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

"Lizardbot"

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

Abstract
Insert abstract here

Submission Year - 2022
Candidate Number - 198732
Project Supervisor - Simon Bowes

Contents

1 Introduction

Add in the intro pretty much directly from the interim report here

2 Project Aims

Why is the robot being modelled instead of physically built?
Why did I choose to use Unity?

2.1 Primary Objectives

2.1.1 Robot Design

Get from interim report - explain overall design and why those decisions were made e.g. simplistic design

2.1.2 Robot Movement

Basic overview of why each component will move the way it does. Tie each point back to how they are founded (or not founded) in natural algorithms.
Include jumping here

2.1.3 Terrain Generation

The terrain will be static - why?
Why will I have three separate terrains? - Octopus
What is the importance of having a smooth terrain?

2.1.4 AI

How will the AI work? Genetic algorithm outline
Dynamic systems theory
Damage minimisation
How do I want the AI to manipulate the relationship between the body and movement?
Why do I want there to be a relationship between the two? - article Simon sent
How am I going to test the relationship?
How will the robot be measured?

2.2 Extension Objectives

2.2.1 Vision

How would a rudimentary visual system reduce damage to the robot?
How would this move the AI from a reactive to proactive mechanism?

2.2.2 Terrain Friction

How do snakes work with different frictions?

2.2.3 UI

How could a UI help lower the threshold to the project and make it easier to 'work with' the AI?

2.2.4 Flexible Tail

What are the advantages of having a flexible tail?

3 Project Relevance

3.1 Salamandra Robotica II

Insert from interim report

3.2 Agama Robot

Insert from interim report

3.3 tbc

Find a team that have modelled a robot vs building one

4 Requirements Analysis

Insert from interim report - needs some work
Add section on the constraints of this project

5 Professional and Ethical Considerations

Insert from interim report with more reference to code of conduct

6 Implementation

6.1 Terrain

Three terrains were generated using Procedural Toolkit [Syomus, 2021] to test the performance of the robot across various environments: rough, uneven, and smooth. These categories were inspired by the those used by the Octopus robot [Cianchetti, 2015].

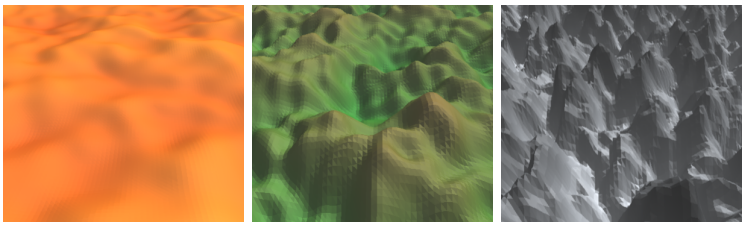

Figure 1: Examples of the three terrain types. From left to right: Smooth, Uneven, Rough.

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. 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.
The smooth terrain is deliberately featureless to test the behaviour of the robot in a simple environment. 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. It is worth noting that the robot is still being tested on three terrains with some common properties (e.g. gravity) and these factors may introduce bias in the AI. This is a reasonable situation as long as applications of the Lizardbot are further modelled on encounterable terrains to allow the AI to adapt the robot accordingly. For proof of concept the sample set of terrains is sufficient.

6.2 Body

To create a snakelike body, each body module is attached to the previous module by a configurable joint [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 movement of one section is translated to those behind it - similar to dragging a piece of string along the ground. This is shown in figure x: the head rotates and, after a delay, creates the same angle in the sections behind it.
The former applies a forward force as determined by the drive velocity parameter whilst the latter applies a velocity to each module using the following equation:

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

For rotating sections $i = 0, ..., m$, where $m \leq n$, the value of w will be calculated using S or C as specified.

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

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

This central pattern generator (CPG) approach allows each module to react to the velocity and angle of the previous section. The equation for the CPG originated in Tony Dear's multi-link snake robot: a robot with a similar modular design with passive joints connecting the modules. [Dear et al., 2020] Lizardbot utilises the same math to calculate the velocity 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 and cos to produce the serpentine motion. For robots with serpentine motion disabled, S and C will be assigned randomly to the rotating modules.

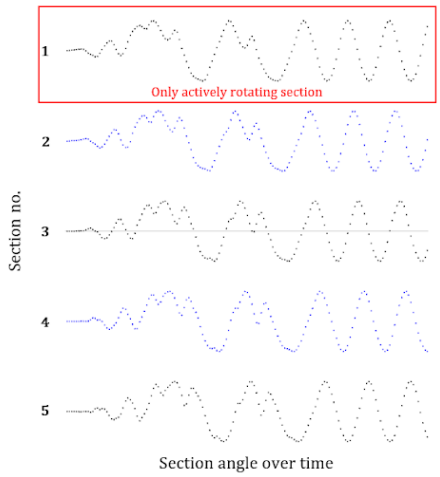

Figure 2: Demonstration of body motion with a single rotating section at the head of the body.

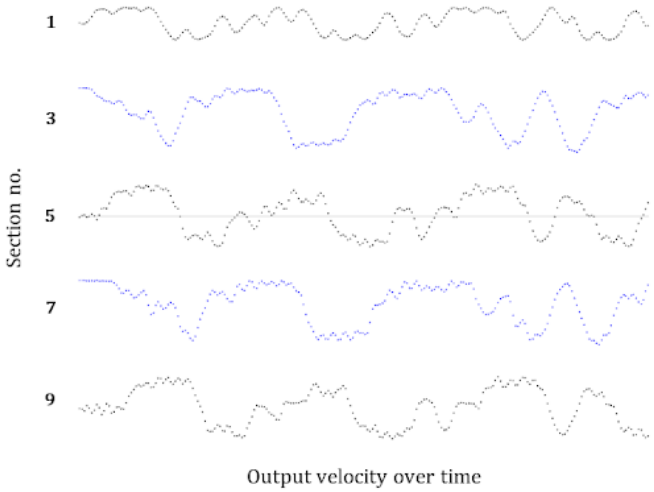

Figure 3: The output velocities generated by each body module with serpentine motion enabled.

As each section 'reacts' to the previous one whilst using the opposing equation, the velocity applied 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. It is simply reacting to the body that came before it.

6.3 Tail

The use of a tail has the potential to counterbalance the body and provide stability as the lizardbot 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.

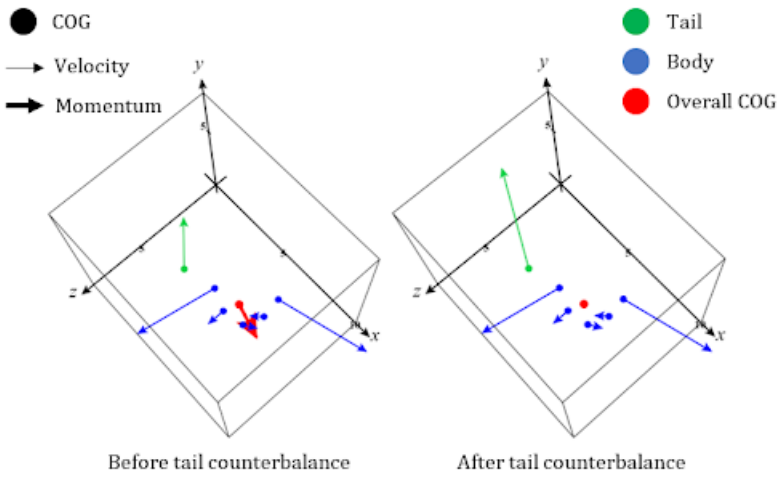

Figure 4: A representation of the tail being adjusted to conserve the angular momentum of a robot. Diagram created using CalcPlot3D [Seeburger, 2021]

For all body parts $i = 0, ..., n$, the radius of the path of motion 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 counterproductive for a feature whose goal was to stabilise the robot. Additionally, by calculating the magnitude around 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 and the location of the tail maintains this property, alongside keeping the motion of the tail in the same plane as the rest of the body. For the lizardbot, this assumption could be removed in the future. Since symmetry is not a required property for non-uniform bodies, the tail could be placed anywhere that has equal mass either side of the tail - or placed randomly to see what effect this has on the robot. Who am I to say that a tail cannot be located on the head?

6.4 Legs

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

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

6.5 Performance

Insert from trapped algorithm log - get performance from GA

6.6 Genetic Algorithm

Insert from GA log

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

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

where R denotes a randomly generated number. 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.

6.7 Dynamic Systems

Insert from DST log

7 Results

How does mutating the body / movement independently work?
How does the addition of the tail help?
How does initiating the body with a serpentine motion established affect the outcome?
What happens when the body is set as static?
What is the outcome when the legs are out of sync from the body?
Starting with defaults, what parameters does the AI mutate to? What is the corresponding behaviour for this?
Does the AI converge on the same parameters when started with different defaults?

8 Conclusion

9 References

Primary References

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.

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*.
http://dx.doi.org/10.1177/0278364919898503
Accessed: 2022-04-11.

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.

Software References

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.

Code References

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

T, V. (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.

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

10 Appendices

10.1 Code of Conduct