Evolutionary Robotics: Tackling the Reality Gap Through Gait Optimization in Simulated Agents

Trym Auren
Department of Informatics
University of Oslo
Oslo, Norway
tsauren@ifi.uio.no

Jonas Romundstad
Department of Informatics
University of Oslo
Oslo, Norway
jonasrom@ifi.uio.no

Øystein Øverbø
Department of Informatics
University of Oslo
Oslo, Norway
oaoverbo@ifi.uio.no

Kyrre Glette
Department of Informatics
University of Oslo
Oslo, Norway
kyrrehg@ifi.uio.no

Abstract—**In the domain of evolutionary robotics, the reality gap continues to impede the seamless transition of optimized behaviors from simulation to physical systems. Simulation is necessary to accelerate development and testing of physical robots. Overcoming the reality gap would enable one to be confident that a robot with high performance in simulation would have a high performance in reality. Having this confidence would speed up the development of physical robots. The research behind this paper aims to investigate two things:**

1. Whether evolving the locomotion of a robot in a Unity environment with obstacles will mitigate the reality gap.

2. Whether evolving the locomotion of a robot with different physics engines in Unity makes the robot more robust to changes in physics, such as the change experienced when transitioning from simulation to reality.

The initial goal of the experiment was to quantify the reality gap when a robot is evolved using the above-mentioned approaches. Our target robot for the experiments was the QuteeV2 12-DOF robot, while also using a simpler crawler robot (part of the ML-agents toolkit), to become familiar with Unity. The research was initially supposed to measure reality gap. However, this turned out to be unachievable due to time constraints. Therefore, we are rather comparing evolution algorithms, fitness functions and frequency settings, whos goal is to optimize the gaits of the crawler. The difference in solutions found by the algorithms are compared, revealing that CMA-ES lays one step ahead of traditional evolutionary algorithms.

Index Terms—**Evolutionary Robotics, Reality gap, DEAP, CMA-ES, Simulation, Unity, ML-Agents**

I. Introduction

There are several reasons why developing a robot in a virtual environment is preferred over developing in reality: Evolving robots in reality is time-consuming; expensive because of wear and tear; requires access to the robot and requires human interaction, such as starting and stopping. Therefore, simulation is the preferred method when attempting to improve the gaits of a robot. However, there is one problem with this approach: Often, when deploying a robot in reality that had high performance in simulation, it shows lower performance or no performance at all in reality [9]. This phenomenon has limited evolutionary robotics from achieving its intended goal of creating working robots using concepts from biological evolution. The reality gap is normally caused by differences in sensing and actuation in a simulated environment and reality [15].

A. Some words on simulation

Computer gaming has been a big driving force for AI: Artificial agents have learnt how to beat humans in games like Atari [11], Go and Chess [14] - to mention some. There has also been improvements in computational power, thanks to the usage of GPUs, traditionally used for computer gaming [5]. Evolutionary robotics is no exception: using simulation environments developed for game development yields a user-friendly and interactive experience [8]. However, issues like non-realistic interaction between physical objects and motor dynamics, can lead to robots developed in simulation exploiting the shortcomings of the physics engines [9]. In the field of evolutionary robotics, robots learns by interacting with the simulation environment, and will therefore pick up certain peculiarities from the simulation environment that are usually not present in reality [13].

This project was initially intended to explore whether utilizing several physics engines in the Unity simulator will yield a more robust robot. The choice of Unity as the simulation environment, was based on Unity's own machine learning API (ML-agents), and its Python API [8]. A crawler-robot is used in the experiments, because of its simple construction, and because it is already defined in the ML-agents toolkit. The robot, which has a symmetric shape, can be controlled using a simple periodic wave. The simple controller was chosen to quickly be able to quantify reality gap across physics engines and/ or environments containing obstacles, without the need for extensive controller development.

II. Related work

Anticipating major advantages in overcoming the reality gap, researchers have engaged in a wide and creative array of studies to connect simulations more closely with real-world conditions. Of course, improving physics engines is the most intuitive solution. However, a fully accurate physics engine is yet to be implemented. There is also a certain trade-off between accuracy and run-time. If the simulator must be as slow as reality to increase precision, the advantages with simulation diminishes. However, multiple strategies exist to tackle the reality gap issues:

A. Robustness improvement

A common solution to promote more transferable agents, is to add different forms of noise when training the agent in simulation. This approach can be found as early as in 1995, together with the first mention of 'reality gap' in Jakobi *et al.* [7]. They found that injecting Gaussian distributed noise to the simulation environment lead to more robust controllers. Controllers developed in simulation environments with noise containing the same magnitude as reality, experienced increased transferability [7]. In the findings by Peng *et al.* [12], it was shown that other forms of noise, such as sudden latency in movement or sudden adjustment of physical weight of body parts, can lead to more robust physical interaction between agents and objects. Other strategies involves using obstacles to promote more robust gaits: Glette *et al.* [4] shows that using small objects to make the ground more uneven provides robustness of locomotion in a *QuadraTot* robot. Collins *et al.* found that switching between different physics engines during training of agents may improve transferability, due to some physics engines being better at certain tasks [2]. The above mentioned are ways to improve agents developed solely in simulated environments.

B. Sim + reality

In addition to solely working in simulation, findings by Kadian *et al.* [9], Koos *et al.* [10] and Collins *et al.* [2] evolves around using both simulation and reality in some way. One approach is to have the training algorithm optimize both fitness in simulation *and* reality [10]. Then, the robot must of course be trained both in simulation and reality. This enables reduction of time spent in reality by using a simulator, while also enabling feedback from reality conditions. Another possibility is to optimize both simulation fitness and the probability that the evolved behaviour will succeed in reality [9]. There are several advantageous returns by doing this: The ability of physics engines to replicate the physics of reality can be evaluated and tuned to better match reality conditions. Algorithms may optimize simulation fitness and reality fitness, effectively reducing reality gap. However, these approaches still requires a physical setup.

C. Our approach and choice of simulator

To stay within scope of this research project, attempts will not be made to use any physical appliances. Rather, quantization by a physics engine and visual examination will be utilized to measure reality gap.

Robotics research commonly utilizes domain-specific simulation platforms, including MuJoCo, V-Rep (CoppeliaSim), Habitat, and PyBullet, to mention some. Each platform excels in particular tasks such as locomotion or manipulation [2]. Nonetheless, their specialization limits the variety of visuals, objects, and user-friendly programming interfaces they offer. On the contrary, the Machine Learning Agents platform (ML agents) [8] by Unity simplifies machine learning experimentation, allowing for relatively easy integration with Python. Unity also offers easy altering of physical objects, and building complex environments. Another advantage of using Unity for simulations is the flexibility of switching between different physics engines, such as the default PhysX, as well as alternatives like Havok and Unity Physics.

However, it turns out that the Havok physics engine requires a paid subscription. This makes Havok irrelevant to this low-budget project. Unity Physics does not work as expected. Due to incompatibilities with other libraries, it is also deemed irrelevant. PhysX is therefore the only physics engine used in the experiments.

III. Method

Locomotion of a crawler robot is controlled by a simple sine function, as in Koos *et al.* [10]. Amplitude, phase and frequency are given as arguments to the sine function, to produce a series of periodic movements. The arguments to the sine function - the genotype - is held in the crawlers genome. The produced sine wave is the phenotype, while the genome is evolved through three evolutionary algorithms: One traditional algorithm implementing the usual steps parent selection, mating and mutation; another based on Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), as described in [6]; the last one, CMA-ES-bipop extends CMA-ES by introducing additional, small *bi-populations*. The experiments conducted measures fitness (distance from starting point to stopping point) for different evolutionary approaches, with different parameter configurations. In addition to quantified fitness, qualified fitness is measured through a visual examination of the evolved agents' locomotion. See the figure below (fig. 1) for a visualization of the experiment setup.

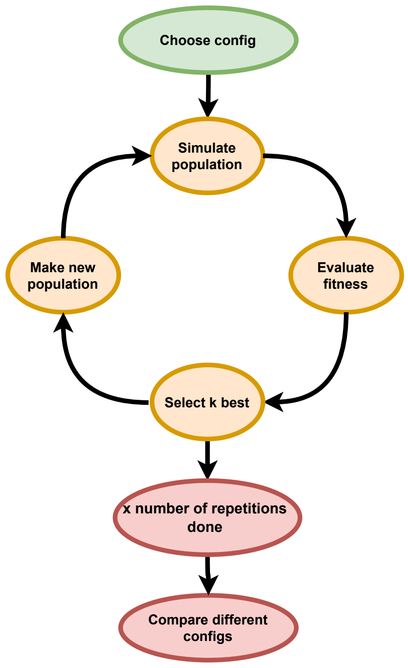

Fig. 1: This setup is repeated in total 120 times, 10 times per configuration for 12 configurations. See table II for the different configurations.

A. Robot configuration

The crawler robot used in these experiments originates from the example assets of the Unity ML-agents [8] package. The robot is equipped with a four-legged configuration, each consisting of two segments: an upper leg and a lower leg. The upper legs have two degrees of freedom, enabling a range of motion horizontally and vertically relative to the crawlers face. The lower legs, on the other hand, have one degree of freedom, limited to movement in the x direction within the leg's coordinate system. This facilitates simple forward and backward swinging motions. In total, 12 different controllers are needed to control all movement directions. More on that below. See fig. 2 for a figure of the crawler in a simple Unity environment.

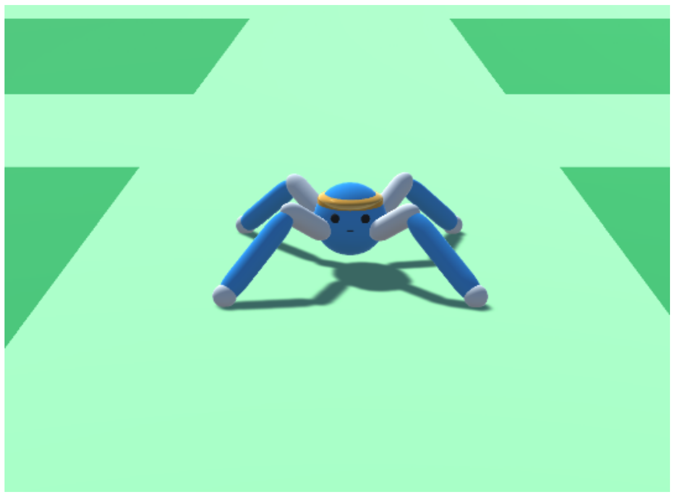
Fig. 2: Crawler robot in a simulation environment with a floor plane. The crawler can be found in the example assets in the Unity ML Agents library.

B. Locomotion

For all movement directions, a sinusoidal wave is generated. The set of the three parameters forming the sinusoidal function are all parts of the genotype of one crawler, and will be adjusted through evolution. I.e., the crawler has 12 sets of parameters, each set producing a series of movement. This results in the genome consisting of $3 \cdot 12 = 36$ floating point numbers, if all parameters for all movement directions are evolved. Some of the experiments involves synchronous frequency across limbs. This setup reduces the genome to 25 floating point numbers. The sinusoidal generation function is:

$$\theta(t,i) = A(i) \cdot \sin(f(i) \cdot t + \phi(i)) \tag{1}$$

where the target angle θ of joint i at time t is $\theta(t,i)$. Amplitude of joint i is denoted $A(i)$, the frequency of joint i is denoted $f(i)$, and the phase shift of joint i is denoted $\phi(i)$.

The sine function outputs a series of discrete numbers, given the three parameters. This series of discrete number are the target angles of the joint. All the values in the series holds values in the range $[-A, A]$. These values are then converted to a target angle in Unity. All array values above the movement restrictions are capped by Unity. This means that if a target angle of $3\frac{\pi}{2}$ is produced for a limb constrained $[-\pi, \pi]$, the target angle is set to π. fig. 3 displays the angle limits of each controller for a single leg, where a target angle of 0 is in the middle of the range.

To better understand the movement generation, one can consider an example to illustrate how the list of target angles is created. Assume the amplitude, frequency, and phase shift are all set to 1. A sine wave is generate, by sampling 2000 times over 4 periods. Each of these samples represents a target angle. The selection of how many samples significantly influences the duration required for each generation's simulation. So in this example, all legs will start in a position, then fully contract before fully extending, ending up in the starting position. If 1000 elements of the sine wave is chosen instead, the generation simulation time would double, because one individual would be simulated for twice the time steps, totalling two periods of the sine wave. Similarly, selecting 750 elements would result in a 1.5 increase in simulation time compared to using 500 elements.

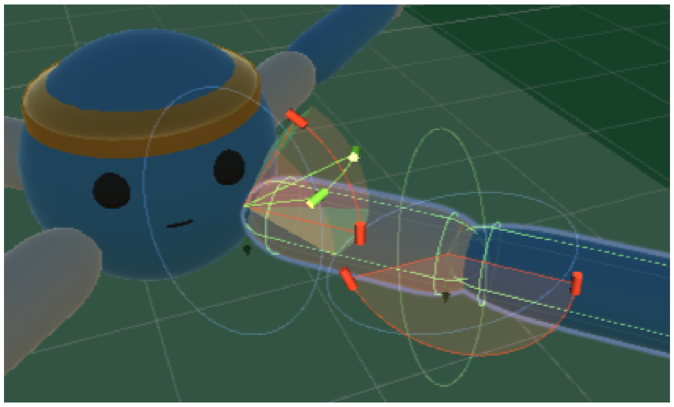
Fig. 3: Visualization of the angle constraints of one leg of the Unity ML-Agents Crawler robot.

C. Evolution

For the evolutionary algorithms, the library Distributed Evolutionary Algorithm in Python (DEAP) is chosen because it offers easy altering of evolution parameters and few lines of code to get started [3]. DEAP has several example algorithms implemented. The three algorithms mentioned above are implemented to closely resemble the examples.

The three algorithms differ in the following ways: CMA-ES employs a population generation scheme where a population is drawn from a distribution which changes over generations. The algorithm moves the distribution in the direction of the best solutions. If there exists good solutions in two outer locations - i.e., two different genome configurations has high fitness - the population drawn will have big variety. If all the good solutions are centered at one location, the population drawn will have low variety. See figure fig. 4 for a visual explanation of CMA-ES.

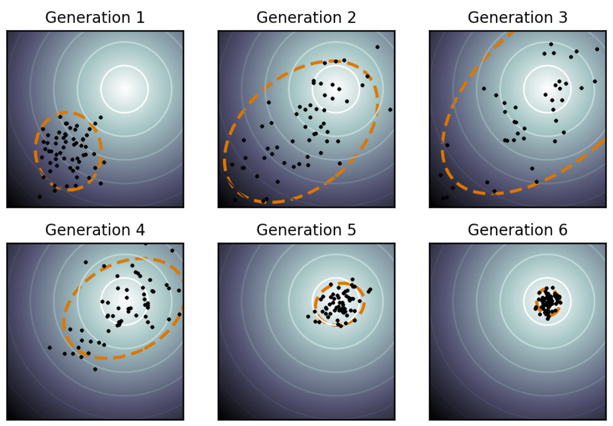

Fig. 4: Visualization of CMA-es. The distribution moves towards the best solutions. Source: [1]

CMA-ES-bipop works in the same way, but at the same time keeps two populations (regimes): one regime where the population size is increasing with generations, and another where the population size is small. The algorithm switches back and forth between evaluating the two regimes. This allows for more exploration than the classic CMA-ES approach. On the other hand, the simple EA implementation consists of the classic approach commonly referred to as μ, λ, and employs parent selection, crossover and mutation - with full population replacement at each generation.

All three algorithms were seeded. All algorithms are ran with the same starting population and the same population size. The simple EA uses a tournament selection with size 3, where the number of tournaments were equal to a static population size of 900. As mating method, two point crossover is utilized. Both the simple EA and the CMA-ES ran for the same number of generations, while the CMA-ES-bipop ran for more generations, as it keeps track of two populations and therefore needs more iterations to reach results. See table I.

Parameter	**Simple EA**	**CMA-ES**	**CMA-ES-bipop**
Population size	900	900	900
Number of generations	100	100	100
Initial genome range	[0,4]	[0,4]	[0,4]
Standard deviation of initial population	1	1	1
Crossover rate	1		
Mutation rate	0.2		

TABLE I: Hyper-parameters for the different algorithms

D. *Fitness evaluation*

The initial concept for quantifying the reality gap centered around measuring the net displacement of the simulated crawler robot in Unity compared to its real-world counterpart, measured over a set number of time steps. This approach focuses on the straight-line distance from the starting point to the endpoint, rather than the total path traveled, to ensure that fitness is not influenced by non-linear or swirling movements. A bigger difference in net displacement equals a bigger reality gap, which gives an indication of how well the physics engine mimics reality. Since the quantification of the reality gap does not become reality, we are only using net displacement in simulation as the fitness metric. Two approaches were taken to measure net displacement: Either net displaced euclidean distance, or net displace distance only in the direction of the axis pointing straight forward from the starting position of the crawler - the z-axis. All experiments are run with both fitness functions. The reasoning behind exclusively emphasizing the z-axis in certain experiments is to encourage the robot to move linearly, thereby preventing the robot from going backwards or sideways.

IV. Experiments and Results

The study comprises twelve distinct experimental setups (configurations), which can be seen in table II. All configurations were ran 10 times with different seeding each time. Both the average (solid line) and the standard deviation (background color) across runs is plotted. See fig. 5 for results from simple EA configurations, fig. 6 for results from CMA-ES configurations, and fig. 7 for CMA-ES-bipop results. Video of the best individuals from each of the 10 runs, for the different configurations can be found at the YouTube channel for the project. The authors suggests watching the video while reading this section.

Config	**Algorithm**	**Fitness function**	**Frequency synch.**	
			True	**False**
1	Simple EA	one direction	✓	
2				✓
3	CMA-ES	one direction	✓	
4				✓
5	CMA-ES-bipop	one direction	✓	
6				✓
7	Simple EA	two directions	✓	
8				✓
9	CMA-ES	two directions	✓	
10				✓
11	CMA-ES-bipop	two directions	✓	
12				✓

TABLE II: Experimental setups with different algorithms and evaluation criteria.

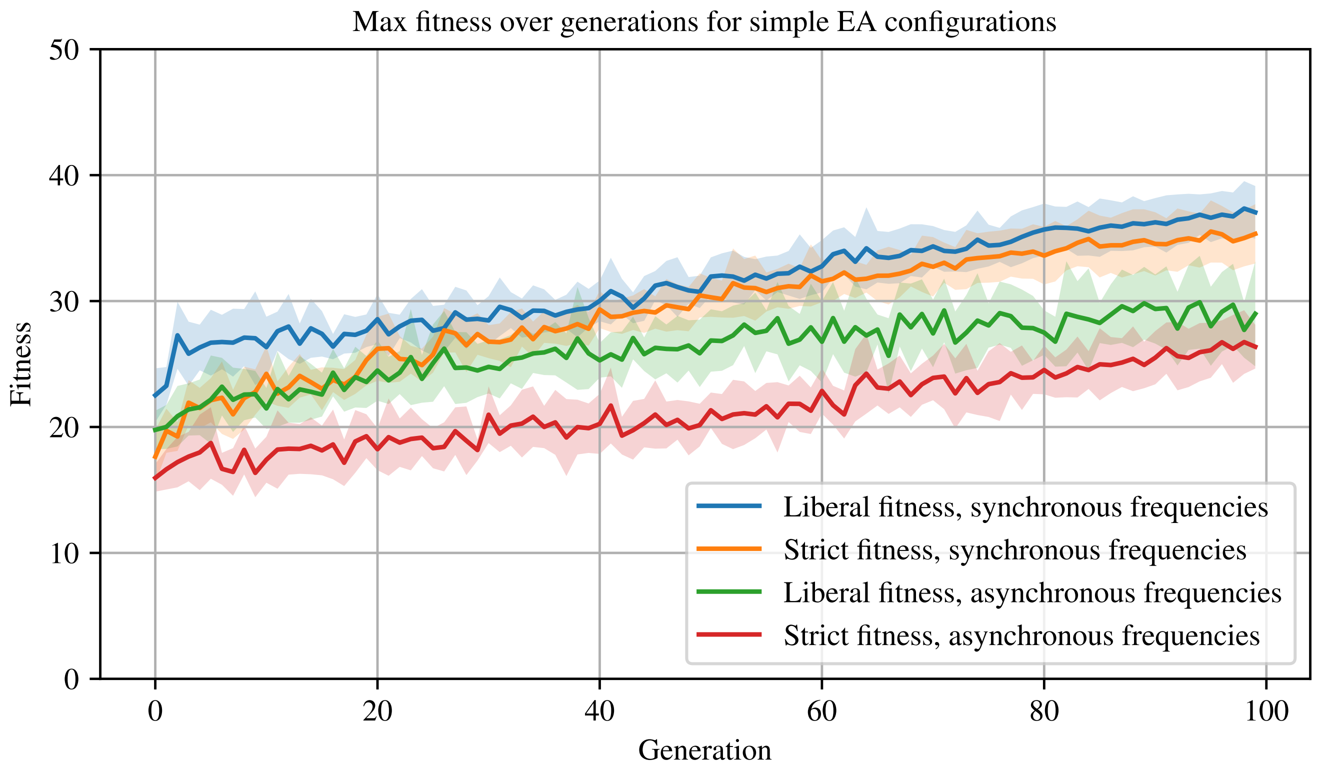

Fig. 5: Max fitness for the simple EA over 100 generations with population of 900. The solid line is the mean of 10 runs, while the coloured area is the deviation of the 10 runs.

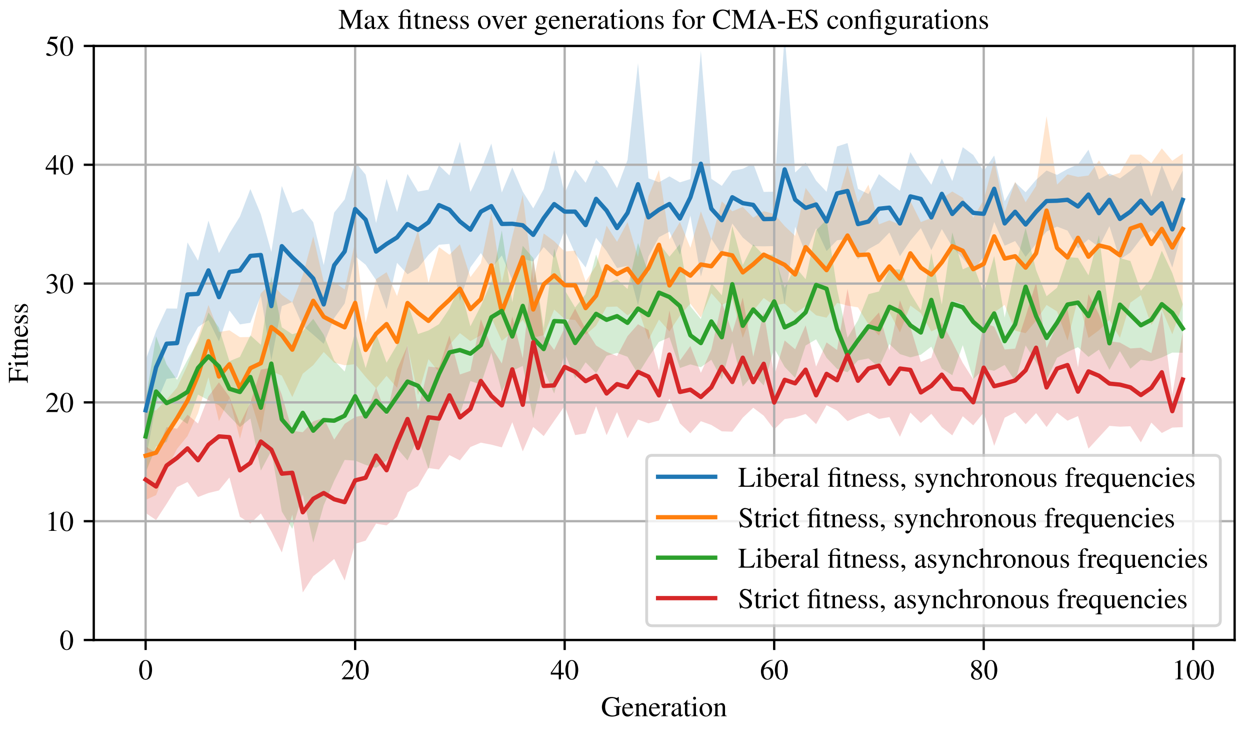

Fig. 6: Max fitness for CMA-ES over 100 generations with population of 900. The solid line is the mean of 10 runs, while the coloured area is the deviation of the 10 runs.

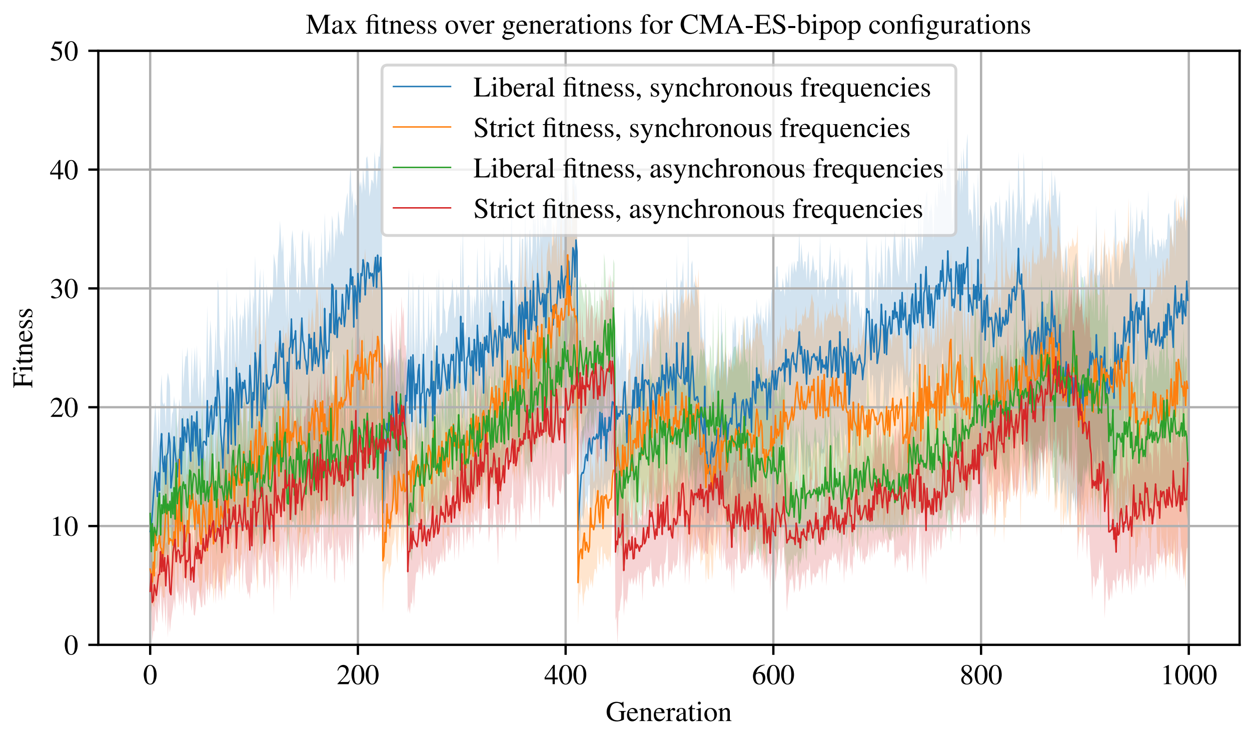

Fig. 7: Max fitness for CMA-ES-bipop over 100 generations with population of 900. The solid line is the mean of 10 runs, while the coloured area is the deviation of the 10 runs.

A. *Quantitative fitness*

From examining the plots, one can observe that both the simple EA implementation and the CMA-ES implementation achieve the same highest fitness averaged across runs. The CMA-ES-bipop implementation performs a bit under the two. The configuration achieving the highest fitness for all algorithms was config 7, 9 and 11 (see table II), where fitness function was fitness in all directions and the controllers were evolved with equal frequencies for all movement directions. Performing second best is the configurations where fitness is measured only in one horizontal direction, while the frequencies are still evolved to be synchronous across limbs and movement directions (config 1,3,5 in table II). Since the two highest scoring configurations has synchronous frequency and the lowest performing configurations (config 2,4,6,8,10,12 in table II) has asynchronous frequency, it is possible to extract that equal frequency for all limbs results in highest fitness. CMA-es-bipop performs a bit weaker than CMA-ES and the simple EA, but follows the same pattern regarding frequency. Examining the impact of different fitness functions, it is observable that crawlers evolved with a fitness function focused on movement in just one direction exhibited almost comparable fitness scores to those evolved with a multi-directional fitness function. Typically, these single-direction-focused crawlers demonstrated a proficiency in straight walking behavior. However, despite achieving similar fitness levels, their gaits can still be unrealistic. This indicates that while the direction-focused fitness function effectively encourages straight-line locomotion, it does not necessarily guarantee the naturalness or realism of the gait.

B. *Qualitative fitness*

When observing the video recordings of the locomotion of the crawler robots in the simulation tool, it can be seen that the difference in visual appearance of gaits is a lot bigger than the difference in fitness obtained. The qualitative fitness is therefore examined below.

CMA-ES qualitative fitness: The best performing configuration (config 9 in table II) produces gaits deemed realistic for 8/10 of the runs. For the second best performing configuration (config 3 in table II) and third best (config 10 in table II) configurations, approximately 5/10 of the runs produce unrealistic gaits that exploits the environment. For the worst configuration (config 4 in table II), 10/10 solutions produces gaits that exploits the environment.

Simple EA qualitative fitness: 10/10 runs for all configurations produced unwanted gaits - that is gaits that are not looking transferable to reality. All solutions have learnt to make flips by using one or two legs to accelerate off the ground. These unwanted gaits are present to a much higher degree in the simple EA runs, than in the CMA-ES runs.

CMA-ES-bipop qualitative fitness: The CMA-ES-bipop algorithm slides in between the two above mentioned. Disregarding fitness function, this algorithm produced wanted gaits for 1/10 solutions when the frequencies were synchronous. Asynchronous frequencies did not produce any gaits looking transferable.

V. Discussion

A. Algorithm Performance and Behaviors

- The contrasting behaviors between CMA-ES and the two other algorithms are noteworthy. Even though the fitness function configurations are equal for the two algorithms, CMA-ES manages to find more realistic gaits than the others. This may be due to the fact that CMA-ES often reaches a higher fitness score for a few runs, see the blue background color spikes in fig. 6. This slight increase in fitness may be essential for filtering out non-realistic gaits. The same explanation may also explain why CMA-ES-bipop is also able to produce realistic gaits for some runs.
- Taking cost into account when evaluating performance, CMA-ES-bipop performs a lot worse than the two others. It requires a significant amount of generations to reach worse fitness scores - on average. However, if looking to the best configuration, it beats the simple EA, and should therefore be considered to use in this type of optimisation problems.
- There is a big contrast between CMA algorithms and the simple EA when it comes to consistency in runs. The deviation is much bigger for CMA-ES and CMA-ES-bipop runs, indicating that these algorithms may be more sensitive to randomness than the simple EA. It can also indicate that they are more willing to explore wider than the simple EA, therefore achieving better solutions for the best runs, and worse solutions for the worst runs.
- Another interesting observation when studying the video recordings of the evolved crawler, is that the crawlers evolved using both the simple EA and CMA-ES-bipop differs from the ones evolved using CMA-ES in frequency and gaits. The frequency of the most fit crawlers are generally higher than the least fit crawlers, if looking at the quantified fitness. However, when looking at the qualified fitness, the crawlers using a high frequency for locomotion seems to exploit the friction of the simulation environment.

B. Reality gap

The initial goal of finding ways to reduce the reality gap could not be achieved due to time constraints, leaving a critical area of evolutionary robotics research unaddressed. While simulation offer a safe and cost-effective environment for initial testing, the true test of any robot controller lies in its performance in the real world. The general misalignment between simulation and reality will most likely imply that the results achieved in this research are not transferable to the real world. However, as some of the configurations produces what looks like natural gaits, some of the controller strategies found may work in a really simple reality condition. The choice to control locomotion with sinusoidal waves and the decision to lock or vary the frequency across limbs have significant implications. Frequency locking could potentially simplify the control problem, but might lead to less adaptable and robust gaits. In contrast, allowing for different frequencies might enable more complex and potentially more effective locomotion patterns. I.e., evaluating the solutions found in a more complex environment may yield solutions without frequency locking.

C. Limitations

There are many limitations present, mainly due to time constraints. First, the experiments would benefit from a more thorough hyper parameter search. The hyper parameters chosen are not backed up by any literature, but more or less chosen randomly. Second, since the experiments are already pretty biased by different parameters chosen, putting constraints on more of the Unity related configurations, such as torque and angles, could speed up evolution by narrowing the search space. Third, the runs are paralleled, but the evaluation of each population is sequential. Running evaluation of population batches in parallel, could further speed up execution, allowing for speed up of simulation time.

VI. Conclusion and future work

A. Conclusion

The primary objective of this project was to explore strategies for minimizing the reality gap in robotic simulations. Initially aimed at a different topic, the project's focus shifted due to time constraints, emphasizing the optimization of realistic gaits using various evolutionary algorithms. This shift entailed experimenting with different fitness functions and frequency settings, to evaluate their impact on the gaits of simulated robotic agents. Our findings reveal a significant variance in the gaits produced, ranging from highly unrealistic to more natural. The CMA-ES algorithm emerged as the most effective, consistently achieving the best fitness scores and the most natural gaits. This was despite a tendency for the solutions to exhibit atypical frequency patterns. In contrast, the simpler EA algorithm yielded unnatural gaits characterized by agents performing flips rather than walking, showing its limitations in this optimisation problem. CMA-ES-bipop presented a mixed performance. While it ranked between the simple EA and CMA-ES in terms of fitness, it fell short in producing realistic gaits, as the simple EA. In conclusion, this project highlights the relationship between algorithmic choices and the realism of simulated gaits. While CMA-ES shows promise, there remains a significant area for further research, particularly in refining the simulation environments to better mimic real-world physics and robot dynamics.

B. Future work

The ML-agents toolkit from Unity and the DEAP library provides easy evolution of robots, providing a framework for reality gap research. It would be interesting to quantify the results obtained by using different algorithms to actually measure how big the reality gap is for different algorithm configurations. There is also need for extensive hyper-parameter tuning, to be able to further improve the results. Constraining the maximum torque for each limb in Unity could also benefit the project, as some of the solutions seems to use a lot of torque when doing flips or other unnatural gaits. Constraining the maximum target angle further could also help finding more realistic gaits.

VII. Ethics and broader impacts statement

- The experiments conducted in this study pose minimal risk to their immediate environment. Simulating robots within the Unity framework is inherently safe and poses no physical dangers. However, there can be some minor risks if working with the physical robot, as the humans involved are prone to shocks or mechanical injuries.
- All the data utilized in this study is generated by the research team. There exists a potential for bias in the results produced, given that the researchers are actively involved in both the examination of results and the fine-tuning of various parameters. These include the robot's design, choice of platform, and the selection of related works that inform the research.
- The advancement of intelligent systems always carries the risk of these systems being misused to the detriment of individuals or other organisms. This research primarily aims to enhance the efficiency of robot motion controller development and reality deployment. While the immediate scope of the research is not harmful, it is reasonable to assume that the techniques developed here could be adapted to more effectively develop harmful, intelligent systems in the future.
- Employing Unity for robotics simulations has the potential to fortify Unity's market position, possibly at the expense of smaller, open-source initiatives. Unity is already a dominant player in the game development industry and has faced criticism for prioritizing commercial interests over open-source contributions. This can especially be an issue if Unity enacts a paywall after significant frameworks have already been built. The decision to use Unity was made based on the existing expertise within the research group.
- As the project demands processing power to speed up simulation robots, high-performance computing (HPC) clusters were utilized. HPC clusters are energy-intensive. According to ML CO2 Impact, our usage resulted in the emission of approximately 5 kg of CO2. Specifically, we utilized machine learning nodes at the University of Oslo, equipped with 4x Nvidia A100 GPUs. Power consumption in calculation projects, such as ours, goes beyond functional considerations in society. As the tech industry's hunger for power grows with advancements in AI, big data, and other resource-intensive fields, it becomes crucial for us to ethically and responsibly manage our resources, not just for immediate economic benefits but with the long term effects in mind to ensure a sustainable future.

VIII. References

[1] *CMA-ES*. In: *Wikipedia*. Page Version ID: 1183723226. Nov. 6, 2023. URL: https://en.wikipedia.org/w/index.php?title=CMA-ES&oldid=1183723226 (visited on 11/16/2023).

[2] Jack Collins, David Howard, and Jürgen Leitner. *Quantifying the Reality Gap in Robotic Manipulation Tasks*. Nov. 7, 2018. DOI: 10.48550/arXiv.1811.01484. arXiv: 1811.01484[cs]. URL: http://arxiv.org/abs/1811.01484 (visited on 09/03/2023).

[3] Felix-Antoine Fortin. "DEAP: Evolutionary Algorithms Made Easy". In: ().

[4] Kyrre Glette, Andreas Leret Johnsen, and Eivind Samuelsen. "Filling the reality gap: Using obstacles to promote robust gaits in evolutionary robotics". In: *2014 IEEE International Conference on Evolvable Systems*. 2014 IEEE International Conference on Evolvable Systems. Dec. 2014, pp. 181–186. DOI: 10.1109/ICES.2014.7008738.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Google-Books-ID: omivDQAAQBAJ. MIT Press, Nov. 10, 2016. 801 pp. ISBN: 978-0-262-33737-3.

[6] Nikolaus Hansen and Andreas Ostermeier. "Completely Derandomized Self-Adaptation in Evolution Strategies". In: *Evolutionary Computation* 9.2 (June 2001). Conference Name: Evolutionary Computation, pp. 159–195. ISSN: 1063-6560. DOI: 10.1162/106365601750190398. URL: https://ieeexplore.ieee.org/document/6790628/authors#authors (visited on 10/05/2023).

[7] Nick Jakobi, Phil Husbands, and Inman Harvey. "Noise and the reality gap: The use of simulation in evolutionary robotics". In: *Advances in Artificial Life*. Ed. by Federico Morán et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 704–720. ISBN: 978-3-540-49286-3. DOI: 10.1007/3-540-59496-5_337.

[8] Arthur Juliani et al. *Unity: A General Platform for Intelligent Agents*. May 6, 2020. DOI: 10.48550/arXiv.1809.02627. arXiv: 1809.02627[cs, stat]. URL: http://arxiv.org/abs/1809.02627 (visited on 10/04/2023).

[9] Abhishek Kadian et al. "Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance?" In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020). Conference Name: IEEE Robotics and Automation Letters, pp. 6670–6677. ISSN: 2377-3766. DOI: 10.1109/LRA.2020.3013848.

[10] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. "The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics". In: *IEEE Transactions on Evolutionary Computation* 17.1 (Feb. 2013). Conference Name: IEEE Transactions on Evolutionary

Computation, pp. 122–145. ISSN: 1941-0026. DOI: 10.1109/TEVC.2012.2185849.

[11] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. Dec. 19, 2013. DOI: 10.48550/arXiv.1312.5602. arXiv: 1312.5602[cs]. URL: http://arxiv.org/abs/1312.5602 (visited on 10/04/2023).

[12] Xue Bin Peng et al. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018 IEEE International Conference on Robotics and Automation (ICRA). ISSN: 2577-087X. May 2018, pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528.

[13] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Berlin: Springer, 2008. ISBN: 978-3-540-23957-4.

[14] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (Dec. 7, 2018). Publisher: American Association for the Advancement of Science, pp. 1140–1144. DOI: 10.1126/science.aar6404. URL: https://www.science.org/doi/full/10.1126/science.aar6404 (visited on 10/04/2023).

[15] Francisco J. Varela and Paul Bourgine. *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. Google-Books-ID: pWsNJkdZ4tgC. MIT Press, Apr. 2, 1992. 546 pp. ISBN: 978-0-262-72019-9.