School of Electronic and Electrical Engineering
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

ELEC3875

Project Report

Generation of Robot Gaits using Low Level Parameters

Joshua Davy

SID: 2011483789	Project No. 33
Supervisor: Dr Craig A. Evans	**Assessor:** Roger Berry

School of Electronic and Electrical Engineering
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

ELEC3875 Individual Engineering Project

Declaration of Academic Integrity

Plagiarism in University Assessments and the Presentation of Fraudulent or Fabricated Coursework

Plagiarism is defined as presenting someone else's work as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance.

Fraudulent or fabricated coursework is defined as work, particularly reports of laboratory or practical work that is untrue and/or made up, submitted to satisfy the requirements of a University assessment, in whole or in part.

Declaration:

- I have read the University Regulations on Plagiarism [1] and state that the work covered by this declaration is my own and does not contain any unacknowledged work from other sources.
- I confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes.
- I confirm that details of any mitigating circumstances or other matters which might have affected my performance and which I wish to bring to the attention of the examiners, have been submitted to the Student Support Office.

[1] Available on the School Student Intranet

Student Name:	Joshua Davy	Project No.	33
Signed:	[illegible]	Date:	21/05/20

Generation of Robot Gaits Using Low Level Parameters

Final Report

Joshua Davy

el17jd@leeds.ac.uk

May 2020

1 Abstract

Traditionally the gaits of legged robots are inspired by what is seen in nature and on assumptions of what properties the optimum gait would have. This involves kinematic and dynamic analysis of the robot which becomes increasingly difficult with robots with greater degrees of freedom. Further, real actuators and joints are not ideal and are also difficult to mathematically model accurately. In this project the aim is to use optimisation techniques to allow a robot to develop its own gait free from human bias of what the optimum gait would look like. This technique means no kinematic or dynamic analysis of a robot is necessary, as the robot will attempt to learn a policy of moving its joints to optimise its gait. The project will consist of two main sections, firstly the design and build of a suitable robotic platform for experimentation and secondly, the investigation and design of suitable software to control the robotic platform.

2 Acknowledgements

I would like to thank Dr Craig A. Evans for supervising this project. His help and support have been vital to this project's success, and his ability to help debug and overcome countless issues I've faced, has saved me many painful hours. I would also like to thank Andrew Bilbrough for his technical help. Due to the COVID-19 outbreak, the final stages of this project had to be completed from my family home. I would therefore like to thank my mum for her tolerance of having robotic testing in her kitchen, the dog of which, was less understanding.

Contents

List of Figures

List of Tables

List of Algorithms

3 Introduction

3.1 Project Background

The recent work of groups such as Boston Dynamics and MIT is well known with many videos featuring legged robots such as Atlas [1] and Mini Cheetah [2] going viral on social media. Although mechanically these robots are very capable, the crux of the problem lies in the control. Legged robotics need to quickly react to changes, understand and process their environment, and plan their movements accordingly.

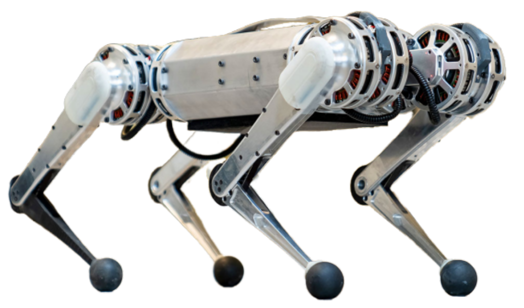

Figure 1: MIT Mini Cheetah [2]

To this end, often machine learning techniques are applied for the purpose of robotic control. Learning techniques allow a robot to develop its own control processes through experimentation with its environment. Machine learning is the study of algorithms designed to complete an objective without being given explicit instructions on how to do so. A machine learning algorithm aims to complete its objective instead by forming patterns found by training [3]. In this project, the focus is on gait generation for a legged robot. Traditionally the gaits of legged robots are inspired by what is seen in nature and on assumptions of what properties the optimum gait would have. More recently, machine learning techniques are being applied to the generation of gaits to overcome the issue of designing and optimising a gait by hand. These machine learning techniques require large amounts of test data in order to produce good results and therefore often rely on simulations where the collection of data is fast and requires no human intervention. The issue of simulations is they require accurate mathematical models of the robot and its environment which are not always attainable. In fact, it is impossible for any simulation to perfectly model the circumstances a robot will face in reality, and a robot whose gait performs well in simulation may perform poorly in the real world. This issue known as the *reality gap* is an active part of robotics research [4]. Several solutions exist such as Automatic Domain Randomisation [5] or Intelligent Trial and Error (IT&E) [6].

Of course, these issues could be avoided entirely if instead of simulation, real robots were used for the evaluation of gaits. This comes with several disadvantages. Firstly, the speed it takes to evaluate the gait. In simulation, gait evaluation can take place as fast as the computer is able to process it allowing thousands of evaluations in the time it would take a real robot to complete one. Further, multiple simulations can run in parallel allowing faster evaluation. Although multiple real robots can be used this comes at significant financial cost. Secondly, in a simulation it may be perfectly reset, whereas in reality this is impossible meaning no two real gait evaluations will be under the same conditions. Identical gaits tested multiple times may perform drastically differently in the real world. For these reasons, the optimisation of gaits for state of the art legged robots rely on simulations for training or a mixture of simulation and real world testing (Sim-to-Real [7]).

This project considers the problem of gait generation under the following constraints:

- No simulation of the robot and it's environment. All gait evaluations will need to use the real robot.

- No kinematic or dynamic analysis of the robot will be first derived so low level parameters of the gait will be used. This means, for example, instead of commanding the position of the foot of the robot, the raw control signals to the robot motors will be used.

With these constraints the robot platform design is independent from the gait control meaning the robot's physical properties or environment could change and the gait generation process would still be suitable.

As discussed above, these constraints mean the generation of gaits for a modern dynamic robot in the real world would be infeasible as the training process would take a impractical amount of time. Therefore, for this project the scope is limited as follows:

- The used robot platform will be statically stable. This will mean that the robot is never in a position where another state is unfeasible to move to. Further, the training process will require less human intervention as the robot cannot topple.

- The gait control will sensor less and the environment constant (single state). This means the robot is not expected to make specific plans for obstacles or path plan. That is not to say that a gait that can deal with obstacles will not be more advantageous. The gait will therefore be a constant cyclic pattern, the properties of which will be generated.

3.2 Problem Definition

In this approach, the problem of gait generation is framed as an optimisation problem

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, f(h(g(\pi))).$$

π is the policy, a vector of values that define the current gait. g is the gait parameterisation; this maps the the policy to the cyclic sequence of control values which form the gait. h is the evaluation function, the testing of a gait on the real robot. Due to the evaluation taking place on the real robot, h is slow to evaluate and likely to be noisy due to uncontrollable disturbances. f is the fitness function. This maps the gait evaluation to a fitness value. The higher the value of the fitness function will correspond to a better performing gait. π^* is the global maximum. The theoretical policy with the highest fitness and therefore the best performing policy.

The project has therefore been broken down as follows:

- Firstly, the design and testing of a 2D simulation of a simplified robot as a initial experiment to test the theory and aim to guide the approach for the real robot.
- The design and build of a suitable robotic platform complete with electronics and wireless interface software.
- The design of several gait parameterisation methods. This will allow comparison between methods.
- The investigation of suitable optimisation methods.
- The comparison of these optimisation processes with different methods of gait parameterisations in various scenarios.

In semester one, the focus was on initial design for the robotic platform, the 2D simulation, sourcing parts, and producing a PCB to host the control electronics. Due to unexpectedly long lead times on certain key components, assembly of the platform was pushed to the start of the second semester.

3.3 Key Terms

Gait - The pattern of steps a robot makes to walk.

Forward Kinematics - The calculation of the position of the foot of the robot leg from its joint angles.

Inverse Kinematics - The calculation of the joint angles of the robot leg from a desired foot position.

Parameter - A single parameter describes one property of the robot's gait.

Policy - The set of parameters that fully describe the robot's gait. The policy has a size of n.

Gait Parameterisation - The function that maps the policy to the sequence of joint positions that form the gait.

Fitness - The resulting score of testing the gait on the real robot. A higher score corresponds to a better gait.

Parameter Space - The n dimensional space of possible parameter values mapping to the corresponding fitness. The goal of any of the machine learning algorithms is to explore this parameter space in search of the maxima.

Local Maxima - A policy in the parameter space which has a higher fitness then the surrounding points.

Global Maxima - The policy with the highest fitness in the parameter space. The goal of optimisation is to find this global maxima.

4 2D Simulation of Gait Optimisation

4.1 Simulation Design

A simple 2D simulation of a robot with a singular leg (A *crawler* robot) was created in order to test the theory and experiment in simulation.

The code is written in Python and makes use of Pymunk [8] a Python wrapper for the Chipmunk physics simulation software. The gait was parametrised as 5 sets of pairs of joint angles (for each of the two joints in the leg). The gait is then formed by cycling through the pairs of joint angles. The policy is then evaluated by cycling through it on the simulated robot five times and the fitness is the distance travelled. Optimisation is performed by stochastic hill climbing where each generation is formed by random variations of the last [9]. Variations of the policy are formed by varying each parameter with a Gaussian distribution $\mathcal{N}(\theta_i, 0.1)$ where θ_i is the current parameter value. The below link is a video demonstrating the simulation:

`https://youtu.be/vynnbg6c0SE`

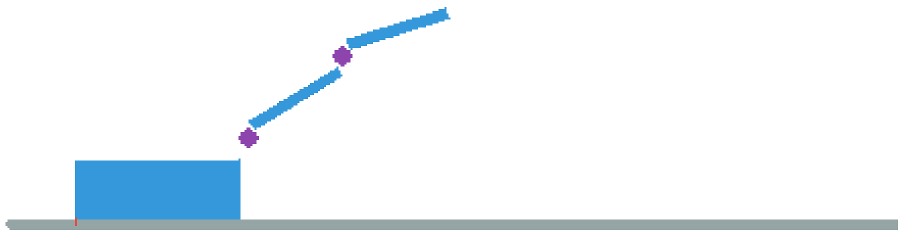

Figure 2: The Single Legged Simulated Robot

Algorithm 1: Stochastic Hill Climbing

```
$currentPolicy \leftarrow randomInitialPolicy()$;
**while** !$done$ **do**
    $polices \leftarrow create10RandomVariations(currentPolicy)$;
    $evaluate(polices)$;
    $currentPolicy \leftarrow getHighestScoring(polices)$;
**end**
```

4.2 Results

Results were discussed in depth in the interim report. The main conclusions of the simulation were despite the simplicity of the optimisation process the formation of well performing gaits was possible in low number of evaluations. Due to the inherit randomness of the algorithm, runs could be radically different in results. With some runs, the algorithm could converge to a policy with a fitness of over 1000, some runs could not get past 300. This clearly illustrates the local maxima problem of optimisation where the algorithm gets stuck at a maxima and cannot improve its policy from there. In Figure 3 we see a typical run of the simulation. At first the robot makes little progress in forming a gait until around the 5th generation where there is a radical increase in policy fitness. After generation 50 the gait no longer makes any meaningful increase in fitness. This shows the algorithm is stuck at a local maxima as we know better policies do exist yet the algorithm can longer improve.

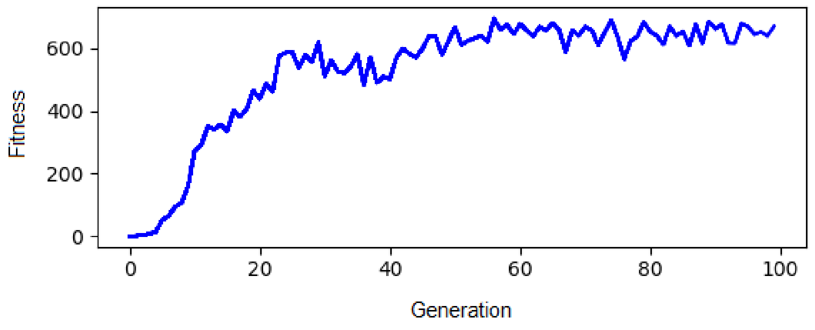

Figure 3: Simulation Results

5 Platform Design and Build

The physical robot needs to be specifically designed in order to suit the gait training process. Mainly there needs to be a way of accurately measuring velocity feedback from the robot. The system also must be stable in any position it may find itself in. This is due to the extensive training process the robot will have to on go to optimise it's gait. In [9], training was in the order of hours therefore the robot needs to be able to continue training with minimum human intervention.

5.1 Assembly

The designed platform has two 3-DOF legs at the front with wheels with mounted encoders at the rear. The purpose of the wheels is to keep the platform statically stable and prevent the robot toppling. The encoders mounted to the wheels provide the feedback to the system allowing the measurement of the robot's distance travelled, velocity and acceleration. Initially all servos were Hitec HS-422, however in initial testing the robot did not have the torque to lift it self off the ground on one leg. The 'knee' joints of the robot (joint two) were replaced with high torque Hitec HS-645MG servos to compensate for this issue.

A small caster wheel was added to the front of the robot to improve stability (not shown in CAD). In experimentation the effect of this caster wheel will be tested.

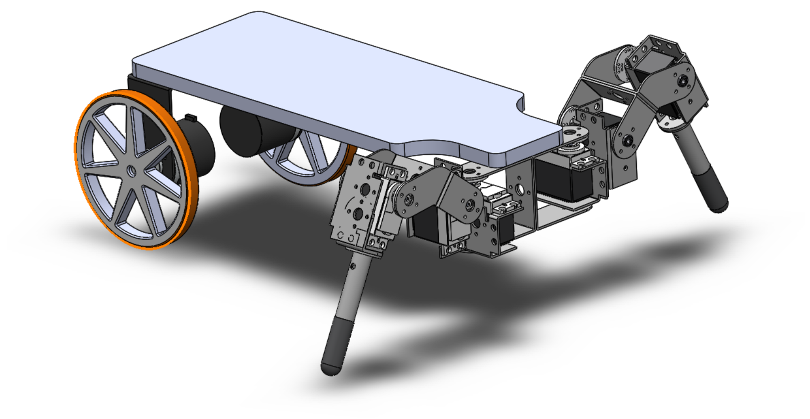

Figure 4: Final Platform CAD

Component	**Description**
Lynxmotion 3DOF T-Hex Leg Kit Pair	Robot Legs.
4 x Hitec HS-422 Servos	Servos for joints one and three in each leg.
2 x Hitec HS-645MG	Servos for joint two in each leg.
2 x Aluminum Long "C" Servo Bracket	Brackets for connecting to robot legs to base.
Robot Base	Laser cut 8mm acrylic robot base.
2 x 400P/R encoders with mounting brackets.	Encoders for mounting wheels.
2x 106mm aluminium wheels with rubber coating.	Wheels for rear of the robot. High friction.

Table 1: Summary of Mechanical Components

5.2 Robot Control Board

For control of the robot a custom PCB was designed to host the main circuitry of the robot. The board contains a ESP32 [10] microcontroller which was selected for its WiFi and Bluetooth capabilities, as well as it's low power consumption and high processing power. The board contains mounting point for a inertial measurement unit, a few buttons and LEDs for general use, ten servo mounts as well as encoder connections. The Robot Control Board was designed to be a general control board for servo based robot projects. This, along with the interface software to be open sourced for others to use.

The initial PCB design (Revision 1.0.0) had several small oversights. Once the design had been fully tested and all issues had been found, a second revision with several new components and key changes were added (Revision 1.1.0). Below lists the issues faced with Revision 1.0.0 and the changes made in Revision 1.1.0.

Issue	**Fix**
Servo and encoder pins were mislabelled on silkscreen.	Silkscreen text was changed to the correct pin names.
Encoder 1 header orientation was opposite to encoder 2.	Encoder 1 header orientation flipped.
Encoder 1B pull up resistor connected to Pin 12 of ESP32 causes the processor to fail to boot.	Switched to internal pull up resistors for the encoders.
Pins used for tactile buttons do not have internal pull ups.	External pull up resistors added.
Servos cut out due to lack of current.	Two 10mF capacitors added to smooth voltage. Separated microcontroller and servo power rails. Screw terminals added.
IMU used is hard to source and relatively expensive.	Switched to a more common and cheaper IMU.

Table 2: Summary of Changes in Revision 1.1.0 of the Robot Control Board.

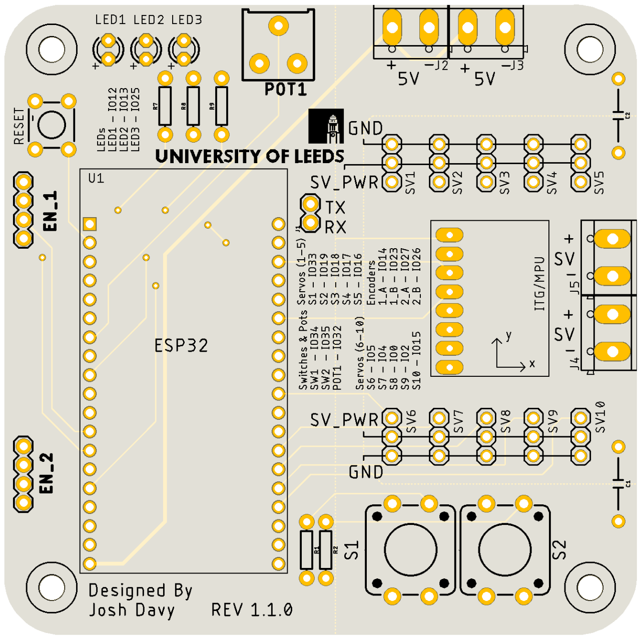

Figure 5: Revised PCB Design

5.3 Power

Originally, the intent was to use a generic 5V 3A rechargeable battery to power the platform. In initial testing, it was found that this was unsuitable due to not being able to handle large current spikes caused by the servos. Added parallel capacitors made little difference. This power loss would cause the Bluetooth connection between the PC and the robot to be reset and the connection would have to be reestablished. For the purpose of experimentation, a bench power supply was instead used to power the robot with a 3 meter cable. With the second revision of the PCB, the servo and microcontroller power rails were separated. A separate small USB power bank was then used to power the microcontroller. This prevents issues caused by voltage drop when the servos draw large currents. A lithium battery and regulator circuit could be added if wireless capabilities became necessary however for use in the lab, the bench power supply was adequate.

6 Platform Interface Software

For control of the robot, a Bluetooth link between a PC and the ESP32 microcontroller is used. The ESP32 software processes and executes the instructions sent by the PC. These instructions are very low level and correspond to, for example, setting servo angles or reading the encoder values. This way, the majority of processing can be done on the PC end and will allow fast changes to the program without having to reprogram the microcontroller. The interface software communicates using the CMDMessenger serial protocol [11], allowing for two way wireless communication between the ESP32 and PC. The aim was to keep this software as well as the PCB as general as possible so they could be used in other robotic projects without significant change.

The ESP32 has built-in Bluetooth capabilities allowing a wireless serial link between the host PC and microcontroller. The ESP32 was programmed using the Arduino framework with the PlatformIO IDE. This was chosen due to the existence of existing libraries for interface with servos and encoders. The PC software is written in Python and has accessor functions allowing the control of the robot's actuators and reading of the sensors as python data types. In-built error detection can detect when the established connection has been lost.

Command	**Parameters**	**Return Command**	**Return Parameters**	**Description**
hello		*hello_return*		For testing the connection.
set_servo	*servoID (0-9), angle (0-180)*	none		Set servo angle.
set_led	*ledID (0 to 2), value (0 or 1)*	none		Set LED state.
get_button	*buttonID (0 - 1)*	*button_return*	*value (0 or 1)*	Gets the state of a button
get_pot		*pot_return*	*value (0 - 255)*	Gets button value.
get_encoder	encoderID (0 or 1)	*encoder_return*	*value*	Returns the encoder count.
		error		Unexpected command.

Table 3: CMDMessenger [11] Commands

7 Gait Parameterisation

In order to optimise, the robot's gait needs to be represented as a vector, referred to as a policy. Mathematical optimisation of the gait over the domain will aim to find the policy which leads to the optimum gait. Two different ways of representing the gait are compared. Both gaits aim to use low level parameters as opposed to the solving of inverse kinematics of the robot, and having the gait parameters as properties of the cyclic shape of the foot's movement. In the design of the robot gait parameterisation, a balance has to be struck between a design which allows as much freedom as possible for the robot to adapt to it's environment and keeping the dimensionality of the policy low. Too low a dimensionality would mean the robot's gait would have limited variations and there possibly would not exist a suitable gait in the domain for the current environment. Too high a dimensionality would mean a very large domain, and due to the time taken for the evaluation of the robot optimisation, it would struggle to converge on a suitable gait. To help reduce dimensionality of the gait and speed up training time, each leg will perform the same movement pattern with the legs moving out of phase with each other. Therefore, the gait parameterisation only represents the movement of a single leg (three joints).

7.1 Linear Interpolated Gait

In this policy the gait is represented as n sets of joint angles. The gait is formed by cycling through these sets of joint angles with intermediate steps being linearly interpolated with the full policy π as

$$\pi = [[\theta_{1,1}, \theta_{2,1}\theta_{3,1}], ..., [\theta_{1,n}, \theta_{2,n}\theta_{3,n}],$$

where $\theta_{j,s}$ is the j^{th} joint in the s^{th} step in the gait. n represents the number of steps in the gait. Possible values of $\theta_{j,s}$ are limited to the 0 to 180 degree range of the servo motors.

The policy dimensionality is therefore

$$d = no.of joints \times n,$$

where in this case, the no. of joints is three. In testing, linearly interpolated gaits with three and four steps are compared.

7.2 Oscillatory Gait

In this policy, inspiration is taken from animal gaits. In animal gaits it can be observed that each joint moves in a roughly sinusoidal pattern see Figure 6 Each joint has the same frequency, but the amplitudes and relative phases vary. The aim is to use this information to parameterise the robot's gait.

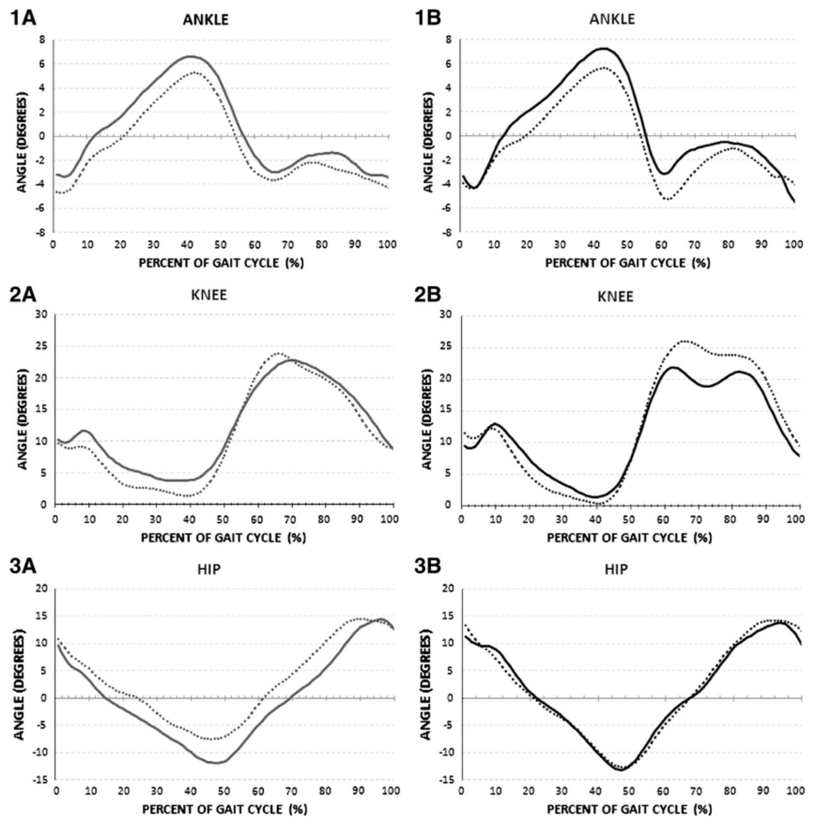

Figure 6: Human joint angles at comfortable (A) and fast (B) walking pace. [12]

Therefore each joint is modelled as:

$$\theta_{j,t} = A_j sin(\omega t + \phi_j)$$

with the full policy π as:

$$\pi = [[A_1, \phi_1], [A_2, \phi_2], [A_3, \phi_3]$$

where j is the the joint index, t is time, and ω is the fixed angular frequency (constant for all joints). A_j and ϕ_j are the joint parameters. A_j is limited to the range 0 to 90 to stay within the servo limits. ϕ_j is limited to 0 to 180 degrees so joints can be in any phase with any other joint. The dimensionality of the policy is therefore:

$$d = no.of joints \times 2$$

7.3 Fitness Function

To compare policies they are represented as 200 discrete sets of joint angles to be sent to the robot. For evaluation, the robot cycles through this sequence of joint angles with a fixed time step of 0.025 seconds between them, to allow time for the servos to move between points. The robot runs this sequence three times and returns the change in encoder measurements which are then used to calculate fitness. There is a 3 second pause between each gait evaluation to let the robot settle back to a stable state. During this time the robot lifts its legs vertically to rest on its wheels. Each evaluation takes 18 seconds.

The testing of a policy returns the difference in encoder values for each wheel

$$c_1, c_2 = h(g(\pi))$$

where c_1 and c_2 is the pulse count from the left and right encoder respectively at the end of the evaluation. This corresponds to the total rotation of each wheel during the gait evaluation.

To measure the quality of the gait we define the fitness function as follows:

$$f(c_1, c_2) = (c_1 + c_2) - \alpha |c_1 - c_2|$$

The fitness function is designed to produce low values for gaits with more desirable traits. The first part $(c_1 + c_2)$ will produce high values for gaits that travel the furthest distance. The second part $-\alpha |c_1 - c_2|$ produces high values for gait's that travel in a straight line as it punishes gaits with large differences between the encoder values. α is a constant. With the 400 pulses per rotation encoders used a value of 0.05 was used for α. The decision on this value is fairly arbitrary as it represents the weighting of importance of distance travelled to how straight the gait is.

8 Gait Optimisation Techniques

Optimisation will seek to find the optimum policy for the robot. The theoretical optimum policy is the one that produce the highest fitness. The evaluation of each gait takes 18 seconds on the real robot. The robot needs to be supervised and readjusted in between evaluations. This limits the amount of gait evaluations can be made. For the purpose of comparison, each optimisation algorithm has a maximum of 100 evaluations on the robot. Including adjustment time this means each optimisation algorithm will have 30 minutes to produce the best gait possible. Another further challenge is the noise present in the results. Although efforts are made to make each gait evaluation have the same environment noise in the results are inevitable. The ideal optimisation algorithm would make the most out of its limited gait evaluations, be resistant to noise in these results and produce a policy with the highest fitness possible. For the purpose of optimisation all policy parameters are scaled to the range $[0, 1]$.

8.1 Hill Climbing with Random Starts

As seen in the simulation (Figure 3), hill climbing, despite its simplicity is an effective optimisation algorithm. The main issue seen in the simulation was the issue of local minimas leading to different runs of the algorithm producing very different results. Therefore, the algorithm here is modified to firstly evaluate five random policies then choose the best one to begin hill climbing with. This will prevent the algorithm starting with a poor initial policy and lead to a better chance of reaching a good policy in the limited gait evaluations. Hill climbing, due to its simplicity and somewhat random approach provides a good benchmark to compare more complex optimisation processes.

Algorithm 2: Stochastic Hill Climbing with Random Starts

```
$polices \leftarrow fiveRandomPolicies();$
$currentPolicy \leftarrow getHighestScoring(polices);$
while !$done$ do
    s $polices \leftarrow create10RandomVariations(currentPolicy);$
    $evaluate(polices);$
    $currentPolicy \leftarrow getHighestScoring(polices);$
end
```

8.2 Genetic Optimisation

Genetic optimisation is a technique based on natural evolution [13]. Policies exist in a *population* that go through *crossover* and *mutation* operations in order to improve the population of policies. A crossover operation aims to replicate breeding in the population where two parent policies are selected and a child policy is produced by combining parent parameters. A mutation operation aims to replicate natural genetic mutation in evolution by randomly adjusting parameters in policies in the hope that this will improve the policy. This allows exploration of the parameter space in the search of the global maximum.

Algorithm 3: Simple Genetic Optimisation

```
$population \leftarrow tenRandomPolicies();$
while !$done$ do
    $populationFitnesses \leftarrow evaluateAll(population);$
    $parent1 \leftarrow highestScoring(population, populationFitnesses);$
    $parent2 \leftarrow$
      $secondHighestScoring(population, populationFitnesses);$
    $childPolicy \leftarrow crossover(parent1, parent2);$
    $population \leftarrow replaceWorstPolicyWith(childPolicy, population);$
    $population \leftarrow randomMutations(population);$
end
```

In this implementation the population was of 10 policies. Crossover operations were performed by combing the first p parameters of $parent1$ and the last $p-1$ of $parent2$, where p is uniformly selected in $p \in [0, n]$, where n is the policy dimensionality. Mutation is performed on 4 random policies in the population per iteration, with half of those parameters being randomly changed with a Gaussian distribution $N(\mu = \theta_i, \sigma^2 = 0.3)$ where θ_i is the current parameter value. Genetic algorithms have the advantage that they will cover more of the parameter space in search of an optimum policy. The mutation aspect allows small improvements to the policy and crossover allows policies to be combined so aspects of each that work well could come together to form a improved policy.

8.3 Bayesian Optimisation

Bayesian optimisation is an effective way of optimising gaits on real robots [14] due to working well with low dimensionality, and expensive optimisation problems. Hill climbing and genetic algorithms can be thought of as trial and error approaches which rely on randomness to improve their policy. Bayesian optimisation is different; instead the optimisation algorithm aims to create a model that represents the real world results of the robot. The optimisation algorithm uses its evaluations to improve the quality of this model. This model is much faster to evaluate then the real robot evaluation $f(h(g(\pi)))$ allowing the optimisation process to make fast approximate gait evaluations. Therefore, allowing the limited real evaluations to be polices which it will believe have higher fitness.

Algorithm 4: Bayesian Optimisation

```
$model = GaussionProcessRegression();$
while $!done$ do
    $policy \leftarrow acquisitionFunction(model);$
    $policyFitness \leftarrow evaluate(policy);$
    $model \leftarrow improveModel(policy, policyFitness, model);$
end
```

The following notes are adapted from [15] The model is formed by a Gaussian process $p(\pi)$. This model aims to replicate $f(h(g(\pi)))$, the real robot evaluation.

A Gaussian process models noise in the results as a Gaussian distribution

$$p(\pi) = m(\pi) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

where $m(\pi)$ is the mean (the expected) value of $p(\pi)$ and ϵ is noise in the results. Therefore p is distributed as

$$p \mid m \sim \mathcal{N}(m(\pi), \sigma_\epsilon^2)$$

A Gaussian distribution is therefore defined in two parts. Firstly, the mean function $m(\pi)$ that returns the mean value of p at π and secondly, k the covariance function that returns the covariance between two polices $k(\pi, \pi')$. In the multidimensional case, for a data set of policies $\boldsymbol{\pi}$ the evaluation function is modelled as

$$p \sim \mathcal{N}(m(\boldsymbol{\pi}), \mathbf{K})$$

where $\mathbf{K}$ is the covariance matrix of all policies in $\boldsymbol{\pi}$

$$K_{ij} = k(\pi_i, \pi_j)$$

The Gaussian process therefore does not just return the expected value of the evaluation function given a policy, but the expected mean and variances over a Gaussian distribution. As more evaluations are completed on the robot, the data set $\boldsymbol{\pi}$ can be updated, improving the model. The policy to evaluate is chosen by the acquisition function. The expected improvement function $EI(\pi)$ is used here. The next policy to evaluate is the one where the expected improvement is the greatest out of the domain of all possible polices.

$$EI(\pi) = E[max(0, p(\pi) - p(\hat{\pi})]$$

$$\Pi = \operatorname*{argmax}_{\pi} EI(\pi)$$

where Π is the next policy to evaluate and $\hat{\pi}$ is the current optimum policy.

The advantage of this optimisation technique is that less gait evaluations will be needed as the optimisation algorithm can use the model to predict those which will perform poorly and not evaluate them and focus search on the areas where the expected fitness is high. Bayesian optimisation is widely used in the tuning of neural network hyperparameters as these are expensive to evaluate and have noise in the results. The implementation of Bayesian optimisation was provided by the Scikit-Learn library [16].

9 Results

9.1 Flat Surface Optimisation

Initial testing was on a flat smooth surface with no constraints on the robot. The caster wheel is used for all tests. Comparison is made between the three presented optimisation techniques and three styles of gait parameterisation.

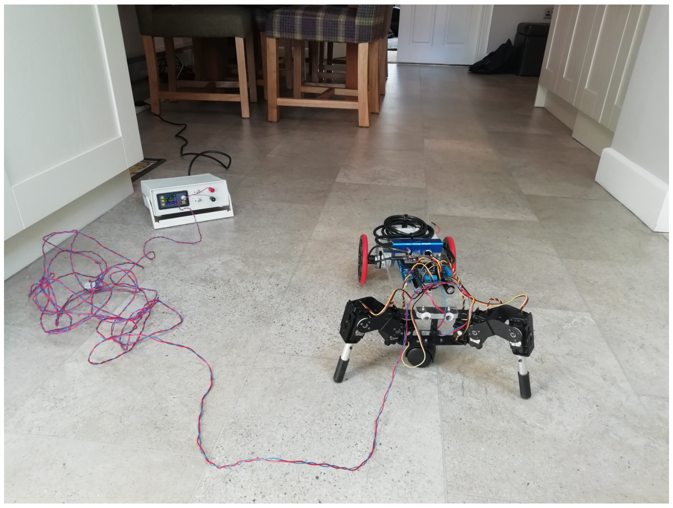

Figure 7: Experimental Setup for Flat Surface Optimisation

9.1.1 Hill Climbing with Random Starts

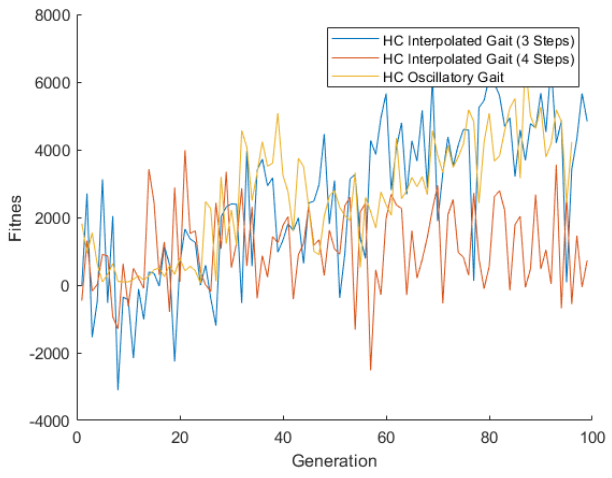

Figure 8: Comparison of Gait Parameterisations with Hill Climbing Optimisation

Hill climbing performed well at gait optimisation with three step interpolated and oscillatory gaits, producing gaits that travelled a good distance (around 1.5m per evaluation) and moved smoothly and sensibly in a straight line. The five random starts allowed the hill climbing process to start with a good policy and help converge. However, hill climbing struggled with the higher dimensional interpolated 4 steps gait. The initial five random starts did not produce a good starting point policy for the robot and therefore the optimisation process struggled to converge to a good policy as seen in the low end fitness. Despite hill climbing's simplicity it does perform well however, is not as robust as other optimisation processes due to its reliant on randomness. Despite this, hill climbing performed well in the noisy environment of gait generation and for the lower dimensional gaits produced good results in a small number of evaluations.

9.1.2 Genetic Optimisation

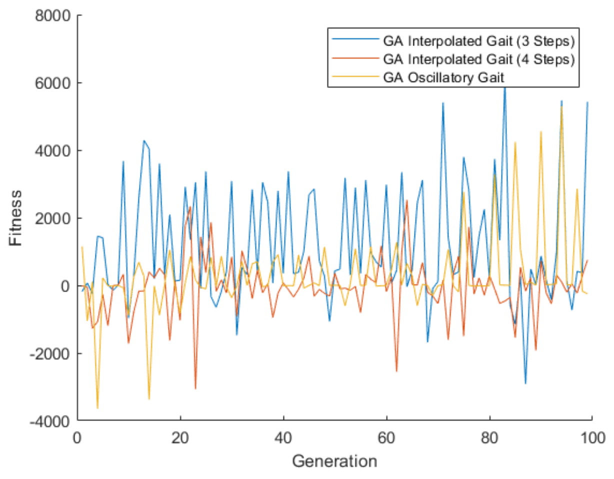

Figure 9: Comparison of Gait Parameterisations with Genetic Optimisation

Genetic optimisation performed poorly in all cases. With the final population of polices barely improving from the start point in terms of fitness. Some occasional good policies can be observed but can be attributed to noise and not the effectiveness of the optimisation process. These policies were not explored further and reflects the poor performance of the genetic algorithm. This poor performance was also seen in the work in work of [9]. Genetic optimisation does not suit well to gait generation. It fails to deal with noise in the readings and converge with the small amount of available evaluations. However, the volume of the parameter space explored by the algorithm was higher than the other two optimisation processes and with enough time would have likely of converged. A further weakness of the genetic algorithm was the *crossover* operation. Crossover of policies may make sense in large dimensionality optimisation problems but here not as much. For example, two polices could perform well but have leg movements out of phase of each other. Combining these polices would not produce a better child policy. Genetic algorithms are better suited to large dimensional, cheap evaluation optimisation problems where there is less noise in results.

9.1.3 Bayesian Optimisation

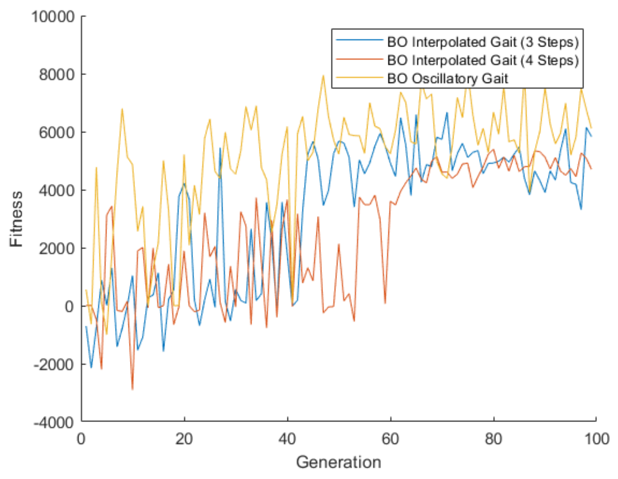

Figure 10: Comparison of Gait Parameterisations with Bayesian Optimisation

Bayesian optimisation was found to be the most effective of the explored optimisation algorithms for gait generation. In this process the highest fitness gaits were observed. In all gait parameterisations, gaits were observed which moved smoothly and fast, travelling up to 2m in a single evaluation in the case of oscillatory gait. Due to Gaussian process regression modelling the noise in the fitness values, the optimisation process could take that into account when exploring the parameter space. Looking at the output of the gait generations, a clear breakthrough point can be observed in all cases. This is where the optimisation process has found a policy that performs well and is robust to slight variations. From then on only small tunings of that policy are needed in order to improve it as can be seen in the smaller amount of variation after the breakthrough. Bayesian optimisation scales well with policy dimensionality, although it is expected that this would break down as dimensionality increases. The disadvantage of Bayesian optimisation is its computational complexity and would be difficult to implement on a simple microcontroller due to the computational cost of evaluating the Gaussian model. A demonstration of gait generation with Bayesian optimisation is available here: `https://youtu.be/q9s3zgLysts`

9.1.4 Interpolated Gait

The interpolated gait parameterisation worked well as way of defining a robots gait. It struck a good balance between allowing a lot of variation in possible gaits while also converging to a good policy under optimisation. The movement of the gait could occasionally jolt as the robot moves between steps and was not as smooth as the sine wave movements of oscillatory gait. The choice of number of steps depends on how complicated the movements of the robot are required. With higher steps comes higher dimensionality however and slows the optimisation process. The dimensionality of the policy is $no.of joints \times no.of Steps$ so could scale badly to higher dimensional robots.

9.1.5 Oscillatory Gait

Oscillatory gait produced the best performing policy in terms of fitness. Because all joint movement were based on sinusoidal waves all gaits, even those that performed badly moved smoothly without jolts. However, this can be a disadvantage as sometimes jolt movements may be an advantage to a gait. The policy dimensionality scales well with no. of joints and therefore would be better suited for higher degrees of freedom robots then the interpolated gait. However, this comes at the expensive of limitations of the complexity of produced gait as all joints are locked to sinusoidal movements.

9.1.6 Experimental Issues

Some unforeseen issues were observed with the experimental setup. Firstly, the amount of intervention needed to right the robot if a poor performing gait caused the robot to topple. This significantly increased the time required to perform the experiments. In total, it took over 5 hours to test all optimisation process with all gait parameterisations. Secondly, the robot occasionally started developing undesirable polices that produced a high fitness but were poor gaits. For example, in the initial run of hill climbing with four interpolated steps, the robot began to develop polices which involved moving forward then lifting the wheel off the ground and letting it spin, before performing the same with the other wheel. This returned a high end fitness as both wheels will have a large encoder difference from their initial state. When this happened, the run was restarted.

9.2 Optimising with Constrained Movement

9.2.1 Without Stabilising Wheel

Testing without the front caster wheel lead to poor results. The hope was the robot would learn a policy which walked and lifted its front off the ground. However, the robot would develop a policy which involved dragging its front across the ground. This is caused by several factors. Firstly, the fitness function cannot punish policies that involve dragging the robot as it is based on the pure encoder count. Even if the robot did learn a policy that lifted its front, it would likely still have a lower fitness than polices that dragged. This could be solved by using measurements from the IMU to reward polices highly that did lift the front. However, a more complicated fitness function leads to a more complicated optimisation surface and would likely increase the required number of evaluations to form a good policy. Secondly, the process of lifting the front off the ground would take several steps. This makes the necessary gait complex as it must combine the lifting and walking.

9.2.2 Using Lower Degrees of Freedom

The gait training process was tested with the robot legs locked forward with the upper joints fixed in position. This constrained the movement of the robot so a radically different style of gait was needed compared to higher degree of freedom modes. Similarly to the attempts without the stabilising wheel, the robot struggled to generate a gait in this scenario as it required the robot both lifting it's front off the ground and walking forward. The servos struggled to hold the necessary position against the robots weight.

9.2.3 With Only One Leg

Testing with only one leg attached to the robot and the stabilising wheel resulted in mixed results. For the robot to return a high fitness from a gait there must be little difference between encoder counts (to encourage the gait to develop to move in a straight line). This caused issues with generating gaits with one leg as the robot would move in a curve as the leg pushed against the ground. For the robot to return a high fitness, the leg would need to move in a very specific manner in order to keep the encoder difference low. The optimisation process failed to find a gait capable of this in the low number of evaluations. The gaits that developed tended to curve away from the side with the leg.

9.3 Alternative Terrain Optimisation

Instead of using a flat surface, the gait training process was tested on a more challenging environment. Grass provided a much more difficult environment for the robot to train in. Friction is obviously much higher and the terrain uneven and tall compared to the size of the robot. Bayesian optimisation was tested with both 3 and 4 step interpolated gaits for the training process, due to this technique performing well in flat surface testing. With 3 steps, the robot struggled to form a well performing gait in the environment. The 3 steps of gait were not enough to form a cyclic pattern of placing the foot moving it across the ground, lifting and moving forward. With 4 steps a much more viable gait was formed with the robot moving in a sensible pattern in a straight line. The servo motors of the robot struggled under the uneven terrain as more torque was needed to move the robot's wheels forward compared to the smooth flat surface. Some of these issues can be observed in this video: `https://youtu.be/qL6CXBV9FY8`

10 Conclusion

In this project, the process of gait generation for real robots has been explored. Firstly, a 2D simulation was created which acted as a proof of concept for the project. In simulation, the hill climbing optimisation algorithm was applied to a simple model robot to develop a gait and proved the feasibility of the project. Secondly, a suitable gait training platform was developed. This platform used commercially available parts allowing quick assembly. A custom PCB was designed to house the microcontroller and support electronics and Bluetooth interface software was developed for communication between the robot and a host PC. The PCB and communication software was deliberately made very general in the hope it may be applied to other robotics projects in the future. Several ways of parameterising the gait of the robot were conceptualised and suitable optimisation methods were investigated. Finally, gait parameterisations and optimisation methods were experimented with in various environments to test their effectiveness and to make comparisons between them.

As seen in the work of [14] and [17], Bayesian optimisation provides a power tool for gait training. The ability for the process to model the evaluation function and make predictions on future gaits allow it to effectively form a suitable gait in the noisy expensive environment. The gait parameterisations presented provide a strong method for represented robotic gaits at a low level and are not reliant on the robot kinematics or dynamics to be solved. These methods provide a good balance between allowing many different gaits available whilst also keeping the dimensionality low to allow optimisation processes to find quality gaits in the low number of evaluations available.

Despite this, these techniques have large limitations. Firstly, the time involved to train the gaits cannot compare to the speed of parallelised simulations which can perform at hundreds of evaluations per minute. Secondly, here the fitness function has been kept very simple; this simplifies the optimisation surface and helps the robot find a suitable gait more quickly. The issue of this simple fitness function is that it can return poor gaits with high fitness, as it is difficult to write a simple fitness that can well define all the desired properties of gait (e.g. stability, speed, smoothness etc...). It was also seen how some gaits learnt to exploit the fitness function to find adversarial gaits.

11 Future Work

Higher dimensional gaits could be explored which would allow more complex movements. The oscillatory gait could be expanded by having joint movements defined as a sum of multiple sinusoids. This will allow Fourier approximations of functions. However, higher dimensional polices would likely increase the required number of evaluations to converge. A mechanical reset system to allow gait training to require no human intervention would make higher numbers of gait evaluations feasible. As seen in testing, the optimum robot gait very much depends on its environment. It would therefore be useful to have a robot attempt to adapt its gait as its environment changes. The robot could therefore could use sensors to develop uncertainty in the effectiveness of the current gait and adapt according. The robot could also sense the new environment is one its encountered before and recall a suitable gait.

Most of the optimisation processes discussed here aim to return the policy with the overall highest fitness. It would be more desirable if the returned policy was one with a high fitness but also some resistance to variation. This gait would be more robust. Therefore, the investigation of a optimisation process that aims to find the most robust highest scoring policy would further research in this field.

These techniques could be applied to situations where simulations are unfeasible to design or poorly reflect the real world scenario. For example, soft robotic actuators are difficult to mathematically model due to their lack of structure and flexibility [18]. This makes the creation of simulated models of these actuators impossible. Instead, techniques similar to those presented here could be applied to train a real world robot. With the advent of techniques such as additive manufacturing, robots can be rapidly prototyped and tested. The techniques here could be used not only to adapt the gait of the robot, but also to adapt the design of the robot itself. It would have to be seen if results could be obtained in a low number of evaluations to make this feasible.

Learning to optimise techniques represent optimisation as a reinforcement learning problem. These learnt optimisers have been shown to outperform Bayesian optimisation in tasks such as neural network hyperparameter tuning [19]. These learnt optimisers could be applied to gait generation tasks. It could be investigated how well optimisers trained for other expensive noisy tasks would apply to gait generation. Further, these optimisers could be trained with gait generation on a variety of different simulated robots. These could be compared with optimisers that have been trained on unrelated tasks.

12 References

[1] Boston Dynamics, Atlas Robot, Available at https://www.bostondynamics.com/atlas, 2020

[2] Ben Katz, A Low Cost Modular Actuator for Dynamic Robots, Massachusetts Institute of Technology, 2018

[3] Stuart Russell Peter Norvig, Artificial Intelligence: A Modern Approach, First Edition, 1995, pp. 5

[4] Jack Collins, Ross Brown, Jurgen Leitner and David Howard. Traversing the Reality Gap via Simulator Tuning, 2020; arXiv:2003.01369.

[5] OpenAI, Ilge Akkaya et al, Solving Rubik's Cube with a Robot Hand, 2019; arXiv:1910.07113.

[6] Antoine Cully, Jeff Clune, Danesh Tarapore and Jean-Baptiste Mouret. Robots that can adapt like animals, 2014; arXiv:1407.3501. DOI: 10.1038/nature14422.

[7] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez and Vincent Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots, 2018; arXiv:1804.10332.

[8] Pymunk Documentation, Release 5.6.0, Available at "https://buildmedia.readthedocs.org/media/pdf/pymunk/latest/pymunk.pdf"

[9] Kohl, Nate Stone, Peter, 2004, Machine Learning for Fast Quadrupedal Locomotion, 78712-1188

[10] Espressif Systems, ESP32 Datasheet, Version 3.4, Available at https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[11] Thijs Elenbaas, CMDMessenger, 2017, Available at https://github.com/thijse/Arduino-CmdMessenger

[12] Polese JC, Teixeira-Salmela LF, Nascimento LR, Faria CD, Kirkwood RN, Laurentino GC, Ada L, The effects of walking sticks on gait kinematics and kinetics with chronic stroke survivors; 10.1016/j.clinbiomech.2011.08.003

[13] Belter, Dominik Skrzypczynski, Piotr, 2010, A biologically inspired approach to feasible gait learning for a hexapod robot. Applied Mathematics and Computer Science. 20. 69-84. 10.2478/v10006-010-0005-7.

[14] Daniel Lizotte, Tao Wang, Michael Bowling and Dale Schuurmans , Univ. Alberta, Automatic Gait Optimization with Gaussian Process Regression, IJCAI-07-944

[15] Thomas Huijskens, Bayesian optimization with scikit-learn, 2016, Available at https://thuijskens.github.io/2016/12/29/bayesian-optimisation/

[16] Pedregosa et al, Scikit-learn: Machine Learning in Python, 2011,JMLR 12, pp. 2825-2830

[17] Calandra, R., Seyfarth, A., Peters, J. et al. Bayesian optimization for learning gaits under uncertainty. Ann Math Artif Intell 76, 5–23 (2016). https://doi.org/10.1007/s10472-015-9463-9

[18] Z. Gong, J. Cheng, K. Hu, T. Wang and L. Wen, "An inverse kinematics method of a soft robotic arm with three-dimensional locomotion

for underwater manipulation," 2018 IEEE International Conference on Soft Robotics (RoboSoft), Livorno, 2018, pp. 516-521, doi: 10.1109/ROBOSOFT.2018.8405378.

[19] Bergstra, James; Bardenet, Remi; Bengio, Yoshua; Kegl, Balazs, Algorithms for hyper-parameter optimization, 2011, Advances in Neural Information Processing Systems