UNIVERSITY OF CAPE TOWN

Department of Mechanical Engineering

RONDEBOSCH, CAPE TOWN, SOUTH AFRICA

FINAL REPORT 2020

PROJECT
No: 68

ONLINE GAIT ADAPTATION OF THE RARL HEXAPOD

by
MLRCHR001
Christopher Mailer

Project Brief

Humans and animals can rapidly adapt to unforeseen changes in their environments and the way they interact with them. If a limb is injured, walking style will immediately change to apply less pressure to that limb, while still ensuring we travel in the desired direction. This will occur naturally and in a matter of minutes even if we have never experienced a compromised limb before. The same goes for navigating slippery or uneven ground.

Robots, however, typically have rigid behaviours designed for specific scenarios. This allows them to excel in those scenarios, but also leads to failure when presented with unexpected changes such as a jammed actuator, bent joint, or different terrain, ultimately requiring human intervention. This poses a significant issue for remote or hazardous environments where human intervention is not possible. For robots to succeed in the real world, the ability to quickly and effectively adapt to changes in their environment is essential.

This project aims to investigate current means of online gait adaption in robotics and implement this system in the RARL Hexapod, with the goal of allowing it to adapt to servo failure or varying terrain.

Supervisor: Ms Leanne Raw

Co-supervisor: Dr Geoff Nitschke

Word Count: 18594

Plagiarism Declaration

Plagiarism Declaration

I, Christopher Mailer, declare that;

i. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.

ii. I have used the IEEE. convention for citation and referencing. Each significant contribution to, and quotation in, this report / project from the work(s) of other people has been attributed and has been cited and referenced.

iii. This report/project is my own work.

iv. I have not allowed and will not allow anyone to copy my work with the intention of passing it off as his or her own work.

Christopher Mailer
November 13, 2020

Acknowledgements

Throughout this project I have received a tremendous amount of support from people to whom I am truly indebted.

First and foremost I would like to thank my supervisor, Ms Leanne Raw for the unwavering support and mentorship throughout this project and this tumultuous year. Your assistance and encouragement to pursue tests on the physical platform resulted in invaluable practical robotics experience, and without your initial development of the hexapod platform this project would certainly not have been possible.

I would like to thank my co-supervisor, Dr Geoff Nitschke for the insight and guidance in developing this project, and for introducing me to a truly fascinating field within robotics. Furthermore, this project would not have been possible without the incredible computing resources from your research programme.

I would also like to thank my friends, Daniel Ferrini, Brandon Reabow, and Joel Philpott for the laughs, motivation, and thought provoking discussions.

Most importantly, I would like to thank my parents, Dana and Robin Mailer for their unbounded love, guidance, and support throughout my life and education. I am eternally grateful.

Abstract

The RARL hexapod at the University of Cape Town is a developed robotic platform intended for search and rescue missions. The currently implemented tripod gait speed drops by 60% with the failure of 1 leg, even with 5 functional legs still remaining. This is not conducive to mission success. The aim of this project was therefore to develop and implement a system which would enable the RARL hexapod to adapt its gait and retain mobility when faced with the failure of any of its legs. The Intelligent Trial and Error (IT&E) algorithm, comprising of MAP-Elites followed by Map-based Bayesian Optimisation, was identified from literature as an approach best suited for the unique hardware constraints of the RARL hexapod platform. To implement this approach a dynamics simulation of the hexapod was developed with the PyBullet physics engine which could be used by the MAP-Elites evolutionary algorithm. In producing this simulation an accurate model of the hexapod was also developed along with a custom kinematic gait controller which could produce a wide variety of gaits. MAP-Elites was modified to run on a multi-node computing cluster, and the diverse collection of gaits, known as behaviour-performance maps, were evolved on the South African Lengau high performance computing cluster using the dynamics simulator. Ten independent behaviour-performance maps were generated to evaluate experimental reproducibility which required 400 hours of computing time in total. Four failure scenarios of increasing complexity involving 1 and 2 fully failed legs were defined to evaluate the performance of adaptation against the default tripod gait currently implemented on the hexapod robot. Adaptation was tested in simulation with 210 independent experiments as well as on the real platform with 6 independent experiments. In both simulation and in reality the adapted gait was significantly faster than the default tripod gait with and without leg failures, and occurred on average in under 2 minutes with a maximum duration of 3 minutes. These results matched those of previous research. Gait adaptation with IT&E on the RARl hexapod was successful and reproducible. This project validates the use of the gait adaptation algorithm on the RARL hexapod, and thereby also validates the applicability of IT&E with MAP-Elites to different legged robotic platforms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Robots in the Field

Robots have the ability to operate in distant or hostile environments which would otherwise be entirely inaccessible, or pose a severe risk to humans [1]. Activities in these environments can range from search and rescue operations, disaster response, to exploration and gathering of scientific data. An early example of this is a legged robot named Dante II, developed by The Robotics Institute at Carnegie Mellon University in 1994, which rappelled into the previously unchartered caldera of the active volcano Mt. Spurr in Alaska, collecting data about the volcanic gas composition whilst researchers and operators were safely located over 120 km away, depicted in Figure 1.1 [2].

Figure 1.1: Photograph of the legged robot Dante II rappelling down a steep slope into the volcano Mt. Spurr adapted from [2]

Since 1997, 4 rovers (Sojourner, Spirit, Opportunity, and Curiosity) have been sent to explore the surface of Mars and gather scientific data. Opportunity was operational for 15 years, and Curiosity is currently still operational [3]. The first use of search and rescue robots was on the 11th of September 2001 during the terrorist attacks on the World Trade Center. The robots assisted rescue personnel in locating and saving the lives of 10 people [4]. Robots are also being used to find and remove nuclear waste from the unit 2 reactor at the Fukushima Daiichi power plant after the meltdown on the 11^{th} of March 2011 [5].

1.1.1 Challenges of Robots in the Field

Distant or hostile environments present a unique set of challenges for robots. Wireless communication is often limited with intermittent connectivity or low bandwidth [1]. Autonomy is therefore essential in these field robots to quickly respond to changes in their environments

without waiting for intervention from a human operator [3]. Even without communication delays, a strategy relying solely on human operators has to deal with the issue of human operator fallibility [6].
A lack of hardware reliability and robustness has also been identified as a significant issue for robots in the field [7]. Reliability is particularly pertinent as repair is often not an option [2]. The failure rate for components in harsh and uncertain environments is also relatively high [6][8]. It is therefore essential that robotic systems be developed which can cope with the unexpected failure of a component.

1.2 Project Description

The University of Cape Town's Robotics and Agents Research Lab (RARL) has developed a hexapod robot for potential use in search and rescue scenarios [9][10][11][12]. The RARL hexapod has no means of detecting or mitigating the failure of one of its 18 servos, resulting in it being rendered immobile in the event of failure. Robots in industry are limited to preprogrammed failure mitigation strategies, however these become infeasible for complex systems and are largely sensor dependent for fault diagnosis, adding cost [7]. This project aims to overcome this fault tolerance limitation by implementing automatic adaptation of the RARL hexapod's gait to retain mobility in the event of a servo failure. There are currently two state-of-the-art approaches to online adaptation, namely Intelligent Trial and Error (IT&E) [13] and Gradient Based Adaptive Learner (GrBAL) [14]. Both of these approaches have been tested on physical robots and are applicable to the RARL Hexapod. For adaptation to be evident, the robot will need to remain mobile and maintain a faster walking speed than a standard tripod gait [15] with a servo failure, for a fixed gait frequency. This adaptation will also need to occur in a time frame of minutes for it to be applicable to real-life scenarios such as search and rescue missions.

1.3 Project Objectives

In order for this online gait adaptation to be successful, first and foremost it needs to result in a gait which performs better than the originally deployed gait with the failure. The performance of a gait is typically based on the speed of the hexapod body[13][14] with the deployment gait commonly being the tripod gait[15][12]. Secondly this adaptation needs to occur quickly as deployment scenarios are typically time critical [5]. The existing approaches on average take less than 2 minutes to achieve better performance than the reference gait for a variety of failures[13][16][14]. Similar or better adaptation time than the existing approaches is desired. Finally the approach needs to be deployable on the current RARL Hexapod[12], making use of the existing sensors, communication channels, and gait controller.

The objectives for this project to be successful are therefore;

- The adapted gait must be faster than the reference gait (tripod gait) with the failure
- The adaptation process must occur in under 2 minutes
- The algorithm must work with the available sensor feedback data
- The algorithm must integrate with the existing RARL hexapod platform

Chapter 2

Literature Review

2.1 Legged Robots

Only about half of the earths surface is accessible to current wheeled and tracked vehicles, whilst a much greater fraction is accessible to legged animals [17]. This is due to wheeled vehicles requiring continuous paths of support with the ground, presenting significant challenges on rough unprepared terrain where surface discontinuities will result in a loss of support and traction [17]. Legged vehicles however only require isolated footholds. The ground between these footholds does not need to be prepared as it can simply be stepped over, presenting an enormous advantage in navigating unprepared terrains [17]. Furthermore legs decouple the motion of the feet and the body, allowing for body stability over rough terrain [17].

2.1.1 Stability

The stability of legged robots can be divided into static stability and dynamic stability. Static stability is achieved when the centre of mass of the robot is within the vertical projection of the polygon created by the contact points of the supporting legs, called the support polygon. A minimum of three supporting legs is required to achieve static stability. Animals occasionally exhibit statically stable gaits when moving slowly. Dynamic stability is achieved through continuous motion and active balancing. This allows for gaits which are no longer restricted to always having 3 feet on the ground around the centre of mass. Animals and humans commonly exhibit dynamic stability. [17]

2.1.2 Locomotion

Locomotion in a legged robot is achieved through a sequence of support and swing phases of the feet as shown in Figure 2.1.

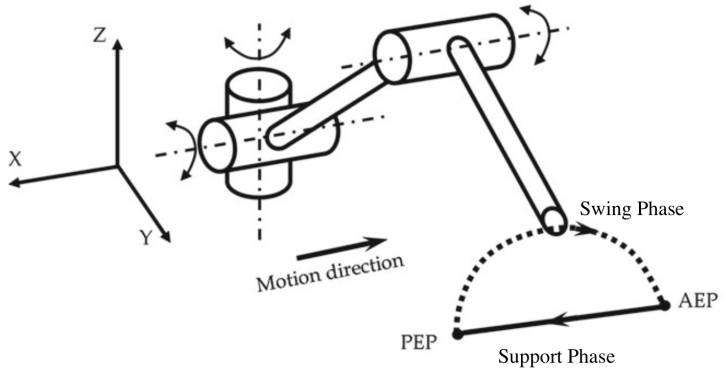


Figure 2.1: Kinematic schematic of leg phases during locomotion adapted from [18]

During the support phase the foot is in contact with the ground producing the reaction force required to support and propel the robot forwards. The foot remains in the same place on the ground and moves relative to the base from the anterior extreme position (AEP) to the posterior extreme position (PEP) as shown in Figure 2.1. In the swing phase the foot is lifted and moved to the next foothold, and the cycle is repeated. [18]

2.2 Hexapod Robots

2.2.1 Morphologies

Hexapod robots are a class of legged robots with six legs. Hexapod morphologies can be grouped into rectangular and circular. Rectangular hexapods are inspired by insects and have 3 legs positioned on either side of the sagittal plane. This morphology exhibits the best walking performance in the direction parallel to the sagittal plane. Circular hexapods have the legs evenly distributed around the body and exhibit similar walking performance in all directions. [15]

Table 2.1: Advantages and disadvantages of 6 legged robots compared to robots with a fewer or greater numbers of legs, adapted from [15]

Advantages	**Disadvantages**
• Statically stable during walking • Efficient during stable walking • Multiple gaits • Legs can be used for manipulation tasks • Motion flexibility • Redundancy	• Complex locomotion and control • Typically slower speed than robots with fewer legs • Expensive due to complex design, actuation, and control systems

2.2.2 Gaits

As a result of having 6 legs, hexapod robots can exhibit approximately 40 million leg phase sequence permutations [18]. Consequently, gait synthesis presents a uniquely complex problem that depends on a number of factors such as terrain type, stability requirements, speed requirements, mobility requirements, power requirements, and potentially also leg failures [18]. Gaits can be classified into periodic and non-periodic. Periodic gaits repeat the same sequence of steps every cycle. For multi-legged robots, coordination of all of the legs is essential [18]. For periodic gaits this is characterised by the phase offset of a legs cyclic motion relative to a reference leg. Periodic gaits can be further divided into regular, and irregular gaits. Regular gaits are when each leg spends the same amount of time per cycle in the support phase; that is they have the same duty factor (β) [18].

$$\beta = \frac{t_{support}}{t_{cycle}} \tag{2.1}$$

Figures 2.2,2.3, and 2.4 are statically stable regular periodic gaits typically implemented on hexapod robots and often observed in nature [15]. Gaits are best described with footfall diagrams where the white blocks represent the leg support phase and the black blocks represent the swing phase, with time progressing from left to right. They are a concise way of visualising both the duty cycles and phase offsets of each of the legs.

Tripod Gait

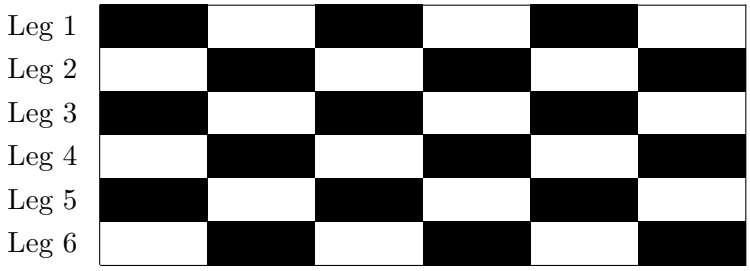


Figure 2.2: Footfall diagram for 3+3 tripod gait

In Figure 2.2 three legs have the same phase offset relative to leg 1. Additionally, three legs are on the ground at all times meeting the minimum requirement for static stability.

Quadruped Gait

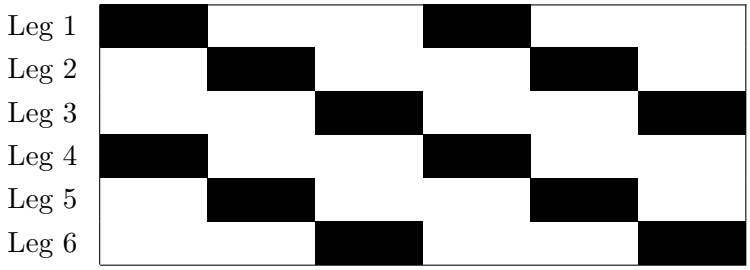


Figure 2.3: Footfall diagram for 4+2 quadruped gait

In Figure 2.3 two legs have the same phase offset relative to leg 1. Additionally, four legs are on the ground at all times.

Wave Gait

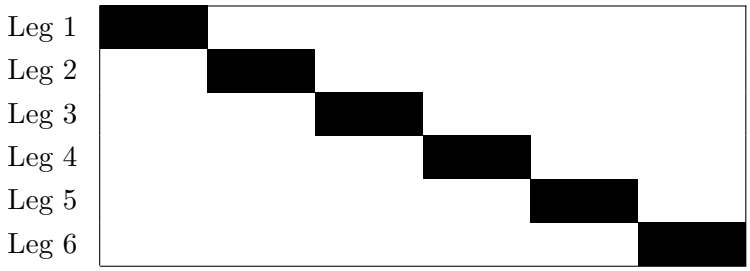


Figure 2.4: Footfall diagram for 5+1 wave gait

In Figure 2.4 each leg has a different phase offset relative to leg 1. Additionally, five legs are on the ground at all times providing the maximum stability for a hexapod robot during locomotion. The noteworthy characteristics of each of these gaits are summarised in Table 2.2 for comparison.

Table 2.2: Summary of common hexapod gait characteristics [15]

Gait	**Duty Cycle (β)**	**Support Legs**	**Speed**
Tripod	1/2	3	Fast
Quadruped	2/3	4	Moderate
Wave	5/6	5	Slow

2.3 Simulation in Robotics

Simulation is an integral part of robotics as it allows for the quick and cheap testing of a robot without it ever needing to physically exist; avoiding the possibility of injury, damage, and unnecessary design changes [19]. Simulators can often perform the required physics computations faster than real-time also allowing for movements to be evaluated faster than if they were performed on the actual robot [20]. There is a growing need for model-based approaches to control and optimisation due to robots operating in increasingly complex and uncertain environments, rendering hand-designed controllers insufficient [21].

2.3.1 Contact Dynamics

The early robotics simulators initially did not include means of computing the dynamics of contacts. This was severely limiting for legged robotics as the contacts between their feet and the ground are the primary means of interaction with the environment and locomotion [21]. Solving the contact dynamics is a computationally complex with no analytical solution due to its non-convexity and discontinuity [22]. This discontinuity can be seen in figure 2.5. The

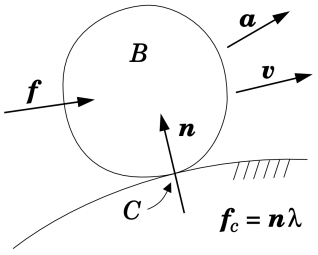


Figure 2.5: Generalised contact dynamics [22]

reaction force (n) and friction force (f_c) are only present when the two rigid bodies are in contact.

$$f_c = \begin{cases} n\lambda & n > 0 \\ 0 & n = 0 \end{cases} \tag{2.2}$$

Modern simulators overcome this by approximating the coulomb friction cone, using a variety of numerical methods [23], and by applying temporal discretisation [22]. The accuracy of the simulation can therefore be increased by decreasing the time-step, however for a given duration this requires more computations, increasing the runtime of the simulation. This presents a trade-off for applications requiring both fast and accurate simulations. Additionally the bounds of rigid bodies are typically approximated with a convex hull volume. This is so that the Gilbert-Johnson-Keerthi (GJK) algorithm [24] can be applied to test for a collision.

2.3.2 The Reality Gap

There are typically discrepancies between simulations and reality as a result of the need to approximate complex physical phenomena (eg: gear backlash, torque characteristics, sensor delay, object deformation, contact dynamics, etc.). This often results in controllers developed in simulation not working when transferred to a real robot [25]. This has presented a significant challenge in robotics fields reliant on simulation and is referred to as the "Reality Gap". There are numerous approaches to reducing the reality gap, however a first step is to increase the accuracy of the simulation and better model components such as actuators [26]. Although, the extent to which this can happen is limited by the corresponding increase in computation time [20].

2.3.3 Physics Engines

In a simulator the physics engine is responsible for computing the dynamics and contacts. The equations governing rigid body motions a fundamental part of the computations being done by a physics engine [22].

$$\vec{F} = m\vec{a} \tag{2.3}$$

$$\vec{T} = \boldsymbol{I}\vec{\alpha} \tag{2.4}$$

A physics engine integrates these fundamental equations to obtain the velocities and displacements of the rigid bodies. Accurate mass and inertia properties are therefore essential to the accurate modelling of the dynamics of a system. Systems in reality are continuous, however continuous computations are not possible. Instead the system is discretised into distinct time-steps with the state of the system being computed sequentially at each of these time-steps. A physics engine can therefore be thought of as a continuous loop. The components of this continuous loop are shown in figure 2.6.

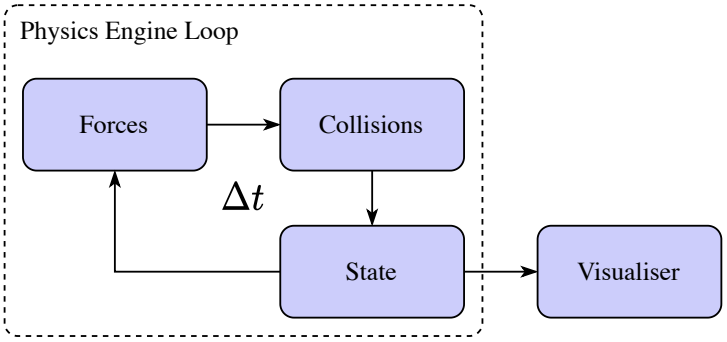


Figure 2.6: Physics engine continuous loop

For each iteration of the loop the physics engine computes the forces, contacts and resulting state for a time-step (Δt). Each iteration of the loop is commonly referred to as a "step". Once the state has been computed for a simulation step this state is typically sent to a visualiser to be displayed to a user. [19]

There are a number of physics engines which support rigid body dynamics, collision detection, and are able to load articulated bodies. Popular state-of-the-art physics engines commonly used in robotics [27][28] are listed in table 2.3.

Table 2.3: Summary of state-of-the-art physics engines for robotics

Specification	**Bullet** [29]	**ODE** [30]	**MuJoCo** [31]	**DART** [32]
Release Date	2006	2001	2015	2012
Author	E. Coumans	R. Smith	E. Todorov	J. Lee et al.
License	Zlib	GPL/BSD	Proprietary	BSD
Language	C/C++	C++	C	C++
API	C++/Python	C	C	C++
Visualiser	Yes	Yes	Yes	Yes
Contacts	Hard/Soft	Hard/Soft	Soft	Hard
Solver	MLCP	LCP	Newton/PGS/CG	LCP
Integrator	Euler	Euler	Euler/RK4	Euler
Coordinates	Minimal	Maximal	Minimal	Minimal

MLCP - Mixed Linear Complementary Problem
LCP - Linear Complementarity Problem
PGS - Projected Gauss-Seidel
RK4 - Fourth Order Runge-Kutta
CG - Conjugate Gradient

2.4 Existing Online Adaptation Approaches

Typical approaches to handling failures in robots involve detecting the failure with sensors and implementing the corresponding preprogrammed contingency plan [33]. Prior work has been done on designing fault tolerant gaits for hexapod robots and fault detection methods, however these are body morphology specific and limited to a maximum failure of two adjacent legs [34]. Hand designing specific gaits for each failure scenario is however not a robust approach for an 18 DOF hexapod due to the vast number of failure combinations which could occur, and does not guarantee an optimal solution [21]. There are currently two state-of-the-art approaches to the problem of online adaptation in legged robotics.

2.4.1 Gradient-Based Adaptive Learner

Overview

The Gradient-Based Adaptive Learner (GrBAL) approach [14], developed by the Berkeley Artificial Intelligence Research (BAIR) Lab uses meta-reinforcement learning (Meta-RL) to simultaneously train an agent to both walk and rapidly adapt its dynamics model to match changes in its environment. Reinforcement Learning (RL) is one of three subclasses of machine learning where an agent learns what sequence of actions to take in an environment to maximise some notion of a cumulative reward [35]. For a walking robot this reward could be the distance travelled or the walking speed. This can be formalised mathematically as a Markov Decision Process shown in Figure 2.7.

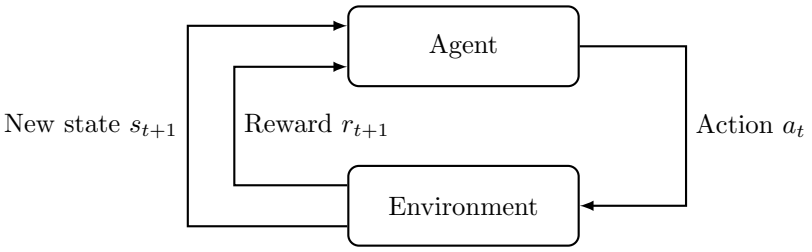


Figure 2.7: Reinforcement learning agent-environment interaction diagram [35]

For the agent in Figure 2.7 to be successful, it needs to choose actions (a_t) to take in the environment which result in the greatest reward (r_{t+1}). Model-based RL accomplishes this in part by learning the probability of an agent transitioning to a new state (s_{t+1}) if a specific action (a_t) is taken in the current state (s_t), referred to as the dynamics model $p(s_{t+1}|s_t, a_t)$ [35]. To learn the dynamics model millions of different state and action sequences are required as learning is typically achieved with a slow gradient based approach. Consequently simulation is an integral part of RL.

The GrBAL approach [14], tackles the problem of gait synthesis using model-based RL and a simulation of the robot. This results in the development of an optimal gait without relying on hand-designed controllers. The performance of model-based RL is highly dependent on the accuracy of the learned dynamics model as it is used when deciding which actions to take [35]. In GrBAL the dynamics model is approximated with an artificial neural network (ANN). This allows it to be fine-tuned using sensor data whilst the robot is operating, thereby enabling online adaptation. Typically performing stochastic gradient descent (SGD) to fine-tune an already trained ANN is however very sample inefficient. GrBAL overcomes this by not only training the dynamics model to accurately predict state transitions, but by simultaneously training it to also be fast at adapting given recent data. This dual objective is trained for by selecting a small subset of consecutive (M+K) state transitions from prior simulated experience, using the first half to update the dynamics model weights, and then optimising these weights to be

good at predicting the remaining state transitions. During training time random environmental variations are introduced such as actuator failures or different terrains to ensure adaptation is required between the M and K state transitions. These variations do not need to exactly match a failure experienced on the physical robot as the approach allows for generalisation. Once the dynamics model has been trained it is then transferred to a physical robot where recent sensor data is continuously used to fine-tune it to the current environment as shown schematically in Figure 2.8. If a failure occurs, mismatch between model predictions and actual sensor data are used to readjust the weights of the model with SGD and backpropogation to once again result in accurate predictions and optimal performance. Adaptation can therefore occur each time new sensor data is available, which is often hundreds of times a second, resulting in adaptation times of under one second. [14]

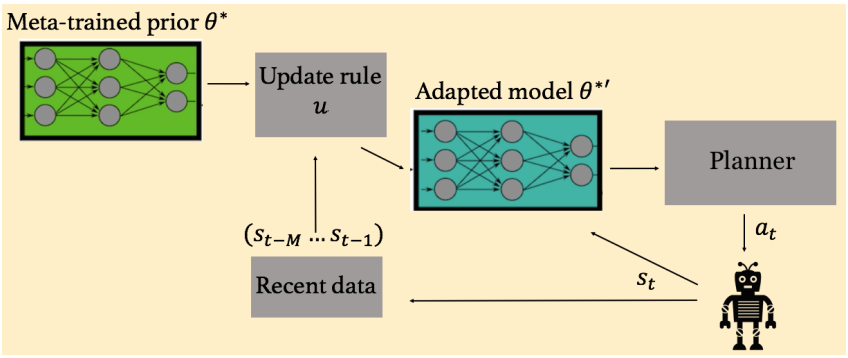


Figure 2.8: Diagramatic representation of GrBAL online adaptation of prior trained ANN [14]

Implementation

This approach was tested on a 2 degree-of-freedom hexapod robot depicted in Figure 2.12, enabling it to successfully adapt to a broken leg, novel terrains and slopes, pose miscalibration, and unbalanced payloads. The sensory feedback of the robots state was a 24-dimensional vector composed of centre of mass positions and velocities, centre of mass pose and angular velocities, back-EMF readings of motors, encoder readings of leg motor angles and velocities, and battery voltage. This sensor feedback was used to update the dynamics model of the robot used in the RL algorithm, thereby driving adaptation. The dynamics model was a neural network with 3 hidden layers of 512 units each and ReLU activations. Tests were also done on multiple simulated agents including a 4 legged, 12 degree-of-freedom ant robot which successfully adapted to crippled legs. [14]

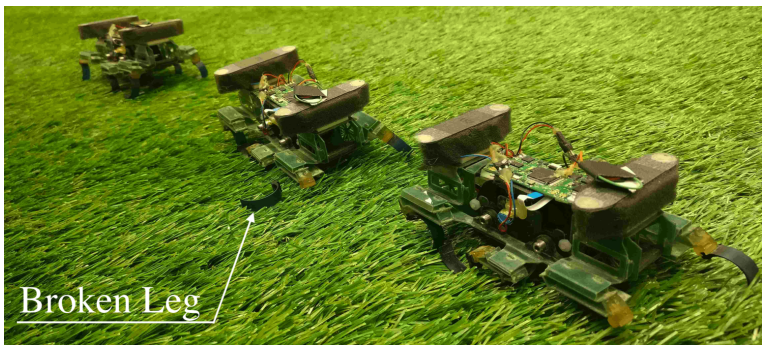


Figure 2.9: Image of the Millirobot used to test GrBAL [14]
Note the failed front right leg

2.4.2 Intelligent Trial-and-Error

Overview

The Intelligent Trial-and-Error (IT&E) algorithm [13], developed by the ResiBots research group, centres around a quality diversity (QD) algorithm, a subclass of evolutionary algorithms, called the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) [36]. Evolutionary computing is a diverse class of global optimisation algorithms inspired by natural processes such as genetic mutation and survival of the fittest [37]. Quality diversity algorithms are optimisation algorithms, however instead of finding a single global optimum, they find a diverse set of optimal niches.

Before a mission the MAP-Elites optimisation algorithm is run for millions of iterations on a simulation of the robot with varying gait parameters, resulting in a diverse and high performing set of possible gaits. The performance of the gaits is described by the walking speed in the simulation, and gaits are distinguished by the proportion of time each foot spends on the ground. Consequently MAP-Elites finds the optimal gait parameters for walking with 6 legs, 5 legs, 4 legs, etc. and a range in between. These gait parameters and their expected performance from simulation are stored in a behaviour-performance map. During the mission, if the robot detects that the current gait is no longer performing as expected, possibly due to actuator failure, it starts exploring alternative gaits in the map to find the new best one. This process is guided by a Bayesian Optimisation algorithm with the behaviour-performance map as a prior. Bayesian optimisation is another global optimisation algorithm typically used for noisy expensive-to-evaluate black-box functions in as few evaluations as possible [38]. It achieves this by approximating the objective function with a gaussian process (GP) [38]. Gaits from the behaviour-performance map are evaluated in short episodes of walking and their measured performance is used to update the GP. This process repeats until a new optimal gait in the behaviour-performance map is found, typically taking under 2 minutes. Therefore, if the performance drop was the result of the failure of an actuator in a leg, the IT&E approach is likely to settle on a gait in the behaviour-performance map which is characterised by very little use of that failed leg. In summary IT&E divides adaptation into first exploring all the possible ways a robot can perform a task, storing a diverse set of these optimal solutions, and then finding which solution works best after a change such as an actuator failure. This approach is summarised in Figure 2.10. [13]

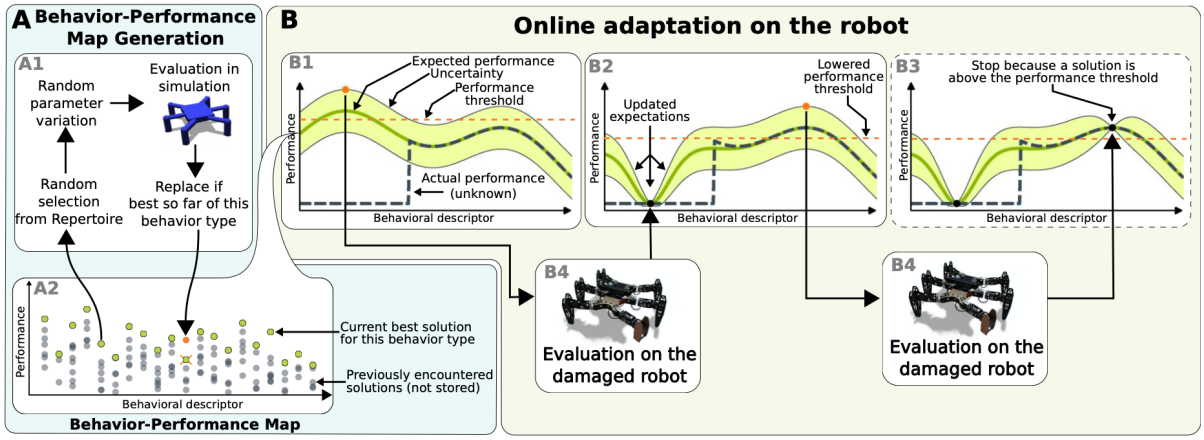


Figure 2.10: Overview of the Intelligent Trial-and-Error algorithm, adapted from [13]

Implementation

The IT&E algorithm was tested on an 18 degree-of-freedom lightweight rectangular hexapod robot, enabling it to successfully adapt to 5 different leg failures, each in under 2 minutes with a walking speed of 50% of the original speed. The sensor feedback on the robot was an RGB camera which was used to determine its walking speed with a Simultaneous Localisation and Mapping (SLAM) algorithm. [13]

2.4.3 Summary of Existing Approaches

The key components of the aforementioned online adaptation approaches are summarised in Table 2.4.

Table 2.4: Summary of existing adaptation approaches

Characteristics	**IT&E** [13]	**GrBAL** [14]
Date Published	2015	2019
Approach	Trial & Error	Meta-Learning
Gait Generation	MAP-Elites	Meta Reinforcement Learning
Deployment Prior	Behaviour-performance map	Meta-trained ANN
Adaptation	Bayesian Optimisation	Stochastic Gradient Descent

The GrBAL and IT&E approaches both feature the same structure, albeit with entirely different algorithms. Both approaches treat adaptation as a dual optimisation problem and use pre-computation to reduce the optimisation time during the mission. Both approaches have been designed for implementation on an actual robot and have been physically tested, and are therefore relevant to the RARL hexapod robot.

2.5 High Performance Computing

Applications such a nontrivial dynamic simulations coupled with evolutionary algorithms typically require substantial computing power to produce results within a reasonable timeframe. The Central Processing Unit (CPU) on a computer is responsible for performing computations. These computations are performed sequentially and the rate at which they are performed is determined by the speed of the CPU, known as the frequency or clock speed. Computing power can be increased by using a faster CPU, however the extent to which this can be done is limited by the speed of light, quantum mechanics, and thermodynamics [39]. The alternative is to use multiple CPU's and distribute the computational load amongst these CPU's. This is referred to as parallel computing as each CPU works on a task in parallel to the other CPU's. Only computational tasks which can be temporarily split up into independent subtasks can benefit from parallel computing. Large collections of CPU's are called computer clusters and are typically organised into nodes connected over a fast local network where each node is a computer containing multiple CPU's[39]. These nodes typically share a common storage system. A parallel computing task can be split up and performed simultaneously on each CPU's in the cluster providing a massive performance increase compared to a single computer.

2.6 RARL Hexapod

This project aims to continue the research and development of the hexapod robot in UCT's RARL lab, depicted in Figure 2.11. This Hexapod has gone through a number of design iterations [9][10][11][12] and is now a highly capable robot able to grasp objects, wirelessly charge itself, and maintain a level body on undulating terrain. The hexapod has been designed with the goal of being used in search and rescue scenarios, and will therefore benefit from the ability to complete missions with unexpected faults.

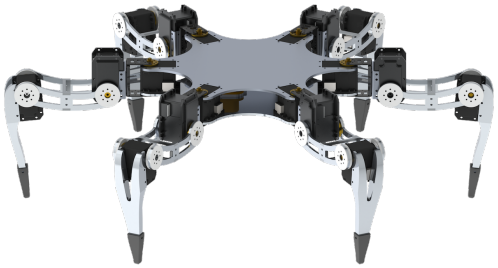


Figure 2.11: SolidWorks rendering of the RARL hexapod robot showing circular leg configuration

2.6.1 Mechanical Design

The RARL hexapod is a circular symmetrical design with legs evenly spaced around the base as seen in Figure 2.11. Each leg has three degrees-of-freedom and is actuated by two Dynamixel RX-28 servos at the coxa and tibia joints, and one higher torque (see Table 2.5) Dynamixel RX-64 servo at the femur joint shown in Figure 2.12.

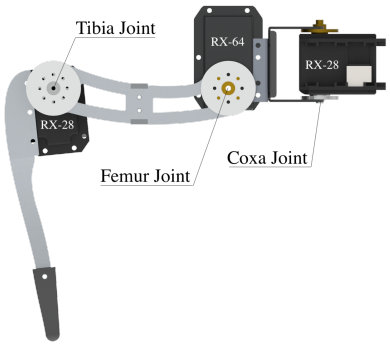


Figure 2.12: SolidWorks rendering of RARL hexapod leg morphology showing actuators and their arrangement

Table 2.5: Dynamixel servo characteristics

Characteristics	**RX-28**[40]	**RX-64**[41]
Weight (g)	72	116
Max Holding Torque ($kgf \cdot cm$)	37.7	64.0
Resolution	0.29°	0.30°
Recommended Voltage (V)	14.4	18
Speed ($sec/60°$)	0.126	0.162

These actuators are high quality, high torque servomechanisms with both position and speed control. Additionally they can provide feedback for angular position, angular velocity, temperature, supply voltage, and load torque using the RS485 Asynchronous Serial Communication Protocol. [40][41]

2.6.2 Electronic System

The hexapod is controlled by an onboard STM32F407VGT6 microcontroller, programmed in C++, which is responsible for computing the leg trajectories, IMU body stabilisation, and communicating with the control computer. Communication is achieved with a RF1100-232 transceiver at a baud rate of 19200 and 10 dBm, affording approximately 110 m line-of-sight operation. The hexapod is powered with six cylindrical 18650 lithium-ion cells in series providing a supply of 25.2 V. [12]

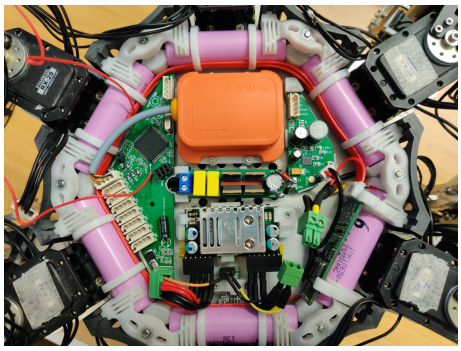


Figure 2.13: Photograph of the RARL hexapod electronic system and component arrangement adapted from [12]. The 18650 batteries are purple, the MTi-28A33G35 IMU is orange, and the STM32F407VGT6 microcontroller is on the left green PCB.

2.6.3 Inertial Measurement Unit

The Xsens MTi-28A33G35 IMU on the hexapod is currently the primary source of high quality sensor data. An Inertial Measurement Unit (IMU) refers to a device which commonly contains a three-axis accelerometer and a three-axis angular rate transducer. An accelerometer measures the external specific force acting on the sensor, or its acceleration relative to its instantaneous rest frame. On earth the specific force consists of both the sensor's acceleration and the earth's gravity. An angular rate transducer measures the rate of change of the sensors orientation. Using both of these sensors with an initial position and orientation it is possible to fully describe the motion of a rigid body in space through dead reckoning as described by the process in Figure 2.14. [42]

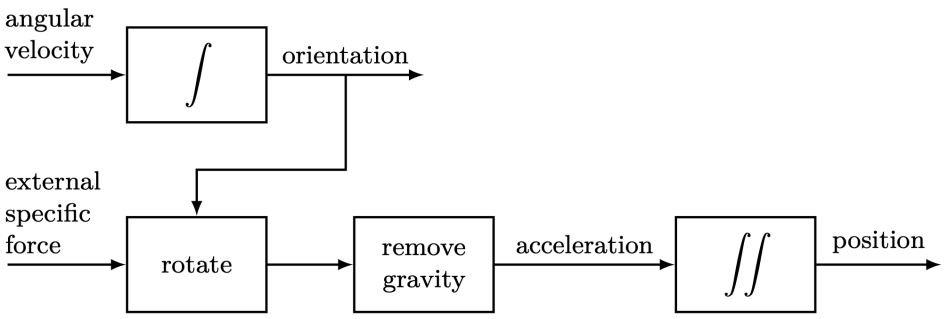


Figure 2.14: Schematic of the process of dead reckoning adapted from [42]

Inertial sensors are often used for navigation where the position and orientation of a device over time are of interest [42]. Dead-reckoning is however significantly more complicated in reality due to the accumulation of sensor errors after integration, referred to as integration drift [42]. The inertial sensors are therefore often combined with drift-free sensors such as GPS or a Magnetometer [42]. The Xsens MTi-28A33G35 is a miniature IMU with integrated 3D magnetometer, capable of providing drift-free 3D orientation as well as calibrated 3D acceleration, 3D rate of turn and 3D earth-magnetic field data, using a Kalman Filter (XKF-3) for sensor fusion at a default rate of 100 Hz [43]. This data is very useful for the analysis of dynamic movements and is often used in human motion tracking [42].

2.6.4 Gait Controller

The gait controller on the hexapod is a feed-forward kinematic controller which recieves high level motion commands which are used to produce a sequence of servo joint angles which are then transmitted to the the servos. This process is described in Figure 2.15.

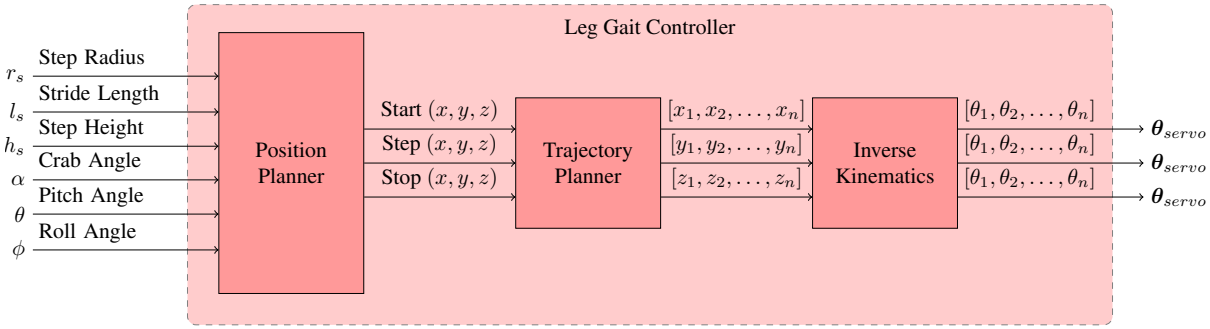


Figure 2.15: Schematic of hexapod gait controller for each leg

The trajectory planner in Figure 2.15 uses a sixth-order polynomial and discretisation to produce a sequence of foot positions between the three calculated via points. These foot positions are then converted to servo joint angles with inverse kinematics developed by [12] which are then repeatedly sent to the servos at a fixed rate producing smooth motion. The current default gait is a tripod gait due to its combination of stability and speed. [12]

2.6.5 System Overview

The integration of the previously described systems is shown in Figure 2.16.

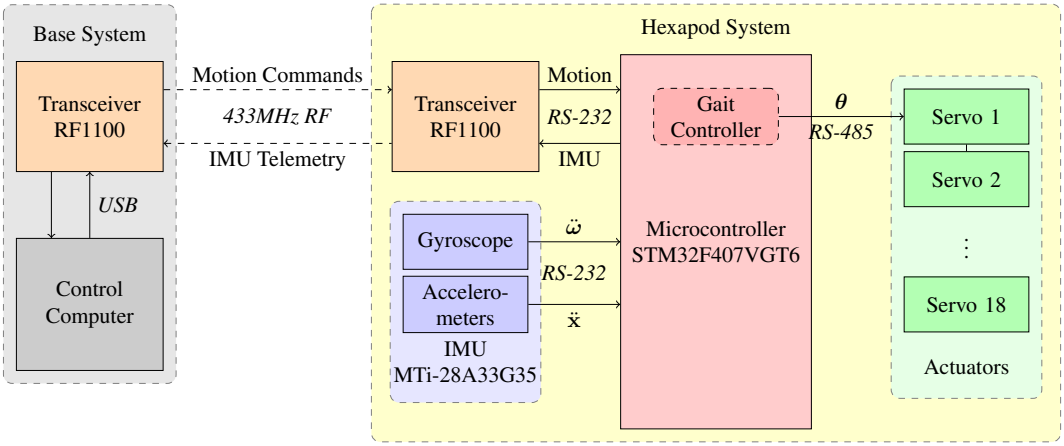


Figure 2.16: Diagram overview of the hexapod system excluding power supply system

The significant components in the RARL hexapod system are summarised in Table 2.6

Table 2.6: Hexapod robot specifications summary for significant components

Characteristic	**Specification**
Degrees of Freedom	18
Programming Language	C++
Microcontroller	STM32F407VGT6
Actuators	6 × Dynamixel RX-64 Servos 12 × Dynamixel RX-28 Servos
Communications	RF1100-232 433MHz Transceiver
Sensors	Xsens Motion Tracker MTi-28A33G35
Power Supply	6 × 18650 lithium-ion cells (25.0 V)
Default Gait	Tripod

Chapter 3

Design

3.1 Adaptation Approach Selection

The RARL hexapod can provide sensor feedback data via its IMU and accepts walking commands through the RF1100 transceiver. These systems can be modified to work with GrBAL or IT&E. The characteristics of the physical tests for each approach will provide insight into which is best suited for the RARL hexapod. These are collated in Table 3.1.

Table 3.1: Summary of physical robot tests for IT&E [13] and GrBAL [14]

Characteristics	**IT&E**	**GrBAL**
Robot Name	Creadapt Hexapod	Millirobot
Robot DOF's	18	2
Adaptation Time	30 s to 120 s	$\approx$ 1 s
Sensor Feedback	Body velocity (1D)	Body position, body velocity, body pose, body angular velocity, motor back-EMF readings, joint angles, joint velocities, battery voltage (24D)
Failures	Shortened leg, unpowered leg, one leg missing, two legs missing, rudimentary leg repair	Removed leg, novel terrains, novel slopes, pose miscallibration, unbalanced payloads

The RARL Hexapod has an IMU (6D) and torque sensors in each servo (18D). The torque sensors however cannot be read whilst simultaneously sending position commands [12]. Therefore the only source of sensor data whilst in motion is the IMU. The hexapod therefore cannot provide the same level of sensor feedback data dimensionality used in the GrBAL tests (24D). The IT&E algorithm however only requires 1 dimensional feedback on the walking speed of the hexapod. This can be achieved with the IMU through integration of the linear accelerations [42]. Additionally IT&E was tested on a legged robot with a similar number of degrees-of-freedom to the RARL hexapod. For these two reasons the IT&E algorithm was selected as the best suited adaptation approach.

3.1.1 Requirement Specification

Based on the requirements of the selected IT&E adaptation algorithm and the previous implementation in [13] the following are required in the solution developed in this project.

Table 3.2: Adaptation functional requirements specification

Requirement	**Specifications**	**Justification**
Gait Controller	• $\approx$ 36 unique parameters • Performs common gaits	6 parameters per leg provided sufficient variation in [13]. The RARL hexapod [12] uses a kinematic approach to gait control. Common gaits such as the tripod gait serve as benchmarks.
Walking Feedback	• Body velocity along axis	IT&E requires a simulation of the hexapod for gait generation with MAP-Elites [13].
Simulation	• See table 3.3	IT&E requires velocity feedback for online evaluation and selection of gaits [13].
Gait Generation	• 40 million evaluations • 5 s simulated gait evaluations	IT&E Required these specifications for gait generation in a prior implementation [13].
Adaptation	• $<$120 s to find optimal gait • Each gait trial $\approx$ 5 s	The IT&E approach achieved the following adaptation performance in prior tests [13]. Equivalent performance is therefore required to indicate a successful implementation.

A simulation of the hexapod robot is a fundamental requirement of the IT&E adaptation approach, but also has its own requirements. It was therefore separated into its own table.

Table 3.3: Simulator software requirement specifications

Requirement	Specifications	Justification
	Inputs	
Hexapod model	• Exact link inertial properties • Exact link dimensions	PyBullet accepts URDF files [29]. An undamaged configuration is required for gait generation and a damaged configuration is required for adaptation testing. Exact link dimensions and masses are required to ensure accuracy of the dynamics.
	Outputs	
Collision detection	• Overturning • Leg collisions • Foot-ground contact detection	Motions which result in overturning or leg collisions could damage the robot. It is therefore required that the simulator have a way of detecting these scenarios.
	Interfaces	
Visualiser	• 3D • Optionally enabled • Robot tracking	Visual feedback is an important component for evaluating simulated motions. It however presents unnecessary computation during gait generation and is therefore required to be optional.
	Performance	
Accuracy	• RX-28 servo characteristics • RX-64 servo characteristics	Accurately modelling actuators was identified as the first means of reducing the reality gap [26].
Parallelisation	• Makes full use of number of available CPU's • 1 simulation per CPU	MAP-Elites supports parallelisation to reduce computation time [36]

3.1.2 Implementation Concept

Similar to the IT&E approach used in [13] a collection of diverse and high performing gaits will be evolved using MAP-Elites and a simulation of the RARL hexapod on a high performance computer. To produce a diverse set of gaits a custom parameterised gait controller is required which can be implemented in both the simulation and on the physical robot. On the current hexapod platform a few gait parameters are already wirelessly transferred between the robot and the control computer such as the body height and the walking speed, depicted in figure 2.16. This communication channel can simply be expanded to include additional parameters for the custom gait controller. The IMU readings will also be sent back to control computer. The control computer can therefore be used for the majority of the computation performing the dead-recking with the IMU data and selecting alternative gaits with M-BOA. This leaves the limited computation power of the hexapod microcontroller to be used solely for receiving gait parameters, producing the desired gait, and transmitting IMU data. The collection of diverse and high performing gaits generated with MAP-Elites will be stored in a `.dat` file and transferred to the control computer with any typical file transfer means such as a thumb-drive. A graphical depiction of this implementation concept can be found in appendix A.

3.2 Algorithm Theory

3.2.1 MAP-Elites Algorithm

The MAP-Elites algorithm [36] forms a fundamental part of the IT&E adaptation approach being implemented in this project with the behaviour-performance maps being analogous to the robots intuition about the different ways it can walk. A review of the theory behind the algorithm is therefore essential as it guides later design and analysis.

MAP-Elites is an evolutionary algorithm developed by Jean-Baptiste Mouret and Jeff Clune. At its crux evolutionary algorithms are optimisation algorithms where the goal is to find the maximum of a black box function. This is expressed mathematically as the following.

$$\underset{\mathbf{x}}{\operatorname{argmax}}\{f(\mathbf{x})\} \tag{3.1}$$

The process used to maximise this function draws inspiration from evolutionary theories of nature. A successful member will survive and proliferate, while unsuccessful members will quickly die off. Through this process of natural selection and survival of the fittest the population of solutions gradually increases in performance. The unknown function (f) is treated as the measure of fitness or performance of a particular solution ($\mathbf{x}$) and new solutions are created by selecting high performing solutions from the current population, and combining their qualities to produce child solutions ($\mathbf{x}'$). The higher performing child solutions replace the lower performing parent solutions within the population, and the cycle repeats. Termination occurs after a certain number of iterations or once a desired performance is achieved.

MAP-Elites takes this a step further and rather than generating a single high performing solution, it generates a diverse collection of high performing solutions which are each different from one another. In order to differentiate solutions the fitness function needs to provide a description of a solution called the behaviour descriptor ($\mathbf{b}$) in addition to the performance of the solution (p). This behaviour descriptor can be a multi-dimensional parameter. The fitness function is therefore required to be in the following form.

$$p, \mathbf{b} = f(\mathbf{x}) \tag{3.2}$$

Using this behaviour descriptor ($\mathbf{b}$) the high performing solutions in the current population are organised into separate bins or niches. New child solutions can then only replace a lower performing parent solution if it is in the niche which corresponds to their behaviour descriptor. If this is the case then the child solution becomes the new high performing solution for that particular niche and the parent solution is removed. Once gain the cycle is repeated similar to traditional evolutionary algorithms.

The diversity is ensured by dividing the multidimensional behaviour descriptor space evenly into a specified number of niches. Even partitioning of the multi-dimensional space is achieved with a Centroidal Veroni Tesselation (CVT) [44].

Similar to traditional evolutionary algorithms MAP-Elites combines parent solutions from separate niches to produce child solutions. To combine the child solutions MAP-Elites uses Simulated Binary Crossover.

The product of MAP-Elites is a diverse archive of different high performing solutions for the original function (f) based on the multi-dimensional behaviour descriptor. Hence the name MAP-Elites which stands for the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites). This diverse archive of solutions is referred to in this report as a behaviour-performance map or simply a map for short.

In this project MAP-Elites will be used to generate a diverse collection of high performing gaits for the RARL hexapod robot. The fitness function (f) in this context therefore relates some collection of gait parameters ($\mathbf{x}$) to a gait performance (p) and a gait behaviour descriptor ($\mathbf{b}$).

3.2.2 Pseudocode

The pseudocode for the MAP-Elites algorithm from [36] is shown in listing 1. This pseudocode is reproduced in the report for reference.

Algorithm 1 MAP-Elites [36]

1: $(\mathcal{P} \leftarrow \emptyset, \mathcal{X} \leftarrow \emptyset)$ ▷ Create an empty, N-dimensional map to store elites
2: **for** $i = 1 \rightarrow I$ **do** ▷ Repeat for I iterations
3: **if** $i < G$ **then** ▷ Randomly initialise map for G iterations
4: $\mathbf{x}' \leftarrow$ random
5: **else**
6: $\mathbf{x} \leftarrow$ random selection from $\mathcal{X}$
7: $\mathbf{x}' \leftarrow$ random mutation of $\mathbf{x}$
8: $p, \mathbf{b} \leftarrow f(\mathbf{x}')$
9: **if** $\mathcal{P}(\mathbf{b}) = \emptyset$ or $\mathcal{P}(\mathbf{b}) < p$ **then**
10: $\mathcal{P}(\mathbf{b}) \leftarrow p$
11: $\mathcal{X}(\mathbf{b}) \leftarrow \mathbf{x}'$

3.2.3 M-BOA Algorithm

The Map-based Bayesian Optimisation algorithm [13] is what enables adaptation to an unknown failures by finding the optimum solution in the behaviour-performance map generated with MAP-Elites. The theory behind this algorithm is therefore also fundamental to this project and subsequent design decisions.

Bayesian optimisation is another algorithm used for the optimisation of black box functions. It is typically used for functions which take a long time (minutes to hours) to evaluate, and therefore aims to minimise the number of evaluations. This function can also be represented as the following.

$$\underset{\mathbf{x}}{\operatorname{argmax}}\{f'(\mathbf{x})\} \tag{3.3}$$

Bayesian optimisation works by building a surrogate for the fitness function (f') with a gaussian process. Prior data is used to approximately fit this surrogate model. A probabilistic gaussian process is used as can represent values at certain points as well as their uncertainty. A new point to evaluate on the function is selected based on the highest estimate predicted by the gaussian process, otherwise known as the Upper Confidence Bound (UCB) acquisition function. The prior gaussian process is then updated with this true value and the process is repeated until a stopping criteria is met. With more trials the gaussian process becomes a better and better approximation of the black box function to be optimised.

M-BOA is simply standard bayesian optimisation with the only difference being that it finds the optimum solution ($\mathbf{x}$) within a behaviour-performance map generated with MAP-Elites based on a modified fitness function (f'). This can be represented mathematically as the following.

$$\underset{\mathbf{x}\in\mathcal{X}}{\operatorname{argmax}}\{f'(\mathbf{x})\} \tag{3.4}$$

Where $\mathcal{X}$ is the set of diverse and high performing solutions created with MAP-Elites. This allows a high performing solution to be found within the behaviour performance map using an alternate fitness function (f') to the one originally used to generate the map in the first place (f).

$$f'(\mathbf{x}) \neq f(\mathbf{x}) \quad where \quad \mathbf{x} \in \mathcal{X} \tag{3.5}$$

This fitness function only needs to be in the following form and does not need to provide a descriptor like the fitness function used for MAP-Elites.

$$p = f'(\mathbf{x}) \tag{3.6}$$

The previous performances of the solutions in the map ($\mathcal{P}$) evaluated with f serve as the prior/initial guess for the gaussian process which is iteratively updated using the new performances returned from f'. This is analogous to an expectation vs reality scenario. M-BOA terminates once the expectation is within a certain percentage of the reality.

Pseudocode

The pseudocode for the M-BOA algorithm from [13] is shown in listing 2. This pseudocode is reproduced in this report for reference.

Algorithm 2 M-BOA [13]

$\forall \mathbf{x} \in map$ ▷ Use the behaviour-performance map from MAP-Elites
$P(f'(\mathbf{x})|\mathbf{x}) = \mathcal{N}(\mu_0(\mathbf{x}), \sigma_0^2(\mathbf{x}))$ ▷ Fit a gaussian process P to approximate f'
where
$\mu_0(\mathbf{x}) = \mathcal{P}(\mathbf{x})$ ▷ Initialise the mean prior from the map
$\sigma_0^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x})$ ▷ Initialise the variance prior from the map
while $\max(\mathbf{P_{1:t}}) < \alpha \max(\mu_t(\mathbf{x}))$ **do**
    $\mathbf{x}_{t+1} \leftarrow \text{argmax}_x(\mu_t(\mathbf{x}) + \kappa\sigma_t(\mathbf{x}))$ ▷ Select solution based on UCB
    $p_{t+1} \leftarrow f'(\mathbf{x})$ ▷ Evaluate solution on modified fitness function
    $P(f'(\mathbf{x})|\mathbf{P}_{1:t+1}, \mathbf{x}) = \mathcal{N}(\mu_{t+1}(\mathbf{x}), \sigma_{t+1}^2(\mathbf{x}))$ ▷ Update the gaussian process
    where
    $\mu_{t+1}(\mathbf{x}) = \mathcal{P}(\mathbf{x}) + \mathbf{k}^T\mathbf{K}^{-1}(\mathbf{P}_{1:t+1} - \mathcal{P}(\mathcal{X}_{1:t+1}))$ ▷ Update the mean
    $\sigma_{t+1}^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k}$ ▷ Update the variance
    $\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_{t+1}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_{t+1}, \mathbf{x}_1) & \dots & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix}$ ▷ Observation correlation matrix
    $\mathbf{k} = \begin{bmatrix} k(\mathbf{x}, \mathbf{x}_1) & k(\mathbf{x}, \mathbf{x}_2) & \dots & k(\mathbf{x}, \mathbf{x}_{t+1}) \end{bmatrix}$

3.2.4 IT&E Algorithm

Intelligent Trial and Error algorithm from [13] is the combination of a behaviour-performance map generated with MAP-Elites and the M-BOA algorithm to produce rapid adaptation. This works in a variety of situations where an expected performance does not match a measure performance, however for this project it will be applied to adapt the RARL hexapod robots gait to leg failures.

The approach begins by generating a diverse set of high performing gaits using MAP-Elites. These gaits are generated using a simulation of the robot as MAP-Elites requires a vast number of evaluations and a simulation can be evaluated faster than real time. A gait controller will take in parameters and move the legs of a simulated hexapod, and the simulator will determine the dynamics of that movement and the resulting performance and behaviour descriptor of the gait.

These gait parameters can then be transferred to the real robot. When a failure occurs the performance of a gait will typically drop. At this point the expected performance will no longer match the performance expected based on the simulation as there were no failures in the simulation. M-BOA can then be used to find a new gait within the behaviour-performance map which performs optimally with this unexpected failure. M-BOA only requires a one dimensional fitness value describing the performance of a gait and therefore does not require complex sensor data from the physical robot. M-BOA works by trying gaits from the map and seeing which perform as expected and which do not. This process of trial and error is where the adaptation algorithm gets its name.

Pseudocode

The pseudocode for the IT&E algorithm from [13] is shown in listing 3. This pseudocode is reproduced in this report for reference.

Algorithm 3 IT&E [13]

1: $\mathcal{X}, \mathcal{P} = \text{MAP-Elites}(f(\mathbf{x}))$ ▷ Create behaviour-performance map
2: **while** in mission **do**
3: **if** significant performance drop ($f \rightarrow f'$) **then** ▷ Failure scenario
4: M-BOA($f'(\mathbf{x}), \mathcal{X}, \mathcal{P}$) ▷ Find compensatory gait in map

3.3 Hexapod Model

A model of the hexapod robot is required to describe to the physics engine how the robot looks and moves. This model must accurately represent the physical properties of the hexapod robot in order for the dynamics simulation to be accurate. This chapter outlines the development of this model for the hexapod robot.

3.3.1 Structure

PyBullet is able to load URDF, SDF, and MJCF robot model files. The URDF format stands for Universal Robot Description Format and it an open-source format developed for use within the Robot Operating System (ROS) [45]. ROS is widely accepted as the industry standard for robotics middleware. It would therefore be beneficial to future development of the hexapod platform to have an accurate model of the robot which is also compatible with ROS. A URDF file uses the Extensible Markup Language (XML) format with the extension `.urdf` to describe the appearance and kinematics of a robot in a human and machine readable format. The elements are arranged in a tree structure originating from the root element, with attributes to specify particular properties. The URDF file begins with the prologue identifying the XML version and character encoding. The root element in a URDF file is the `robot` which is made up of `links` which are connected by `joints`. This general structure for the URDF file is shown below in listing 3.1.

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <robot name="hexapod">
3    <link> ... </link>
4    <link> ... </link>
5    <link> ... </link>
6    ...
7    <joint> .... </joint>
8    <joint> .... </joint>
9    <joint> .... </joint>
10 </robot>
```

Listing 3.1: URDF Structure

The hexapod robot is comprised of a base and 6 legs. Each of these 6 legs consists of 3 links connected by 3 joints. The first link on the leg from the body is called the coxa, followed by the femur, and the tibia. The end of the tibia serves as the foot. This structure is shown as a tress diagram in figure 3.1. Each of the links and joints is defined by a separate element in the URDF file.

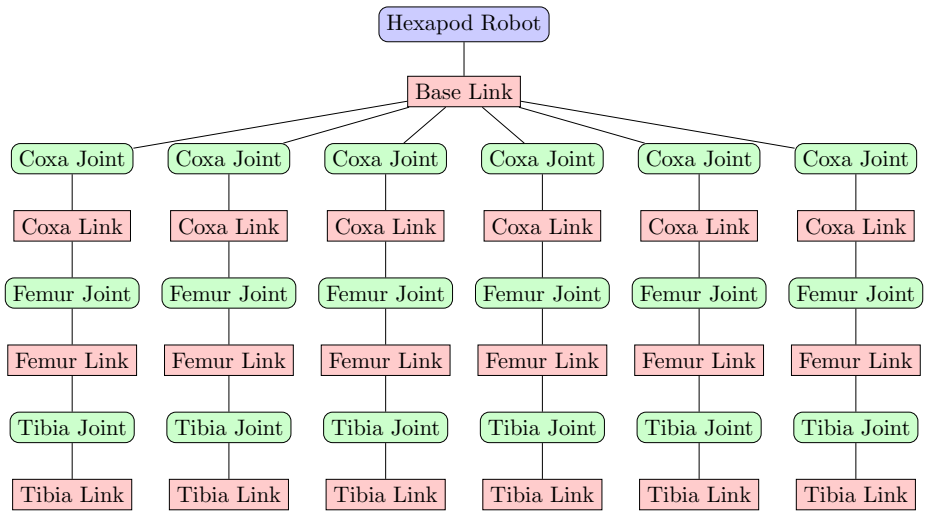


Figure 3.1: Hexapod kinematic tree structure
Links are coloured in red and joints in green

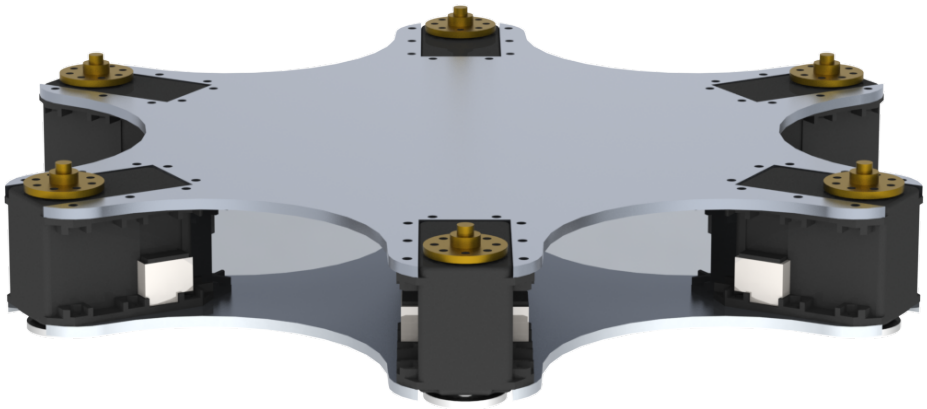
3.3.2 Links

Link elements in the URDF contain information about the rigid bodies which make up the robot. This includes the inertia, visual appearance, and collision bounds. This link information is stored as elements according to the XML structure shown in listing 3.2.

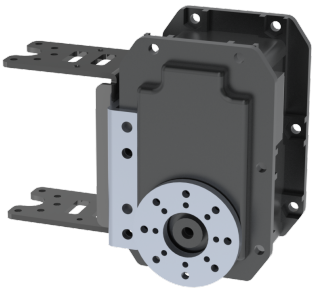
```
<link name="...">
  <inertial>
    <mass value="0" />
      <origin rpy="0 0 0" xyz="0 0 0" />
      <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>

    <visual>
      <origin rpy="0 0 0" xyz="0 0 0" />
      <geometry>
        <mesh filename="package://urdf/meshes/example.stl" />
    </geometry>
    <material name="" />
  </visual>

  <collision>
    <origin rpy="0 0 0" xyz="0 0 0" />
      <geometry>
        <mesh filename="package://urdf/meshes/example.STL" />
      </geometry>
  </collision>
</link>
```

Listing 3.2: Link Element XML Structure

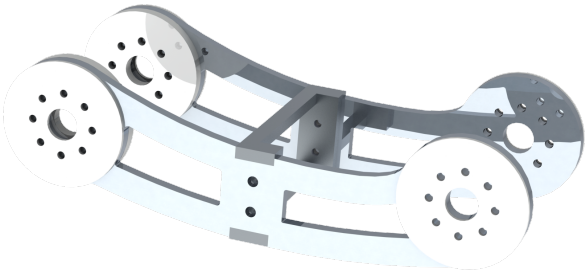
Even though the servos are the joints, they have mass and form part of the links, adding to the inertia and collision bounds. Each of the legs of the hexapod are identical, differing only in their attachment position to the body. The whole hexapod therefore is comprised of only 4 unique rigid bodies; the body, coxa, femur, and tibia. The coordinate frame for each of the links is defined relative to the joint coordinate frame according to the URDF convention. The 4 unique links which make up the hexapod are shown in figure 3.2.



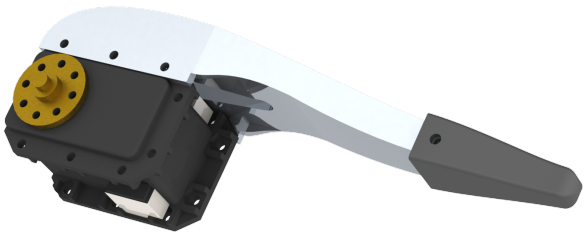
(a) Base link



(b) Coxa link



(c) Femur link



(d) Tibia link

Figure 3.2: Hexapod links
Not to scale

Naming

URDF convention requires that the main body of the robot be named `base_link` and that all subsequent link names be unique. For simplicity, each link was named link with a suffix of its leg number followed by its position from the base link, separated by underscores. For example the femur on leg

$$\text{link\_}\langle \textit{leg number} \rangle\text{\_}\langle \textit{position from base} \rangle$$

Inertia

The inertia properties of each of the links is a particularly important aspect when ensuring that the model accurately represents reality. This information is contained within the `inertial` XML element of a link within the URDF. SolidWorks was primarily used to calculate the inertia properties as the solid model included accurate material properties and was up-to-date. However, the mass of the base link could not be estimated using SolidWorks as the model was missing a number of large components, namely the batteries, IMU, transceiver and printed circuit board. Instead these component masses were estimated using alternative methods detailed in table 3.4.

Table 3.4: Mass estimate for base link

Component	**Source**	**Unit Mass** (g)	**Qty**	**Mass** (g)
LiPo Battery	Datasheet	45	6	270
RX-28 Servo	Datasheet	72	6	432
Xsens IMU	Datasheet	50	1	50
Chassis	SolidWorks	258	2	516
Transceiver	Datasheet	12	1	12
PCB	Estimate	20	2	40
Total Mass				1320

The masses for the remaining coxa, femur, and tibia links could simply be estimated using SolidWorks. The masses for all of the links are summarised in table 3.5.

Table 3.5: Link masses

Link	**Mass** (kg)
Base	1.320
Coxa	0.146
Femur	0.058
Tibia	0.154

These values for the link masses are included in the URDF link element under mass value. The mass for each of these links is not uniformly distributed and therefore the average point at which it acts, or centre of mass, also needs to be calculated and included in the URDF. The position of the centre of mass is given relative to the link origin, which is the joint coordinate frame as shown in figure 3.3.

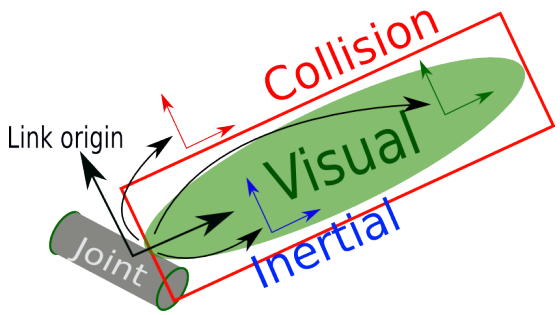


Figure 3.3: Link URDF structure from [45]

Table 3.6: Centre of mass position for each link type relative to the joint frame

Link	$\mathbf{x}_{CoM}$ (m)	$\mathbf{y}_{CoM}$ (m)	$\mathbf{z}_{CoM}$ (m)
Base	0	0	-6.800×10^{-4}
Coxa	4.992×10^{-2}	-1.400×10^{-4}	1.457×10^{-2}
Femur	5.478×10^{-2}	-3.960×10^{-3}	0
Tibia	3.309×10^{-2}	7.960×10^{-3}	-2×10^{-5}

The rotational inertia properties of each of the links are also calculated using SolidWorks. In addition to the mass and centre of mass properties, the inertia tensor of a rigid body is required to fully describe the inertial properties. The inertia tensor describes the distribution of mass within a rigid body. It can be written in the following matrix form.

$$\mathbf{I} = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix} \tag{3.7}$$

The URDF structure requires that the inertia tensor be given at the centre of mass and aligned to the link coordinate frame. This was computed for each of the 4 joints using SolidWorks. The IMU, PCB, and Transceiver were assumed to be point masses at the centre of the base. This assumption was made as they only make up a small proportion of the total mass and they are centred relative to the base frame. The servos, batteries, and aluminium chassis were not assumed to be point masses at the centre as they make up a significant portion of the mass and

are not centred. The inertia tensors for the links are stated below.

$$\mathbf{I}_{base} = \begin{pmatrix} 5.255 \times 10^{-3} & 0 & -1 \quad \times 10^{-11} \\ 0 & 5.255 \times 10^{-3} & -1.300 \times 10^{-10} \\ -1 \quad \times 10^{-11} & -1.300 \times 10^{-10} & 1.015 \times 10^{-2} \end{pmatrix} \mathrm{kg\,m^2} \tag{3.8}$$

$$\mathbf{I}_{coxa} = \begin{pmatrix} 6.023 \times 10^{-5} & -6.480 \times 10^{-8} & 9.306 \times 10^{-7} \\ -6.480 \times 10^{-8} & 6.045 \times 10^{-5} & -3.629 \times 10^{-7} \\ 9.306 \times 10^{-7} & -3.629 \times 10^{-7} & 4.573 \times 10^{-5} \end{pmatrix} \mathrm{kg\,m^2} \tag{3.9}$$

$$\mathbf{I}_{femur} = \begin{pmatrix} 3.893 \times 10^{-5} & 1.706 \times 10^{-6} & 0 \\ 1.706 \times 10^{-6} & 1.248 \times 10^{-4} & 0 \\ 0 & 0 & 9.600 \times 10^{-5} \end{pmatrix} \mathrm{kg\,m^2} \tag{3.10}$$

$$\mathbf{I}_{tibia} = \begin{pmatrix} 3.191 \times 10^{-5} & 1.071 \times 10^{-5} & -1.034 \times 10^{-7} \\ 1.071 \times 10^{-5} & 1.890 \times 10^{-4} & 4.968 \times 10^{-8} \\ -1.034 \times 10^{-7} & 4.968 \times 10^{-8} & 1.927 \times 10^{-4} \end{pmatrix} \mathrm{kg\,m^2} \tag{3.11}$$

$$\tag{3.12}$$

Visual

The visual element describes the appearance of the link. The visual geometry element can contain primitive shapes or the filename of a mesh file. A mesh file is particularly useful as the hexapod geometry is complex and meshes will provide an accurate visual representation. The supported mesh types within the UDRF are STL and DAE files. SolidWorks only supports exporting models to STL format so this mesh format was used. To simplify positioning of the mesh elements the STL files were exported with their origins as the joint frame so no translations or rotations were required. The meshes need to be stored within the UDRF folder, and their filename is specified as follows.

```
package://urdf/meshes/mesh.STL
```

Collision

The collision element of the URDF defines the collision bounds used in computing contacts. Just like the visual element, the collision geometry element can also be defined by an STL mesh file. The collision element was therefore almost identical to the visual element of the link, and both use the same STL file.

3.3.3 Joints

Joint elements in the URDF file describe the kinematics properties of the joints in the robot. They specify which two links they join, the position of the joint, the axis of rotation, and the limits of the joint. This joint information is stored as elements according to the XML structure shown in listing 3.3.

```
<joint name="..." type="revolute">
    <origin rpy="0 0 0" xyz="0 0 0" />
    <parent link="..." />
    <child link="..." />
    <axis xyz="0 0 1" />
    <limit effort="0" lower="0" upper="0" velocity="0" />
  </joint>
```

Listing 3.3: Joint Element XML Structure

The joint element has a name and type attribute. For consistency naming of the joints followed the same convention to naming of the links with a suffix of the leg number followed by the joint position from the base link, separated by underscores.

$$\text{joint\_}\langle \textit{leg number} \rangle\text{\_}\langle \textit{position from base} \rangle$$

A variety of joint types are supported in the URDF format, however the hexapod is comprised entirely of revolute servo motors, so the revolute joint type is used for every joint type attribute.

Origin

The origin of a joint defines where the joint acts, and is specified as a transformation relative to the origin of its parent link, and includes both orientation and position values. These were calculated in SolidWorks and are summarised in table 3.7.

Table 3.7: URDF relative joint origins and positions

Joint	**r** (rad)	**p** (rad)	**y** (rad)	**x** (m)	**y** (m)	**z** (m)
Coxa	α	0	0	$0.125 \cdot \cos(\alpha)$	$0.125 \cdot \sin(\alpha)$	-1.400×10^{-2}
Femur	π	0	0	5.317×10^{-2}	0	0
Tibia	0	0	0	1.019×10^{-1}	0	0

The values in table **??** are given in the same units expected in the UDRF file. The positions of the Tibia joints around the base are a function of the leg number and therefore the angle of the leg around the base, expressed as α.

$$\alpha = (N_{leg} - 1) \cdot \frac{\pi}{3} \tag{3.13}$$

Link Hierarchy

One of the fundamental roles of a joint element in the URDF is to connect two links. These two links are specified in the parent and child elements within the joint. These elements take a link name as an attribute. The tree structure of the hexapod in figure 3.4 best describes the relationships between joints and their links along with their corresponding names within the URDF file.

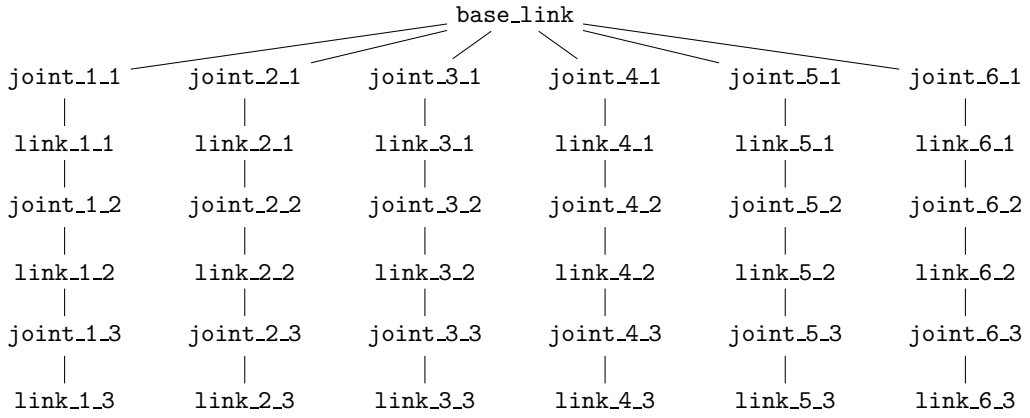


Figure 3.4: Hexapod tree structure with link and joint names

Axis

This element defines the axis of rotation for revolute joints. Due to the Denavit-Hartenberg convention being used to define the joint frames, the axis of each joint is simply the z axis in the joint frame.

```
axis xyz="0 0 1"
```

Limits

Lastly a joint requires limits to be specified for the extent of rotation, torque, and velocity. Each servo has a maximum operating angle of 300°. This operating angle exceeds the range of motion allowed due to the physical structure of the robot and links. Collisions of links in the hexapod are highly undesirable as they could result in permanent damage, therefore the limits of each of the joints were specified to ensure that a link cannot collide with its parent link. This range of motion was determined in SolidWorks and is summarised in table 3.8.

Table 3.8: Joint limits

Joint	**Lower limit** (rad)	**Upper limit** (rad)
Coxa	−1.745	1.745
Femur	−2.618	2.269
Tibia	−2.967	2.269

The torque and velocity limits for the joints are determined by the type of servo at that joint and the rated characteristics provided in the datasheets [41][40]. These servo characteristics are dependent on the supply voltage. The power distribution circuit in the hexapod supplies 15V to each of the servos. Therefore the characteristics from the datasheets were linearly scaled to this voltage to ensure an as accurate approximation of their performance as possible. These torque and velocity limit characteristics are summarised in table 3.9.

These values are stated in the default units for the URDF file. For revolute joint types, the effort limit is the maximum torque the servo can produce and the velocity limit is the maximum rotational speed of the servo.

Table 3.9: Joint servo characteristics at supply voltage

Joint	**Servo Type**	**Effort Limit** ($\mathrm{N\,m}$)	**Velocity Limit** ($\mathrm{rad\,s^{-1}}$)
Coxa	Dynamixel RX-28	3.464	7.686
Femur	Dynamixel RX-64	5.194	5.289
Tibia	Dynamixel RX-28	3.464	7.686

3.3.4 Package Structure

Only a certain file structure is supported for combining the UDRF file with its assets such as the meshes. The following file and folder structure was used for the hexapod URDF.

```
urdf
├── hex.urdf
└── meshes
    ├── base_mesh.stl
    ├── coxa_mesh.stl
    ├── femur_mesh.stl
    └── tibia_mesh.stl
```

3.4 Gait Controller

The gait controller is responsible for generating the joint motions required to achieved the foot trajectories for a given gait. The current gait controller on the hexapod is a feed-forward kinematic controller which only performs the tripod gait. To be able to exhibit a diverse set of gaits a new gait controller was designed. This section describes the design of this controller.

3.4.1 Structure

The controller must compute the required joint angles and speeds for each of the 18 joints of the hexapod for each time-step of the simulator. The gait controller on the hexapod achieves this by precomputing the sequence of joint angles for each leg with multiple nested for loops. This is sufficient on the embedded system, however, the controller will be initialised millions of times for the evolutionary algorithm and should therefore be as fast as possible in computing the joint trajectories. To ensure this, the majority of the joint trajectory computations were rephrased as matrix operations. This allowed the use of the Python library NumPy[46] which is substantially faster than iterating through python lists and calculating each time-step.

3.4.2 Parameter Space

To simplify deployment of this system on the existing platform the gait controller for the simulator closely resembles the implementation programmed on the hexapod microcontroller with a few fundamental differences. The MAP-Elites algorithm optimises the walking speed over the gait parameter space whilst producing a variety of diverse solutions. It is therefore essential to first define the space of gait parameters. These parameters were identified in the literature review whilst exploring hexapod gaits. The gait parameter space therefore includes the ability to modify the support polygon, and the footfall sequence, as well as some additional parameters.

Body Velocity (v) – It is undesirable for each of the legs to be trying to drive the base at their own speed, therefore a global body velocity parameter is defined.

Body Height (h_b) – Another parameter influencing a gait is the height of the body. To ensure variability this cannot be fixed to a specific value.

Leg Radius (r) – The leg radius parameter is the horizontal distance from the start of the hexapod leg, or the coxa joint, to the centre of the legs gait trajectory. It essentially determines the placement of a foot relative to the body during the support phase and therefore indirectly influences the gait's support polygon. The support polygon plays a fundamental role in the stability of a gait and this parameter will allow for the variability of this.

Leg Angle Offset (ϕ) – The leg angle offset parameter works in conjunction to the aforementioned radius parameter to adjust the position of the centre of the legs gait trajectory. The two parameters can be thought of as a polar coordinate definition of the centre of the foot path. The angle offset is given relative to the default radially outwards orientation of each of the legs. This ensures the support polygon of the gait is fully adjustable.

Leg Step Height (s) – The step height parameter refers to the maximum height of the foot at the top of the swing phase of the gait cycle. Rougher uneven terrain requires that the feet be lifted higher to step over obstacles. This is however limited by the orientation of the leg as a leg far from the body has less range of motion than a leg closer to the body. This is therefore another parameter which has an influence on the gait and requires optimisation for each leg.

Leg Duty Factor (β) – The duty factor parameter refers to the portion of time a leg spends in the support phase out of the whole gait cycle. This parameter is often referred to in the literature when identifying gaits and is therefore an essential parameter in producing a wide variety of gaits. The parameter is also unique to each leg.

Leg Phase Offset (ψ) – A gait is a cyclic motion repeating after a fixed period of time. Each leg has its own cyclic motion. This parameter influences where each leg is in its gait cycle relative to the other legs gait cycles and is often depicted in the literature as a shift in the footfall diagram. This is therefore another fundamental parameter in producing a variable gait controller and is defined for each leg.

One constant was defined for the gaits in this project and that is the gait period. This was fixed at 1 second as leaving this variable had undesirable effects during gait generation. Each leg having a unique period was also explored however the gaits were only optimised for a short duration and the legs would go out of phase if left longer. Table 3.10 summarises the 32 unique gait parameters.

Table 3.10: Gait controller parameters

Gait Parameter	**Symbol**	#
Body velocity	v	1
Body height	h_b	1
Leg radius	r	6
Leg angle offset	ϕ	6
Leg step height	s	6
Leg duty factor	β	6
Leg phase offset	ψ	6
Total parameters		32

3.4.3 Foot Trajectory

Initial Points

Gait generation begins by first defining keys points within a desired foot path. These points are the start of the support phase, the start of the swing phase and the top point of the swing phase. With these initial points the rest of the gait path can be filled in. These points are given in the leg coordinate frame.

$$\boldsymbol{p}_{start} = \begin{bmatrix} x & y & z \end{bmatrix}^T = \begin{bmatrix} r_i \cdot \cos(\phi_i) + \frac{l_s}{2}\cos(-\theta_i + \alpha) \\ r_i \cdot \sin(\phi_i) + \frac{l_s}{2}\sin(-\theta_i + \alpha) \\ -h_b + 0.014 \end{bmatrix} \tag{3.14}$$

$$\boldsymbol{p}_{mid} = \begin{bmatrix} x & y & z \end{bmatrix}^T = \begin{bmatrix} r_i \cdot \cos(\phi_i) \\ r_i \cdot \sin(\phi_i) \\ -h_b + 0.014 + s_i \end{bmatrix} \tag{3.15}$$

$$\boldsymbol{p}_{end} = \begin{bmatrix} x & y & z \end{bmatrix}^T = \begin{bmatrix} r_i \cdot \cos(\phi_i) - \frac{l_s}{2}\cos(-\theta_i + \alpha) \\ r_i \cdot \sin(\phi_i) - \frac{l_s}{2}\sin(-\theta_i + \alpha) \\ -h_b + 0.014 \end{bmatrix} \tag{3.16}$$

where:

$$l_s = vT\beta_i \tag{3.17}$$

$$\theta = (n_{leg} - 1)\frac{\pi}{3} \tag{3.18}$$

Swing Trajectory

The swing phase of the hexapod robot is generated using a 6^{th} order polynomial. A 6^{th} order polynomial is used as it eliminates spikes in jerk at the start and end points which occur in lower order polynomials [12]. This polynomial is shown in equation 3.19. The velocities for this path are given by equation 3.20.

$$p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 + a_6 t^6 \tag{3.19}$$

$$\dot{p}(t) = a_1 t + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 + 6a_6 t^5 \tag{3.20}$$

To solve for the coefficients a number of constraints were applied. These constraints were that the path must start at $\boldsymbol{p}_{start}$, end at $\boldsymbol{p}_{end}$, last t_f seconds, and pass through $\boldsymbol{p}_{mid}$ at $\frac{t_f}{2}$ seconds. This resulted in the following coefficients.

$$\boldsymbol{a}_0 = \boldsymbol{p}_{start} \tag{3.21}$$

$$\boldsymbol{a}_1 = \boldsymbol{\emptyset} \tag{3.22}$$

$$\boldsymbol{a}_2 = \boldsymbol{\emptyset} \tag{3.23}$$

$$\boldsymbol{a}_3 = \frac{2}{{t_f}^3}(32(\boldsymbol{p}_{mid} - \boldsymbol{p}_{start}) - 11(\boldsymbol{p}_{end} - \boldsymbol{p}_{start})) \tag{3.24}$$

$$\boldsymbol{a}_4 = -\frac{3}{{t_f}^4}(64(\boldsymbol{p}_{mid} - \boldsymbol{p}_{start}) - 27(\boldsymbol{p}_{end} - \boldsymbol{p}_{start})) \tag{3.25}$$

$$\boldsymbol{a}_5 = \frac{3}{{t_f}^5}(64(\boldsymbol{p}_{mid} - \boldsymbol{p}_{start}) - 30(\boldsymbol{p}_{end} - \boldsymbol{p}_{start})) \tag{3.26}$$

$$\boldsymbol{a}_6 = -\frac{32}{{t_f}^6}(2(\boldsymbol{p}_{mid} - \boldsymbol{p}_{start}) - (\boldsymbol{p}_{end} - \boldsymbol{p}_{start})) \tag{3.27}$$

The polynomials in equations 3.19 and 3.20 need to be computed for each time-step (Δt) during the swing cycle. In order to compute the aforementioned swing trajectory polynomial over n time-steps the following time matrix was defined.

$$\boldsymbol{t} = \begin{bmatrix} 0 & (\Delta t)^1 & (2\Delta t)^1 & \dots & (n\Delta t)^1 \\ 0 & (\Delta t)^2 & (2\Delta t)^2 & \dots & (n\Delta t)^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & (\Delta t)^5 & (2\Delta t)^5 & \dots & (n\Delta t)^5 \\ 0 & (\Delta t)^6 & (2\Delta t)^6 & \dots & (n\Delta t)^6 \end{bmatrix} \tag{3.28}$$

where:

$$n = \frac{t_{swing}}{\Delta t} = \frac{(1-\beta)T}{\Delta t} \tag{3.29}$$

The points for the swing trajectory can therefore be calculated using the following equation where $\mathcal{P}$ is the sequence of points which make up the trajectory.

$$\mathcal{P}_{swing} = \boldsymbol{A} \cdot \boldsymbol{t} = \begin{bmatrix} \boldsymbol{p}_1 & \boldsymbol{p}_2 & \dots & \boldsymbol{p}_{n-1} & \boldsymbol{p}_n \end{bmatrix} \tag{3.30}$$

where:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_0 & \boldsymbol{a}_1 & \dots & \boldsymbol{a}_5 & \boldsymbol{a}_6 \end{bmatrix} \tag{3.31}$$

The sequence of velocities $\mathcal{V}$ can be calculated with a similar method.

$$\mathcal{V}_{swing} = \boldsymbol{A}' \cdot \boldsymbol{t} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \dots & \boldsymbol{v}_{n-1} & \boldsymbol{v}_n \end{bmatrix} \tag{3.32}$$

where:

$$\boldsymbol{A}' = \begin{bmatrix} \boldsymbol{a}_1 & 2\boldsymbol{a}_2 & 3\boldsymbol{a}_3 & 4\boldsymbol{a}_4 & 5\boldsymbol{a}_5 & 6\boldsymbol{a}_6 & \boldsymbol{\emptyset} \end{bmatrix} \tag{3.33}$$

Rephrasing these computations as matrix operations allows them to be computed using NumPy which is specifically optimised for matrix operations and will be substantially faster than the multiple nested for loops.

Support Trajectory

The support trajectory is the straight line motion between the start and end points with a constant velocity. The positions will therefore be evenly spaced and the NumPy function `linspace` can be used to generate this trajectory. The velocity is the same at each point and is computed using the following equations.

$$t_{support} = T\beta \tag{3.34}$$

$$\boldsymbol{v}_{support} = (\boldsymbol{p}_{end} - \boldsymbol{p}_{start}) \oslash t_{support} \tag{3.35}$$

$$\mathcal{V}_{support} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \dots & \boldsymbol{v}_{n-1} & \boldsymbol{v}_n \end{bmatrix} \tag{3.36}$$

3.4.4 Leg Inverse Kinematics

Once the desired trajectory of a foot is know, this needs to be transformed from the leg coordinate frame to the joint coordinate frame before it can be sent as an angle to each of the servos. This is achieved with inverse kinematics. The inverse kinematic solution is only required for one leg when using the legs coordinate frame.

$$x_e, y_e, z_e \rightarrow \theta_1, \theta_2, \theta_3 \tag{3.37}$$

For this it is essential to first kinematically represent the leg of the hexapod. Figure 3.5 is a kinematic representation of the hexapod leg in the leg coordinate frame with the joint axes and frames defined according to the Denavit-Hartenberg convention.

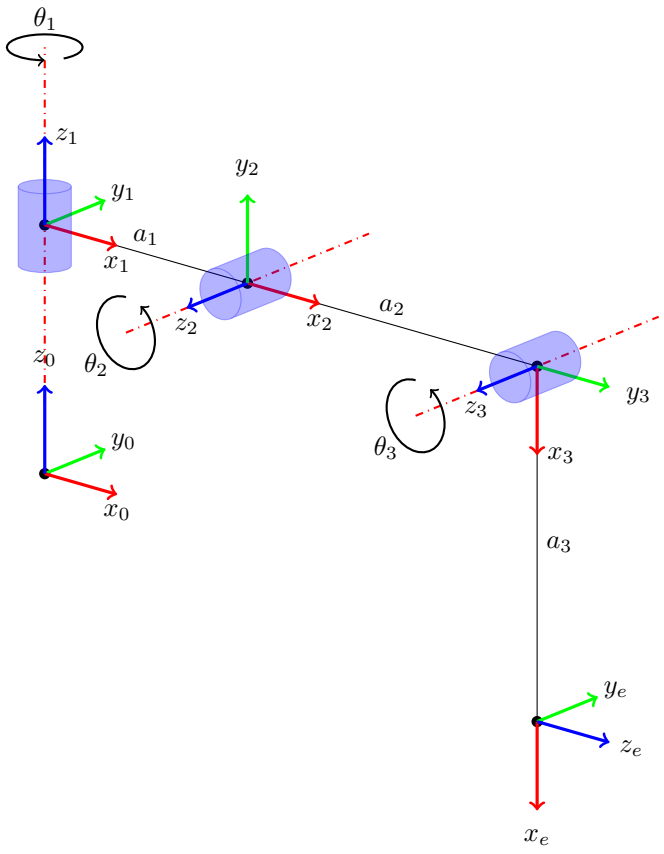


Figure 3.5: Hexapod leg kinematic representation

The following equations follow the convention defined in figure 3.5. The inverse kinematic solution for the platform was developed in [12] and is reproduced below. The *NumPy* function `arctan2` was used when implementing the inverse kinematics in *Python* to ensure an output of $\pm 180°$.

$$\theta_1 = \arctan2(y_e, x_e) \tag{3.38}$$

$$\theta_2 = \arctan2(z_e, \sqrt{(x_e - a_1c_1)^2 + (y_e - a_1s_1)^2}) - atan2(a_3s_3, a_2 + a_3c_3) \tag{3.39}$$

$$\theta_3 = \arctan2(s_3, c_3) \tag{3.40}$$

where:

$$a_1 = 5.317 \times 10^{-2}\,\text{m} \qquad a_2 = 1.019 \times 10^{-1}\,\text{m} \qquad a_3 = 1.474 \times 10^{-1}\,\text{m} \tag{3.41}$$

$$c_i = \cos(\theta_i) \qquad s_i = \sin(\theta_i) \tag{3.42}$$

$$c_3 = \frac{(x_e - a_1c_1)^2 + (y_e - a_1s_1)^2 + {z_e}^2 - {a_2}^2 - {a_3}^2}{2a_2a_3} \qquad s_3 = \pm\sqrt{1 - {c_3}^2} \tag{3.43}$$

The NumPy library was again used when implementing the inverse kinematics in Python. Consequently, the inverse kinematics can be computed on a single point or on an array of points to produce an array of joint angles.

Velocity control of the servos on the hexapod was also implemented in **??**. To ensure the closest possible match with reality the controller for the simulator should also include velocity control. The relationship between the end effector velocity and the joint angular velocities is required for velocity control.

$$\dot{x}_e, \dot{y}_e, \dot{z}_e \rightarrow \dot{\theta_1}, \dot{\theta_2}, \dot{\theta_3}$$

This relationship was also developed in [12] and is reproduced below.

$$\dot{\theta_1} = \frac{\dot{y}_e c_1 - \dot{x}_e s_1}{a_1 + a_3c_{23} + a_2c_2} \tag{3.44}$$

$$\dot{\theta_2} = \frac{1}{a_2}(\dot{z}_e c_2 - \dot{x}_e c_1 s_2 - \dot{y}_e s_1 s_2 + \frac{c_3}{s_3}(\dot{z}_e s_2 + \dot{x}_e c_1 c_2 + \dot{y}_e c_2 s_1)) \tag{3.45}$$

$$\dot{\theta_3} = -\frac{1}{a_2}(\dot{z}_e c_2 - \dot{x}_e c_1 s_2 - \dot{y}_e s_1 s_2 + \frac{a_2 + a_3c_3}{a_3 s_3}(\dot{z}_e s_2 + \dot{x}_e c_1 c_2 + \dot{y}_e c_2 s_1)) \tag{3.46}$$

where:

$$c_{ij} = \cos(\theta_i + \theta_j) \qquad s_{ij} = \sin(\theta_i + \theta_j) \tag{3.47}$$

3.4.5 Singularity Avoidance

The inverse mapping of the end effector in cartesian space to joint space can often become problematic. Only a small subset of the cartesian space is reachable by the leg, but the inverse kinematic equations accept the full cartesian space. It is therefore possible to command the leg to a position which cannot be reached which can often result in unpredictable solutions such as infinite joint velocities. These solutions are highly undesirable so a simple check is made to ensure that the cartesian position is reachable. This is done by converting the joint commands back to the cartesian space with forward kinematics and checking whether the resulting positions match the desired positions. If they do not match, then the leg was commanded outside of its reachable area and the gait controller throws an error.

3.4.6 Leg Forward Kinematics

In order to implement the aforementioned singularity avoidance the inverse kinematic solution relating the joint angles to the end effector position was required. This was developed in [12] and is reproduced below.

$$x_e = \cos(\theta_1)(L_1 + L_3\cos(\theta_2 + \theta_3) + L_2\cos(\theta_2)) \tag{3.48}$$

$$y_e = \sin(\theta_1)(L_1 + L_3\cos(\theta_2 + \theta_3) + L_2\cos(\theta_2)) \tag{3.49}$$

$$z_e = L_3\sin(\theta_2 + \theta_3) + L_2\sin(\theta_2) \tag{3.50}$$

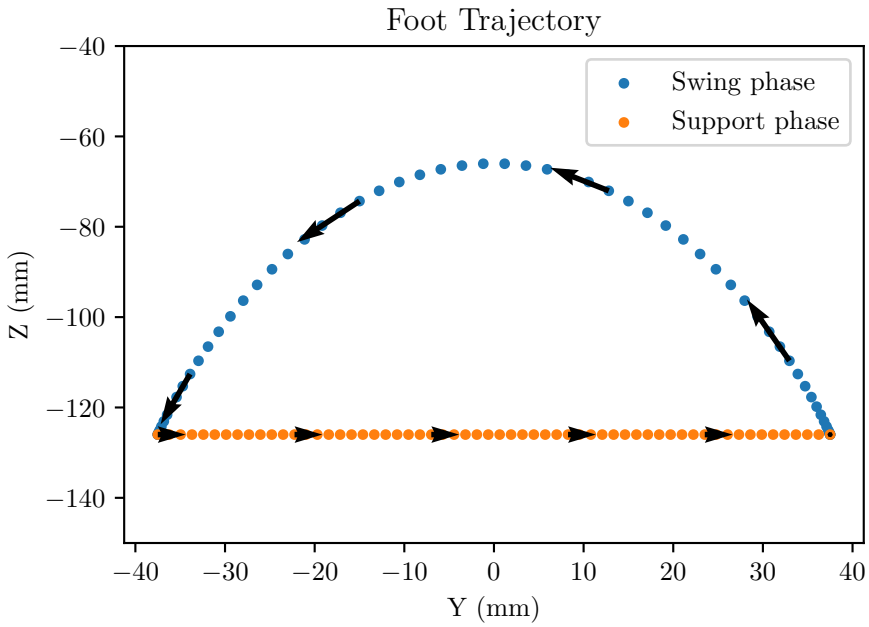


Figure 3.6: Gait controller foot trajectory plot

3.4.7 Validation

The foot trajectory calculations were checked by plotting the computed points and their corresponding velocities. This is shown in figure 3.6.

The resulting trajectory corresponded to what was intended. The spacing of the points also corresponds to the desired velocity. The goal for this gait controller is to be a highly variable parameterised controller which can be used with MAP-Elites to develop a diverse set of gaits. To validate the variability of the designed controller, the 3 hexapod gaits identified in literature were reproduced. The parameters for each of these gaits is detailed in table 3.11, 3.12, and 3.13.

3.4.8 Code

The gait controller was successfully implemented in Python using the NumPy linear algebra library. The code is in the file is called `kinematic.py` within the controllers folder.

Table 3.11: Tripod gait controller parameters

Leg	r	ϕ	s	ψ	β
1	0.15	0.0	0.06	0/2	1/2
2	0.15	0.0	0.06	1/2	1/2
3	0.15	0.0	0.06	0/2	1/2
4	0.15	0.0	0.06	1/2	1/2
5	0.15	0.0	0.06	0/2	1/2
6	0.15	0.0	0.06	1/2	1/2

Table 3.12: Wave gait controller parameters

Leg	r	ϕ	s	ψ	β
1	0.15	0.0	0.06	0/6	1/6
2	0.15	0.0	0.06	1/6	1/6
3	0.15	0.0	0.06	2/6	1/6
4	0.15	0.0	0.06	3/6	1/6
5	0.15	0.0	0.06	4/6	1/6
6	0.15	0.0	0.06	5/6	1/6

Table 3.13: Quadruped gait controller parameters

Leg	r	ϕ	s	ψ	β
1	0.15	0.0	0.06	0/3	2/3
2	0.15	0.0	0.06	1/3	2/3
3	0.15	0.0	0.06	2/3	2/3
4	0.15	0.0	0.06	0/3	2/3
5	0.15	0.0	0.06	1/3	2/3
6	0.15	0.0	0.06	2/3	2/3

3.5 Simulator Development

This section outlines the development of a simulator for the hexapod robot. This simulator connects the hexapod model to the gait controller so that gaits can be simulated. It additionally needs to be able to interface with the MAP-Elites evolutionary algorithm for the gait generation.

3.5.1 Physics Engine Selection

The physics engine is a fundamental component of a robot simulator requiring careful selection based on its application and the trade-off between accuracy and speed. Researchers at the Robotics Systems Lab at ETH Zurich developed a benchmark test for physics engines called SimBenchmark [28] which combines the speed and accuracy performance of a physics engine into a single metric for a given test, greatly assisting with comparison. The results of each of these tests have been normalised with a maximum score of 5, and are summarised in Table 3.14.

Table 3.14: SimBenchmark [28] speed-accuracy results for relevant tests

Test	**Bullet**	**ODE**	**MuJoCo**	**DART**
Friction model	3	0	1	0
Single-body elastic collision	2	3	0	1
Single-body hard contact	1	2	1	1
Single-body energy	2	3	0	1
Articulated-robot-system speed	3	1	4	2
Articulated-robot-system momentum	2	5	4	1
Articulated-robot-system energy	3	2	5	1
Average Score (/5)	2,3	2,3	2,1	1,0

Based on the results in Table 3.14 the Bullet and ODE physics engines both offer the best combined speed and accuracy performance, and will therefore be suited for this project. Bullet offers an API in Python called PyBullet[29], while ODE offers one in C. Python is preferred over C for its clean syntax, higher level of abstraction from hardware, object-oriented approach, and availability of external libraries [47]. Therefore the Bullet physics engine Python API, PyBullet [29] will be used as the physics engine for the hexapod simulator.

3.5.2 Structure

The hexapod simulator was designed independently of the gait controller. This modular design allowed each component to be designed and tested independently of one another. The responsibility of the simulator was to get the desired joint angles and speeds from the gait controller and move the hexapod model to the corresponding positions. The simulator was also required to interface with MAP-Elites. It therefore needs to provide the performance of the gait and the descriptor.

3.5.3 Initialisation

Physics Engine

PyBullet is designed using a client server API architecture. The client is exposed to the user and can be used to send commands to the server. The server is the physics engine which executes

the clients commands and returns the status of the simulation. The first step in creating the simulation is to connect a client to a physics engine.

```
import pybullet_utils.bullet_client as bc
import pybullet as p

client = bc.BulletClient(connection_mode=p.DIRECT)
client = bc.BulletClient(connection_mode=p.GUI)
```

Listing 3.4: Initialise simulation connection to PyBullet

The GUI connection mode starts a Graphical User Interface (GUI) along with the physics engine. The server state is then also rendered in a 3D visualiser window; a crucial requirement defined in the simulator Software Requirement Specification (SRS) in table 3.3. The DIRECT connection mode instead sends commands directly to the physics engine without any visualisation layer. Each instance is created based on whether the users desires a visualisation or not satisfying the SRS in table 3.3. The `BulletClient` class is used to initialise the client rather than the default implementation. This class is responsible for creating the connection to the physics engine and is also responsible for allocating computing resources. Each physics engine is single threaded and can therefore run on a single CPU. The `BulletClient` allocates each simulation to an available CPU. This function is essential for instantiating and running multiple simulations in parallel without them interfering. This satisfies the parallel computing requirement in the SRS in table 3.3.

Environment

Once a PyBullet physics engine has been instantiated and connected to, the environment within the physics engine is initialised. This is done first by adding gravity, then loading the ground plane, and lastly loading the hexapod robot from the URDF file.

```
import pybullet_data

client.setGravity(0, 0, -9.81)
groundId = client.loadURDF('plane.urdf')
client.changeDynamics(groundId, -1, lateralFriction=1.0)
position = [0, 0, controller.body_height]
orientation = client.getQuaternionFromEuler([0, 0, 0])
flags = p.URDF_USE_INERTIA_FROM_FILE | p.URDF_USE_SELF_COLLISION
hexId = client.loadURDF('/urdf/hex.urdf', position, orientation, flags=flags)
```

Listing 3.5: Initialise simulation environment

The `URDF_USE_INERTIA_FROM_FILE` flag was set to ensure that the inertia properties from the URDF file are used. Without this flag PyBullet will recalculate the inertia properties based on the mesh files assuming constant density, resulting in the incorrect dynamics. The second flag `URDF_USE_SELF_COLLISION` was set to ensure that collisions between links within the hexapod are not ignored. This is important as leg collisions are undesirable and should register as a collision. The default flat ground plane was used. Modelling a specific ground surface would introduce complexity outside the intended scope of this project.

Joints

Joint initialisation is done in the functions `get_joints` and `init_joints`. By default the joints are initialised to their zero positions based on the URDF file. They are instead initialised to their starting positions based on the gait controller. This is achieved with the function `resetJointState` to ensure dynamics are ignored. The information about all of the joints is also extracted and stored as this is needed in other parts of the simulator and is often not used by default with PyBullet. If a joint is specified as failed this is also where configuration for the failure scenario is done.

Links

Link initialisation is done in the functions `get_links` and `init_links`. The links are extracted from the URDF file. The only information required about the links are their indices used in PyBullet. PyBullet approximates the collision bounds of the links with a convex hull. This results in the femur links colliding with the base before their full range of motion has been achieved due to the concave sections in between the legs on the base. To get around this the collisions between the base link and the femurs is filtered with `setCollisionFilterPair` during initialisation of the links. The femur and base should not collide as a result of the joint limits assigned when defining the hexapod model.

3.5.4 Friction

The friction coefficient between the feet of the hexapod and the ground is not known as the ground surface can vary, however this information is required for the physics engine. A `lateralFriction` coefficient of 1 was assigned for the ground plane and the tibia links. This was done to minimise sliding and slipping of the feet.

3.5.5 Joint Control

The joints in the physical hexapod robot are controlled with 18 servo motors. These servo motors receive desired position and desired velocity commands and move to satisfy both of these commands. Their movement is governed within the servos by an internal Proportional Integral Derivative (PID) control loop to ensure that the control commands are maintained [40][41]. This actuation method is very common in robotics and PyBullet provides a built-in `POSITION_CONTROL` mode to replicate this. This control function accepts a desired position command and a desired velocity command and uses a Proportional Derivative (PD) controller to move to these commands. While this PD control method is not identical to the PID controller used in the servos it provides a reasonable approximation. At each simulation step new desired joint positions and velocities for each of the 18 servos are fetched from the gait controller. These positions and velocities are sent to the physics engine using the function `setJointMotorControl2`. The function also accepts maximum torque and speed parameters. These are essential in ensuring the simulation models the actuators accurately. The joint torques and speeds from the URDF model of the hexapod are used.

3.5.6 Simulation Stepping

The progression of time in the simulation is achieved with the `step` function which is summarised in 3.6. This function is responsible for advancing the simulation one time-step. PyBullet uses a default time-step of $\frac{1}{240}$ s. This time-step is a balance between performance and accuracy and was therefore left as default. The `step` function gets the joint angles and velocities from the controller and moves the joints to their corresponding positions using the aforementioned method. Collisions between the links in the robot and between the ground are also queried. If ones of these collisions has occurred the step function raises an error to meet the SRS in table 3.3. The visualiser camera position is also updated to the robots position to enable tracking. Finally the physics engine is stepped and the current time of the simulator is incremented. This is a fundamental function in the simulation and is therefore detailed in listing 3.6.

```
10 def step(self):
11  for index, joint in enumerate(self.joints):
12  joint_index, lower_limit, upper_limit, max_torque, max_speed = joint
13
14  if (joint_index is None) or (index in self.locked_joints) or (index in self.failed_joints): continue
15
16  joint_angle = joint_angles[index]
17  joint_speed = joint_speeds[index]
18
19  joint_angle = min(max(lower_limit, joint_angle), upper_limit)
20  joint_speed = min(max(-max_speed, joint_speed), +max_speed)
21
22  self.client.setJointMotorControl2(self.hexId, joint_index, p.POSITION_CONTROL, targetPosition=joint_angle,
        force=max_torque, maxVelocity=max_speed)
23
24  if self.collision_fatal:
25   if self.__link_collision() or self.__ground_collision():
26    raise RuntimeError("Link collision during simulation")
27
28  if self.visualiser_enabled:
29   self.client.resetDebugVisualizerCamera(cameraDistance=self.camera_distance, cameraYaw=self.camera_yaw,
       cameraPitch=self.camera_pitch, cameraTargetPosition=self.base_pos())
30
31  self.client.stepSimulation()
32  self.t += self.dt
```

Listing 3.6: Step function

3.5.7 Supporting Legs

Information about which of the legs is in contact with the ground is required to determined a description of a gait and produce the footfall diagram. A leg ground contact was determined by querying the contacts between the tibia links and the ground plane with `getContactPoints`. The function `supporting_legs` returns an array of leg contact booleans where the element index corresponds to the leg number.

$$\begin{bmatrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{bmatrix} \tag{3.51}$$

3.5.8 Performance Optimisation

The simulator is run for millions of iterations so any performance improvements which reduce runtime in a single simulation will be compounded and significantly beneficial. PyBullet includes a time profiling feature which shows a breakdown of what is using computing time. This was run for a 5 second simulation of the hexapod walking to determine if any performance improvements could be made. The time profile is shown in figure 3.7.

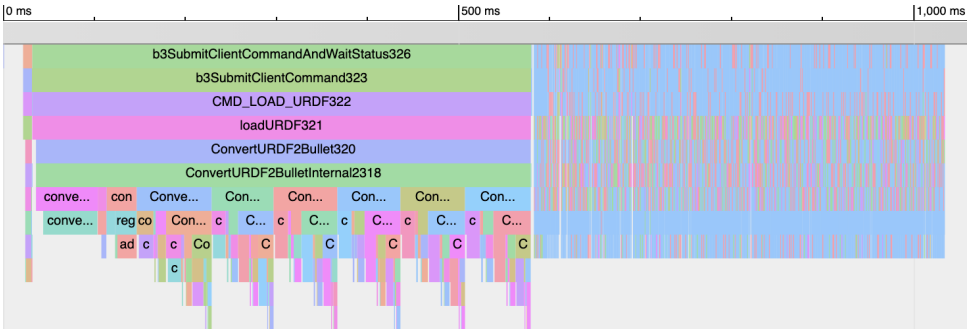


Figure 3.7: PyBullet unoptimised time profile

The time profile in figure 3.7 shows that 540 ms is being used on the function `loadURDF` which is more than half of the simulation duration. The most complex component of the URDF file is the STL mesh files for each of the links. The loading time of the meshes can be shortened by reducing the mesh complexity. An STL file approximates the surface of a solid object with triangles as this is the minimum representation of a 2D surface. The complexity of an STL file is determined by the quantity of triangles. Multiple triangles are needed to approximate curved surfaces, therefore simplifying the links into flatter surfaces would substantially reduce the mesh complexity and the simulator runtime. This geometry simplification was done for all of the links in SolidWorks and is shown in figure 3.9. Care was taken to ensure that the overall bounds of the links remained unchanged so as to not have an effect on collisions. Additionally the geometry of the end of the tibia which contacts the ground was left unchanged to not alter the foot contact dynamics. Table 3.15 shows the differences in the number of triangles and the file size of the STL files as a result of the geometry simplification. From table 3.15 is it evident that the geometry simplifications resulted in a dramatic decrease in the number of triangles and therefore the complexity of the STL files. The time profile for the same test simulation was generated using the simplified mesh files in the URDF and is shown in figure 3.8.

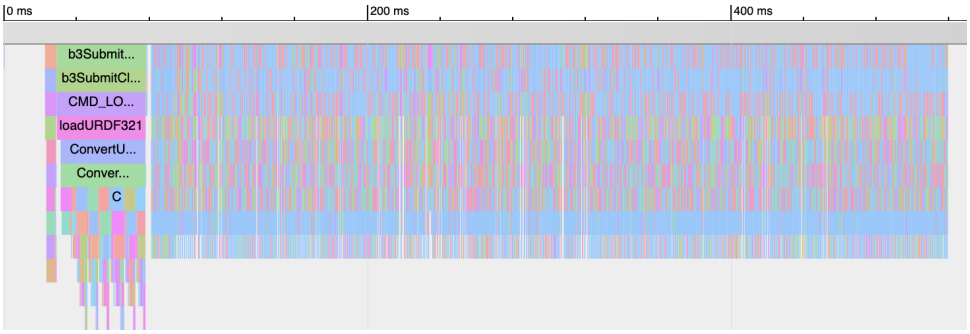
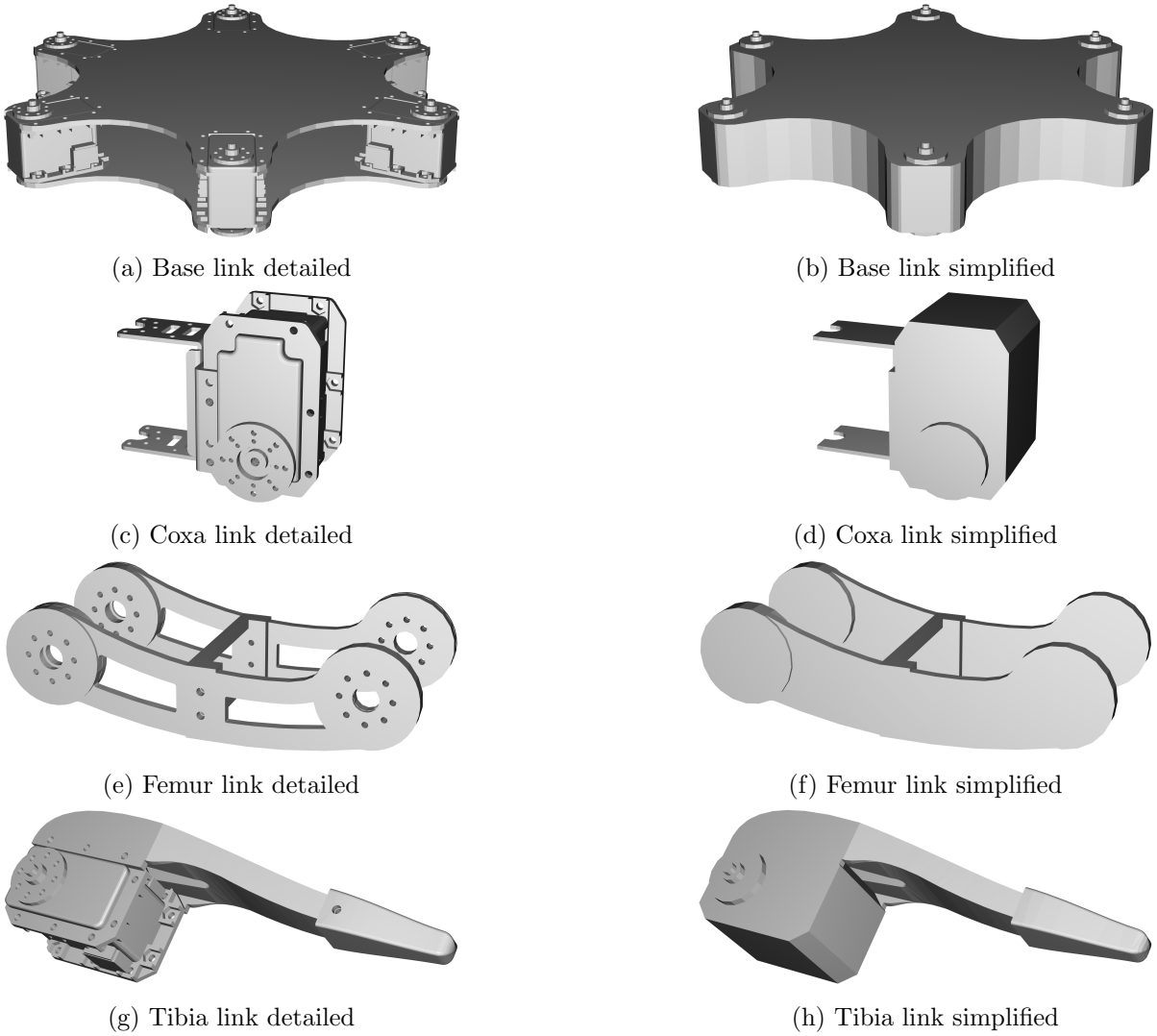


Figure 3.8: PyBullet optimised time profile

The simplification of the mesh files reduced the time used for `loadURDF` to 49 ms, a tenth of the original duration. The runtime of the simulation was more than halved as a result of this optimisation.



(a) Base link detailed

(b) Base link simplified

(c) Coxa link detailed

(d) Coxa link simplified

(e) Femur link detailed

(f) Femur link simplified

(g) Tibia link detailed

(h) Tibia link simplified

Figure 3.9: Link mesh geometry simplification

Table 3.15: Link STL mesh complexity comparison

Link	**Detailed**		**Simplified**	
	Triangles	Size	Triangles	Size
Base	38972	1.9 MB	2252	115 KB
Coxa	13460	676 KB	440	25 KB
Femur	10692	537 KB	658	37 KB
Tibia	7350	369 KB	1500	78 KB

3.5.9 Validation

The correct functioning of the simulator was validated by monitoring the movements of the hexapod in the visualiser. An image of the visualiser is shown in figure 3.10 with each of the link types coloured differently so they can be easily distinguished.

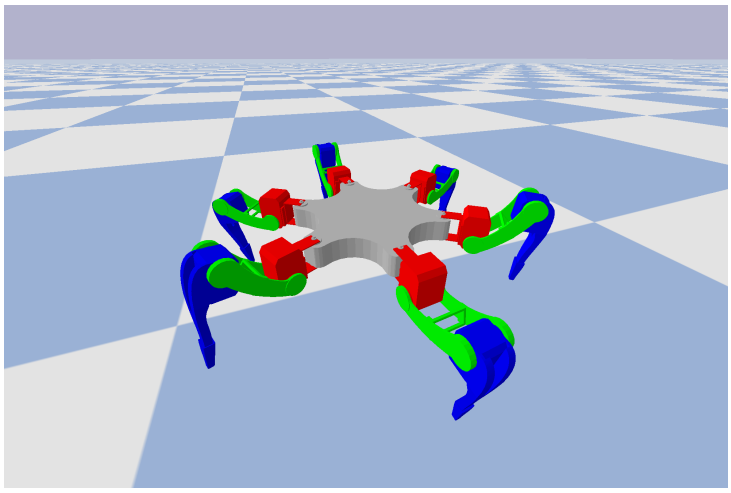


Figure 3.10: Hexapod simulator visualiser

The foot contacts were validated by simulating a gait and ensuring the foot contact sequence corresponded returned by `supporting_legs` to the pattern expected in the footfall diagram for that gait. The regular tripod gait specified in table 3.11 was simulated for 3 s with a period of 1 s for all of the legs. The resulting footfall diagram generated from the simulator foot contacts is shown in figure 3.11.

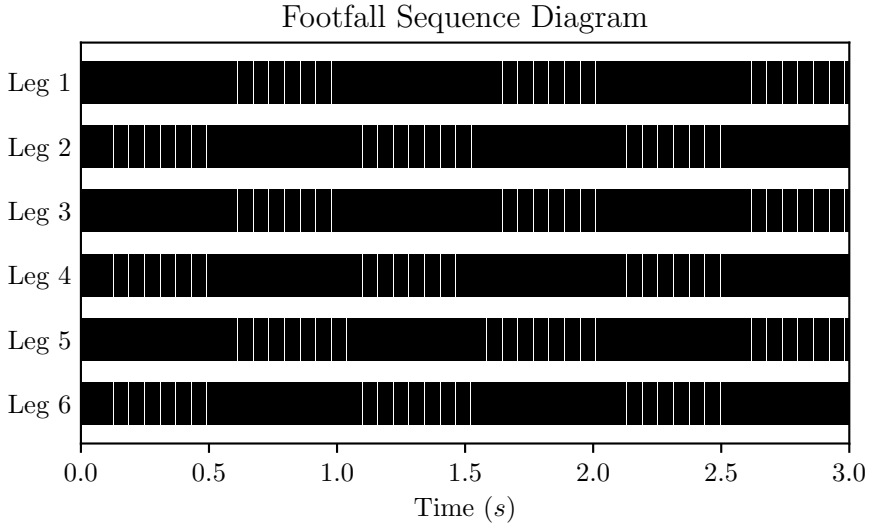


Figure 3.11: Simulator foot contact sequence diagram
The instance when a foot is in contact with the ground is shown in black

The foot contact sequence in figure 3.11 matches the tripod gait footfall diagram identified from literature in figure 2.2. All of the feet in each tripod are in contact with the ground, each foot has a consistent contact duration, and pattern matches the desired gait. The only issue is the slight overlap of sequences, however this can be attributed to the transition between tripods. This simultaneously serves as additional validation of the correct functioning of the

gait controller.

Chapter 4

Implementation

This chapter details the integration of the simulation with the MAP-Elites algorithm and the generation of the behaviour-performance maps for use in the online adaptation algorithm.

4.1 Interfacing with MAP-Elites

At its crux the MAP-Elites algorithm works by trying a set of gait parameters on the hexapod, seeing how they perform and what gait they produce, and recording this to slowly generate a diverse map of high performing gaits through an evolutionary process. The MAP-Elites algorithm is however not limited to hexapod robots and can be used to find diverse sets of optimal solutions for any general function ($\mathcal{F}$), provided the function returns a fitness (f) and behaviour descriptor ($\mathbf{b}$) for a given set of parameters ($\mathbf{x}$).

$$f, \mathbf{b} = \mathcal{F}(\mathbf{x}) \tag{4.1}$$

Due to the vast quantity of gait parameters to be tested, the map generation process is done with the prior developed simulator and gait controller. Therefore in order to interface the simulator and controller with MAP-Elites they need to be combined into a function of the above form which returns a fitness and a descriptor.

4.1.1 Fitness

The fitness or performance of a gait (f) can be quantified by a variety of metrics such as robot speed, stability, cost of transport, etc. based on the aims of the platform. The aim of this project is for the hexapod to remain mobile irrespective of leg failures. The experiments in [13] looked at the velocity of the robot along an axis to gauge the performance or fitness of the gait. This same metric is used for this project to remain consistent to the implementation in [13] in order to compare results and also to avoid the complexity associated with fitness function design. The fitness is therefore returned from the simulator as the distance traveled along the x-axis which can be divided by the simulation duration to calculate the average velocity of the robot. The robot gait is simulated for 5 seconds with 5 gait cycles to avoid transient motions associated with sudden starts and ends.

4.1.2 Behaviour Descriptor

The behaviour descriptor is what MAP-Elites uses to distinguish one gait from another and it is crucial in the development of diversity. In addition to diversifying, the descriptor also provides a substantial reduction in the dimensionality for the adaptation problem. Again there are a variety of metrics by which to distinguish the gaits resulting from a set of parameters such as the leg duty factor, body orientation, foot ground reaction force, leg angles, etc. The duty factor

descriptor was used in [13] and found to be suitable for legged robot adaptation. Furthermore using the same descriptor ensures consistency between implementations allowing comparison of results. A gait is therefore described by the percentage of time each foot spends in contact with the ground, represented mathematically as the following.

$$\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_6 \end{bmatrix} = \begin{bmatrix} \frac{\sum c_1(t)}{n_t} \\ \vdots \\ \frac{\sum c_6(t)}{n_t} \end{bmatrix} \tag{4.2}$$

Where $c_1(t)$ represents a boolean of the foot contact with the ground at time-step t, and n_t is the number of time-steps. The function `supporting_legs()` in the simulator was built for this purpose and returns an array of boolean values representing whether the leg is in contact with the ground at that time instance. The final descriptor is the average of the boolean contact sequence for each leg over the duration of the simulation. This can be thought of as the average colour of the footfall diagram per leg.

4.1.3 Gait Parameters

MAP-Elites produces the behaviour-performance maps by adjusting the parameters provided to the gait controller, represented by $\mathbf{x}$. The gait parameters were selected during the gait controller design and their justifications can be found there. There are a total of 32 parameters; 2 parameters describing whole body motion, and 5 parameters describing the individual motions of each of the 6 legs. MAP-Elites is therefore configured to optimise over these 32 parameters resulting in $\mathbf{x}$ being of the following form;

$$\mathbf{x} = [x_1, x_2, \ldots, x_{31}, x_{32}] \tag{4.3}$$

where: $x \in [0, 1]$

The issue with this is that the gait controller parameter value scales and ranges are not compatible with the provided range of values for x of [0,1]. Additionally the range of values for the body height is not the same as the range of values for the leg offset angles. To interface the gait controller with MAP-Elites it is therefore essential to select a range for each of the parameters and scale x to a range accepted by the gait controller. However, this is open to unintentionally over-constraining the gait controller to a desired gait and reducing the space of optimal solutions. To avoid this care was taken when defining each of the parameter ranges to ensure that the justification was predominantly based on a morphological limitations to limit unintended biases.

Leg Radius Range – The leg radius range was based on the maximum possible extension of the leg of the hexapod shown in figure 4.1. Based on this the allowable range for this parameter was chosen to be 0mm to 300mm. The lower bound of 0mm was chosen as negative values could result in the gait path being inside or underneath the hexapod body.

Leg Offset Angle Rangle – The offset angle range for a leg is limited by the range of motion of the coxa joint. This was found to be -90° to +90° using the SolidWorks model and is summarised in hexapod model development section in table 3.8.

Step Height – The leg workspace shown in figure 4.1 was also used to determine the step height range. It was taken to be half of the maximum vertical reach for the leg; approximately 200mm.

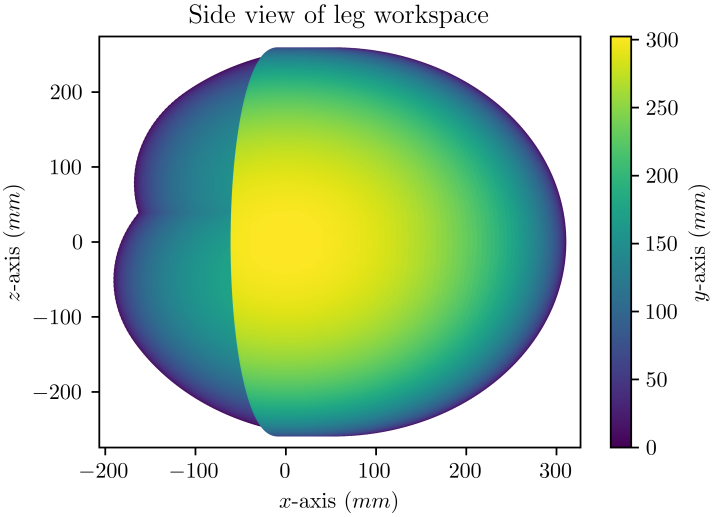


Figure 4.1: Side view of the hexapod leg workspace in the joint coordinate frame

Phase Offset – In the gait controller the phase offset was defined as a value between 0 and 1 for each of the legs and therefore does not need to be changed to interface with MAP-Elites.

Duty Factor – In the gait controller the duty factor was also defined as a value between 0 and 1, and therefore also does not need to be changed to interface with MAP-Elites.

Body Velocity – The upper bound for the body velocity was set to 0.5 m/s as it was found experimentally that it was not possible to produce gaits faster than this as legs would collide with each other and the body. This was later confirmed as the fastest gait found by MAP-Elites without collisions was around 4.5 m/s.

Body Height – Similar to the step height the maximum body height was set to half of the maximum vertical reach for a leg, resulting in a range of 0mm to 200mm.

These gait parameter ranges are summarised in table 4.1.

Table 4.1: Gait parameter range

Parameter	**Range**			**Units**
Foot Velocity	0	to	0.5	$\mathrm{m\,s^{-1}}$
Body Height	0	to	200	mm
Radius	0	to	300	mm
Offset Angle	-1.745	to	1.745	rad
Step Height	0	to	200	mm
Phase Offset	0	to	1.0	-
Duty Factor	0	to	1.0	-

The function `rescale(x)` is responsible for transforming the parameters from MAP-Elites into the ranges defined in table 4.1 for the gait controller.

4.1.4 Evaluation Function

The pseudocode in listing 4 describes the function used to evaluate a set of gait parameters ($\mathbf{x}$) in MAP-Elites and return the fitness (f) and the descriptor ($\mathbf{d}$) for that gait.

Algorithm 4 Gait evaluation function

function EVALUATE_GAIT($\mathbf{x}$)
  $\mathbf{x}' \leftarrow$ reshape($\mathbf{x}$) ▷ Parameters range adjusted for controller
  $\mathcal{C} \leftarrow$ controller($\mathbf{x}'$) ▷ Controller initialised with parameters
  $\mathcal{S} \leftarrow$ simulator($\mathcal{C}$) ▷ Simulator initialised with controller
  $f \leftarrow 0$ ▷ Initialise fitness
  $\mathbf{c} \leftarrow [...]$ ▷ Initialise array to store contact sequence
  for $t \leftarrow 0 \dots 5s$ **do**
    $\mathcal{S}$.step() ▷ Step simulation
    if collision **then**
      return 0, $\emptyset$
    $\mathbf{c}$.append($\mathcal{S}$.supporting_feet()) ▷ Update foot contacts
    $f \leftarrow \mathcal{S}$.base_position().x ▷ Update distance travelled along x-axis
  $\mathbf{d} \leftarrow$ sum($\mathbf{c}$)/length($\mathbf{c}$) ▷ Convert contact sequence into duty factor
  $\mathcal{S}$.terminate() ▷ Terminate simulation to free up resources
  return f, $\mathbf{d}$

The Python implementation of this function can be found in appendix 5.2.

4.2 Generating Maps

The function described in listing 4 is passed as a parameter into MAP-Elites and is used in generating the behaviour-performance maps. This is shown in the Python code listing below. The Python implementation of MAP-Elites was used as it allows for easy interfacing with the simulation.

```
import map_elites.cvt as cvt_map_elites
archive = cvt_map_elites.compute(6, 32, evaluate_gait, n_niches, max_evals, log_file, params)
```

Listing 4.1: MAP-Elites Python usage

Along with the evaluation function, MAP-Elites requires a number of other parameters. The parameters used in this implementation are summarised in table 4.2 and are shown alongside the parameters used in the previous adaptation implementation by Cully et al. for comparison.

The number of controller dimensions could not be identical due to the different constraints of the kinematic controller compared to the sinusoidal periodic controller used in the previous implementation. The differences in the CVT sample size, batch sizes, and random initialisation percentage are explained in subsequent sections.

4.2.1 Centroids

The Centroidal Voronoi Tessellation (CVT) using the K-means clustering method is used to evenly divide up the multidimensional gait descriptor space into the desired number of distinct niches. K-means clustering is an iterative algorithm and the higher the number of samples, the more accurate the final result, but also the more memory required to compute. Therefore a high memory Google Compute Engine Virtual Machine (VM) was created to compute an accurate CVT. The specifications of this computer can be found in appendix **??**.

Table 4.2: MAP-Elites parameters

Parameter	**Cully et al.[13]**	**Current**
Map Dimensions	6	6
Controller Dimensions	36	32
Number of Niches	40000	40000
Evaluations	4×10^7	4×10^7
CVT Samples	4×10^5	1.500×10^7
Batch Size	100	2390
Random Init	400	1%
Random Init Batch	100	2390
Parallel	True	True
Cache CVT	True	True
Parameter Min	0.0	0.0
Parameter Max	1.0	1.0

Using this VM a CVT was computed with 15 million samples with a peak RAM usage of 81% of 624 GB. Generating the CVT uses a substantial amount of computing resources and takes a significant portion of time. Therefore one CVT was generated using the VM specified in table B.1 and cached for use across all of the map generations. This file is called `centroids_40000_6.dat`.

4.2.2 Migration to a Cluster

MAP-Elites has to run a simulation to test each gait parameter, and it runs a total of 40 million of these gait parameter iterations. Consequently a substantial amount of CPU time goes into the generation of a single map, with the prior implementation taking 2 weeks per map on a hyperthreaded 16-core desktop computer. Furthermore, each map begins randomly so multiple maps need to be generated to analyse the stochasticity and investigate reproducibility.

The MAP-Elites algorithm supports parallelisation with each CPU running a separate simulated gait evaluation. This provides an enormous speed-up as multiple gait simulations can be run simultaneously, increasing computation without increasing runtime. This is implemented with the `map` function from the Multiprocessing library in Python.

The South African Lengau Cluster was used to generate each of the maps. The cluster enabled access to 240 cores across 10 nodes. The full specifications of the cluster can be found in table B.2 This reduced the computation time of a single map to approximately 40 hours. MAP-Elites supports parallelisation, however this was on multiple cores on a single machine. A cluster is made up of multiple machines communicating through a shared network. The Multiprocessing library will not be able to utilise all of the available cores as it cannot communicate to other nodes across this shared network. To communicate between nodes the cluster supports the Message Passing Interface (MPI). The Multiprocessing library was therefore replaced with the MPI for Python[48] library which was relatively uncomplicated as it also provided a `map` function which handled evaluation of the simulations across all of the available CPU's. The generate map Python script was run under command line control of `mpi4py.futures`. This would initialise an MPI process on each core, with one process being assigned the master and the others being assigned as workers. The master was responsible for distributing simulations to the workers who execute them and return the result. For a 240 CPU configuration this meant that 239

simulations could be run simultaneously.

4.2.3 Batch Size Selection

With MAP-Elites a batch represents the number of gait simulations to evaluate before updating the behaviour-performance map. Batches in MAP-Elites serve no purpose other than to group simulations for parallel evaluations. To ensure that no cores were left idle an initial batch size of 239 was selected with each core being assigned 1 simulation. This however resulted in the map generation not finishing within the 48 hour job time limit. It was speculated that the overhead associated with distributing the simulations from the master to the workers over the network and receiving the results was larger than anticipated. To mitigate this the batch size was increased to 2390, reducing communication frequency between the master and worker nodes by an order of magnitude. This solved the issue and map generation completed within the 48 hour job time limit. How the algorithm was parallelised on a computing cluster is shown in figure 4.2.

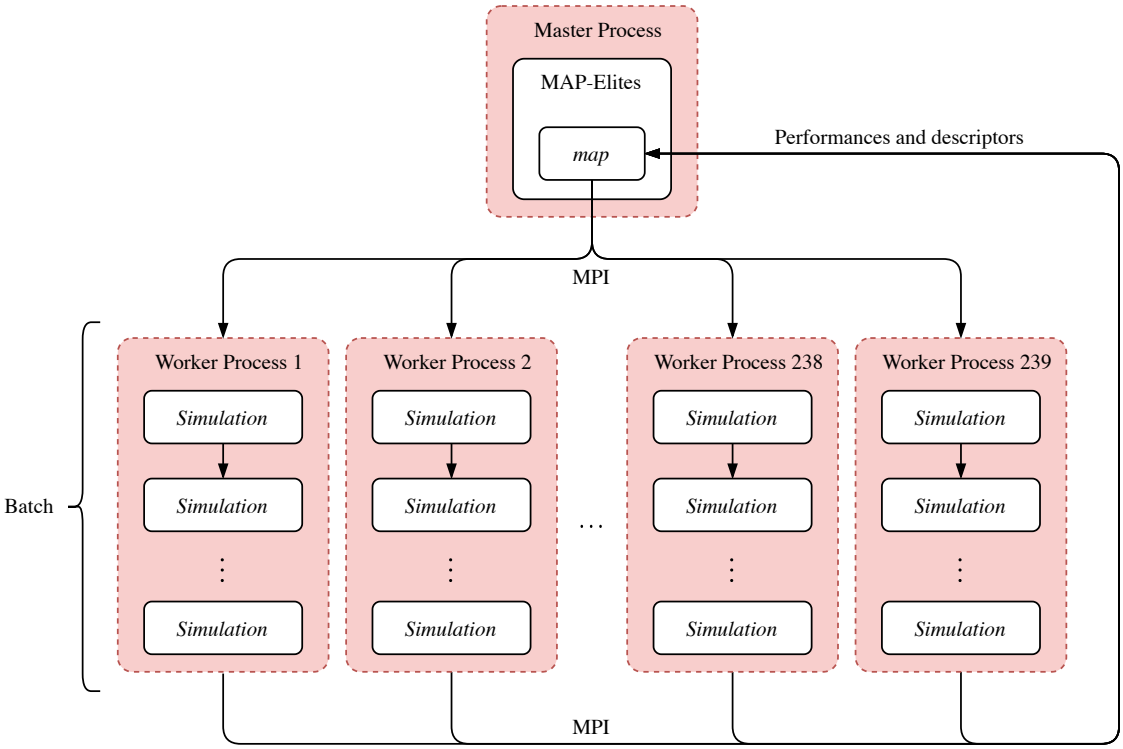


Figure 4.2: Cluster parallelisation configuration

A batch size of 2390 simulations results in 10 sequential simulations on each of the 239 worker process before the gait performance and descriptor results are returned to the MAP-Elites algorithm on the master process.

4.2.4 Map Initialisation

The previous implementation initialised the behaviour-performance map with 400 randomly generated gaits. The Python version of MAP-Elites randomly generates gaits until a desired percentage of the map has been filled. The impact of different percentages of random initialisation was not known so one map was generated with 1% random initialisation and the other with 10% random initialisation for comparison. A graph showing how the number of generated gaits changes with time for both initialisation percentages is shown in figure 4.3.

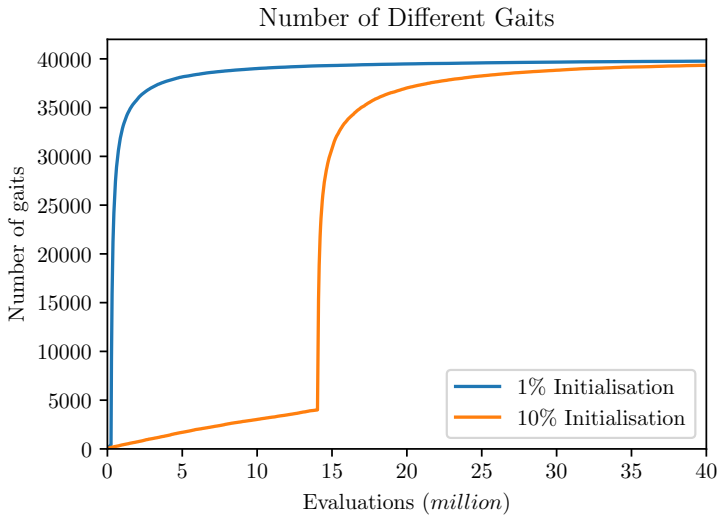


Figure 4.3: Progression of the number of different gaits for 1% and 10% random map initialisation

Both maps reach approximately the same final number of different gaits. The percentage of initialisation also appears to have no effect on the rate of gait diversification once random initialisation ends. The 14 million evaluations used to reach a 10% initialisation could possibly be better used to optimise gaits already in the map. Figure 4.4 shows the impact of these wasted evaluations on the mean performance.

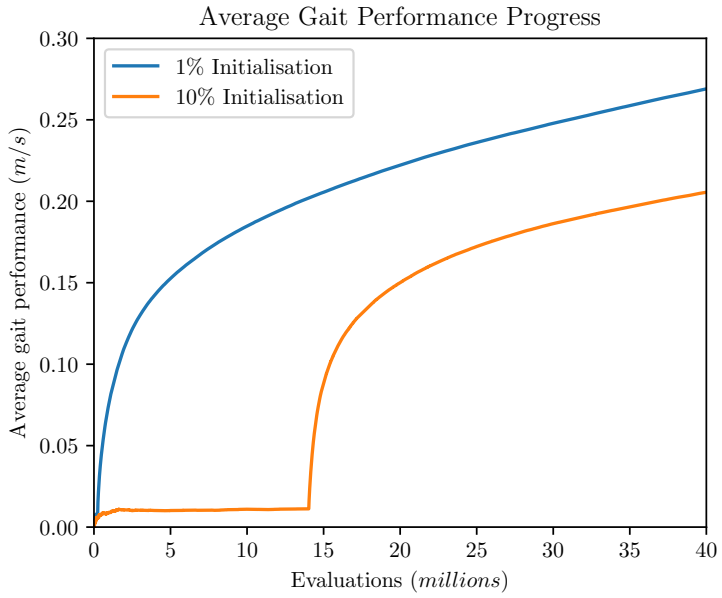


Figure 4.4: Progression of the mean gait fitness for 1% and 10% random map initialisation

Figure 4.4 shows that random map initialisation produces low performing gaits and provides minimal performance improvements. A larger infill percentage simply delays the optimisation resulting in a lower average final gait performance in the map. From figure 4.3 and figure 4.4 a random initialisation of 1% appears adequate so this value was used for the final maps as shown in table 4.2.

4.2.5 Base Collisions

Gaits where the legs collided were to be avoided as this could result in damage when implemented on the physical robot. To ensure this, the simulator was queried for collisions between the legs at each time-step, and if one had occurred, the simulator was terminated and a fitness of 0 was returned. After generation of the first test map it was evident that collisions between the base of the robot and the ground had been overlooked; MAP-Elites had produced a number of high performing gaits by simply sliding along the ground on the base. This result was unintended and a check for collisions between the base and the ground was subsequently added with the same procedure as the leg collisions. This was effective in disincentivising this behaviour.

4.2.6 Map Stochasticity

Each map generation begins by initialising a certain percentage of the map with randomly generated gaits, acting as a seed for the MAP-Elites algorithm. Each map will therefore be different and unique. For experimental repeatability 10 maps were generated on the Lengau cluster with identical configurations to investigate the effects of the stochasticity.

4.2.7 Output Files

MAP-Elites produces two data files. The first of these files is the log file which records the overall performance of the map after each batch. After each batch a row is appended to the log file with the following format.

$$\text{row}: \quad \overbrace{i_1}^{\text{iterations}} \quad \overbrace{n_2}^{\text{niches}} \quad \overbrace{f_3}^{\text{max}} \quad \overbrace{\overline{f}_4}^{\text{mean}} \quad \overbrace{\tilde{f}_5}^{\text{median}} \quad \overbrace{f_6}^{5^{\text{th}}\ \%}$$

Figure 4.5: MAP-Elites log file row format

The above log file is used for evaluating the progress of MAP-Elites. The second output file is the behaviour-performance map. This file contains the highest performing gait per gait niche, with each gait niche being on a separate row. The structure for each row is shown below.

$$\text{row}: \quad \overbrace{f_1}^{\text{fitness}} \quad \overbrace{d_2 \quad \dots \quad d_7}^{\text{descriptor}} \quad \overbrace{c_8 \quad \dots \quad c_{13}}^{\text{centroids}} \quad \overbrace{p_{14} \quad \dots \quad p_{45}}^{\text{gait parameters}}$$

Figure 4.6: MAP-Elites behaviour-performance map file row format

The descriptor for a gait as well as the closest centroid in the CVT it was assigned are stored for each gait.

4.2.8 Map Generation Performance

The CHPC completed generation of the 10 maps in 402 hours with an average peak RAM usage of 151 GB and an average CPU utilisation of 39.5%. The detailed performance metrics for each map can be found in table B.2. The low CPU utilisation can potentially be explained by the following; If a simulation terminates prematurely due to a collision or a controller singularity then a CPU will be left idle until the remaining worker processes in the batch finish executing as the distribution of a batch of simulations and collection of the results happens synchronously. Asynchronous allocation of simulations should therefore improve efficiency.

The aim of MAP-Elites is to produce a diverse set of high performing gaits. The performance of the gaits within the maps is therefore an important metric in gauging the success of MAP-Elites. The performance progression for the distribution of the 10 independently generated maps is plotted in figure 4.7.

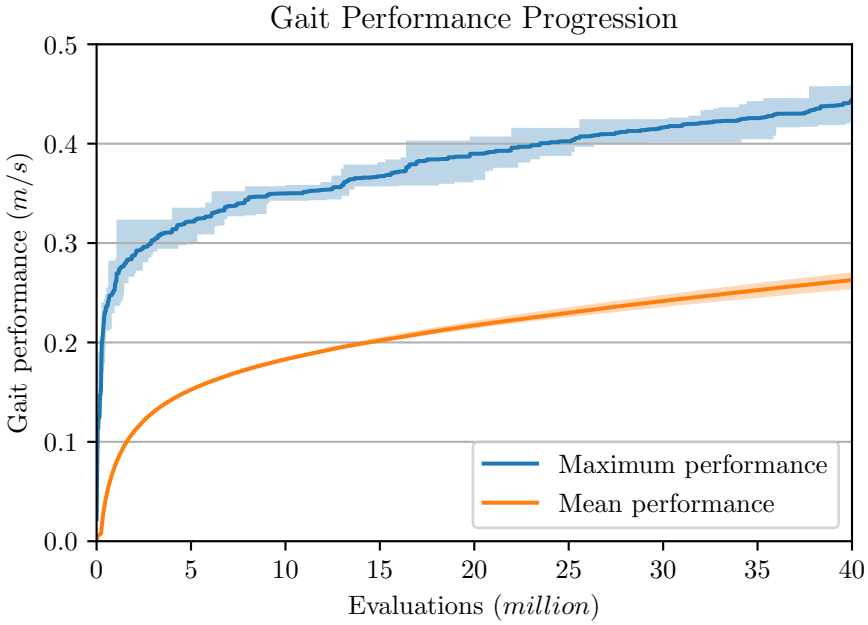


Figure 4.7: Gait performance progression for 10 independent maps
The shaded regions show the distribution across the 10 independently generated maps and the solid lines show the mean of the distribution for the 10 maps.

There is only a small amount of variation in the gait fitness between the 10 independently generated behaviour-performance maps. The variation between the mean fitness of the maps is barely noticeable. This shows that MAP-Elites is able to consistently generate behaviour-performance maps for the Hexapod using the custom simulator and gait controller. Additionally the final performance of the maps after the 40 million iterations is on average 0.44 m/s which is close to the maximum foot velocity of 0.5 m/s assigned in 4.1. The selected number of iterations for MAP-Elites is therefore sufficient to find an optimum.

The second objective of MAP-Elites is to produce a diverse set of gaits. Its performance in this regard is therefore also evaluated. The progression of the number of unique gaits for each map is plotted in figure 4.8

Each map has 40000 niches with each niche corresponding to a unique gait. An average of 39756 unique gaits is found across the 10 independently generated maps. Approximately 99% of the total unique gait niches have therefore been filled for each of the maps, showing a high degree of diversity. This validates the ability of the custom gait controller to produce a wide variety of gaits. Additionally the shaded region showing the maximum and minimum region is barely visible further validating the consistency between independently generated maps.

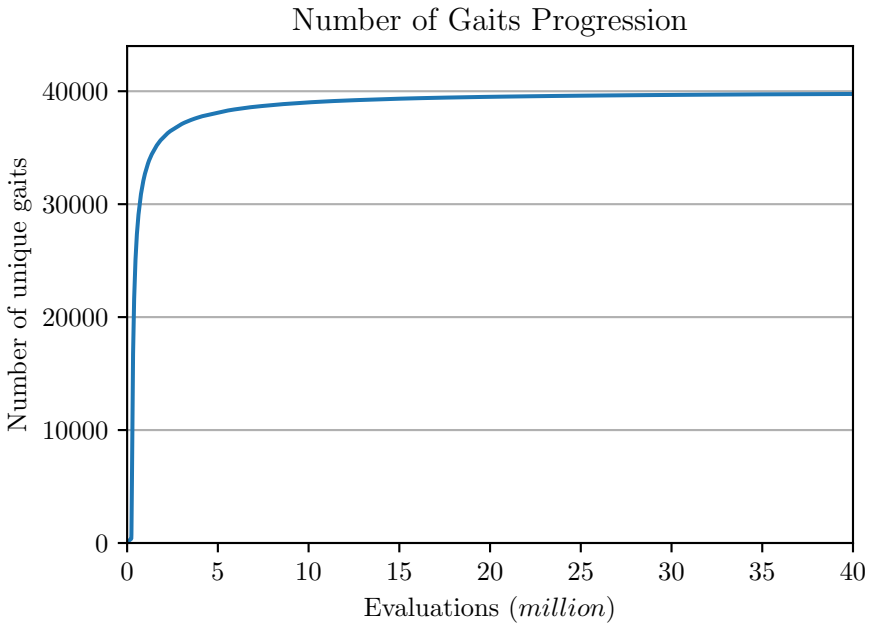


Figure 4.8: Number of gaits progression for 10 independent maps
The shaded region shows the distribution across the 10 independently generated maps and the solid line shows the mean of the distribution for the 10 maps.

4.3 Hexapod Physical Implementation

This section describes the implementation of the dynamic gait controller on the RARL Hexapod platform. The code for the platform was developed by [12] and a substantial amount of work was done to provide flexible high-level control of the platform, abstracting the low-level code associate with embedded programming. This drastically simplified modification of the code.

4.3.1 Gait Controller Modification

The hexapod code was written in C++ and made use of classes for a modular software design. Each leg was associated with a `Leg` class which was responsible for generating the servo position sequence, computing inverse kinematics, and communicating with the servos for that leg. As a result of this modular design modification of the code to produce unique gait trajectories for each leg was dramatically simplified.

The `makePath` function within the `Leg` class accepts the gait parameters for that leg such as its radial distance from the body or the step height and generates the corresponding trajectory. In the prior system the same parameters were used across all of the legs resulting in a symmetrical gait. These parameters were hand selected.

Rather than the same value being passed to all of the legs, an array of unique parameters was stored in the `Hexapod` class with each leg being passed its corresponding parameter.

This function was the opportunity to provide unique parameters for each leg

makePath calls the setPoints function within the Path class which generates the start, mid, and end points for the leg trajectory. The makePath function within the `Path` class uses the start, mid, and end points to generate the remaining points for the swing and support trajectories. The only modification required for the `makePath` function was the inclusion of a dutyFactor and gaitPeriod parameter to control the duration of the leg's trajectory and the ratio of swing phase to support phase.

Old	New
strideLength	footVelocity
bodyHeight	bodyHeight
direction	crabAngle
radialDistance	radialDistance
stepHeight	stepHeight
slopePitch	slopePitch
slopeRoll	slopeRoll
	gaitPeriod
	angleOffset
	phaseOffset
	dutyFactor

The trajectory for a joint is stored as a sequence of joint angles.

$$\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_n] \tag{4.4}$$

The desired motion is achieved by iterating through this sequence at a fixed controller frequency and sending the current position to the required servo. The sequence is repeated to produce a continuous cyclic motion. The previous gait controller used a frequency of 14 Hz as this was adequate for the intended application. The simulator used to develop the gaits for adaptation ran at a frequency of 240 Hz. To remain consistent the gait controller on the hexapod should use the same controller frequency, however this resulted in the trajectory arrays exceeded the available RAM on the microcontroller. A frequency of 120 Hz was instead used. This frequency was modified by setting the period of the gait controller timer to 8 ms.

$$T_{gait} = \frac{1}{120\ Hz} \approx 8\ ms \tag{4.5}$$

4.3.2 Servo Speed Control

The speed control of the servos on the physical system was not producing the desired motions with some servos moving slower than others even with the same speed commands. There was insufficient time remaining to find the root of this issue so speed control was removed. This was done by setting the servos to their maximum speeds as specified in table 3.9 in section 3.3.3. The sequencing of the servo positions provided acceptable foot velocity control due to the relatively high gait frequency.

The differences between the simulated hexapod gait controller and the real hexapod gait controller are summarised below in table 4.3.

Table 4.3: Hexapod simulated and real implementation comparison

Characteristic	**Simulated**	**Real**
Controller frequency	240 Hz	120 Hz
Joint control	position, velocity	position

4.3.3 Interfacing with M-BOA

The Map Based Bayesian Optimisation Algorithm (M-BOA) requires feedback on the walking performance of the hexapod robot to update the gaussian process and guide the selection of compensatory gaits, as well as a means of transferring new gait parameters to the robot.

Performance Feedback

It was initially intended that the IMU would provide velocity feedback through dead reckoning. This would only need to be an average velocity over 5 seconds so integration drift would be negligible. Due to time constraints this system could not be implemented. As an alternative the distance travelled was manually measured and divided by the walk duration to get the average velocity which was then inputted into M-BOA. The benefit of this solution was that the adaptation could be tested without influence from additional variables such as the accuracy of the dead reckoning.

Parameter Transfer

The gait parameters outputted from the M-BOA algorithm are manually inputted into the Hexapod code and uploaded to the microcontroller through an ST-Link debugger. This was due to time constraints, however the initially proposed system could definitely still be implemented.

Table 4.4: Leg failure scenario joint angles

Joint	**Angle**
Coxa	0°
Femur	90°
Tibia	-165°

Chapter 5

Testing & Results

5.1 Approximating Failure

This section details the method used to approximate leg failures for the simulated and real hexapod adaptation tests.

5.1.1 Leg Failure

When an animal hurts its leg it will often lift the limb out of the way so that it doesn't get bumped and remove it entirely from its gait [13]. A similar behaviour was used to approximate a failure of the hexapod leg. The leg is lifted up out of the way so that it does not interfere with the other functional legs or provide a means of support on the ground. The leg was also left attached to retain the associated dead weight. The failure position of a leg is shown in figure **??**

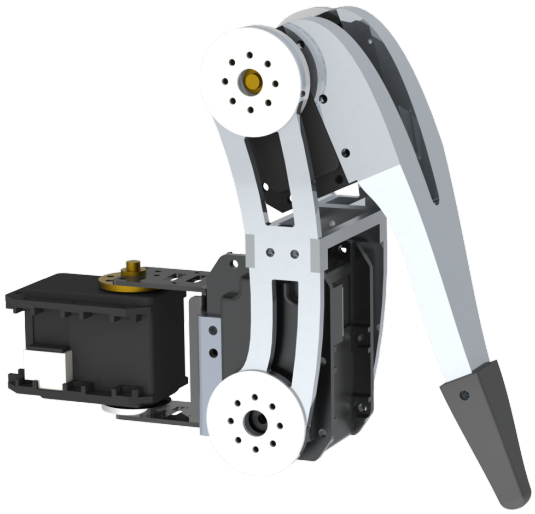


Figure 5.1: Leg failure position

Figure 5.2 additionally shows the actual position of the leg on the real hexapod.

This failure scenario can be readily implemented on the hexapod by commanding the joint angles specified in table 5.1. The leg is actively held in this position to stop it falling down and interfering with the gait.

5.1.2 Failure Scenarios

For this project only 1 and 2 leg failure scenarios were investigated. If 3 legs are failed on the hexapod that only leaves 3 functional legs remaining. During walking one of those functional

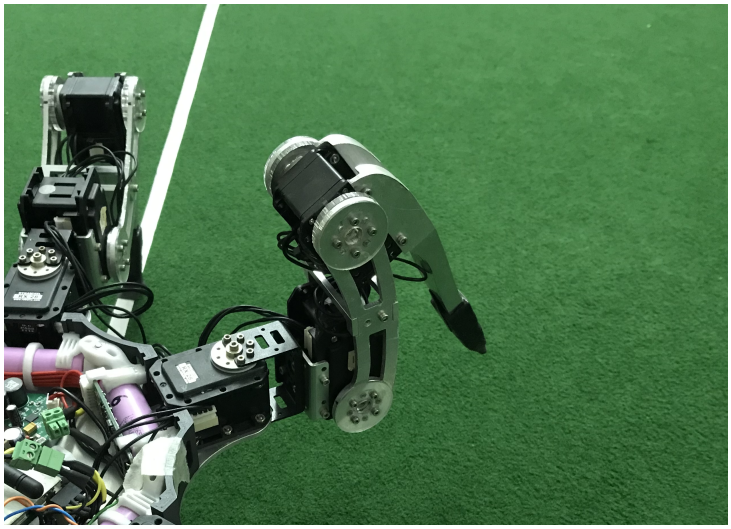


Figure 5.2: Hexapod leg failure position

Table 5.1: Failed leg joint angles

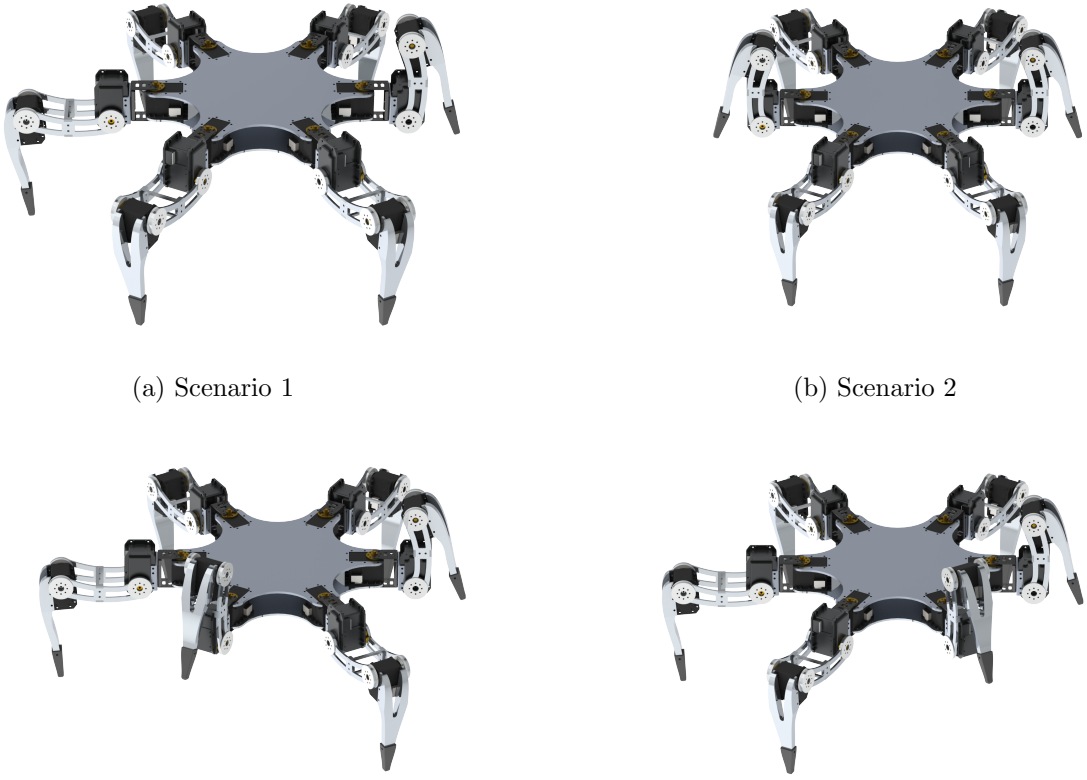
Joint	Failure Angle
Coxa	0°
Femur	90°
Tibia	-165°

legs needs to be lifted for the swing phase. This leaves only 2 legs on the ground reducing the support polygon to a line. The hexapod is not dynamically balanced and would therefore simply topple over. It is unreasonable to expect a compensatory gait with half of all the legs failed.

With 1 failed leg there is only a single configuration, however with 2 failed legs there are three configurations that the failures can occur in. The failed legs can be opposite, separated by one functional leg, or adjacent. These have been called failure scenarios and there are a total of 4 failure scenarios when considering 2 or fewer leg failures. These failure scenarios are shown in figure 5.3.

Failure Orientation

Within each failure scenario there are 6 (3 for scenario 2) possible orientations the failure can occur in around the body. For example, in scenario 3 legs 2 and 4 could be failed or legs 3 and 5. Therefore, all possible orientations are considered for each failure scenario to ensure that the entire distribution of failures is simulated.



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

Figure 5.3: Simulated failure scenarios
The failed legs are shown in their raised failure position

5.2 Simulated Adaptation Experiment

5.2.1 Experimental Scenarios

Testing gait adaptation is simulation is substantially faster than in reality. Consequently significantly more tests could be conducted. Each of the failure scenarios were tested with each of the independently generated maps for all possible failure configurations around the body. This resulted in a total of 210 independent adaptation tests as shown in table:5.2

Table 5.2: Simulated experiment configurations

Failure	**Orientations**	**Maps**	**Experiments**
Scenario 1	6	10	60
Scenario 2	3	10	30
Scenario 3	6	10	60
Scenario 4	6	10	60
Total			**210**

5.2.2 Experimental Setup

The M-BOA adaptation algorithm is run with the same parameters as [13].

$$\sigma^2_{noise} = 0.001$$
$$\alpha = 0.90$$
$$\rho = 0.4$$
$$\kappa = 0.05$$

Each gait is simulated for 5 s and the walking performance (average velocity) is calculated for this duration. Consequently the duration of adaptation is simply the number of trials multiplied by the duration per trial of 5 s. Collisions between the base and ground were also ignored unlike during map generation. The real hexapod platform has no means of determining collisions with the ground so this cannot be included for the adaptation phase.

To evaluate statistical significance

5.2.3 Control Test

The original gait implemented on the hexapod serves as a control test to compare the results of adaptation. The original gait is a tripod gait with a commanded velocity of 0.3 m/s and a body height of 140 mm. Table 3.11 describes the leg parameters to produce this gait. The gait is tested in the failed and functional configurations. For the failed case only a single leg failure (scenario 1) is considered. This scenario has the lowest loss in functionality and therefore any of the other scenarios would simply perform worse. The tripod gait performance with failure scenario 1 was simulated for all 6 failure orientations. The average speed for these gaits is calculated the same way as for the adaptation tests.

5.3 Adaptation in Simulation

This section presents the results of the simulated adaptation of the hexapod platform. The adaptation performance was evaluated on the resulting gait performance and the number of trials required for adaptation. Additionally the impact on adaptation performance of the stochastic behaviour-performance map was investigated.

5.3.1 Performance After Adaptation

In order for the adaptation solution to be considered effective the adapted gait needs to perform better than the originally deployed gait with the failure. The adapted performance for each of the failure scenarios is compared with the default deployed gait across all of the failure scenarios in figure 5.4.

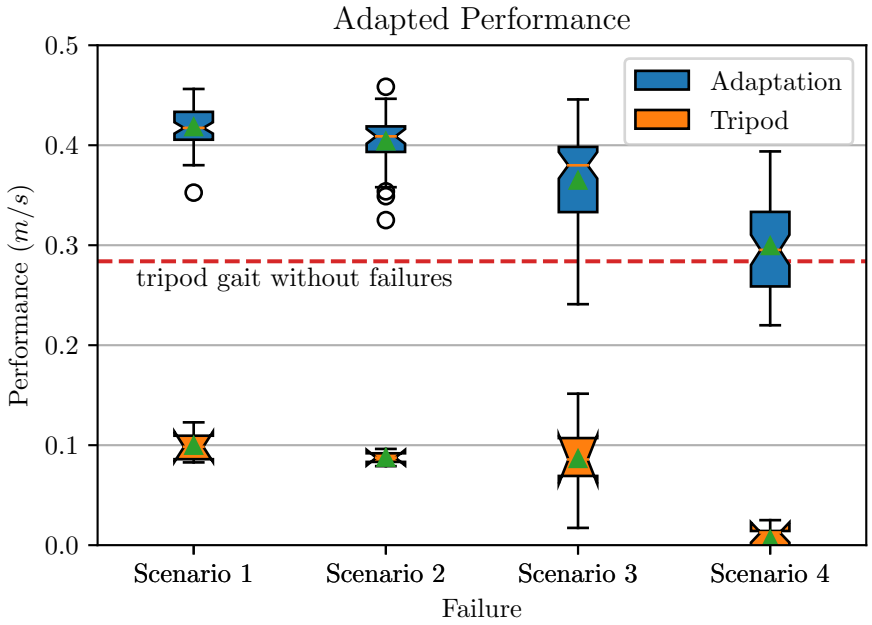


Figure 5.4: Performance after simulated adaptation
The box and whisker plots provide a summary of the distribution of the results for each previously defined failure scenario. The green triangles show the means of each distribution. The dashed red line shows the mean performance of the the tripod gait without any leg failures.

From figure 5.4 it is clear that the gait adaptation algorithm resulted in a significantly ($p < 10^{-18}$) higher performing gait on the simulated hexapod than the fixed tripod gait across all of the 4 failure scenarios. The adaptation algorithm therefore results in a better performing gait than the default gait for all 4 failure scenarios, regardless of the behaviour-performance map used or the orientation of the failure relative to the walking direction.

Another notable result is that after adapting to a failure, the resulting gaits have similar or greater performance than even the default tripod gait without any failures. In failure scenarios 1, 2, and 3 the adapted gait's performance was significantly ($p < 10^{-19}$) better than the fully functional tripod gait. The adapted performance in scenario 4 was better but only marginally ($p \approx 0.015$). Therefore the adapted gaits are higher performing than a default tripod gait irrespective of leg failures.

5.3.2 Adaptation Duration

For adaptation to be applicable in real world scenarios it needs to occur quickly. A threshold value of 2 minutes was selected based on the previous performance results for a different robotic platform in [13]. The number of trials, and therefore duration duration of adaptation for each failure scenario is shown in figure 5.5.

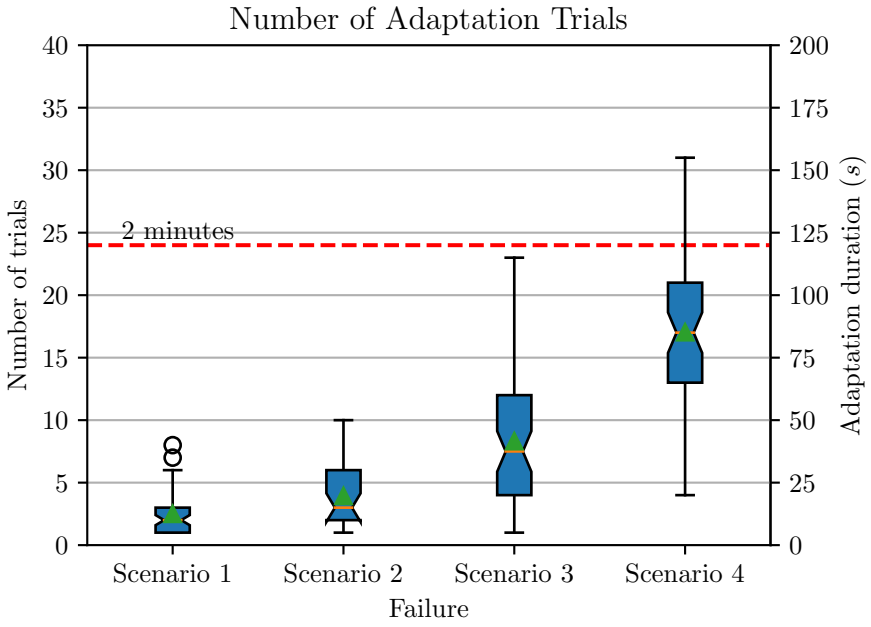


Figure 5.5: Number of trials for simulated adaptation
The box and whisker plots provide a summary of the distribution of the results for each previously defined failure scenario. The green triangles show the means of each distribution. The dashed red line shows the desired adaptation time.

The mean adaptation time for all of the experiments was significantly ($p < 10^{-11}$) lower than the desired 2 minute threshold. A few tests in scenario 4 took longer than 2 minutes to adapt with the longest being 2 minutes and 35 seconds, however statistically these will be infrequent. Therefore, based on simulation the hexapod gait can be adapted in under 2 minutes irrespective of the failure being experienced, the map used, and the orientation of the failure.

These results include all 10 of the independently generated behaviour-performance maps generated with MAP-Elites and the distribution of data is therefore statistically representative of what can be expected for independent reproductions of these experiments.

5.3.3 Impact of Behaviour-Performance Map

Ten independent behaviour-performance maps were generated as each map has an intrinsic stochasticity as a result of the random initialisation. Due to this random variation some maps could potentially perform better than others for adaptation. Figure 5.6 shows the variation in adapted performance that can be expected with identical independent generations of the behaviour-performance maps. The whole space of failure scenarios and orientations is considered as in reality any failure is possible.

Some of the maps in figure 5.6, particularly 1 and 10, resulted in very similar mean performance after adaptation, while other maps, particularly 4 and 7, were significantly ($p \approx 0.018$) different. Additionally the performance range can vary substantially depending on the map used with some maps exhibiting very consistent performance irrespective of the failure and others

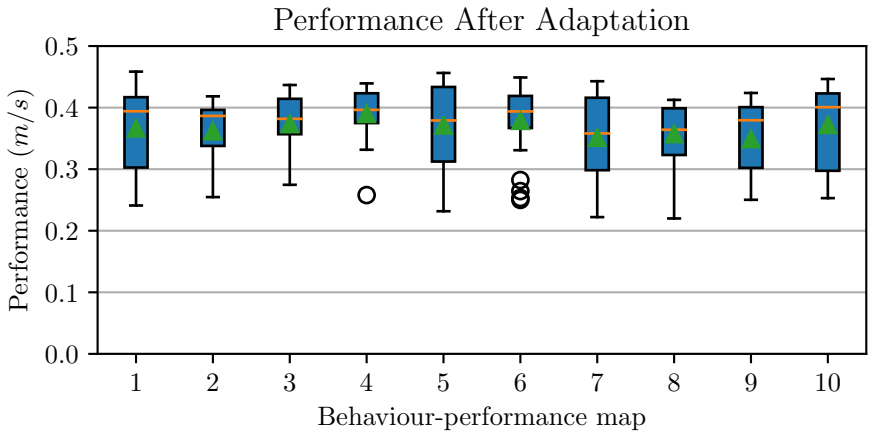


Figure 5.6: Performance after adaptation per map
The box and whisker plots provide a summary of the distribution of the performance after adaptation across all of the failure scenario and orientation combinations for each map. The green triangles show the means of each distribution.

producing a wider range of performances.

The impact of map stochasticity on the number of trials required for adaptation is also of interest. Figure 5.7 shows the variation in the number of trials required to adapt between each of the independently generated behaviour-performance maps. Again the whole space of failure scenarios and orientations is considered.

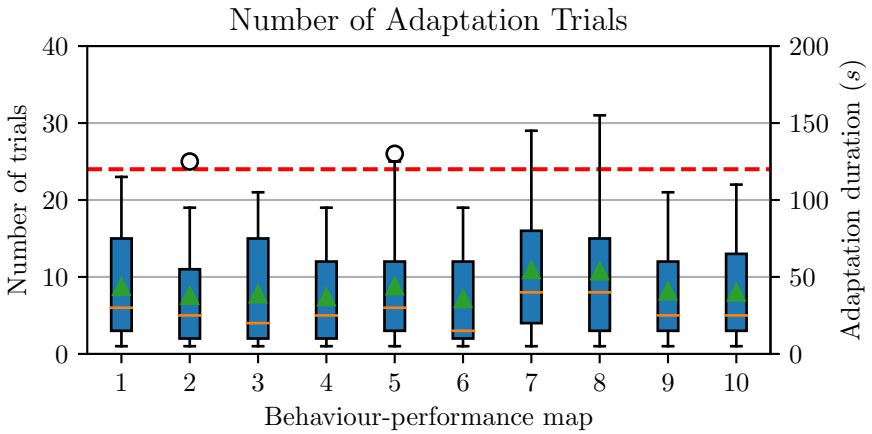


Figure 5.7: Number of trials for adaptation per map
The box and whisker plots provide a summary of the distribution of the number of trials required for adaption across all of the failure scenario and orientation combinations for each map. The green triangles show the means of each distribution. The dashed red line shows the threshold duration of 2 minutes

Unlike with the performance after adaptation the difference between maps with the mean number of adaptation trials is not statistically significant ($p > 0.14$). Therefore generation of additional behaviour-performance maps will not substantially alter the number of trials required for adaptation.

5.3.4 Discussion

The simulated results show that adaptation of the hexapod platform with IT&E resulted in faster gait than the tripod gait for all of the investigated failure scenarios involving 1 and 2 failed legs. Furthermore this adaptation occurred in under the desired 2 minutes. These results are also consistent with those published by [13].

The selection of the behaviour-performance map being used will affect the final performance of the adapted gait, therefore the best performing map should be selected if multiple maps are generated.

The adapted performance gradually decreased for scenario numbers 2 to 4, indicating and increase in failure scenario complexity. A potential explanation for this is that the higher failure scenarios have a greater weight imbalance due to the closer spaced dead-weight legs. Failure scenario 4 appears as though it is almost about to tip over while failure scenario 2 has an even stance. Interestingly failure scenario 1 and 2 performed similarly even with the one has twice as many failed legs as the other.

In robotics there is typically a discrepancy between simulation and reality, termed the "Reality Gap". This was discussed during the literature review. Experiments on the physical hexapod platform are therefore essential in determining the performance of the approach.

5.4 Real Adaptation Testing

5.4.1 Experiment Configuration

Due to time limitations replicating all of the experiments done in section 5.2 was not possible. Instead only two failure scenarios were considered to reduce the number of experiments. Failure scenario 1 and 2 were selected as they required the lowest number of trials during adaptation in simulation. A higher number of adaptation trials will increase the duration of the experiments which was undesirable. Furthermore they include both the failure of 1 leg and 2 legs. The behaviour-performance map which required the lowest number of adaptation trials in simulation for the corresponding leg failure was used as the behaviour performance map for the real tests. Furthermore legs 1 and 4 were picked as the legs to be failed. There was no logic behind this decision as in reality any leg could fail. In addition to the failure scenarios, adaptation experiments without failures were also included. The goal of these experiments is to determine whether the maps generated in simulation can be deployed on the real robot. They therefore also need to be functional even before a failure occurs.

The experiment configurations and their corresponding experiment numbers are shown in table 5.3.

Table 5.3: Experiment configuration

Experiment	Map	Leg Failure
1	1	None
2	2	None
3	2	Leg 1
4	2	Leg 4
5	4	Leg 1 & 4
6	6	Leg 1 & 4

5.4.2 Control Experiment

The default tripod gait shown in table 3.11 was used as the control experiment as this gait is currently implemented on the hexapod. This would provide an indication of the relative performance of adaptation to no adaptation. The tripod gait with the corresponding failures was measured in 10 independent tests.

5.4.3 Experimental Setup

For this project the performance of a gait is measured as the average velocity along an axis, with the average velocity calculated as the distance travelled divided by the walk duration. In the simulations the velocity is taken along the x-axis. This was replicated in reality by creating two perpendicular lines on the floor of the lab with tape. The Hexapod was then aligned with its forward direction along the one line (vertical), and the centre of its base at the intersection as shown in figure 5.8.

This alignment is identical to the simulation to ensure that the results are consistent can be compared. The alignment to the axes shown in figure 5.8 was done after the legs of the hexapod had move to their initial positions for a given gait to ensure consistency. This was done as the standing manoeuvre sometimes resulted in a slight misalignment of the base. A 10 second delay in the code provided sufficient time to complete this alignment. The distance travelled was measured parallel to the commanded direction from the marked point to the

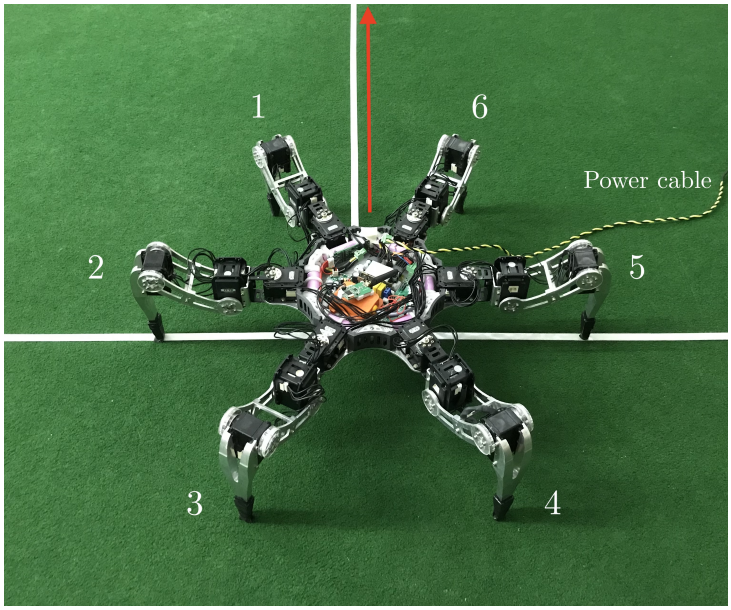


Figure 5.8: Hexapod alignment with ground axes
The red arrow is the commanded walking direction for the Hexapod

horizontal line to mimic the measurement along only a specific axis. The final position of the hexapod base was marked on the lab floor with white stickers directly underneath its centre, and the distance travelled was measured with a tape measure. Measurements were rounded to the nearest 5 mm. The Hexapod has an onboard battery system, however this was not used as a single experiment could last for over an hour and the runtime of the system was not known. A constant supply voltage of 25.0 V was instead used ensuring consistency between experiments. This was provided by a power supply rated for 15 A. The power cable can be seen in figure 5.8 and was held out of the way of the legs during gait trials. Gaits which resulted in the hexapod moving behind the start line were given a distance travelled of 0 m. The flooring of the lab was used as the ground surface and it is a green polyester felt carpeting. The same M-BOA parameters as the previous implementation [13], and the simulated experiments were used for consistency and to compare results.

$$\begin{aligned} \sigma^2_{noise} &= 0.001 \\ \alpha &= 0.90 \\ \rho &= 0.4 \\ \kappa &= 0.05 \end{aligned}$$

5.4.4 Test Procedure

A single experiment is made up of multiple gait trials. The exact procedure for each trial is outlined below and was repeated until the adaptation criteria were met.

1. Upload gait parameters from M-BOA to microcontroller
2. Power cycle Hexapod
3. Wait for legs to move to starting positions
4. Align Hexapod base with ground axes
5. Wait for 5 second walk to complete
6. Mark centre position of Hexapod base
7. Move Hexapod to start
8. Measure perpendicular distance from start line to marker
9. Record distance travelled and input into to M-BOA

The distance travelled results for each experiment can be found in section 5.2.

5.5 Adaptation in Reality

This section presents the results from the testing of adaptation on the real hexapod platform. Six independent adaptation experiments were conducted for 2 failure scenarios and one scenario with no failures.

5.5.1 Performance After Adaptation

The performance after adaptation for each of the 6 experiments is shown in figure 5.9. The results are presented with the performance results of the default tripod gait with each of the corresponding failure scenarios as well as the functional scenario for comparison.

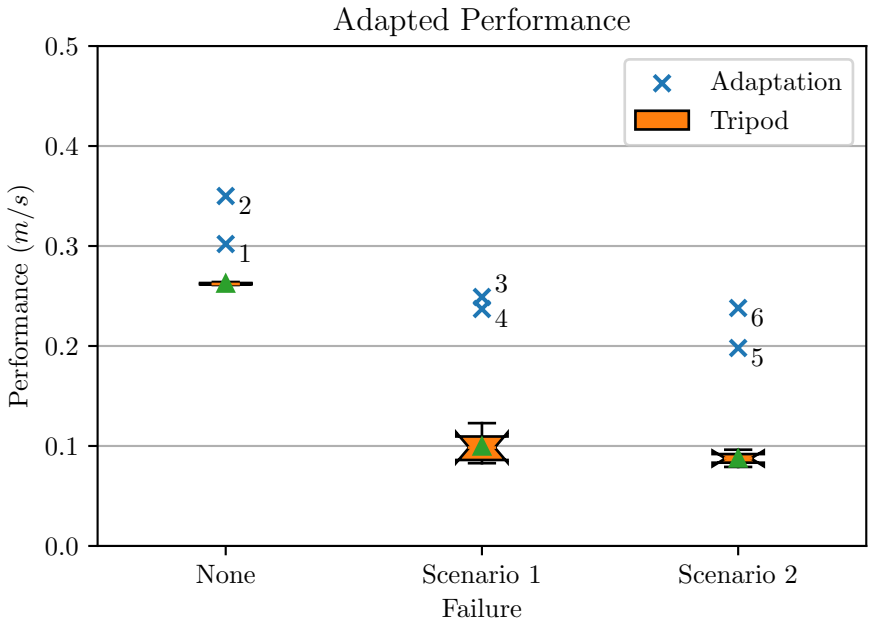


Figure 5.9: Adapted gait performance in reality
The performance of the tripod gait is shown by the box and whisker plots and the gait performance from the adaptation experiments are shown by the x's. The numbers for the adaptation experiments are presented alongside the corresponding result. The experiment numbers can be found in table 5.3. The green triangles show the mean of the tripod data distribution.

Even without any leg failures both of the adapted gait experiments outperformed the default tripod gait on the hexapod platform. This is to be expected as the adapted gait are the result of an optimisation algorithm, while the default tripod gait is just hand-picked parameters. Additionally, the adapted gait performance for both failure scenarios substantially outperforms the default tripod gait. The adapted gait performance from experiment 6 actually performs slightly better than the adapted gait in experiment 4 even though experiment 6 had 2 failed legs while experiment 4 only had 1 failed leg. The mean adaptation performance was significantly ($p \approx 10^{-4}$) greater than the the mean performance of the default tripod gait across all 3 failure scenarios.

Experiments 3 and 4 use the same map with the only difference between these experiments being the orientation of the failed leg around the body as seen in table 5.3. Therefore, the gait resulting from adaptation to a failed front left leg will perform similarly to the gait resulting from a failed back right leg.

Between experiments 5 and 6 the only difference was the map used. Experiment 5 uses map

4 and experiment 6 uses map 6. Both maps had similar performance after adaptation to failures in simulation, however their performance differs slightly in reality.

5.5.2 Adaptation Duration

The number of trails required for adaptation on the real hexapod, was recorded for each of the 6 independent experiments and is shown in figure 5.10.

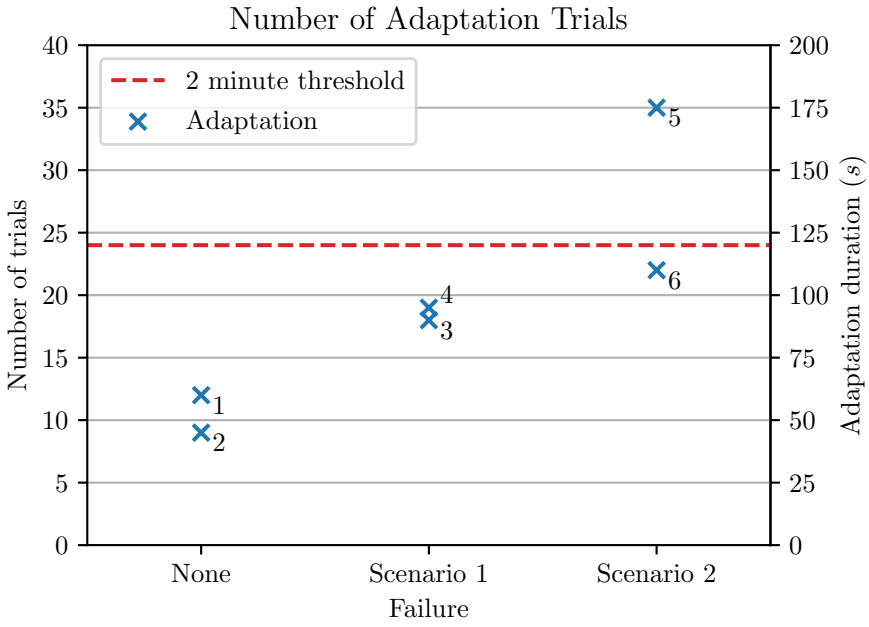


Figure 5.10: Number of adaptation trials in reality
The adaptation experiments are shown by the x's. The number for the adaptation experiment is presented alongside the corresponding result. The experiment numbers can be found in table 5.3

Only experiment 5 exceeded the maximum 2 minute adaptation threshold. This was for the more challenging adaptation scenario of 2 failed legs compared to 1, however adaptation still occurred in less than 3 minutes. However, experiment 6 was done for the same failure scenario but managed to adapt in less than the 2 minute threshold. These experiments differ in the behaviour-performance map used.

The results for experiment 3 and 4 are again closely spaced indicating that the orientation of the failure has little effect on the number of adaptation trials and therefore the duration.

5.5.3 Comparing Simulation and Reality

In the previous section the performance after adaptation was evaluated on the simulated hexapod. It is useful to compare the results collected from the simulated hexapod to the results collected on the real hexapod. These simulated and real adapted gait performance results are displayed together in figure 5.11 to aid comparison.

Most notable from these results is that there is a substantial performance drop even without any failures. A gait which produced a certain performance in reality cannot match the same performance in simulation.

The real performance is also consistently lower than the simulated performance after adaptation. The performance difference also appears to be similar between failure scenarios and corresponds to the performance difference without failure.

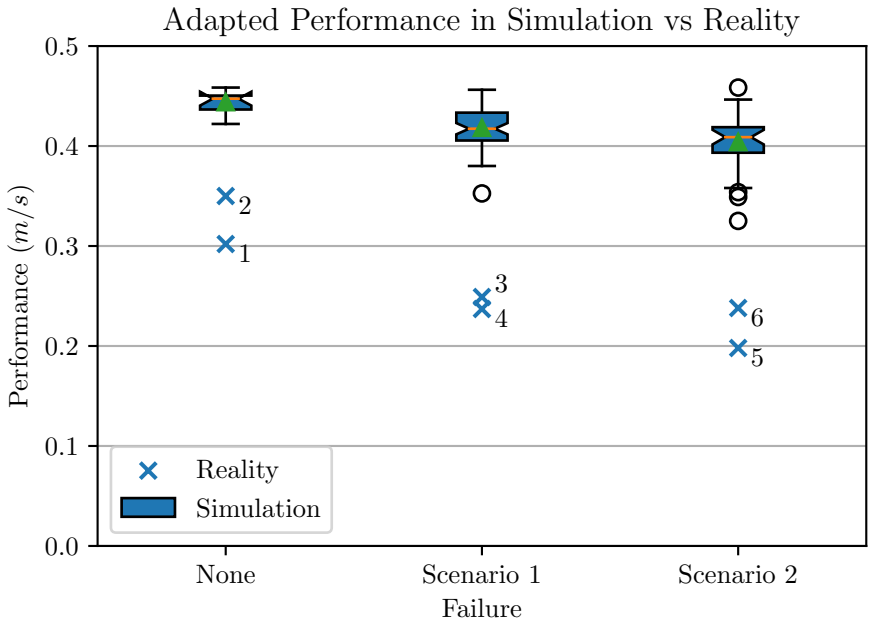


Figure 5.11: Adapted gait performance in simulation versus reality
The performance in simulation is shown by the box and whisker plots and the adaptation experiments are shown by the x's. The full distribution of the simulated results is shown to give an idea of the spread of the data. The green triangle shows the mean of the simulation distribution

In addition to the variation in adapted gait performance between simulation and reality, the variation between the number of trials required for adaptation should also be considered. The number of trials required for adaptation in both simulation and reality is shown in figure 5.12.

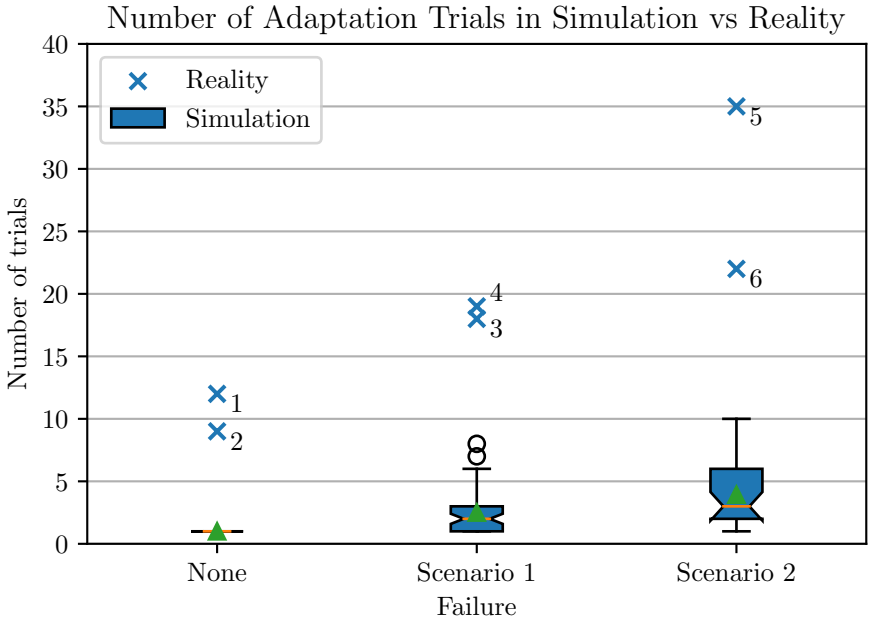


Figure 5.12: Number of adaptation trials in simulation versus reality
The green triangle shows the mean of the simulation distribution

Only 1 trial is required to adapt to the hexapod without any failures in simulation. This is understandable as the map was generated using the simulated hexapod without any failures and therefore the first tested gait performs as expected. In reality however the first tested gait does not perform as expected and requires multiple adaptation trials.

Significantly more trials are required to adapt in reality than in simulation. The number of trials also increases from scenario 1 to scenario 2 which is consistent with the results in simulation.

5.5.4 Direction Deviation

The final position of the hexapod at the end of a gait trial was marked with a white sticker before the distance was measured. Figure 5.13 shows the distribution for experiment 3 of the final positions during adaptation trials.

Figure 5.13: Adaptation trial final positions
Best performing trial is marked with red circle

While the best performing adaptation trial is close to the centre line, and therefore walked relatively straight, some of the other trials resulted in substantial deviation from the centre line.

5.5.5 Discussion

First and foremost these results show that adaptation with the IT&E algorithm consistently results in a higher performing gait than the default tripod gait on the real hexapod platform irrespective of the failure scenario. Furthermore this adaptation occurred in under 24 trials (2 minutes) on average with the longest taking 35 trials (3 minutes and 55 seconds) making it applicable in real-world scenarios. These results also correspond to those in [13] serving as further validation of their approach. The results also confirm that the custom kinematic controller created for this project can be used with MAP-Elites and is variable enough to produce compensatory gaits to a variety of failure scenarios.

The orientation of the failure around the body relative to the walking direction has little effect on the resulting adapted performance as well as on the number of trials required. The use of a different map however does result in a marginally different adapted performance.

The adapted performance gradually decreased in both simulation and reality for the

The difference between the simulation and reality was noticeable and much more substantial than the results from Cully et al [13]. Their robot required less than 6 trials to adapt from simulation to reality without any failures, while less than 12 trials were required for this project.

It was found that the feet of the hexapod often slipped on the lab floor when testing a gait, but did not slip in simulation with the same gait. The floor of the lab was very different to the perfectly smooth and uniform ground plane used in simulation which would explain the difference in gait performance in reality. Accurately modelling the ground surface is particularly challenging and will very often not be possible in new and uncertain real-world environments. The ground plane in simulation was also assigned a high friction coefficient to limit sliding gaits and dragging of the feet. An unintended consequence of this is that gaits could have developed which were reliant on the high friction coefficient and used it to their advantage to propel them forwards. However, even with this larger sim-to-real discrepancy the IT&E algorithm was able to adapt and consistently find a higher performing solution than the default tripod gait. The only downside of the larger discrepancy was that it required a greater number of trials to find a compensatory gait.

Chapter 6

Conclusion

The aim of this project was to implement online gait adaptation on the RARL hexapod robot to retain mobility even with leg failures. The IT&E was identified from existing approaches to be best suited to the constraints of the hexapod platform. This approach was successfully implemented on the hexapod platform, albeit with manual performance feedback and gait parameter transmission.

Implementation of the approach required development of a custom kinematic gait controller and a dynamic simulator for the hexapod. Both of these were developed successfully and integrated with the IT$E adaptation approach. Ten independent behaviour-performance maps were also successfully generated on the CHPC computing cluster allowing for a statistical evaluation of the success of the approach.

The hexapod robot was successfully able to adapt to 1 and 2 leg failure scenarios in both simulation and reality. The adapted gaits were more than 50% faster than the default tripod gait currently implemented on the system. Additionally the adapted gaits were also faster without any failures. This exceeded the objectives for adaptation set out at the beginning of the project. Adaptation took on average less than 2 minutes with the longest taking just under 3 minutes making the approach applicable to real-life scenarios.

The project serves also serves to validate the applicability of the IT&E approach to different robotic systems.

Chapter 7

Recommendations

7.0.1 Project Changes

There was insufficient contact time with the platform to be able to implement velocity estimation using the IMU and transferral of the gait parameters over remote communications, however this was intended. Implementation of these system would significantly speed up experimentation time and allow for the collection of more results on the physical platform in future work.

The adaptation duration can be shortened substantially by shortening the duration of a trial. For this project 5 seconds was used to remain consistent with a previous implementation, however this value was assumed. Future work could investigate the optimal duration for a trial to still find an alternative gait but also minimise the adaptation duration.

7.0.2 Future Work

The hexapod base stabilisation was left out during this initial analysis. Consequently the resulting gaits were relatively unstable. A recommendation for future work would be to include the stabilisation system in the simulation and investigate the stability of the adapted gaits.

This project only dealt with the hexapod walking in a single direction as the performance of a gait is a function of the commanded direction. To rephrase, a gait optimised to walk forwards will not also be optimised to walk to the right. This is a substantial limitation as robots in real situations will need to be able to walk optimally in a variety of directions. This issue could be tackled by developing separate maps for different tasks such as turning and switching based on the commands sent from the base computer. Further work in this aspect would be very beneficial.

It was thought that the gait which used less of a failed leg would be the optimum gait found during adaptation however this was not the case. There is therefore no correlation between the chosen descriptor and the failure scenario. It would therefore be worthwhile for other descriptors to be investigated.

The MAP-Elites algorithm was very inefficient and could be sped up substantially by by a performance improvement. Investigation into means of increasing efficiency on the cluster and therefore runtime would be beneficial.

References

[1] J. Bellingham and K. Rajan, "Robotics in remote and hostile environments.," in *318*. Monterey Bay Aquarium Research Institute, 7700 Sandholdt Road, Moss Landing, CA 95039, USA. jgb@mbari.org, 2007, pp. 1098–1102.

[2] J. E. Bares and D. S. Wettergreen, "Dante ii: Technical description, results, and lessons learned," in *18*. SAGE Publications, 1999, pp. 621–649.

[3] M. Bajracharya, M. W. Maimone, and D. Helmick, "Autonomy for mars rovers: Past, present, and future," in *41*. IEEE, 2008, pp. 44–50.

[4] A. Davids, "Urban search and rescue robots: From tragedy to technology," in *17*. IEEE, 2002, pp. 81–83.

[5] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and M. Fukushima, "Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots," in *30*. Wiley Online Library, 2013, pp. 44–63.

[6] C. G. Atkeson, B. Babu, N. Banerjee, D. Berenson, C. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, and P. Franklin, "What happened at the darpa robotics challenge, and why?" In *1*. 2016.

[7] J. Carlson and R. Murphy, "How ugvs physically fail in the field," in *21*. Institute of Electrical and Electronics Engineers (IEEE), 2005, pp. 423–437.

[8] B. Dhillon, A. Fashandi, and K. Liu, "Robot systems reliability and safety: A review," in *8*. Emerald, 2002, pp. 170–212.

[9] T. Booysen and S. Marais, "The development of a remote controlled, omnidirectional six legged walker with feedback," ser. 2013 Africon, IEEE, 2013, pp. 1–1.

[10] T. Booysen and F. Reiner, "Gait adaptation of a six legged walker to enable gripping," in. 2015.

[11] S. Marais, A. Nel, and P. Robinson, "Reflex assisted walking for a hexapod robot," ser. Reflex assisted walking for a hexapod robot, IEEE, 2016, pp. 1–6.

[12] R. Christopher, "Mathematical modelling and control system development of a remote controlled, IMU stabilised hexapod robot," M.S. thesis, UCT, Cape Town, 2019.

[13] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," in *521*. Springer Science and Business Media LLC, 2015, pp. 503–507.

[14] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in. 2018.

[15] X. Ding, Z. Wang, A. Rovetta, and J. Zhu, "Locomotion analysis of hexapod robot," in *Climbing and walking robots*, InTech, 2010, pp. 291–310.

[16] K. Chatzilygeroudis, V. Vassiliades, and J.-B. Mouret, "Reset-free trial-and-error learning for robot damage recovery," in *100*. Elsevier BV, 2018, pp. 236–250.

[17] M. H. Raibert, *Legged Robots that Balance*. MIT Press, 1986.

[18] G. Carbone and F. Gomez-Bravo, *Motion and Operation Planning of Robotic Systems*. Springer, 2015.

[19] L. Zlajpah, "Simulation in robotics," in *79*. Elsevier BV, 2008, pp. 879–897.

[20] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," in *17*. Institute of Electrical and Electronics Engineers (IEEE), 2013, pp. 122–145.

[21] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx," in *2015 IEEE international conference on robotics and automation*, ser. ICRA, IEEE, 2015, pp. 4397–4404.

[22] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer, 2014.

[23] R. Featherstone and D. Orin, "Robot dynamics: Equations and algorithms," ser. Robot dynamics: equations and algorithms, IEEE, 2000, pp. 826–834.

[24] C. Ong and E. Gilbert, "The gilbert-johnson-keerthi distance algorithm: A fast version for incremental motions," in. 1997.

[25] J. Collins, D. Howard, and J. Leitner, "Quantifying the reality gap in robotic manipulation tasks," in. 2019.

[26] J.-B. Mouret and K. Chatzilygeroudis, "20 years of reality gap: A few thoughts about simulators in evolutionary robotics," in *GECCO '17: Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. 20 years of reality gap: a few thoughts about simulators in evolutionary robotics, New York, NY, USA: ACM, 2017, pp. 1121–1124.

[27] S. Ivaldi, V. Padois, and F. Nori, "Tools for dynamics simulation of robots: A survey based on user feedback," in. 2014, p. 1402.7050v1.

[28] D. Kang and J. Hwangho, *Simbenchmark: Physics engine benchmark for robotics applications*, `https://leggedrobotics.github.io/SimBenchmark/`, 2018.

[29] E. Coumans and Y. Bai, *Pybullet, a python module for physics simulation for games, robotics and machine learning*, `http://pybullet.org`, 2016.

[30] R. Smith, *Open dynamics engine (ODE)*, `https://ode.org`, 2001.

[31] E. Todorov, *Multi-joint dynamics with contact (MuJoCo)*, `http://www.mujoco.org`, 2015.

[32] J. Lee, *Dynamic animation and robotics toolkit (DART)*, `https://dartsim.github.io`, 2012.

[33] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer, 2016.

[34] J.-M. Yang, "Gait synthesis for hexapod robots with a locked joint failure," in *23*. Cambridge University Press (CUP), 2005, pp. 701–708.

[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. A Bradford Book, 2018.

[36] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," in. 2015.

[37] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," in *3*. Frontiers Media SA, 2016.

[38] P. I. Frazier, "A tutorial on bayesian optimization," in. 2018, p. 1807.02811v1.

[39] M. Baker and R. Buyya, "Cluster computing at a glance," in *1*. Prentice Hall, 1999, p. 12.

[40] Robotis, *Dynamixel RX-28 user manual*, 2006.

[41] ——, *Dynamixel RX-64 user manual*, 2006.

[42] M. Kok, J. D. Hol, and T. B. Schön, "Using inertial sensors for position and orientation estimation," in *11*. Now Publishers, 2017, pp. 1–153.

[43] X. T. BV, *Mti and mtx user manual and technical documentation*, 2009.

[44] V. Vassiliades, K. Chatzilygeroudis, and J.-B. Mouret, "Using centroidal voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm," in. 2016, p. 1610.05729v2.

[45] S. A. I. L. e. al., *Robotic operating system*, `https://www.ros.org`, 2018.

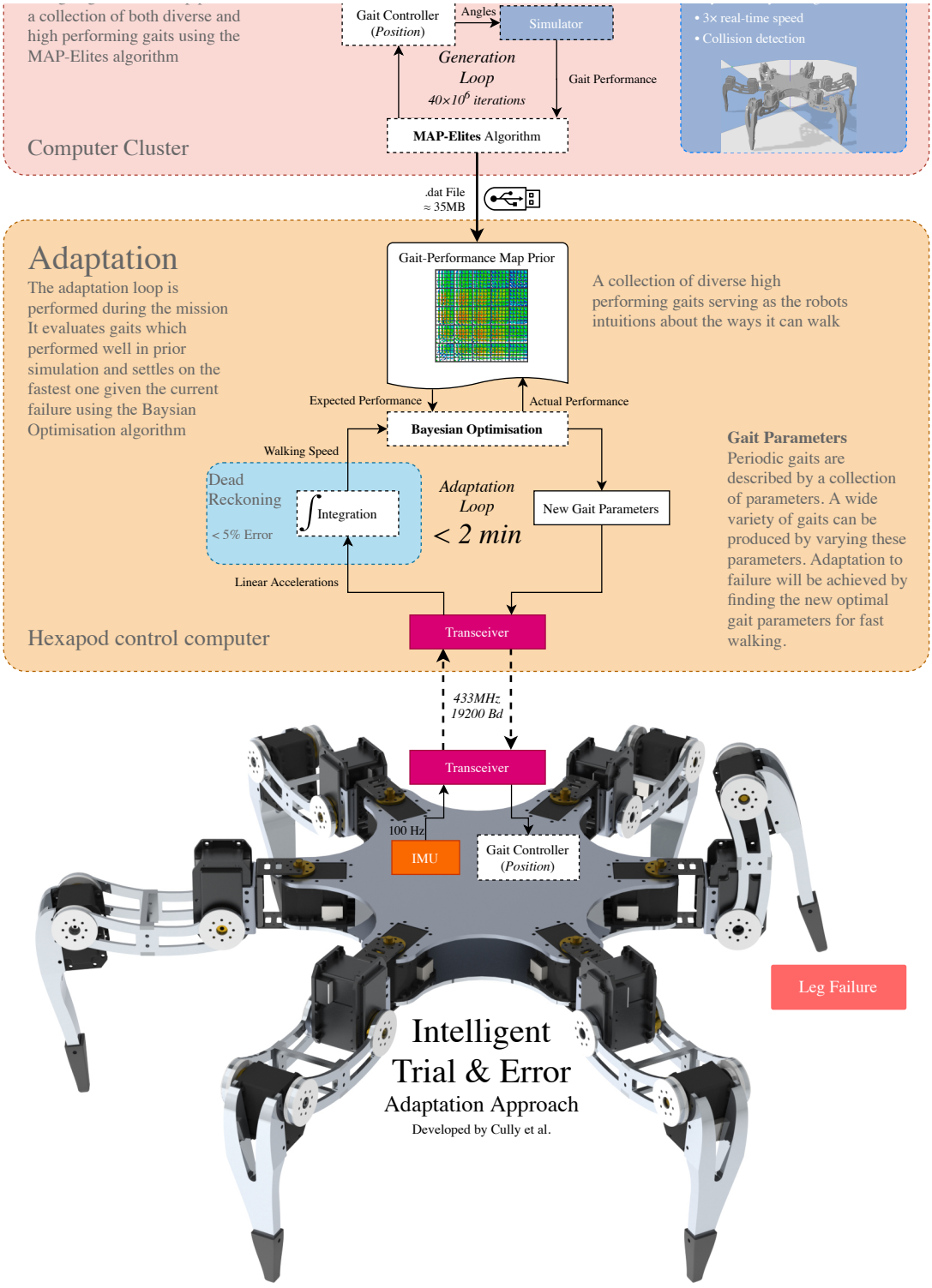
[46] C. Harris, K. Millman, S. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. Smith, R. Kern, M. Picus, S. Hoyer, M. van Kerkwijk, M. Brett, A. Haldane, J. Del Rıo, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. Oliphant, "Array programming with NumPy," in *585*. Springer Science and Business Media LLC, 2020, pp. 357–362.

[47] K. Reitz and T. Schlusser, *The Hitchhiker's Guide to Python*. "O'Reilly Media, Inc.", 2016.

[48] L. Dalcın, R. Paz, and M. Storti, "Mpi for python," in *65*. Elsevier, 2005, pp. 1108–1115.

Appendix A

Implementation Infographic



Appendix B

Computer Specifications

Table B.1: Google Compute Engine CVT Virtual Machine Specifications

CPU(s)	1
Processor	6^{th} Gen Intel Xeon
Clock speed	2.0GHz
RAM	624 GB
OS	Debian 9 (Stretch)

Table B.2: CHPC Lengau Cluster Queue Specifications

CPU(s)	240
Nodes	24
Processor	5^{th} Gen Intel Xeon
Clock speed	2.6 GHz
RAM	128 GB
OS	CentOS 7.0

Appendix C

Supplementary Results

C.1 Results

C.1.1 Map Generation

Table C.1: CHPC Performance Results

Map	**CPU time**(h)	**Runtime**(h)	**RAM**(GB)	**CPU Utilisation**
1	3697.8	40.0	151.03	38%
2	3868.7	40.2	150.93	40%
3	3899.9	41.0	150.95	39%
4	3884.4	40.8	151.00	39%
5	3870.0	40.8	150.98	39%
6	3863.2	39.6	150.94	40%
7	3853.4	39.5	150.93	40%
8	3897.7	40.1	150.88	40%
9	3920.8	40.1	151.00	40%
10	3879.0	40.0	150.80	40%
Avg.	3863.5	40.2	150.94	39.5%

C.1.2 Simulated Adaptation

This table is used to select which map to use per experiment

Table C.2: Map Generation Results

Map	**Max Perf.**(m/s)	**Unique Gaits**
1	0.457	39734
2	0.422	39769
3	0.445	39776
4	0.448	39743
5	0.458	39759
6	0.450	39757
7	0.451	39754
8	0.434	39745
9	0.427	39766
10	0.447	39753
Avg.	0.444	39756

Table C.3: Number of adaptation trials

Map	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
1	2.7	15.8	10.2	3.7
2	1.5	15.0	7.5	5.0
3	2.0	16.2	7.3	3.3
4	2.7	12.7	9.5	2.0
5	2.8	17.7	8.5	3.7
6	1.8	16.3	5.8	2.0
7	3.3	24.0	7.0	7.7
8	2.7	21.2	11.2	5.3
9	2.8	15.3	8.5	3.7
10	2.7	16.2	7.7	3.0

Table C.4: Adapted performance

Map	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
1	0.429	0.295	0.346	0.423
2	0.400	0.305	0.370	0.382
3	0.421	0.318	0.366	0.403
4	0.429	0.344	0.382	0.418
5	0.440	0.281	0.369	0.417
6	0.426	0.296	0.395	0.416
7	0.417	0.257	0.357	0.395
8	0.391	0.311	0.356	0.383
9	0.406	0.285	0.341	0.379
10	0.424	0.300	0.365	0.425

Table C.5: Experiment configuration

Experiment	**Map**	**Failure**
1	1	None
2	2	None
3	2	Leg 1
4	2	Leg 4
5	4	Leg 1 and 4
6	6	Leg 1 and 4

Table C.6: Experiment 1 results - Adapting to no leg failures using Map 1

Iteration	**Map Index**	**Fitness** (m)
1	13851	0.660
2	32178	1.130
3	9155	0.865
4	886	0.540
5	25225	0.500
6	28407	0.640
7	8572	0.515
8	31447	0.545
9	1944	1.190
10	389	0.330
11	10443	0.530
12	540	1.510

Table C.7: Experiment 2 results - Adapting to no leg failures using Map 2

Iteration	**Map Index**	**Distance** (m)
1	9336	1.170
2	22852	0.580
3	19470	0.245
4	26460	1.340
5	11448	0.440
6	29507	0.720
7	622	0.390
8	19892	0.510
9	161	1.335
10	4353	0.350
11	22766	1.340
12	14188	0.510
13	11564	1.180
14	440	1.150
15	16087	1.090
16	18178	1.130
17	2002	0.985
18	22827	1.040

Table C.8: Experiment 3 results - adapting to failure of leg 1 using Map 2

Iteration	**Map Index**	**Distance** (m)
1	9336	0.650
2	22852	0.790
3	19470	0.245
4	26460	0.925
5	29507	0.790
6	23549	1.035
7	1873	0.510
8	16224	0
9	2002	0.820
10	22766	0.420
11	11302	0.360
12	25284	0.310
13	161	1.245
14	14188	0.745
15	4980	0.465
16	11564	0.475
17	32691	0.910
18	1270	0.140

Table C.9: Experiment 4 results - adapting to failure of leg 4 using Map 2

Iteration	**Map Index**	**Distance** (m)
1	9336	1.185
2	22852	0.285
3	19470	1.160
4	18117	0.075
5	23549	0.060
6	29507	0.310
7	622	0.800
8	4353	0.845
9	14188	0.710
10	5732	0.580
11	4980	0.240
12	18178	0.190
13	28598	0.030
14	22083	0.585
15	2735	0.320
16	11564	0.370
17	1873	0.160
18	1059	0
19	440	0.455

Table C.10: Experiment 5 results - adapting to failure of leg 1 and 4 using Map 2

Iteration	**Map Index**	**Distance** (m)
1	10156	0.940
2	15396	0.075
3	1313	0
4	10027	0.440
5	5464	0.530
6	3314	0
7	2549	0.055
8	22966	0.580
9	9064	0.505
10	1727	0.235
11	22176	0
12	1017	0
13	32776	0.170
14	9113	0
15	2064	0.410
16	7338	0.730
17	32006	0.010
18	2154	0.985
19	18706	0
20	4809	0.575
21	18067	0.175
22	28217	0.990
23	1116	0.360
24	4641	0.670
25	2872	0
26	8702	0.050
27	18014	0.420
28	1251	0.655
29	35634	0
30	2383	0.050
31	2801	0.580
32	384	0.275
33	15398	0.085
34	9167	0.625
35	7820	0.880

Table C.11: Experiment 6 results - adapting to failure of leg 1 and 4 using Map 2

Iteration	**Map Index**	**Distance** (m)
1	284	0.855
2	9802	0.305
3	1722	0.860
4	5197	0.765
5	63	0.230
6	11478	0.225
7	9140	0.605
8	1901	0.280
9	9780	0
10	15552	0.150
11	25291	0.515
12	19079	1.190
13	12002	0.200
14	19229	0.100
15	22533	0.300
16	22742	1
17	14915	0.100
18	23	0.260
19	4593	0.265
20	36336	0.445
21	2080	0.630
22	23834	0.135

Appendix D

Graduate Attribute Documentation

D.1 Ethics Clearance

Application for Approval of Ethics in Research (EiR) Projects
Faculty of Engineering and the Built Environment, University of Cape Town

ETHICS APPLICATION FORM

Please Note:
Any person planning to undertake research in the Faculty of Engineering and the Built Environment (EBE) at the University of Cape Town is required to complete this form **before** collecting or analysing data. The objective of submitting this application *prior* to embarking on research is to ensure that the highest ethical standards in research, conducted under the auspices of the EBE Faculty, are met. Please ensure that you have read, and understood the **EBE Ethics in Research Handbook** (available from the UCT EBE, Research Ethics website) prior to completing this application form: http://www.ebe.uct.ac.za/ebe/research/ethics1

APPLICANT'S DETAILS		
Name of principal researcher, student or external applicant		Christopher Mailer
Department		Mechanical Engineering
Preferred email address of applicant:		mlrchr001@myuct.ac.za
If Student	Your Degree: e.g., MSc, PhD, etc.	B.Sc. Mechanical and Mechatronic Engineering
	Credit Value of Research: e.g., 60/120/180/360 etc.	46
	Name of Supervisor (if supervised):	Ms Leanne Raw
If this is a researchcontract, indicate the source of funding/sponsorship		NA
Project Title		Online Gait Adaptation of the RARL Hexapod

I hereby undertake to carry out my research in such a way that:

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

APPLICATION BY	Full name	Signature	Date
Principal Researcher/ Student/External applicant	Christopher Mailer		20/05/2020
SUPPORTED BY	Full name	Signature	Date
Supervisor (where applicable)	Leanne Raw		24/05/2020

APPROVED BY	Full name	Signature	Date
HOD (or delegated nominee) Final authority for all applicants who have answered NO to all questions in Section 1; and for all Undergraduate research (Including Honours).			
Chair: Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the questions in Section 1.			

D.2 Risk Identification Form

Department of Mechanical Engineering
Initial Risk Identification Form

Rev 1.1

- **Completion of this Risk Identification Form is required before any detail design or manufacturing is started.**
- **The purpose is to identify special safety requirements upfront, and to ensure that the necessary safety knowledge have been acquired, or specialist have been contracted.**
- **Failure to complete this form properly may result in refusal to perform your actual test / run your equipment.**
- **This form is incomplete without the attached supporting documentation as required for certain conditions.**

Student Name	Christopher Mailer
Supervisor	Ms Leanne Raw
Project title and number	Project 68: Online Gait Adaptation of the RARL Hexapod

Section A: Vessels under pressure

1. Does your design/apparatus have an operating pressure above 50 kPa(g), and holds gas or two-phase fluid heated to above atmospheric saturation conditions? *If YES, then complete the remaining section.*		NO
2. Study the South African classification of pressure equipment (SANS 347:2007). Note that a container made up of welded pipes is considered a pressure vessel.	Sign initials	
3. Does your pressure vessel fall within the Standard Engineering Practice (category 0)? *Attach a brief report showing the calculations performed and steps followed to arrive at the classification.*		
4. If **YES** to Question 3, you acknowledge the following: • Your final design will make use of appropriate pressure vessel calculations, and will be properly documented and verified by your supervisor. • You will complete a hydraulic test of 1.25 times the design pressure using water before actual testing or operation. This test will be witnessed by your supervisor and a photo of the pressure gauge reading will be included in your report. • Alternatively a pneumatic test may be performed in an enclosed environment. • If your setup is a refrigeration system, you will study the requirements of SANS 10147, and include in your final report the key aspects applicable to your design.	Sign initials	
5. if **NO** to Question 3, you acknowledge the following: • A suitable certified pressure vessel design & manufacturing company have been identified and has agreed to ensure all pressure regulations are met. *Proof of this must be attached.* • Your final report will contain a detail pressure vessel design pack, including a test certificate and certificate of compliance by the consulting company.	Sign initials	

Section B: Electrical installations

1. Does your design/apparatus make use of electricity above 50V where any element of the electrical design/manufacturing is done by yourself, i.e. not part of a bought-out component? *If YES, then complete the remaining section.*		NO
2. Read the Machine Lab Safety Rulebook (on Vula), and then consult Mr Maysam Soltanian (Electrical Engineering Machines Lab) regarding specific requirements. *Attach a brief summary of discussion outcomes and requirements.*	Sign initials	
3. Have you been advised to contract an external electrician to assist?		
4. if **YES** to Question 3, you acknowledge the following: • An electrician has been identified and he has agreed to assist. *Proof of this must be attached.* • You will include the electrical certificate in your final report.	Sign initials	
5. if **NO** to Question 3, you acknowledge the following: • You will generate proper electrical diagrams of your setup. • You will continue to consult Mr Soltanian w.r.t. wiring, equipment and testing. • You will have your Supervisor and/or Mr Soltanian witness the first power-on event.	Sign initials	

Section C: Hazardous chemicals and fuels

NO PROJECTS INVOLVING THE STORAGE OF HEATED FUEL MAY BE DONE

1. Does your design/apparatus make use of materials (chemicals, gasses, etc.) which may be hazardous to health, or the environment? **OR** Does your setup involve the use of flammable fuel, including LP Gas? *If YES, then complete the remaining section.*		NO
2. Consult with Mrs Penny Louw (CME lab) or Prof Genevieve Langdon (BISRU) or A/Prof Chris von Klemperer (Composites) regarding the specific requirements in terms of handling, storage, record keeping etc. *Attach a brief summary of discussion outcomes and requirements.*	Sign initials	
3. Attach the Material Safety Declaration Sheet (MSDS) for any hazardous material to be used.	Sign initials	

I have read all attached documentation and is satisfied that the necessary safety considerations will be adhered to during the physical execution of the project.	**Supervisor Signature**	**Date**
		11/06/2020

D.3 Risk Assessment Form

Department of Mechanical Engineering rev 1.0

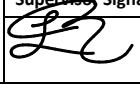
Risk Assessment Form

- **Risk Assessments are not required for simple workshop activities covered by the Safety Declaration. However permission must be obtained from the Workshop Manager before commencing any activity in the Workshop.**
- **A new Risk Assessment form needs to be completed for each major activity within your project.**
- **You are required to include this document (signed) in your bound project submission and mount a copy next to any rig / apparatus you are using.**

Your Name	Christopher Mailer
Your Supervisor	Ms Leanne Raw
Project title and number	Online Gait Adaptation of the RARL Hexapod - 68
Area Safety Warden	Ms Leanne Raw

This Section to be completed by the student (Must be typed and the declaration signed)			
Location where the activity will be done	RARL		
Describe the activity (use attachment with diagrams if needed)	Implementation and testing of the gait adaptation algorithm on the hexapod platform		
Names of persons involved in this activity	Christopher Mailer		
Describe in detail the risks you (and others) will face during this activity and the potential consequences of your activities	The risk of pinching from the servos. They are high torque servos and will severely hurt a pinched finger. The risk of spilling fluids on the hexapod electronics and permanently damaging them. Risk of Battery shorting and swelling. If the batteries start swelling there is a risk of explosion.		
Does this activity involve the use of any materials (chemicals, gasses, etc.) which may be hazardous to health, or the environment	**No**	Yes	*If Yes list the chemicals / gasses to be used and attach the MSDS(s) for these materials.*
Does this activity involve any equipment / device designed or built by you which is to be plugged into mains electricity?	**No**	Yes	*If Yes please consult with Mr Richard Whittemore or* Mr Maysam Soltanian (Electrical Machines Lab) *before connecting to the mains / switching on.*
Does the activity involve any new equipment / devices designed by you which will be pressurized with a gas or volatile liquids?	**No**	Yes	*If Yes please check the relevant SANS Pressurised Equipment Regulations and consult with A/Prof Fuls before testing.*
What precautions are required to protect against the risks detailed above?	Ensure that the hexapod platform is not handled when powered on and that it is placed at least half a meter away from dangling clothing and body extremities. Ensure that fluids are not on the same working surface as the hexapod platform. To limit risk of battery explosions the robot will initially be powered from an external power supply. Battery powered tests will only begin once tests to validate operation have been completed.		
Describe the personal protective equipment (PPE) required during this activity – specify in detail.	None		
Describe the shutdown procedure in detail.	Send termination command from control PC. Turn off power switch. Disconnect power supply/batteries.		
Describe any relevant emergency procedures, e.g. spillage response etc.	Response to spillage – Disconnect power supply and move hexapod to clean surface.		

I declare that I am aware of the risks associated with this activity and will take all necessary steps to mitigate these risks.	**Student Signature**	**Date**
		12/10/2020

I am aware of the student's intended activity, and have provided the necessary guidance, inputs, and oversight.	**Supervisor Signature**	**Date**
		12/10/2020

This section to be completed by the Area Safety Warden		
Level of supervision required (Please tick relevant block)	A = work may not take place without supervisor/warden present.	
	B = work may not take place without a 2nd party present.	
	C = no specific extra supervision requirements.	
I am satisfied that the student is aware of the risks associated with this activity and grant approval for it to proceed.	**Signature**	**Date**

D.4 Impact of Engineering Activity

Department of Mechanical Engineering

MEC4110W – Final-Year Project

Impact of Engineering Activity Rev5

By completing this assessment, you will help to demonstrate that you have met the requirement of ECSA's Graduate Attribute 7: Impact of Engineering activity. If it is determined that you have not successfully completed this task, you will be required to rework and resubmit this document. Should this still not be considered acceptable, you will not pass MEC4110W.

You are required to include this document in your Interim Report as an appendix. Your supervisor will read it and then sign as the reviewer.

In the space provided below, please write up to 300 words on how the technology in your project impacts on society, and then sign and date the form. You can consider "society" in one of or all three spheres: 1) your fellow students and other staff members, 2) the institution more generally, and/or 3) the broader society. You may need to consider the "downstream" impact of the technology in your project if you are undertaking a focused research-based project.

This project ultimately contributes to existing work on improving robots abilities to adapt to and mitigate unforeseen circumstances in uncertain environments. The project is currently limited to unexpected failures on a hexapod robot, however it can very quickly extended to unexpected terrains or unplanned interactions with people. This downstream impact of this is the development of robots with more robust behaviours, able to operate outside of the controlled environments of laboratories and factories. With more robust behaviours robots will be better suited to replace humans in dangerous tasks such as firefighting, disaster response, and search & rescue, thereby reducing risk to human lives.
An adverse effect of more widespread use of robots is the potential replacement of humans in tasks driven by the commercial benefit rather than morality, resulting in unnecessary unemployment and job displacement. Additionally, with more widespread use of robots and therefore closer interaction with peoples, comes the potential for malfunctions leading to the potential harm of nearby people.

This project also serves to continue work and development on the hexapod robot at the university, further developing the platform and its features for future research and contributing to the goal of the advancement of learning. Subsequent students will be able to use components developed in this project such as the simulator to further their projects and research. Additionally, a direct output of this project is a poster which will serve to inform students about the platform, stimulating future engagement and development. This all ultimately contributes to the production of more research at the university and the elevation of South Africa as a research hub within Africa.

Signed	Date
	18/06/2020

Reviewer's comments: _____

_____

_____

Reviewer's signature: _____ Date: _____