The Reality Gap: Implementing NEAT with a Scribbler Robot

Alison Rosenzweig and Emma Remy

December 2017

Abstract

We explore the "reality gap" between experiments with a physical Scribbler robot and experiments with a simulated Scribbler robot. In particular, we try to identify and eliminate the causes of differences between the physical and simulated implementations. Our experiments use the evolutionary algorithm NEAT with both objective and novelty fitness metrics with the ultimate goal of the robot's path covering its given environment. We do not succeed at fully eliminating the reality gap, but we find that the gap can be reduced by matching the simulation and real implementation as much as possible in setup, calibrating the path length, and adding noise to the simulated sensors. We also explore whether coverage in a noisy environment (such as the physical world) is a deceptive task. We hypothesize that the remaining gap can be attributed to the oversimplification of the robot geometry in our simulation, and to more complex noise and physics in the real world than the simulation. Identifying the extent of the reality gap can inform future work with this simulation, as well as suggesting that in some research we can not rely too heavily on simulation alone as results may not be realistic. That is, while using simple simulations is useful in generating hypotheses and performing some types of research, they may not reliably reflect the results of the experiment with a physical robot.

1 Introduction

Simulations are extremely useful in the field of evolutionary robotics due to the amount of computational time that goes into often very theoretical research. However, for many long-term research goals of robotics as well as for any practical applications, physical robots are necessary. Replicating a simulated experiment in the real world is difficult. This is often because simulations reflect an "ideal" and simplified version of reality [1]. The difference between the real and simulated implementations is called the *reality gap*. As Leyden et al. describe in their 2000 paper, the reality gap can make simulated results much less useful and can increase the time and work required for an experiment contradicting the goal of simulations to make such experiments easier. Leyden et al. develop a control system for their robot in simulation only to discover that the success of this system did not transfer to the physical robot. They also find that not understanding the relationship between the simulation and reality can lead to wasting effort and resources on issues related to quirks of the simulation that would not be problematic when using the real robot in a physical environment. In our work, we also find that simplifications in the simulation can lead to many engineering challenges that would not occur if all testing occured on the physical robot. However, there are many important advantages of using robotic simulations, particularly with evolutionary algorithms, like the dramatic decrease in time required for each experiment and preventing damage

to the robot in early stages of evolution [2]. It is therefore still important to understand the extent to which we can use simulations as surrogates for physical experiments and to close the gap between the two to expand the utility of our simulated experiments.

We explore the extent of the reality gap in the context of small-scale experiments using NEAT [3], an evolutionary algorithm for developing increasingly complex neural network "brains" to maximize a fitness metric. We expect that, with fine-tuning, it will be possible to match our results with our simulated and real robots, as there is some precedent for this. In their 1995 paper, Jakobi, Husbands, and Harvey show that by adding noise to a simulation, they can evolve a strategy in simulation, download it onto a physical robot, and get nearly identical results [4]. An important similarity between our experiments is the focus on the unreliability of the physical robot's sensors. Jakobi et al. focus on the robot's sensors in their efforts to close the reality gap and allow this transfer learning to work. They attribute their success in transferring learned strategies to the physical robot in large part to their addition of noise to the simulated robot's sensors. Their experiment differs from ours in one key way – they are testing the reality gap by transferring learning done in simulation to a physical robot and environment whereas we evolve distinct robot brains in simulation and in reality and compare the results of this evolution. Instead of transferring the "brain" learned in simulation to the physical robot, we execute the same evolutionary algorithm (NEAT with an objective function or a novelty score) with both our simulated and physical robots. There is a lot of random variation inherent in these evolutionary experiments; they do not produce precisely the same results even when repeated in the same environment and with constant parameters. Thus, we attempt to generate comparable strategies rather than replicate exact results between our simulation and physical implementation. In addition to this major difference between our experimental setups, it is also important to note that Jakobi et al. use a different evolutionary algorithm (their own genetic algorithm which includes combining genotypes of individual "brains" in a population rather than simply using NEAT), different fitness functions or goals for their robot "brains" (obstacle avoidance and light seeking rather than grid coverage), and a different physical robot (the Khepera robot rather than the Scribbler).

We use Kenneth O. Stanley's Python implementation of NEAT [5] with both a simple objective function and using novelty search [6]. Our robot is a Scribbler [7]. Our simple objective function and fitness metric is grid coverage. In other words, we hope the robot "brains" will learn to cover as much of a rectangular world as possible with its path.

Minimizing the reality gap is typically a very iterative process. A researcher creates a simulated experiment, implements it in reality, identifies the reality gap, and then adjusts the simulation and/or real implementation so that they match. This process is then repeated with the goal of the simulated and real experiments converging to give the same results. Though it's often not possible for simulated and real experimental results to converge completely, minimizing the reality gap is important to the utility of the simulated experiments; if there is not a reasonable correlation between the simulated and physical robots, simulations can not be used to accelerate research or replace physical experiments where a physical robot may be harmed.

We are somewhat limited in our process, as we do not have the ability to alter our physical Scribbler robot. Instead, we can only modify the simulation to try to best approximate the physical robot, its sensors, and its surroundings. We find that the reality gap is not entirely surmountable in this context, although it can be shrunk with alterations to path length, sensor calibration, and the addition of noise to the simulation. However, we hypothesize that the reality gap is maintained by the geometry of the representation of the Scribbler in Jyro, as well as complexity of noise and

Figure 1: The Scribbler 2 robot, shown here without the additional Fluke.

physics that are not accounted for in the simulation.

2 Experiments

2.1 Methodology

We perform a series of experiments, all using the same robot, basic implementation of NEAT, and environment. In both the simulation and physical experiments, we change the fitness metric being used between a simple objective and a novelty score. In the simulation we also vary the amount of noise in the inputs.

Our robot is a Scribbler, a simple robot with two wheels and a hole in the middle to place a pen, as shown in Figure 1. We also use a Fluke, an accessory to the Scribbler that contributes features like Bluetooth capability and three infrared (IR) obstacle sensors. For our simulation, we use a simulated version of the Scribbler built into the Jyro Python Robot Simulator [8]. We define our own sensors to match those on the Fluke: three sensors on the front of the Scribbler, facing straight ahead, with a maximum obstacle detecting distance of just under 9 inches (.2 meters), as determined by sensor experiments described later.

Our real and simulated environment is a rectangular world with an underlying grid. In the simulation, we do this simply with a world class that defines the shape and borders of the environment. In the physical implementation, we use wooden blocks to limit the movement of the robot and define the boundaries of its environment. The full world is 24 inches by 42 inches (.609 by 1.07 meters), or an 8 by 14 grid of 3 inch by 3 inch squares. However, the robot's path is defined by the location of its center point (where the pen sits in the physical robot) and the shape of the robot limits its ability to reach the outer ring of grid squares. Thus, while our entire world is the full 8 by 14 grid, our effective grid (the reachable world) is 6 by 12 (see Figure 2). Similarly, we use a grid class to keep track of the underlying grid in our simulation.

Our fitness function encourages the robot to cross through as many grid squares as possible since we use the proportion of grid squares reached as a measure of *coverage* of the world. We measure this out of the effective grid, rather than the entire grid. If the robot were to pass through all of the reachable squares in the grid, it would have a coverage score of 1.0. When using NEAT

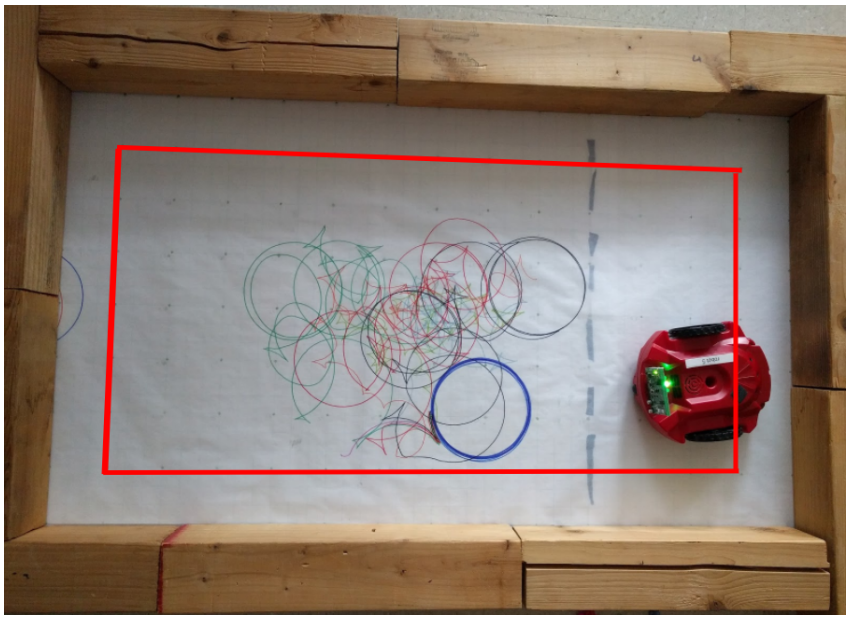

Figure 2: The world is limited by wooden blocks, but the Scribbler is only able to reach the inner grid, as shown by the red outline.

with the simple objective function, this coverage score would also be used as the fitness score.

Our implementation of NEAT uses the parameters in Table 2.1. Note that we are limited in population size and number of generations by the time-consuming nature of running experiments with a real robot rather than just a simulation. The inputs are the obstacle sensor value indicating the closest obstacle, the stall sensor (0 if not stalled, 1 if stalled), and a measure of time. When we use novelty as our fitness function, we add an additional bias input. The neural networks output a direction for the robot composed of a move and a turn.

For each individual in each generation, the robot moves for a number of steps (we pass this as an input to our network as an indication of time), we record the coverage score of the path created, and score it depending on the fitness metric we are using. In our real world implementation, we

Parameter	**Setting**
Generations	5
Population size	8
Input nodes	3*
Hidden nodes	0
Output nodes	2
Prob. to add link	0.1
Prob. to add node	0.05

Table 1: NEAT parameter settings used in the experiments. Note that the number of input nodes increases to 4 when using the novelty fitness metric, as we add a node for bias.

record each path by placing a differently colored pen in the Scribbler and having it draw its path on tracing paper. We change the paper after each generation, so we have one sheet of paper with 8 colors of paths for each generation. We start the robot at the same initial point for every path.

Our simulation is implemented with Jyro in Python 3, and our real implementation uses Myro in Python 2 [8, 9]. We transmit to the Scribbler using a Bluetooth connection.

2.2 Path Calibration

The simulated and real robots move for a different number of steps in each behavior, in order to normalize the length of the paths. The real Scribbler requires some wait time (.5 seconds) when sending move commands, as it takes some time for the robot to perform the command after it is transmitted via Bluetooth. During the wait time, the Scribbler continues to move. The simulation does not require this lag, and we noticed in preliminary experiments that the simulated paths often appear shorter than the real paths.

Rather than adding the wait time to the simulation, we calibrate the length of the paths between reality and simulation by having both robots move 5 steps straight ahead at the maximum possible speed. We compare the lengths of the resulting paths (on average, 13.75 inches with the real Scribbler and 19.7 inches with the simulated Scribbler) and determine that 87 steps with the simulation are equivalent to 100 steps with the real Scribbler (with the same velocity).

Thus, each individual in the simulation draws a path for 87 steps, and 100 steps in the real implementation.

2.3 Sensor Calibration

Our calibration of sensors is two-fold: we match the simulated sensor style to the Fluke's sensors, and we attempt to quantify the noise of the real sensors.

The Scribbler's IR sensors give a high value when they are close to an obstacle. Our simulated sensors are built from generic Jyro obstacle sensors, so we invert the simulated sensor values in order to match the real robot's sensors. We normalize the values by dividing over the highest sensor value (.2 for the simulation, 1700 for the real robot – if the normalized sensor is greater than 1, we use 1), and select the highest of the three sensor values to pass as an input into the neural network.

As has been observed in many other works exploring the reality gap, noise in the environment is a major contributor to differences in performance between reality and simulation [4]. In order to address this, we try to understand the noisiness of the Scribbler's sensors by placing the Scribbler at varying distances away from an obstacle and capturing fifty sensor values at that distance. Graphical representations of these results are in Figure 3.

Notice that the sensor values are reasonable for the first 6-9 inches, and after that are extremely random, and often just fall to zero. This is why we have limited the sensor distance in the simulation to around 9 inches. In addition, we run the simulation with varying levels of noise added to the obstacle sensors.

Noise is added by randomly generating numbers between -1 and 1, multiplying them by the level of noise (e.g. 10% noise is multiplied by .1), and adding to the sensors. From here, we select the highest sensor value as usual to be the input to the neural network.

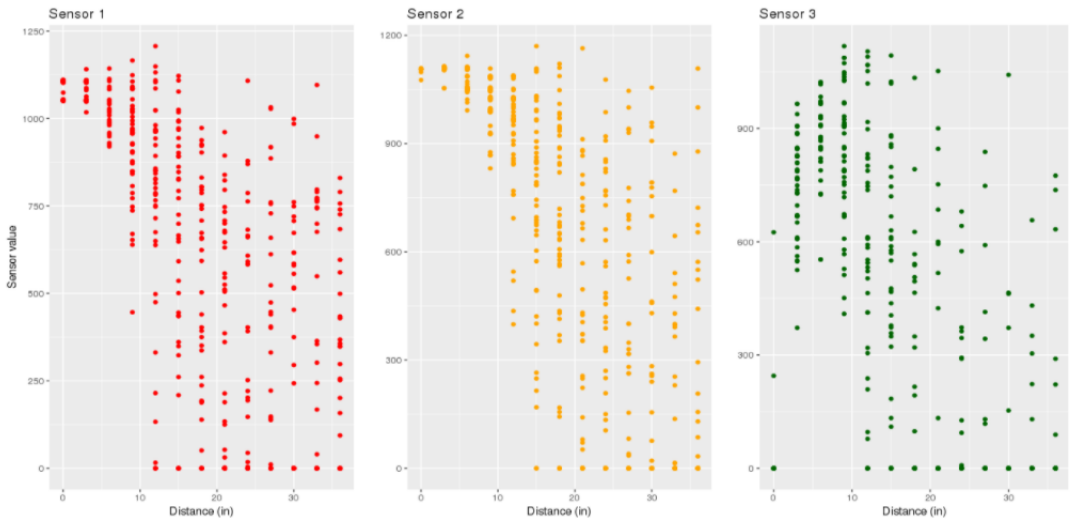

Figure 3: Each graph is the values from one sensor, for each of the three IR obstacle sensors on the Scribbler's Fluke. We vary the distance from the object by increments of 3 inches.

2.4 Fitness

We use two different fitness functions in our experiments. The first is coverage, so the overall goal and the fitness function reflect the same objective. We keep track of the coverage of an individual's path by counting the number of grid squares it crosses through. In the simulation, this is a fairly trivial task. With the real counterpart, we manually count the number of grid squares. A coverage score is considered successful if it is high, as the overall goal is to maximize coverage. This version of NEAT is called objective NEAT, and it has the potential downfall of getting stuck in local maxima instead of converging to a global maximum.

The second fitness function we use is *novelty*. Rather than rewarding individuals for high coverage scores, they are rewarded for being different from previous behaviors. This is particularly useful for tasks that are deceptive, like those with local maxima that are far from the global maximum. We perform novelty search by keeping an archive of the most novel previous behaviors. When an individual with a behavior more novel than the other behaviors in the archive is found, it replaces the oldest behavior in the archive. To measure novelty we define the behavior of an individual to be the sorted list of grid squares that individual passes through (repeats of grid squares are not included). We calculate a metric for the distance between two behaviors by summing over the pointwise distance between each element in the shorter behavior. If the behaviors differ in length, the distance is then padded with the maximum pointwise distance between two points in the world. We choose to sort the behaviors because it makes the real implementation feasible – we record the grid squares passed through by a path, rather than having to track the order and directionality of the path.

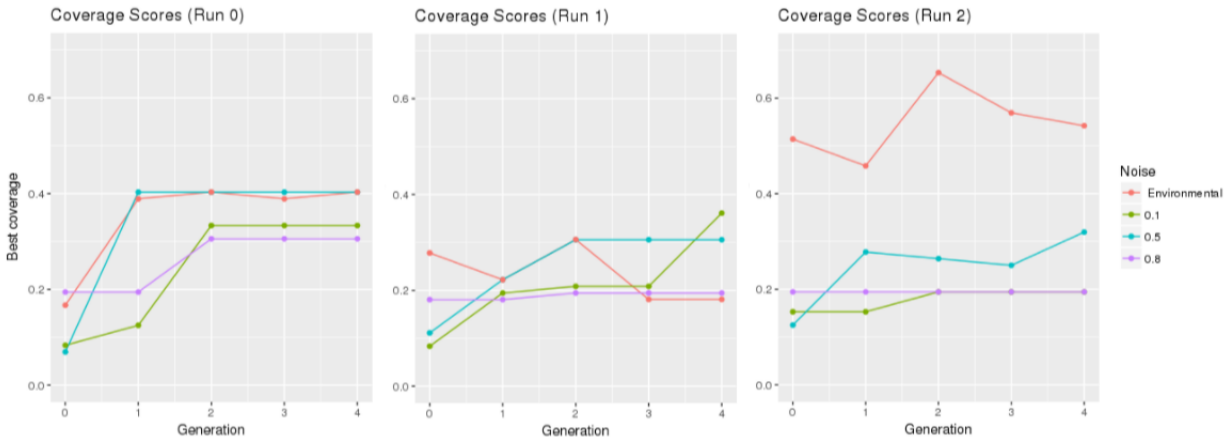

Figure 4: The four variations of objective NEAT across three runs. "Environmental" noise indicates the real run, rather than a run in simulation.

3 Results

3.1 Objective NEAT

We perform objective NEAT four ways: with the physical Scribbler, and with the simulated Scribbler with 10%, 50%, and 80% noise in the sensors. See Figure 4 for a graphical overview of our coverage scores across all variations for each of our three runs. Refer to Figure 5 for the paths from one run with the physical Scribbler, and 6 for the best paths from a run of the simulation. We notice that the robot often, after discovering one relatively well-scored behavior, gets "stuck". For example, in Figure 5, the robot learns to go along the edge of the world, but this behavior doesn't have very much room for improvement and thus the robot's coverage scores plateau.

Most of the phenotypes (neural network structures) NEAT produces are just the initial three input nodes and two output nodes, although occasionally one or two hidden nodes are added, or connections added, such as in Figure 7.

Although we do not perform enough runs to make comparisons with statistical significance, it seems that the physical robot's success is more unstable than the simulation. That is, the physical Scribbler achieves the highest coverage scores of any run, but also (if it is stuck in a different local maximum) can end up with similar or worse coverage scores than the simulated runs. The simulated runs, on the other hand, seem to have much more consistent results. This may be a result of randomness alone, but it also seems that the simulation with 50% noise performs better then the simulation with 10% or 80% noise. Note that this is the level of noise used in Leyden et al.'s successful bridging of the reality gap [4]. The simulation with 80% noise seems to have the most difficulty improving across generations, as it plateaus at a low coverage in most runs.

3.2 NEAT with Novelty

As with objective NEAT, we perform NEAT using novelty with the physical robot, and then in simulation with 10%, 50%, and 80% noise. See Table 3.2 for an overview of our results across all

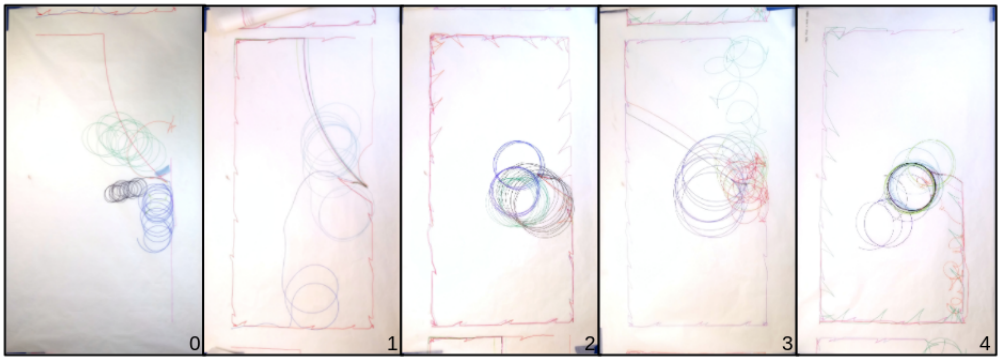

Figure 5: Each generation from a run of objective NEAT with the physical Scribbler. Every individual from each generation is represented here in a different color pen. Notice that the robot in this run learns to go along the outside of the world.

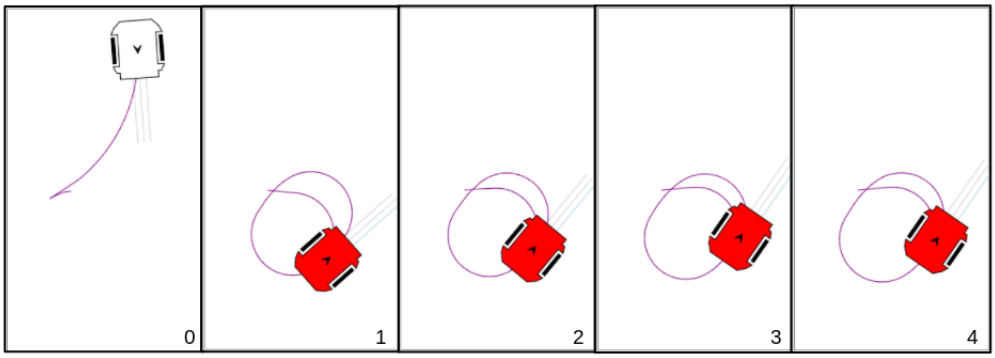

Figure 6: The highest-scoring individual from each generation from a run of objective NEAT with the simulated Scribbler. Notice that the robot learns to do a loop.

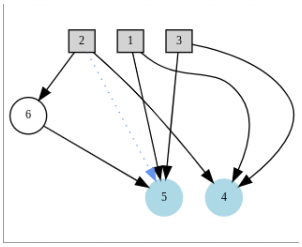

Figure 7: The phenotype from the final generation of the first run of objective NEAT with 10% noise. This is a common structure for the final generation, but phenotypes across generations are typically just the original input and output nodes.

	Run 0	Run 1	Run 2
Physical robot	.431	.583	.333
10% Noise (Sim)	.154	.222	.166
50% Noise (Sim)	.208	.432	.389
80% Noise (Sim)	.180	.333	.333

Table 2: The greatest coverage achieved in each of three runs for each method.

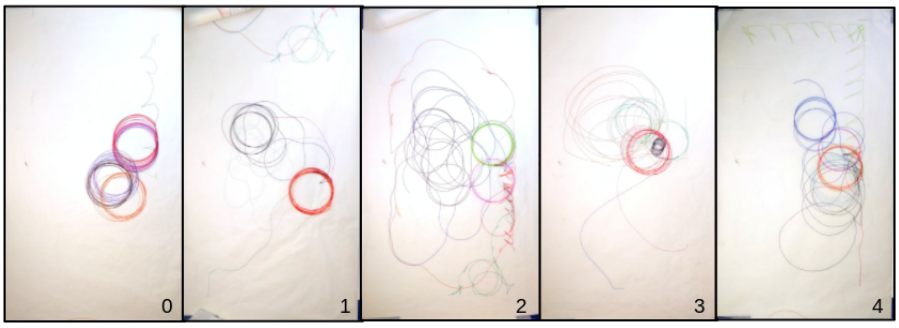

Figure 8: Each generation of individuals from one run of NEAT with novelty with the physical Scribbler. Notice the large variety in behaviors.

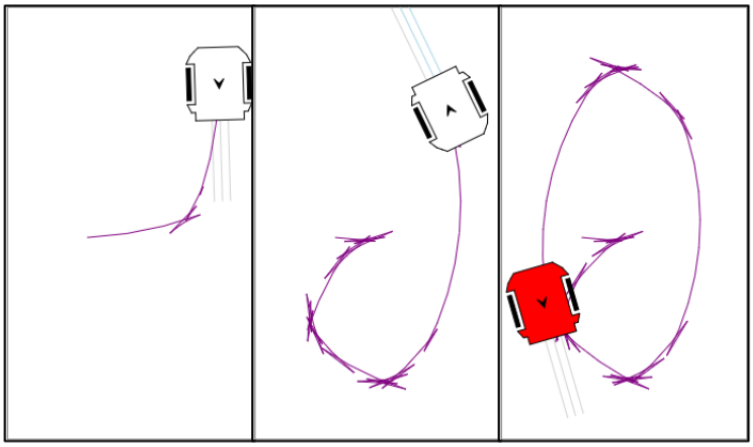

Figure 9: Some of the individuals with better coverage scores from a run of NEAT with novelty in the simulation with 50% noise.

variations. Refer to Figure 8 for all individuals in a run with the physical Scribbler, and Figure 9 for the best individuals from a simulated run with 50% noise. Notice that there is a much larger variety of behaviors with novelty than there were with objective search.

The phenotypes display similar patterns to the phenotypes for objective NEAT: typically, the resulting phenotypes just have the initial input and outputs, but sometimes a hidden node or two is added.

Overall, it seems like the physical robot performed better than the simulation with novelty as a fitness metric. However, the behaviors generated in the simulation with novelty seem more effective than the behaviors generated with objective NEAT. As before, it seems like 50% noise may perform better than other noise levels.

4 Discussion

Overall, we find that we do not fully succeed at closing the reality gap and replicating our simulated experiments with a real robot. We achieve comparable coverage scores in some cases, but not in others, and the generated paths look very different between the real implementation and the simulation.

With objective NEAT, it seems that the physical Scribbler gives more unstable results than the simulations, even with varying levels of noise. The higher coverage scores from NEAT with novelty using the physical robot suggest that it may have more volatile results here as well, as instability using is an advantage of novelty search. This suggests that our rudimentary implementation of noise is not adequate in fully capturing the noise of the real world. It also may suggest that coverage in the physical world is a more deceptive task than coverage in simulation, although we do not have enough data to make any conclusions about this.

It seems that our method of generating a novelty score biases behaviors to move toward lower-numbered grid squares. This is an interesting result but it likely does not impact the extent of the reality gap as this bias exists in both simulation and reality. This is because the behaviors are sorted and compared pointwise in calculating our novelty score. That is, if we have two behaviors that are identical except one crosses through $(0, 0)$ and the other does not, the "distance" between the two behaviors will be large because the identical portions of the behaviors are not aligned and therefore not compared when calculating distance. In contrast, if two behaviors are identical except one crosses through $(11, 5)$ and the other does not, the distance between them will be low because the identical portions of their behaviors do align in this case simply as an artifact of the method of sorting.

We also notice that the physical Scribbler is affected by other factors (outside of environmental noise) that the simulation is not. Especially notable is *drift*. The physical Scribbler's motors are not perfect, so when the motor controls would (if there were no error) have it turn in place or in a loop, it drifts slightly. This can result in artificially high coverage scores in the physical experiments for the same brain. Similarly, the physical Scribbler drifts or skids when bumping into a wall while the simulated Scribbler remains stationary once it has stalled. The simplification of the simulated Scribbler's geometry also contributes to this kind of error. While in reality, because the Scribbler is not rectangular and can slip along the wall, if the real Scribbler hits a wall and stalls it is capable of successfully turning in place. Currently, this is not possible in simulation.

From behaviors like those in Figure 6, it seems like the simulation is not as capable at creating paths with complex shapes as the physical Scribbler. We hypothesize that this is partially due to

the Jyro's representation of the Scribbler in simulation. The robot is defined with a rectangular bounding box, and a collision occurs when this bounding box intersects with an obstacle (i.e. a wall). On the other hand, the physical Scribbler has rounded corners and therefore is capable of turning against walls, skidding against them at angles, and other complex behaviors that are limited by the rectangular bounding box of the Jyro Scribbler.

5 Conclusions and future work

We find that the reality gap is not entirely surmountable with our methods. However, we are able to reduce it through an iterative process of discovering the existing gap (in our preliminary experiments, which are not discussed here), modifying the simulation and physical environment to account for these differences, and again performing our experiments. The next step is to consider the gaps that, in this work, we discover still exist, and modify our implementations to account for them.

One aspect of this is the geometry of the Scribbler's representation in Jyro. Currently, it is represented using a simple rectangular bounding box, which is not accurate to the shape of the physical robot. There are two ways we can approach this: we can modify the representation of the Scribbler in simulation so that its bounds are closer in shape to the physical Scribbler, or we can modify the shape of the physical Scribbler. In modifying the simulation, we might use several bounding boxes to represent the bounds, rather than a single rectangle, to allow for more complex turns against walls. We could also accomplish this by replacing the bounding box with a bounding circle or (with more complex adjustments to the Jyro code) a bounding polygon. In modifying the physical robot, we might place a physical representation of the simulated bounding box around the Scribbler, such as a cardboard rectangle. In doing so, we hope to match the turning abilities of the simulated and physical robots.

It seems that our representation of noise is also not sufficient in capturing the complexity of noise in the real world, based on the instability of the results of our physical experiments. In order to adjust this, we may use a more complex function for calculating noise, or try to fully quantify the noise discovered in the experiments displayed in Figure 3. We also notice that the physics of the real world is not entirely represented in the simulation. We may try to surmount this by coding some noise into the motor controls or more directly into the position of our simulated robot to account for the drift observed with the real Scribbler.

In our future work, we would like to perform many more runs of each method in order to allow for robust statistical comparisons, as we are not able to do this here. We can also modify the parameters of our implementation of NEAT in order to allow for the evolution of more complex phenotypes in a small number of generations, as this may give us more successful (or at least more interesting) results. Additionally, we should alter our calculation of the novelty score to prevent the bias toward moving toward lower-numbered grid squares. We can do this by not sorting the behaviors, or by using a different algorithm for comparing distance (such as sequence alignment). We may also transfer neural networks from the simulation to the physical implementation, as in Jakobi et al., as this will also allow for more robust comparisons between simulation and reality [4].

Overall, we discover that the original, unmodified simulation is not an accurate representation of the physical world. This suggests that, in future experiments using this simulation, researchers can not assume that experimental results in simulation will transfer to the physical world.

References

[1] Daniel Toal Mark Leyden and Colin Flanagan. Pitfalls of simulation for mobile robot controller development. *IFAC Proceedings*, 2000.

[2] D. A. Sofge, M. A. Potter, M. D. Bugajska, and A. C. Schultz. Challenges and opportunities of evolutionary robotics. *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2003.

[3] Kenneth Stanley. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21, 2004.

[4] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. *European Conference on Artificial Life*, 2005.

[5] The NEAT users page. www.cs.ucf.edu/∼kstanley/neat.html. Accessed: 2014-05-01.

[6] Joel Lehman and Kenneth Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2), 2011.

[7] The scribbler robot. www.parallax.com/product/28136. Accessed: 2017-12.

[8] Jyro python robot simulator. jyro.readthedocs.io/en/latest/Jyro%20Simulator.html. Accessed: 2017-12.

[9] Myro reference manual. wiki.roboteducation.org/Myro. Accessed: 2017-12.