Using NEAT to Stabilize an Inverted Pendulum

Awjin Ahn and Caleb Cochrane

May 9, 2014

Abstract

The inverted pendulum balancing problem is a classic benchmark problem on which many types of control implementations are tested, as well as being an apt analog to actual applications in which the system being controlled is naturally unstable. As a means of aiding the design process, and in an effort to explore alternative controller implementations, adaptive control systems have been developed to automatically tune control system parameters. We hypothesized that the Neuroevolution of Augmenting Topologies (NEAT) technique will evolve effective neural network controllers for an inverted pendulum system. NEAT was, in fact, successful in producing controllers that provided consistent balancing motion. Tweaking the fitness function provided us with even smoother and efficient solutions. Furthermore, to test the robustness of NEAT, we introduced random disturbances during evolutions. NEAT was able to generate controllers that handled these disturbances well, despite needing more generations.

1 Introduction

1.1 Overview

A common problem of particular interest in the study of control theory is the design of controllers for systems that are naturally unstable. Furthermore, the design of precise and robust controllers for such systems require deep knowledge of control theory principles, and can thus be difficult to design. As an alternative, it is possible to use learning systems and adaptive methods to aid in the design of effective control systems. In particular, artificial neural networks (ANNs) may be used in place of conventional controllers, and careful design of the network topology (use of recurrence, feed-forward connections, hidden units, etc.) can be used to design highly effective ANN controllers. Finally, Neuroevolution of Augmenting Topologies (NEAT) can further simplify the design task, as the topology of the control ANN does not need to be hard coded by the designer [3].

1.2 The Inverted Pendulum

We want to be able to design a control system capable of balancing an inverted pendulum on a cart. In the following sections, we discuss two different approaches to this problem, and the motivations behind them.

1.2.1 Classic Control Solutions

Of classic importance in the study of system control is the *unstable system*. An unstable system may have a theoretically infinite or constantly increasing output due to a finite input. For example, a ball resting on a hill constitutes an unstable system, as a small push in any direction will cause the ball to begin accelerating down the hill. Similarly, balancing an inverted pendulum is difficult, because a small perturbation in any direction will start the pendulum falling faster and faster. This class of problems has several real world applications, such as controlling the trajectory of a rocket immediately after launch, or the steering controls of fighter jets.

Precise controller implementations are traditionally designed through the application of control theory. By characterizing the response of the system with a transfer function, one can design a variety of control systems, such as a PID (proportional, integral, derivative) controller [5]. PID controllers are effective for inverted pendulum control. A block diagram of a PID controller is shown in figure 1:

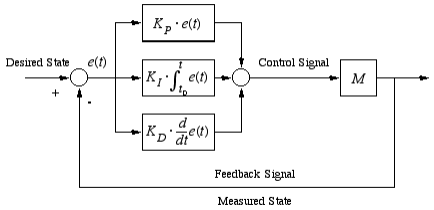


Figure 1: A Basic PID Controller. The current system state is fed back to the input, to generate an error signal, $e(t)$, which is in turn used to calculate the input to the system

Other classic control theory solutions include state-space representations and pole placing, whereby feedback gains can be calculated by using a matrix representation of the system characteristics [4]. It is important to note that each of these implementations relies on feeding back the current state of the system to the controller, where this state information is scaled and combined to generate an output for the system being controlled.

1.2.2 Adaptive Control and Neural Networks

Rather than designing controllers fully by hand and making them tailored to very specific environments, it is possible to design controllers with built-in learning. For example, a method called *gain scheduling* automatically sets system parameters by first breaking the

control task into several sub-tasks, according to differing sets of operating conditions, and making linear approximations for each region [6],[2].

A system controller can also be modeled with an artificial neural network. Just like a classic controller, a neural network can take current system states and calculate the next controller input to effectively control a system. Using a modified back-propagation algorithm and careful consideration of network topology, neural network controllers tested on benchmark control system tests have comparable performance to classic implementations [1]. Furthermore, design of a *dynamic* neural network can achieve desirable control system properties such as disturbance rejection, or the ability to easily respond to sudden changes, through more complex learning algorithms [6]. Each of these techniques is designed to supply greater assistance in the design of a control system, while maintaining robust control over the system.

1.2.3 Taking Adaptive Control Even Further

The neural network control techniques discussed above all aid in the design of control systems by adapting system parameters, but the designer must still carefully consider the particular topology and training algorithms to achieve robust control. Alternatively, we hypothesize that it is possible to reduce even topological considerations through the use of NEAT to evolve an effective topology.

NEAT works with an initially small population of neural networks. Over the course of several generations of evolutions, the topology of the networks is *complexified* to achieve more and more sophisticated behaviors. During each generation, the most fit networks (as assessed by a fitness metric) reproduce and create new generations. During this step, topologies of the network are altered through mutation and crossover, potentially adding connections between nodes, or adding nodes themselves, producing more complex offspring for the next generation. NEAT is also uniquely able to maintain a diverse population through *speciation*—compartmentalization of types of networks, using historical markings—to protect emerging strategies and give younger individuals a chance to thrive. Speciation also allows more similar networks to produce offspring more efficiently [3].

By instantiating a population of controller networks with a NEAT algorithm, we hope to be able to evolve effective inverted pendulum balancing strategies. Furthermore, we believe that recurrence is likely to be an important aspect of an effective controller, based on the natural back-and-forth motion used by humans when balancing similar systems.

2 Methods

2.1 Hypothesis and Overview

We hypothesize that the NEAT algorithm will develop effective controllers for balancing an inverted pendulum. We explore the efficacy of NEAT in three ways:

1. We implement a penalty for moving too much in order to encourage less erratic (and more physically feasible) solutions.

2. We allow the generation of recurrent connections in order to test whether recurrent connections are beneficial.

3. We evolve networks to handle disturbances to the system.

2.2 Simulator

In order to evolve and evaluate a NEAT population of controller networks, we built a simulated environment of a pendulum balanced on a cart using the pygame graphics library. (A screenshot of our finished simulator is shown in figure 2). To ensure that the controllers developed a physically feasible control system, we carefully derived the equations of general plane motion. These equations describe the motion of the pendulum for each timestep, namely the translational position, velocity, and acceleration (x_c,v_c, and a_c) and the angle and rotational velocity and acceleration (θ,ω,and α), and make sure that both the force generated by the base and the pull of gravity on the pendulum are accurately represented.

Each ANN generated by NEAT has 4 inputs and 1 output. The 4 inputs fed into the ANN by the simulator are the linear velocity of the rod, the angle of the pendulum relative to the vertical, the linear velocity of the base, and the linear position of the base. The ANN generates an output -1 to 1, as the activation function is $tanh(inputs)$. The output is scaled by a factor of 10, then passed back to the simulator which moves the pendulum by the output amount.

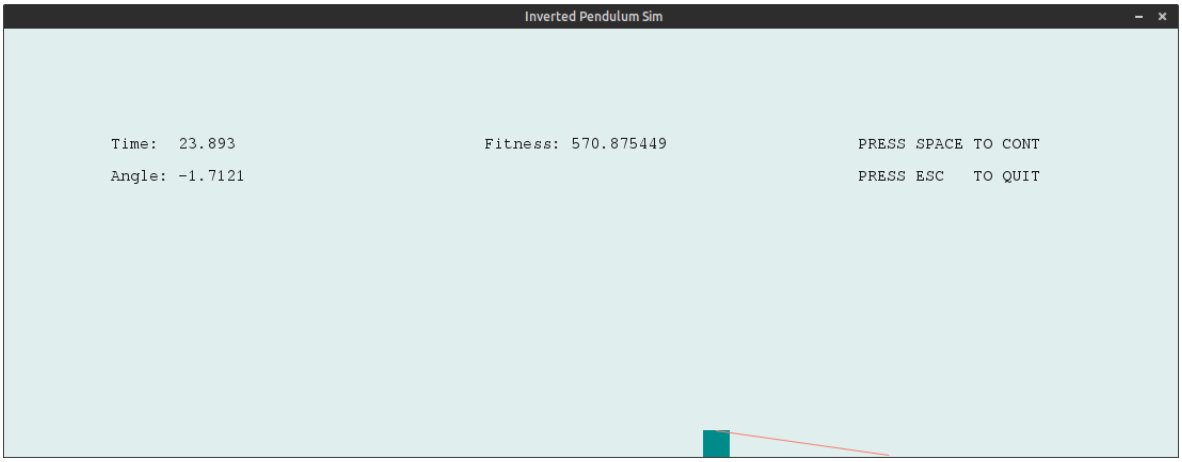


Figure 2: Inverted Pendulum Simulator (Human Control)

We refactored the open source code for NEAT and integrated it into our simulator. Normally, NEAT's evolutions depend on error generation and propagation, but we decided to measure the fitness of an ANN using time:

$$fitness = (duration\ of\ time\ balanced)^2$$

This ensured that controllers that could balance the pendulum for longer periods of time received exponentially greater rewards.

2.3 Experiments

To test our hypotheses, we conducted trials of at most 30 generations for each of the following experiments. At the beginning of each ANN evaluation, the pendulum was set to a random angle in the range of [-0.01, 0.01] radians. The NEAT algorithm stopped whenever an ANN was generated that could balance the pendulum for 120 seconds, which we deemed as a reasonable maximum fitness. Videos of the best evolved topologies, as well as demonstrations of the simulator, are available online at: http://goo.gl/c2JToV.

2.3.1 Experiment 1: Distance Penalty

After some experimentation with NEAT training, we noticed a general trend in the evolved controllers: since the fitness was measured solely via time, the controllers were moving extremely rapidly back and forth. This was effective in terms of balancing the pendulum for a long time, but was an unfeasible solution in terms of energy, sustainability, and realistic physical limits. Therefore, we tested an alternative fitness function in contrast to the original one, in which moving greater distances incurs a higher penalty. The new function is as follows:

$$fitness = (duration\ of\ time\ balanced)^2 - \frac{total\ distance\ travelled}{10}$$

2.3.2 Experiment 2: Recurrence Enabled

In attempts to evolve a controller with even more regular movements, we allowed NEAT to form recurrent connections. We used the new fitness function described above and compared the results to those obtained using only feedforward connections.

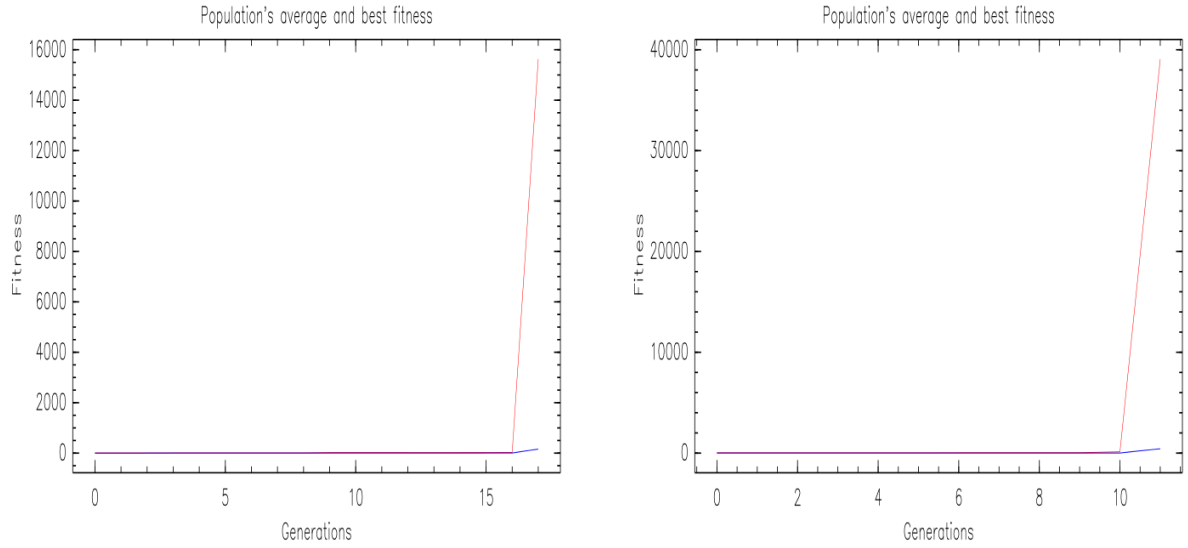
2.3.3 Experiment 3: Disturbances to the System

We added disturbances to the system in order to see if NEAT could evolve a robust controller. The disturbance consisted of a consistent push to either the left or the right for 5 time steps. The disturbances occurred at every 100th time step, but the direction and force of the push were randomly generated.

3 Results

3.1 Experiment 1: Distance Penalty

In an effort to get more smooth control of the inverted pendulum, we altered the fitness function by penalizing excessive movement. In this way, NEAT's solutions can be more conservative with its solutions, rather than oscillating rapidly. The fitnesses for these trainings are shown below in figure 3, though it is important to notice that the scale for the two fitnesses is completely different, due to the altered fitness metric. The qualitative results are also discussed in the following sections.



(a) Fitness function without distance penalty (b) Fitness function with distance penalty

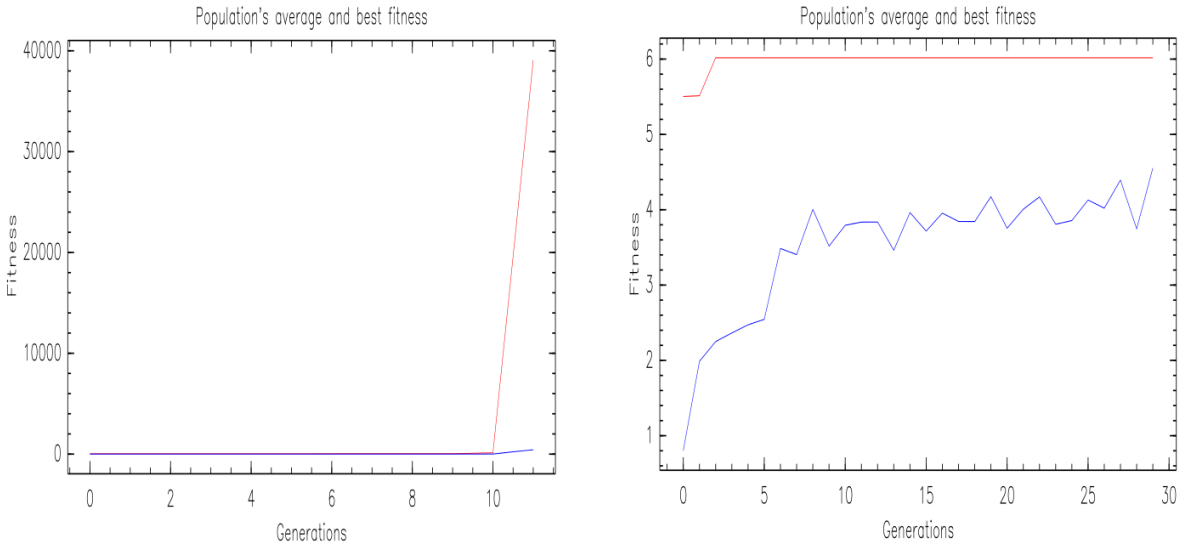
Figure 3: Fitness of controllers trained with different fitness functions

Figure 3 shows that NEAT is able to evolve a successful controller within 30 generations, for two different fitness metrics. Though they only represent one evolution, the trials shown in figure 3 are very typical runs for each method. These figures mostly serve to show that both fitness metrics could be used to develop a successful controller. Observing the qualitative results, however, provides a much better illustration of the difference in behavior resulting from the two functions, which we address in the discussion section.

3.2 Experiment 2: Recurrence Enabled

After performing two training runs to determine the effect of recurrence, the plots of average fitness for two training runs are shown below, side-by-side, in figure 4.

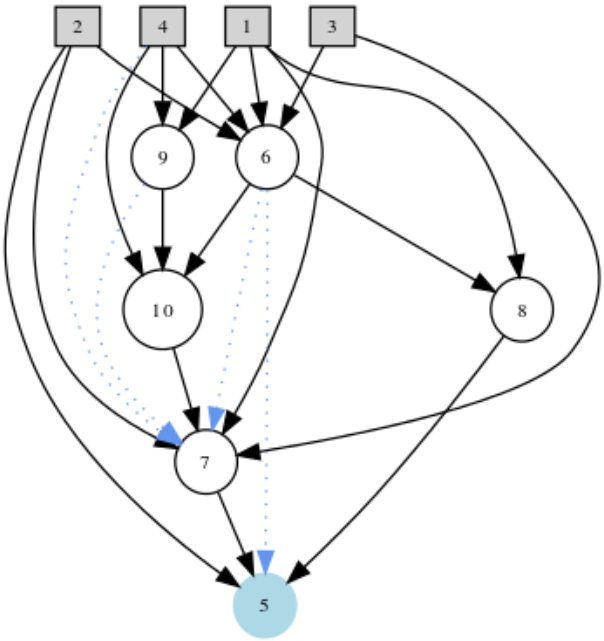
The topologies of the learned networks can also be examined below in figure 5



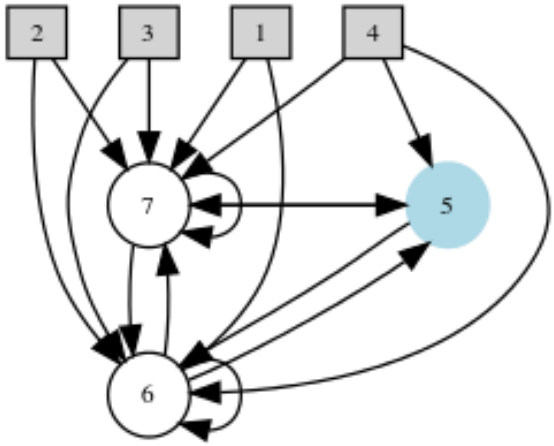
(a) Recurrence disabled

(b) Recurrence enabled

Figure 4: Controllers trained with recurrent connections enabled and disabled. Note radical difference in scale between the two fitnesses



(a) Controller Topology, Recurrence disabled



(b) Controller topology, Recurrence enabled

Figure 5: Controllers trained with recurrent connections enabled and disabled

The non-recurrent solution was able to generate a solution within 30 generations, as seen in figure 4a, and was able to do so consistently, but NEAT with recurrence enabled actually failed to evolve a controller, as shown in figure 4b, and failed consistently. This contradicts our hypothesis that recurrent connections in the networks would be instrumental in the development of feedback control systems. The recurrent trials of NEAT likely failed because the extra search space created by allowing recurrent connections made the problem too difficult for NEAT, at least for our modest population size and number of generations. We also realized that the non-recurrent network implementation still constitutes a feedback control system, because the inputs to the network can be considered as feedback on the current state of the system. Therefore it is actually very reasonable that NEAT without recurrent connections has an easier time finding solutions, since the recurrence options only serve to complicate the problem.

3.3 Experiment 3: Disturbances to the System

We also experimented with implementing a system that can handle random disturbances by training NEAT with random "pushes" to the pendulum system. After twenty-four generations of evolution, a controller was developed that was able to keep the pendulum balanced for the entire allotted time interval. The average fitness across the 24 generations is shown below, in figure 6, as well as the topology developed by the fittest controller in 7.

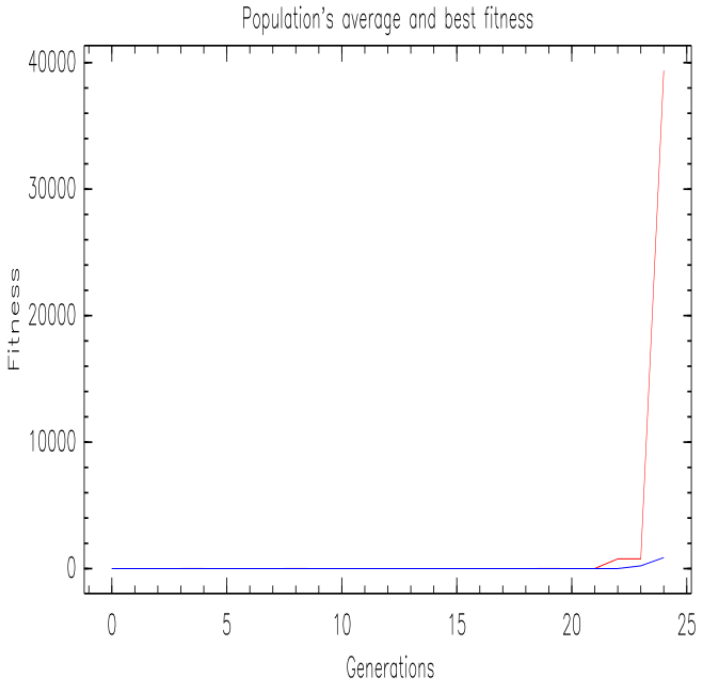


Figure 6: The average fitness of the NEAT population across 24 generations of evolution

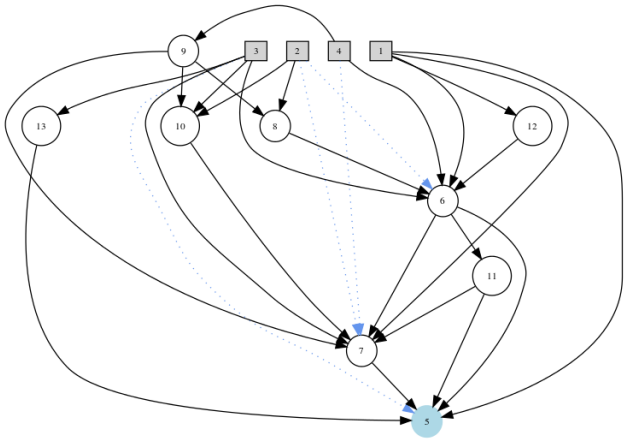


Figure 7: The Topology of a network evolved with disturbance rejection

By training the NEAT population on an environment with constant, random disturbances, we also developed a controller to successfully handle disturbance rejection, as shown in 6. We can also examine the topology of the highest fitness network in figure 7, essentially the disturbance-rejection controller evolved by NEAT. Compared to the topologies of the standard controllers developed in experiment 2, in figure 5 and 5, the topology is much more complex, reflecting the fact that this is a much harder problem to solve.

4 Discussion

For most of our trials, we examined the changes in fitness over the course of evolution to check whether or not a maximum network controller achieved a viable solution to the problem. However, our fitness graphs are somewhat uninformative, beyond demonstrating whether or not a solution is found, because our trials were designed to terminate once a viable solution is found. Also, the learning curve of the population is very steep, because the network essentially either has a correct implementation, or it doesn't—here is surprisingly little middle ground. As a result, fitness rates tend to look like very steep step functions. This is unsurprising, as the act of balancing a pendulum requires a very subtle and deliberate motion that has little room for error.

Rather than looking at the fitness curves, we can get a much better sense of the actual behavior under different conditions by watching the highest fitness networks in action in the simulator. With a simple fitness function of time-squared, NEAT was able to evolve effective but somewhat spastic controller for the inverted pendulum system. Using a slightly more sophisticated fitness function, we were able to achieve much smoother controllers. We also tested recurrent networks, to test our hypothesis that this would enable better controllers, but found that recurrent networks were unable to develop successful controllers. The disturbance handling was quite capable and smooth, and almost indistinguishable from the controller evolved with a distance penalty.

5 Conclusion

In order to aid in the design of complex control systems, many engineers have turned to adaptive methods to assist in the tuning and structural design of the controller. Many such solutions use careful design of neural networks to automatically design system parameters. Our approach was to use the NEAT algorithm to aid in the design of network-controller topologies by automatically evolving the structure of the networks. We have shown that it is possible to use this method to design highly effective and robust controllers, with important features such as disturbance rejection.

6 Future Work

Our work with the evolution of system controllers using NEAT feels very much like just scraping the tip of the iceberg. For future research, it would be highly informative to compare the efficacy of NEAT-evolved controllers against other control methods, such as classic, hand-designed, PID controllers, and other adaptive methods, such as the Kalman Filter. We would also like to apply evolved controllers to actual hardware applications.

7 Acknowledgements

We would like to thank the following individuals:

1. Eric Liu, for sharing his research on hand-designed controllers, especially concerning physical parameters and kinematic equations for an actual inverted pendulum system
2. Professor Faruq Siddiqui, for helping us with the Kinematics of our simulator, and
3. Professor Lisa Meeden, for providing guidance and feedback throughout the semester, and helping with implementation details as they arose.

References

[1] Tarek Aboueldahab and Mahumod Fakhreldin. "Identification and Adaptive Control of Dynamic Nonlinear Systems Using Sigmoid Diagonal Recurrent Neural Networks". In: *Intelligent Control and Automation* 2 (2011), pp. 176–181.

[2] Lingji Chen and Kumpati S. Narendra. "Identication and Control of a Nonlinear Dynamical System Based on its Linearization: Part II". In: *American Control Conference, 2002* 1 (2002), pp. 382 –387.

[3] Joel Lehman and Kenneth O. Stanley. "Abandoning Objectives: Evolution through the Search for Novelty Alone". In: *Evolutionary Computation* 19.2 (2011), pp. 189–223.

[4] Eric Liu. "Rotary Inverted Pendulum". In: *Swarthmore College* (2013).

[5] Francesco Corucci Pasquale Buonocunto. "Real-time PID Control of an Inverted Pendulum". In: *University of Pisa* (2012).

[6] Gregory L. Plett. "Adaptive Inverse Control of Linear and Nonlinear Systems Using Dynamic Neural Networks". In: *IEEE Transactions on Neural Networks* 14.2 (2003), pp. 360–376.