Jacob Brown
University of Sussex - Informatics
Computer Science with Artificial Intelligence

“Lizardbot”

A reptile-inspired model of robots optimised to navigate rough terrain

Abstract

Lizardbot aims to provide a solution derived from natural processes for robots traversing irregular environments. Unity software was used to create a model of these robots with realistic physics interaction. The design combined a serpentine motion of the body, the use of a tail for stabilisation, and a lizardlike gait. An AI sought an optimal design by evolving the robots, mutating the physical structure and movement algorithms in tandem. Lizardbot was able to find an optimisation on the less extreme terrains: a single spherical body module with emergent rolling behaviour. Lizardbot demonstrated that a nature-inspired algorithm will not necessarily produce a naturalistic result.

Submission Year - 2022
Candidate Number - 198732
Project Supervisor - Simon Bowes
https://github.com/jacobgeorge26/lizardbot

Statement of Originality

This report is submitted as part requirement for the degree of Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged. I hereby give permission for a copy of this report to be loaned out to students in future years.

Jacob Brown

Acknowledgements

I would like to thank the supervisor for this project, Simon Bowes, for his advice and support throughout this year (and indeed my entire degree). I have always enjoyed his modules and it was a genuine pleasure to be able to implement some of the theory that he has taught me over the years into Lizardbot. Thankyou for your part in a very interesting degree!

Professional and Ethical Considerations

This project will be an isolated model without any user testing, personal data, vulnerable people, protected characteristics, or medical data. While it does focus on reptiles there will not be any experimentation on any animals. Referenced studies will have faced their own ethical review and will largely use videos of the animal being studied. All testing will be conducted by comparing various inputs to the code and analysing the results.
With these considerations, this project has been classed as low-risk.

This project will abide by the **BSC Code of Conduct**[1] regarding:

- Public Interest;
- Professional Competence and Integrity;
- Duty to Relevant Authority;
- Duty to the Profession.

[1] `https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf`

Contents

1 Introduction

Nature can often inspire elegant and efficient solutions to non-natural problems. For example, the tunnel-building behaviours of ants can be used to generate algorithms to manage traffic flow in a city. This can be much more efficient than deploying a group of developers to design an elaborate network to structure how the population navigates the city. Instead, the nests of ants could be studied and their tunnel-building algorithm applied to the problem [Plataforma SINC, 2008].

On a climbing trip to the Isle of Portland I took a break from repeatedly falling off the cliff to watch a lizard. It was attempting to jump from the path onto a nearby rock. Every time it failed it paused, then tried again with its tail in a completely different position. It was naturally using its tail to counterbalance its body as it jumped, and learning from previous attempts. The addition of a tail to a robot can produce an efficient jumping robot with the application of relatively simple maths. A prime example of this is shown in the *'UC Berkeley Leaping Lizard & Robot'* video [UC Berkeley, 2012]. This behaviour can be observed elsewhere in nature; when a praying mantis jumps it swings its body and abdomen to ensure it hits its target [Burrows et al., 2015], or when cats twist their bodies to reorient themselves to land on their feet [Diamond, 1988].

In this project, the characteristics of various reptiles will be applied to a model of a robot - "Lizardbot" - to optimise the physical design and algorithm it uses to navigate rough terrain. For robots used in applications such as bomb disposal or interplanetary exploration, the environments that they face can be highly unpredictable. The margin for error is often low to nonexistent, as there can be little physical access to the robot to fix any issues. These robots require a design and an algorithm optimised to enable them to traverse any landscape they are presented with. The stabilising tail of a lizard, alongside other reptilian characteristics, could provide a nature-inspired solution for a robot targeting rough terrain.

The robots will be modelled due to the lack of the resources required to build a physical robot. Even if it were possible to conduct physical testing, for applications such as extraterrestial robots the testing would still depend on simulations.
To construct a virtual model of the robot, the popular gaming engine Unity [Unity, 2021c] will be used. The Unity engine has built-in physics, collision, and terrain features that bring complex interactions within scope.

2 Project Design

This project will focus on determining how various reptilian characteristics influence the success of the model as it navigates a terrain. The objective is to explore the 'ideal' design of a robot, and the relationship between this body and its navigation of a randomly generated terrain. The basic performance metric will be how far it successfully moves across the terrain (in a straight line from the origin) before it gets stuck. This final state may be bouncing back and forth between two points or hitting a section of the terrain that it cannot progress past; the algorithm to determine if the robot is trapped will consider both of these possibilities.
Within this bigger picture goal there will be several smaller experiments into how various features and approaches influence the outcome.

Lizardbot will be constructed as a model of a robot due to the lack of the resources required to build a physical robot. Having prior experience with Unity [Unity, 2021c] I was confident that its physics system would provide realistic behaviour. Given this, the free access, vast range of functionality and excellent documentation, the decision was made to proceed using Unity.

2.1 Robot Design

The body of the robot be constructed of a series of independent modules / sections located linearly behind the head. This design *"provides the ability of traversing in irregular environments, something that surpasses the mobility of the conventional wheeled, tracked and legged types of robots"* [Kelasidi and Tzes, 2012, p. 536].
Legs will be added at random positions perpendicular to the body and rotate 360° to push the body forward. A tail, added behind the body, will rotate to counterbalance the motion of the body. Every joint will have a constraint defining how far it is capable of rotating around each axis.

There will be an option to generate robots with random parameters, from the number of body modules and legs, to the force with which they drive. Additionally, a boolean option will determine if the robot is constructed 'uniformly' - i.e. asserting whether it must be symmetrical with all legs / body modules of equal size and mass to each other. The ideal parameters for both the physical design and movement of a robot will be explored.

To simplify the model of the robot it will be constructed of basic shapes. The body will be a string of spheres, whilst the legs and tail will be spheroids. The decision against cubes/cuboids is to prevent the shapes from 'catching' on anything, as this could interfere with the results. Of course, with a physical robot there will be this interaction to contend with; for the purpose of the model it has been assumed that every interactable surface is uniform and can be smoothed for simplicity.

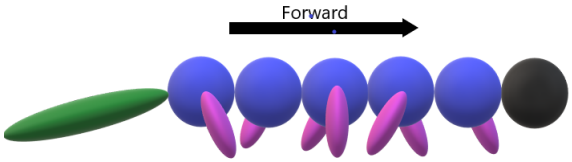

Figure 1: *An example of a randomly generated robot (viewed from the side).*

2.1.1 Body

The fundamental movement pattern of the body sections will mimic the serpentine motion of a snake, as shown in Figure 2.

Figure 2: *Diagram of a snake using a serpentine method of locomotion [Alexander, 2012].*

To achieve this, each module will have the ability to drive forward and/or rotate and will have its own set of parameters (speed, for example). To improve the fluidity of the motion a central pattern generator, an approach similar to a (CPG) approach will be taken. A CPG *"can produce rhythmic motor patterns ... in the absence of sensory or descending inputs that carry specific timing information"* [Marder and Bucher, 2001, p. 986]. To clarify, the body will not implement a true CPG: the algorithm will utilise a similar approach that avoids any sensory understanding of its worldspace or timing.

There will be an option to maintain serpentine motion by alternating the direction of rotation between the rotating sections, and evenly spacing these sections across the body (see Figure 3). The algorithm will not be constrained to this serpentine motion so may move away from this behaviour as it mutates. Nonetheless, due to the embodiment of the robot it is expected that the motion will remain snakelike.

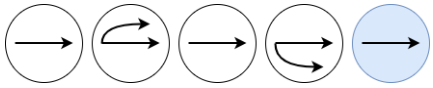

Figure 3: *The driving & rotation values used to maintain a serpentine motion.*

The decision to have the body oscillate (as opposed to remaining static whilst the legs provide the entirety of the motion) is supported by Jeongryul Kim [Kim et al., 2019]. Due to a lack of degrees of freedom (DOFs) in legged robots a robot may struggle to maintain its posture and direction of movement. *"A possible solution to such lack of DOFs caused by underactuation may be additional motions of the body"* [Kim et al., 2019, p. 130]. It is important to note that the referenced study was analysing bipedal robots when this conclusion was drawn. To test the assumption that moving the body with the legs improves performance, another experiment will be conducted to explore the impact of the body remaining static.
An alternative solution could be to add additional joints for the legs to increase the DOFs. For Lizardbot, a form that mimicked the structure of a lizard was prioritised whilst considering the positioning of leg joints. The offer of freedom in the leg placement remains as a project limitation. Regardless, other projects, such as the Salamandra Robotica II, have adopted a similar approach with promising results. The Salamandra used - and inspired - the combination of leg rotation and body oscillation [Crespi et al., 2013]. The Salamandra is covered in more depth in 3.3.

2.1.2 Tail

The motion of the tail will mirror that of the 'Agama robot' whereby the tail flicks up quickly in the opposite direction of the trajectory of the body [Libby et al., 2012]. The focus of this study was for stabilising a robot as it jumped but it is expected that this counterbalancing will improve the success of the robot. To test this, an experiment will be performed with and without a tail to determine how it impacts the distance the robot is able to cover.

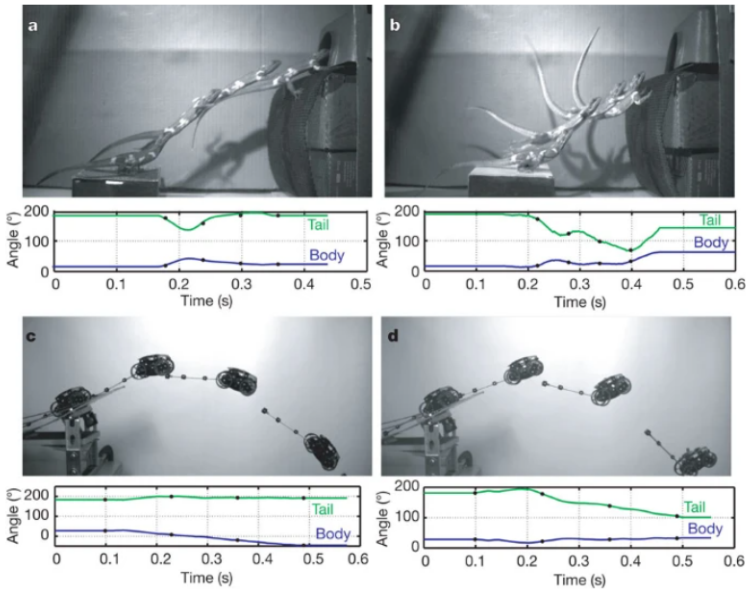

Figure 4: *An Agama lizard compared to the Agama robot whilst jumping [Libby et al., 2012].*

2.1.3 Legs

The approach to the fundamental movement of the legs is inspired by the Salamandra Robotica II [Crespi et al., 2013]. Rotating each leg 360° in its joint is a simpler mechanism to push the body forward than a biological folding limb. Though the design is somewhat detached from a biological leg, it is hoped that the implementation of a gait will restore a lizardlike motion.
To replicate a lizard's gait, diagonally opposite legs will be accelerated in rhythm with the motion of the body.

Figure 1. A diagram of a lizard taking a step, showing how bending the body contributes to the length of the step.

Figure 5: *A representation of the relationship between a lizard's gait and its body oscillation [Alexander, 2012].*

2.2 AI Design

2.2.1 Genetic Algorithm

Evolution can be used as a *"method for designing innovative solutions to complex problems"* [Mitchell, 1998, p. 4]. Lizardbot aims to emulate the biological evolution of lizards to find optimal solutions among the many thousands of possible robot permutations: a mimicry of natural selection. A genetic algorithm (GA) is a form of artificial intelligence that more closely models the characteristics of natural selection. GAs can be used to *"[abstract] biological evolution"* [Yang, 2021, p. 1], an approach that is aligned with the naturalistic approach sought by Lizardbot . As Xin-She Yang discusses, GAs are also suited to Lizardbot's broader optimisation problem, as they are fit for independent agents operating parallel to each other and for complex problem spaces [Yang, 2021]. Additionally, Lizardbot is not searching for a global optimum: not all robots in the population are expected to succeed in converging on a solution, and those that do may offer differing solutions. With this freedom, a *"GA will have a good chance of being competitive with or surpassing other "weak" methods"* [Mitchell, 1998, p. 116]. However, the parameter values used by the GA can undermine its success: *"inappropriate choice will make it difficult for the algorithm to converge or it will simply produce meaningless results"* [Yang, 2021, p. 2]. To mitigate this, the GA will undergo hyperparameter tuning to determine suitable values.

An agent's parameters are broken down into 'genes' that can be manipulated by three main processes: selection, mutation, and recombination (or crossover) [Mitchell, 1998].

Lizardbot utilises a slightly different selection process to tradtional GAs. Conventionally, all agents would be considered for recombination/mutation each generation. Instead, an algorithm will determine when a robot is stuck and terminate the generation of that robot in isolation - evolving the population asynchronously.

The mutation stage randomly adjusts a selection of the genes to allow random variations to form in the population. Mutation can *"[insure] the population against permanent fixation at any particular locus"* [Mitchell, 1998, p. 129]. Essentially, mutation avoids convergence within the population and encourages the formation of new solutions.

The recombination stage mimics the genetic crossover that occurs when two animals produce offspring. The 'breeding' will occur between the selected robot and another chosen either for its physical or movement similarity.

Several alternative recombination methods were also considered. A *Triad* approach breeds the input robot with two others: a robot selected for its physicality and another selected for its movement. The offspring of three successful robots is expected to be more successful than the conventional two as it will provide more genetic diversity during recombination. A parallel in nature is the multiple matings found in Harvester ants, improving colony performance via *"increased genetic variance within the worker population of the colony"* [Wiernasz et al., 2004, p. 1601]. Even with multiple matings in nature there will still be the traditional two genetic inputs. However, there is a connection between the two methods and their **opportunity** for genetic diversity. The *Triad* method also aims to strengthen the relationship between body and movement. Evolution Gym's co-design GA mutated these two mechanisms in tandem and produced higher-performing robots in most tasks being tested [Bhatia et al., 2021]. This is discussed in more depth in 3.4.

Another recombination method is a *Lizard* option whereby the mate is selected proximally using a characteristic with no direct impact on performance. Male spiny-footed lizards appear to favour redder females - a trait that appears to signal sexual maturity [Belliure et al., 2018]. For these lizards the body colour appears to signal sexual maturity, a physiological trait that can impact on the success of the population but is not as directly linked as traits usually considered to be desirable (signs of health or fighting ability, for example) [Lappin and Husak, 2005].

For this project, the desirable trait is the body colouration as it is for spiny-footed lizards, though Lizardbot loves blue as opposed to red. In theory, this method should create local 'ecosystems' in the population over time as robots that explore the same area of the terrain each iteration will always interact with the same selection pool.

The genetic algorithm aims to optimise the parameters of Lizardbot and, once an optimal robot has been found, could be removed from the project. It has no impact on the behaviour of the robot mid-generation in reaction to specific circumstances.

2.2.2 Dynamic Movement

Dynamic systems theory uses the causal webs of an environment (the action of an agent influences other entities that in turn have their own causal relationships) to determine the *"sequences of states that could take the system from one location in state space to another"* [Clark, 2001, p. 122].
The overall idea is that the robot will explore how to move in a direction through trial and error, as opposed to being provided with rigid rules. Essentially, it will 'collect' behaviours. If a specific set of parameters produced a desired behaviour (move left, for example) then this can be activated with increasing force (see Figure 6) to steer the robot left across the terrain. This approach may result in emergent complex behaviours that would otherwise have been a mammoth task to implement as a set of instructions.

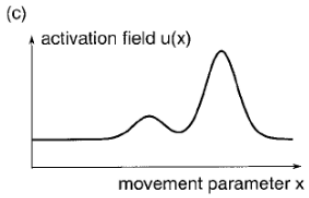

Figure 6: *An example of a movement being activated: weakly at first, then again with a stronger activation to produce a greater movement [Erlhagen and Schöner, 2002].*

Thelan and Smith explored the behaviour of babies taking steps on a treadmill and concluded that there was a complex relationship between the body, environment and behaviour [Thelan and Smith, 1994]. This relationship can be argued as both advantageous for Lizardbot, and a potential constraint to the performance that dynamic movement can offer. Dynamic movement relies on assumptions about this relationship, as it is not feasible to store every factor that culminated in a behaviour. Thus the system must be reduced to a select few control parameters. This reduces the worldspace into a very narrow definition revolving around the control parameters (velocity, for example). This vast oversimplification renders the dynamic approach unsuitable as the primary movement mechanism. However, it may still complement the movement algorithms discussed in section 2. Andy Clark discusses the idea of "partial programs" [Clark, 1997] - *"minimal instruction sets that maximally exploit the inherent (bodily and environmental) dynamics of a controlled system"* [Clark, 2001, p. 133]. This approach will be taken by the dynamic movement algorithm by building on the more rigid movement algorithms instead of replacing them. As long as Lizardbot's adaptive behaviour can work alongside other causal influences then dynamic movement is expected to improve its performance.

2.3 Terrain Design

Three terrains will be generated to test the robot in increasingly difficult environments. The terrains will be randomly generated once and then stored and used as a control variable in experiments. The terrain types $[Smooth, Uneven, Rough]$ are inspired by those used to test the Octopus robot [Cianchetti, 2015], as they demonstrate the efficacy of a robot in increasingly difficult test scenarios. For Lizardbot, they will also show how generalised evolved solutions are to new environments.

The inclusion of a smooth terrain is crucial in considering the situatedness of the robot. Herbert Simon provided an elegant example of the importance of this consideration: an ant is observed making its way back to its nest across a beach. Its route is 'a sequence of irregular, angular segments' that suggests some level of complexity in the ant's behaviour. However, the beach for the ant is a much harsher environment than it is for a human. It is more likely that 'its complexity is really a complexity in the surface of the beach, not a complexity in the ant' [Herbert, 1996]. Thus, the situatedness of the robot could culminate in behaviours that are not of its own making and are instead caused by its relationship with the terrain. The smooth terrain should reduce the role of the environment and allow for emergent behaviours to be prescribed to the robot itself.

2.4 Extension - Vision

Thus far, there has been little focus on the collisions that a robot may encounter, risking damage when translated to a physical robot. When a robot becomes stuck it will likely find itself bouncing against the nearby terrain, forcing the prioritisation of durability in the design of the robot. If the robot were given a rudimentary visual system it could instead anticipate collisions and adjust its course to avoid them. *"Most animals respond avoidantly and directionally to the abstract visual stimulus ... which specifies the approach of an object and impending collision"* [Schiff, 1965, p. 1]. Implementing a visual system to the Lizardbot could unlock a variety of more complex naturalistic behaviours.

3 Related Work

There has been a vast amount of research conducted into optimising robots for various environments, including many that explored designs derived from nature. Several robots took a similar approach to Lizardbot, focussing primarily on reptilian characteristics.

3.1 Modular Snake Robot

The work of Ye et al. [Ye et al., 2010] found that a modular snake robot was *"more efficient to get into complicated environments"* [Ye et al., 2010, p. 453]. Their aim was to test the abilities of such a robot in environments too harsh for humans. Each module operated independently in a manner similar to the body movement algorithm covered in 2.1.1. Each module was attached to the adjacent modules via a joint that rotated to create a wave through the body, whilst a servo drove the module forward. Their robot used a cosine function to manoeuvre the body, whereas Lizardbot will make use of an approach closer to that of a central pattern generator.
The modular snake robot successfully navigated any obstacles with a height less than the height of a module. It also demonstrated the constraints that a physical robot can face (for example, the error rate in the servos increased with amplitudes over 4cm). Lizardbot aims to explore similar methods of movement to improve its environmental adaptivity, without the physical limits faced by the modular robot. The addition of legs and a tail to the modular design is intended to enable the robot to overcome more complex obstacles in its environment. Meanwhile the AI will, theoretically, allow the Lizardbot to adapt to its situation more than the modular snake robot was able to.

3.2 Agama Robot

The tail-assisted pitch control used to build the Agama lizard-inspired robot heavily influenced the tail motion of the Lizardbot [Libby et al., 2012]. It constantly assesses the momentum of the robot and adjusts the motion of the tail to counterbalance this force. Applying this research may reduce the risk of the robot overbalancing, a situation that could damage a physical robot. The Agama study found that *"the robot with [proportional–derivative] feedback tail control maintained a nearly constant body angle by swinging its tail upward and incurred 72% less rotation after a perturbation than did the robot without tail control"* [Libby et al., 2012, p. 181]. This is a promising result for stabilising the Lizardbot.

3.3 Salamandra Robotica II

The relevant goal of the Salamandra Robotica II [Crespi et al., 2013] was to *"advance robotics design for bimodal and efficient locomotion"* [Crespi et al., 2013, p. 309]. The design of this robot heavily inspired various elements of the design of the Lizardbot: the full rotation of the legs and the CPG-inspired oscillation of the body.

One of the key differences between the Salamandra and the Lizardbot is the method for turning. The former used calculated asymmetric oscillations to produce a curving trajectory (see Figure 7) while the latter will dynamically derive an activation function for a turn.

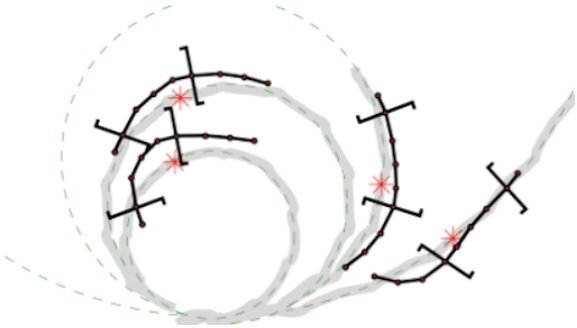

Figure 7: *The results of various curved trajectories of the Salamandra [Crespi et al., 2013].*

Another difference was the Salamandra's passive tail. The tail fin was only used to test its effect on speed whilst swimming; removing it decreased the speed by 63%. The rigidity of the tail will be matched by this project but, as previously discussed, it will rotate to counterbalance the motion of the rest of the body.

3.4 Evolution Gym

Evolution Gym [Bhatia et al., 2021], unlike the previous works covered in this section, modelled their robots rather than building them. It shared a common goal with Lizardbot: optimise the movement and body together using a co-design algorithm.

Evolution Gym likewise employed a (much simpler) genetic algorithm. Evolution was conducted synchronously using the top $x\%$ of the population and did not feature recombination. Other optimisation algorithms were tested against the GA on a series of both locomotion and manipulation tasks and the evolved robots were compared to a hand-crafted selection. The former were able to find a balance between their structure and motion in a way that the hand-crafted robots could not. Interestingly, the autonomously generated robots resembled *"existing natural creatures"*[Bhatia et al., 2021, p. 1] - a result that it is hoped Lizardbot will achieve too.

4 Requirements Analysis

It is difficult to predict what the specific applications of this project would be and what range of possible constraints they may have. Are elements of the physical design restricted? What kind of terrain is the robot going to be placed in? Are there other factors that need to be accounted for (gravity or bodies of water, for example)? This project is designed to be broad such that it can be adapted for a narrower use case. The requirements focus on flexibility and reusability of Lizardbot to lay the groundwork for future research. Lizardbot could be tailored to create a robot for bomb disposal, planet exploration, or even a robot hoover that can handle stairs! The features that a target system would include are outlined here, divided into those that are feasible and those that form a limitation of Lizardbot.

4.1 Functional Requirements

4.1.1 Realistic Physics

The robot will need to interact realistically with the terrain around it such that the results are translateable to a real world scenario. If it were to remain on a ledge in a precarious position that any physical robot would have immediately fallen off then this is not a realistic model. Evolution Gym similarly sought a realistic physics engine to accurately represent the forces acting on their robot model [Bhatia et al., 2021]. They opted for a mass-spring dynamics system, whereas Lizardbot will adopt a rigid body dynamics system via the Unity Physics [Unity, 2018]. This is an advanced system that is expected to model collisions and gravity to a high degree of accuracy.

4.1.2 Transferable Design

It would be beneficial for target users applying Lizardbot to a physical robot if there were a direct correlation between the physical structure of a robot and the model it were derived from. Lizardbot has a vague shape but the underlying structure would be transferable to a blueprint for a robot. The model will deliberately use realistic constraints to avoid outcomes that cannot be replicated (for example, the maximum degrees of freedom will be bound).

4.1.3 Collision Avoidance

Regardless of its purpose, any user will want their robot to avoid behaviour that risks damage. This function can be somewhat achieved by having the AI predict the upcoming terrain and proactively decide how it will navigate it. The AI should introduce a bias toward robots that minimise their collision force: by avoiding collisions and falls, the largest risks can be reduced.
Additionally, the model could approach this objective via the visual system that has been established as a possible extension to the AI. The work of Lihui Wang [Wang et al., 2013] demonstrated the success of such a system: a vision sensor relayed information about potential incoming collisions such that they could be avoided. These early reactions prevented the need for more dramatic responses during active collisions.

4.1.4 Functional AI

It is important that the AI is capable of evolving an input population to a higher performing one over time. The constraints placed on it may limit its success; nonetheless it should display an upward trend.
The AI should implement a suitable performance metric that does not introduce unnecessary bias towards certain physical structures or behaviours to allow for more freedom in the resultant robots. Over time, it should demonstrate an improved performance in the overall population, translating to robots successfully navigating further across the terrain than their predecessors.

4.1.5 Dynamic Movement

It would be advantageous for the user if Lizardbot were able to 'learn' from prior behaviour and reactivate this to explore farther across terrains. Dynamic movement aims to address this requirement by implementing a 'partial program' approach [Clark, 2001] to add a dynamic element above the fundamental movement algorithms. The dynamic movement should determine which inputs resulted in a motion in a given direction, using these same inputs to progress the robot across the terrain. This is predicted to provide a drastic improvement on the performance of a population.

4.1.6 Multiple Terrains

The robots evolved by the AI should show a similar performance when placed into a new terrain, to demonstrate that their motion is optimised for a range of environments as opposed to simply the one they were trained in. The importance of testing across increasingly complex surfaces was highlighted by the Octopus robot [Cianchetti, 2015]. Additionally, the inclusion of a featureless terrain will provide insight into the role that the environment has on the emergent behaviour.

4.2 Project Limitations

4.2.1 Physical Testing

A model is excellent for testing theories but can overlook the 'real physics' that becomes apparent when building a physical robot. For example, the modular snake robot needed to ensure that the robot had *"enough space for installing the joint driving mechanism and the circuit modules"* [Ye et al., 2010, p. 450]. Unfortunately, building a physical robot would require time and expensive resources. It will not be possible to test the performance on an actual robot. This will hopefully be compensated for by having a realistic physics system and a model whose design is directly transferable to the blueprint for a robot.

4.2.2 Customisable Input

A target user may already have known constraints for the design of their physical robot and wish to see this represented in Lizardbot. Likewise, the ability to model a specific environment may be valuable for modelling the effectiveness of a robot in a terrain it is going to encounter. The parameters used by the project will be contained to *Config* files that can be adjusted with minimal programming knowledge. However, it will not be possible to accomodate every criterion so a user may still need to extend the project to their needs. The current expectation is that the user will require some knowledge of Unity to interact with the project. The addition of a UI may be desirable in future to lower the technological threshold.

5 Implementation

5.1 Terrain Generation

Three terrains were generated using Procedural Toolkit [Syomus, 2021] to test the performance of the robot across various environments: rough, uneven, and smooth. It is worth noting that the three terrains do share some common properties (gravity, for example) and these factors may introduce bias in the AI. For proof of concept the sample set of terrains is sufficient.

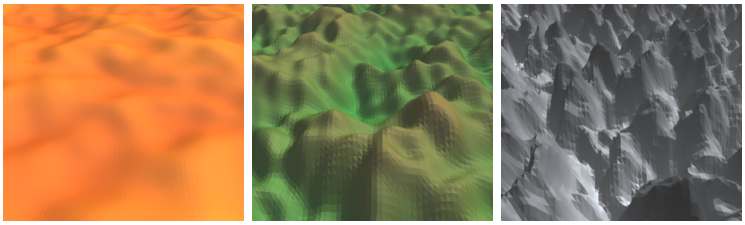

Figure 8: *Examples of the three terrain types. (From left to right) Smooth, Uneven, Rough.*

At one point the height of the terrain was proportional to the number of sections of the robot, a similar method to that of the Octopus robot [Cianchetti, 2015]. However, as the terrain is a control variable the heights were switched to a static value: $Smooth = 8, Uneven = 16, Rough = 24$. Overall, the rougher the terrain, the higher and more closed in it is. Most of the development of the robot was conducted on the smooth and uneven terrains, as the rough terrain aims to provide a more extreme environment with which to test the efficacy of the AI.

5.2 Robot Implementation

5.2.1 Body

To create a snakelike body, each body module is attached to the previous module by a configurable joint [Vladimir T, 2020] and has two methods of movement: driving and rotation. The physical design of the robot creates a fluid motion before any complex movement is applied. With the joint structure, the motion of one section is translated to those behind it, similar to dragging a piece of string along the ground. This is shown in Figure 9a: the head rotates and, after a delay, creates the same angle in the sections behind it.

The central pattern generator (CPG) inspired approach allows each module to react to the velocity and angle of the previous section. The velocity v to apply to each of the n sections is calculated using the below equation. This was adopted from Tony Dear's multi-link snake robot: a robot with a similar modular design with passive joints connecting the modules [Dear et al., 2020].

$$\overrightarrow{v_i} = \overrightarrow{v_{i-1}} + \frac{n}{2}\overrightarrow{w}$$

Lizardbot utilises the same equation to calculate the velocity as the multi-link snake robot but with one distinction: Dear's robot split the velocity vector into its axes, using cos for the x axis and sin for the y. Lizardbot instead calculates the vector as a whole and alternates the rotating sections between sin (S) and cos (C) to produce the serpentine motion. For robots with serpentine motion disabled, S and C will be assigned randomly to the m rotating modules (where $m \leq n$).

The value of w will be calculated using S or C as specified.

$$S : \overrightarrow{w} = sin\overrightarrow{\theta_{i-1}} + sin\overrightarrow{\theta_i}$$

$$C : \overrightarrow{w} = cos\overrightarrow{\theta_{i-1}} + cos\overrightarrow{\theta_i}$$

It is worth noting that every velocity calculated (for a body module, leg or tail) is multiplied by a parameter that can be mutated by the AI. Mostly it affects the primary axis of rotation, which allows the path of motion to be adapted over time.

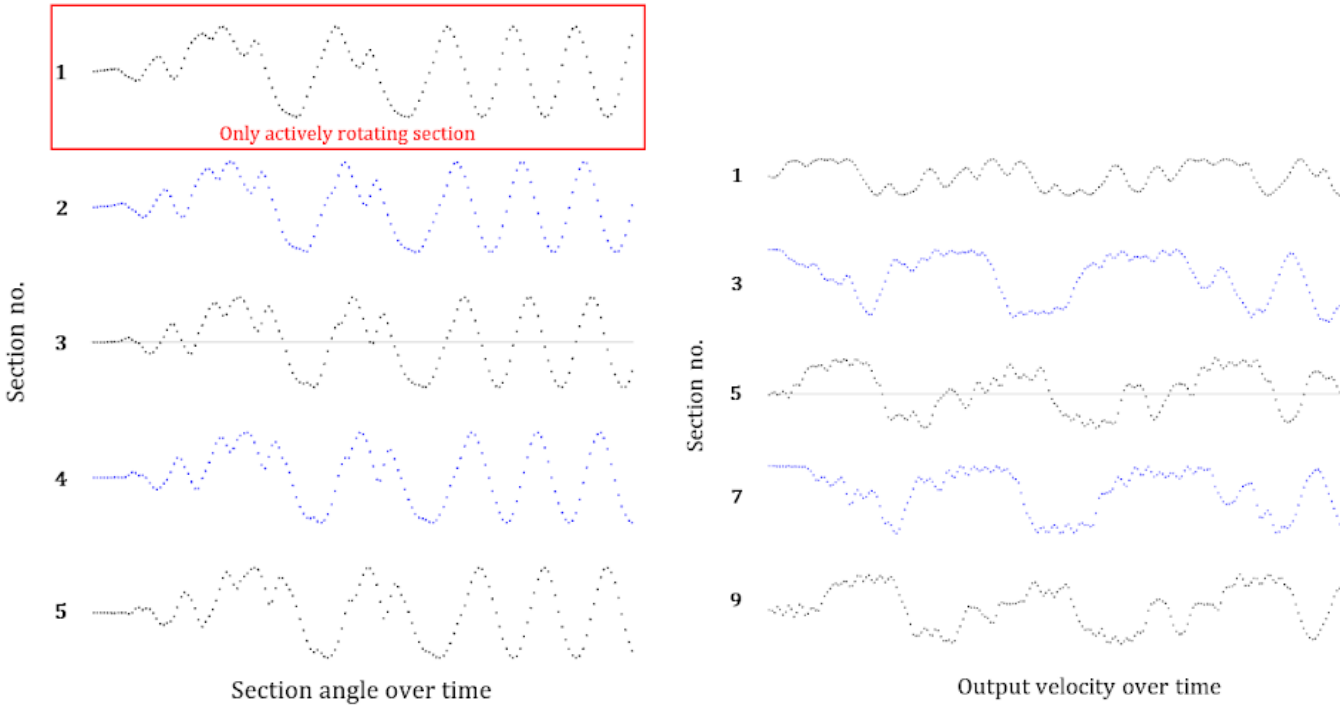

Figure 9: *(a) Left, (b) Right*
(a) Demonstration of body motion with a single rotating section at the head of the body.
(b) The output velocities generated by each body module with serpentine motion enabled. Modules using S are shown in black, C in blue.
Created using Grapher [NWH Coding, 2017].

For serpentine motion, each section 'reacts' to the previous one whilst using the opposing equation to output a velocity that is almost an inversion of its predecessor. Moving back through the body there appears to be more fluctuation in the values as more noise is introduced through each application of the equation. The advantage of using this recursive approach is the incredibly organic behaviour that it produces. The first prototypes of the project used hardcoded timings and velocities to try and mimic a serpentine motion and the rigidity of the code was evident in the behaviour. With the above equation applied, the motion of the body appears completely natural. As the Lizardbot slithers through troughs in the terrain or wriggles whilst stuck on a ridge, it is easy to forget that it has no awareness of its surroundings. Each module is simply reacting to the modules that come before it in the body.

5.2.2 Tail

As discussed in 2.1.2 and 3.2, a tail has the potential to counterbalance the body and provide stability as the robot moves. The Agama robot used the angular momentum of the body to calculate how to calibrate the tail vertically as the robot jumped. Lizardbot implemented a similar approach in three dimensions by calculating the total momentum of the robot around its centre of gravity (COG) at each frame, and adjusting the velocity of the tail accordingly.

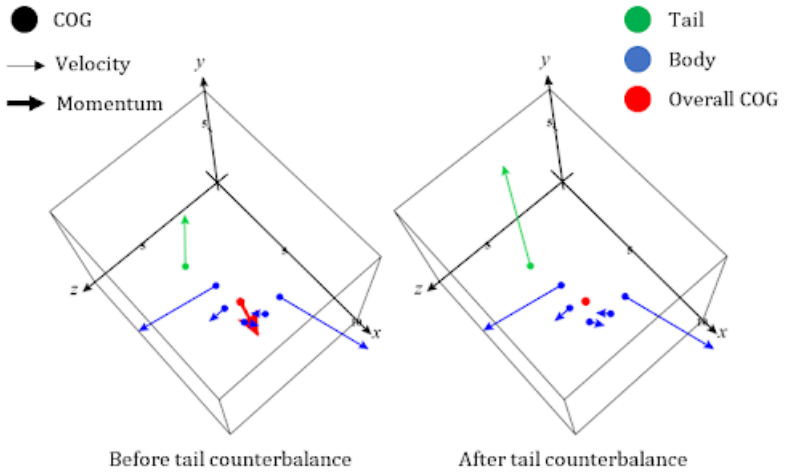

Figure 10: *A representation of the tail being adjusted to conserve the angular momentum of a robot. Diagram created using CalcPlot3D [Seeburger, 2021].*

For all body parts $i = 0, ..., n$, the radius of the path of motion r is the distance from the individual COG x to the COG of the overall robot.

$$r = |x_i - \frac{1}{n} \sum_i^n m_i x_i|$$

The total angular momentum L of the robot is calculated using the above values of r, the mass m and the velocity v of each body part [Rafay, 2022].

$$L = \sum_i^n r_i m_i v_i$$

To conserve momentum, the velocity of the tail is calculated by inverting L and dividing it by the tail's mass and distance from the overall COG.

$$v_t = -\frac{L}{r_t m_t}$$

Another simplified approach was considered whereby the overall velocity of the robot was counterbalanced instead. However, this was found to create sharp changes in the velocity of the tail that could cause it to fling the entire body into the air. Whilst this showed promising behaviour for the basis of a jumping motion, it was counter-productive for a feature whose goal was to stabilise the robot. Additionally, by accounting for the COG, any difference in mass between components is taken into consideration. Thus, the tail is able to counterbalance any body structure (assuming that the motion of the tail is not physically blocked by the position of a body part).

The design of the tail assumes that nature has already selected for the optimal location by placing the tail at the back of a creature. This assumption seems intuitive: most animals, including lizards, are symmetrical. The location of the tail maintains this property whilst keeping the motion of the tail in the same plane as the rest of the body. In future, this assumption could be removed to explore how the position of the robot affects the robot.

5.2.3 Legs

Legs were added to Lizardbot in an attempt to model the gait of a lizard, as outlined in 2.1.3. Inspired by the design of the Salamandra Robotica [Crespi et al., 2013], each leg is designed to rotate in a circle to push the body forward. The physical design of the leg matches the elliptical shape of the tail and uses the same configuration of customisable mass and length. Currently, there are only two attachment points for a leg on each body module: one each side, perpendicular to the body joints. For a uniform body, the legs are placed symmetrically along the body and given equal length and mass; they are constructed randomly for non-uniform bodies.

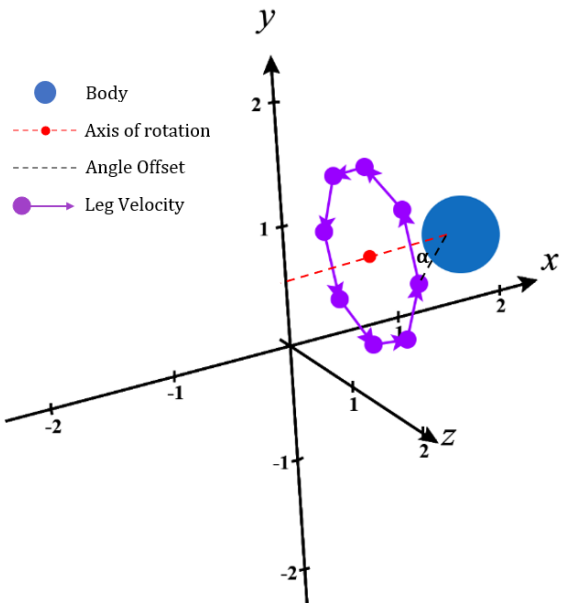

Figure 11: *An illustration of the rotation of a leg around an axis of rotation. Only one attachment point is shown - another would be available on the other side of the body along the x axis. Diagram created using CalcPlot3D [Seeburger, 2021].*

The leg rotates around an axis perpendicular to the body section it is attached to, as shown in Figure 11. The size of the circle it follows is determined by the angle it is offset by when attached to the body (α). The rotation follows a series of conceptual points spaced 30° apart around centre point D (shown by the red point on the axis of rotation in Figure 11). These points are generated using the following equation:

$$P = D + V cos\theta + U cos\theta$$

Where V and U are two vectors perpendicular to each other in a plane through D and the target circle, and $0 \leq \theta < 360$ [David K, 2018].
To calculate the desired velocity of the leg, the point P_i closest to the current position of the leg is found. From this, the new velocity can be calculated using the vector to the next point on the circle.

$$v_i = g(P_{i+1} - P_i)$$

If the relevant gene is active, the gait multiplier g increases the velocity when the body is turning away from the leg, decreasing it if the body is turning toward the leg.

Currently the resultant leg motion is a lot of twitching. It is possible that an alternative joint mechanism would allow for Lizardbot to stand on its legs and appear more natural. It would also

be interesting to explore how the location of the legs impacts the robot as a whole. Currently the attachment points are based on lizards (indeed, most animals) and the years of evolution that have led to their structure. However, the offer of more variety in the design of the robot may offer alternative solutions to navigate the terrain.

5.3 Artifical Intelligence Implementation

A population of Lizardbots uses a genetic algorithm (GA) to "evolve" over a series of generations. The aim of this GA is to improve the performance of the robots over time. The GA is performed when a robot is declared to be stuck by the trapped algorithm.

5.3.1 Performance

The metric used to measure the performance of a robot can heavily influence the outcome of the genetic algorithm. As the desired outcome is a robot capable of navigating the terrain, the base measurement used is the furthest distance (by magnitude) that it has travelled in a given generation from its spawn point within the terrain.

Two additional parameters are used to add context to this base measurement. The first rewards robots that move quickly. The current performance is multiplied by the average speed (distance / time) of the robot since it spawned.

The second penalises robots that cause large collisions. Every time a body part encounters a collision it triggers a method that will measure the force that the body part has experienced. Unity's built-in physics system returns the impulse I of a collision. Using this, the force can be found by dividing the impulse by time: $f = \frac{I}{t}$ [Serlite, 2016]. If the force is found to be above a threshold then the robot will be penalised by deducting 10% from the performance. This ensures that the robot is not evolving toward a behaviour that carries it across the terrain efficiently but would cause damage to itself in the physical world. An example of this is an iteration whereby the robots were observed 'flicking' their tails rapidly downward, causing the robot to be thrown into the air and across the terrain. Whilst this was effective at getting them to the edge of the terrain within seconds, this approach would shatter most robots. This threshold can be tailored to the needs of the robot, such that robots with shielding or soft bodies can be given a higher threshold to allow more risky behaviour.

A flaw in the performance metric is that it does not differentiate between the routes that robots take. If one robot moved quickly but erratically, it may be rewarded equally to one that moved slowly and directly. However, this metric appears to provide a suitable balance between rewarding robots that travel the farthest, further rewarding those that move efficiently, and reducing behaviours that would damage a physical robot.

When referring to the performance of a population, the mean performance of the top 25% is being measured. It was found that analysing the entire population added too much noise to the results, as when a robot is mutated and respawned its performance returns to zero.

5.3.2 Trapped Algorithm

It is important for the AI to know when the robot is stuck to terminate its current generation. This trapped behaviour can take many forms: from bouncing against the same point in the terrain to circling itself.

The locations of the robot over the last t seconds are analysed. The algorithm finds the minimum and maximum values for each axis to draw a conceptual cube around the points the robot has visited in the last t seconds. These cubes depict the worldspace the robot has recently explored. The variance of the data set is calculated for the volume of the cube rather than the coordinates themselves. If the variance of the volumes of the cubes converges to zero then the robot is considered to be trapped.

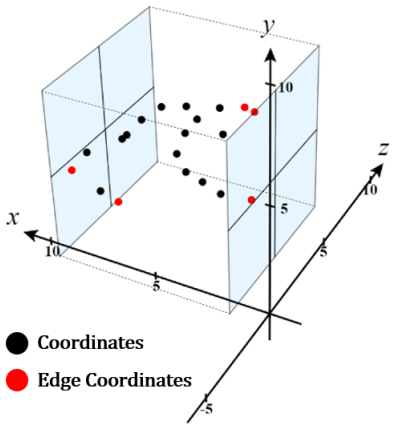

Figure 12: *A representation of the cube constructed around the last 20 locations a robot has visited (captured twice a second). Diagram created using CalcPlot3D [Seeburger, 2021].*

The red points in Figure 13 align with behaviours that can reasonably be classed as 'trapped', however it is important to note that this algorithm was implemented with some bias toward certain behaviours (looping, for example). For this project, the identification of a trapped robot appears sufficient and is expected to produce an AI that will train away from continually exploring the same area for too long. The algorithm has other potential applications in finding looping patterns in any problem that can be assigned a worldspace. For example, a neural network could be analysed with this algorithm to determine when it is 'stuck' whilst performing a task, or a search algorithm could implement it to avoid excessively exploring within a section of the graph.

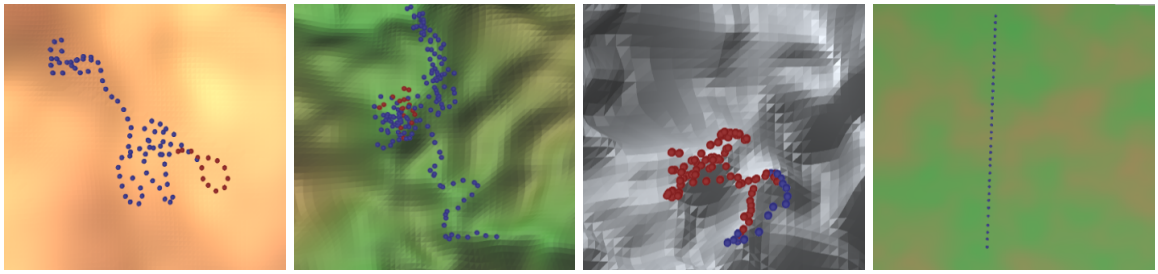

Figure 13: *A demonstration of the trapped algorithm using the same robot and $t = 20$ on each terrain.*
Each blue point represents the location of the robot being captured. Red indicates that the robot is trapped (and would normally have been disabled and passed to the genetic algorithm).
(From left to right) Smooth, Uneven, Rough, Flat. The latter used a robot with rotation disabled to ensure that a robot travelling in a straight line would not be flagged.

5.3.3 Genes

The manipulatable characteristics of a robot are distinguished by a *Gene* class. Each variable is instantiated with a default, minimum, and maximum value. Additionally, they are given a type (established in the enum class *Variable* whereby negative enum values are physical properties and positive are movement).

This class handles any erroneous situations that arise and allows Boolean values to be stored as a float. If the *Get* method of a Boolean *Gene* is called, then true will be returned for values greater than 0.5. This allows Boolean genes to be mutated slowly (for example, from 0.35 to 0.6) without having to make a single jump from one value to the other.

5.3.4 Genetic Algorithm

When the trapped algorithm determines that a robot is stuck it is disabled and passed to the GA.

The mutation cycle of a robot refers to how many evolutions should take place before the resultant robot is compared to the robot it initially branched from. When a mutation cycle is complete, the highest performing of the two is selected as the input for the GA. The reasoning behind this is to allow a robot to mutate to a lower performing robot temporarily, as this may allow it to evolve into a more successful robot a few generations later.

Once the input robot has been determined, this robot is then recombined. There are several methods of recombination available but the overall process is: select k robots using a given method and "breed" the genes of one robot in this pool with those of the input robot.
The available recombination methods are:

1. *Physical*
   The selection pool consists of robots that are physically similar to the input robot. The acceptable margin of difference is increased until k robots are found. This method aims to evolve the input robot toward another that is built similarly and is outperforming its current design.

2. *Movement*
   The selection pool consists of robots that are moving in a similar way to that of the input robot. As above, the acceptable margin of difference is incrementally widened if necessary. This method ignores the structure of the robot and instead evolves it towards another that is following the same movement rules but more effectively.

3. *Triad*
   This approach is more artificial than others. Two robots are selected: one each from the physical and movement methods. The genes of these two robots are then recombined with the input robot. As discussed in 2.2.1, nature does demonstrate examples of more than two agents mating. Lizardbot here explores how multiple mating partners might be beneficial when (unlike nature) all contributers are able to input their genetics.

4. *Lizard*
   This method aims to mimic a more lizardlike breeding process. Those with bluer body colours are selected, ignoring those in the population on the other side of the terrain. *Nearby* is relative to the spawn point of the robot, and limited to those in the same terrain type.

5. *Performance*
   In line with more conventional GAs, the highest performing k robots are collected and a random robot within this pool is selected for recombination. Methods 1-3 aim to evolve the input robot toward a more successful version of its current form. This method instead disregards its current design and looks at the entire population instead.

6. *Random*
   A random robot is selected from the population, to act as a control method.

7. *Any*
   Methods 1-5 make an assumption that using the same method for every generation is optimal. This method instead randomly selects one of the above methods for a single generation.

Each of the input robot's n genes is recombined as follows:

$$G(1)_i = R^{[0,1]} < r \longrightarrow \begin{matrix} R^{[0,1]} < 0.5 \longrightarrow G(1)_i \\ R^{[0,1]} \geq 0.5 \longrightarrow G(2)_i \end{matrix}$$

Where $i = 0, ..., n$. R denotes a randomly generated number in the range $[a, b]$. G(1) refers to the input robot, whilst G(2) is the selected robot. For *Triad* recombination G(2) is randomly chosen from either of the two selected robots, with equal probability.

The recombined robot is then mutated. The genes are divided into physical or movement properties to allow the mutation to be limited to one or the other, or both. The mutation rate m is used to avoid mutating every single gene.
When a gene is mutated its value is adjusted as follows:

$$G_i = R^{[0,1]} < m \longrightarrow (max(G_i) - min(G_i))R^{[0.01,0.1]}G_i$$

The mutation could be improved by switching from an adjustment of $1 - 10\%$ of the gene range, to instead using Gaussian noise [Chopra, 2019].

The GA contains four major parameters that influence its function: recombination rate, mutation rate, selection size, and mutation cycle. To determine the optimal values for these, 47 iterations were run with randomly generated values and the performance of the population measured. Each iteration contained 50 robots running for 600 seconds. The desired outcome was a performance curve that sloped upwards over time. Peaks and troughs were expected as the successful robots returned to the spawn point; the overall pattern of the performance was key.

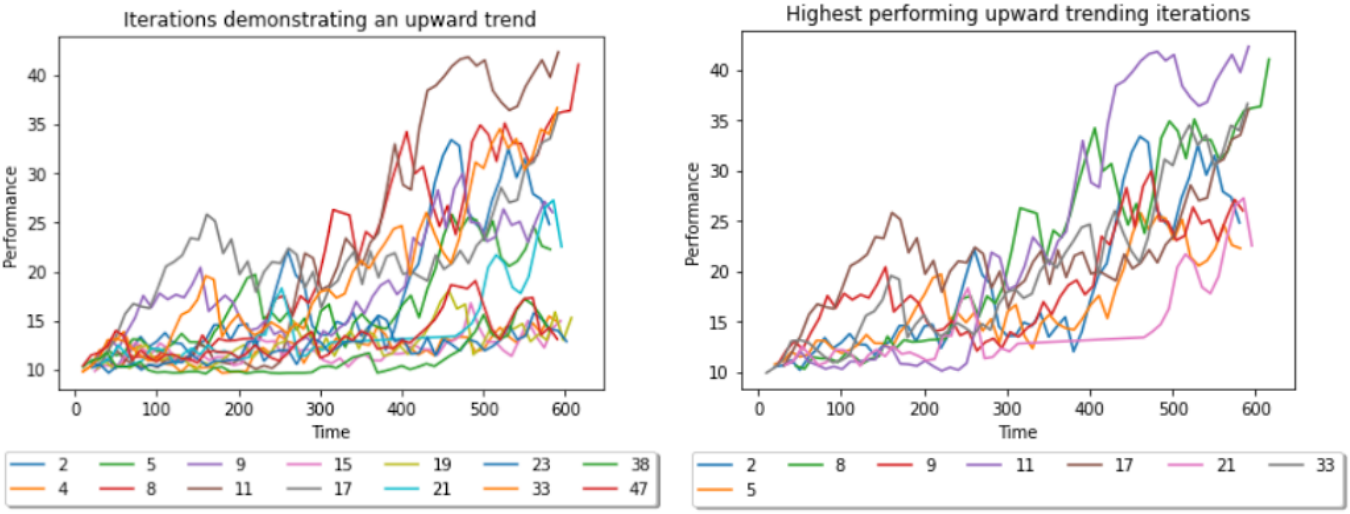

Figure 14: *(Left) All iterations demonstrating a promising upward trend.*
(Right) Iterations from the left graph filtered to those that reached a performance > 20.
Graphs created using Colab [Google, 2022], Pandas [pandas, 2022], MatPlotLib [matplotlib, 2021], and NumPy [NumPy, 2021].

Upon analysing the four values being used for the highest performing upward-trending iterations it appeared that successful values had been found through an issue in the code. When generating random values, Unity cycles through the same values if not provided with a seed [Arycama, 2015]. As the four values were consistently the first four values generated they had been repeatedly set as $[2, 0.77, 0.31, 9]$. Of the 8 iterations that showed an upwards trend and performed well, 7 were these values. Additionally, there was only a single iteration using these values that did not perform well. Given the apparent success of these GA values, these were selected as the default parameters.

It would be useful to have different GA parameter values tailored to each combination of the recombination and mutation methods. Currently, effective values have been found whilst the *Any* option was selected for both in an attempt to find parameters suitable for all permutations. However, this may favour certain combinations more than others and there is currently no data on this. Rather, these base values are assumed to be appropriate and the effectiveness of the methods themselves judged relative to them.

5.3.5 Dynamic Movement

As outlined in 2.2.2, a dynamic approach uses the historical movement of the robot to 'learn' how to move in a given direction again. Take a robot that moves left by rolling over. Due to the reactive nature of the algorithms used for movement, it can be assumed that applying the same velocities from the start of the roll would produce another leftward motion. The dynamic movement algorithm saves these velocities for use when a robot needs to move left again.

To achieve this, the worldspace around a robot is divided. *"The Fibonacci lattice is a simple way to very evenly distribute points [on the] surface of a sphere"* [Roberts, 2020, sec. 0]. Using the implementation of the Fibonacci sphere provided by Fnord [Fnord, 2014], a series of n points were constructed in a sphere around the robot. The value of n impacted the efficacy of the dynamic movement; too low a value of n caused the robot to occasionally veer off to one side of the intended direction. Meanwhile, too high a value of n was unnecessarily expensive and resulted in the majority of the points containing *null* values. $n = 50$ was selected as an appropriate granularity.

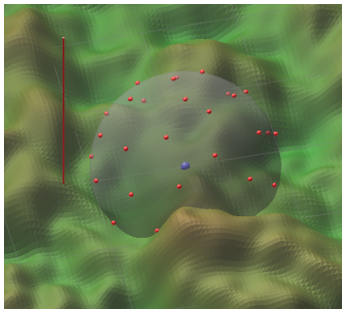

Figure 15: *A sphere created using a Fibonacci lattice ($n = 50$) around a centre point shown in blue.*

Each point on the sphere represents a direction in which the robot could move, as a vector from the centre of the sphere to the point. To 'remember' which velocities resulted in a motion in a given direction, the position of the robot is analysed every t seconds. The sphere point that most closely corresponds to the direction in which it has moved over the last t seconds is found. If there are already velocities stored for this point then the set of values that moved the robot the furthest distance in the time interval are saved. This process of saving the velocities used to move toward a point is illustrated in Figure 16.

It is worth noting that the orientation of the sphere is relative to the robot. When a conceptual sphere is constructed around the robot, it is adjusted to match its rotation. This way, a point directly to the left of a robot with rotation $[30°, 45°, 0°]$ will be in the same relative location to it at $[-20°, 65°, -50°]$.

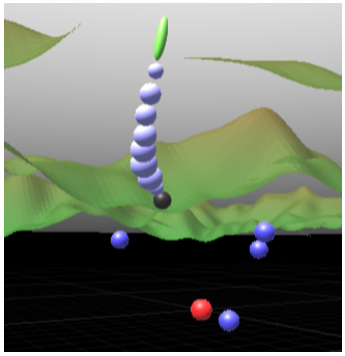

Figure 16: *Each blue point represents a set of stored velocities for a robot that moved left before turning right and moving downward.*

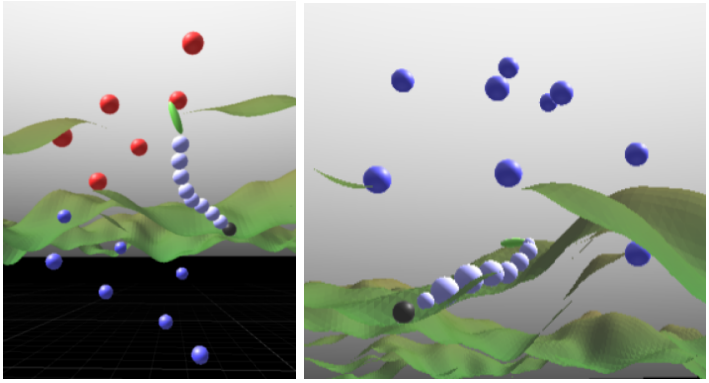

Figure 17: *(Left) A wedge of points that would take the robot away from its spawn point. Those shown in red are those remaining after filtering for the height of the terrain.*
(Right) An example of the terrain height filtering missing a point.

There are two situations in which the motion of the robot is adjusted:

1. Routine adjustment
   The goal of this adjustment is to keep the robot moving away from its spawn point. Every x seconds the vector between the spawn point and the current position of the robot is calculated. The sphere around the robot is filtered using this vector to output a vertical wedge of points moving away from the origin. Applying any of the velocities saved at these points would theoretically direct the robot further across the terrain.
   The points are further filtered by removing those that have not had any velocities stored yet and those that would move the robot below the level of the terrain. Due to the varying height of the terrain, three measurements are taken at incremental distances from the robot. If the direction of the point at this distance would take it below the height then it is removed from the selection. This method aims to take a snapshot of the rough height in a given direction and is liable to miss points that should be eliminated if the peak is between the snapshots.
   From the filtered points, a single one is selected and the velocities applied to the robot.
   Frequent adjustment may affect the ability of the robots to explore new means of movement. To reduce this, the regular adjustment is only enacted if the robot is not already moving toward any of the points in the initial wedge.

2. Scaled adjustment
   This adjustment is enacted when the trapped algorithm determines that the robot is stuck. At this point it is worth mentioning how the activation of the velocities is implemented. For the routine adjustment the applied velocities will be multiplied by a static activation rate. When a robot is trapped, this activation function is increased exponentially in an attempt to free the robot. Failing that, it is mutated and respawned.

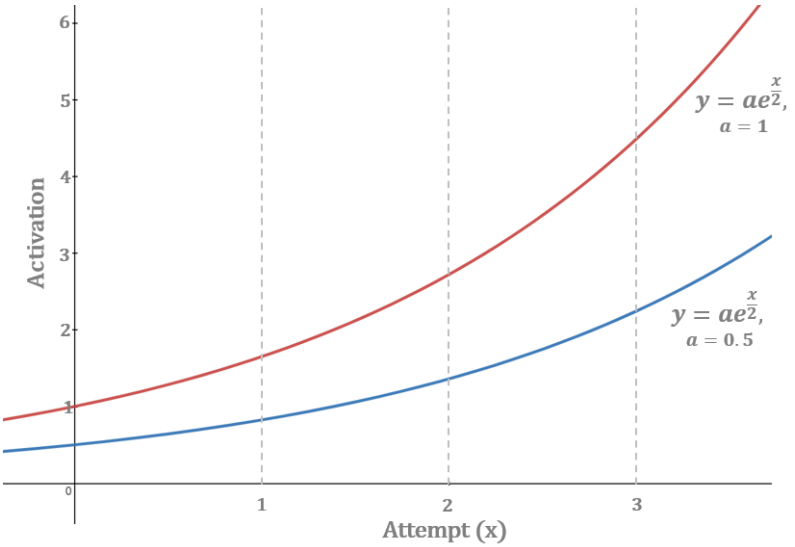

Figure 18: *The activation multiplier used for each attempt at scaled adjustment. The static activation rate is shown as α. Graph created using Desmos [Desmos, 2022].*

It is expected that the dynamic movement algorithm will favour robots that are already performing well. All stored velocities are cleared when the robot is mutated, thus the robots will begin "learning" from scratch every time they become trapped. For robots with sufficient data about their movement, the scaled activation will theoretically enable them to progress in the terrain more often, thus reducing the mutation frequency. As they spend longer in the terrain, less successful robots will have longer to recombine with them. It is predicted that the dynamic movement will both improve the performance of any individual robot, and increase the rate of improvement across a population.

6 Results

For each experiment discussed in this section the population was capped at 50 robots. This is an unfortunate constraint of the project due to technical limitations. The project exceeded the technical capabilities of the computer being used when a larger population was attempted. As such, the data is a snapshot of what Lizardbot may be capable of at a larger scale.

All graphs are made using Colab [Google, 2022], Pandas [pandas, 2022], MatPlotLib [matplotlib, 2021], and NumPy [NumPy, 2021].

6.1 Body Motion

To test the effectiveness of the body motion, a population of 50 robots with 10 modules (no legs or tail) were measured with three forms of motion. All experiments measuring the efficacy of body parts were conducted on smooth terrain to avoid the situatedness of the robot impacting the results.
Random oscillation set random modules to rotate. Due to the embodiment of the robots this motion still resembled a coiling snake. This method outperformed a static body (driving only) which appeared to curve the body around the contours of the terrain. At the slightest gradient this caused the robot to become stuck.
As Figure 19 shows, the serpentine motion showed ∼5x the success of the random oscillation. Thus, serpentine motion became the default motion used in future experiments.

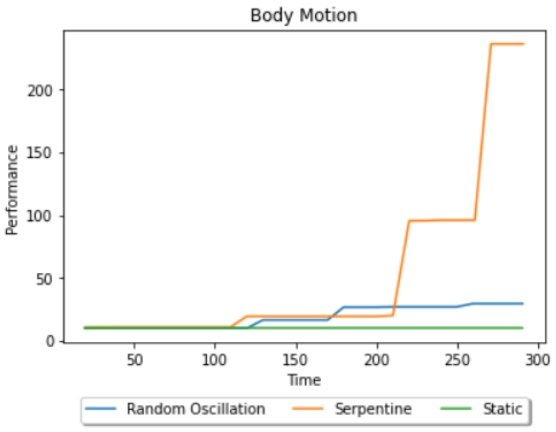

Figure 19: *The results of different methods of body motion.*

6.2 Counterbalancing Tail

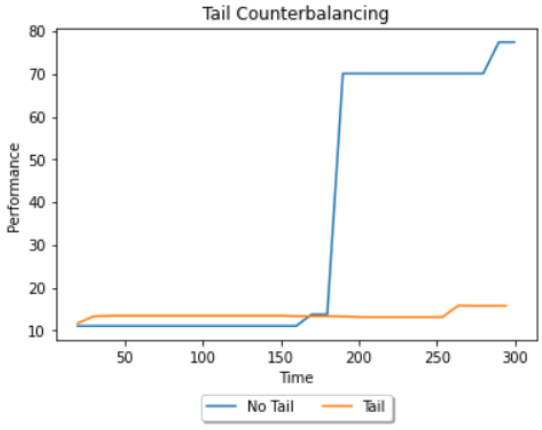

Figure 20: *The impact of tail counterbalancing.*

The impact of a counterbalancing tail was analysed by comparing the performance of a tail-less serpentine body against one with a medium-length tail. The results were damning: the tail significantly limited the population. There are several theories behind this. The first is that the physical parameters of the test tail were unsuitable. It would be useful to repeat this experiment with varying tail lengths and masses to determine if there is a tail form that complements the body. The second is that a rigid tail may be a less effective design. The tail could be observed colliding with the terrain at times which could have thrown the robot off its path of motion. The Agama robot tested a flexible robot and found that after a perturbation a rigid tail produced 72% less impact; a flexible tail resulted in 85% less rotation [Libby et al., 2012]. Given the performance of a serpentine body, applying this to a modular counterbalancing tail may render success.

6.3 Leg Gait

To test the function of the legs, a serpentine 10-module tail-less body was compared against another with 6 legs positioned symmetrically. The latter robot was tested with and without the use of a gait. The results determined that both populations with legs performed terribly. This was not so surprising: as explained previously, the implementation of the legs has some room for improvement.
Interestingly, the use of a gait flatlined the performance. Given a more effective foundation for the legs, it may be that the gait concept would have worked. The Salamandra used a phase rotation between the body and legs to form a gait [Crespi et al., 2013]. This more coordinated approach may have been more suitable than the simple increase in velocity in reaction to the motion of the body.

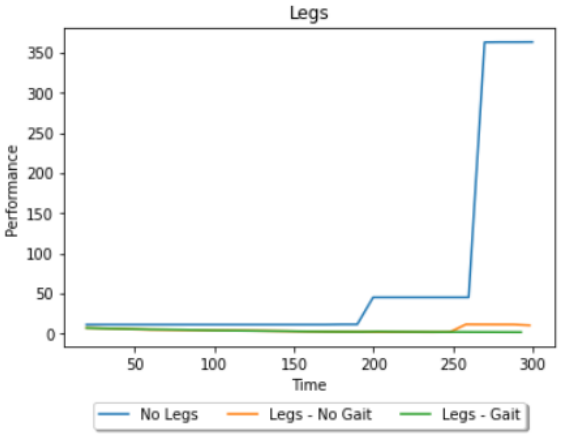

Figure 21: *The impact of the addition of legs with/without a lizardlike gait.*

6.4 Uniform vs Non-uniform Robot

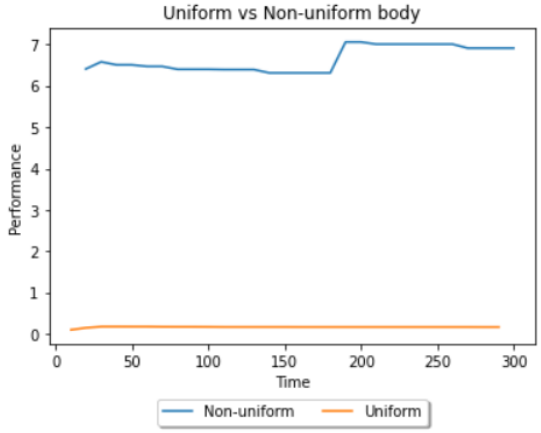

Figure 22: *The results of a uniform vs non-uniform body.*

A population of 50 non-uniform robots were created and left to execute for 300s. At this point they were toggled to become uniform, respawned and measured again.
The results, shown in Figure 22, suggest that forcing a uniform body hindered the robots. There was next-to-no progress made by the uniform version of the robots after 300s.
The ability to force a uniform body was retained in the AI but, given the performance shown here, it is unlikely that this robot would remain in the population for long.

6.5 Mutation Constraints

As explained in 2.2.1, there is the option to restrict the GA to only movement or physical genes. There are 18 permutations available between the mutation constraints and the recombination methods. Providing data for each would have been time-consuming and strained an already struggling laptop. Instead, the most effective constraint (using *Any* recombination) was found and used for 6.6. This was found to be the manipulation of all genes, which produced consistently high results.

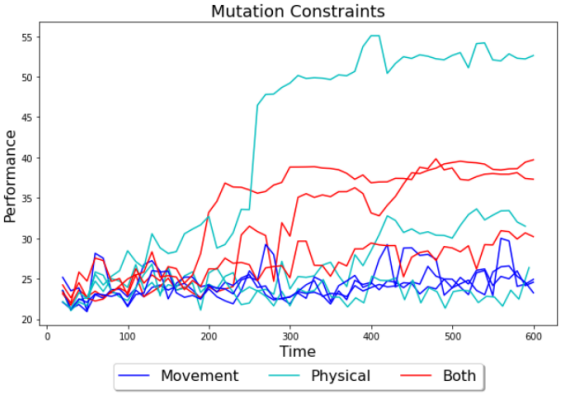

Figure 23: *The results of the GA constrained to physical/movement genes.*

6.6 Recombination Methods

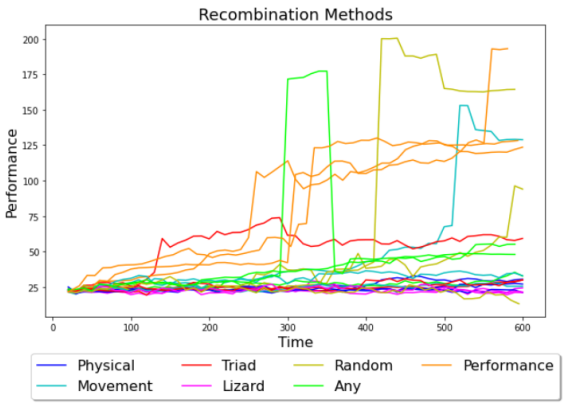

Figure 24: *The results of each recombination method.*
An outlier was excluded for a Random iteration; a single value was recorded at 531.8821

The *Physical, Movement, Triad* and *Lizard* recombination methods all produced disappointing results. *Triad* achieved a single sucessful iteration, but was rejected as a viable option given the lack of progress elsewhere. One explanation for the demise of these recombinations is that the selection methods created localised breeding groups within the population. It may be that with active incest prevention [Mitchell, 1998] these recombination methods would show more promise.
Any recombination demonstrated a consistent (if slow) upward trend but still appeared less successful that the *Random* control group.
The GA demonstrated consistently high performance with *Performance* recombination. Figure 24 shows the clear success of the top 25% of the population. Beyond this, it was observed that almost every robot was consistently reaching the edge of the uneven terrain. However, there are still some fundamental issues with it.

A major issue is that not all genes have equal weight. If the gene that determines whether a uniform body is maintained mutates from false to true then this has a significant effect on the robot. The physical properties of almost all body and leg parts will be adjusted, resulting in a much larger adjustment to the robot than those caused by mutating other genes. Similar large-scale adjustments are applicable when moving to serpentine motion. To mitigate this, it would be useful to apply weights to each gene. Thus, the mutation range could be scaled accordingly, or the genes categorised such that a change to a heavily weighted gene would limit any changes to other genes. This would allow an isolated analysis of the effect that such a large mutation has on the performance.

There is an assumption within the GA that asynchronous evolution is superior to synchronous evolution. The alternative would be to allow every robot to operate for x seconds before mutating a percentage of the population in parallel. This would ensure that every robot experiences the same number of generations and remove any assumptions in the algorithm used to declare a robot as trapped. Intuitively, the current approach is more appropriate as it does not leave unsuccessful robots continuing past the point where they have been identified as trapped. It is also more natural: we do not wait for a person's 30th birthday before allowing them to have children based on how successful they have been thus far in their lives. However, there is no data on the performance of Lizardbot evolving synchronously to support this intuition.

6.7 Dynamic Movement

The addition of dynamic movement had an interesting impact on the performance graph. Seemingly, it results in a high-performing solution significantly better than those without. The exception to this (first graph in Figure 25) is suspected to be due to the GA (with dynamic movement) not having enough time to finish converging. Dynamic movement comes at a cost: it creates a delay in finding a solution and converges to a single solution with no further improvement.

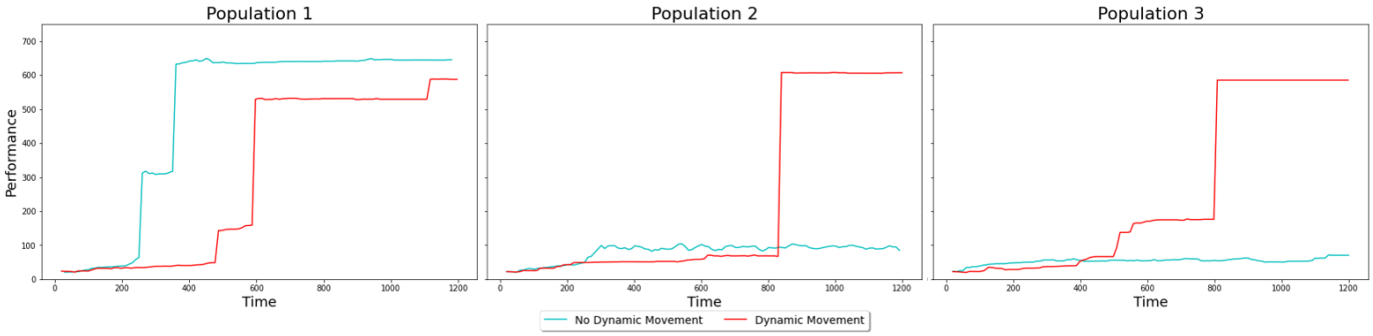

Figure 25: *The impact of dynamic movement on three separate randomly generated populations. Uneven terrain was used, and a GA with Performance recombination and no mutation constraints.*

The dynamic movement could be further improved by performing a more comprehensive analysis of the entire terrain around the robot and making a more informed decision about the direction the robot should move in. This could further prevent the robot from becoming trapped as it avoids higher ground, or alternatively uses a larger activation to "jump" onto it.

6.8 Evolved Robots

The final experiment that was conducted explored how the GA and dynamic movement performed across each of the three terrains. Much of the testing had been focussed on the smooth and uneven terrains so the use of the rough terrain would provide insight into the overall success of Lizardbot. Unsurprisingly, the performance on the two easier terrains was exemplary: one iteration on the uneven surface reached a performance of 15946.23! This result was replicated and is not considered an outlier.
However, progress was slow on the rough terrain. This suggests that Lizardbot in its current form is limited to moderate environments rather than the harsher ones it was targeting.

It is possible that Lizardbot could be improved by repeating the honing of the GA parameters on the rough terrain. It was presumed that calibrating it on the uneven terrain would produce a scaleable GA; this assumption may have introduced bias towards easier environments.

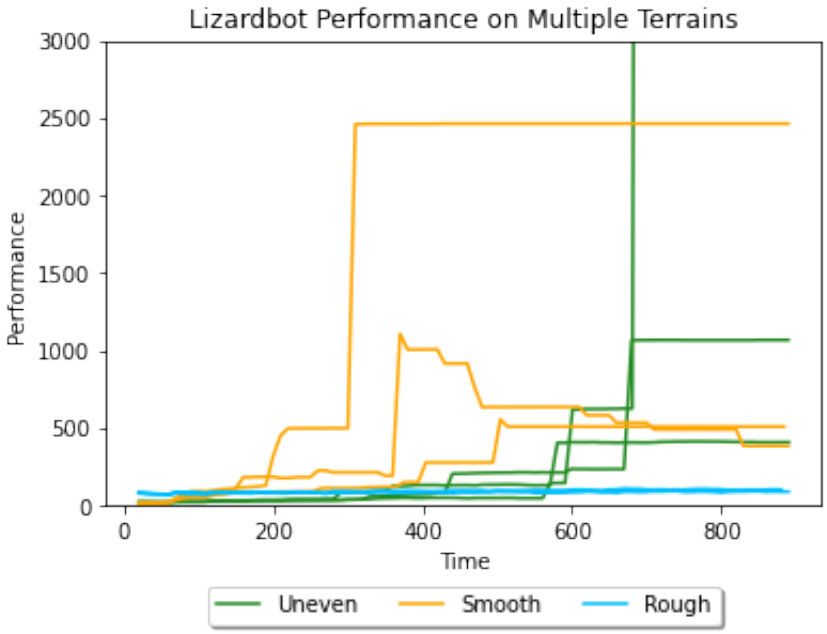

Figure 26: *The outcome of the GA (with dynamic movement) across each of the three terrains. The uneven graph that is not fully visible continues to* 15946.23 *and remains within range of that value.*

Where the results get interesting is in the form of the final evolved robots. The first two terrains evolved the entire population to a single module that drove with speed between the troughs. There was no intention of producing these results but the fumbling balls were undoubtedly an optimal solution. These sped across the terrain - even 'climbing' over peaks - and consistently hit the edge of the terrain within seconds. This speed is likely the reason behind the 16k performance shown by the uneven terrain.

There was a slight differentiation between the resultant robots from the smooth and uneven terrains. The latter produced heads of similar size to each other. Contrastingly, the former gave rise to robots that were both smaller and varied in size. Though a single module has been found as an optimal design, there are still variations within this structure that can tailor it to its environment.
The GA on the rough terrain deviated from this rolling ball result, instead constructing a short string of modules, a tail, and symmetrical legs for 2/3 robots. It is worth taking these resultant robots with a pinch of salt due to their performance.

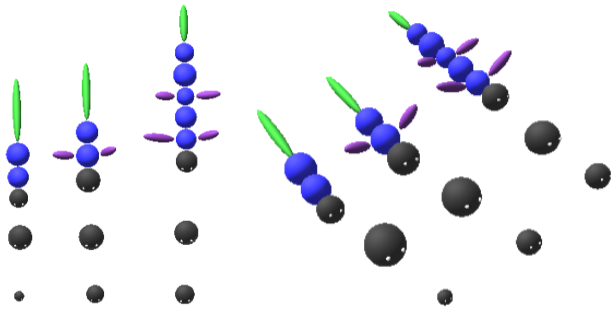

Figure 27: *The highest performing robots evolved by the GA. The rows correspond to each terrain: rough, uneven and smooth respectively (from top to bottom).*

A behavioural analysis was conducted by placing these 9 robots back onto each terrain (with the GA disabled). As expected, all 6 balls immediately began rolling across the smooth terrain. On uneven, only the larger ones were able to traverse the environment whilst the smaller ones (evolved on the smooth terrain) became stuck in a trough. Thus far, the bodied robots had remained still.
On the rough terrain the rolling behaviour became unproductive. All six balls immediately rolled into a basin and remained there. For the others, it appeared that the tail helped the robots 'push' themselves up steep slopes and the twitching legs provided sudden boosts in velocity. It is possible that more productive legs would have greatly improved the performance here by pushing them further than the (very) short distance they achieved.

7 Conclusion

The goal of this project was to produce a nature-inspired model of a robot that could navigate rough terrain. The majority of the inspiration was reptilian, supported by research indicating the efficient and adaptable motion of snakes and lizards. The modular body, counterbalancing tail and lizardlike gait had already been proven to be effective by works such as the Agama robot and Salamandra Robotica.

The results of Lizardbot corroborated the efficacy of serpentine motion. However, the data suggested that the movement of the tail and legs were counterproductive to the success of the robots. Seemingly, the snakelike design had been implemented more effectively than the lizard-inspired design aspects.

A genetic algorithm was used to mimic a biological evolutionary process. The GA was found to be limited regarding which terrains it could find a suitable solution for. The goal was optimisation for navigating rough terrain; the outcome was a GA that can optimise for moderate terrain. However, the reasons behind this could be overcome given more time. The parameters used were targeted at the uneven terrain - which produced phenomenal results - and as such a GA tailored to the harsher environment may yield a higher performance. Ideally, a single GA would work across any terrain.
Lizardbot could further be improved by exploring the option of a flexible tail (see 6.2) or by applying the algorithm to a larger population to reduce the formation of localised breeding groups. Additionally, the introduction of a visual system (see 2.4) could provide proactive collision avoidance to complement the collision penalisation in the performance metric.

Dynamic movement encouraged the robots to manouevre away from their spawn point. This feature was proved to improve the performance of the population at the cost of a convergence on a single solution. Whilst effective, this may have restricted Lizardbot from finding more abstract solutions.

The evolution of the robots produced unexpected results that deviated from the natural evolution of lizards. The optimal solution was found to be a single ball rolling with speed across the terrain. Whilst the basis of the project was to produce a nature-inspired solution, an effort was deliberately made to not enforce or bias towards a naturalistic robot. Evolution does not always offer an efficient or elegant solution; the laryngeal nerve in giraffes is a prime example of this unintelligent design. This nerve connects the brain to the larynx in most mammals but diverts under the heart instead of taking the direct path through the head and neck. In giraffes this results in a highly inefficient 5m laryngeal nerve [Wedel, 2011]. Arguably, the emergence of such non-natural robots suggests the successful avoidance of bias in Lizardbot. Perhaps in thousands of years lizards too will have evolved into a rolling sphere.

8 References

Primary References

Alexander, R. M. (2012). Locomotion of reptiles. *Herpetological Bulletin*.
https://www.thebhs.org/publications/the-herpetological-bulletin/issue-number-121-autumn-2012/3-1-locomotion-in-reptiles/file
Accessed: 2022-04-13.

Belliure, J., Fresnilli, B., and Cuervo, J. (2018). Male mate choice based on female coloration in a lizard: the role of a juvenile trait. *Behavioural Ecology*.
https://academic.oup.com/beheco/article/29/3/543/4839801?login=false
Accessed: 2022-04-15.

Bhatia, J. S., Jackson, H., Tian, Y., Xu, J., and Matusik, W. (2021). Evolution gym: A large-scale benchmark for evolving soft robots. *NeurIPS*.
https://openreview.net/pdf?id=lM2971LAwV
Accessed: 2022-04-15.

Burrows, M., Cullen, D., Dorosenko, M., and Sutton, G. (2015). Mantises exchange angular momentum between three rotating body parts to jump precisely to targets. *Current Biology*.
https://doi.org/10.1016/j.cub.2015.01.054
Accessed: 2022-04-12.

Chopra, P. (2019). Reinforcement learning without gradients: evolving agents using genetic algorithms. *Towards Data Science*.
https://towardsdatascience.com/reinforcement-learning-without-gradients-evolving-agents-using-g
Accessed: 2022-04-15.

Cianchetti, M. (2015). Bioinspired locomotion and grasping in water: the soft eight-arm octopus robot. *Bioinspiration & Biomimetics*.
https://iopscience.iop.org/article/10.1088/1748-3190/10/3/035003
Accessed: 2022-04-10.

Clark, A. (1997). Being there: Putting brain, body and world together again. *MIT Press*.

Clark, A. (2001). Mindware: An introduction to the philosophy of cognitive science. *Oxford University Press*, pages 120–139.

Crespi, A., Karakasiliotis, K., Guignard, A., and Ijspeert, A. J. (2013). Salamandra robotica ii: An amphibious robot to study salamander-like swimming and walking gaits. *IEEE*.
https://ieeexplore.ieee.org/document/6416074
Accessed: 2022-04-12.

Dear, T., Buchanon, B., and Abrajan-Guerrero, R. (2020). Locomotion of a multi-link non-holonomic snake robot with passive joints. *The International Journal of Robotics Research*.
https://journals.sagepub.com/doi/10.1177/0278364919898503
Accessed: 2022-04-11.

Diamond, J. (1988). Why cats have nine lives. *nature*.
https://www.nature.com/articles/332586a0
Accessed: 2022-04-12.

Erlhagen, W. and Schöner, G. (2002). Dynamic field theory of movement preparation. *Psychological Review*.
https://www.researchgate.net/publication/11287764_Dynamic_field_theory_of_movement_preparation
Accessed: 2022-04-13.

Herbert, S. (1996). The sciences of the artificial. *MIT Press*, page 51.
https://monoskop.org/images/9/9c/Simon_Herbert_A_The_Sciences_of_the_Artificial_3rd_ed.pdf
Accessed: 2022-04-11.

Kelasidi, E. and Tzes, A. (2012). Serpentine motion control of snake robots for curvature and heading based trajectory - parameterization. *IEEE*.
https://ieeexplore.ieee.org/document/6265693
Accessed: 2022-04-13.

Kim, J., Kim, H., Kim, Y., Park, J., Seo, T., Kim, H. S., and Kim, J. (2019). A new lizard-inspired robot with s-shaped lateral body motions. *IEEE*.
https://ieeexplore.ieee.org/document/8902059
Accessed: 2022-04-13.

Lappin, A. K. and Husak, J. (2005). Weapon performance, not size, determines mating success and potential reproductive output in the collared lizard (crotaphytus collaris). *The American Naturalist*.
https://www.journals.uchicago.edu/doi/10.1086/432564
Accessed: 2022-04-14.

Libby, T., Moore, T., Chang-Siu, E., Li, D., Cohen, D., Jusufi, A., and Full, R. (2012). Tail-assisted pitch control in lizards, robots and dinosaurs. *nature*.
https://www.nature.com/articles/nature10710
Accessed: 2022-04-13.

Marder, E. and Bucher, D. (2001). Central pattern generators and the control of rhythmic movements. *Current Biology*.
https://pubmed.ncbi.nlm.nih.gov/11728329/
Accessed: 2022-04-13.

Mitchell, M. (1998). An introduction to genetic algorithms. *MIT Press*, pages 4, 8, 116.

Plataforma SINC (2008). Self-steering vehicle designed to mimic movements of ants. *ScienceDaily*.
https://www.sciencedaily.com/releases/2008/09/080917074130.htm
Accessed: 2022-04-12.

Roberts, M. (2020). How to evenly distribute points on a sphere more effectively than the canonical fibonacci lattice. *Extreme Learning*.
http://extremelearning.com.au/how-to-evenly-distribute-points-on-a-sphere-more-effectively-than
Accessed: 2022-04-12.

Schiff, W. (1965). Perception of impending collision: A study of visually directed avoidant behavior. *Psychological Monographs*.
https://doi.apa.org/doiLanding?doi=10.1037%2Fh0093887
Accessed: 2022-04-13.

Thelan, E. and Smith, L. (1994). A dynamic systems approach to the development of cognition and action. *MIT Press*, pages 263–266.

UC Berkeley (2012). Uc berkeley leaping lizard & robot. *YouTube*.
https://www.youtube.com/watch?v=jOUAIRbrv6s
Accessed: 2022-04-12.

Wang, L., Schmidt, B., and Nee, A. (2013). Vision-guided active collision avoidance for human-robot collaborations. *Manufacturing Letters*.
https://www.sciencedirect.com/science/article/pii/S2213846313000023
Accessed: 2022-05-07.

Wedel, M. (2011). A monument of inefficiency: The presumed course of the recurrent laryngeal nerve in sauropod dinosaurs. *Acta Palaeontologica Polonica*.
https://doi.org/10.4202/app.2011.0019
Accessed: 2022-04-19.

Wiernasz, D., Perroni, C., and Cole, B. (2004). Polyandry and fitness in the western harvester ant, pogonomyrmex occidentalis. *Molecular Ecology*.
https://onlinelibrary.wiley.com/doi/10.1111/j.1365-294X.2004.02153.x
Accessed: 2022-04-15.

Yang, X.-S. (2021). Genetic algorithms. *Nature-Inspired Optimization Algorithms*.
https://www.sciencedirect.com/topics/engineering/genetic-algorithm
Accessed: 2022-04-15.

Ye, X., Niu, Y., Wang, H., and Meng, T. (2010). Locomotion control for a modular snake robot over rough terrain. *IEEE*.
https://ieeexplore.ieee.org/document/5567368
Accessed: 2022-04-13.

Software References

Bracket, P. (2022). Texmaker latex editor 5.1.2. *TexMaker*.
https://www.xm1math.net/texmaker/
Accessed: 2022-04-12.

Desmos (2022). Graphing calculator - desmos. *Desmos*.
https://www.desmos.com/calculator
Accessed: 2022-04-12.

einarro (2020). Texcount web service (version 3.2.0.41). *TeXcount*.
https://app.uio.no/ifi/texcount/online.php
Accessed: 2022-04-12.

GitHub (2022). Github. *GitHub, Inc.*
https://github.com/
Accessed: 2022-04-12.

Google (2022). Colaboratory. *Google*.
https://colab.research.google.com/?utm_source=scs-index
Accessed: 2022-04-12.

jgraph (2022). drawio-desktop. *GitHub*.
https://github.com/jgraph/drawio-desktop/releases/tag/v17.4.2
Accessed: 2022-04-12.

matplotlib (2021). Matplotlib 3.5.1. *matplotlib*.
https://matplotlib.org/stable/index.html
Accessed: 2022-04-12.

NumPy (2021). Numpy 1.22.0. *NumPy*.
https://numpy.org/
Accessed: 2022-04-12.

NWH Coding (2017). Grapher - graph, replay, log. *Unity Asset Store*.
https://assetstore.unity.com/packages/tools/utilities/
grapher-graph-replay-log-84823#description
Accessed: 2022-04-12.

pandas (2022). Pandas 1.4.2. *pandas*.
https://pandas.pydata.org/
Accessed: 2022-04-12.

Seeburger, P. (2021). Calcplot3d. *LibreTexts*.
https://c3d.libretexts.org/CalcPlot3D/index.html
Accessed: 2022-04-11.

Syomus (2021). Proceduraltoolkit. *GitHub*.
https://github.com/Syomus/ProceduralToolkit
Accessed: 2022-04-10.

Unity (2020). Textmeshpro 3.0.6. *Unity Technologies*.
https://docs.unity3d.com/Packages/com.unity.textmeshpro@3.0/manual/index.html
Accessed: 2022-04-12.

Unity (2021a). Code coverage 1.1.1. *Unity Technologies*.
https://docs.unity3d.com/Packages/com.unity.testtools.codecoverage@1.1/manual/index.html
Accessed: 2022-04-12.

Unity (2021b). Recorder 2.5.7. *Unity Technologies*.
https://docs.unity3d.com/Packages/com.unity.recorder@2.5/manual/index.html
Accessed: 2022-04-12.

Unity (2021c). Unity 2021.1.10. *Unity Technologies*.
https://unity3d.com/unity/whats-new/2021.1.10
Accessed: 2022-04-12.

Unity (2022). Unity test framework 1.1.31. *Unity Technologies*.
https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html
Accessed: 2022-04-12.

Visual Studio (2021). Visual studio community 2019 version 16.10. *Microsoft*.
https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes-v16.10
Accessed: 2022-04-12.

Yughues (2021). Yughues free metal materials. *Unity Asset Store*.
https://assetstore.unity.com/packages/2d/textures-materials/metals/yughues-free-metal-materials-12949#description
Accessed: 2022-04-12.

Code References

aldonaletto (2013). Rotate a vector around a certain point. *Unity Forums*.
https://answers.unity.com/questions/532297/rotate-a-vector-around-a-certain-point.html
Accessed: 2022-04-12.

Arycama (2015). Random.range always generates the exact same numbers in start(). *Unity Forums*.
https://answers.unity.com/questions/1072318/randomrange-always-generates-the-exact-same-number.html
Accessed: 2022-04-12.

Brackeys (2017a). Generating terrain in unity - procedural generation tutorial. *YouTube*.
https://www.youtube.com/watch?v=vFvwyu_ZKfU
Accessed: 2022-04-12.

Brackeys (2017b). Start menu in unity. *YouTube*.
https://www.youtube.com/watch?v=zc8ac_qUXQY
Accessed: 2022-04-12.

Çetin, S. T. (2015). What is the proper way to handle data between scenes? *Game Development*.
https://gamedev.stackexchange.com/questions/110958/what-is-the-proper-way-to-handle-data-betwee
Accessed: 2022-04-12.

cjddmut (2015). Colorhsv. *GitHub*.
https://gist.github.com/cjddmut/fefe5dac35cccfceabec
Accessed: 2022-04-12.

damien_oconnell (2009). Click+drag camera movement. *Unity Forums*.
https://forum.unity.com/threads/click-drag-camera-movement.39513/
Accessed: 2022-04-12.

David K (2018). Plot points around circumference of circle in 3d space given 3 points. *StackExchange*.
https://math.stackexchange.com/questions/3007243/plot-points-around-circumference-of-circle-in-
Accessed: 2022-04-12.

DitzelGames (2018). Fixed, spring, hinge, character & configurable joint explained - unity tutorial. *YouTube*.
https://www.youtube.com/watch?v=MElbAwhMvTc
Accessed: 2022-04-12.

Draco18s no longer trusts SE (2018). unity - how two objects with rigidbody can pass through each other? *StackOverflow*.
https://stackoverflow.com/questions/48461267/unity-how-two-objects-with-rigidbody-can-pass-thro
Accessed: 2022-04-12.

FK22 (2017). Write data from list to csv file. *Unity Forums*.
https://forum.unity.com/threads/write-data-from-list-to-csv-file.643561/
Accessed: 2022-04-12.

Fnord (2014). Evenly distributing n points on a sphere. *StackOverflow*.
https://stackoverflow.com/questions/9600801/evenly-distributing-n-points-on-a-sphere
Accessed: 2022-04-12.

Goodacre, A. (2016). How to change object's layer at runtime in unity? *StackOverflow*.
https://stackoverflow.com/questions/40869566/how-to-change-objects-layer-at-runtime-in-unity
Accessed: 2022-04-12.

Halvarsson, K. (2021). Unity3d check if a point is to the left or right of a vector. *StackOverflow*.
https://stackoverflow.com/questions/65794490/unity3d-check-if-a-point-is-to-the-left-or-right-o
Accessed: 2022-04-12.

Microsoft (2022). Streamreader class. *Microsoft Documentation*.
https://docs.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=net-6.0
Accessed: 2022-04-14.

Mörk, F. (2011). How to list all variables of class. *StackOverflow*.
https://stackoverflow.com/questions/6536163/how-to-list-all-variables-of-class
Accessed: 2022-04-12.

mqp (2009). Identify if a string is a number. *StackOverflow*.
https://stackoverflow.com/questions/894263/identify-if-a-string-is-a-number
Accessed: 2022-04-14.

nobizzle (2019). Update() vs. fixedupdate() for object rotation. *Unity Forums*.
https://answers.unity.com/questions/1677930/update-vs-fixedupdate-for-object-rotation.html
Accessed: 2022-04-12.

Rafay, K. (2022). Angular momentum calculator. *Omni Calculator*.
https://www.omnicalculator.com/physics/angular-momentum
Accessed: 2022-04-11.

Serlite (2016). Getting collision contact force. *StackOverflow*.
https://stackoverflow.com/questions/36387753/getting-collision-contact-force
Accessed: 2022-04-12.

slandau (2011). Efficient way to remove all whitespace from string? *StackOverflow*.
https://stackoverflow.com/questions/6219454/efficient-way-to-remove-all-whitespace-from-string
Accessed: 2022-04-12.

superpig (2012). How to save a material at run time? *Unity Forums*.
https://forum.unity.com/threads/how-to-save-a-material-at-run-time.138318/
Accessed: 2022-04-12.

tomvds (2008). Euler, quaternions, radians, degrees... huh? *Unity Forums*.
https://forum.unity.com/threads/euler-quaternions-radians-degrees-huh.53407/
Accessed: 2022-04-11.

Unity (2018). Introduction to unity physics. *Unity Documentation*.
https://docs.unity3d.com/Packages/com.unity.physics@0.0/manual/index.html
Accessed: 2022-05-07.

Unity (2019). Joints. *Unity Documentation*.
https://docs.unity3d.com/Manual/Joints.html
Accessed: 2022-04-12.

Unity (2022a). Prefab utility. *Unity Documentation*.
https://docs.unity3d.com/ScriptReference/PrefabUtility.html
Accessed: 2022-04-12.

Unity (2022b). Unity documentation. *Unity Technologies*.
https://docs.unity.com/
Accessed: 2022-04-12.

user91669 (2017). Change the length of footnote line? *StackExchange*.
https://tex.stackexchange.com/questions/405195/change-the-length-of-footnote-line
Accessed: 2022-04-16.

Vladimir T (2020). Luna tech series: A deep dive into unity configurable joints. *Luna Labs*.
https://medium.com/luna-labs-ltd/luna-tech-series-a-deep-dive-into-unity-configurable-joints-96
Accessed: 2022-04-11.

9 Appendices

9.1 Progress Logs

The status of Lizardbot was reported as the project progressed. The progress logs for Lizardbot can be found here:
`https://github.com/jacobgeorge26/lizardbot/pull/10`

The details of the discussions with the project supervisor, Simon Bowes, can be found here:
`https://github.com/jacobgeorge26/lizardbot/pull/9`

9.2 Project Proposal

The initial project proposal can be found here:
`https://docs.google.com/document/d/17SvBZN9ctZutAxGG5i6fcNnbd7-2hrrye14ARTxWUa4/edit?usp=sharing`