Efficiency Through GPU-based Co-Evolution of Control and Pose in Evolutionary Robotics

Kasper Stoy

IT University of Copenhagen
ksty@itu.dk

Abstract

A key challenge in evolutionary robotics is the computational cost of evolutionary runs. The high computational cost forces researchers to rely on power-hungry computer clusters and, even with these, researchers often are faced with long evaluation cycles that make development of evolutionary experiments a time consuming and tedious effort. In this paper we address this challenge on two fronts. We have developed an evolutionary robotic engine where all individuals are evaluated in parallel using a thread-based implementation on a graphical processing unit (GPU). This engine allows us to run an evolutionary robotics experiment in seconds on a modest laptop. The second avenue of exploration is that we have used this engine to study the role of initial robot poses in fitness evaluation. We find that if we co-evolve initial pose and controller competitively, we can reduce the evaluation period of individuals significantly. Combined the evolutionary robotics engine and the co-evolutionary approach are significant demonstrations of how to make evolutionary robotics more computationally efficient.

Introduction

In evolutionary robotics (Nolfi and Floreano, 2000; Doncieux et al., 2015) it is common to take advantage of the parallel processing power of graphics processing units (GPUs). However, often it is sub-components that are accelerated such as the simulator or maybe the evaluation of a neural network controller. However, individuals in a population are still predominately evaluated sequentially. Instead, what we propose is that all individuals in a population are evaluated in parallel on a GPU. This has drastic speedup potential because we can evaluate n individuals of a population on a single GPU. The enabling technology is that modern GPU allows for implementation of complex algorithms. Although one still must observe some restrictions such as avoiding branching in the code if possible. Doing this we can do full evolutionary runs in the order of seconds even on low-cost laptops making evolutionary robotics more accessible. We follow a general trend in machine learning where increased parallelisation leads to dramatic speed-ups. A prominent example of this is (Rudin et al., 2021) that demonstrate dramatic speed-ups on a reinforcement learning walking task.

Using this GPU-accelerated evolutionary robotics engine we continue our work started in (Stoy, 2021). The work is done in the context of the canonical task of evolving an obstacle avoidance behaviour for a simple differential-drive mobile robot. We use a challenging variation of this task where obstacles are rare due to the environment being a large, empty arena. The observation we build our work on is that it is important to consider where in the environment a robot is placed at the onset of fitness evaluation. If, for instance, an individual is placed far from obstacles either the evaluation will be unproductive, or the evaluation period must be so long that the robot actually encounters an obstacle as part of a fitness evaluation. As pointed out in our initial paper the solution is not to place the robot in a fixed position close to an obstacle, because this will lead to overfitting and may prevent the robot from handling obstacles approached from other angles or distances. Conventionally, the solution is to place the robot at a random position. While this works it also means that many fitness evaluations are counter-productive e.g., an individual scoring high fitness simply because it started far from obstacles and not because it has learned the desired behaviour. Instead, we proposed to co-evolve the initial position with the controller in a competitive setup. The idea is that the co-evolutionary system will try to find initial positions which exposes the weakness of the controller at a given time in the evolutionary process. We found in previous work that this is indeed the case. The contribution in this paper is that we show that we can reduce the evaluation period dramatically from 200 seconds of simulated time to 25 seconds of simulated time.

In conclusion we find that with the combination of a GPU-accelerated evolutionary robotics engine and a significant reduction in evaluation period due to co-evolution of initial pose and controller we can make a highly efficient evolutionary robotics setup.

Related work

While work on accelerating aspects of robotics such as simulators (Liang et al., 2018), visions systems, and neural network controllers (Pierson and Gashler, 2017) are abundant

the use of GPUs to accelerate a full evolutionary robotics setup is not explored. One step in this direction is in work by Makoviychuk et al. (2021) that demonstrates significant effect of putting both simulation and neural network controller on a GPU. However, compared to our work only one individual can be evaluated at a time. Another related piece of work is that of Ohkura et al. (2014) who implemented an evolutionary swarm robotics setup on a GPU. However, again here the individual in this case the swarm is evaluated sequentially.

Controlling the initial pose can be viewed as a simple way to control the difficulty of an evolutionary challenge. From this point of view the approach belongs to a small class of approaches that tries to make learning easier by ramping up complexity as the evolutionary process develops. The simplest is incremental evolution where the task itself is made more complex over evolutionary timescales (Gomez and Miikkulainen, 1997; Rossi and Eiben, 2014). Similarly, it is also possible to make the environment (Wang et al., 2019) or the robot's morphology (Bongard, 2011) change over evolutionary time. However, these approaches require encoding of the task or environment to be used by the evolutionary process and thus are complex, but arguably can also handle more complex, open-ended problems. In contrast, we only encode the pose which is simple and practical and well suited for problems where the task-environment is given.

An interesting development is that robots start to have GPUs on-board making it possible to potentially run evolutionary runs in real-time on on-board the robots themselves (Jones et al., 2015).

Parallel Implementation

We programmed our evolutionary robotics engine from scratch and based it on the Khepera IV robot (Soares et al., 2016) which has eight infrared sensors. For the environment we use a walled, square arena with varying side lengths. In the following we describe how elements of the evolutionary engine is assigned to threads on the GPU. The evaluation of an individual consists of running the following update loop:

Sensor update. The robot has eight sensors for each sensor a GPU thread is responsible for finding the nearest element of the environment (modelled as a polygon) and calculate the corresponding distance.

Controller update. The controller is a simple neural network which maps sensor inputs to motor outputs. A single thread is responsible for calculating this mapping.

Simulation update. A simple kinematic simulation of the differential drive robot is implemented which is calculated by one thread.

Collision update. A single thread checks for a collision.

This update loop is run in parallel for an entire generation of individuals meaning that for most update steps a number of threads equal to the number of individuals in a generation is used. The only exception is the sensor update which uses eight threads per individual. This means that for our 256 individuals to run in parallel we need 8 times 256 equals 2048 threads. This is easily achievable as even our Quadro T1000 laptop GPU has 896 cuda cores of 32 threads each which equals a total of 28672 threads. Hence, we have many more threads for further parallelism if desired. The parallel implementation allows us to update all our 256 individuals in parallel and thus cut computation time by 99.6%.

While not the focus here we have also implemented the evolutionary algorithm itself on the GPU. However, this is not critical as most of the computation time is spent in the update loop described above where the individuals are evaluated. The update loop runs for every $0.1s$ in our implementation which means that for a simulated time period of $200s$ the update loop is repeated 2000 times.

Experimental Setup

The evolutionary robotic engine follows that of Mondada and Floreano (1996). The controller consists of two perceptrons with eight inputs, two recurrent connections between the outputs and one bias unit. The corresponding chromosome consists of 22 real-valued genes encoding the weights of the neural network. For the initial configuration population we use a chromosome of 3 genes encoding position and orientation. The fitness function is summed for each time step and is calculated as:

$$fitness = V(1 - \sqrt{\Delta v})(1 - i) \quad (1)$$

Where V is the average velocity, Δv is the difference in velocities between left and right wheel, and finally i is the activity of the most active sensor (the sensor where an obstacle is closest). All parameters are normalised to be between 0 and 1.

The fitness function for the population of initial configurations is one minus the fitness of the controller. Hence, the system is a competitive co-evolutionary system.

For the evolutionary process we use a single point crossover with a probability of 0.6, a mutation with probability 0.02 which replaces a gene with a random number which for the weights is normalised to be between 8 and -8. We use an elitism of 8 and the rest of a generation is selected using tournament selection with a tournament size of 2. We use 256 individuals and run the process for 300 generations where each individual is evaluated. For the initial pose part, we use 128 individuals and an elitism of 4. The initial configuration is limited to be inside the arena and at least a robot diameter from the wall ($10cm$), which is just outside of sensor range which is $15cm$ from the centre of the robot. For the co-evolutionary experiment, we evolve the

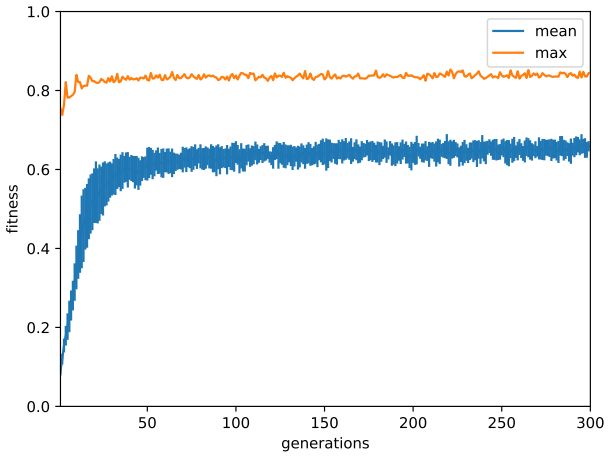

Figure 1: The fitness of the evolutionary runs based on random initial poses in the small environment. The figure shows the maximum fitness (top) and the average of ten evolutionary runs (bottom). The dark area corresponds to the standard deviation around the mean.

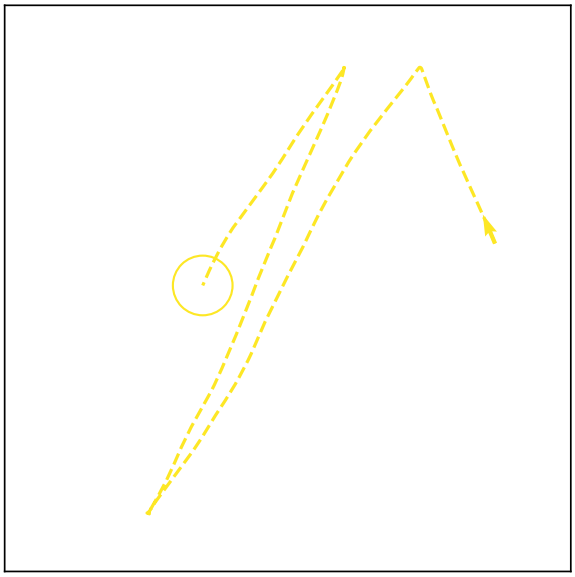

Figure 2: The robot arena (1m x 1m) and an example trajectory of the best individual of one of the evolutionary runs. The arrow represents the starting pose and the circle the final pose.

controller and the initial configuration alternately one generation at a time. For fitness evaluation the most fit individual from the other population is used. All code is implemented in CUDA (Nickolls et al., 2008) and all experiments are performed on a modest laptop with an Intel Core i7-10750H CPU running at 2.60GHz and a Quadro T1000 Graphical Processing Unit.

Results

Random Pose - Small Environment

The first experiment represents the base case and reproduces a result on evolving obstacle avoidance from literature. All individuals are evaluated in an arena with side length $1m$ and the evaluation period is $200s$. In Figure 1 the maximum and the average fitness of ten evolutionary runs can be seen. It shows that the algorithm consistently improves in the beginning and find a solution to the obstacle avoidance task. An example of a found solution can be seen in Figure 2 which shows that once a robot encounters the wall it turns sharply to get away from the wall. The experiment shows that in a small environment compared to the length of the evaluation period a standard evolutionary approach can find a solution and that our implementation works.

Random Pose - Timing

In the second experiment we measured the wall clock runtime of the evolutionary process as a function of the period each individual was evaluated. The result can be seen in Table 1. The Table shows that if each individual is evaluated for a simulated period of 400 seconds the evolutionary process takes 4.8 seconds. At the other end of the spectrum if each individual is evaluated for only 12 seconds of simulated time the evolutionary process takes 0.4 seconds. In other words, the GPU implementation allows us to run a full evolutionary run in 2.6 seconds for the run documented in the previous section where the evaluation period was 200 seconds. The table also documents that if we reduce the evaluation period needed to evaluate an individual, we have potential for further speed up.

simulation time	12	25	50	100	200	400
run-time	0.4	0.5	0.8	1.4	2.6	4.8

Table 1: This table shows the run-time of the evolutionary process as a function of the evaluation time of individuals. Both are measured in seconds.

Random pose - Large Environment

We now rerun the first experiment with the only exception that we increase the size of the environment such that it has a side length of $8m$. However, while evolution strictly takes place in the larger environment, we additionally validate each individual in the small environment to make the absolute fitness values comparable to the first experiments. The validation fitness results are shown in Figure 3. The fitness graph shows that a solution cannot be found to the obstacle avoidance task in the large environment and inspection of the trajectory shows that the robot simply crashes

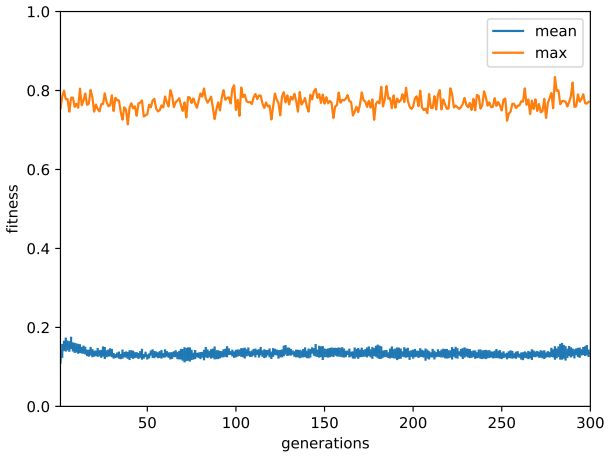

Figure 3: The fitness of the evolutionary runs based on random initial poses in the large environment The maximum fitness and the average with errors bars of ten standard evolutionary runs.

into the wall the first time it encounters it. The reason evolution fails is that a randomly placed individual in the large environment is unlikely to encounter an obstacle (the wall) and hence can score high fitness without having learned the correct behaviour. Furthermore, individuals which are randomly placed close to a wall will have a low fitness and be out-competed by individuals which were not placed close to a wall. Hence, the optimal solution from the evolutionary algorithms point of view is an individual that goes fast and straight.

Random Pose - Evaluation Period and Environment Size

In order to explore the potential for further speed-up, we now examine the sensitivity of the random pose-based evolutionary approach to arena size and evaluation period. For each set of parameters, we run the evolutionary approach ten times and count the number of times that the resulting controller can successfully avoid the wall in the validation environment (1m x 1m) for 200 seconds. If the robot within this time touches the wall the run is a failure. If not, it is a success. The results can be seen in Table 2. As the size of the arena increases (that is, the side length of the arena increases) the number of successful evolutionary runs decreases. Similarly, as the evaluation period of individuals decreases the number of successful evolutionary runs also decreases. In fact, we find that for large arena sizes and short evaluation periods evolution fails to find a solution. The only exception is a single successful run with an arena side length of 8m and an evaluation period of 50m. However, the other nine runs with these parameters fail hence we consider this

	1m	2m	4m	8m
12s	0	0	0	0
25s	0	0	0	0
50s	40	0	0	10
100s	100	90	0	0
200s	90	100	50	0
400s	70	100	100	50

Table 2: The percent of successful runs using the baseline evolutionary robotics setup as a function of arena side length and evaluation time of each individual.

	1m	2m	4m	8m
12s	30	20	30	50
25s	90	100	90	90
50s	70	100	90	80
100s	100	60	90	90
200s	100	90	80	100
400s	100	90	100	70

Table 3: The percent of successful runs using co-evolution of pose and controller as a function of arena side length and evaluation time of each individual.

an outlier. In general, the variation in the data stems from the random initialisation of weights and the random poses of individuals. Overall, the experiment shows that the baseline evolutionary approach does not converge on a solution for large arena sizes and short evaluation periods and thus we cannot obtain speed-up this way.

Co-evolved Pose - Evaluation Period and Environment Size

In the last set of experiments, we rerun the experiment from the previous section with the exception that we use the co-evolutionary approach where the initial pose is co-evolved with the controllers. In Table 3 we see the results of running the co-evolutionary setup using the same parameters as for the baseline setup. We can see that the number of successful runs is independent of arena size. This is because the co-evolutionary process finds initial poses close to the wall independent of the arena size. Hence, the performance of the co-evolutionary process is independent of arena size. If we look at the evaluation period, the process is also robust to changes of this again because the robot starts close to the wall and comparative fitness can quickly be established. First when the evaluation period becomes 12 seconds the process fails. This is because with the short evaluation time the robots do not have time to move away from the wall and obtain a higher fitness compared to an individual that does not move at all. In fact, from observing the trajectories we find that most of the individuals that succeeds in these runs tend to stay in one spot and hence does solve the task, but are low fitness scoring individuals.

Generally, we find that for the baseline evolutionary setup solutions can reliably be found if the arena size is less than 2m and the evaluation period is higher than 100s. While for the co-evolutionary setup we can find solutions for any arena size if the evaluation period is 25s or longer. Hence, in conclusion we find that the co-evolutionary process can find a solution to the obstacle task independent of arena size. Similarly, we find that we can reduce the evaluation periodic drastically without influencing the performance. This leads to much lower computational costs and thus faster run times of the evolutionary optimisation process.

Discussion

While the two efforts in this paper are directed at making evolutionary robotics more computationally tractable, it is worth to note that co-evolution of course is more computationally costly than a standard evolutionary setup because we both must do fitness evaluation of the controller and the pose. However, our results show that even though the algorithm is more time consuming the result is still that it overall is faster.

A short-coming of the work is that we have not considered the reality gap (Jakobi et al., 1995). However, we do feel on safe ground as related work has shown that for simple systems such as the one demonstrated here it is possible to overcome the reality gap.

An important question is if our approach generalises to more complex tasks. For the GPU acceleration we think this is the case because many simulators already take advantage of GPU acceleration. Hence, all that is needed is to allow several simulators to run in parallel on the same GPU. While this in theory is simple, the practical challenge may be significant. However, future simulator developers may well bear in mind that many simulations should be able to run in parallel.

The other generalisation question relates to whether the idea of co-evolution of initial pose and controller scales to more complex tasks or not. This is less clear. The challenge is that in many cases it is easy to come up with poses or in general configurations that makes solving a task for a controller impossible. E.g., for a walking task a specifically unfortunate configuration would be one where the robot has lost its balance and therefore is unrecoverable no matter what actions the controller decides to take. For these tasks, the configurations the co-evolutionary system should have access to, should likely be reduced which requires some hand-tuning.

Conclusion

In this paper we introduced the idea of fully parallelising an evolutionary robotics engine making it possible to run it on a GPU. This immediately led to drastic increase in performance. Specifically, we found for our relatively simple evolutionary setup that we could run a full evolutionary run in less than 2.6 seconds on a modest laptop. We used this engine to explore if co-evolution of robot pose and controller could reduce evaluation period and thereby further increase performance. We found this to be the case and was able to reduce the evaluation period to 25s for all arena sizes. Overall, we find the approaches to be promising and we think it is a timely topic for further research to understand how to reduce computational costs of evolutionary robotics setups.

References

Bongard, J. C. (2011). Morphological and environmental scaffolding synergize when evolving robot controllers: Artificial life/robotics/evolvable hardware. In *Proc. of the 13th Annual Conf. on Genetic and Evolutionary Computation*, page 179–186, New York, NY, USA. ACM.

Doncieux, S., Bredeche, N., Mouret, J.-B., and Eiben, A. E. G. (2015). Evolutionary robotics: What, why, and where to. *Frontiers in Robotics and AI*, 2:4.

Gomez, F. and Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342.

Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. *Lecture Notes in Computer Science*, 929:704–720.

Jones, S., Studley, M., and Winfield, A. (2015). Mobile GPGPU acceleration of embodied robot simulation. In *Artificial Life and Intelligent Agents*, pages 97–109, Cham. Springer International Publishing.

Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M., and Fox, D. (2018). GPU-accelerated robotic simulation for distributed reinforcement learning. In *Proc. of The 2nd Conf. on Robot Learning, PMLR*, volume 87, pages 270–282.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., and State, G. (2021). Isaac gym: High performance gpu-based physics simulation for robot learning.

Mondada, F. and Floreano, D. (1996). Evolution and mobile autonomous robotics. *Towards Evolvable Hardware. Lecture Notes in Computer Science*, pages 221–249.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press.

Ohkura, K., Yasuda, T., Matsumura, Y., and Kadota, M. (2014). GPU implementation of food-foraging problem for evolutionary swarm robotics systems. In *Swarm Intelligence*, pages 238–245, Cham. Springer International Publishing.

Pierson, H. A. and Gashler, M. S. (2017). Deep learning in robotics: a review of recent research. *Advanced Robotics*, 31(16):821–835.

Rossi, C. and Eiben, A. (2014). Simultaneous versus incremental learning of multiple skills by modular robots. *Evol. Intel.*, 7:119–131.

Rudin, N., Hoeller, D., Reist, P., and Hutter, M. (2021). Learning to walk in minutes using massively parallel deep reinforcement learning. *CoRR*, abs/2109.11978.

Soares, J. M., Navarro, I., and Martinoli, A. (2016). The Khepera IV mobile robot: Performance evaluation, sensory data and software toolbox. In *Robot 2015: 2nd Iberian Robotics Conf.*, pages 767–781. Springer.

Stoy, K. (2021). Co-evolution of Initial Configuration and Control in Evolutionary Robotics. In *ALIFE 2021: The 2021 Conference on Artificial Life*. 70.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. (2019). Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions.