DURVAL CAMARGO RANGEL DE SOUZA BRITO

USO DO MÉTODO Q-LEARNING PARA APRENDIZADO DE UM ROBÔ QUADRÚPEDE.

LAVRAS – MG

2019

DURVAL CAMARGO RANGEL DE SOUZA BRITO

USO DO MÉTODO Q-LEARNING PARA APRENDIZADO DE UM ROBÔ QUADRÚPEDE.

Relatório técnico apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Graduação em Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

Prof. Wilian Soares Lacerda

Orientador

LAVRAS – MG

2019

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Brito, Durval Camargo Rangel de Souza

Uso do método Q-Learning para aprendizado de um robô quadrúpede. / Durval Camargo Rangel de Souza Brito. – Lavras : UFLA, 2019.

74 p. : il.

TCC(graduação)–Universidade Federal de Lavras, 2019.

Orientador: Prof. Wilian Soares Lacerda.

Bibliografia.

1. Aprendizagem de máquina. 2. Robótica móvel. 3. Q-Learning. I. Lacerda, Wilian Soares. II. Título.

CDD-808.066

DURVAL CAMARGO RANGEL DE SOUZA BRITO

USO DO MÉTODO Q-LEARNING PARA APRENDIZADO DE UM ROBÔ QUADRÚPEDE.
MACHINE LEARNING METHOD USED IN TEACHING A QUADRUPEDAL ROBOT HOW TO WALK

Relatório técnico apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Graduação em Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

APROVADA em 19 de Novembro de 2019.

Prof. Leonardo Silveira Paiva Banca Um UFLA
Prof. Dimitri Campos Viana Banca Dois UFLA

Prof. Wilian Soares Lacerda
Orientador

LAVRAS – MG
2019

Dedico este trabalho à minha mãe e ao meu padrinho. Graças a eles, tive a oportunidade de começar a fazer robótica no 1o ano do ensino fundamental e agora, culmina na apresentação deste trabalho.

AGRADECIMENTOS

Agradeço à minha mãe, por me apoiar sempre e por dedicar tanto à minha educação. Agradeço ao meu padrinho por me proporcionar incontáveis opurtunidades que contribuíram para minha construção pessoal. Agradeço à minha madrinha por sempre me apoiar e comemorar todas as minhas conquistas. Agradeço ao meu orientador, por me ajudar e encarar o desafio que foi esse projeto. Agradeço à minha família por serem todos tão dedicados, inteligentes e unidos. Agradeço aos meus amigos e amigas que estiveram presentes comigo durante a faculdade, nos melhores e piores momentos desta jornada incrível.

O conhecimento torna a alma jovem e diminui a amargura da velhice. Colhe, pois, a sabedoria. Armazena suavidade para o amanhã.
(Leonardo Da Vinci)

RESUMO

Este trabalho registra a implementação do método de aprendizado de máquina por reforço, conhecido como Q-Learning, que foi utilizado para ensinar um robô quadrúpede a se locomover. Para isto, foi construído um robô com quatro patas com o intuito de servir como modelo na implementação do algoritmo. Para o algoritmo do Q-Learning funcionar, é necessário uma tabela de estados e ações para armazenar as recompensas de cada ato, chamada de Tabela-Q. De modo a simplificar o problema, foram criados 2 estados para cada motor, na forma de 2 vetores de suporte contendo posições dos servomotores, assim um bit pôde representar a posição de um servomotor. A união dos bits de estados dos servomotores, permitiu que um número inteiro fosse utilizado como estado do robô, facilitando a indexação da Tabela-Q. As ações foram pares de motores a serem acionados juntos. Devido a uma limitação da memória interna do controlador do robô, foi necessário fazer o treinamento fora do próprio controlador. Dessa forma ao implementar o algoritmo em Python, fez-se necessário criar uma série de regras que simulassem a realidade, penalizando o robô, caso caia, e recompensando-o, caso consiga andar para frente. Após executar o algoritmo dez vezes, alguns resultados diferentes apareceram e todos fizeram o robô andar, sem violar as regras impostas no "ambiente virtual".

Palavras-chave: Aprendizado de Máquina. Q-Leaning. Robô Quadrúpede.

ABSTRACT

This work registers the implementation of a machine learning algorithm via reinforcement learning, known as Q-Learning, that was used in teaching a quadrupedal robot how to walk. For that, a four-legged robot was built to work as a model for the algorithm's implementation. For the Q-Learning algorithm to work, it needs a Table with the robot states' and actions, to store the reward for each action in each state, called Q-Table. In order to simplify the problem, two states are created for each motor in the form of two support vectors containing positions of the servo motors, allowing for a bit to represent the position of the servo motor. Placing those bits in a sequence allowed the use of an integer as the robots' state, assisting with the indexing of the Q-Table. Actions were set to be pair of motors triggered at the same time. Due to a lack of internal memory the training was made outside the controller unit. When implementing the algorithm in Python, a set of rules that simulate the behaviour of the robot in the real world was needed, so it could penalize the robot when it falls and rewarded it when walking forwards. After executing the algorithm a ten times, some different results arise and all of them make the robot walk, without violating the rules imposed in the "virtual environment".

Keywords: Machine Learning. Q-Learning. Quadrupedal Robot.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Desde o início da humanidade, o homem procura modos de facilitar sua vida, produzir mais e ter mais conforto. Tarefas manuais, como arar a terra, foram substituídas por animais e hoje são feitas por máquinas, aumentando a produtividade. Ainda assim, muitos veem nos robôs, máquinas capazes de tornar o humano obsoleto, ao passo que o objetivo é tornar a vida humana mais fácil.

Robôs executam algoritmos, que são uma série de instruções. A maioria destes algoritmos se baseiam na lógica clássica, o conhecido "se... então... senão..." que, apesar de resolver grande parte dos problemas, pode se tornar inviável facilmente, já que a capacidade de resolver problemas é limitada por quem programou e criou as instruções. Nem sempre problemas tem uma lógica clara e definida, como por exemplo reconhecer rostos, escritas à mão (BISHOP, 2006) ou até mesmo aprender a jogar um jogo como GO (SUTTON; BARTO, 2017). Neste âmbito é que a aprendizagem de máquina entra e torna-se extremamente relevante, por sua capacidade de agrupar, identificar e criar padrões, sendo apenas necessário dados específicos para que algoritmos aprendam a resolver problemas complexos sozinhos.

Poucos robôs tem o comportamento humano de aprendizado contínuo e a introdução da inteligência artificial, utilizada para a movimentação de robôs, torna-se cada vez mais necessária para ir além do que a robótica permite. Robôs assim já estão em desenvolvimento, como é o caso do ANYmal (HWANGBO et al., 2019), um robô quadrúpede que utiliza aprendizado de máquina por reforço, com uma política de treinamento baseado em redes neurais artificiais, permitindo que ele execute tarefas de forma mais precisa, eficiente e ágil. Neste trabalho, foi discutido e implementado um algoritmo de inteligência artificial capaz de aprender a locomover um robô quadrúpede.

1.1 Objetivos

1.1.1 Objetivos Gerais

O objetivo deste trabalho aplicar o método Q-Learning de aprendizagem de máquina por reforço para ensinar um robô quadrúpede a se locomover. Este objetivo tem como finalidade o engrandecimento pessoal.

1.1.2 Objetivos específicos

Em função da complexidade do objetivo geral, este será dividido em partes menores para que fique mais compreensivo:

- Estudo e identificação do melhor método de aprendizagem de máquina a se aplicar, considerando características físicas do robô para que seja possível a aplicação do aprendizado;
- Construção do robô considerando a simplificação necessária para utilizar o aprendizado de máquina;
- Implementação do algoritmo de treinamento, baseado no aspecto construtivo do robô, e treino da inteligência artificial;
- Aplicação do resultado do treinamento no robô, verificação da funcionalidade e análise de performance.

1.2 Motivação

A finalidade deste trabalho é o crescimento pessoal, pesquisando e aprendendo sobre aprendizado de máquina por reforço e aplicando o conhecimento na construção de um robô quadrúpede que se locomova via o resultado do treinamento da inteligência artificial.

1.3 Organização do texto

O trabalho será dividido em quatro partes:

1. Revisão literária, onde são citados trabalhos correlatos, e mostrada sua relevância para a produção deste trabalho;

2. Definições, onde terá uma breve descrição dos conceitos utilizados neste trabalho;

3. Materiais e Métodos, onde será exposto o que foi utilizado e também o como foi feito o trabalho em si;

4. Resultados, onde será exposto as saídas do treinamento, os gráficos que representam a funcionalidade do método e explicado o resultado alcançado no trabalho;

5. Conclusão, onde será resumido os resultados obtidos e explicitado o que trabalhos futuros podem acrescentar, tanto na parte construtiva quanto no método de treinamento do robô.

2 REVISÃO

2.1 Conceitos

Serão abordados alguns pontos importantes para o projeto, definindo os conceitos dos tópicos seguintes para melhor entendimento do trabalho.

2.1.1 Robótica

Robótica é um ramo da ciência dedicada ao estudo, projeto e desenvolvimento de robôs. A robótica une 3 áreas do conhecimento, sendo elas: mecânica, eletrônica e computação. Um robô é composto por:

- Sistema de ação mecânica, ou seja, parte que gera movimento;
- Sistema de sensoriamento, que adquire informações do ambiente. O modo de obter os dados pode ser virtualmente ou fisicamente;
- Sistema de controle, parte que processa os dados obtidos e evite comandos ao sistema mecânico para agir no ambiente.

Robôs podem ser autônomos ou semi-autônomos e são muito utilizados para executar tarefas perigosas, cansativas ou repetitivas. Na Figura 2.1 é mostrado um manipulador com intuito educacional. Nele são utilizados servomotores, máquinas que contém diversos componentes:

- Motor, que representa seu atuador.
- Sensor, que indica a posição do motor que normalmente é um potenciômetro. A qualidade desse potenciômetro influencia na estabilidade, na precisão e exatidão do servomotor.
- Circuito de controle, que modula a tensão aplicada para manter o motor na posição correta.

Figura 2.1 – Manipulador

Fonte: (ROBOCORE, 2019)

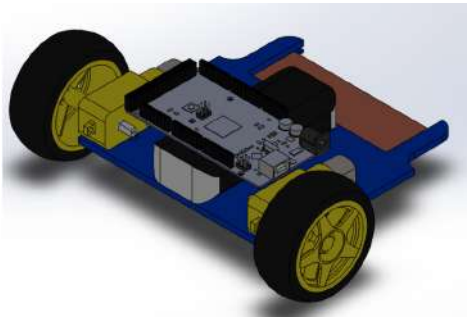
2.1.2 Robótica móvel

Robótica móvel é aquela que se dedica a aplicar os conhecimentos de robótica com o objetivo de movimentar o próprio robô, mudando seu referencial em relação a um referencial inicial. Robôs móveis terrestres podem ser construídos com rodas, esteiras ou pernas e cada tipo tem suas particularidades, vantagens e desvantagens. A escolha para cada projeto se dá com base nos objetivos, restrições e desafios.

2.1.2.1 Rodas

Modelo mais comum, por ser simples de se implementar. A principal desvantagem é sua dificuldade com terrenos irregulares. O número de rodas pode variar, apesar de o mais conhecido ser o de 4 rodas. Na Figura 2.2 tem-se um exemplo de robô com 3 rodas.

Figura 2.2 – Robô com rodas

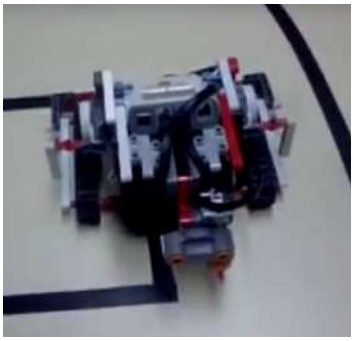


Fonte: do autor 2018

2.1.2.2 Esteiras

Modelo menos comum que rodas, apresenta grande superfície de contato com o chão, o que dificulta fazer curvas deixando-o menos ágil. Mas sua vantagem sobre as rodas é que consegue um desempenho bom em terrenos irregulares. Na Figura 2.3 há um robô com esteiras, utilizado para ultrapassar obstáculos em uma competição de robótica.

Figura 2.3 – Robô com esteiras



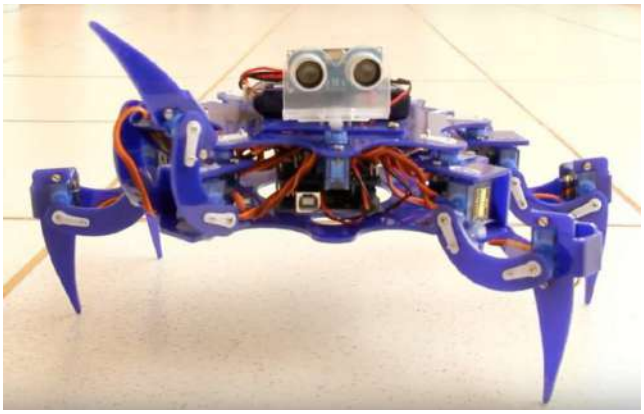
Fonte: do autor 2013

2.1.2.3 Pernas

Modelo mais complexo devido à grande quantidade de partes móveis. A velocidade, agilidade e complexidade variam com a quantidade de pernas e nú-

mero de graus de liberdade que cada perna tem, ou seja, o número de articulações das pernas do robô. A Figura 2.4 contém um robô hexápode montado e programado por um estudante da UFLA.

Figura 2.4 – Robô Hexápode



Fonte: (SILVEIRA, 2018)

2.1.3 Microcontroladores

Microcontroladores são feitos para serem utilizados em aplicações embarcadas, ou seja, para que sejam programados e adicionados periféricos com fins específicos (GRIDLING; WEISS, 2007). Um microcontrolador é composto por algumas partes sempre presentes:

- Processador;
- Memória;
- Portas de Input/Output (I/O);

2.1.3.1 Comunicação

Alguns microcontroladores se comunicam via protocolos específicos, permitindo comunicação com maior número de dispositivos, que não caberiam apenas nas portas I/O do próprio microcontrolador.

2.1.3.2 Protocolo I^2C

O I^2C é um protocolo de comunicação do tipo mestre/escravo. É bastante simples, utilizando apenas duas linhas, uma para a transmissão de dados, e outra para o clock (velicidade/momento) de comunicação. O canal de dados é bidirecional, ou seja, em certo instante pode ser utilizado pelo mestre para transimitir e em outro momento pode ser utilizado por um escravo para mandar dados para o mestre. O protocolo também tem algumas características como (IRAZABAL; BLOZIS, 2003):

- Enderaçamento único para cada dispositivo conectado;
- Permite detecção de colisões caso mais de um mestre tente transmitir ao mesmo tempo;
- Cada "palavra"enviada, contém 8 bits e o dispositivo que recebe o dado deve informar que está recebendo os dados corretamente a cada byte recebido;
- Filtro para evitar picos repentinos;
- Checagem de comunicação a cada byte transmitido;

2.1.4 Aprendizagem de Máquina

Aprendizagem de máquina é uma aplicação de inteligência artificial que utiliza dados, modelos matemáticos e algoritmos, para se adaptar e melhorar por experiência própria. Existem várias abordagens à aprendizagem de máquina, dentre elas, as mais conhecidas são: métodos supervisionados, métodos não-supervisionados e métodos por reforço (BISHOP, 2006).

2.1.4.1 Métodos supervisionados

Os métodos supervisionados são mais intuitivos, o algoritmo tem acesso à resposta que deveria dar para cada entrada, e se ajusta de modo a acertar na maioria

das vezes. Esta abordagem é uma das mais utilizadas e costuma trazer resultados mais robustos, porém é necessário que existam dados para seu treinamento. Alguns algoritmos conhecidos são (BISHOP, 2006):

- Perceptron;
- Redes Neurais Artificiais;
- Máquinas de Vetores de Suporte;

2.1.4.2 Métodos não-supervisionados

Os métodos não supervisionados recebem como entrada dados qua não contém uma resposta uma resposta esperada para o treinamento. A partir dos dados disponíveis, é feita a identificação de padrões por modelos matemáticos e estatísticos, permitindo criar agrupamentos de dados parecidos(clustering), detecção de anomalias, entre outros (BISHOP, 2006).

2.1.4.3 Métodos por reforço

Para os métodos por reforço é necessário um objetivo explícito, considerando o problema como um todo, diferentemente dos outros métodos, que não deixam explícito qual o objetivo final ou como seriam úteis. A diferença para o método supervisionado é que não é necessário ter o par entrada/saída, já que se baseia em recompensas para atingir um objetivo. Para cada ação tomada, o ambiente recompensa ou penaliza o algoritmo, que tenta maximizar sua recompensa. Então, qual ação deve ser tomada quando ainda não se tem conhecimento do que fazer? A solução é criar um paralelo entre a exploração e usufruto. A exploração é testar ações aleatoriamente de modo a entender melhor o ambiente, o usufruto é tomar a ação que resulta na maior recompensa (SUTTON; BARTO, 2017).

2.1.4.4 Funcionamento do Q-Learning

O método de aprendizagem por reforço, chamado de Q-Learning, consiste em criar uma tabela com todos os estados possíveis em seu índice e as ações executáveis nas colunas. Temos então uma Tabela Q com os estados S e as possíveis ações A. Para cada ação A tomada, gera-se um novo estado S' e para este novo estado, há uma ação A' que trará a maior recompensa. A Tabela Q armazena o valor das recompensas para os atos executados em cada estado. Atualiza-se o valor da tabela com uma porcentagem, representada pelo valor α e o melhor resultado possível para o estado seguinte a partir da ação tomada, visto na Equação 2.1 (SUTTON; BARTO, 2017). Um exemplo de Tabela Q pode ser vista no Quadro 2.1 em que A_n é uma ação a ser tomada, S_m é um estado possível e R_{mn} é a recompensa para tomar a ação A_n no estado S_m.

Quadro 2.1 – Exemplo Tabela Q

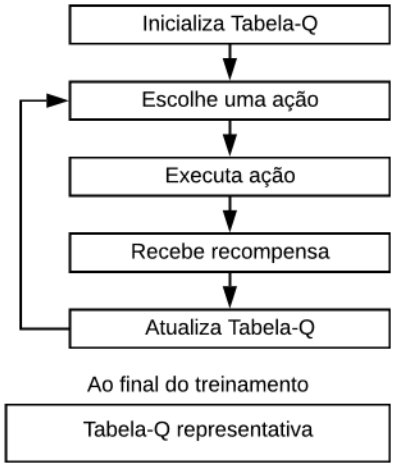
	A_1	A_2	A_3
S_1	R_{11}	R_{12}	R_{13}
S_2	R_{21}	R_{22}	R_{23}
S_3	R_{31}	R_{32}	R_{33}

Fonte: do autor 2019

$$Q(S,A) = Q(S,A) + \alpha[R + \gamma max(Q(S',A')) - Q(S,A)] \tag{2.1}$$

O algoritmo do Q-Learning funciona do seguinte modo: dado um estado inicial, executa uma ação, vai para o próximo estado, recebe uma recompensa do ambiente, atualiza a Tabela Q pela Equação 2.1 e reinicia o processo, como é visto na Figura 2.5. O treinamento pode ou não ser infinito, dependendo implementação do algoritmo. Em muitos casos, o treinamento termina depois de uma quantidade máxima de iterações ou quando certas condições são atingidas.

Figura 2.5 – Algoritmo do Q-Learning



Fonte: do autor 2019

2.2 Trabalhos correlatos

A robótica móvel pode ser dividida em modelagem cinemática, que estuda o movimento dos corpos sem considerar as forças agindo sobre os objetos, e dinâmica, que estuda as forças atuantes nos corpos movimentados. Tais modelagens são bastante complexas e costumam ser simplificadas para aplicaçãoes específicas, como feito para a modelagem dinâmica aplicada a robôs quadrúpedes em (PIZZIOLO et al., 2004), onde tal simplificação obtém uma resposta em regime permanente próxima do real.

A variação do ambiente no qual o robô se encontra pode trazer problemas quando utila-se modelos simplificados, já que costumam ser orientados à área de trablho do robô. Para isso, algoritmos de inteligência artificial são utilizados para que o robô possa se adaptar de acordo com o ambiente no qual se encontra. Em (SELVATICI; COSTA, 2007), comportamentos do robô são adaptáveis e modificados por algoritmos de aprendizagem de máquina. Nesse trabalho, é utilizada uma arquitetura baseada em comportamentos, assim criaram-se módulos de

reações que se adaptam com o ambiente à sua volta utilizando aprendizagem por reforço.

O aprendizado por reforço é aplicado em robôs com pernas para a coordenação de seus atuadores, simulado e implementado em (SANTOS; JúNIOR, 2012). Nesse trabalho, é aplicado tanto para um robô quadrúpede quanto para um robô trípede, provando a generalização do método. O resultado é executado e testado empiricamente e comparado com as simulações feitas. Isso demonstra a aplicabilidade de se empregar algoritmos de aprendizagem de máquina para robôs com pernas, otimizando sua locomoção.

3 MATERIAIS E MÉTODOS

Este capítulo tem como finalidade expor os materiais utilizados no projeto, bem como explicitar a metodologia utilizada para a criação dos algoritmos, de modo a facilitar o melhor entendimento dos resultados obtidos.

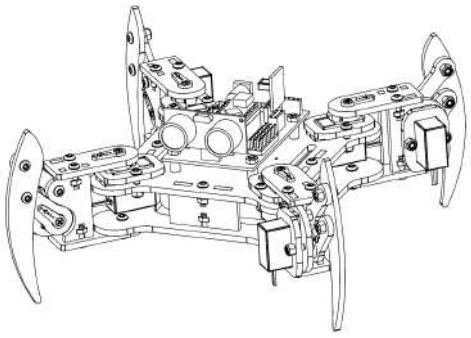
3.1 Projeto

Foi escolhido um robô quadrúpede com dois graus de liberdade em cada membro e consiste em: esqueleto, atuadores, bateria e controladores. O esqueleto é apenas a estrutura que representa o robô, os atuadores são os servomotores, a bateria alimenta os circuitos e os controladores são: o Arduino Mega 2560 e o Driver PCA9685, para controlar os servomotores.

3.1.1 Estrutura

A estrutura escolhida foi a do projeto mePed (PIERCE, 2016), como na Figura 3.1, em acrílico, mas com algumas pequenas modificações que serão vistas na Sessão 3.2.

Figura 3.1 – Modelo esquemático do mePed



Fonte: (PIERCE, 2016)

3.1.2 Servomotores

Foi escolhido o servomotor TowerPro MG90S da Figura 3.2 por ter um torque maior, ser mais confiável e ter engrenagens de metal, tornando-o mais durável (TOWERPRO, 2014).

O servomotor é utilizado em conjunto com o Arduino, que envia comandos a ele em PWM. Bibliotecas específicas do Arduino ajudam a simplificar o controle, permitindo que um ângulo seja convertido no valor de PWM e enviado ao servomotor.

Figura 3.2 – Servomotor TowerPro MG90S

Fonte: (TOWERPRO, 2014)

3.1.3 Bateria

A bateria escolhida foi um pack de bateria da multilaser de 2500mah e 4.8V. Pelo robô ter apenas 8 motores, não houve necessidade de uma bateria com potência maior.

3.1.4 Controladores

Os controladores são responsáveis por aplicar o algoritmo implementado em software. O Arduino contém as posições dos motores e os encaminha à placa PCA9685, que executa os comandos nos atuadores.

3.1.4.1 Arduino

O Arduino é uma plataforma de prototipagem eletrônica, com um microcontrolador programável. Como o Arduino tem portas de entrada e saída, é possível programá-lo de modo a obter dados do ambiente, processá-los e dar uma resposta aos atuadores. Para o projeto foi utilizado o Arduino Mega como o da Figura 3.3, pois o aluno já o possuía.

Figura 3.3 – Arduino Mega 2560

Fonte: (FILIPEFLOP, 2019)

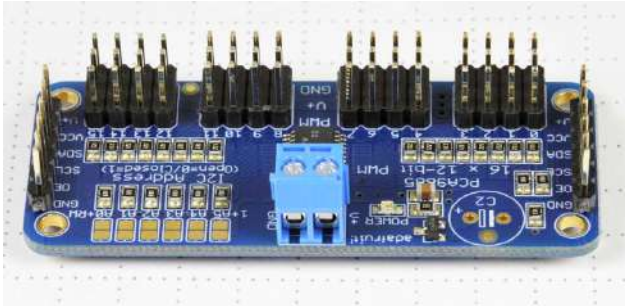
3.1.4.2 PCA9685

O módulo PCA9685 da Figura 3.4 é utilizado como um organizador de fios que separa o circuito dos servomotores do Arduino. O módulo utiliza o protocolo de comunicação I^2C, explicado na Sessão 2.1.3.2, para controlar até 16 servomotores de uma vez (BAUERMEISTER, 2017).

3.2 Construção

A parte construtiva do robô tem o intuito de demonstrar empiricamente que o método de machine learning ensina o robô a andar de modo natural. Salienta-se que o objetivo não é ter um robô capaz de vencer obstáculos ou correr, mas

Figura 3.4 – Controlador PCA9685



Fonte: (BAUERMEISTER, 2017)

sim, um robô com a capacidade de andar, tornando viável a aplicação do machine learning.

3.2.1 Montagem

Partindo do projeto original do robô (PIERCE, 2016), foram feitas pequenas modificações e correções:

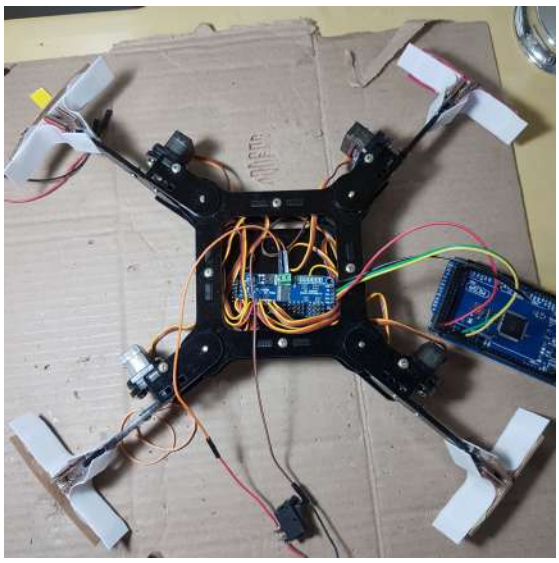
1. O corpo do robô foi posicionado com o vão para cima, como visto na Figura 3.5.

2. Os servomotores das patas foram dispostos de modo à caixa de engrenagem dos servomotores apontarem para a mesma direção, estabelecendo assim, a frente do robô, como visto na Figura 3.5.

3. Para manter o equilíbrio do robô, foram colados na ponta das patas, com cola quente, pedaços plásticos e papelão, como visto na Figura 3.6.

4. Para corrigir um erro do projeto original, foi parafusado diretamente na pata, o braço em cruz do servomotor, como visto na Figura 3.7. Isso foi necessário, pois o projeto original possui parafusos que se soltam devido ao constante movimento e, em razão disso, algumas peças do projeto original não foram utilizadas. Adaptação ilustrada na Figura 3.8.

Figura 3.5 – Robô visto de cima



Fonte: do autor 2019

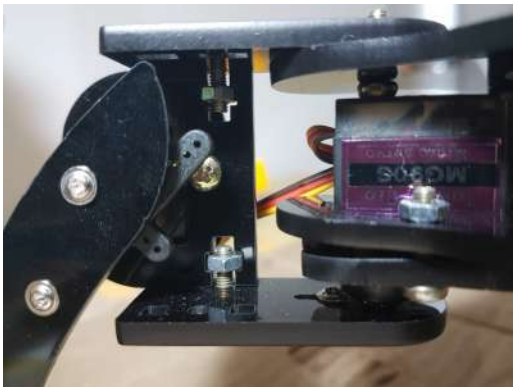
Figura 3.6 – Pata com apoio colado

Fonte: do autor 2019

Figura 3.7 – Pata com braço em cruz

Fonte: do autor 2019

Figura 3.8 – Junção da pata com o quadril



Fonte: do autor 2019

3.3 Implementação

Para o controle do robô, foi necessário executar um código no Arduino, para desempenhar o movimento dos membros. A cada junta é dado um número de 0 a 7, que corresponde ao servomotor que controla aquela junta como na Figura 3.9. Os servomotores das juntas do robô são divididos em dois tipos: patas e quadris. Cada pata pode estar para baixo, tocando o chão, ou para cima, sem tocar o chão. Cada quadril está ligado em uma pata e permite que esta seja movimentada para frente ou para trás.

Também são definidos vetores com posições, em graus, de máximo e de mínimo para cada servomotor; estes vetores são chamados de servomax e servomin, respectivamente. O valor máximo é fixado de modo a ser a posição de uma pata para cima ou um quadril para trás. O valor mínimo é definido de modo a ser a posição de uma pata para baixo ou um quadril para frente.

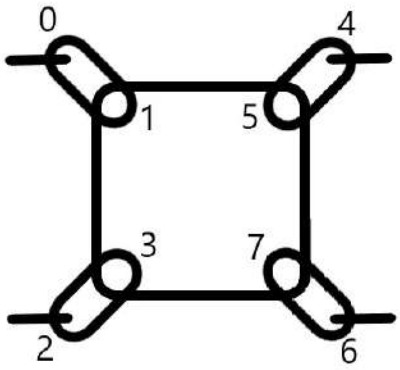
O código é simplificado para não ser necessário calcular a posição e velocidade dos motores; apenas utiliza-se bits, onde 0 corresponde ao valor da posição mínima e 1 o da posição máxima. Assim, a função "walk" do código em Arduino do Apendice B, recebe um vetor de bits, sendo a posição do vetor correspondente ao número do motor, e procura qual a nova posição do servomotor nos vetores de máximo ou mínimo, a depender do valor do bit.

Observa-se que o valor mínimo da posição do servomotor nem sempre é menor que seu respectivo valor de máximo, apenas representa o estado do motor e seu valor em graus para que se enquadre na regra pré-definida. Um exemplo hipotético, que se encaixa na situação descrita, seria: uma pata X toca o chão quando está em 200^{o} e está para cima quando se encontra em 240^{o}, ao passo que uma pata Y toca o chão quando está em 240^{o} e está levantada em 200^{o}. Portanto o valor mínimo da pata X será 200^{o} e o máximo 240^{o}, já a pata Y terá o valor mínimo de 240^{o} e máximo de 200^{o}.

Como não há controle da posição exata na qual se encontra o motor, já que não há um sensor de posição para cada servomotor, é necessário fixar um tempo de espera entre um passo e outro, para que apenas dois motores estejam em movimento ao mesmo tempo, executando apenas uma ação por vez; o tempo utilizado foi de 300ms.

O código do Arduino, utiliza uma biblioteca que converte os valores em graus para o valor em pwm, utilizado pelo servomotor, e também permite que seja enviado via protocolo I^2C para os servomotores.

Figura 3.9 – Diagrama de numeração de servomotores do Robô



Fonte: do autor 2019

3.4 Machine Learning

Dentre os vários tipos e métodos aplicáveis de Machine Learning descritos no Capítulo 2, o escolhido foi o aprendizado por reforço chamado Q-Learning. Métodos supervisionados, simplesmente não teriam sentido, já que não há uma base de dados que tenha a resposta para como andar, no caso específico do robô criado. Métodos não supervisionados tem uma melhor serventia para trabalhar com agrupamento de dados, por isso também não seriam úteis, já que não existem dados neste caso. O tipo mais interessante é o de aprendizagem por reforço, assim, o robô pode explorar seu modo de andar e aprender com seus erros, cumprindo com o objetivo do trabalho, mesmo sendo um ambiente virtual.

3.4.1 Q-Learning

Apesar do algoritmo o Q-Learning ser simples, como visto no Tópico 2.1.4.4, é necessário adicionar uma "camada" extra chamada de episódio, onde o algoritmo funcione durante um período determinado. Dentro de cada episódio, há um limite máximo de épocas que acontecem. Um episódio acaba quando atinge

o valor limite de épocas ou quando ocorre um evento que torna necessário começar novamente. Época é cada iteração do algoritmo Q-Learning de treinamento, ou seja, cada momento que uma ação é escolhida.

O objetivo é buscar quais ações tomar para maximizar a recompensa final. Mas como maximizar a recompensa quando não se tem nenhum conhecimento do ambiente? Este é o dilema da exploração/usufruto: tomar uma ação já conhecida, que traz boas recompensas; ou tomar uma ação aleatória que pode ou não trazer melhores recompensas. Uma forma de mitigar este dilema, é criar uma política de ganância progressiva, onde inicialmente explora-se muito e, ao longo do aprendizado, passa a usufruir mais do conhecimento já adquirido (SUTTON; BARTO, 2017).

Para esta política de ganância, utiliza-se uma medida de desconfiança, representada pela letra ε e, antes de uma ação ser escolhida, é gerado um valor aleatório uniforme e comparado com o valor atual de ε, caso seja menor, opta por uma ação aleatória e caso seja maior, opta pela ação que retorna a maior recompensa (VIOLANTE, 2018). A cada época, a falta de confiança na Tabela Q deve diminuir muito pouco, permitindo que o robô explore mais o ambiente no início e, à medida que aprende e comete menos erros, consiga dar mais passos em cada episódio até o ponto em que a confiança é total e o robô não escolhe ações aleatórias. A implementação do código em linguagem Python, pode ser vista no Apêndice A e melhor compreendido em um fluxograma na Figura 3.10.

Como o treinamento é feito offline (fora do controlador do robô), foi necessário criar a Tabela Q e um "ambiente virtual". A Tabela Q deve ter os estados em seu índice, como explicado na Sessão 2.1.4.4; como o robô tem 8 motores, foi utilizado um número inteiro para representar o valor binário dos estados dos motores. A pata pode estar para cima representado pelo binário 1, ou para baixo, representado pelo binário 0. O quadril pode estar para frente, representado pelo binário 0, ou para trás, representado pelo binário 1. A combinação dos 8 motores

resulta em uma sequência de 8 bits, que pode ser interpretado como um número inteiro de 0 a 255. Além dos estados, a Tabela Q contém as ações possíveis que o robô pode executar. Foi escolhido que as ações são o movimento de dois motores ao mesmo tempo, consequentemente, as colunas são as combinações possíveis de quais motores movimentar.

O ambiente virtual consiste em uma série de regras que ditam a recompensa que cada ação tomada pode trazer, esse ambiente é representado na função "calculateReward" do código no Apêndice A. Veja as regras a seguir:

- Se o robô estiver com duas patas consecutivas levantadas no novo estado, ele cai, portanto é recompensado negativamente no valor de -20 e o episódio acaba;

- Se o robô mantém a pata no chão e movimenta o quadril desta pata, da frente para trás, ele é recompensado positivamente no valor de +1 para cada pata em que isto ocorre, pois está se deslocando para frente;

- Se o robô mantém a pata no chão e movimenta o quadril desta pata, de trás para frente, ele é recompensado negativamente no valor de -1 para cada pata em que isto ocorre, pois está se deslocando para trás;

- Se o robô mantém a pata levantada e move o quadril desta pata, de trás para frente, ele é recompensado positivamente no valor de +1 para cada pata em que isto ocorre;

- Se o robô mantém a pata levantada e move o quadril desta pata, da frente para trás, ele é recompensado negativamente no valor de -1 para cada pata em que isto ocorre;

- Caso mantenha uma pata no chão e o quadril desta pata não movimentar e, além disso, manter outra pata no chão, com o respectivo quadril movimen-

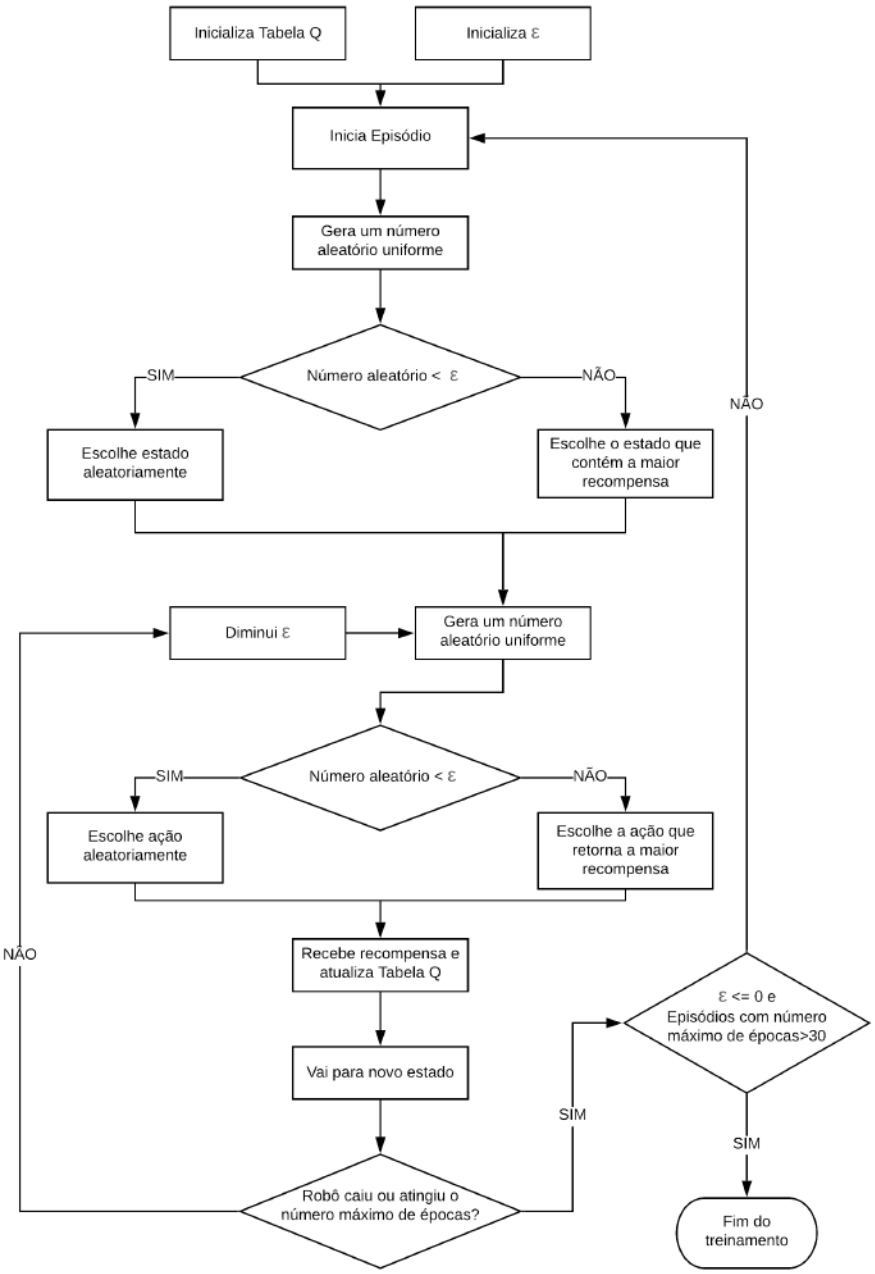
tando de modo a andar para frente, ocorre o arrasto de uma pata no chão, sendo assim, ele é recompensado negativamente no valor de -20.

Os valores de recompensa foram escolhidos arbitrariamente de modo a penalizar muito os casos que realmente comprometem a mobilidade ou integridade do robô, penalizar pouco quando apenas não atinge o objetivo de se locomover para frente e recompensar pouco quando se atinge o objetivo. A recompensa ser pequena tem o propósito de treinar lentamente, para que não seja rapidamente influenciado pelas escolhas aleatórias.

É possível inferir que a Tabela Q pode se tornar extensa ao adicionar mais patas ou mais graus de liberdade a cada pata, já que isso geraria mais estados possíveis, mais ações possíveis e mais regras a serem adicionadas para o cálculo das recompensas. Por isso o robô escolhido foi um quadrúpede com 2 graus de liberdade em cada pata, para diminuir a quantidade de estados e simplificar o problema.

Cada execução do código de treinamento tem como saída um documento ".txt" no formato a ser utilizado como entrada no código do Arduino, visto no Apêndice B, sendo apenas necessário copiar e colar no código e carregá-lo no Arduino para que seja executado pelo robô. Basicamente, cada linha contém um vetor (entre chaves) que, em cada posição, tem um valor binário de 0 ou 1. A posição no vetor é referente ao motor do robô, como no diagrama que aparece na Figura 3.9 e o valor binário indica o estado para qual o servomotor deve ir, como explicado anteriormente na Sessão 3.3. O número seguinte ao "//" é o número inteiro representado pelo binário que constitui os estados dos motores do robô, explicado na Sessão 2.1.4.4. Assim, ao copiar esta saída para dentro de um vetor no código do controlador, é gerada uma matriz. Essa matriz é lida pelo programa para executar cada linha como a sequência de passos a serem executados pelo robô.

Figura 3.10 – Fluxograma do treinamento

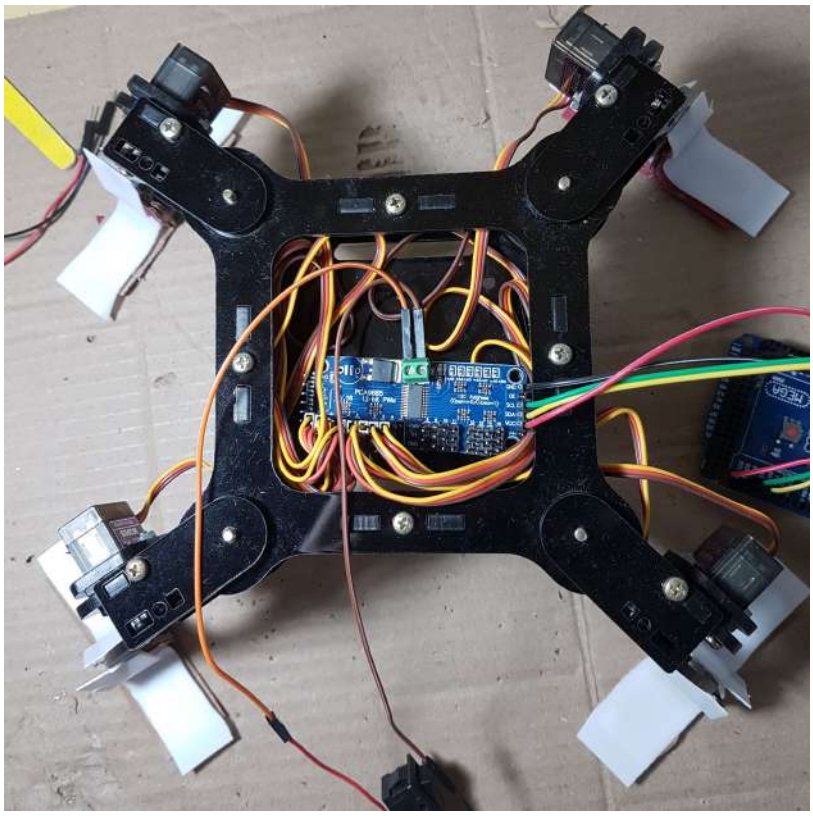


Fonte: do autor 2019

4 RESULTADOS

O robô montado pode ser observado por cima, na Figura 4.1, de frente, na Figura 4.2 e de baixo na Figura 4.3.

Figura 4.1 – Robô em pé visto por cima



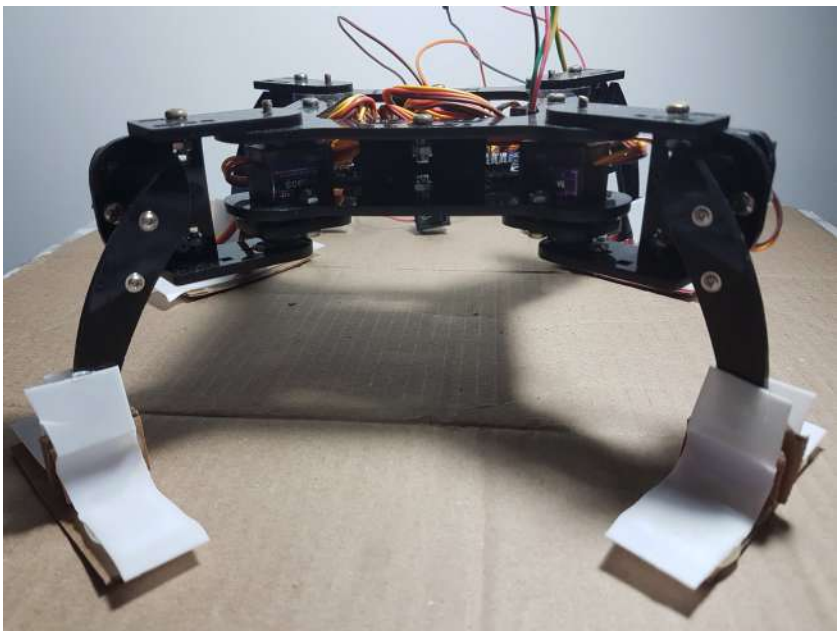
Fonte: do autor 2019

4.1 Execuções do Q-Learning

O código em Python do Apêndice A foi executado dez vezes. Como a exploração é aleatória, o resultado pode variar a cada vez que o código é rodado e alguns resultados diferentes são criados.

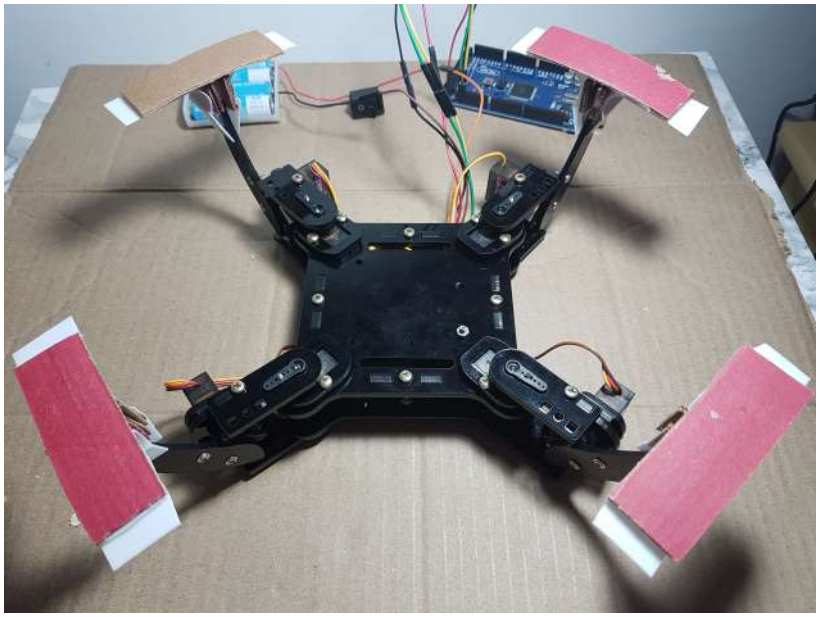
A execução 0, tem sua saída no Quadro 4.1 e os gráficos para melhor entedimento do treinamento nas Figuras 4.4, 4.5, 4.6 e 4.7. A execução 1, tem

Figura 4.2 – Robô visto de frente



Fonte: do autor 2019

Figura 4.3 – Robô visto de baixo



Fonte: do autor 2019

sua saída no Quadro 4.2 e os gráficos para melhor entedimento do treinamento nas Figuras 4.8, 4.9, 4.10 e 4.11. A execução 2, tem sua saída no Quadro 4.3 e os gráficos para melhor entedimento do treinamento nas Figuras 4.12, 4.13, 4.14 e 4.15. A execução 3, tem sua saída no Quadro 4.4 e os gráficos para melhor entedimento do treinamento nas Figuras 4.16, 4.17, 4.18 e 4.19. A execução 4, tem sua saída no Quadro 4.5. A execução 5, tem sua saída no Quadro 4.6. A execução 6, tem sua saída no Quadro 4.7. A execução 7, tem sua saída no Quadro 4.8. A execução 8, tem sua saída no Quadro 4.9. A execução 9, tem sua saída no Quadro 4.10.

Os Quadros 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9 e 4.10 apresentam as saídas de cada treinamento em forma de tabela, em que a coluna "nº"contém o número inteiro que representa cada estado, as colunas de 0 a 7 representam o número da junta em questão vista no diagrama da Figura 3.9 e cada coluna armazena a posição que esta junta deve estar, no formato de bit, como explicado na Sessão 3.3. Cada linha destes Quadros representam um passo ou ação a ser executada pelo robô.

Os Gráficos 4.4, 4.8, 4.12, 4.16, tem em seu eixo vertical o valor de ε e, em seu eixo horizontal, o número do episódio. Os Gráficos 4.5, 4.9, 4.13 4.17, tem em seu eixo vertical o número de passos dados e, em seu eixo horizontal, o número do episódio. Os Gráficos 4.6, 4.10, 4.14, 4.18, tem em seu eixo vertical o valor da média das recompensas recebidas e, em seu eixo horizontal, o número do episódio. Os Graficos 4.7, 4.11, 4.15, 4.19, tem em seu eixo horizontal o número de passos no episódio e em seu eixo vertical a média das recompensas recebidas no episódio.

4.1.1 Discussão

Os resultados do Q-Learning são bastante distintos e alguns chamam atenção. As execuções 1, 8 e 9, vistas nos Quadros 4.2, 4.9 e 4.10 respectivamente,

Quadro 4.1 – Saída da execução 0

nº	0	1	2	3	4	5	6	7
194	1	1	0	0	0	0	1	0
214	1	1	0	1	0	1	1	0
148	1	0	0	1	0	1	0	0
52	0	0	1	1	0	1	0	0
44	0	0	1	0	1	1	0	0
109	0	1	1	0	1	1	0	1
73	0	1	0	0	1	0	0	1
67	0	1	0	0	0	0	1	1

Fonte: do autor 2019

Quadro 4.2 – Saída da execução 1

nº	0	1	2	3	4	5	6	7
56	0	0	1	1	1	0	0	0
121	0	1	1	1	1	0	0	1
97	0	1	1	0	0	0	0	1
193	1	1	0	0	0	0	0	1
131	1	0	0	0	0	0	1	1
151	1	0	0	1	0	1	1	1
22	0	0	0	1	0	1	1	0
28	0	0	0	1	1	1	0	0

Fonte: do autor 2019

Quadro 4.3 – Saída da execução 2

nº	0	1	2	3	4	5	6	7
195	1	1	0	0	0	0	1	1
215	1	1	0	1	0	1	1	1
150	1	0	0	1	0	1	1	0
20	0	0	0	1	0	1	0	0
60	0	0	1	1	1	1	0	0
125	0	1	1	1	1	1	0	1
105	0	1	1	0	1	0	0	1
65	0	1	0	0	0	0	0	1

Fonte: do autor 2019

apresentam o mesmo resultado. As execuções 5 e 7, vistas nos Quadros 4.6 e 4.8 respectivamente, apresentam o mesmo resultado. As execuções 2,3 e 6, vistas nos Quadros 4.3, 4.4 e 4.7 respectivamente, apresentam o mesmo resultado, conside-

Quadro 4.4 – Saída da execução 3

nº	0	1	2	3	4	5	6	7
195	1	1	0	0	0	0	1	1
215	1	1	0	1	0	1	1	1
150	1	0	0	1	0	1	1	0
20	0	0	0	1	0	1	0	0
60	0	0	1	1	1	1	0	0
125	0	1	1	1	1	1	0	1
105	0	1	1	0	1	0	0	1
65	0	1	0	0	0	0	0	1

Fonte: do autor 2019

Quadro 4.5 – Saída da execução 4

nº	0	1	2	3	4	5	6	7
56	0	0	1	1	1	0	0	0
121	0	1	1	1	1	0	0	1
97	0	1	1	0	0	0	0	1
67	0	1	0	0	0	0	1	1
194	1	1	0	0	0	0	1	0
214	1	1	0	1	0	1	1	0
148	1	0	0	1	0	1	0	0
28	0	0	0	1	1	1	0	0

Fonte: do autor 2019

Quadro 4.6 – Saída da execução 5

nº	0	1	2	3	4	5	6	7
60	0	0	1	1	1	1	0	0
125	0	1	1	1	1	1	0	1
105	0	1	1	0	1	0	0	1
65	0	1	0	0	0	0	0	1
195	1	1	0	0	0	0	1	1
130	1	0	0	0	0	0	1	0
150	1	0	0	1	0	1	1	0
20	0	0	0	1	0	1	0	0

Fonte: do autor 2019

rando que é um resultado cíclico, ou seja, a sequência é a mesma, ainda que não se inicie no mesmo valor. O interessante é que todos respeitam as regras impostas e fazem o robô andar ao ser testado fisicamente. Isto implica que o algoritmo

Quadro 4.7 – Saída da execução 6

nº	0	1	2	3	4	5	6	7
60	0	0	1	1	1	1	0	0
125	0	1	1	1	1	1	0	1
105	0	1	1	0	1	0	0	1
65	0	1	0	0	0	0	0	1
195	1	1	0	0	0	0	1	1
215	1	1	0	1	0	1	1	1
150	1	0	0	1	0	1	1	0
20	0	0	0	1	0	1	0	0

Fonte: do autor 2019

Quadro 4.8 – Saída da execução 7

nº	0	1	2	3	4	5	6	7
60	0	0	1	1	1	1	0	0
125	0	1	1	1	1	1	0	1
105	0	1	1	0	1	0	0	1
65	0	1	0	0	0	0	0	1
195	1	1	0	0	0	0	1	1
130	1	0	0	0	0	0	1	0
150	1	0	0	1	0	1	1	0
20	0	0	0	1	0	1	0	0

Fonte: do autor 2019

Quadro 4.9 – Saída da execução 8

nº	0	1	2	3	4	5	6	7
56	0	0	1	1	1	0	0	0
121	0	1	1	1	1	0	0	1
97	0	1	1	0	0	0	0	1
193	1	1	0	0	0	0	0	1
131	1	0	0	0	0	0	1	1
151	1	0	0	1	0	1	1	1
22	0	0	0	1	0	1	1	0
28	0	0	0	1	1	1	0	0

Fonte: do autor 2019

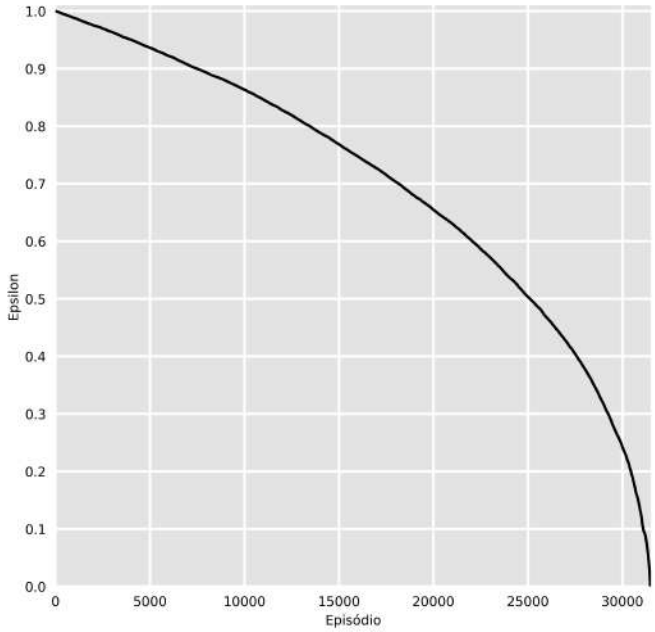
converge e demonstra sua capacidade de aprender a andar. Não são mostrados os gráficos de todas as execuções, pois são extremamente parecidos, adicionando pouco ou nada de conteúdo.

Quadro 4.10 – Saída da execução 9

nº	0	1	2	3	4	5	6	7
56	0	0	1	1	1	0	0	0
121	0	1	1	1	1	0	0	1
97	0	1	1	0	0	0	0	1
193	1	1	0	0	0	0	0	1
131	1	0	0	0	0	0	1	1
151	1	0	0	1	0	1	1	1
22	0	0	0	1	0	1	1	0
28	0	0	0	1	1	1	0	0

Fonte: do autor 2019

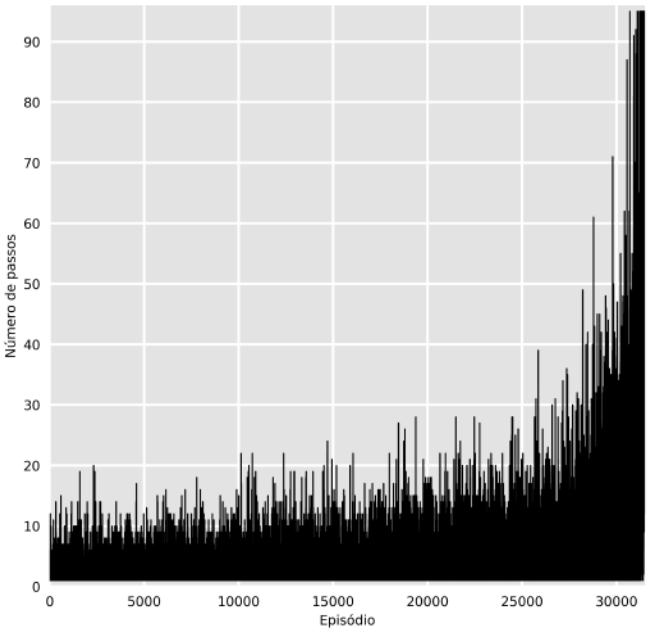
Figura 4.4 – Evolução da desconfiaça ε na execução 0



Fonte: do autor 2019

É importante que os gráficos de evolução da desconfiança ε, vistos nas Figuras 4.4, 4.8, 4.12, 4.16 e os gráficos de número de passos por episódio, vistos dos Gráficos 4.5, 4.9, 4.13 4.17, sejam analisados em conjunto. Para cada época (indicando cada passo), o valor de ε decresce, deste modo, à medida que mais passos são dados em cada episódio, o ε decresce mais rapidamente. Isso fica evidenciado nos dois gráficos, em que, inicialmente poucos passos são dados sem

Figura 4.5 – Número de passos por episódio na execução 0

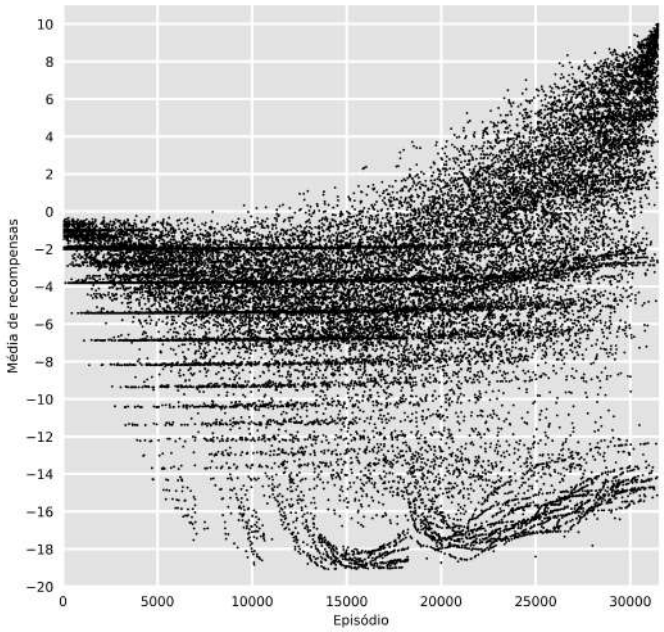


Fonte: do autor 2019

que o robô caia e o epsilon decresce pouco; ao final do treinamento, mais passos são dados sem que o robô caia, implicando em um decréscimo acelerado de ε.

Os gráficos da média das recompensas por episódio, vistos nas Figuras 4.6, 4.10, 4.14, 4.18, mostram bem o funcionamento da Equação 2.1. Espera-se que o robô caia constantemente no início do treinamento, o que faria o ambiente penalizá-lo em -20, como explicado na Sessão 2.1.4.4, porém esse não é o valor que é visto nos primeiros episódios, isso ocorre devido à taxa de aprendizado α e à falta de conhecimento prévio das recompensas futuras. Por isso, soma-se 0 para a maior recompensa do próximo passo, na maioria dos casos iniciais, o que torna o valor αR dominante na equação inicialmente, deixando apenas uma pequena porcentagem da recompensa que o ambiente deu, realmente influenciar na Tabela Q. À medida que a tabela passa a ter mais valores, o termo $\gamma maxQ(S',A')$ tende a ter maior influência, criando uma certa "barriga"no gráfico, que diminui

Figura 4.6 – Média de recompensas por episódio na execução 0

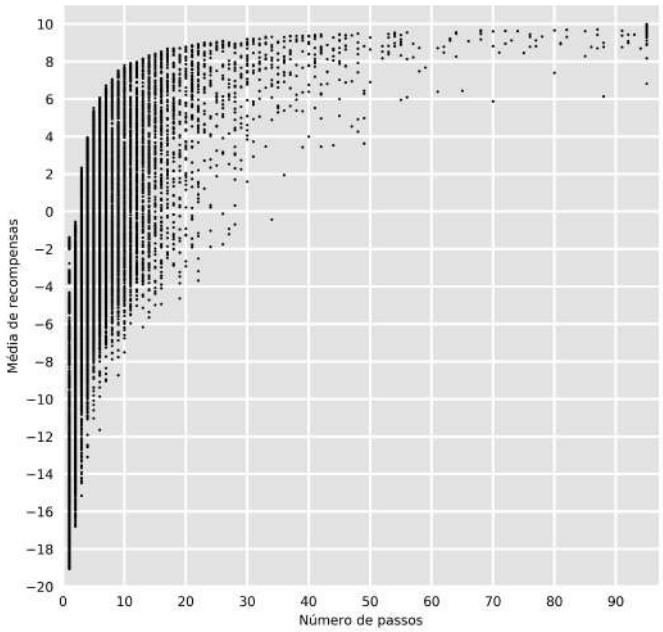


Fonte: do autor 2019

de amplitude à medida que o ε diminui, aumentando o número de passos dados, o que aumenta o valor médio das recompensas recebidas.

Os gráficos de média de recompensas por número de passos, vistos nos Graficos 4.7, 4.11, 4.15, 4.19, mostra como o algoritimo otimiza a recompensa recebida. Fica evidente que, ao aumentar o número de passos, a média das recompensas recebidas é maior. Para um baixo número de passos dados, há uma grande amplitude de valores; ao revisitar os gráficos explicados anteriormente, observa-se que no gráfico da média de recompensas por episódio há uma "faixa" de recompensas bastante baixas, quase que determinando um limite inferior para a média das recompensas; esta faixa se reflete no gráfico da média das recompensas por número de passos, justamente nessa grande amplitude, devido ao fato de que já está bastante treinado e erros são mais penalizados. Isto prova a convergência do algoritmo, em que procurar a maior recompensa também induz um maior número de passos a serem dados.

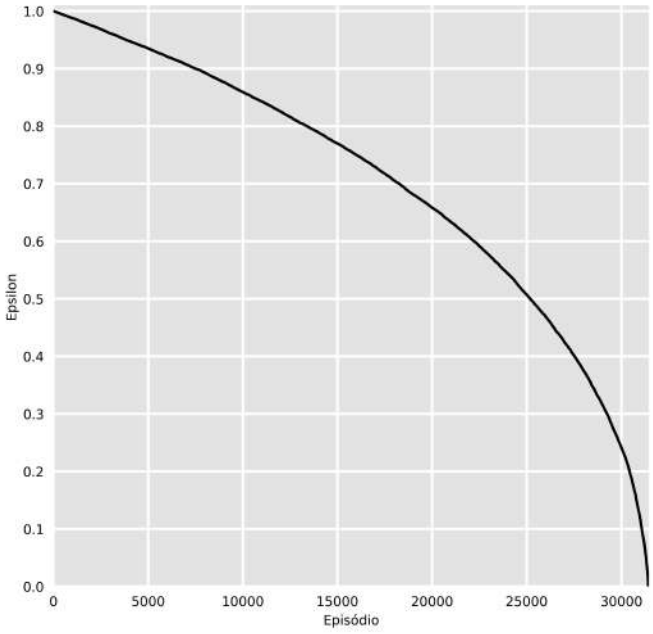
Figura 4.7 – Média de recompensas por número de passos na execução 0



Fonte: do autor 2019

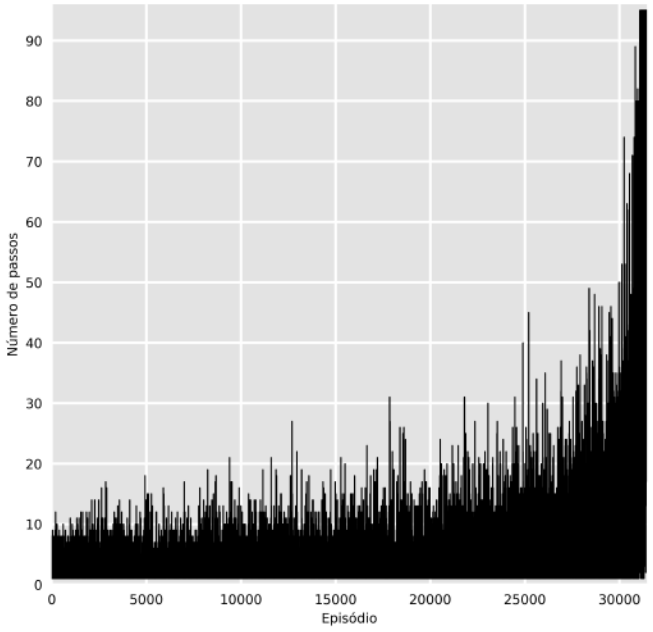
Vale ressaltar que, os pontos onde há o maior número de passos possíveis, não necessariamente refletem a melhor sequência de ações. Justamente por isso o algoritmo leva em consideração os 30 episódios finais, em que executa o maior número de passos possíveis, tendo absoluta confiança na Tabela Q, ou seja, $\varepsilon <= 0$, permitindo que o algoritmo execute exatamente a melhor sequência de ações que foi encontrada.

Figura 4.8 – Evolução da desconfiaça ε na execução 1



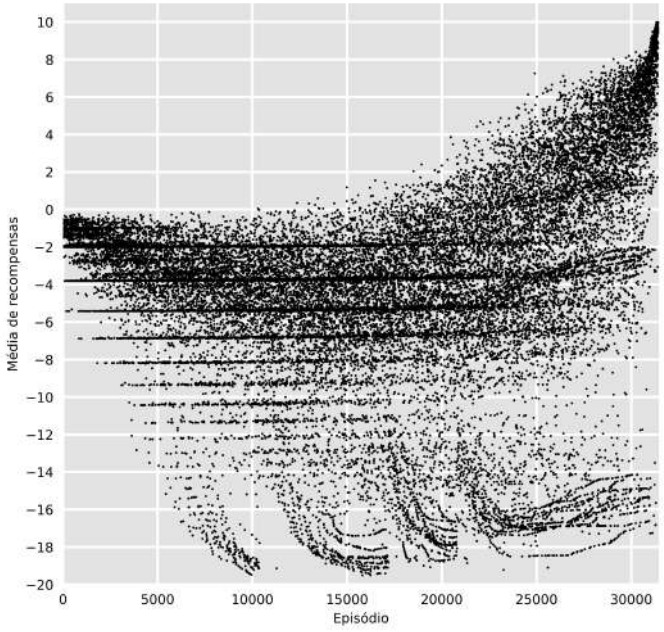
Fonte: do autor 2019

Figura 4.9 – Número de passos por episódio na execução 1



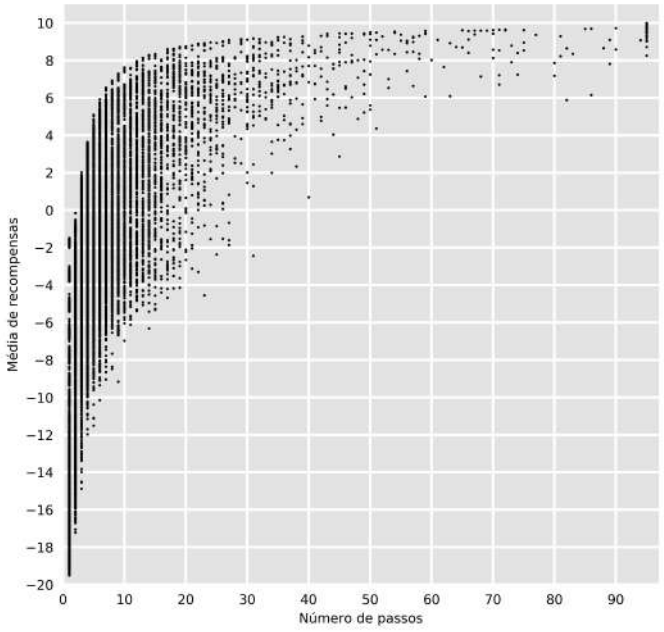
Fonte: do autor 2019

Figura 4.10 – Média de recompensas por episódio na execução 1



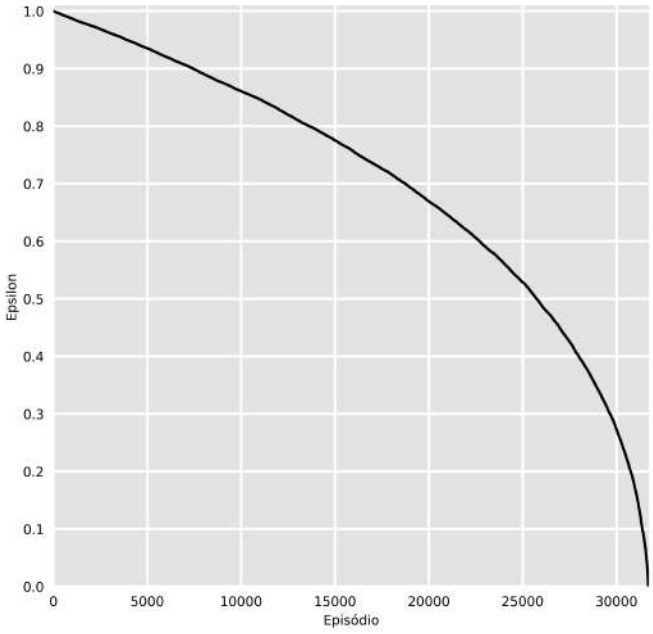
Fonte: do autor 2019

Figura 4.11 – Média de recompensas por número de passos na execução 1



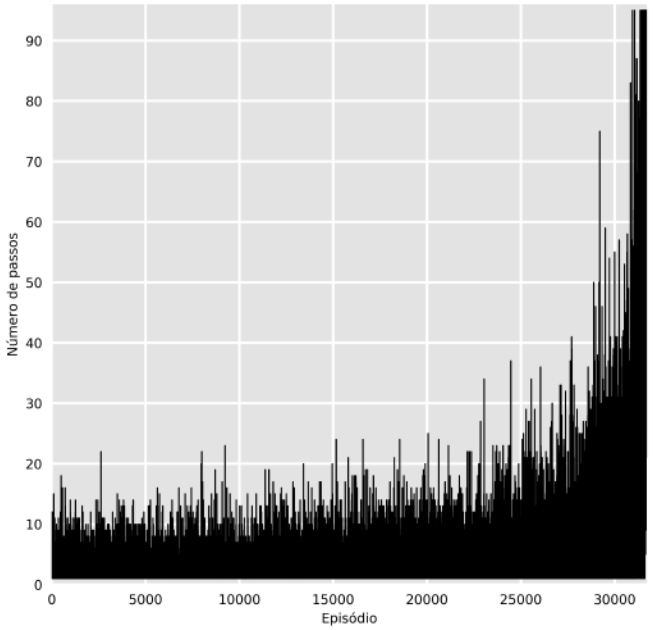
Fonte: do autor 2019

Figura 4.12 – Evolução da desconfiaça ε na execução 2



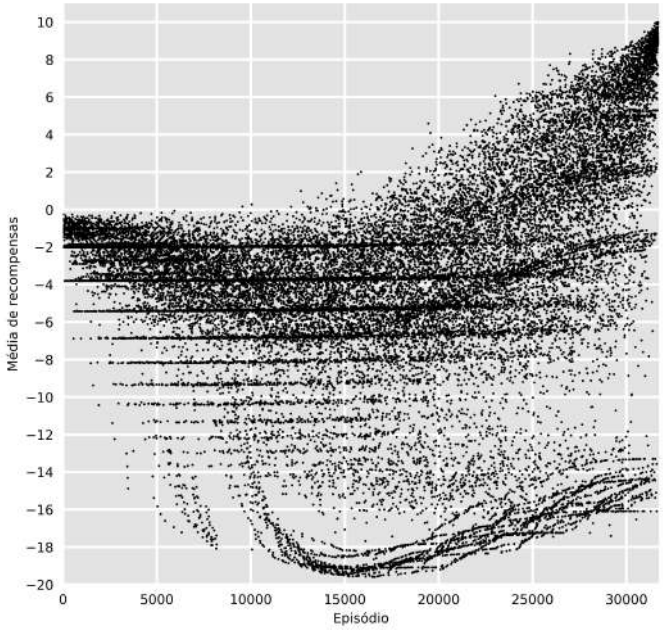
Fonte: do autor 2019

Figura 4.13 – Número de passos por episódio na execução 2



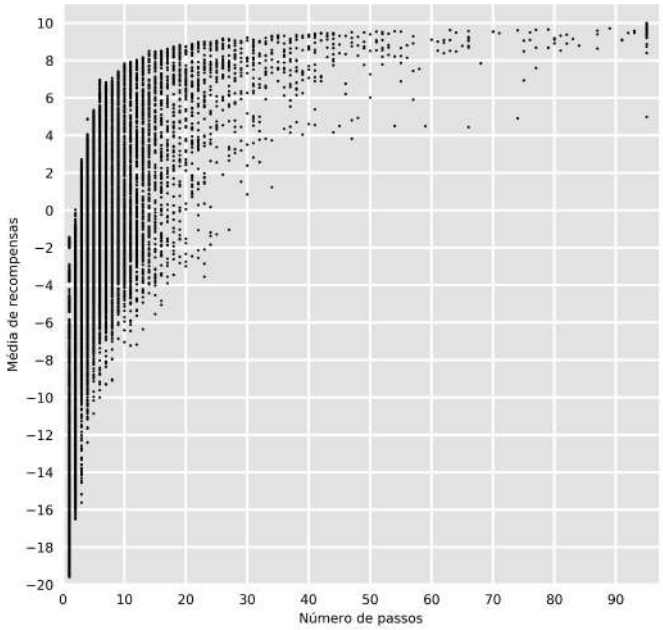
Fonte: do autor 2019

Figura 4.14 – Média de recompensas por episódio na execução 2



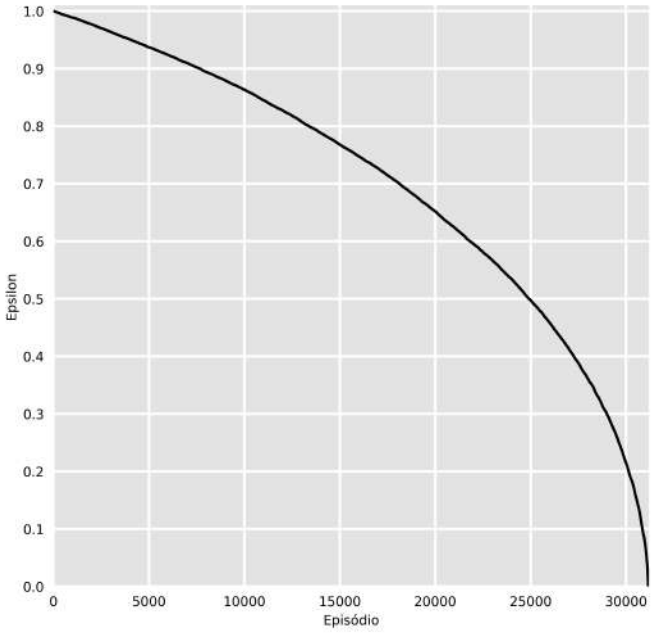
Fonte: do autor 2019

Figura 4.15 – Média de recompensas por número de passos na execução 2



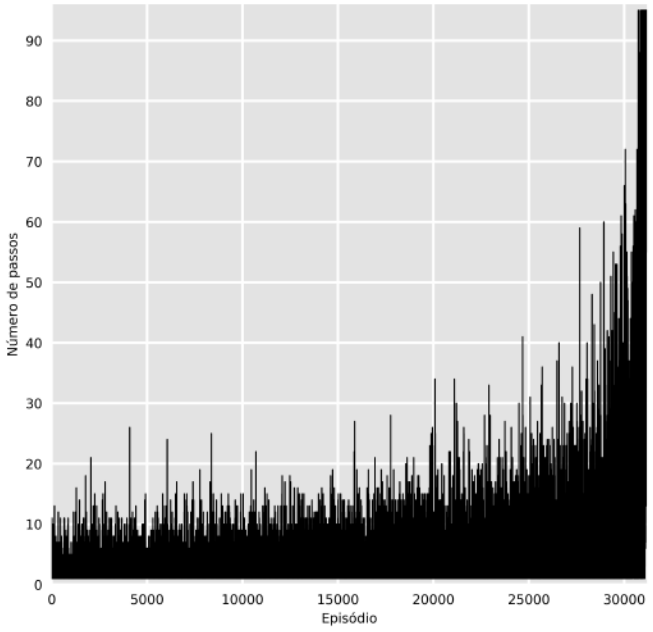
Fonte: do autor 2019

Figura 4.16 – Evolução da desconfiaça ε na execução 3



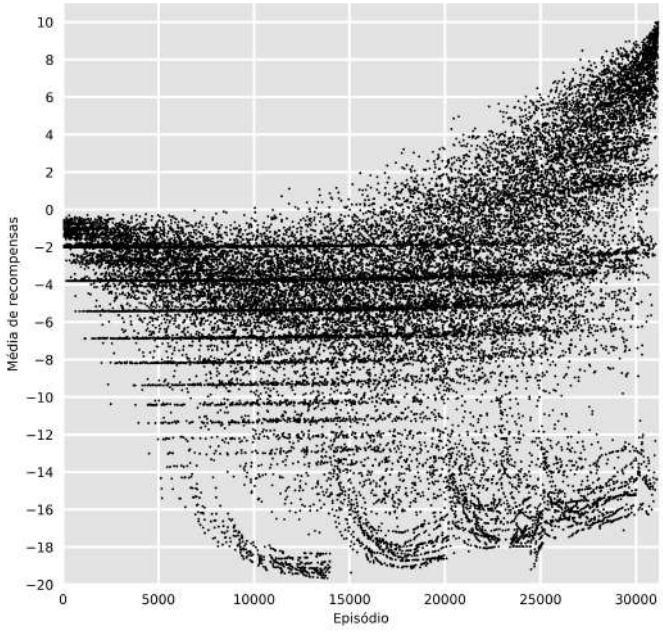
Fonte: do autor 2019

Figura 4.17 – Número de passos por episódio na execução 3



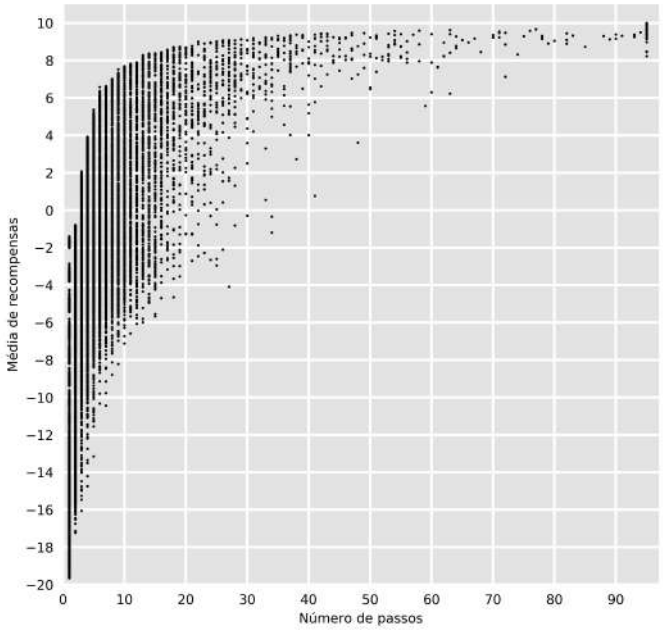
Fonte: do autor 2019

Figura 4.18 – Média de recompensas por episódio na execução 3



Fonte: do autor 2019

Figura 4.19 – Média de recompensas por número de passos na execução 3



Fonte: do autor 2019

5 CONCLUSÃO

O objetivo deste trabalho foi aplicar o método de machine learning por reforço, conhecido como Q-Learning, no treinamento de um robô quadrúpede para que este aprenda a se locomover. O resultado bastante satisfatório deve-se à simplicidade do algoritmo e à simplificação de variáveis complexas. Tal simplificação, que consiste em padronizar os tempos entre cada movimento e as posições dos servomotores, facilita o entendimento do problema.

O treinamento offline, não impediu o algoritmo de convergir, apenas implicou na criação de um ambiente virtual para que haja uma "percepção" da realidade a partir de regras impostas. A vantagem de o algoritmo ser offline é que o poder computacional do processador de computador é bem maior que o poder computacional de um microprocessador, permitindo que o algoritmo termine o processamento muito mais rapidamente. A desvantagem é não ser possível acompanhar e visualizar a evolução do robô ao longo de seu treinamento, apenas o resultado final.

5.1 Trabalhos futuros

Alguns algoritmos mais avançados como o Deep Q-Learning, que interliga Redes Neurais com o Q-Learning, podem eliminar a necessidade de simplificação mas tornam o problema ainda mais complexo.

Uma sugestão para um futuro projeto é a eliminação prévia de estados não viáveis, como os que já detém duas ou mais patas consecutivas levantadas. Caso ocorra uma suficiente redução no tamanho da tabela Q, o treinamento online pode se tornar viável, mostrando fisicamente a evolução ao longo do tempo.

REFERÊNCIAS BIBLIOGRÁFICAS

BAUERMEISTER, G. Como usar o módulo PWM Servo 16 canais. **FilipeFlop**, ago. 2017. Acessado em: 22/10/2019. Disponível em: <https://www.filipeflop.com/blog/como-usar-modulo-pwm-servo-16-canais/>.

BISHOP, C. **Pattern Recognition and MachineLearning**. Cambridge CB3 0FB, U.K.: Springer, 2006. Disponível em: <http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf>.

FILIPEFLOP. Placa MEGA 2560 R3 + Cabo USB para Arduino. out. 2019. Acessado em: 22/10/2019. Disponível em: <https://www.filipeflop.com/produto/placa-mega-2560-r3-cabo-usb-para-arduino/>.

GRIDLING, G.; WEISS, B. **Introduction to Microcontrollers**. 1.4. ed. Vienna University of Technology, Institute of Computer Engineering and Embedded Computing Systems Group, 2007. Disponível em: <https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf>.

HWANGBO, J. et al. Learning agile and dynamic motor skills for legged robots. **Science Robotics**, jan. 2019.

IRAZABAL, J.-M.; BLOZIS, S. **I^2C MANUAL**. [S.l.], 2003. Disponível em: <https://www.nxp.com/docs/en/application-note/AN10216.pdf>.

PIERCE, S. **mePed v2 Assembly Manual**. [S.l.], 2016. Acessado em: 28/10/2019. Disponível em: <http://www.meped.io/sites/default/files/2016-09/mePed_v2_Assembly_Manual_0.pdf>.

PIZZIOLO, T. A. et al. Análise de simplificação na modelagem dinâmica aplicada a robôs quadrúpedes. **Sociedade Brasileira de Automática**, set. 2004. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-17592004000300006&lng=en&tlng=en>.

ROBOCORE. **BRAÇO Robótico RoboARM**. [s.n.], 2019. Acessado em: 23/10/2019. Disponível em: <https://www.robocore.net/loja/kits/braco-robotico-roboarm>.

SANTOS, J. L. d.; JúNIOR, C. L. N. Coordenação dos atuadores das pernas de robôs móveis usando aprendizado por reforço: simulação e implementação. **Sociedade Brasileira de Automática**, fev. 2012. Disponível em: <http://www.scielo.br/scielo.php?frbrVersion=2&script=sci_arttext&pid=S0103-17592012000100007&lng=en&tlng=en>.

SELVATICI, A. H. P.; COSTA, A. H. R. Aprendizado da coordenação de comportamentos primitivos para robôs móveis. **Sociedade Brasileira de**

Automática, jun. 2007. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-17592007000200004&lng=en&tlng=en>.

SILVEIRA, M. A. **Estudo, Montagem e Controle de um Robô Móvel Hexápode**. Lavras - MG: UFLA, 2018.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 2. ed. Cambridge, Massachusetts: The MIT Press, 2017. Disponível em: <http://incompleteideas.net/book/bookdraft2017nov5.pdf>.

TOWERPRO. **Tower Pro MG90S**. [S.l.], 2014. Acessado em: 12/10/2019. Disponível em: <https://www.towerpro.com.tw/product/mg90s-3/>.

VIOLANTE, A. Simple Reinforcement Learning: Q-learning. **freeCodeCamp**, mar. 2018. Acessado em: 10/10/2019. Disponível em: <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56>.

APÊNDICE A – Código de treinamento em python

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 12 19:06:39 2019

@author: durval
"""
import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
import itertools

def calculateReward(prevstep,newstep,totalmotors):
    # Generates the bit value of the state number
    prevstep = ('{0:0'+str(totalmotors)+'b}').format(prevstep)
    newstep = ('{0:0'+str(totalmotors)+'b}').format(newstep)

    # Sets the values to better understanging of the logic
    # A hip is either to the front or to the back
    # A feet is either up or down
    front = down = '0'
    back = up = '1'
    reward=[0]

    # Sets to continue the episode
    continuar = True

    # The following number to the member of the robot
    # 'lbf','lbh','lff','lfh','rbf','rbh','rff','rfh'
    #   0  ,  1  ,  2  ,  3  ,  4  ,  5  ,  6  ,  7

    # If there are any two consecutive feet up,
    # the robot will fall
    if ((newstep[0]==newstep[2]==up) or
        (newstep[2]==newstep[6]==up) or
        (newstep[4]==newstep[6]==up) or
        (newstep[0]==newstep[4]==up)):
        continuar = False # End of episode
        reward = -20
        return [reward,continuar]

    hipmoves = 0
    drag = 0
    for member in range(0,8,2):
        if prevstep[member]==newstep[member]==down:
            # From the front to back pushing the robot forwards
            if (prevstep[member+1]==front
                and
                newstep[member+1]==back):
                reward.append(1)  # Going forwards
                hipmoves+=1
            # From the back to front pushing the robot backwards
            if (prevstep[member+1]==back and
                newstep[member+1]==front):
                reward.append(-1) # Going backwards
            # Hip does not move, dragging the feet on the ground
            if (prevstep[member+1]==newstep[member+1]):
                drag+=1
        if prevstep[member]==newstep[member]==up:
            if (prevstep[member+1]==back and
                newstep[member+1]==front):
                reward.append(1)  # Going forwards
```

```
            if (prevstep[member+1]==front and
                newstep[member+1]==back):
                reward.append(-1) # Going backwards

    if (drag>0 and hipmoves>0):
    # Penalizes for dragging the feet on the ground
        reward.append(-20)

    reward = sum(reward)
    return [reward,continuar]

def act(state,action,totalmotors):
    state = ('{0:0'+str(totalmotors)+'b}').format(state)
    for move in action:
        move = int(move)
        # Changes the state of that motor
        if state[move]=='0':
            # Changes the state from 0 to 1
            state = state[:move]+'1'+state[move+1:]
        else:
            # Changes the state from 1 to 0
            state = state[:move]+'0'+state[move+1:]
    # Calculates the value of the new state
    state = int(state,2)
    return state

def Qupdate(qtable,state,action,lr,gamma,totalmotors):
    # Gets the state after doing a certain action
    newstate = act(state,action,totalmotors)
    # Calculates the reward given for this newstate
    [reward,continuar] = calculateReward(state,
                                         newstate,
                                         totalmotors)
    # Calculates the expected reward
    # after given newstate is done
    expected = qtable.loc[newstate,:].max()
    # Calculates the update value
    update = lr*(reward+(gamma*expected)-qtable.loc[
                                             state,
                                             action])
    # Updates the Q-value
    reward = qtable.loc[state,action]+update
    qtable.loc[state,action] = reward
    return [newstate,continuar,reward]

def qlearn(qtable,state,lr,gamma,epsilon,totalmotors):
    # Exploration or exploitation
    if (random.uniform(0,1)<epsilon):
        # Exploration
        action = random.choice(qtable.columns)
    else:
        # Exploitation
        action = qtable.loc[state,:].idxmax()

    return Qupdate(qtable,
                   state,
                   action,
                   lr,
                   gamma,
                   totalmotors
                   )

def reset(qtable,epsilon):
    # Resets the robgot and now will try a new approach
```

```
    if (random.uniform(0,1)<epsilon):
        # New approach is random
        state = random.choice(qtable.index)
    else:
        # New approach is the best starting
        # reward for the first action
        maximum =  qtable[qtable==qtable.max().max()]
        maximum.dropna(axis='index',how='all',inplace=True)
        maximum.dropna(axis='columns',how='all',inplace=True)
        maximum = maximum.idxmax(axis='columns')

        # Selects the action(s) that gives the best reward
        try:
            action = str(random.choice(maximum.values))
        except:
            action = str(maximum)
        # Selects the state that will most likely
        # result in that action
        try:
            state = random.choice(
                    qtable.loc[:,action].idxmax().values
                    )
        except:
            state = qtable.loc[:,action].idxmax()
    return state

def looping(path):
    return pd.Series(
                data=list(
                dict.fromkeys(path)
                ),
                name='state'
                )

if __name__=='__main__':
    run = 9
    nummotors = 2
    totalmotors = 8
    legs = ['lbf','lbh','lff','lfh','rbf','rbh','rff','rfh']
    legsnum = [x for x in range(totalmotors)]
    sets = []
    for subset in itertools.combinations(legsnum,nummotors):
        s = ''
        for sub in subset:
            s+=str(sub)
        sets.append(s)

    # All possible states permutations o 2 states
    # for each of the 8 motors 2^8
    states = [x for x in range(256)]

    # Initializes the Q-Table
    qtable = pd.DataFrame(data=0,index=states,columns=sets)
    # Number of episodes
    episodes = 100000
    # Number of epochs
    epochs = int((totalmotors*(totalmotors*3)/nummotors)-1)
    # Untrustfulness in the Q-table
    epsilon = 1.0
    # Keeping track of the evolution of trust
    epsilons = [epsilon]
    # Learning rate
    lr = 0.1
    # Discount factor
```

```
gamma = 0.9

# Saves the failing of the episode
continues = []
# Saves the paths made
paths = []
# Saves the last path
path = []
# Saves rewards
rewards = []
r = []
count = 0
print("Training...")
for episode in range(episodes):
    print("Starting episode {}:".format(episode))
    #Resets to a starting position
    state = reset(qtable,epsilon)
    path = [state]
    reward = []
    for epoch in range(epochs):
        # newstate done becomes the new state
        [state,continuar,rewardaux] = qlearn(
                                        qtable,
                                        state,
                                        lr,
                                        gamma,
                                        epsilon,
                                        totalmotors
                                        )
        path.append(state)
        reward.append(rewardaux)
        if continuar:
            if epsilon>0:
                epsilon-=0.00001
        else:
            # A well treined robot, should not fall
            # Exploring more
            if epsilon<0.1:
                epsilon = epsilon+0.0001
            print(
"Robot fallen in epoch {} with epsilon: {}".format(
                                            epoch,
                                            epsilon
                                            )

                 )
            break
    loop = looping(path)
    epsilons.append(epsilon)
    continues.append(int(continuar))
    paths.append(path)
    rewards.append(np.mean(reward))
    r.append(reward)
    if (epsilon<=0):
        count+=1
    else:
        count=0
    if count>30 and rewards[-1]>0:
        break
fig,ax = plt.subplots(figsize=[8,8])
ax.plot(epsilons,linewidth=2,c='black')
ax.set_facecolor('#e3e3e3')
ax.spines["top"].set_visible(False)
ax.spines["bottom"].set_visible(False)
ax.spines["right"].set_visible(False)
```

```
ax.spines["left"].set_visible(False)
plt.yticks(np.arange(0,1.1,0.1))
plt.tick_params(axis="both",
                which="both",
                bottom="off",
                top="off",
                labelbottom="on",
                left="off",
                right="off",
                labelleft="on"
                )
plt.xlabel('Episódio')
plt.ylabel('Epsilon')
plt.xlim(0,len(epsilons))
plt.ylim(0,max(epsilons)+0.01)
plt.grid(True,linewidth=2,c='white')
plt.savefig('Results/Epsilons{}.png'.format(run),
            dpi=500,
            bbox_inches='tight')
plt.close()

psize = [len(p)-1 for p in paths]

fig,ax = plt.subplots(figsize=[8,8])
ax.plot(psize,linewidth=1,c='black')
ax.set_facecolor('#e3e3e3')
ax.spines["top"].set_visible(False)
ax.spines["bottom"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
plt.yticks(np.arange(0,epochs+1,10))
plt.tick_params(axis="both",
                which="both",
                bottom="off",
                top="off",
                labelbottom="on",
                left="off",
                right="off",
                labelleft="on")
plt.xlabel('Episódio')
plt.ylabel('Número de passos')
plt.xlim(0,len(psize))
plt.ylim(0,max(psize)+1)
plt.grid(True,linewidth=2,c='white')
plt.savefig('Results/PathSize{}.png'.format(run),
            dpi=500,
            bbox_inches='tight')
plt.close()

fig,ax = plt.subplots(figsize=[8,8])
ax.plot(rewards,'.',c='black',markersize=1.5)
ax.set_facecolor('#e3e3e3')
ax.spines["top"].set_visible(False)
ax.spines["bottom"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
plt.yticks(np.arange(-20,15,2))
plt.tick_params(axis="both",
                which="both",
                bottom="off",
                top="off",
                labelbottom="on",
                left="off",
                right="off",
```

```
                labelleft="on")
plt.xlabel('Episódio')
plt.ylabel('Média de recompensas')
plt.xlim(0,len(rewards))
plt.ylim(-20,max(rewards)+1)
plt.grid(True,linewidth=2,c='white')
plt.savefig('Results/MeanReward{}.png'.format(run),
            dpi=500,
            bbox_inches='tight')
plt.close()

fig,ax = plt.subplots(figsize=[8,8])
ax.plot(psize,rewards,'.',c='black',markersize=2)
ax.set_facecolor('#e3e3e3')
ax.spines["top"].set_visible(False)
ax.spines["bottom"].set_visible(False)
ax.spines["right"].set_visible(False)
ax.spines["left"].set_visible(False)
plt.yticks(np.arange(-20,15,2))
plt.xticks(np.arange(0,100,10))
plt.tick_params(axis="both",
                which="both",
                bottom="off",
                top="off",
                labelbottom="on",
                left="off",
                right="off",
                labelleft="on")
plt.ylabel('Média de recompensas')
plt.xlabel('Número de passos')
plt.ylim(-20,max(rewards)+1)
plt.xlim(0,max(psize)+2)
plt.grid(True,linewidth=2,c='white')
plt.savefig('Results/MRxP{}.png'.format(run),
            dpi=500,
            bbox_inches='tight')
plt.close()

qtable.to_csv('Results/Q-table{}.csv'.format(run))

#To create the vectors for Arduino
x=[]
for i in loop.values:
    l = ('{0:0'+str(totalmotors)+'b}').format(i)
    laux = ''
    for k in l:
        laux+=(k+',')
    x.append(("{"+str(laux)+"}, //"+
                str(i)).replace(',}','}'))
pd.Series(
          data=x,
          name='steps'
          ).to_csv(
                  'Results/steps{}.txt'.format(run),
                  sep=';',
                  header=False,
                  index=False)
```

APÊNDICE B – Código do controlador do robô

```
/*
Autor: Durval
Data de criação: 10/09/2019
Compilador: Arduino 1.8.9
*/
#include "HCPCA9685.h"
#define servonum 8
#define lbf 0 //left back feet
#define lbh 1 //left back hip
#define lff 2 //left front feet
#define lfh 3 //left front hip
#define rbf 4 //right back feet
#define rbh 5 //right back hip
#define rff 6 //right front feet
#define rfh 7 //right front hip

#define I2CAdd 0x40 // Default address of the PCA9685 Module
// Uses the default address 0x40

HCPCA9685 HCPCA9685(I2CAdd);
                        //lbf,lbh,lff,lfh,rbf,rbh,rff,rfh
//down or front
unsigned int servomin[] = {240,170,240,170,  0, 70,  0, 70};
//up or back
unsigned int servomax[] = {200, 70,200, 70, 40,170, 40,170};

int T = 100;

void walk(int steps[servonum],int T){
  for (int servo=0;servo<servonum;servo++){
    if (steps[servo]==1){
      HCPCA9685.Servo(servo,servomax[servo]);
    }
    if (steps[servo]==0){
      HCPCA9685.Servo(servo,servomin[servo]);
    }
    delay(T);
  }
}

void setup() {
  Serial.begin(9600);
  /* Initialise the library and set it to 'servo mode' */
  HCPCA9685.Init(SERVO_MODE);
  /* Wake the device up */
  HCPCA9685.Sleep(false);
  delay(1000);
  T = 100;
  int walking[servonum];
  for (int servo=0;servo<servonum;servo++){
    if((servo%2)==0){
      walking[servo] = 0;
    } else{
      walking[servo] = 1;
    }
  }
  walk(walking,T);
  T=10;
}

int walking[][servonum] = {
{0,0,1,1,1,1,0,0}, //60
```

```
{0,1,1,1,1,1,0,1}, //125
{0,1,1,0,1,0,0,1}, //105
{0,1,0,0,0,0,0,1}, //65
{1,1,0,0,0,0,1,1}, //195
{1,0,0,0,0,0,1,0}, //130
{1,0,0,1,0,1,1,0}, //150
{0,0,0,1,0,1,0,0}, //20
};

int steps = (sizeof(walking)/sizeof(walking[0]));
int tdelay = 300;
void loop() {
  for (int i=0;i<steps;i++){
    walk(walking[i],T);
    delay(tdelay);
  }
}
```