

Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Learning a Structured Locomotion Algorithm for Hexapod Robots

Jiří Hronovský

**Supervisor: Ing. Teymur Azayev
May 2022**

I. Personal and study details

Student's name: **Hronovský Jiří** Personal ID number: **491868**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Cybernetics**
Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Learning a Structured Locomotion Algorithm for Hexapod Robots

Bachelor's thesis title in Czech:

Učení strukturovaného algoritmu pro chůzi na hexapoda

Guidelines:

The task is to implement a locomotion algorithm which is a hybrid of manually-designed and learnable elements which can be optimized using random search algorithms, giving an interpretable locomotion structure in contrast to end-to-end trained black box neural network policies. The task deliverables are as follows:

- 1) Implement an algorithm for hexapod locomotion which is centered around a learnable state-machine architecture for gait-phase (stance, swing, etc) transition for each leg. The algorithm has to use exteroceptive information of the terrain for foot placement and feedback to compensate for slippage and other factors.
- 2) Use an off-the-shelf Reinforcement learning or any other Random search algorithm to optimize locomotion parameters on various simulated terrains.
- 3) Evaluate the locomotion algorithm on several rough terrains in simulation, showing where it is suitable and in which cases it fails.
- 4) Compare proposed algorithm against an end-to-end learned unstructured neural network algorithm (can be without use of exteroceptive data) on several terrains in simulation.
- 5) Optionally test locomotion algorithm on analogous real platform.

Bibliography / sources:

- [1] Karim, Ahmad & Gaudin, Thibaut & Meyer, Alexandre & Buendia, Axel & Bouakaz, Saida. (2013). Procedural Locomotion of Multi-Legged Characters in Dynamic Environments. Journal of Visualization and Computer Animation. 24. 3-15. 10.1002/cav.1467.
- [2] Homberger, Timon & Bjelonic, Marko & Kottege, Navinda & Borges, Paulo. (2016). Terrain-Dependant Control of Hexapod Robots Using Vision. 10.1007/978-3-319-50115-4_9.
- [3] Campos, Ricardo & Matos, Vitor & Santos, Cristina. (2010). Hexapod locomotion: A nonlinear dynamical systems approach. 1546 - 1551. 10.1109/IECON.2010.5675454.

Name and workplace of bachelor's thesis supervisor:

Ing. Teymur Azayev Vision for Robotics and Autonomous Systems FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **28.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Teymur Azayev
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I would like to thank my thesis supervisor, Ing. Teymur Azayev, for his guidance, support, and motivation during my work on this thesis.

I would also like to give thanks to doc. Ing. Jan Bauer, Ph.D. for a consultation on servomotors.

Lastly, I would like to thank my family for their immense support during my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of inspiration used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses. used in the bibliography.

Prague, May 20, 2022

Abstract

This thesis focuses on generation of hexapod locomotion using a learnable structured algorithm consisting of a neural network that generates control parameters to a hand-designed movement generator. The proposed method is compared with a variant in which the neural network is replaced by a simple hand-designed gait controller. The proposed method is also compared with an end-to-end neural network. All neural networks are trained using an evolution strategy optimizer. The results of training and evaluation on five terrains show that the control parameters given by the trained neural network give a higher performance than the hand-designed control parameters. The proposed method also gives smoother results than the end-to-end neural network and trains faster. This method provides a robust foundation with predefined behavior, but can adapt to different terrains without having to redesign the implementation.

Keywords: hexapod, neural network, structured algorithm, adaptive locomotion, evolution strategy, simulation

Supervisor: Ing. Teymur Azayev

Abstrakt

Tato práce se zaměřuje na generování chůze hexapoda pomocí učení strukturovaného algoritmu skládajícího se z neuronové sítě, která generuje řídicí parametry ručně navrženého generátoru pohybu. Navrhovaná metoda je porovnána s variantou, kde je neuronová síť nahrazena ručně navrženým ovladačem chůze. Navrhovaná metoda je také porovnána s neuronovou sítí, která přímo řídí úhly kloubů hexapoda. Všechny použité neuronové sítě jsou trénovány pomocí evoluční strategie. Výsledky tréninku a hodnocení na pěti terénech ukazují, že řídicí parametry generované naučenou neuronovou sítí dávají lepší výsledky než ručně navržené řídicí parametry. Navrhovaná metoda také dává hladší výsledky než samotná neuronová síť a učí se rychleji. Tato metoda poskytuje robustní základ s předdefinovaným chováním, ale je schopná se adaptovat na různé terény, bez nutnosti změnit implementaci.

Klíčová slova: hexapod, neuronová síť, strukturovaný algoritmus, adaptivní chůze, evoluční strategie, simulace

Překlad názvu: Učení strukturovaného algoritmu pro chůze hexapoda

Contents

1 Introduction	1
1.1 Related work	2
2 Methods	5
2.1 Movement generator	5
2.1.1 Movement generator pipeline .	5
2.1.2 Hexapod description	6
2.1.3 Point cloud sampling	7
2.1.4 Locomotion generation	8
2.1.5 Direct kinematics	13
2.1.6 Inverse kinematics	14
2.2 Neural network (NN)	17
2.3 Policy	18
2.3.1 Learnable Structured Algorithm (LSA)	19
2.3.2 Structured Algorithm (SA) . .	25
2.3.3 Learnable Unstructured Algorithm (LUA)	26
2.4 Environment	28
2.4.1 Physics simulation	28
2.4.2 Hexapod representation	29
2.4.3 Terrain representation	29
2.5 Training	30
2.5.1 Training pipeline	31
2.5.2 Optimization method	32
2.6 Evaluation	32
2.6.1 Distance reached	32
2.6.2 Smoothness	33
2.6.3 Power consumption	34
2.7 Terrain set	35
3 Results	37
3.1 Results on terrain (a)	38
3.2 Results on terrain (b)	39
3.3 Results on terrain (c)	39
3.4 Results on terrain (d)	40
3.5 Results on terrain (e)	40
4 Discussion	41
5 Conclusion	45
Bibliography	47

Figures

<p>2.1 Hexapod description. The arrow pointing forward. 7</p> <p>2.2 Hexapod leg description. 7</p> <p>2.3 Point cloud approximation by heightmap. 8</p> <p>2.4 Visualization of leg range. Purple cube is the center point. Green circle represents horizontal cut through a vertically oriented cylinder. 8</p> <p>2.5 Visualization of different scenarios. Leg is assumed to be in stance phase: (a) leg is stuck, (b) leg is not stuck because it is not on the edge of the range, (c) leg is not stuck because the dot product of torso moving direction and Foot position is positive. 9</p> <p>2.6 Visualization of foot placement defined by Equations 2.3 and 2.5. 11</p> <p>2.7 Speed characteristics. Scores are defined in Section 2.6. 12</p> <p>2.8 Leg with coordinate frames for each particular joint based on DH-notation. Each frame is labeled with a number. 14</p> <p>2.9 Flipping leg form convex orientation to concave orientation. 16</p> <p>2.10 Diagram of NN. 17</p> <p>2.11 ReLU activation function. 18</p> <p>2.12 Diagram of learnable structured algorithm. Green color highlights the learnable part, red color highlights the structured part. 19</p> <p>2.13 Average sum of all leg policy rewards during training each leg parameter vector separately. 21</p> <p>2.14 Visualization of bilateral weight sharing. 22</p> <p>2.15 The course of separately learning the gait-phase transitions with bilateral sharing. 22</p> <p>2.16 Demonstration of observation. Each number indicates from which leg the value should be taken. 23</p> <p>2.17 Average policy reward during supervised learning. 23</p> <p>2.18 Diagram of structured policy. 25</p>	<p>2.19 Tripod gait. The leg groups can be distinguished by the red and green color. 26</p> <p>2.20 Diagram of Unstructured learnable policy. 27</p> <p>2.21 Caption 28</p> <p>2.22 Hexapod in PyBullet simulation. 29</p> <p>2.23 Generated point clouds. 30</p> <p>2.24 Terrain representation in PyBullet. 30</p> <p>2.25 Reward during training on terrain (d). 31</p>
--	--

Tables

<p>2.1 Control inputs to the movement generator..... 9</p> <p>2.2 Parameters of the DH-notation. 13</p> <p>2.3 Parameters of DH-notation used in experiments. 14</p> <p>2.4 Two architectures of gait-phase neural networks. 19</p> <p>2.5 State representation for gait-phase neural network. 20</p> <p>2.6 Architectures of torso height NN. 24</p> <p>2.7 Observation representation for torso height NN..... 24</p> <p>2.8 Architectures of leg height NN. . 25</p> <p>2.9 Observation representation for leg height NN..... 25</p> <p>2.10 Two architectures of end-to-end unstructured neural networks. 27</p> <p>2.11 Observation representation for end-to-end neural network..... 28</p> <p>2.12 Terrains that the policies were trained and evaluated on. 35</p> <p>3.1 Comparison of all policies on terrain (a). Movement generator has speed set to 0.3. D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red indicates worst in class. Blue indicates best in class. . 38</p> <p>3.2 Comparison of all policies on terrain (a). Movement generator had speed set to 1.0. D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class. 38</p> <p>3.3 Comparison of all approaches on terrain (b). D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class. 39</p>	<p>3.4 Comparison of all approaches on terrain (c). D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class. 39</p> <p>3.5 Comparison of all approaches on terrain (d). D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class. 40</p> <p>3.6 Comparison of all approaches on terrain (e). D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class. 40</p>
--	---

Chapter 1

Introduction

As electronics becomes smaller and computational power grows, research into legged robots is expanding rapidly. It is practically impossible to coordinate the movement of many joints without computer control, especially on rough terrains [1]. Rough terrains are probably the biggest motivation for developing legged robots. There are many places on Earth's surface that can be accessed by animals, but lots of them are inaccessible by wheeled and tracked vehicles [2]. This is why most legged robots are bio-inspired, because we can take advantage of the long natural evolution.

Advantages of legged robots according to [1]:

1. Legs can step over obstacles and up and down stairs and therefore can overcome even very rough terrain.
2. Legs can overcome soft terrain where the wheels could dig themselves.
3. Legs can cause less damage to some surfaces than wheels and tracks.
4. Legged robots can level themselves on rough terrain better, which can be, for example, beneficial for some types of payload.

Legged robots also have disadvantages:

1. They are much more complex because they require more actuators, sensors, and on-board computers to be controlled properly.
2. Very often, the actuators consume energy because they have to actively compensate for gravity even when standing still.
3. Generally, legged robots are slower than wheeled robots.

Most physically studied and physically tested in recent years were bipeds, quadrupeds, and hexapods. In this thesis, I decided to work with hexapod robots because they offer a great amount of stability and do not require as much dynamic control as quadrupeds. However, the proposed concept should be applicable in a wider scope.

There are many approaches to generate hexapod locomotion. Before machine learning was used in this field, many algorithms have been proposed that manually-designed the locomotion. However, very often they have to be specifically designed for some type of terrain, and redesigning the algorithm can be a long process and requires knowledge of the code or a qualified person. In recent years, machine learning has become a very popular approach. Machine learning enables adaptation to various terrains, but in most situations it cannot be manually changed without having to retrain the network, which can take a lot of time and can also lead to unwanted results.

In this thesis, I study whether an algorithm that is a hybrid between hand-designed and machine learning approaches is a viable solution in cases where an ordinary person needs to improve the performance of hexapod locomotion without having to change the code. The structured algorithm should provide a robust foundation that constrains movement to mitigate the unwanted results that a neural network can give. The structured algorithm should also allow the neural network enough room to adapt to various terrains.

To study the problem, I implemented a structured learnable algorithm for locomotion generation for a hexapod robot. The structured algorithm consists of a neural network that generates control parameters to a hand-designed movement generator. The movement generator calculates joint angles for hexapod legs based on exteroceptive data and the control parameters that consist of gait-phases, torso height, and leg lifting heights. It also compensates for slippage and other factors. Exteroceptive data are represented by a point cloud. The movement generator uses the point cloud for foot placement. To evaluate the performance, I generated five custom terrains. The results are compared to a variant of the proposed method in which the neural network is replaced by a hand-designed gait controller. The results are also compared with an end-to-end neural network. The parameters of all neural networks are optimized using an evolution strategy optimizer.

1.1 Related work

Generating locomotion for hexapod robots is a well-documented topic. One way is to hand-design an algorithm that generates locomotion using inverse kinematics and exteroceptive data [3],[4],[5],[6]. In [7] they generate adaptive terrain locomotion for animation purposes by voxelizing the terrain and pathfinding the legs through it. In [8] gait is controlled by designing a non-linear oscillator for each leg, and the frequencies of the individual oscillators were then controlled to generate the locomotion. Oscillators were also used in [9]. Machine learning approaches have become popular in recent years [10]. In [11] reinforcement learning was used to classify the type of terrain and based on that the gait was adjusted. Closely related work to this thesis is [12] where they used a neural network to learn gait patterns but without

exteroceptive data such as point cloud for foot placement. In [13], [14] they adapt locomotion to terrain using open loop gait control. [15] also trained structured policy but represented by Graph Neural Network. A decentralized neural network was used in [16] to learn each leg separately.

Chapter 2

Methods

In this chapter, I describe what methods were used during this research. This includes the design of algorithms, simulation environment, optimization methods, terrain generation, *etc.*

2.1 Movement generator

In this section, I describe how the movement generator was implemented and how all its individual core parts work.

The movement generator is initialized with the hexapod parameters. The movement generator was created to generate locomotion based on:

- Control input¹
- Current state of the hexapod (position, orientation, joint angles)
- Control parameters (gait-phases, torso height, leg lifting heights)
- Point cloud
- Speed

The movement generator takes inputs and performs a step in which it calculates the next position of the torso and also the next positions of the feet of the legs. Movement generator then calculates the joint angles using inverse kinematics and returns them as output.

2.1.1 Movement generator pipeline

I demonstrate the workings of the movement generator on Algorithms 1 and 2. All parts are described in the following sections.

¹Control input defines the direction in which the movement should be generated

Algorithm 1 Movement generator step

```
Estimate surface normal
Update torso pitch and roll based on the surface normal
if no leg is stuck then
    Move torso based on control input
    Update torso yaw based on control input
end if
Update torso height based on control parameter
Calculate torso transformation matrix
for each leg do
    Update leg (see Algorithm 2)
end for
return all joint angles
```

Algorithm 2 Leg update

```
Calculate direct kinematics
Calculate target foot position
if swing-phase then
    if foot height < required height parameter then
        Increase foot height
    else
        if distance from footXY to targetXY < threshold then
            Interpolate between footXY and targetXY
        else
            Interpolate between foot and target
        end if
    end if
end if
if stance-phase then
    Set foot height to the sampled height from the heightmap
end if
Clamp foot position to leg range
Calculate inverse kinematics
return joint angles
```

■ 2.1.2 Hexapod description

Hexapod is a six-legged robot. The legs were labeled according to their position, as can be seen in Figure 2.2.

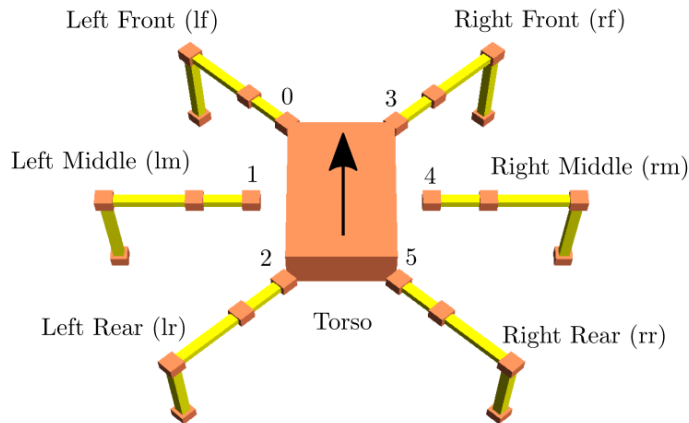


Figure 2.1: Hexapod description. The arrow pointing forward.

In this thesis, each leg has 3 joints and 3 links. To describe the hexapod leg, the terminology that can be seen in Figure 2.2 is used.

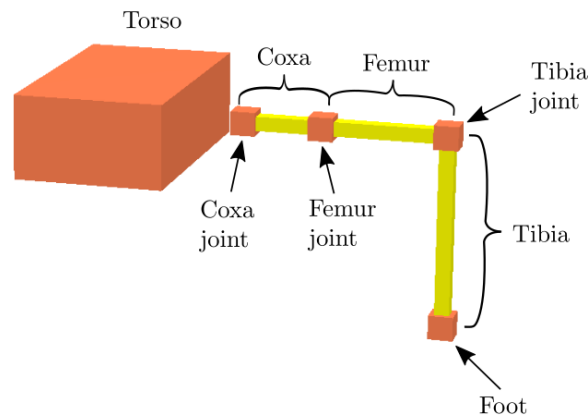


Figure 2.2: Hexapod leg description.

■ 2.1.3 Point cloud sampling

To make the sampling from the point cloud easier and more efficient, I implemented a heightmap that divides the region of the point cloud into a grid. Each grid tile stores the height of the highest point inside that tile, as can be seen in Figure 2.3. The tiles are stored in a 2D array. Sampling the heightmap at some point is then faster because the array can be accessed directly by converting the point's coordinates. If the point cloud is not dense enough to fill every tile of the heightmap, the heights of the empty tiles are calculated by iteratively interpolating the neighboring tiles.

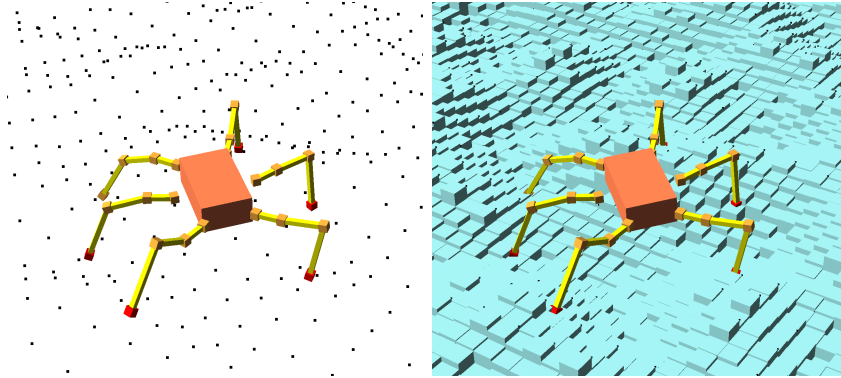


Figure 2.3: Point cloud approximation by heightmap.

2.1.4 Locomotion generation

In this section, I describe how locomotion was generated.

Definition of leg being stuck

Every leg is restricted to a range in which its foot must remain. This range is a cylinder with a center point and a radius.

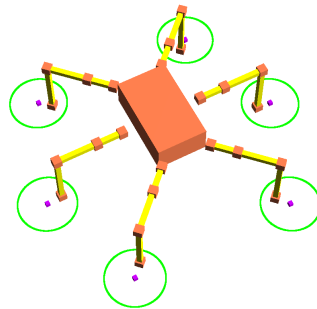


Figure 2.4: Visualization of leg range. Purple cube is the center point. Green circle represents horizontal cut through a vertically oriented cylinder.

For leg to be stuck, these criteria must be satisfied:

1. The foot must be at the edge of its range.
2. The leg has to be in the stance-phase.
3. The angle between the torso moving direction and the direction from the center point to the foot position must be greater than 90 degrees *i.e.*, the dot product must be negative.

These criteria are visualized in Figure 2.5.

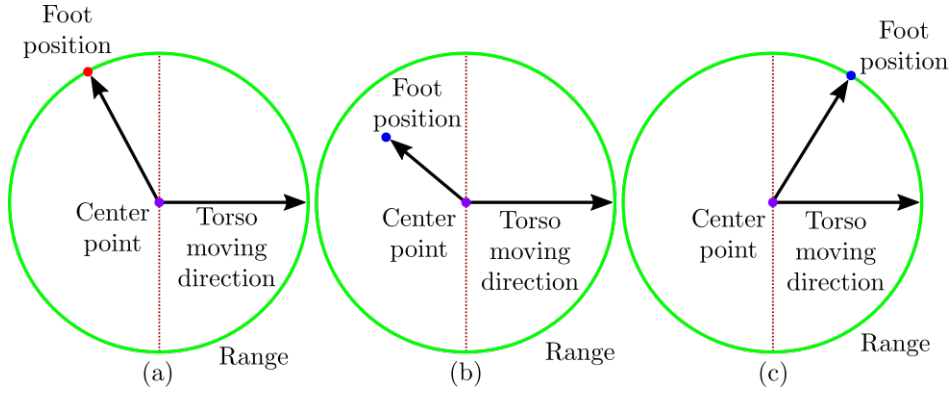


Figure 2.5: Visualization of different scenarios. Leg is assumed to be in stance phase: (a) leg is stuck, (b) leg is not stuck because it is not on the edge of the range, (c) leg is not stuck because the dot product of torso moving direction and Foot position is positive.

■ Torso movement

Locomotion is generated by moving the torso, while the legs have to compensate for the offset of the center of mass. The torso can be moved only in the xy-axis. If a leg is stuck, the torso cannot continue moving and has to wait for the leg to be put into the swing-phase. The movement is calculated based on the control input. All control inputs are relative to the hexapod frame. There are seven control inputs, as can be seen in Table 2.1. Inputs are discretized for simplicity, but in reality the direction can be \mathbb{R}^2 if more precise control is needed.

Input name	Direction (x,y)	Turn direction
Nothing	(0, 0)	0
Forwards	(1, 0)	0
Backwards	(-1, 0)	0
Left	(0, 1)	0
Right	(0,-1)	0
Turn left	(0, 0)	1
Turn right	(0, 0)	-1

Table 2.1: Control inputs to the movement generator.

The torso is moved by adding the control input direction or the rotation scaled by the speed.

Movement generator calculates the torso height as follows:

$$h_{t_z} = \frac{1}{6} \sum_{j=1}^6 t_{j_z} + h_d + h_i \quad (2.1)$$

where

- h_{t_z} is the torso height in world coordinates,
- $t_{1_z}, t_{2_z}, \dots, t_{6_z}$ are z-coordinates of the target positions,
- h_d is the default height value,
- h_i is the torso height control parameter.

■ Gait-phases

Locomotion had two gait-phases:

■ Stance-phase

In stance-phase, the leg is on the ground. Because the algorithm is designed to work in global coordinates, the foot position stays the same for the entire duration of the stance-phase. The only thing the leg has to handle during stance-phase is inverse kinematics to adjust for potentially moving torso.

■ Swing-phase

Swing-phase is divided into 3 sub-phases:

1. The leg foot is lifted by increasing its z-coordinate until it reaches the desired height.
2. The leg foot is translated towards the target foot position by interpolating the xy-coordinates of the current foot position and the xy-coordinates of the target foot position.
3. The leg foot is placed by interpolating the current foot position and the target foot position.

Leg height when lifting is calculated as follows:

$$h_l = h_t + h_d + h_i \quad (2.2)$$

where

- h_l is the leg height in the world coordinates,
- h_t is the terrain height sampled from heightmap,
- h_d is the default lifting height,
- h_i is the leg lifting height control parameter.

■ Target foot position calculation

Foot placement depends on the control input. The following examples can be seen in Figure 2.6.

When the input is *Nothing*, the foot target is placed to the center point of the range. When the input does not involve turning, the foot target is placed at the edge of the range in the desired direction:

$$\mathbf{t}_{xy} = \mathbf{c}_{xy} + r \frac{\mathbf{d}}{\|\mathbf{d}\|} \quad (2.3)$$

where

- \mathbf{t}_{xy} is the xy-coordinate of the foot target,
- \mathbf{c}_{xy} is the xy-coordinate of the range center point,
- \mathbf{d} is the direction,
- r is the leg range.

When the control direction involves turning, then the target foot position is calculated as:

$$\mathbf{w} = (\mathbf{J}_{coxa} - \mathbf{c}) \times \mathbf{z}_{world} \quad (2.4)$$

$$\mathbf{t}_{xy} = \mathbf{c}_{xy} + r d \frac{\mathbf{w}_{xy}}{\|\mathbf{w}_{xy}\|} \quad (2.5)$$

where

- \mathbf{t}_{xy} is the xy-coordinate of the foot target,
- \mathbf{c}_{xy} is the xy-coordinate of the range center point,
- \mathbf{J}_{coxa} is the coxa joint position,
- d is the turning direction,
- r is the leg range,
- \mathbf{z}_{world} is the z-axis of the world coordinate system.

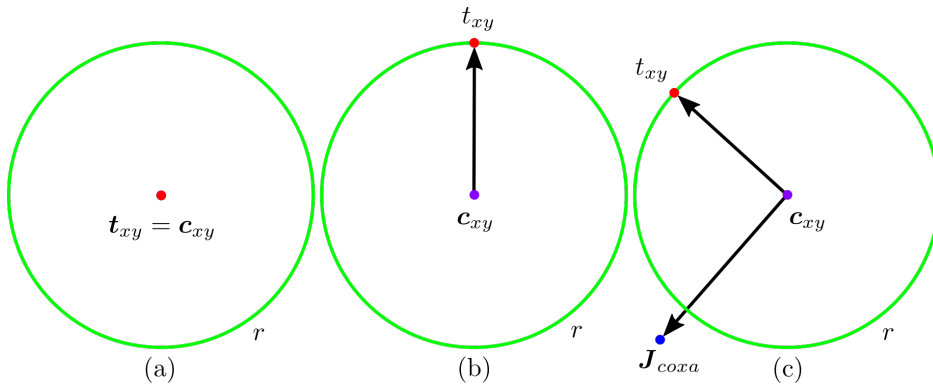


Figure 2.6: Visualization of foot placement defined by Equations 2.3 and 2.5.

After calculating the xy-coordinates of the target foot position, the z coordinate is sampled from the heightmap.

■ Surface normal estimation

The surface normal is estimated based on target foot positions. This is because the torso is oriented relative to the terrain based on the surface normal. In this way, each leg has a similar chance of reaching its foot target. The surface normal is calculated as:

$$\bar{\mathbf{t}} = \frac{1}{6} \sum_{i=1}^6 \mathbf{t}_i \quad (2.6)$$

$$\mathbf{n} = \sum_{i=1}^6 \sum_{j=1}^6 c(\mathbf{t}_i - \bar{\mathbf{t}}) \times (\mathbf{t}_j - \bar{\mathbf{t}}) \quad (2.7)$$

where

- \mathbf{n} is surface normal,
- \mathbf{t}_i and \mathbf{t}_j are the target foot positions if i-th and j-th leg,
- $\bar{\mathbf{t}}$ is the median of the target foot positions,
- $c = \begin{cases} 1.0, & ((\mathbf{t}_i - \bar{\mathbf{t}}) \times (\mathbf{t}_j - \bar{\mathbf{t}}))_z \geq 0 \\ -1.0, & ((\mathbf{t}_i - \bar{\mathbf{t}}) \times (\mathbf{t}_j - \bar{\mathbf{t}}))_z < 0 \\ 0.0, & i = j \end{cases}$.

■ Speed

Speed is a parameter of the movement generator that controls the amount of distance that the torso and legs can move in one step.

The speed characteristics can be seen in Figure 2.7.

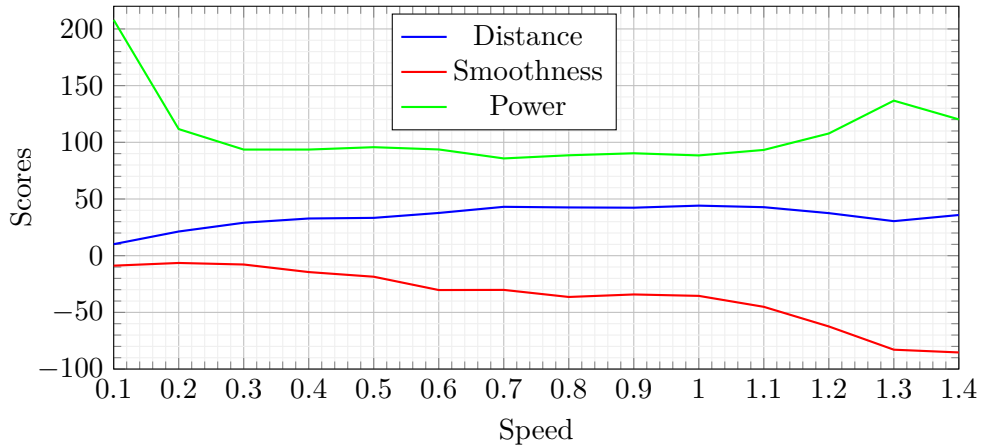


Figure 2.7: Speed characteristics. Scores are defined in Section 2.6

Based on Figure 2.7, the most energy efficient locomotion can be achieved with a speed of around 1.0, where the power consumption per distance is the smallest, the distance reached is the highest and the smoothness does not yet drop.

2.1.5 Direct kinematics

There are multiple ways to calculate direct kinematics. I used the Denavit Hartenberg notation (DH-notation) because it is used to describe serial kinematic chains. It uses 4 parameters to describe the spatial relationship between successive coordinate frames. (More about DH-notation can be found in [17])

The description can be obtained by following two constraints for each two successive frames:

1. The axis x_j is perpendicular to the axis z_{j-1} .
2. The axis x_j intersects the axis z_{j-1} .

The 4 parameters and operations associated with them are described in Table 2.2.

Parameter	Operation
θ	rotation around z-axis
d	translation along z-axis
a	translation along x-axis
α	rotation around x-axis

Table 2.2: Parameters of the DH-notation.

These parameters do not include the actual angle of the joint ϕ , therefore, ϕ must be added to θ :

$$\gamma = \theta + \phi \tag{2.8}$$

There is a transformation matrix $\mathbf{T} \in \mathbb{R}^{4 \times 4}$ for each parameter, and by multiplying them together we can transform the coordinates from frame i to frame $i - 1$:

$$\mathbf{T}_i^{i-1} = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) \cos(\alpha) & \sin(\gamma) \sin(\alpha) & \alpha \cos(\gamma) \\ \sin(\gamma) & \cos(\gamma) \cos(\alpha) & -\cos(\gamma) \sin(\alpha) & \alpha \sin(\gamma) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

where i is the number of frames to which we are transforming the coordinates.

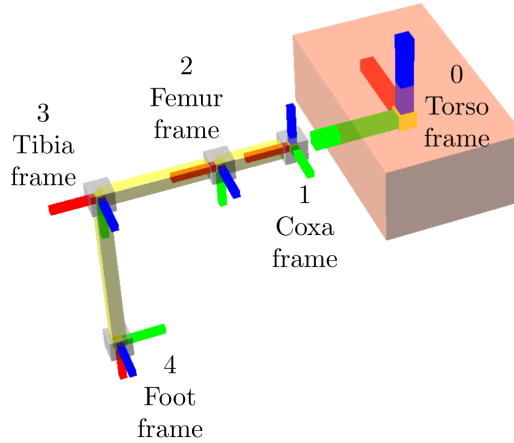


Figure 2.8: Leg with coordinate frames for each particular joint based on DH-notation. Each frame is labeled with a number.

To get the coordinates of the foot frame \mathbf{P}^4 in the torso frame, we must first transform the coordinates from the coxa frame to the body frame by matrix \mathbf{T}_1^0 and then use the matrices obtained from the DH-notation such that:

$$\mathbf{P}^0 = \mathbf{T}_1^0 \mathbf{T}_2^1 \mathbf{T}_3^2 \mathbf{T}_4^3 \mathbf{P}^4 \quad (2.10)$$

These parameters were used in all experiments:

Frame number	Frame name	θ	d	a	α
1	coxa	0	0	0.063	-90
2	femur	0	0	0.100	0
3	tibia	90	0	0.160	0

Table 2.3: Parameters of DH-notation used in experiments.

All parameters can be passed into the movement generator during initialization and can therefore be changed very easily ².

2.1.6 Inverse kinematics

The inverse kinematics of the legs of the hexapod was performed by Cyclic Coordinate Descent (CCD). I chose this method because it is simple to implement, computationally fast, and numerically stable. The number of links can be changed very easily without changing the implementation. Another important aspect of why I chose an iterative method over an analytic one is that it gives a solution even if the leg cannot reach the desired position by giving the closest solution.

² θ can be changed in GUI when setting the 0 angle joint position. The parameter "a" corresponds to the length of the leg segment

Method is used in 2D in a way that builds links from the target point³ to the root⁴ point. In each iteration, links are rotated one by one to point towards the next joint and translated so that they connect with the next link. This is done for all links, and then all links are translated so that the last processed link joint matches the root point. I demonstrate the process in Algorithm 3.

Algorithm 3 CCD algorithm

```

1: J : n+1 joints, first being the target point
2: l : n lengths of links
3: Pr : root point
4: for  $i = 1, 2, \dots, N$  do
5:   for  $i = 1, \dots, n$  do
6:      $\mathbf{v} = \mathbf{J}[i] - \mathbf{J}[i-1]$  - Find vector to the next joint
7:      $\mathbf{v}_l = l[i] \cdot \frac{\mathbf{v}}{\|\mathbf{v}\|}$  - Set length of v to the link length
8:      $\mathbf{J}[i] = \mathbf{J}[i-1] + \mathbf{v}_l$  - Update joint position
9:   end for
10:
11:   $\mathbf{d} = \mathbf{P}_r - \mathbf{J}[n]$  - Find vector to the base root point
12:
13:  for  $i = 1, 2, \dots, N$  do - Translate all joints by d
14:     $\mathbf{J}[i] = \mathbf{J}[i] + \mathbf{d}$ 
15:  end for
16:
17:  if  $\|\mathbf{J}[n] - \mathbf{P}_r\| < \text{some threshold}$  then
18:    Satisfactory solution found
19:  end if
20: end for

```

In this thesis, the CCD is implemented so that the coordinates of the target position \mathbf{P}^w are first transformed into 2D by rotating the leg into the XZ plane. To obtain the 2D coordinates of the target position \mathbf{P}^w the position must first be transformed into the coxa frame:

$$\mathbf{P}^1 = \left(\mathbf{T}_0^w \mathbf{T}_1^0 \right)^{-1} \mathbf{P}^w \quad (2.11)$$

where

- $\mathbf{T}_0^w \in \mathbb{R}^{4 \times 4}$ is the transformation from the world frame to the hexapod torso frame,
- $\mathbf{T}_1^0 \in \mathbb{R}^{4 \times 4}$ is the transformation from the hexapod torso frame to the coxa frame.

The angle of the coxa joint can then be calculated as:

$$\phi_{coxa} = \text{atan2}(P_y^1, P_x^1) + \theta_{coxa} \quad (2.12)$$

³In this case the desired foot position

⁴In this case the coxa joint

where θ_{coxa} is the DH parameter of the coxa link, which can be found in Table 2.3. \mathbf{P}^1 then can be transformed into the XZ plane by rotating it by $-\phi_{coxa}$ around the z-axis with the matrix \mathbf{R}_z .

$$\mathbf{P}_{XZ}^1 = \mathbf{R}_z \mathbf{P}^1 \quad (2.13)$$

And transformed into the femur frame like:

$$\mathbf{P}_{XZ}^2 = \mathbf{P}_{XZ}^1 + \begin{bmatrix} l_{coxa} \\ 0 \\ 0 \end{bmatrix} \quad (2.14)$$

where l_{coxa} is the length of the coxa link. 2.14 can be done because the coxa joint axis is vertical and lies in the XZ-plane. The representation of \mathbf{P}_{XZ}^2 in 2D is then:

$$\mathbf{P}_{2D}^2 = \begin{bmatrix} P_{XZx}^2 \\ P_{XZz}^2 \end{bmatrix} \quad (2.15)$$

This coordinate transformation was done because CCD operates in 2D. CCD is then run with the target point \mathbf{P}_{2D}^2 , the root point being at the origin, and two randomly initialized points $\mathbf{J}_{tibia}, \mathbf{J}_{foot} \in \mathbb{R}^2$.

If the CCD solution is in a convex configuration as can be seen in Figure 2.9, then the leg is flipped to the concave configuration as:

$$\mathbf{b} = \mathbf{J}_{foot} - \mathbf{J}_{femur} \quad (2.16)$$

$$\mathbf{w} = \mathbf{J}_{tibia_{convex}} - \mathbf{J}_{femur} \quad (2.17)$$

$$\mathbf{w}_{rej} = \mathbf{w} - \frac{\mathbf{w}^T \mathbf{b}}{\|\mathbf{b}\|^2} \mathbf{b} \quad (2.18)$$

$$\mathbf{J}_{tibia_{concave}} = \mathbf{J}_{tibia_{convex}} - 2\mathbf{w}_{rej} \quad (2.19)$$

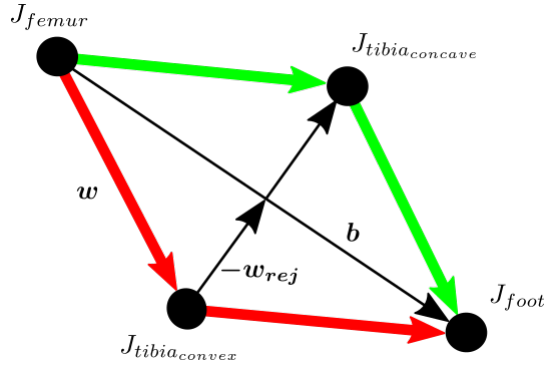


Figure 2.9: Flipping leg form convex orientation to concave orientation.

Finally joint angles are calculated using the atan2 function:

$$\phi_{femur} = -atan2(\mathbf{J}_{tibia_y}, \mathbf{J}_{tibia_y}) + \theta_{femur} \quad (2.20)$$

$$\phi_{tibia} = -atan2(\mathbf{J}_{foot_y} - \mathbf{J}_{tibia_y}, \mathbf{J}_{foot_y} - \mathbf{J}_{tibia_y}) - \phi_{femur} + \theta_{tibia} \quad (2.21)$$

where θ_{femur} and θ_{tibia} are the DH parameters of the corresponding links 2.3.

2.2 Neural network (NN)

Neural networks (NN) or also artificial neural networks (ANN) have become very popular in machine learning. NNs are used in many applications, including business, economics, science, engineering, medicine, climate, etc. [18]

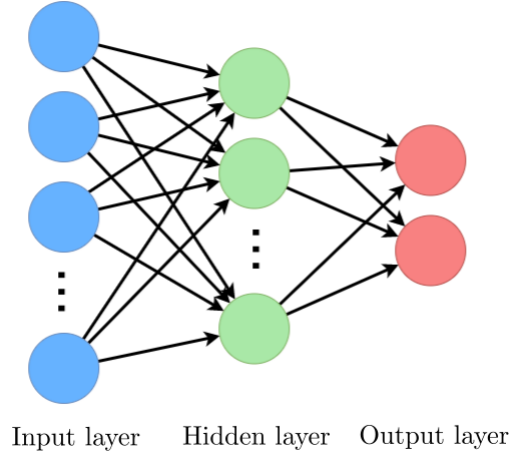


Figure 2.10: Diagram of NN.

NN have an input layer, an output layer and zero or more hidden layers, as seen in Figure 2.10. The forward pass of a single layer is calculated as follows.

$$f(\mathbf{x}) = \sum_{i=1}^m \sum_{j=1}^n w_{ij} x_j + b_i \quad (2.22)$$

where

- $\mathbf{x} \in \mathbb{R}^n$ is the input,
- w_{ji} is the element of matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ in row i and column j ,
- b_i is the i -th element of the vector \mathbf{b} representing bias.

Equation 2.22 can be written in matrix format as:

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b} = \begin{bmatrix} \mathbf{W} & \mathbf{b} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \mathbf{Q} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \quad (2.23)$$

where $\mathbf{Q} \in \mathbb{R}^{m \times n+1}$. The matrix \mathbf{Q} contains all parameters of a particular layer. The parameters of all layers can be written in one vector $\boldsymbol{\theta}$. I will refer to $\boldsymbol{\theta}$ as the parameter vector (or weights) of NN or the parameter vector of the policy if NN is part of that policy.

As an activation function, I used Rectifier Linear Unit (ReLU), which can be seen in Figure 2.11.

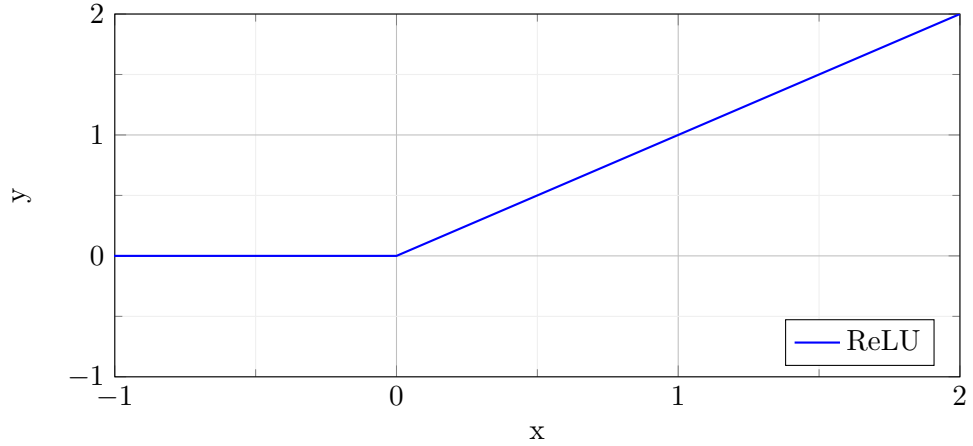


Figure 2.11: ReLU activation function.

Definition of the ReLU function is as follows:

$$f(x) = \max(0, x) \quad (2.24)$$

In this thesis, I used mainly architectures with zero hidden layers *i.e.* Single Layer Perceptron. If I tested NN with hidden layers, the ReLU was used only for these hidden layers.

2.3 Policy

In this thesis, the policy is used to generate the locomotion of the hexapod robot by controlling its joints. All joints are position controlled. The policy generates the target angle positions based on the current observation:

$$\pi(\mathbf{o}_t) = \boldsymbol{\phi}_{t+1} \quad (2.25)$$

where

- π is the policy,
- $\mathbf{o}_t \in \mathbb{R}^n$ is the observation at time t ,
- $\boldsymbol{\phi}_{t+1} \in \mathbb{R}^{18}$ are the joint target positions at time $t + 1$.

In this thesis, the policy was represented by three algorithms.

2.3.1 Learnable Structured Algorithm (LSA)

This is the algorithm (policy) proposed in this thesis. It has two parts:

1. Movement generator centered around state-machine for gait-phase transitions. Torso height and leg lifting heights are also parameterized and controlled through an input. A detailed description of the movement generator can be found in Section 2.1.
2. Learnable NNs that calculate the control parameter to the movement generator based on the observation. There is a separate NN for each input, as can be seen in Figure 2.12. NNs are separate because they can be optionally excluded from the decision process and replaced with the solution proposed in Section 2.3.2.

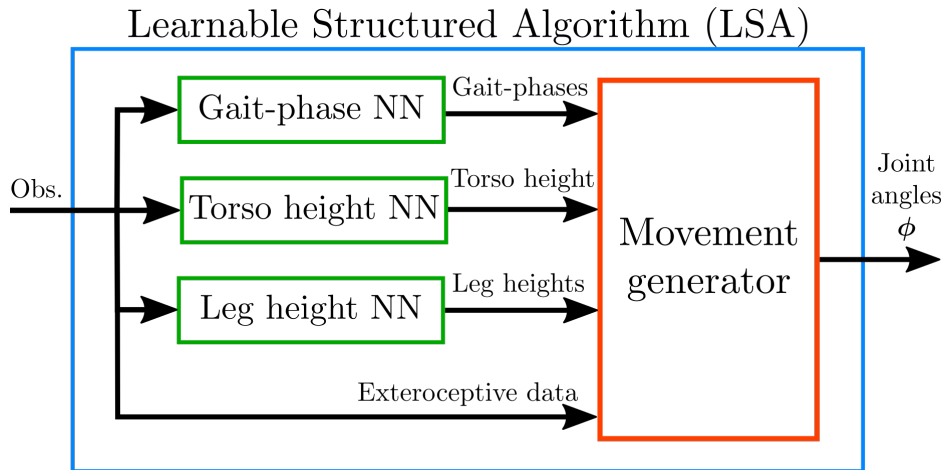


Figure 2.12: Diagram of learnable structured algorithm. Green color highlights the learnable part, red color highlights the structured part.

In following sections I describe architectures of the three NNs.

Gait-phase transition neural network (GP)

Motivation behind learning gait-phase transitions was to adapt the gait pattern of the legs to different terrains.

I kept the NN architecture simple without hidden layers. I experimented with two types of output layers, as can be seen in Table 2.4.

	Input layer	Output layer	Number of parameters
1.	25	2	52
2.	25	1	26

Table 2.4: Two architectures of gait-phase neural networks.

The first architecture in Table 2.4 was designed as a classification problem in which each output neuron represented either swing-phase or stance-phase:

$$\text{gait phase} = \begin{cases} \text{swing-phase, } a_1 > a_2 \\ \text{stance-phase, } a_1 \leq a_2 \end{cases}$$

where $\mathbf{a} \in \mathbb{R}^2$. The motivation behind this was that more parameters would make the NN more expressive.

The second architecture was designed in such a way that if the output was smaller than or equal to 0 the gait-phase was classified as stance-phase or swing-phase if the output was greater than 0:

$$\text{gait-phase} = \begin{cases} \text{swing-phase, } a > 0 \\ \text{stance-phase, } a \leq 0 \end{cases}$$

where $a \in \mathbb{R}$. This concept ended up being sufficient to learn locomotion and was used in the experiments in the rest of the thesis.

Observation vector $\mathbf{o} \in \mathbb{R}^{25}$ was the same for both architectures. A detailed representation of the observation can be seen in Table 2.5.

Name	Dimension
Gait-phases	6
Stuck states	6
Feet ground contacts	6
Distance to the target position	1
Surface normal	3
Leg joint angles	3
Final dimension	25

Table 2.5: State representation for gait-phase neural network.

Learning the gait-phase transitions was the most challenging part. The main problem was that the reward landscape was very flat with very narrow optimum spikes. This was caused by the design of the movement generator. Even when a single leg got stuck (defined in Section 2.1.4), the entire movement was on hold. This caused saturation in the reward.

To tackle the problem, I experimented with four solutions:

1. Learning each leg separately

This divided the parameter vector into six smaller parts which reduced the number of parameters to optimize at once. Each leg had its own optimizer. This solution sped up the learning process because the optimizer could not break the parameters of the other well-trained legs while

trying to improve badly trained legs.

Because gait-phase policies could no longer be rewarded collectively, I had to introduce individual rewards for legs. The legs were rewarded for good transitions and penalized for bad transitions. To reduce fast switching between the phases, the legs were slightly penalized for each transition.

A problem emerged with this approach because each policy was trained separately, but it was uncertain whether they will work together. Because each leg policy had as an input observation gait-phases of other legs. Badly trained legs could act randomly and, therefore, influence the other legs.

This caused an almost endless training loop where the reward constantly oscillated, as can be seen in Figure 2.13.

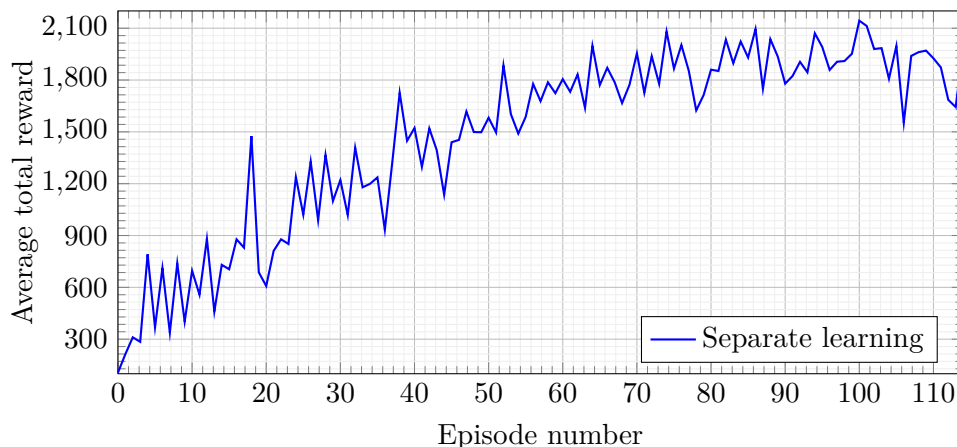


Figure 2.13: Average sum of all leg policy rewards during training each leg parameter vector separately.

Although some improvement can be seen in Figure 2.13 with an increasing number of episodes, this method is highly impractical because it does not guarantee convergence to a usable result. The trained policy was rarely usable.

2. Bilateral weight sharing

This aimed to reduce the number of parameters by symmetrically sharing the policy with the opposite legs (Figure 2.14) in both cases of training all parameters together and training them separately.

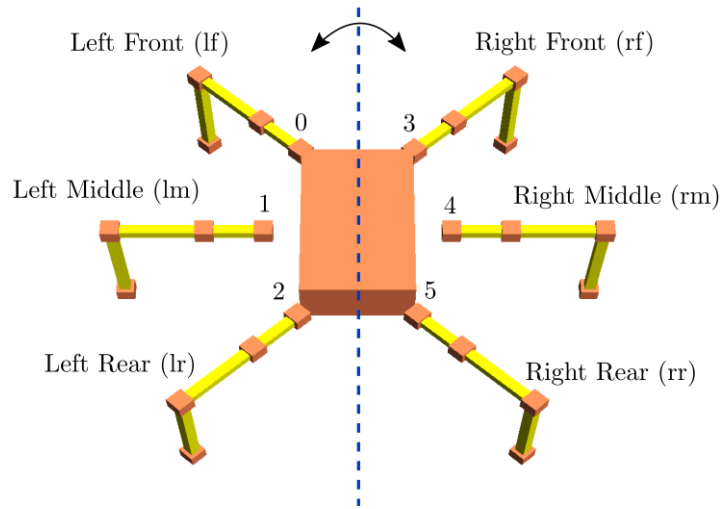


Figure 2.14: Visualization of bilateral weight sharing.

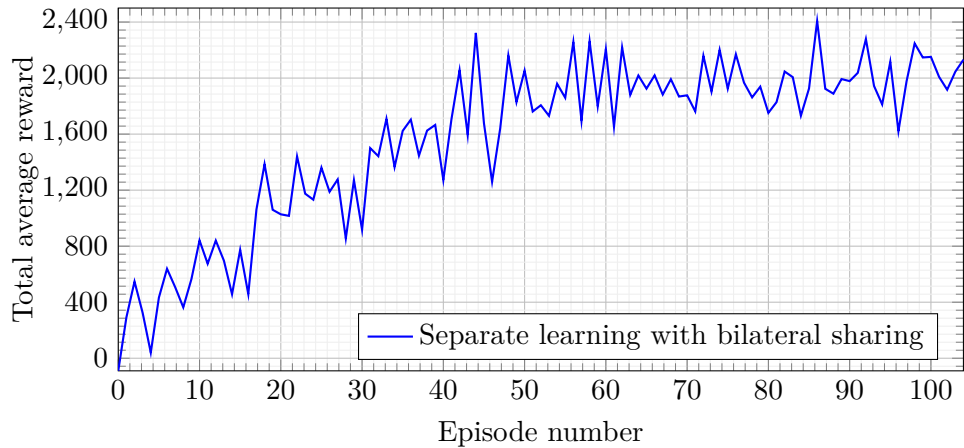


Figure 2.15: The course of separately learning the gait-phase transitions with bilateral sharing.

The observation vector had to be changed to represent the same state for both legs that share the same policy. This includes gait-phases, stuck-states, and feet ground contacts. The observation was structured as shown in Figure 2.16.

In case of training each leg separately, this did not help as can be seen in Figure 2.15. In case of training the three legs together, this had a positive effect on training speed, but no major results were achieved.

3. Supervised learning

The idea behind this approach was to avoid the unwanted observation dependency that could arise from individual training.

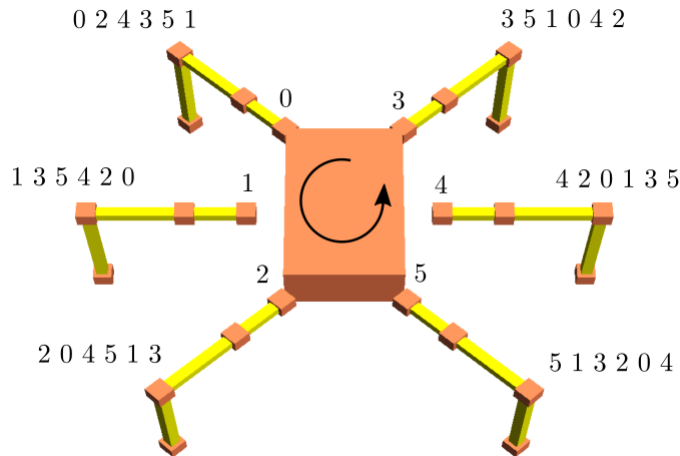


Figure 2.16: Demonstration of observation. Each number indicates from which leg the value should be taken.

The policy was trained using the controller described in Section 2.3.2 as a reference. The training pipeline remained the same, except that the gait-phases were determined by the controller. Based on the observation returned by the environment, the policy tried to predict whether the leg was in the swing or stance phase. If the prediction matched with the controller output, the policy gained reward. In this way, the policy could learn basic tripod gait and then could be trained optimizer with a very small training rate.

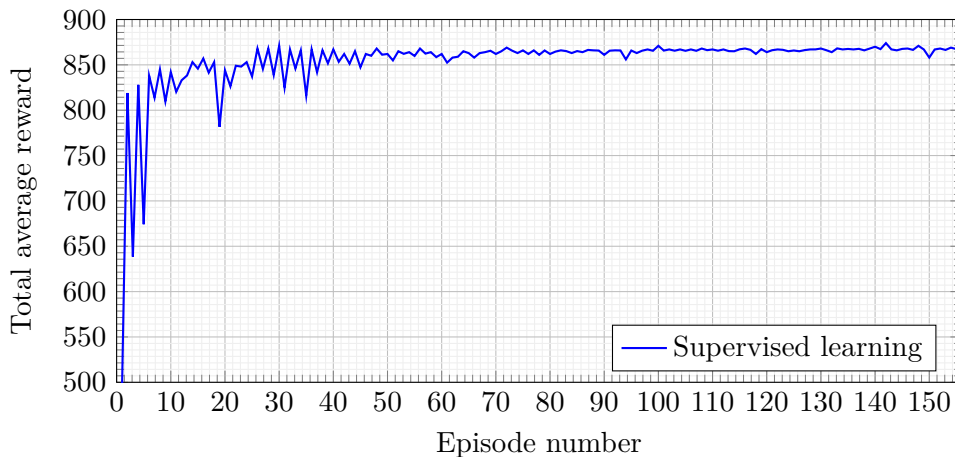


Figure 2.17: Average policy reward during supervised learning.

As can be seen in Figure 2.17, the policy gains the most rewards in the beginning. In this specific case, the simulation steps were set to 150

steps. In each step, the policy made a prediction and if the prediction matched the controller prediction, the policy gained a +1.0 reward. This meant that the maximum reachable reward was 150 for each leg, which was 900 in total. In Figure 2.17 the policy trains very fast during the first 40 episodes but then begins to learn very slowly. The result was usable most of the time, but unsupervised learning on rougher terrains usually diverged and made the method impractical.

4. Sharing one policy between all of the legs

This is one step further from the bilateral parameter sharing. This addresses all the previous issues. Advantages are that it uses the least amount of parameters and because all the legs use the same policy, the observation and reward are consistent. The best results were achieved with this approach. Basic locomotion could be achieved even during the first training episode. This was the method I ended up using in this thesis.

■ Torso height neural network (TH)

Torso height above the terrain is another parameter of the movement generator that can be learned. To speed up the learning, the movement generator has a default height value already predefined, and the policy learns to adjust this value by adding its output to it. The idea behind it was to enable adjustment to rough terrain and increase performance in situations where some legs cannot reach the target position on the ground or when overcoming tall obstacles.

For learning the torso height I used architecture which can be seen in Table 2.6

	Input layer	Output layer	Number of parameters
1.	25	1	26

Table 2.6: Architectures of torso height NN.

Representation of the observation vector $\mathbf{o} \in \mathbb{R}^{24}$ can be seen in Table 2.7.

Name	Dimension
Gait-phases	6
Stuck states	6
Feet ground contacts	6
Surface normal	3
Torso orientation	3
Torso height sampled from heightmap	1
Final dimension	25

Table 2.7: Observation representation for torso height NN.

■ Leg height neural network (LH)

The concept of learning leg height is similar to the torso height. Lifting the legs higher is beneficial on terrains with great height differences. On mainly flat terrains, lifting legs lower could be faster and more power efficient.

	Input layer	Output layer	Number of parameters
1.	8	1	9

Table 2.8: Architectures of leg height NN.

Representation of the observation vector $\mathbf{o} \in \mathbb{R}^7$ can be seen in Table 2.9.

Name	Dimension
Surface normal	3
Leg joint angles	3
Leg ground contact	1
Current leg height	1
Final dimension	8

Table 2.9: Observation representation for leg height NN.

■ 2.3.2 Structured Algorithm (SA)

The structured algorithm is similar to the LSA except that the input to the movement generator comes from the controller, as can be seen in Figure 2.18. This is the first of the two baseline algorithms that were evaluated for comparison with the proposed LSA.

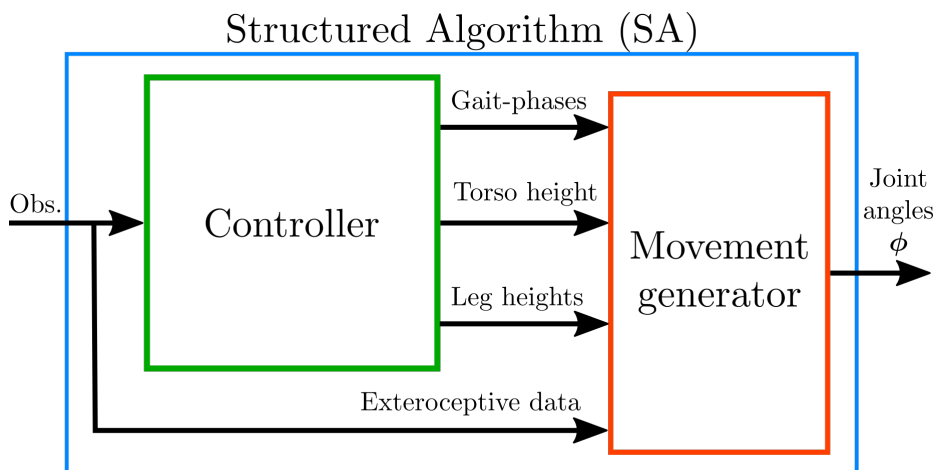


Figure 2.18: Diagram of structured policy.

■ Controller

The controller was created to test the basic functionality of the movement generator.

To control the gait-phases, the controller generates a basic tripod gait. In tripod gait, the legs are divided into two groups, as can be seen in Figure 2.19. The advantage of this gait is that there are always at least three legs on the ground to maintain balance.

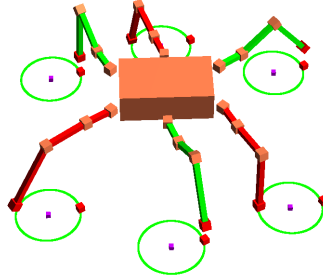


Figure 2.19: Tripod gait. The leg groups can be distinguished by the red and green color.

I demonstrate the gait-phase transition decision process implemented in the controller on Algorithm 4.

Algorithm 4 Gait-phase transition

```

1: if Group 1 is in stance phase and some leg from Group 2 is stuck then
2:   Put Group 2 in swing phase
3: end if
4: if Group 2 is in stance phase and some leg from Group 1 is stuck then
5:   Put Group 1 in swing phase
6: end if
7:
8: for leg in All legs do
9:   if leg distance from target foot position < some threshold then
10:    Put leg in stance pahase
11:   end if
12: end for

```

The torso height and leg lift heights were left as default values because the default values were designed to be versatile.

■ 2.3.3 Learnable Unstructured Algorithm (LUA)

This algorithm is represented by an end-to-end neural network (E2E_NN). All joint angles are controlled directly by the E2E_NN, as can be seen in Figure 2.20. This is the second of two baseline algorithms that were implemented

and evaluated for comparison with the proposed LSA.

The disadvantages are that it requires a lot of reward shaping to achieve the desired locomotion. In some cases, it is very difficult and time consuming to restrict unwanted movements.

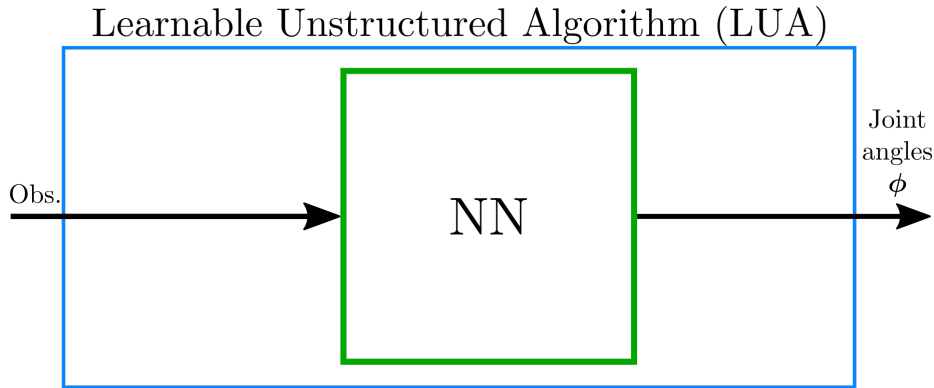


Figure 2.20: Diagram of Unstructured learnable policy.

I experimented with two NN architectures, as can be seen in Table 2.10.

	Input layer	Hidden layer	Output layer	Number of parameters
1.	51	0	18	936
2.	51	20	18	1418

Table 2.10: Two architectures of end-to-end unstructured neural networks.

The first architecture in Table 2.10 without hidden layers was sufficient for training on various terrains and trained faster due to the smaller number of parameters.

In this setup, the policy directly controlled the 18 joint angles of the legs of the hexapod:

$$\mathbf{a} \in \mathbb{R}^{18}$$

The angles were calculated as a forward pass based on the current observation. A detailed representation of the observation $\mathbf{o} \in \mathbb{R}^{51}$ can be seen in Table 2.11.

Name	Dimension
Torso velocity	3
Torso orientation (roll, pitch, yaw)	3
Torso angular velocity (roll, pitch, yaw)	3
Joint angles	18
Joint velocities	18
Feet ground contacts	6
Final dimension of parameters	51

Table 2.11: Observation representation for end-to-end neural network.

2.4 Environment

I created an environment with similar API to an OpenAI Gym. The environment is initialised with point cloud, hexapod parameters, maximum steps, noise level, and starting position.

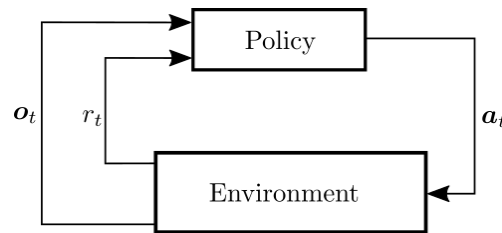


Figure 2.21: Caption

The environment has two functions:

- reset - resets the environment and returns the initial observation
- step - takes action as an input, steps the physics simulation and return observation, reward and information whether the objective was reached

Objective is reached after a certain number of iterations ⁵ or also if the torso pitch or roll exceeds a certain threshold.

2.4.1 Physics simulation

For simulating the physics, I used PyBullet Real-Time Physics Simulator [19]. PyBullet is a fast and easy-to-use Python module for robotics simulation and machine learning.

⁵Number of iterations could be set at the start of the training via the GUI

■ 2.4.2 Hexapod representation

The hexapod robot is created based on the Unified Robot Description Format (URDF) file which is generated from the hexapod parameters. To generate the URDF file from hexapod parameters I wrote a Python script. In this way, I can quickly change the hexapod parameters using my GUI and automatically generate the URDF file.

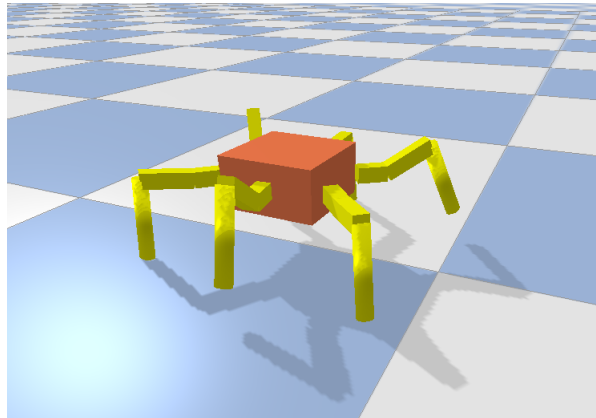


Figure 2.22: Hexapod in PyBullet simulation.

Hexapod parameters are:

- positions of the coxa joints ⁶,
- length of all links of the leg,
- position of the centre point,
- leg ranges,
- leg orientations,
- DH-parameters.

■ 2.4.3 Terrain representation

The terrain was represented by a point cloud. To generate the point cloud, I wrote a custom terrain generator. Using this generator, I could control the position, slope, density, size, and scale of the point cloud.

Three types of point cloud could be generated:

- Simplex - To generate this type of point cloud, I used the OpenSimplex noise Python package ⁷. As an extra input, it takes a seed.

⁶Coxa joints are also the leg mounting points.

⁷<https://github.com/lmas/opensimplex>

- Rocks - This point cloud also used OpenSimplex noise, but the heights were discretized.
- Stairs - This point cloud is a series of steps, as can be seen in Figure 2.23. The position, length, and height of these steps can be controlled.

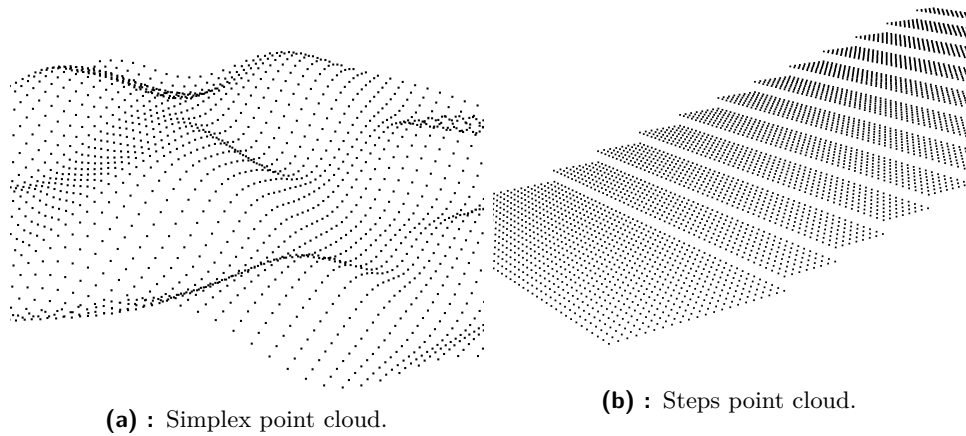


Figure 2.23: Generated point clouds.

In PyBullet, the terrain was represented as a concave collision shape. The point cloud was first converted to an OBJ file format, and this file was used to initialise the terrain.

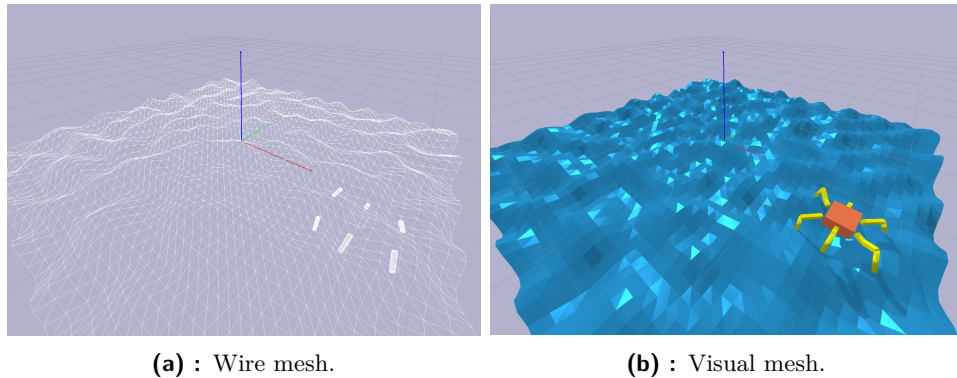


Figure 2.24: Terrain representation in PyBullet.

2.5 Training

Structured learnable algorithms *i.e.* LSA, SA+GP, SA+TH, and SA+LH were trained using sparse rewards in the form of the distance reached in the x-axis. The number of simulation and evaluation iterations differed from one terrain to another.

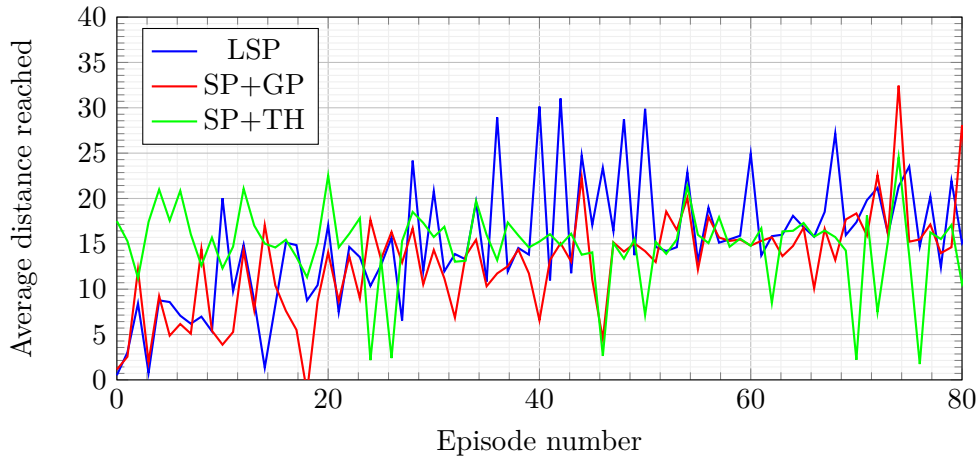


Figure 2.25: Reward during training on terrain (d).

2.5.1 Training pipeline

In this section, I explain how the training pipeline was structured.

Training was divided into episodes. At the beginning of each episode, the optimizer is asked for a certain number of solutions ⁸. After each episode, the optimizer performs an optimization step. During one episode, the requested potential solutions are evaluated. The evaluation is carried out by stepping the environment until it returns the information that the objective was reached (Section 2.4). The evaluation of a solution can be executed more than once based on the number of evaluation iterations ⁹, which can be useful when the observation contains noise. Gait-phases, torso height, and leg heights can be learned separately or together.

Algorithm 5 Training pipeline

```

1: obtain observation by initializing PyBullet environment
2: for episode = 1, 2, ..., N do
3:   for solution = 1, 2, ..., J do
4:     for evaluation iteration = 1, 2, ..., E do
5:       for simulation iteration = 1, 2, ..., S do
6:         calculate foot position from observation using DKT
7:         use policy to generate action from observation
8:         update leg desired positions
9:         calculate 18 leg joint angles using IKT
10:        get observation by stepping PyBullet simulation
11:      end for
12:    end for
13:  end for
14: end for

```

⁸Number of solutions could be set at the start of the training via the GUI

⁹Number of evaluation iterations can be set at the start of the training via the GUI

2.5.2 Optimization method

For optimisation, I used the Python Nevergrad optimisation library [20]. Nevergrad is a gradient-free optimisation platform that implements all kinds of evolution strategy (ES) optimizers. These optimizers can optimize continuous, discrete, or mixed parameters. In this thesis, the optimizer is used to update the continuous policy parameter vector θ . All of these optimizers are implemented with an ask-tell interface. I demonstrate the usage of the ask-tell interface in Algorithm 6.

Algorithm 6 Ask-Tell pipeline

```

1: for iteration = 1, 2, ..., N do
2:   Ask optimizer for potential solutions
3:   for every solution do
4:     Calculate loss of the solution
5:   end for
6:   Tell optimizer the calculated losses
7: end for

```

Some of the optimizers implemented in Nevergrad are:

1. Random Search
2. Particle Swarm Optimizer (PSO)
3. Covariance Matrix Adaptation Evolution Strategy (CMA-ES)
4. Hammersley Search

In this thesis, I used CMA-ES as it is considered state-of-the-art in evolutionary computation [21].

2.6 Evaluation

All evaluation scores were normalised by the number of evaluation steps and the number of simulation iterations because different kinds of terrain required a different number of simulation and evaluation iterations.

To evaluate performance, I used a score based on the distance reached on the x-axis, smoothness and power consumption.

2.6.1 Distance reached

This is the main score of the evaluation. LSA is constrained and performs well with regard to smoothness, therefore, distance is a good benchmark. If

the learned policy is not efficient, it will not be able to travel as far.

$$r_d = \frac{1}{SE} \sum_{i=1}^E d_i \quad (2.26)$$

where

- r_d is the reward for the distance reached,
- E is the number of evaluation iterations,
- S is the number of simulation iterations,
- d_i is the distance at the end of the evaluation iteration i .

2.6.2 Smoothness

Because the LUP is not directly constrained, the resulting locomotion can be rough. Smoothness is calculated by summing all accelerations over all simulation steps. This includes torso acceleration and torso angular acceleration. This score will always be less than or equal to zero. This means that the larger the value, the smoother.

Performance cannot be evaluated purely based on smoothness because if the policy outputs zero action, the smoothness value will be greater. Therefore, calculating smoothness per distance is more intuitive score.

The smoothness is calculated as follows:

$$r_s = -\frac{1}{r_d} \frac{1}{SE} \sum_{i=1}^E \sum_{j=1}^{S-1} \|\mathbf{a}_{ij} + c\boldsymbol{\epsilon}_{ij}\|^2 \quad (2.27)$$

where

- r_d is the distance reached (Equation 2.26),
- E is the number of evaluation iterations,
- S is the number of simulation iterations,
- $\mathbf{a}_i \in \mathbb{R}^3$ is the torso acceleration in step i ,
- $\boldsymbol{\epsilon}_i \in \mathbb{R}^3$ is the torso angular acceleration in step i ,
- $c \in \mathbb{R}$ is some constant that changes the ratio.

Because PyBullet does not support acceleration readouts, Equation 2.27 can be written using velocities $\mathbf{v}_i \in \mathbb{R}^3$ and angular velocities $\boldsymbol{\omega}_i \in \mathbb{R}^3$ as:

$$r_s = -\frac{1}{r_d} \frac{1}{SE} \sum_{i=1}^E \sum_{j=1}^{S-1} \left\| \frac{\mathbf{v}_{ij+1} - \mathbf{v}_{ij}}{t_{ij+1} - t_{ij}} + c \frac{\boldsymbol{\omega}_{ij+1} - \boldsymbol{\omega}_{ij}}{t_{ij+1} - t_{ij}} \right\|^2 \quad (2.28)$$

In this way, I calculated the smoothness of the trajectory in each simulation step.

■ 2.6.3 Power consumption

Since it is not a real model, the power consumption can be simplified. PyBullet does not simulate electric servomotors, so I had to use only the quantities available. PyBullet returns joint torques at each step. Because the servomotor torque is proportional to the current, the power consumption was defined as the sum of all the torques from all the simulation and evaluation steps. To make the power consumption score more intuitive, I divided it by the distance reached defined in Section 2.6.1. In this way, it says how efficient the locomotion is.

$$r_p = \frac{1}{r_d} \frac{1}{SE} \sum_{i=1}^E \sum_{j=1}^{S-1} \sum_{k=1}^{18} |\tau_{ijk}| (t_{ij+1} - t_{ij}) \quad (2.29)$$

where

- r_d is the distance reached (Equation 2.26),
- E is the number of evaluation iterations,
- S is the number of simulation iterations,
- $\tau_{ijk} \in \mathbb{R}$ is the k -th joint torque in simulation step j in evaluation iteration i .

2.7 Terrain set

Training and evaluation was carried out on several terrains. Terrains can be seen in Table 2.12.

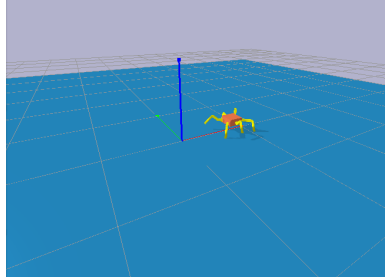
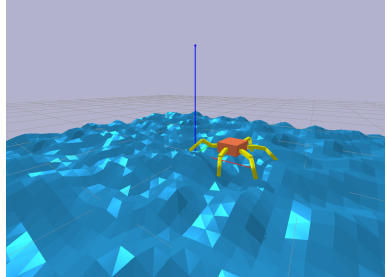
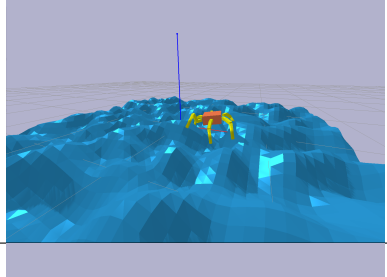
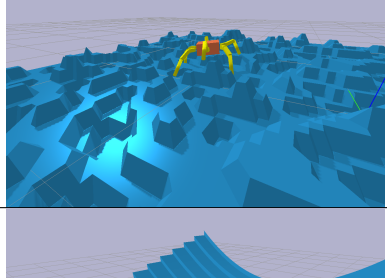
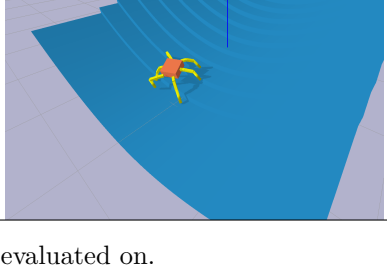
Label	Name	Description	Picture
(a)	Flat	A regular flat terrain.	
(b)	Bumps	This terrain features bumps with small height differences that should show how policies react compared to flat terrain.	
(c)	Big Bumps	Bumps on this terrain have more prominent height differences, some of them are as tall as the hexapod itself.	
(d)	Rocks	This terrain features sharp edges and is the roughest terrain.	
(e)	Stairs	This terrain features stairs that should show whether the policy is able to climb. To truly focus on the climbing aspect, the terrain is made into a U-shape to help the policies to keep forward orientation.	

Table 2.12: Terrains that the policies were trained and evaluated on.

Chapter 3

Results

In this chapter, I present the results of the trained algorithms introduced in Section 2.3. More in-depth analysis of the results can be found in the discussion in Chapter 4.

To further study the performance of the LSA, the individual NNs of the LSA were also evaluated separately to see which provided the greatest benefit to the locomotion. In that case, the SA was used and the particular control parameter of the movement generator was generated by the NN.

The following policies were trained, evaluated and compared:

- Learnable Unstructured Algorithm (LUA)
- Learnable Structured Algorithm (LSA)
- Structured Algorithm (SA)
- Structured Algorithm with learnable gait-phases (SA+GP)
- Structured Algorithm with learnable torso height (SA+TH)
- Structured Algorithm with learnable leg heights (SA+LH)

Policies were evaluated based on the following scores:

- Maximum distance reached in the x-axis. (D)
- Smoothness per distance (S_d)
- Power consumption per distance (P_d)
- Subjectively from a visual point of view.¹

The definitions for the first three individual scores can be found in Section 2.6. The terrains on which the policies were evaluated can be found in Section 2.7

¹Video: <https://youtu.be/xbi8Kqam6Nc>

3.1 Results on terrain (a)

First, all approaches were trained and evaluated on the flat ground to obtain a basic overview of performance. The results were evaluated using two speed values.

The results obtained with an speed equal to 0.3 can be seen in Table 3.1.

Policy	D	S_d	P_d
LUA	135.21	-339.17	61.88
LSA	32.58	-9.19	86.67
SA	28.58	-7.40	94.85
SA+GP	31.76	-11.66	86.65
SA+TH	29.41	-8.93	93.32
SA+LH	30.89	-14.14	96.19

Table 3.1: Comparison of all policies on terrain (a). Movement generator has speed set to 0.3. D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red indicates worst in class. Blue indicates best in class.

The results in Table 3.1 show that the LUA performed best with respect to the distance reached and power consumption. LUA was by a large margin the worst in smoothness. Visually, the LUA movements were very fast and shaky. The smoothest policy was the SA. From the structured algorithms, the LSA performed the best. Visually, the structure algorithms were very smooth, natural looking, and almost indistinguishable from each other.

Policy	D	S_d	P_d
LUA	135.21	-339.17	61.88
LSA	56.59	-68.52	75.54
SA	46.50	-36.65	84.64
SA+GP	62.01	-35.31	64.00
SA+TH	52.86	-30.50	74.81
SA+LH	54.25	-41.26	76.20

Table 3.2: Comparison of all policies on terrain (a). Movement generator had speed set to 1.0. D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class.

In this case, the structured algorithms moved faster, which had a positive effect on the distance reached and the power consumption per distance and a negative effect on the smoothness. Visually, there was a noticeable difference between the two speeds, but the locomotion looked natural.

3.2 Results on terrain (b)

Policy	D	S_d	P_d
LUA	50.46	-653.42	162.68
LSA	43.98	-98.36	97.75
SA	28.34	-87.13	143.19
SA+GP	37.24	-85.34	114.46
SA+TH	30.17	-118.68	137.81
SA+LH	28.34	-105.33	139.05

Table 3.3: Comparison of all approaches on terrain (b). D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class.

As can be seen in Table 3.3, the LUA performed the best with respect to the distance reached, but was the worst in smoothness and power consumption. The distances reached by the structured algorithms were similar, while LSA performed the best and SA and SA+LH performed the worst. LSA also had the smallest power consumption. From a visual perspective, the LUA has a problem in maintaining a consistent trajectory.

3.3 Results on terrain (c)

Policy	D	S_d	P_d
LUA	26.36	-839.74	318.13
LSA	32.03	-160.51	135.52
SA	20.83	-173.52	194.49
SA+GP	32.79	-168.71	132.49
SA+TH	20.64	-193.89	196.69
SA+LH	22.37	-174.81	184.01

Table 3.4: Comparison of all approaches on terrain (c). D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class.

This terrain has greater height differences than terrain (b). Some of them were as high as the hexapod itself. LSA and SA+GP had very similar performances, having the best in class results between them. The trajectories of the structured algorithms were more in accordance with the shape of the terrain than on the terrain (b). LUA had a hard time training on this terrain and had a problem maintaining a consistent trajectory.

3.4 Results on terrain (d)

Policy	D	S_d	P_d
LUA	15.31	-1906.96	643.16
LSA	23.03	-210.42	190.03
SA	16.31	-202.74	262.66
SA+GP	19.62	-228.30	236.77
SA+TH	17.11	-249.76	261.26
SA+LH	20.17	-164.65	192.82

Table 3.5: Comparison of all approaches on terrain (d). D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class.

This was the roughest terrain with very steep slopes and relatively sharp edges. The LUA had problems training on this terrain and had the worst results. The best results with regard to distance reached and power consumption had the LSA. The smoothest was SA+LH with not that bad results regarding distance and power consumption. Visually, all algorithms had a problem maintaining a straight trajectory.

3.5 Results on terrain (e)

Policy	D	S_d	P_d
LUA	9.37	-2374.00	818.58
LSA	8.24	-481.10	588.32
SA	6.97	-251.06	603.37
SA+GP	9.96	-219.38	419.85
SA+TH	6.98	-373.71	651.10
SA+LH	15.29	-139.04	259.58

Table 3.6: Comparison of all approaches on terrain (e). D is the distance reached, S is the smoothness per distance, P is the power consumption per distance. Red highlights worst in class. Blue highlights best in class.

Terrain (e) was designed to test the ability of the policies to climb. The results show that SA+LH performed the best in all aspects. Other approaches had a hard time. The worst performed the algorithms without the ability to control the leg lifting heights.

Chapter 4

Discussion

First, I compared all algorithms on flat ground to obtain an initial picture of performance. Data were collected using two speeds. (more about speed in Section 2.1.4)

I also wanted to verify several assumptions:

1. SA will be smoother and more power efficient than LUA.
2. SA+GP will perform about as well as SA or slightly worse.
3. SA+TH will perform about the same as the SA.
4. SA+LH will perform better than SA because the optimal leg height on flat terrain is as low as possible but not too low to drag the feet.

In the first assumption, I was able to verify the fact that SA was smoother than LUA, however, according to Table 3.1, the LUA was approximately 50 % more efficient than SA with speed equal to 0.3. The assumption was that faster movement would require larger accelerations in servomotors, which would increase the power consumption. This was not the case because the SA is constrained by the movement generator and with 0.3 speed the movement of the legs is very slow compared to the LUA. The servomotors have to produce torque to keep the legs off the ground. The limit example of this would be that when the hexapod is standing still, the servomotors have to keep the hexapod standing up, but the hexapod is not generating any movement, so the power per distance increases to infinity. As can be seen in Table 3.2, when the speed is set to 1.0, the power per distance decreases, but at the cost of smoothness. This argument is also supported by the results of the algorithms that use the movement generator. The speed characteristic is shown in Figure 2.7.

I did not verify the second assumption because the SA+GP managed to outperform the SA. In the case of the speed equal to 1.0, it even outperformed all other structured approaches by a great margin, as can be seen in Table 3.2. The reason behind it was that with increasing speed, smoothness decreased, and slippage and other negative factors became more prominent. SA+GP was

able to learn to eliminate these factors by transitioning between gait-phases more appropriately.

The third assumption was also incorrect. Changing the torso height on flat terrain had a similar effect because SA+TH learned to mitigate negative factors.

The fourth assumption was verified. SA+LH was able to improve the movement generator. This approach also had a lower smoothness. It was because during locomotion some legs touched the ground.

Training on more rough terrains (b) and (c) showed that LUA had a hard time learning consistent locomotion. This was due to the lack of exteroceptive data and the large number of parameters.

The results obtained on terrain (d) show that on the roughest terrain the LUA performance dropped below all structured approaches. The causes of this were lack of exteroceptive data and slow training. Unlike terrains (b) and (c) this terrain had really prominent height differences. It was very easy for the legs to get stuck. SA+TH did not show good results despite the fact that the height of the torso was an important aspect on this terrain. This was because the hexapod got stuck more on smaller obstacles due to the low lifting height. This terrain required complex adaptation in all aspects, therefore, the best performance can be seen from LSA.


Results show that with increasing roughness, the distance reached decreased, the smoothness per distance decreased, and the power consumption per distance increased. This was expected.

The results obtained on terrain (e) show that SA and SA+TH were unable to climb the stairs. There were three reasons for this. The first reason was that the movement generator sampled the point cloud using the heightmap. The resolution around sharp edges, such as stairs, was not high enough to correctly approximate the terrain. This caused a large number of slippages and bad gait transitions of SA. SA+GP was able to slightly improve performance by adapting the gait, which addressed the problem stated above. The second reason was that the movement generator did not implement logic to avoid placing feet on the edges. The third reason was that the default value of leg lifting height was too small to reliably place the foot on the stairs. This third reason was addressed by SA+LH, as can be seen in Table 3.6, which significantly improved performance.

LSA did not perform as well as SA+LH, even though it has the same ability. This was because a higher number of parameters resulted in slower learning. LSA would have been able to increase its performance even further if it was given a longer time to train.

This shows that by learning certain aspects of the movement generator, performance can not only be improved, but even new use cases can be created.

In general, LSA and all its variants were able to perform better than SA, even beyond expectations. From a visual perspective and based on the results, the structured algorithms performed substantially better with respect to smoothness, leading to more consistent locomotion and trajectories. Structured algorithms also trained faster and did not require reward shaping to achieve these results.



Chapter 5

Conclusion

In this thesis, I designed a learnable structured algorithm (LSA) for generation of hexapod locomotion consisting of a neural network that calculates control parameters for a manually-designed movement generator. The control parameters were gait-phase transitions (GP), torso height (TH), and leg lifting heights (LH). The movement generator used exteroceptive terrain data in the form of a point cloud for foot placement. A feedback loop was designed to compensate for slippage and other factors.

For comparison I implemented a structured algorithm (SA), which was a variant of LSA in which I replaced the neural network with a hand-designed controller to generate the control parameters. The controller generated a basic tripod gait and constant torso height and leg lifting heights. To further study the performance of LSA, three more variants were created. Each variant implemented either GP, TH or HL. The remaining control parameters were generated by the controller created for SA. I also implemented a learnable unstructured algorithm (LUA) that was represented by an end-to-end neural network.

The parameters of the neural networks were optimized using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES).

As a physics simulator, I used PyBullet. In total, six algorithms were evaluated on five custom terrains. The algorithms were evaluated based on the distance reached in the x-axis, smoothness per distance, power consumption per distance, and subjectively based on appearance. The definitions of scores can be found in Section 2.6.

The LUA performed best on smoother terrains. It was able to generate very fast and power-efficient locomotion, but at the cost of smoothness and maintaining the trajectory to the point where it would make it impractical or even impossible to use on a real platform. The main limitations were the time-consuming reward shaping and parameter optimization using the black-box evolution strategy method. More reward shaping would have been necessary to achieve smoother motion with smaller joint velocities. Further-

more, carefully choosing only the most useful observation, thus reducing the number of parameters would also speed up the learning process.

The results show that the proposed LSA performed better than SA on all terrains. The structured approaches were much smoother and more natural looking than LUA. They did not require reward shaping and were able to learn from sparse rewards. Because the movement generator constrains the motion of the LSA, it would be safe to use on a real platform even with zero training. LSA was able to learn basic locomotion even during the first few iterations. The SA failed on stairs due to the low default leg lifting height and ineffective GP transitions and was outperformed by the SA+LH which involved learning the leg lifting heights. This shows that not only the performance can be improved, but even new use-cases can be enabled by the proposed method.

The assignment was satisfied, except that the optional deployment on a real platform could not be performed due to lack of time.

Probably the biggest limitation in this thesis was the optimization method. It would be worth trying reinforcement learning instead of evolution strategy optimizers in future research. If the learning process was faster and more efficient, there would be a lot of room for training other aspects of the movement generator, such as:

- Torso orientation relative to the terrain,
- Leg range,
- Center point position for each leg,
- Foot placement using heightmap as low-resolution image.

Other improvements could be made to the movement generator. Because the proposed solution is limited by the movement generator, there may be a better line between robustness and versatility.

Future work could comprise the implementation of the solution described in [11]. This would make it possible to switch between trained policies based on terrain estimation.



Bibliography

- [1] D. J. Todd, *Walking machines: an introduction to legged robots*. Springer Science & Business Media, 2013.
- [2] M. H. Raibert, “Legged robots,” *Communications of the ACM*, vol. 29, no. 6, pp. 499–514, 1986.
- [3] M. Zangrandi, S. Arrigoni, and F. Braghin, “Control of a hexapod robot considering terrain interaction,” *CoRR*, vol. abs/2112.10206, 2021.
- [4] A. J. P. A. Chávez and J. H. A. Alcántara, “Kinematic and dynamic modeling of the phantomx ax metal hexapod mark iii robot using quaternions,” in *2021 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pp. 595–601, 2021.
- [5] M. Bjelonic, N. Kottege, and P. Beckerle, “Proprioceptive control of an over-actuated hexapod robot in unstructured terrain,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2042–2049, IEEE, 2016.
- [6] P. Cizek, D. Masri, and J. Faigl, “Foothold placement planning with a hexapod crawling robot,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Vancouver, BC, Canada), pp. 4096–4101, IEEE, 2017.
- [7] A. A. Karim, T. Gaudin, A. Meyer, A. Buendia, and S. Bouakaz, “Procedural locomotion of multilegged characters in dynamic environments,” *Computer Animation and Virtual Worlds*, vol. 24, no. 1, pp. 3–15, 2013.
- [8] R. Campos, V. Matos, and C. Santos, “Hexapod locomotion,” in *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, (Glendale, AZ, USA), pp. 1546–1551, IEEE, 2010.
- [9] L. Minati, M. Frasca, N. Yoshimura, and Y. Koike, “Versatile locomotion control of a hexapod robot using a hierarchical network of nonlinear oscillator circuits,” *IEEE Access*, vol. 6, pp. 8042–8065, 2018.

- [10] H. Hu and Y. Liu, “Blind adaptive gait planning on non-stationary environments via continual reinforcement learning,” in *2021 IEEE International Conference on Unmanned Systems (ICUS)*, pp. 280–284, 2021.
- [11] T. Azayev and K. Zimmerman, “Blind hexapod locomotion in complex terrain with gait adaptation using deep reinforcement learning and classification,” *Journal of Intelligent & Robotic Systems*, vol. 99, no. 3, pp. 659–671, 2020.
- [12] A. S. Lele, Y. Fang, J. Ting, and A. Raychowdhury, “Online reward-based training of spiking central pattern generator for hexapod locomotion,” in *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, pp. 208–209, 2020.
- [13] J. Vice, G. Sukthankar, and P. K. Douglas, “Leveraging evolutionary algorithms for feasible hexapod locomotion across uneven terrain,” *arXiv preprint arXiv:2203.15948*, 2022.
- [14] D. Belter and P. Skrzypczyński, “A biologically inspired approach to feasible gait learning for a hexapod robot,” 2010.
- [15] T. Wang, R. Liao, J. Ba, and S. Fidler, “Nervenet: Learning structured policy with graph neural networks,” in *International Conference on Learning Representations*, 2018.
- [16] M. Schilling, K. Konen, F. W. Ohl, and T. Korthals, “Decentralized deep reinforcement learning for a distributed and adaptive locomotion controller of a hexapod robot,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5335–5342, IEEE, 2020.
- [17] P. Corke, “Denavit-hartenberg notation for common robots,” *PeterCorke: Brisbane, Australia*, 2014.
- [18] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, “State-of-the-art in artificial neural network applications: A survey,” *Heliyon*, vol. 4, no. 11, p. e00938, 2018.
- [19] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning.” <http://pybullet.org>, 2016–2021.
- [20] J. Rapin and O. Teytaud, “Nevergrad - A gradient-free optimization platform.” <https://GitHub.com/FacebookResearch/Nevergrad>, 2018.
- [21] I. Loshchilov and F. Hutter, “CMA-ES for hyperparameter optimization of deep neural networks,” *CoRR*, vol. abs/1604.07269, 2016.