Racing Hexapods: Q-learning vs Uninformed Search

William Bryk

December 12, 2016

1 Introduction

Legged robots have several advantages over their wheeled counterparts, including navigating rugged terrain and avoiding obstacles. Finding highly adaptable leg algorithms would benefit a wide range of disciplines, from disaster relief to helping around the house. However, although legged animals move with ease, humans have struggled to find efficient algorithms for 4-legged (quadruped) and 6-legged (hexapod) robotic creatures, among others, given an arbitrary shape, orientation, and weight distribution. In order to find the most efficient walking algorithm, a robot self-learning approach is often more practical than a programmer hardcoding joint movements.

The goal of this project was to test how quickly and efficiently a hexapod could learn to walk. Unlike other methods of teaching hexapods found in the literature, such as genetic algorithms, neural networks, or reinforcement learning of a parameter space, this project focused on Q-learning of a state, action space. The hexapod used a Q-learning algorithm to balance exploration of new movements and exploitation of the movements that were successful.

What distinguishes this project from others that discuss reinforcement learning in the literature is the methodology with which the reinforcement learning was defined. The biggest difficulty in finding efficient gaits for hexapods is the sheer number of possible movements. With 3 servos on 6 legs, there are 18 degrees of freedom, which gives an exponential number of possible gaits. The purpose of this project was to develop a new method of decreasing this state space, such that direct reinforcement learning was not just manageable, but better than simple search methods.

The Q-learning hexapod raced against other hexapods that employed uninformed search methods of walking such as greedy search and uniform cost search. In this way, the Q-learning algorithm was given an objective to beat.

2 Background and Related Work

A vast array of machine learning algorithms have been applied to the quadruped and hexapod locomotion problem. An examination of the hexapod-related literature reveals all sorts of algorithm types, from evolving genetic algorithms to find the fittest gait parameters (1), to neural networks that contain layers of nodes representing mathematical functions (2). Reinforcement learning is another popular type of category of algorithm to solve this problem.

Within the reinforcement learning category, there are a number of different approaches to solving the large dimensionality of the problem. Many of the methods employ reinforcement learning

not on the state space that represents the physical state of the legs, but instead on parameters related to the gait, such as using policy gradient reinforcement learning of a parameter space, where the parameters include details such as the fraction of time the feet are on the ground and the shape of the loci the feet can move through (3). Other methods try to use reinforcement learning directly on the positions of the joints, but reduce the state space by discretizing the possible joint positions of the legs (4). Others have tried to select beforehand possible configurations of the leg in relation to the body, and use the vector of these positions for each leg as the state space (5).

What I did not encounter in the literature however, is an approach that focuses on Q-Learning over the feet positions with respect to the leg's symmetric home position.

3 Problem Specification

The problem this project addresses is the speed and efficiency with which Q-Learning can teach a hexapod to walk. While some evolutionary algorithms or deep neural networks require a significant amount of training, in the real world, it is necessary to learn quickly and adapt to new environments without the need for thousands of hours of training. In more testable, formal terms, the problem is whether a hexapod can be trained within a few hundred steps to perform equally to a Hexapod agent using a reasonable walking algorithm, such as always taking the action with highest reward.

4 Approach

Despite the use of normal Q-Learning, what was developed in this study was a completely new approach to the hexapod problem of dimensionality. Using the techniques presented in this paper, a three-dimensional problem was translated into a two-dimensional problem with more constraints, that led to a small enough state space to Q-learn very fast. Whereas in other studies, thousands (3) (4) or even hundreds of thousands of simulations of learning were used (5), the hexapod in this project performed on par with greedy search algorithms often in less than 500 steps.

To model the hexapod, the states were vectors $(p_1, p_2, p_3, p_4, p_5, p_6)$ of the locations of the six feet relative to their specific home position – not the six leg orientations as other studies deal with (meaning, 6*3 degrees of freedom were brought down to 6 * 2). Home position of a foot is defined as the foot's relative position to the base during the starting position of the hexapod, when the feet are 60 degrees apart, and at a certain user-defined distance from the center. p_1 through p_6 are integers that encode an (x, y) relative displacement from foot home position (there is a possibleStates array, and these integers are the indices). Not all (x, y) displacements are possible, or else the state space would be infinitely large. Instead, the possible displacements that are allowed are a discretized square grid of possible spaces, where the leg home position is in the center. The user can change the number of divisions per side length of the square grid (this brings the 6*2 degrees of freedom down to only 6*1).

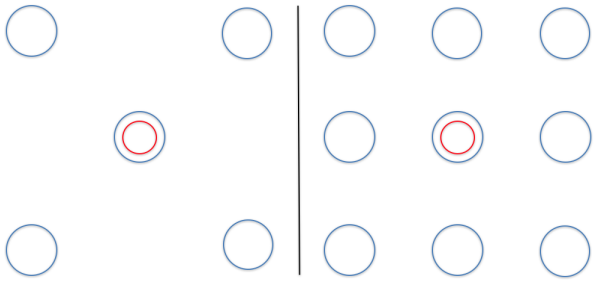

Figure 1: Home Position (red), and other possible states (blue). On the left, side division is 2, on the right side division is 3. 2 is used for the experiments in this project, because the state space is much smaller for 2

An action moves the feet of two symmetric legs to another one of the possible (x,y) options deterministically. If leg 1 moves to (x,y), then its symmetric leg will move to (-x, y). The state space is designed to ensure that (-x,y) option will always be a legal state (this brings the 6 degrees of freedom down to only 3). The symmetry for a given hexapod is determined by the particular walking gait. This project studies the results of four walking gaits, defined by the orientation of the robot and the symmetric relationship between the legs. The orientation can either have 3 legs on the left and three legs on the right, or three legs in front and three legs in the back. The symmetry can either be with respect to the origin or respect to the vertical.

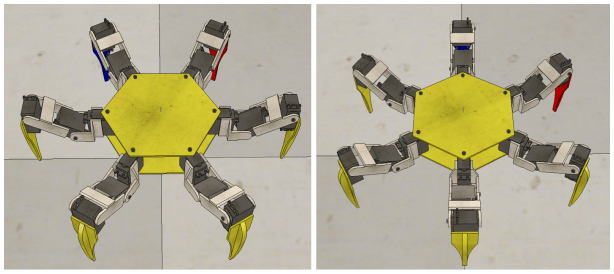

Figure 2: Four walking gaits. Two orientations of the robot – LeftRight (left image) and FrontBack (right image), and for each one, the legs move symmetrically either with respect to the origin or with respect to the vertical. Gait Names: 1. LeftRight-Origin, 2. LeftRight-Vertical, 3. FrontBack-Origin, 4. FrontBack-Vertical

The hexapod is always tested by moving forward. To ensure that the robot knows which way is forward with respect to its own frame, the user must define a leg that is on the right of the center and one that is on the left of the center. This choice decides whether the gait is LeftRight or FrontBack. The forward direction is in the middle. For example, the red leg in Figure 2 is leg 1 and the blue is leg 2. To define the forward direction for the LeftRightOrigin or LeftRightVertical walking gait, the right leg is leg 1 and the left leg is leg 2. If leg 1 is defined as the right leg and leg 3 the left, then you get the FrontBack gaits instead. All displacement vectors, action vectors, and state vectors, are then defined in this robot reference frame.

Q-Learning was done on the state-action space using an epsilon-greedy policy, where epsilon decays linearly with each episode of training. The data structure was a dictionary that mapped

a string representing the state and action to a value. The dictionary had functions that made it similar to the counter dictionaries we used in class where values initialize to zero when keys are declared. The reward function was defined as the displacement in the forward direction. The robot cannot have displacement to the side, because of the symmetric movements of the legs.

For the uninformed search, a greedy algorithm (one step ahead) and two step ahead search were used. The greedy hexapod agent simulates each possible next action, and then chooses the action that gives the greatest reward. The two-steps ahead agent simulates all the next possible two actions, and chooses the first action of the pair that gives the highest reward. Thus greedy search has knowledge of the simulation, and two-steps ahead has better knowledge. This is what makes Q-Learning agent's ability to compete using blind learning all the more impressive.

To simulate the hexapod's movements, the robot simulator V-REP was used, with code written in LUA programming language. A framework of functions was set up so that the user can simply input which leg and which new state it should move to, and the functions will move it to that new position in the simulation according to the correct reference frame. Additionally, every time after legs move, a function is called to move the center of the hexapod base to the center of the new feet positions. This way the center of mass follows the feet, and Q-Learning need only focus on the feet positions. When given a new position to move a foot of one of the legs, V-REP's Inverse Kinematics functions determine the required joint movements of the leg. In order to move the foot from (x,y) position to another, a helper function was written that would move the foot to the halfway point between the current position and the newposition with some stepHeight in the z direction, and then the foot position would move to the new position. This setup is all that is needed to reduce the problem of hexapod movement in 3D space to a two-dimensional problem more approachable to Q-Learning methods.

Algorithm 1 Q-Learn

```
procedure OBSERVEEPISODES()
    for episodes = 1, NUMEPISODES, 1 do
        decayEpsilon(episodes, NUMEPISODES)
        for steps = 1,STEPSPEREPISODE,1 do
            state = getState()
            action = getAction()
            reward = getReward(state, action)
            nextState = getState()
            sample = reward + DISCOUNT * computeValueFromQValues(nextState)
            QValues[state,action] = (1 - ALPHA) * getQValue(state, action) + ALPHA * sample
end procedure
```

Algorithm 2 Move according to QValues

```
procedure MOVEACCORDINGTOQVALUES()
    for i = 1,NUMRACESTEPS,1
        action = getAction()
        leg = action[1]
        actionIndex = action[2]
        actionVector = getActionRelLeg(leg, actionIndex)
        moveLeg(legTargets[leg], stepHeight, actionVector[1], actionVector[2])
        moveLeg(legTargets[legPairs[leg]], stepHeight , -actionVector[1], actionVector[2])
end procedure
```

Algorithm 3 Greedy algorithm.

```
procedure GREEDYSEARCH(state)
    rewardsFromActions = getNextStatesAndRewards(state)
    saveBestAction = getIndexOfMaxReward(rewardsFromActions)
    actions = getLegalActions(state)
    action = actions[saveBestAction]
    return action
end procedure
```

Algorithm 4 UCS2Generation algorithm

```
procedure UCS2GENSEARCH(state)
    rewardsFromActions = getNextStatesAndRewards(state)
    for nextState in rewardsFromActions do
        rewardsFromActions2 = getNextStatesAndRewards(nextState)
        saveBestActionArray.append( getMaxReward(rewardsFromActions2))
    saveBestAction = getIndexofMaxSumRewards(saveBestActionArray, rewardsFromActions)
    actions = getLegalActions(state)
    action = actions[saveBestAction]
    return action
end procedure
```

5 Experiments

For the race between the Q-learning Agent, Greedy Search Agent, and Two-Steps Ahead Agent, a race track was built to better visualize who's winning. Each hexapod has its own particular code that makes it follow its own algorithm. Each hexapod takes 50 steps (where a step is actually two symmetric legs moving at the same time). The total distance the agents travelled was recorded, as well as the distance at which the first agent finishes the 50 steps (some agents take longer to finish calculations). This race was done for four gates, after training the Q-learning agent for both 500 steps, and 1000 steps. 2 divisions were used for the size of the state space – meaning, there are 5 states, the 4 corners of the square around the leg's home position and the leg home position itself.

Additionally, it was important to check how each walking gait changes as the number of train-

ing trials spent learning grows. For each walking gait, the hexapod Qlearns for 250 steps and then saves its Qvalues into a text file. Then it Qlearns for 250 more steps using the saved Qvalues and updates the Qvalues in the text file. It does this until it has taken a total of 2000 steps. Again, 2 divisions were used for the size of the state space for these simulations.

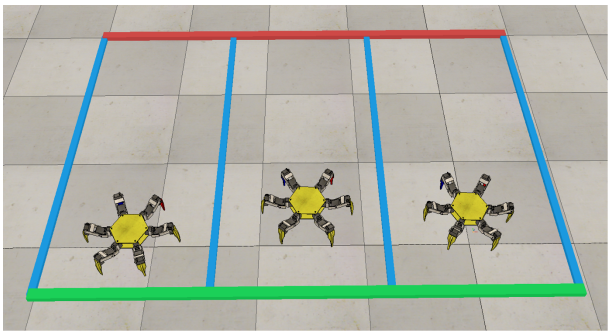

Figure 3: Race setup. Green line is starting line, red line is finish. From right to left: Q-learning agent, Greedy Agent, Two-Steps Agent

Table 1: Racing Data

Gait Type	Distance Type	Q-500 trials	Q-1000 trials	Greedy Agent	Two-Steps Agent
LeftRight-Origin	Finished Distance	.25	.325	.375	.04
LeftRight-Origin	Total Distance	.25	.325	.525	Unfinished
LeftRight-Vertical	Finished Distance	.05	.15	.425	.075
LeftRight-Vertical	Total Distance	.05	.15	.525	Unfinished
FrontBack-Origin	Finished Distance	.075	.375	.35	.05
FrontBack-Origin	Total Distance	.075	.375	.475	Unfinished
FrontBack-Vertical	Finished Distance	.275	.275	.325	.05
FrontBack-Vertical	Total Distance	.275	.275	.475	Unfinished

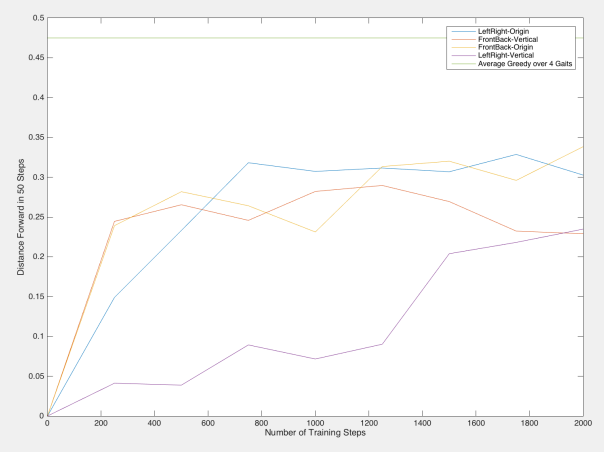

Figure 4: Distance Travelled After 50 Steps vs Number of Previous Trial Steps

5.1 Results

For the race, two distances were measured, with units of .5 per square length within the grid. The first is the distance that all agents have gone when the first agent finishes the 50 steps – always the Q-learning agent, since its calculations are faster. The second is the total distance the agent travels in 50 steps (the Two-Steps Ahead Agent is too slow to wait for so it is recorded as unfinished). The motivation for two different distance measurements is that the hexapods are taking steps at different rates in real time, so it's more practical to judge them on two metrics. Since looking up Q-Learned tables is quicker than Greedy simulating all next actions, the difference between Greedy agent's and Q-Learning agent's finished distances is smaller than their total distance. Greedy dominates the Q-Learning in total distance, but Q-Learning walks just as fast in real time, despite taking more steps. Greedy maximizes each step's reward, but takes much longer to decide an action.

It is clear that the Q-Learning Agent improves substantially in the first few hundred trials, up to around 1000. Based on Figure 4, it seems that the improvement flattens out quickly before 1000 steps of learning. The Q-learning actually beats the Greedy Agent for the LeftRight-Origin walking gait after 1000 trials under the metric of finished distance. This is surprising since the robot had only 1000 trials to explore $(5^3 * 3 * 4) = 1500$ (state, action) pairs (5 possible positions for each foot in the state, 3 leg choices for the next action and 4 possible action moves because it can't move to its own location). Two-Step Agents are impractically slow to calculate even with this relatively small state space.

All the walking gaits have pretty similar results, except for LeftRight-Vertical. It get off to a slow start, but eventually Q-Learns its way to a satisfactory gait.

The average of Greedy Agent's 50 foot distance is also included in Figure 4. While it is higher than all walking gaits, the Q-Learning curves slowly approach.

6 Discussion

Its probable that the plateau in Figure 4 is really a small incline, and that when you zoom out, it steadily grows for a long time before truly flattening out. This is because Q-Learning will eventually converge to the optimum gait.

There are many Q-Learning parameters and simulation parameters that could be tested in future study. For this project, they were kept steady with a discount of .9 and alpha of .3. It is very possible that tuning these parameters will result in faster Q-Learning. There are also many variables that will affect the walking speed and form, such as the length of the divisions for the possible feet positions, and the number of divisions. The Qlearning for one combination of variables might not carry over to another.

Greedy Search Agent does fair better than Q-learning Agent in most gaits, except for LeftRight-Origin according to the finished distance metric. Though it is remarkable that Q-learning Agent comes so close to an agent that is allowed to simulate one step ahead.

Other future work would include: Testing informed search algorithms using a reasonable heuristic, controlling a physical quadruped using the simulation (since this simulation does not capture the fuzzy reality of real-world sensors), and making a generalized Q-learning algorithm for a robotic creature of any shape and any number of legs.

All in all, I think it was a successful project in that it achieved what I set out to achieve, which was to teach a robot to walk from scratch and to test possible walking gaits. The addition of uninformed search agents came about because I wanted to be able to compare the Q-Learned performance to some other algorithm that makes for a reasonable walking gait. The inclusion of uninformed search agents also helped demonstrate the practicality of Q-Learned gaits, considering how slow they took to calculate compared to a QValue-based action selection algorithm.

A System Description

Appendix 1

First, you must download V-REP software: `http://www.coppeliarobotics.com/downloads.html`. It is a quick and easy download. Then download the files CS182FinalReport-QLearning.ttt and CS182FinalReport-Race.ttt and Hexapod-Qdata.txt. The first demonstrates a hexapod Qlearning through 500 steps. The second demonstrate a race between a QLearned agent, Greedy Agent, and UCS2Generation agent. You will need to place these files in the scene folder within the VREP files. It will be called something like: V-REP-PRO-EDU-V3-3-2-Mac/scenes.

To start VREP, go to the command line and cd into V-REP-PRO-EDU-V3-3-2-Mac or your computer's equivalent. Then run the command ./vrep.app/Contents/MacOS/vrep if you have a Mac, or the equivalent for the computer you have. This is if you want VREP to print variables to the command line. Otherwise you can just open the VREP application directly.

Once open, go to the file menu and open one of the two scenes that you put in the scene folder. Once you open one of these scenes, you can press play to begin the simulation. You can investigate the code by clicking the document icon in the middle of the left panel. The code is located in the Threaded file, not the main file. Note: If the race simulation of CS182FinalReport-Race.ttt is too slow, you can select the leftmost hexapod (the UCS2Generation agent) and press delete.

To run CS182FinalReport-Race.ttt, you will need the Qvalues for the Qlearning robot. That's what is in Hexapod-Qdata.txt. You need to download this file and then copy the path to that file

in your computer. Then go to the one line in the code that says "io.open" and paste your full path to the file (just replace the current file path that is there). Then you can run the race. I already uploaded Qvalues to the text file by training with the LeftRight-Origin gait on 500 steps. If you don't change the path name, then VREP will freeze.

For CS182FinalReport-QLearning.ttt, you can just press play, and the Q-Learning will start from scratch. I chose the FrontBack-Vertical walking gate for this file, but you can go into the code and change the gait. You can do this by searching for the part of the file with the comment "CHANGE CONSTANTS HERE". You only have to change legPairs and the right vertex value. You choose which legs are symmetric to which by matching the index of the array with the value (in LUA index starts at 1). The red foot is 1 and the blue foot is 2. The legs go from 1 to 6. For FrontBack gaits, use right-v-num = 6, for LeftRight gaits, use right vertex = 1. Make sure to rotate either the your perspective or the robot itself so that you see it walking straight depending on which value you chose for the vertices.

In case your computer is not letting you use the .ttt scene files, I have attached the CS182FinalReport-QLearning.ttt code on the course website. To get this code into the simulation, you first have to drag the hexapod model from the V-REP interface. The one I used is located under the Model Browser in robots/mobile. You just drag hexapod1 into the scene. Once it is in the scene, press the scripts logo in the middle of the left sidebar. You will need to modify the Threaded script and delete the Non-threaded script. Double click the threaded script and paste in the code. Double click the non-threaded script and delete all the code inside. Then you can press the play button, and it will start QLearning.

B Group Makeup

Appendix 2: I am the only one who worked on this project

C Bibliography

(1) http://oak.conncoll.edu/parker/papers/SMC2011_Quad.pdf
(2) http://yosinski.com/media/papers/Yosinski2011EvolvedGaits.pdf
(3) http://www.cs.utexas.edu/~pstone/Papers/bib2html-links/icra04.pdf
(4) http://sysplan.nams.kyushu-u.ac.jp/gen/papers/CDC2001/CDC2001_080.pdf
(5) https://www.scribd.com/document/62787480/Q-LearningHexapod

(Used PDF as bibliography so they are easier for you to navigate to)