39
Seleção	p. 40
Mutação	p. 40
<i>Crossover</i>	p. 40
Sobrevivência dos Mais Aptos	p. 40
5.2 Particle Swarm Optimization	p. 41
6 Tecnologias Utilizadas	p. 44
6.1 USARSim	p. 44
6.1.1 Nvidia PhysX	p. 44
6.1.2 UDK Editor	p. 46
6.1.3 UnrealScript	p. 46
6.1.4 Implementação de Robôs em USARSim	p. 48
6.1.5 Comunicação entre Agentes e USARSim	p. 48
6.2 Robonova-I	p. 49
7 Modelo de Simulação	p. 52
7.1 Modelos CAD	p. 52
7.2 Configuração no UDK Editor	p. 53
7.2.1 Implementação no USARSim	p. 53
8 Programa do Agente	p. 56
8.1 Comunicação	p. 57

8.2 Interface	p. 58
8.3 Controle	p. 59
9 Processo de Otimização	p. 60
9.1 Modelos de Caminhada	p. 60
9.2 Configuração dos Algoritmos de Otimização	p. 61
9.2.1 Configuração do Algoritmo Genético	p. 61
9.2.2 Configuração do Particle Swarm Optimization	p. 62
9.3 Montagem do Experimento Simulado	p. 64
9.4 Resultados e Discussões	p. 66
10 Transferência para o Robô Real	p. 73
10.1 Desafios Envolvidos	p. 73
10.2 Processo de Transferência	p. 73
11 Validação com Outro Robô	p. 78
11.1 RoboCup 3D Soccer Simulation League	p. 78
11.2 Implementação	p. 80
11.3 Processo de Otimização	p. 81
12 Conclusões e Trabalhos Futuros	p. 84
12.1 Conclusão	p. 84
12.2 Trabalhos Futuros	p. 84
Referências	p. 86

Lista de Figuras

1	Espectro para a nota C5 tocada em uma flauta doce Yamaha YRS-23Y obtido a partir da FFT em 1 segundo de gravação.	p. 28
2	Espectro para um G5 (azul) e um G6 (verde) tocados em uma flauta doce Yamaha YRS-23Y obtido a partir da FFT em 1 segundo de gravação.	p. 28
3	Espectro para a nota D5 tocada em um <i>whistle</i> Clarke SBDC obtido a partir da FFT em 1 segundo de gravação.	p. 30
4	Espectro para um G5 (verde) e um G6 (azul) tocados em um <i>whistle</i> Clarke SBDC obtido a partir da FFT em 1 segundo de gravação.	p. 31
5	Espectro para a nota C5 tocada em um pífano Yamaha YRF-21 obtido a partir da FFT em 1 segundo de gravação.	p. 33
6	Espectro para a nota C4 tocada em uma flauta transversal Hallelu HFL-200 obtido a partir da FFT em 1 segundo de gravação.	p. 34
7	Espectro para um G4 (verde) e um G6 (azul) tocados em uma flauta transversal Hallelu HFL-200C obtido a partir da FFT em 1 segundo de gravação.	p. 35
8	Espectro para um A4 (azul) e um A5 (verde) tocados na ocarina escolhida, obtido a partir da FFT em 1 segundo de gravação.	p. 38
9	Junta de revolução.	p. 45
10	Diferenças entre modelos físicos gerados com K-DOF (os modelos físicos são apresentados em <i>wireframe</i>).	p. 47
11	Robonova-I. Extraído de (18).	p. 49
12	Juntas do Robonova-I. Extraído de (18).	p. 50
13	Servomotores Hitec HSR-8498HB. Extraído de (18).	p. 50
14	Detalhe das costas do Robonova-I mostrando a placa MR-C3204 instalada. Extraído de (18).	p. 51

15	Modelo completo montado do Robonova-I.	p. 52
16	Modelo completo explodido do Robonova-I.	p. 53
17	Juntas implementadas no modelo de simulação.	p. 54
18	Robonova-I simulado dentro do USARSim! (USARSim!).	p. 55
19	Camadas do programa do agente.	p. 56
20	ExampleMap.	p. 64
21	Sistema de coordenadas no USARSim.	p. 65
22	Resultados da otimização com Modelo Simples e AG! (AG!).	p. 67
23	Resultados da otimização com Modelo com Movimento de Braços e AG!	p. 68
24	Resultados da otimização com Modelo Complexo e AG!	p. 68
25	Resultados da otimização com Modelo Simples e PSO! (PSO!).	p. 69
26	Resultados da otimização com Modelo com Movimento de Braços e PSO!	p. 69
27	Resultados da otimização com Modelo Complexo e PSO!	p. 70
28	Sequência de imagens para ilustrar a caminhada aprendida com Modelo Simples e PSO!	p. 71
29	Sequência de imagens para ilustrar a caminhada aprendida com Modelo com Movimento de Braços e PSO!	p. 72
30	Sequência de imagens para ilustrar a caminhada aprendida com Modelo Complexo e PSO!	p. 72
31	Sequência de imagens para ilustrar a caminhada final do Robonova real. A taxa de quadros usada para gravação foi de 30 quadros por segundo.	p. 76
32	Ambiente do experimento de medida de velocidade do robô real.	p. 77
33	Robô Nao simulado chutando uma bola no SimSpark! (SimSpark!).	p. 78
34	Esquema das juntas do Nao simulado do SimSpark!	p. 79
35	Resultados da otimização com PSO! realizada no ambiente da Soccer 3D! (Soccer 3D!).	p. 82

Lista de Tabelas

1	Especificação dos servomotores Hitec HSR-8498HB. Extraído de (18).	p. 49
2	Especificação da placa MR-C3024. Extraído de (6).	p. 51
3	Peças implementadas como objetos do tipo “Part”.	p. 54
4	Mapeamento das juntas dos modelos teóricos para as do Robonova (veja a Figura 17 para nomenclatura utilizada para as juntas do Robonova).	p. 61
5	Limites para os valores das contantes dos modelos de caminhada.	p. 62
6	Valores dos parâmetros usados nas execuções de AG!	p. 62
7	Valores dos parâmetros usados nas execuções de PSO!	p. 64
8	Limites para os valores das contantes dos modelos de caminhada.	p. 66
9	Resultados dos Experimentos.	p. 67
10	Limites para os valores das contantes dos modelos de caminhada.	p. 71
11	Comparação entre diversas caminhadas para o Robonova-I.	p. 77
12	Valores dos parâmetros da execução de PSO! para otimização dentro do domínio da Soccer 3D!	p. 81
13	Limites do espaço de busca para no caso da otimização realizada dentro da domínio da Soccer 3D!	p. 81
14	Parâmetros aprendidos para o Modelo Complexo executado no Nao simulado do SimSpark!	p. 82
15	Resultados de medidas de distâncias percorridas no sentido positivo do eixo X. Realizou-se 10 experimentos de 15 segundos para cada caminhada.	p. 83

Lista de Algoritmos

1	Pseudocódigo do Algoritmo Genético.	p. 41
2	Pseudocódigo do <i>Particle Swarm Optimization</i>	p. 43
3	Subrotina que simula “choques” das partículas com os limites do espaço de busca.	p. 63

Lista de Listagens

Lista de Abreviaturas

FFT Transformada rápida de Fourier

1 Introdução

1.1 Motivação

1.2 Objetivos

1.3 Organização do Texto

2 *Fundamentação Teórica*

3 Proposta de Classificador

4 *Instrumentos analisados*

Todo programa de computador é, essencialmente, um algoritmo que transforma um sinal de entrada em um sinal de saída, portanto, entender os sinais de entrada é fundamental para o desenvolvimento e a depuração de qualquer programa de computador.

No desenvolvimento deste trabalho diversos limites terão de ser impostos aos sinais sonoros de entrada. Serão estabelecidos limites, por exemplo à quantidade máxima de notas tocadas por segundo, à nota mais grave e mais alta, à intensidade mínima para um som ser considerado uma nota, etc. Entretanto, dada a multíitude de instrumentos existentes, é virtualmente impossível estabelecer limites para os exemplos dados que satisfaçam aos sinais gerados por todos os instrumentos.

Assim, embora o escopo deste trabalho seja a transcrição musical automática de instrumentos monofônicos temperados, independentemente da forma com que o som é gerado neles, as análises feitas a partir deste capítulo serão voltadas a um sub-conjunto específico de instrumentos. Espera-se que este sub-conjunto seja representativo o bastante para que o algoritmo final desenvolvido seja também aplicável a diversos outros instrumentos. Este capítulo apresenta os instrumentos escolhidos e justifica as escolhas.

4.1 Características de interesse

Ao escolher os instrumentos, algumas características desejadas foram levantadas, essas são apresentadas a seguir:

4.1.1 Mecanismo de produção de som

Deseja-se que os instrumentos analisados possuam mecanismos de produção de som simples e para os quais já existam modelos matemáticos confiáveis. Isso tornará possível embasar as decisões tomadas em modelos matemáticos que representam o mundo real, e

não apenas em suposições feitas com base em teoria musical.

4.1.2 Timbre

O timbre é, talvez, a característica mais marcante de um instrumento, e pode variar muito mesmo entre instrumentos de uma mesma família. Por esses motivos, deseja-se que os instrumentos analisados possuam timbres consideravelmente distintos. Isso ajudará a elaborar algoritmos que funcionem em uma gama diversa de instrumentos.

4.1.3 Tom

Em um único instrumento, uma mesma nota pode ser tocada de diversas formas, ou tons, dependendo da sensação que o músico deseja transmitir. Por exemplo, uma nota pode ser suave ou agressiva; viva ou fria. É interessante que os instrumentos escolhidos sejam capazes de gerar uma vasta gama de tons, também para que os algoritmos elaborados sejam pouco específicos aos instrumentos escolhidos.

4.1.4 Popularidade

Para que os algoritmos desenvolvidos possuam aplicações reais, é interessante que eles funcionem bem para instrumentos populares, portanto faz sentido escolher instrumentos que sejam conhecidos e bastante utilizados.

4.2 Instrumentos escolhidos

Com base nas características levantadas, a classe dos instrumentos de sopro foi escolhida. Além de populares, a produção de som por esses instrumentos já é um processo bem conhecido e que pode ser aproximado por modelos majoritariamente lineares (9). Estes instrumentos também possuem timbres bastante característicos e distintos uns dos outros. Ainda, instrumentos como a flauta transversal permitem ao instrumentista realizar uma vasta gama de tonalidades dependendo de sua embocadura, controle de respiração, etc.

Dentro da classe dos instrumentos de sopro, os seguintes instrumentos foram selecionados:

1. Flauta doce

2. *Tin whistle*
3. Pífano
4. Flauta transversal ocidental
5. Flauta transversal chinesa (*dizi*)
6. Ocarina

Cada instrumento será apresentado partindo de uma breve descrição histórica, seguida de características de interesse para o trabalho e de uma análise espectral de algumas notas tocadas no instrumento. Nota-se que o espectro gerado por cada instrumento pode variar de acordo com a habilidade do instrumentista e com as técnicas que ele deseja executar (8). Entretanto, consideramos que essas variações são desprezíveis quando comparadas às causadas pelo uso de instrumentos diferentes e, portanto, podem ser desconsideradas nesse capítulo, cujo propósito é apenas comparar o espectro de diferentes instrumentos e discutir os resultados esperados para cada um.

4.3 Flauta doce

A flauta doce é um instrumento de origem medieval, sua popularidade em apresentações diminuiu consideravelmente com o surgimento de instrumentos voltados para orquestras. Hoje, por ser um instrumento relativamente simples e fácil de se tocar, a flauta doce é amplamente utilizada no ensino de música.

Neste trabalho, será utilizada uma flauta doce soprano germânica afinada em dó maior (modelo Yamaha YRS-23Y). Este modelo possui oito buracos, é feito de plástico e é amplamente utilizado por iniciantes. O termo "soprano" remete ao fato da nota mais grave produzida por esta flauta ser um C5, enquanto o termo "germânica" remete à forma com que os buracos na flauta devem ser utilizados para produzir notas específicas.

4.3.1 Características

A flauta doce utilizada é capaz de produzir 3 oitavas cromáticas¹ a partir do C5.

¹A maioria dos instrumentos de sopro não possui um limite superior para as notas produzidas (9). Os limites apresentados aqui são os dados pelo fabricante do modelo utilizado, e refletem o que é pedido no repertório clássico.

Uma característica que facilitará este trabalho é que a estrutura geométrica da flauta doce é totalmente controlada pelo fabricante (5). Seu bocal é feito de forma a direcionar o jato de ar produzido pelo flautista contra uma superfície afiada. Com isso, o flautista não precisa se preocupar em direcionar o jato de ar gerado, e tem que apenas controlar a pressão exercida. O bocal da flauta doce também é projetado para facilitar o controle da pressão exercida e, como resultado, o flautista é capaz de produzir duas oitavas com modificações mínimas no jato de ar que produz, e sem ter de modificar sua geometria. Isso também significa que a tonalidade da flauta doce varia pouco, embora fatores como ressonância na cavidade oral do instrumentista ainda permitam a ele leves modificações no tom (5).

Por outro lado, a flauta doce não possui nenhum mecanismo para ajudar o flautista a pressionar os buracos em seu corpo. Assim, embora seja relativamente fácil produzir e alternar entre as notas da escala de dó maior, produzir uma escala cromática é uma tarefa difícil que requer muito controle do posicionamento dos dedos, sendo necessário encobrir certos buracos levemente, com um alto grau de precisão. Ainda, pode ser necessário alternar muitos dedos entre notas cromáticas, o que facilmente produz estados transientes desagradáveis em frases tocadas em *legato*.

4.3.2 Análise espectral

A figura 1 mostra o espectro obtido em 1 segundo de gravação da nota mais grave produzida pela flauta doce (C5). Observa-se que na flauta doce o componente fundamental é o mais acentuado, sendo seguido por quatro harmônicos cuja amplitude decai em cerca de 10dB/harmônico. A partir do quinto harmônico o comportamento deixa de seguir um padrão simples. Ainda, o máximo em 521Hz indica que a flauta doce utilizada está afinada com o A4 em cerca de 438Hz, o que sugere a afinação padrão de 440Hz com uma pequena margem de erro que pode ter sido causada, por exemplo, por diferenças de temperatura. Por fim, é interessante observar que os harmônicos da flauta doce seguem a série:

$$H_n = H_0(n + 1) \quad (4.1)$$

Sendo H_0 o harmônico fundamental. Outros instrumentos podem seguir a série:

$$H_n = H_0(2n + 1) \quad (4.2)$$

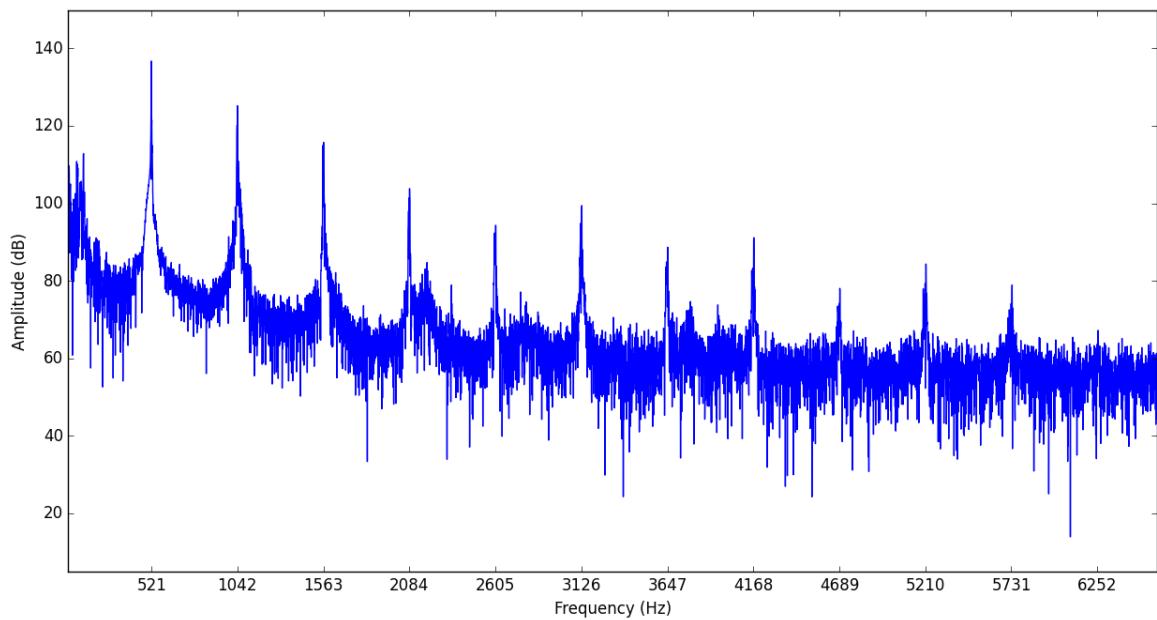


Figura 1: Espectro para a nota C5 tocada em uma flauta doce Yamaha YRS-23Y obtido a partir da FFT em 1 segundo de gravação.

Uma explicação física para os instrumentos de sopro gerarem uma série ou a outra pode ser encontrada em (9).

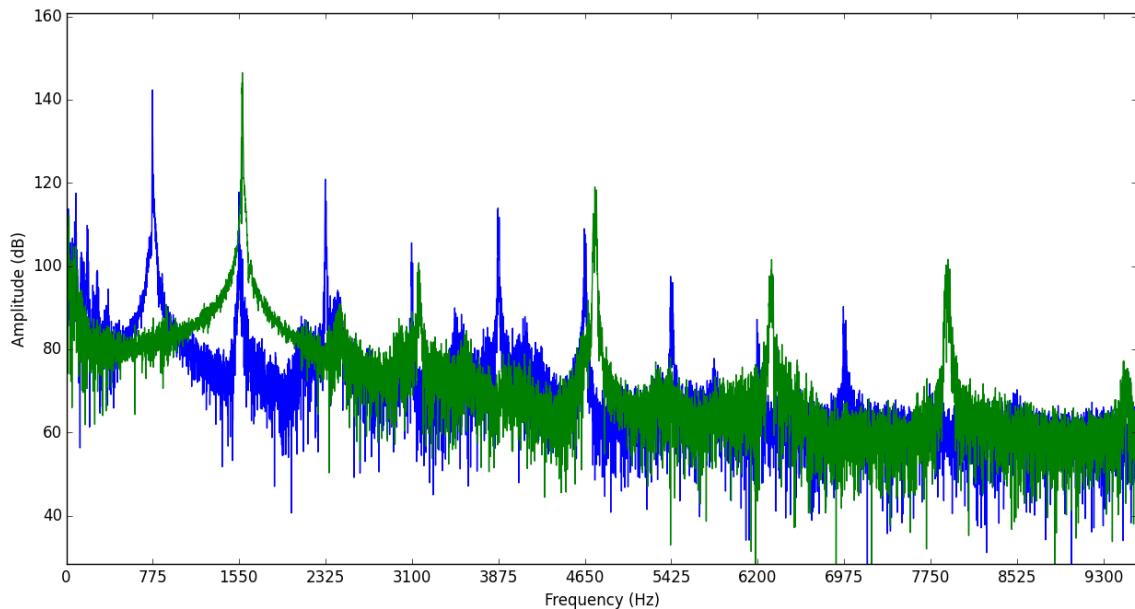


Figura 2: Espectro para um G5 (azul) e um G6 (verde) tocados em uma flauta doce Yamaha YRS-23Y obtido a partir da FFT em 1 segundo de gravação.

Já a figura 2 compara o espectro de um G5 e de um G6 uma oitava acima. É

interessante observar que a amplitude relativa entre os harmônicos é bastante diferente da amplitude relativa para a nota C5 da figura 1, sendo o segundo harmônico mais intenso que o primeiro. Apesar disso, tanto o G5 quanto o G6 possuem amplitudes relativas semelhantes entre si. Outro fator interessante é que os harmônicos produzidos não são posicionados exatamente nas mesmas frequências.

Por fim, observamos que a nota G6 tem uma leve tendência de gerar componentes posicionados nos harmônicos de um G5 que não são seus próprios harmônicos. Isso pode ser visto na frequência de (2325Hz), que é um múltiplo da frequência fundamental do G5 775Hz , mas não é um múltiplo do G6 (1550Hz). Isto dificultará a distinção dessas duas notas no algoritmo de detecção de frequência fundamental.

4.3.3 Conclusões

Espera-se que o programa desenvolvido nesse trabalho seja capaz de identificar muito bem músicas em dó maior tocadas nessa flauta, mas que encontre dificuldades em músicas cromáticas, especialmente se houver muitas frases em *legato*. Espera-se que essa dificuldade seja mais relacionada à erros cometidos pelo instrumentista do que de erros do algoritmo.

4.4 *Tin Whistle*

O *tin whistle* é um instrumento de sopro popular na música celta, seu corpo costuma ser feito de estanho, daí o nome. A música popular celta é conhecida por ritmos rápidos e frases com muitos ornamentos, e isso se reflete nas características do *whistle*.

Nesse trabalho será utilizado um *whistle* Clarke SBDC afinado em Ré maior, que possui 6 buracos.

4.4.1 Características

O *whistle* utilizado é capaz de produzir 2 oitavas nas escalas de Ré maior ou Sol maior, a partir de um D5.

O *whistle* é bastante semelhante à flauta doce, mas é um instrumento de confecção mais simples e seu bocal ajuda muito menos o instrumentista a controlar o jato de ar, sendo mais fácil gerar notas desafinadas ou desgradáveis. Por outro lado, os buracos no

corpo do *whistle* são mais simples, e é muito fácil transitar entre as notas na escala para a qual ele foi afinado. Isso permite ao instrumentista tocar passagens muito rápidas e com diversos ornamentos sem gerar muitos transientes desagradáveis.

4.4.2 Análise espectral

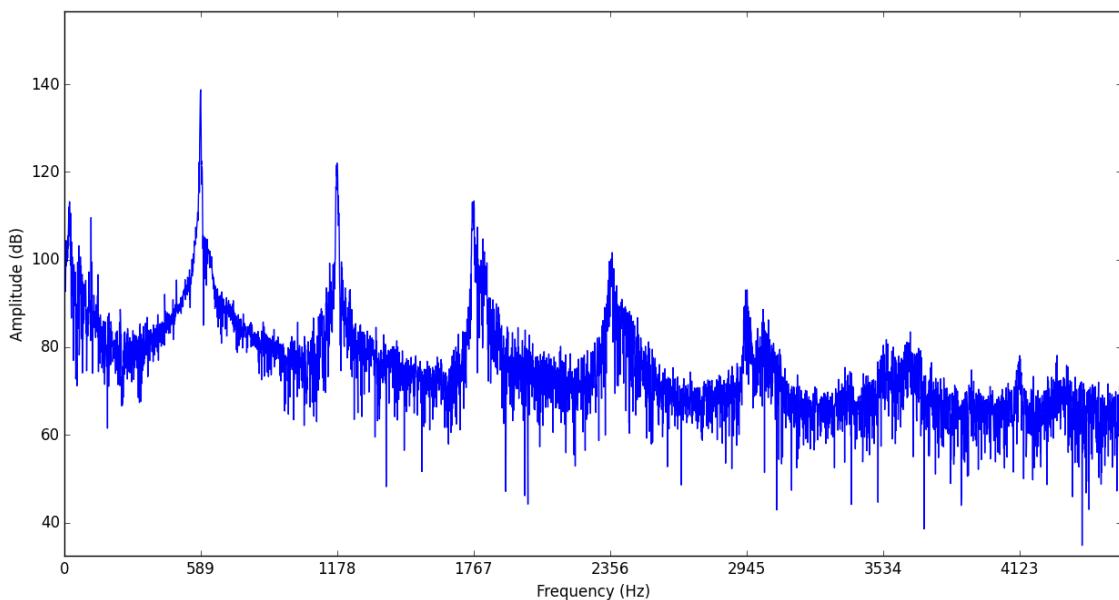


Figura 3: Espectro para a nota D5 tocada em um *whistle* Clarke SBDC obtido a partir da FFT em 1 segundo de gravação.

Na figura 3 podemos ver que, relativamente à flauta doce, o *whistle* gera menos harmônicos significativos. Além disso, os picos nas frequências dos harmônicos ”vazam” mais para as frequências vizinhas. Isso pode ser atribuído ao bocal mais primitivo do *whistle*, que direciona menos o jato de ar produzido pelo instrumentista, fazendo com que parte da energia do jato seja desperdiçada e gere componentes indesejados.

Na figura 4 podemos comparar a nota G6 com a nota G5 no *whistle*. É interessante observar que na nota G6 os ruídos são muito mais intensos do que na G5, lembramos que no *whistle* é necessário aumentar consideravelmente a intensidade do jato de ar para trocar entre oitavas, como o bocal do *whistle* não faz nenhum ajuste no jato de ar, a maior intensidade causa um desperdício maior de energia, que se reflete no ruído.

Adicionalmente, nota-se que a nota G6 gera um componente harmônico na frequência fundamental da nota G5, mas com amplitude reduzida. Isso é esperado pois as duas notas possuem o mesmo dedilhado e, portanto, a forma geométrica do instrumento permite que

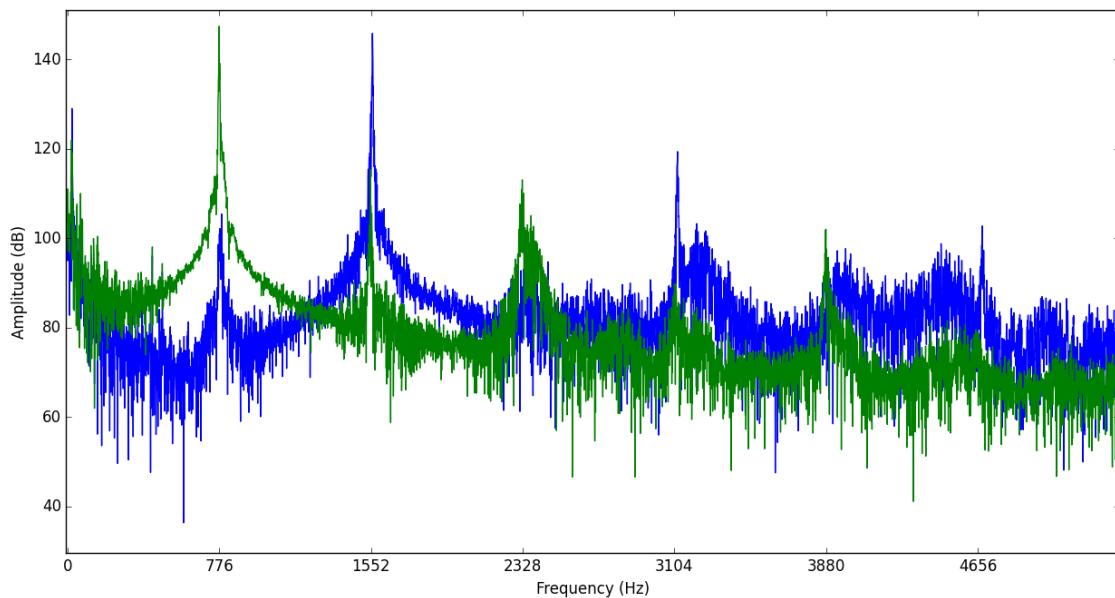


Figura 4: Espectro para um G5 (verde) e um G6 (azul) tocados em um *whistle* Clarke SBDC obtido a partir da FFT em 1 segundo de gravação.

esse componente seja formado. Esse detalhe pode causar erros na classificação correta da oitava de uma nota tocada.

4.4.3 Conclusões

Relativamente a flauta doce, espera-se que nos sinais provenientes do *whistle* seja mais fácil identificar passagens rápidas e com ornamentos, mas que para músicas lentas a precisão seja menor, devido a maior dificuldade em se controlar o jato de ar. Ainda, espera-se uma dificuldade adicional nas notas mais agudas, que possuem mais ruído e componentes nas mesmas frequências das notas correspondentes na oitava inferior.

4.5 Pífano

O pífano é uma flauta transversal pequena com timbre estridente. É um instrumento bastante simples, sendo desprovido de um bocal e, portanto, delegando o controle do jato de ar totalmente ao instrumentista.

Nesse trabalho será utilizado um pífano Yamaha YRF-21 afinado em Dó maior, que é feito de plástico e possui 9 buracos em seu corpo, sendo um destinado ao sopro.

4.5.1 Características

O pífano utilizado é capaz de produzir cromaticamente as notas entre C5 e E7.

O pífano escolhido é bastante semelhante à flauta doce, tanto em material quanto na posição dos dedilhados. A maior diferença é o fato do pífano ser desprovido de um bocal e de ser empunhado transversalmente. A ausência de um bocal permite ao instrumentista controlar sua embocadura de forma a variar muito o tom do instrumento. Assim, as dificuldades apresentadas para a flauta doce se mantém, sendo acrescentadas dificuldades em identificar precisamente as notas, que podem sofrer muitas variações de tom.

Nota-se, em especial, que a transição entre oitavas no pífano é muito mais difícil de se realizar do que nos instrumentos apresentados anteriormente. Tanto na flauta doce quanto no *whistle* essa transição pode ser feita usando um dedilhado levemente diferente e acelerando um pouco o jato de ar. Entretanto, como o pífano não possui um bocal para direcionar o jato de ar, o instrumentista precisa alterar consideravelmente a sua embocadura de forma a gerar um jato mais rápido que continua atingindo a superfície do instrumento no ângulo correto e com a largura adequada. Esse aspecto será analisado espectralmente a seguir.

4.5.2 Análise espectral

Na figura 5 temos o espectro para um C5, a nota mais grave atingida pelo pífano. É possível reconhecer cerca de 8 harmônicos, sendo os 3 primeiros bem mais significativos. Ainda, nota-se que os harmônicos a partir do terceiro são acompanhados de amplitudes consideráveis em suas redondezas, de forma semelhante ao que ocorre no *whistle*, e contrária à flauta doce.

4.5.3 Conclusões

Relativamente a flauta doce, espera-se que nos sinais provenientes do pífano seja mais difícil identificar precisamente a frequência fundamental das notas, e que ocorram erros relativos à oitava da nota tocada. Esses problemas se reduzem de acordo com a habilidade do instrumentista.

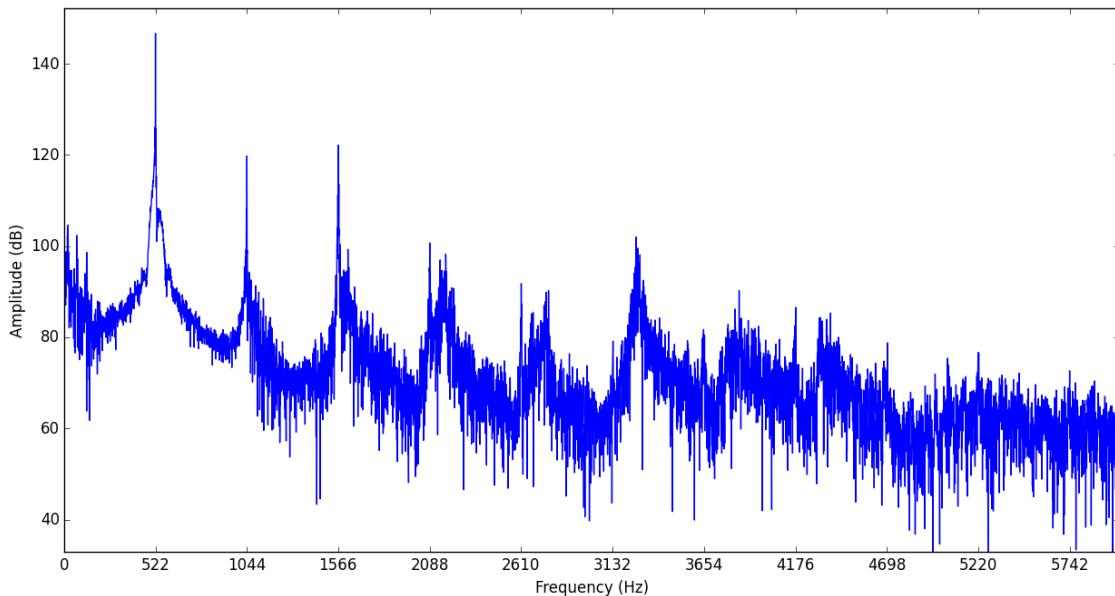


Figura 5: Espectro para a nota C5 tocada em um pífano Yamaha YRF-21 obtido a partir da FFT em 1 segundo de gravação.

4.6 Flauta transversal ocidental

A flauta transversal ocidental é um dos instrumentos de sopro mais populares em orquestras. Foi feito a partir da modificação de flautas transversais medievais (semelhantes ao pífano). As principais melhorias introduzidas foram um corpo de metal, que é capaz de gerar notas muito mais altas e que podem ser carregadas sobre os demais instrumentos de uma orquestra, e um mecanismo de botões e molas que deixa os dedilhados muito mais confortáveis, pois os botões podem ser projetados para ocuparem posições confortáveis sem prejudicar a afinação do instrumento, além de facilitarem a tarefa de encobrir totalmente os buracos.

Nesse trabalho será utilizada uma flauta transversal afinada em Dó maior, modelo Hallelu HFL-200.

4.6.1 Características

A flauta transversal utilizada é capaz de produzir cromaticamente três oitavas a partir do C4.

Em relação aos instrumentos apresentados anteriormente, nota-se que a flauta transversal escolhida é capaz de produzir notas mais graves. Analisar essas notas será uma

tarefa mais difícil, primeiro porque a diferença absoluta da frequência entre duas notas separadas por um semitom será menor e, segundo, porque a frequência menor aumenta o tempo em que o instrumento se encontra em estados transitórios (9).

Em relação à flauta doce e ao *whistle*, a ausência de um bocal dificulta a identificação das notas tocadas, pois há uma variedade muito maior de tons que podem ser produzidos, assim como no pífano. Por outro lado, a construção mais cuidadosa do instrumento e os botões mecânicos ajudarão a produzir tons mais limpos do que no pífano, e facilitarão a identificação de notas em passagens rápidas e com *legatos*.

4.6.2 Análise espectral

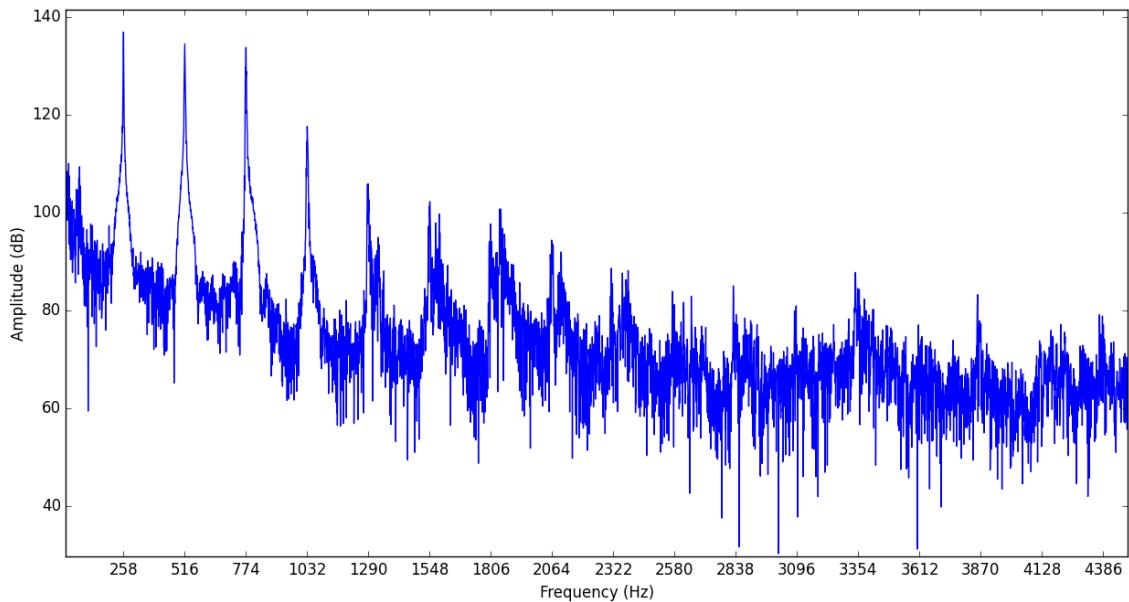


Figura 6: Espectro para a nota C4 tocada em uma flauta transversal Hallelu HFL-200 obtido a partir da FFT em 1 segundo de gravação.

Na figura 6 vemos o espectro para um C4, a nota mais grave produzível na flauta transversal. Já na figura 7 vemos as notas G4 e G6 na mesma flauta.

Primeiramente, observamos que na nota C4 os três primeiros harmônicos possuem amplitude bastante próxima. A nota C4 na flauta transversal é relativamente difícil de ser tocada, sendo necessário um controle bastante preciso do jato de ar produzido. Pequenos erros no jato podem transformar a nota em um C5, o que justifica a amplitude relativamente alta do segundo harmônico (que seria o fundamental para um C5). Além disso, os harmônicos mais altos não possuem picos tão bem definidos quanto os mais

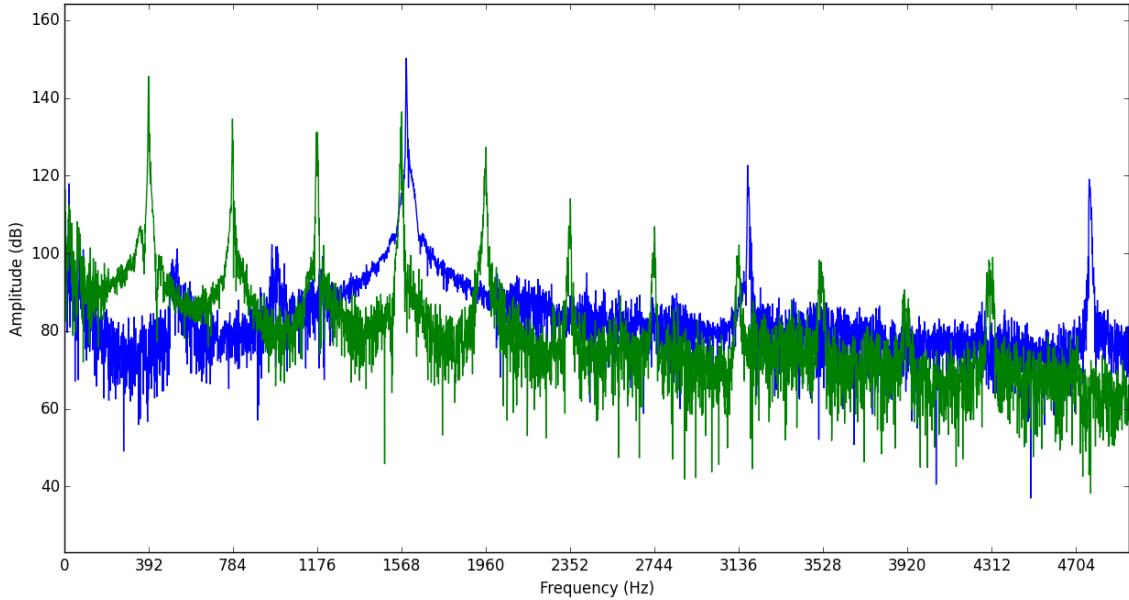


Figura 7: Espectro para um G4 (verde) e um G6 (azul) tocados em uma flauta transversal Hallelu HFL-200C obtido a partir da FFT em 1 segundo de gravação.

baixos, o que indica um som menos limpo do que o da flauta doce, por exemplo.

Comparando a figura 6 com a 7, podemos ver que a nota G4, apesar de estar na mesma oitava que a C4, não apresenta as dificuldades apresentadas anteriormente, possuindo um espectro parecido com os produzidos pela flauta doce, em que cada harmônico possui um pico bem definido no espectro.

Por fim, vemos que a nota G6, mesmo estando duas oitavas acima da nota G4, não apresenta um ruído significativamente maior e nem gera componentes sub-harmônicos significativos (dificuldades presentes, por exemplo, nas notas altas do *whistle*). Nota-se que a flauta transversal é um instrumento feito para ser ouvido sob uma orquestra e é, portanto, mais potente. A energia perdida no jato de ar produzido pelo flautista é menos significativa. Além disso, a ausência de um bocal permite ao flautista ajustar livremente a embocadura, de forma a aproveitar bem a energia do jato de ar e controlar os componentes sub-harmônicos.

4.6.3 Conclusões

Espera-se que a identificação das notas na flauta transversal seja mais fácil que nos demais instrumentos a partir da segunda oitava (C5-C7), desde que o flautista seja hábil o bastante para controlar sua embocadura corretamente. Na oitava inferior (C4-B4) podem

haver mais erros devido à estados transientes e à maior proximidade entre as frequências fundamentais. Por fim, espera-se ser mais fácil identificar passagens cromáticas na flauta transversal do que nos outros instrumentos, devido aos botões mecânicos que facilitam os dedilhados para notas acidentais.

4.7 Flauta transversal chinesa (*dizi*)

O *dizi*, comumente conhecido como flauta transversal chinesa, é um instrumento popular na china, sendo utilizado tanto em música popular quanto em óperas e orquestras chinesas. Costuma ser feito artesanalmente a partir de uma única peça de bambu, que é furada nas posições desejadas para os dedilhados.

Nesse trabalho será utilizado um *dizi* artesanal de baixa qualidade importado da china.

4.7.1 Características

O *dizi* utilizado é capaz de produzir um subconjunto específico de notas entre E5 e B7. Seu temperamento não gera notas separadas por semitons, o que dificultará muito a tarefa de classificá-las.

Em relação ao pífano, a produção artesanal do *dizi* faz com que as notas produzidas possuam várias imperfeições, tanto na afinação, pela posição dos buracos, quanto no tom, pela maior dificuldade em se encobrir totalmente os buracos. Ainda, o *dizi* escolhido possui 11 buracos em seu corpo, e seus dedilhados são pouco confortáveis, sendo difícil tocar passagens rápidas com precisão

Por não ser muito adequado à música ocidental, ele será utilizado brevemente para testar os tons produzidos por instrumentos pouco comuns na música ocidental, e com afinações exóticas.

4.7.2 Análise espectral

4.7.3 Conclusões

Espera-se que a identificação correta das notas tocadas no *dizi* seja bastante difícil e que seja feita com sucesso apenas para músicas mais lentas, a menos que o instrumentista

seja muito habilidoso. Ainda, espera-se que seja difícil obter as notas exatas devido à afinação não temperada do *dizi*.

4.8 Ocarina

A ocarina é um instrumento de sopro globular geralmente feito de cerâmica. É um dos instrumentos mais antigos conhecidos, e pode ter diversas formas.

Nesse trabalho será utilizada uma ocarina modelo "TOTMC Legend of Zelda Ocarina of Time Triforce Link TOLO248" de 12 buracos afinada em Dó maior.

4.8.1 Características

A ocarina escolhida é capaz de produzir cromaticamente as notas no intervalo A4-F6.

O corpo globular da ocarina dificulta a criação de harmônicos, sendo a frequência fundamental muito mais intensa do que as mais altas. Entretanto, espera-se que a manufatura em cerâmica apresente mais imperfeições do que os instrumentos em plástico e metal, o que pode ser constatado, por exemplo, no formato irregular dos buracos. Também nota-se que os dedilhados na ocarina são pouco confortáveis, sendo muito fácil não encobrir corretamente alguns dos buracos.

4.8.2 Análise espectral

Na figura 8 comprovamos que apenas o harmônico fundamental é significativo na ocarina.

4.8.3 Conclusões

Como na ocarina apenas a frequência fundamental é considerável, espera-se que as notas identificadas estejam na oitava correta. Entretanto, pode haver erros na identificação de notas separadas por semitons, pois a decisão terá de ser tomada com a informação de um único harmônico.

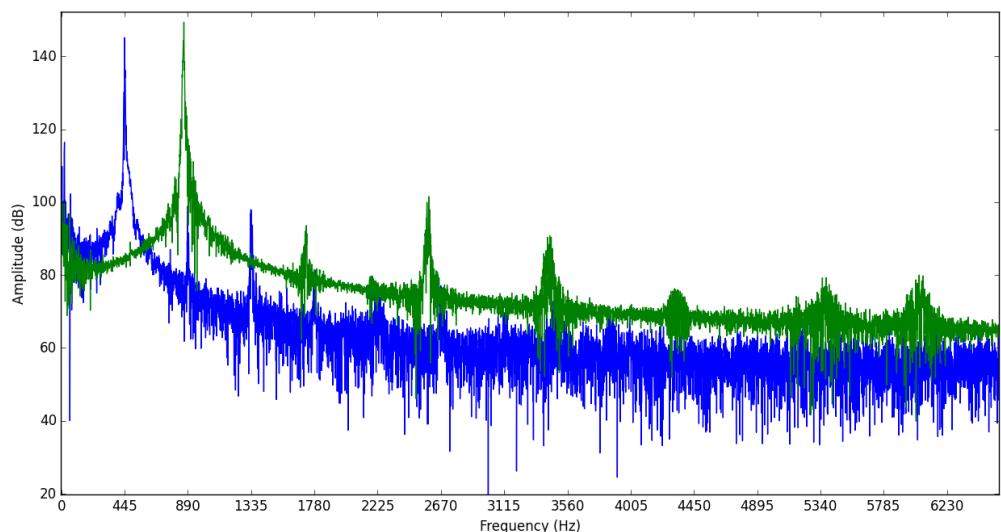


Figura 8: Espectro para um A4 (azul) e um A5 (verde) tocados na ocarina escolhida, obtido a partir da FFT em 1 segundo de gravação.

5 Algoritmos de Otimização

5.1 Algoritmo Genético

Algoritmos Genéticos (**AG!**s) são inspirados na Teoria da Evolução de Darwin (12). Nestas técnicas, mantém-se uma população de soluções candidatas, chamadas cromossomos, que “evoluem” guiadas por uma medida de qualidade (função de *fitness*) através da aplicação de operações inspiradas pela evolução natural: seleção, mutação, *crossover* e sobrevivência dos mais aptos. Inicialmente, a população é composta por S_i (parâmetro do AG) cromossomos gerados aleatoriamente.

Existem diversas variantes de **AG!**. A seguir, explica-se os detalhes de **AG!** nas formas que são utilizadas com mais frequência. No final, apresenta-se também um pseudocódigo para **AG!**. Uma descrição mais geral de **AG!** pode ser encontrada em (20).

Cromossomo

Um cromossomo é um candidato à solução ótima. Cada parte do cromossomo é chamada gene. A boa codificação do problema em cromossomo é um fator fundamental para o sucesso do algoritmo. No caso do problema de otimizar um conjunto de parâmetros, a codificação em cromossomo óbvia é representar uma dada instância por um vetor, em que cada posição (gene) corresponde a um parâmetro específico.

Função de *Fitness*

É uma função que, dado um cromossomo, retorna um número real (*fitness*) que mede o quanto bom é este cromossomo para resolver o problema. Em geral, assume-se a convenção que quanto maior o *fitness*, melhor o cromossomo.

Seleção

A cada geração uma parte da população é escolhida para reprodução. Nesse processo, a probabilidade de um cromossomo ser escolhido é geralmente proporcional ao seu *fitness* (seleção por roleta). Outras formas de selecionar os que irão se reproduzir também são aplicadas. O número de quantos indivíduos R (parâmetro do **AG!**) são escolhidos para reprodução a cada geração é um parâmetro do **AG!**.

Mutação

Corresponde ao fator de caminhada aleatória do algoritmo. Dado um cromossomo, essa operação itera sobre cada gene e o troca por um valor aleatório (dentro do domínio do gene) com uma certa probabilidade de mutação p_m (parâmetro do **AG!**).

Crossover

Dados dois cromossomos escolhidos para se reproduzir, é interessante (assim como ocorre na Natureza) que, ao invés de se ter filhos como cópia idêntica dos pais, faça-se uma recombinação dos genes dos pais de modo a produzir cromossomos com características mistas. Assim, o *crossover* em um **AG!** segue um procedimento análogo ao *crossover* biológico: escolhe-se um ponto de quebra e cria-se um cromossomo filho tomando-se a parte do primeiro pai antes do ponto de quebra e a do segundo após esse ponto. Observe que esse processo pode ser generalizado para múltiplos pontos de quebra, embora comumente se utilize um único ponto.

Sobrevivência dos Mais Aptos

Como a cada iteração ocorre reprodução, a população tende a aumentar indefinidamente. Para evitar isso, assume-se um tamanho de população máximo S_m (parâmetro do **AG!**) e elimina-se os menos aptos (com menores valores de *fitness*) a cada geração. Outra forma de realizar esta operação envolve escolher probabilisticamente os sobreviventes com probabilidade de escolha dependente do valor de *fitness*.

Algoritmo 1: Pseudocódigo do Algoritmo Genético.

```

begin
    Populacao  $\leftarrow$  PopulacaoAleatoria( $S_i$ );
    Fitnesses  $\leftarrow$  CalcularFitnesses(Populacao);
    while critério de parada não satisfeito do
        Pais  $\leftarrow$  Selecao(Populacao, Fitnesses,  $R$ );
        Filhos  $\leftarrow$  Crossover(Pais);
        Populacao  $\leftarrow$  Populacao  $\cup$  Filhos;
        Populacao  $\leftarrow$  Mutacao(Populacao,  $p_m$ );
        Fitnesses  $\leftarrow$  CalcularFitnesses(Populacao);
        Populacao  $\leftarrow$  MaisAptos(Populacao, Fitnesses,  $S_m$ );
    end
end

```

5.2 Particle Swarm Optimization

PSO! é um algoritmo de otimização iterativo que busca uma candidata à solução ótima conforme uma medida de qualidade. O algoritmo trabalha com um espaço de busca em D dimensões limitado inferiormente por \mathbf{l} e superiormente por \mathbf{u} . Inicialmente, sorteia-se P “partículas” aleatoriamente tal que a posição \mathbf{x}_i e a velocidade \mathbf{v}_i de cada partícula p_i satisfazem as equações 5.1 e 5.2, respectivamente.

$$\mathbf{x}_i(d) \in [\mathbf{l}(d), \mathbf{u}(d)], d = 1, \dots, D \quad (5.1)$$

$$\mathbf{v}_i(d) \in [-|\mathbf{u}(d) - \mathbf{l}(d)|, |\mathbf{u}(d) - \mathbf{l}(d)|], d = 1, \dots, D \quad (5.2)$$

Assim, a cada iteração, cada posição de partícula é avaliada pela função de medida de qualidade $f : \mathbb{R}^D \rightarrow \mathbb{R}$ e atualiza-se \mathbf{b}_i , a melhor posição da partícula p_i até então, e \mathbf{g} , a melhor posição global de partícula até o momento, segundo f . A partir deste ponto, assume-se que o objetivo é maximizar $f(\mathbf{x})$. Note que isso não é limitação, pois se o objetivo é minimizar $f(\mathbf{x})$, basta executar o algoritmo com $g(\mathbf{x}) = -f(\mathbf{x})$. Por fim, atualiza-se a velocidade e a posição de cada partícula segundo as equações 5.3 e 5.4, respectivamente.

$$\mathbf{v}_i \leftarrow \omega \mathbf{v}_i + \varphi_p r_p (\mathbf{b}_i - \mathbf{x}_i) + \varphi_g r_g (\mathbf{g} - \mathbf{x}_i) \quad (5.3)$$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i \quad (5.4)$$

Em que ω , φ_p e φ_g são parâmetros do algoritmo, e r_p e r_g são número reais aleatórios entre 0 e 1. O método prossegue até que algum critério de parada seja atingido (número máximo de iterações, tempo máximo, limite de processamento, etc.). Um pseudocódigo para **PSO!** é apresentado no Algoritmo 2.

O **PSO!** tem vantagens por não impor requisitos sobre o problema a ser otimizado além do conhecimento do espaço de busca e de uma medida de desempenho. Entretanto, seu comportamento não é bem compreendido e não há garantia de convergência para solução ótima. Além disso, o método sofre de tendência a convergir para máximos locais.

Algoritmo 2: Pseudocódigo do *Particle Swarm Optimization*.

```

begin
    for  $i \leftarrow 1, \dots, P$  do
        for  $d \leftarrow 1, \dots, D$  do
             $\mathbf{x}_i(d) \leftarrow \text{random}(\mathbf{l}(d), \mathbf{u}(d));$ 
             $\Delta \leftarrow |\mathbf{u}(d) - \mathbf{l}(d)|;$ 
             $\mathbf{v}_i(d) \leftarrow \text{random}(-\Delta, \Delta);$ 
        end
         $\mathbf{b}_i \leftarrow \mathbf{x}_i;$ 
        if  $f(\mathbf{b}_i) > f(\mathbf{g})$  then
             $\mathbf{g} \leftarrow \mathbf{b}_i;$ 
        end
    end
    while critério de parada não satisfeito do
        for  $i \leftarrow 1, \dots, P$  do
            for  $d \leftarrow 1, \dots, D$  do
                 $r_p \leftarrow \text{random}(0, 1);$ 
                 $r_g \leftarrow \text{random}(0, 1);$ 
                 $\mathbf{v}_i(d) \leftarrow \omega \mathbf{v}_i(d) + \varphi_p r_p (\mathbf{b}_i(d) - \mathbf{x}_i(d)) + \varphi_g r_g (\mathbf{g}(d) - \mathbf{x}_i(d));$ 
            end
             $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i;$ 
            if  $f(\mathbf{x}_i) > f(\mathbf{b}_i)$  then
                 $\mathbf{b}_i \leftarrow \mathbf{x}_i;$ 
                if  $f(\mathbf{b}_i) > f(\mathbf{g})$  then
                     $\mathbf{g} \leftarrow \mathbf{b}_i;$ 
                end
            end
        end
    end
end

```

6 *Tecnologias Utilizadas*

6.1 USARSim

USARSim! é um simulador de Robótica de alta fidelidade. A versão atual é baseada no **UDK!** (**UDK!**) (10), kit de desenvolvimento da *engine* de jogos Unreal. A Unreal usa como engine de Física a Nvidia PhysX (15), uma das mais avançadas *engines* de Física de tempo real. Com isso, é possível simular interações mecânicas, tais como colisão e atrito, com boa precisão. Inclusive, consegue-se modelar com boa fidelidade o comportamento de robôs humanóides com grande número de graus de liberdade, como demonstra (27).

O simulador já possui diversos modelos de robôs, sensores e atuadores. Além disso, a implementação de novos modelos é muito facilitada pelas ferramentas providas pelo **UDK!**, em especial uma ferramenta de edição chamada **UDK!** Editor e uma linguagem de *script* de alto nível denominada UnrealScript. A seguir, são apresentados os conceitos da **UDK!** e do **USARSim!** relevantes para o entendimento deste trabalho.

6.1.1 Nvidia PhysX

A simulação de Física da PhysX mantém uma cena em que os objetos são atualizados iterativamente, em que a cada iteração injeta-se um passo de simulação (no caso de tempo real, o intervalo de tempo desde a última atualização). Este comportamento discreto introduz erros de Física quando comparado ao que seria uma simulação contínua, logo deseja-se que o passo de simulação seja o menor possível para permitir uma maior fidelidade.

Em simulações em que a Física é crítica, como a simulação de um robô humanóide, a qualidade da simulação degrada muito com a redução da taxa de atualização da Física, como mostra (27). Observe que a simulação na Unreal ocorre em tempo real, o que torna o passo limitado pela capacidade de processamento disponível.

Uma cena de PhysX contém três importantes aspectos: *actors* (atores), *materials* (materiais) e *joints* (juntas). *Actors* definem objetos físicos capazes de interagir com o mundo e com outros objetos. *Actors* podem ser estáticos (fixos no mundo; geralmente usados para partes do cenário) ou corpos rígidos dinâmicos (usados para objetos móveis).

A cada *actor* pode estar associado um formato (modelo) físico. Note que este é desvinculado do modelo gráfico que é renderizado pela *engine* gráfica. Na realidade, para um dado objeto, em geral convém utilizar um modelo físico mais simples que o gráfico, pois cálculos de Física consomem bastante processamento; usar o modelo gráfico para Física é impraticável na maioria dos casos. Ademais, um *actor* também possui um tensor de inércia I_{body} e uma massa M localizada em seu centro de massa C_M .

Materials descrevem as propriedades de uma superfície (e.g. coeficiente de atrito) de um *actor*. Essas propriedades ditam o que ocorre quando dois *actors* colidem. *Joints* conectam dois corpos rígidos e limitam o movimento entre estes. PhysX possui diversos tipos de juntas (classificadas conforme as restrições aplicados aos corpos que unem). Neste trabalho, a única junta utilizada é a *revolute joint* (junta de revolução), que conecta dois corpos por um eixo como mostra a Figura 9.

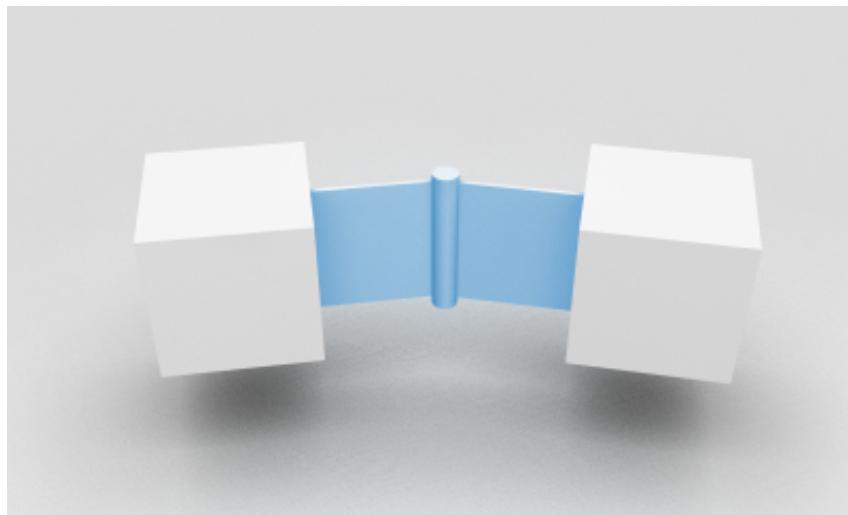


Figura 9: Junta de revolução.

Uma das tarefas mais importantes de uma engine Física é detectar quais colisões ocorrem em um dado momento eficientemente. Há dois desafios principais a serem superados nisso: detecção de colisão polígono por polígono (extremamente custosa) e dado n objetos presentes em uma cena, precisa-se, a princípio, testar colisão entre $\binom{n}{2}$ pares de objetos.

Para resolver o primeiro problema, a *engine* mantém uma caixa delimitadora (*bounding box*) para cada modelo físico. Essa caixa é formada pelos limites mínimos e máximos

do modelo em cada uma das coordenadas. Assim, para verificar se dois modelos colidem, o algoritmo verifica primeiro se as caixas delimitadoras se sobrepõem; se isso não ocorre, pode-se afirmar que os modelos certamente não colidem e evita-se o custoso teste polígono por polígono.

Para evitar testar colisão entre todos os pares de modelos de uma cena, PhysX divide o espaço em partições, de modo que há necessidade de verificar colisões apenas com outros objetos presente na mesma partição ou no máximo em partições vizinhas.

6.1.2 UDK Editor

O **UDK!** Editor é a ferramenta principal de edição do **UDK!**. Por ser uma ferramenta do UDK, ela é voltada à criação de jogos. Assim, possui diversas funcionalidades para este fim, como manipulação de modelos 3D, criação de cenários etc.

Para esse trabalho, a funcionalidade mais importante é a auto-geração de modelos físicos a partir de modelos gráficos 3D. Para isso, o editor provê diversos métodos. Um dos mais convenientes é o K-DOF. O algoritmo do K-DOF basicamente toma K planos alinhados com K eixos passando pelo centro da malha gráfica e aproxima esses planos o máximo possível da malha sem que ocorra intersecção com esta. Pode-se escolher K dentre as seguintes opções:

- 6: caixa alinhada com os eixos principais (X, Y e Z);
- 10: caixa com 4 arestas chanfradas – pode-se escolher dentre arestas alinhadas com os eixos X, Y ou Z;
- 18: caixa com todas as arestas chanfradas;
- 26: caixa com todas as arestas e cantos chanfrados.

A Figura 10 exibe as diferenças entre as opções de K-DOF para a peça do peito do Robonova.

6.1.3 UnrealScript

A UnrealScript é uma linguagem de *script* proprietária da *engine* Unreal. Foi projetada por Tim Sweeney especificamente para desenvolvimento de jogos. As principais características da UnrealScript são:

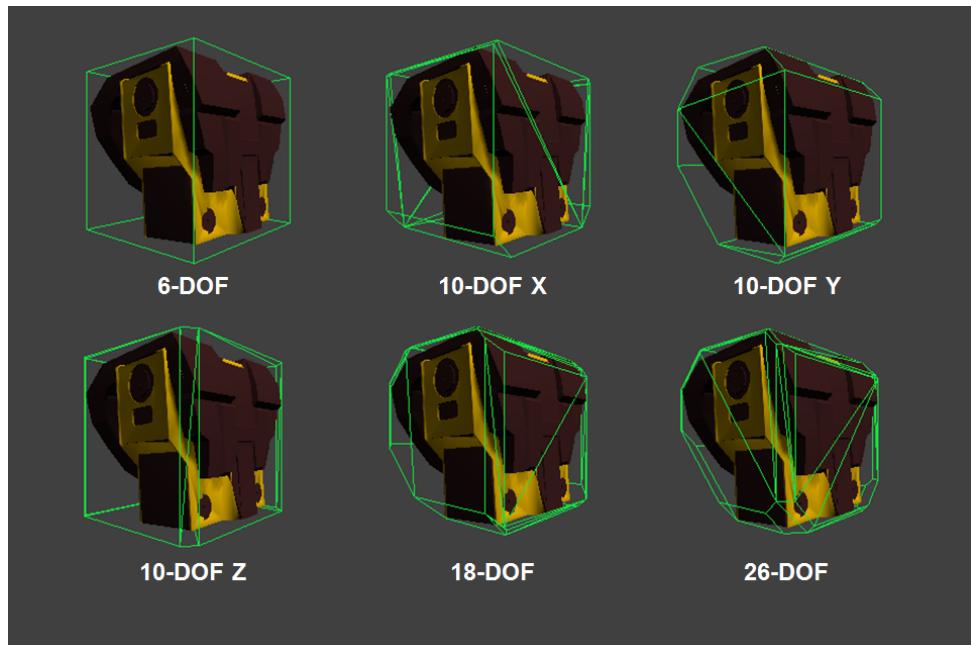


Figura 10: Diferenças entre modelos físicos gerados com K-DOF (os modelos físicos são apresentados em *wireframe*).

- Orientada a objetos;
- Não há suporte a herança múltipla, ou seja, uma classe só poder herdar de no máximo uma outra classe;
- Não existe o conceito de ponteiro. Trabalha-se com referências, assim como em Java;
- Programador não precisa se preocupar com desalocação de memória dinâmica, pois há um *garbage collector*;
- Forte detecção de erros durante tempo de compilação;
- Estilo de sintaxe similar a C/C++/Java;
- Suporte a sobrecarga de operadores;
- Não há suporte a sobrecarga de métodos;
- Suporte nativo a conceitos importantes para desenvolvimento de jogos: tempo, estados, eventos, propriedades, rede etc.

Como pode-se perceber pelas características listadas, muitas das decisões de projeto da UnrealScript foram inspiradas pela linguagem Java. Inclusive, Tim Sweeney experimentou

usar Java para a Unreal antes de decidir criar uma linguagem própria. Por fim, destaca-se que a UnrealScript é usada apenas para programar a parte de alto nível. O baixo nível, como as partes de renderização gráfica e de simulação de Física, que exigem mais desempenho, é programado em C++.

6.1.4 Implementação de Robôs em USARSim

A implementação de modelos novos de robôs em USARSim envolve basicamente os passos:

1. Construção de modelos **CAD!** que representam o robô;
2. Geração dos modelos físicos a partir dos modelos **CAD!**;
3. Programação do modelo do robô em UnrealScript;
4. Definição de partes extras, como sensores e atuadores.

6.1.5 Comunicação entre Agentes e USARSim

A comunicação entre agentes e o simulador é feita através de trocas de mensagens **TCP/IP!** seguindo protocolo próprio do **USARSim!**. Todas as mensagens trocadas contêm um tipo e uma lista de segmentos seguindo o formato “TIPO segmento1 segmento2...”, em que:

- TIPO: refere-se ao tipo da mensagem e deve ser escrito em letras maiúsculas. O protocolo define 5 tipos de mensagens: INIT, STA, SEN, DRIVE e CONF;
- Segmentos: são pares nome-valor escritos da forma “nome valor” que representam os dados da mensagem em si. Por exemplo, no caso do segmento “Orientation 0.0,0.0,0.0”, “Orientation” é o nome e “0.0,0.0,0.0” é o valor.

Note que o tipo e a lista de segmentos são separados por um espaço. Os segmentos são também separados entre si por um espaço. Dentro de um segmento, o nome e valor também são separados por um espaço, assim espaços dentro do nome ou do valor não são permitidos. Com isso, o segmento “Orientation 0.0, 0.0, 0.0” é inválido. Para indicar fim de mensagem, deve-se adicionar “\r\n” no final da cadeia de caracteres.

6.2 Robonova-I

O Robonova-I é um modelo de robô humanoíde de baixo custo desenvolvido pela empresa japonesa Hitec Robotics (18). A Figura 11 exibe o robô visto de costas e de frente. O Robonova-I é dotado de 16 graus de liberdade, distribuídos da seguinte forma: 3 em cada braço (2 no ombro e 1 no cotovelo) e 5 em cada perna (2 na coxa, 1 no joelho e 2 no pé). A Figura 12 mostra a disposição das juntas, assim como o eixo de rotação de cada uma.



Figura 11: Robonova-I. Extraído de (18).

Cada junta é implementada por um servomotor Hitec HSR-8498HB, apresentado na Figura 13. Tratam-se de servomotores especificamente desenvolvidos para o Robonova-I e são relativamente baratos, o que garante o baixo custo do robô. A Tabela 1 apresenta as principais características do HSR-8498HB.

Interface	Protocolo HMI! , PWM!
Voltagem	6,0 V
Velocidade Máxima	60° em 0,20 s @ 6,0 V
Torque	10 kg·cm
Ângulo de Operação	Máximo de 180°
Peso	55 g
Dimensão	40 x 20 x 47 mm

Tabela 1: Especificação dos servomotores Hitec HSR-8498HB. Extraído de (18).

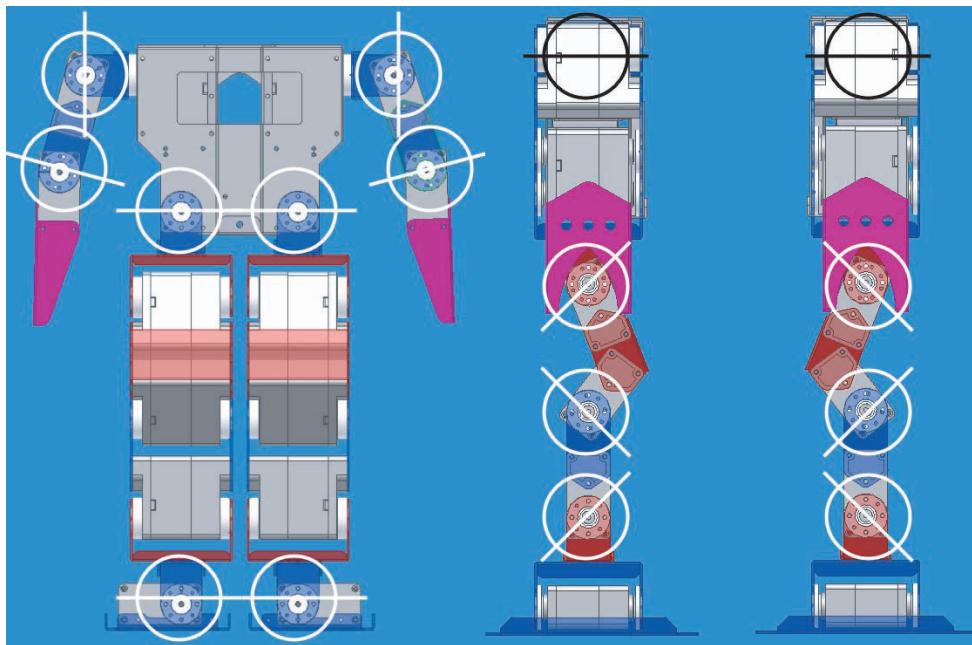


Figura 12: Juntas do Robonova-I. Extraído de (18).



Figura 13: Servomotores Hitec HSR-8498HB. Extraído de (18).

O único sensor que vem juntamente com o kit é um sensor infravermelho, que é instalado na cabeça do robô, para receber comandos de um controle remoto, que também faz parte do kit. Porém, o fabricante produzia alguns tipos de sensores compatíveis, como sensor de toque, girômetro e sensor de luz.

O controle do robô é feito por uma placa MR-C3024, que fica instalada nas costas do robô, conforme mostra a Figura 14. A CPU da placa é um Atmel ATMEGA128, o que é uma limitação caso se deseje ter um robô humanóide completamente autônomo. O ponto forte da MR-C3024 é ter uma capacidade de **E/S!** razoável, principalmente suporte a 24 **PWM!**s, o que é suficiente para controlar todos os 16 servomotores do kit e ainda permite a adição de mais 8. A Tabela 2 resume as características da MR-C3024.

A programação é feita com uma linguagem proprietária chamada RoboBASIC, especificamente criada para o Robonova-I. A linguagem tem a vantagem de possuir primitivas

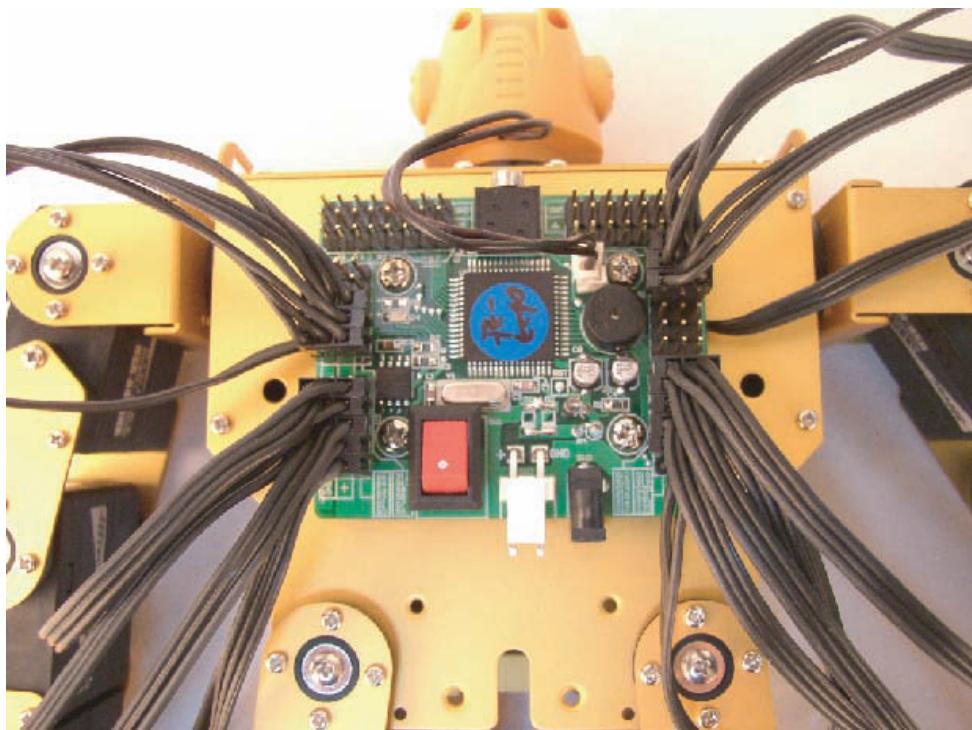


Figura 14: Detalhe das costas do Robonova-I mostrando a placa MR-C3204 instalada. Extraído de (18).

CPU	Atmel ATMEGA128 8bit RISC
Máx Servos	24
Conversores A/D	8
Memória de Programa	32 Kbytes
Peso	0,2 kg
Dimensão	55 x 50 x 15 mm

Tabela 2: Especificação da placa MR-C3024. Extraído de (6).

específicas para utilizar os recursos do Robonova-I e da MR-C3024. A programação feita é passada do PC para a MR-C3024 por comunicação serial RS-232 com uso de um cabo próprio. Outra opção de controle é mandar comandos via RS-232 para mover diretamente os servomotores seguindo protocolo próprio da MR-C3204 (1).

7 *Modelo de Simulação*

7.1 Modelos CAD

Os modelos **CAD!** foram confeccionados em um trabalho anterior (13). Os modelos foram construídos com uso do *software* Autodesk AutoCAD (3) a partir de medidas das peças de do Robonova-I real tomadas com um paquímetro. Cada peça foi feita como um modelo **CAD!** separado.

Em seguida, as peças foram importados no *software* Autodesk 3D Studio Max (2), onde materiais (gráficos) com padrões de cores simples foram criados e assinalados conforme as cores correspondentes no robô real. As Figuras 15 e 16 apresentam o modelo final montado e o explodido, respectivamente. Como pode-se verificar, teve-se o cuidado de confeccionar o modelo com boa precisão, para que se consiga um comportamento em simulação o mais fiel possível ao robô real.

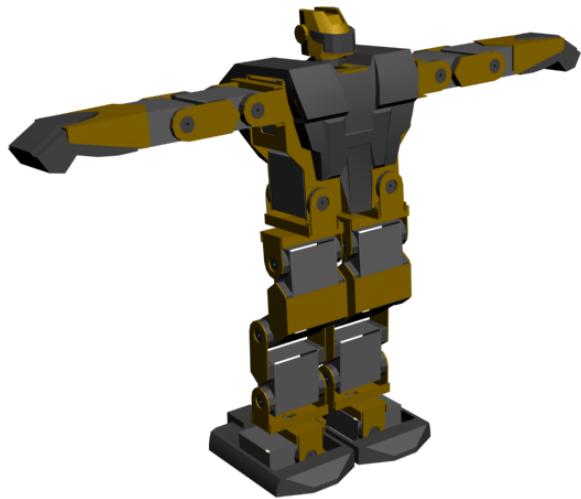


Figura 15: Modelo completo montado do Robonova-I.

Finalmente, cada modelo foi exportado como um arquivo FBX (4) separado, porque esse formato permite importação direta no **UDK!** Editor.

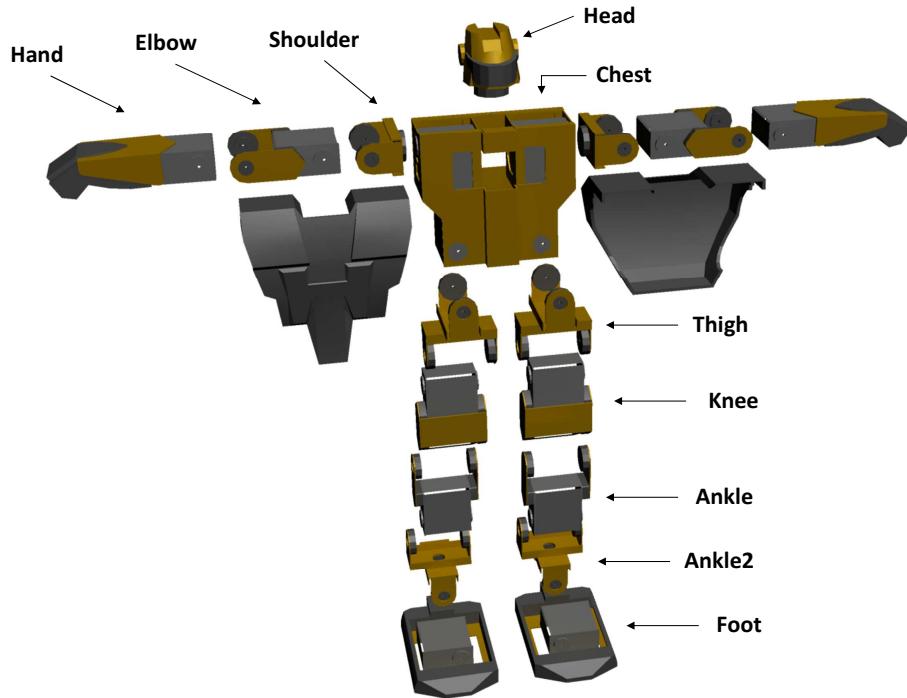


Figura 16: Modelo completo explodido do Robonova-I.

7.2 Configuração no UDK Editor

Para ter-se um modelo de simulação o mais fiel possível à realidade, optou-se pelo método *26-DOF simplified collision* para auto-gerar os modelos físicos a partir dos gráficos.

Embora os materiais já tivessem sido associados às peças anteriormente, foi necessário repetir esse processo dentro do **UDK!** Editor. Após a configuração de todos as peças, exportou-se o conjunto de peças como um pacote UPK (Unreal Package).

7.2.1 Implementação no USARSim

Para concluir a implementação do robô em **USARSim!**, faltava ainda criar a classe do robô em UnrealScript e configurar os sensores. A classe em UnrealScript do Robonova-I foi inspirada na implementação em **USARSim!** do robô Albebaran Nao (27).

As peças foram implementadas como objetos do tipo “Part”, que permite definir o modelo (gráfico e físico), massa e uma posição em relação à origem do robô (ou um deslocamento em relação a uma outra Part tomada como “pai”). A Tabela 3 apresenta todas as peças implementadas.

Em seguida, as juntas foram implementadas como juntas de revolução (objetos do

Peca (Part)	Massa (g)
Head	27
Chest	337
LeftShoulder	6
LeftElbow	65
LeftHand	65
RightShoulder	6
RightElbow	65
RightHand	65
LeftThigh	23
LeftKnee	135
LeftAnkle	44
LeftAnkle2	23
LeftFoot	83
RightThigh	23
RightKnee	135
RightAnkle	44
RightAnkle2	23
RightFoot	83

Tabela 3: Peças implementadas como objetos do tipo “Part”.

tipo “RevoluteJoint”). Um objeto do tipo RevoluteJoint une dois objetos do tipo Part por um eixo de rotação. A Figura 17 mostra os nomes dados às juntas implementadas.

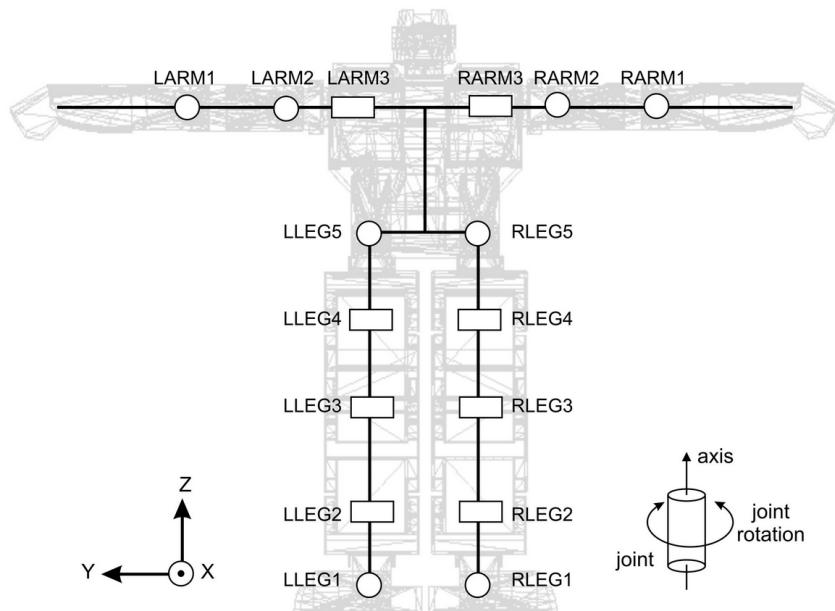


Figura 17: Juntas implementadas no modelo de simulação.

Devido a imprecisões dos modelos físicos, foi necessário desabilitar colisão entre peças adjacentes (considera-se que duas peças são adjacentes se são ligadas por uma junta). As

demais colisões foram mantidas ativas.

Para percepção, colocou-se uma câmera na cabeça e sensores de toque nos pés. O **USARSim!** já possui esses sensores implementados, logo foi necessário apenas configurá-los. Além disso, acoplou-se um sensor especial do **USARSim!** chamado “GroundTruth” no tronco do robô. Esse sensor provê posição e orientação globais livres de ruído. Embora essas informações sejam impossíveis de se obter em um robô real, elas são muito úteis em processos de Aprendizado ou Otimização.

A Figura 18 apresenta o Robonova-I dentro do **USARSim!**; os modelos físicos são mostrados em *wireframe* por linhas rosas. No canto superior esquerdo é mostrada a visualização a partir da câmera do robô.



Figura 18: Robonova-I simulado dentro do **USARSim!**.

8 Programa do Agente

O agente foi implementado em C++ usando a biblioteca Qt (14). Essas tecnologias foram escolhidas pois permitem implementação *cross-platform* e um alto desempenho em tempo de execução. Para permitir que boa parte do código fosse comum entre simulação e robô real, dividiu-se o programa do agente em três camadas (vide Figura 19):

- **Comunicação:** implementa a comunicação de baixo nível com sensores e atuadores. Interface e implementação são diferentes entre simulação e robô real;
- **Interface:** provê uma abstração para sensores e atuadores. Mesma interface na simulação e no robô real, mas implementações diferentes;
- **Controlador:** determina quais posições devem ser enviadas para as juntas do robô a cada momento. Interface e implementação iguais entre simulação e robô real.

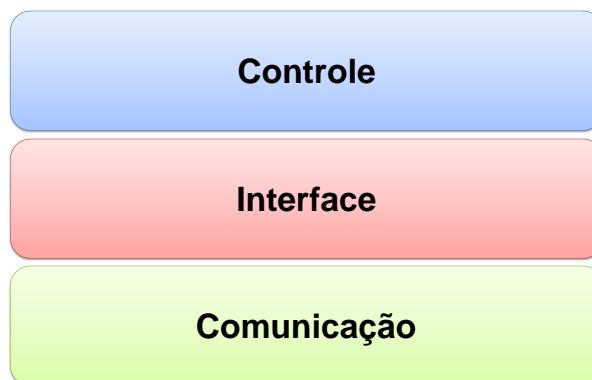


Figura 19: Camadas do programa do agente.

É importante observar que a camada de Interface provê uma abstração para a camada superior (Controle), de modo que, a princípio, pode-se usar o mesmo controlador para o robô simulado e para o real. Já as camadas de Comunicação e de Interface devem ser implementadas em cada plataforma.

Tal arquitetura é suficiente para os propósitos desse trabalho. Todavia, caso deseje-se que o robô realize tarefas mais complexas, como jogar futebol, as seguintes alterações são interessantes:

- Adição de um componente de **Modelo** à camada de Controle. Esse componente seria responsável por processar as percepções e gerar modelos mais elaborados sobre o mundo e sobre o agente, de modo a prover informações importantes, como posições globais do agente, de outros agentes e de objetos. Com isso, seria adequado passar a chamar essa nova camada de **Modelo e Controle**;
- Adição de uma camada de **Cognição**, que deveria ficar acima das demais. Tal camada trataria de, a partir dos modelos providos pelo componente de Modelo, determinar o melhor movimento (andar, girar o corpo, levantar-se etc.) a ser executado e notificar a camada de Controle disto.

A seguir, cada camada efetivamente implementada é detalhada. Nisso, deve-se destacar um fato: fazia parte do plano inicial utilizar o mesmo programa de agente no robô simulado e no real, tanto que a arquitetura explicada foi idealizada de modo a permitir isso. Entretanto, o único *hardware* de processamento disponibilizado foi a placa original do Robonova-I (MR-C3204) que é muito limitada e permite programação apenas em linguagem RoboBASIC. Com isso, implementou-se o programa de agente aqui descrito apenas na simulação.

Embora isto torne a separação do programa em camadas irrelevante para a proposta deste trabalho, esta arquitetura ainda é interessante para servir como base para futuros trabalhos em que se possua o *hardware* de processamento necessário.

8.1 Comunicação

A camada de comunicação implementa a comunicação direta de baixo nível com sensores e atuadores. Como essa comunicação difere muito entre robô simulado e real, é difícil estabelecer padrões para essa camada, portanto considerou-se interface e implementação diferentes entre as duas plataformas.

Para a simulação, essa camada deve consistir de um socket **TCP/IP!** para comunicação com o servidor e de um *parser* para transformar as mensagens brutas recebidas em uma estrutura mais fácil de lidar. Desse modo, a implementação é composta por duas classes:

- USARSocket: implementa um socket **TCP/IP!** para comunicação com o servidor do **USARSim!**. Faz a separação dos dados recebidos na comunicação em mensagens usando a indicação de fim de mensagem “\r\n”, como especificado no protocolo. A funcionalidade de socket propriamente dita é herdada de QTcpSocket, que a implementa de modo *cross-platform*;
- USARMessage: representa uma mensagem que segue o protocolo do USARSim. A classe possui dois construtores: um que faz o *parsing* de uma mensagem bruta recebida do servidor e a separada em seus constituintes (tipo e segmentos) e outro que recebe os constituintes e gera uma mensagem no protocolo esperado pelo servidor.

8.2 Interface

Esta camada trata as estruturas intermediárias recebidas da camada de Comunicação e provê uma abstração para sensores e atuadores independente de se o programa está sendo executado no robô simulado ou no real. Como a camada lida com estruturas da camada de Comunicação, a implementação tem de ser diferente entre as duas plataformas. Porém, como a intenção é prover uma abstração independente de plataforma para a camada de Controle, a interface deve ser a mesma entre robô simulado e real.

Para conseguir esse efeito, o mecanismo usado em Orientação a Objetos é criar classes abstratas para representar a interface e então criar classes concretas que herdam destas classes abstratas e que implementam as funcionalidades desejadas em cada caso. Portanto, criou-se as seguintes classes abstratas:

- RobotPerceptor: funciona como uma espécie de sensor especial que contém informações do corpo do robô, como nível da bateria e posições das juntas;
- Sensor: representa um sensor genérico. Sensores devem ser implementados através de especialização desta classe, com exceção do RobotPerceptor;
- Perception: provê uma interface única para toda a percepção;
- Action: representa uma ação genérica. Não inclui comandos para as juntas;
- ActionHandler: provê uma interface única para toda a atuação (incluindo as juntas).

No robô simulado, essa camada tem basicamente duas responsabilidades. No lado da percepção, deve interpretar as estruturas do tipo USARMessage da camada de Comunicação e atualizar o estado interno do agente. Em relação à atuação, deve receber

uma requisição para movimentação de junta e criar uma estrutura do tipo USARMessage que será passada para a camada de Comunicação. Para isso, as principais classes criadas foram:

- GroundTruth: implementa sensor GroundTruth. Sensor exclusivo do robô simulado;
- USARPerception: implementação concreta de Perception. Interpreta uma USAR-Message recebida da camada de Comunicação e atualiza o estado interno do robô;
- InitAction: inicia um modelo de robô na simulação;
- GetStartPosesAction: requisita ao servidor as localizações padrões em que o robô pode ser iniciado no mapa em questão;
- ReconnectAction: reconecta o agente ao simulador;
- USARActionHandler: implementação concreta de ActionHandler. Recebe ações (incluindo movimentação de junta), cria uma USARMessage correspondente e a envia para a camada de Comunicação.

8.3 Controle

O objetivo desta camada é implementar os movimentos do robô humanóide a partir do controle de suas juntas. Como a camada de Interface provê uma abstração para sensores e atuadores, o código desta camada pode ser o mesmo tanto no robô simulado quanto no simulado (considerando que o comportamento físico do robô simulado seja próximo o suficiente do real).

Dado o escopo do trabalho, implementou-se apenas um controlador para a caminhada do robô. Porém, a implementação feita já provê uma estrutura para implementações futuras de novos movimentos, como girar o corpo, levanta-se, chutar uma bola etc. As principais classes criadas foram:

- TFSGaitController: controlador para caminhada baseada em **SFT!** (**SFT!**);
- TFSGaitGenerator: classe abstrata que representa um gerador de caminhada baseada em **SFT!**. Dado o instante de tempo atual, gera as posições das juntas a partir das equações do modelo de caminhada.

9 Processo de Otimização

9.1 Modelos de Caminhada

Para verificar os efeitos da adição de movimento de braços e movimento no plano coronal, decidiu-se por implementar geradores para os 3 modelos propostos por Shafii (21, 22, 23), conforme explicitado no Capítulo ??.

Entretanto, após alguns testes, optou-se por modificar a função proposta por Shafii para os ombros. A modificação proposta envolve considerar que as amplitudes no sentido positivo e negativo podem ser diferentes para o movimento do braço, assim como é feito para a perna no sentido frente-trás. Desse modo, $\theta_o(t)$ passa a ser descrita pela equação (9.1).

$$\theta_o(t) = \begin{cases} -D_- \sin\left(\frac{2\pi t}{T}\right), & t \in \left[kT, \frac{T}{2} + kT\right), k \in \mathbb{Z} \\ -D_+ \sin\left(\frac{2\pi t}{T}\right), & t \in \left[\frac{T}{2} + kT, (k+1)T\right), k \in \mathbb{Z} \end{cases} \quad (9.1)$$

Note que essa modificação aumenta em 1 o número de constantes a serem determinadas para os modelos que possuem movimento de braços. Portanto, os modelos de caminhada implementados são os seguintes:

- Modelo Simples (MS): usa apenas as juntas na direção frente-trás das pernas;
- Modelo com Movimento de Braços (MB): além das juntas do “Modelo Simples”, utiliza as juntas do ombro (com trajetória para o ombro esquerdo definida pela equação (9.1));
- Modelo Complexo (MC): além das juntas do Modelo com “Movimento de Braços”, adiciona as juntas da coxa no sentido lateral (plano coronal).

Junta nos Modelos Teóricos	Junta no Robonova
Coxa Frente-trás Esquerda	LLEG4
Joelho Esquerdo	LLEG3
Pé Frente-trás Esquerdo	LLEG2
Ombro Esquerdo	LARM3
Coxa Lateral Esquerda	LLEG5
Pé Lateral Esquerdo	LLEG1
Coxa Frente-trás Direita	RLEG4
Joelho Direito	RLEG3
Pé Frente-trás Direito	RLEG2
Ombro Direito	RARM3
Coxa Lateral Direita	RLEG5
Pé Lateral Direito	RLEG1

Tabela 4: Mapeamento das juntas dos modelos teóricos para as do Robonova (veja a Figura 17 para nomenclatura utilizada para as juntas do Robonova).

Observe que o Robonova possui os graus de liberdade necessários para implementar os 3 modelos. A Tabela 4 apresenta o mapeamento feito das juntas do modelo teórico para as do Robonova (veja a Figura 17 para nomenclatura utilizada para as juntas do Robonova).

9.2 Configuração dos Algoritmos de Otimização

Um algoritmo de Otimização fornece um *framework* para resolução de problemas de otimização, mas deve-se definir o que significa certos elemento do algoritmo para o problema em específico.

O problema de Otimização em questão envolve a determinação de constantes reais relacionada ao modelo de caminhada. É interessante definir um domínio para cada uma das constantes a fim de evitar valores sem sentido para o problema (que resultem em ângulos ou velocidades além do domínio de operação dos servomotores ou que certamente levem à queda do robô). Após testes manuais, determinou-se os limites para as constantes do modelo apresentados na Tabela 5 (as equações (??), (??), (9.1) e (??) apresentam detalhes sobre as constantes mencionadas).

9.2.1 Configuração do Algoritmo Genético

A seguir, são apresentados como os conceitos de **AG!** são definidos no problema em específico:

Constante	Mínimo	Máximo
O_c	-1,5	0
A	0,01	1,0
B	0,01	1,0
O_j	0	2,0
C	0	1,0
$\tau = t_2/T$	0,1	0,9
T	0,1	0,7
D_+	0	0,6
D_-	0	0,6
E	0	0,5

Tabela 5: Limites para os valores das constantes dos modelos de caminhada.

- Cromossomo: a escolha mais óbvia para a definição de cromossomo é um vetor com as constantes, em que cada gene representa uma constante;
- Função de *fitness*: explicitada a seguir, na Seção 9.3;
- Mutação: altera um gene (no caso, uma das constantes) por um valor real aleatório dentro dos limites da Tabela 5.

Os demais conceitos de AG (seleção, *crossover* e sobrevivência dos mais aptos) saem naturalmente da definição de cromossomo.

Por fim, a Tabela 6 apresenta os valores dos parâmetros usados nas execuções de **AG!**.

Parâmetro	Valor
S_i	20
S_m	20
R	10
p_m	0,05

Tabela 6: Valores dos parâmetros usados nas execuções de **AG!**.

9.2.2 Configuração do Particle Swarm Optimization

A escolha natural é considerar que cada partícula em um espaço n -dimensional, em que n é o número de constantes a serem determinadas. Assim, cada posição de partículas é um vetor de constantes. O espaço de busca é dado pelos limites da Tabela 5. Já a função de medida de qualidade é apresentada na Seção 9.3.

No **PSO!** pode ocorrer das partículas atingirem velocidades muito altas e saírem do espaço de busca. Para evitar esses problemas, fez-se modificações no **PSO!** apresentado na Seção 5.2. Primeiramente, logo após a atualização de velocidade de uma partícula, limita-se sua velocidade com uso da equação (9.2).

$$\mathbf{v}_i(d) \leftarrow \min(\mathbf{u}(d) - \mathbf{l}(d), \max(\mathbf{l}(d) - \mathbf{u}(d), \mathbf{v}_i(d))), d = 1, \dots, D \quad (9.2)$$

A outra modificação busca forçar as partículas a permanecerem sempre dentro do espaço de busca. Para tal, considera-se que as partículas sofrem “choques mecânicos inelásticos” ao tentarem cruzar os limites do espaço de busca. A partir desta inspiração da Mecânica, criou-se a subrotina apresentada no Algoritmo 3, que é executada logo após ser aplicada a limitação de velocidade. Note que o uso dessa subrotina introduz o parâmetro ε , ao qual deu-se a denominação de “coeficiente de restituição” para seguir a analogia mecânica.

Algoritmo 3: Subrotina que simula “choques” das partículas com os limites do espaço de busca.

```

begin
  for  $i \leftarrow 1, \dots, P$  do
    for  $d \leftarrow 1, \dots, D$  do
      if  $\mathbf{x}_i(d) > \mathbf{u}(d)$  then
         $\delta \leftarrow \mathbf{x}_i(d) - \mathbf{u}(d);$ 
         $\mathbf{x}_i(d) \leftarrow \varepsilon(\mathbf{u}(d) - \delta);$ 
         $\mathbf{v}_i(d) \leftarrow -\varepsilon\mathbf{v}_i(d);$ 
      end
      if  $\mathbf{x}_i(d) < \mathbf{l}(d)$  then
         $\delta \leftarrow \mathbf{l}(d) - \mathbf{x}_i(d);$ 
         $\mathbf{x}_i(d) \leftarrow \varepsilon(\mathbf{l}(d) + \delta);$ 
         $\mathbf{v}_i(d) \leftarrow -\varepsilon\mathbf{v}_i(d);$ 
      end
    end
  end
end
```

Por fim, a Tabela 7 apresenta os valores dos parâmetros usados nas execuções de **PSO!**.

Parâmetro	Valor
P	20
ω	0,9
φ_p	0,6
φ_g	0,8
ε	0,7

Tabela 7: Valores dos parâmetros usados nas execuções de **PSO!**.

9.3 Montagem do Experimento Simulado

Para utilizar um algoritmo de Otimização para o “aprendizado” das melhores constantes para os modelos de caminhada, é necessário montar um experimento em que seja dado ao robô um período de tempo para tentar andar e depois seja aplicada uma métrica para avaliação do desempenho da caminhada com o dado conjunto de constantes.

O mapa do simulador escolhido para os experimentos foi o “ExampleMap” (vide Figura 20), porque possui uma grande área com terreno plano e sem presença de obstáculos. A localização escolhida para iniciar o robô no mapa foi a “RobotStart1”, que é onde o robô se encontra na Figura 20.

Como o robô inicia com as juntas em posições diferentes das posições neutras da caminhada (quando os senos valem zero), adicionou-se um tempo de preparação em que os ângulos das juntas são interpolados linearmente das posições iniciais para as neutras da caminhada. Experimentalmente, verificou-se que 1,5 segundo era adequado para esse tempo de ajuste.

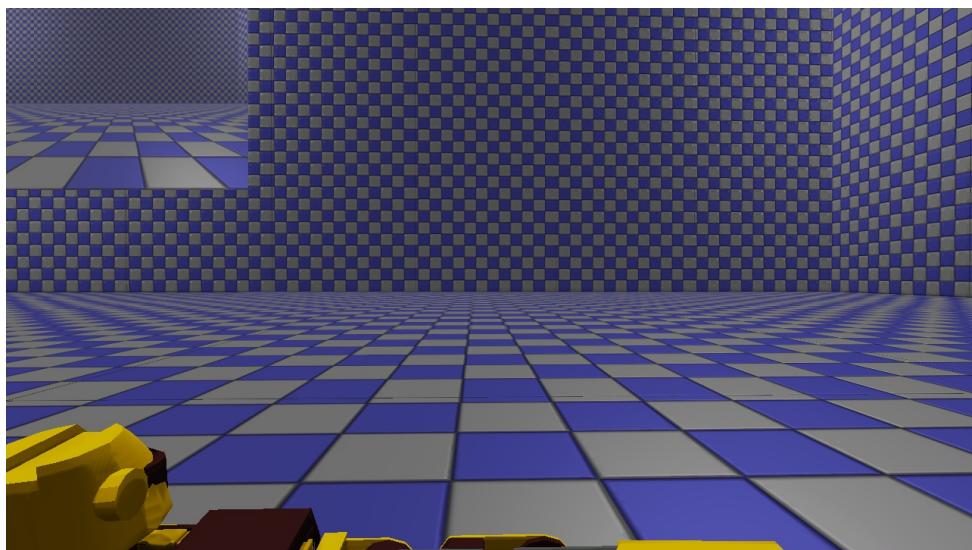


Figura 20: ExampleMap.

Depois, deixava-se o robô andar por 20 segundos, o que podia ser interrompido prematuramente caso o robô caísse. Considera-se queda se a orientação do robô passar de 0,6 radianos em relação aos eixos X ou Y (vide Figura 21). Por fim, media-se o desempenho do robô. Em resumo, cada experimento de simulação segue os passos:

1. Iniciar o Robonova na localização “RobotStart1”;
2. Esperar 1,5 segundo para preparação;
3. Esperar 20 segundos ou o robô cair, o que ocorrer primeiro;
4. Calcular desempenho com a equação (9.3).

$$D = (x - x_o) - |y - y_o| + 0,1 \times \Delta t - \sum P_i \quad (9.3)$$

Na equação (9.3), (x_o, y_o) e (x, y) são respectivamente as posições inicial e final do robô no plano do mapa; o sistema de coordenadas usado é o mostrado na Figura 21. Δt é a quantidade tempo (em segundos) que o robô permaneceu sem cair. $\sum P_i$ representa o somatório das possíveis punições (mostradas na Tabela 8). O objetivo das punições é tratar casos muito indesejados.

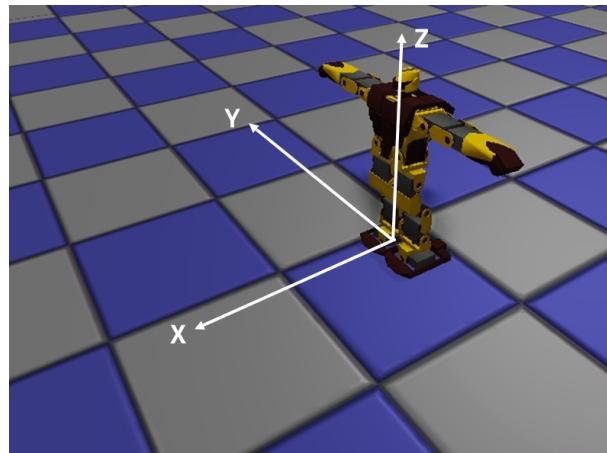


Figura 21: Sistema de coordenadas no USARSim.

Após algumas execuções dos experimentos simulados, verificou-se o seguinte:

- Percebeu-se que havia uma dificuldade grande do robô romper a transição brusca de quando ele estava parado para o primeiro passo. Ocorria com frequência de uma caminhada que se tornava bem instável quando o robô entrava em regime cair no primeiro passo;

Punição	Significado	Valor
P_1	Queda	50
P_2	Posição inicial instável	80
P_3	Praticamente não se moveu	60

Tabela 8: Limites para os valores das contantes dos modelos de caminhada.

- Às vezes, ocorria de uma caminhada pouco estável, mas muito rápida, ter a sorte de se manter em pé durante os 20 segundos do teste e receber uma avaliação muito boa. Isto era problemático principalmente para o **PSO!**, que é fortemente guiado pelo melhor desempenho encontrado.

Para resolver estes dois problemas, implementou-se duas heurísticas propostas em (23):

- Ao invés de começar o movimento já com as amplitudes no máximo, no começo do movimento reduz-se todas as amplitudes dos movimentos por um fator que aumenta linearmente de 0 a 1. Após alguns testes, determinou-se que uma duração de 3 períodos (3 primeiros passos) era adequada para este tempo. Ou seja, o robô começa dando passos curtos e vai aumentando o tamanho do passo gradativamente até atingir o valor de regime;
- Ao invés de usar o desempenho de uma única execução do experimento para alimentar a otimização, passou-se a utilizar uma média de 3 execuções.

9.4 Resultados e Discussões

A Tabela 9 apresenta os resultados dos experimentos. Como **AG!** e **PSO!** envolvem aleatoriedade, o mais justo para comparação seria realizar diversas execuções de cada algoritmo para cada modelo de caminhada e tirar uma média. Entretanto, isso exigiria uma disponibilidade de poder de processamento bem superior à que o autor tinha disponível. Assim, cada linha da Tabela 9 se refere a uma única instância de execução. Apesar disso, acredita-se que os resultados apresentados permitem fazer as seguintes considerações qualitativas:

- A adição de movimento de braços provoca melhora na qualidade da caminhada (se os parâmetros estiverem ajustados corretamente);

- A adição de movimento no plano coronal provoca melhora ainda maior na qualidade da caminhada (novamente, se os parâmetros estiverem ajustados corretamente);
- Quanto mais parâmetros são adicionados ao modelo, mais difícil é a convergência dos algoritmos, como esperado.

Algoritmo de Otimização	Modelo de Caminhada	Melhor Desempenho	Número de Testes
AG	Simples	6,16	1585
AG	Com Movimento de Braços	8,3	1678
AG	Complexo	5,77	1636
PSO	Simples	5,58	860
PSO	Com Movimento de Braços	7,2	1449
PSO	Complexo	7,99	1493

Tabela 9: Resultados dos Experimentos.

As Figuras 22, 23 e 24 apresentam como varia o melhor *fitness* até então pelo número de testes para experimentos com AG. As Figuras 25, 26 e 27 fazem o mesmo para o PSO. A Tabela 10 exibem os melhores conjuntos de parâmetros encontrados para cada caso.

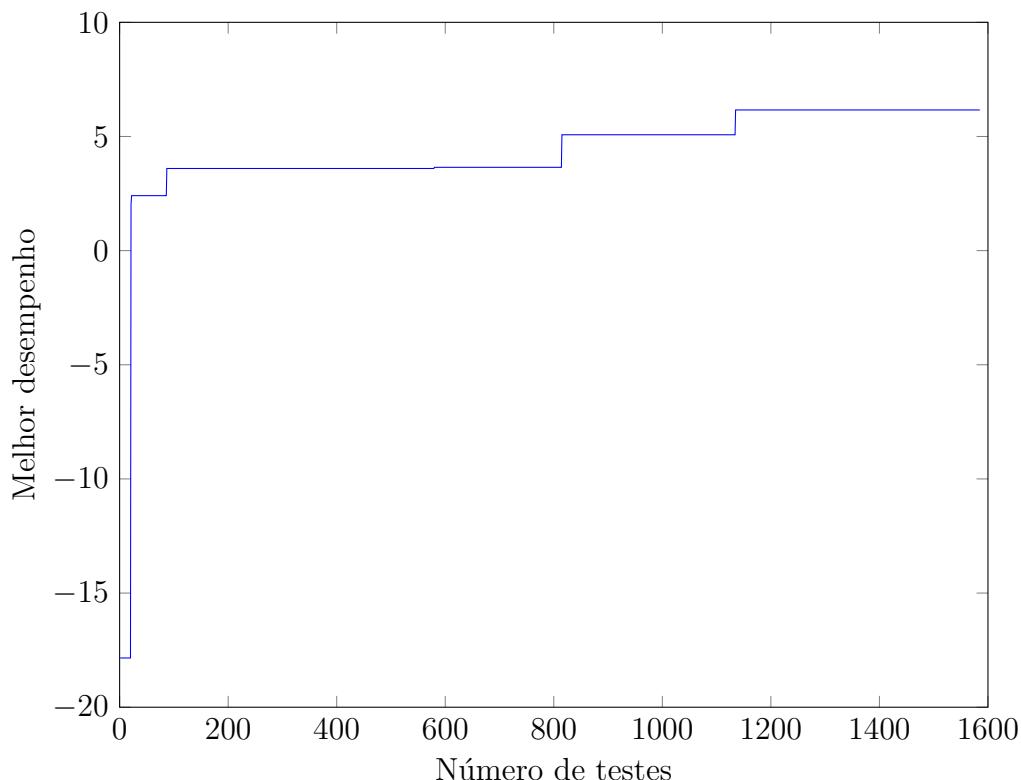


Figura 22: Resultados da otimização com Modelo Simples e **AG!**.

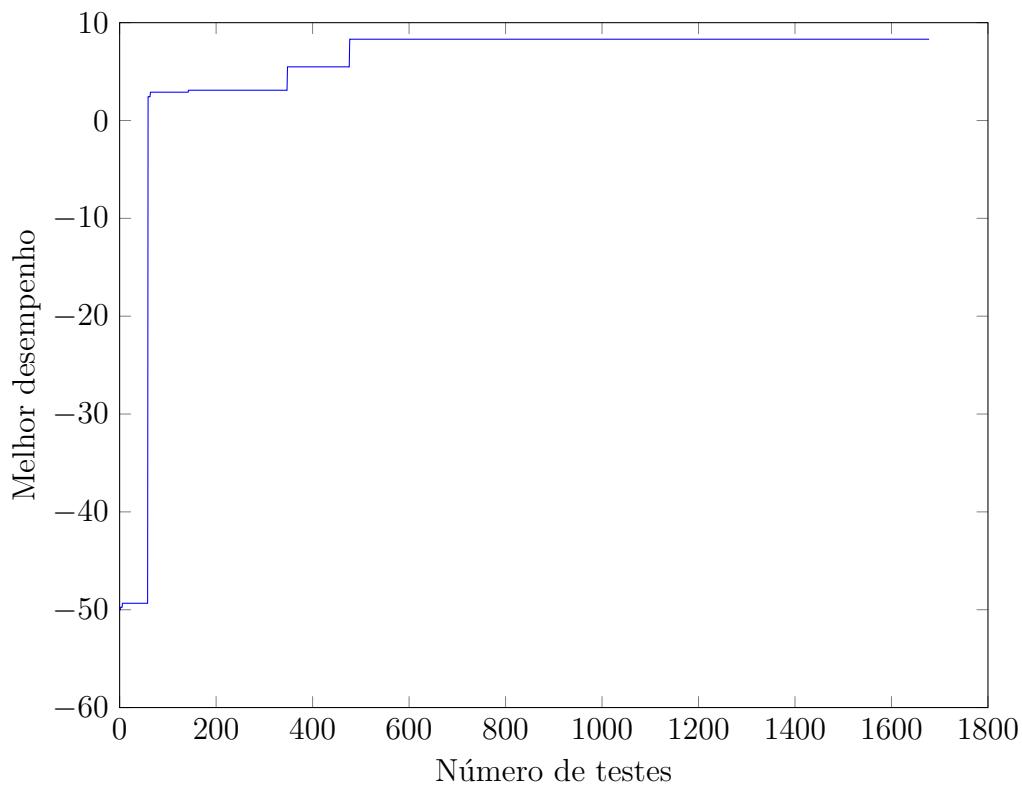


Figura 23: Resultados da otimização com Modelo com Movimento de Braços e **AG!**.

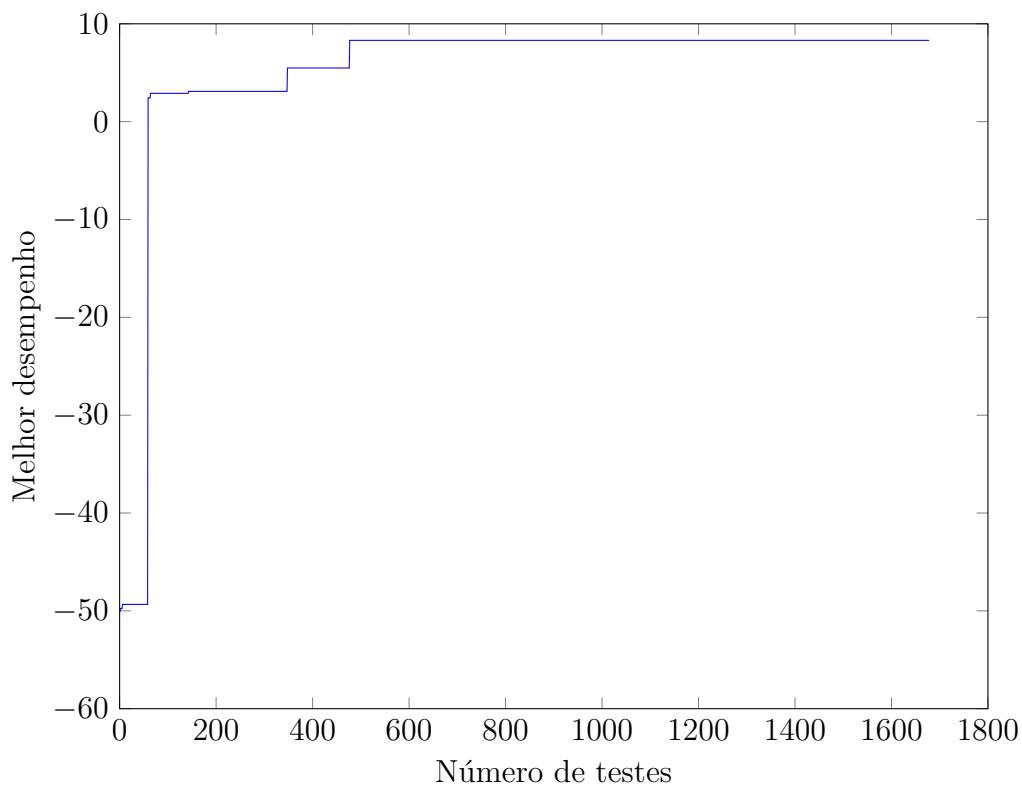


Figura 24: Resultados da otimização com Modelo Complexo e **AG!**.

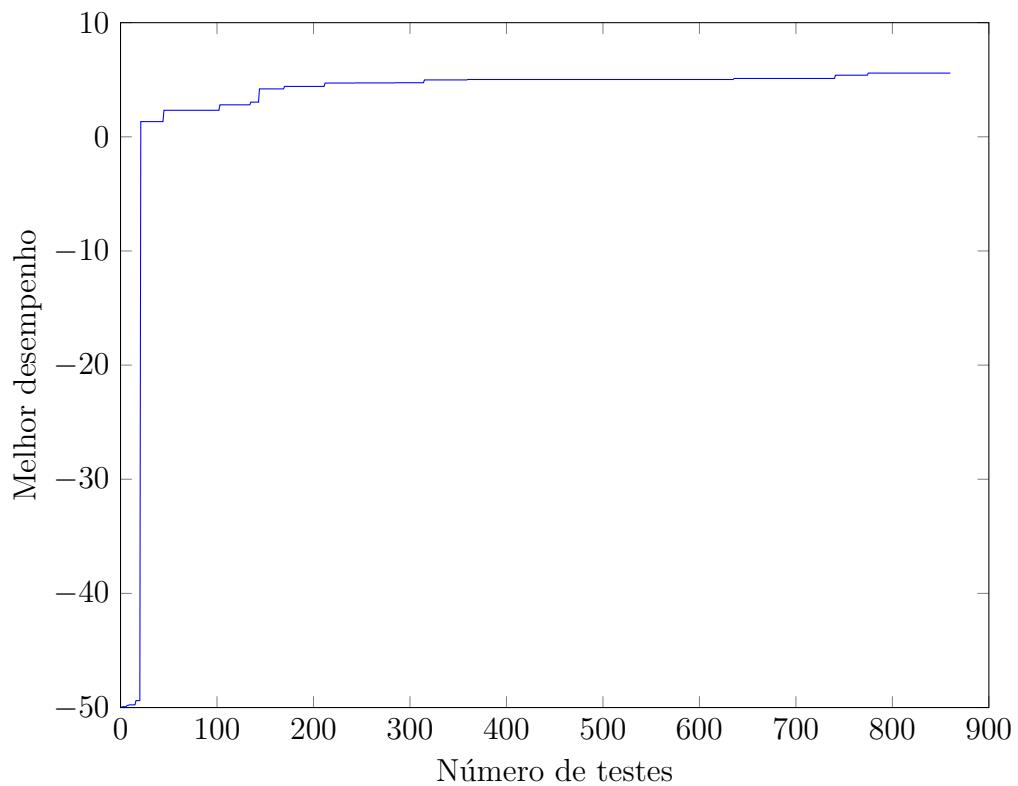


Figura 25: Resultados da otimização com Modelo Simples e **PSO!**.

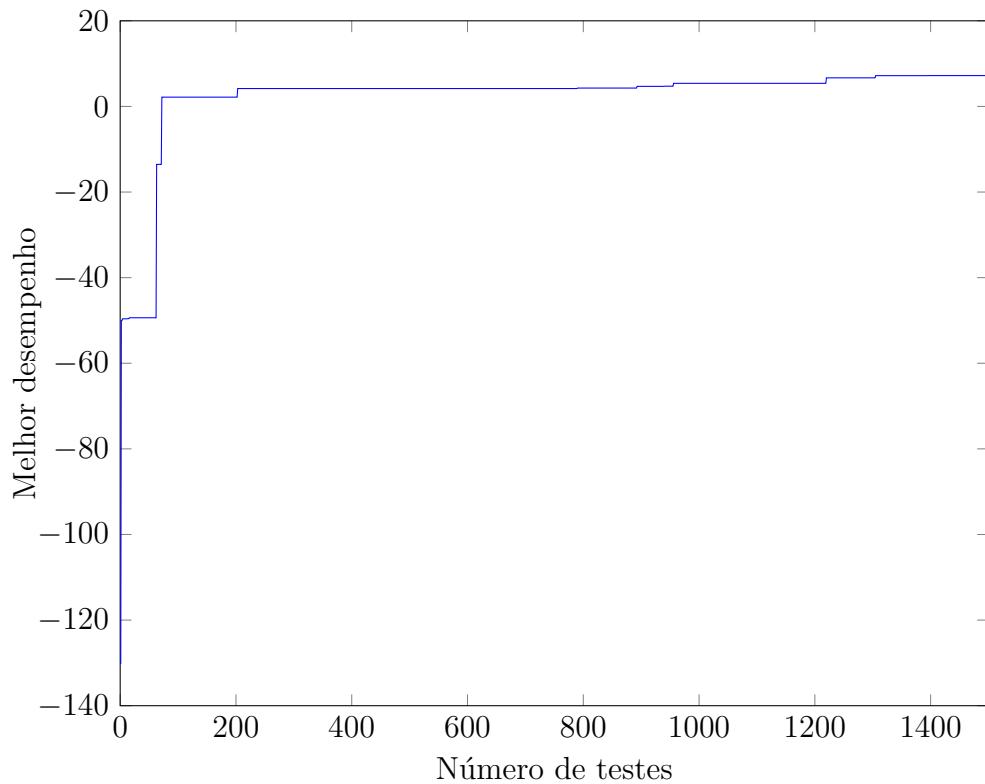


Figura 26: Resultados da otimização com Modelo com Movimento de Braços e **PSO!**.

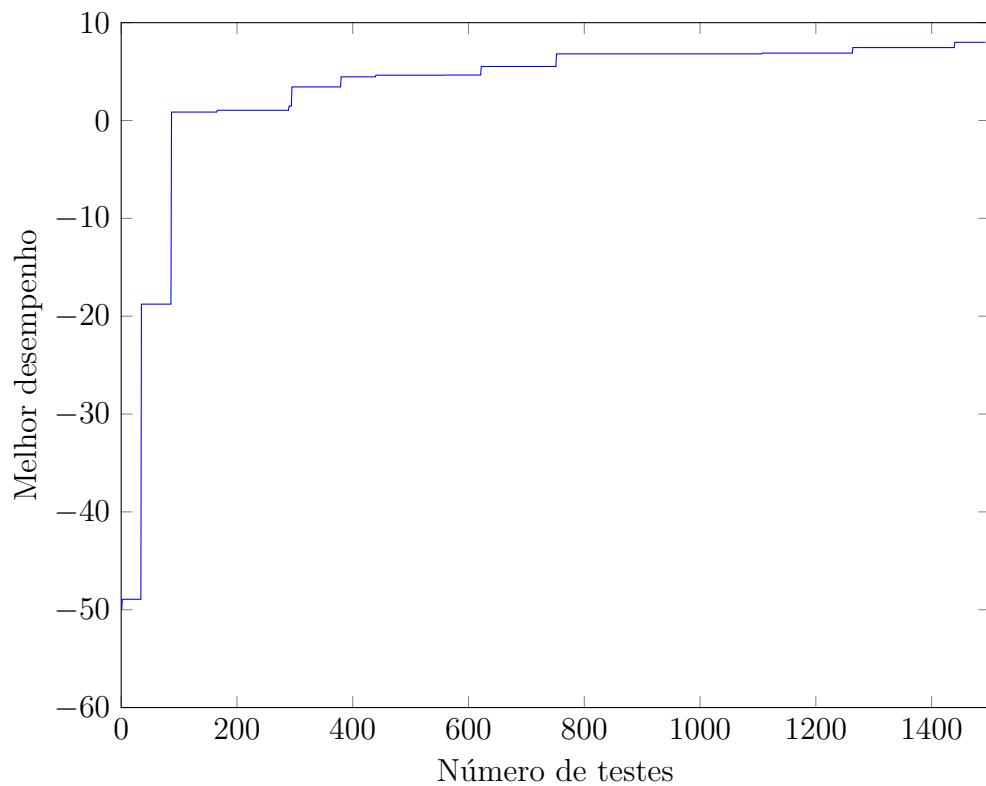


Figura 27: Resultados da otimização com Modelo Complexo e **PSO!**.

Por fim, as Figuras 28, 29 e 30 apresentam sequências de imagens para ilustrar a visualização das caminhadas aprendidas com **PSO!** para cada modelo. A taxa de quadros utilizada para a gravação foi de 15 quadros por segundo.

Constante	MS+AG	MB+AG	MC+AG	MS+PSO	MB+PSO	MC+PSO
O_c	-0,69	-1,2	-0,96	-0,64	-0,66	-0,87
A	0,23	0,33	0,37	0,27	0,5	0,51
B	0,43	0,37	0,68	0,35	0,56	0,7
O_j	0,98	1,6	0,99	0,85	0,83	1,23
C	0,4	0,51	0,76	0,3	0,6	0,46
$\tau = t_2/T$	0,38	0,25	0,22	0,39	0,38	0,34
T	0,34	0,1	0,4	0,44	0,39	0,38
D_+	—	0,58	0,58	—	0,17	0,54
D_-	—	0,14	0,55	—	0,57	0,39
E	—	—	0,04	—	—	0,05

Tabela 10: Limites para os valores das contantes dos modelos de caminhada.

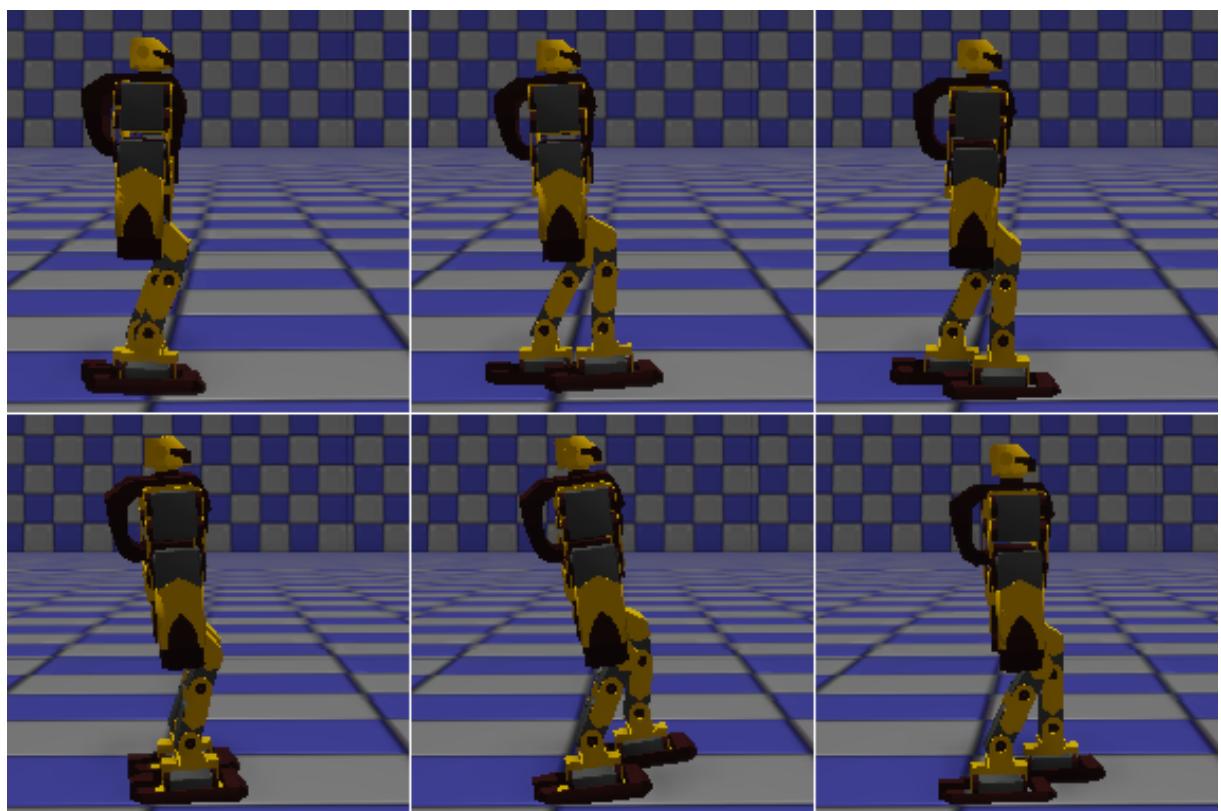


Figura 28: Sequência de imagens para ilustrar a caminhada aprendida com Modelo Simples e PSO!.

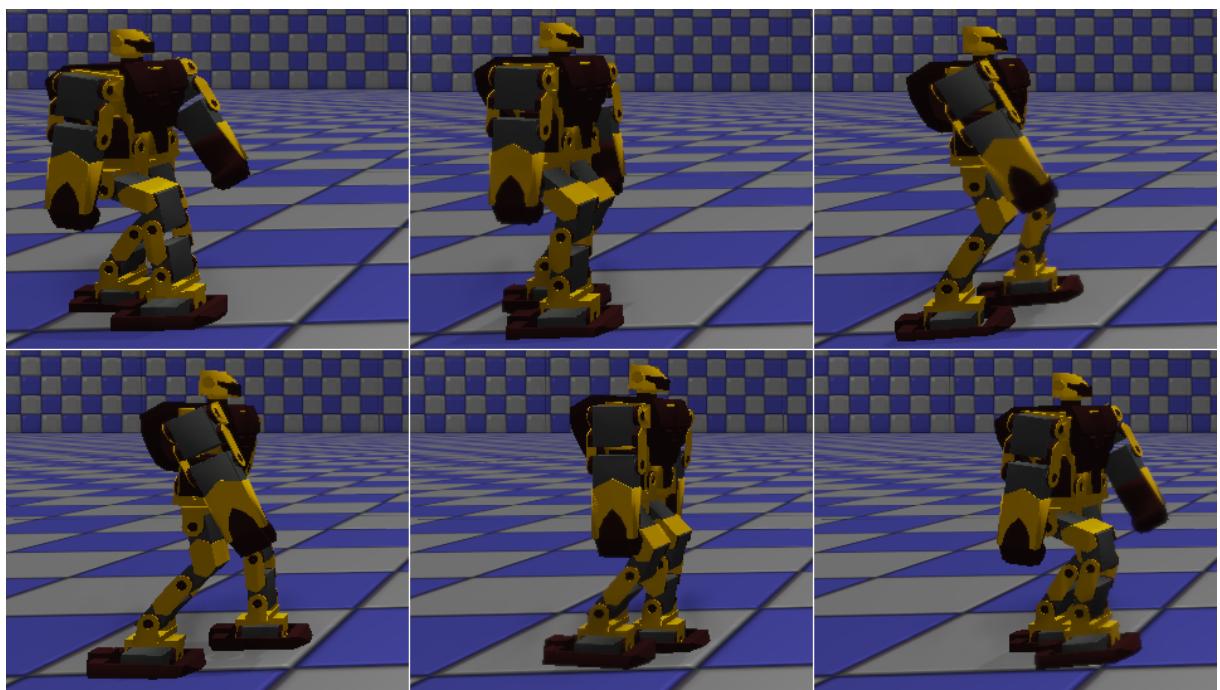


Figura 29: Sequência de imagens para ilustrar a caminhada aprendida com Modelo com Movimento de Braços e PSO!.

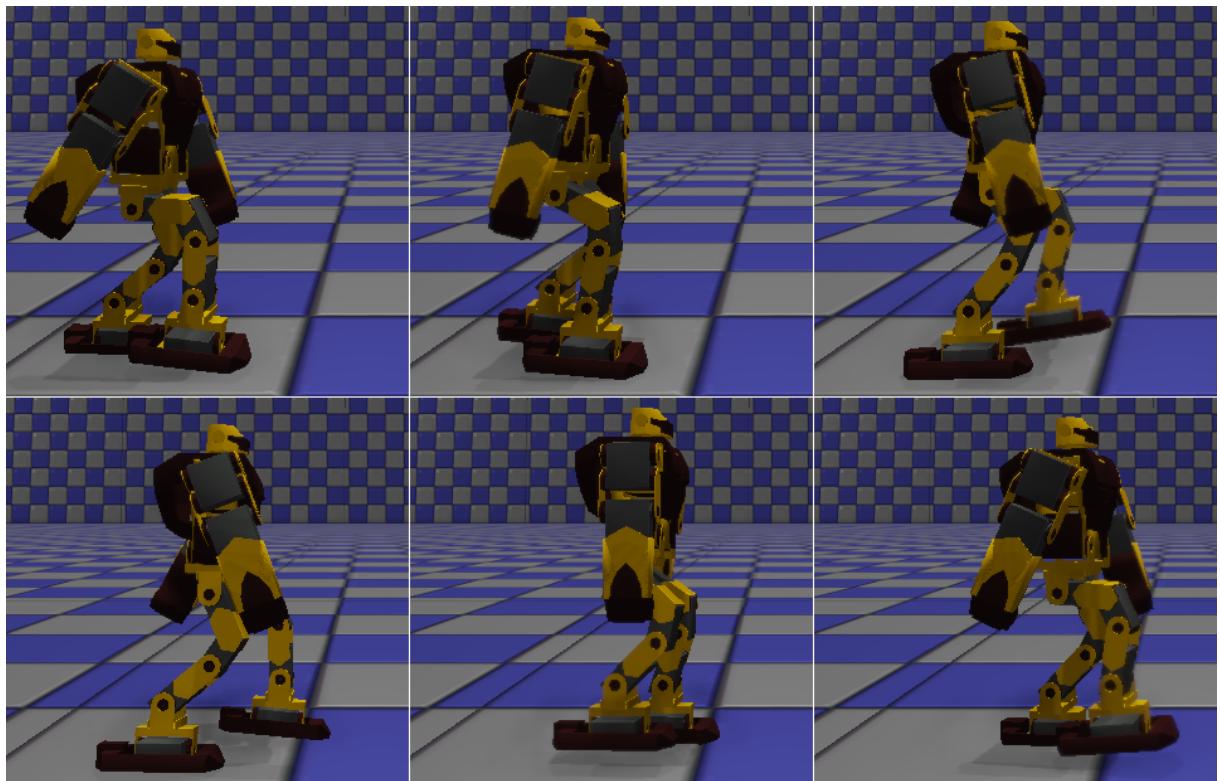


Figura 30: Sequência de imagens para ilustrar a caminhada aprendida com Modelo Complexo e PSO!.

10 Transferência para o Robô Real

10.1 Desafios Envoltos

Apesar do cuidado tomado em fazer um modelo físico o mais próximo possível do real e da PhysX ser uma das melhores *engines* de Física da atualidade, há diversas limitações na modelagem que não foram tratadas:

- Modelos físicos são aproximações dos modelos CAD;
- O coeficiente de atrito do solo usado para otimização no simulador não é necessariamente o mesmo que um solo em que o robô real caminhará;
- Embora sejam todos do mesmo modelo, os servomotores do robô real possuem pequenas diferenças de torque e velocidade devido a questões de construção;
- No robô real, existem folgas mecânicas que não são modeladas na simulação.

Locomoção humanóide é um problema complexo em que pequenas variações nas condições como as citadas acima podem tornar o comportamento muito diferente. Assim, espera-se que a caminhada com os parâmetros aprendidos em simulação não tenha um desempenho tão bom no robô real, mas que, após alguns ajustes manuais, obtenha-se uma boa solução para o problema.

10.2 Processo de Transferência

Como comentado na Seção ??, o plano inicial era utilizar o mesmo código C++ tanto em simulação quanto no robô real. Para isso, esperava-se substituir a placa original do Robonova (MR-C3024) por uma placa Gumstix (11). Esta placa roda uma versão embarcada do sistema operacional Linux, o que permitiria compilar e executar a maior parte do código do controlador construído para o robô simulado. Infelizmente, não foi

possível obter esse *hardware*. Desse modo, alterou-se o plano para programação na MR-C3024.

A programação para a MR-C3024 deve ser feita em RoboBASIC, conforme comentado na Seção 6.2. O recursos dessa linguagem são bem limitados e ela não possui suporte ao uso direto da função seno, que é necessária para a implementação das equações que descrevem as trajetórias angulares das juntas.

Portanto, decidiu-se criar um programa em linguagem C# que, a partir das equações do modelo de caminhada e um passo de tempo, gera código em RoboBASIC com os valores numéricos para as posições das juntas já calculados. O código gerado é formado basicamente por dois comandos da RoboBASIC: o MOVE, que envia novas posições para todos os servomotores, e DELAY, que bloqueia a execução por um tempo em milisegundos. A Listagem 10.1 mostra o código gerado para a caminhada com Modelo com Movimento de Braços.

Listagem 10.1: Código em RoboBASIC para a caminhada de Modelo com Movimento de Braços.

```
' 3 PASSOS INICIAIS

MOVE24 ,79,140,90, , ,100, , , , ,100, , , , , ,82,137,90, ,
DELAY 20
MOVE24 ,81,140,87, , ,101, , , , ,96, , , , , ,80,137,92, ,
DELAY 20
MOVE24 ,82,137,90, , ,100, , , , ,100, , , , , ,79,140,90, ,
DELAY 20
MOVE24 ,80,137,92, , ,96, , , , ,101, , , , , ,81,140,87, ,
DELAY 20
MOVE24 ,79,140,90, , ,100, , , , ,100, , , , , ,86,133,90, ,
DELAY 20
MOVE24 ,84,140,85, , ,102, , , , ,92, , , , , ,81,134,93, ,
DELAY 20
MOVE24 ,86,133,90, , ,100, , , , ,100, , , , , ,79,140,90, ,
DELAY 20
MOVE24 ,81,134,93, , ,92, , , , ,102, , , , , ,84,140,85, ,
DELAY 20
MOVE24 ,79,140,90, , ,100, , , , ,100, , , , , ,89,130,90, ,
DELAY 20
MOVE24 ,86,140,82, , ,103, , , , ,88, , , , , ,83,131,95, ,
```

```

DELAY 20
MOVE24 ,89,130,90, , ,100, , , , ,100, , , , , ,79,140,90, ,
DELAY 20
MOVE24 ,83,131,95, , ,88, , , , ,103, , , , , ,86,140,82, ,
DELAY 20

' MOVIMENTO PERIÓDICO

walk_loop:

MOVE24 ,79,140,90, , ,100, , , , ,100, , , , , ,89,130,90, ,
DELAY 20
MOVE24 ,89,140,80, , ,103, , , , ,84, , , , , ,84,127,97, ,
DELAY 20
MOVE24 ,92,127,90, , ,100, , , , ,100, , , , , ,79,140,90, ,
DELAY 20
MOVE24 ,84,127,97, , ,84, , , , ,103, , , , , ,89,140,80, ,
DELAY 20

GOTO walk_loop

```

Testes foram feitos para avaliar a velocidade de resposta do microcontrolador e dos servomotores na prática. Com os servomotores em velocidade máxima, determinou-se 20 milisegundos como um bom passo de tempo. Passos menores faziam os servomotores vibrarem excessivamente durante a operação.

Com isso, a primeira tentativa foi tentar a caminhada com “Modelo Simples” com os parâmetros aprendidos em simulação. A caminhada funcionou, mas o desempenho foi bem inferior ao da simulação: a caminhada ficou lenta, pouco estável (robô caia com frequência) e curvava excessivamente para a direita.

Através de alguns testes foi possível perceber que apenas a redução do período já melhorava a qualidade da caminhada. Empiricamente, chegou-se ao valor de 0,08 segundos como adequado para o período. A caminhada assim obtida mostrou-se bem estável, de modo que quedas se tornaram raras quando em solo plano.

Entretanto, ainda verificou-se que ela curvava muito. Atribui-se isso principalmente ao deslizamento no solo devido a uma amplitude muito grande dos passos. Desse modo, testes foram realizados reduzindo-se os valores das amplitudes de todas as juntas percen-

tualmente até encontrar um percentual de 60% de redução como adequado. A velocidade da caminhada assim obtida foi de 18,08 cm/s.

Então, implementou-se movimento de braços e verificou-se uma melhora de desempenho, com a velocidade passando a 21,49 cm/s. Também, a trajetória do robô passou a curvar menos.

A Tabela 11 resume as velocidades das caminhadas com os diferentes modelos. Para efeitos de comparação, mostra-se também as velocidades das caminhadas originais do kit Robonova e de uma versão melhorada disponível em (19). Observe que a caminhada final gerada nesse trabalho supera em velocidade todas as outras testadas. A Figura 31 apresenta uma sequência de imagens que ilustra a caminhada após todas as modificações feitas.



Figura 31: Sequência de imagens para ilustrar a caminhada final do Robonova real. A taxa de quadros usada para gravação foi de 30 quadros por segundo.

As medidas das velocidades das caminhadas foram feitas a partir de quanto o robô conseguia caminhar em 5 segundos. 10 testes foram feitos para cada tipo de caminhada. As medidas de distância foram feitas com uso de uma trena no chão e as de tempo com uso de um cronômetro digital. A Figura 32 mostra o ambiente do experimento montado.

Caminhada	Média da Velocidade (cm/s)	Desvio Padrão da Velocidade (cm/s)	Número de Quedas (em 10 testes)
Modelo Simples	18,08	1,05	0
Modelo com Movimento de Braços	21,49	0,9	0
Forward Walk (fabricante)	2,7	0,25	0
Fast Walk (fabricante)	6,35	3,12	5
New Fast Walk (19)	19,2	1,3	0

Tabela 11: Comparação entre diversas caminhadas para o Robonova-I.



Figura 32: Ambiente do experimento de medida de velocidade do robô real.

11 Validação com Outro Robô

Para verificar a validade das técnicas utilizadas neste trabalho em outro modelo de robô, foi feito um procedimento análogo com o robô Nao simulado do domínio da RoboCup 3D Soccer Simulation League.

11.1 RoboCup 3D Soccer Simulation League

A **Soccer 3D!** é uma das categorias da competição RoboCup (7). Na **Soccer 3D!**, 22 agentes (11 em cada time) jogam futebol em um ambiente simulado, seguindo regras semelhantes às do futebol humano.

O simulador de Robótica utilizado é o **SimSpark!** (16). Esse simulador utiliza a **ODE! (ODE!)** (25) como sua *engine* de Física, que permite uma simulação precisa de interações mecânicas, como atrito e colisão. Isso permite modelar robôs humanóides complexos, como o Albebaran Nao (17), que possui 22 graus de liberdade. A Figura 33 mostra o robô Nao simulado dentro do **SimSpark!**.



Figura 33: Robô Nao simulado chutando uma bola no **SimSpark!**.

Diferentemente do USARSim, em que a simulação ocorre em tempo real, sendo o passo de tempo limitado pelo poder de processamento da máquina, a simulação no **SimSpark!**

ocorre a um passo fixo de 20 milisegundos. A comunicação entre agentes e servidor é por **TCP/IP!** (**TCP/IP!**) de acordo com um protocolo específico.

Um dos objetivos da competição é fomentar pesquisas em controle de humanóides, assim a atuação é feita a partir do envio de comandos de velocidade para as juntas. Note que a implementação de controle de posição para as juntas fica a cargo do agente. A Figura 34 apresenta um esquema das juntas do Nao simulado.

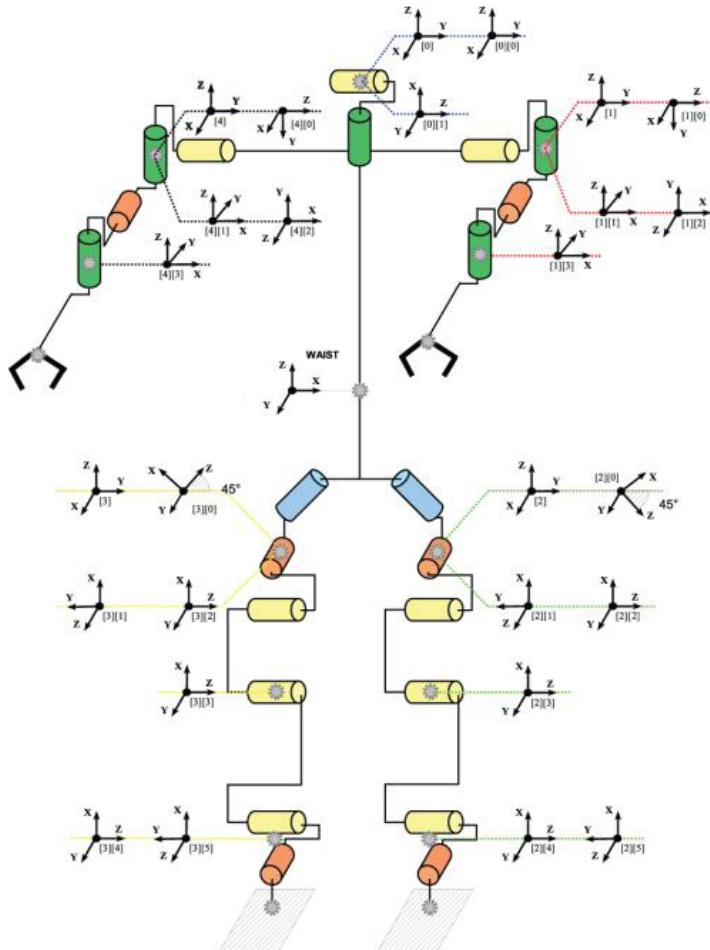


Figura 34: Esquema das juntas do Nao simulado do **SimSpark!**.

Já a visão é simplificada e não é necessário fazer processamento de imagem, pois o servidor envia medidas de distância e ângulo para cada objeto dentro do campo de visão do agente (que abrange 120° na horizontal). Outros sensores (acelerômetro, girômetro, sensores de pressão nos pés etc.) presentes no Nao real estão também presentes nessa versão simulada. Note que ruído é adicionado às medidas dos sensores para simular o que ocorre no caso de sensores reais. Ademais, para se comunicarem entre si, os agentes podem “falar” e “ouvir” mensagens limitadas a 20 bytes.

11.2 Implementação

Como trata-se de uma competição, a **Soccer 3D!** possui uma comunidade de tamanho razoável. Alguns times mais experientes liberam códigos de agentes base para servir de exemplo e facilitar a entrada de novos times.

Como o objetivo era apenas usar a **Soccer 3D!** para validação das técnicas utilizadas e não construir um novo programa de agente, optou-se pelo uso de um time base. Após a análise de alguns dos códigos providos em (24), escolheu-se o **magma-AF!** (**magma-AF!**), que é desenvolvido pelo time magmaOffenburg da Hochschule Offenburg University of Applied Sciences. O código do **magma-AF!** é escrito em Java com o uso de boas práticas de Engenharia de Software.

O **magma-AF!** é um agente funcional que joga futebol de modo autônomo com uma estratégia bem simples. Assim, várias das questões de baixo nível necessárias foram aproveitadas: comunicação com o servidor, controlador de posição, verificação de se o robô caiu etc.

Entretanto, no **SimSpark!**, o Nao não possui um sensor que dê a localização global do agente, assim como faz o GroundTruth do USARSim. Desse modo, o agente deve inferir sua posição a partir das medidas dos sensores que possui, principalmente da câmera. Considerando que o problema de Localização é um dos mais complexos da Robótica Móvel e que o **magma-AF!** usa um método muito simples de localização, as estimativas de posição feitas pelo agente certamente não seriam adequadas para o processo de treinamento.

A solução para isso foi implementar o que geralmente é chamado na liga de “treinador”. Esse treinador se conecta ao servidor como se fosse um monitor (um programa para visualização do jogo), logo recebe posições e orientações globais de todos os objetos. Com isso, pode-se obter a posição do agente desejado. O treinador implementado foi inspirado no código provido em (26). Para facilitar a troca de informações, optou-se por implementar o treinador no mesmo processo Java do agente, mas sendo executado em uma *thread* própria.

Depois disso, a caminhada com Modelo Complexo foi implementada conforme as estruturas providas pelo **magma-AF!**.

11.3 Processo de Otimização

O processo de otimização foi montado de modo semelhante ao explicado no Capítulo 9, inclusive a função de medida de qualidade utilizada foi a mesma. No caso, executou-se otimização apenas para o Modelo Complexo e com **PSO!**. Os parâmetros usados para o **PSO!** foram os apresentados na Tabela 12. Os limites para do espaço de busca são mostrados na Tabela 13.

Parâmetro	Valor
P	50
ω	0,7
φ_p	0,5
φ_g	0,8
ε	0,8

Tabela 12: Valores dos parâmetros da execução de **PSO!** para otimização dentro do domínio da **Soccer 3D!**.

Constante	Mínimo	Máximo
O_c	-1,0	0
A	0,01	1,0
B	0,01	1,0
O_j	0	1,5
C	0	1,5
$\tau = t_2/T$	0,1	0,9
T	0,1	2,0
D_+	0	1
D_-	0	1
E	0	0,5

Tabela 13: Limites do espaço de busca para no caso da otimização realizada dentro da domínio da **Soccer 3D!**.

A Figura 35 apresenta como o melhor desempenho até então varia com o número de testes. A Tabela 14 mostra os valores da melhor partícula encontrada, que foi avaliada com qualidade igual a 9,39. Por fim, a Figura 36 mostra uma sequência de imagens apresentando a caminhada aprendida.

Para avaliar o desempenho da caminhada aprendida, decidiu-se compará-la com a caminhada provida no **magma-AF!**. Assim, executou-se 10 experimentos simulados de 15 segundos para cada caminhada. Ao fim dos 15 segundos, a distância percorrida pelo robô no sentido positivo do eixo X era medida. A Tabela 15 resume os resultados obtidos.

Constante	Valor
O_c	-0,36
A	0,51
B	0,21
O_j	0,81
C	0,72
$\tau = t_2/T$	0,36
T	0,27
D_+	0,23
D_-	0,31
E	0,12

Tabela 14: Parâmetros aprendidos para o Modelo Complexo executado no Nao simulado do **SimSpark!**.

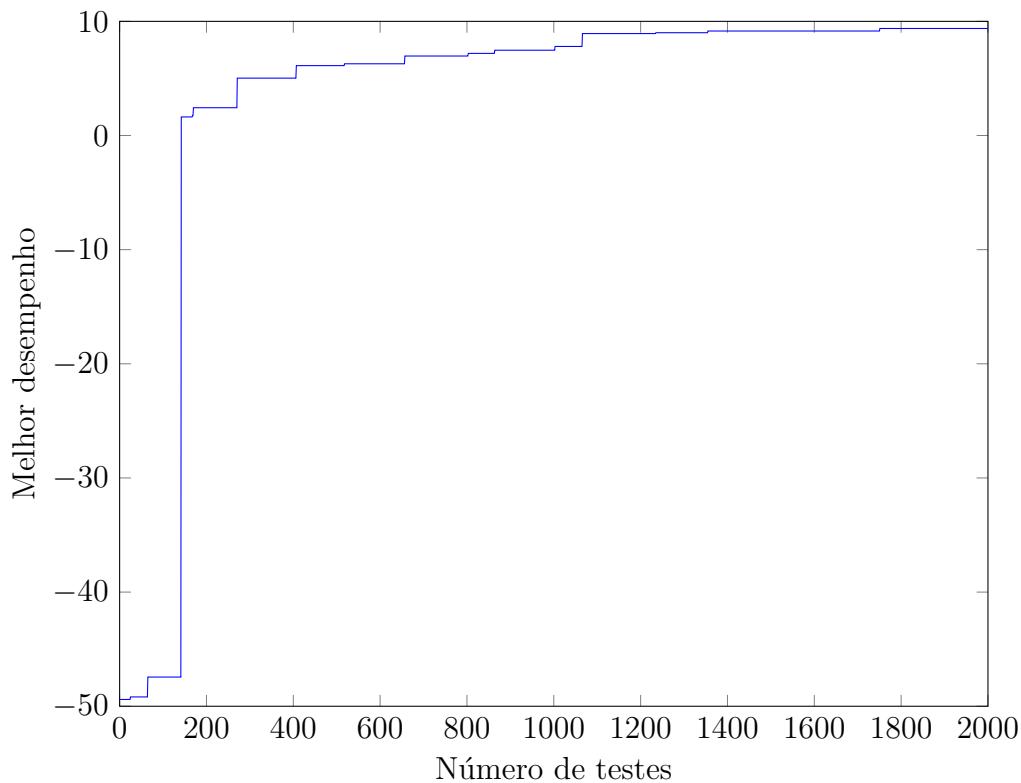


Figura 35: Resultados da otimização com **PSO!** realizada no ambiente da **Soccer 3D!**.

Observe que a caminhada aprendida é bem superior à do time base, sendo 89% mais rápida.

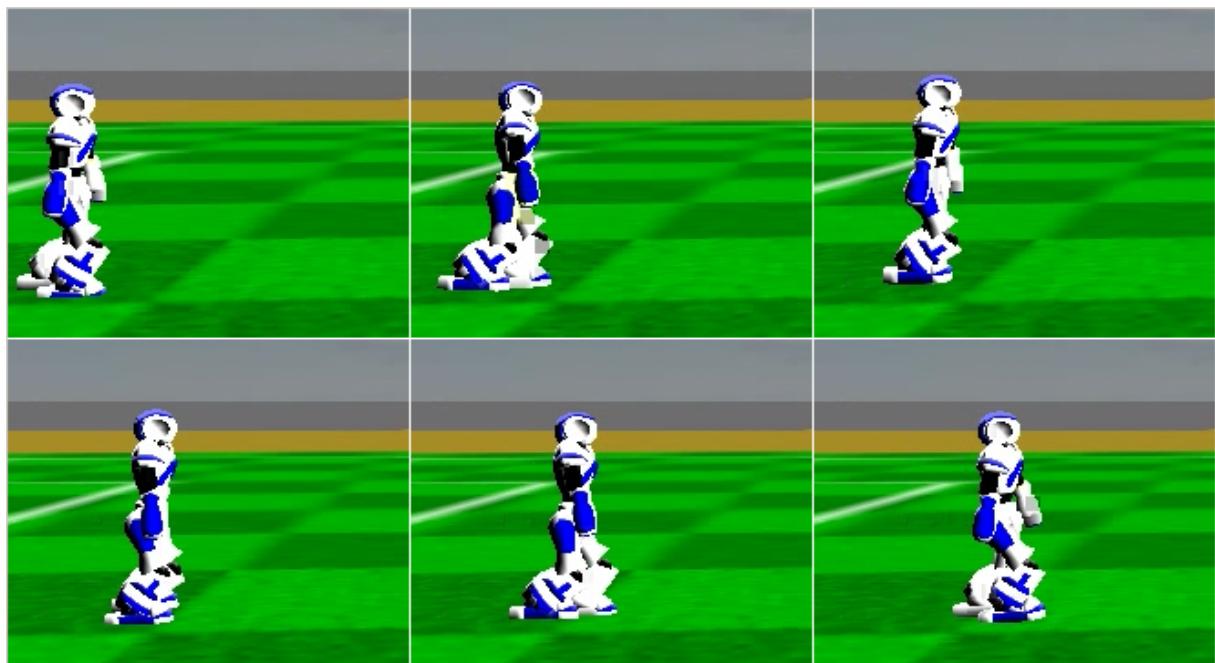


Figura 36: Sequência de imagens para ilustrar a caminhada aprendida para o Nao simulado do **SimSpark!**. Há uma diferença de tempo de 0,1 segundo entre uma imagem e a posterior.

	Caminhada	Média	Desvio Padrão
Modelo Complexo	7,41	0,55	
magma-AF	3,93	0,4	

Tabela 15: Resultados de medidas de distâncias percorridas no sentido positivo do eixo X. Realizou-se 10 experimentos de 15 segundos para cada caminhada.

12 Conclusões e Trabalhos Futuros

12.1 Conclusão

Nesse trabalho, caminhadas rápidas e estáveis foram confeccionadas para um robô Hitec Robonova-I. Para isso, usou-se um modelos parametrizados baseados em **SFT!** (**SFT!**) propostos em (22, 21, 23). Na abordagem seguida, parâmetros adequados para os modelos foram aprendidos com uso de 2 algoritmos de otimização: **AG!** (**AG!**) e **PSO!** (**PSO!**). Como aprendizado de locomoção diretamente no robô seria inconveniente e poderia provocar danos ao *hardware*, construiu-se um modelo do Robonova-I no **USARSim!** (**USARSim!**) e realizou-se o processo de aprendizado em simulação. Então, adaptou-se as caminhadas aprendidas para o robô real por ajustes manuais. Testes realizados com a melhor caminhada obtida mostraram que seu desempenho é bem superior à caminhada original do fabricante.

Posteriormente, uma validação adicional foi feita no domínio da **Soccer 3D!** (**Soccer 3D!**), que é uma categoria de futebol de robôs humanóides simulado da competição RoboCup. A **Soccer 3D!** é baseada no **SimSpark!** (**SimSpark!**) e atualmente usa um modelo simulado do robô Aldebaran Nao. Para evitar o desenvolvimento de uma interface com o **SimSpark!**, como foi feito no caso do **USARSim!**, utilizou-se o código base **magma-AF!** (**magma-AF!**). Assim, o processo de aprendizado foi executado e a melhor caminhada obtida se mostrou rápida e estável, inclusive superando significativamente o desempenho da caminhada provida no código base.

12.2 Trabalhos Futuros

Desde seu início, a intenção desse trabalho era criar uma estrutura para diversos outros trabalhos de pesquisa em robôs humanóides na instituição. Exatamente por isso, houve um cuidado especial em deixar os desenvolvimentos feitos bem documentados e em se fazer uma boa estrutura de programa de agente, embora essas questões não fossem tão

importantes dentro do escopo desse trabalho.

Em relação ao controle de humanóides especificamente, são propostas as seguintes frentes de trabalho futuro:

- Implementação de outros movimentos, como levantar-se, girar o corpo etc.;
- Idealização e execução de experimentos e métricas que avaliem o quanto o comportamento do robô simulado difere do real. A partir disso, seria interessante a realização de ajustes finos no modelo de simulação a fim de se obter um comportamento o mais próximo possível do robô real;
- Em otimização heurística, é difícil determinar de antemão qual o melhor método para um determinado problema. Assim, seria interessante tentar utilizar outros métodos além de **AG!** e **PSO!**;
- Estudo, desenvolvimento e implementação de novos modelos de caminhada. Em especial, caminhadas omnidirecionais, por serem mais flexíveis;
- Estudo, desenvolvimento e implementação de estratégias de balanço com base em sensores para “fechar a malha” da caminhada e torná-la mais estável e menos suscetível a pertubações como irregularidades no solo. Para isso, seria necessário antes a adição de sensores julgados interessantes, como acelerômetro, girômetro e sensores de pressão nos pés.

Em relação a outras linhas ligadas a humanóides, o autor tem interesse especial em continuar este trabalho em uma pós-graduação no sentido de construir robôs autônomos jogadores de futebol para a liga Humanoid KidSize da RoboCup, o que na realidade era objetivo implícito desde o início. Inclusive, um grupo de alunos de já está formado dentro da ITAndroids, grupo para competições de Robótica do ITA, para este projeto.

Referências

- 1 Hitec MR-C3024 to PC Protocol for RoboNova.
- 2 Autodesk. Autodesk 3D Studio Max. <http://usa.autodesk.com/3ds-max/>, 2012.
- 3 Autodesk. Autodesk Autocad. <http://usa.autodesk.com/autocad/>, 2012.
- 4 Autodesk. Autodesk FBX. <http://usa.autodesk.com/fbx/>, 2012.
- 5 R. Auvray, B. Fabre, and P.-Y. Lagrée. Harmonics generation in flute-like instruments. In *Proceedings of the International Symposium on Musical Acoustics*, pages 40–43, 2014.
- 6 David Buckley. MR-C3024 ROBOT CONTROLLER.
- 7 RoboCup Federation. RoboCup. <http://www.robocup.org/>, 2012. [Online; acessado em 25 de outubro de 2012].
- 8 N. H. Fletcher. Acoustical correlates of flute performance technique. *Journal of the Acoustical Society of America*, pages 233–237, 1975.
- 9 N. H. Fletcher. Air flow and sound generation in musical wind instruments. *Annual Review of Fluid Mechanics*, pages 123–146, 1979.
- 10 Epic Games. Unreal Development Kit. <http://www.unrealengine.com/udk/>, 2008–2012. [Online; acessado em 25 de outubro de 2012].
- 11 Gumstix. Gumstix. <http://www.gumstix.com/>, 2012. [Online; acessado em 25 de outubro de 2012].
- 12 J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
- 13 Jackson Paul Matsuura, Esther Luna Colombini, and Alexandre da Silva Simões. Desenvolvimento de Robô Humanóide Autônomo de Baixo Custo. In *Simpósio Brasileiro de Automação Inteligente (SBAI) 2007*, 2007.
- 14 Nokia. Qt. <http://qt.nokia.com/>, 2012. [Online; acessado em 28 de outubro de 2012].
- 15 Nvidia. Physx. <http://www.geforce.com/hardware/technology/physx>, 2012. [Online; acessado em 28 de outubro de 2012].
- 16 Oliver Obst and Markus Rollmann. SPARK - A Generic Simulator for Physical Multiagent Simulations. *Computer Systems Science and Engineering 20*, pages 347–356, 2005.

- 17 Albebaran Robotics. Nao H25: Humanoid Robot Platform. <http://www.aldebaran-robotics.com/en/Discover-NAO/nao-datasheet-h25.html>, 2012. [Online; acessado em 25 de outubro de 2012].
- 18 Hitec Robotics. Robonova-I English Instruction Manual.
- 19 Hitec Robotics. New source code with fast walk and fast turn. http://www.robonova.de/store/support/index.php?_m=downloads&_a=viewdownload&downloaditemid=93&nav=0,5,7, 2006.
- 20 Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2009.
- 21 Nima Shafii, Siavash Aslani, Omid Mohamad Nezami, and Saeed Shiry. Evolution of Biped Walking Using Truncated Fourier Series and Particle Swarm Optimization. In *Baltes, J., Lagoudakis, M.G., Naruse, T.; Ghidary, S.S. (eds.) RoboCup 2009*, pages 344–354, 2010.
- 22 Nima Shafii, Ali Khorsandian, Abbas Abdolmaleki, and Bahram Jozi. An Optimized Gait Generator Based on Fourier Series Toward Fast and Robust Biped Locomotion Involving Arms Swing. In *Proceedings of the IEEE International Conference on Automation and Logistics*, 2009.
- 23 Nima Shafii, Luís Paulo Reis, and Nuno Lau. Biped Walking Using Coronal and Sagittal Movements Based on Truncated Fourier Series. In *RoboCup 2010: Robot Soccer World Cup XIV, Lecture Notes in Computer Science*, pages 324–335, 2010.
- 24 SimSpark. SimSpark Agents. <http://simspark.sourceforge.net/wiki/index.php/Agents>, 2012. [Online; acessado em 25 de outubro de 2012].
- 25 Russell Smith. Open Dynamics Engine. <http://www.ode.org/>, 2012. [Online; acessado em 25 de outubro de 2012].
- 26 TinMan. TinMan. <http://code.google.com/p/tin-man/>, 2012. [Online; acessado em 25 de outubro de 2012].
- 27 Sander van Noort and Arnoud Visser. Validation of the dynamics of an humanoid robot in USARSim. In *Proceedings of the Performance Metrics for Intelligent Systems Workshop (PerMIS'12)*, March 2012.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO TC	2. DATA	3. REGISTRO N°	4. N° DE PÁGINAS N PAGINAS
5. TÍTULO E SUBTÍTULO: Desenvolvimento de um classificador de postagens em rede social utilizando Processamento de Linguagem Natural			
6. AUTOR(ES): Luiz Filipe Martins Ramos Bernardo Monteiro Rufino			
7. INSTITUIÇÃO(ES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ES): Instituto Tecnológico de Aeronáutica - ITA			
8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR: Inteligência Artificial, Processamento de Linguagem Natural, Redes Bayesianas			
9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO: Solicite preenchimento dos campos 2, 3 e 9 – envie este formulário para doc.pt@ita.br			
10. APRESENTAÇÃO:		<input checked="" type="checkbox"/> Nacional	<input type="checkbox"/> Internacional
ITA, São José dos Campos. Curso de Graduação em Engenharia de Computação. Orientador(es): Prof. Dr. Paulo André Lima de Castro. Publicado em 2015			
11. RESUMO:			
12. GRAU DE SIGILO: <input checked="" type="checkbox"/> OSTENSIVO <input type="checkbox"/> RESERVADO <input type="checkbox"/> SECRETO			