TRABALHO DE CONCLUSÃO DE CURSO

Melhoramento de marcha para robô quadrúpede utilizando estratégia de aprendizagem por reforço

Júlia Cabral Diniz Braz

Brasília, dezembro de 2018

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE CONCLUSÃO DE CURSO

Melhoramento de marcha para robô quadrúpede utilizando estratégia de aprendizagem por reforço

Júlia Cabral Diniz Braz

Trabalho de Conclusão de Curso submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro Eletricista

Banca Examinadora

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

Prof. Geovany Araújo Borges, ENE/UnB _____
Examinador interno

Prof. Adolfo Bauchspiess, ENE/UnB _____
Examinador interno

FICHA CATALOGRÁFICA

BRAZ, JÚLIA CABRAL DINIZ
Melhoramento de marcha para robô quadrúpede utilizando estratégia de aprendizagem por reforço [Distrito Federal] 2018.
xvi, 69 p., 210 x 297 mm (ENE/FT/UnB, Engenheiro, Engenharia Elétrica, 2018).
Trabalho de Conclusão de Curso - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Robô Quadrúpede	2. Aprendizagem por Reforço
3. Melhoramento de Marcha	4. Simulação de Marcha
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

BRAZ, J. C. D. (2018). *Melhoramento de marcha para robô quadrúpede utilizando estratégia de aprendizagem por reforço*. Trabalho de Conclusão de Curso, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 69 p.

CESSÃO DE DIREITOS

Júlia Cabral Diniz Braz
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedicatória

Dedico esse trabalho aos meus pais, Aline e Rubens, aos meus irmãos, Clara e Rubens Filho, aos meus avós, Jarbas e Lúcia, e a um grande amigo, Miguel.

Júlia Cabral Diniz Braz

Agradecimentos

Agradeço aos meus pais, Aline e Rubens, que sempre me apoiaram em tudo, mesmo que as vezes de maneiras diferentes das quais eu esperava. Eu não teria chegado muito longe se não fossem por vocês me mandado ir estudar e na maiorias das vezes brigando comigo após eu não obedecer. Agradeço a minha irmã, Clara, que está sempre comigo e que, além de irmã, sempre será minha melhor amiga. Eu costumava odiar quando as pessoas diziam que nós eramos uma pessoa só, mas agora só lembro disso com carinho. A gente é realmente muito parecida, tanto fisicamente quanto no temperamento. Agradeço ao meu irmão que, mesmo tão diferente de mim, me suporta e tenta lidar comigo da melhor maneira. Agradeço aos meus avós, Jarbas e Lúcia, que sempre disseram que sonhavam em ir na minha formatura. Agradeço a Xixinha, minha eterna babá. Agradeço aos amigos que fiz ao longo da minha graduação e com os quais me diverti bastante, em festinhas ou noite de estudos, em especial, Miguel Pachá, meu melhor amigo desde o primeiro dia de aula do primeiro semestre, Luisa Moreira, Isabella Chrisostomo, Anna Caroline Lopes, Monique Nogueira. Agradeço aos meus colegas de curso com os quais dividi muitas risadas no CA. Agradeço as pessoas que conheci estagiando no Itamaraty, Débora, Marcelo, Caio, Anny, Elynara e Laryssa, guardo muitos lembranças boas de lá. Por fim, agradeço ao meu orientador, Alexandre Romariz, por me dar a oportunidade de participar desse projeto.

Júlia Cabral Diniz Braz

RESUMO

Esse trabalho tem como objetivo o melhoramento de um modelo de marcha proposto em um trabalho anterior, utilizando-se estratégias de aprendizado por reforço. Para isso foram aplicados dois métodos diferentes, o primeiro foi a utilização do algoritmo de diferença temporal e o segundo, um algoritmo de gradiente de política. Para a avaliação dos algoritmos eles foram, primeiramente, aplicados em ambiente simulado. No primeiro método, o valor dos estados são aproximados conforme os estados são visitados e uma implementação com parâmetros contínuos se mostrou inviável. Deste modo, apenas três parâmetros foram escolhidos para serem aprendidos e esses foram discretizados em passos de 0,4, mas mesmo com estas restrições impostas, foi visto que para se obter uma estimativa da função valor eram necessárias muitas horas de simulação. No segundo método o tempo de simulação diminuiu consideravelmente, e com poucas iterações, foi possível a obtenção de melhores resultados. A marcha usada para inciar o algoritmo conseguia percorrer uma distância de 61cm e a marcha final encontrada nesse trabalho chegou a andar 80cm, um aumento de 30%.

Palavras Chave: Quadrúpede, Marchas, Aprendizado por reforço, Gradiente de política, Diferença temporal.

ABSTRACT

This work aims to improve a gait model proposed in an earlier work, using reinforcement learning strategies. For this, two different methods were applied, the first being the use of the time difference algorithm and the second, a policy gradient algorithm. For the evaluation of the algorithms they were applied in a simulated environment first. In the first method, the value of the states is approximated as the states are visited and an implementation with continuous parameters proved to be impractical. Thus, only three parameters were chosen to be learned and these were discretized in steps of 0.4, but even with these restrictions, it was seen that in order to obtain an estimate of the value function, many hours of simulation were necessary. In the second method the simulation time decreased considerably and with few iterations it was possible to obtain better results. The march used to start the algorithm could walk a distance of 61cm and the final gait found in this work reached 80cm, an increase of 30%.

Keywords: Quadruped, Gait, Reinforcement Learning, Policy Gradients, Temporal difference.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

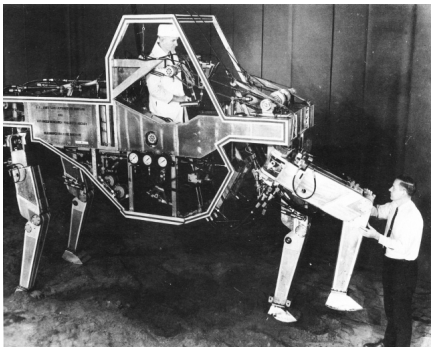
LISTA DE ALGORITMOS

1 INTRODUÇÃO

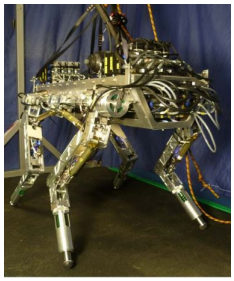
1.1 CONTEXTUALIZAÇÃO

1.1.1 Robôs quadrúpedes

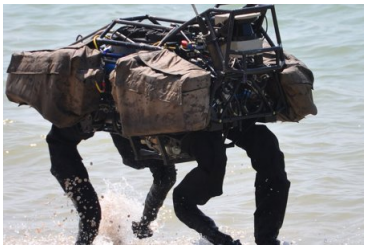
(a) *The Mechanical Horse [7]*



(b) Quadrúpede da *General Electric* [7]



(c) HyQ [8]



(d) *Big Dog* da *Boston Dynamics*

Figura 1.1: Robôs quadrúpedes ao longo do tempo.

O primeiro robô do tipo quadrúpede foi projetado por L. A. Rigg no século XIX, o *The Mechanical Horse*. Esse robô foi pensado de modo que seus movimentos seriam feitos por um sistema de engrenagens, onde uma pessoa poderia pedalar. Ele foi patenteado em 14 de fevereiro de 1893 mas não há evidências de que foi realmente construído [7].

Em 1968, surge o quadrúpede da *General Electric*, desenvolvido por R. Mosher, esse foi um dos primeiros veículos a ser capaz de executar diferentes marchas. Esse robô era humanamente controlado através de um sistema de alavancas. Por ter um sistema de movimentação muito complexo e o controle do robô ser composto por 12 graus de liberdade, sua movimentação era feito apenas por um profissional especializado. Deste modo, apesar do quadrúpede da *General Electric* ter capacidade de superar obstáculos e sua boa mobilidade em terrenos difíceis, ficou claro que era necessário ter um sistema de controle por computador. Na mesma década, surge o primeiro quadrúpede totalmente controlado por computador, o *"Phoney Poney"*, criado na Universidade do Sul da Califórnia por Andrew Frank, Bob McGee e Rajko Tomovic.

Com o desenvolvimento da robótica nos últimos anos, atualmente, temos robôs quadrúpedes multiarticulados, autônomos e com capacidade de mudança de marcha para transpor obstáculos. Robôs quadrúpedes são estudados devido a sua diversas aplicações. Uma aplicação possível para robôs desse tipo é no resgate de vítimas em áreas de risco. A estrutura do robô de quatro patas permite alcançar locais que não seriam possíveis com robôs com rodas. O robô *HyQ (Hydraulic Quadruped)*, desenvolvido no Instituto Italiano de Tecnologia, foi criado com o objetivo de acessar ambientes de risco que antes só podiam ser acessado por humanos [8].

Outra aplicação interessante para uma plataforma quadrúpede é a de transporte de cargas. Sua estrutura é ideal para essa aplicação devido a distribuição do peso entre as patas proporcionando um alivio a carga existente nos motores. A implementação mais famosa de um robô quadrúpede focado no transporte de cargas é o *BigDog*, da *Boston Dynamics*.

1.1.2 Histórico da Plataforma

Desde 2006, são realizados trabalhos para a implementação e aperfeiçoamento de um robô quadrúpede no Laboratório de Automação e Robótica (LARA) da Universidade de Brasília (UnB). Os dois principais objetos de estudo, idealizados com a construção dessa plataforma, eram robótica terrestre e robótica comportamental. Ao longo dos anos vários trabalhos utilizaram esse dispositivo como foco. A Tabela 1.1 lista os trabalhos desenvolvidos.

Autores	**Ano**	**Título**
COTTA, G. H.; RAULINO NETO, L.	2006	Realização de uma plataforma para estudo de robótica comportamental baseada em quadrúpedes
CALMON, A. du P.; PINHEIRO, N. C.; FERREIRA, R. U.	2006	Desenvolvimento de um robô-cachorro comportamental: percepção e modelagem comportamental
SOUTO, R. F.	2007	Modelagem cinemática de um robô quadrúpede e geração de seus movimentos usando filtragem estocástica.
BATISTA, G. F.; CARDOSO, I. F.	2007	Adequação de um sistema de locomoção de um robô quadrúpede para avaliação de algoritmos de aprendizagem
DE NOVAIS, N. A.; TOSCANO, R. A.	2007	Estudo de locomoção de uma plataforma quadrúpede utilizando sensoriamento inercial e geração de padrões de movimento.
RAMOS, E.G.	2008	Desenvolvimento da plataforma quadrúpede geração de software e eletrônica.
PAIVA, R. C.	2012	Osciladores neurais para comando de marcha de um robô quadrúpede e robô humanoide.
SANTOS, J. H. de S.	2016	Plataforma quadrúpede Uma nova estrutura para robô quadrúpede do LARA.
PORPHIRIO, C. de F.; SANTANA, P. H. M.	2017	Geração de marchas para a plataforma quadrúpede utilizando algoritmo genético.
FLORIANO, B. R de O.	2017	Desenvolvimento e interação de um sistema de controle de equilíbrio para um robô quadrúpede.
DOS SANTOS, P. M .F; DUARTE, V. A.	2018	Aprendizado de marchas factíveis para um robô quadrúpede utilizando algoritmo genético multiobjetivo.

Tabela 1.1: Projetos desenvolvidos utilizando a plataforma quadrúpede

No ano de 2006, Cotta e Raulino Neto [9] construíram a plataforma quadrúpede para o estudo de robótica comportamental, onde estudaram uma abordagem da cinemática direta e inversa da plataforma. No mesmo ano, Calmon, Pinheiro e Ferreira [10] desenvolveram um modelo, utilizando aprendizado por reforço, para possibilitar uma interação do robô com agentes externos. Este trabalho resultou na melhoria do robô, como a implementação de um sistema de sensoriamento.

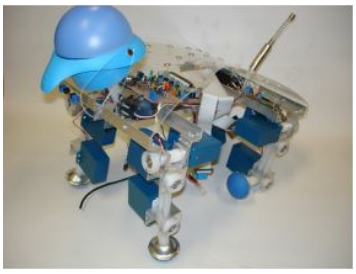
Em 2007, Souto [3] deu continuidade aos estudos do modelo comportamental, utilizando filtragem estocástica. Ainda em 2007, Batista e Cardoso [11] alteraram os servos e introduziram acelerômetros para avaliar os algoritmos de aprendizagem. Em seguida, Novais e Toscano [12] adicionaram um IMU (*Inertial Meansuriment Unit*) e um conjunto de sensores para detecção de contato nas patas.

Em 2008, Ramos [13] reestruturou as placas de acionamento dos serviços do robô e transcreveu os códigos, anteriormente implementados em *MatLab*, para linguagem C.

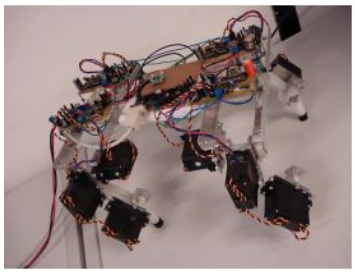
Paiva [14], em 2012, estudou a geração de marcha bioinspirada, utilizando osciladores neurais, na plataforma quadrúpede e em plataformas bípedes. Mas devido ao acréscimo de peso do robô o torque dos servos era inferior ao demandado pelo robô e as marchas não puderam ser testadas no solo. Devido a isso, Santos [15], em 2016, promoveu uma reestruturação completa da plataforma.

Porphiro e Santana [16], em 2017, adicionaram um módulo de *Arduíno* para realizar medições do acelerômetro e dos sensores de força, além disso, desenvolveram algoritmos que permitiam o controle do modelo robótico, no ambiente de simulaçõ do *software V-REP*, pelo *MatLab*. Ao mesmo tempo, Floriano [17] desenvolveu um trabalho focado no controle de equilíbrio da plataforma, permitindo que o dispositivo continue a execução da marcha mesmo após um distúrbio.

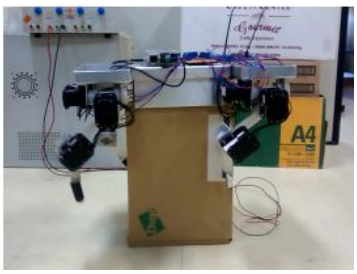
Em 2018, Dos Santos e Duarte [4], promoveram a busca de marchas factíveis ao robô, utilizando algoritmo genético multiobjetivo, ao final do projeto a plataforma conseguiu realizar uma marcha adequada ao seu funcionamento.



(a) Primeira versão desenvolvida [9]



(b) Quadrúpede após o trabalho de Batista e Cardoso [11]



(c) Plataforma reestruturada Santos [15]

Figura 1.2: Histórico da plataforma quadrúpede do LARA.

1.1.3 Aprendizado de Máquina

Arthur Samuel [18] definiu aprendizado de máquina como o "campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados". Ou seja, pode-se utilizar algoritmos para coletar dados e aprender com os dados, levando em consideração o histórico para fazer predições sobre várias situações. Algoritmos desse tipo constroem um modelo se guiado apenas pelos dados e, não mais, seguindo instruções programadas.

O interesse no aprendizado de máquina se deve a alguns fatores como: os crescentes volume e variedade de dados disponíveis, o processamento computacional mais barato e poderoso, o armazenamento de dados acessível etc. Isso significa que, com o aprendizado de máquina é possível produzir modelos que são capazes de analisar dados, de forma mais rápida e automaticamente, e entregar resultado precisos.

Não é difícil encontrar históricos da utilização do Aprendizado de Máquina na vida real, algumas de suas utilizações consiste em: carros autônomos do *Google*, as ofertas e recomendações online da *Netflix, Facebook e Amazon*.

Algumas empresas surgiram com o objetivo de fazer pesquisas na área de aprendizado de máquina. *DeepMind Technologies Limited*, da *Google*, é uma empresa com o foco em pesquisas e desenvolvimento de máquinas de inteligência artificial. A empresa criou uma rede neural que aprende a jogar jogos eletrônicos de uma forma semelhante a dos humanos. Seu programa, AlphaGo, chegou a ganhar de um jogador profissional de Go pela primeira vez. Um programa mais genérico, AlphaZero, conseguiu superar os programas mais poderosos que jogam Go, depois de algumas horas de jogo contra si mesmo usando aprendizado por reforço.

Essa companhia chegou, também, a desenvolver um inteligência artificial capaz de aprender como andar e pular sem nenhum tipo de experiência anterior. Essa tecnologia usa um sistema de aprendizado por reforço para melhorar sua movimentação a cada passo. Nesse software foi apenas programado a capacidade de interação com o ambiente, ou seja, o agente entender seus arredores e o incentivo de seguir sempre em frente.

Em vídeo [1] é visto como o *software* evoluiu a cada novo obstáculo encontrado, aprendendo novas maneiras de se mover e otimizando seu deslocamento a cada passo. Assim, ele se mostra capaz até mesmo de lidar com cenários completamente novos sem grandes dificuldades.

1.2 DEFINIÇÃO DO PROBLEMA

Em 2018, Dos Santos e Duarte [4] criaram um modelo de marcha adequada ao funcionamento da plataforma quadrúpede, essa marcha é composta por 4 poses, que são descritas por um conjunto de parâmetros. Para a obtenção dos parâmetros ideias foram necessários mais de 500 testes de marcha na plataforma.

[1] https://www.youtube.com/watch?v=dng6SHJkGk0

Neste projeto busca-se uma forma mais eficiente de aprendizado, utilizando-se do mesmo modelo de marcha.

1.3 OBJETIVOS DO PROJETO

O objetivo do projeto é, utilizando do modelo de marcha criado por Dos Santos e Duarte [4], encontrar marchas mais fluidas através da implementação de algoritmos de aprendizado por reforço, com auxilio de um ambiente de simulação.

1.4 APRESENTAÇÃO DO MANUSCRITO

O capítulo 2 apresenta uma revisão bibliográfica dos conceitos teóricos utilizados, nele é apresentado um resumo sobre aprendizado por reforço e seus tipos de algoritmos, a estrutura da plataforma, a definição da marcha, e o *software* de simulação. O capítulo 3 traz a metodologia utilizada no desenvolvimento do projeto. No capítulo 4 apresentam-se os resultados obtidos, como apresentação dos resultados de cada algoritmo utilizado e comparação das marchas obtidas. E por fim, o capítulo 5 é onde se encontra as conclusões do projeto.

2 FUNDAMENTOS TEÓRICOS

Este capítulo apresenta um resumo sobre os fundamentos utilizados como referência para este trabalho. No qual expõe-se os conceitos de aprendizagem por reforço e seus principais algoritmos, breve resumo da estrutura da plataforma utilizada e seus dispositivos embarcados (sensores, Arduíno e Raspberry PI), como a marcha foi definida por Dos Santos e Duarte [4] e, ainda, é feito um breve comentário sobre o software de simulação V-REP.

2.1 APRENDIZAGEM POR REFORÇO

Aprendizado por reforço (RL, do inglês *Reinforcement Learning*) é diferente de aprendizado supervisionado, o tipo de aprendizado de máquina mais estudado atualmente [1]. No aprendizado supervisionado o agente aprende a partir de um conjunto de dados pré-definidos por um supervisor externo. Cada exemplo é uma descrição de uma situação com uma especificação – rótulo – da ação a ser tomada. O objetivo deste tipo de aprendizado é o agente extrapolar, ou generalizar, sua resposta para que ele tome a ação correta para situações que não estão em seu banco de dados. Esse tipo de aprendizado não é adequado para um problema iterativo. Em problemas iterativos é difícil manter um base de dados que contenha todos os exemplos das ações a serem em todas as situações em que o agente tem que enfrentar, dessa forma o agente deve ser capaz de aprender a partir de sua própria experiência.

RL também é diferente de aprendizado não supervisionado, em que tipicamente um agente pode automaticamente encontrar encontrar padrões e relações em um conjunto de dados não rotulados. Pode parecer que RL seja um tipo de aprendizado não supervisionado pois ele não necessita de exemplos de comportamentos corretos, mas o RL tenta maximizar um sinal de recompensa em vez de encontrar padrões. Aprendizado por reforço é sinônimo de aprendizado por interação, uma vez que o agente aprende diretamente da interação com o ambiente onde está inserido [19]. Neste processo, não é dito ao agente qual ação deve ser tomada, ao invés disso ele deve descobrir qual ação irá retornar uma melhor recompensa.

Um dos problemas encontrados no RL, e não em outros tipos de aprendizado de máquina, é o paradigma entre *exploration* e *exploitation*, ou seja, o agente pode escolher tomar ações que já foram exploradas no passado que foram efetivas em obter recompensas (*explotation*), ou preferir "testar"novas combinações de ações ótimas em uma sequência de estados não realizadas anteriormente procurando uma recompensa maior mas não imediata (*exploration*). O agente tem que se aproveitar do que já foi experimentado para conseguir recompensas, mas também tem que explorar para que seja selecionada realmente a melhor ação no futuro. O dilema entre usufruir ou explorar é que nenhum dos dois pode ser usado exclusivamente sem que o agente falhe em atingir

seu objetivo. É necessário que uma variedade de ações sejam tomadas e que as que pareceram atingir melhores resultados sejam escolhidas progressivamente.

Grande parte das pesquisas envolvendo aprendizado por reforço é baseada no processo de decisão de Markov. Um agente – tomador de decisões – encontra o problema, ou melhor, a oportunidade, de influenciar o comportamento de um sistema probabilístico com o tempo. Ele faz isso ao tomar decisões ou ações. Seu objetivo é encontrar uma sequencia de ações que faz o sistema operar otimizadamente à respeito de critérios de atuação previamente estabelecidos e levando em conta que o estado do sistema futuro à tomada de decisão depende do estado atual. Consequentemente, as decisões devem antecipar as oportunidades e custos (ou recompensas) associadas também ao estado futuro do sistema [20].

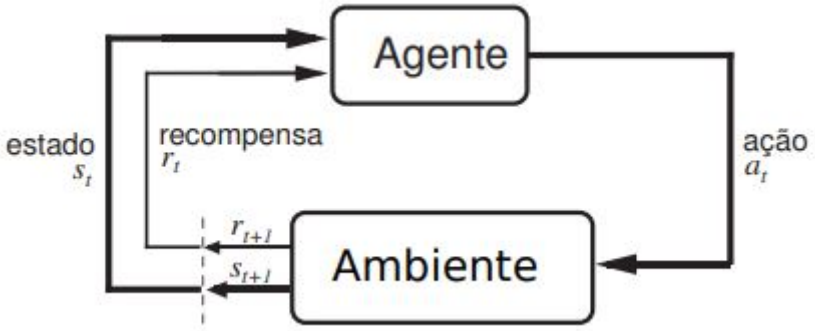


Figura 2.1: Interação agente-ambiente em um MDP. (Adaptado de [1])

2.1.1 Processos de decisão de Markov

Um processo de decisão de Markov (MDP, do inglês *Markov Decision Process*) é uma tupla $< S, A, T, R >$, onde S representa o conjunto finito de estados do ambiente, A é o conjunto finito de ações do agente, T é a função de transição, ou seja, ela denota a probabilidade de uma ação levar o agente de um estado atual para um estado estado futuro e R é uma função que retorna a recompensa por se tomar uma ação a partir de um estado [21]. A interação agente-ambiente do MDP pode ser melhor visualizada na Figura 2.1.

2.1.1.1 Estado e ação

Em um intervalo de tempo discreto t, o agente ocupa um estado que é formado pelas percepções do agente. S representa o conjunto de estados em que o processo pode estar. Se o tomador de decisões observa o sistema em $s \in S$, ele pode escolher uma ação do conjunto de ações que podem ser executadas naquele estado s, A_s [1].

2.1.1.2 Recompensa e probabilidade de transição

De acordo com a propriedade de Markov, a dinâmica do futuro, transições e recompensas são completamente dependentes do estado atual, isto é, uma ação $a \in A$ em um estado $s \in S$ resulta em um estado s', baseado na função de transição $T : S \times A \times S \to [0, 1]$. A probabilidade de parar em um estado s' após a realização de uma ação a é denotada como $T(s, a, s')$. A função de recompensa $R : S \to \Re$ retorna a recompensa $R(s, a)$ após o agente tomar uma ação a partir de um estado s. A função de transição T e a função de recompensa R, juntas, são definidas como o modelo que o agente tem do ambiente [22].

2.1.2 Sub-elementos principais no aprendizado por reforço

No RL, além do agente e do ambiente podemos identificar outros quatro sub-elementos principais no aprendizado por reforço: uma *política*, um *sinal de recompensa*, uma *função valor*, e, opcionalmente, *modelo do ambiente* [1]. Estes sub-elementos são definidos a seguir.

2.1.2.1 Política

Uma politica de aprendizado π define o comportamento do aprendiz em um determinado tempo. Em outras palavras, proporciona ao agente uma indicação para a seleção de uma ação baseado em qualquer estado, é a organização dos estados, $s \in S$, e das ações, $a \in A(s)$, para a probabilidade $\pi(s, a)$ de se tomar uma ação a enquanto se estiver no estado s. Corresponde ao que é definido na psicologia como conjunto de regras "estímulo-resposta" ou associações [1]. Em alguns caso a politica de aprendizado pode ser apenas uma simples função e em outros pode envolver problemas complicados de computação. Para se otimizar a interação do agente com o ambiente, as consequências futuras de uma política de aprendizado devem ser conhecidas [23].

2.1.2.2 Sinal de recompensa

Um sinal de recompensa é o que define o objetivo de um problema de aprendizado por reforço. A cada intervalo de tempo t, o sinal de recompensa é simplesmente um número, $R_t \in \Re$ [1]. Em geral, o agente tem como meta maximizar o retorno esperado, onde o retorno G_t é definido como uma função da sequência de recompensas. No seu caso mais simples o retorno é apenas a soma das recompensas, mas quando a interação agente-ambiente não se divide em intervalos (tarefas continuas) essa não será a melhor estratégia pois o intervalo de tempo final seria $T = \infty$.

É, então, introduzido o conceito de desconto γ que pode assumir valores entre $[0, 1]$. Deste modo, o agente tenta selecionar ações em que as somas das recompensas descontadas que recebe ao longo do tempo é maximizada. O retorno pode é definido em [1] como a soma descontada dos

recompensas subsequentes:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{2.1}$$

onde T é o intervalo de tempo final. Há a possibilidade de $T = \infty$ e de $\gamma = 1$ (apenas a soma das recompensas) mas elas não podem ser satisfeitas ao mesmo tempo.

O sinal de recompensa define os eventos que estão ou não corretos para o agente [23]. Em uma abordagem biológica, podemos classificar as recompensas como eventos de prazer ou dor [1]. Um sinal de recompensa é a primeira base para se alterar uma política; se uma ação selecionada pela política retorna uma recompensa baixa, então é possível que uma alteração na política retorne melhores resultados.

2.1.2.3 Função valor

Enquanto o sinal de recompensa indica se o agente executou ou não uma boa ação logo após esta ter sido executada, a função valor de um estado, $V(s)$, descreve o que é bom a longo prazo. Em outras palavras, indica a o quão é bom para o agente estar em determinado estado em longo prazo considerando os estados que podem vir a seguir e as recompensas de cada um destes estados. Um cenário possível é quem um estado forneça uma baixa recompensa mas tenha um alto valor, isso pode acontecer pois os estados que o seguem levam a altas recompensas. A situação inversa também pode ocorrer.

Diferentemente do como o sinal de recompensa pode ser comparado com prazer e dor a função valor corresponde a quanto se está satisfeito em estar em dado estado. O único proposito de se calcular a função valor é alcançar mais recompensas [1]. Ações são feitas de modo ao agente se encontrar em um estado com o maior valor, e não que obtenha uma maior recompensa, pois estas ações proporcionam um maior retorno à longo do tempo. A maioria dos algoritmos de aprendizado por reforço são baseados na estimativa da função valor.

Ao contrário de recompensas que são dadas diretamente pelo ambiente o valor de um estado deve ser estimado a partir de uma sequência observações que o agente faz ao longo de sua vida útil. O valor de um estado s sujeito a uma política π, $V_\pi(s)$, é o valor esperado quando se começa em um estado s e desde então segue-se uma política π. V_π segundo [23] pode ser definido como:

$$V_\pi = \sum_a \pi(s,a) \sum_{s'} T^a_{ss'}(R^a_s + \gamma V_\pi(s')). \tag{2.2}$$

Onde $\gamma \in [0,1]$ é o fator de desconto que é utilizado para ponderar futuras recompensas. Se $\gamma = 1$ corresponde a recompensa imediata, caso contrário, se $\gamma = 0$, então corresponde a soma esperada das futuras recompensas. A Equação 2.2 é conhecida como a Equação de Bellman para V_π, ela expressa a relação entre o valor do estado corrente com os seus sucessores. A equação de Bellman calcula a média sobre todas as possibilidades, pesando cada estado de acordo com a sua

probabilidade de ser atingido.

2.1.2.4 Modelo do ambiente

O quarto elemento de um sistema de aprendizado por reforço é um modelo do ambiente. Esse modelo "imita" o comportamento do ambiente, ou melhor, ele possibilita inferir sobre como o ambiente vai se comportar. O modelo pode até mesmo supor qual poderá ser o próximo estado e recompensa se dada ação a for selecionada em um estado s. Eles são usados para planejar, ou seja, um meio de decidir um sequência de ações que consideram possíveis situações antes mesmo que elas sejam experimentadas. Modelos para solucionar o problema de aprendizado por reforço que usam modelos e planejamento são chamados de *model-based* enquanto os outros métodos são considerados *model-free* e usam métodos de tentativa e erro ou em vez de fazer um planejamento.

2.1.3 Dilema *exploration* vs *exploitation*

Um dos métodos mais simples para balancear entre (*explotation*) e (*exploration*) é se utilizando do valor de uma ação, que serve para a seleção de qual ação deve ser selecionada.

Se o valor verdadeiro de uma ação é a recompensa após ela ser tomada o jeito mais fácil de estimar o valor dessa ação é tomando a média de todas as recompensas recebidas:

$$Q_t(a) = \frac{\text{soma das recompensas quando } a \text{ foi selecionada antes de } t}{\text{número de vezes que } a \text{ foi selecionada antes de } t} \tag{2.3}$$

O valor de $Q_t(a)$ é definido inicialmente como 0 e conforme o denominador aumenta o valor de $Q_t(a)$ converge para $q_*(a)$.

Para seleção de uma ação é levado em conta aquela que tem o maior valor de ação, ou seja, é tomada uma ação gulosa e dizemos que o agente está escolhendo ações que ele sabe que levam a boas recompensas (*explotation*). Ao selecionar ações não gulosas – que não tem o maior valor de ação – dizemos que o agente está explorando, pois dá a oportunidade de melhorar a estimativa destas ações (*exploration*).

Uma ação gulosa é definida como:

$$A_t \doteq \arg\max_a Q_t \tag{2.4}$$

onde $\arg\max_a$ denota qual ação deve ser tomada para que a expressão seja maximizada. Seleção de ações gulosas leva em conta o conhecimento para selecionar ações e maximizar a recompensa imediata; não perdem tempo atrás de ações com valores inferiores que podem vir a ser melhores.

Uma alternativa simples para o dilema de (*explotation*) e (*exploration*) é selecionar uma ação

gulosa quase sempre, mas de vez em quando, com uma pequena probabilidade ϵ selecionar uma ação de modo aleatório entre todas as outras opções, ou seja, selecionar uma ação não gulosa (*exploration*). Esse método é chamado de ϵ-greedy.

2.2 ALGORITMOS DE APRENDIZADO POR REFORÇO

2.2.1 Programação Dinâmica

O termo programação dinâmica (DP, do inglês *Dynamic Programming*) se refere a uma coleção de algoritmos que pode ser usado para determinar uma política ótima se dado um modelo perfeito do ambiente com um MDP [1]. Algoritmos clássicos de PD são bastante limitados em RL, devido à sua suposição de modelo de ambiente perfeito e custo computacional [22].

É dito que uma política π é melhor que outra π' se o valor esperado de π é maior ou igual para todos os estados, ou seja, $\pi \geq \pi' \Leftrightarrow V_\pi(s) \geq V_{\pi'}(s)$ para todo $s \in S$. Se existe uma política π que é melhor ou igual a todas as outras políticas, então π é ótima.

Para se encontrar uma política ótima, primeiro tem-se que achar uma função de avaliação ótima. A função de avaliação ótima é única e foi definida em [23] como:

$$V^* = \max_a \sum_{s'} T^a_{ss'}(R^a_s + \gamma V_\pi(s')), \forall s \in S, \tag{2.5}$$

o que define o valor da função valor ótima de um estado é a recompensa instantânea somado ao valor descontado do próximo estado, considerando que foi selecionada a melhor ação.

Dado a função valor ótima de um estado, pode-se especificar a política ótima da seguinte maneira:

$$\pi^*(s) = \arg\max_a \left(\sum_{s'} T^a_{ss'} \left(R^a_{ss'} + \gamma V^*(s') \right) \right). \tag{2.6}$$

A ideia chave de DP, e de aprendizado por reforço em geral, é o uso da função valor para organizar uma estrutura e buscar por boas políticas. Estes algoritmos são obtidos transformando as equações de Bellman, equação (2.2), em atribuições, isto é, em regras de atualização para melhorar as aproximações das funções de valor desejadas. Isso pode ser feito conforme [1]:

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} T^a_{ss'} \left(R^a_{ss'} + \gamma V_k(s') \right) \tag{2.7}$$

Primeiramente é considerado como calcular a função valor V^* para uma política arbitrária π. Isso é chamado de avaliação de política, ou problema de predição. Em cada iteração se atualiza o valor de cada estado uma vez para se alcançar novas aproximações da função valor V_{k+1}, com a equação (2.7).

Um conhecido algoritmo de programação dinâmica é o iteração de políticas. Este algoritmo é inicializado com uma política aleatória e, para cada estado, é verificado se uma melhor ação pode ser realizada. Desta forma, alterando apenas um par estado-ação por vez para um com recompensa maior, o algoritmo pode garantir melhorar a política geral do agente.

2.2.2 Métodos de Monte Carlo

Neste tipo de algoritmo não é necessário completo conhecimento do ambiente, apenas experiência através de amostras de estados, ações e recompensas de iterações reais ou simuladas com o ambiente. Aprender diretamente com experiências reais é um diferencial pois não é necessário nenhum tipo de conhecimento anterior da dinâmica do ambiente, e mesmo assim é possível obter um comportamento ótimo.

Métodos de Monte Carlo (MC) são maneiras de resolver o problema de RL baseando-se na média de amostras de retorno, para isso métodos de MC são utilizados em tarefas episódicas – definidas como uma sequência (ou cadeia) de estados visitados por um agente, desde o estado inicial até o final. É ao final destes episódios em que a função valor é estimada e as políticas avaliadas e reformadas.

Quando o modelo do sistema é desconhecido, é necessário aprender a função valor da ação, $Q_\pi(s, a)$, ou seja, o retorno esperado a partir do estado s, tomando-se a ação a, e depois seguindo a política π, não apenas o valor $V_\pi(s)$ do estado. $Q_\pi(s, a)$ é definida em [23] como:

$$Q_\pi(s, a) = \sum_{s'} T^a_{ss'}(R^a_s + \gamma V_\pi(s')) \tag{2.8}$$

Deste modo a equação (2.5) pode ser reduzida a: $V^* = \max_a Q_\pi(s, a)$. A partir dessas duas equações chegamos ao valor ótimo da função valor da ação:

$$Q^*(s, a) = \max_a Q_\pi(s, a) = \sum_{s'} T^a_{ss'}(R^a_s + \gamma \max_a Q^*(s', a')). \tag{2.9}$$

O procedimento para obtenção dessa função consiste em gerar episódios a partir de uma política inicial π_0. A partir do conhecimento da sequência de recompensas é possível estimar o retorno proveniente de cada par estado-ação, $Q_{\pi_0}(s, a)$, a medida em que eles são visitados. Após um número suficiente de episódios pode-se obter a estimativa aproximada da função valor da política. Quanto mais amostras, melhor a convergência para o valor esperado [2].

A melhoria da política neste tipo de algoritmo é simplesmente a escolha de uma ação gulosa – ação de maior valor – para cada estado. Neste momento que a função valor da ação é importante, pois quando não se tem o modelo do ambiente não haveria como saber o efeito de uma ação sobre o ambiente[24]. Sabendo deste valores é apenas necessário escolher a ação que maximize $Q_\pi(s, a)$ para um dado estado s.

Assim como o método de programação dinâmica o método de Monte Carlo também consiste na intercalação de avaliação e melhoria da política. Mas esse tipo de algoritmo possui algumas limitações, um delas está no fato em que apenas após um episódio é que se pode alterar a política, outra limitação é devido a grande variância da estimativa do retorno, pois consiste no acúmulo de recompensas estocásticas ao longo de todo um episódio[24]. É importante considerar também que neste método temos o problema do dilema *exploration* vs *exploitation*, explicitado na sessão 2.1.3, ou seja, assim que os valores começam a ser estimados há o risco da melhoria de política de ignorar ações de menor valor estimado, mas com valor real maior.

2.2.3 Diferença temporal

Aprendizado por diferença temporal (TD, do inglês *Temporal Difference*) é uma combinação das ideias dos métodos de programação dinâmica e Monte Carlo. Como nos métodos de Monte Carlo, TD pode aprender diretamente da experiência, sem um modelo do ambiente. Como nos métodos de DP, TD atualiza a função valor a cada intervalo de tempo a partir da experiência adquirida no tempo anterior, mas sem que haja necessidade de um episódio inteiro [1].

Os métodos de TD esperam apenas uma transição de estado para atualizar V_{S_t}. O método mais simples de TD – conhecido como TD(0) ou DT de um passo – atualiza o valor da seguinte maneira [25]:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)), \tag{2.10}$$

onde α é a taxa de aprendizagem e γ é o fator de desconto da diferença entre a percepção de valores no instante t e $t+1$. O ajuste no valor do estado s_t é baseado no retorno imediato r_{t+1} e na estimativa do próximo estado $V(s_{t+1})$.

Um dos métodos de TD mais usados é o *SARSA (state, action, reward, state, action)*, que é classificado como *on-policy*, ou seja, o valor de q_π é estimado continuamente para a política de comportamento π, e ao mesmo tempo π é modificada a partir dos valores de q_π.

A estimativa de $Q_\pi(s, a)$ pode ser feita essencialmente através do mesmo método da equação (2.10), mas agora considerando transições de pares estado/ação para estado/ação:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)). \tag{2.11}$$

Outro método bastante utilizado é o *Q-learning*, classificado como *off-policy*, neste caso a função aprendida de estado/ação Q se aproxima diretamente de Q^*, independente da política sendo seguida. Mas a política não deixa de ter efeito no aprendizado uma vez em que é ela que define qual pares estado/ação serão visitados.

Um passo de aprendizado neste método é tido como:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)). \tag{2.12}$$

Estes métodos de RL podem encontrar problemas quando o espaço de estados é grande. Um problema comum é que, devido ao grande número de estados, a estimativa da função valor Q, em sua forma tabular, pode ser inviável por causa da grande quantidade de memória necessária para armazenar a tabela e, além disso, apenas convergir após cada estado ser visitado várias vezes [25].

2.2.4 Diferença temporal com elegibilidade

Algoritmos de diferença temporal com elegibilidade TD(λ) utiliza elementos da programação dinâmica, dos métodos de Monte Carlo e métodos de diferença temporal utilizando um mecanismo denominado traço de elegibilidade. Traço de elegibilidade é um mecanismo de registro temporário da ocorrência de um determinado evento – visita a um estado ou seleção de uma ação – de forma a classificá-los como elegíveis para mudança no aprendizado em andamento.

O traço de elegibilidade para dado estado s no instante t é denotado por $e_t(s) \in \Re^+$. A cada passo t, o traçado de elegibilidade de todos os estados decai um fator de $\gamma\lambda$ e o traço correspondente ao estado visitado s_t é incrementado em uma unidade [1]:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s), & \text{se } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1, & \text{se } s = s_t \end{cases} \tag{2.13}$$

onde γ é o desconto, já citado em seções anteriores, e λ é o decaimento do traço de elegibilidade ($0 \leq \lambda \leq 1$), que determina como o futuro da recompensa vai ser considerado.

Esse traço é dito acumulativo [2], porque a cada visita ao estado ele é acumulado mas decai gradualmente enquanto o estado não é visitado, isso fica mais claro na Figura 2.2.

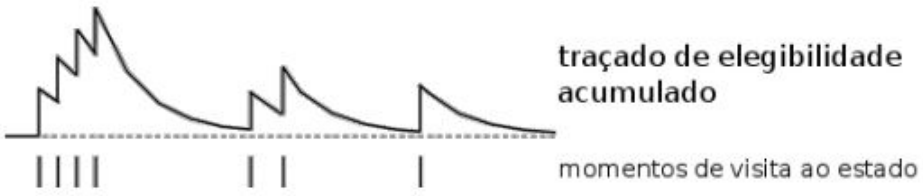


Figura 2.2: Traço de elegibilidade, mostrando acumulação ao longo do tempo [2].

O conjunto de traços indicam quão recentemente um estado foi visitado a qualquer instante de tempo e consequentemente indica quanto os estados são importantes para que o aprendizado ocorra.

TD(λ) é obtido pela combinação do traço de elegibilidade dado pelo equação (2.13) e com o método de atualização do função valor dado na equação (2.10), deste modo:

$$V(s_t) \leftarrow V(s_t) + \alpha\left[(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))\right] e_t(s), \text{ para } \forall s \in S. \tag{2.14}$$

Essas atualizações podem ser feitas a cada passo de tempo t (sistema "*on-line*") ou realizadas ao

final de cada episódio (sistema "*off-line*").

O traço de elegibilidade unifica e generaliza métodos de TD e Monte Carlo. Quando $\lambda = 1$, temos um método de Monte Carlo, pois o crédito dado a todos os estados decaem na razão de γ, e quando $\lambda = 0$, temos todos os traços nulos no instante t, exceto o correspondente a s_t , e a equação (2.14) corresponde a do método de atualização do algoritmo mais simples de TD, descrito na seção anterior.

2.2.5 Métodos de gradiente de política

Os métodos explicitados anteriormente têm em comum a necessidade de se estimar a função valor das ações para então selecionar ações baseados nestes valores, ou seja, a política nem sequer existiria sem a estimativa da função valor da ação.

Nos métodos de gradiente de política o que é aprendido é uma política parametrizada que é capaz de selecionar ações sem que seja necessário calcular o valor da ação. Uma função valor ainda pode ser utilizada para aprender a política parametrizada, mas não é necessária para selecionar uma ação [1]. Uma das vantagens desse tipo de algoritmo é que ele pode ser facilmente aplicado em problemas com grande espaço de estados devido a sua alta capacidade de generalização [26].

Em métodos de gradiente de política, a probabilidade de executar uma ação a em um estado s é representada por uma função π da ação a e do estado s, parametrizada por um conjunto de parâmetros θ [24]:

$$Pr\{a|s\} = \pi_\theta(s, a). \tag{2.15}$$

Um retorno G é obtido pelo agente após uma sequência de ações dependente de uma política. Logo, uma varição $\Delta\theta$ nos parâmeros da política deve causar uma variação correspondente ΔG no retorno:

$$\Delta G \approx \nabla_\theta G \cdot \Delta\theta. \tag{2.16}$$

Onde $\nabla_\theta G$ é o gradiente da retorno em relação aos parâmetros da política. Sabendo que o vetor gradiente aponta na direção de maior crescimento da função e, deste modo, ao se variar o parâmetros θ na direção do gradiente, com um passo α, o crescimento é garantido [24].

A expressão final para a atualização dos parâmetros da política é dado por:

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta G, \tag{2.17}$$

sendo α a taxa de aprendizado do algoritmo.

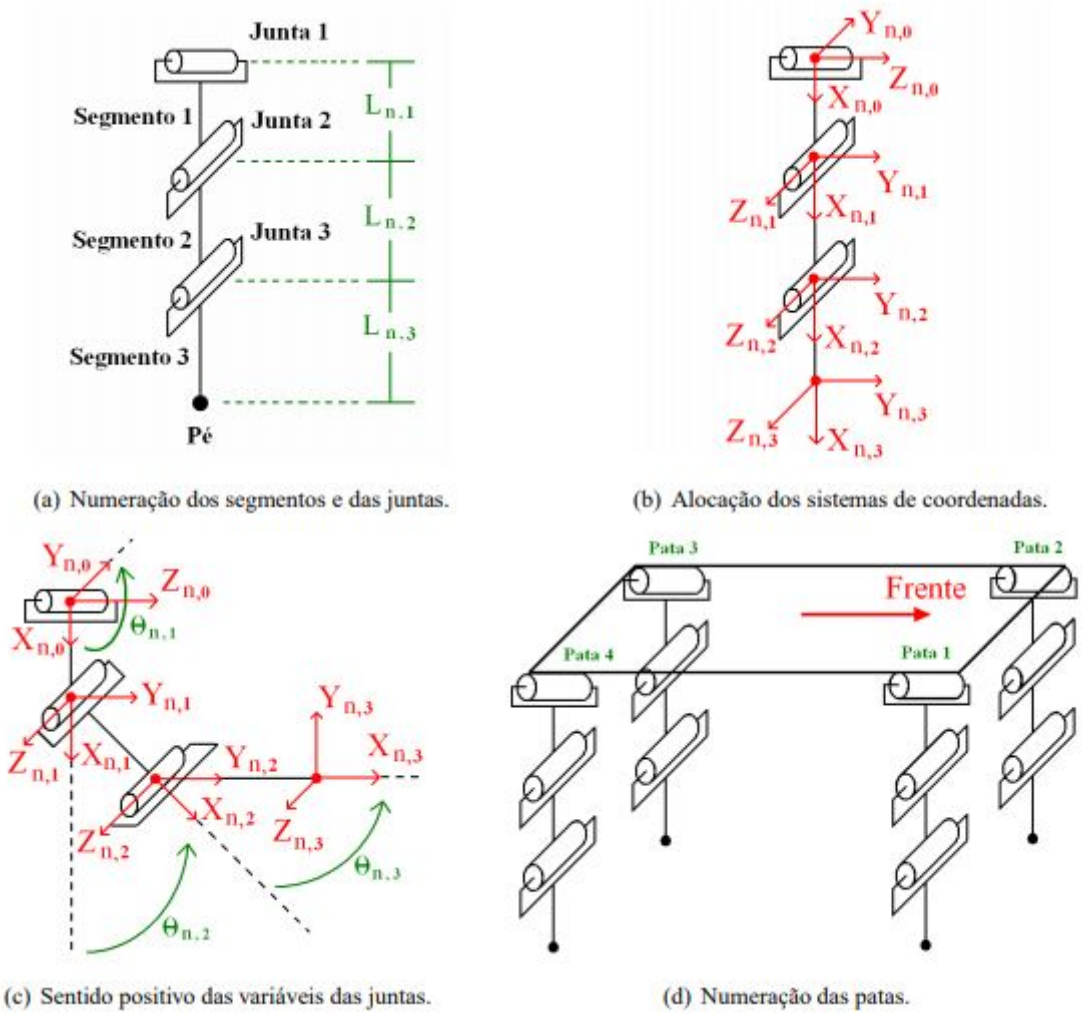
Devido ao modelo do sistema não ser conhecido, não há uma forma exata de se calcular $G(\theta)$ ou o gradiente $\nabla_\theta G(\theta)$, então, é necessário estimar o gradiente a partir de recompensas recebidas por experiência.

2.3 ESTRUTURA DO QUADRÚPEDE

A plataforma quadrúpede é um dispositivo robótico composto de quatro patas, cada uma possuindo três servos, totalizando doze graus de liberdade. A Figura 2.3 representa a disposição dos servos na plataforma e a convenção de numeração das patas.

A estrutura atual da plataforma do quadrúpede do LARA é fruto uma reestruturação foi feita por Santos [15], onde o robô possui uma estrutura de alumínio e patas de nylon. Algumas melhorias foram realizadas por [16] como: fixação de conexões e adição de borracha nas extremidades das patas.

Os servos utilizados são do modelo *Dynamixel* RX-28 da *Robotis*, a alimentação desse servo deve ser de no mínimo 12 Volts, a tensão recomendada pelo fabricante é de 14,8 Volts [6]. O torque máximo de servo é de 3,7N.m.



(a) Numeração dos segmentos e das juntas.

(b) Alocação dos sistemas de coordenadas.

(c) Sentido positivo das variáveis das juntas.

(d) Numeração das patas.

Figura 2.3: Sistemas de coordenadas do robô [3].

2.3.1 Sensoriamento

O sistema de sensoriamento do robô foi feito por Porphiro e Santana [16]. O objetivo do sensoriamento é permitir a obtenção de informações sobre o atual estado do robô, como, por exemplo, quais patas estão em contato com o solo. Atualmente o sistema implementado no robô quadrúpede oferece medidas sobre a aceleração, por meio de um acelerômetro, e medidas das forças nas extremidades das patas, utilizando um sensor resistivo de força.

A estimação da força aplicada nas patas do robô se faz necessária por dois motivos principais: a avaliação da carga que cada ponto de apoio do robô está submetido e análise do instante de contato entre a pata e o solo. Para medir a força exercida por cada pata do robô são utilizados sensores resistivos de força do tipo FSR 400. Estes sensores estão localizados entre a pata do robô e a bota (ponteira de borracha para cadeira - 1/2"). Segundo o manual do fabricante [27], esse sensor possui uma resolução de 0,2N e uma faixa de atuação entre 0,2 e 20N.

Um acelerômetro é necessário para que seja possível ter uma orientação precisa do centro de massa do robô, e com isso, prever se o robô está seguindo a trajetória desejada. Além disso, a estabilidade do robô é obtida a partir do controle da rolagem e da arfagem do centro de massa da plataforma.

Na plataforma foi utilizado o Acelerômetro Digital ADXL345. Seu funcionamento se dá por meio do efeito capacitivo, proporcionado por um sistema microeletrônico que consiste de uma massa móvel, acoplada a conjunto de superfícies de silício policristalino. Acelerômetros são dispositivos sensíveis a diferença entre a aceleração do sensor e a aceleração da gravidade local. Quando o acelerômetro se encontra parado e sem angulações de rolagem ou arfagem a aceleração medida é de 1g no eixo y.

2.3.2 Sistemas embarcados

Nos trabalhos de Porphirio e Santana [16] e Santos [15] foi definido que o quadrúpede seria controlado por uma integração entre *Raspberry PI* e o *Arduíno*, em uma relação mestre-escravo. A função do Arduíno é coletar dados dos sensores de força e do acelerômetro e enviá-los ao *Raspberry PI*, via serial. O *Raspberry PI* é responsável por controlar os servo motores a partir de uma biblioteca da *Dynamixel*, em C++.

O *Arduíno Uno* é uma placa composta por um microcontrolador do tipo ATmega328, 14 pinos de entradas digitais e 6 pinos de entrada analógica, além de uma entrada USB, conexão para energia do tipo *power jack* podendo ser alimentado com tensão limite 20 V. É uma placa com fim didático, simples de usar mas não deixa de ser flexível o suficiente para projetos com um certo nível de complexidade.

O *Raspberry Pi* é um microcomputador do tamanho de um cartão de crédito de baixo custo e grande capacidade que conta com um processador ARM Cortex-A53 de 1,2Ghz que roda a 900Mhz e conta com um 1GB de RAM. Devido às suas pequenas dimensões é uma alternativa

para se embarcar em um robô.

2.4 DEFINIÇÕES DA MARCHA

O trabalho realizado de Dos Santos e Duarte [4] buscava desenvolver marchas factíveis para a plataforma quadrúpede utilizando algoritmo genético. Em um primeiro momento a busca consistia em encontrar um conjunto de ângulos para as juntas do robô, mas devido ao grande espaço de busca não foram obtidos bons resultados. Na segunda tentativa foram modeladas 4 poses para as patas do quadrúpede utilizando cinemática inversa, essas poses foram inspiradas em caminhadas de cavalos e podem ser vistas na Figura 2.4, e a busca consistia em encontrar a melhor combinação de poses. Na terceira abordagem, a ordem das poses foi fixada e a busca se restringiu a otimizar parâmetros relacionados a essas poses.

Os parâmetros a serem encontrados influenciam as extremidades (*end-effectors*) das patas. Eles são: ângulos de aberturas das poses 2 ($angle - front$) e 3 ($angle - back$) e posição no espaço da extremidade durante pose de transição, pose 1 (x_c, y_c, z_c).

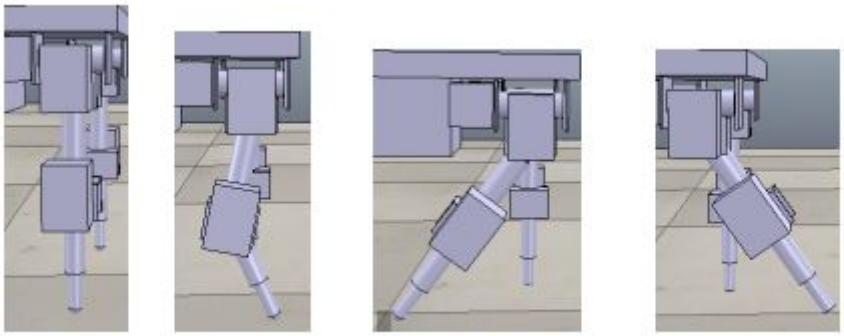


Figura 2.4: Poses das patas geradas por cinemática inversa [4].

2.4.1 Cinemática inversa

De acordo com Erthal [28] a cinemática inversa de posição é o processo de obtenção das coordenadas das juntas, dadas as coordenadas externas do efetuador, em outras palavras, obter os ângulos que os atuadores terão que girar ou a distância que terão que se deslocar para que um ponto de interesse seja atingido pela sua extremidade. Matematicamente, ela se resume na solução da equação:

$$\vec{X} = f(\vec{q}) \tag{2.18}$$

Onde $\vec{X}$ é o vetor coordenadas externas ou espaciais, dado por $\vec{X} = [x_1\ x_2\ \ldots\ x_M]$, M representa o número de coordenadas necessárias para representar o efetuador no espaço externo. $\vec{q}$ é o vetor coordenadas das juntas, dado por $\vec{q} = [q_1\ q_2\ \ldots\ q_N]^T$, cada coordenada corresponde a

um grau de liberdade e N representa o número graus de liberdade do robô.

Segundo [28] a cinemática inversa pode ser utilizada em duas situações. Uma é no planejamento de trajetórias quando ela é feita em coordenadas externas e essas coordenadas devem ser convertidas em coordenadas de juntas para o controle da posição e do movimento de cada junta. A segunda aplicação é na geração de trajetórias em tempo real, ou seja, a trajetória é definida de acordo com sinais captados por sensores.

A obtenção da cinemática inversa é complexa por se tratar da solução de sistemas de equações não-lineares. A solução pode não existir e, se existir, ela pode não ser única. A cinemática inversa diferencial é a solução da equação

$$\delta\vec{X} = \frac{\partial f(\vec{q})}{\partial q} \cdot \delta q, \tag{2.19}$$

onde $\delta\vec{X}$ e $\delta\vec{q}$ representam os deslocamentos infinitesimais no efetuador e nas juntas.

A solução é dada por:

$$\delta\vec{q} = \vec{J}^{-1} \cdot \delta\vec{X} \tag{2.20}$$

o que resulta no problema da inversão da matriz Jacobiana.

2.4.2 Modelagem das poses

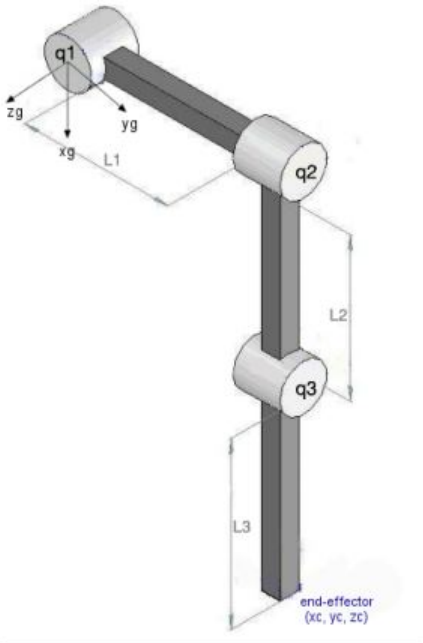


Figura 2.5: Modelo de pata [4].

Como foi dito anteriormente, a partir da cinemática inversa, as poses foram modeladas, ou seja, foi necessário encontrar um conjunto de equações que descrevessem o comportamento dos ângulos dos atuadores em função da extremidade da pata. O modelo da pata utilizado pode ser encontrado na Figura 2.5. Tomando θ_1 como o ângulo da junta q_1, θ_2 o ângulo da junta q_2, θ_3 o

ângulo da junta q_3, (x_c, y_c, z_c) o ponto no espaço da extremidade. L_2 o tamanho da conexão entre as juntas q_2 e q_3 e L_4 o tamanho da conexão entre a junta q_3 e a extremidade. No robô real L_1, a distância entre q_1 e q_2, é desprezível. Dados os eixos de referência (x_g, y_g, z_g) da 2.5, os ângulos das juntas são dados por:

$$\begin{aligned}
\theta_1 &= \tan^{-1}(\tfrac{y_c}{x_c}) \\
\theta_2 &= \tan^{-1}(\tfrac{z_c}{x_c}) - \tan^{-1}\left(L_3\cos(\theta_1)\tfrac{\sin(\theta_3)}{L_2\cos(\theta_1)+L_3\cos(\theta_1)\cos(\theta_3)}\right) \\
\theta_3 &= \pi - \cos^{-1}\left(\tfrac{-x_c^2-z_c^2+(L_3\cos(\theta_1))^2+(L_2\cos(\theta_1))^2}{2L_2L_3\cos^2(\theta_1)}\right)
\end{aligned} \tag{2.21}$$

Com essas equações em mãos, tem-se um código que recebe como parâmetro o end-effector. Primeiramente o código realiza a leitura de um arquivo de texto em que estão definidos os parâmetros de uma marcha. É gerada, então, uma marcha a partir dos parâmetros obtidos. Essa marcha consiste na descrição dos ângulo de cada motor, que são as soluções da equação (2.21). A ordem das poses são restringidas em cada pada da seguinte maneira:

- **Pata 1:** Pose 3 $\rightarrow$ Pose 4 $\rightarrow$ Pose 1 $\rightarrow$ Pose 2;
- **Pata 2:** Pose 1 $\rightarrow$ Pose 2 $\rightarrow$ Pose 3 $\rightarrow$ Pose 4;
- **Pata 3:** Pose 3 $\rightarrow$ Pose 4 $\rightarrow$ Pose 1 $\rightarrow$ Pose 2;
- **Pata 4:** Pose 1 $\rightarrow$ Pose 2 $\rightarrow$ Pose 3 $\rightarrow$ Pose 4;

Essa ordem de poses é chamado de ciclo e ele é repetido duas vezes. Tem-se ao final um arquivo texto que contém matriz 10×12, em que as linhas são as poses do robô e as colunas, os ângulos de cada junta. O robô inicia o movimento, depois que todos os seus servos foram calibrados, na posição zero e termina novamente com os servos na posição zero.

2.5 SOFTWARE DE SIMULAÇÃO

Como as interações de um robô com o ambiente são definidas por diversas variáveis e condições a previsão de seu comportamento se torna uma tarefa difícil. Segundo Kelton [29], o termo simulação se refere a uma ampla coleção de métodos e aplicativos para imitar o comportamento de sistemas reais. A simulação, na robótica, tem como objetivo validar o comportamento de um robô por meio de uma modelagem simplificada do mundo real. Uma das vantagens da simulação é o seu uso no desenvolvimento da programação *off-line* – elaboração de programas em um computador que serão posteriormente carregados no robô – onde é possível que sejam feitos testes dos programas antes da execução no robô real [30]. O uso de simuladores é vantajoso quando o robô está inserido em ambientes de risco para o operador e/ou desgaste do equipamento, ou para evitar preparações requeridas do equipamento real como, por exemplo trocas de bateria e reposicionamentos [31].

São diversos os programas para simulação de robôs, podemos citar: *Open HRP, Gazebo, Webots* e *V-REP*. Segundo Rohmer, Singh e Freese [32] mesmo todos competindo em quesitos de funcionalidade, o *V-REP* se destaca por oferecer uma grande gama de técnicas de simulação e pela portabilidade de seus modelos e controladores. Por essa razão o *software V-REP* foi escolhido por Porphirio e Santana [16] para simulação da plataforma quadrúpede.

O *V-REP* permite parâmetros de simulação sejam manipulados por API (Interface de Programação de Aplicativos) remotas. Essas interfaces permitem que toda a movimentação do robô, assim como e parâmetros do ambiente de simulação, sejam controlados a partir de aplicações externas em linguagem de programação escolhida. Conforme consta no manual do *software V-REP* [33], esta API remota proporciona mais de 100 funções que proporcionam o controle do robô em simulação, e essa interação com o V-REP pode acontecer de maneira síncrona ou assíncrona.

3 MÉTODOS EMPREGADOS

Neste capítulo trata-se sobre as metodologias aplicadas para o desenvolvimento do otimização da marcha. Primeiramente foi feita a implementação de um algoritmo de diferença temporal e depois foi utilizado outro algoritmo, o de gradiente de política, os dois em ambiente simulado. Após o método de gradiente de política se mostrar hábil e de fácil aplicação criou-se um programa em Python para implementa-lo na plataforma física.

3.1 INTRODUÇÃO

A partir do modelo de marcha, criado Dos Santos e Duarte [4], esse trabalho se propõe a otimizar parâmetros, mencionados na seção 2.4, para a geração de uma marcha satisfatória de acordo com avaliações específicas. Para isso foram utilizados dois métodos diferentes de aprendizado por reforço.

Em primeiro momento foi utilizado um algoritmo de diferença temporal com traço de elegibilidade para a otimização destes parâmetros, mas ao se mostrar demorado e não muito efetivo outro algoritmo foi utilizado. Este segundo algoritmo é baseado no gradiente de política. Este se mostrou mais rápido e eficaz, sendo factível até mesmo em uma aplicação real, sem a necessidade de se fazer simulações prévias como no algoritmo anterior.

3.2 DIFERENÇA TEMPORAL COM TRAÇO DE ELEGIBILIDADE

O algoritmo de diferença temporal foi escolhido para ser aplicado pois o agente, nosso robô quadrúpede, precisa aprender apenas os melhores valores para os parâmetros que são usados para gerar uma marcha. Considerando que o estado é um conjunto de parâmetros o importante é determinar qual o melhor estado (valores ótimos do conjunto de parâmetros), a ação a ser tomada em cada estado não é importante. Como visto no seção 2.2.3, neste tipo de algoritmo se é calculado o valor de um estado $V(s)$.

Para a geração de uma marcha são utilizados 5 parâmetros -— *angle-back, angle-front,* x_c*,* y_c*,* z_c -— mas para uma melhor análise da eficiência deste algoritmo foram variados apenas 3 deles. Quando mais parâmetros são variados maior é o espaço de estados e a análise dos valores dos estados se torna muito complexa e o tempo de simulação muito extenso. Para a aplicação deste tipo de algoritmo cada parâmetro que está sendo analisado foi discretizado em um intervalo específico. Isso é necessário para que o espaço de estados seja reduzido e, além disso, evitar valores que desconfiguram as características de cada pose. Os valores de mínimo e máximo de

cada parâmetro foi definido no trabalho de Dos Santos e Duarte [4]. Os intervalos de valores que cada parâmetro pode assumir estão resumidos abaixo:

- $angle - back$ e $angle - front$: Estes parâmetros são os ângulos em que a pata do robô faz nas poses 2 e 3, que podem ser vistas na Figura 2.4. Eles que se mantiveram constante, foram dados a eles um valor de $30°$, pois nas marchas obtidas por Dos Santos e Duarte [4] eles se aproximavam deste valor;
- x_c: Representa a posição x que a extremidade da pata do robô deve alcançar em um pose intermediária. Limitado no intervalo [17,21] cm, com um passo de 0,4.
- y_c: Representa a posição y que a extremidade da pata do robô deve alcançar em um pose intermediária. Limitado no intervalo [-4,0] cm, com um passo de 0,4.
- z_c: Representa a posição z que a extremidade da pata do robô deve alcançar em um pose intermediária. Limitado no intervalo [0,4] cm, com um passo de 0,4.

O objetivo deste trabalho é a otimização dos valores de determinados parâmetros buscando a geração de uma marcha que atinja resultados satisfatórios em relação a uma avaliação específica. Em um primeiro momento essa avaliação é a distância máxima percorrida pelo robô no eixo y. Nossa função objetivo é, então, a maximização desta distância.

Na aplicação deste algoritmo o problema de aprendizado é definido como um MPD, que é composto de:

- S: Estados s que são um conjunto de parâmetros, $s(x_c, y_c, z_c)$;
- A: Ações $a(s)$: pode incrementar o valor de um parâmetro, decrementar o valor de um parâmetro, e continuar no mesmo estado.
- $T(s, a, s')$: Função de transição determinística que permite o incremento ou decremento de apenas um parâmetro e ainda permite o sistema ficar no mesmo estado.
- r: Uma recompensa negativa é dada ao robô quando ele falha em andar certa distância mínima, ou quando cai, e uma recompensa positiva é determinada como uma função da distância percorrida.

Nesta caso, ao trabalharmos com apenas 3 parâmetros, $s(x_c, y_c, z_c) \in S$, haverá 3 ações de incremento de parâmetro, 3 ações de decremento e uma ação para ficar no mesmo estado, totalizando 7 ações $a(s) \in A$. O algoritmo para otimizar estes parâmetros utilizando TD(λ), é mostrado no Algoritmo 3.1.

Algoritmo 3.1: Algoritmo TD(λ)

Resultado: Valores dos estados

Iniciar $V(s)$ arbitrariamente;
para $\forall a \in A$ **faça**
    $Q(a) \leftarrow 0$
fim
enquanto *o número total de episódios não é atingido (N)* **faça**
    Estado inicial s escolhido de modo aleatório;
    para $\forall s \in S$ **faça**
        $e(s) \leftarrow 0$;
    fim
    $score \leftarrow$ Simulação da marcha gerada pelo estado s;
    se $score \leq 0,3$ **então**
        $r \leftarrow$ recompensa negativa ;
    senão
        $r \leftarrow$ recompensa positiva ;
    fim
    enquanto *o número total de marchas não é atingido (M)* **faça**
        $a \leftarrow$ ação do estado s derivada de uma política ϵ-greedy ;
        se *a ação escolhida faz com que os parâmetros são do intervalo estabelecido* **então**
            r=-200;
            Atualização do valor de $Q(a)$;
            Retorna-se ao inicio do ciclo para outra ação ser escolhida ;
        fim
        Ao se tomar a ação a e observar o estado s':
        $\delta \leftarrow r + \gamma V(s') - V(s)$;
        $e(s) \leftarrow e(s) + 1$;
        para $\forall s \in S$ **faça**
            $V(s) \leftarrow V(s) + \alpha\delta e(s)$;
            $e(s) \leftarrow \gamma\lambda e(s)$
        fim
        $score \leftarrow$ Simulação da marcha gerada pelo estado s';
        se $score \leq 0,3$ **então**
            $r \leftarrow$ recompensa negativa ;
        senão
            $r \leftarrow$ recompensa positiva ;
        fim
        Atualiza-se o valor de $Q(a)$;
        $s \leftarrow s'$
    fim
fim

Ao passo de 0,4 cm cada parâmetro pode assumir 11 valores diferentes e assim o número total de estados é de 1.331 ($11 \times 11 \times 11$). Na tentativa de que todos estes esses estados sejam visitados ao menos uma vez, o número de marchas geradas deve ser bem maior que o número de estados. Outra maneira de tentar garantir essa visita é que ao inicio de cada episódio o estado inicial seja escolhido de forma aleatória.

Na Tabela 3.1 temos as variáveis de aprendizado, que foram escolhidas de acordo com trabalhos anteriores que utilizaram a mesma estratégia como o trabalho de Silva, Perico, Costa e Bianchi [34].

Número de episódios (N)	200
Marchas por episódio (M)	30
Taxa de aprendizado (α)	0,1
Taxa de exploração (ϵ)	0,5
Decaimento da elegibilidade (λ)	0,05
Desconto (γ)	0,90

Tabela 3.1: Configuração das variáveis de aprendizado para o algoritmo de TD(λ).

Ainda se baseando no trabalho de Silva, Perico, Costa e Bianchi [34] foi definida uma função exponencial para a recompensa para separar parâmetros que geram marchas que percorrem menores distâncias:

$$r = \begin{cases} -100 & \text{se } score \leq 0,3 \\ K_1 10^{K_2 \times score} & \text{de outro modo} \end{cases} \tag{3.1}$$

onde K_1 e K_2 são constantes usadas para influenciar o reforço. K_1 é a constante de ganho linear que assume o valor de $0,33$ e K_2 é a constante de ganho exponencial, definida como 5. O $score$ é a distância total percorrida pelo quadrúpede no eixo y.

Para a escolha de ações foi utilizada uma politica ϵ-greedy em que $\epsilon = 0,5$, deste modo, metade das vezes se escolhe a ação de acordo com valor da ação $Q(a)$, definido como a média de todas as recompensas recebidas quando esta ação foi selecionada. Na outra metade a ação é escolhida de modo aleatório, para que haja uma exploração de outros estados.

3.3 GRADIENTE DE POLÍTICA

Ao se verificar que o algoritmo anterior demorava muito tempo para encontrar os valores dos estados, foi buscado um outro tipo de algoritmo, um que fosse mais rápido e em que não houvesse a necessidade de discretizar os parâmetros. Uma escolha lógica seria a utilização de um algoritmo de gradiente de política devido a sua capacidade de se trabalhar em grande espaço de estados.

Neste algoritmo deixamos de olhar os parâmetros com o estado do agente mas consideramos cada conjunto de parâmetros possíveis como uma definição de uma política *open-loop* que pode ser executada pelo robô. Para estimar o gradiente da politica é necessário considerar que a politica é diferenciável em respeito a cada parâmetro.

Como não se sabe nada sobre a real forma da política, não é possível calcular exatamente o gradiente. Estimar o gradiente por amostragem empiricamente pode ser uma tarefa difícil e dado o tamanho do espaço de busca, muito demorado. Deste modo, teríamos o mesmo problema do algoritmo anterior e a utilização deste novo não traria nenhuma vantagem. No trabalho de Kohl e Stone [5] é apresentado um método muito eficiente para calcular a estimativa do gradiente de política. Assim como em nosso trabalho, eles também buscam a otimização de parâmetros que geram uma marcha. Deste modo, a metodologia utilizada por eles pode ser aplicada a este projeto com algumas adaptações.

O método utilizado pode ser considerado uma derivação dos métodos de gradiente de política padrões. Ao utilizarmos métodos de gradiente de política a convergência em busca da política ótima apenas se aproximará do máximo local, em contraste com a técnica utilizada anteriormente em que converge para um máximo global.

Nesta técnica consideramos um vetor de parâmetros inicias $\pi = \{angle - back, angle - front, x_c, y_c, z_c\}$ e em seguida é estimada a derivada parcial da função objetivo de π em relação a cada parâmetro. Isso é feito primeiramente avaliando n políticas próximas de π $(R_1, R_2, \cdots, R_n)$, que são geradas aleatoriamente. Essas políticas são definidas como:

$$R_i = \{angle - back + \Delta_1, angle - front + \Delta_2, x_c + \Delta_3, y_c + \Delta_4, z_c + \Delta_5\}, \tag{3.2}$$

onde Δ_n é escolhido de modo aleatório entre três opções $\{+\epsilon_n, 0, -\epsilon_n\}$. Cada ϵ_n é um valor fixado de acordo com o parâmetro a qual ele está ligado.

Assim como na técnica anterior a política é avaliada por meio de simulação a qual nos retorna uma nota $(score)$ da marcha descrita por cada política. Essa nota pode ser tanto a distância percorrida por y, a variação no eixo x,a variação no eixo z, ou até mesmo uma média ponderada de todos estas medidas. A função objetivo pode ser maximizar, ou até mesmo minimizar, estes valores.

Depois que todas as políticas são avaliadas são estimadas as derivadas parciais em relação a cada um dos 5 parâmetros. Isso é feito agrupando cada R_i em um dos 3 conjuntos para cada umas das dimensões:

$$R_i \in \begin{cases} S_{+\epsilon,n} & \text{se o enésimo parâmetro de } R_i \text{ é } p_n + \epsilon_n \\ S_{0,n} & \text{se o enésimo parâmetro de } R_i \text{ é } p_n + 0 \\ S_{-\epsilon,n} & \text{se o enésimo parâmetro de } R_i \text{ é } p_n - \epsilon_n \end{cases} \tag{3.3}$$

É computado um nota média $Avg_{+\epsilon,n}$,$Avg_{0,n}$ e $Avg_{-\epsilon,n}$ para $S_{+\epsilon,n}$, $S_{0,n}$ e $S_{-\epsilon,n}$ respectivamente. Essas três médias representam qual é o beneficio de se alterar o enésimo parâmetro por $+\epsilon_n$, 0 ou $-\epsilon_n$. Esse mecanismo é melhor ilustrado na Figura 3.1, que mostra um exemplo desse processo em apenas uma dimensão, onde cada R_i é agrupado em uma das três categorias, dependendo do valor do parâmetro (π_1) de cada (R_i).

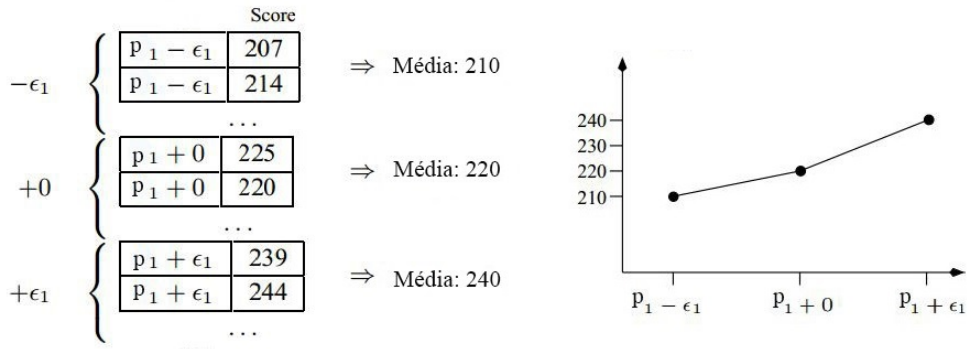


Figura 3.1: Exemplo do processo para estimativa do gradiente em uma dimensão. (Adaptado de [5])

Usamos essas médias para ajustar um vetor A com 5 variáveis, onde:

$$A_n = \begin{cases} 0 & \text{se } Avg_{+0,n} > Avg_{+\epsilon,n} \text{ e } Avg_{+0,n} > Avg_{-\epsilon,n}, \\ Avg_{+\epsilon,n} - Avg_{-\epsilon,n} & \text{de outro modo} \end{cases} \tag{3.4}$$

O vetor A é, então, normalizado e multiplicado por um escalar de passo η, para que o ajuste seja fixo a cada iteração. Ao final adicionamos A ao vetor π, e uma nova iteração é iniciada. O Algoritmo 3.2 mostra todos os passos utilizado para obtenção dos parâmetros ótimos.

Algoritmo 3.2: Algoritmo do gradiente de política

Resultado: Parâmetros ótimos

$\pi \leftarrow$ Política inicial;
enquanto *o numero de iterações total não é alcançado* **faça**
  M políticas são geradas aleatoriamente;
  para $\forall R_i$, *onde* $i = 1, 2, \cdots M$ **faça**
    $score_i$ obtido através de simulação
  fim
  para $\forall n$, *onde* $n = 1, 2, \cdots 5$ **faça** `// n vai até 5 pois são utilizados 5 parâmetros na geração da marcha`
    $Avg_{+\epsilon,n} \leftarrow$ média dos $scores$ para todos R_i que tenham tido uma variação positiva para o enésimo parâmetro;
    $Avg_{0,n} \leftarrow$ média dos $scores$ para todos R_i que tenham tido uma variação zero para o enésimo parâmetro;
    $Avg_{-\epsilon,n} \leftarrow$ média dos $scores$ para todos R_i que tenham tido uma variação negativa para o enésimo parâmetro;
    se $Avg_{+0,n} > Avg_{+\epsilon,n}$ *e* $Avg_{+0,n} > Avg_{-\epsilon,n}$ **então**
      $A_n \leftarrow 0$
    senão
      $A_n \leftarrow Avg_{+\epsilon,n} - Avg_{-\epsilon,n}$
    fim
  fim
  $A \leftarrow \frac{A}{|A|} \times \eta$;
  $\pi \leftarrow \pi + A$;
fim

Quando o algoritmo foi implementado a cada iteração eram avaliadas 5 marchas, como as simulações no *V-REP* possuem muito ruído cada marcha era simulada 3 vezes. A cada vez que o algoritmo era rodado eram feitas 10 iterações. O algoritmo era rodado pelo menos duas vezes, se em uma nova rodada pelo menos uma política teve $score$ maior que todas as políticas da rodada anterior o algoritmo era rodado mais uma vez, caso contrario não se era mais rodado e a política que obteve maior $score$ na rodada anterior era considerada a política ótima.

Foram usadas varias funções objetivos diferentes para esse método de aprendizado, que estão descritas abaixo:

- Maximização da distância no eixo y. Para isso o $score$ é considerado a distância total percorrida neste eixo.
- Minimização da variação no eixo z, ou seja, minimizar o balanço da altura do robô na tentativa de penalizar políticas em que o robô acaba por cair. Para isso o $score$ é o inverso da diferença entre o maior e menor valor lido pelo GPS no eixo z de coordenadas. Se utiliza do inverso pois o algoritmo visa maximizar o $score$.
- Minimização da variação no eixo x, pois esperasse que o robô ande apenas no eixo y, se há uma diferença grande no eixo x que dizer que a marcha gerada pende para um algum lado. Utiliza-se novamente o inverso da diferença entre os valores máximo e mínimos lidos pelo GPS.
- Maximização da média ponderada da distância percorrida em y, inverso da variação em z e inverso da variação em x. Neste caso buscamos que todas as características buscadas individualmente anteriormente sejam satisfeitas em uma única marcha.

3.4 PLATAFORMA REAL

A plataforma real, neste trabalho, foi utilizada principalmente para a avaliação dos resultados obtidos nas simulações. Como o objetivo da otimização era apenas o $score$ obtido não era muito importante que a distância percorrida pelo robô real fosse exatamente igual ao resultado da simulação, esperava-se apenas que a mudança nos parâmetros gerassem marchas melhores nas duas plataformas.

Para aferir as medidas das marchas por simulação foi utilizado o mesmo local de teste definido por Dos Santos e Duarte [4]. A Figura 3.2 ilustra como foi montado esse local de testes. A posição inicial do robô era demarcada com fita isolante e ao final de uma marcha era medido com fitas métricas o deslocamento total nos eixos x e y usando uma das patas do robô como referência. Neste teste não era possível fazer uma medida da variação em z – balanço da altura do robô – e apenas era avaliada qualitativamente a marcha, ou seja, se a marcha balançava apenas o suficiente para manter uma marcha fluida.

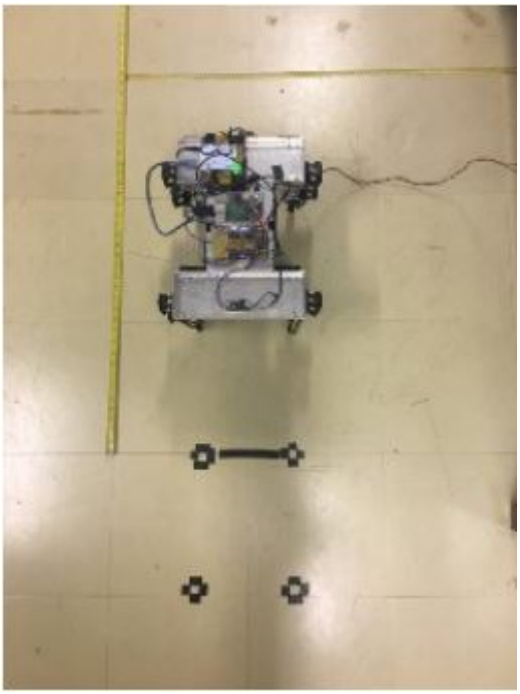


Figura 3.2: Local para testes com a plataforma real [4].

Para a geração da marcha no robô foi utilizada a interface de treinamento criada por Dos Santos e Duarte [4] com algumas adaptações. O código implementado por eles visava automatizar o máximo possível de processos para gerar e realizar as marchas no robô. Para iniciar a geração de uma marcha o código nos dava algumas opções: finalizar o programa, liberar os motores após a execução da marchas, calibrar o robô ou executar a marcha mantendo fixos após a execução. Após a geração da marcha os dados coletados da marcha eram copiados para uma pasta nomeada pelo usuário.

As alterações no código em *Python* foram necessárias para que os testes da marcha fossem mais eficientes. Primeiramente, uma nova opção na inicialização do código foi adicionada, a liberação da marcha sem que nenhuma marcha fosse executada. Isso foi necessário pois após o robô travar na tentativa de gerar uma marcha o programa podia ser rodado novamente e ao selecionar essa opção soltar os motores e, em seguida, recalibrar o robô e gerar outra marcha sem a necessidade de desligar o robô da fonte de alimentação, como era feito anteriormente quando situação semelhante ocorria. Outro ponto alterado no código foi a adição da escolha de salvar ou não os dados da marcha.

3.4.1 Aprendizado na plataforma

A fim de compensar as diferenças entre a simulação e aplicação real, foi escrito um código em *Python* para que o aprendizado ocorra diretamente na plataforma. Esse código implementa o método de gradiente de política. Apenas o método de gradiente de política é viável de ser aplicado diretamente na plataforma, porque, diferentemente do método de diferença temporal, nele não há

a necessidade da geração de um número muito grande de marchas para que o aprendizado ocorra.

O código funciona a partir de um conjunto de parâmetros inicias, após ser feita uma conexão SSH com a plataforma entra-se com o número de marchas a serem avaliadas. Para a geração das marchas outro código em *Python* é chamado e arquivos de texto que descrevem essas marchas são criados e alocados em uma pasta na plataforma. Para a execução das marchas é chamado um programa principal em C++, escrito por Porphirio e Santana [16] e modificado por Dos Santos e Duarte [4], que é responsável pela calibração do robô e execução das marchas descritas no arquivo de texto.

Essas marchas são executadas uma por uma e após sua execução é dada uma nota manualmente. Essa nota pode ser a distância percorrida ou uma medida qualitativa. É, então, executado Algoritmo 3.2 que retorna novos parâmetros de modo que a marcha gerada por eles seja melhor que a inicial. Por fim, pode-se alterar os parâmetros inicias para os obtidos pelo código para que seja feita nova iteração.

3.4.2 Problemas enfrentados

Em primeiro momento, ao se tentar executar marchas, o robô após dar os primeiros passos acabava caindo. Isso ocorria pois ao alcançar uma posição em que o robô se abaixa para dar um impulso para frente os motores acabavam não tendo torque para levantar o peso do robô. Ao se verificar o código principal e constatar que o torque limite dado aos motores era o valor máximo, e que este valor era o suficiente para que, em teoria, o robô conseguisse se levantar, foi levantada a hipótese de que o problema poderia estar na fonte de alimentação da plataforma.

A alimentação do robô estava sendo feita através de uma fonte de bancada com um valor de tensão de $13,5$ Volts. Ao se constatar problema da força dos motores outra alternativa de alimentação foi buscada. Foi decidido pelo uso de uma bateria de 12 Volts, tensão suficiente para a alimentação o funcionamento do robô. Com a mudança o robô passou a conseguir se levantar.

Apesar da mudança na fonte outro problema era recorrente, quando algumas marchas eram executadas, os motores de uma das patas eram liberados e por isso o robô pendia para um lado e caia. Alguns testes foram feitos com o *software RoboPlus* que permite uma fácil interface com os motores. Na opção *Dynamixel Wizard* é possível gerenciar facilmente os motores, incluindo calibração e acesso a tabela de controle, onde é possível testá-los. A interface do *software* pode ser vista na Figura 3.3.

Foi verificado que todos os robôs respondiam bem aos comandos, inclusive os motores da pata que estavam se desligando, verificando assim que não havia problemas. Deste modo, o problema podia estar na alimentação dos servos.

Após testes com o voltímetro para verificar se havia queda de tensão foi notado que os fios de alimentação dessa pata ficavam muito tensionados, e que em alguns movimentos, a conexão era interrompida momentaneamente. Por esta razão, quando o robô tentava executar algumas marchas

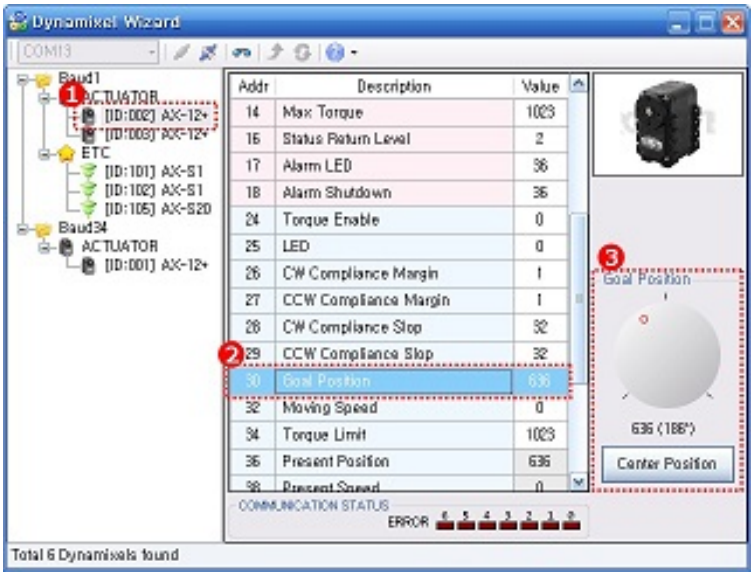


Figura 3.3: Interface do *software* utilizado para testes nos motores [6]

a pata se desligava, mas ao tentar executar outra marcha ela poderia rodar sem problemas.

Foram corrigidos todos os problemas e o robô estava executando todas as marchas, mas ainda se movia com dificuldade devido a folgas nos parafusos das patas. Todos os parafusos foram apertados e finalmente as marchas geradas nas simulações puderam ser avaliadas na plataforma real.

4 RESULTADOS

Aqui se expõe e se analisam os resultados obtidos dos algoritmos de otimizações propostos: diferença temporal e gradiente de política. O resultado é composto de gráficos da estimativa da função valor gerado pelo primeiro método, gráficos das resultados a cada iteração no segundo método e comparação das marchas obtidas por meio de gráficos de posições dos motores, de sensores e quadros de imagens que representam a execução das marchas.

4.1 DIFERENÇA TEMPORAL

A marcha proposta por Dos Santos e Duarte [4] para o quadrúpede consiste em 4 poses para cada pata em sequência definida. Um conjunto de 5 parâmetros pode alterar algumas características dessas poses. O objetivo do algoritmo de aprendizagem consiste em encontrar melhor conjunto de valores destes parâmetros de acordo com uma função objetivo.

Para a aplicação do algoritmo de diferença temporal – em que se busca uma estimativa para os valores de um estado – o conjunto de parâmetros descreve o estado do nosso robô e a função objetivo é a maximização da distância percorrida pelo robô no eixo y. Os parâmetros a serem otimizados pelo algoritmo descrevem a posição no espaço (x_c, y_c, z_c) da extremidade da pata na pose da Figura 2.4b, a pose de transição. Os parâmetros que descrevem os ângulos de abertura das outas poses não foram considerados na tentativa de restringir o espaço de busca, que, de outro modo, seria muito extenso.

A execução do algoritmo demorou mais de 15h horas e foram feitas 6 mil simulações. Ao final obtivemos como resultado a estimativa dos valores dos estados, $V(x_c, y_c, z_c)$. Apesar dos parâmetros serem valores discretos com um passo de $0,4$ o número de estados possíveis chega a $1,331$. Para a obtenção de estimativas mais precisas seriam necessárias mais iterações visto que alguns estados não chegaram a ser visitados ou foram visitados poucas vezes.

Nos gráficos na Figura 4.1 temos as estimativas para os valores dos estados em z_c está fixo em 0, $V(x_c, y_c, 0)$. A ánalise desses gráficos nos permite entender que o algoritmo foi capaz de encontrar parâmetros responsáveis por marchas que percorrem maiores distâncias. Áreas com cores mais quentes correspondem a um conjunto de parâmetros que tem valores maiores, Uma melhor visualização dessas áreas é encontrada nos gráficos de contorno.

Na região mais vermelha dos gráficos da Figura 4.3 é onde se tem o maior valor, neste caso, $V(20.6, -3.6, 0)$. Através do gráfico 3D é visto que para esse conjunto de parâmetros função valor se encontra em um pico, ou seja, conjunto parâmetros não possuem valores tão altos. Isso ocorre porque, a fim de se restringir o espaço de busca, o passo dado aos parâmetros foi alto e os desempenhos das marchas são muito sensíveis a variações.

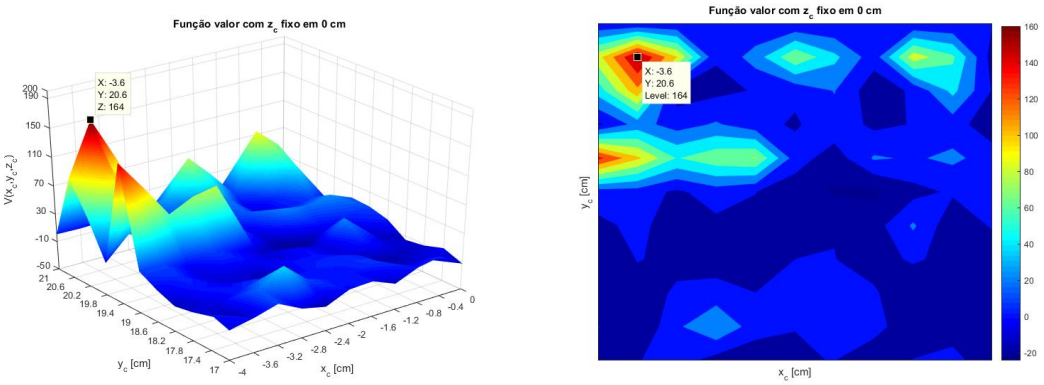


Figura 4.1: Estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0.

Na Figura 4.2 temos o caso em que z_c está fixo em $0,4$. Novamente reparamos que o há um pico na função valor. É neste pico o que se encontra o segundo maior valor da função valor, sendo próximo ao valor do conjunto $(20.6, -3.6, 0)$. Isso mostra que mais de um conjunto de parâmetros podem resultar em marchas que satisfazem a função objetivo. O menor valor encontrado é mostrado na Figura 4.3. Assim como é visto nos outros gráficos, parâmetros adjacentes não possuem valores tão baixos.

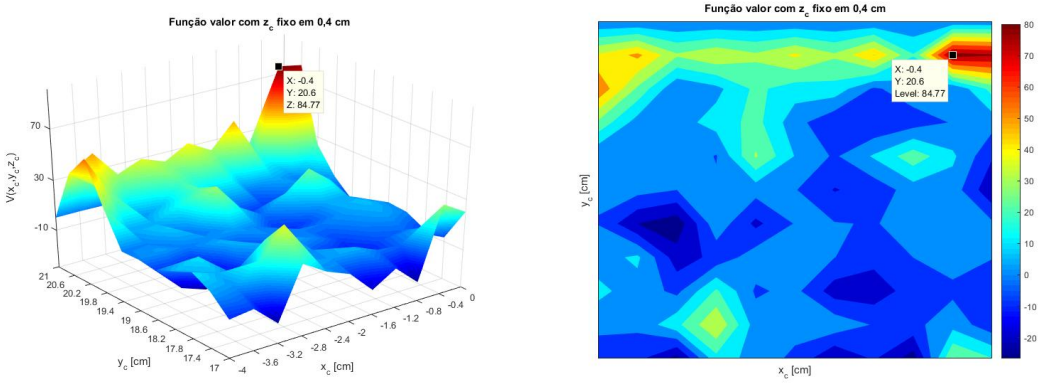


Figura 4.2: Estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0,4.

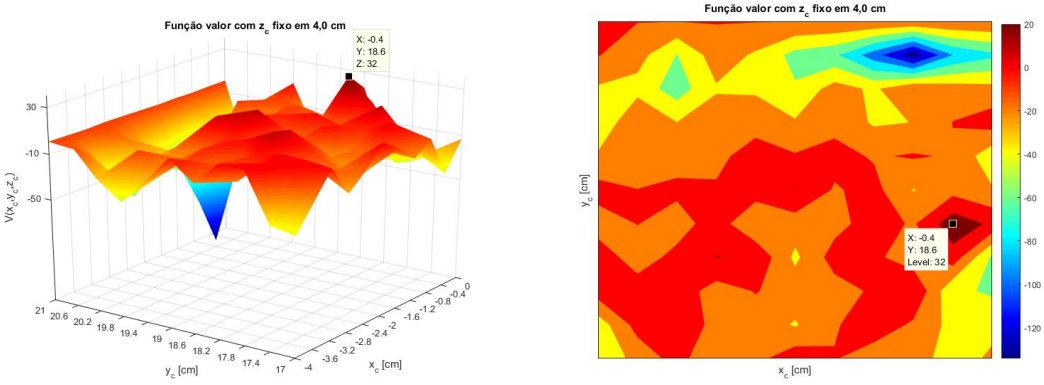


Figura 4.3: Estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 4

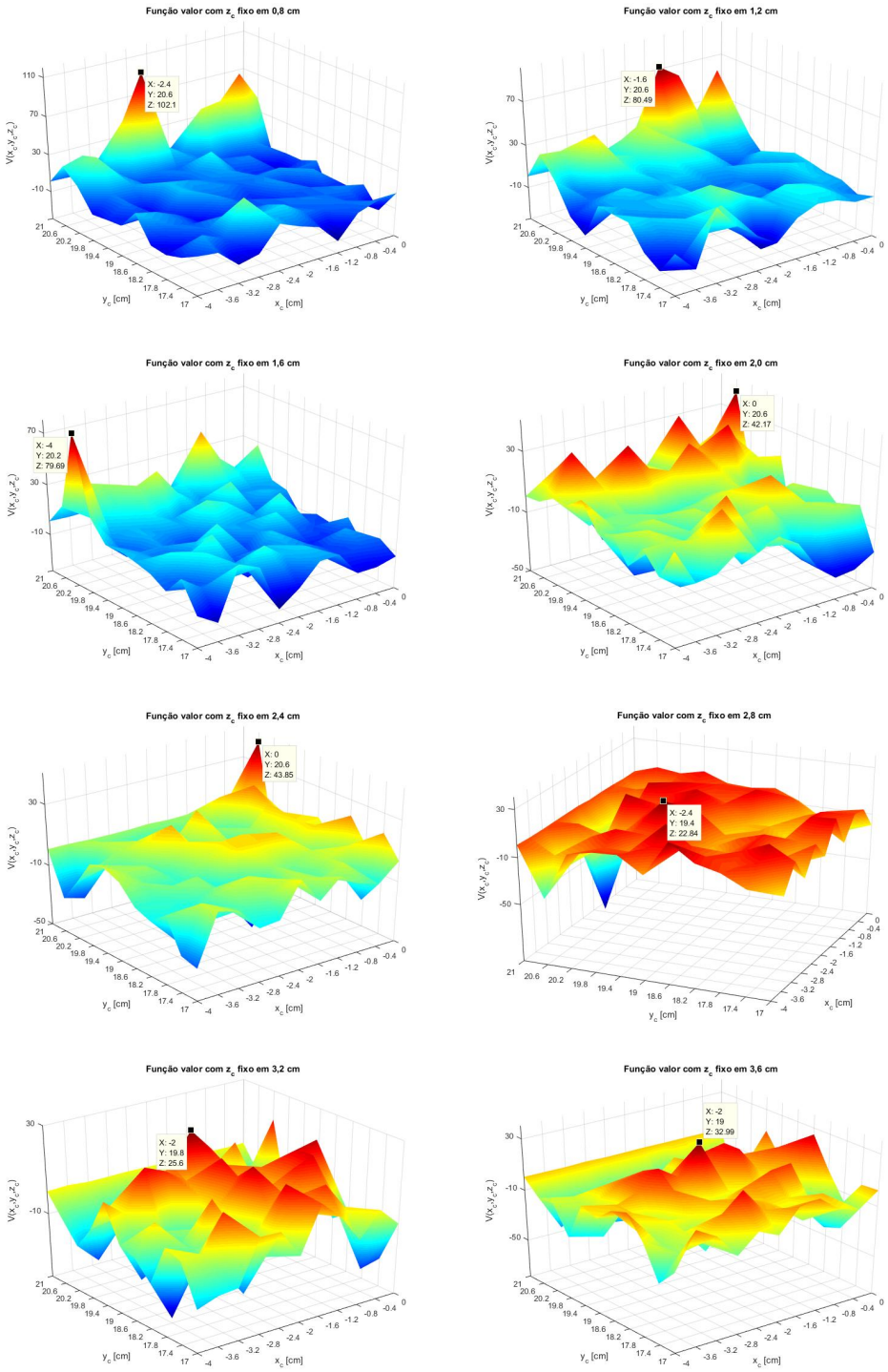


Figura 4.4: Estimativa dos valores dos estados em função de x_c e y_c, para os demais possíveis valores de z_c

Os gráficos quando z_c assume outros valores podem ser vistos na Figura 4.4. Ao se olhar todos os gráficos lado o lado é notado que conforme z_c aumenta menores são os valores máximos em cada gráfico. Isso mostra que o algoritmo aprendeu que se o objetivo é gerar marchas que maximizam a distância percorrida, conjuntos de parâmetros que tenham z_c menores são mais

efetivos. Todos os gráficos da função valor se encontram em tamanhos maiores no ANEXO I.

Para avaliar os resultados do algoritmo foram feitas simulações usando os parâmetros obtidos para a geração da marcha. Parâmetros que retornaram maiores valores foram avaliados. Os gráficos da Figuras 4.5 e 4.6 mostram os gráficos dos sensores de GPS da simulação. É possível perceber que em ambas os gráficos a posição y é crescente. As duas marchas andaram quase a mesma distância, 51 e 50 cm.

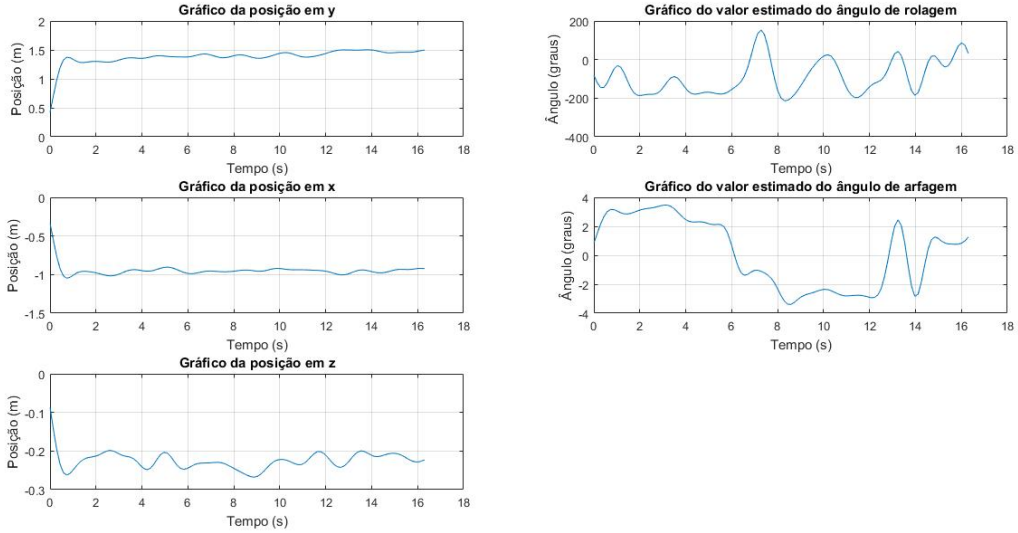


Figura 4.5: Valores dos sensores de GPS e dos ângulos de arfagem e rolagem para a marcha que obteve o maior valor.

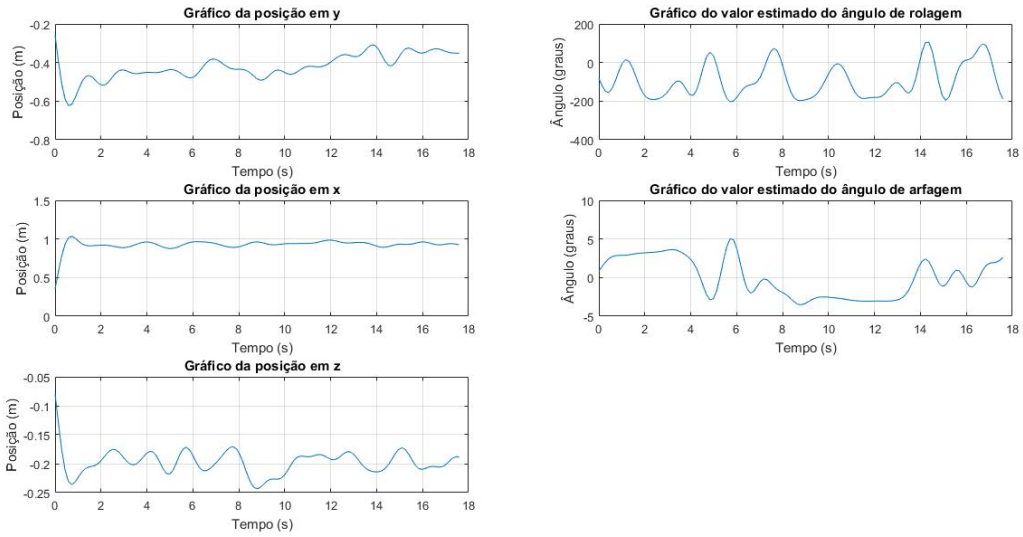


Figura 4.6: Valores dos sensores de GPS e dos ângulos de arfagem e rolagem para a marcha que obteve o segundo maior valor.

Os gráficos nas Figuras 4.7 e 4.8 mostram as posições dos motores nas marchas. Como a sequência de poses é fixa, não há uma grande diferença na trajetória de cada motor, mas fica

explícito que as transições nos motores 1, 4, 7 e 10, são feitas de modo mais abrupto na segunda marcha. Na Figura 4.9 há uma comparação entre as poses de transição das marchas.

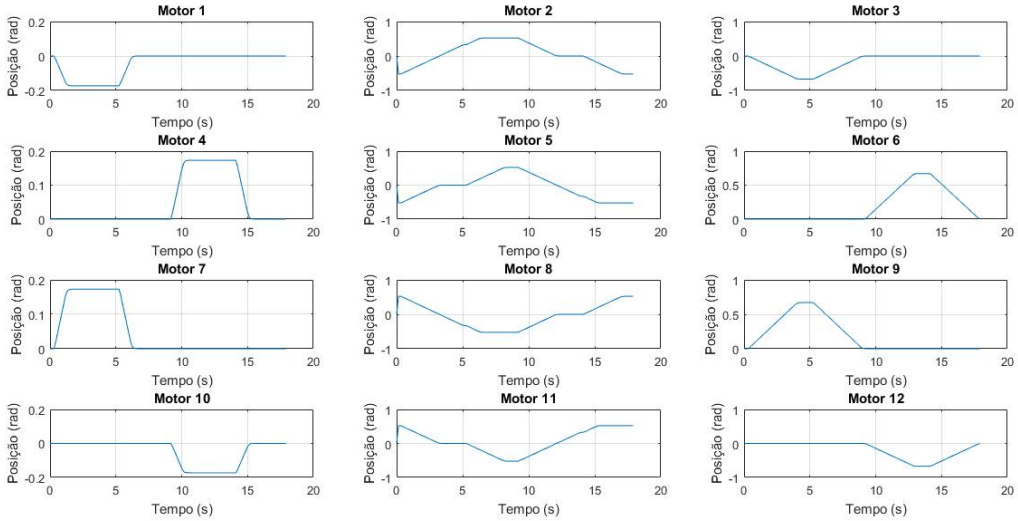


Figura 4.7: Posição dos motores para marcha que obteve o maior valor.

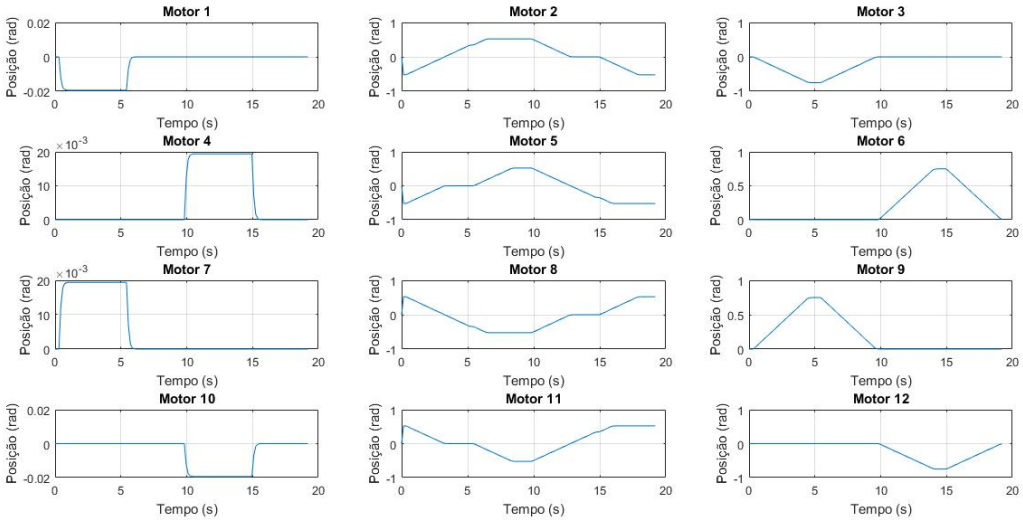


Figura 4.8: Posição dos motores para marcha que obteve o segundo maior valor.

Ainda avaliando o algoritmo, foi simulada a marcha advinda de estado que obteve menor valor. A Figura 4.10 mostra que assim como se esperava ao se gerar essa marcha, o robô acaba caindo ao dar o primeiro passo.

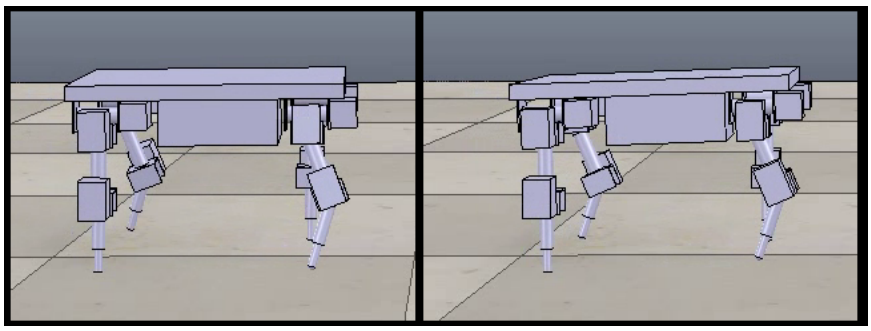


Figura 4.9: Comparação entre marchas que obtiveram o maior valor (esquerda) e o segundo maior valor. Distâncias percorridas de 50 e 51cm respectivamente.

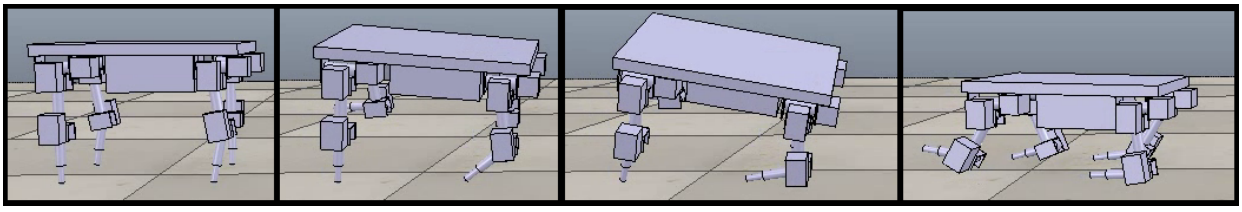


Figura 4.10: Marcha que obteve menor valor.

4.2 GRADIENTE DE POLÍTICA

4.2.1 Maximização da distância percorrida em y

Neste método, o conjunto de parâmetros não é mais tido como uma descrição do estado do quadrúpede. Para a aplicação do método de gradiente de política os parâmetros são considerados como a política. O algoritmo utilizado propõe uma maneira de estimar o gradiente da política para otimizar marchas conforme função de objetivo definida. Diferentemente da estratégia utilizada anteriormente, não há a necessidade de se restringir o espaço de busca, mas em compensação apenas obtemos as marchas ótimas locais.

Em primeiro momento, a função objetivo foi apenas maximizar a distância percorrida no eixo y. Para iniciar a execução algoritmo de gradiente de política, é necessária a escolha de uma política inicial, uma opção natural é os parâmetros obtidos por Dos Santos e Duarte [4], pois a marcha achada por eles obtém resultados satisfatórios e a convergência desse algoritmo depende de um bom chute inicial.

Ao utilizarmos algoritmo de gradiente de política foram obtidos resultados satisfatórios com poucas iterações. A distância máxima percorrida pela marcha inicial em simulação era por volta de 47cm, após cerca de 30 iterações, avaliando 5 marchas por iteração, a distância percorrida na marcha final chegou a 65cm.

A Figura 4.11 mostra os resultados de duas execuções do algoritmo sendo iniciando com os parâmetros obtido por Dos Santos e Duarte [4]. As iterações que possuem valor zero mostram que obtiveram como resultados parâmetros que não conseguiam gerar poses factíveis ou em que o robô cai. Na primeira tentativa (simulação 1) foram necessárias 20 iterações para se encontrar

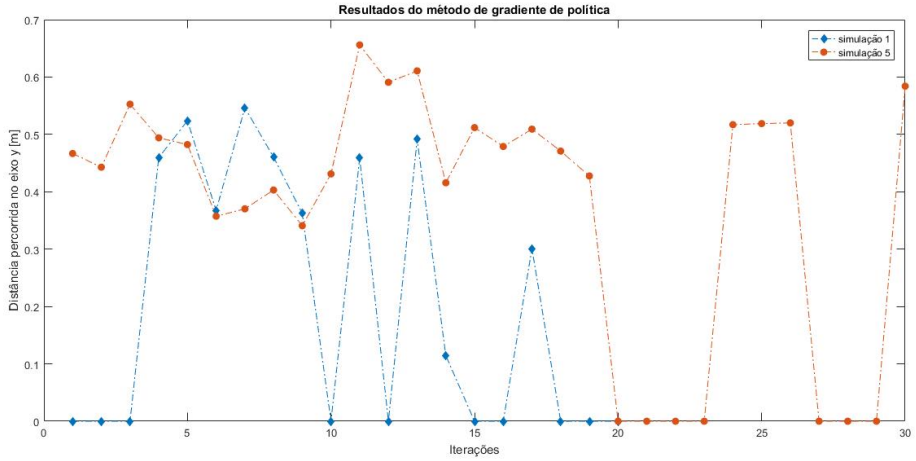


Figura 4.11: Distância percorrida ao final de cada iteração das simulações 1 e 5.

um valor máximo de 55cm. Na segunda tentativa (simulação 5) foram necessárias 30 iterações e foi encontrado um valor máximo de 65cm.

As variações no processo de aprendizado pode ser uma consequência de ruídos das medidas obtidas pelo *V-REP*. Outra fonte de variação pode estar no fato de um passo muito grande utilizado no processo de aprendizado para ajustar a política ao final de cada iteração. Esse passo serve para acelerar a taxa de aprendizado, mas se for muito grande, pode acabar gerando essa variação, uma vez que faz com que o agente vá a uma política ruim.

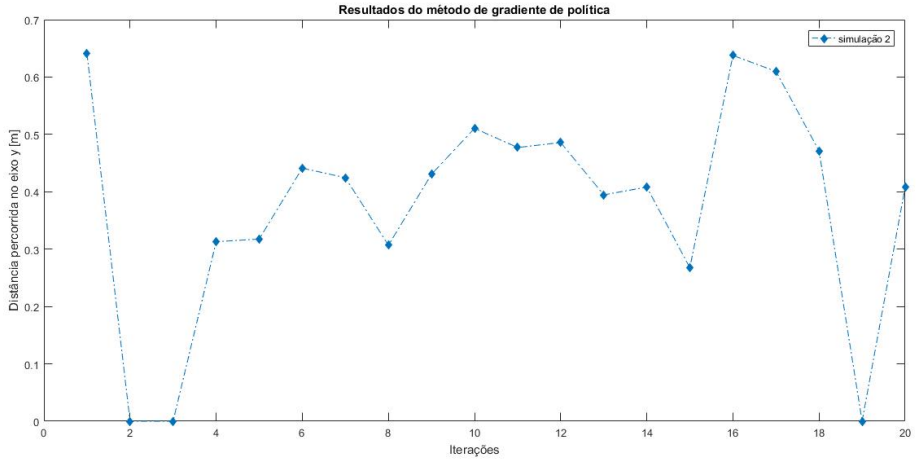


Figura 4.12: Distância percorrida ao final de cada iteração da simulação 2.

Na segunda simulação, resultados obtidos mostrados na Figura 4.12, se tentou verificar se era possível encontrar marchas que levassem a um deslocamento maior do que o obtido na simulação 1, para isso o algoritmo foi executado tomando como política inicial o resultado desta. Como pode ser visto, houve uma grande variação nos resultados de cada iteração mas ao final foi obtido um valor de 65cm, assim como na simulação 5.

Por fim os algoritmos foram executados usando parâmetros resultantes da simulação 1 como

parâmetros iniciais. Os resultados destas duas simulações (3 e 4) podem ser vistos nas Figuras 4.13 e 4.14. Novamente é visto uma grande variação nos valores de cada iteração. Ao final, os resultados obtidos foi uma distância total percorrida de 71cm para a simulação 3 e 72cm para simulação 4. Esses resultados corroboram em que ao se iniciar as iterações em políticas boas pode-se obter melhores resultados. A partir deles também podemos verificar que ao se iniciar em políticas diferentes podemos ter resultados diferentes pois nesse método apenas se consegue alcançar a política ótima local.

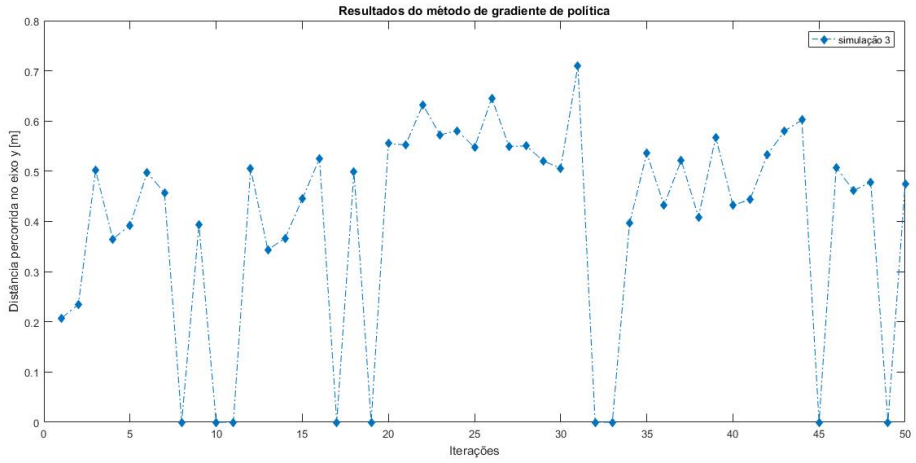


Figura 4.13: Distância percorrida ao final de cada iteração da simulação 3.

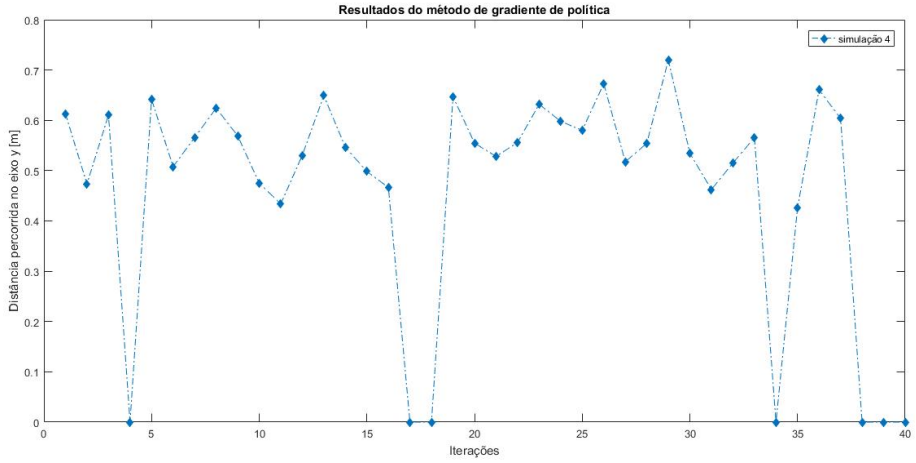


Figura 4.14: Distância percorrida ao final de cada iteração da simulação 4.

4.2.2 Minimização da variação em z

Nesta etapa, foi utilizada a como função objetivo a minimização da variação da altura do robô na tentativa de evitar marchas em que o robô cai. Como o algoritmo foi escrito de maneira a maximizar a nota da marcha, a nota foi dada como o inverso da diferença entre o maior e o menor valor válido lido pelo GPS no eixo de coordenadas z.

Novamente a execução do algoritmo na simulação 6 é iniciada com os parâmetros obtidos por Dos Santos e Duarte [4]. Pode ser visto na 4.15 que foram necessárias 40 iterações para se encontrar o valor máximo de $9,22$, o que consiste em uma variação de aproximadamente 11cm.

Para a simulação 6, foi alterado apenas um dos parâmetros afim de verificar se a convergência ocorreria no mesmo número de iterações. Na simulação 7, foi obtido um resultado de $9,11$, bem próximo do que foi achado na simulação anterior, mas em apenas 30 iterações. Isso ocorre pois de acordo com o que foi verificado no método de diferença temporal, marchas melhores são geradas por conjunto de parâmetros que possuam um menor valor de x_c, parâmetro que foi alterado na política inicial. Outro fator que pode ter acelerado é que não são vistas iterações que não conseguiram gerar marchas. O resultado dessa simulação pode ser visto na Figura 4.16.

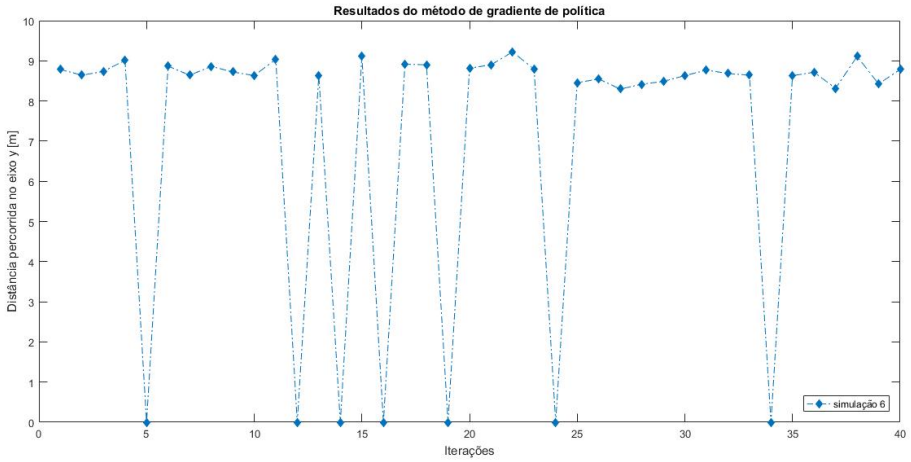


Figura 4.15: Inverso da variação em z ao final de cada iteração da simulação 6.

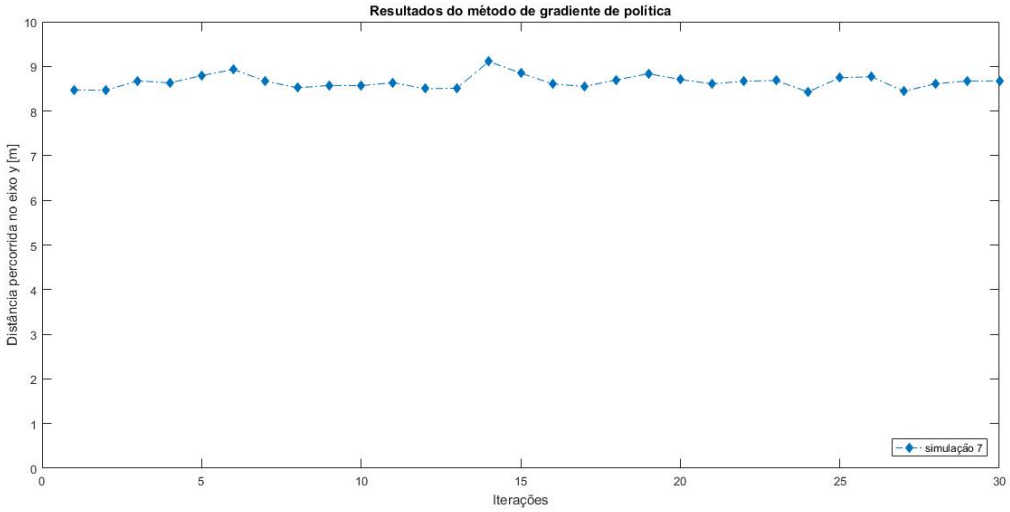


Figura 4.16: Inverso da variação em z ao final de cada iteração da simulação 7.

4.2.3 Minimização da variação em x

Para a minimização da variação em x, foi utilizada a mesma estratégia para o caso em z. As simulações utilizaram as mesmas políticas inicias do caso anterior. Os resultados podem ser vistos nas Figuras 4.17 e 4.18. Assim como ocorreu nas simulações em z, ao se alterar apenas um parâmetro, os resultados obtidos foram muito parecidos entre si. Na simulação 8 os parâmetros finais tiveram uma nota de $4,98$ e na simulação 9 essa nota foi de $4,94$ mas, a convergência se mostrou mais rápida após alteração do parâmetro. A simulação 8 foi completa ao final de 40 iterações e a 9 foram necessárias 30.

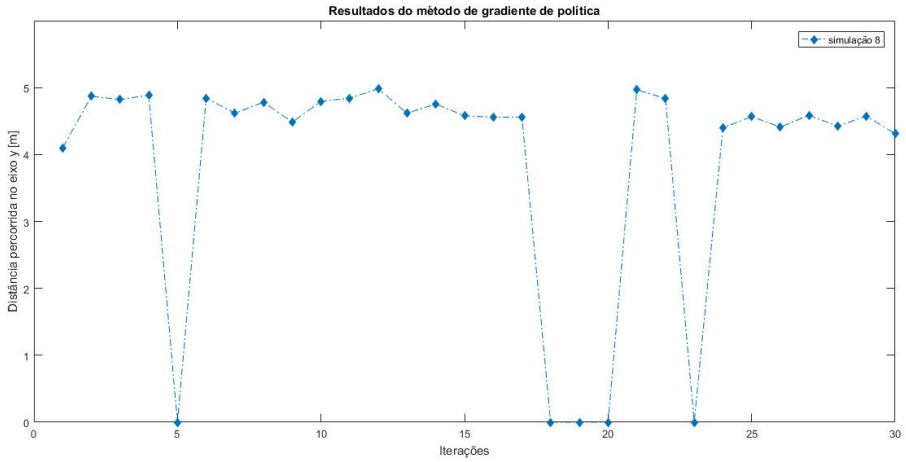


Figura 4.17: Inverso da variação em y ao final de cada iteração da simulação 8.

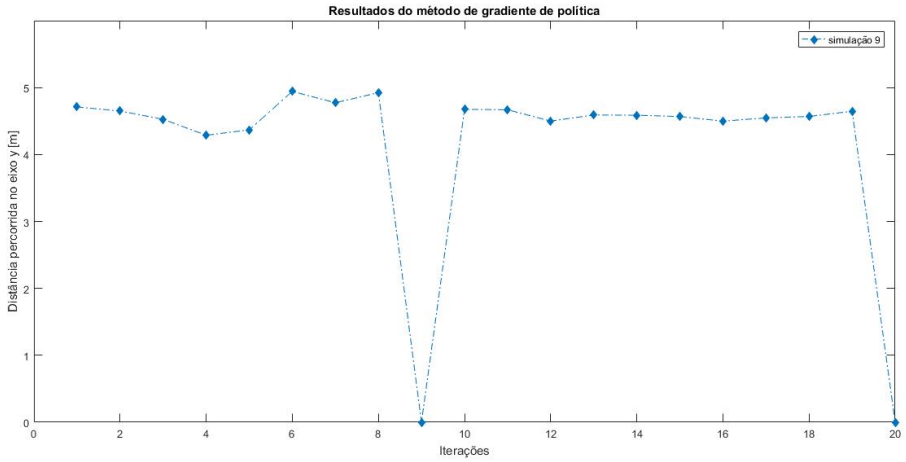


Figura 4.18: Inverso da variação em y ao final de cada iteração da simulação 9.

4.2.4 Média ponderada dos *scores* em x, y e z

Por fim, a busca por parâmetros ótimos tenta otimizar todas as funções objetivos de uma vez. Assim, se busca parâmetros que gerem uma marcha que além percorrer uma maior distância, não

tenha muita variação nos eixos x e z. Para isso, usou-se uma média ponderada de todos os $scores$. Foi dado peso 3 para nota da distância percorrida, peso 2 para o inverso da variação em x e peso 1 para o inverso da variação em z.

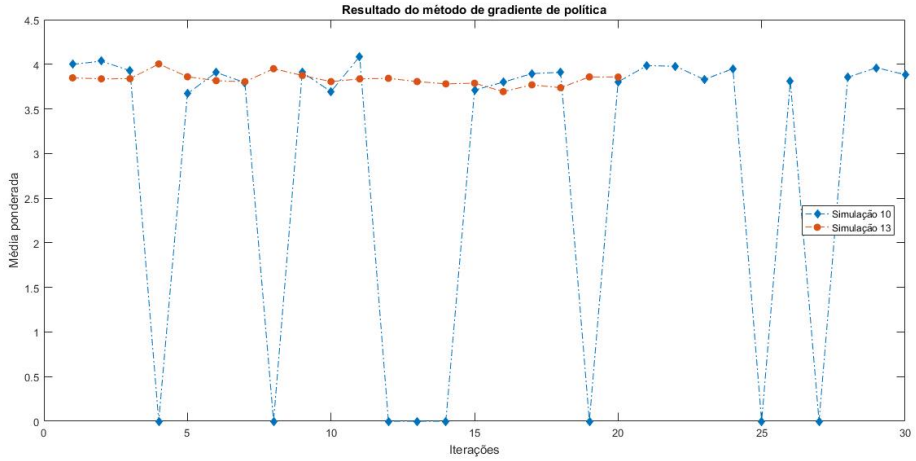


Figura 4.19: Resultado da média ponderada ao final de cada iteração das simulações 10 e 13.

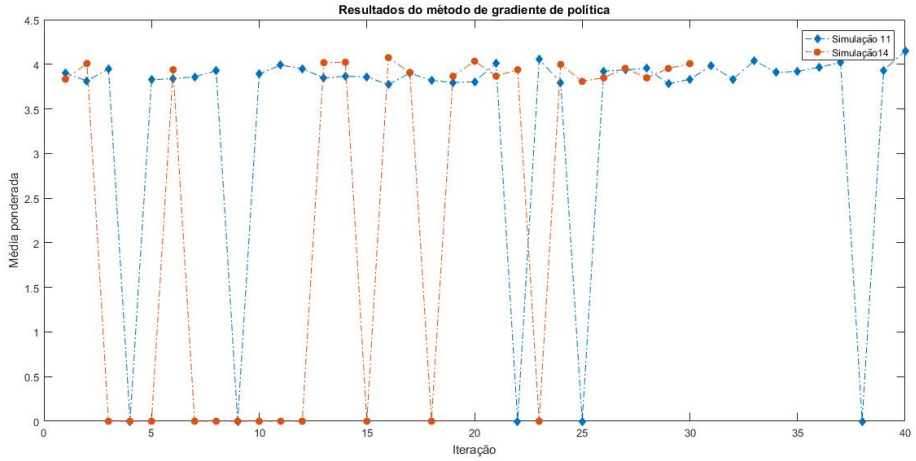


Figura 4.20: Resultado da média ponderada ao final de cada iteração das simulações 11 e 14.

As simulação 10 e 13, Figura 4.19, se iniciam com a mesma política que as simulações 1 e 6. Com os parâmetros resultantes da simulação 10 se iniciou a simulação 12, Figura 4.21. Finalmente para a simulações 11 e 14, Figura 4.20, apenas um parâmetro é mudado em relação a politica inicial da simulação 10. Nas simulações 10 e 13 foram obtidas notas $4,08$ e $4,15$, sendo que a segunda convergiu mais rapidamente. Na simulação 12 a nota final foi de $4,07$ e para as simulações 11 e 14, foram obtidos $4,00$ e $4,07$. Todas as marchas obtiveram resultados parecidos e, excluindo as marchas com nota zero, foi observada pouco variação nos valores a cada nova iteração.

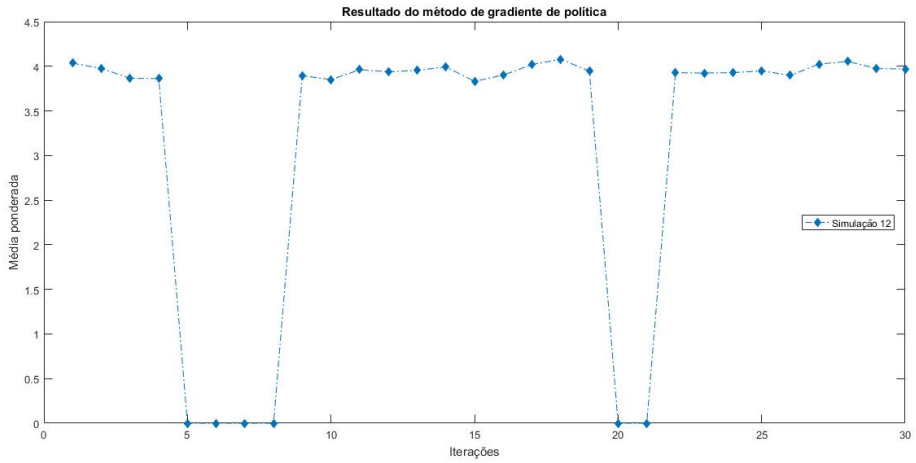


Figura 4.21: Resultado da média ponderada ao final de cada iteração das simulação 12.

4.3 PLATAFORMA REAL

4.3.1 Aprendizado na plataforma

Foi feita uma tentativa de aplicação do método de gradiente de politica diretamente na plataforma real. Devido a problemas no algoritmo implementado e na estrutura mecânica não foi foram obtidos resultados favoráveis.

Para o processo de aprendizado na plataforma real é necessário uma constante supervisão ao robô, isso porque a inicialização o programa *Python* responsável pelo aprendizado é feita de forma manual a cada iteração. Em uma iteração é definido, o número de marchas que será avaliada. Após a execução cada marcha é necessário que seja dada uma nota a ela, neste caso foi utilizada a distância total percorrida no eixo y, e em seguida realocar o robô no ponto de partida para que nova marcha seja executada. Essa interação constante com o robô acaba tomando muito tempo e o processo de aprendizado muito demorado.

Outro problema encontrado ao e tentar a implementação está no fato de serem utilizadas baterias de apenas 12Volts para a alimentação do robô, essa é a tensão mínima para os motores do robô. Ao terem sido feitas por volta de 7 iterações, onde se avaliava 5 marchas, a tensão nessa bateria caiu para 11,1Volts, uma tensão insuficiente para que os motores respondam bem. A partir desse momento o robô não conseguia terminar de rodar a maioria das marchas a serem avaliadas.

Além de tudo problemas na estrutura do robô foram encontradas. Ao serem feitos alguns movimentos o fio que faz a alimentação de uma das patas é tensionado e acaba por desligar a pata, quando isso ocorre é necessário reiniciar a iteração. Problemas nos parafusos das patas também foram encontrados. Após serem rodas um numero considerável de marchas o parafuso da junta q_3 de uma das patas começa a se soltar e, consequentemente, a influenciar negativamente no desempenho das marchas.

Quando tentamos aplicar o algoritmo na plataforma poucas as marchas geradas para serem

avaliadas na iteração conseguiam obter resultados satisfatórios, o que impede uma rápida convergência dos parâmetros a valores ideais. Isso pode ser devido que na implementação do método de gradiente de política na plataforma os valores de ϵ_n para gerar novas marchas são os mesmos mas o valor de de η é maior na tentativa de se ter um aprendizado mais rápido. A alteração tanto nos valores de ϵ_n como de η pode ser uma boa solução para esse problema.

4.3.2 Avaliação das marchas adquiridas nas simulações

A última etapa para a avaliação da método de gradiente de política consiste em aplicar as marchas obtidas em simulação na plataforma real e verificar se nela também ouve um melhora na marcha de acordo com a função objetivo para a qual a ela foi gerada. Como muitos resultados obtidos nas simulações são parecidos apenas alguns foram selecionados para serem avaliado na plataforma.

Na Tabela 4.1, tem-se a relação das simulações avaliadas, os parâmetros que cada simulação gerou e os resultados obtidos utilizando a plataforma no local de treino descrito na seção 3.4. As marchas geradas podem ser vistas em video[1].

Simulação	Parâmetros					y [cm]	x [cm]
	$angle - back$	$angle - front$	x_c	y_c	z_c		
0	27.92	32.06	20.68	-3.38	0.43	61	1,5
3	24.41	36.33	19.93	-4.17	-0.67	68	7,5
5	27.18	33.85	21.43	-2.46	-0.01	78	6
6	28.44	32.70	21.08	-3.60	2.65	52	-12
8	28.37	33.80	21.62	-1.92	3.05	16	-14
10	27.14	32.06	21.55	-2.20	2.17	0	0
12	25.83	32.30	21.20	-4.50	-0.31	80	5

Tabela 4.1: Resultados das marchas aplicadas na plataforma real.

A marcha 0 é a marcha resultante do trabalho de Dos Santos e Duarte[4] que foi aplicada na plataforma para efeito de comparação, já que quase todas as marchas foram adquiridas usando ela como política inicial.

A marcha 10 tem valores medidos igual a zero pois ao tentar aplicar essa marcha na plataforma ela acabou ficando de cabeça para baixo. Essa marcha foi resultado de uma simulação em que tentava maximizar a média ponderada dos scores. Visto que se utilizando de apenas o resultado da média não sabemos a que proporção cada fator está sendo melhorado, pode ser tanto a distância percorrida ou a variação em z.

A marcha 8 também não obteve resultados condizentes com a simulação, uma vez que a variação em x chegou a 16cm e ele foi obtida na tentativa de minimizar esse valor. A marcha 6 foi obtida na tentativa de minimizar a variação da altura do robô, não tendo qualquer compromisso com a distância percorrida e variação em x, deste modo, os valores obtidos em x e y não foram

[1] https://youtu.be/99tqD5Nkres

discrepantes uma vez que não se esperavam bom valores nessa marcha.

Discrepâncias encontradas entre os resultados da simulação e da aplicação real podem ser explicada pelo fato de que a simulação não é um retrato fiel da realidade e que os parâmetros bons para o modelo no *V-REP* pode não ser o melhor para a aplicação real. Os resultados para distância percorrida pela marcha 3 e 5 são uma prova disso, nas simulações a marcha 3 percorre uma distância maior que a marcha 5 mas na aplicação esse cenário se inverte.

As marchas 3, 5 e 12 foram as melhores e com resultados condizentes com a simulação. A marcha 12 especialmente, nela verificamos que quando o robô vai se levantar da posição mais agachada ele se joga para frente, diferentemente da maioria das outras marchas obtidas na plataforma, em que há um pequeno impulso para trás ao fazer este mesmo movimento. Essa pose pode ser vista nas Figuras 4.22, 4.25 e 4.28.

Nas Figuras 4.24, 4.26 e 4.29 é possível comparar os ângulos de rolagem e arfagem e nas Figuras 4.23, 4.26 e 4.29 é possível ver a posição que cada motor atinge durante cada marcha. Pode se verificar comparando as imagens que as marchas 5 e 12 são bem parecidas, apesar de serem derivadas de funções objetivos diferentes. Isso se deve ao fato do peso da média ponderada ser maior para o valor da distância percorrida em y.

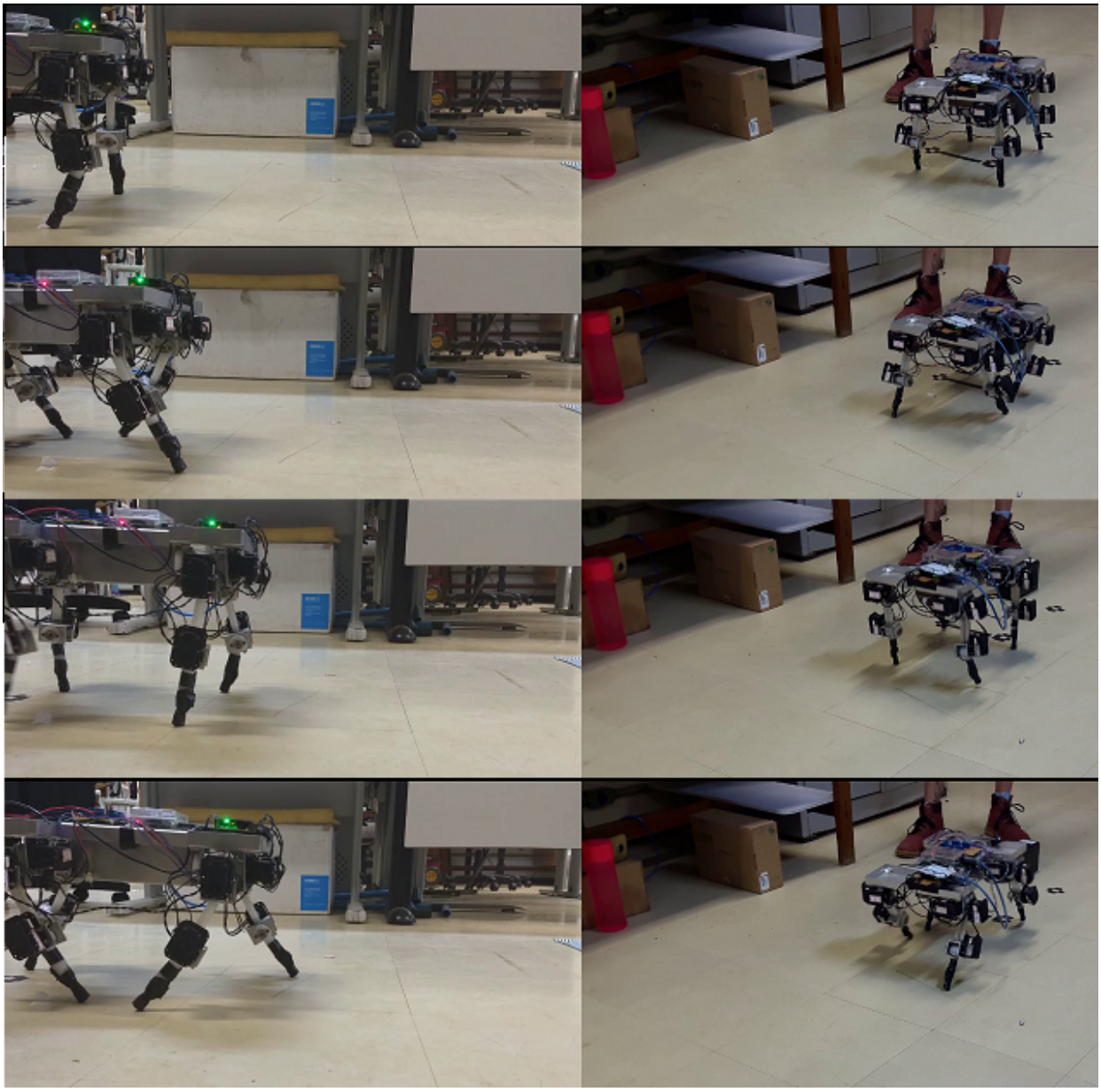


Figura 4.22: Sequência de imagens da execução da marcha obtida por Dos Santos e Duarte [4]. <https://youtu.be/99tqD5Nkres>

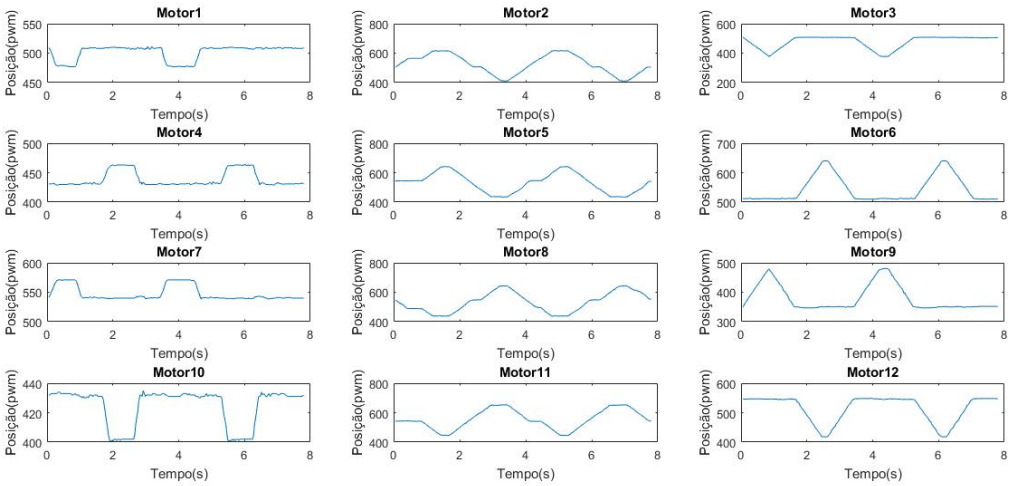


Figura 4.23: Posição dos motores para a marcha obtida por Dos Santos e Duarte [4].

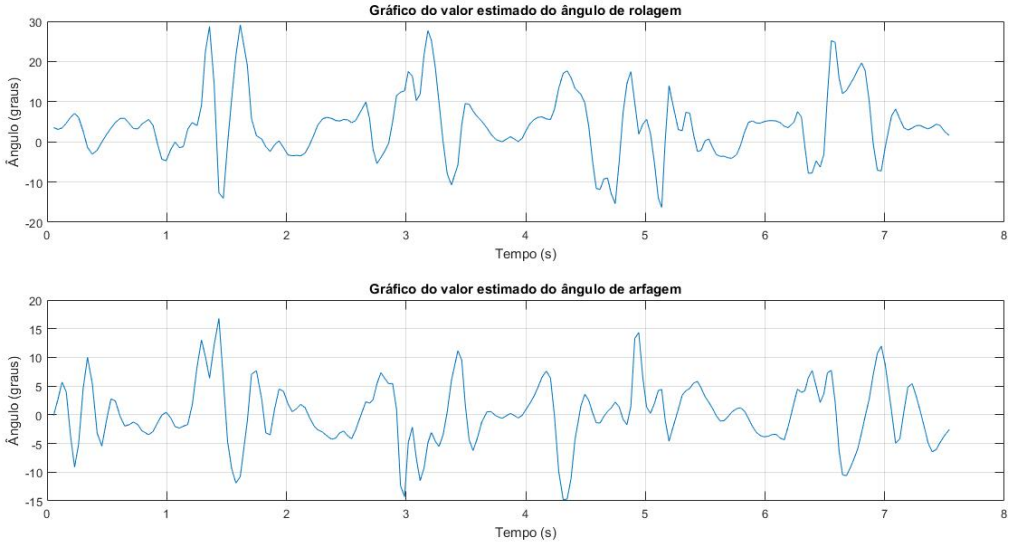


Figura 4.24: Valores dos ângulos de rolagem e arfagem ao longo do tempo para a marcha obtida por Dos Santos e Duarte [4].

Figura 4.25: Sequência de imagens da execução da marcha obtida na simulação 5. <https://youtu.be/s9zqs6otwWk>

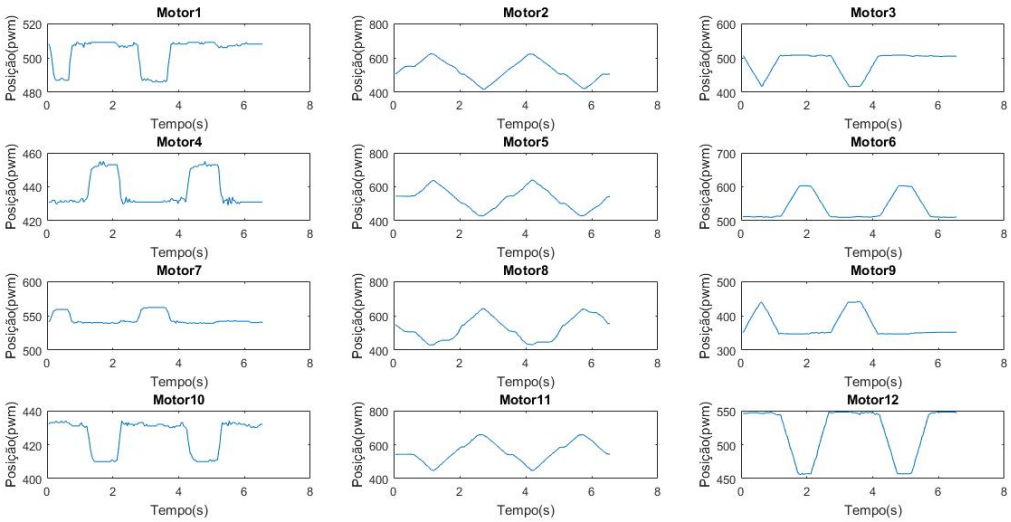


Figura 4.26: Posição dos motores para a marcha obtida na simulação 5.

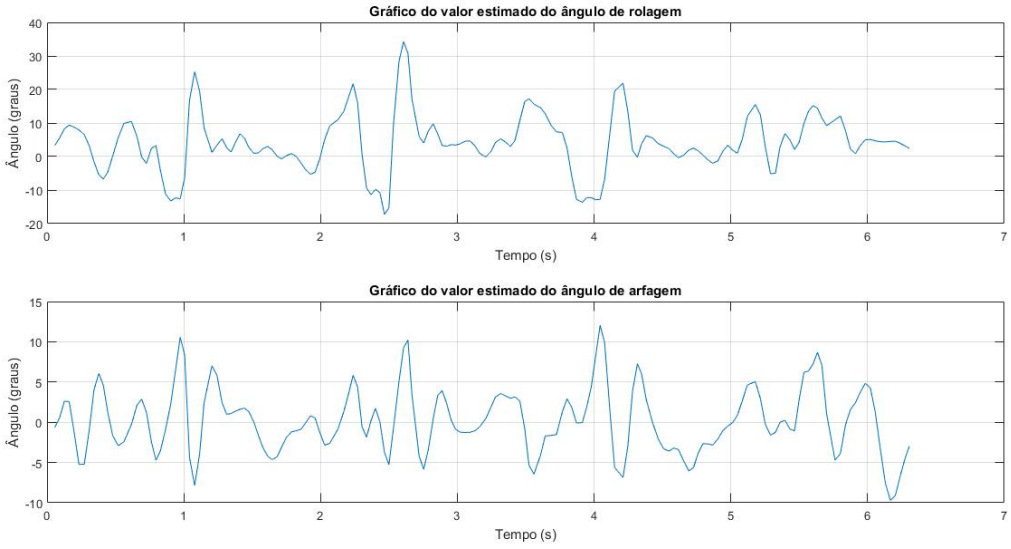


Figura 4.27: Valores dos ângulos de rolagem e arfagem ao longo do tempo para a marcha obtida na simulação 5.

Figura 4.28: Sequência de imagens da execução da marcha obtida na simulação 12. <https://youtu.be/qOC-ekQdmzg>

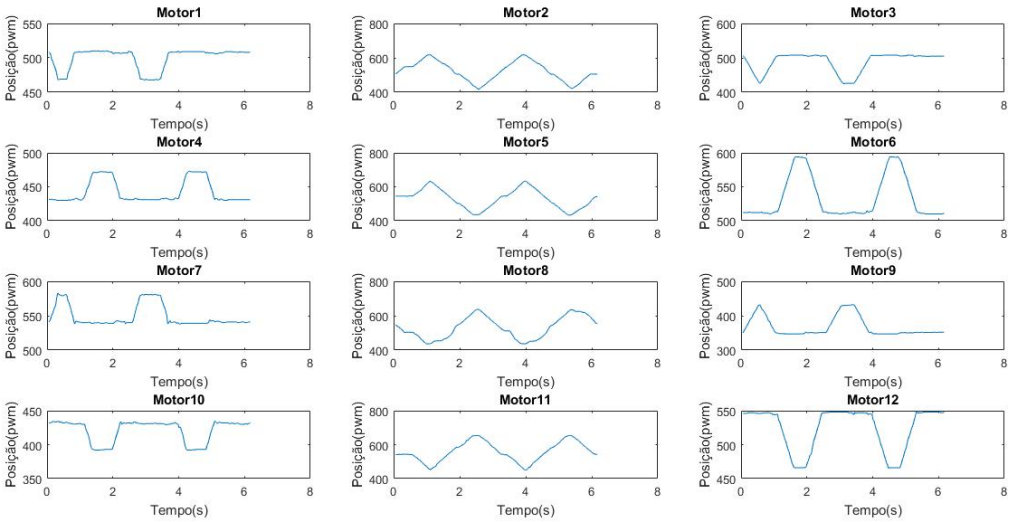


Figura 4.29: Posição dos motores para a marcha obtida na simulação 12.

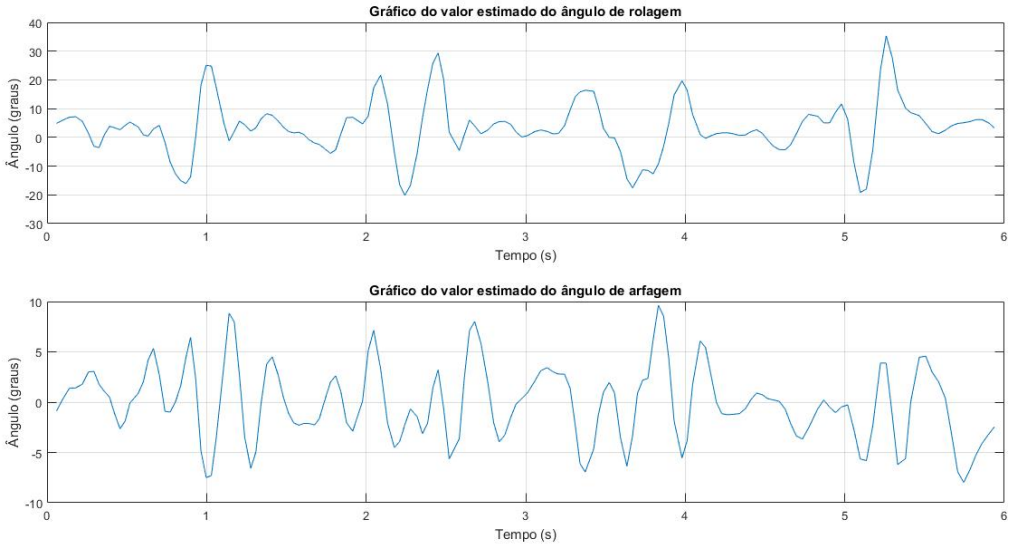


Figura 4.30: Valores dos ângulos de rolagem e arfagem ao longo do tempo para a marcha obtida na simulação 12.

5 CONCLUSÕES

Este trabalho mostrou que a estrategia de aprendizado de reforço que foi capaz de otimizar os valores dos parâmetros do gerador de marcha de um robô quadrúpede em um ambiente simulado. Para a otimização foram utilizados dois algoritmos diferentes, os quais cada mostrou vantagens e desvantagens.

O primeiro método aplicado foi o de diferença temporal, onde os parâmetros da marcha são considerados o estado do robô. Neste método, o valor dos estados são aproximados conforme os estados são visitados e uma implementação com parâmetros contínuos seria inviável. Apenas três parâmetros foram escolhidos para serem aprendidos e esses foram discretizados em passos de 0,4.

Com as restrições impostas, foi visto que para se obter uma estimativa da função valor eram necessárias muitas horas de simulação. Neste trabalho cada parâmetro a ser aprendido poderia assumir 11 valores, totalizando 1331 possíveis estados, foram feitas 6000 simulações, totalizando mais de 15h, e mesmo assim nem todos os estados foram visitados ou foram visitados pouca vezes. Os parâmetros foram discretizados a passos muito grandes e, sendo a marcha muito sensível a estes parâmetros, foi visto que outras algoritmos de aprendizado poderiam ser mais eficientes. Mas mesmo assim, com este algoritmo foi possível encontrar relações entre os parâmetros e maximização da distância percorrida.

O segundo método utilizado foi o de gradiente de política. Neste algoritmo os parâmetros não são mais tomados como estados e sim uma política que o robô executa. O objetivo, então, consiste em achar a política ótima. Para isso o algoritmo utilizado propõe uma forma eficiente de se estimar o gradiente de política, em outras palavras, encontrar a direção em que cada parâmetro deve ser ajustado para que a função objetivo seja maximizada. Com a utilização deste algoritmo o tempo de simulação diminuiu consideravelmente, passando a durar no máximo uma hora e meia. Com poucas iterações foi possível a obtenção de resultados bem melhores. A marcha usada para inciar o algoritmo conseguia percorrer uma distância de 61cm e a marcha final encontrada nesse trabalho chegou a andar 80cm, um aumento de 30%.

A desvantagem no uso deste segundo método está no fato de que, diferentemente do método utilizado anteriormente, a busca pelos melhores parâmetros converge apenas para um ótimo local. Isso é comprovado ao se iniciar o algoritmo escolhendo política diferentes, ao final os parâmetros obtidos são diferentes. Apesar disso os resultados são parecidos o que sugere que foi atingido o máximo da capacidade para esse modelo de marcha.

As simulações foram de suma importância para a implementação do aprendizado. Não seria possível a aplicação do primeiro método diretamente na plataforma pois ele requer um número muito grande de marchas a serem avaliadas. O segundo método é de fácil aplicação mas a tentativa de utilizá-lo diretamente na plataforma encontrou muitos problemas, mas resultados obtidos por

esse método através da simulação e, então, aplicados na plataforma se mostraram satisfatórios.

Foram vários os problemas encontrados para a implementação diretamente na plataforma. A utilização de uma bateria de apenas 12Volts, tesão mínima para funcionamento dos motores, foi um dos problemas um vez que se fazer por volta a sétima iteração a bateria estava gerando apenas 11,1Volts e o robô não conseguia mais terminar as marchas. No trabalho de Porphirio e Santana [16] foi concluído que para uma melhor execução dos movimentos do robô seriam necessária a utilização de uma bateria de 18Volts, para uma completa autonomia o ideal seriam bateria de lítio ligadas em paralelo. Melhorias na estrutura mecânica do robô também seriam necessárias uma vez que era necessário apertar parafusos da pata frequentemente. Soluções paliativas para problemas na estrutura foram feitas em trabalhos anteriores mas elas deixaram de funcionar a longo prazo. O ideal seria uma reestruturação das patas do robô.

De forma geral, o trabalho realizado representa um progresso satisfatório, uma vez que deu continuidade ao trabalho de Dos Santos e Duarte [4] propondo uma nova maneira de otimizar os parâmetros da marcha modelada por eles.

Para dar continuidade a este trabalho são sugeridos alguns temas. Melhorias mecânicas na plataforma devem ser feitas para que sejam obtidas melhores marchas uma vez que as patas do robô tem um problema de falta de atrito com o chão, fazendo o robô andar de modo arrastado. Outro trabalho que poderia ser desenvolvido seria a criação de um sistema de treinamento automático. Nele o robô poderia realizar medidas automáticas e se colocar de volta na posição inicial, diminuindo ou excluindo a necessidade de constante supervisão. Sugere-se a implementando um sistema de GPS em adição ao sistema de sensoriamento já existente. Outro ponto que pode ser explorado em trabalhos futuros e a modelagem de novas poses e assim criar marchas mais fluidas.

REFERÊNCIAS BIBLIOGRÁFICAS

1 SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. 2. ed. Cambridge: The MIT Press, 2018. (Adaptive Computation and Machine Learning). ISBN 9780262193986.

2 ARRUDA, R. L. S. *Uma arquitetura híbrida aplicada em problemas de aprendizagem por reforço*. Dissertação (Mestrado) — Universidade Estadual de Campinas, Campinas, 2012.

3 SOUTO, R. F. *Modelagem cinemática de um robô quadrúpede e geração de seus movimentos usando filtragem estocástica*. Monografia (Graduação) — Universidade de Brasília, Brasília, Julho 2007.

4 SANTOS, P. M. F. dos; DUARTE, V. A. *Aprendizado de marcha para um robô quadrúpede utilizando algoritmo genético multiobjetivo*. Monografia (Graduação) — Universidade de Brasília, Brasília, Julho 2018.

5 KOHL, N.; STONE, P. Policy gradient reinforcement learning for fast quadrupedal locomotion. In: IEEE. *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. New Orleans, 2004. v. 3, p. 2619–2624.

6 ROBOTICS. *ROBOTIS E-Manual*. Disponível em: <http://emanual.robotis.com/docs/en/software/rplus1/dynamixel_wizard/>. Acesso em: 30 nov. 2018.

7 MACHADO, J. T.; SILVA, M. F. An overview of legged robots. In: MME PRESS ANKARA, TURKEY. *International symposium on mathematical methods in engineering*. [S.l.], 2006.

8 SEMINI, C.; TSAGARAKIS, N. G.; GUGLIELMINO, E.; FOCCHI, M.; CANNELLA, F.; CALDWELL, D. G. Design of hyq–a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, SAGE Publications Sage UK: London, England, v. 225, n. 6, p. 831–849, 2011.

9 COTTA, G. H.; RAULINO NETO, L. *Realização de uma plataforma para estudo de robótica comportamental baseada em quadrúpedes*. Monografia (Graduação) — Universidade de Brasília, Brasília, 2006.

10 CALMON, A. du P.; PINHEIRO, N. C.; FERREIRA, R. U. *Desenvolvimento de um robô-cachorro comportamental: percepção e modelagem comportamental*. Monografia (Graduação) — Universidade de Brasília, Brasília, 2006.

11 BATISTA, G. F.; CARDOSO, I. F. *Adequação de um sistema de locomoção de um robô quadrúpede para avaliação de algoritmos de aprendizagem.* Monografia (Graduação) — Universidade de Brasília, Brasília, 2007.

12 DE NOVAIS, N. A.; TOSCANO, R. A. *Estudo de locomoção de uma plataforma quadrúpede utilizando sensoriamento inercial e geração de padrões de movimento.* Monografia (Graduação) — Universidade de Brasília, Brasília, 2007.

13 RAMOS, E. *Desenvolvimento da plataforma quadrúpede geração de software e eletrônica.* Monografia (Graduação) — Universidade de Brasília, Brasília, 2008.

14 PAIVA, R. C. *Osciladores neurais para comando de marcha de um robô quadrúpede e robô humanoide.* Monografia (Graduação) — Universidade de Brasília, Brasília, 2012.

15 SANTOS, J. H. de S. *PLATAFORMA QUADRUPEDE: Uma nova estrutura para robô quadrúpede do LARA*. Monografia (Graduação) — Universidade de Brasília, Brasília, Julho 2016.

16 PORPHIRIO, C. de F.; SANTANA, P. H. M. *Geração de marchas para a plataforma quadrúpede utilizando algoritmo genético*. Monografia (Graduação) — Universidade de Brasília, Brasília, Junho 2017.

17 FLORIANO, B. R. de O. *Desenvolvimento e interação de um sistema de controle de equilíbrio para um robô quadrúpede.* Monografia (Graduação) — Universidade de Brasília, Brasília, 2017.

18 SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, IBM, v. 3, n. 3, p. 210–229, 1959.

19 SHABANI, J.; HONARVAR, A.; MOROVATI, M.; ABDOLLAHIAN, G.; MAHMOODI, M. Reinforcement learning in soccer simulation. In: *Workshop on Adaptability in Multi-Agent Systems and The First RoboCup Australian Open 2003 (AORC-2003)*. Sidney: Commonwealth Scientific and Industrial Research Organization, 2003.

20 PUTERMAN, M. L. *Markov decision processes: discrete stochastic dynamic programming*. 1. ed. New Jersey: Wiley-Interscience, 1994. (Wiley Series in Probability and Statistics). ISBN 0471619779,9780471619772.

21 PELLEGRINI, J.; WAINER, J. Processos de decisão de markov: um tutorial. *Revista de Informática Teórica e Aplicada*, XIV, n. 2, 2007.

22 GRUNITZKI, R. Um estudo sobre aprendizado por reforço multiagente e reward shaping. Trabalho em curso de pós-graduação. 2014. Disponível em: <https://www.maxwell.vrac.puc-rio.br/19637/19637_4.PDF>.

23 MARTINS, P. S. *Aprendizado de Máquina para Otimização de Parâmetros em Sistemas Baseados em Conhecimento.* Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Florianópolis, Junho 2003.

24 AZZOLINI, A. G. *Aprendizado por reforço em modelos probabilísticos de redes imunológicas para robótica autônoma*. Dissertação (Mestrado) — Universidade Estadual de Campinas, Campinas, 2011.

25 SILVA, R. R. da. *Aprendizado por reforço relacional para o controle de robôs sociáveis*. Dissertação (Mestrado) — Universidade de São Paulo, São Carlos, Janeiro 2009.

26 SENCIANES, A. E.-F. *Gradient-based reinforcement learning techniques for underwater robotics behavior learning*. Tese (Doutorado) — University of Girona, December 2011.

27 INTERLINK ELETRONICS. *Fsr 400 series data sheet*.

28 ERTHAL, J. L. et al. *Estudo de metodos para a solução da cinematica inversa de robos industriais para implementação computacional*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Florianópolis, Outubro 1992.

29 KELTON, W. D. *Simulation with ARENA*. New York: McGraw-hill, 2002.

30 REDEL, R.; HOUNSELL, M. d. S. Implementação de simuladores de robôs com o uso da tecnologia de realidade virtual. In: UNIVALI. *IV Congresso Brasileiro de Computação. IV CBCOMP*. Itajaí–SC, 2004. v. 1, p. 398–401.

31 HOSS, A.; HOUNSELL, M.; LEAL, A. B. Virbot4u: Um simulador de robô usando x3d. *I Simpósio de Computação Aplicada*, p. 1–15, 2009.

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

33 COPPELIA ROBOTICS. *Virtual Robot Experimentation Platform (V-REP) User Manual*.

34 SILVA, I. J.; PERICO, D. H.; COSTA, A. H.; BIANCHI, R. A. Using reinforcement learning to optimize gait generation parameters of a humanoid robot. *XIII Simpósio Brasileiro de Automação Inteligente*, 2017.

APÊNDICES

I. GRÁFICOS DO MÉTODO DE DIFERENÇA TEMPORAL

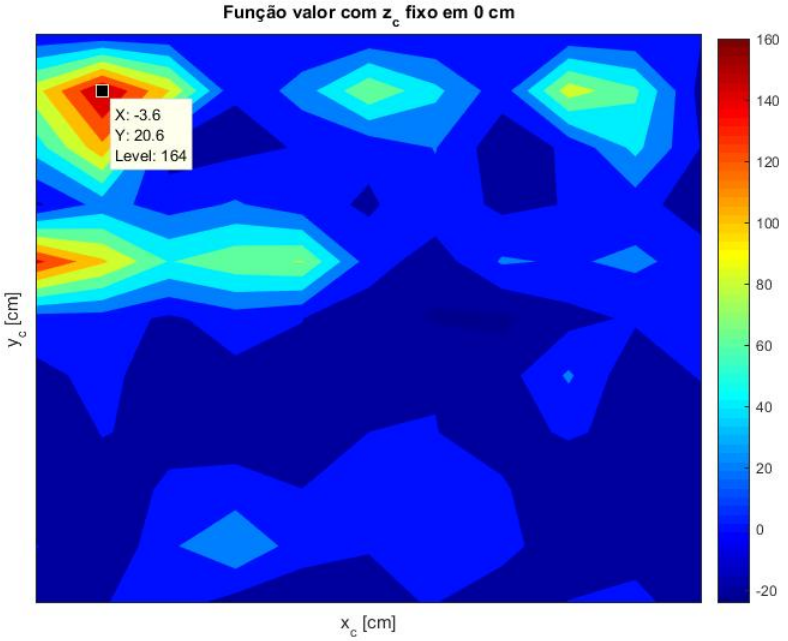


Figura I.1: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0.

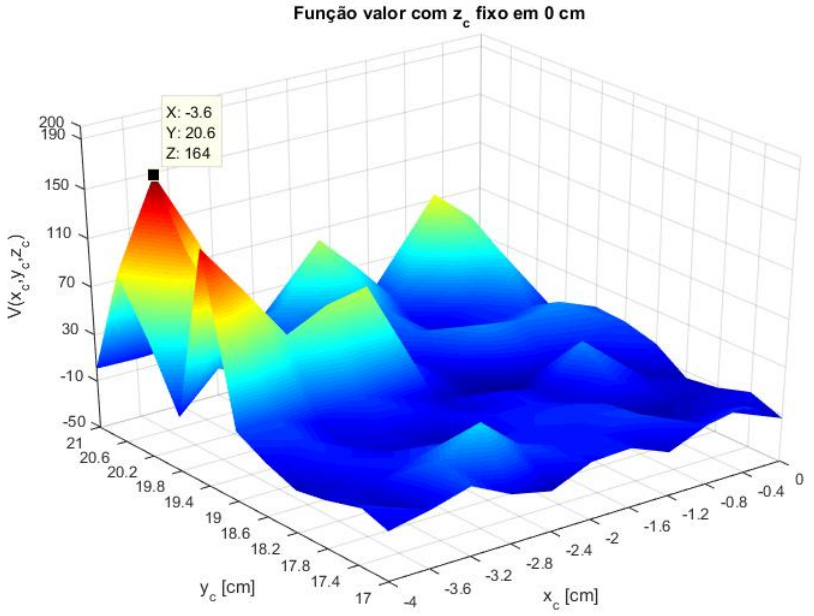


Figura I.2: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0.

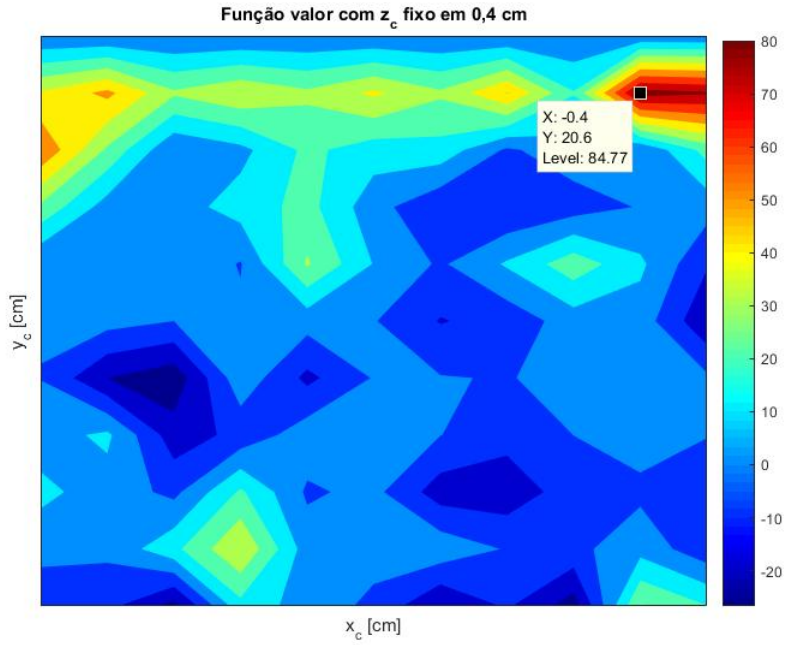


Figura I.3: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0,4.

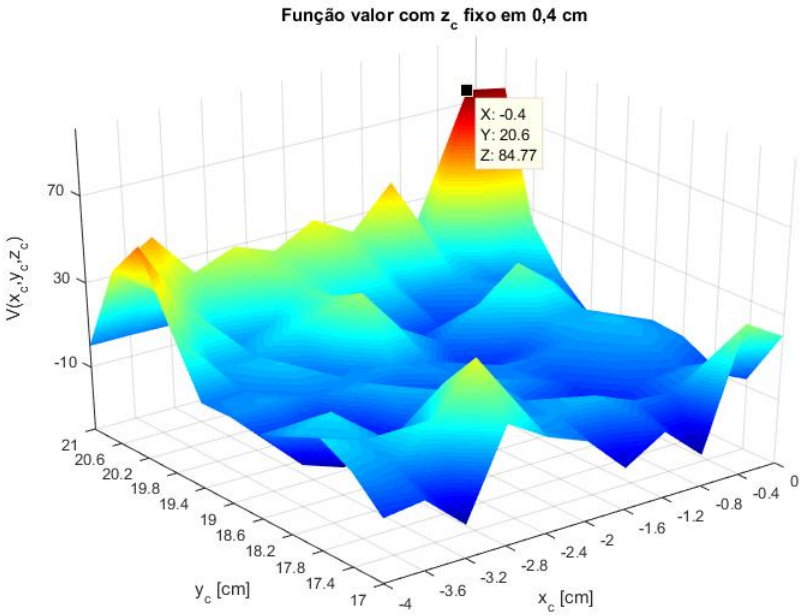


Figura I.4: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0,4.

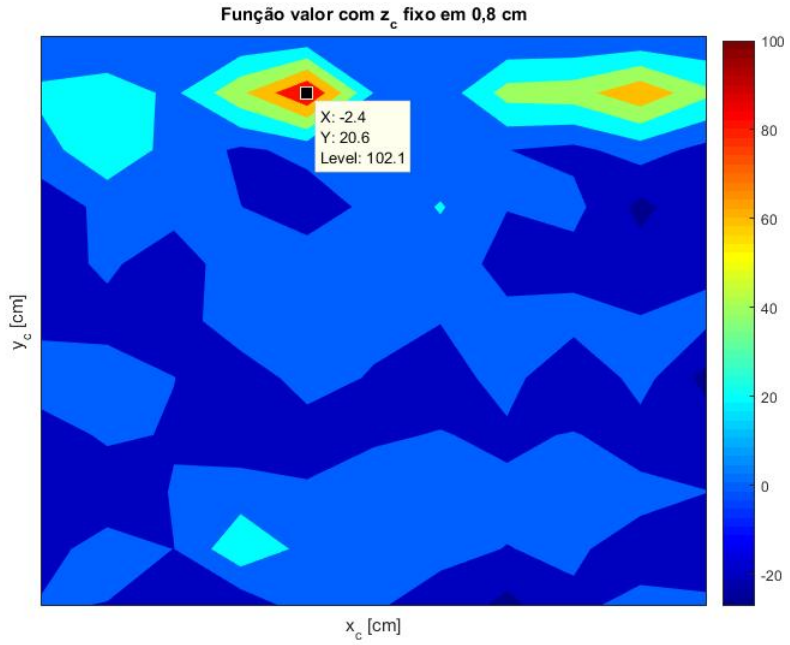


Figura I.5: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0,8.

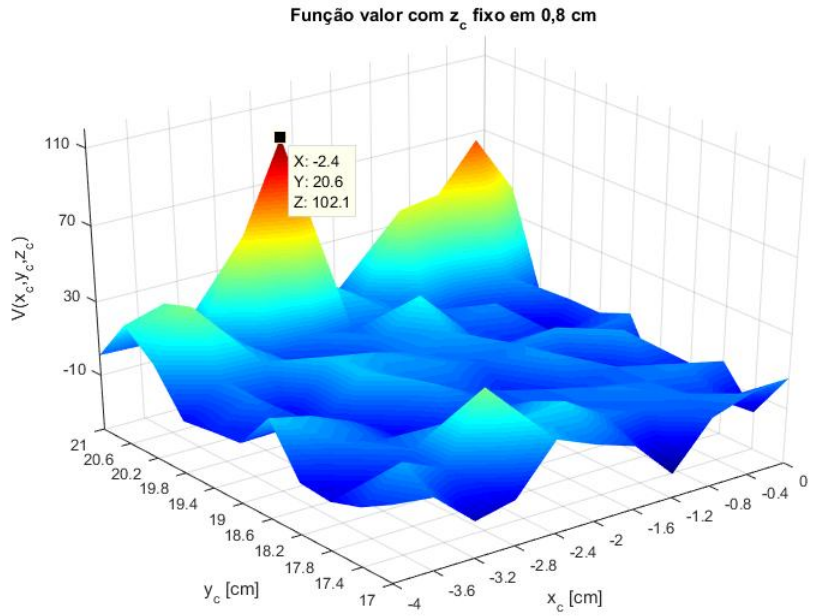


Figura I.6: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0,8.

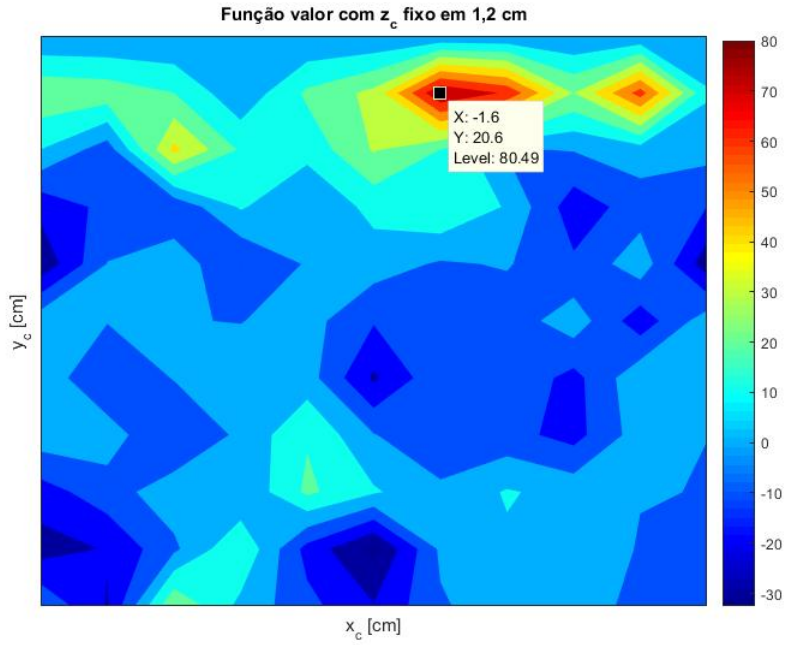


Figura I.7: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 1,2.

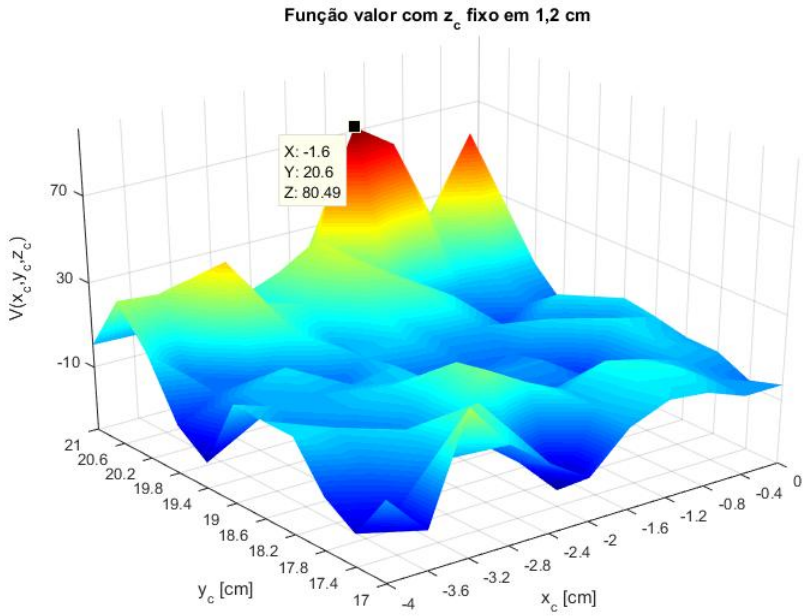


Figura I.8: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 1,2.

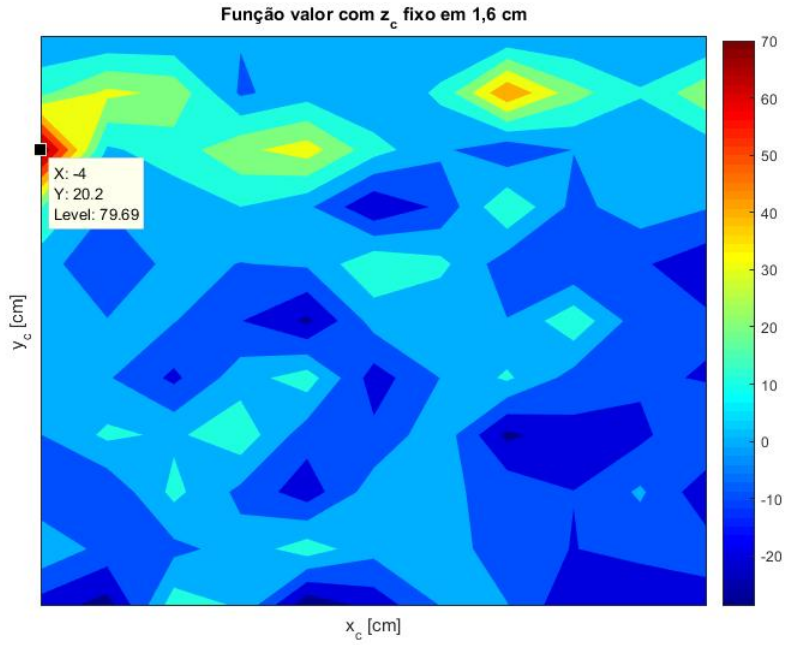


Figura I.9: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 1,6.

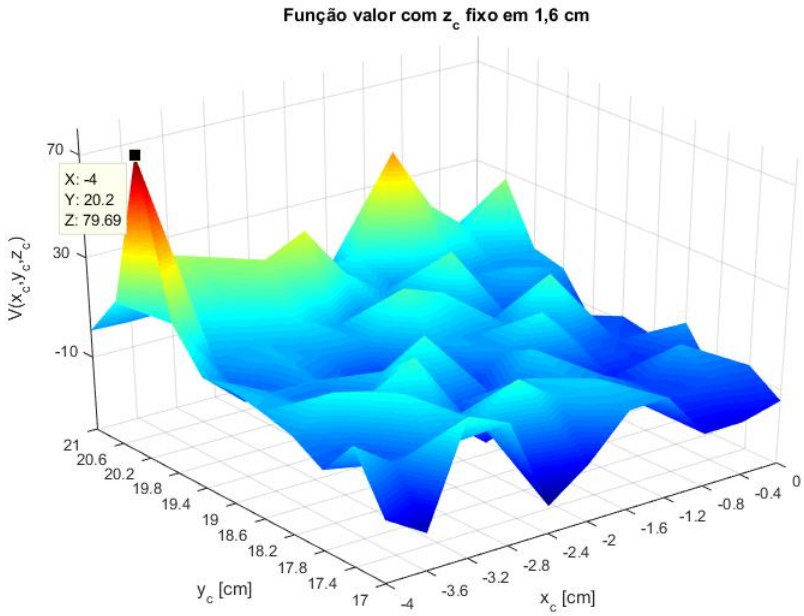


Figura I.10: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 1,6.

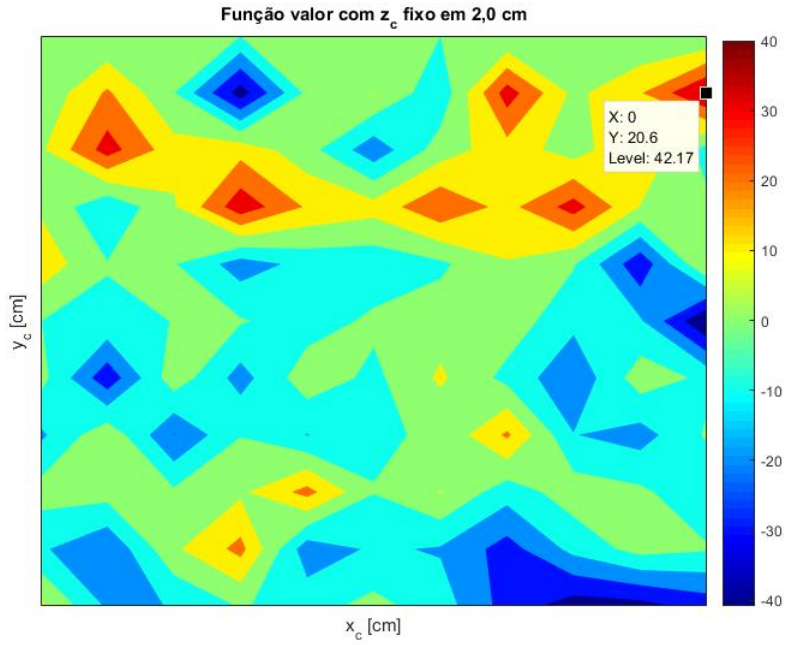


Figura I.11: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 2,0.

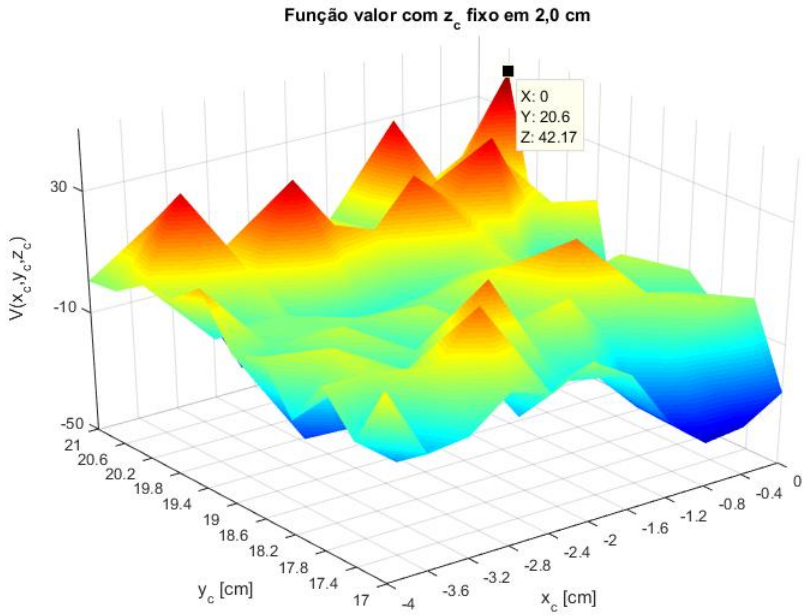


Figura I.12: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 2,0.

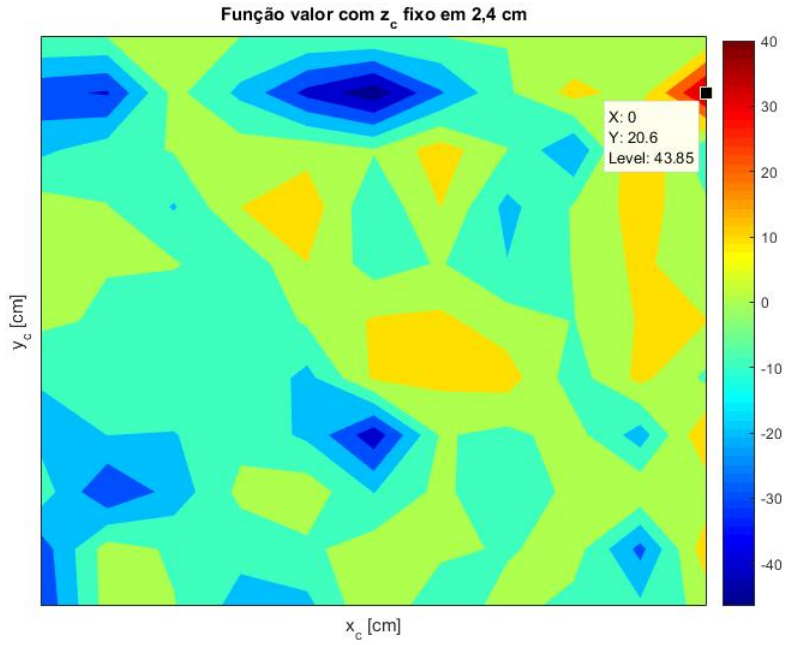


Figura I.13: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 2,4.

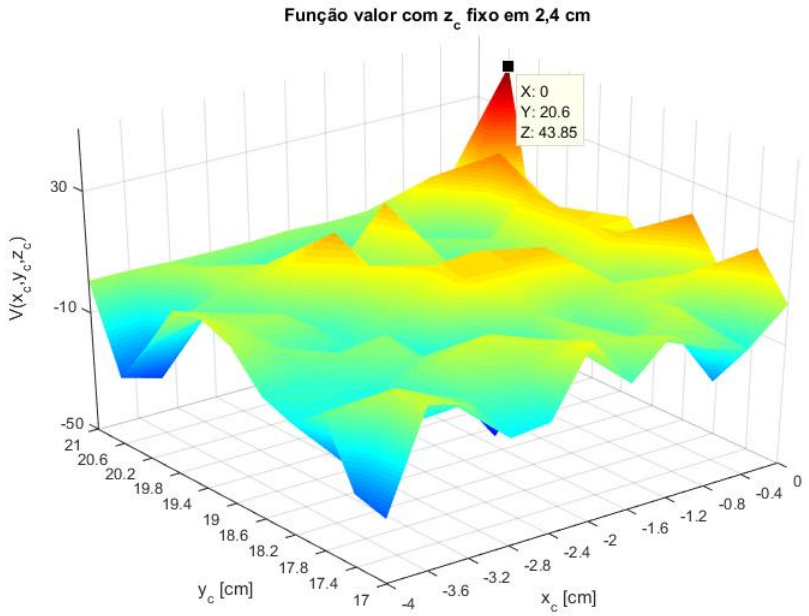


Figura I.14: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 2,4.

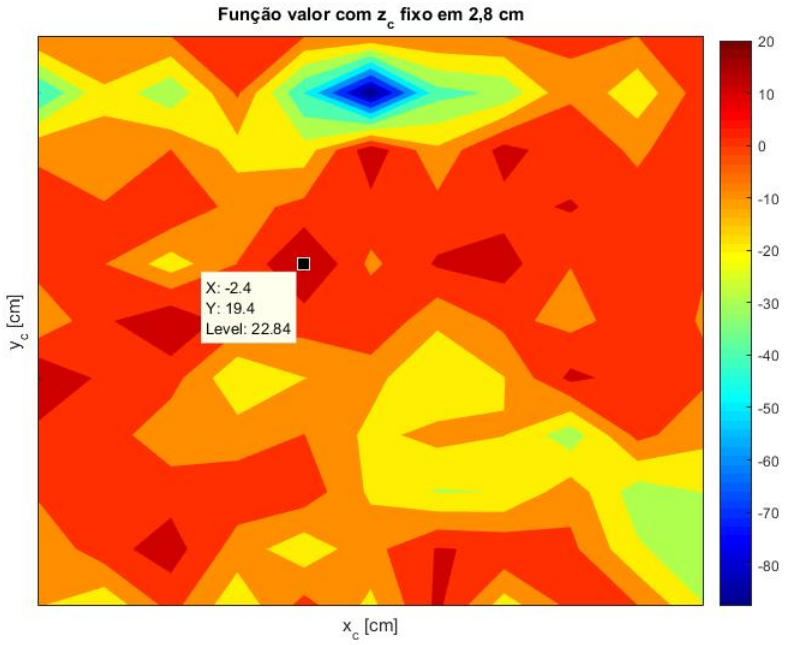


Figura I.15: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 2,8.

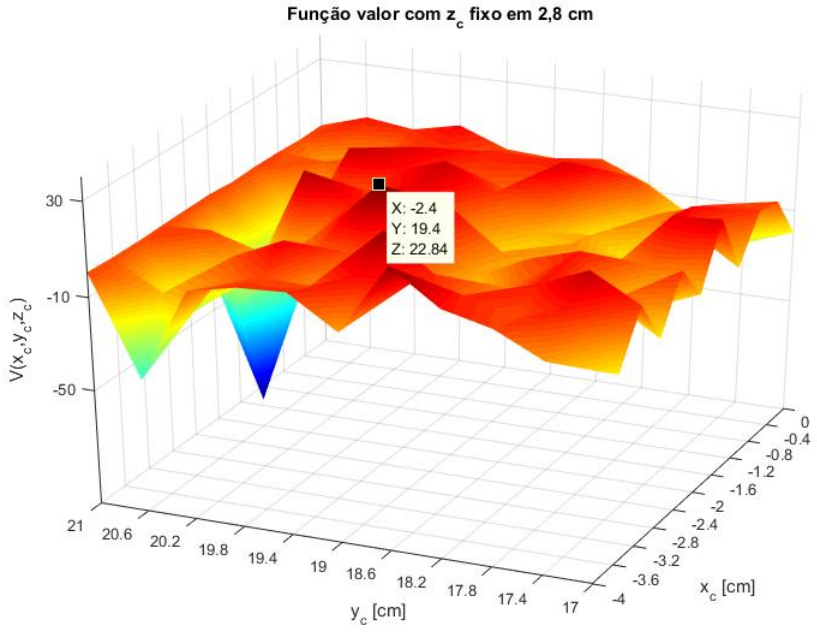


Figura I.16: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 2,8.

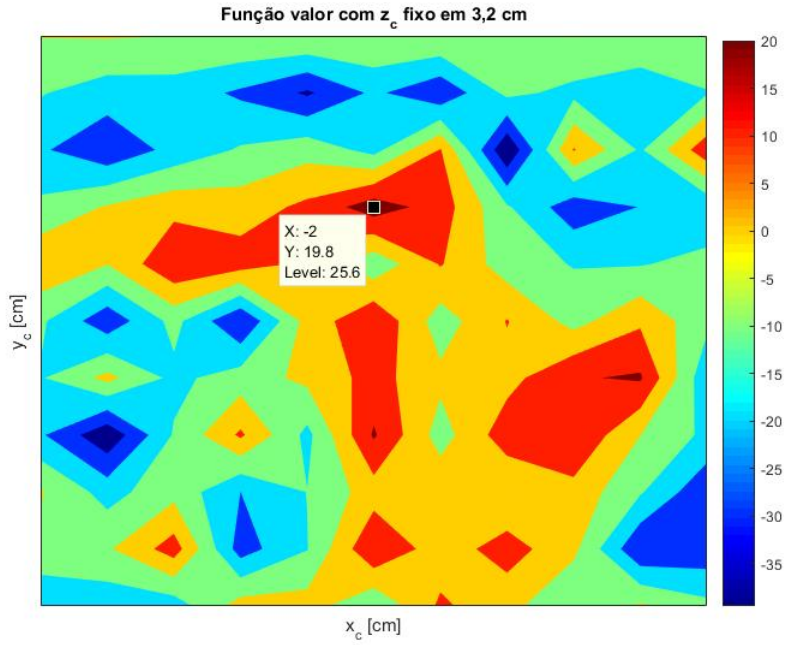


Figura I.17: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 3,2.

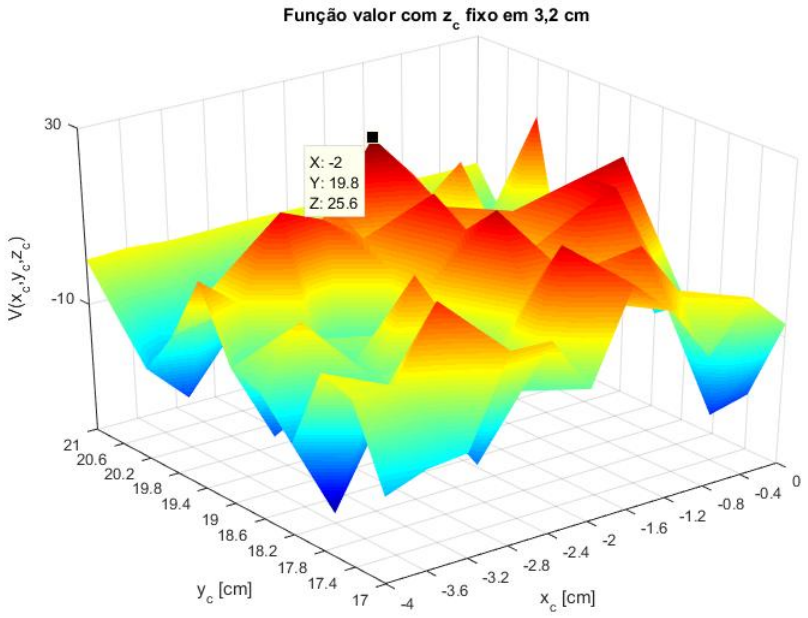


Figura I.18: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0.

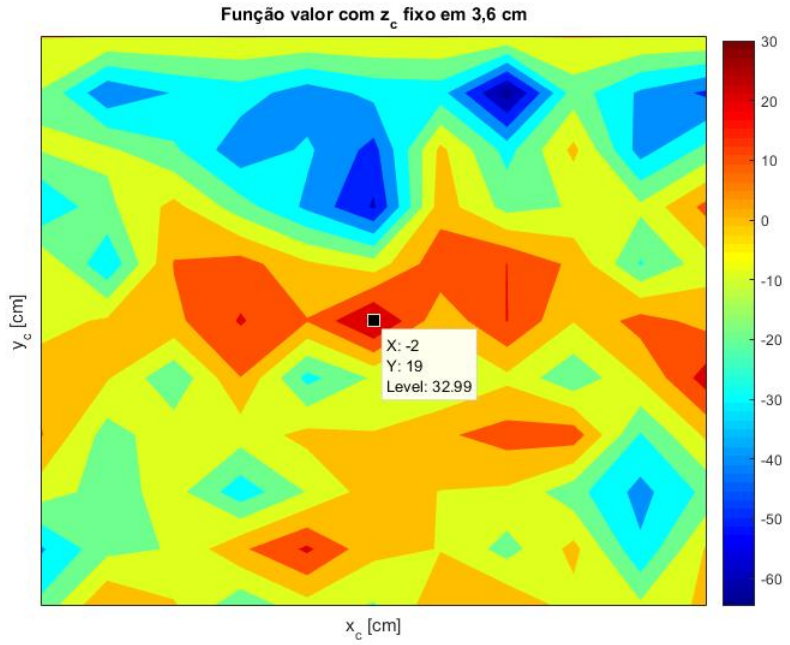


Figura I.19: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 3,6.

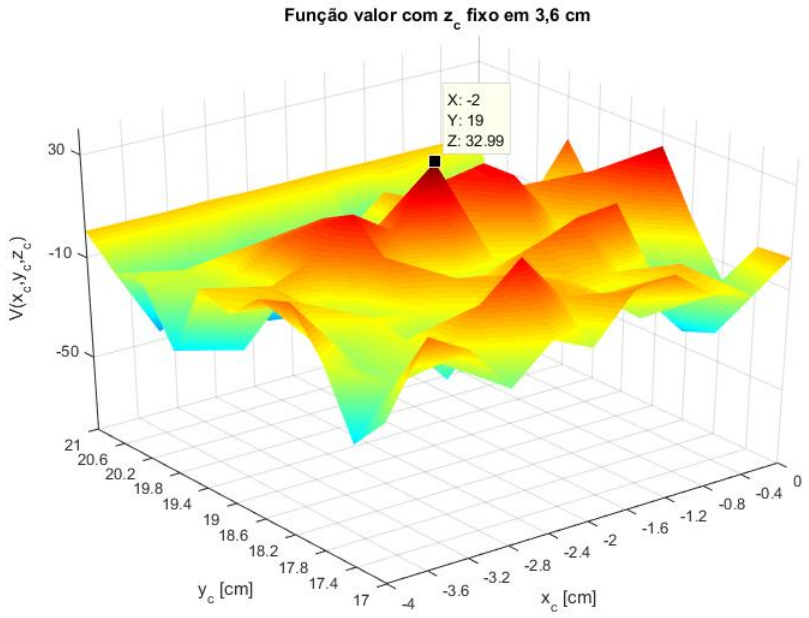


Figura I.20: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 0.

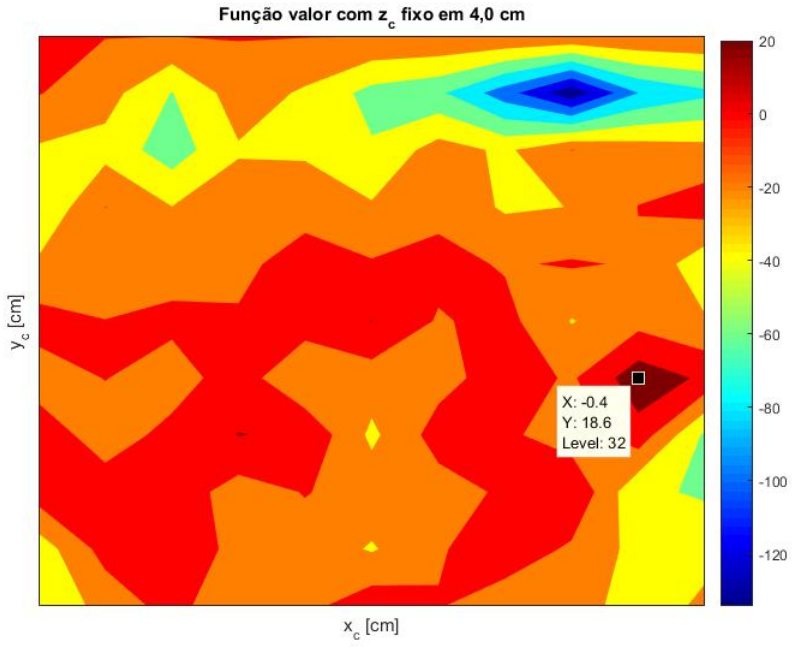


Figura I.21: Gráfico de contorno da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 4,0.

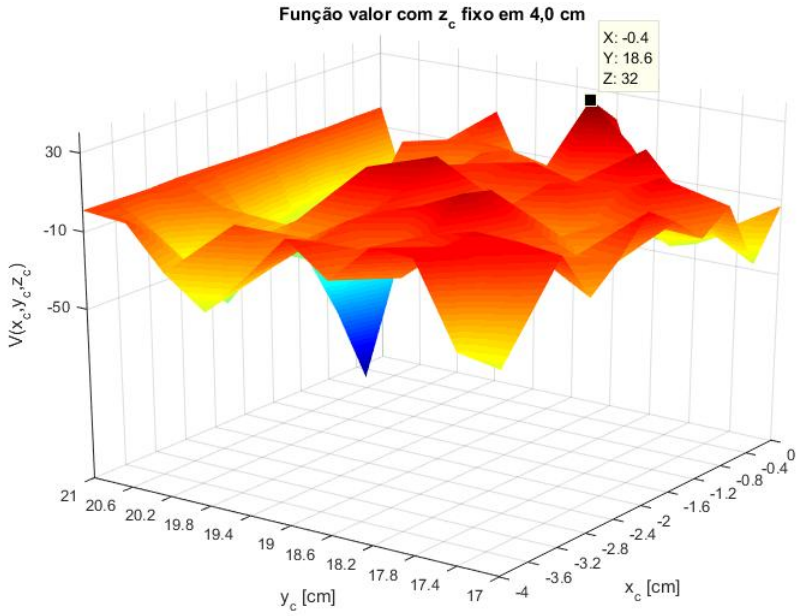


Figura I.22: Gráfico 3D da estimativa dos valores dos estados em função de x_c e y_c, z_c fixo em 4,0.

II. DESCRIÇÃO DOS CÓDIGOS UTILIZADOS

II.1 MATLAB

move_uma_junta.m: Código no matlab que serve para testar o funcionamento das juntas no V-REP. Para ele funcionar deve-se iniciar o V-REP primeiro e depois executar o código no matlab.

cria_pose_unica.m: Código no MatLab que chama a simulação no V-REP. Ele recebe do aplicativo 5 variáveis que são parâmetros das poses criadas e retorna as funções de score (que podem ser modificadas de acordo com seu objetivo).

inverse_kinematics.m: Código no MatLab que implementa cinemática inversa de uma pata do robô. Recebe como entrada parâmetros, que é a posição que se deseja colocar a ponta da pata do robô e retorna os ângulos de junta1 (“ombro” alto), junta2 (“ombro” baixo) e junta3 (“joelho”).

remAPi.m;remoteApi.dll; remoteApiProto.m: Arquivos necessário para realizar a comunicação do MatLab com o V-REP.

rl.m: Código no MatLab que implementa o algoritmo de gradiente de política. Recebe como entrada parâmetros iniciais e o número de marchas a serem avaliadas, ao final retorna novos parâmeros.

td.m: Código no MatLab que implementa o algoritmo de diferença temporal. Retorna a estimativa da função valor de cada estado.

action.m: Lista de ações que podem ser tomadas no algoritmo de diferença temporal.

II.2 PLATAFORMA REAL

gait_generation.py: Código *Python* que implementa cinemática inversa de uma pata do robô. Recebe como entrada parâmetros, que é a posição que se deseja colocar a ponta da pata do robô e retorna os ângulos de junta1 (“ombro” alto), junta2 (“ombro” baixo) e junta3 (“joelho”).

main.cpp: Código *C++* responsável pela calibração do robô e execução de marchas descritas em arquivos de texto.

iniciar.py: Código *Python* que chama os códigos gait_generation.py e main.cpp. Recebe como entrada um arquivo txt que contêm 5 parâmetros, retorna outro arquivo de texto que descreve a marcha e a executa.

rl.py: Código *Python* que implementa o algoritmo de gradiente de política. Recebe como entrada parâmetros iniciais e o número de marchas a serem avaliadas, é necessário entrar com a nota de cada marcha após a execução de cada uma e, então, retorna novos parâmetros.