Mobilisation of an Octopod Robot and Optimisation using an Evolutionary Algorithm

By Joshua Graham Bowen Foulkes BSc

A Thesis presented for the Master's degree of Robotics

Department of Computer Science

University of Birmingham

United Kingdom

12th September 2022

Acknowledgements

I would like to thank the university and its lecturers for the help given such as the knowledge to be able to undertake this thesis. I would like to thank my project supervisors for the help given and to keep me on track. I would like to thank my mum and dad for the moral support and the endless cups of teas to get me through this.

Table of Contents

Abstract

This thesis aims to explore the design and creation of a walk with the goal of the mobilisation of an octopod robot, previously designed for my BSc. The algorithms used by the walk are also examined and formalised. The introduction of an evolutionary algorithm for optimisation of the walk's characteristics, without hindering other factors of the movement, are also included. Previous papers are researched for improved walk hyperparameters such as an altered gait, called the tripod gait, and their effects. The eventual introduced algorithm affected the outcome on the simulation by improving the speed by over 260 per cent with only a minor effect on stability. The combination of the tripod gait and training from the evolutionary algorithm improved the maximum speed by over 450 per cent.

Keywords: octopod, evolutionary algorithm, machine learning, gait, robot, walk, Webots, hyperparameter optimisation, mobilisation

List of Abbreviations

The list below describes the relevant significance and meaning of the acronyms and abbreviations used throughout the thesis.

Abbreviation	Meanings
CAD	Computer-Aided Design
EA	Evolutionary Algorithm
FK	Forward Kinematics
GUI	Graphical user interface
IK	Inverse Kinematics
NDT	Non-Destructive Testing
RL	Reinforcement Learning
RRR	Rotation Rotation Rotation
STEVE	Specific Terrain Evaluation Verification Equipment
URDF	Unified Robot Description Format

Introduction

For my dissertation I created a robot spider named STEVE - Specific Terrain Evaluation Verification Equipment - with the intention it is used in search and rescue operations to access difficult locations. The spider had eight legs each driven by three servos. This spider had a built-in camera, the robot could be operated remotely via Bluetooth and had a GUI for controlling and monitoring the robot and sensors. However, a functionality that wasn't achieved was to make it walk.

This was in part due to lack of expertise and the lack of simulation software; this means there was no safe environment for testing the walk. I didn't want to break the servos as this would ensue a large cost due to the amount of trials needed. Therefore, I was hesitant in testing and developing a walk. I had no Non-Destructive Testing (NDT) possibility.

The aim of this project is to create a simulated environment and recreate the spider robot in a simulated environment. Then the goal is to succeed in getting it to walk using "manual" methods as well as to see if the walk can be optimised, to improve aspects such as the robot's speed. These outcomes of both methods will then be compared to observe if any improvements have been made.

The difficulty of this project will be balancing how much of the walk will be the initial basic walk and how much control the machine learning will have. Too little control and it won't be effective, too much and the training will take too long and may not converge at all.

This thesis details the original robot's design to give background knowledge. The simulation setup will also be covered which involves how the robot was transferred to a simulated environment and setting up the controller which allows for the software to control the robot. Following sections detail the method used for the initial walk. The final section of the methodology will cover the machine learning, which method was chosen and why. How the method was implemented is also included and the outcomes of this learning. The final section will be a discussion and comparison of the effects of the machine learning on the proficiency of the robot and how it compares to the initial walk. There also will be a comparison to the outcomes from other related papers using alternate hyperparameters.

Further Background Material

This section will cover the background material needed to contextualise the thesis, its task, and the need for it to be achieved.

➢ Main Inspiration

The main inspiration for the project was from a video [1] of the results of a paper [2] and where demonstrated, were a few bipedal agents walking using a muscle system. It was even more interesting seeing these agents being able to adapt and walk over uneven terrain and adapt to external perturbations.

➢ STEVE Robot Specification

The spider robot, nicknamed STEVE, was created as part of my BSc dissertation. It was designed as a cost-effective mobile robot for the purpose of navigating terrain with the intent for search and rescue tasks and their environments [3].

STEVE is a robot with eight legs, each leg having three servos, the first servo controlling horizontal rotational movement at the hip and the two other servos control the vertical movement at the hip and at the knee, each leg has a button on the base of the foot used to measure if there is contact with the ground. The robot also includes a camera that was streamed to a computer where a GUI was created. This GUI showed sensor readouts such as the battery percentage.

The whole system runs using an integrated circuit (IC) which controls the legs, manages the Bluetooth communication, and carries out the main part of the logic.

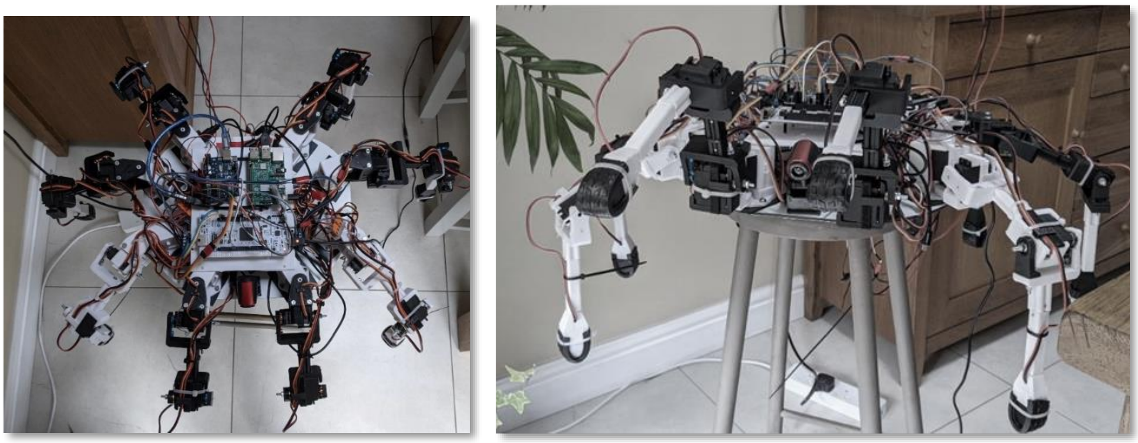


Figure 1: Top view and front view of STEVE

The spider robot needs a massive update to integrate the use of ROS, and therefore the algorithms used in this thesis will **not** be implemented on the robot in this thesis.

Compared to the original paper the legs have been changed slightly to have more responsive buttons on the base of the feet. Additional sensors such as angle feedback and the gyro were added to the simulation which are not present on the real robot.

➢ Previous Works

There are many works covering the problem of getting robots to walk, most are focused upon hexapods rather than octopods. Some use machine learning to improve aspects such as stability.

1. *"Flexible Muscle-Based Locomotion for Bipedal Creatures"* [2]

This paper demonstrates a muscle-based control method for simulated bipeds. The paper has multiple different type of bipeds from ones seen in nature, such as humans, to some biped structures that are fabrications. The controller finds different gaits based upon target speed, terrain, and external perturbations.

This paper was the main inspiration for the thesis, it shows the use of optimisation for better biped motion with the ability for adaptation. It also demonstrates the power of simulation with the ability to run tests quickly and efficiently.

As mentioned before, the main takeaway was the inspiration. On a software side, the systems are very different as this system uses a muscle-based control method. The optimisation is also not using machine learning.

2. *“Fundamentals of Hexapod Robot”* [4]

This chapter of a project log covers the fundamental concepts of a hexapod robot. This includes the kinematic, stability and walking gaits.

Although this project involves a hexapod rather than an octopod, many of the fundamentals for the octopod are covered, such as the kinematics which uses a similar leg structure. Some concepts such as the walking gait can be easily adapted to eight legs by keeping these principles and extending them.

This project report was used to understand some of the basic principles of a legged robot. It also includes information that could be used later to improve the robot’s versatility for difficult terrain.

3. *“Hexapod Robot Static Stability Enhancement using Genetic algorithm”* [5]

The paper involves using a genetic algorithm to improve the stability of the gait of a hexapod robot. This paper covers two methods for improving stability; the first being where “*the range of stability margins is evaluated for all gaits*” and the second being the evolutionary algorithm to enhance “*the static stability by getting the best stability margins for* [a] *hexapod robot*”. The forward and inverse kinematics for the hexapod are also derived.

This paper is another source for the kinematics but also marries the idea of using an evolutionary algorithm to improve a certain aspect of the hexapod’s gait. The main difference being the improvement of the stability rather than a combination of speed and stability. Due to this difference, the fitness function used cannot be implemented directly.

The conclusion of the paper shows that after training, the tripod gait had the lowest error value and therefore best stability. This is interesting as it goes against some previous sources [4] which states that the wave gait is the most stable but also slowest.

4. *“An evolutionary approach to gait learning for four-legged robots”* [6]

In this paper a Sony AIBO robot, a four-legged robot is used. The gait parameters are trained via an evolutionary algorithm and optimised with the goal to have a greater forward velocity.

This paper relates the use of an evolutionary algorithm for parameter optimisation with the task of creating an effective gait for a legged robot.

This paper was used to find the parameters for the robot that need to be trained and some default values to use.

A very similar paper is [7] where the robot used again is the Sony AIBO robot. Here the goal is to tune a parametrised gait. This paper formulates the issue as a policy gradient reinforcement learning problem.

Analysis and Specification

As stated previously the main regret was - at Bachelor’s degree level - I was unable to get the spider robot walking. This was due to issues such as possible breakages in the servos.

With the use of simulation, these issues are negated as it is a safe environment for testing that also allows for changes to be verified quickly and efficiently. This allows for Non-Destructive Testing which is vital in the design and creation of robots and their algorithms.

The initial goal is for the robot to achieve a walk not only forward but in multiple directions in simulation. When the simulation is transferred to the actual robot this will allow it to move. Additional movements such as rotation may be added.

For the training the goal is to create an algorithm that will modify the walk via its hyperparameters to improve the speed of the movement without sacrificing too much stability and accuracy.

Improved hyperparameters are to be researched and applied to the robot. The original training algorithm can then be applied to this new walk to optimise it further.

Methodology: Design and Implementation

Design:

The initial method was to use a reinforcement learning algorithm. This would create a neural network of all: The inputs to the network being the robot's sensors and the outputs being the values of the joints in each leg. This network would be applied to the simulated robot which would be trained by running the simulation and be given a fitness depending upon its performance. This would inform the neural network where changes would need to be applied to the network to improve performance. This idea was proposed to my project supervisor who massively discouraged it, as it would take a lot of time. A new method was then proposed by my supervisor.

The new method is to have the robot do a slow and a stable walk using "manual" methods. The hyperparameters of this walk would be trained using machine learning to make the walk faster and more efficient.

Software

All of this project will be done in simulation, so choosing the correct software is paramount. Before this project I was using ROS for a previous project which required some simulation, this led me to using Gazebo. I therefore chose to use Gazebo for this project. The robot was recreated and loaded into Gazebo where a main issue became apparent, the robot started to slightly move on its own. Due to being unable to fix this issue and some other issues such as ease of use, I chose to discard the idea of using Gazebo.

After carrying out some research I found a new piece of software, Webots. It accomplishes what I needed, is easy to use, has good documentation, the code runs directly from my computer but interacts with the software easily. It has a slew of sensors and actuators, worked with ROS(if needed at a later date) and there was already a library for reinforcement learning.

Setting up the Simulation

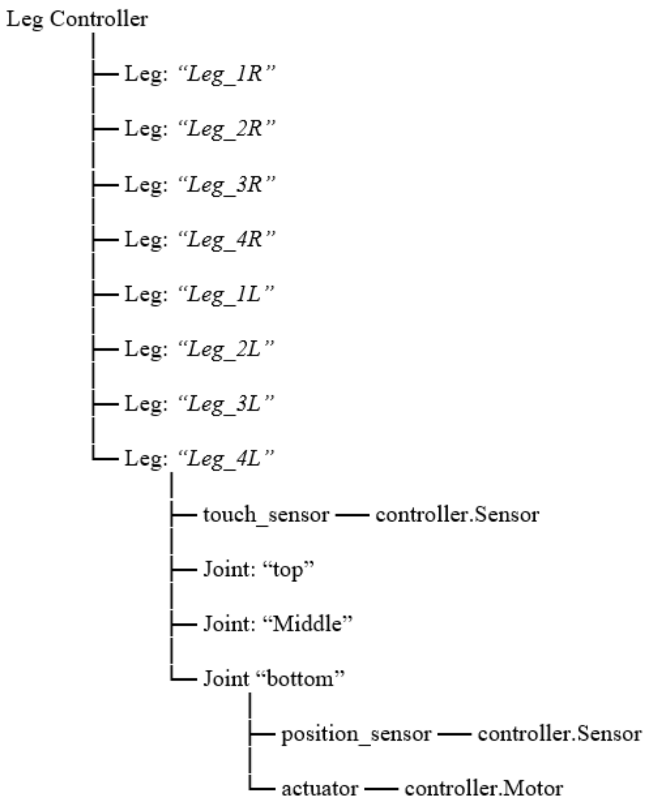
The first step was to setup the software so that the robot could be simulated. The software, Webots uses ".proto" files to represent agents such as robots. This means that STEVE needs to be represented via a proto file. I created a URDF (Unified Robot Description Format), of STEVE as this is the standard format to describe a robot for use with ROS and Gazebo.

The URDF of the leg was created using the legs dimensions and the STL files of the sections in the leg for a visual representation. The body of the robot was created in a similar manor. The collisions of each section of the leg were defined using simple shape geometry by either using a box, cylinder, or sphere, this simplifies the calculation of the collisions. For the body an additional section was added called the "others section" represented on the URDF by a box. This box is a simplification taken of the components and their weights. The URDF file gives an accurate representation of the robot including its mass and weight distribution which is represented using the inertia tensor. The inertia tensor informs the program how the mass is distributed in the part for accurate simulation. This tensor can be calculated using a program such as MeshLab. The URDF file also allows for the position and strength of servos to be accurately simulated, meaning if the walk can function in the simulation, it should be able to run on the physical robot; this is rarely the case. This is due to simplifications of the physics engine the simulation uses and some of the predictable unpredictability of the real world.

To convert from URDF to ".proto" a prebuilt library was used, "urdf2proto" [8]. This python library runs via command line and easily and accurately converted the URDF to a ".proto" file. There were some minor issues, such as to make the URDF file "nicer", macros were used, which are standard for URDF. The python library didn't recognise these macros, therefore they had to be removed, which resulted in a much longer file with lots of "ugly" repetition. Another issue was that the URDF uses STL files for the visual representation of each component, and when the CAD software exports these files, they are in mm but when they are imported into the software they are in meters. This issue was quickly and easily resolved by scaling the files by a factor of 0.001. During the creation of the proto file the additional sensors such as a gyro were added.

Controller Setup

The next stage is for the code to be able to move the robot. A class was created which contained each leg and within each leg, three servos - one for each joint – they were assigned their respective Webots controller function. This allows for the value of the servo to be set and read. Structure of the class follows the tree structure below:



```
Leg Controller
        |
        ├── Leg: "Leg_1R"
        |
        ├── Leg: "Leg_2R"
        |
        ├── Leg: "Leg_3R"
        |
        ├── Leg: "Leg_4R"
        |
        ├── Leg: "Leg_1L"
        |
        ├── Leg: "Leg_2L"
        |
        ├── Leg: "Leg_3L"
        |
        └── Leg: "Leg_4L"
                |
                ├── touch_sensor ── controller.Sensor
                |
                ├── Joint: "top"
                |
                ├── Joint: "Middle"
                |
                └── Joint "bottom"
                        |
                        ├── position_sensor ── controller.Sensor
                        |
                        └── actuator ── controller.Motor
```

Figure 2 Tree structure of controller class

Once all these issues were sorted, it meant STEVE was in simulation and ready for learning.

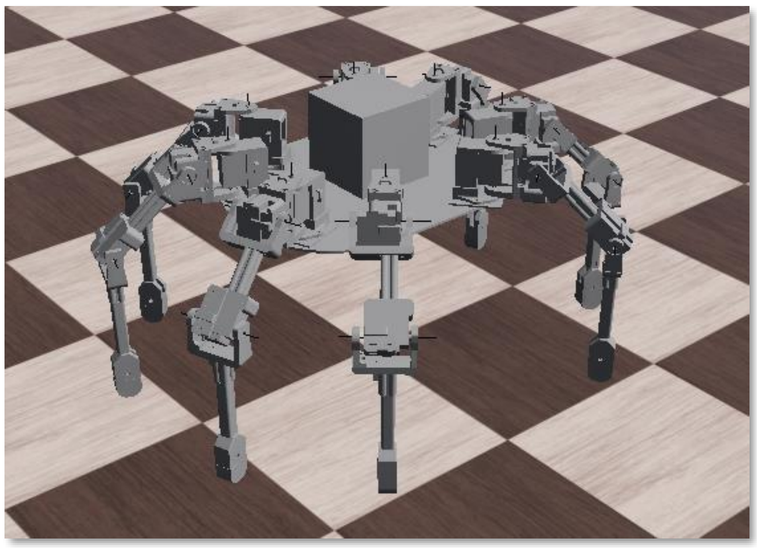


Figure 3 STEVE in simulation

The Basic Walk

The aim of this stage is to have the spider robot do a slow but stable walk. The parameters of this walk will then be passed onto a machine learning algorithm to optimise it by improving the walk speed. Other specifications might be added to improve the path that the robot walks.

➢ Additional Files

Some additional files and functions were needed that would be useful which had to be created. These were defined at the start of the file and made part of the control class. The functions that are defined are:

- The coordinate translator
- The forward kinematics of the leg
- The inverse kinematics of the leg

➢ Coordinate Translator

The coordinate translator takes a position given in terms of the robot base frame and translates it to a coordinate in terms of the desired leg's coordinate frame. The inputs are the desired coordinate to translate in terms of a 3x1 array, the array index of the leg (which is the same as the order in the controller class), the desired direction, meaning if the translation is from the body to the leg or the leg to the body. The index of the leg is required to get the position of the base frame and its rotation.

The equation below is used to translate coordinates from the body frame 0 to the leg frame 1 [9]:

$$p^1 = -{R_1^0}^T o_1^0 + {R_1^0}^T p^0 \tag{1}$$

Where p^0 is the input coordinates, o_1^0 is translation from the body to the leg, R_1^0 is the rotation from the body to the leg in radians and p^1 is the output coordinates.

The equation to translate coordinates from the leg frame to the body frame is below:

$$p^0 = o_1^0 + R_1^0 p^1 \tag{2}$$

➢ Forward Kinematics

The forward kinematic of the leg allows for the joint space to be mapped to the task space. This is to know the position and orientation of the robot's end effector given the angles of the joints.

The arm first needs to be converted into Denavit–Hartenberg parameters, which *"are the four parameters associated with a particular convention for attaching reference frames to the links of a spatial kinematic chain, or robot manipulator."* [10] This means the robot leg can be represented using four parameters, being alpha, theta, d and r (sometime also called a). These four parameters can then be used in the array equation below:

$$\begin{bmatrix} & & & \\ & R & & T \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta cos\alpha & sin\theta sin\alpha & rcos\theta \\ sin\theta & cos\theta sin\alpha & -cos\theta sin\alpha & rsin\theta \\ 0 & sin\alpha & cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The R 3x3 matrix is the rotation matrix and the T column is the x, y, and z coordinates of the end effector.

➢ Inverse Kinematics

Inverse kinematics maps from the task space to the joint space, meaning a set of input coordinates and rotation can be inputted and a set of joints will be returned which when applied to the robot will move it to the desired location. This process is more complicated than FK(forward kinematics) as IK(inverse kinematics) has many different possibilities that lead to the same point.

There are multiple different solutions for the inverse kinematics problem. One solution is to use the pseudo-inverse Jacobian matrix and to iterate the movement of the leg until the desired position is reached. This solution can adapt to different structures of legs.

A simpler solution which doesn't use the pseudo-inverse Jacobian or needs to iterate the movement of the leg is the one which uses standard equations but is fixed to the leg structure it is designed for.

The equations for inverse kinematics are gathered from [11] and are for the RRR leg structure below:

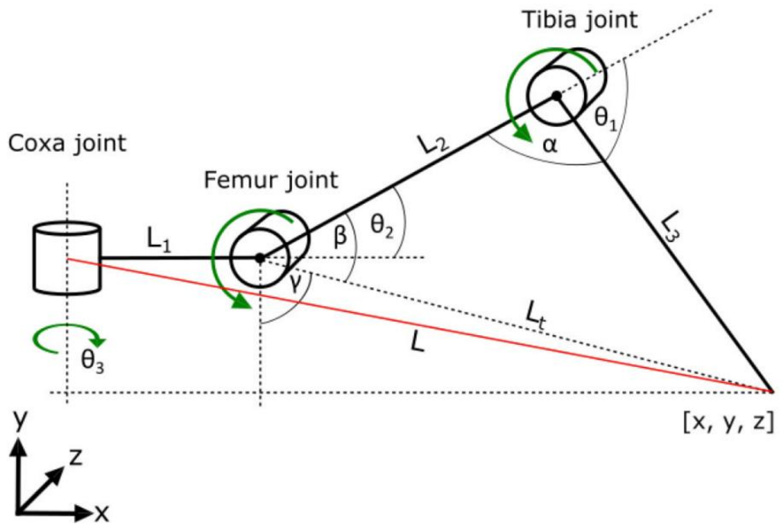


Figure 4: Structure of leg [11]

The leg is similar to that of the spider robot apart from the coordinate frame:

Reference Coordinates	**Spider Robot coordinates**
X	X
Y	Z
Z	Y

The modified equations for IK:

$$L = \sqrt{x^2 + y^2} \tag{4}$$

$$L_t = \sqrt{(L - L_1)^2 + z^2} \tag{5}$$

$$\gamma = \arctan\left(\frac{L - L_1}{z}\right) \tag{6}$$

$$\beta = \arccos\left(\frac{L_3^2 - L_2^2 - L_t^2}{-2L_2L_t}\right) \tag{7}$$

$$\alpha = \arccos\left(\frac{L_t^2 - L_2^2 - L_3^2}{-2L_2L_3}\right) \tag{8}$$

$$\theta_1 = 90 - \alpha \tag{9}$$

$$\theta_2 = 90 - (\gamma + \beta) \tag{10}$$

$$r\theta_3 = \arctan\left(\frac{y}{x}\right) \tag{11}$$

There is a possibility that the leg can't reach the desired coordinates. In that case it will return a value of 404 for each joint. This removes the issue of the software crashing due to a mathematical error created by an unreachable solution.

➢ Self-Collision Avoidance

One of the most important challenges that needed to be addressed is the possibility of legs colliding with each other when walking. This issue would hinder the movement forwards and could damage the legs and its servos.

To stop this, the lateral hip servos is limited to certain angles. Which means the legs will never collide.

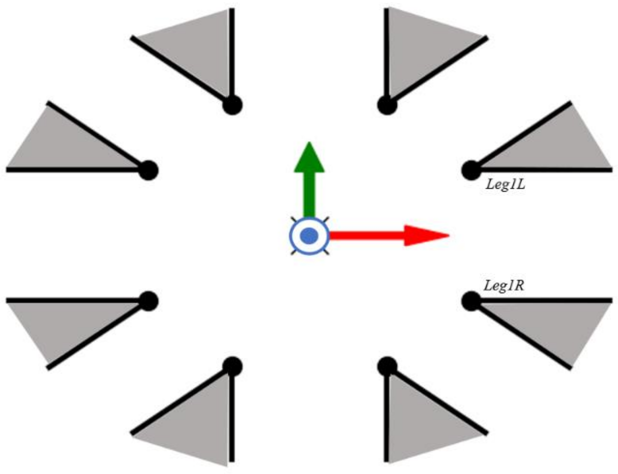


Figure 5: Restricted movement area of each leg

The area between the black line shown in grey on the image shows the task space of the leg, where the leg can move within.

Each angle that each leg can move is 45 degrees laterally at the hip.

This solution of limiting the angles ensures that the task space of each leg will not overlap.

This solution is simple but not very efficient as some area for the leg to move in is lost.

A different solution would be to know where each leg is and for the robot to adapt by modifying the gait. This solution avoids collision whilst unaffecting any movement and the stability of the robot, permitting the task space to overlap and for each leg a larger movement area.

Another solution would be to train this behaviour out of the robot by using machine learning, which it would do as part of the learning; due to the collision of legs would affect the distance travelled. It would therefore train itself to avoid this behaviour.

A preferable method, if there was more time for development to the previously mentioned to allow for the overlapping of joint space whilst avoiding self-collision, would be the following. The legs can be approximated into 2D rectangles from the top view, with the width W equal to the max width of the leg. Due to the length L of the leg not being fixed in can be calculated using the Pythagoras equation.

$$L = \sqrt{x^2 + y^2} \tag{12}$$

Where x and y are the x and y coordinates of the end effector, which are gathered using forward kinematics. The square would have its inner short edge focused on the origin of the leg with the outer short edge placed on the so that the rectangle is lined up with the leg.

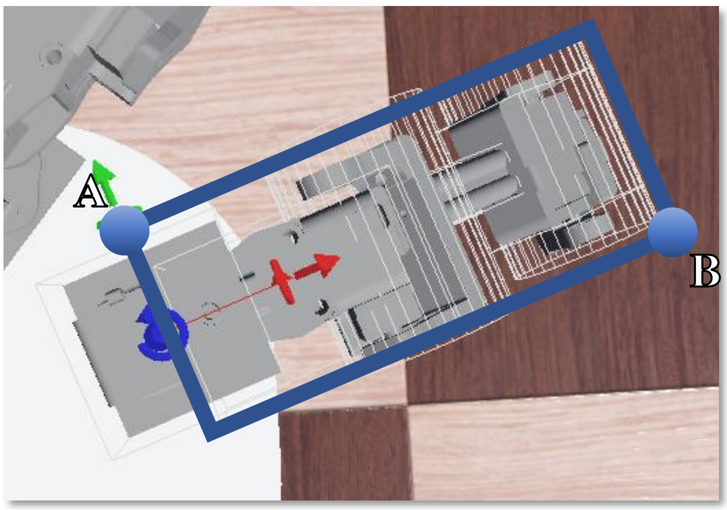


Figure 6: Simplified collision box of leg

Once the coordinates of the corners A and B are calculated, this can be done using simple trigonometry. This square can then be checked throughout the predicted movement of the leg for any overlap with any other squares. This is achieved using the pseudocode below, where A1 and B1 are the corners of square 1 and A2 and B2 are the corners of square 2 and the goal is to check if there is any overlap of both squares [12]:

```
1.     For each movement between start and stop point:
    1.1.  Calculate A1 and B1
    1.2.  Calculate A2 and B2
    1.3.  Overlap =0
    1.4.  If A1.x == B1.x or A1.y == B1.y or A2.x == B2.x or A2.y == B2.y:
        1.4.1. Overlap = 1
    1.5.  If A1.x > B2.x or A2.x > B1.x:
        1.5.1. Overlap = 1
    1.6.  If  B1.y > A2.y or B2.y > A1.y:
        1.6.1. Overlap = 1
```

The output of this will be if the leg is overlapping therefore the trajectory of the leg needs to be changed. This change could be a slight delay to the leg's gait to give one of the legs time to move or to recalculate the start point to avoid a collision.

➢ Forward Walk

The main principle of the forward walk is to get every foot to move in the same direction at the same speed. An offset also needs to be added to the cycle of each leg to ensure that every leg is off the ground at different times. This is called a gait. The gait needs to make sure that the centre of mass is always within the robot's support polygon. This is the area created by the feet touching the ground. This issue is more important with fewer legs but with eight legs this is very easily done.

The basic gait to use is the wave gait. Only one foot leaves the ground at one time. This is the gait that the robot will start off with. Another type of possible gait is the ripple gait. [4]

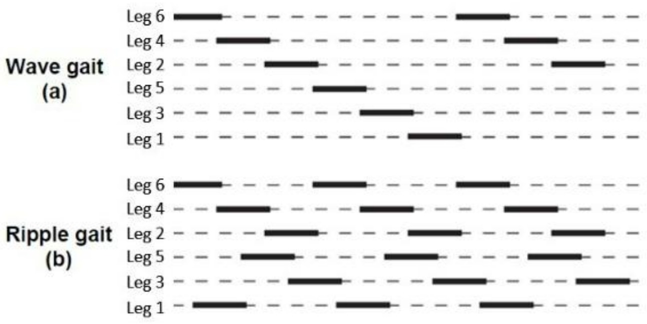


Figure 7: Wave and ripple gait [4]

The wave gait is used as it is the simplest to create and most stable gait [4]. It is also one of the slowest gaits.

The motion of each leg is broken up into two phases: The swing phase and the stance phase. The stance phase is when the foot is on the ground moving the robot. The swing phase is when the leg is in the air returning to the point at the start of the stance phase.

The basic motion will follow a half ellipsis pattern.

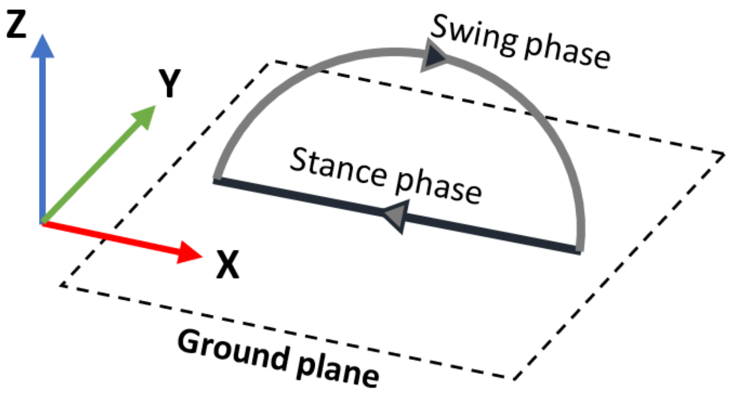


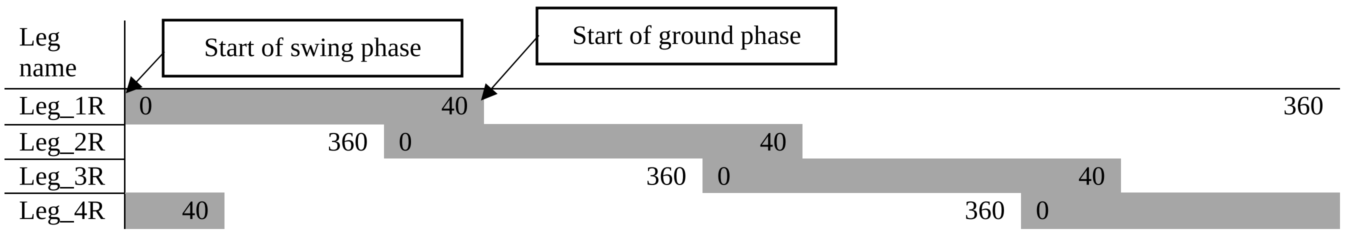
Figure 8: Breakdown of movement

An important part of this system is that it needs to be easily adaptable to different variables such as the walk height, the direction, the speed and maybe a custom gait.

This can be carried out simply by having all the legs push in the desired direction at the desired speed.

The first step is to create the cycle for the leg to follow the system which will cycle through the numbers 0 to 360 where 0 is the start of the swing phase, each leg will be swinging for 40 units, here shown in grey. They will be on the ground for 320 units.

A unit is the unspecified amount of time, the length of a unit will depend on the amount of time a cycle will take. This could be an issue if a more powerful computer is used as the loop in the code will have a shorter period. The system therefore needs to be limited so that the period of the code is a fixed amount of time.



The ratio between the swing phase isn't massively important, the numbers are estimates inspired from previous works [4] [13]. The machine learning will tweak these values to be more efficient.

This means at any point in time it is known whether the leg should be in the air or on the ground and how far it is along its cycle.

From this the total distance the leg is travelling when it is on the ground can be calculated by multiplying the desired movement speed by the number of units when it is in the stance phase.

Then a path needs to be created for the leg to follow. This is done by going from a start point and creating an end point, this being the travelled distance away at the angle of which the robot needs to move. The start and stop coordinates are then checked using the IK function. To do this the coordinates are checked whether the leg can move there and then checked if the legs will collied . If the points are valid, they are returned by the function. If one of the points isn't valid, then the points are moved along each of the edge boundaries of the leg until both points become valid.

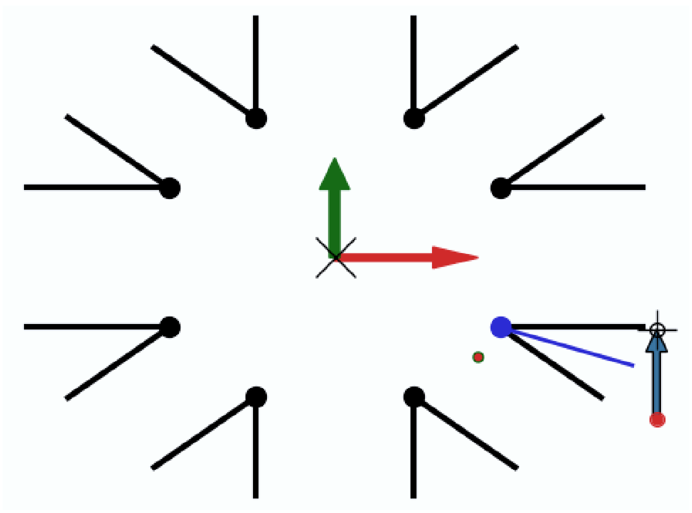


Figure 9: Diagram of start and stop coordinate function

The diagram shows the process used by the function in this case for leg 1, denoted using the blue dot. The smaller red dot represents the initial input point. It can be observed that the point is outside of the black boundary line of leg 1 therefore it is moved and iterated until it and the end point, the black cross within the boundary. The blue arrow represents the motion of the leg from the start to the end.

The leg is then moved to the start point at the start of the ground phase, then moved in the desired direction and at the desired speed.

When ground phase has ended and goes into the swing phase the leg needs to go from its current position to the start position. The current position is read using the FK function. From this the direction and velocity that the leg needs to travel can be calculated using the length of the swing phase and the distance from the two points. But the leg needs to leave the ground on the return journey. If not, it will pull the body back and effect the robot trajectory.

To determine the height H of the leg's end effector at any point during the swing phase the equation below is used:

$$H = WH + ratio * \sqrt{\left(\frac{dist}{2}\right)^2 - \left(\left|\frac{dist}{2} - currdist\right|\right)^2} \tag{13}$$

Where WH is the current walk height, the $dist$ is the distance from the end point to the start point and the $currdist$ is the current distance from the end of the leg to the starting position and this is read through the swing movement.

The ratio value controls how high the return path is. The below graphs were created using Desmos [14] with a walk height WH of 0 and dist equalling 10.

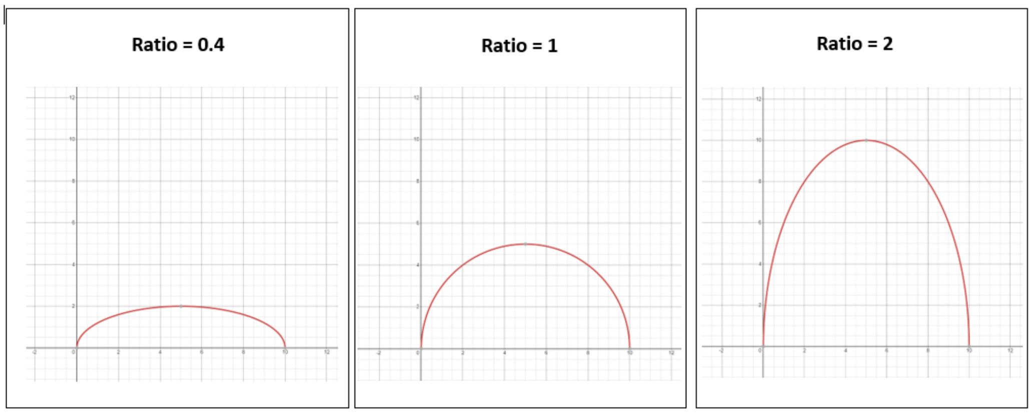


Figure 10: Effect of ratio on ellipse height

As you can see as the ration is increased the path becomes more elliptical and when decreased the path becomes flatter. A higher ratio would allow for the robot to clear taller objects and be better adapted for terrain with possible objects.

The main walking process follows the process below:

```
1.  For each leg i:
    1.1.  Calculate start and stop coordinates using previous function
    1.2.  Get the leg cycle to see if the leg i is in stance or swing phase
    1.3.  Calculate the velocity v by using the speed and angle of direction
    1.4.  If leg cycle equal to 0:
          1.4.1. Set desired position des_pos to start point
    1.5.  If the leg cycle is in swing phase:
          1.5.1. Calculate the amount of time left in swing phase
          1.5.2. Read current position using forward kinematics
          1.5.3. Calculate velocity needed to cover distance between current position and initial position using
                 time left in swing phase
          1.5.4. Overwrite velocity v with new velocity calculated
    1.6.  Calculate desired position des_pos using velocity v and current position of leg i
    1.7.  If the leg cycle is in swing phase:
          1.7.1. Calculate current distance curr_dist between current point and initial point
          1.7.2. Use current distance curr_dist and equation( 13) to calculate height of desired position des_pos
    1.8.  Translate des_pos from base frame to leg frame of leg i giving coordinate translate_des_pos
    1.9.  Take translate_des_pos and calculate desired angles leg_angles to reach position
    1.10. If leg_angles have value 404:
          1.10.1.   Pass
    1.11. Else:
          1.11.1.   Send leg_angles to controller of leg i
2.  Increase cycle value
3.  Set cycle value to 0 of cycle equal to max value
```

The result of this is a slow but stable walk in any desired direction and speed. https://youtu.be/pJ6C2kFfydc

The process used for the rotation is similar.

➢ Rotation

Rotation of the body is based off a similar system as the walking. All the legs need to be moving in the same direction at the same speed with different offsets. Here the speed is defined as the degrees per second and the direction means moving along a circular path with all the rotations centred around the same point.

The whole rotation system mainly uses two equations. The first one allows for a point to be rotated a certain number of degrees around the centre of 0,0. For example given the radius of the circle r, point A and the rotation θ; point B can be calculated.

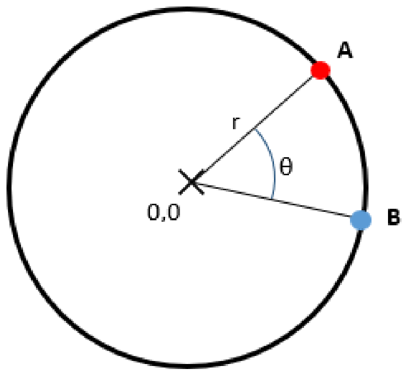


Figure 11: Diagram of degree movement equation

$$B = \begin{bmatrix} r * \sin(\theta + atan2(x,y)) \\ r * \cos(\theta + atan2(x,y)) \end{bmatrix} \tag{14}$$

Where x and y are the coordinates of point A.

The second equation finds the two intersection points $(x_{1/2}, y_{1/2})$ of a line going through a circle given a point in 2D space (x,y), an angle at which the line crosses the point θ and the radius of the circle r [15].

$$m = \left(\frac{\cos(\theta)}{\sin(\theta)}\right) \tag{15}$$

$$c = y - (m * x) \tag{16}$$

$$x_{1/2} = \frac{-m * c \pm \sqrt{r^2(-m^2) - c^2}}{-m^2} \tag{17}$$

$$y_{1/2} = \frac{c \mp \sqrt{r^2(-m^2) - c^2}}{-m^2} \tag{18}$$

The steps for the actual rotation.

The first step is to calculate the angle of the arc that leg will move along. This is simply done by calculating the product of the rotation speed and time.

The next stage is to find the point at which the radius from the centre of the robot and the middle line between the robot's leg's max operation angles intersect. This is achieved using the equations setup above. This will return two points to find which one is within the operational range of the leg, by using the atan2 function in order to check whether it is the operational range of the leg. Then two points are created going in opposite directions using the equation(*14*). The angle used is halve the angle of the arc initially calculated in both clockwise and anti-clockwise direction. If both points are not within the leg's operational range, then the radius is increased until both points are valid.

The gait used for the rotation is the same wave gait as the one used for the forward walk.

Which Machine Learning Algorithm to use?

The two main algorithms being considered are Reinforcement Learning and an Evolutionary algorithm. The next section each algorithm will be introduced and then compared.

➢ Evolutionary Algorithm

Evolutionary algorithms are "*optimization based output tecniques, whereas learning algorithms are advanced linear equation solver or complex regression algorithms"* [16]. They work by creating a population where each individual is a representation of a solution. Each solution is then evaluated using a fitness function. The fitness function returns a value for how well each individual performs. Individuals are then selected proportionally to their fitness and then reproduced to create new individuals; these new individuals are slightly mutated and then are added to the new population. This process is repeated and then solutions will converge to an optimum.

The greatest issue with this technique is the necessity for numerous individuals which can be costly processing wise. Another aspect that needs to be planned is the representation of the solutions and how this will affect the agent in the simulated environment.

➢ Reinforcement Learning

Reinforcement learning is a type of machine learning where the algorithm is given a set of inputs and outputs and by using these inputs, also called 'observations' must chose certain outputs to maximise a goal or fitness function. This particular technique has advantages such as only needing one agent, but this also means that the training process can be longer than evolutionary algorithms as only one solution is being explored. It can also cause the algorithm to become stuck in a local maximum. It is the machine learning algorithm I have the least knowledge of. Reinforcement learning is best suited for solving problems that require sequential decision making [17].Reinforcement learning approaches such as some PPO methods map from action space to state space via a neural network.

➢ Comparison

A main disadvantage of using reinforcement learning is that I have less experience and would be relying on a library to do that main processing.

An evolutionary algorithm was chosen for specific reasons. The biggest draw towards an evolutionary algorithm was its flexibility [18]. Some reinforcement learning which use neural networks uses regression to improve and train its networks. This means that the fitness function for RL needs to be differentiable [18]. Whilst an EA algorithm can have any function or equation as its fitness function, meaning "*EAs can be applied to any optimization problem where you can encode solutions, define a fitness function that compares solutions and you can stochastically change those solutions*" [19].

At the point where the type of machine learning used needed to be decided there was no incline into if the fitness function was going to be differentiable meaning that a gradient-based solution may not be applied such as RL. The EA flexibility means that they are straightforward to integrate and implement [18]. Another benefit of EA is the built-in process of generations means logs of the performance for each iteration can be recorded.

A major drawback to RL is that it can take longer to converge to a solution due to it only exploring one solution at one time. An EA has multiple agents meaning it has a greater ability to travel the search space. RL algorithms are more likely to get stuck in a local minima due to only one solution being investigated and its use of gradient descent increasing the amount of time needed for the algorithm to be developed to avoid these local minima or to restart the training. In terms of performance, some as AutoML by Google [20], an evolutionary algorithm that can "*outperform more expensive gradient-based methods*" [18].

General Overview of an Evolutionary Algorithm (EA)

An evolutionary algorithm used for parameter optimisation will follow this flowchart.

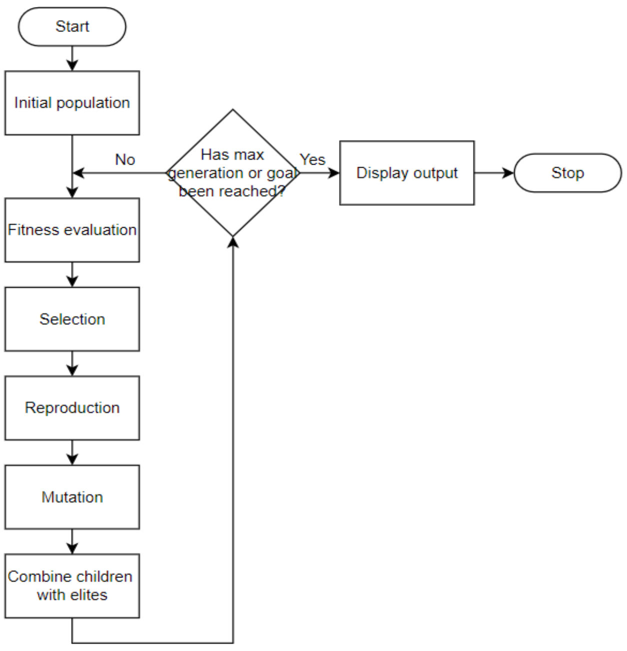


Figure 12: Flowchart of Evolutionary algorithm

➢ Initial Population

The first step of an evolutionary algorithm is to create the initial population. This is achieved by generating a matrix of size n by m where n is the desired population size and m is the number of parameters to train. These parameters are random numbers of either uniform or normal distribution within the range specified for each parameter. Each parameter can be a float, integers, or a binary value.

➢ Fitness Evaluation

Then comes the main loop of the algorithm. It starts with a fitness evaluation of all the individuals. This evaluates performance of each agent. For this particular case the fitness will return a high value for a good performance and a low value for a bad performance.

Rewards are positive values added to the total, these reward positive behaviours. Whilst costs are negative values taken off from the total fitness, these are used to supress undesired actions.

How much each cost effects the fitness depends on the factor assigned to each. For example, squaring a cost will mean the algorithm is punished very heavily for this action.

➢ Selection

Afterwards selection is needed. This is where the individuals are sorted depending upon their fitness. The top percent of the individuals are automatically transferred to the next generation. These are called the elites.

Then two random individuals are selected with their chances of selection being proportional to their fitness. These two are the parents used for reproduction.

➢ Reproduction

Reproduction involves taking the parent agents and picking randomly between each of their hyperparameters using a coin flip and the selected hyperparameters being transferred to the child. This creates a child; this child is then mutated.

➢ Mutation

Mutation involves adding some more randomness to the child. This allows for the child to explore different possible hyperparameter combinations to try and find a better one, rather than exploiting the current ones. This is achieved by iterating over each of the child's genes (hyperparameters) and modifying it with a chance of 1 over the number of genes.

The process of selection, reproduction, and mutation is repeated until the number of children and elites is equal in size to the original population.

The whole process is stopped if the desired goal of the algorithm is reached or it has reached the max number of generations.

After each generation the population and its genes(hyperparameters) are saved to a file, this is called a checkpoint. Checkpoints are used in case of a failure to limit the amount of training lost. The data from the checkpoint may also be used to generate graphs to show trends in the data.

Implementation of an Evolutionary Algorithm

The hyperparameters of the evolutionary algorithm are shown:

- Max fit: 10000

  This value is just set to be high as the number of generations needed will vary and cannot be calculated.

- Elites' percentage: 40%

  The elite percentage controls how many individuals are copied and passed over to the next generation. From previous experience with evolutionary algorithms, 40% seems adequate as it creates a good balance of exploitation vs exploration.

- Population size: 10

  The population controls the numbers of individuals in the population for each generation. A larger population can lead to better results but will mean each generation will take longer.

- Fitness time: 5

  The fitness time is number of seconds the simulation will run, 5 seconds is chosen as it allows for two full runs of the algorithm when each run is 360 units.

The only sections that needs to be changed and adapted to fit the simulation are the initial population and the fitness evaluation.

The initial population used for the algorithm is generated by creating a set of hyperparameters for each agent in the population. The basic starting parameters used for the basic walk are defined in an array with their type, either float or integer, their starting value, and their range.

Each hyperparameters for each agent starts as the initial hyperparameter value with a slight modification. Initially the values were completely uniform over all the value's range. This caused the solutions to be unfocused and take too long to converge. They were changed later on to start as the default value with mutations.

Fitness evaluation needed to be completely redone to fit the simulation. For the fitness the walking algorithm will run using the created hyperparameters for 5 seconds or however long the fitness time specifies. After the 5 seconds some aspects of the simulation will be measured and used to calculate the fitness of the current representation of the hyperparameters.

The aspects to measure will depend entirely on what parts of the algorithm need to be punished and what parts need to be rewarded. The basic fitness would be to simply return the distance in the forwards direction that the robot has travelled.

The mutation works by generating a random number with uniform distribution within the hyperparameters range. Initially the randomness's range would only be within 10% of the range with a normal distribution focused on the parameter. This, however, created an issue where the value wouldn't explore enough other solutions.

Hyperparameters to Train

The hyperparameters to be trained are required to be selected purposefully. These parameters are a mix of values that have to be set and some parameters which were used in other papers were brought over and implemented. In [7] there are hyperparameters such as the height of the return of the leg, gait of the walk, the height of the body, etc… due to the robot here being a four legged robot which a different design, with different legs for its front and back pair some parameters need to be translated and some liberties needed to be taken and some hyperparameters in the papers to not be used.

All the hyperparameters are listed:

- The speed of the spider

  The goal will be to maximize this value whilst trying to get a walk that resembles something achievable, reasonable, and stable.

- The swing times

  The swing times is the amount of time the leg stays in the air. This value is currently the same for all legs, but this may change if it has a positive effect on the robot's performance.

- The start points of the swing phase

  This parameter controls the start of the swing phase of each leg. This parameter and the previous of the swing time are what defines the gait of the robot which heavily affects the max speed.

- The initial location for the start coordinates of the legs, each being an x,y value for each leg.

  This will be the first search point for the walking algorithm when looking to create a movement for the leg. If the point given doesn't work, then the algorithm will generate one using the initial looking distance.

- The initial looking distance when creating the location for the foot to move

  This determines the distance from the initial point for the algorithm to search when the initial points fail to give valid start and end points.

- The walk height of the spider

  This was initially left fixed as it caused issues initially but as the robot's performance improved it was allowed to change the height. The walk height will also affect the torque needed for each joint.

- The height of eclipse for the return path of the leg

  This hyperparameter determines the height of the return path. It will be more valuable if the robot needs to be able to adapt to terrain where the height of the return path will allow it to overcome obstacles. It will have little effect on the actual speed. This hyperparameter of similar design has been used in other papers [7].

Some hyperparameters are left fixed, such as the rotation of the direction of movement of the spider.

The robot always started at 0,0 in the world's coordinate frame.

Log of Outcomes

This section will cover the different major outcomes which have come out of the testing sessions over multiple runs and how this affected the algorithm but mainly how this effected how the fitness is measured via the fitness function.

- **Outcome 1:**

The first iteration of the training has the fitness function where the reward is the final value of the y position of the agent. This is so that the agent is only rewarded for movement in the forward direction. The value of y position is multiplied by 1000. This converts the distance from meters into millimetres, to make the distance more important and allows for some costs to be removed whilst keeping the fitness positive.

The outcome of this was random movement but the general movement was in the correct direction.

- **Outcome 2:**

The next fitness equation was to punish the sideways movement by subtracting the x position multiplied by 10 and squared. This creates a very large cost for going in a sideways direction at a greater distance - more than 0.1 m. This error of 0.1 m is to give the algorithm some “wiggle room” to improve training outcomes and reduce training times need for a stable walk.

The results were still random movement similar to outcome 1 but with less sideways drift.

- **Outcome 3:**

For this test the fitness was not changed but the population was increased from initially being 5 to 10. Therefore, there are more agents which will help with exploration.

The outcome for this was an improvement with less “random” movement. But an issue has arisen. When the leg start points are generated and the point is too close to the body the leg will do an unusual movement where it will initially move in the correct direction towards the body. It will then stop and move the opposite direction whilst still being on the ground. This is an issue as the body will move forward and then be pushed back. After some investigation it was deduced that this issue is caused by the inverse kinematics being carried out using a solution created from equations, as these equations do not take into account some of the issues caused by the inverse kinematics problem having many solution for the same problem. The fix for this would be to redo the inverse kinematics. Another method is to train this behaviour out of the robot.

When the issue of the leg pushing on the ground occurs the body of the robot will rotate slightly. This rotation can be used to punish the robot for rotation. This is achieved by cumulatively adding the yaw rotation of the robot at every time step and then subtracting it from the reward gathered.

- **Outcome 4:**

The fitness function for this outcome had the addition of the penalising for the yaw rotation. This change fixed the issue of the push back. The robot now walks straight along the Y axis. A new problem needs to be addressed. When walking the robot will ‘jiggle’.

This issue can be solved using the onboard gyro. This gives the movement in the each of the rotational axis. Similar to the previous outcome all the rotations measured from the gyro can be cumulatively added and then subtracted from the fitness.

- **Outcome 5:**

Not much progress was made after some training. The results were barely an improvement from the original walk. This issue may be due to the punishment from the gyro being too high. To solve this the cost of the gyro was halved.

- **Outcome 6:**

Two training sessions were done with the change to the fitness being the halving of the gyro cost.

The first training session resulted in a walk where the spider would use its back legs to slide along the ground. This was efficient but not realistic. Due to the nature of the goal of the algorithm sometimes the system will find a way of getting a high fitness by exploiting the inevitable simulator simplifications. This is unfortunate and unavoidable.

The second training session result had an excellent result. The walk was efficient and quick whilst keeping the body stable. This meets the original design requirements.

These two outcomes show that due to the very nature of using random values to mutate the agents, the behaviours can be very different despite being given the same inputs.

Final Fitness Function

After the changes done to the fitness function after each outcome of each training session. The final fitness function is shown below.

$$fitness = 1000 * Y - (X * 10)^2 - rotation - \frac{gyro}{2} \tag{19}$$

Where Y is the final y coordinate of the robot in the world. X is the final x coordinate. Where $rotation$ is the rotation log and $gyro$ is the gyro log

Improved Hyperparameters

This section will cover the values for the hyperparameters gathered from previous research. The aim will be to see if these parameters will improve performance. The performance of these hyperparameters will be compared to that of the original basic walk and the walk which was trained.

Some parameters are quite specific to the robot and therefore alternative values for these parameters may not be possible to be researched.

- The height of eclipse for the return path of the leg

This value doesn't affect the speed considerably as the leg will always take the correct amount of time to return to the starting value. It is more useful if the test is taken further and applied to rough terrain.

- The swing times
- The start points of the swing phase

These parameters will have the most effect on the speed of the robot.

It was noticed that the gait of the robot when trained was little affected, therefore the need to initially manually set the gait might hopefully push the robot in a better direction.

The gait that will be used is the tripod gait specified in [11]. This gait is commonly used by six legged insects [21]. This gait has three legs off the ground when in the swing phase. This the minimum amount for full stability when walking. This gait - of a maximum of three legs off the ground - is similar to that of an actual spider [13]. However, the spider's gait of only three legs on the ground at once was not used as the swing and support times seemed to be very unspecific and random. The tripod gait will therefore be adapted to the spider robot so that four legs are on the ground at once, rather than three such as the gait used by actual spiders. This is done as it increases the size of support polygon where the centre of mass needs to be situated for the robot to be stable.

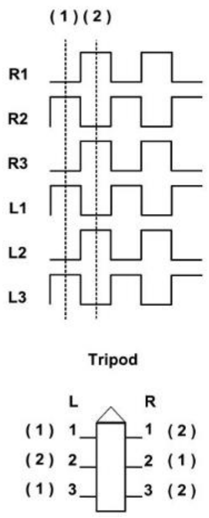
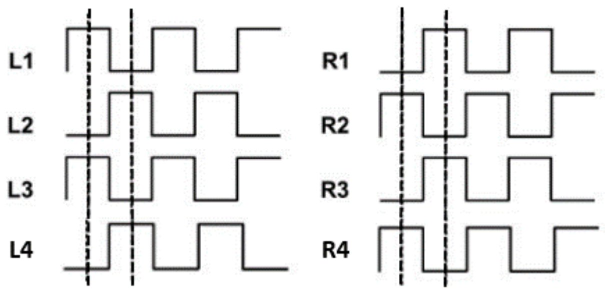


Figure 13: Tripod gait [11]

The tripod gait involves having opposite legs on the ground.

To adapt this gait for eight legs another level set will be added which are opposites to the gait of the previous leg. Meaning leg R4 will have the inverse gait than that of R3. The result being this gait.



This gait also has the swing stance times as equal. Therefore, the swing time will be initially equal to 180 which is half of the 360-unit cycle used by the basic walk. The training algorithm will be able to modify this, but the swing and stance lengths will stay equal.

Some values are specific to the spider therefore will stay with the same default values. These hyperparameters are:

- The initial looking distance when creating the location for the foot to move.
- The initial locations for the start coordinates of the legs, each being an x,y value for each leg.
- The walk height of the spider.

➢ Training

This new set of initial hyperparameters will be a new start point for the training, to teach a walk that is more efficient.

Using the fitness function created from the training of the basic walk, these new parameters can be trained and adjusted slightly so that they fit and can find the most efficient walk. The fitness of this new walk can be easily compared to that of the basic walk and trained basic walk using the fitness function.

There was however a complication that appeared during the training of the researched improved gait. Due to this gait gaining more distance, it improves the reward from forwards distance as expected. This means that the effect of the cost due to the orientation and gyro are less potent, reducing the total fitness by a smaller ratio. This is evident as when the walk is slower at the start of the training the robot is more stable and moves straighter, but as the training progresses and the distance travelled increases, the robot becomes more unstable by a minimal amount.

For the sake of fairness when compared with the previous training, the fitness function will not be changed further.

There are some possible fixes. The simplest is to multiply the cost by a value. This solution is flawed as then another value is unknown and needs to be tuned and balanced. An alternative solution would have the cost function take a share of the positive outcome rather than a straight value. Probably the best solution would be to use a different type of evolutionary algorithm such as a NSGA-II algorithm which allows for a multi-objective fitness function. This would eliminate the need to tune the fitness function. The NSGA-II algorithm is a multi-objective optimisation evolutionary algorithm. This means it can optimise a solution where the fitness function has multiple conflicting objectives. The agents trained are sorted not via a number returned from a fitness function, such as the evolutionary algorithm used here, but via non-dominated sorting and a crowding function. This allows for optimisation of multiple objectives without having to assign a value to each objective to scale how important they are [22].

Testing

Here is a list of various test done carried out during development and how the results of each test had an effect.

Name	*Goal*	*Method*	*Outcome*	*Implication*
Weight test	The goal of this test was to see if the servos were prefect with infinite strength or are limited and reacted appropriately and physics based	The method for testing was to set the mass of the "others section" to a large weight of 99kg	The servos are not perfect and are physics based depending upon the mass of each component	The servos in the simulation were accurate to actual servos, meaning if the robot can move in the simulation, then it can move in real life
Gazebo test	Check URDF within Gazebo, to check viability of Gazebo	Import URDF into gazebo and test visually	The robot slowly rotates or "flies into space"	Need to find new simulation software
Self-collision test	Check robot's self-collisions, this would improve accuracy	Import robot and enable self-collision	The robot behaves erratically, this may be due to some collisions interacting with each other	Needs to be fixed but will disable self-collisions until a fix is found
Evolutionary algorithm(EA) test	Check if EA is working as designed	Giving the EA the simple fitness function of reducing a value via the sum of 5 other values, the outcome should be the minimisation of all 5 values	All 5 values minimised correctly	This implies that the EA algorithm is running correctly and can be developed further for optimising the walk
Start and stop point test	To see if invalid points are being moved until the start and stop points are both valid	By giving the function an invalid value and check output points if they are valid	The initial point was changed and the start and stop points returned were both valid	The function runs correctly and therefore can be used in the main algorithm
Inverse kinematics(IK) testing	Check whether the angles returned by the IK function are correct	Input coordinates to function which returns angles. Input returned angles into MATLAB sim of arm, cross validate returned coordinates with initial coordinates	The angles returned were correct	The IK function can be used
Individual leg movement testing	Validate the movement of each leg	Run the movement on each leg individually and then in groups to make sure the movement is correct	The movement was correct but in the wrong direction	The fix for this issue is to invert/flip the angle in which the robot moves

Project Management

The methods used for the research and development of the thesis were appropriate for what they were intended to achieve. More research on some of the other techniques for parameter optimisation could have resulted in a different methodology. Nonetheless the test results achieved, using the method as detailed, gave an outcome that satisfied the principal aim of the report. The final methodology used was primarily due to my previous experience and understanding of evolutionary algorithms and it has delivered effective outcomes that satisfy the aims of the report.

The strategy used to tackle the processes required for effective outcomes, such as deciding whether to use reinforcement learning or an evolutionary algorithm, was to initially do research on some of the solutions used currently, and then to evaluate and test each. This is achieved by evaluating the pros and cons from a variety of sources [18] [17] and/or by testing each method directly. From these actions a methodology was chosen, and the reasons are indicated. . The chosen method was then implemented.

Even after the initial methodology testing and evaluation there was still a loss of time of approximately two weeks because the first method chosen for achieving a walk that was chosen was later removed and abandoned in its entirety.

A daily work logbook was kept where the progress of each stage, notes on outcomes and testing and current thoughts were written down. This enabled information to be focused on what was relevant. The major changes and detail of this logbook were then added to the main thesis at suitable intervals to reflect the work timeline progress.

The hardware used was perfectly adequate to carry out the training needed for the evolutionary algorithm. It is evident that more processing power would have increased the speed of training which could have resulted in results being achieved faster. Had the time for training been able to be reduced it could have meant that more time was dedicated to adding improvements such as torque balancing. This is conjecture, but the outcomes that were achieved satisfied the principal goals of the report.

There were failures in the processes, and a change that would have helped would have been an earlier implementation of a checkpoint system. There were some initial delays in the work timeline prior to the checkpoint system but once implemented it enabled an improved workflow.

One issue that occurred was the miscommunication between the measurement metrics of some of the software parameters. The measurements in the STL files were understood differently, with some software interpretating the measurements in meters and other software in millimetres. For some aspects, such as the visual representation of components this did not cause a major problem and was rectified quickly, but it was important for the creation of the inertia tensors. These tell the software how each component reacts to the world around it. Solving this was time consuming but essential.

Results and Evaluation

Basic Walk

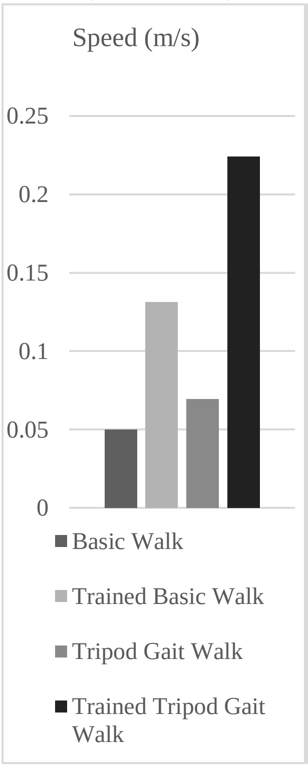
The outcome is that the basic walk is slow but steady - it can go in any direction using the ripple gait, only one leg is moved at one time. It can vary its speed up to a maximum value. The rotation also allows for the robot to turn, and with some alterations can allow the robot to turn a corner rather than having to rotate. Other parts of the walk can be changed such as the walking height and the height of the leg when in the swing phase.

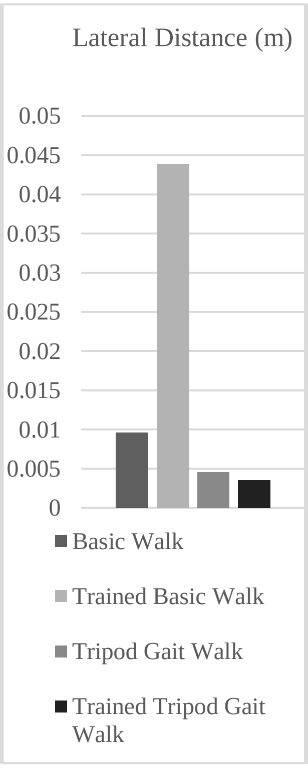
Training

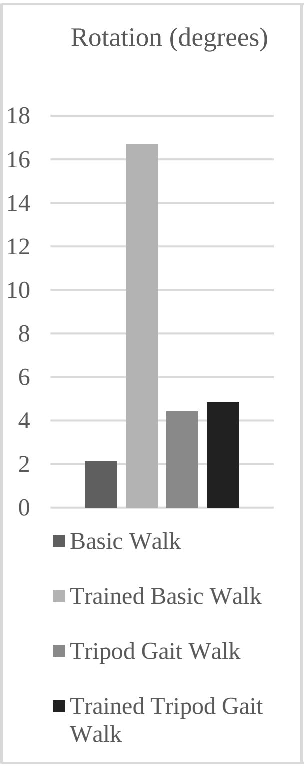
The results are of the final distances, speeds, and stabilities – in the form of the rotation error and gyro value - of the two walks with the different gaits. The results also show the outcome of these gaits after they have been trained using the same fitness function as (*19*)

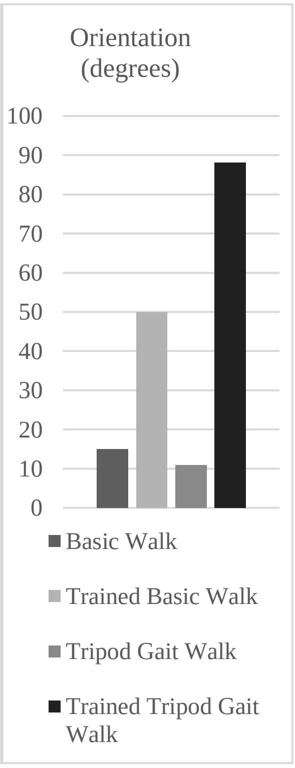
Outcome	**Fitness**	**Forward Distance (m)**	**Lateral Distance (m)**	**Rotation (degrees)**	**Orientation (degrees)**	**Speed (m/s)**
Basic Walk	239.13	0.2488	0.00096	2.1223	15.0661	0.04976
Trained Basic Walk	614.83	0.6567	0.04386	16.7115	50.0109	0.13134
Tripod Gait Walk	336.06	0.3459	0.00453	4.4279	10.8409	0.06918
Trained Tripod Gait Walk	1071.10	1.1200	0.00353	4.8228	88.1472	0.22400

Below are graphs for easier comparison. The goal is to maximise the speed whilst minimising the lateral distance, the rotation, and the orientation.









Personally created video of the four outcomes: https://youtu.be/pJ6C2kFfydc

The final hyperparameters are in the GitHub repository under log outcomes.

Comparison

From the outcomes it can be clearly demonstrated the effectiveness of using machine learning such as an evolutionary algorithm. The speed of the robot after training has more than doubled in both cases. This did affect the rotation error and stability but only via a marginal amount which is excusable, the error appears to be large in the graphs but is minor when viewed on video of the outcome. This could reduce even further with the use of stronger more balanced cost to rotation and orientation.

Discussion

The issue that can be noticed in the video is the self-collision of the robot. Some of the legs when moving collide with each other. This was thought to be rectified in the code by limiting the angle, but the error persists. This may be due to the inverse kinematics solution being a form which uses equations rather than the version which iterates the position of the leg to reach the desired goal. A fix and improvement would be to replace it for a more general method, and this was initially considered but replaced for form using equations version as it is quicker as it doesn't rely on iterating to a goal, rather than just using maths. In the simulation, self-collision is disabled as this would cause some unrealistic and unexpected bugs from the physics engine.

Another issue that can be observed in the video is the balance of the use of the legs. This would cause more stress on other legs. Another issue is that not all the feet touch the ground when they are in the stance phase. This also influences the force on each leg. If the agent is pushed more to balancing out the torques on each joint, then hopefully this could result in a walk where the effort is more shared between the legs.

To improve testing more processing power could be used. This would allow for the fitness tests, which are the current bottleneck in the system, to be able to run faster thus allowing for more generations for the agents to train in less overall time.

The current training is only for when the robot is moving in the forward direction. These changes might not be positive when the robot is moving sideways. This will be difficult to train to get the best hyperparameters due to the infinite number of possible directions. To remedy this the direction for training could be limited to certain angles and when training, an angle could be chosen at random.

Future Work

A next step for the algorithm would be the ability for it to adapt to different terrains. This could be expanded upon with the use of the contact sensors on the feet.

Further research would investigate more advanced gaits to improve the speed and efficiency of the walk. An example being the bipod gait used in [21]. This gait in testing was found to be faster on level ground than the gait used by actual insects such as flies.

The major next step will be to get an algorithm onto the spider robot for further testing. As stated at the start of the thesis this requires for the robot to be redesigned slightly on the physical side and a major redesign is needed on the software side. There needs to be the addition of the sensors to measure the value of the legs' joints. For the software side, the whole system needs to be redesigned using ROS rather than using a single IC.

Conclusion

The goal of the thesis was to achieve a walk on a pre-built and designed octopod robot and then to investigate how to improve performance of aspects of the movement such as speed without affecting stability. Improvements would be carried out by machine learning such as an evolutionary algorithm. Further improvements are investigated to see if the use of different parameters of the walk gathered from other sources such as papers would affect this performance.

The conclusion allowed for development of a walk using manual methods with the customisation of the movement such as the speed, direction and walk height. This outcome was a method that was stable but slow.

The next phase was the application of the machine learning to the forward movement of the robot and the introduction of different parameters, such as a different gait gathered from sources including other papers. This showed that there was a clear increase in speed with a minor effect on stability.

The outcomes from the application of a different gait are that there is an increase in speed of 140 per cent. With the use of machine learning - in this case an evolutionary algorithm - there was an increase in speed in both cases of over 260 per cent. With both the combination of the new gait, parameters, and machine learning for optimisation there was an increase in speed of 450 per cent, with the speed going from 0.05 m/s to 0.22 m/s.

The effect to stability by machine learning was minor and for the steadier gait it was inconsequential. However, reduced the error in the rotation of the robot.

Throughout the learning process, manual and later machine learning there were ongoing issues with leg collision. I attempted to solve this issue by limiting the movement of the leg in the code, but the problem persisted. The impact was negated in simulation by removing self-collisions so that I could continue with the core objective.

This will lead the way to the investigation in using this research for the improvements of the mobility of the octopod robot with the use of machine learning. It will lead the ability to move over uneven terrain and for the movement to be optimised in all directions.

The contribution of this thesis are the ideas presented is the design and application of a walk for an octopod robot. The other contribution is the application of machine learning to improve the forward velocity of the robot.

In conclusion, this thesis shows that machine learning, in particular evolutionary learning, has a key role in developing movement of robots.

The future for the application of machine learning for optimisation of robot movement is an ever-growing area improving the efficiency not only to the current walks of robots but also designing improved walks compared to those already present in nature around us [21].

My additional notes are that these more efficient walks are useful as it means the robot can traverse faster which can be vital in applications such as search and rescue. This could also lead to walks that are more efficient in the sense of preserving battery life. Longer times between charge could allow longer search times. Balancing the forces over multiple legs can reduce structural stress leading to a longer lifespan of a robot.

Table of Figures

References

[1] Thomas, “Flexible Muscle-Based Locomotion for Bipedal Creatures,” 25 November 2013. [Online]. Available: https://www.youtube.com/watch?v=pgaEE27nsQw. [Accessed 09 june 2022].

[2] T. Geijtenbeek, M. van de Panne and F. van der Stappen, “Flexible Muscle-Based Locomotion for Bipedal Creatures,” SIGGRAPH Asia, 2013.

[3] J. Foulkes, “S.T.E.V.E Specific Terrain Evaluation Verification Equipement,” Plymouth, 2021.

[4] C. S. Gurei, “3. Fundamentals of Hexapod Robot,” 29 June 2017. [Online]. Available: https://hackaday.io/project/21904-hexapod-modelling-path-planning-and-control/log/62326-3-fundamentals-of-hexapod-robot/discussion-170498. [Accessed 21 July 2022].

[5] H. Z. Khaleel and F. A. Raheem, “Hexapod Robot Static Stability Enhancement using Genetic Algorithm,” *Al-Khwarizmi Engineering Journal,* vol. 11, no. 4, pp. 44- 59, 2015.

[6] S. Chernova and M. Veloso, “An evolutionary approach to gait learning for four-legged robots,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566),* Sendai, Japan, 2004.

[7] N. Kohl and P. Stone, “Policy gradient reinforcement learning for fast quadrupedal locomotion.,” 2004. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1307456. [Accessed 26 July 2022].

[8] Cyberbotics, “urdf2webots,” 2020. [Online]. Available: https://github.com/cyberbotics/urdf2webots. [Accessed 09 06 2022].

[9] Univeristy of Birmingham, *Advanced Robotics: Homogeneous Transformation Translation + Rotation,* Birmingham, 2022, p. 3.

[10] Wikipedia, “Denavit–Hartenberg parameters - Wikipedia,” 2022. [Online]. Available: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters. [Accessed 21 July 2022].

[11] M. Žák, J. Rozman and F. V. Zbořil, “Design and Control of 7-DOF Omni-directional Hexapod Robot,” 17 December 2020. [Online]. Available: https://www.degruyter.com/document/doi/10.1515/comp-2020-0189/html?lang=de. [Accessed 2022 July 21].

[12] GeeksforGeeks, “Find if two rectangles overlap,” 06 Aug 2022. [Online]. Available: https://www.geeksforgeeks.org/find-two-rectangles-overlap/.

[13] X. Hao, W. Ma, C. Liu, Y. Li, Z. Qian, L. Ren and L. Ren, “Analysis of Spiders' Joint Kinematics and Driving Modes under Different Ground Conditions,” Appl Bionics Biomech, 11 December 2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6935789/#:~:text=These%20findings%20showed%20that%20when,spiders%20better%20stability%20in%20walking..

[14] “Desmos Studio,” [Online]. Available: https://www.desmos.com/about. [Accessed 22 July 2022].

[15] E. Hartmann, *Geometry and Algorithms for,* Darmstadt: Technische Universität Darmstadt, 2003, p. 17.

[16] A. Yayik, “How are "Evolutionary Algorithms" different from "Learning Algorithms"?,” 11 June 2015. [Online]. Available: https://www.researchgate.net/post/How_are_Evolutionary_Algorithms_different_from_Learning_Algorithms/557941a45f7f713ebc8b4584/citation/download.

[17] Neelarghya, “Reinforcement Learning vs Genetic Algorithm — AI for Simulations,” 26 July 2021. [Online]. Available: https://medium.com/xrpractices/reinforcement-learning-vs-genetic-algorithm-ai-for-simulations-f1f484969c56.

[18] D. Devansh, “Why you should implement Evolutionary Algorithms in your Machine Learning Projects,” 22 December 2021. [Online]. Available: https://medium.com/mlearning-ai/why-you-should-implement-evolutionary-algorithms-in-your-machine-learning-projects-ee386edb4ecc.

[19] nbro, “What is the difference between reinforcement learning and evolutionary algorithms?,” 2 June 2020. [Online]. Available: https://ai.stackexchange.com/questions/21584/what-is-the-difference-between-reinforcement-learning-and-evolutionary-algorithm.

[20] E. Real and C. Liang, “AutoML-Zero: Evolving Code that Learns,” 9 July 2020. [Online]. Available: https://ai.googleblog.com/2020/07/automl-zero-evolving-code-that-learns.html.

[21] P. Ramdya, R. Thandiackal, R. Charney, T. Asselborn, R. Benton, A. J. Ijspeert and D. FLoreano, “Climbing favours the tripod gait over alternative faster insect gaits,” 17 February 2017. [Online]. Available: https://www.nature.com/articles/ncomms14494.

[22] P. Calle, “NSGA-II explained,” 24 October 2017. [Online]. Available: https://oklahomaanalytics.com/data-science-techniques/nsga-ii-explained/. [Accessed July 2022].

Appendices

- Video of demonstration of walking: https://youtu.be/pJ6C2kFfydc
- Link to GitHub repository of code: https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2021/jxf139.git