CS 206 Final Project - Dexterous Tree

Ben Crystal

May 7, 2019

Contents

1 Introduction and Goals

The goal of the "Dexterous Tree" robot was to create a segmented pillar that would dodge objects in its environment when it falls. This could have many applications in the real world, generally relating to safety precautions. For example, on a large scale construction site, there may be several cranes that are hundreds of meters tall, and several buildings constructed of long I-beams, as can be seen in Figure 1. If these were to fall or collapse, it would be to the benefit of everyone and everything in the vicinity for the falling debris to be able to dodge both pedestrians and expensive machinery. The same goes for the example in Figure 2. When a rocket is sent into or out of orbit, the propulsion is supplemented with detaching boosters. Historically, these boosters would be scheduled to depart from the central rocket so that they would fall into the ocean and could be salvaged later. However, this approach has flaws where once the boosters are dropped, there is nothing protecting a potential boat large aquatic animal from being hit by such objects.

Figure 1: Crane and Construciton Site

Figure 2: Rocket with Departing Boosters

As an undergraduate, I had three weekly updates to accomplish, which came through smoothly. The first update was to generate a vertical robot composed of a minimum of 6 segments, as well as a randomly generated spike field with a minimum distance between spikes of the diameter of a segment of the robot's "trunk". During initial implementations and testing, I was able to implement this method. However, once I realized that pedestrians and machinery that could be in the way of a falling object could be moving in a real world application, I scrapped the minimum-distance requirement to replicate the randomness of a real-world scenario. The second weekly update was to generate a wind perturbation that caused the robot to fall in

the correct direction, and to establish the neural network and light and position sensors that helped guide the robot's fall and give feedback, and all of this was implemented properly. Finally, the third goal was to create a successful network with the original number of "trunk" segments, and then to increase and decrease the number of segments to see how well different-complexities of attachments would interact with a similar environment, and then to analyze the results. This will be discussed in the "Results" section.

Although some of the results and reactions of the robot were unexpected, this was a successful project in teaching a robot how to dodge objects when falling.

2 Implementation

To implement this robot, four major modifications to the quadruped had to occur- a modification to the phenotype, genotype, environment, and fitness function.

Starting with the phenotype, the standard tree was created by stacking cylinders on top of one another, connected by a hinge joint with 180 degree rotation (limiting the joint from moving more than 90 degrees from a linear connection between body segments. Additionally, each of these segments was equipped with both a light and a touch sensor. Originally, a cube-shaped base was implemented so that the tree would fall while "rooted", where the lowest segment would tilt towards the ground and other segments would protrude and lean further and further until it was touched. However, it was difficult to get the base to behave as expected, as it was not fixed in place and would move whenever the robot was perturbed. Eventually, this was removed as it was deemed unnecessary, as the real world applications would most likely be falling from the sky. Instead, a constant force was applied to the top segment of the tree, launching the whole body up and towards the spike field. A ten-segment tall robot can be seen here in Figure 3.

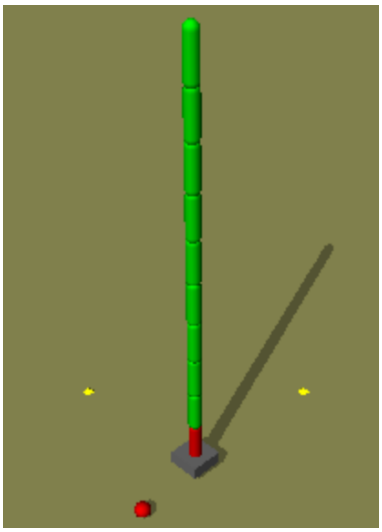

Figure 3: Ten Segment Tall Tree with Detached Base

The genotype was the next to change. Since each segment of the tree's body was equipped with both light and touch sensors, a motor was implemented into each joint between body segments. Additionally, a sensor neuron was implemented for each touch and light sensor placed into the body segments. This helped develop an interconnected network where the robot had the potential to develop an understanding of how much light it was receiving at each individual segment, as well whether or not it was in contact with a ball in the spike field.

The environment needed to be specified to the task at hand as well. Originally, three fields of 20 balls were generated to test each robot so that one randomly generated, difficult field wouldn't have an overwhelming effect on the evolution of the robot. Additionally, each of these three fields had the balls spaced a minimum distance apart from one another. However, it was realized that in a real world environment, infinite possibilities of object to dodge could exist, which very much included the movement of objects to dodge. With such, it was determined that removing the minimum distance and spawning three randomly placed environments of 20 objects would be a sufficient test for fitness. An example environment can be seen in Figure 4.

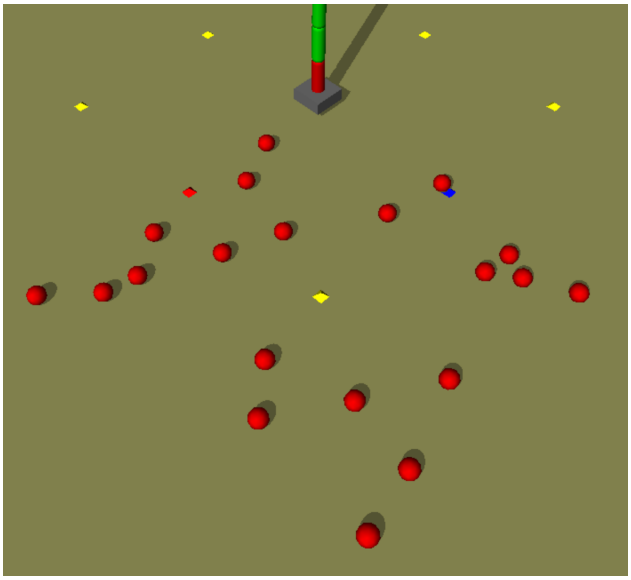

Figure 4: Randomly Generated Field of Spikes

It is important to note that collision groups were established between not only the robot and the balls, but also between the balls and each other. As such, two balls randomly spawned inside one another at a low chance would push each other apart, creating a more dynamic and chaotic environment for the robot to learn in. Additionally, each ball was spawned as a light source, which the robot used to learn to evade them.

The fitness function was fairly straight forward, but tweaking the parameters has been difficult. In essence, it was created to punish the robot for being close to light sources, and punish even harder for touching them. The fitness function is as follows:

$$fitness = fitness - light[-1] - 100 * iftouch$$

In this case, $fitness$ was initialized to 0, $light[-1]$ was the summation of the light values received by each segment, and $iftouch$ was a variable that stored the summation of time instances when any of the robot's

segments were in contact with a ball in the spike field. The 100 multiplier on the fitness function was chosen after running 100 generations of a 1, 10, 100, and 1,000 multiplier and observing the results. This function sums the fitness value of each robot through the three randomly generated environments. This value is then divided by three to normalize the robot's performance in case a particularly difficult environment was generated.

3 Results

The first generation of each forest of trees contains 10 robots with randomly generated weights in each of their synapses and motor neurons. The best performing tree is saved to an external location and can be played back, by the standards of the prior mentioned fitness function. However, since the environment is consistently randomized, a robot that scores well in the first round may have simply fallen into the right place at the the right time, expression not intended but accurate nonetheless. As such, if that "best" robot was saved and reevaluated in a different environment, it may completely ignore the arrangement of a new spike field, and fall in the same orientation directly onto its surroundings, as can be seen here in Figures 5 and 6.

Figure 5: First Generation Tree Falling on Ball (Circled)

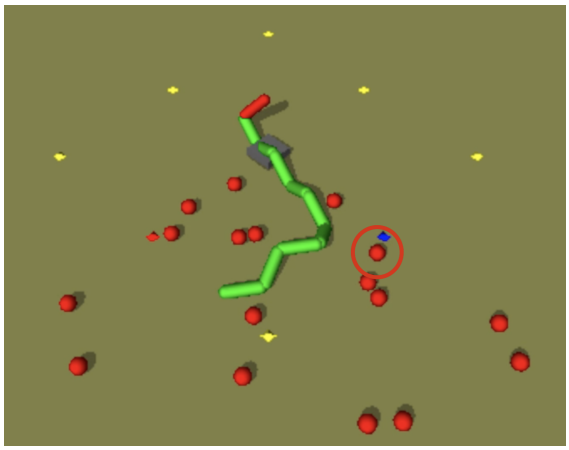

Figure 6: Same First Generation Tree, Ball Knocked Away as Proof of Collision (Circled)

On the contrary, a highly evolved robot that has experienced thousands of environments to form its neural network of connections would have most likely developed what can be thought of as a basic understanding of how to interact with its surroundings and environment. Although these particular trees were evaluated for success on one set of three random environments, after hundreds of generations of evolution, they are more likely capable of dodging balls in their next environment as well. Figures 7 and 8 below show trees that evolved in 1000 generations to roll away from the general light source population while contorting themselves to dodge balls in the way.

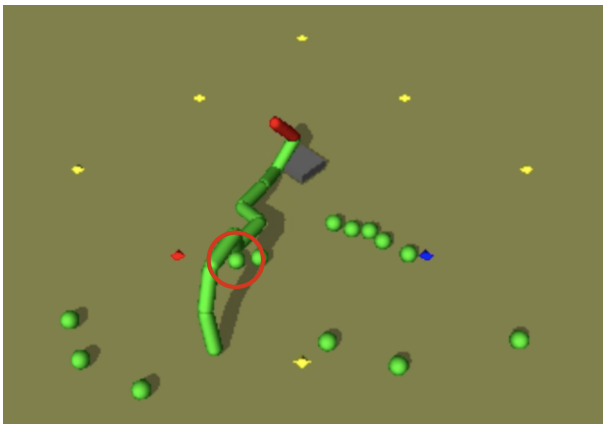

Figure 7: 1000th Generation Tree, Contorting while Rolling to Dodge Ball (Circled)

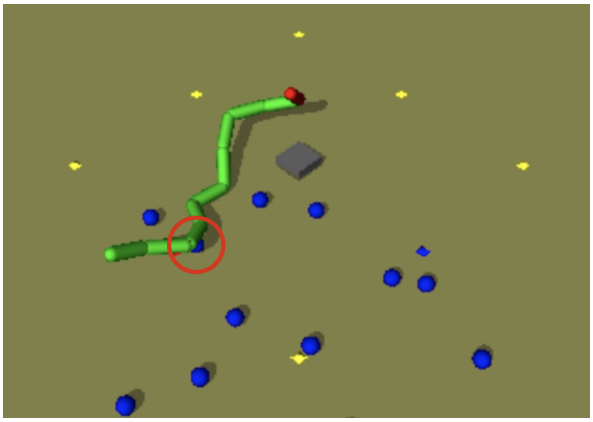

Figure 8: Same 1000th Generation Tree, Launching itself over Ball (Circled)

There still were, however, many robots that were not fit to handle their environments. Figure 9 below displays the fitness score associated with the best tree from each generation in random environments.

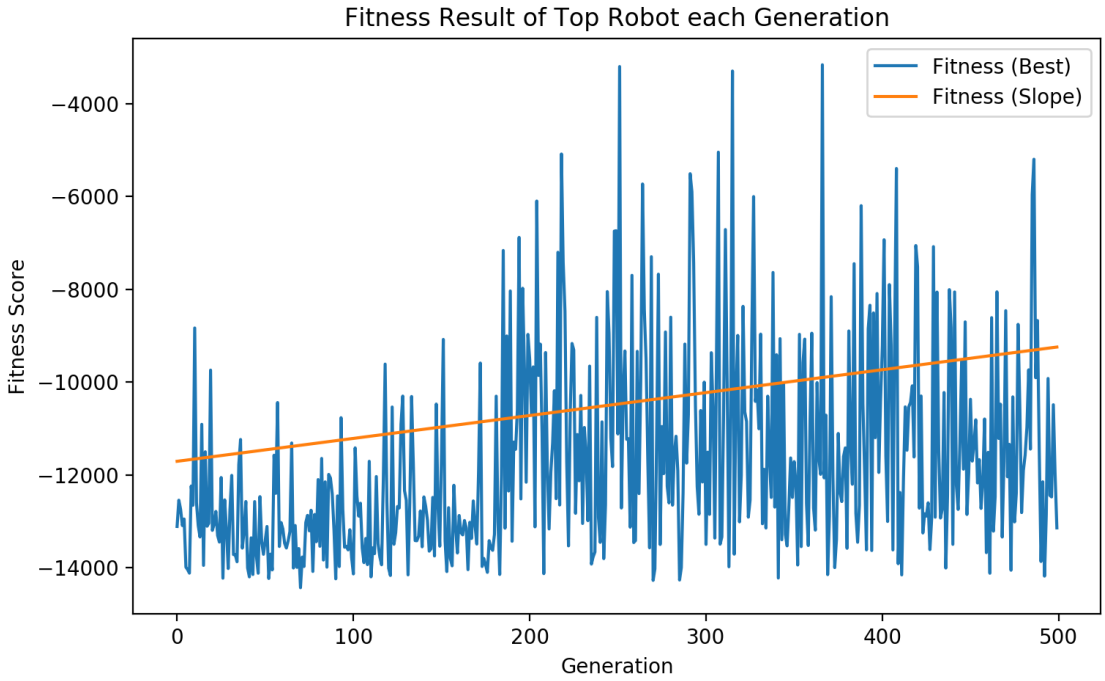

Figure 9: Fitness Results across 500 Generations (Avg. Slope $\approx$ 4.932 per Generation)

At a quick glance, Figure 9 tells the story of a robot that increasingly became more likely to be able to roll away and not make contact with any balls in the spike field, with a relatively low chance of this happening overall. However, one can notive a general spike at arount the 190th generation in some some generations

of this fitness function. This is the point where the robot learned to propel itself, and the plotting that follows shows cases where the robot succeeded (launching itself further away from the field of light) and failed (launched itself either into the field or upwards, falling back into it). These chaotic results lead for a linear regression slope to show an increase in fitness of approximately 4.932 per generation, but this was more due to greater variability in movement than learning to avoid spikes altogether. It can be noted that the frequency of "low" fitness values, e.g. below -13,500, seems to be lower, partly because of the ability to launch themselves and partly because of micro-adjustments learned by the robot in optimal spaces, like those in Figures 7 and 8. In these scenarios, the robot did learn to dodge balls in various environment that had a low density of balls all around it, skewing the light values at each body segment.

4 Discussion

The most difficult part of this project was getting the robot to model potential real-world situations. Many of the robots learned to minimize the light value received at their segments by launching themselves back up into space or out and away from the spike field, rather than by simply falling in the right spot the first time and staying still. This could be catastrophic if implemented into, say, a crane that when it falls would flail around and launch itself back into space so that it was further from the people it was avoiding. Though this problem wasn't initially considered, it should be solved before and further research is put into this project.

The easiest part of this project was constructing the tree robot, but that's not to say that it is done evolving. Scrapping the quadruped and instead creating a hinge-joint connected linear series of segments was a rather easy task that seemed to work on the first try. However, this surely isn't representative of a real world application, as the potential range of angles is 90 degrees in every direction from linear. Most likely, it is difficult to imagine something large and rigid when not in a "falling" mode being capable of flexing to these extreme angles, but it was an arbitrary decision nonetheless, as there's a possibility of real world applications being capable of bending almost to 180 degrees. Additionally, a Tao value of .5 was implemented for all motors between segments, which had very different effects both in different simulation gravities as well as with different length segmented robots, as they'd be attempting to move more mass with the same amount of force.

In the future, if this project were to continue, several changes would occur to make the simulation even better. First and foremost, a change to the fitness function is necessary. Though the results using a weight of 100 times the amount of time instances where the robot was in contact with a ball seemed to be working, it was surely not a fine tuned parameter, especially in regards to another component. This other component that would be very useful to add is either a minimization of height or a minimization in movement after a small, definitive time period. Punishing the robot for height would help skew the behavior so that once the robot hit the ground and has dodged everything successfully, it would stay there and not flail itself around, potentially destroying more of its environment. The same goes for minimizing movement after a given time: once it is on the ground, it is stationary, and it should be an environment that is safe to navigate for those around it in a real world setting, or for the spikes in this simulated environment. One odd addition conceived to the project would be to add a moving target, perhaps something that emitted darkness or a negative light value to the simulation that moved around. This could push the tree to fall towards the target, rather than just to find its own path while falling. One benefit to this in simulation is that this may account for the

movement after falling, where the negative light value may accumulate to a net gain in reward higher than that if the robot learned to launch itself back into the sky, potentially eliminating the need for changes to the fitness function. In a real world application, a target could be set up in a construction site as a "Do Not Enter" zone, where all falling debris is semi-guaranteed to hit, and thus all pedestrians and machinery would be safe if they avoided said zone.

The addition of a moving target would be relatively difficult to add given the current functions in Pyrosim, but in theory it would be easy to add. If one could generate a darkness source, or if there was another parameter besides light (i.e. sound, smell, radiation, etc.) that something could emit and a corresponding sensor that could pick it up, the only problems that would arise from this would be getting a quadruped or another robot of some sort to walk around freely and potentially avoid the spikes themselves so that when the tree would fall the light and negative light wouldn't counteract one another. One more addition that should be included into Pyrosim beyond a second radiating source that can be applied to objects is the ability to fix objects in place. With this particular project, the initial goal was for the balls in the spike field to emulate, as the name implied, a spike field. The environment was envisioned as having the balls half submerged in the ground, and no matter how the robot interacted with them and if they interacted with each other, they would stay in place and only the robot would move. However, the only solution seemed to be to turn collision off between the ball and the ground, or to raise the weight of the objects in the spike field immensely, which sometimes led to unexpected results (e.g. launching overlapping objects into the sky, etc.).

Thank you, Professor Bongard, for providing me with the passion and knowledge needed to learn about this field for my future, and thank you Jack for the help and motivation for progress on the project as well as for discussing inspiration in my future.