Universidade de Brasília

Relatório de Trabalho de Graduação 1

Geração de marchas bioinspiradas para um robô quadrúpede utilizando modelos geradores.

Pedro Matheus Filadelfo dos Santos - 13/0060674
Vitor Alves Duarte - 13/0018546

Professor: Alexandre Romariz

Data de realização: 08/12/2017

1 Resumo

Este trabalho consiste de uma compilação dos estudos realizados para se dar início ao projeto de graduação. Como se deseja gerar uma marcha bioinspirada em um robô quadrúpede utilizando modelos geradores, revisou-se sobre métodos de aplicação desses modelos, como máquina restrita de Boltzmann, *deep belief networks* e redes geradoras concorrentes. Explica-se a metodologia, de como isto será aplicado às marchas do robô, ao utilizar uma biblioteca de inteligência artificial, o TensorFlow e um simulador de dinâmicas robóticas.

2 Introdução

Robôs móveis articulados são de grande utilidade para diversas de tarefas, uma vez que sua configuração permite uma maior adaptação as mudanças do ambiente. Em um comparativo com os robôs tradicionais movidos por rodas, tem-se uma maior acessibilidade, em detrimento da diminuição da velocidade e aumento da dificuldade de estabilização.

O robô articulado, como um todo, possui diversas peculiaridades que dificultam a implementação de uma movimentação mais veloz e estável. Uma delas é a presença de não linearidades. Os modelos propostos, que se baseiam no aprendizado de maquina, permitem lidar com essas não linearidades. Oferecendo uma forma prática e confiável de otimizar a movimentação do robô, por meio da criação e avaliação de uma marcha. Essa marcha consiste em uma sequência de movimentações necessárias de cada estrutura articulada para prover a movimentação do robô.

2.1 Plataforma quadrúpede

O automato que será utilizado no projeto é do tipo quadrúpede e foi disponibilizado pelo Laboratório de Automação e Robótica (LARA). Diversos projetos foram realizados utilizando a mesma plataforma, dentre eles o trabalho mais recente, finalizado em 2017, propôs a criação de marchas utilizando algoritmos genéticos [1]. A Figura 1 mostra o aspecto geral da plataforma no fim desse projeto.

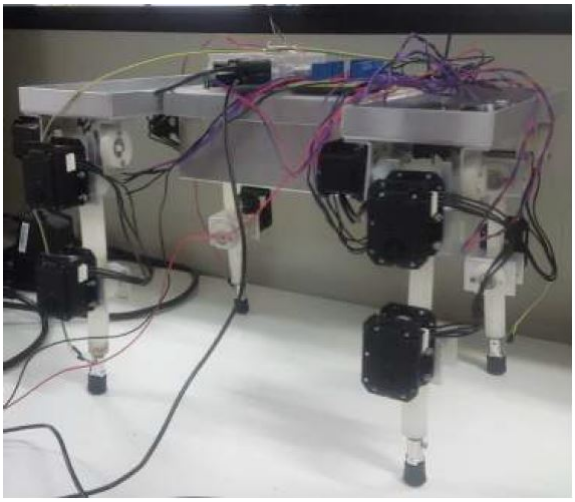

Figura 1: Planta quadrupede do LARA [1].

O sistema embarcado da plataforma é dado por um *Raspberry pi* e um Arduino. A comunicação entre esses dispositivos é dada de forma direta, promovendo uma comunicação do tipo mestre e escravo, onde o Arduino atua como escravo do *Raspberry*. Os dados para a avaliação dos estados são fornecidos por um conjunto de sensores resistivos de força, *encoders* e acelerômetro.

O robô possui três articulações em cada uma de suas quatro patas. Cada articulação pode ser controlada por meio de um servo, totalizando doze graus de liberdade.

A arquitetura de controle de estabilidade da plataforma foi implementada em trabalhos anteriores, portanto assume-se que ao definir uma posição de referência para as juntas, o controle irá garantir que essa posição foi atingida, exceto quando essa posição coloca em risco a estabilidade do sistema.

2.2 Modelos geradores

Neste trabalho pretende-se usar modelos geradores para criar uma marcha bioinspirada para o robô quadrúpede. Dessa forma realizou-se um estudo sobre diferentes técnicas de aplicação desses modelos, tais como: Modelos geradores profundos [4] que englobam máquinas restritas de Boltzmann e suas generalizações; e redes geradoras concorrentes [5] - *Generative Adversarial Nets (GANs)* - que possuem um modelo gerador G, o qual captura a distribuição dos dados de entrada, e um modelo discriminador D que deve estimar a probabilidade de uma amostra ser proveniente dos dados de treinamento ou de G. Explora-se também suas variações e algumas técnicas de aprimoramento.

3 Revisão Bibliográfica

3.1 Modelos geradores profundos

Muitos problemas da área de inteligência artificial exigem que sistemas extraiam representações significativas de entradas sensoriais complexas, como por exemplo reconhecimento de objetos, de discurso, compreensão de linguagem etc. Além disso, argumentos biológicos sugerem que esse sistemas devem possuir arquiteturas profundas, ou seja, modelos compostos de várias camadas de processamento não-linear [4].

Dessa forma, recentemente foi introduzido um algoritmo de aprendizagem não supervisionado para modelos geradores profundos, as *deep belief networks* (DBNs), que são modelos probabilísticos gráficos que contém múltiplas camadas de variáveis escondidas. Cada camada não linear captura padrões mais complexos de dados. A característica chave de uma DBN é seu treinamento camada-por-camada, que pode ser repetido várias vezes para se aprender um modelo probabilístico profundo e hierárquico. As maiores motivações quanto a essas redes são: O aprendizado camada-por-camada pode achar um bom conjunto de parâmetros do modelo relativamente rápido; O algoritmo faz uso eficiente de dados não rotulados, usando os poucos dados rotulados para se realizar um ajuste fino para uma tarefa específica; Existe um jeito fácil fazer uma inferência aproximada [4].

Faz-se então um estudo sobre o funcionamento dessas redes, porém tendo em conhecimento que elas possuem como base a máquina restrita de Boltzmann (RBM), que será a primeira a ser apresentada.

3.1.1 Máquina Restrita de Boltzmann (RBM)

Para se construir uma DBN deve-se primeiro conhecer a máquina restrita de Boltzmann (RBM). Ela é um modelo baseado em energia, que associa uma valor escalar para cada configuração de variáveis de interesse. O aprendizado consiste em modificar a função de energia para se obter os resultados desejáveis [8]. Assim, é definido uma distribuição probabilística pela função de energia:

$$p(x) = \frac{e^{-E(x)}}{Z} \tag{1}$$

Em que Z é um fator normalizador, conhecido como função de partição:

$$Z = \sum_{x} e^{-E(x)} \tag{2}$$

Um modelo baseado em energia pode aprender utilizando o gradiente descendente da log-verossimilhança negativa empírica dos dados de treinamento. Assim, tem-se que:

$$L(\theta, D) = \frac{1}{N} \sum_{x^{(i)} \in D} logp(x^{(i)}) \tag{3}$$

$$\ell(\theta, D) = -L(\theta, D) \tag{4}$$

Em que (3) é a log-verossimilhança e (4) é a função de perda, e usando o gradiente estocástico

$$-\frac{\delta logp(x^{(i)})}{\delta \theta} \tag{5}$$

em que θ são os parâmetros do modelo.

Em alguns casos não se observa x por completo, ou apenas se deseja inserir algumas variáveis não observáveis para aumentar o potencial do modelo. Então se considera uma parte observada x e uma escondida h, assim:

$$P(x) = \sum_h P(x, h) \tag{6}$$

E para mapear isso como na (1) se introduz o conceito de energia livre, definida como:

$$F(x) = -log \sum_h e^{-E(x,h)} \tag{7}$$

Assim tem-se

$$P(x) = \frac{e^{-F(x)}}{Z} \tag{8}$$

com $Z = \sum_x e^{-F(x)}$. Dessa forma, usando o gradiente da log-verossimilhança negativa dos dados, tem-se que:

$$-\frac{\delta logp(x)}{\delta \theta} = \frac{\delta F(x)}{\delta \theta} - \sum_{\tilde{x}} p(\tilde{x}) \frac{\delta F(\tilde{x})}{\delta \theta} \tag{9}$$

É possível ver que o gradiente possui dois termos, relativos a fase positiva e negativa, que se referem ao efeito de aumentar a probabilidade de dados de treino (termo positivo) e de diminuir a probabilidade de amostras geradas pelo modelo (termo negativo).

Geralmente é muito difícil determinar esse gradiente analiticamente uma vez que envolve a computação da esperança sobre todas as possíveis configurações da entrada x sob a distribuição P do modelo. O primeiro passo para tornar

isso tragável é estimar a esperança usando um número fixo de amostras do modelo. Amostras usadas para estimar a fase negativa do gradiente são chamadas de partículas negativas N, assim têm-se

$$-\frac{\delta logp(x)}{\delta\theta} \approx \frac{\delta F(x)}{\delta\theta} - \frac{1}{|N|}\sum_{\tilde{x}\in N}\frac{\delta F(\tilde{x})}{\delta\theta} \tag{10}$$

Assim, basta apenas determinar como extrair as partículas negativas N para praticamente se obter um algoritmo estocástico para o aprendizado de um modelo baseado em energia (EBM). Na literatura, se encontra método de corrente de Markov e Monte Carlo, que se encaixam bem em RBMs.

Sendo assim, finalmente pode-se definir que uma RBM é forma particular do campo aleatório de Markov e é uma rede que contém variáveis visíveis v e escondidas h, que a sua função energia é dada por:

$$E(v,h,\theta) = -v^T W h - b^T v - a^T h = -\sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} v_i h_j - \sum_{i=1}^{D} b_i v_i - \sum_{j=i}^{F} a_j h_j \tag{11}$$

Em que $\theta = W, b, a$ são os parâmetros do modelo, W_{ij} são as interações simétricas entre a variáveis visível i e a variável escondida j, e b_i e a_j são os termos *bias*. Pode-se visualizar graficamente uma RBM na figura 2.

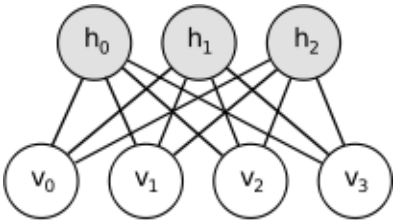

Figura 2: Esquema gráfico de uma RBM. Retirado de [8].

Assim, utilizando as equações (7), (8)e (11), tem-se que

$$P(v;\theta) = \frac{1}{Z(\theta)} e^{b^T v} \prod_{j=1}^{F} (1 + e^{(a_j + \sum_{i=1}^{D} W_{ij} v_i)}) \tag{12}$$

Devido a estrutura da RBM, as variáveis visíveis e escondidas na condicionalmente dependentes entre si, assim pode-se escrever:

$$p(h|v;\theta) = \prod_j p(h_j|v)$$

$$p(v|h;\theta) = \prod_i p(v_i|h)$$

É muito comum se utilizar o caso binário, em que $v_i e h_i \in 0, 1$, e da equação (12) pode-se fazer

$$P(h_j = 1|v) = g(a_j + \sum_i W_{ij} v_i)$$

$$P(v_i = 1|h) = g(b_i + \sum_j W_{ij} h_j)$$

Em que g é a função logística $g(x) = \frac{1}{1+e^{-x}}$. Assim, utilizando a equação (12) novamente, obtém-se os gradientes da log-verossimilhança para uma RBM com unidades binárias:

$$-\frac{\delta log p(v)}{\delta W_{ij}} = E_v[p(h_j|v)v_i] - v_i^{(j)} g(W_j v^{(j)} + a_j) \tag{13}$$

$$-\frac{\delta log p(v)}{\delta a_j} = E_v[p(h_j|v)] - g(W_j v^{(j)})$$

$$-\frac{\delta log p(v)}{\delta b_i} = E_v[p(v_i|h)] - v_i^{(j)}$$

Por fim, deve-se amostrar $P(x)$. Para tanto, pode-se usar uma cadeia de Markov para a convergência, utilizando a amostragem de Gibbs como operador de transição [8]. Dessa forma variáveis visíveis são coletadas simultaneamente dado valores fixos de unidades escondidas, e vice-versa, assim define-se:

$$h^{n+1} \approx g(W' v^{(n)} + a)$$

$$v^{n+1} \approx g(W' h^{(n+1)} + b)$$

Em que n é a etapa na cadeia de Markov. Quando $t \to \infty$, $(v^{(t)}, h^{(t)})$ são garantidamente amostras acuradas de $p(v, h)$. Isso, naturalmente, seria muito custoso, computacionalmente.

Para a acelerar o processo de amostragem pode-se usar a técnica *Constrastive Divergent* (CD-k), em que se inicializa a cadeia de Markov com um exemplo de treinamento. E assim, CD não espera a cadeia convergir, obtendo as amostras depois de k passos da amostragem de Gibbs. Viu-se que, na prática, $K = 1$ tem funcionado muito bem.

3.1.2 Deep Belief Networks (DBNs)

Uma única camada de características binárias não é a melhor forma de se entender a estrutura de dados de entrada de grandes dimensões [4]. Para isso pode-se utilizar mais camadas binárias escondidas, interligadas, construindo assim uma DBN.

Sendo assim, DBNs são modelos geradores probabilísticos que contém várias camadas de variáveis escondidas, as quais capturam correlações de grande ordem entre as atividades de características de camadas inferiores. Em uma rede de duas camadas de variáveis escondidas e uma de visíveis, as duas camadas de cima forma uma RBM. A característica chave desse algoritmo é o treinamento camada-por-camada, que pode ser repetido várias vezes pra se aprender um modelo hierárquico profundo. Além disso se permite um jeito eficiente de realizar inferência aproximada, a qual requere apenas uma passagem de baixo para cima para se inferir os valores das variáveis de níveis altos.

Considerando agora uma DBN com duas camadas de camadas escondidas $h^{(1)}, h^{(2)}$, considerando que a segunda camada possui o mesmo número de variáveis da camada de variáveis visíveis. Sendo assim, as duas camadas de cima formam uma RBM e as camadas inferiores formam uma rede sigmóide direta. Assim, a distribuição conjunta de v, $h^{(1)}$ e $h^{(2)}$ é:

$$P(v, h^{(1)}, h^{(2)}; \theta) = P(v|h^{(1)}; W^{(1)})P(h^{(1)}, h^{(2)}; W^{(2)}) \tag{14}$$

Em que $\theta = \{W^{(1)}, W^{(2)}\}$ são os parâmetros do modelo, $P(v|h^{(1)}; W^{(1)})$ é a rede sigmóide direta e $P(h^{(1)}, h^{(2)}; W^{(2)})$ é a distribuição conjunta definida pela segunda camada. Então, têm-se

$$P(v|h^{(1)} = \prod_i p(v_i|h^{(1)}; W^{(1)}), p(v_i = 1|h^{(1)}; W^{(1)}) = g(\sum_j W_{ij} h_j^{(1)}) \tag{15}$$

$$P(h^{(1)}, h^{(2)}; W^{(2)}) = \frac{1}{Z(W^{(2)})} e^{(h^{(1)^T} W^{(2)} h^{(2)})} \tag{16}$$

Para se realizar o treinamento camada-por-camada, considera-se uma DBN com duas camadas escondidas com parâmetros amarrados $W^{(2)} = W^{(1)^T}$. Dessa forma, usando as equações (14) - (16), têm-se que a distribuição conjunta sobre $v, h^{(1)}$ da DBN é:

$$P(v, h^{(1)}; \theta) = \frac{1}{Z(W^{(1)})} e^{(\sum_{ij} W_{ij}^{(1)} v_i h_j^{(1)})} \tag{17}$$

Que é idêntico a distribuição sobre $v, h^{(i)}$ definido para uma RBM.

Assim, para realizar o treinamento camada-por-camada, usa-se uma pilha de RBMs, treinando, primeiramente, a camada inferior com parâmetros $W^{(1)}$, como descrito na seção anterior. Depois, se inicializa os pesos da segunda camada (superior) com $W^{(2)} = W^{(1)^T}$, garantindo assim, que a segunda camada da DBN é tão boa quanto a primeira.

Para uma distribuição aproximada $Q(h^{(1)}|v)$ a log-verossimilhança da DBN com duas camadas escondidas tem o seguinte formato:

$$\begin{aligned} logP(v;\theta) &= log\sum_{h^{(1)}} Q(h^{(1)})\frac{P(v,h^{(1)};\theta)}{Q(h^{(1)})} \geq \sum_{h^{(1)}} Q(h^{(1)})\left[log\frac{P(v,h^{(1)};\theta)}{Q(h^{(1)})}\right] \\ &= \sum_{h^{(1)}} Q(h^{(1)})\left[logP(h^{(1)};W^{(2)}) + logP(h^{(1)};W^{(1)})\right] + H(Q(h^{(1)})) \end{aligned} \tag{18}$$

Em que $H(.)$ é a função de entropia. A estratégia para o aprendizado do algoritmo é fixar o parâmetro $W^{(i)}$ e tentar aprender o melhor modelo para $P(h^{(i)};W^{(2)})$, maximizando o limite inferior variante da equação (18) com respeito a $W^{(i)}$. Isso então significa maximizar

$$\sum_{h^{(1)}} Q(h^{(i)})logP(h^{(1)};W^{(1)}) \tag{19}$$

Que é equivalente a máxima verossimilhança do treino da segunda camada com vetores $h^{(1)}$ retirados de $Q(h^{(1)}|v)$ como dado de entrada. Dessa forma, um algoritmo para o treinamento recursivo do aprendizado para uma DBN é:

1. Ajustar os parâmetros $W^{(1)}$ da primeira camada RBM aos dados;
2. Fixar o vetor $W^{(1)}$, e usar amostras $h^{(1)}$ de $Q(h^{(1)}|v) = P(h^{(1)}|v, W^{(1)})$;
3. Fixar os parâmetros $W^{(2)}$ que define a segunda camada de características e usar amostras $h^{(2)}$ de $Q(h^{(2)}|h^{(1)}) = P(h^{(2)}|h^{(1)}, W^{(2)})$;
4. proceder recursivamente para as próximas camadas;

Assim que treinadas as camadas da DBN, a distribuição conjunta do modelo P e sua distribuição posterior aproximada são dadas por:

$$\begin{aligned} P(v,h^{(1)},...,h^{(L)}) &= P(v|h^{(1)})...P(h^{(L-2)}|h^{(L-1)})P(h^{(L-1)}|h^{(L)}) \\ Q(h^{(1)},...,h^{(L)}|v) &= Q(h^{(1)}|v)Q(h^{(2)}|h^{(1)})...Q(h^{(L)}|h^{(L-1)}) \end{aligned}$$

Para gerar amostras aproximadas da DBN pode-se utilizar um amostrados de Gibbs, como explicado na seção anterior. E para adquirir uma amostra da

distribuição aproximada posterior Q, pode apenas realizar uma passagem de baixo para cima ativando estocasticamente cada camada superior por turno. Para tanto pode-se escolher a seguinte aproximação $\tilde{Q}$, completamente fatorizada:

$$\tilde{Q}(h^{(1)}, ..., h^{(L)}|v) = \prod_{l=1}^{L} \tilde{Q}(h^{(l)}|v) \tag{20}$$

Em que

$$\begin{aligned} \tilde{Q}(h^{(1)}|v) &= \prod_j q(h_j^{(1)}|v), q(h_j^{(1)} = 1|v) = g\left(\sum_i W_{ij}^{(i)} v_i + a_j^{(1)}\right) \\ \tilde{Q}(h^{(l)}|v) &= \prod_j q(h_j^{(l)}|v), q(h_j^{(l)} = 1|v) = g\left(\sum_i W_{ij}^{(l)} q(h_i^{l-1} = 1|v) + a_j^{(l)}\right) \end{aligned} \tag{21}$$

$\tilde{Q}$ é obtido por simplesmente substituir as variáveis escondidas estocásticas por probabilidades reais e depois fazer uma passagem determinística de cima para baixo para computar $q(h_j^{(L)} = 1|v)$.

3.2 Redes geradoras concorrentes (GAN)

Nas redes geradoras concorrente, o modelo gerador é confrontado por um modelo discriminador, que deve aprender a diferenciar se uma amostra é da distribuição do modelo ou dos banco de dados. Essa competição leva aos dois lados a melhorarem seus métodos, até que as amostras do modelo gerados sejam indistinguíveis das do banco de dados.

Nesse caso não se faz necessário cadeias de Markov ou inferências aproximadas. Ambas as redes são definidas por perceptrons multicamada e o sistema todo pode ser treinado via *backpropagation*. [5]

Para se aprender a distribuição p_g do modelo gerador sobre os dados x, se define a priori variáveis de ruído $p_z(z)$, e depois se representa um mapeamento ao espaço dos dados como $G(z; \theta_g)$, em que G é uma função diferenciável representada por uma rede perceptron multicamada com parâmetros θ_g. E assim, se define uma segunda rede $D(x; \theta_d)$, que tem como saída um escalar. $D(x)$ representa a probabilidade de x ser proveniente dos dados do que de p_g. Dessa forma, treina-se D para que maximize a probabilidade de atribuir corretamente um rótulo tanto para os exemplos x quanto para as amostras de G. Simultaneamente treina-se G para se minimizar $log(1 - D(G(z)))$. Dessa forma, D e G jogam o *minmimax game*, com função de valor $V(G, D)$:

$$min_G max_D V(D, G) = E_{x \sim p_{data}(x)}\left[logD(x)\right] + E_{z \sim p_z(z)}\left[log(1 - D(G(z)))\right] \tag{22}$$

Otimizar D no laço interno, ao mesmo tempo que se otimiza G, é computacionalmente proibitivo. Dessa forma, se alterna entre k passos de otimização de D e um passo de otimização de G. Isso resulta em D sendo mantido perto de sua solução ótima enquanto G muda devagar o suficiente.

E ainda, na pratica a equação (22) não garante que G aprenda bem. No começo do treinamento, quando G ainda é pobre, D rejeita muitas amostras com alta confiança. Nesse caso, $log(1 - D(G(z)))$ satura. Então, ao invés de treinar G para minimizar $log(1 - D(G(z)))$, pode-se treiná-lo para maximizar $logD(G(z))$, o que garante gradientes maiores no começo do aprendizado.

Para um G fixo, o discriminador D ótimo é

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{23}$$

Além disso, o mínimo global de um critério de treino virtual $C(G)$ é atingido se e apenas se $p_g = p_{data}$. Nesse ponto, $C(G)$ adquire o valor $-log4$, sabendo que $C(G) = maxV(G, D)$.

E por fim, se G e D tem capacidade o bastante, e a cada etapa do algoritmo 1 [5], o discriminador consegue atingir seu máximo dado G e p_g é atualizado de acordo com o critério

$$E_{x\sim p_{data}(x)}\left[logD_G^*(x)\right] + E_{z\sim p_z(z)}\left[log(1 - D_G^*(G(z)))\right]$$

então p_g converge para p_data. As provas dos teoremas e das preposições citadas acima podem ser encontradas em [5].

Existem técnicas de melhoria das GANs, como *Feature Matching*, *minibatch discrimination*, média histórica, aprendizado semi-superviosionado, entre outras, que podem ser encontrados em [6].

Além disso, existe ainda o modelo gerador concorrente de imitação, que consiste no uso de funções de custos de *inverse reinforcement learning*, que são encontrados em [7].

3.3 TensorFlow

TensorFlow ™ é uma biblioteca de código aberto, desenvolvida originalmente por pesquisadores e engenheiros da Google como ferramenta para suas pesquisas em aprendizado de maquina. O enfoque dessa biblioteca consiste na realização de operações em computação numérica utilizando uma abstração em fluxo de dados.

A arquitetura permite, além de uma abstração adequada para as aplicações desejadas, uma melhor otimização do código que é dada principalmente pela possibilidade processamento utilizando unidades de processamento gráfico (GPUs).

Sua presença no mercado em aplicações de sistemas inteligentes é muito relevante, fornecendo uma certa confiança para sua utilização nesse projeto.

4 Metodologia

O processo de obtenção de marchas para o robô articulado divide-se nas seguintes partes: obtenção da base de dados; treinamento; avaliação dos resultados obtidos. É importante ressaltar que por se tratar de um método iterativo, em geral é necessário retornar e realizar alterações na arquitetura do sistema, com base nos resultados obtidos, de forma a aprimorar os resultados.

4.1 Base de Dados

Diferente das aplicações mais comuns utilizando redes geradoras concorrentes, que são baseadas em análise de imagens e textos, neste caso não se tem uma base de dados de fácil acesso para utilizar como parâmetros nas redes. Entretanto, a aplicação escolhida permite a analogia com sistemas biológicos, fornecendo a possibilidade de uma abordagem bioinspirada.

Define-se então a estratégia de criação de marchas inspiradas em animais. Elas servirão como parâmetros de treinamento para a rede. Destaca-se que a ideia é abordar diversos tipos de marchas, promovendo a maior variabilidade possível, visando uma boa generalização nos modelos gerados após o treinamento.

4.2 Treinamento

A aplicação de métodos de sistemas em sistemas físicos apresenta uma complicação, no processo de treinamento as redes não geram resultados confiáveis, podendo inferir em uma grande instabilidade ao sistema.

O erro é um fator necessário e está diretamente associado ao processo de treinamento dos algoritmos geradores. Torna-se inviável realizar essa etapa na planta real, haja vista que uma perda de equilíbrio do robô pode gerar danos irreparáveis na planta.

Portanto decide-se realizar essa etapa em uma plataforma de simulação computacional, permitindo um treinamento mais rápido e que não prejudique a planta real. É importante ressaltar que a simulação consiste em uma simplificação do ambiente real, portanto existirão características que não serão avaliadas nesse processo.

Uma abordagem interessante a ser estudada consiste na divisão do treinamento em duas etapas: o treinamento geral e o de ajuste fino. O treinamento geral se dá apenas no ambiente de simulação e é necessário para a aquisição de informações

gerais do sistema. Essa etapa é finalizada quando nota-se que o sistema possui uma estabilidade satisfatória. Logo depois disso, pode-se realizar o treinamento de ajuste fino, essa etapa é realiza na planta real. O intuito é ajustar as redes geradoras para adquirirem as informações que não foram obtidas na simulação.

4.3 Avaliação dos resultados obtidos

Após possuir uma rede geradora que fornece marchas com certa confiabilidade é possível obter diversas marchas e avaliar o desempenho delas na planta real. Os dois principais critérios de avaliação da qualidade da marcha gerada é a velocidade e a estabilidade do robô. A definição de uma função que avalie a marcha é de suma importância para caracterizar de forma quantitativa o desempenho do sistema com essa movimentação. Esse ponto exige um maior aprofundamento e será feito posteriormente, bem como a definição detalhada do processo de treinamento e implementação do algoritmo.

5 Referências

[1] *PORPHIRIO, C. F.;SANTANA, P. H. M. Geração de marchas para a plataforma quadrúpede utilizando algoritmo genético. jun 2017.*

[2] *PAIVA, R. C. Utilizando osciladores neurais para gerar marcha de um robô quadrúpede e robô humanoide. jul 2012.*

[3] *BELTER, D.; SKRZYPCZYŃSKI, P. A Biologically inspired approach to feasible gait learning for a hexapod robot. 2010.*

[4] *SALAKHUTDINOV, R. Learning Deep Generative Models. 2015.*

[5] *GOODFELLOW, I. J; POUGET-ABADIE, J.; MIRZA, M; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A.; BENGIO, Y. Generative Adversarial Nets. 10 jun 2014.*

[6] *SALIMANS, T.; GOODFELLOW, I.; ZAREMBA, W.; CHEUNG, V.; RADFORD, A.; CHEN, X. Improved Techninques for training GANs. 10 jun 2016.*

[7] *HO, J.; ERMON, S. Generative Adversarial Imitation Learning. 10 jun 2016.*

[8] *RESTRICTED BOLTZMANN MACHINES (RMB). DeepLearning. Disponível em <http://deeplearning.net/tutorial/rbm.html>. Acesso em 07 dez. 2017*