Evolving Neural Networks with NEAT to Control an X-Drive Robot

Brandan Roachell
Dept. of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, TN, USA
broachel@vols.utk.edu

Drew Friend
Dept. of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
Knoxville, TN, USA
afriend3@vols.utk.edu

***Abstract*—With the rise of self-driving technology and other artificially intelligent systems, full autonomy has become an increasingly more desirable yet achievable goal. We explored an application of genetically evolved neural networks to an X-drive robot. Directly controlling the velocities of the four motors with the available information to drive to a target position proved to be a relatively simple problem for the trained model to solve and was feasible in simulation. However, the implementation on a physical robot was far more challenging for various possible reasons.**

I. Introduction and Motivation

Since we both have a strong background in robotics, we thought it would be interesting to apply biologically-inspired algorithms to this field. We decided we would explore how a trained model might behave when presented with a driving task. X-drive has a more interesting control scheme with complicated equations involved, as seen later. Self-driving technology is on the rise, and exploring different avenues of mobility could lead to more interesting, exciting, and optimized applications of this than the traditional approach of "a modern car capable of driving itself." We sought to answer the following question: how feasible is it to evolve a neural network to control an X-drive robot entirely on its own?

X-drive, a subset of holonomic driving mechanisms commonly used in the VEX robotics competition, is a robot control scheme consisting of 4 motor groups each controlling an omni-directional wheel angled at 45° from the X and Y axes of the robot. Being holonomic, this layout allows the robot to rotate independent of its translational movement. It also gives the robot the ability of planar driving, where movement in any direction is possible without needing to change the heading of the robot. These are the equations that allow a human driver or hard-coded approach to drive directly to any point either while turning or facing the same direction. Given the multiple layers of trigonometry and simple math, we expected the model to create something similar, using a multi-layer network to emulate these equations. Given the (x, y) position of a joystick with values between -100 and 100, and a, some scaled turning power from another joystick, the X-drive control scheme is as follows:

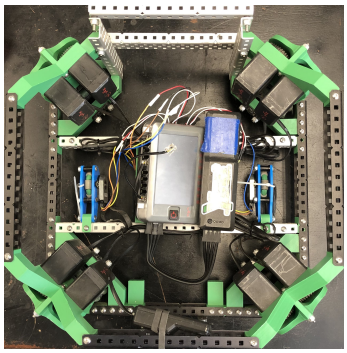


Fig. 1. Our physical X-drive robot.

$$
\begin{aligned}
x_{\text{adj}} &= -y\cos\theta - x\sin\theta \\
y_{\text{adj}} &= -y\sin\theta + x\cos\theta \\
m_1 &= a + y_{\text{adj}} + x_{\text{adj}} \\
m_2 &= a - y_{\text{adj}} + x_{\text{adj}} \\
m_3 &= a - y_{\text{adj}} - x_{\text{adj}} \\
m_4 &= a + y_{\text{adj}} - x_{\text{adj}}
\end{aligned}
$$

where "adj" stands for "adjusted," θ is the heading (angle of the "front") of the of the robot, and m_i is the desired power of motor i.

II. Related Work

In addition to driving simpler robot drivetrains with fewer wheels and more simplified controls, many of the related approaches were focused on other applications of a neural network like computer vision and path optimization. There is little to no existing work that we found regarding controlling any form of holonomic drivetrain. We also did not find anything about training robot movement entirely from a model without predefined drive code. To elaborate, our model is only able to control motor powers—it does not have access to the concepts of "turn" or "drive forward."

Further reading can be found in references [1] [7] [12] [3] [10] [6] [9] [2] [4] [8].

III. Methodology

Due to the nature of both our problem and genetic algorithms, training this model on a physical robot was not feasible within the scope of this project. We instead developed a robot simulator in Python to be used as the problem space for training. This allowed training to happen much quicker than it would have in real time, and Python's package options made it easier to implement code for our purposes. We used NEAT [5] to evolve populations of robots to reach a goal position and angle in our simulation using custom odometry and fitness code. The best network at the end of this process was saved and copied into C++, where we then fit it into the existing drive code for the robot. This workflow is visualized at a high level in Figure 2, and our implementation details can be found in our repository [11].

A single generation of training consisted of the population being initiated—either randomly for generation 1 or based on previous evaluations for all other generations. This population was then subjected to five trials, and the fitnesses found across these trials were averaged for each individual in order to create wider variance in the evaluated fitness scores and differentiate the good generalists from the models that got lucky with goal placement. A trial consists of the individual being placed at the origin with a heading of 0° and being given a goal somewhere within a distance of 10 arbitrary units (but 100 units when generating the plots in Figures 9–11) and a goal angle. The robot is then able to "drive" for 100 iterations, with each previous iteration impacting both the robot's momentum and current position, which are inputs of the network. The fitness was measured with the function below:

$$F(S) = \begin{cases} 1.5 & \text{if } d + \frac{\Delta\theta}{4\pi} < 0.05, \\ 1 - (d + \frac{\Delta\theta}{4\pi}) & \text{otherwise,} \end{cases}$$

where S is the genome, d is the distance from the goal, and $\Delta\theta$ is the absolute difference between the robot and goal angle in radians, measured along the shortest arc.

This function gave the model a bonus above the theoretically perfect score of 1 for getting within a small range of the goal and/or reaching the target angle. This allowed models that routinely do well to average notably higher than the models that fall short. It also prioritized the position over the angle. Because the maximum arc from the current angle was π, dividing by 4π reduces the range of possible angles to $[0, 0.25]$, whereas the distance was unbounded in the negative direction and bounded by 1 in the positive.

The hyperparameters of the genetic algorithm that we landed on were a population size of 500, running for 100 generations, and extremely volatile connections compared to NEAT examples. All activation functions were considered, and the probabilities for adding, deleting, or modifying the nodes and connections were relatively high, as documented in our code. These factors combined to allow our models to find the solutions early and often and spend more time in later generations increasing efficiency.

The five best models from preselected generations were saved and tested in nine driving tasks with different target positions outside of the range of distances initially trained on and a target angle of π radians with 500 iterations to drive. This allowed us to demonstrate the ability of the robot to somewhat generalize driving in any direction, as well as showcase intra-generational variance and model the robot's end behavior when given more timesteps than needed—one of the largest improvements through generations.

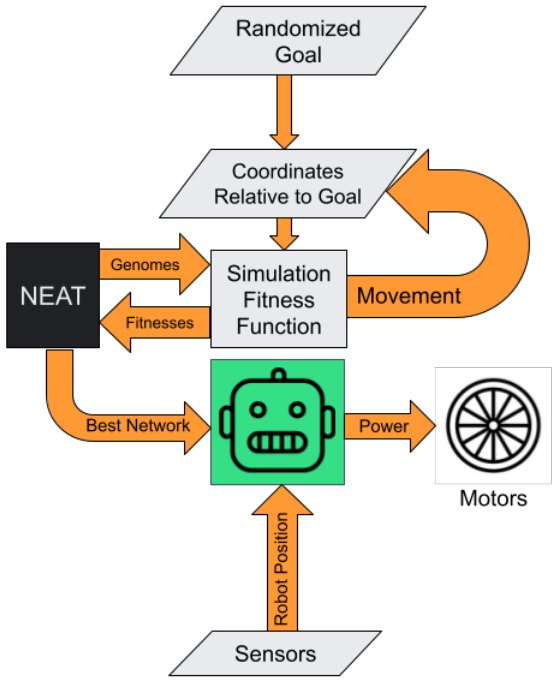


Fig. 2. A diagram of our workflow.

IV. Results

A. Model Results

We were shocked at how quickly the modeled robots were able to find the goal and remain in its vicinity. Our expectation for many of the early generations was to see robots spinning or moving in unpredictable directions given that none of the motors are directly linked to a single coordinate, and they also run against each other. However, as seen in Figure 3, a rough solution was discovered very early.

There is room for improvement in this model. It does not take the most efficient path to a point, instead traveling along a 45° transversal until it shares an x or y coordinate with the goal before following a straight line along the other axis from there. There are also certain directions that the model takes longer to move in than others, as well as directions it cannot travel in directly. Rather, it moves in a zig-zag pattern in adjacent directions, leading to an erratic movement overall (Fig. 4c).

B. Simulation Results

The simulation was completely novel software and a large hurdle to get past in training the AI. It involved translating the necessary robot code from C++ to Python, as well as creating

a fitness function, tracking the position in artificial space, and simulating basic physics concepts like inertia and friction. Overall, the results of this section are mixed. The simulation worked very well insofar as creating a reliable problem space for training purposes. The best fitness was able to increase over time (Fig. 6), and when applied to the visualization software, the models moved approximately as expected and were adaptable to moving targets as well. However, as the following section will cover, the simulation was not as true-to-life as we anticipated.

C. Translation to Physical Robot

The translation to the physical robot base did not go as cleanly as we had hoped. A small initial issue was scaling. The arbitrary units used in the simulation were on the order of magnitude of inches, whereas the encoder counts were small fractions of that. This was easily remedied with some conversions, though it complicated the translation process.

The first systemic issue we found is that the robot frame has warped in the last few months of disuse, causing issues where not all wheels would have equal pressure on the ground. This resulted in spinning out or not moving when it should be. The friction and inertia numbers were also inaccurate, resulting in skewed physics that did not work well on the robot. Additionally, the process of transferring models from the NEAT format to our C++ implementation was not easy, resulting in fewer networks tested physically and the physical testing space was being used by other members of our robotics team during the times that we were testing, decreasing the number of tests that could be run.

V. Discussion

Our work, if not completely successful in the original goal of teaching an AI to drive itself via an evolved neural network, did show great promise and proof of concept. The question of whether or not a robot could be trained to drive a holonomic drive base when given direct control of the motor powers appears to be a resounding yes, and given more time to tune the simulator and a more reliable hardware setup, these results seem very capable of making the transition to the real world.

VI. Conclusion and Future Work

The final goal of the experiment was not realized—due primarily to the scope of the project both in time and resources. However, the biologically-inspired section seems to have worked quite well. One issue that we discovered late in the process is that the bonus may be hampering the ability of the fitness scores to accurately describe performance. Because it is so high, a network that got lucky once or twice would be able to have an average fitness near what the best general-purpose networks could accomplish. The other issue with the fitness function is that it has a small range of "good" values and does not emphasize efficiency as much as we would like. Fitness values in $(0, 1.5]$ mean the network moved in the correct direction, but this is a limited range. This creates a small problem space of models that are working as intended and stops the best-performing models from rising notably past the others.

In the future, we would like to expand upon this project by continuing with our goal of implementing it on a physical robot. The steps to do this include writing a more realistic fitness function and adding more accurate physics to the simulation, such as a deadband of small values that cannot turn the wheel on the physical robot. Additionally, we would need to fix a few small issues on the robot itself to eliminate errors originating from its usual wear and tear. Although an impressive showing of ingenuity to travel in a new direction, we hope to see the robot develop methods more efficient than the erratic zig-zag motions and either straight or 45° angles it uses currently.

References

[1] R. Fierro and F.L. Lewis. Control of a nonholonomic mobile robot using neural networks. *IEEE Transactions on Neural Networks*, 9(4):589–600, 1998. doi:10.1109/72.701173.

[2] Michele Folgheraiter, Giuseppina Gini, Alessandro Nava, and Nicola Mottola. A bioinspired neural controller for a mobile robot. In *2006 IEEE International Conference on Robotics and Biomimetics*, pages 1646–1651. IEEE, 2006.

[3] Tiffany Hwu, Jacob Isbell, Nicolas Oros, and Jeffrey Krichmar. A self-driving robot using deep convolutional neural networks on neuromorphic hardware. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 635–641, 2017. doi:10.1109/IJCNN.2017.7965912.

[4] Namratha Karkera, Mansi Pednekar, Soumya Mokashi, Sushma Kore, and Prajakta Kakade. Autonomous vehicle using artificial neural network. In *Proceedings of the 4th International Conference on Advances in Science & Technology (ICAST2021)*, 2021.

[5] Alan McIntyre, Matt Kallada, Cesar G. Miguel, Carolina Feher de Silva, and Marcio Lobo Netto. neat-python. URL: https://github.com/CodeReclaimers/neat-python.

[6] Akshay Mogaveera, Ritwik Giri, Mihir Mahadik, and Anup Patil. Self driving robot using neural network. In *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, pages 1–6, 2018. doi:10.1109/ICICET.2018.8533870.

[7] Omid Mohareri, Rached Dhaouadi, and Ahmad B. Rad. Indirect adaptive tracking control of a nonholonomic mobile robot via neural networks. *Neurocomputing*, 88:54–66, 2012. Intelligent and Autonomous Systems. URL: https://www.sciencedirect.com/science/article/pii/S092523121200015X, doi:https://doi.org/10.1016/j.neucom.2011.06.035.

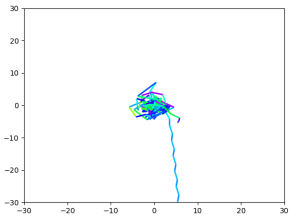
[8] Pascal Morin and Claude Samson. Control of nonholonomic mobile robots based on the transverse function approach. *IEEE Transactions on robotics*, 25(5):1058–1073, 2009.

[9] D R Parhi and M K Singh. Real-time navigational control of mobile robots using an artificial neural network. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 223(7):1713–1725, 2009. doi:10.1243/09544062JMES1410.

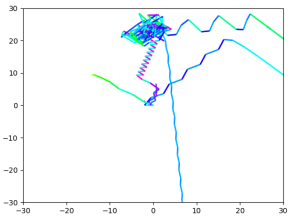
[10] D.A. Pomerleau. Progress in neural network-based vision for autonomous robot driving. In *Proceedings of the Intelligent Vehicles '92 Symposium*, pages 391–396, 1992. doi:10.1109/IVS.1992.252290.

[11] Brandan Roachell and Drew Friend. URL: https://github.com/s4mpl/Bio-Inspired_Final_Project.

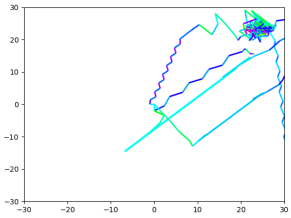
[12] M.K. Singh and D.R. Parhi. Path optimisation of a mobile robot using an artificial neural network controller. *International Journal of Systems Science*, 42(1):107–120, 2011. doi:10.1080/00207720903470155.



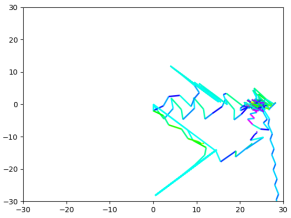
(a) $(0, 0)$



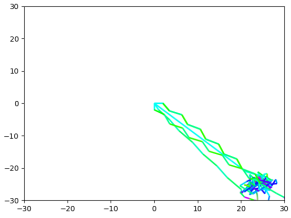
(b) $(0, 24)$



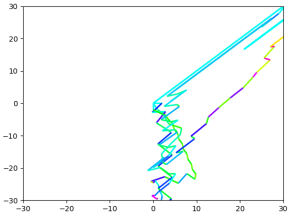
(c) $(24, 24)$



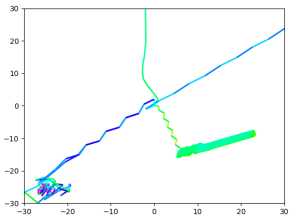
(d) $(24, 0)$



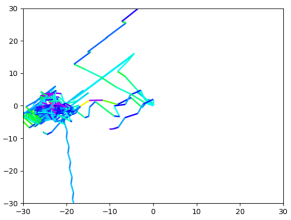
(e) $(24, -24)$



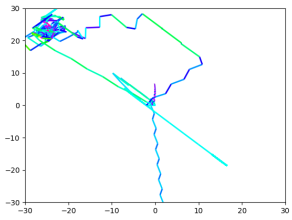
(f) $(0, -24)$



(g) $(-24, -24)$

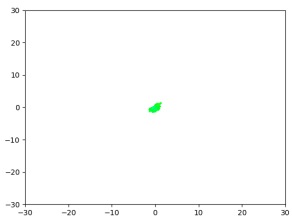


(h) $(-24, 0)$

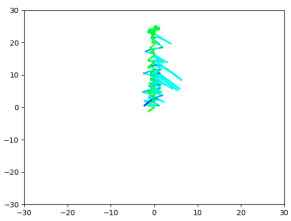


(i) $(-24, 24)$

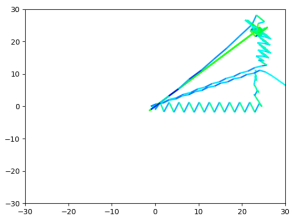
Fig. 3. Performance of the top 5 models after 10 generations for various target positions and a target angle of π radians given 500 iterations. The models were trained on random directions within only 10 units and given 100 iterations. Line segment color represents the current angle between $[-\pi, \pi]$, but the colorbar is unavailable—note that the target angle color is red.



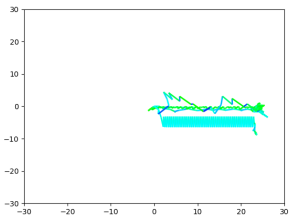
(a) $(0, 0)$



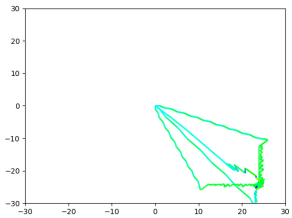
(b) $(0, 24)$



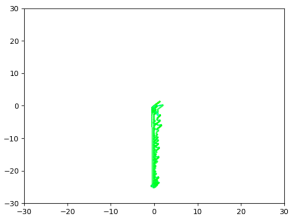
(c) $(24, 24)$



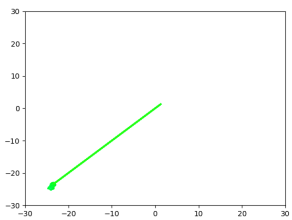
(d) $(24, 0)$



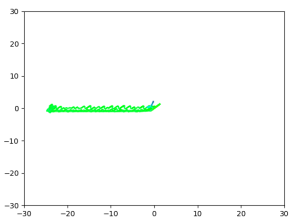
(e) $(24, -24)$



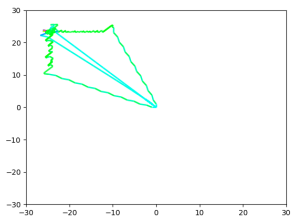
(f) $(0, -24)$



(g) $(-24, -24)$

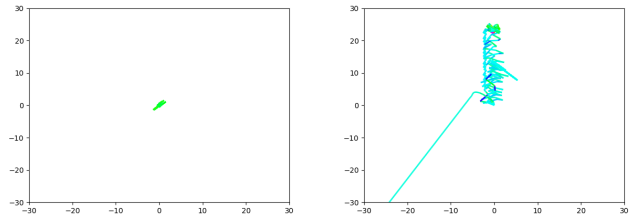


(h) $(-24, 0)$

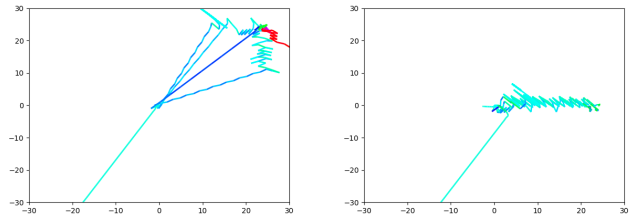


(i) $(-24, 24)$

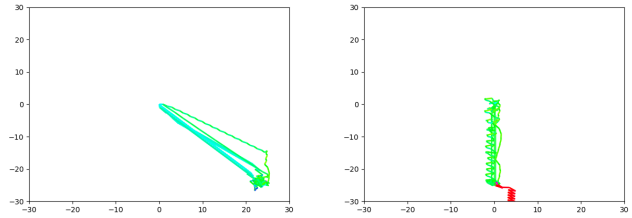
Fig. 4. Performance of the top 5 models after 50 generations for various target positions and a target angle of π radians given 500 iterations. The models were trained on random directions within only 10 units and given 100 iterations. Significant improvements in terms of efficiency and stability of end behavior.



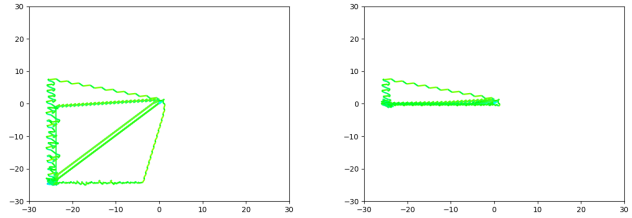
(a) $(0, 0)$ (b) $(0, 24)$



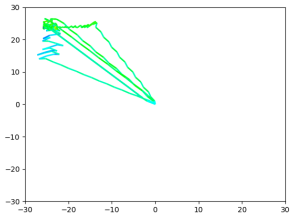
(c) $(24, 24)$ (d) $(24, 0)$



(e) $(24, -24)$ (f) $(0, -24)$



(g) $(-24, -24)$ (h) $(-24, 0)$



(i) $(-24, 24)$

Fig. 5. Performance of the top 5 models after 100 generations for various target positions and a target angle of π radians given 500 iterations. The models were trained on random directions within only 10 units and given 100 iterations. Little improvement from generation 50 in terms of efficiency and stability of end behavior, but some models began to use the extra iterations to achieve the goal angle (in red).

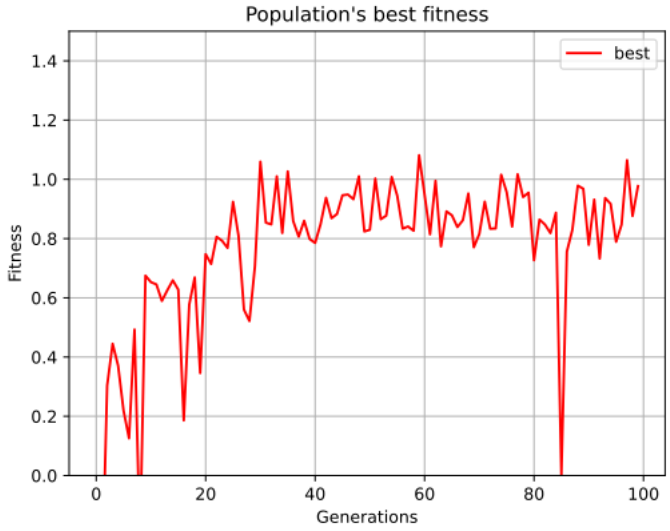


Fig. 6. The best fitnesses from each generation of the original training results (Fig. 3–8).

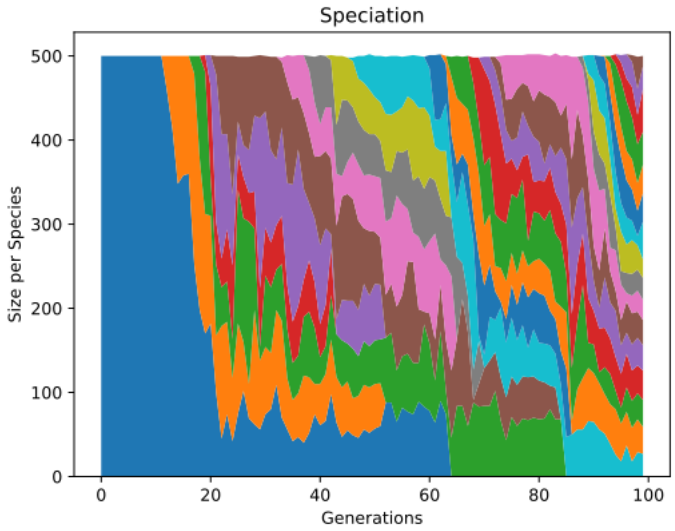


Fig. 7. The speciation of each generation in the original training results (Fig. 3–8).

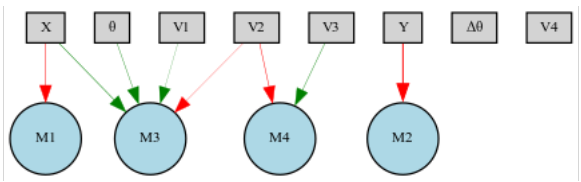


Fig. 8. The best model from the final generation of the original training results (Fig. 3–8).

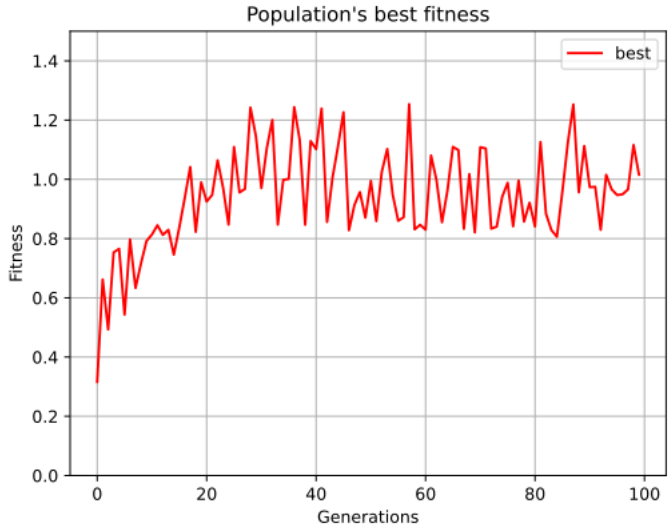


Fig. 9. The best fitnesses from each generation of the results after tweaking the simulation to our live demo environment and retraining (Fig. 9–11). An increase in robot radius and total distance it was trained over (up to 100 units) were found to give better behavior in our demo environment, which slightly differed from the simulation environment in how a robot's position was updated—using real time as a scaling factor for distance traveled in a given timestep—and was our first time fully visualizing the behavior in real time. The increased radius of the robot helped to quell the uncontrollable spinning observed in the original model.

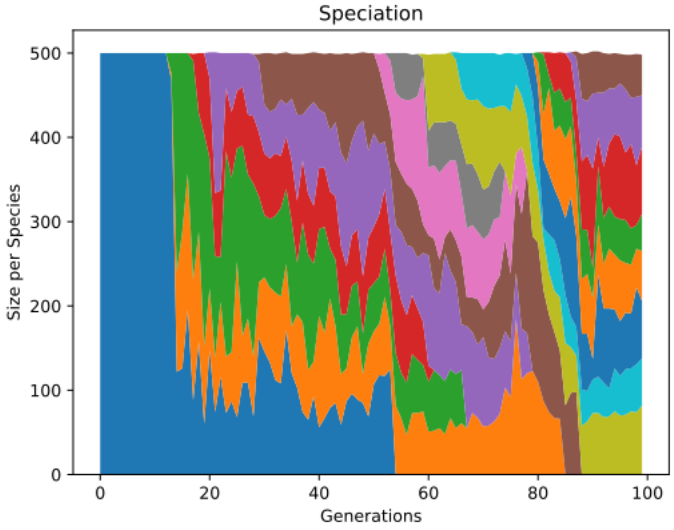


Fig. 10. The speciation of each generation in the results after tweaking the simulation to our live demo environment and retraining (Fig. 9–11).

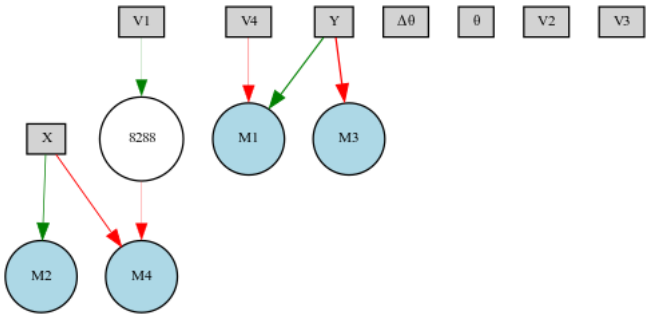


Fig. 11. The best model from the final generation of the results after tweaking the simulation to our live demo environment and retraining (Fig. 9–11).