Mitch Boucher

ECE 497 - Evolutionary Robotics

Dr. Eduardo Izquierdo

11/20/2024

Programming Report #8: Final Project, 3D Inverted Pendulum

The goal for this project was to combine a 3D simulated body, microbial genetic algorithm, and feed-forward neural network to simulate the Inverted Pendulum problem. The inverted pendulum is a system where a single beam pinned at one side tries to hold its center of mass above its pivot point; this is a naturally unstable position, and the system must therefore apply a force to stay balanced. I chose to go towards developing a 3D simulated body due to my interest in robotics that often use 3D models to test designs. Because I am developing a 3D body for the pendulum, I chose to use a microbial genetic algorithm and a feed-forward neural network as I have already developed and tested them. For this project I have used multiple resources including: ChatGPT, my designs from the midterm project, and an online URDF for a pendulum system. I used ChatGPT to get a basic understanding of how to update the position and get values of the URDF components. The online resource to the URDF is in the references at the end of this paper.

I experimented with different versions of creating the pendulum with the best one being to simply create a URDF from scratch as a file. The body (figure 1) in the URDF is defined by 4 links at 3 joints. The links include: the world, the pivot point (represented as a small sphere), the arm, and an arm tracker at the midpoint of the arm. The 3 joints include: the joint between the pivot point and the world, the joint between the arm and the pivot point, and the joint between the arm and its tracker. The arm tracker's point is to measure the beam's position, such as its angle and angular velocity while simulating. The joint between the pivot point and the world is fixed so that the pivot point stays stationary in mid-air. The joint between the arm and the pivot point is a continuous joint with the ability to rotate in the y-axis. The joint between the arm and the arm tracker is fixed at the midpoint of the arm to track its movement. The position of the pendulum arm could be measured using the getLinkState of the arm tracker link. This command was used to obtain the x and y coordinates and the angular velocity of the midpoint of the arm. The angle itself was calculated using the arctangent of the x and y coordinates subtracted by $\frac{\pi}{2}$ to ensure that the upright position is 0 radians. The pendulum was interacted with using the setJointMotorControl2 command by pyBullet. The comamnd was used to set the initial position, inital velocity, and the torque at each interaction. The neural network used was a feed forward network with a singular hidden layer of 10 nodes. This is downsized from my midterm project that used 2 hidden layers, both with 10 nodes each. The evolutionary algorithm used was a microbial genetic algorithm. The genetic algorithm was set to evolve the weights and biases for the neural network For all the simulations I used a population size of 50, recombination rate of

50%, mutation probability of 10%, and 5000 tournaments. Some of the tests I ran for longer durations to see the long-term behavior of some of the systems that seemed repetitive. The inputs to the neural network were the x and y coordinates and the angular velocity. The output of the neural network was the torque to be applied at the pivot point. The system's fitness (equation 1) is determined similarly to the midterm in that it uses the torque applied to the system, the angular velocity, and the current angle. The fitness is set to negative so that the best fitness is 0 which is impossible to reach unless the system was set to start at the top position. The equation to determine the angular velocity from the torque applied to predict the next position is no longer needed as that is solved internally by the 3D simulation. The force is simply applied as torque to the motor controller. To evolve the network, I incorporated the running of the simulation into the fitness function for the genetic algorithm. For each iteration of the fitness function the physics client would be generated, the initial state of the system would be applied, then for each time step the position and velocity of the arm would be recorded, inputted into the network, and the output would update the torque applied to the system. The fitness is then determined by the accumulation of the instantaneous finesses multiplied by the timestep

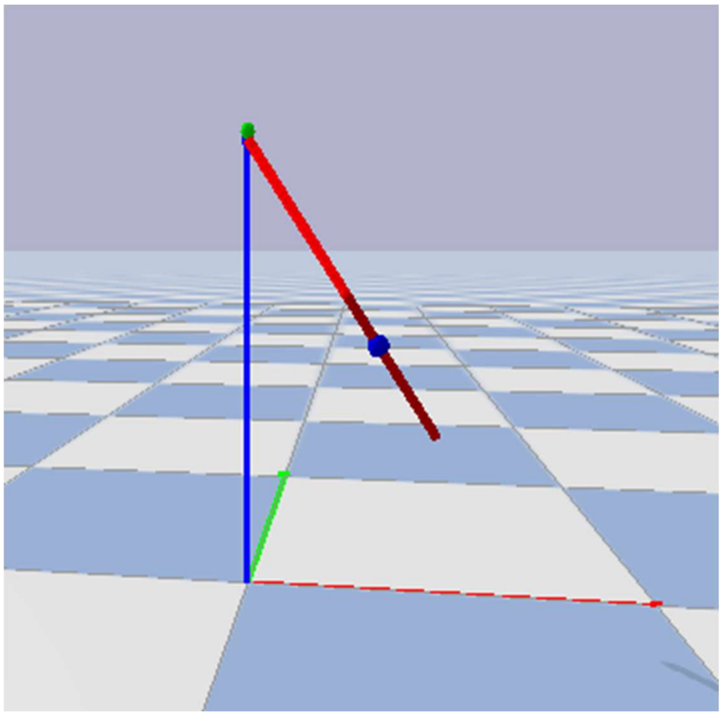

Figure 1: 3D body for inverted pendulum (Green ball = pivot point, Red cylinder = arm, Blue ball = arm tracker)

$$fitness = \theta^2 + 0.1 \cdot \dot{\theta}^2 + 0.001 \cdot u^2$$

Equation 1: θ = normalized angle, $\dot{\theta}$= angular velocity, u = torque

One of the other methods of creating the pendulum that I tested was creating it using Pyrosim. I ran into a few problems with this method, one of the issues was the inability to create a world element to link the pivot point to in order to keep it stable. Another issue was when trying to make various kinds of joints and shaped elements, entirely new commands would need to be made or the URDF would have to be changed by hand after initial generation. Another method used was using pyBullet's createMultiBody command, this skipped the generation of a URDF file and made the limbs one by one and connected them each time the code was executed. The issue with this method is that each joint had to be meticulously pieced together and the joints often had an issue of not excepting the revolute or continuous joint. Another issue was that a lot of tracking was required as the joint link index was needed for the commands.

The evolutionary algorithm was able to successfully produce a neural network that could start from the bottom and balance itself to the top position with the given force range. The fitness of 2 different (figure 2) runs show the growth of the network over time for the system. The successful behavior of the system is visualized as a graph where the values of the system and fitness are recorded for analysis (figure 3). The evolution over 4 different runs shows (figure 4) that if the initial fitness is too low that the system might get stuck and gets harder to evolve as it never reaches the top position securely. The behavior of the system is visualized as both an animation at https://youtu.be/y9MquBI3TnU and a graph (figure 5). The system constantly pushes in one direction making it unable to successfully swing up to the top position against gravity. The visualization of the tests helps ensure the network's success is accurate. The pendulum's position is clearly shown that when it reaches the top, it does not fall over, this shows that the fitness is not being beaten by some loophole within the simulation. To test the system in a non-natural state, I added in the visualization a chance for the system to be pushed randomly. The torque applied is averaged across 0 with a range of –250 to 250 because an instantaneous torque is not enough to affect the system in a meaningful way if it is too small. The graph of the behavior (figure 6) and visualization at https://youtu.be/LErTwfZzhiY show that the system can resist the outside force and if it gets toppled it is still able to correct itself. Another graph showing the results of the behavior with the force push values removed from the graph (figure 7) show the systems attempt to rectify outside force by maximizing the force at points. In the graph, the red lines going off are the results of the chance pushing on the system with such high torque compared to normal. The system swaps between positive and negative torques to balance the pendulum against the random forces.

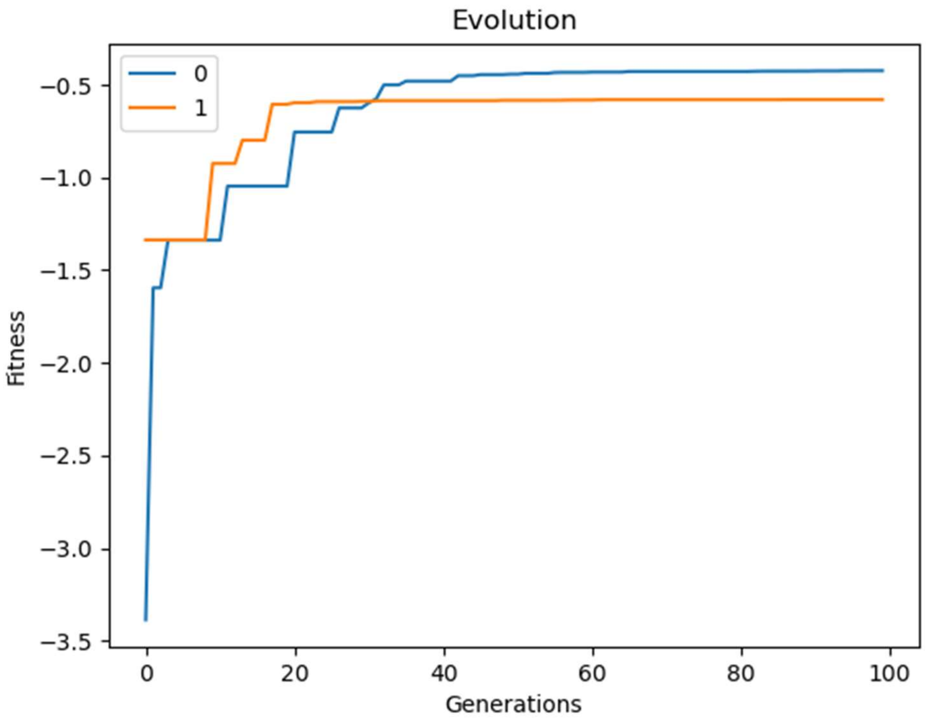

Figure 2: The fitness level for the evolution of the 2 inverted pendulum simulations each with a network with a duration of 200 steps

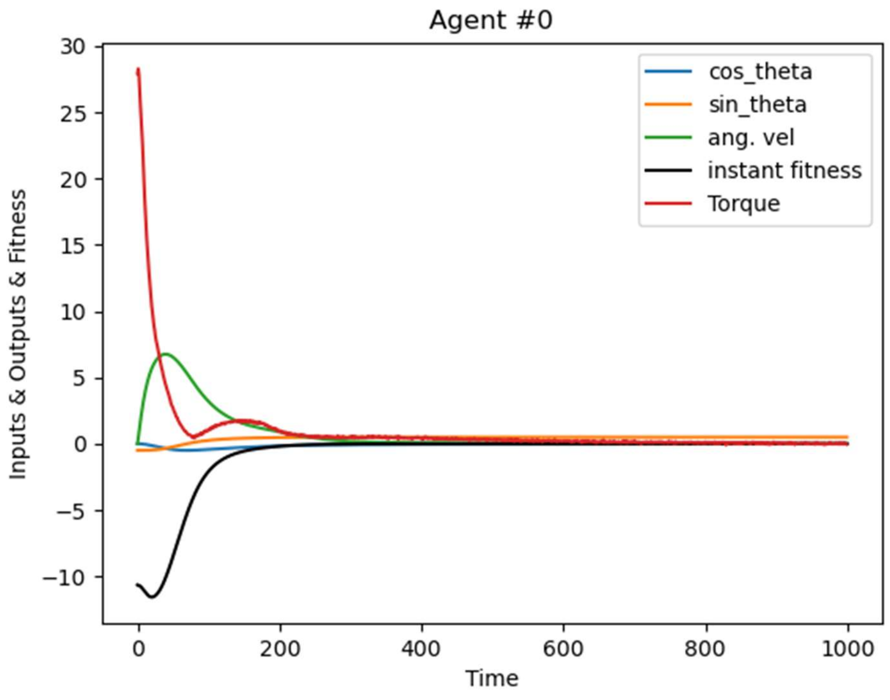

Figure 3: The behavioral simulation of 0th agent of the simulation runs from figure 2

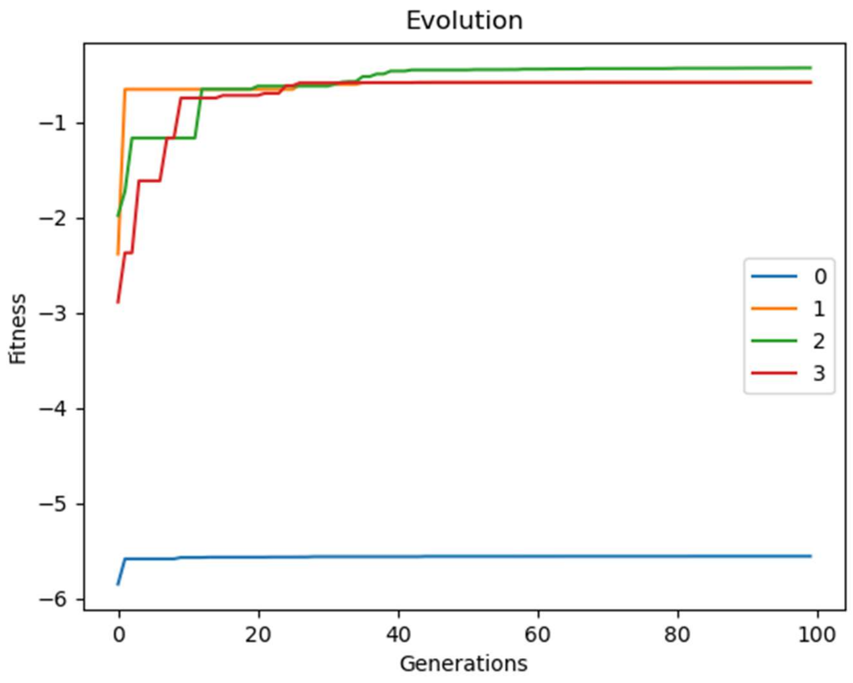

Figure 4: Fitness levels for the evolution of 4 inverted pendulum simulations each with a network with a duration of 20 steps

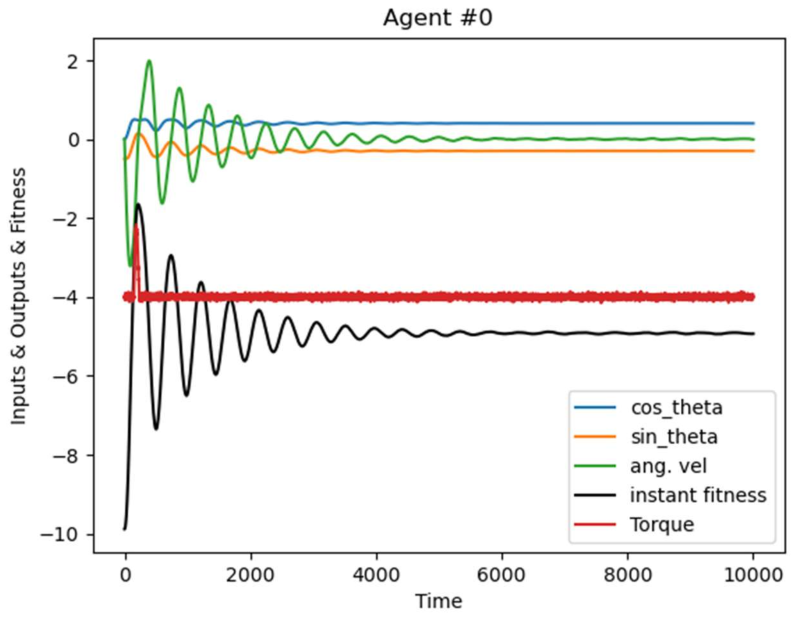

Figure 5: The behavior results of the 0th agent of the simulation group from figure 4. The torque is constantly at the maximum negative value causing the stagnation.

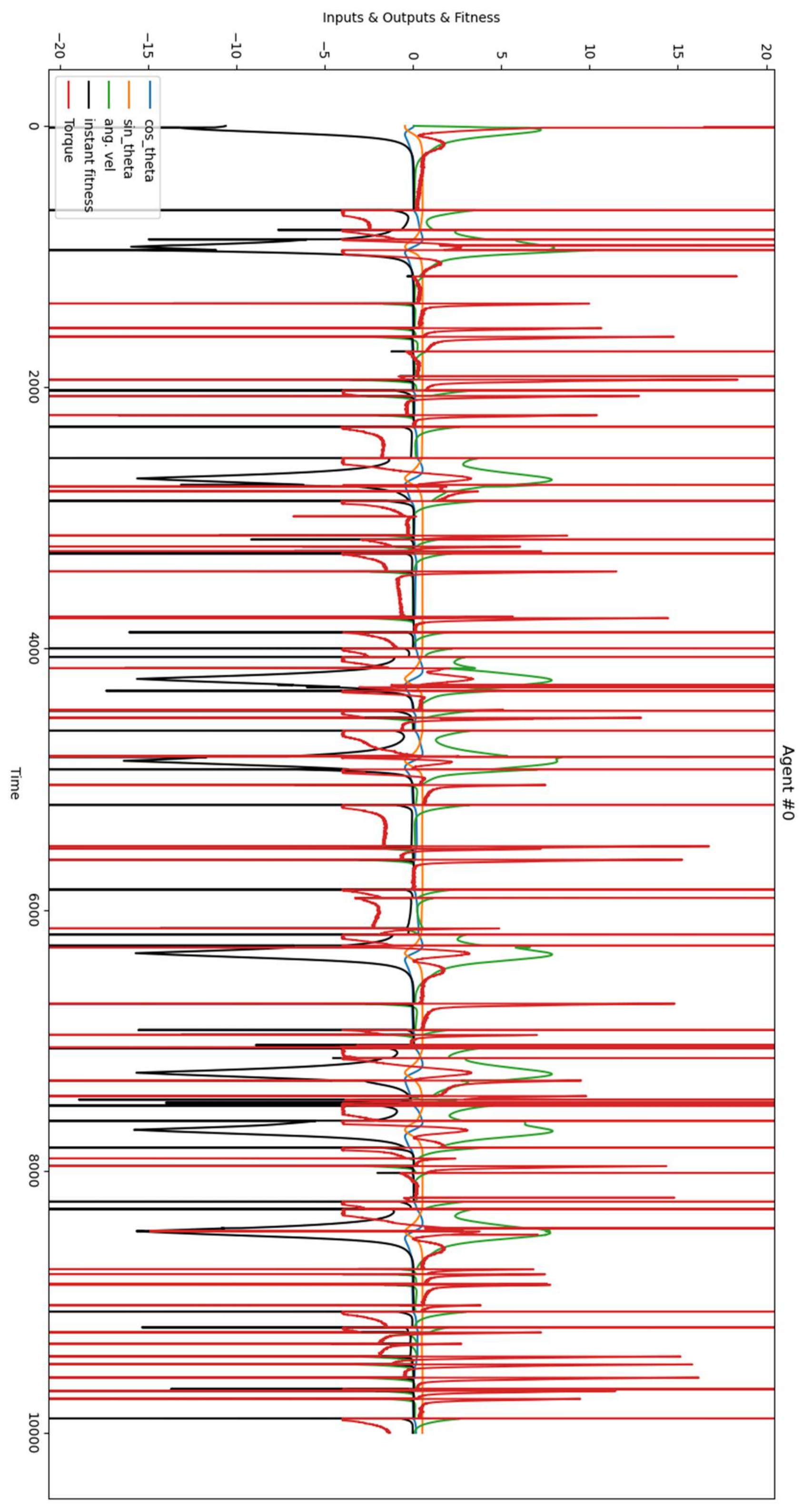

Figure 6: The behavioral simulation of 0th agent of the simulation runs from figure 2 with random outside pushes of torque on the beam

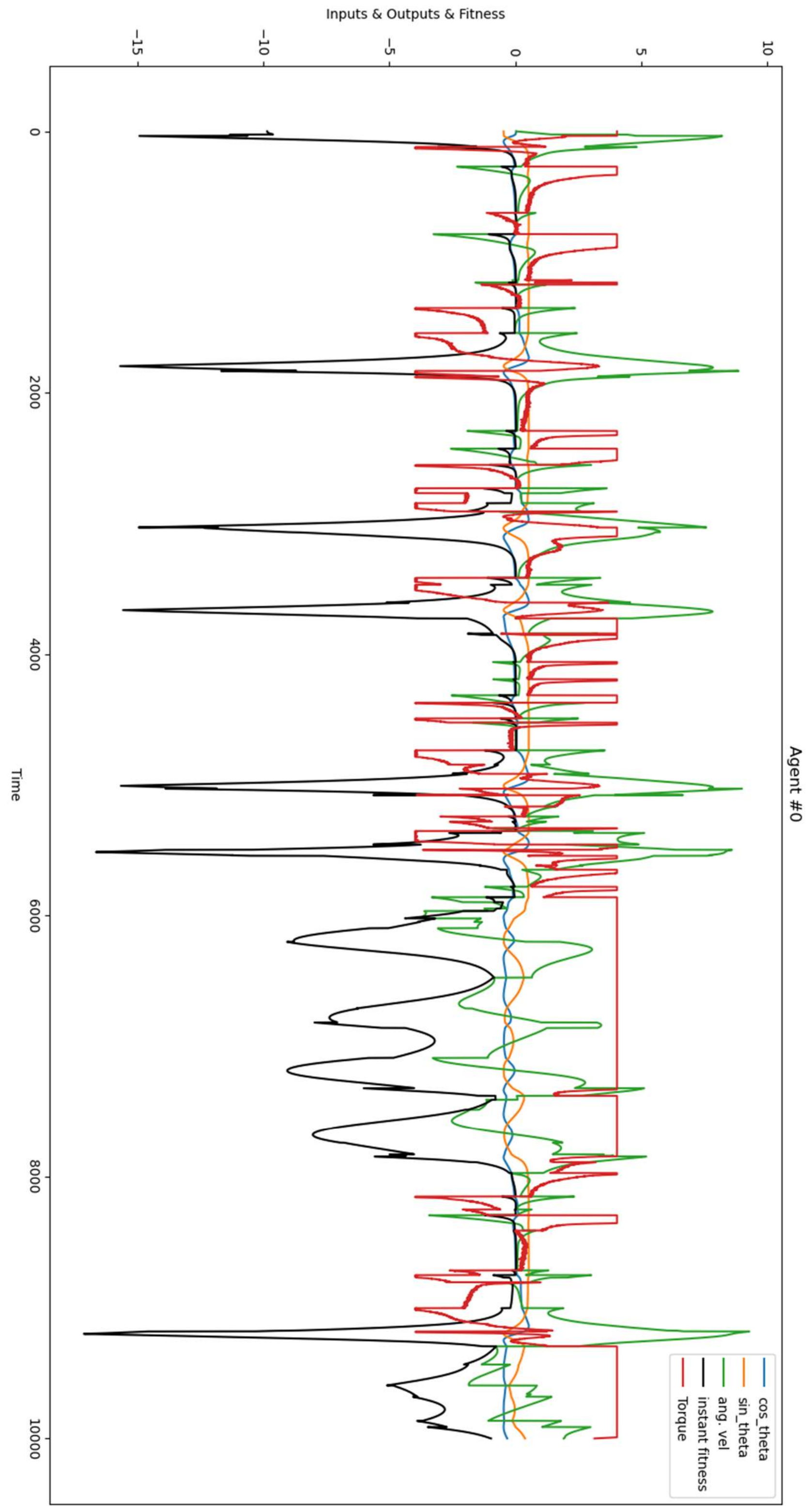

Figure 7: The behavioral simulation of 0th agent of the simulation runs from figure 2 with random outside pushes of torque on the beam without the visualization of the outside torque

In this project I was able to complete what I set out to initially which was to combine a 3D simulated body with a microbial genetic algorithm and a feed-forward neural network. The combine body, controller, and training algorithm was able to also successfully work in getting the pendulum to stay standing straight with it's center of mass over it's pivot point as initially intended. Throughout this project I was able to learn more about 3D simulated bodies, their construction, control and integration with a training network. Previously we created the simulated bodies using Pyrosim but I quickly found that it would be too simple for my use case so I had to explore different forms of body construction. I was also able to create a better method for training the network in control of the body throughout this project and it led to me better understanding how a simulated body that must be constructed each simulation would integrate with a neural network when training. Some shortcomings throughout this project was the amount of time that I spent on the physical design of the pendulum. I was often trying different libraries on generation and changed the design multiple times to see which was better. If I was to do this again I would spend less time with differing construction methods as the amount of time that I spent on how to generate the model would have been better spent on improving the visualization and testing of the simulations. If this project were to be continued there are four main ideas that I would like to go in the direction of. That would be to: expand the axis of rotation that the pendulum was able to move, as this system was only able to rotate along the y-axis; expand the body to be a double pendulum system, as the new structure would lead to likely entirely new behavior and expand the number of sensors needed; improve the environmental details of the system, like the mass of the system or defining the initial inertial values better; or to change the network or training method, as I wasn't able to use or expand on the new methods that we had learned in class such as dynamic networks or NEAT algorithms.

References

[1] https://github.com/RobotLocomotion/drake/blob/master/examples/pendulum/Pendulum.urdf

[2] Evolutionary Algorithms used in the course

[3] Various API's for URDF and pyBullet