Comparison of Techniques for Stabilization of a Triple Inverted Pendulum

Erik Lee and James Perkins

***Abstract*— We present a comparison of a LQR controller and Q-learning on a simulation of a triple inverted pendulum. While the LQR controller was able to balance the pendulum, Q-learning was unable due to memory and computing limitations. We examine the strengths and weaknesses of each method and compare their abilities.**

I. Introduction

The balancing of a triple inverted pendulum [Figure 1] is an important problem in robotics because it mimics the human body and its balancing mechanisms. While most solutions are quite primitive compared to an actual human, there has been some progress in relation to this issue as humanoid robots are becoming more popular for military, care and other applications.

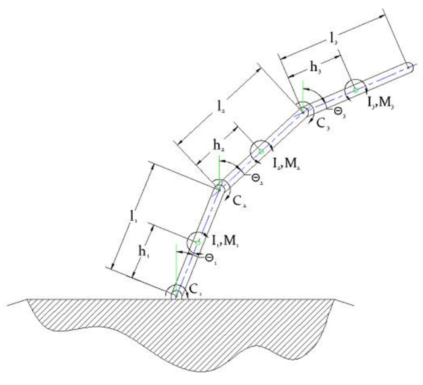


Fig. 1. Schematic of triple inverted pendulum

In this paper, two methods of controlling the triple inverted pendulum in simulation are used and compared. An continuous LQR controller is compared to a state-based discrete Q-learning system. It will be shown that the LQR controller is able to balance the pendulum given certain constraints, while Q-learning was unsuccessful due to system constraints. Both methods, however, may be valid with more work. First, the model of the triple inverted pendulum will be described, followed by the LQR controller implementation. The Q-learning implementation then follows, and the paper concludes with a discussion on the merits of each method.

II. System Model

A mathematical model for a triple inverted pendulum can be derived using the Lagrange differential equations. As shown in Figure 1, the pendulum of concern is pinned and non-actuated on the first link. For purposes of linear control and state space matrix representation, the nonlinear model of a triple inverted pendulum was linearized by removing all nonlinear terms and implementing a small angle approximation for the sine function. The linearized differential equation is

$$\begin{pmatrix} J_1 & l_1M_2 & l_3M_3 \\ l_1M_2 & J_2 & l_2M_3 \\ l_1M_3 & l_2M_3 & J_3 \end{pmatrix} \begin{pmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{pmatrix} + \begin{pmatrix} C_1+C_2 & -C_2 & 0 \\ -C_2 & C_2+C_3 & -C_3 \\ 0 & -C_3 & C_3 \end{pmatrix} \tag{1}$$

$$* \begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{pmatrix} - \begin{pmatrix} M_1g\theta_1 \\ M_2g\theta_2 \\ M_3g\theta_3 \end{pmatrix}$$

where

$$\begin{aligned} M_1 &= m_1h_1 + m_2l_1 + m_3l_1 \\ M_2 &= m_2h_2 + m_3l_2 \\ M_3 &= m_3h_3 \\ J_1 &= I_1 + m_1h_1^2 + m_2l_1^2 + m_3l_1^2 \\ J_2 &= I_2 + m_2h_2^2 + m_3l_2^2 \\ J_3 &= I_3 + m_3h_3^2 \end{aligned}$$

and table 1 displays the values represented by each of the variables.

TABLE I
Triple inverted pendulum system parameters

Symbol	Value
l_1	$110(mm)$
l_2	$170(mm)$
l_3	$160(mm)$
h_1	$70(mm)$
h_2	$90(mm)$
h_3	$80(mm)$
m_1	$1.9(kg)$
m_2	$0.467(kg)$
m_3	$0.292(kg)$
I_1	$9.6x10^-4(kg-m^2)$
I_2	$6.8x10^-4(kg-m^2)$
I_3	$4.1x10^-4(kg-m^2)$

Defining a state vector such that

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix} = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{pmatrix} \tag{2}$$

and plugging in the parameter values for the physical system, the state space representation of the dynamic system is

$$\begin{pmatrix} \dot{X}_1 \\ \dot{X}_2 \\ \dot{X}_3 \\ \dot{X}_4 \\ \dot{X}_5 \\ \dot{X}_6 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 187.9 & -74.6 & 10.3 & -29.9 & 34.4 & -14.7 \\ -176.1 & 220.3 & -76.4 & 47.2 & -104.2 & 66.5 \\ 95.0 & -299.7 & 222.1 & -48.6 & 193.3 & -149.8 \end{pmatrix} \tag{3}$$
$$* \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -171.4 & 127.7 \\ 327.8 & -578.2 \\ -378.0 & 1302.5 \end{pmatrix} \begin{pmatrix} t_{m1} \\ t_{m2} \end{pmatrix}$$

with and representing the motor torques for the second and third pendulum, respectively.

III. Controller Design

Stabilization of an inverted pendulum is a classical control problem that is often used to gauge the quality of a controller. Extensive research in the controls field has shown on multiple occasions that a Linear Quadratic Regulator (LQR) is well suited for inverted pendulum stabilization[6][5]. Hence, a first approach is to implement an LQR control scheme to stabilize the triple inverted pendulum.

The basic idea behind an LQR controller is to minimize a cost function. Given a state space model of the system

$$\dot{x} = Ax + Bu \tag{4}$$

where x variable is a matrix representing the system states and the u variable is a matrix containing the control inputs. In this situation, the control input is defined by the equation

$$u = -kx \tag{5}$$

such that the K variable, a gain matrix, scales the system states to generate an input signal. For this type of controller, the cost function is generally defined as

$$J = \int_0^\infty (x^T Q x + u^T R u) d\tau \tag{6}$$

where Q and R represent weighting matrices. The Q and R matrices are required to be positive definite matrices and in this case are selected to be diagonal matrices[1]. It is clear from the cost function that the diagonal weighting matrices specify how much consideration each state and input is given. That is, a large valued Q matrix and small valued R matrix mean that the changes in the state matrix will be amplified as compared to the changes in the input matrix.

In order to implement an LQR controller, one must select appropriate weighting matrices. For the triple inverted pendulum model, several different weighting matrices were selected and tested. As a first approach, the elements of the Q matrix were selected to be much larger than the elements of the R matrix. This selection translates into a controller that is more sensitive to the states of the system than the control input. The logic behind this choice is that the main design criterion is stability and therefore the system states dictate stability. In foresight, one should expect that this could lead to actuator saturation, since minimal weighting is given to the input values. A first set of tests showed that this stabilized the linearized model of a triple inverted pendulum [Figure 2]. As shown in Figure 3, the base LQR controller is able to stabilize the system when a disturbance input of 1^o is applied

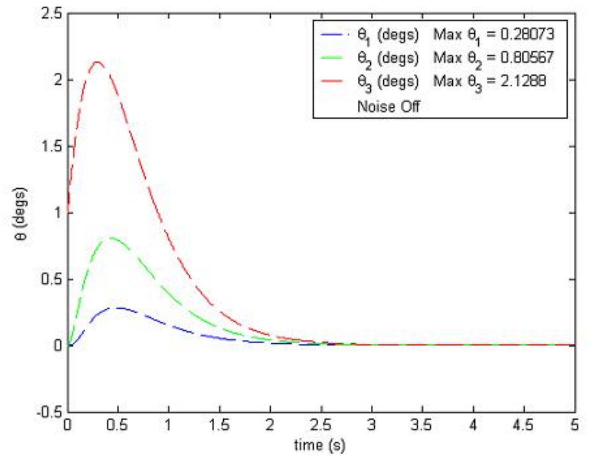


Fig. 2. System response with initial LQR controller

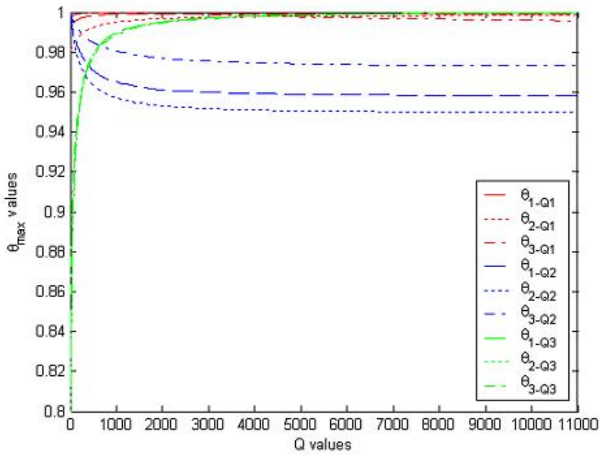


Fig. 3. Dependence of maximum displacement on Q matrix values

to θ_3. Further testing showed that the LQR controller could stabilize the system for disturbances of 2^o and 9^o for θ_2 and θ_3, respectively . It should be noted that disturbances with magnitudes greater than this will cause the system to oscillate outside the 20^o limit, and therefore invalidate the small angle approximation.

Although stabilization of the system has been achieved, there are a number of reasons to extend the capabilities of the controller. First, it is likely that in actual application, the pendulum displacement may exceed the limits stated above. Therefore, extensive tuning of the controller must be performed in order to maximize the stability range of the controller. Second, up to this point the simulations have been run without the inclusion of noise. On a real system, noise is unavoidable. The inclusion of noise will only act to destabilize the system. Hence this is a second reason to tune the controller. Since LQR controllers don't have tunable parameters that directly correlate to meaningful system parameters, it is a relevant first approach to attempt to understand how the tunable parameters affect the system response.

This was performed by holding each of the Q matrix values constant, varying the remaining Q value, and then measuring the output. Small initial angular displacements were given to the system to force some control effort to be input. Instead of recording the actual output signal, the maximum angular displacement was recorded. This was done because the main mandatory design criterion was to stay within the region where the small angle approximation holds. Therefore, recording the maximum displacement, as opposed to the entire signal, better quantifies the stability with respect to this criterion. For these tests, the Q matrix was defined as

$$Q = \begin{pmatrix} Q_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & Q_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & Q_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & Q_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & Q_1 \end{pmatrix} \quad (7)$$

where each of the Q values inside the Q matrix were varied. The tests were performed by holding R constant at the identity matrix and setting each of the values to 1. The remaining Q value in the Q matrix was then varied individually while holding the other Q values at 1. This allows one to quantify how each entry in the Q matrix individually affects the maximum angular displacement. Figure 3 shows the results from this test, where the values are normalized by their maximum value for a given Q value test. The x-axis of the plot shows the Q values, with the red lines corresponding to varying , the blue lines corresponding to varying and the green lines corresponding to varying . Although the plots in Figure 3 are of little quantitative value, the qualitative significance is large. Notice in the figure, by looking at the y-axis, that all of the lines are bounded within 80% of the maximum value. Also, most of the maximum angular displacements are only altered by 4% when varying the Q value. This leads to two conclusions. First, regardless of the Q matrix values, stability will be preserved for small angular displacements. Although, this is a great attribute, it has a downside. That is, regardless of the Q matrix values, large angular displacements can not be overcome. This is true because, if one can not introduce a control effort large enough to destabilize the system, then when large displacements are externally input into the system the controller will not be able to apply a large enough control effort to stabilize the system.

With this information in mind, the next step was to tune the LQR controller in order to increase the robustness of the controller. After much tuning, the gains that produced the most robust controller, of the ones tested, was a Q matrix of,

$$Q = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & .01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & .1 & 0 & 0 \\ 0 & 0 & 0 & 0 & .1 & 0 \\ 0 & 0 & 0 & 0 & 0 & .1 \end{pmatrix} \quad (8)$$

with the R matrix equal to the identity matrix. Using this Q matrix, the range was extended to 4^o and 14^o for θ_1 and θ_3, respectively. Figure 4 shows a plot of the poles, or eigenvalues, of the closed loop system. It is evident that the response of the system will be non-oscillatory since the imaginary part of all of the poles is zero. Since all of the real parts of the eigenvalues are negative, the system will exhibit a stable response. The problematic poles of closed loop system are shown in the zoomed-in portion of Figure 4. These poles present a problem for a number of reasons.

With respect to implementation, these poles could be

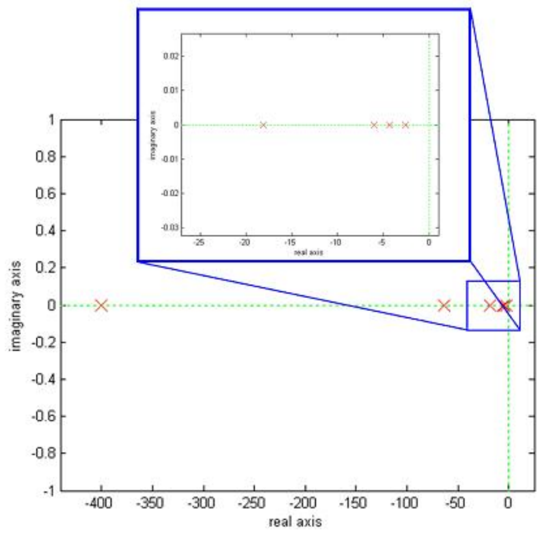


Fig. 4. Poles of closed loop system with LQR controller

shifted on the actual system if the parameter values are different than the model parameters. The problem with this is if the poles shift to the right half plane of the real axis then the system will be unstable. A second problem with these small negative poles is that they produce a solution that responds more slowly. That is, the small argument in the exponential causes the response to be slow with respect to the more negative poles.

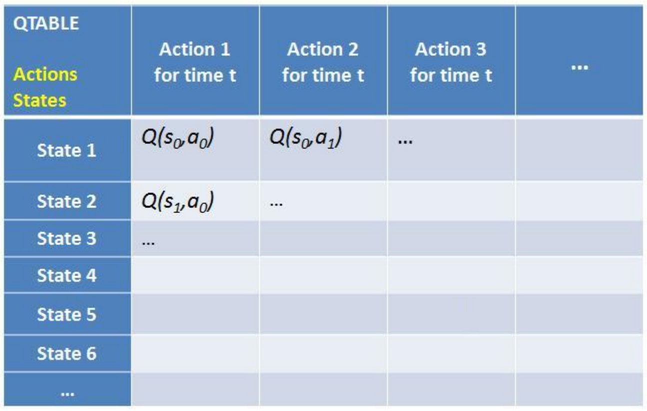
IV. Controlling the Triple Inverted Pendulum with Q-Learning

A. Q-Learning Introduction

An important breakthrough in reinforcement learning was the development of Q-Learning[9]. Q-Learning, an off-policy temporal-difference algorithm, directly approximates the optimal action-value function through basic trial and error. The robot begins with a predefined set of states and actions with no knowledge of the best action to take in a given state in order to reach the goal state. By taking actions from states (with no knowledge), the robot can accumulate rewards through reaching the goal or a state that leads to the goal and punishments by moving to an unwanted state. The Q-learning algorithm is as follows [8]:

```
1. Initialize $Q(s,a)$ arbitrarily
2. Repeat (for each episode)
3.     Initialize $s$
4.     Repeat (For each step of episode)
5.         Choose a from $s$ using a policy
6.         Take action $a$, observe $r, s^{'}$
7.         $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma maxQ(s_{t+1},a) - Q(s_t,a_t)]$
8.         $s \leftarrow s^{'}$
9. until $s$ is terminal
```

where $Q(s_t, a_t)$ is the state/action pair before taking the most recent action, α is the learning constant, r_{t+1} is the reward function, γ is the discount factor and $maxQ(s_{t+1}, a)$ is the maximum Q-Value (from the Q-Table) in the state reached after taking the action. Basically, the robot will initialize the Q Table [Figure 5] and then from its start state choose an action. The action will then be evaluated based on the equation in line 7 of the above algorithm. After



QTABLE Actions States	Action 1 for time t	Action 2 for time t	Action 3 for time t	...
State 1	$Q(s_0,a_0)$	$Q(s_0,a_1)$	...	
State 2	$Q(s_1,a_0)$	...		
State 3	...			
State 4				
State 5				
State 6				
...				

Fig. 5. Example basic Q-Table

it is evaluated, a new action will be chosen from the new state, performed and evaluated. This will be repeated until either the goal is reached, or a terminal unwanted state is reached. The robots position will then be reinitialized and lines 5-8 repeated. With infinite iterations and a policy that chooses all possible actions from all states (i.e. random), the algorithm will converge on a solution where each state has an action with a highest q-value. Although it does not prove the behavior will be optimal, any optimal behavior requires that all state-action pairs continue to be updated[8]. Convergence is also dependent on two factors in the table, the learning constant and discount factor. The learning constant is a value between 0 and 1 that describes the level of importance that is placed on newly acquired rewards. With a higher value, the robots actions, positive or negative, will highly influence the current q-values in the table. The discount factor, also a value between 0 and 1, limits the importance of previously acquired q-values.

1) General Q-Learning Issues and Related Works: One of the biggest problems with Q-Learning is the size of the Q-Table. In a complex system with a large number of states and actions, the table can become quite large requiring a large amount of memory for storing the table. While it is possible to only store the best action for each state when the learning process is complete, during the learning phase the entire table must be used until the values converge to a solution. For a floating point table in C, the size of the table in bytes is approximately equal to

$$size = states * actions * 4; \tag{9}$$

where 4 is the number of bytes for a floating point variable. As the size of the table increases, so does the time it takes for the values to converge to the optimal policy. One factor that affects this is the decision made on which action to take from a given state. A simple, yet slow solution would be to randomly choose actions from every state. While this would explore the entire table in time, a commonly used method known as ϵ-greedy has been used [8] which balances between exploration (i.e. a random choosing of an action) and exploitation (choosing the action with the highest Q-value) with $0 \leq \epsilon \leq 1$. While all actions from all states will be performed, q-values for actions that already have been performed will be reinforced. Sutton and Barto showed that a nonzero ϵ is usually better than a 0 value, but after a period of time spent learning, exploration is less important and the value can be changed to include more exploitation. Work has been performed to choose the action with the most *value* during exploration to improve convergence time. In [3], Q-value sampling has been used to represent the agent's knowledge of the available rewards as probability distributions. [2] has extended this idea with Myopic-VPI and Bayesian filtering and has shown for simple systems an improvement over conventional q-value choice policys. An overview of this issue and some strategies that have been used can be seen in Kaelbling, Littman and Moore [4]. Because of the issue of convergence time, it is often impossible to perform learning on an actual robot. Learning is usually performed in simulation and the solution ported to the robot. Of course, moving from simulation to robot is a difficult due to the many issues with describing a real, complex system in a dynamic world in simulation.

Another issue with Q-Learning and reinforcement learning in general is the reward function. As was described earlier, the only reward given was for the goal state. This value would then "propagate backward" through the table as actions led to that goal state and actions led to the state that led to the goal state and so on. The difficulty with this approach, however, is that it may take a very long time (depending on the number of states and actions) to actually propagate this value backwards through the table. It may be more beneficial to create a proportional reward system that gives rewards or punishments every time an action is taken from a state based on the metric of *how much* the system moved towards the goal state. The problem that lies with this approach is that the programmer may be deciding for the system how it should act from a given state, instead of allowing the system to decide. It may be beneficial to have a reward system that is very basic, so that for complex systems (i.e. a robot walking), the robot can learn how to walk without being constrained by a possibly flawed human-created reward system.

In general, reinforcement learning is performed with an immediate state-action-state reward system, but some systems have used a delayed reward where the agent builds is Q-table based on a reward that can take place in the future. These problems are modeled as Markov Decision Processes [4].

An issue that arises in all system is the problem of discretizing the states in a manner that adequately describes the system in order to make the actions worthwhile. Without enough knowledge of the system, actions can not be properly chosen to achieve the goal. A sign of possible incomplete state description would be a lack of convergence of the Q-Values as well as, of course, inappropriate actions performed from states in the real system. To solve the problem of discretization, Zhou et. al. have described an approach entitled "'Dynamic Fuzzy Continues-Action Q-learning"' that works on continuous domains where the learner has a continuous perception of the state space and can trigger continues actions [11].

B. Implementation Description

A classic Q-learning algorithm was followed for attempting to balance the triple inverted pendulum in this project. Because of the sin approximation explained earlier in this paper and to keep the q-learning controller consistent with the controller described above, each joint in the robot was only able to move between -20 and 20 degrees. States in the system are represented by 6 values: $\theta_1, \theta_2, \theta_3, V_1, V_2$ and V_3

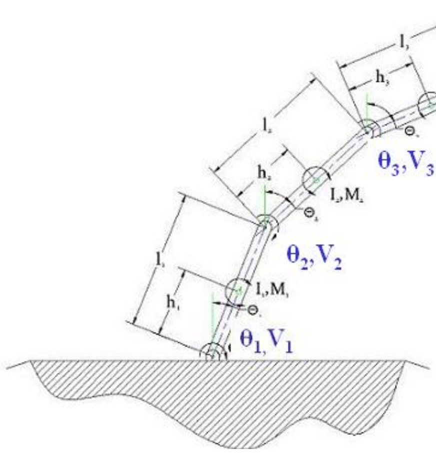


Fig. 6. Schematic of triple inverted pendulum

[Figure 6]. This discretization led to approximately 22 million states. Actions consisted of two torque values for the top two joints for a period of time t. Their minimum and maximum values were also calculated from tests with the model and, although inexact, seemed to be reasonable estimates from model input. Initially, 25 actions were created. The classic learning equation earlier in the paper was used for updates to the Q-Table, with random actions being chosen approximately 70 percent of the time. Both a proportional reward system for every action taken and a reward system that only gave a reward if the goal was reached were implemented.

C. Implementation Issues and Discussion

For most of the Q-learning implementation process, only 4 variables were being used to describe a state $(\theta_1, \theta_2, \theta_3, V_1)$. Through intuition, it was found that two more variables were needed to adequately describe the state. Without V_2 and V_3, the top two joints velocities were not taken into account when performing an action. For example, performing action X in state Y in an instance with $V_2 = a$ might lead us closer to the goal, but performing that same action in the same state with $V_2 = b$ may make the bottom pendulum move outside the 20 degree maximum angle. Thus, the state is not being adequately described because the same action in the same state could lead to two unrealistically different states, possibly ruining the Q-table.

When adding in the two extra variables, as mentioned before, there were approximately 22 million states. With 25 actions and 22 million states, the floating point Q-table was approximately two gigabytes in size, which was too large for the memory constraints of the computer. Because of the time left for the project at the realization of this inadequacy, we were unable to implement the system on an appropriate machine or create a multi threaded algorithm.

It is still a concern that the ranges on the states do not adequately describe the state space in order to balance the triple inverted pendulum. A one degree range on each θ may be too large to balance the pendulum. Because velocities vary greatly when moving up the pendulum and the memory issues late in our project, we were unable to determine which velocity range was appropriate for each pendulum link. Also because of the memory issues, we were unable to determine absolutely whether our actions were correct for the system.

Although there were memory issues, it still may be nearly impossible for the Q-Table to converge to a solution with the large number of states and the current action-selection policy. Another manner, such as suggested in [4] may be more appropriate for such a large system, but it is still questionable as to whether the triple inverted pendulum problem requires too many states for any reasonable computation time. Since it is computed off-line, however, it may be possible. More study would need to be performed to determine this.

Another issue with Q-Learning is that it is difficult to tune the created controller without recalculating the entire table. Because of the issue of compute time, this is a significant problem. Also, when problems arise with the calculated policy, it can be difficult to determine the reasons for failure because of the disconnection between the user created algorithm and the actual learning process. Defining states and actions are somewhat of an inexact science, yet still interrelated, compounding the problem of choosing them correctly. Also, in regards to compute time, multiple values, such as the learning constant, discount factor and ϵ can have a major effect. These values are also chosen based on an educated guess.

In other work, q-learning and other types of reinforcement learning has been implemented on inverted pendulums with some success. Some of this work can be found in [10] and [7].

D. Q-Learning Conclusion

While Q-learning is an interesting concept, it presents problems in regards to memory and compute time. These problems can be worked around, but it may be necessary to define the state space in an unsatisfactory manner. With infinite computing power, Q-learning would be a viable option for a triple inverted pendulum and may still be if implemented correctly.

E. Comparison of Controller and Q-Learning

Although the Q-learning section this section of the project was unsuccessful, the process was helpful in determining the differences in the LQR controller and Q-learning. One advantage of Q-learning over the LQR controller is that nonlinear systems are inherently no different than a linear system in Q-learning. Creating a nonlinear controller can be a difficult and time consuming task for the engineer, while the Q-learning programmer can allow the system to design a controller itself given nonlinearities. Of course, Q-table convergence time with Q-learning can be very time consuming. The difficulty with this is that if the calculated policy is incorrect or inoptimal, the policy must be recalculated and the entire Q-learning process restarted from the beginning. With a LQR controller, values such as gain are easily and rapidly tuned once placed on a real system. Another issue with Q-learning is that the states are discritized. It is possible that the size of the Q-table could severely cripple the learning process, while controllers do not have a similar problem. Also, state discretization may make it difficult to find a set of actions that can effectively allow the robot to reach the goal state. Although untested in this process, as mentioned earlier, work has been performed to create a continuous Q-learning theory[11]. While Q-learning was unable to balance the pendulum and the LQR controller was, more work may show that the Q-learning algorithm, if implemented correctly, can match the performance of the LQR controller. It may be beneficial to perform other types of reinforcement learning such as value iteration to find an optimal policy.

References

[1] Chi-Tsong Chen. *Analog and Digital Control System Design: Transfer-function, State-space, and Algebraic Methods*. Oxford University Press, Inc., New York, NY, USA, 1995.

[2] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *In AAAI/IAAI*, pages 761–768. AAAI Press, 1998.

[3] J.Wyatt. Exploration and inference in learning from reinforcement. In *PhD thesis Department of Artificial Intelligence, University of Edinburgh*, 1997.

[4] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[5] Qing-Rui Li, Wen-Hua Tao, Na Sun, Chong-Yang Zhang, and Ling-Hong Yao. Stabilization control of double inverted pendulum system. *Innovative Computing ,Information and Control, International Conference on*, 0:417, 2008.

[6] M. Mihelj and M. Munih. Double inverted pendulum optimal control-basis for unsupported standing in paraplegia. *Advanced Motion Control, 2002. 7th International Workshop on*, pages 121–126, 2002.

[7] Rudolph Pienaar. Adaptive control of human posture using reinforcement learning. In *PhD thesis Cleveland State University*, 1993.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[9] C.J.C.H. Watkins. Learning from delayed rewards. In *PhD Thesis University of Cambridge, England*, 1989.

[10] Yu Zheng, Siwei Luo, and Ziang Lv. Control double inverted pendulum by reinforcement learning with double cmac network. *Pattern Recognition, International Conference on*, 4:639–642, 2006.

[11] Yi Zhou and Meng Joo Er. A novel q-learning approach with continuous states and actions. *Control Applications, 2007. CCA 2007. IEEE International Conference on*, pages 18–23, Oct. 2007.