The Engineer - Final Project ReadMe

Nicket Mauskar

March 14, 2023

1 Summary

For the final project, I was tasked with using the parallel hill climber to design and evolve a random 3-d morphology and behavior for locomotion. Optimized through 50,000 simulations(10 random seeds that unfold for 500 generations with population size 10) with mutations in between evolution, I discovered that the most optimal phenotype for locomotion tended to be robots containing 3 links total with only 1 link containing a sensor neuron with motor neurons attached.

REFER TO THE VIDEOS BELOW TO SEE ROBOTS IN MOTION:

1. 10-second teaser gif: https://youtu.be/j5XhBtfVzho
2. Final Project Movie(Star Wars Edition): https://youtu.be/GCL6tav1fJI

Limitation 1: It's very difficult to embed videos and gifs into LateX, and the quality gets ruined whenever I tried, so I uploaded the videos to youtube and linked them into the file. Also, the gif is in the repository as well in cas you wanted to see in in gif format. I hope this is not an issue! If github does not allow you to highlight the links, they are here:
Limitation 2: I am now aware that the recordings of the creatures were not supposed to have the sidebars but these recordings were done over the past 2 weeks before I had known it was a criteria for the movie. I put a lot of effort into including as many recordings as possible, and I just do not have the time to go back and run all the simulations and get the videos without the sidebars. Each random seed run took about two hours, so the recordings were done over the course of the full reading week.

2 Reproduction Steps

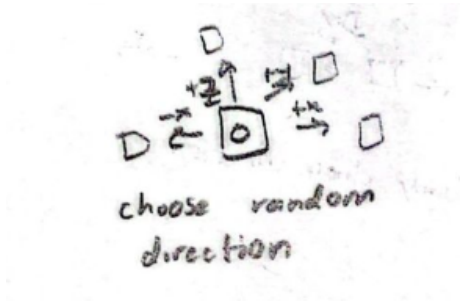
To run the code, simply run 'python search.py'. The parameters are currently set to population sizes of 10 with 500 generations. To change these parameters, please refer to the constants.py file and adjust the values $numberofGenerations$ and $populationSize$.

3 Process

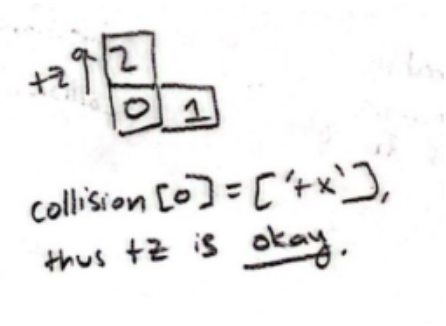
1. **Create randomly sized creature with variable sizes:** I did this by specifying the length/size of the creature to be a random number between 3 and 8. Any higher number would work, but just for the purpose of this submission, I chose 3 and 8. Also, I made the sizes of the cubes to be a random number between .75 and 1.25 to create some variability. *Code for this step found in the init method in Solution.py.*

2. **Assigned neurons randomly and color coded accordingly:** I did this by creating a method in SOLUTION which takes the random number of total links(lets call this *numlinks*), and returns a list of a random length from 1...*numlinks*, in increasing order, of random numbers from 1...*numlinks*. I then used this list as a guide to show which links will have sensor neurons and motor neurons connected to it. I also calibrated the *send_cube* function to assign the colors of the sensored links to green and the nonsensored links to blue. Joint Axis' were randomized and implemented on each joint to provide variability as well. Refer to diagram 3 for visual representation of neural network creation. *Code for this step found in Generate_Body() and the Generate_Brain() function in Solution.py*

3. **Generate a random direction and attach the link to a randomly chosen previous link with a collision check.**

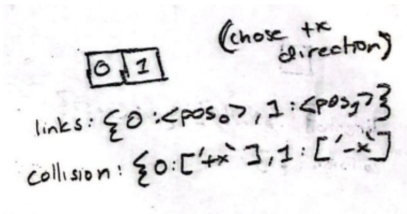


Generated a random number 1-5 which signified the directions of expansion(+x, +y, +z, -x, -y). After this, it randomly chooses a link from a dictionary of already established links (every time I add a link, I add the linkname: position into a dictionary). I also have a collision checker dictionary which keeps track of all the directions that a certain link has expanded to, and before adding another link, I check if the cube has been expanded in the randomly chosen direction already. If it has, then I choose another direction and try again. If not, then I continue to expand the creature with less chances of collisions. Refer to diagram 2 for visual representation of body creation.



Code for this step found in Generate_Body() function in Solution.py. Refer to helper functions create_direction() and collision_checker() for more information.

4. **Constantly update the links dictionary and collision dictionary.** Everytime a new link is placed, I make sure to update these two dictionaries such that any time I add a new link to a random link from before, I have all the position, joint, and axis information of that parent link such that my child link will have the most optimal position and connection with said parent link.



Code for this step found in Generate_Body() function in Solution.py. Refer to helper functions create_direction() and collision_checker() for more information.

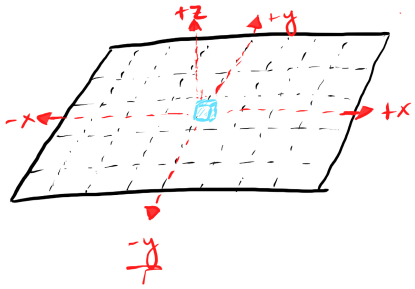
5. **Evolved/Mutated my creature across generations.** In between every generation, the child had to undergo a mutation. This would include either changing the weights of the motor neurons as in past assignments, changing the size of a random existing link in the previous iteration, changing a joint axis among one of the sensor neurons, or adding a new link to the last made link. This way, the creature will undergo mutations in both it's brain(neuron values), as well as it's actual body composition(size, shape, movement). This was done by creating a new body class defined in body.py that stores the information of all of the bodies being created through the evolution's and is able to be changed and recreated into a new body.urdf file during mutation. Refer to diagram 4 for more information. *Code for this step found in Mutate() function in Solution.py. Refer to*

body.py to see how bodies are stored into classes and generated through the mutation function.

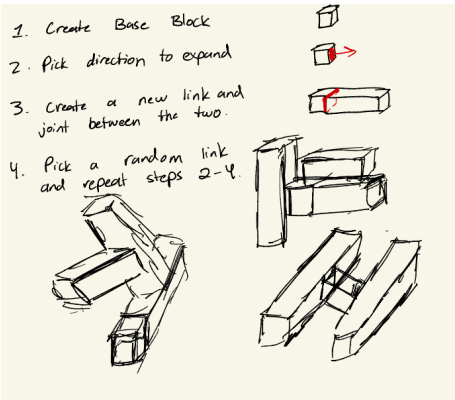
6. **Selected best option between parent and child through fitness function.** Every generation, once the child has undergone the mutation such that it is no longer identical to the parent, the simulation is then run on the child. Since the goal of this project was to design a robot that is optimal for locomotion, the fitness function I used was **maximum distance from the origin**. Thus, if the mutated child has travelled farther from the origin than the parent, then the child would become the parent. If not, then the parent would remain the same, and a different mutation would be applied in the next generation. This step repeated 500 times for 10 different parents for every run. Refer to diagram 5 for visual representation of fitness function and child/parent selection. *Code for this step found in the Evolve, Spawn, and Select functions in parallelHillClimber.py*

4 Diagrams

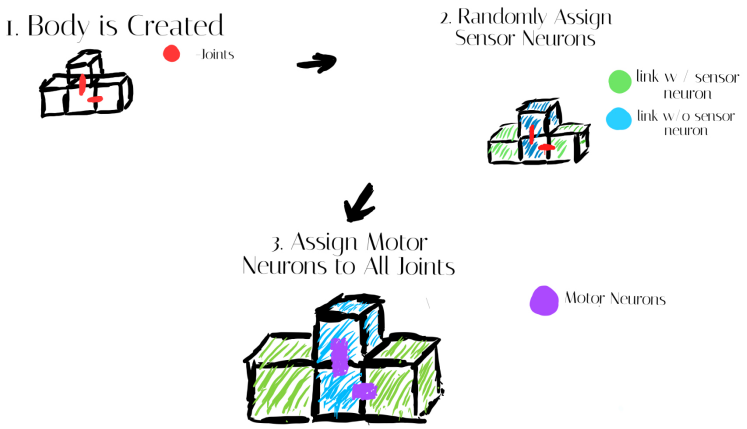
1. The Arena. Robots started creation at the origin(0, 0, 0) and moved in 5 possible directions(x, y, z, -x, -y).



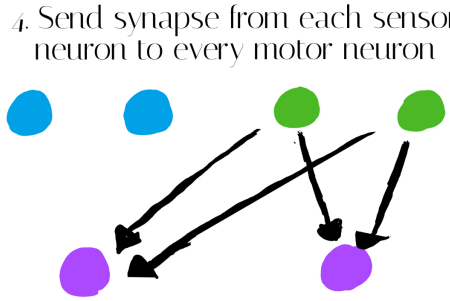
2. Design and Build of 3-d Morphologies



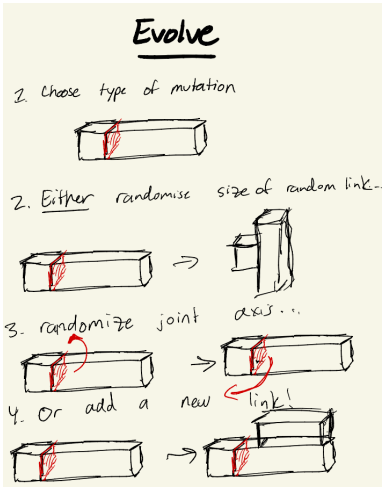
3. Design and Build of Neural Network



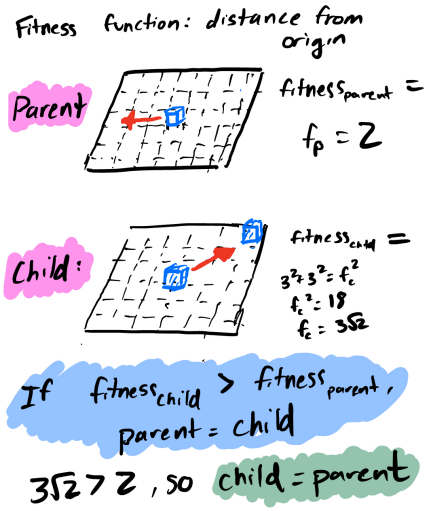
correction: second motor neuron should be attached from the left sensor neuron the the middle link(green to blue), not blue to blue.



4. Evolution/Mutation of 3-d Morphologies



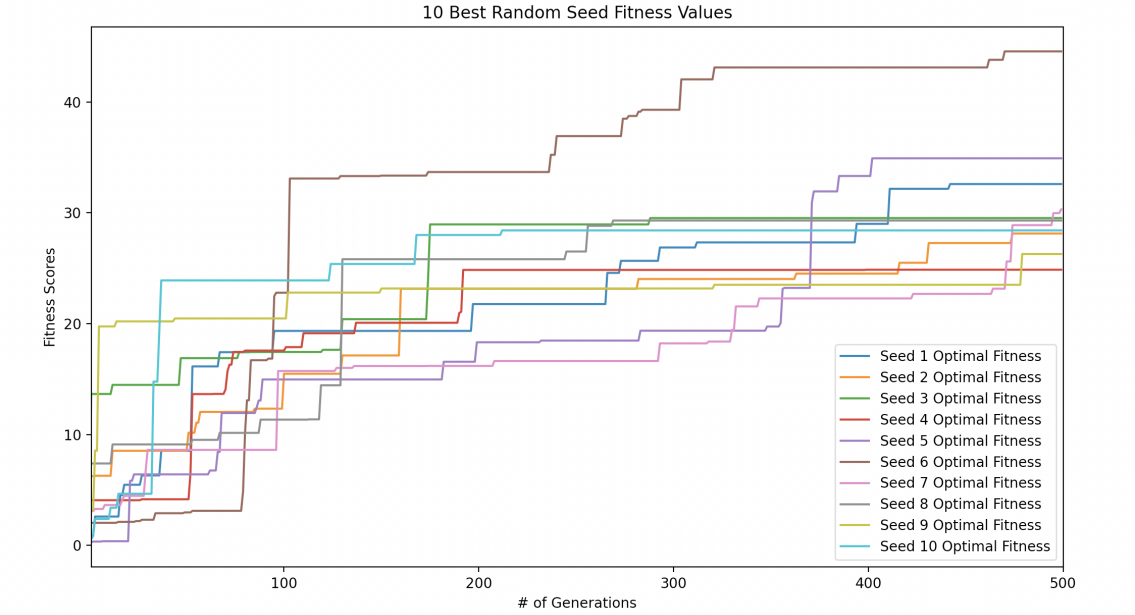
5. Fitness Function and Selection of Parent/Child through Evolution

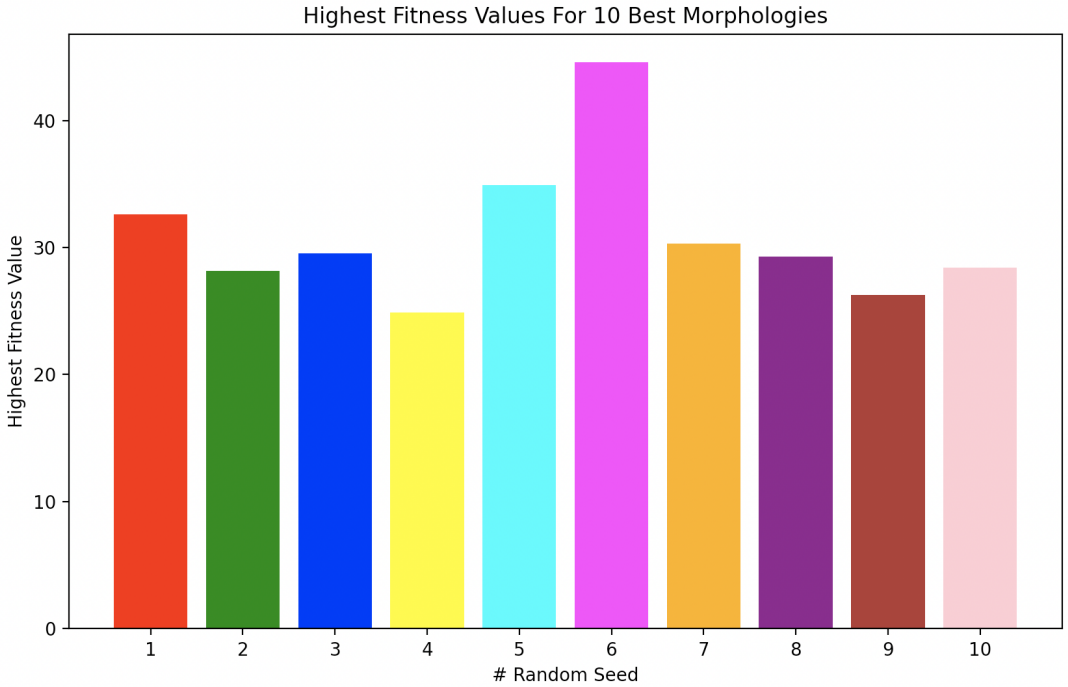


Note: Inspiration for designs from Karl Sims

5 Results

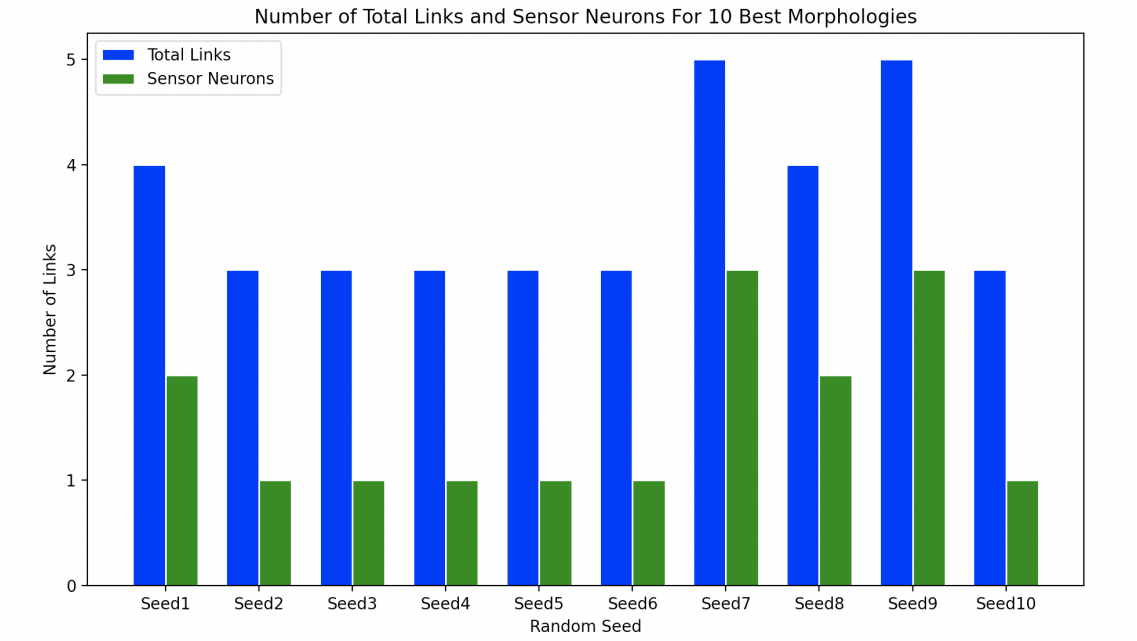
In order to produce I ran 10 random seeds for 500 generations of a population size 10. This means that these results were gathered through 50,000 simulations. Below is a graph that shows the evolution of the most optimal morphology for each of the top 10 seeds. The fitness graphs for all populations for each random seed can be seen in the appendix, and the data for all of the simulations as well as the most optimal sims can be found in grapher.py.


10 Best Random Seed Fitness Values
Fitness Scores
# of Generations
0
10
20
30
40
100
200
300
400
500
Seed 1 Optimal Fitness
Seed 2 Optimal Fitness
Seed 3 Optimal Fitness
Seed 4 Optimal Fitness
Seed 5 Optimal Fitness
Seed 6 Optimal Fitness
Seed 7 Optimal Fitness
Seed 8 Optimal Fitness
Seed 9 Optimal Fitness
Seed 10 Optimal Fitness


Highest Fitness Values For 10 Best Morphologies
Highest Fitness Value
Random Seed
0
10
20
30
40
1
2
3
4
5
6
7
8
9
10

From these graphs, I noticed a few things. First, most of the optimal morphology's ended with a fitness score from 23-30. However, we can see that the most optimal fitness was Seed 6, with a fitness score of 44. It is also interesting how each of the seeds progress over the generations in the sense that Seed 6 seemed to have huge jumps from only generations 80-120 with slight increases from 120-500, while the 4th seed seemed to slowly increase over time with little to no big jumps. Also, it is worth noting that there are many periods in all seeds where the fitness does not increase for 50-100 generations. This is a sign that the mutations may not necessarily be functional, and that sometimes it may take many different types of mutations before finding a beneficial one.

I also wanted to do a little further exploration on how the number of links and sensor neurons may have an effect on the fitness. Below is a graph that shows the number of total links and the number of sensor neurons for the best 10 morphologies.



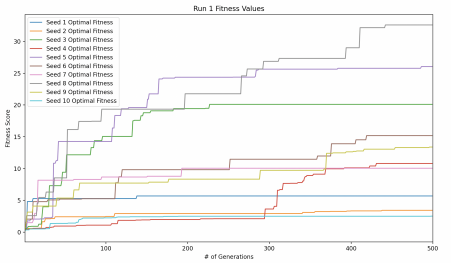
From this graph, I was able to evaluate a hypothesis/idea that I had when starting this final project: that robots with 3 total links seem to be the most optimal. We see that this this is the case from this graph, as 6 of the top 10 morphologies had 3 total links and 1 sensor neuron. Although the links had different sizes and the joints were in slightly different areas, there was definitely a common trend in that the one sensor neuron acted as a powerful motor that propelled the body forward, while the other 2 links provided stability and direction. Even in seed 7, 8, and 9, even though there were more total and sensor neurons, the functionality of the robot was very similar in that the blue

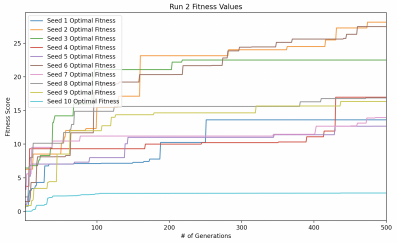
links provided stability while the green links were the ones touching the ground and propelling the robot forward.

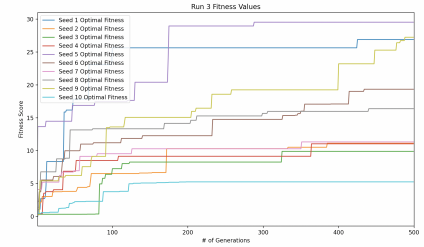
The biggest takeaway would be that more is not always merrier. More often than not, creatures with greater than 5 links, and in turn, more motors and sensor neurons, ended up not performing better than those with only 3 or 4 links and 1 sensor/motor neuron pairing. As such, the bigger the robot, the more complex the interactions between the links become, and in turn, they devolve the locomotion of the robot.

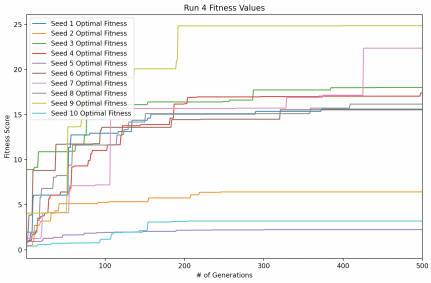
6 Appendix

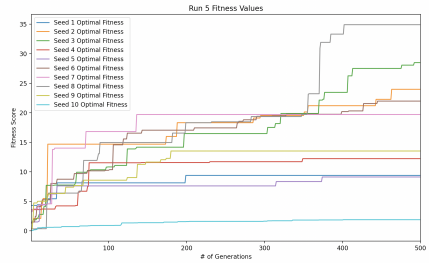
Below are the graphs from all 10 of the random seeds. The parents with the highest fitness at the end were used in the Results section's graphs as they were the most optimal evolution.

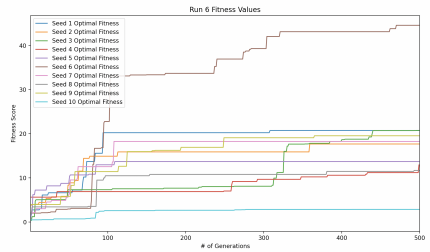


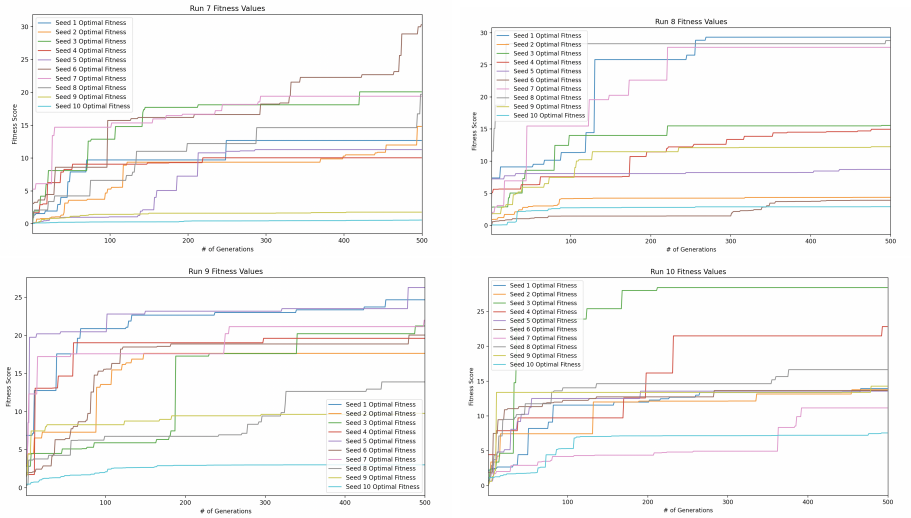












7 sources

1. r/ludobots - https://www.reddit.com/r/ludobots/
2. pyrosim - https://www.thunderheadeng.com/pyrosim