UNIVERSITY OF BIRMINGHAM

Optimising Swinging Motion with a NAO Robot

M.Knee, H.Shaw, S.Bull, H.Gaskin, J.Morris, C.Patmore,
H.Pratten, M.Hadfield, J.Torbett, D.Corless,
H.Withers, K.Evans, E.Dixon, A.Szekely, T.Smith

Date: 24th March 2017

Abstract:
The aim of the investigation was to program a robot to effectively drive a swing. The characteristics of the NAO robot, including the range of motion and strength of the robot, were examined, as well as the properties of the physical swing. These properties included the mass distribution and damping coefficient. Numerical models were built on these values to find the optimal motion to drive the swing. These were split into two categories: swinging from predetermined functions and swinging based on position feedback. A virtual swing was constructed in Webots for which the damping constant was altered until the virtual world closely represented the real world. Swinging from predetermined functions produced a maximum amplitude of approximately 4.5° in the virtual world; the maximum amplitude was unstable and was found to oscillate. The maximum amplitude was limited by the precision of the value for the virtual swings period. No result was obtained from running this function on the real robot due to the robot's processor limitations. Swinging with position feedback was incredibly successful producing a maximum amplitude of 47° in the virtual world and $15.8 \pm 0.2°$ in the real world. The difference seen in these values was as a result of the swings ability to flex and the robot separating from the seat. This model was extended to use the robot's internal sensors to determine its position; using visual sensors the robot was able to build to an amplitude of $9.2 \pm 0.2°$ while the inertial sensors could not be used due to complications in the hip movement. The optimal human swinging motion was analysed but was to be not applicable to the NAO robot due to the different weight distribution of the NAO. Several machine learning algorithms were tested, the most promising of which were two based on neural networks. These algorithms were able to learn how to drive a dumbbell pendulum but were not able to be tested on a Webots virtual world. This was as a result of the instability of the Webots software.

School of Physics and Astronomy
University of Birmingham
Birmingham, B15 2TT

Contents

1 Introduction

Sam Bull

The Aldebaran NAO robot continues humanity's tradition of creating human-like robots to perform certain tasks in a similar manner to a human.

The first modern robots were developed during the Industrial Revolution to automate manufacturing tasks. These were mostly mechanical arms, but in 1810, Friedrich Kaufmann created the first humanoid robot in Dresden, Germany[1]; a trumpet player with bellow for lungs. Its sole purpose was to make a trumpet-like sound, and could only make sounds that it was mechanically able to. This is typical of robots of this time, which were always created for a singular purpose and not intended to develop in the future.

The field of robotics has continued to advance since then, with robots able to perform more and increasingly complex tasks and, in some cases, learning to develop their skills.

The evolution of robots is closely linked to the field of artificial intelligence. In this, artificial "agents" perceive their environments and determine an action to take that best achieves a given goal. William Grey Walter is credited with creating the first independent robot with complex behaviour in Bristol, England in 1948. He used an early idea of neural networks, modelling the connections between brain cells, to create Elmer and Elsie[2]. These three-wheeled robots were described as tortoises due to their appearance and slow movements. They could respond to both light and touch, demonstrating the interaction between two sensory systems, such that they had "behaviour" and could navigate obstacles and return "home".

Recent developments in the fields of robotics and artificial intelligence include computers learning complex human board games and humanoid robots learning to perform tasks usually only seen in humans. Google DeepMind's AlphaGo beat world-class human player Lee Sedol in 2016 in 4 out of 5 games of Go[3], an ancient Chinese game with more possible states than atoms in the universe. Because the game has so many states it is impossible for it to be won by "brute force", i.e. examining all the possibilities and selecting the best moves. Having a computer win such a complex game that often relies on feeling and intuition was a huge breakthrough for DeepMind. It used deep neural networks to mimic expert players and tested its learning by playing millions of games against itself.

Honda's ASIMO is a humanoid robot which can perform tasks such as walking, running, navigating through a room of moving obstacles, traversing flights of stairs and responding to human speech. It combines mechanical ability, such as combating the rotational forces generated by running, with artificial intelligence and processing sensory input, such as knowing where a step is or recognising human speech. In this way robots, have often been used to optimise or analyse tasks usually performed by humans. From the manufacturing robots used in the Industrial Revolution to ASIMO running and talking, humans can use robots to study their actions and find uses for them in everyday life.

Swings are a common sight in children's playgrounds nowadays, and it is known that they have been in the UK since at least 1875[4]. They are usually comprised of a wooden or plastic seat suspended from a bar, or fulcrum, by rope or metal chains. By sitting on the seat and moving the legs and torso forwards and backwards, a person can cause the swing to oscillate like a pendulum and achieve significant height changes. The G-forces associated with this motion are what makes the experience enjoyable, in the same manner as roller coasters, though less extreme.

As well as being an enjoyable outdoor recreational activity, swings are also used in the sport of Kiiking. Invented in Estonia in the 1990s by Ado Kosk, the sport involves a swing with metal bars instead of chains such that the arms remain taught. A person is strapped to the swing by the feet and must repeatedly squat and stand to pump its motion, with the goal being to swing a full 360 degrees over the fulcrum of the swing. The winner is the person who can do so with the longest swing arms, as Kosk observed that longer arms make it harder to swing over the fulcrum. He therefore created telescoping arms to allow for easily increasing the level of challenge. The current record holder in this sport is Ants Tamme with 7.03 metres, earning him a place in

the Guinness Book of World Records[5].

There are therefore many different ways of using a swing, and many different types of swing. By using an Aldebaran NAO robot, this project aims to investigate the physical mechanics of swinging and the various swings and methods of pumping them to achieve maximum oscillations.

2 Numerical Modelling

2.1 Numerical Models

Joshua Torbett

Numerical models were used in this project to investigate what the optimal swinging motion of the robot was, without the need for access to the robot. This is useful as it removes a bottleneck in this project due to having a single robot and to allows other key tasks to be performed on it. Once set up, these models were applied to the robot. The models improved the swinging motion of the robot and data from the swinging motion improved the models, bringing them closer to reality.

2.1.1 Lagrangian and Hamiltonian Mechanics

Lagrangian mechanics were used for the numerical models, which is a reformulation of Newton's Laws by Joseph Louis Lagrange that eliminate the need to calculate forces on isolated parts of a mechanical system. Even though Lagrangian mechanics introduces no new physical principles it is extremely efficient, as it only has as many equations to solve as variables [6].

Lagrangian mechanics allows for the use of convenient variables as long as they obey the constraints of the system. Also if used over Newtonian mechanics only requires a single function of these dynamic variables to describe the motion of the entire system.

This single function is called the Lagrangian, L, of the system.

$$L = T - U. \tag{2.1}$$

Where T is the kinetic energy of the system and V is the potential energy of the system.

Once this Lagrangian is found it can be used to directly obtain the differential equations of motion, without need for any vector force diagrams.

This is done with the Euler-Lagrange equations,

$$\frac{d}{dt}(\frac{dL}{d\dot{q}}) = \frac{dL}{dq}. \tag{2.2}$$

Where q denotes a generalised co-ordinate and $\dot{q}$ the time derivative of it or its generlised velocity.

However, it is important to consider that these equations of motion are for a frictionless system and extra terms will need to be considered to include factors, such as friction in the model.

2.1.1.1 Simple Plane Pendulum

Taking a simple plane pendulum with length, l, and mass, m, illustrated in figure 2.1

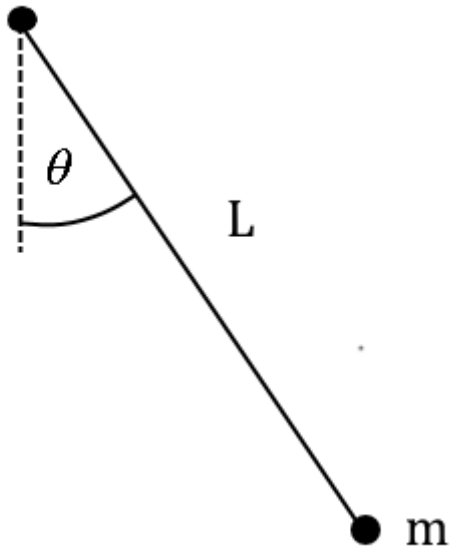


Figure 2.1: Diagram of a Simple Pendulum

To calculate the Lagrangian, L; convenient variables that obey the constraints of the system are picked, for this system the x and y distance of the pendulum bob, from the pivot, are chosen with the following constraints due to the plane pendulum,

$$x = lsin(\theta),$$
$$y = lcos(\theta).$$

Now it is necessary to find the kinetic energy, T, and potential energy, V, for the system.

$$T = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2).$$

Where $\dot{x}$ and $\dot{y}$ are the velocities in the x and y directions respectively.

$$U = -mgy.$$

This is negative because y is defined such that a decrease in y relates to an increase in potential energy.

Although it can be reformulated using the constraints, to reduce the number of dynamic variables and thus equations necessary to solve,

$$L = T - U = \frac{1}{2}ml^2\dot{\theta}^2 + mglcos(\theta). \tag{2.3}$$

With the Lagrangian now defined the Euler-Lagrange equation is used,

$$\frac{d}{dt}(\frac{dL}{d\dot{q}}) = \frac{dL}{dq},$$
$$ml^2\ddot{\theta} = -mglsin(\theta).$$

This can be rearranged to find the equation of motion for θ,

$$\ddot{\theta} = -\frac{g}{l}sin(\theta). \tag{2.4}$$

This equation can almost be used to describe the motion of the system. This equation is simple enough to solve but as there are a number of dynamic variables, the complexity of the equations of motion increase. As a result, it is necessary to decouple this equations of motion to allow them to be used and analysed.

2.1.2 Decoupling 2nd Order ODEs

The decoupling of 2nd Order ordinary differential equations, (ODEs), is necessary for the analysis of more complicated models.

To continue the example of the simple plane pendulum,

$$\ddot{\theta} = -\frac{g}{l}sin(\theta).$$

First a new variable is introduced, ω, which in this context, relates to the angular speed of the system.

$$\omega = \frac{d\theta}{dt} = \dot{\theta}.$$

This creates two 1st order ODEs,

$$\dot{\theta} = \omega$$

and

$$\dot{\omega} = -\frac{g}{l}sin(\theta).$$

Next, standard numerical methods are used to solve these ODEs and thus simulate the motion of the system.

2.2 Custom Runge-Kutta 4th Order

The reason numerical methods are powerful is that after a solution is set up for ordinary differential equations, for each model only the equations of motion will need to be changed instead of needing a specific solution for each. Also, the complexity of the equations of motion, once they are in the decoupled form, will not be a barrier in modelling the motion of that system.

The numerical method chosen was the Runge-Kutta $4th$ order method. After the use of different methods used in GNU Scientific Library Differential equation solver, it was found to be the best middle ground between accuracy and computation time. However, due to the black box nature of those solvers and the issues integrating them with the chosen programming software, a custom Runge-Kutta $4th$ order method was used.

The Runge-Kutta $4th$ order method is an extension of Euler's method, which can be considered as 1st order Runge-Kutta method. It is used to numerically integrate first order differential equations by generating slope approximations using Taylor expansions. [2]

To demonstrate the method, consider a $1st$ order ODE,

$$\frac{dy(t)}{dt} = y'(t) = f(y(t), t), with y(t_0) = y_0.$$

To find the estimate of $y(t+h)$ the weighted sum of the slopes, m, is used.

$$\begin{aligned} y'(t_0 + h) &= y(t_0) + \frac{\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4}{6} h, \\ &= y(t_0) + mh. \end{aligned} \tag{2.5}$$

Where,

$$\mathbf{k}_1 = \mathbf{f}(y(t_0), t_0).$$

The variable $\mathbf{k}_1$ is the slope at the beginning of the step.

$$\mathbf{k}_2 = \mathbf{f}(y(t_0) + \mathbf{k}_1 \frac{h}{2}, t_0 + \frac{h}{2}.$$

The variable $\mathbf{k}_2$ is the estimate of the slope at the midpoint, after using $\mathbf{k}_1$ to step half way through the step.

$$\mathbf{k}_3 = \mathbf{f}(y(t_0) + \mathbf{k}_2 \frac{h}{2}, t_0 + \frac{h}{2}.$$

The variable $\mathbf{k}_3$ is another more accurate estimate for the slope at the midpoint, found by using $\mathbf{k}_2$ to step half way through the step.

$$\mathbf{k}_4 = \mathbf{f}(y(t_0) + \mathbf{k}_3 h, t_0 + \frac{h}{2}.$$

The variable k_4 is an estimate of the slope at the endpoint, found by using $\mathbf{k}_3$ to step fully through the step.

Due to the estimations in the method, after each step there is a difference between the estimated and real values. This error is of order $O(h^5)$ and is called the local truncation error. To model a system many steps will need to be taken, this causes a total accumulated error or global truncation error of the order $O(h^4)$. However as this error is based on the step size, h, as it tends to zero, the result tends towards the true value.

2.3 Pendulum Motion

2.3.1 The Simple Pendulum

Max Hadfield

The simplest possible model of a swinging object is a point mass of mass m, suspended from a pivot by a light inextensible string of length l. This system is known as a simple pendulum and it was used as the basis for all the mathematical models which were created by the Numerical Modelling Group.

The Lagrangian function of a simple pendulum in terms of the coordinate θ is shown in Equation 2.3. Applying the Euler-Lagrange equation leads to the equation of motion for the pendulum

$$\ddot{\theta} = -\frac{g}{l} sin(\theta). \tag{2.6}$$

It can be seen from Equation 2.6 that the mass will undergo an oscillatory motion about $\theta = 0$ when released from an initial displacement. The amplitude of the oscillation is dependent on the length of the swing but not the mass of the pendulum bob.

The restoring force on the bob is given by $F = -mgsin(\theta)$. The small angle approximation states that $sin(\theta) \approx \theta$ as $\theta \to 0$. This approximation has a relative error of less than 1% for angles less than 14°. Using this, the equation for force can be expressed as $F = -mg\theta$. If s is defined as the distance of the pendulum bob from the equilibrium point measured along the arc of motion, the force expression can be written as $F = -\frac{mg}{L}s$, which is in the form of the restoring force of a simple harmonic oscillator with force constant $k = \frac{mg}{L}$.

Using the result that a simple pendulum behaves as a simple harmonic oscillator for small oscillations, an expression for the angular frequency of the pendulum can be obtained.

$$\omega_0 = \sqrt{\frac{k}{m}} = \sqrt{\frac{\left(\frac{mg}{L}\right)}{m}} = \sqrt{\frac{g}{L}} \text{ rad s}^{-1} \tag{2.7}$$

$$f_n = \frac{1}{2\pi}\sqrt{\frac{g}{L}} \text{ s}^{-1} \tag{2.8}$$

Equation 2.8 gives an expression for the frequency at which a pendulum will oscillate in the absence of a driving or damping force, known as its natural frequency. This is a useful quantity for us to consider because resonance occurs when an oscillator is driven with a frequency close to its natural frequency, causing a peak in the oscillation amplitude which can be obtained with a given magnitude of force.

2.3.2 Damped Simple Pendulum

Max Hadfield

The equation of motion of the simple pendulum given in equation 2.6 leads to a continuous oscillation. In reality, resistive forces, such as friction and air resistance, dissipate energy from the system, causing the swing amplitude to decrease over time. To account for these forces, a viscous damping term was added to the equation of motion of the system, as shown in Equation 2.10.

$$\ddot{\theta} = -\frac{g}{l} sin(\theta) - b\omega \tag{2.9}$$

where b is a property of the system known as the damping constant. Dot notation and the small angle approximation were used to express Equation 2.10 as

$$\ddot{\theta} + b\dot{\theta} + \frac{g}{l}\theta = 0. \tag{2.10}$$

This second order differential equation was solved to give

$$\theta = Ae^{-\frac{b}{2m}t} cos(\omega^{'} t + \phi), \tag{2.11}$$

where the angular frequency of the pendulum ω' was given by

$$\omega' = \sqrt{\frac{k}{m} - \frac{b^2}{4m^2}} = \sqrt{\frac{g}{L} - \frac{b^2}{4m^2}}. \tag{2.12}$$

It can be seen from equation 2.11 that the amplitude of a damped pendulum decreases exponentially with time as its energy is dissipated.

2.3.3 Calculating the Damping Constant

Max Hadfield

In order that the swing could be modelled accurately the damping coefficient was to be determined. Data were collected by the Robot Group showing the angle of the swing against time when the robot was released from an initial angle but with no motion programmed to show the natural decay of the swing motion. The times and angles of the peaks of the swing were selected and a curve of the form $\theta = Ae^{-\gamma t}$ was fit to this data using the MATLAB Curve Fitting Tool. It was found that the coefficient $\gamma = 0.00443 \pm 0.00002 s^{-1}$. This could be used in equations of motion by setting the damping constant $b = 2\gamma$. A plot showing the theoretical motion of a damped pendulum with the calculated damping constant, compared to the experimental data from the Robot Group, is shown in figure 2.2. This graph shows that the swing amplitude simulated pendulum decays at the same rate as the real swing, illustrating that this method of modelling the damping was accurate enough for use in the simulations.

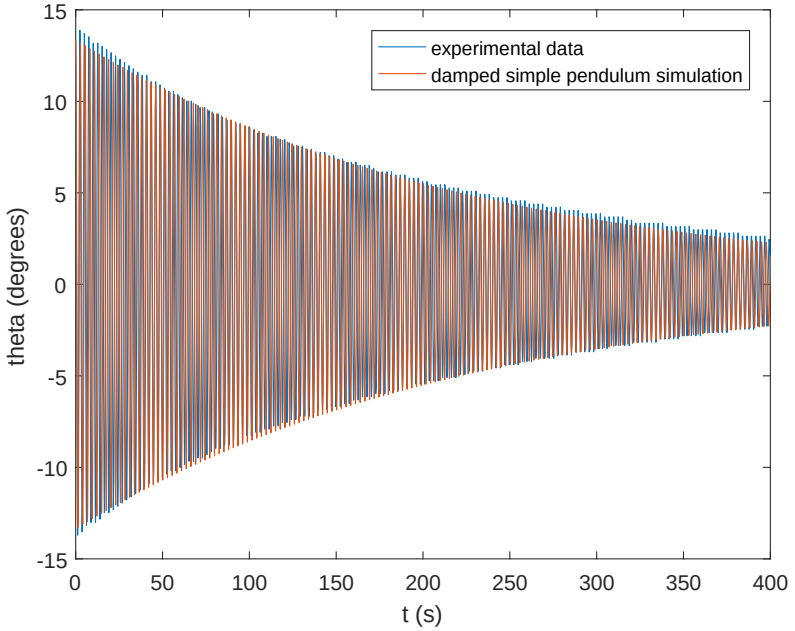


Figure 2.2: A graph showing the motion of a damped simple pendulum with damping constant b = 0.00886 s^{-1} compared to experimental data from the swing.

2.4 Principles of rotational motion

Harry Pratten

The primary principle upon which the methods of controlling rotational motion are based is the conservation of angular momentum. To illustrate how certain parameters can be varied in order to control the motion of the system, consider the flywheel pendulum represented in Figure 2.3.

Taking the pivotal point of the pendulum as the centre of the system, the total angular momentum of the system can be given by

$$\vec{L}_{tot} = \vec{L}_{orb} + \vec{L}_{spin}, \tag{2.13}$$

where $\vec{L}_{spin}$ is the angular momentum resulting from the rotation of the flywheel about its own axis, and $\vec{L}_{orb}$ is the angular momentum resulting from the orbit of the flywheel about the pivotal point of the pendulum.

In terms of variables defined in Figure 2.3, this translates to

$$\vec{L}_{tot} = \frac{1}{2}MR^2\dot{\beta} + Mr^2\dot{\alpha}. \tag{2.14}$$

Consider the system in the absence of gravity, and suppose that its initial state is at rest with respect to the lab frame.

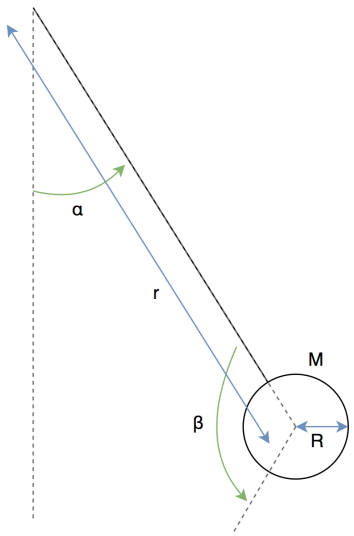


Figure 2.3: Representation of flywheel pendulum system.

If r is held constant, and the flywheel is rotated from within the system, then the resulting final state can be expressed by

$$(\dot{\beta}_i - \dot{\beta}_f) = \frac{2r^2}{R^2}(\dot{\alpha_f} - \dot{\alpha_i}), \tag{2.15}$$

hence the change in angular velocity of the flywheel about the central pivot is governed by the change in its spin by the relation

$$\Delta\dot{\alpha} = -\Delta\dot{\beta}\frac{R^2}{2r^2}. \tag{2.16}$$

This shows that a change in the intrinsic spin of the flywheel is linearly related to the rate of its orbit. However if the spin is varied, the change in angular velocity of the flywheel will act in a direction to effectively counteract this, hence conserving overall angular momentum of the system.

In this sense, the change in rotational velocity of the flywheel can be seen to induce an orbital torque on the system of

$$\tau = I\frac{\Delta\dot{\alpha}}{\Delta t} = I\frac{-\Delta\dot{\beta}}{\Delta t}\frac{r^2}{2R^2}, \tag{2.17}$$

where I is the moment of inertia of the system.

Now consider the case where the system initially has some energy such that $\dot{\alpha} = const \neq 0$ and $\ddot{\alpha} = 0$.

If r is now varied, the resulting change in $\dot{\alpha}$ can be given by

$$\dot{\alpha_f} = \frac{r_i^2}{r_f^2}\dot{\alpha_i}. \tag{2.18}$$

It can also be shown that this variation in r has no effect on $\dot{\beta}$:

$$(\dot{\beta_i} - \dot{\beta_f}) = \frac{2}{R^2}(r_f^2\dot{\alpha_f} - r_i^2\dot{\alpha_i}) = \frac{2}{R^2}(r_f^2\frac{r_i^2}{r_f^2}\dot{\alpha_i} - r_i^2\dot{\alpha_i}) = \frac{2}{R^2}(0) = 0. \tag{2.19}$$

Hence $\Delta\dot{\beta} = 0$ for any change of r.

In the presence of gravity, the motion of the system becomes more complex due to the constant downwards force, so Lagrangian analysis can be used in order to efficiently quantify the motion of the system.

2.5 Building amplitude

Harry Pratten

2.5.1 Representation and Lagrangians of basic theories

The ways of controlling the motion of a rotating system due to a) intrinsic spin about its centre of mass, and b) variation of the position of the centre of mass, can be represented in two simple systems: the dumbbell-pendulum and a simple pendulum of variable length.

2.5.1.1 Rotation model

In order to investigate the effects of rotation of a body about a centre of mass at a distance from the central pivot, consider a system comprised of a dumbbell attached at the end of a pendulum. This is illustrated in Figure 2.4.

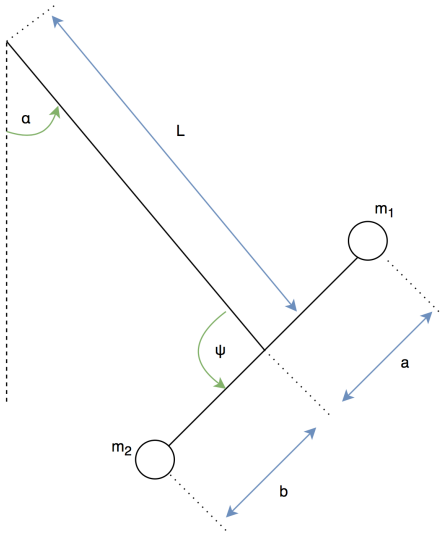


Figure 2.4: Representation of dumbbell pendulum system for rotation variation

where L is the length of the pendulum; a and b are the distances of masses m_1 and m_2 from the end of the pendulum respectively; ψ is the angle of the dumbbell relative to the pendulum; and α is the displacement of the pendulum with respect to the vertical.

The Lagrangian for this system[7] can be given by

$$\begin{aligned}\mathcal{L} &= \frac{m_1}{2}[\dot{\alpha}^2L^2 - 2\dot{\alpha}aL(\dot{\alpha}+\dot{\psi})\cos\psi + a^2(\dot{\alpha}+\dot{\psi})^2] \\ &+ \frac{m_2}{2}[\dot{\alpha}^2L^2 + 2\dot{\alpha}bL(\dot{\alpha}+\dot{\psi})\cos\psi + b^2(\dot{\alpha}+\dot{\psi})^2] \\ &+ g[L(m_1+m_2)(\cos\alpha - 1) + (m_2b - m_1a)(\cos\alpha + \psi)].\end{aligned} \tag{2.20}$$

Now, for simplicity, suppose $m_1 = m_2 = m$ and $a = b$. Then after applying the Euler-Lagrange equation, an equation of motion can be obtained[7], given by

$$\ddot{\alpha} + \frac{a^2}{a^2+L^2}\ddot{\psi} + w_0^2\sin\alpha = 0, \tag{2.21}$$

where

$$w_0 = \sqrt{\frac{gL}{a^2+L^2}}. \tag{2.22}$$

Note the only relation to ϕ in the equation of motion is the dependency upon $\ddot{\phi}$, thus illustrating that it is only the acceleration of the rotation that affects the torque the pendulum feels about the central pivot.

The equation of motion can then be decoupled in order to numerically evaluate.

$$\dot{\alpha} = -\Phi; \tag{2.23}$$

$$\dot{\Phi} = \frac{a^2}{a^2 + L^2}\ddot{\psi} + w_0^2 \sin\alpha. \tag{2.24}$$

2.5.1.2 Length Variation Model

To investigate the result of varying the centre of mass of a pendulum as it oscillates, a simple pendulum was modelled with the ability to vary its length, illustrated in Figure 2.5.

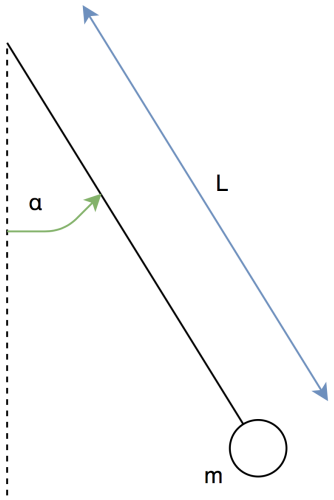


Figure 2.5: Representation of simple pendulum system for centre of mass variation.

The Lagrangian for this system illustrated in Figure 2.5 can then be given by

$$\mathcal{L} = m[\frac{L^2\dot{\alpha}^2}{2} + gL\cos\alpha + \frac{\dot{L}^2}{2}]. \tag{2.25}$$

Further analysis by applying the Euler-Lagrange equation yields the equation of motion

$$\ddot{\alpha} + 2\frac{\dot{L}}{L}\dot{\alpha} + \frac{g}{L}\sin\alpha, \tag{2.26}$$

which can then be decoupled into the form:

$$\dot{\alpha} = -\Phi; \tag{2.27}$$

$$\dot{\Phi} = 2\frac{\dot{L}}{L}\dot{\alpha} + \frac{g}{L}\sin\alpha. \tag{2.28}$$

2.5.2 Results and analysis

2.5.2.1 Resonance

In order to examine the resonant frequency of each system, the position of the dumbbell relative to the pendulum, and the length of the simple pendulum were varied sinusoidally in the rotation and varied length models respectively such that

$$L = L_0 + A\sin(wt), \tag{2.29}$$

and

$$\Psi = B\sin(wt), \tag{2.30}$$

where A and B are arbitrary constants.

For each simulation, both systems were initially slightly displaced from equilibrium. Figure 2.6 illustrates the relative growth in amplitude at frequencies as a function of natural frequency.

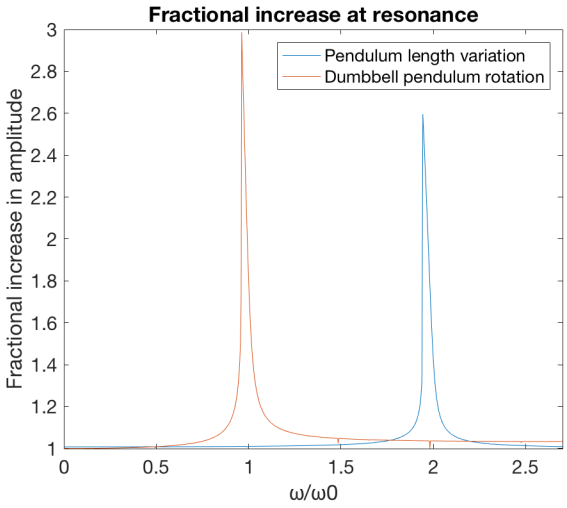


Figure 2.6: Resonance for simple pendulum and dumbbell pendulum.

First consider the simple pendulum of driven length. It can be seen that the length must be varied at a frequency of twice the natural frequency in order to be driven to a maximum amplitude. This translates to decreasing the length upon moving away from equilibrium, and increasing the length upon moving towards equilibrium.

In this sense, as the pendulum moves away from equilibrium, work is done against gravity, hence the system gains this energy. The angular velocity of the system also increases due to the nature of conservation of angular momentum, this results in moving to a greater extrema. Upon moving towards equilibrium, increasing the length may decrease the potential energy and decrease the angular velocity, but the torque from gravity acts to accelerate the pendulum in its desired direction of motion. The energy changes associated with this process are illustrated in Figure 2.7. It follows that for the best driven motion, the length should be decreased

instantaneously over equilibrium and increased instantaneously at maximum amplitude.

For the dumbbell model, it can be seen that the optimum frequency at which to drive the system is the natural frequency. This is due to the torque induced on the system being attributable to the acceleration of the rotation of the dumbbell. Hence if the motion of the dumbbell is of a phase difference of greater than $\pi/2$ and less than $3\pi/4$ with the motion of the pendulum, the torque induced on the system will always act in the direction of motion of the pendulum. In a sense the torque from the motion of the dumbbell superposes with the torque from gravity, driving the system to greater amplitudes. Hence this driving force acts to increase the maximum amplitude.

2.5.2.2 Energy and maintaining amplitude

The dilemma with driving the system perfectly periodically arises as eventually the system will collapse back down to a minimum amplitude, as the driving force falls into/out of phase with the motion of the system, in a process known as parametric resonance. It will then fall back into/out of phase with the system and build back up to a maximum as the process continuously repeats. This is illustrated in Figure 2.7 in an energy plot of the length variation model at resonance.

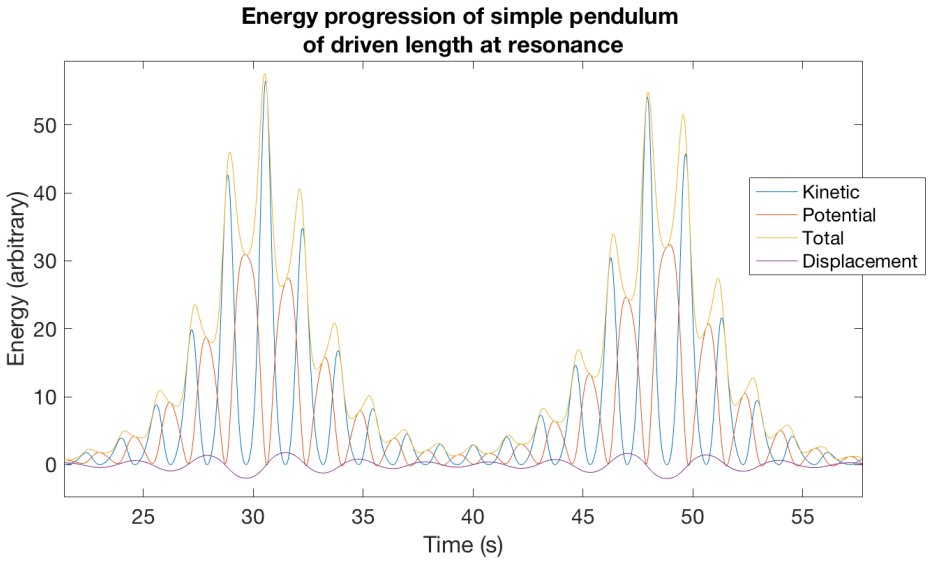


Figure 2.7: Energy of simple pendulum of driven length at resonance

This collapse in amplitude is due to the fact that as the system is driven, the time period of its oscillations is slightly affected (this will be examined in greater deal later on in relation to a more advanced model - the Double Flail), and at high speeds the system begins to behave as a non-linear oscillator.

There are several ways to prevent this collapse of the system whether it be by commencing driven motion as the system reaches a specific amplitude relative to its previous maximum; or by dynamically recalculating the frequency (e.g. taking an average over the previous 3 cycles) of the system and seeding the driving force based on a specific position in the cycle i.e. 'reset' the driving force as a function of the newly calculated frequency each time the system passes through equilibrium, and define the driven motion to start after a proportion of the time period has passed.

Dynamically recalculating the frequency would be preferred in real life as due to damping and a lack of a perfect system, the displacement may not reach a predicted amplitude where the driven force is specified to begin. Frequency however is independent of maximum displacement so dynamically recalculating the time period of an oscillation is a more reliable method of preventing a collapse of maximum amplitude in real life.

2.6 Analytical Models

Daniel Corless

It was discussed earlier in this chapter how certain movements of a mass can be made in order to drive a swing to large amplitudes. This can be done by the mass transferring energy to the system by doing work. The law of conservation of angular momentum plays a key role in the process. The main methods discussed involve upwards and downwards movements of the mass along the length of the swing, and rotation of the mass. An important thing to consider was how these movements could be applied to a robot. Possible ways that these could be applied to gain swinging amplitude were that the robot stands and crouches at the lowest and highest points of the swing respectively, and the robot rotates its upper body and its legs at the highest points of the swing. It was then investigated how this motion of the robot can be modeled as accurately as possible using an analytical model consisting of a system of masses.

The main benefit of such a model is that simulations could be run, and the model could be optimized to determine the motion of the robot required to achieve the highest possible swinging amplitude. The standing and crouching motion could easily be modeled by a mass moving up and down along the length of the swing, whereas the rotating motion could be modeled by having masses that rotate about the seat of the swing. Since the project is focusing on the robot sitting down on the swing, only the rotating motion was considered in the models.

2.6.1 Hinged Single Flail

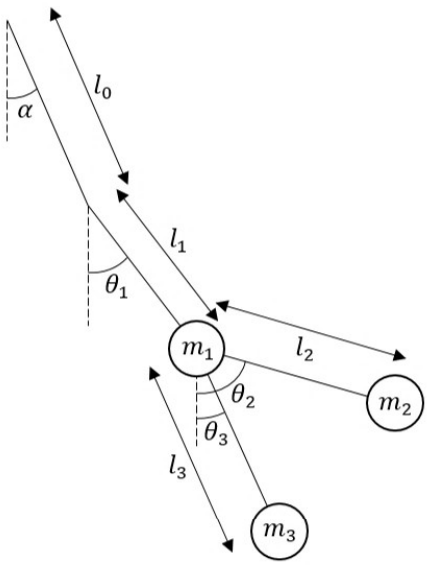


Figure 2.8: Diagram of the Hinged Single Flail model.

In the report of the previous year, the model considered to be the most accurate representation of the robot and swing was the hinged flail model [7], which is shown in Figure 2.8. This model consisted of a mass m_1 that represented the mass of the swing and the rigid part of the robot, and two other masses m_2 and m_3, which represented the upper body and legs of the robot respectively, were allowed to rotate about the seat of the swing. A pivot point was added between the top and the seat of the swing to model the hinge. The total mass $M = m_1 + m_2 + m_3$ was used to simplify mathematical expressions. One problem with this model was that m_1 was not in the position of the center of mass of the swing and the rigid part of the robot, which would realistically be at a higher position, and this is something that was taken into account in future models.

The Lagrangian of this system can be written in terms of generalized coordinates α, θ_1, θ_2 and θ_3, which are the angles from the vertical of the swing below the main pivot point, the swing below the hinge, and the two parts at the end of the flail respectively. The Lagrangian can be found to be

$$\begin{aligned}\mathcal{L} = \frac{M}{2}\big[l_0^2\dot{\alpha}^2 + l_1^2\dot{\theta}_1^2 + 2l_0l_1\dot{\alpha}\dot{\theta}_1\cos(\alpha-\theta_1)\big] + \frac{m_2}{2}\big[l_2^2\dot{\theta}_2^2 + 2l_0l_2\dot{\alpha}\dot{\theta}_2\cos(\alpha-\theta_2)\\ +2l_1l_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\big] + \frac{m_3}{2}\big[l_3^2\dot{\theta}_3^2 + 2l_0l_3\dot{\alpha}\dot{\theta}_3\cos(\alpha-\theta_3) + 2l_1l_3\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)\big]\\ +Mg\big[l_0\cos(\alpha) + l_1\cos(\theta_1)\big] + m_2gl_2\cos(\theta_2) + m_3gl_3\cos(\theta_3),\end{aligned} \tag{2.31}$$

where l_0 is the distance between the main pivot of the swing and the hinge, l_1 is the distance between the hinge and the seat (and m_1), and l_2 and l_3 are the distances from the seat to m_2 and from the seat to m_3 respectively [7]. Applying the Euler-Lagrange equation to this Lagrangian allows the equation of motion for α to be found.

$$\begin{aligned}Ml_0\ddot{\alpha} = -Ml_1\big[\ddot{\theta}_1\cos(\alpha-\theta_1) + \dot{\theta}_1^2\sin(\alpha-\theta_1)\big] - m_2l_2\big[\ddot{\theta}_2\cos(\alpha-\theta_2)\\ +\dot{\theta}_2^2\sin(\alpha-\theta_2)\big] - m_3l_3\big[\ddot{\theta}_3\cos(\alpha-\theta_3) + \dot{\theta}_3^2\sin(\alpha-\theta_3)\big] - Mg\sin(\alpha)\end{aligned} \tag{2.32}$$

If the model was to be investigated without the hinge, then the simplifications $\theta_1 = \alpha$ and $l_0 + l_1 = r$ can be made. This resulted in the equation of motion

$$\begin{aligned}Mr\ddot{\alpha} = -m_2l_2\big[\ddot{\theta}_2\cos(\alpha-\theta_2) + \dot{\theta}_2^2\sin(\alpha-\theta_2)\big]\\ -m_3l_3\big[\ddot{\theta}_3\cos(\alpha-\theta_3) + \dot{\theta}_3^2\sin(\alpha-\theta_3)\big] - Mg\sin(\alpha),\end{aligned} \tag{2.33}$$

which was used to simulate the motion of this model without a hinge. Figure 2.9 shows the variation of α with time from a simulation, with a damping term and a damping constant of 0.00443 s^{-1} added to the equation of motion. As a consequence, the swing amplitude slowly builds up to a maximum value of 27°. This demonstrates that the model is effective in driving a swing, and is therefore a good basis for creating more advanced models.

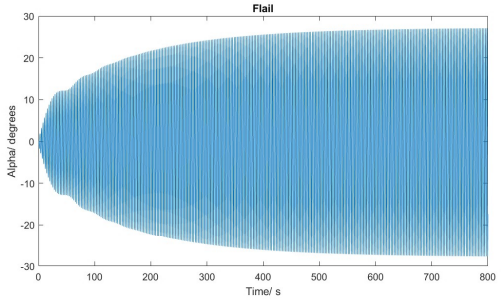


Figure 2.9: A graph showing the variation of alpha over time for the Single Flail model.

2.6.1.1 Driving the Single Flail with a Sinusoidal Motion

Max Hadfield

Using the unhinged Simple Flail model described above, the properties and effectiveness of different movement patterns were investigated. The values of l_2, l_3, θ_2, θ_3, m_1, m_2 and m_3 were set to approximate the mass distribution of the robot, based on data from the NAO H25 specification [8]. m_1 represented the swing and the robot's thigh, m_2 represented its torso, head and arms, and m_3 represented its tibia and feet. All movements which could be made by the Robot involved the rotation of one of its joints between two angles, therefore two different methods of varying the joint angle were investigated.

The first movement pattern tested was a sinusoidal motion between a maximum and minimum angle. The values $\theta_2 = 225°$ and $\theta_3 = 45°$ were set and both joint were made to oscillate around these equilibrium points with an amplitude of 15°. This motion was simulated using a range of angular frequencies and the maximum swing angle achieved each time was recorded.

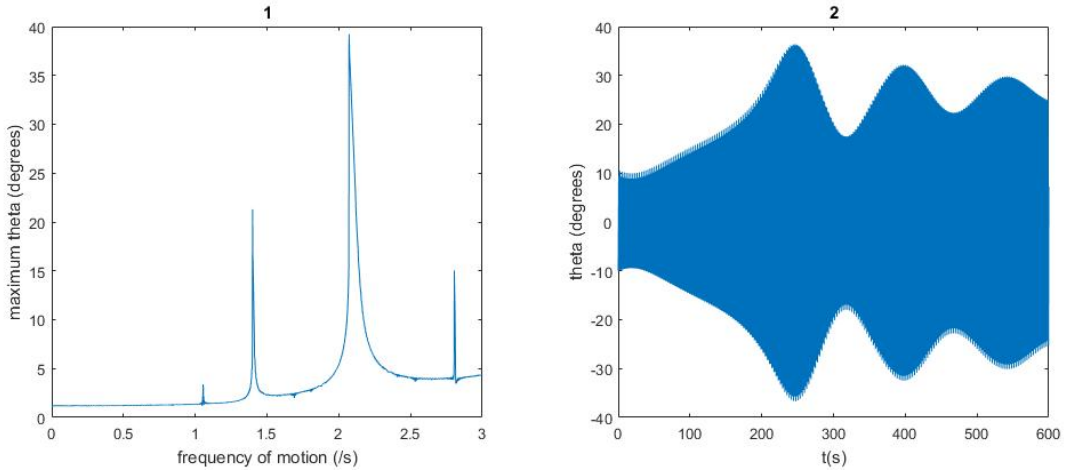


Figure 2.10: Plots showing the maximum swing angle achieved with a sinusoidal motion for a range of motion frequencies, and an example of the evolution of theta over time when the model is driven with a sinusoidal motion at the optimum frequency.

Subplot 1 in Figure 2.10 shows the maximum swing angle reached with the sinusoidal driving motion over a range of frequencies. It can be seen that there were four peaks in the maximum swing amplitude. The frequencies at which these peaks occurred were $f_1 = 1.1\ s^{-1}$, $f_2 = 1.4\ s^{-1}$, $f_3 = 2.1\ s^{-1}$ and $f_4 = 2.8\ s^{-1}$

(numbered from left to right). Calculating the natural frequency of a simple pendulum of length 1.815m using Equation 2.8 gives $f_n = 0.37$. f_2, f_3, f_4 are all approximately even multiples of f_n indicating that these peaks are due to resonance effects. Figure 2.6 suggests that resonance at even multiples of the natural frequency is due to the varying the position of the centre of mass so that is believed to be the cause here. f_1 is approximately an odd multiple of f_n, suggesting that this peak was a resonance caused by the change in angular momentum caused by the movement.

Subplot 2 in Figure 2.10 shows the evolution of the swing angle over a 10 minute period when driven at the optimum frequency. It can be seen that the amplitude takes 245 seconds to build up to its maximum amplitude. This then dies down and builds back up again due to the effects of the driving motion falling in and out of phase with the motion of the swing, discussed above in "Energy and Maintaining Amplitude".

An investigation was also carried out into the effect of changing the range of angle through which θ_2 and θ_3 were able to move. This was tested at multiple frequencies both higher and lower than the optimum frequency. The results obtained are shown in Figure 2.11.

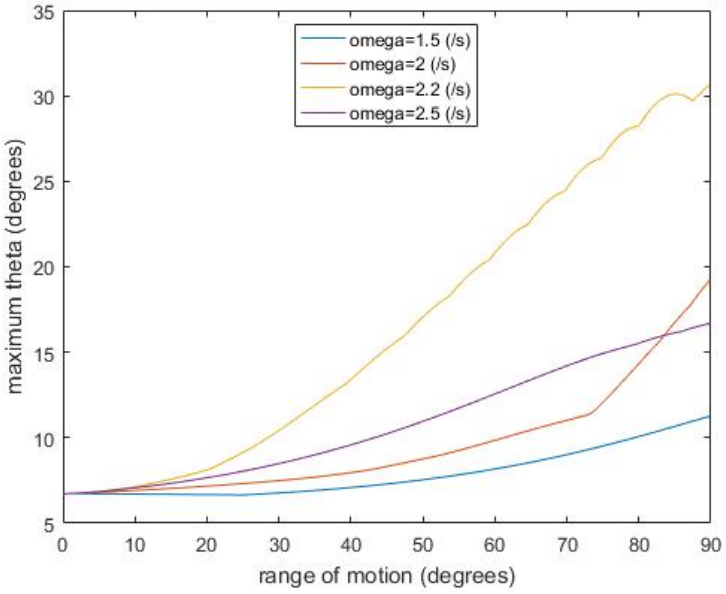


Figure 2.11: A graph showing the variation in maximum swing amplitude as the range of motion of the sinusoidal was varied at multiple angular frequencies.

The results shown in Figure 2.11 show that, for all of the angular frequencies tested, increasing the range of angles through which the joints could move increased the maximum angle the robot could swing to. This is because a larger range of motion increases the amount that the position of the centre of mass can be changed, allowing a larger torque to be generated. Also if the limb is moving with a given frequency, increasing the range of motion increases the speed that the limb travels at, providing an increase in torque. This provides evidence that all movements should be carried out across the largest angle range possible.

2.6.1.2 Calibrating a Step Function To The Real Robot's Movement

Max Hadfield

The second movement pattern used to drive the Single Flail model was based on a square wave. This motion was believed to be advantageous to the sinusoidal motion for multiple reasons: the smoothness with which the

robot was able to move would not significantly affect the motion, it was considered to be more similar to the motion with which a human would drive a swing and it has been shown in literature to be more effective [9].

A true square wave with step-like motion would have infinite velocity and acceleration when it moved between angles, this would not be achievable by the robot and would cause the simulations to produce unrealistic results, therefore the step-like motions which were used in simulations by the Numerical Modelling group were set up to replicate the motion that occurred when the robot moved its limbs between their maximum and minimum angle at its maximum safe speed. Data showing this were collected by the robot group and used to calibrate the step-like functions used.

The step-like movement pattern that was used to drive the Single Flail model was created by defining a constant value for the angular acceleration that could be produced by the motor, and a maximum angular velocity that the joints could move at. The angular acceleration was set so that the robot could accelerate to its maximum angular velocity from rest in 0.25 seconds. The maximum angular velocity was set to 3 radians per second as this was found to agree with the experimental data. The calibrated step function was set to produce a motion centred at -43° with a range of 14.3° and a time period of 3 seconds to replicate the motion performed by the robot in the Robot Group's experimental data. Figure 2.12 shows plots comparing the angle and velocity output by the step function to the experimental data. It can be seen that this method of recreating the robot's motion provided a close approximation to its real movement.

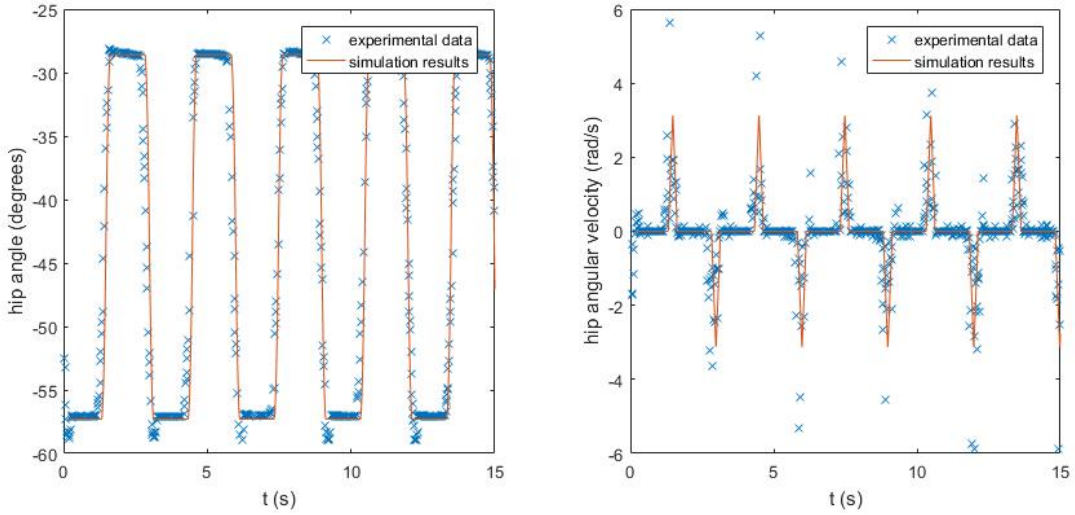


Figure 2.12: Plots showing the angle and velocity produced by the calibrated step function compared to experimental data.

2.6.1.3 Driving the Single Flail with a Step Function

Max Hadfield

The step function described above was used to drive the Single Flail model. As before the motion was simulated across a range of frequencies and the maximum swing angle achieved was recorded.

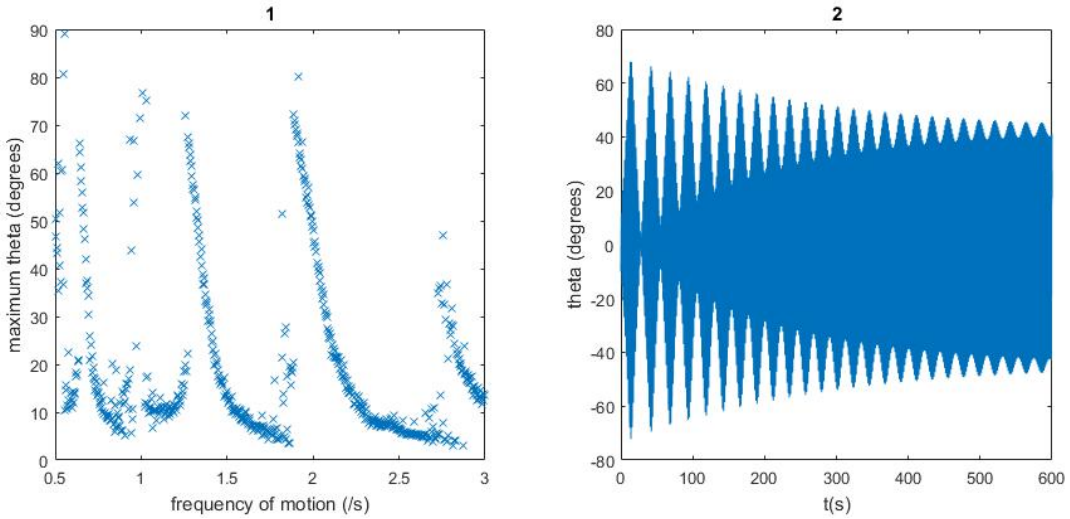


Figure 2.13: Plots showing the maximum swing angle achieved with the realistic step function motion for a range of motion frequencies, and an example of the evolution of theta over time when the model is driven with this motion at frequency is 2 s^{-1}.

Subplot 1 in Figure 2.13 shows the maximum swing angle reached with the realistic step function over a range of frequencies. The results produced showed a higher degree of variance than with the sinusoidal motion, this was believed to be due to the pendulum responding chaotically to the motion. It can be seen that there were five peaks in the maximum swing amplitude with approximate frequencies of 0.7 s^{-1}, 1 s^{-1}, 1.4 s^{-1}, 2.0 s^{-1}, and 2.8 s^{-1}. These frequencies can be seen to also correspond to multiples of the natural frequency, $f_n = 0.37$, and it can be seen that four of the peaks are at approximately the same frequency as the resonance peaks with the sinusoidal motion, suggesting they were caused by the same phenomenon. It was concluded that the other peak at 0.7 s^{-1} was also due to resonance but was too small to show up in Figure 2.10.

Subplot 2 shows the evolution of the swing angle over time when the simulated robot performed the step function motion at a frequency of 2 s^{-1}. It can be seen that the swing rose to its maximum angle much quicker than with the sinusoidal motion, taking only 14 seconds to reach its peak. The effects of parametric resonance are also seen as the swing amplitude rises and falls repeatedly over time.

It can be seen from Figure 2.13 that there were multiple movement frequencies at which the step function motion produced a higher swing angle than the sinusoidal motion's maximum swing of 36°. Therefore from these results it can be concluded that a periodic motion with sharp changes between the minimum and maximum angle would produce the greatest swing angle. It can also be seen from figures 2.10 and 2.13 that the frequency of the driving motion was a crucial factor in achieving a large swing amplitude, so motions programmed on the robot should be carried out at multiples of the natural frequency of the swing to take advantage of resonance.

2.6.2 Double Flail

Daniel Corless

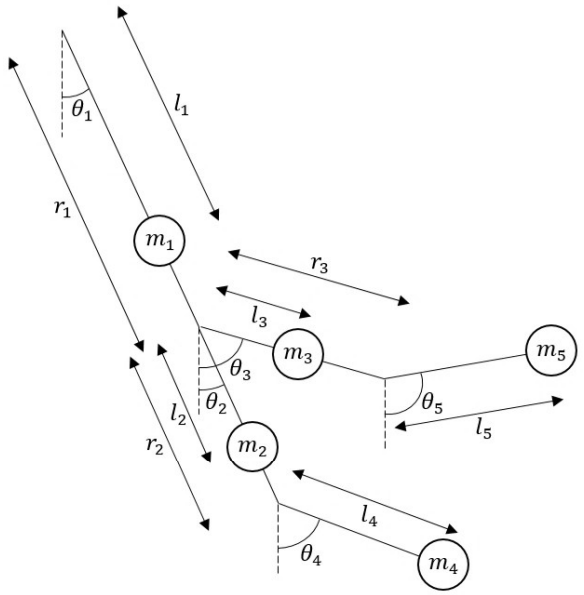


Figure 2.14: Diagram of the Double Flail model.

This model built on the flail model by introducing two extra masses to more accurately represent the mass distribution of the robot as shown in Figure 2.14. The model also allowed the masses to be positioned at the center of mass of the real masses that they represented, which is an improvement on the flail model. Like before, m_1 represented the mass of the swing and the rigid part of the robot, but m_2 and m_3 represented the upper leg and the torso of the robot and m_4 and m_5 represented the lower leg and the head. Thus, these limbs of the robot could be more accurately modelled. This model does not have a hinge, so it was compared with the flail model without a hinge.

The Lagrangian of this system can be written in terms of generalized coordinates θ_1, θ_2, θ_3, θ_4 and θ_5, which are the angles from the vertical of the swing below the main pivot point, the two parts of the flail connected to the seat and the two parts at the end of the flail respectively. Derivations of all equations for this model can be found in Appendix A.1. Using the standard method, the Lagrangian of the system is found to be

$$\begin{aligned}\mathcal{L} = \frac{m_1}{2} l_1^2 \dot{\theta}_1^2 + \frac{(m_2+m_3+m_4+m_5)}{2} r_1^2 \dot{\theta}_1^2 + \frac{m_2}{2}\left[l_2^2\dot{\theta}_2^2 + 2r_1 l_2 \dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\right] \\ + \frac{m_3}{2}\left[l_3^2\dot{\theta}_3^2 + 2r_1 l_3 \dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)\right] + \frac{m_4}{2}\left[r_2^2\dot{\theta}_2^2 + l_4^2\dot{\theta}_4^2 + 2r_1 r_2 \dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\right. \\ \left. + 2r_1 l_4 \dot{\theta}_1\dot{\theta}_4\cos(\theta_1-\theta_4) + 2r_2 l_4 \dot{\theta}_2\dot{\theta}_4\cos(\theta_2-\theta_4)\right] + \frac{m_5}{2}\left[r_3^2\dot{\theta}_3^2 + l_5^2\dot{\theta}_5^2\right. \\ \left. + 2r_1 r_3 \dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3) + 2r_1 l_5 \dot{\theta}_1\dot{\theta}_5\cos(\theta_1-\theta_5) + 2r_3 l_5 \dot{\theta}_3\dot{\theta}_5\cos(\theta_3-\theta_5)\right] \\ + m_1 g l_1\cos(\theta_1) + (m_2+m_3+m_4+m_5) g r_1\cos(\theta_1) + (m_2 l_2 + m_4 r_2) g\cos(\theta_2) \\ + (m_3 l_3 + m_5 r_3) g\cos(\theta_3) + m_4 g l_4\cos(\theta_4) + m_5 g l_5\cos(\theta_5),\end{aligned} \tag{2.34}$$

where l_1 and r_1 are the distances from the main pivot of the swing to m_1 and the seat respectively, and l_2 and l_3 are the distances from the seat to m_2 and from the seat to m_3 respectively, r_2 and r_3 are the lengths of

the upper leg and the torso of the robot respectively, and l_4 and l_5 are the length of the lower leg and height of the head respectively. Applying the Euler-Lagrange equation then gives the equation of motion for θ_1 to be found.

$$\begin{aligned}\left[m_1 l_1^2 + (m_2 + m_3 + m_4 + m_5) r_1^2\right] \ddot{\theta}_1 = -m_2 r_1 l_2 \left[\ddot{\theta}_2 \cos(\theta_1 - \theta_2) + \dot{\theta}_2^2 \sin(\theta_1 - \theta_2)\right] \\ -m_3 r_1 l_3 \left[\ddot{\theta}_3 \cos(\theta_1 - \theta_3) + \dot{\theta}_3^2 \sin(\theta_1 - \theta_3)\right] - m_4 r_1 \left[r_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) + r_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2)\right. \\ \left. + l_4 \ddot{\theta}_4 \cos(\theta_1 - \theta_4) + l_4 \dot{\theta}_4^2 \sin(\theta_1 - \theta_4)\right] - m_5 r_1 \left[r_3 \ddot{\theta}_3 \cos(\theta_1 - \theta_3) + r_3 \dot{\theta}_3^2 \sin(\theta_1 - \theta_3)\right. \\ \left. + l_5 \ddot{\theta}_5 \cos(\theta_1 - \theta_5) + l_5 \dot{\theta}_5^2 \sin(\theta_1 - \theta_5)\right] - \left[m_1 g l_1 + (m_2 + m_3 + m_4 + m_5) g r_1\right] \sin(\theta_1)\end{aligned} \tag{2.35}$$

Joshua Torbett

The model was improved by using parameters from the robot, to make it more representative of the form and motion of the robot.

Table 2.1: Robot parameters in the context of the double flail

Mass	(kg)	Length	(m)
m_1	0.4207	l_1	1.815
m_2	2.2068	l_2	0.2115
m_3	0.7794	l_3	0.1000
m_4	0.6838	l_4	0.1144
m_5	1.2148	l_5	0.1481

The r values are based on the l values,

$$r_1 = l_1;$$
$$r_2 = l_2/2;$$
$$r_3 = l_3/2;$$
$$r_4 = l_4/2;$$
$$r_5 = l_5/2.$$

The robot parameters also place constraints on how the various angles change,

$$\theta_2 = \theta - 2.319 - \theta_{2max} Z,$$

where $\theta_{2max} = 0.236$ radians.

$$\theta_3 = \theta + 1.570 - \theta_{3max} Z,$$

where $\theta_{3max} = 0.000$ radians.

$$\theta_4 = \theta_2 - 0.0785 - \theta_{4max} Z,$$

where $\theta_{2max} = 0.593$ radians.

$$\theta_5 = \theta_3 - 0.733 - \theta_{5max} Z,$$

where $\theta_{2max} = 0.820$ radians.

Where these values are in radians and Z is a dimensionless factor that varies between -1 and 1 to give the full ranges of possible angles.

With these parameters this becomes a common orientation of the masses for representing the robot.

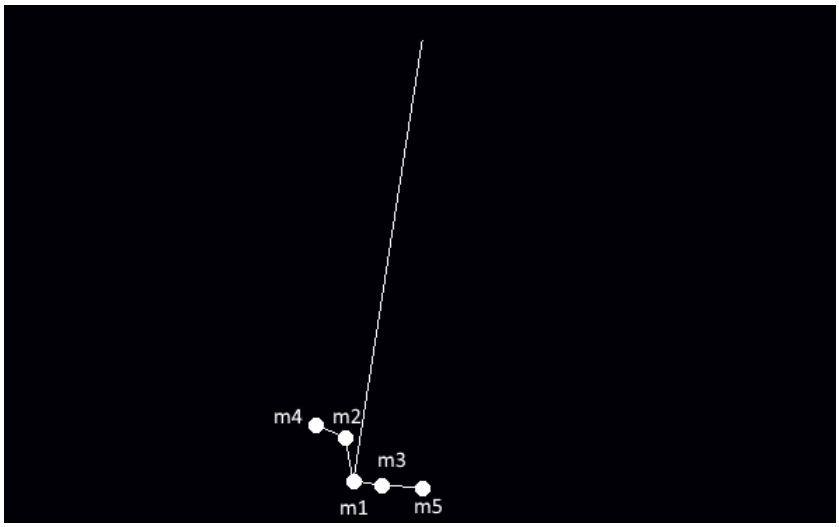


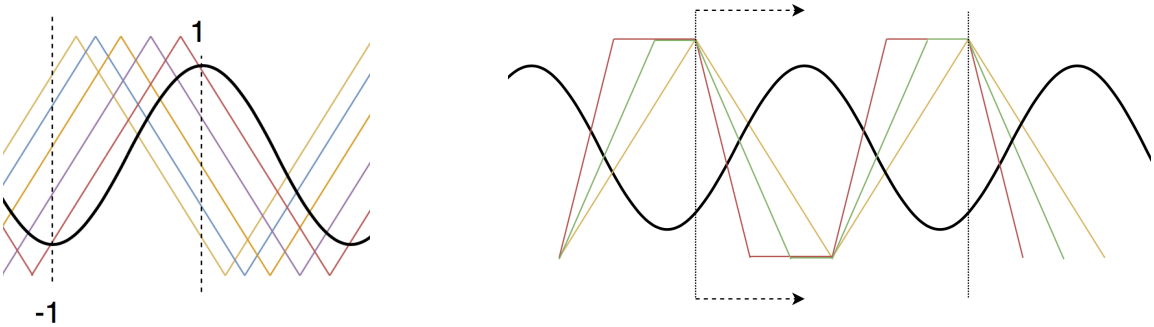
Figure 2.15: Double Flail model simulation, see 'Graphical Simulations of Models'. Here the double flail is rotated to one extreme, i.e. $Z = -1$

Harry Pratten

2.6.3 Results and analysis

2.6.3.1 Variation of generated torque at different points within the cycle

In order to examine both the best place in the cycle to move the robot, and how to move the robot, the term Z controlling the motion of the Double Flail was varied at different rates between the two extrema. This is illustrated in Figure 2.16.



(a) Varied position of sawtooth movement within cycle.

(b) Variation of applied torque at different points within the cycle.

Figure 2.16: Rotational movement of the double flail throughout an oscillation. The sinusoidal wave is given as the displacement of the swing over time, and a single sawtooth wave represents the angular position of the double flail between its two extremes.

The point in the swings cycle at which the double flail started to move was varied from *-(the positive maximum amplitude from the previous oscillation)* up to *+(positive maximum amplitude from previous oscillation)*. Intuitively, a starting point of -0.9 corresponds to commencing movement of the double flail just after the system has reached maximum amplitude; conversely a starting point of 0.9 corresponds to commencing movement just before the system reaches a maximum amplitude. This is illustrated by the number of sawtooth waves which can be seen to move through the cycle in Figure 2.16a, each one representing the motion of the double flail as it commences at different points within the cycle.

After one possible movement of the double flail has been iterated through the whole cycle, the speed of rotation was increased, represented in Figure 2.16b, again where the coloured lines represent the angular position of the double flail throughout the cycle due to varied speeds of the rotation. The reason that the movement of the double flail appears to stop is due to the limit on its rotation as it has been constrained by parameters as similar as possible to that of the robot in real life.

Effectively iterating through all possible speeds at which the real life robot could move has been completed, before iterating the point in an oscillation at which these movements are begun through half an oscillation (the half which would act to increase amplitude). The maximum amplitude reached after 21 oscillations for each case was recorded. The results of this are shown in Figure 2.17.

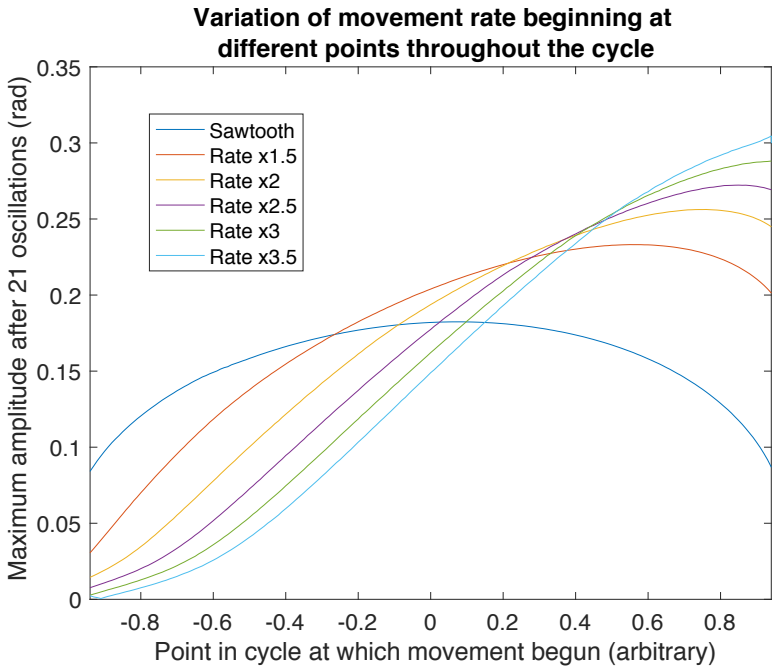


Figure 2.17: Maximum amplitude with different movement rates throughout an oscillation.

Figure 2.17 shows that although it is more efficient to apply torque to the system at equilibrium - when the movement of the double flail is essentially of a phase difference of $\pi/2$ with the displacement of the system; it is more effective to vary the rotation of the double flail at a faster rate closer to the maximum amplitude. In this sense an impulse of torque is felt just before reaching an extreme, and another impulse of torque is felt just after reaching the extreme, both acting in the direction of motion of the system.

It can be seen that depending on the point in the cycle at which the double flail begins to move, it is better to move at different rates. This is due to the point within the cycle at which the movement finishes, and hence another impulse of torque acts on the system, but acts in the opposite direction. So the optimum movement will have the greatest speed of rotation of the double flail, with the movement minimising the time after passing through equilibrium at which the double flail begins its motion and the time before reaching equilibrium the motion ceases. The optimum motion for each rate of movement corresponds to finding the best balance between these factors.

2.6.3.2 Variation of period with varied torque

As discussed previously, the primary reason a system such as this is unable to be driven perfectly periodically with respect to time, is due to the fact that a driving force causes distortion in the time period of an oscillation. The effect of various movements on the time period of oscillations was investigated by simulating how long different driving methods took to reach 21 oscillations, illustrated in Figure 2.18.

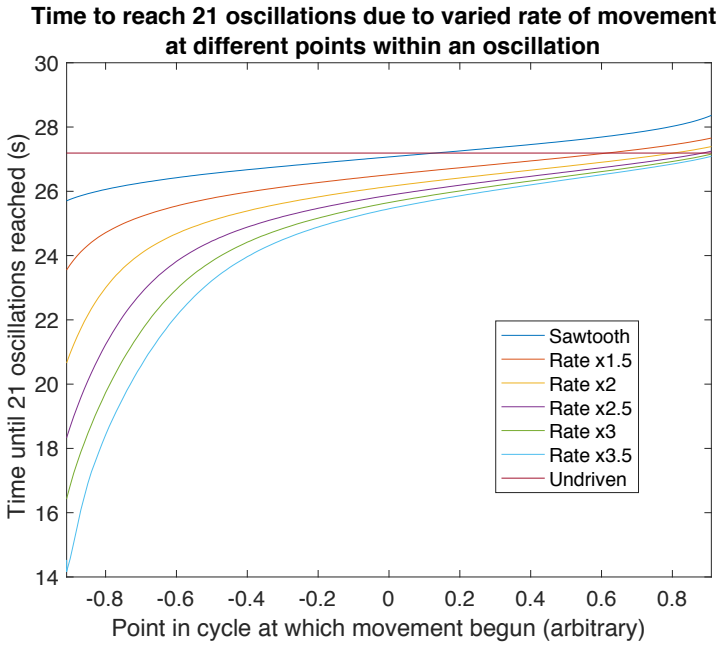


Figure 2.18: Distortion of time period with varied movement.

In Figure 2.18 it can be seen that the frequency of oscillations is largely effected by the driving motion of the system. It is trivial to see this by considering the effects of one burst of toque per half oscillation i.e. using a sawtooth wave to control the rotation of the double flail.

Consider a torque on the system acting at a time just after the system has reached maximum amplitude. This will act to accelerate the system down towards equilibrium, hence decreasing the time is takes to complete this half of the oscillation. If the torque is applied just after the system reaches every maxima, the period of the oscillations will be shortened in this way. This method is illustrated in Figure 2.16a by observing the sawtooth plot with a peak at the relative point -0.9 within the cycle.

Conversely, if a torque is applied to the system just before it reaches a maxima (at point 0.9 in the cycle as illustrated in Figure 2.16a), then this will act to accelerate the system away from equilibrium, taking a longer time to come to rest at a maximum. In this way a torque just before a maxima acts to increase the period of the system.

2.6.3.3 Optimal motion

Accounting for the fact that different methods of driving the system vary the time period of oscillation, it is possible to examine which method will not only drive the system to the greatest amplitude in the fewest number of oscillations, but which method will increase the amplitude in the fastest possible time.

From the data taken in examining the effects of different speeds of angular motion of the double flail, and the time taken to reach a certain number of oscillations for each of these rates; a relation can be obtained between the speed of movement, the point in an oscillation at which movement is commenced, and the increase in amplitude per oscillation per unit time due to these. The results are shown in Figure 2.19.

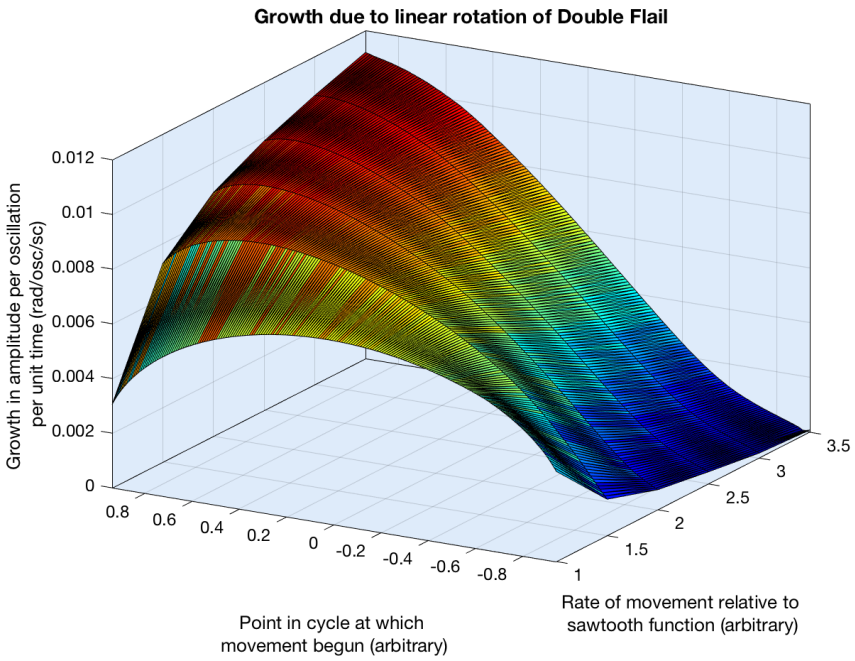


Figure 2.19: Growth in amplitude due to possible motions.

Figure 2.19 effectively shows that the increase in amplitude due to increased rate of rotation of the double flail on approaching maximum displacement will both cause the amplitude of the system to grow more, both with each oscillation and unit of time.

Taking the minimum time at which the robot can rotate between two extremes to be 0.48s, this corresponds to a maximum rate of 2.5 in the context of the program used, i.e. 2.5 times faster than a sawtooth method of movement. Looking at Figure 2.19, it can be seen that in order to get to the greatest amplitude in the shortest number of oscillations with a maximum rate of movement of 2.5 (times faster than a sawtooth movement), the double flail must move at this maximum rate, at a point in the cycle corresponding to an amplitude of 0.89

times the previous maximum. This movement is illustrated in Figure 2.20.

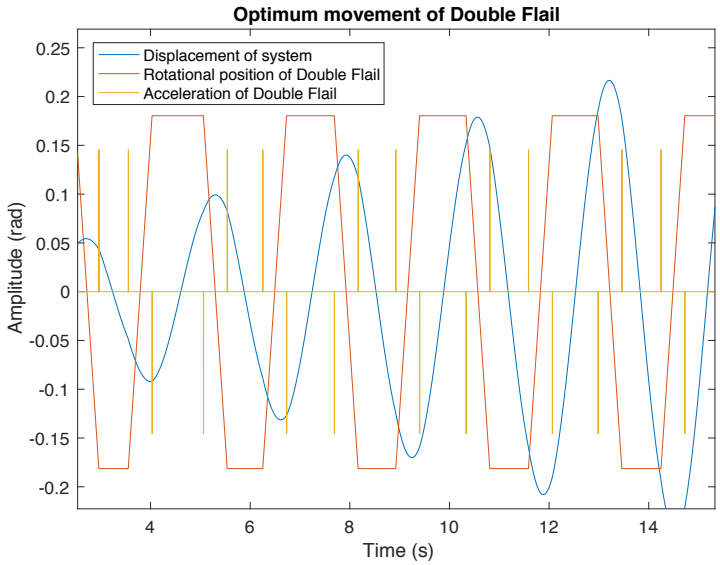


Figure 2.20: Optimum movement of double flail.

Figure 2.20 shows the displacement of the system over time, however with arbitrary values for the rotational position of the double flail and acceleration of the double added to illustrate its motion.

The double flail can be seen to begin rotating at a ratio of 0.89 of the previous maximum amplitude upon moving away from equilibrium and finish rotating at almost the same amplitude, however this time with the system moving towards equilibrium. This acceleration, shown in yellow in Figure 2.20, produces an impulse of torque on the system which acts to drive the amplitude.

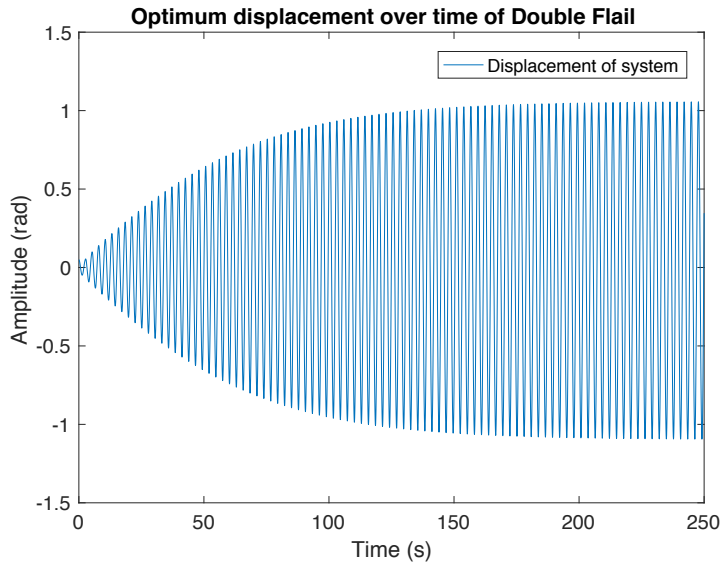


Figure 2.21: Displacement over time with optimum motion.

Using this motion, the amplitude can be seen to reach 1.0536 rad after 250 s, as illustrated in Figure 2.21 which shows a plot of the optimum motion over time.

2.6.4 Double-Hinged Double Flail

Daniel Corless

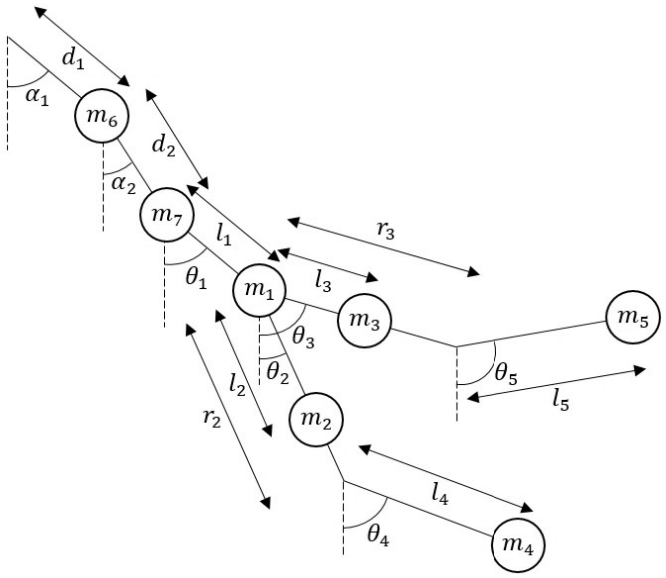


Figure 2.22: Diagram of the Double-Hinged Double Flail model.

This was the most advanced model being investigated. It was similar to the Double Flail model, but it also included two hinges on the swing in order to simulate a chain as shown in Figure 2.22. This introduced two additional free-to-move coordinates, α_1 and α_2, which described the angles to the vertical of the swing below the main pivot of the swing and below the first hinge respectively, with θ_1 now being the angle to the vertical of the swing below the second hinge. There were also two additional masses m_6 and m_7 that were placed at the positions of the upper and lower hinges respectively to represent the mass of the swing. For simplicity, m_1 was placed at the position of the seat, and l_1 was the distance from the second hinge to the seat, but all other variables and constants had the same meaning as in the Double Flail model. To shorten the following equations, M will now be defined as the sum of m_1, m_2, m_3, m_4 and m_5. Derivations for the Lagrangian and equations of motion can be found in Appendix A.2. The Lagrangian of this system is given by

$$\begin{aligned}\mathcal{L} = \frac{M}{2}\big[&d_1^2\dot{\alpha}_1^2 + d_2^2\dot{\alpha}_2^2 + l_1^2\dot{\theta}_1^2 + 2d_1d_2\dot{\alpha}_1\dot{\alpha}_2\cos(\alpha_1-\alpha_2) + 2d_1l_1\dot{\alpha}_1\dot{\theta}_1\cos(\alpha_1-\theta_1)\\ &+2d_2l_1\dot{\alpha}_2\dot{\theta}_1\cos(\alpha_2-\theta_1)\big] + \frac{m_2}{2}l_2^2\dot{\theta}_2^2 + \frac{m_3}{2}l_3^2\dot{\theta}_3^2 + \frac{m_4}{2}\big(r_2^2\dot{\theta}_2^2 + l_4^2\dot{\theta}_4^2\big)\\ &+\frac{m_5}{2}\big(r_3^2\dot{\theta}_3^2 + l_5^2\dot{\theta}_5^2\big) + (m_2l_2 + m_4r_2)\big[d_1\dot{\alpha}_1\dot{\theta}_2\cos(\alpha_1-\theta_2) + d_2\dot{\alpha}_2\dot{\theta}_2\cos(\alpha_2-\theta_2)\\ &+l_1\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\big] + (m_3l_3 + m_5r_3)\big[d_1\dot{\alpha}_1\dot{\theta}_3\cos(\alpha_1-\theta_3) + d_2\dot{\alpha}_2\dot{\theta}_3\cos(\alpha_2-\theta_3)\\ &+l_1\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)\big] + m_4l_4\big[d_1\dot{\alpha}_1\dot{\theta}_4\cos(\alpha_1-\theta_4) + d_2\dot{\alpha}_2\dot{\theta}_4\cos(\alpha_2-\theta_4)\\ &+l_1\dot{\theta}_1\dot{\theta}_4\cos(\theta_1-\theta_4) + r_2\dot{\theta}_2\dot{\theta}_4\cos(\theta_2-\theta_4)\big] + m_5l_5\big[d_1\dot{\alpha}_1\dot{\theta}_5\cos(\alpha_1-\theta_5)\\ &+d_2\dot{\alpha}_2\dot{\theta}_5\cos(\alpha_2-\theta_5) + l_1\dot{\theta}_1\dot{\theta}_5\cos(\theta_1-\theta_5) + r_3\dot{\theta}_3\dot{\theta}_5\cos(\theta_3-\theta_5)\big] + Mg\big[d_1\cos(\alpha_1)\\ &+d_2\cos(\alpha_2) + l_1\cos(\theta_1)\big] + (m_2l_2 + m_4r_2)g\cos(\theta_2) + (m_3l_3 + m_5r_3)g\cos(\theta_3)\\ &+m_4l_4g\cos(\theta_4) + m_5l_5g\cos(\theta_5) + \frac{m_6}{2}d_1^2\dot{\alpha}_1^2 + \frac{m_7}{2}\big[d_1^2\dot{\alpha}_1^2 + d_2^2\dot{\alpha}_2^2\\ &+2d_1d_2\dot{\alpha}_1\dot{\alpha}_2\cos(\alpha_1-\alpha_2)\big] + m_6gd_1\cos(\alpha_1) + m_7g\big[d_1\cos(\alpha_1) + d_2\cos(\alpha_2)\big],\end{aligned} \tag{2.36}$$

where d_1 and d_2 are the distances from the main pivot of the swing to the first hinge and from the first hinge to the second hinge. As the main pivot of the swing and the hinges are all unconstrained, the Euler-Lagrange equation was then respectively applied to α_1, α_2 and θ_1 to get the equations of motion for each angle.

$$\begin{aligned}(M+m_6+m_7)d_1\ddot{\alpha}_1 = -M\big[&d_2\ddot{\alpha}_2\cos(\alpha_1-\alpha_2) + d_2\dot{\alpha}_2^2\sin(\alpha_1-\alpha_2) + l_1\ddot{\theta}_1\cos(\alpha_1-\theta_1)\\ &+l_1\dot{\theta}_1^2\sin(\alpha_1-\theta_1)\big] - (m_2l_2 + m_4r_2)\big[\ddot{\theta}_2\cos(\alpha_1-\theta_2) + \dot{\theta}_2^2\sin(\alpha_1-\theta_2)\big]\\ &-(m_3l_3 + m_5r_3)\big[\ddot{\theta}_3\cos(\alpha_1-\theta_3) + \dot{\theta}_3^2\sin(\alpha_1-\theta_3)\big] - m_4l_4\big[\ddot{\theta}_4\cos(\alpha_1-\theta_4)\\ &+\dot{\theta}_4^2\sin(\alpha_1-\theta_4)\big] - m_5l_5\big[\ddot{\theta}_5\cos(\alpha_1-\theta_5) + \dot{\theta}_5^2\sin(\alpha_1-\theta_5)\big]\\ &-m_7d_2\big[\ddot{\alpha}_2\cos(\alpha_1-\alpha_2) + \dot{\alpha}_2^2\sin(\alpha_1-\alpha_2)\big] - Mg\sin(\alpha_1) - (m_6+m_7)g\sin(\alpha_1)\end{aligned} \tag{2.37}$$

$$\begin{aligned}(M+m_7)d_2\ddot{\alpha}_2 = -M\big[&d_1\ddot{\alpha}_1\cos(\alpha_1-\alpha_2) - d_1\dot{\alpha}_1^2\sin(\alpha_1-\alpha_2) + l_1\ddot{\theta}_1\cos(\alpha_2-\theta_1)\\ &+l_1\dot{\theta}_1^2\sin(\alpha_2-\theta_1)\big] - (m_2l_2 + m_4r_2)\big[\ddot{\theta}_2\cos(\alpha_2-\theta_2) + \dot{\theta}_2^2\sin(\alpha_2-\theta_2)\big]\\ &-(m_3l_3 + m_5r_3)\big[\ddot{\theta}_3\cos(\alpha_2-\theta_3) + \dot{\theta}_3^2\sin(\alpha_2-\theta_3)\big] - m_4l_4\big[\ddot{\theta}_4\cos(\alpha_2-\theta_4)\\ &+\dot{\theta}_4^2\sin(\alpha_2-\theta_4)\big] - m_5l_5\big[\ddot{\theta}_5\cos(\alpha_2-\theta_5) + \dot{\theta}_5^2\sin(\alpha_2-\theta_5)\big]\\ &-m_7d_1\big[\ddot{\alpha}_1\cos(\alpha_1-\alpha_2) - \dot{\alpha}_1^2\sin(\alpha_1-\alpha_2)\big] - Mg\sin(\alpha_2) - m_7g\sin(\alpha_2)\end{aligned} \tag{2.38}$$

$$\begin{aligned}Ml_1\ddot{\theta}_1 = -M\big[&d_1\ddot{\alpha}_1\cos(\alpha_1-\theta_1) - d_1\dot{\alpha}_1^2\sin(\alpha_1-\theta_1) + d_2\ddot{\alpha}_2\cos(\alpha_2-\theta_1)\\ &-d_2\dot{\alpha}_2^2\sin(\alpha_2-\theta_1)\big] - (m_2l_2 + m_4r_2)\big[\ddot{\theta}_2\cos(\theta_1-\theta_2) + \dot{\theta}_2^2\sin(\theta_1-\theta_2)\big]\\ &-(m_3l_3 + m_5r_3)\big[\ddot{\theta}_3\cos(\theta_1-\theta_3) + \dot{\theta}_3^2\sin(\theta_1-\theta_3)\big] - m_4l_4\big[\ddot{\theta}_4\cos(\theta_1-\theta_4)\\ &+\dot{\theta}_4^2\sin(\theta_1-\theta_4)\big] - m_5l_5\big[\ddot{\theta}_5\cos(\theta_1-\theta_5) + \dot{\theta}_5^2\sin(\theta_1-\theta_5)\big] - Mg\sin(\theta_1)\end{aligned} \tag{2.39}$$

The problem with these equations is that they contain terms with second order time derivatives for coordinates that are free to move, meaning that decoupling them to apply Runge Kutta integration is not as simple as in previous examples. By substitution of the equations into each other it is possible to eliminate these terms. By doing this and using trigonometric identities such as $\sin^2(A) + \cos^2(A) = 1$ and $\cos(A) + \cos(B) = \frac{1}{2}\big[\cos(A-B) + \cos(A+B)\big]$, the equation of motion for α_1 can be written as

$$\begin{aligned}
&\Big[M + m_6 + m_7 + \frac{(M+m_7)\cos(\alpha_1-\alpha_2)}{m_7 + M\sin^2(\alpha_2-\theta_1)}\big[-M\sin(\alpha_1-\theta_1)\sin(\alpha_2-\theta_1) - m_7\cos(\alpha_1-\alpha_2)\big] \\
&+\frac{(M+m_7)\cos(\alpha_1-\theta_1)}{m_7 + M\sin^2(\alpha_2-\theta_1)}\big[M\sin(\alpha_1-\alpha_2)\sin(\alpha_2-\theta_1)\big]\Big]d_1\ddot{\alpha}_1 = -M\big[d_2\dot{\alpha}_2^2\sin(\alpha_1-\alpha_2) \\
&+l_1\dot{\theta}_1^2\sin(\alpha_1-\theta_1)\big] - (m_2l_2+m_4r_2)\big[\ddot{\theta}_2\cos(\alpha_1-\theta_2) + \dot{\theta}_2^2\sin(\alpha_1-\theta_2)\big] - (m_3l_3 \\
&+m_5r_3)\big[\ddot{\theta}_3\cos(\alpha_1-\theta_3) + \dot{\theta}_3^2\sin(\alpha_1-\theta_3)\big] - (m_4l_4)\big[\ddot{\theta}_4\cos(\alpha_1-\theta_4) + \dot{\theta}_4^2\sin(\alpha_1-\theta_4)\big] \\
&-(m_5l_5)\big[\ddot{\theta}_5\cos(\alpha_1-\theta_5) + \dot{\theta}_5^2\sin(\alpha_1-\theta_5)\big] - Mg\sin(\alpha_1) - m_7d_2\dot{\alpha}_2^2\sin(\alpha_1-\alpha_2) \\
&-(m_6+m_7)g\sin(\alpha_1) - \frac{(M+m_7)\cos(\alpha_1-\alpha_2)}{m_7 + M\sin^2(\alpha_2-\theta_1)}\Big[\sin(\alpha_2-\theta_1)\Big[-M\big[d_1\dot{\alpha}_1^2\cos(\alpha_1-\theta_1) \\
&+d_2\dot{\alpha}_2^2\cos(\alpha_2-\theta_1)\big] - (m_2l_2+m_4r_2)\big[\ddot{\theta}_2\sin(\theta_2-\theta_1) + \dot{\theta}_2^2\cos(\theta_2-\theta_1)\big] - (m_3l_3 \\
&+m_5r_3)\big[\ddot{\theta}_3\sin(\theta_3-\theta_1) + \dot{\theta}_3^2\cos(\theta_3-\theta_1)\big] - (m_4l_4)\big[\ddot{\theta}_4\sin(\theta_4-\theta_1) + \dot{\theta}_4^2\cos(\theta_4-\theta_1)\big] \\
&-(m_5l_5)\big[\ddot{\theta}_5\sin(\theta_5-\theta_1) + \dot{\theta}_5^2\cos(\theta_5-\theta_1)\big] - Mg\cos(\theta_1) - Ml_1\dot{\theta}_1^2\Big] + m_7d_1\dot{\alpha}_1^2\sin(\alpha_1-\alpha_2) \\
&-m_7g\sin(\alpha_2)\Big] - \frac{(M+m_7)\cos(\alpha_1-\theta_1)}{m_7 + M\sin^2(\alpha_2-\theta_1)}\Big[Md_1\dot{\alpha}_1^2\cos(\alpha_1-\alpha_2)\sin(\alpha_2-\theta_1) \\
&+\frac{M^2}{M+m_7}\big[l_1\dot{\theta}_1^2\cos(\alpha_2-\theta_1)\sin(\alpha_2-\theta_1)\big] - \frac{M\sin(\theta_1-\alpha_2)}{M+m_7}\Big[(m_2l_2+m_4r_2)\big[\ddot{\theta}_2\sin(\theta_2-\alpha_2) \\
&+\dot{\theta}_2^2\cos(\theta_2-\alpha_2)\big] + (m_3l_3+m_5r_3)\big[\ddot{\theta}_3\sin(\theta_3-\alpha_2) + \dot{\theta}_3^2\cos(\theta_3-\alpha_2)\big] + (m_4l_4)\big[\ddot{\theta}_4\sin(\theta_4-\alpha_2) \\
&+\dot{\theta}_4^2\cos(\theta_4-\alpha_2)\big] + (m_5l_5)\big[\ddot{\theta}_5\sin(\theta_5-\alpha_2) + \dot{\theta}_5^2\cos(\theta_5-\alpha_2)\big]\Big] - \frac{m_7}{M+m_7}\Big[(m_2l_2 \\
&+m_4r_2)\big[\ddot{\theta}_2\cos(\theta_1-\theta_2) + \dot{\theta}_2^2\sin(\theta_1-\theta_2)\big] + (m_3l_3+m_5r_3)\big[\ddot{\theta}_3\cos(\theta_1-\theta_3) + \dot{\theta}_3^2\sin(\theta_1-\theta_3)\big] \\
&+(m_4l_4)\big[\ddot{\theta}_4\cos(\theta_1-\theta_4) + \dot{\theta}_4^2\sin(\theta_1-\theta_4)\big] + (m_5l_5)\big[\ddot{\theta}_5\cos(\theta_1-\theta_5) + \dot{\theta}_5^2\sin(\theta_1-\theta_5)\big]\Big] \\
&-Md_2\dot{\alpha}_2^2\sin(\theta_1-\alpha_2) - Mg\cos(\alpha_2)\sin(\theta_1-\alpha_2)\Big].
\end{aligned} \tag{2.40}$$

Using similar methods yields the equations of motion for α_2 and θ_1 in a similar form.

$$\begin{aligned}
&\big[m_7 + M\sin^2(\alpha_2-\theta_1)\big]d_2\ddot{\alpha}_2 = \sin(\alpha_2-\theta_1)\Big[-M\big[d_1\ddot{\alpha}_1\sin(\alpha_1-\theta_1) + d_1\dot{\alpha}_1^2\cos(\alpha_1-\theta_1) \\
&+d_2\dot{\alpha}_2^2\cos(\alpha_2-\theta_1)\big] - (m_2l_2+m_4r_2)\big[\ddot{\theta}_2\sin(\theta_2-\theta_1) + \dot{\theta}_2^2\cos(\theta_2-\theta_1)\big] - (m_3l_3 \\
&+m_5r_3)\big[\ddot{\theta}_3\sin(\theta_3-\theta_1) + \dot{\theta}_3^2\cos(\theta_3-\theta_1)\big] - (m_4l_4)\big[\ddot{\theta}_4\sin(\theta_4-\theta_1) + \dot{\theta}_4^2\cos(\theta_4-\theta_1)\big] \\
&-(m_5l_5)\big[\ddot{\theta}_5\sin(\theta_5-\theta_1) + \dot{\theta}_5^2\cos(\theta_5-\theta_1)\big] - Mg\cos(\theta_1) - Ml_1\dot{\theta}_1^2\Big] \\
&-m_7d_1\big[\ddot{\alpha}_1\cos(\alpha_1-\alpha_2) - \dot{\alpha}_1^2\sin(\alpha_1-\alpha_2)\big] - m_7g\sin(\alpha_2)
\end{aligned} \tag{2.41}$$

$$\begin{aligned}\big[Mm_7 + M^2\sin^2(\alpha_2-\theta_1)\big]l_1\ddot{\theta}_1 = M\big[d_1\ddot{\alpha}_1\sin(\alpha_1-\alpha_2)\sin(\alpha_2-\theta_1) + d_1\dot{\alpha}_1^2\cos(\alpha_1-\alpha_2)\sin(\alpha_2-\theta_1)\big] \\ + \frac{M^2}{M+m_7}\big[l_1\dot{\theta}_1^2\cos(\alpha_2-\theta_1)\sin(\alpha_2-\theta_1)\big] - \frac{M\sin(\theta_1-\alpha_2)}{M+m_7}\Big[(m_2l_2+m_4r_2)\big[\ddot{\theta}_2\sin(\theta_2-\alpha_2) \\ +\dot{\theta}_2^2\cos(\theta_2-\alpha_2)\big] + (m_3l_3+m_5r_3)\big[\ddot{\theta}_3\sin(\theta_3-\alpha_2) + \dot{\theta}_3^2\cos(\theta_3-\alpha_2)\big] + (m_4l_4)\big[\ddot{\theta}_4\sin(\theta_4-\alpha_2) \\ +\dot{\theta}_4^2\cos(\theta_4-\alpha_2)\big] + (m_5l_5)\big[\ddot{\theta}_5\sin(\theta_5-\alpha_2) + \dot{\theta}_5^2\cos(\theta_5-\alpha_2)\big]\Big] - \frac{m_7}{M+m_7}\Big[(m_2l_2 \\ +m_4r_2)\big[\ddot{\theta}_2\cos(\theta_1-\theta_2) + \dot{\theta}_2^2\sin(\theta_1-\theta_2)\big] + (m_3l_3+m_5r_3)\big[\ddot{\theta}_3\cos(\theta_1-\theta_3) + \dot{\theta}_3^2\sin(\theta_1-\theta_3)\big] \\ +(m_4l_4)\big[\ddot{\theta}_4\cos(\theta_1-\theta_4) + \dot{\theta}_4^2\sin(\theta_1-\theta_4)\big] + (m_5l_5)\big[\ddot{\theta}_5\cos(\theta_1-\theta_5) + \dot{\theta}_5^2\sin(\theta_1-\theta_5)\big]\Big] \\ -Md_2\dot{\alpha}_2^2\sin(\theta_1-\alpha_2) - Mg\cos(\alpha_2)\sin(\theta_1-\alpha_2)\end{aligned} \tag{2.42}$$

Substitution for $\ddot{\alpha}_1$ from Equation 2.40 into these equations results in there being expressions for each angular acceleration with no second order time derivatives for free-to-move coordinates, meaning that these equations could be easily decoupled using the same method as for previous models. After adding damping terms to the equations of motion with a damping constant of 0.00443 s^{-1}, a simulation was run using similar robot movements to those used for the Double Flail model. It must be noted that because the mass of the swing was modeled using point masses at the hinges rather than an even distribution of mass along the swing, the lower part of the swing was caused to oscillate about the hinges unrealistically, illustrated in Figure 2.23.

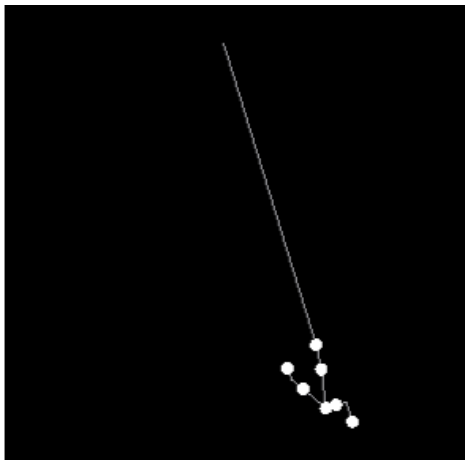


Figure 2.23: A single frame from a graphical simulation with realistic mass values $m_1 = 0.421\ kg$, $m_6 = 1.291\ kg$ and $m_7 = 1.082\ kg$. The angle between the swing above and below the hinges can clearly be seen, causing oscillations of the lower part of the swing about the hinges.

However, the effect of this reduced as the masses at the hinges were reduced, and these oscillations were quite small when using realistic values for the masses, so this was not a major problem. Another issue was that the masses were positioned such that the centre of mass of the swing was a bit lower than it was in reality. However, this did not produce a significant difference and using mass values of 0.421 kg, 1.291 kg and 1.082 kg for m_1, m_6 and m_7 respectively was a reasonable approximation for the mass distribution.

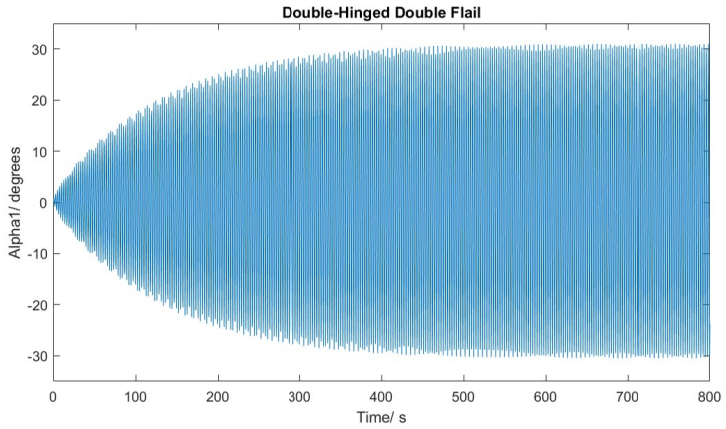


Figure 2.24: A graph of the variation of angle α_1 over time. The amplitude slowly builds over time and plateaus at 30°.

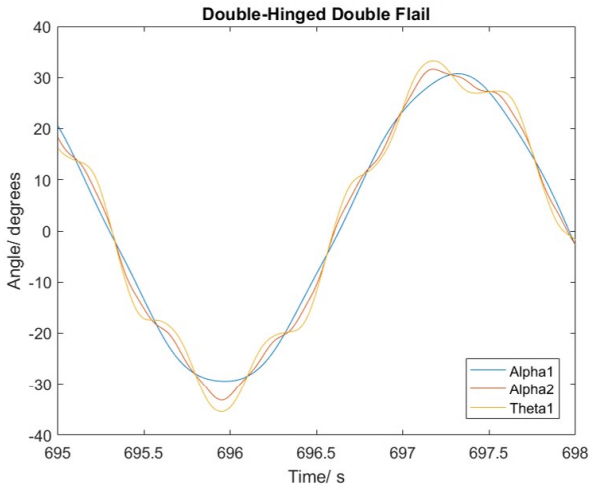


Figure 2.25: A graph of the variation of angles α_1, α_2 and θ_1 over three seconds in the simulation.

Figure 2.24 shows the output of the simulation for α_1 over time. Figure 2.25 shows the variation of angles α_1, α_2 and θ_1 over a short period of time, and the output for α_1 and α_2 further illustrates the high frequency oscillations present in the model. The swinging amplitude can be seen to reach a maximum of about 30°, which is greater than the amplitude reached with the hinges removed, which was found to be about 27°. This suggests that adding hinges to the swing does give an increase in amplitude. However, it is unclear how much this is due to the discussed oscillations. Therefore, if the mass distribution was more evenly distributed along the swing as is the case in reality, then the increase in amplitude might have been less noticeable. Such a system could therefore be investigated in the future. As with the Double Flail model, the maximum amplitude was considerably greater than the amplitude of 15°reached by the real robot, which indicates that there could have been additional causes of damping in the real system that had not been accounted for. Another reason for the greater amplitude could be that the torques applied to the system in the model had been overestimated. However, comparisons of the model with data from the real robot indicated that the torques applied in the model were quite realistic, meaning that the difference in maximum amplitude was likely to be due to damping.

2.7 Graphical Simulations of Models

Joshua Torbett

To test the models an object orientated program was written in C++, set up as universal program that can graphically simulate any of the investigated models.

This is done by each model producing (x, y) position data for each mass in the model. Then the program reads in this data and the masses are graphically simulated and relevant lines drawn to and between masses depending on the model.

The structure of the program works in two main parts the control window, in which the type of model is selected and the location and name of data file.

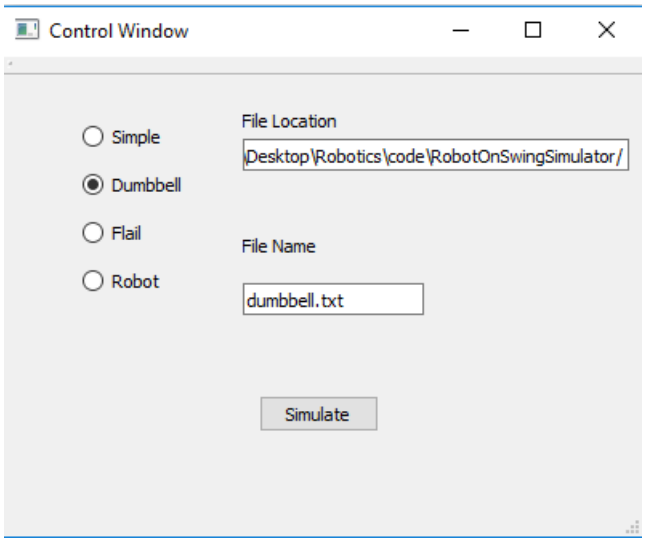


Figure 2.26: Control Window

Once this information is in place, the simulate button will produce a display window with a graphical simulation based off the earlier parameters.

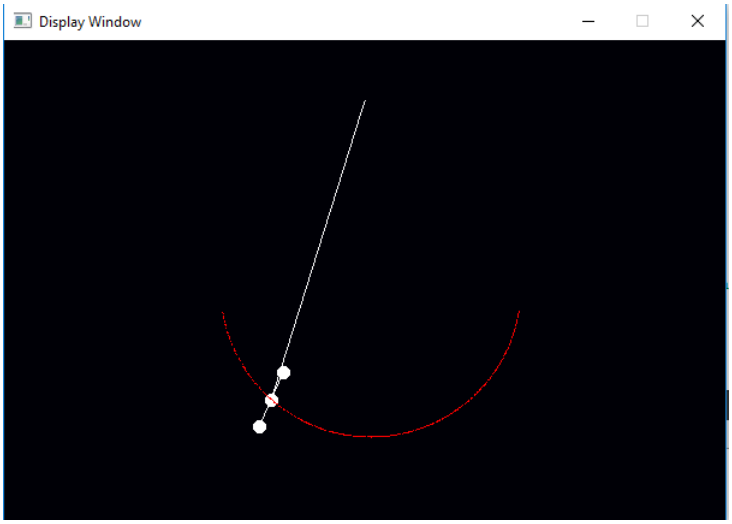


Figure 2.27: Display Window

As it was hard to distinguish from only data and graphs if the motion acts as intended. These graphical simulations allow for checking the general motion of the masses is accurate, such as the red tracing line seen in figure 2.27, that traces the mass connected to the pivot.

Also motions that add energy to the system need to happen at the correct point in the cycle, this can be checked and for models such at the virtual robot that angles of body parts are as expected. Additionally due to object oriented nature of the program multiple simulations can be ran at the same time.

In summary the program gives another method to test the models are accurate, motions to add energy to the system occur at the right point in the cycle and allows the direct comparison of models of different type or parameters.

2.8 Conclusion

Max Hadfield

The Numerical Modelling Group developed a successful method through which systems designed to represent a moving robot on a swing could be simulated. The systems were analysed using Lagrangian mechanics to obtain a pair of coupled first order differential equations. These were included into a C++ program which allowed the movement of the robot to be programmed, and then solved the equations of motion using a 4th order Runge Kutta method. A flexible GUI was developed which could be used to display the output from the simulation, making it easy to identify and fix errors in the simulations. Using this system, models of increasing complexity were simulated and analysis of the results was carried out so that advice could be given to the robot group regarding how best to move the robot.

A simulation of an undriven damped simple pendulum was used to determine that a damping term of -0.00886ω rad s^{-2} gave an accurate representation of the effect of resistive forces on the angular acceleration of the swing. The effect of a rotating body on the motion of a pendulum was investigated using the Dumbbell Model (shown in Figure 2.4). This model showed that varying the angular momentum of the robot could provide torque to the swing, and it was found that for this specific model that the largest amplitude was achieved when the movement frequency was equal to the natural frequency of the pendulum. The effect of varying the position

of the centre of mass of a body on a pendulum was investigated using the Length Variation model. This showed that moving the robot's centre of mass could provide torque to the swing, and it was found for this specific model that the largest swing angle occurred when the movement frequency was twice the natural frequency.

A Single Flail model (shown in Figure 2.8) was created, this model was driven using both sinusoidal and step-like motions. From this investigation it was found that motions should be carried out across the largest angle range possible and should be performed at frequencies which are multiples of the natural frequency of the swing to take advantage of resonance effects. It was also found that the step-like motion was able to produce larger swing amplitudes than the sinusoidal motion. A Double Flail model (shown in Figure 2.14) was investigated by varying the position of a sawtooth motion with the swing's motion and varying the point in the cycle at which torque was applied. From these investigations it was concluded that the optimum movement was for the robot to begin rotating just before the swing reached a peak, and finish rotating as soon as possible after the peak had been reached.

Investigations were carried out into the effect on unfixing the hinges on the swing so that it behaved more similarly to a chain swing than a rigid pendulum. This was done through the simulation of a Double-Hinged Double Flail model (shown in Figure 2.22). Due to time constraints full analysis was not carried out on this model, however initial investigations suggested that, when driven with the same movement pattern, the Double Hinged Double Flail produced a slightly higher swing amplitude than the Double Flail.

3 Robot and Swing Properties

3.1 Introduction

Katherine Evans

This section of the report covers familiarisation with the capabilities of the NAO robot with regards to its range of motion and strength and also how to connect to the robot and control its motions. The physical properties of the swing within the laboratory were also measured including the damping through use of the angle encoder. The robot's sensors were investigated to provide confirmations of potential feedback methods that would eliminate the need for external inputs in swinging motions which was a key aim of the investigation.

3.1.1 Connecting to the NAO

The most beneficial way to connect to the NAO so that all software and libraries were accessible was to set up a wireless connection to one of the computers within the Robotics Laboratory. With some assistance from laboratory staff it was possible to give this computer, which also had the angle encoder of the swing connected for later use, a secondary network connection to the router used by the robot. The robot, with the wireless connection via the router, then had a fixed IP address which could be used in all Python scripts aiming to communicate with the real robot. This method also allowed communication with the NAO robot via the router with personal laptops by adding their MAC addresses to the whitelist in the router.

3.1.2 Choregraphe and Communicating with the NAO

Aldebaran Robotics' programming software, Choregraphe, provides a simple way to create and execute behaviours on the NAO or a virtual robot through the use of box scripts connected together in a flow diagram sequentially or in parallel. The prewritten behaviour boxes allowed for familiarisation with the possible behaviours and how they are interpreted through NAOqi (see section 3.2 for further details on the use of NAOqi). The Choregraphe graphical user interface had many benefits early on in the investigation especially in allowing all joint angles and sensor values to be easily monitored continuously using the 'Memory Watcher Panel'. It should be noted that Choregraphe v1.14.5 is needed for compatibility with the V4 NAO provided, however, it was found that to communicate with a robot in Webots 8, a later version, Choregraphe v2.1.4 is needed.

Initially, Choregraphe was used as the sole way of communicating with the robot. To get the robot to move to a set position it was placed into said position, this was saved within the Timeline and the command was run as needed. The angles of joints were also controlled and monitored through Choregraphe. Python scripts could be written and run through script boxes in just the same way as the preprogrammed ones included in the library and it would have been possible to continue relying on Choregraphe to provide the communication with the NAO as all the API of the NAOqi can be accessed within Choregraphe. However, incorporating the angle encoder feedback for swinging motion later in the investigation became the first motivation to move away from using Choregraphe and instead control the robot directly from a Python script as there were difficulties importing the encoder library within Choregraphe. To make this possible, the Python 2.7 SDK 1.14.5 is needed so that it is compatible with the version of the NAO used.

3.2 Controlling The Robot

Harry Withers

The aim when controlling the robot was to write a single Python program that could be used to control both simulated and real robot with minimal alteration. This is in contrast to last year's approach of writing a model

straight into the Webots controller file and then writing separate code using the same base model to control the real robot [7]. The new approach cut down the amount of code having to be written by half and increased the efficiency of the project.
The code used to control the robot was from the NAOqi Python API provided by Aldebaran Robotics. NAOqi is the name of the software that runs on the robot and therefore the API provides a method of running code both on a laptop and on the robot itself [10]. This API is based on a number of key concepts in the NAOqi framework. When a robot is started, either real or simulated, it runs an executable called a broker [10]. A broker is an object that provides access, via a proxy in this case, to the robot's modules and their functions [10]. The main module used for controlling the robot is the `ALMotion` module which is simply a class that contains functions for altering the robot's position. In order to control the module, a proxy is used which acts as the class it represents and contains all of the module's available functions [10]. This proxy belongs to the class `ALProxy` provided in the NAOqi API; a proxy can be connected to a module using the following line of Python code

```
moduleProxy = ALProxy('module_name','robot_ip',robot_port)
```

where the robot IP address for a simulated robot is 127.0.0.1 and the port is 9559 for both real and simulated. This line of code was used in every program in order to connect to the robot.

3.2.1 SwingAPI

In order to test numerical models and machine learning techniques an API needed to be created to control both the real and virtual robot. This was the SwingAPI which included a number of functions to allow the user to move the robot in a predefined manner. Every function in the API required that the user pass to it a proxy to the robots ALMotion module.
Each function in the API stems from `moveLimbs`, a function which calls `setAngles` in the NAOqi API. This function required the user to enter a fraction of the joint motors maximum speed at which the joint would be moved. In order to change the angle of a joint, that limb must first be made to be stiff and so this function first calls the `setStiffness` function also in the NAOqi API. Finally, this function sends the thread to sleep for a passed period of time allowing the robot to complete the movement.
Four more functions were included in this API, two of which were the `position1Dynamic` and `postition2Dynamic` functions which, when called, would move the robot into either position 1 or position 2 as shown in Figure 3.1. These functions both call `moveLimbs`, explained earlier. In the earlier attempts of controlling the robot, using the functions `position1Mono` and `position2Mono`, every joint was moved with a single call. This also meant that every joint was set to move at the same speed. At lower speeds in particular, this causes a problem as the joints with a smaller range to travel, such as the hip pitch, complete the movement in a smaller time frame than the joints with a larger range, such as the knee pitch. This would cause the legs of the robot to swing out of phase with the hip and thus swinging became ineffective.
The dynamic functions were introduced to solve this issue; a user passes to these functions the speed they would like the knee pitch to travel at. The function then calculates what the speed of the other joints should be based on the ratio of their range to the knee joint. This ensures that even at low speeds, every joint completed its movement in the same time frame.

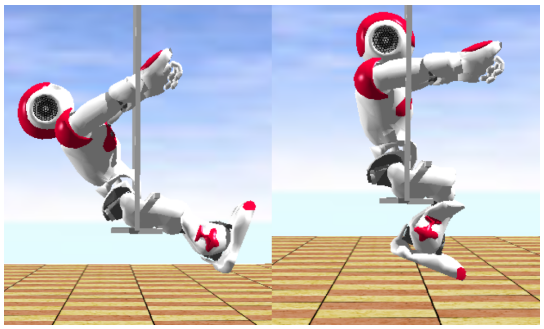


Figure 3.1: A picture showing the 2 positions that were available to the robot. The left is position 1 and the right position 2.

3.3 Strength testing

3.3.1 Method

Thomas Smith

The strength of the robot was obtained because we were thinking about adding weights on the limbs of the robot to improve its swinging abilities. The values that were obtained for the stall torque can be used to get the maximum possible weight that could added before the motor starts to stall, but this was decided against because it would put strain on the motors which would cause the robot to get hotter faster as well as possibly breaking the robot if too much weight was added. This extra weight would also mean that the robot would be using more power so it will run out of battery faster. This if the robot was running for long periods of time, like was needed for the machine learning side of the project, the robot would have needed to be plugged into the mains more that without the added weights. The stall torque was used in some of the models of the robot as a maximum value for torque that the motors can create. This experiment had to be done carefully because the limits of the robot were being tested, so if the robot overexerted itself by pushing too hard against the Newton meter when making measurements, the gears could have been damaged. It was necessary not to take too many readings of the same motor in a short period of time because the motor heated up very fast while taking the readings. For example, after one measurement of the knee pitch motor the temperature, obtained from the internal sensors, went from 55 °C to 65 °C. This made it hard to repeats measurements.

There were a number of ways that the strength of the robot could have been obtained. The stall torque of the motors was used to get a measure of the strength of the robot's motors. Stall torque is the torque produced by a motor when the rotational speed is 0. The theoretical values for the motors stall torque was found from NAO online documentation [11]. These values were not going to be correct for the robot because as the motors are used, they lose some of their efficiency due to overuse. To get the real values for stall torque of the motors some tests needed to be performed. There are many ways that this could be achieved, for example weights could have been added to the limbs incrementally until the motor stalls but this would take longer and require multiple readings. The stall torque was found by attaching the robot into the seat, the same way it would be in the swing, and clamped the seat on a table then an electric Newton Meter, that could measure tension and compression, was placed it 90 °to the limb, then by using Choregraph the limbs were moved but they were block it with the Newton Meter causing the motor to stall. The angle between the Newton meter was not exactly 90 °, the angle between the limb and the Newton meter needed to be recorded. To do this, the initial angle of the limb was recorded from Choregraphy and the angle between the limb and the Newton meter was at 90 ° and was clamped in place. Then once the limb has been moved, the angle is then recorded and the difference between the initial and final angle of the limb is the amount the angle between limb and the Newton meter changes. The Newton meter then yielded a value in grams for the force exerted by the motor, which was then multiply this by acceleration due to gravity to get it in units of Newtons. The compression setting was used because it was easier for us to position the Newton Meter in front of the robot than behind it and the values

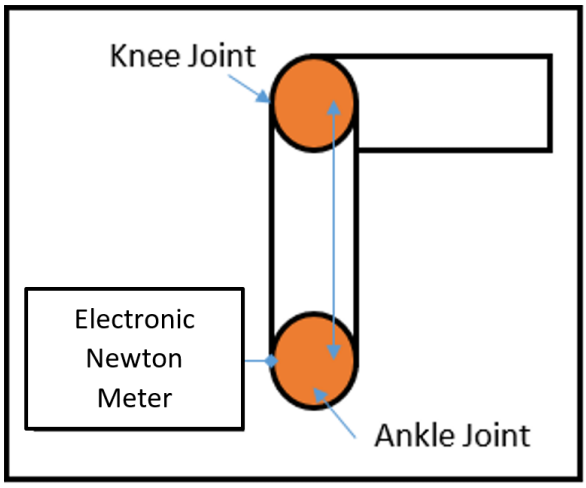


Figure 3.2: Equipment set-up for Strength Tests.

that were found from the compression and tension settings were the same in initial testing. Figure 3.2 shows the equipment set-up for the strength test.

The length between the joint and where the readings were taken from was needed to convert the force into a torque. It was possible to choose any point, and measure the distance between the motor and the point. However, this would have made thing difficult because the exact position of the motor was hard to see from the outside. Additionally, the exterior of the robot is very uneven so it would be difficult to measure because the Newton meter needed to be perpendicular to the limbs of the robot. This was meant that the error on the length used to get the torque would be very large. There was a better way of getting length. The specifications [12] of the robot were used to get the length; this was done by using the lengths between joints. For the Knee Pitch the length between the Knee Joint and the Ankle Joint was used, for the elbow the length between the elbow joint and the wrist joint, for the hips the length that was used was from the hips to the middle of the back. With the hips it was possible to use any point on the back because it was much flatter than the rest of the robot. For the hips, it was a bit different from the other joints because all the other joints the left and right ones were done separately but because the robot was fastened to the seat which was attached to the table so they could not be done separately so instead one reading for their combined strength was taken. Doing it from joint to joint is a good way of taking measurements it not just because it is a known length but it also means that the Newton meter is placed on the ball of the joint which is uniform compared to the rest of the body, which makes it easier to get the Newton meter to be at 90 °to the limb.

3.3.2 Results and conclusions

Adam Szekely

The theoretical results for the strength of the motors could be found using data on the Aldebaran website and by using the equation,

$$\hat{\tau} k = \hat{l} \wedge \hat{F}, \tag{3.1}$$

where τ is the theoretical stall torque of the motor (otherwise known as the instantaneous torque); $\hat{F}$ is force vector quantity that describes the weight generated on the Newton meter multiplied by gravitational acceleration of the Earth; $\hat{l}$ is a vector quantity that describes the distance between the pivot point of a motor and the point at which the force is exerted, and k is the speed reduction ratio. Where the speed reduction ratio, is given by,

$$Speed\ reduction\ ratio = \frac{Number\ of\ teeth\ on\ the\ driven\ gear}{Number\ of\ teeth\ on\ the\ driving\ gear}. \tag{3.2}$$

The experimental instantaneous torque (referred to as torque throughout this section), $l \wedge F$, should be equal to the theoretical stall torque, τ, multiplied by the speed reduction ratio, k. The results are displayed in Table 3.1.

Table 3.1: A table containing the quantities required to find torque from theoretical data found on the Aldebaran website for the major motors on the NAO H25 robot [11]

Moving component	Theoretical stall torque, $\boldsymbol{\tau}$ (Nm)	Theoretical stall torque error (Nm)	Speed reduction ratio (k)	Resultant theoretical torque, $\boldsymbol{\tau}k$ (Nm)	Resultant torque error (Nm)
Shoulder (Pitch)	0.0143	±0.0011	150.27	2.15	±0.17
Elbow (Roll)	0.0143	±0.0011	173.22	2.48	±0.20
Knee (Pitch)	0.0680	±0.0054	130.85	8.90	±0.71
Torso (Combined Hip Pitches)	0.1360	±0.0109	130.85	17.80	±1.42

.

The results for torque, displayed below, were obtained using the method described in the previous section. The equation,

$$\tau = Flsin\theta, \tag{3.3}$$

was used to calculate the torque, where τ is the torque produced by one of the motors of the robot; F is the gravitational acceleration multiplied by the weight value produced on the Newton meter by the motor; l is the distance between the pivot point of the motor and the point at which the force is exerted. Finally, θ is the angle between the length for which torque is measured and the direction perpendicular to the ground.

Table 3.2: A table containing the requisite quantities for the torque calculations for the listed joints, corresponding to the relevant motor on the NAO H25.

Moving Component	Mass (kg)	Length of moving component (m)	Angle Θ, at which the moving component is to the normal force (degrees)	Torque (Nm)	Torque error (Nm)
Right shoulder	1.177	0.105	83.00	1.20	$\pm$0.17
Left shoulder	1.126	0.105	81.00	1.15	$\pm$0.16
Right elbow	1.265	0.114	39.00	0.89	$\pm$0.13
Left elbow	1.482	0.114	50.00	1.27	$\pm$0.18
Right knee	4.870	0.103	71.00	4.65	$\pm$0.68
Left knee	4.030	0.103	80.00	4.01	$\pm$0.59
Torso	3.702	0.161	55.00	4.79	$\pm$0.48

An assumption was made that the thigh of the robot, when seated, was parallel to the floor. This meant that it could be assumed that angular value of the motor, given on Choregraphe, could be taken to be the true angle that the moving limb was relative to the ground. Gravity was taken to be $9.81 \mathrm{m} s^{-2}$ as standard. Though the Newton meter measured weight to an accuracy of ±0.001kg, the error taken on the weight measurement was taken to be ±0.005kg, due to it being necessary to cover the compression point on the Newton meter with Blu-Tack to protect the plastic casing of the robot. The error given on the length was ±0.015m, which was twice the diameter of the compression point attached to the Newton meter. This was an estimate of human error of its placement. The angular error was taken to be ±0.05°, as this was the highest degree of accuracy of Choregraphe. The total error was calculated using the standard error formula on equation 3.3. The largest error contribution was produced as a consequence of the positioning of the Newton meter along the moving section of the robot. The other error contributions were over an order of magnitude smaller, yielding them negligible,

meaning the error was effectively,

$$\sigma_\tau = \pm\Big(9.81 \times M \times sin\theta \times 0.015\Big). \tag{3.4}$$

On comparing the theoretical and experimental data, it was clear that the results are within an order of magnitude of one another and that the theoretical values are significantly larger than those calculated using experimental data. This is illustrated below.

Table 3.3: A table illustrating a comparison of torque between data found on the Aldebaran website and data obtained via experimental means [11].

Moving component	Experimental torque (Nm)	Experimental torque error (Nm)	Theoretical torque (Nm)	Theoretical torque error (Nm)	Experimental torque as a percentage of theoretical torque	Percentage errors
Right shoulder	1.20	±0.17	2.15	±0.17	55.8	±9.1
Left shoulder	1.15	±0.16	2.15	±0.17	53.5	±8.6
Right elbow	0.89	±0.13	2.48	±0.20	35.9	±6.0
Left elbow	1.27	±0.18	2.48	±0.20	51.3	±8.3
Right knee	4.65	±0.68	8.90	±0.71	52.3	±8.7
Left knee	4.01	±0.59	8.90	±0.71	45.1	±7.5
Torso	4.79	±0.48	17.80	±1.42	26.9	±3.5

Table 3.3 illustrates the experimental torque as a percentage of the theoretical torque, along with accompanying errors calculations, obtained using the standard method. Most of the experimental data pertaining to the limbs of the robot appear to be around 50 percent of the theoretical values. Clearly two outliers are contained within Table 3.3, the right elbow, which was likely a consequence of human error, elaborated on later in the section, and the torso, which was 12.32 percent of the expected theoretical value. Exactly the same method was performed in taking measurement of the torso, nevertheless, this particular measurement had idiosyncrasies exclusive to itself. All other measurements involved just one motor, whereas the torso measurement involved two as explained in the previous section. These motors changed the *hip pitch*. It was assumed that since they were acting in tandem, in exactly the same direction, the resultant theoretical torques could be summed and would be twice the value of one of the *hip pitch* motors, however this was not the case. The *hip pitch* motors had exactly the same properties as those of the *knee pitch*, this was not reflected in the experimental data, where the torso generated around the same torque as a single knee movement, rather than twice as much, as anticipated. When the measurement was taken, the angle of the torso to the horizontal was quoted as 55°, however, visually, the angle looked much closer to 90°. The torso was affected by this more than the other measurements. If the angle was taken to be 90°, as it appeared visually, the torque obtained from this measurement would have been 5.85 Nm (see results from table 3.2), which would increase the percentage of theoretical data to 32.9 percent. This was still smaller than anticipated. The only explanation available is that either Choregraphe had an inbuilt limit on the force exerted by the robot at one time, meaning that a simple addition of torques of the motors is not reflective of reality. This may also be the reason for the experimental torque being 50 percent of the theoretical torque for the other joints.

Motor degradation may also be culpable for the lower experimental results. The NAO H25 robot central to

the investigation, at the time of measurements, was at least two years old. It had been used for two eleven week periods by the previous groups. As a result, the internal hardware may have deteriorated, leading to the large disparity between experimental and theoretical data. Additionally, overheating of joints commonly occurs in the robot, causing the readings given on the Newton meter to reduce, even if exactly the same measurement was repeated. A detailed experiment of how temperature of the joints affects the torque produced by the motors was not conducted for fear of damaging the robot. But a few measurements were taken illustrating a gradual decline in the torque produced. This hypothesis was supported by the investigation performed by the previous year's group [7].

An investigation to see whether altering the motor speed of a particular joint was conducted to determine whether increasing motor speed would result in larger torque values. However, no change was apparent, this was due to the fact that acceleration used in calculating instantaneous torque was always g and that the limb was static when the torque measurement was taken. From,

$$F = ma, \tag{3.5}$$

it is apparent that if mass is constant and acceleration is constant, the force, and by extension, the instantaneous torque will be constant irrespective of the speed at which the motor precesses.

Human error in reading measurements may have been a contributing factor in the difference between torque values. Obtaining the length measurements involved was straightforward. Joint to joint length measurements could be found on the Aldeberan website[12]; the pivot points where the Newton meter was placed for measurement were symmetrical ball and socket joints, meaning the length values found online could be taken to be accurate length measurements for the torque calculations. However, ensuring the readings taken on the Newton meter matched up to the angular position of the limb given on Choregraphe was difficult since the Newton meter reading had to be taken visually. In this process, a person had to hold a Newton meter in place against the opposing force generated by the motor of the robot. One of two thing would often happen, firstly, the robot would push the Newton meter out of position or, secondly, the robot itself would change position, forcing itself backwards. Naturally, if this happened, the measurement was retaken. Nevertheless, keeping the robot stiffened (a condition required for the robot to move) caused the joints of the robot to overheat, leading to a reduced weight measurement being produced on the Newton meter. Due to the regularity of these occurrences and limited time with the robot, waiting for the robot to cool down after a failed measurement was not practical. Furthermore, even if everything appeared static, the reading on the Newton meter would still sporadically fluctuate. This combined with the heating of the motor and the ageing of robot could be conducive to the smaller than anticipated experimental torque values obtained.

Ideally, if the time was available, waiting for joints to cool to the same initial temperature before every measurement would produce more accurate strength test results. Furthermore, if it was possible to measure the angle of the joints in unison with the mass measurements produced on a Newton meter, a more accurate experimental torque could be found. Moreover, though effort was made to align the position of the robot to the ground, so the torso was perpendicular to the ground and the thigh parallel, the angles given by Choregraphe were not the true angles that the joints were relative to the ground. Rather, the angles represented how far a motor had rotated relative to its own fixed point. For example, an assumption was made that when the robot was seated, the angle produced on Choregraphe was the true angle it was to the ground. This was only an approximation. If there was some way to calculate the angle offset, the angular measurement would increase in accuracy. Having a larger sample of measurements would have also proved beneficial in isolating and eliminating anomalous results, for example the right elbow measurement. If torque is used in future groups' code to run the robot, it would be prudent to repeat the strength tests to account for deterioration in the motors and make the improvements described.

3.4 Range of Motion

3.4.1 Method

Katherine Evans

The absolute values for the maximum angle ranges of all joints can be found in the specification [13], however the seat and handle used placed further limitations on these. To determine the actual extremes of positions that would be possible during any chosen motion, the robot was placed in a proposed position and the actuator values of all relevant joints were recorded.

First, a typical seated swinging position was considered with the handle of the swing in place. This would allow torso motion by rotating the hip pitch joints and varying arm joints as well as swinging of the legs from the knee joints. However in this arrangement the variation in torso position is severely limited by holding the robot's hands in place on the handle which ultimately gives less freedom of the swinging motions that could be trialled. To overcome some of the limitations on the motion the handle of the swing was removed. This is a perhaps an unrealistic swinging motion as the hands of the robot are not holding on to the swing at any point so it is not representative of human swinging, but seeing as the legs were already fastened to the seat for stability it was decided that this should be trialled regardless as replicating human motion was not the sole aim of the study.

Once the robot was connected to a computer with the Choregraphe software installed it was possible to save specific positions and read the actuator values of all the joints involved in these using the Memory Watcher panel. To do this the robot was physically moved into what were determined to be the maximum and minimum angles of joints in a given swinging motion and these were monitored and recorded. The key joints observed were the knees for the swinging of the legs and, as the upper legs were fixed to the swing seat, the hip pitch for the torso motion. For the positions with the handle of the swing in place the actuator values of the arm joints also had to be considered as the limits on these need to be observed to ensure there is no damage to the robot during movements. The maximum and minimum values obtained could then be used to within numerical models varying positions of different masses so that these more closely reflected possible motions.

3.4.2 Results

The results of motion tests with and without the handle are shown in tables 3.4 and 3.5 respectively. The joints included are those that vary with the motion and may cause damage to the robot if values are exceeded due to collision with parts of the swing or other body parts. For motion with the handle in place the hands were held fixed to the bar using cable ties and the values of the Wrist Yaw and Hand angles were set as constant by this positioning. The recorded values show some asymmetry between corresponding left and right joints which simply arose from the method of physically moving the robot into a position but these angles could easily be mirrored in applications at later stages. The angles recorded here are taken as upper limits to ensure safe motion and therefore magnitudes were always rounded up or down for the minimum and maximum respectively. For this reason errors on these values are insignificant.

Table 3.4: Limits of angles for motion with the handle in place on the swing. Maximum refers to extension (Definitions of angle direction are illustrated in the joint specifications [13].)

Joint	Min. Angle /rad	Max. Angle /rad
Hip Pitch	-1.01	-0.51
Knee Pitch	1.59	-0.09
R Shoulder Roll	-0.99	-0.10
L Shoulder Roll	1.01	0.11
R Elbow Yaw	0.34	0.05
L Elbow Yaw	-0.41	-0.08
R Elbow Roll	1.54	0.05
L Elbow Roll	-1.54	0.05

Table 3.5: Limits of angles for motion without the handle on the swing.

Joint	Min. Angle /rad	Max. Angle /rad
Hip Pitch	-1.18	-0.54
Knee Pitch	1.59	-0.09

The handle was at first removed in the hope that it would significantly increase the range of hip motion possible. However, after further tests it was found that if the robot was allowed to move to these extremes its centre of mass being too far forward caused it to rock off of the slots on the seat and, when moving, its momentum often resulted in collisions with the back of the seat resulting in unstable motion. The results for the hip pitch given in Table 3.5 are the adjusted positions to prevent instabilities on the swing, hence why the range does not appear significantly larger than that with the handle. It was still beneficial to remove the handle from the swing to reduce the number of joints that needed to be considered when trialling motions, as with the handle in place all the arm joints must also be controlled. To ensure stability, which the handle previously provided, the robot was attached to a harness whilst on the swing so that if it became detached from the seat it would not be damaged.

3.5 Swing Designs

The swing used in this investigation was provided from the previous year's work [7], designed for a robot executing seated motion.

3.5.1 Swing measurements

Adam Szekely

In order to build a virtual swing to model the robot code on Webots, it was necessary to measure the masses and lengths of each individual moving component of the swing. The rods of the physical swing could either be locked into place so that the swing, in essence, behaved liked a pendulum. Alternatively, by removing screws, the swing could be made to behave like a triple pendulum. The motivation behind this style of construction was to try to mimic human hand placement on a chain swing. When a person places their hands on a swing, the length of chain gripped by the person effectively behaves as if it is rigid. Having three separate rods goes some way to being able to model this scenario and, at the same time, avoids the complicated and mathematically chaotic enterprise of trying to model the motion of a chain swing.

Ostensibly, taking swing measurements seemed to be a simple task. This was the case for the solid version of the swing, where the only axis of rotation was where the pendulum was attached to the wall. The process involved dismantling the swing and then weighing both the rods and seat. However, when considering the swing

with three smaller independently moving rods, the weighing of components became difficult due to sections of the components contributing mass to separate moving parts of the rod. This is illustrated in Figure 3.3.

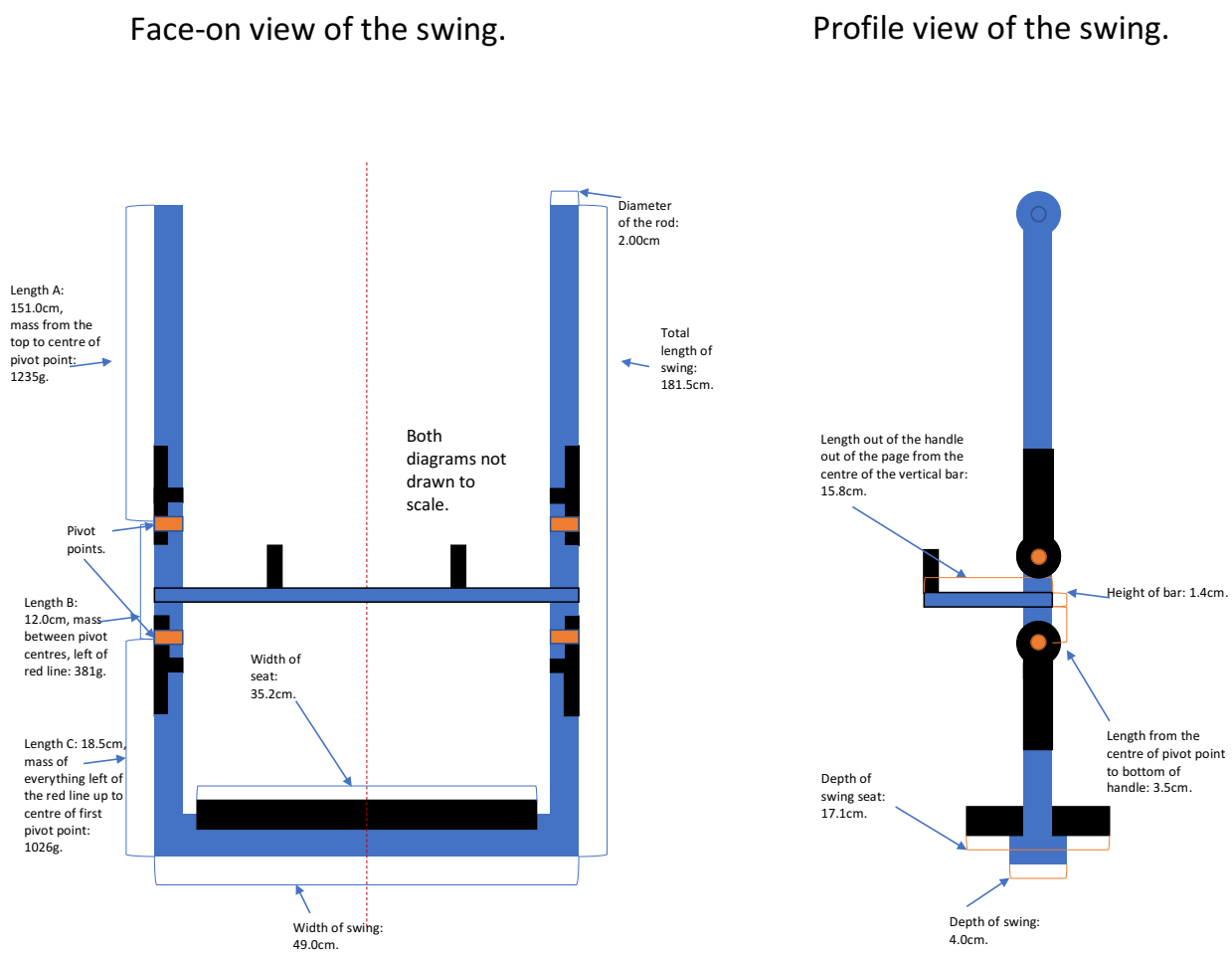


Figure 3.3: A figure displaying both the face on view and profile view of the swing the robot is seated upon in two dimensions, with pertinent mass and length measurements labelled.

In order to model the triple pendulum version of the swing on Webots, it was necessary to obtain the masses for lengths, A, B and C, labelled in Figure 3.3. An ingenious method was suggested by project supervisor, Mr Ross Griffin, to perform this task: Archimedes' Principle. After dismantling the swing, all of the components were weighed. Afterwards, one of the identical black components, pictured in Figure 3.3, was fully submerged in a measuring cylinder filled with water to discover how much water it displaced. Following this, the same component was partially submerged so that only half the circular end of the black component was in water. By assuming the component was of uniform density, the mass contribution to each length could be found by multiplying the fraction of the water displaced by the mass of the whole component. This process was repeated for the silver (blue in Figure 3.3) length between the black components. The respective masses found were summed together and thus the masses of lengths A, B and C were identified. Note that half the handle weight was added to Length B and that half the seating was added to Length C. The data is displayed below in Table 3.6.

In Table 3.6, the total mass of the swing is the combined mass of all of the components doubled, as there was a twin length parallel to each rod. These measurements were used in building the swing in Webots - other length measurements can be found in Figure 3.3.

Table 3.6: A table illustrating the real lengths and mass data required for building a virtual swing in Webots.

Length name	Total length of component (m)	Error on length (m)	Mass (kg)	Mass error (kg)
A	1.510	± 0.005	1.235	± 0.001
B	0.120	± 0.005	0.381	± 0.001
C	0.185	± 0.005	1.026	± 0.001
Total swing	1.810	± 0.015	5.293	± 0.006

3.5.2 Damping Constant

Elise Dixon

The damping constant and natural frequency of the system are parameters useful in producing models for optimum movement as well as implementing simulations to find estimates of the amplitude of swinging in a steady state. These constants were found experimentally by watching the oscillation and decay of the swinging angle when released from an initial amplitude.

3.5.2.1 Method

The angle encoder was set up to record data once released from an angle, while supplied with a voltage of approximately 6.6V. The resolution of the encoder's output is dependent on the supplied voltage, thus the errors can be taken as the minimum resolution found between two values, which in this case is 0.2°. The robot was cable tied onto the swing and stiffened, so it would not move and affect the swing's motion. The robot was moved into a neutral position with his centre of mass remaining above the seat, to maintain stability. The swing was then released from an initial angle of (13.9±0.2)°, and allowed to decay for 400s, as this was found to produce a significant decay for analysis. This data was fit using the equation for a damped harmonic oscillator:

$$\theta = Ae^{-\lambda t}\cos(\omega t + \phi) \tag{3.6}$$

The results were plot using MATLAB's *cftool* data fitting tool, and the values found for the parameters are shown in table 3.7 below. No uncertainty values for ω were produced by the fit, so the error is given as the precision of the parameter output.

3.5.2.2 Results

Table 3.7: Table of fitting parameters provided by *cftool* for the fitting of Equation 3.6 .

Parameter	Value
A	(13.4±0.2) °
λ	(4.43±0.02) x10^{-3} s^{-1}
ω_n	(2.453±0.001) rad s^{-1}
φ	(0.127±0.002)rad

The time period of oscillations is therefore found to be (2.561±0.001) s. From Figure 3.4, it can be seen that the damping seems to fit sufficiently well with the data, however the parameter of the amplitude is inconsistent with the known amplitude taken directly from the maximum theta in the data. For this reason, further investigation was done to check if air resistance was having an effect on the results.

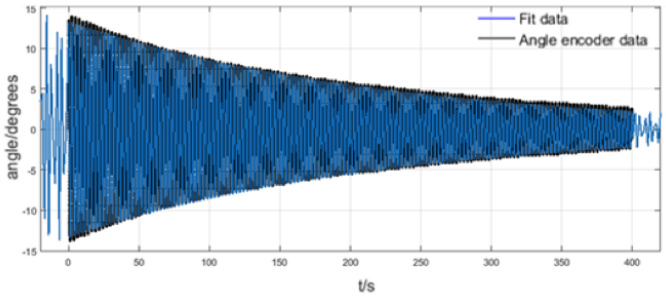


Figure 3.4: The output of *cftool* fitting tool, fitting equation 1 with the data produced by the angle encoder while swinging with the robot on the seat.

3.5.2.3 Air Resistance Considerations

To investigate the effect of air resistance, a MATLAB program was written to numerically compare a decay where the equations of motion contained an air resistance term, to a decay where air resistance is neglected. The oscillations are produced using the following equations for the angular acceleration, angular velocity, and angle respectively:

$$\alpha(t+\delta t) = \frac{-k\theta(t) - c\omega(t) - r\omega^2(t)\hat{\omega}}{m} \tag{3.7}$$

$$\omega(t+\delta t) = \omega(t) + \alpha(t)\delta t \tag{3.8}$$

$$\theta(t+\delta t) = \theta(t) + \omega(t)\delta t \tag{3.9}$$

where the constants are given by:

$$k = \omega_n^2 m, \tag{3.10}$$

the system's "spring constant,"

$$c = \frac{-\lambda}{2m}, \tag{3.11}$$

the damping coefficient,

$$r = \frac{\rho C_D A}{2}, \tag{3.12}$$

the coefficient related to air resistance,[14] where ρ is the density of the air, C_D is the drag coefficient, A is the surface area of the object which pushes against the air, m is the mass of the swing and robot combined, and ω_n is the natural frequency of the system.

A rough estimation of r was made by assuming ρ=1.225 kgm^{-3}, the density of air at sea level. [15] C_D was estimated to be similar to the drag coefficient of a human standing, approximately 1. By approximating the robot's surface area as half the area of the rectangle enclosing its dimensions in any given position, the area A was estimated as half of the robot's height multiplied by its width, [16] multiplied by 0.5. This produced r=0.0245kgm^{-1}.

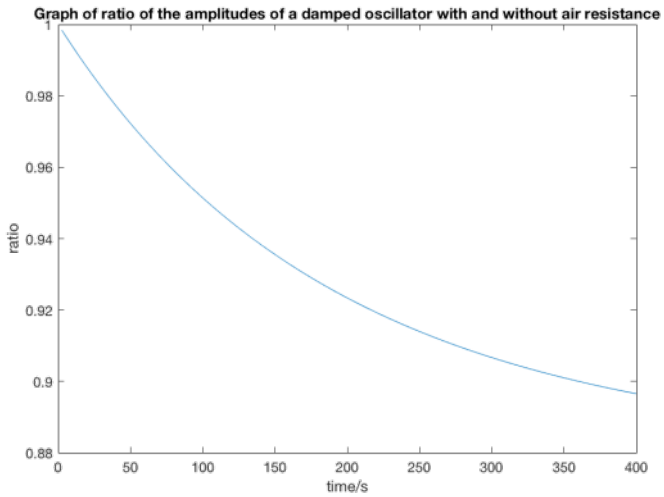


Figure 3.5: Graph of the ratio comparing the amplitudes of two numerically produced oscillations with and without air resistance. The peaks of the oscillations were found using MATLAB's *findpeaks* function. k and c were found using fitting parameters shown in Table 3.7, and r was estimated as described above.

This numerically produced oscillation was compared to another, which neglected the air resistance term from the equation of motion. The ratio of the peaks of these two signals was plotted against time, to see what effect the air resistance term had on the decay. It was found that after 400s, the ratio was reduced to 0.8965, showing that there is an order of 10% difference in the amplitude of oscillations over a reasonable time period. Since the change produced by an air resistance effect is small, and cannot be simulated in Webots, it was decided to neglect any further consideration.

3.5.2.4 Conclusion

The swing with the robot on was found to have a natural time period of (2.561±0.001)s, and a damping decay constant of $(4.43 \pm 0.02) \times 10^{-3} s^{-1}$. Air resistance was considered to improve the decay model; however, this was found to decrease the amplitude by only 10% over 400s given the estimations made in the drag coefficients. Small disparities in the natural frequency of the system will arise depending on the position of the robot, as altering the position and thus the centre of mass will affect the effective length of the "pendulum" system and thus the time period. It was also considered that some energy may be lost due to the flex of the main swing axis, as there was a component of rotation around the vertical axis of the swing.

3.6 Sensor Investigations

Katherine Evans

3.6.1 Introduction

A significant aim of the investigation was to achieve swinging motion independent of external sensors, instead utilising those available to the NAO itself such as its inertial and visual sensors. Before feedback from these internal sensors could be implemented into a motion code the capabilities of different sensors and methods to record values needed to be investigated.

3.6.2 Inertial Sensors

The inertial unit of the NAO consists of a two axis gyrometer and a digital 8-bit, three axis accelerometer [17] located within the robot's torso [18]. Figure 3.6 indicates the defined axes and the gyrometer and accelerometer sensors return the acceleration for, and rotation about, these axes respectively. These devices have precisions quoted as 5% with an angular speed of $\sim 500°/s$ for the gyrometer, and 1% precision with an acceleration of $\sim$ 2g for the accelerometer where g is the acceleration due to gravity [18].

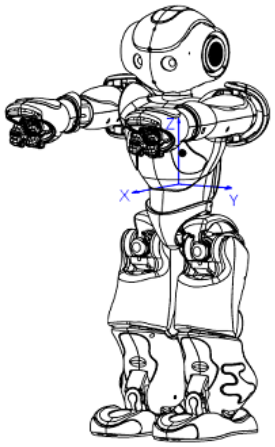


Figure 3.6: Axes for inertial unit [18]

Within Choregraphe it is simple to monitor and record sensor values via the memory watcher panel. However, sensor values recorded using Choregraphe could not be recorded alongside the angle of the swing and therefore a simple python script to record the values of these sensors was written which also included the angle of the swing from the angle encoder. To do this ALMemory was used to retrieve values from the robot for each sensor by creating a proxy and selecting the relevant memory key,

```
Value= ALProxy("ALMemory",robot_ip,robot_port) .getData(memory_key).
```

This method returned raw data for the memory keys used and, by comparison to that taken via Choregraphe and realistic expectations of the angular velocity of the swing, it is clear that some calibration is needed to return a real value for the accelerometer and gyrometer as expected in m/s^2 and rad/s respectively. As it was only the periodic behaviour of the sensors rather than an absolute value for acceleration or angular velocity needed to monitor swing position, these scalings and offsets were ignored at this stage.

3.6.2.1 Gyrometer Results

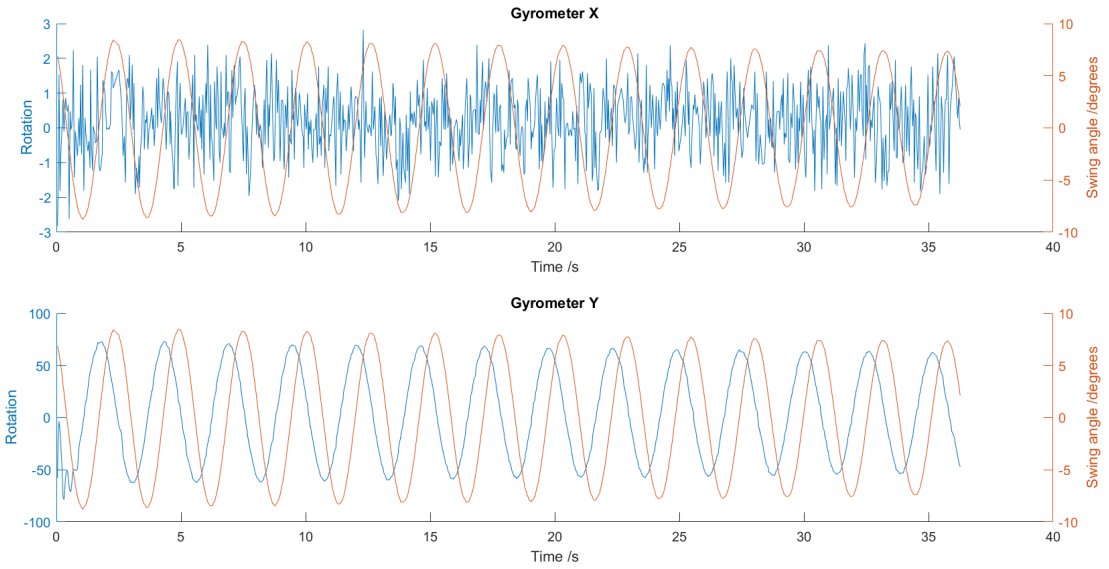


Figure 3.7: Raw gyrometer values and swing angle against time for swing motion

Figure 3.7 shows the results for the x and y gyrometers readings of the swing motion plotted alongside the swing angle. The periodicity of the y gyrometer is clearly the same as that of the swing but $\frac{\pi}{2}$ out of phase as expected as it is reading the angular velocity of the swing. This evidently could be used to determine the swing velocity and therefore the position within the cycle. Gyrometer x gives readings of the swing motion from side-to- side which is not useful for determining information about the typical swinging motion. However, as swing flex from left to right when the robot switches positions is later discussed as a potential reason for energy loss and deviation from simulation results this could perhaps be further investigated utilising this gyrometer to see the extent of this motion.

3.6.2.2 Accelerometer Results

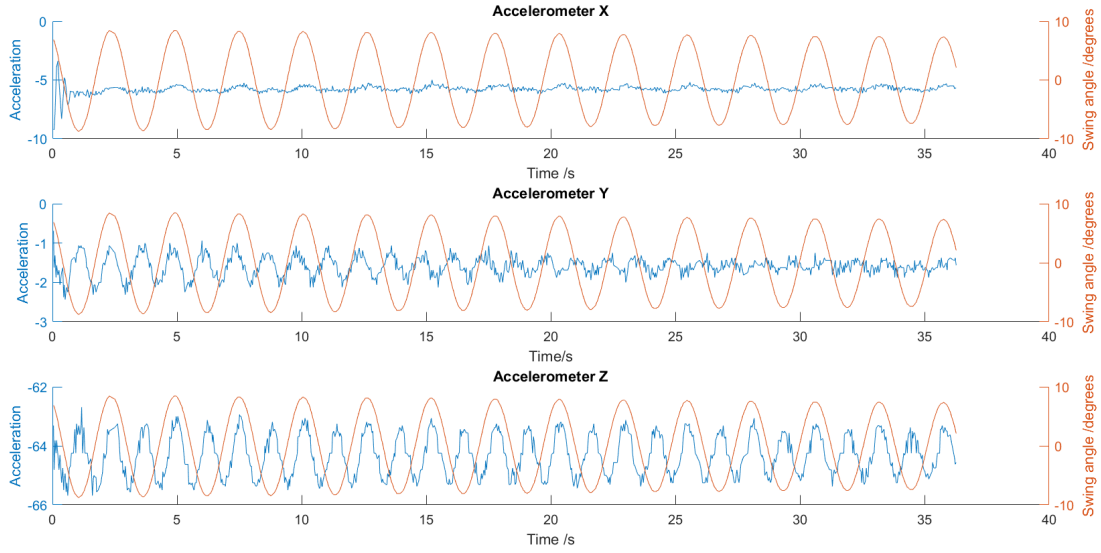


Figure 3.8: Raw accelerometer values and swing angle against time for swing motion

Figure 3.8 shows the results for the x, y and z accelerometers plotted alongside the swing angle. All three accelerometers do exhibit periodic nature, even accelerometer x although the variations are very small and hard to distinguish above noise. The y and z accelerometer also appear noisy, so although their periodicities mean that they show potential for instructing when the robot should change position, they do not seem as reliable for implementation as gyrometer y.

3.6.2.3 Consideration of Effect of Torso Motion

Torso motion was vital to obtaining the maximum torque to increase the amplitude of the swing within this investigation. Due to the fact the inertial unit is located in the centre of the NAO's torso, this motion would clearly affect its gyrometer and accelerometer readings. It was therefore necessary to begin investigations into how the swing motion could be isolated from the readings of the chosen sensor: gyrometer y.

The torso moved through variation of the hip pitch due to the fact the upper legs of the robot were fixed to the seat. Gyrometer y readings were therefore expected to consist of a sum of contributions from both the angular velocity of the swing and the hip rotational velocity. To investigate the extent of the effect of the hip angle variation on the gyrometer readings the swing seat was removed and clamped to a desk. The robot was then placed in the usual seated position on this and code providing smooth periodic motion of its hip joints was run. The hip pitch, gyrometer y and time were all recorded throughout the motion. The variation in hip angle was set to be sinusoidal between the maximum and minimum angular positions previously determined with the swing seat limitations.The rate of variation of hip pitch angle, referred to as the hip velocity, was simply calculated from the change in hip pitch over time and fitting this, using MATLAB's curve fitting tool, to the differential of the expected hip pitch variation is shown in figure 3.9. Clearly the so called 'hip velocity' is actually the velocity with which the torso is moving due to hip pitch variation but to avoid any confusion with overall torso motion, including that due to the swing in later readings, this was always referred to as a simply hip velocity to keep it clear that the contribution arises from the hip motion.

As the robot was sat on a desk in these tests the gyrometer reading should only be affected by the hip velocity. Therefore the fit from the hip velocity, with some scaling factor, was fitted to the gyrometer readings and the results of this are shown in figure 3.10. It can be seen that the hip velocity has the opposite sign to the raw gyrometer value and needs to be scaled by a factor of roughly 200. This gave a good indication for the

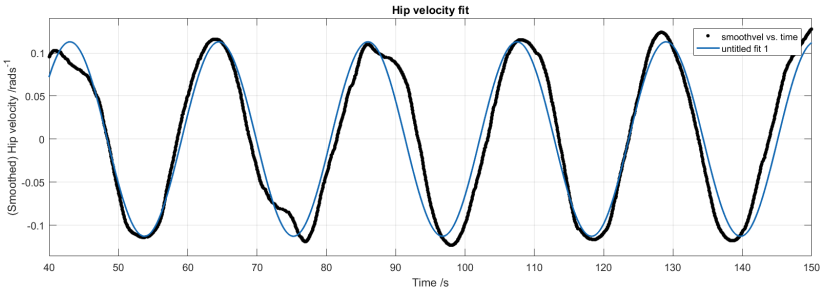


Figure 3.9: Fitting hip velocity. (f(x) = a*cos(k*x+c) with a=0.113 ±0.002, k= 0.2923 ± 0.0004 and c=-0.002138 ± 0.04)

feasibility of removing the torso motion affects from the total gyrometer readings through the addition of the gyrometer value and the scaled hip velocity at any point.

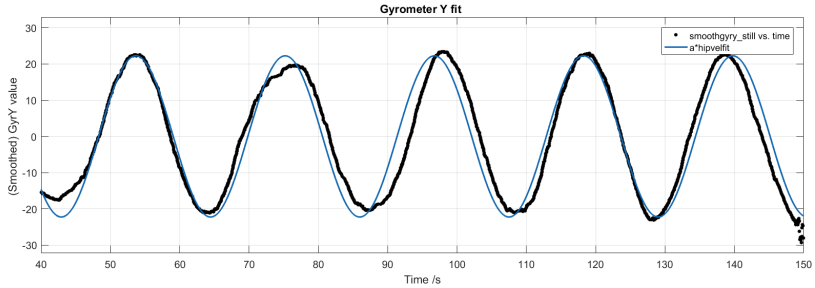


Figure 3.10: Fitting scaled hip velocity fit to gyrometer y data. b=-200 with a*cos(k*x+c) from previous hip velocity fit.

The trialled motion at this stage had a significantly longer period than that of the swing so it was easy to disentangle the swing motion from plots of this data with the robot executing the motion whilst on the swing. However early models showed that an effective swing motion would be one that involved the robot changing position twice per period so any further investigation of this motion would not be particularly applicable at a later stage where the tested motions were expected to lead to the swing motion being harder to distinguish.

3.6.3 Visual Sensors

Elise Dixon

Investigation into the vision sensors was conducted in part due to the promising results of vision feedback swinging presented in the preceding report. [19] It is also noted that a human would possess a combination of sensory feedback in order to make decisions about their swinging motion. A human has a sense of balance, analogous to the inertial sensors of the robot, as well as vision.

3.6.3.1 Camera Properties

The robot has two video cameras in its head, one situated in order to capture a field of view almost parallel to the horizontal axis of the head, and one angled downwards in order to capture objects and obstacles below the head. In this instance, use of the first camera will be investigated primarily, however the two cameras are identical so results can be applied to the angled camera given appropriate positioning of the head.

Both cameras have an angular field of view of 47.64°about the y axis and 60.97°about the z axis, as shown

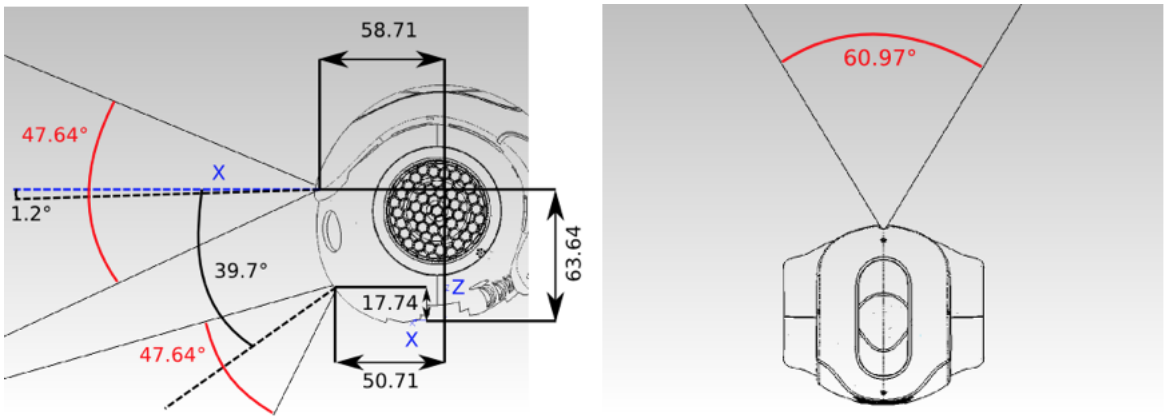


Figure 3.11: Positions and angular field of views of the two cameras relative to the centre bottom of the head and angular field of views of both cameras from above [20]

in 3.11. They have a frame rate of 30 frames per second, a resolution of up to 1280x960, and a focus range of 30cm to infinity. [20]

3.6.3.2 Vision API

The cameras are capable of a variety of operations including recording videos, detecting, and recognising objects. Further details of the camera's functions can be found on the Aldebaran website. [21] Informed by tests completed in last year's report concerning the suitability of vision detection methods, the vision module chosen for following investigations was [19] ALLandMarkDetection. This allows the robot to recognise particular patterned landmarks (Naomarks) made available on the Aldebaran website, an example of which is shown in Figure 3.12 . Once a Naomark is detected, variables are created containing the time stamp of the image detection, the size of the Naomark in the x and y axis, the position, and the orientation of the Naomark about the vertical axis. All positions and sizes are measured in terms of camera angles, output in radians. The API can also be used to distinguish between marks using the Mark ID.

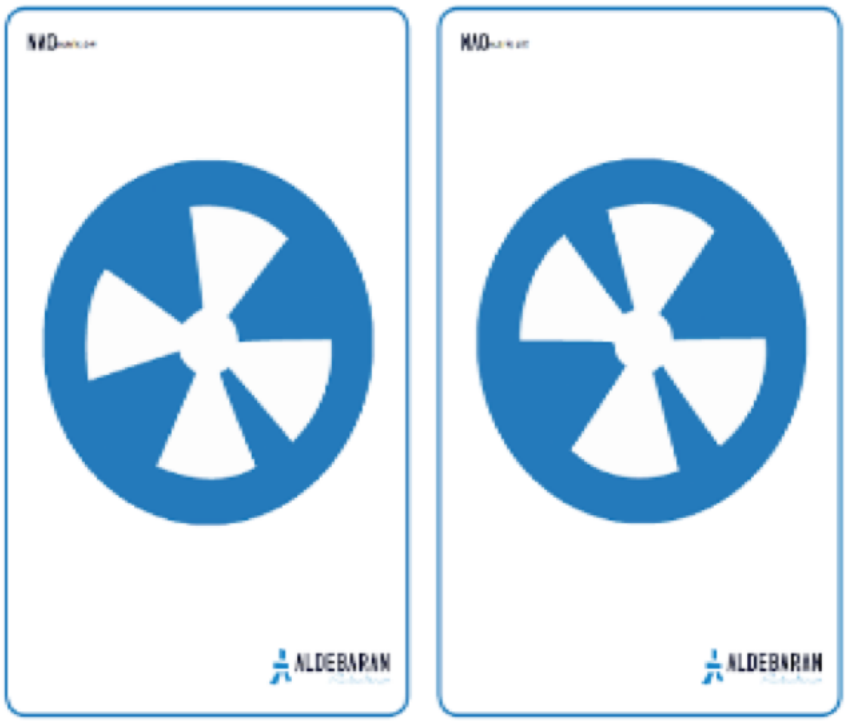


Figure 3.12: An example of a Naomark, with the ID numbers of 64 and 68. The robot is able to detect this landmark, its ID, and quantities relating to its position in the field of view. [21]

4 Human Motion

4.1 Human Motion on the Robot

4.1.1 Fieldwork and Data Analysis

Adam Szekely

As the original objective of the project was to get the robot to swing using human motion, a day was taken to go to a local park to acquire footage documenting how the human body could generate a swinging motion. The swing used was situated in Selly Park. A number of items were brought to the park in preparation for filming the swinging. The essential inventory used included a functioning camera (an iPhone 6), paper, Sellotape, a laboratory book, a clamp with an accompanying stand, 22 mm diameter Climafex Pipe Insulation (bought at Homebase) and a heavy backpack.

The swing lengths were predominantly made of solid rods but had small lengths of chain attaching the seat to the rods. In an attempt to limit the effect of the chains on the swinging motion, the rod lengths and attached chains were enveloped in pipe insulation, which were secured with Sellotape, to make the swing behave more rigidly. The phone was placed in the clamp attached to the stand, which was positioned as far back as possible to try to ensure that both swing and person remained in view for the duration of the recordings. Naturally, the phone was positioned so it was directly facing the swing and parallel to the floor. Paper was stuck to joints of interest of the person so that *autotracking*, discussed later in the section, could be employed when analysing the human motion. Five videos were made: the first video involved swinging normally from a displaced position; the second, swinging from a stationary position at equilibrium; the third, swinging with a backpack from a displaced position; the fourth, swinging from a stationary position at equilibrium with a backpack and the fifth,

swinging whilst standing on the swing.

The motivation for exploring different variations of swinging was that multiple models pertaining to different modes of swinging were created. For instance, some models involved 'self-start' and others assumed an external initial force. Wearing a backpack was, upon reflection, counter productive as it made the weight distribution of a person further diverge from that of the robot. The largest variation in weight was in the torso. For the NAO H25 robot, the torso made up nearly twenty percent of its mass [8], by contrast, for the average female human body, the torso comprises around fifty percent of total body mass [22]. As a consequence, it was questionable whether human motion when swinging, if used by the robot, would aid it in swinging. In spite of these reservations, the investigation was pursued.

The footage was uploaded on to a computer. Free to download software named *Tracker* was used in analysing the videos captured [23]. A version of the software was available to use for Macintosh, Windows and Linux. A set of axes and *calibration stick* could be placed on every video so that position and distance could be gauged. Following this, the *point mass* was used so that the software created a template image and autotracked the template, attempting to find a match for every individual video frame. Often the autotracking would fail, there were a multitude of reasons for this happening. Firstly, the view of the template image, when at the extrema of the swing periods, was impeded by the physical frame of the swing. Secondly, the non-uniform lighting meant that images that ostensibly looked alike had a low percentage similarity, preventing a match from being found. This meant that manual tracking had to be used, which was tedious, time consuming and less accurate. For each video, point masses were tracked on the foot, knee, hip and shoulder of the subject. The software could extract an assortment of information but for the purposes of the project, only the angle, θ (the angle of the point mass relative to one of the axes), and the x and y displacements from the axes were of interest. Figure 4.1, illustrates both the x and y displacements with respect to time for all point masses in the video, *swingingfromdisplacedposition.trk* (see accompanying videos).

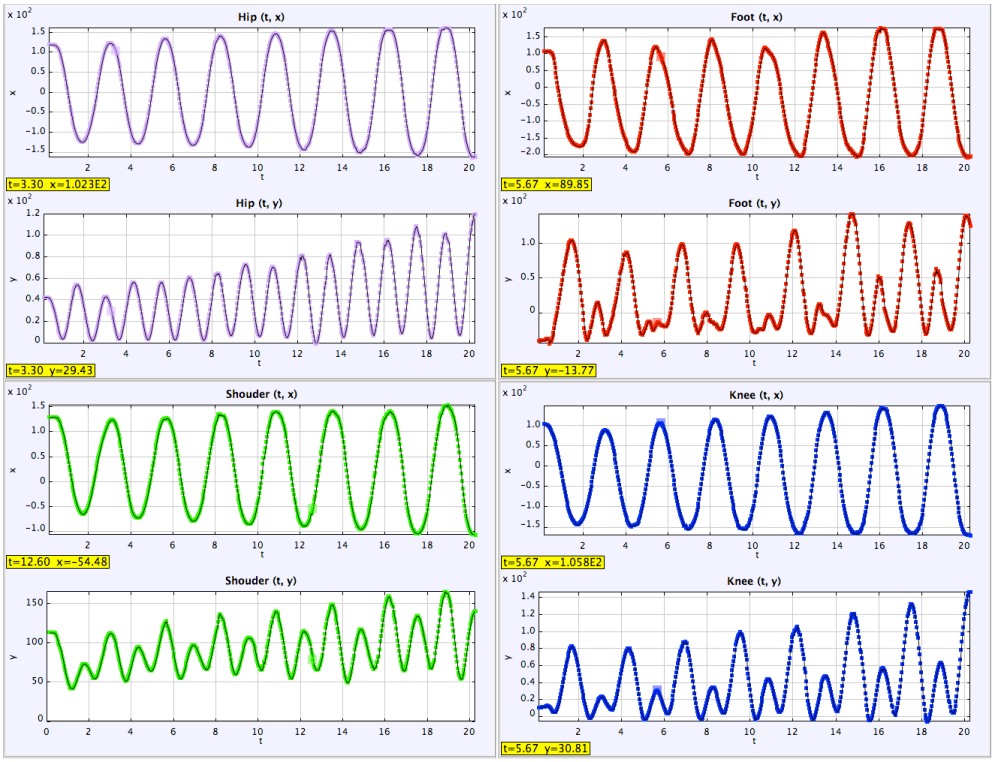


Figure 4.1: Plots made using *Tracker* to show the position of key human joints when swinging from a displaced position, where the set of axes were centred on the position of the swing when at equilibrium. Note that displacement is in units of centimetres and time is in units of seconds.

Figure 4.2: Method for obtaining the angles from the video.

Clearly, the displacement in the x-direction for all the joints follows a periodic sinusoidal motion with a marginally increasing amplitude to account for the slight increase in motion generated by human movement. The increase in amplitude is more obvious in the plots mapping y-displacement with respect to time. The human movements which were the catalysts for range of motion can be seen in the y-plots in Figure 4.1. It was optimum that at the extrema of every period, for the person to raise their centre of mass as far from the ground as possible to harness maximum gravitational potential energy in order to increase the amplitude of motion. This can be seen clearly in the y-plots, titled knee and foot in Figure 4.1. The larger peaks correspond to when the feet and knees of the person have travelled as far forward as possible - with raised outstretched legs. The smaller peaks describe the other extremum, where the person if displaced as far back as possible - where the feet and knees are folded back to raise the position of the centre of mass in the y-direction. This is the reason for every alternating peak possessing similar characteristics as opposed to all the peaks being similar. The shoulder, by contrast represents the position of the torso and the head. The torso behaves in an opposite fashion to the legs, as it effectively acts as a counterbalance, ensuring stability of the person on the swing. Ideally, to gain maximum amplitude, the torso would also be raised at the extrema of the periods of motion. However, due to the need maintain balance, the y-displacement from the ground is actually reduced to prevent the human from being thrown from the swing. The hip plot is, in essence, synonymous with the movement of the swing so cannot change independently of the motion of the swing.

It was possible to fix the set of axes on a video to a point mass so that the angular position of one mass could be found in the reference frame of a secondary mass. Using this method, the angles of the torso and the feet, relative to thigh, could be found. The method will be elaborated upon in the following section.

4.1.2 Data extraction and implementation of human motion onto Nao H25

Thomas Smith

After the videos were taken and the joints were tracked, they needed to be analysed and the angle of the knee pitch and the hip pitch needed to be found and converted into a format that can be read by the robot. The easiest way to find the angles for knee pitch, was to centre the axis at the knee or hip and measure the angles between the x axis and the other points. To get the angle for the knee pitch, labelled B in Figure 4.2, the angles from the x axis to the foot and angle between the x axis to the hip, which are the 2 blue angles on the right, are summed. To get the angle for the hip pitch, the axis was centred on the hip and the angles for the knee to the axis and the shoulder and the axis was found, which are the two blue angles on the left of Figure 4.2, and their sum is equal to minus the hip pitch which is the angle labelled A. From this, a list of angles of hip pitch and knee pitch for each frame was gotten and was saved onto a text file.

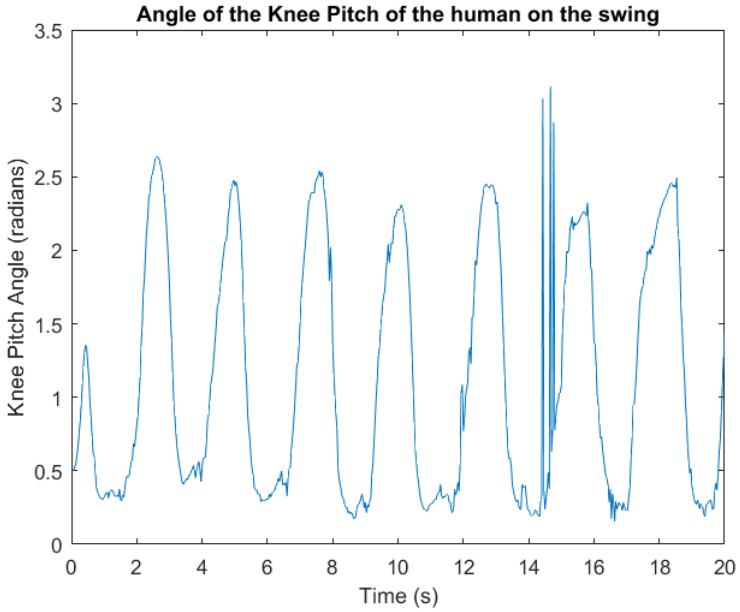


Figure 4.3: Plot of the Knee Pitch against time from the human swinging.

The plots of these angles, plotted against time, are shown in Figures with Figure 4.3 being the angle of the knee pitch and Figure 4.4 being the angle of the hip pitch both plotted against time. In Figure 4.3 there are a few anomalous results that do not match the fix the curve. These anomalous results were most likely from tracking software because the for those frames the tracking software probably thought that the markers were in different positions to where they were. These results could not just be ignored because this method of swinging is time based and removing point would cause the robot to start moving out of time and would cause the legs to move slightly before the back. To account for this the three anomalous data points were replaced by three point that fit the curve.

A piece of code was written that would read the angles consecutively. To do this, the data from the file needed to be converted from a string to a float. This was done by using the function rstrip('\n'). This was needed to be used because there is a return between each reading and it is impossible to convert a string into a float when there is a return function in it. As well as getting rid of the returns between each result there needed to be a wait function between each reading. The amount of time that the program waits needs so that one oscillation is equal to half the natural period of the swing. The wait time is given by,

$$T = \frac{Pt}{2p} \tag{4.1}$$

where T is the time between each frame P is the natural period of the swing, p is the period of oscillation from the video and t is the time of each frame of the video. This causes problems with natural frequency of the swing, which is needed to get the robot to swing properly. This means that, as it is it cannot be easily used on any swing unless you calculate the natural frequency of the swing.

If the robot just read off all the angles consecutively, only once, the robot would swing for a couple of seconds. To make the robot able to swing forever the code would need to loop forever. To do this, the robot could have just read the whole file again but the start and the end are out of phase. Instead of this the last oscillation was taken and saved into separate files, one for the knee pitch and one for the hip pitch. They were then adjusted so that they had the same amount of data points each. Then the number of data points in each file was summed and then half the natural period of the swing was divided by the number of points to get the time that the program need to wait between each movement to get parametric resonance.

Three different types of swinging were looked at. These were displaced sitting, sitting from equilibrium and

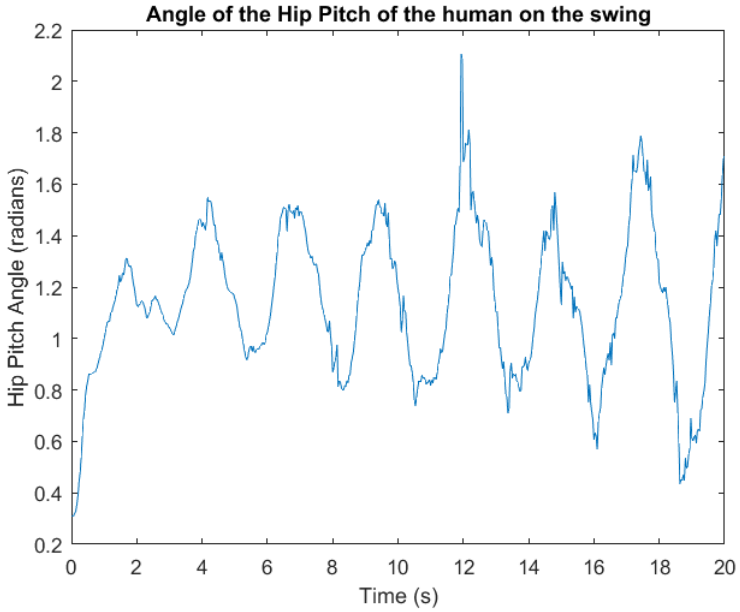


Figure 4.4: Plot of the Hip Pitch against Time from the human swinging.

standing from displaced. The angles for the knee pitch and hip pitch were able to be calculate for each frame of the video. With these angles the robot could be moved to simulate the human swinging. The two important types of swinging that were used were from the two videos of the person sitting while swinging. This is because the real robot was only used while sitting down.

The data from the video of the person standing was the most complicated to analyse. In the standing video, unlike the others, the ankles also had to be changed, as it moved, so it does not fall. As well as this, the data from the standing video was incomplete because there was no tracking on the shoulder. It was not possible to track the shoulder because it went out of the cameras view. So even if the video was gone through frame by frame, moving the point mass to the shoulder for each frame it was impossible to obtain a track of the shoulder. This meant that the angle for the hip pitch was not obtained. To get around this, the back was made to be always 90 °to the seat. To do this, the hip pitch was set to minus half the value of the knee pitch and ankle pitch to half the value of the knee pitch. This made the code for this simpler than the other 2 types of swinging because the code only needed to read form one file.

When implementing human motion onto Webots, the max angle that the swing reaches was needed to be obtained. Initially the virtual angle encoder was going to read the angle of the swing the same way the real angle encoder does. Unfortunately, it was not usable in the end because it would mess up the timing of the program and would make it so the movements were not at the right time. This is because the computer took a considerable amount of time to write the values to a file, compared to the time that it needs to wait between each reading. It might have been possible to change the timing of the program to compensate for the extra time needed to obtain angle but this would have required trial and error. If the time taken to write the data was more than the wait time, then it would have been impossible. Instead a high definition video of the robot swinging in Webots was taken from the side of the swing. Then Tracker was used to get the angle by placing the origin on the top of the swing and tracking the seat of the swing. The problem with this method of obtaining the angle was that it took a lot of time to obtain the data because you need to go through the video frame by frame to make sure that the point mass stays on the seat, adjusting it when it moves off the seat, this also limited the amount of time that the robot could record because, as the video gets longer, the amount of time taken for the program to do each frame increases. It is also not as accurate as the angle encoder because it cannot track the exact same point in each frame. This causes fluctuation. This meant that something that tracker can track, needed to be added to the swing or the robot. It was decided to add blue dot without physics to the side of

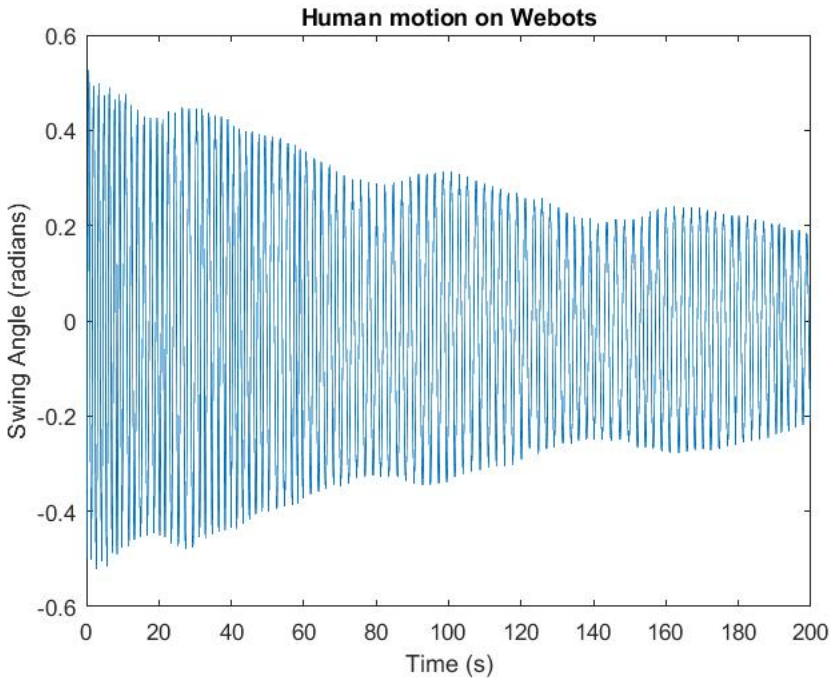


Figure 4.5: First 200 seconds of the human motion on the robot from Webots.

the swing. It needed to have physics off so that it does not have and mass, which would have cause the natural frequency of the swing to be changed, and it doesn't interact with the robot or the swing.

Unfortunately, when testing the code on Webots the amplitudes that the robot reaches are nowhere near what was expected. Initially it was thought that the timing of the code was off because the time taken for the program to read from the file was not considered. The time taken for the code to read off and move the robot was taken and subtracted from the wait time but this didn't improve the amplitude that much. The amplitude of the swinging while sitting from a displaced position reaches of 4 °after 10 minutes and the one function from sitting only reaches a maximum of 2 °. Figure 4.5 shows the first 200 seconds of the swinging. These values are much less than that of the human that the motion was taken from. The standing model seemed to be better than the sitting ones, with it being able to maintain an amplitude of around 20 °. Unfortunately, this is because the world that was used in Webots was different because connectors were able to be used to attach the robot to the swing as well as this the damping on the swing was an order of magnitude less so it amplitude decays slower.

The reason for the difference between the human swinging and the robot swinging is because the robot and the person have different weight distribution. The robot has more of its mass in its legs while a human will have more of its mass in its chest, this means that the human would not be moving in the best way for the robot. As well as the weight distribution being different, the swing that was used was not the same as the one the robot swings on. This is because the robot used a solid swing and it was hard to find a solid swing, to try and simulate this tubing was placed on a normal swing. This will cause the swings to act slightly differently. The reason that the robot was not able to swing optimally was that when the videos were taken the person swinging was just swinging without thinking about it, they were not trying to get optimal motion, instead they were just swinging like a normal human would. This was one reason that the robot was not able to swing as well as the other models. If this part was done again, it might be a good idea to have the human on the swing to have weights on its legs to make the human a closer model of the robot, as well as getting the swing that is used to be as close to the one the robot is in as possible. This is why the swinging of the robot from the human motion was not optimal.

If the code was converted to be used on the real robot, changes to the timing in the code and the IP address that it thinks the robot is at, would also need to be. In addition to this, the range of angle the robot is able to go would need to be changed. The timing of the code would need to be changed because the virtual swing and the real swing have different natural frequencies. The IP address needed to be changed so it is actually sending it to the robot instead of Webots. The range of angles would be needed to be changed because the real robot and the one in Webots have different ranges while seated. This is because the real robot has the seat that it needed to stay in, and when the back moved to far back the robot can comes out of the seat and runs the risk of falling off the swing. As well as this, the seat stops the legs from being bent back as far as Webots will allow. It was decided against testing this model on the real robot because time was an issue and there were multiple other models that where more successful that needed to be tested.

A problem that was faced is that the bar used to keep the robot in place, that was in the legs, was pushing hard against the robot which would sometimes push the robot into the seat, causing the robot to stop responding. This was only a problem with the Webots model and was not a problem with the real robot. There was another problem that the robot had. While execution the human motion functions the robot would sometimes unstiffen and stop moving. This would normally happen after around 20 seconds which made it hard to get a value for the maximum amplitude because it would stop working before it reached its maximum. To get around this, the code needed to re-stiffen the robot once it has unstiffened. The best way to do this is to have it stiffen the joints at the start of the loop.

4.2 Self-Start

Thomas Smith

One thing that the robot was not very good at, was starting its self. If you tried to start the swinging on its own, by only swinging backwards and forwards, it takes some time to reach a respectable amplitude. It would be more useful to us if it could quickly reach an amplitude because it would allow for more test and models to be ran on the robot because there would be less time waiting for it to reach the initial amplitude. If the robot is unable to start its self, it is more reliant on humans. This means that one way that the robot was made more self-reliant is looking into self-starting. It is also a way that the robot can be made more human like because a human is able to start swinging on its own.

There are a few possible ways that the robot could start its self. The first and the simplest way would be for the robot to rotate backwards and forwards generating torque and using that to swing. This method of self-start could easily be simulated as something like a dumbbell on a pendulum. This method is good because it could be done without any help from humans or other pieces of equipment but it takes much longer than the other methods. In addition, this method is not how humans start swinging.

The second method of self-start that was looked at was to kick off a block to start swinging. This was a good way to self-start but it still requires some human interaction because someone needs to move the block away after it kicks off otherwise the robot will crash into the block when it swings back. Also a block is needed, which makes it require additional equipment, which makes it less self-reliant. This is still reliant on something besides the swing and it still required human interaction so at this point the robot might as well just be pulled back by a person. From this method the robot only reached an amplitude, on Webots, of about 7 °. Because of the small amplitude that was reached and it still requires human interaction and the extra equipment needed, there is little point in using this method, it was much easier and better to push it to start it instead of using this method.

The ideal self-start would not require anything done by a person or any extra equipment. The closest thing to this, that was considered, was to have the robot to walk its self-back while sitting on the seat and then releasing when it gets to a certain height like a human would. The problem with this is that the robot only has short legs so the robot can't reach the floor of most swings so it couldn't be implemented without any other equipment. As well as this the amount that the robot can move while seated is very limited compered to how a human can move. Therefore, it's hard to replicate.

This would be a good area for future years to look into, but it should not be a main focus of the project because not much work has been done on self-starting and it is a way that you can get the robot to be more self-reliant and humanlike but it is an area that is not essential to the project so too many resources shouldn't be wasted on it.

Harry Withers

5 Modelling Swinging in Webots

Before running code on a real robot, it is advantageous to first test the planned motion on a simulated robot. This is because any damage done to a simulated robot during testing can simply be undone by resetting the simulation where, on the real robot, hundreds of pounds worth of repairs may have been needed. Simulation also allows the advantage of revealing which models are not worth testing on the real robot and highlighting others that yield more promising results. The final advantage of testing on a simulated robot is that many motions can be tested a large number of times without any wearing down any of the robot's joints prolonging the life of a expensive piece of equipment.

5.1 Introduction to Webots

The simulation program used in this project was Webots 8.5.3, a development environment that can be used to model, program and simulate mobile robots [24]. Webots uses a tree-like structure of nodes to represent every object in the simulation and their properties with all of this information being stored in a world, or .wbt, file. This world file also contains information regarding the viewpoint and position of the observer.
Controller files are used in Webots in order to manipulate nodes, and so the simulation, and can be written in a range of languages including C and Python [25]. Every robot in the simulation is represented in the world by a robot node which can be assigned a controller file. In this case of the NAO robot, Webots has included a pre-built robot with it's own controller file which allows the simulated robot to be operated in the same way as the real robot [26]. After a time step the world is updated according to the physics engine and controller files and the result displayed.

5.2 Building the Virtual Swing

In order to simulate the NAO robot attempting to swing, the world must contain both a robot, provided by Webots, and a swing. Last year's group provided us with a number of Webots world files containing a variety of swings they had used for last year. However, last year's approach to simulation was very different from this year's, as explained later, and so it was decided that we should construct a new swing. It was decided that we would build two swings: a solid swing and a chain swing with two joints loosened in case the investigation was furthered to incorporate a chain swing.
Figure 5.1 shows the node structure of the solid swing used to test the robot's motions. This figure shows the tree-like structure Webots uses to represent every robot and object in the world. The tree-like structure does not allow a node to have more than one parent which became an issue when attempting to build the swing as the seat needed to be a child of both the left and right arms. To solve this problem, Figure 5.1 shows a connector on both the seat and the left arm. In Webots, connectors form a physical connection between the two objects they are attached to so long as a set of conditions are met including distance between the objects. This meant, however, that in order to start the swing in an elevated position, both arms had to be independently set to the same starting angle.
The structure of the chain swing is very similar to that of the solid swing only with extra hinge joint nodes added at two points along the arms. Once again, a pair of connectors was used to complete the swing.
It was important that the joints at the top of the swing had the same damping coefficient as in the real world

so that each simulation was as accurate as possible. When applied to the virtual world, the damping coefficient calculated in Section 3.5.2 was found to be too small producing a half life of around 3800 seconds compared to the 170 seconds expected in the real world. Multiplying the calculated damping coefficient by the ratio of these half lives produced a damping coefficient of approximately 0.1. Figure 5.2 shows that the damping in the virtual world, using this new coefficient, closely resembles the expected damping in the real world.

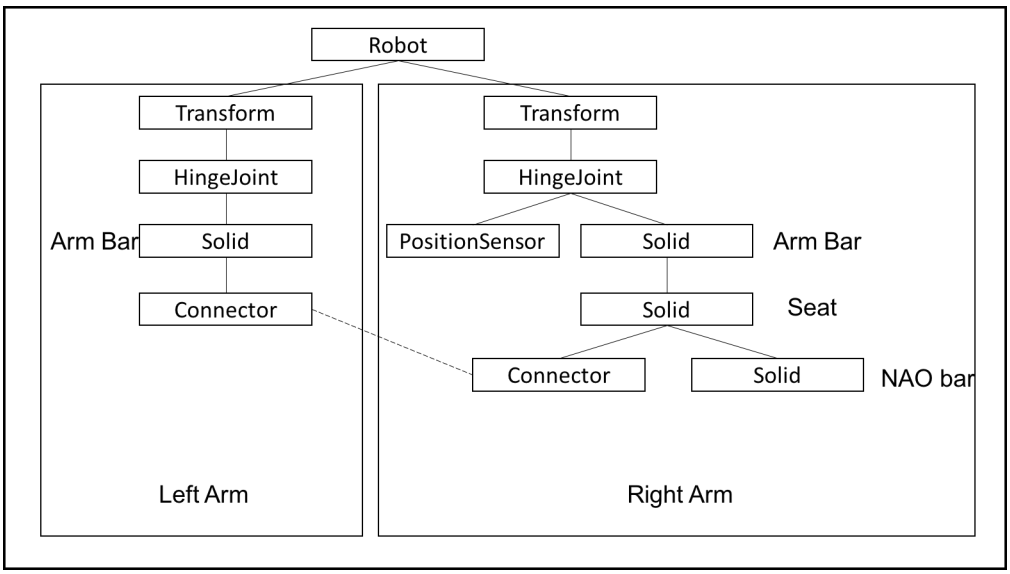


Figure 5.1: A diagram showing the node structure of the solid swing. This diagram shows the connectors at the bottom of the tree that attached the seat to the left arm.

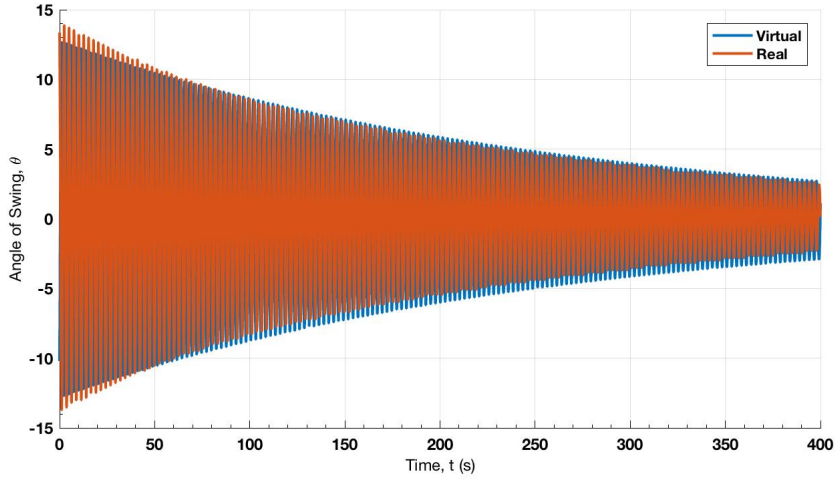


Figure 5.2: A graph showing a comparison between the damping of both the real and virtual swing. This shows that the behaviour of the virtual swing closely follows that of the real swing.

5.2.1 Virtual Angle Encoder

A number of the models tested on the robot, especially those using machine learning, required constant updates on the angle of swing. This meant that the file controlling the robot must have been able to request and receive

the current position of the swing. This was a relatively simple task when using last years approach as an emitter and receiver pair could be used to send data between two controller files and hence the position of the swing sent to the robot.
Our approach required a Python file to be able to connect to the swing robot from outside Webots to query the swings position. This was achieved by using the swings controller file to set up a transmission control protocol (TCP) server allowing clients to connect and query the position of the swing. A TCP connection is a one-to-one connection made with an IP address and port number exactly like connecting to the robot. This connection can be split into two key concepts: the client and the server. As the server was also a controller file for a robot it was still responsible for constantly stepping the simulation to allow it to run, if this did not happen then the NAOqisim controller was unable to properly start and the simulated robot could not be connected to. This required the use of multiple threads; one to step the simulation and one to handle all communication. The simulation thread was assigned a function that simply called the swing robot's step function in a continuous loop that was broken only if the simulation were to crash in some way. The communication thread, meanwhile, would also continuously loop each time checking if there was any information in the TCP socket's buffer indicating that server had been sent a request from a client. If a request had been sent then the position of the swing was retrieved from the position sensor attached to the pivot joint and sent back to the client as a string.
The client, on the other hand, was written as an API so any user could easily connect to the swing without having to set up or have an understanding of sockets and TCP connections. This was done by creating an object name `SwingProxy`. This object, upon instantiation, would attempt to make a connection to the IP address and port number passed to it. In the case of the virtual angle encoder these were as follows:

$$\begin{aligned} IP &: 127.0.0.1 \\ PORT &: 5005 \end{aligned}$$

In order to receive the position of the swing, the `get_angle` function had to be called. This function would send the keyword "update" to the server which would trigger the server to send back the position of the swing. This function had the same name as the function used by the real angle encoder in order to minimise the amount of code needing to be rewritten. In order to adapt any code written for the virtual world to the real world, the `SwingProxy` object had only to be replaced by the imported encoder.
The machine learning subgroup required the ability to be able to revert the world to its initial state from a Python file. In order to achieve this aim, the swing had to be constructed as a special kind of Webots robot called a supervisor. According to the Webots documentation, a supervisor is a robot that is also able to control the simulation as a human would and can therefore be used to revert the world [27]. A function, named `revert_world`, was written into the client which, when called, sends the keyword "revert" to the server. The server would then call the `simulationRevert` function in the supervisor class causing both controllers to stop running and the virtual robot and swing to return to their start positions. Finally, the client thread was put to sleep for 10 seconds to allow time for the controllers to fully restart to prevent failed connection errors.

6 Swinging from Predetermined Functions

6.1 Achieving Smooth Periodic Motion

The first numerical model tested on the robot was time based. The numerical modelling subgroup supplied an equation for the position of each of the joints at a certain time based on the period of the swing in question. These equations were as follows

$$\theta_{head} = 0.590 \sin \omega t - 0.0785 \tag{6.1}$$
$$\theta_{hip} = -0.236 \sin \omega t - 0.749 \tag{6.2}$$
$$\theta_{knee} = 0.820 \sin \omega t + 0.730 \tag{6.3}$$

Where ω is the natural frequency of the pendulum and all angles are in radians. These equations were used in the `smoothPeriodic` function in the `TimeSwing.py` file. In order to achieve smooth motion, this function split

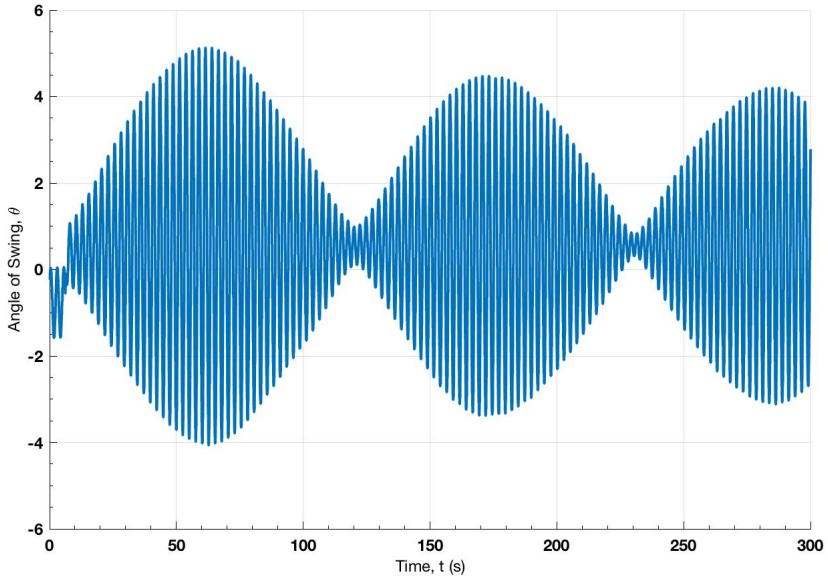


Figure 6.1: A graph showing the result of running a function based on smooth periodic motion on the virtual robot. This graph shows that the robot underwent parametric resonance as it attempted to drive the swing.

each period into a number of steps. A rest interval was then calculated that should be passed to the `moveLimbs` using the following equation

$$\Delta t = \frac{T}{N}. \tag{6.4}$$

A loop was then started for every oscillation that would pass the current time to the equation, set the angle and increment the time by the rest interval until the period had been reached. This was repeated for a number of oscillations specified by the user.

6.2 Results in Webots

Applying this function to the virtual robot created the graph shown in Figure 6.1. This graph shows the amplitude of the robots swing grow to maximum of approximately 4.5 degrees. At this point the robot moves out of phase with the swings motion and the amplitude drops. It is expected that, with a more accurate period of the swing, a high amplitude could be achieved but it would still oscillate. In order to prevent the collapse of the amplitude, the robot would need to dynamically change its motion in order to stay in phase with the motion of the swing as explained in Section 2.5.2.2.

6.3 Results with the NAO

It was not possible to test the smooth, time based model on the real robot because of two real world limitations. The first was that if each period was split into too few intervals then the robot would reach each subsequent position significantly before the next instruction was given. This meant the robot would come to a stop between every incrementation and so it's motion would appear discontinuous. The second limitation was that the number of steps required to produce smooth motion would also cause the period of the robot's motion to drop significantly. This was because the real robot was unable to process such a high number of commands in a short time frame. A potential solution to this problem would be to write the model in C as this is a language able to run much faster at run time once it has been compiled.

7 Swinging with Feedback

7.1 Angle Encoder Feedback

Harry Withers

7.1.1 Implementing Numerical Model

The next numerical model to test was one in which the robot was able to determine the current angle of swing. The position in this case was provided by both the real and virtual angle encoders. This model worked by determining when the robot was at the bottom of the swing by comparing the previous and current angle. If the product of these two angles was less or equal to zero then the swing must just have passed the bottom of the swing. At this point a timer was started; at 0.1 seconds before one quarter of the period the robot was moved to a one of the two pre-set positions. The position the robot was to be changed to was determined by examining the sign of the angle of swing at the point. If the angle of swing was positive then the robot was set to position 1, conversely if the angle was negative the robot was set to position 2.
It was important that the period of the swing was known accurately to ensure that the robot changed position at the optimal moment. This was achieved by first feeding the program an initial estimate of the period. The robot would then determine the period every full swing and use this to calculate a running average of the period of the swing. This adaptation means the robot is able to use this model to drive a swing with any period. The Python file `PositionSwingDynamic.py` contains the code for this approach.

7.1.2 Results in Webots

Using this model, the robot was able to drive the swing to produce the graph seen in Figure 7.1. This graph shows an initial growth to around 20 degrees in only 5 minutes; eventually this graph shows that the amplitude of swing comes to a maximum at approximately 47 degrees. At this point, the robot's movement no longer produces enough of a driving force at the extreme positions to overcome friction and drive the swing further.

7.1.3 Results with the NAO

When this method was applied to the real NAO robot, the graph shown in Figure 7.2 was produced. This graph, taken over a period of 7.5 minutes, shows the robot driving the swing to an amplitude of $15.8 \pm 0.2°$. This is considerably less than the 20 degrees obtained in a similar time frame in the virtual world. The difference in these 2 values is caused by a couple of inefficiencies in the real world. Firstly, in the real world, the swing was able to flex from left to right, an effect that was particularly noticeable when the robot moved from one position to the other. This caused a large amount of energy to be lost which both slowed the growth of the amplitude and also capped it at a lower value. Secondly, the real NAO robot was not as well attached to the swing as in the virtual world causing the robot's bottom to briefly rise up off of the seat when moving between positions. This caused a further energy loss leading to a reduced drive.

7.2 Vision Feedback

Elise Dixon

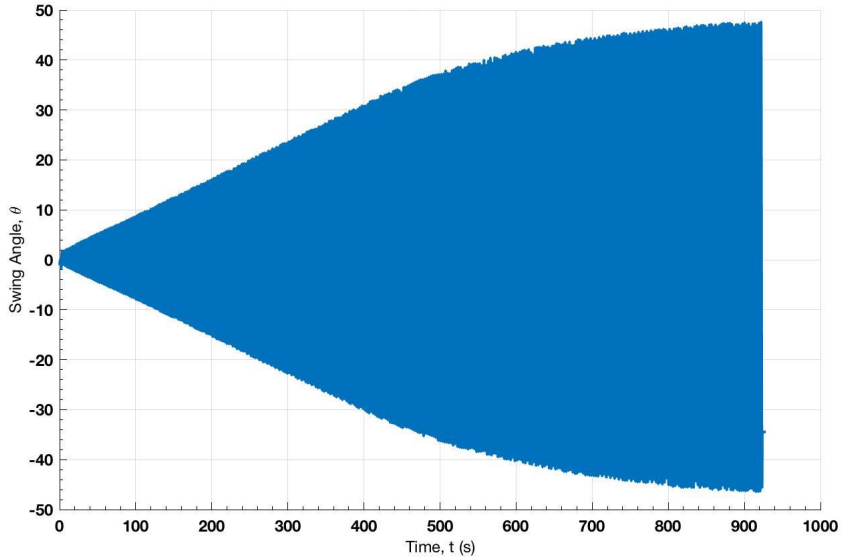


Figure 7.1: A graph showing the result of running a function based on position on the virtual robot. This graph shows a maximum amplitude of approximately 47 degrees before the driving force of the robot can no longer overcome the frictional forces of the swing.

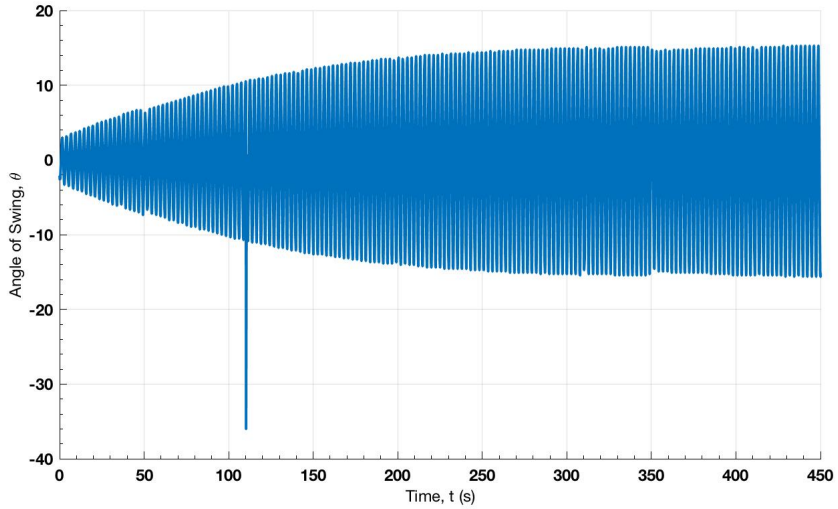


Figure 7.2: A graph showing the result of running a function based on position on the real robot. This graph shows a maximum amplitude of $15.8 \pm 0.2°$ before the driving force of the robot can no longer overcome the frictional forces of the swing.

7.2.1 Stationary torso initial tests

A Python script was written to check for Naomarks at regular intervals and output alpha and beta (the angular positions about the vertical and horizontal axes respectively), sizeX, and sizeY. The robot was then placed on the swing without any motor movement, and swung from an initial amplitude. A Naomark was placed directly ahead of the robot, and the video watcher in Choregraphe was used to ensure the mark could be seen during a sufficient range of the swing motion in order to determine the period of oscillation. The data from the angle encoder was also recorded at each interval in order to compare the relationship between each data output, and the position of the swing. It was found during testing that the robot occasionally detected Naomarks which were not present. This may have been due to the robot mistaking random objects in its field of view for Naomarks. This caused issues in the data as their related shape information outputs were undesired in the dataset. It was therefore necessary to add variables to hold expected landmark IDs, and implement appropriate checks in the code before a landmark's data was added to the dataset.

Once a successful output was produced, it was found that the beta angle varied over a larger range than the alpha angle, as can be seen in Figure 7.3. The reason for this is that, as displayed in Figure 7.4, the Naomark's position in the beta angle range is related directly to the position of the swing due to the tilting of the robot's horizontal axis relative to the ground and therefore the Naomark. The alpha angle, however, varies only if the Naomark is not placed exactly parallel to the robot's motion, and arises as shown in Figure 7.4 due to the change in proximity of the robot to the horizontally displaced Naomark between the two extremes of motion. Both angles, in theory, should be useful in determining the angular position of the swing, however the alpha angle depends on a sufficient horizontal displacement of Naomark, whereas the beta angle varies irrespective of Naomark positioning, making it the easier method to generalise for any further use.

The size of the Naomark was also tested by the same method, with results shown in Figure 7.3. The variables sizeX and sizeY were found to be largely similar due to the symmetrical size of the Naomark, so only sizeX was considered in subsequent tests. As the size of the Naomark only varies by approximately 0.03 radians where the beta angle varies by approximately 0.2 radians, it was decided that beta location angle produces the better variable to consider in determination of the swing angle.

When compared to the swing angle, it was found that beta always recognised a change in direction slightly later than the angle encoder. This was investigated further by fitting both of the signals shown in Figure 7.5 on MATLAB's *cftool* using sinusoidal functions with a phase difference. This phase difference was converted to a time delay using the frequency of the oscillation, and was found to be (0.16 ± 0.01)s. Each individual time delay between the peaks was found using MATLAB's *findpeaks* function, and plotted to investigate whether the delay was consistent over various time periods. Although there is a variation in the delays over each period, the variation appears random rather than time dependent. It was therefore concluded that it was acceptable to use the delay found using *cftool* in every instance. The source of this delay was considered, and may have been due to an imag processing delay. The decision was made to add a delay into the code to compensate for this disparity, rather than change the resolution of the images, as it was clear that a reduction in resolution could result in greater loss of accuracy concerning the turning points of the position values. The frame rate may also produce a small contribution, as the image updates every 0.03s, which is greater than the sampling rate of the data.

7.2.2 Moving Torso Test

The tests described previously account only for a robot with a stationary torso. This case limits the field of view to directly in front of the robot. Thinking ahead to the implementation of sensor feedback, it is necessary for the robot to detect Naomarks while it carries out movements between two positions. The extended position, where legs are up and torso leans back, should occur momentarily before reaching the backward maximum on the swing as suggested by the Numerical Modelling subgroup. This position, however, limits the visibility of any Naomark placed in front of the swing. Different tests were taken to find another useful position of Naomark. It was found that a Naomark placed just to the side of the top of the swing was visible at the backward maximum

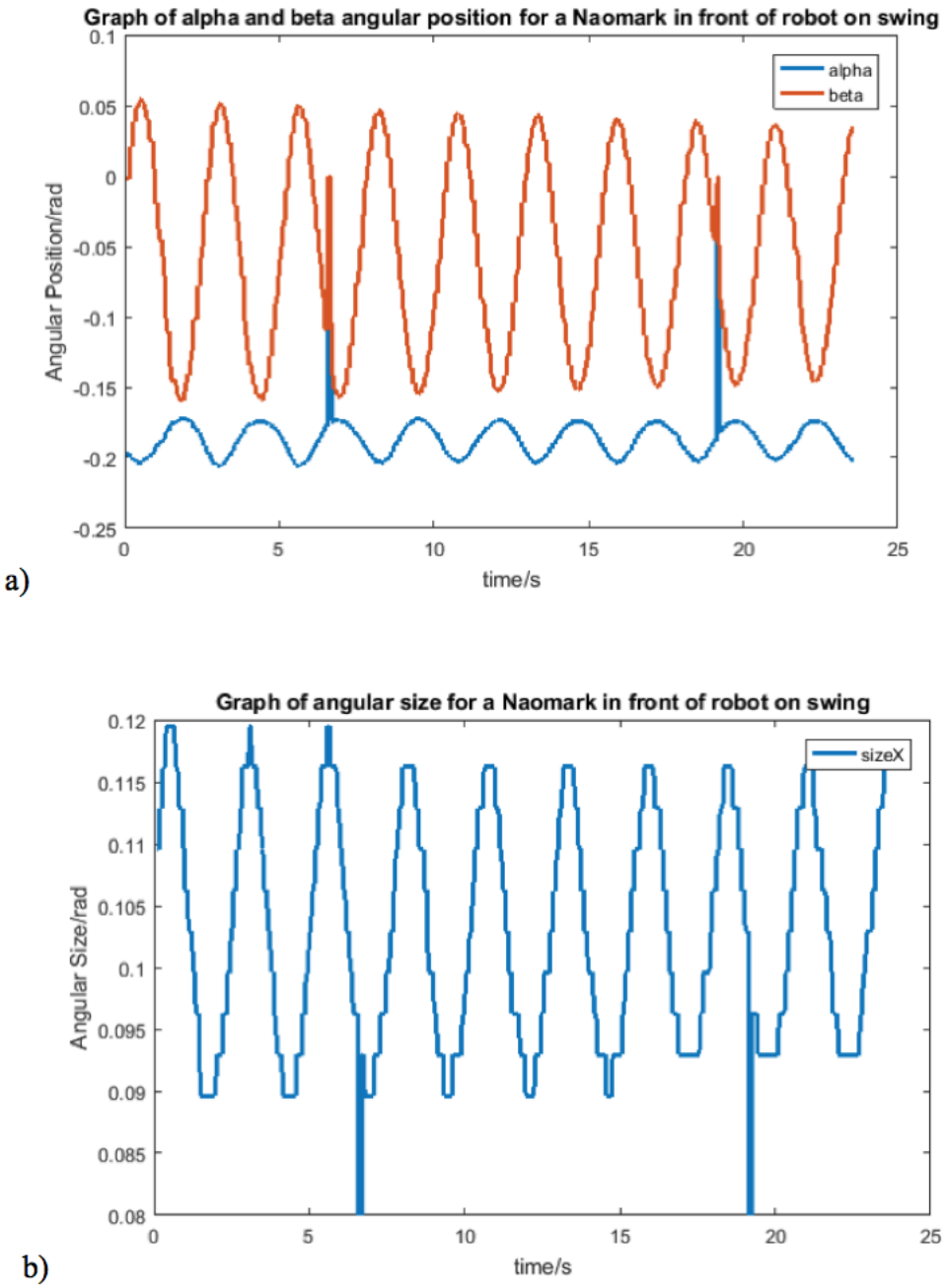


Figure 7.3: a) Graph of alpha and beta angular positions of a Naomark in front of the robot on the swing. The spikes are due to the robot detecting no Naomark on a particular interval, thus outputting a zero value for alpha and beta positions. b)Graph of sizeX, the angular size of the Naomark in the robot's field of view. Lower spikes are due to the robot detecting no Naomark.

when in the extended position.

Tests were then required to track the visibility of these two Naomarks at different points in the swinging motion. Due to complications caused when the robot attempts to run from angle encoder feedback whilst sending sensor data, it was necessary to run time based models instead. This was done by initiating code at the same moment as the swing was released from a displacement, and changing the robot's position every half period. This caused complications as the robot's movements eventually comes out of phase with the swing, and

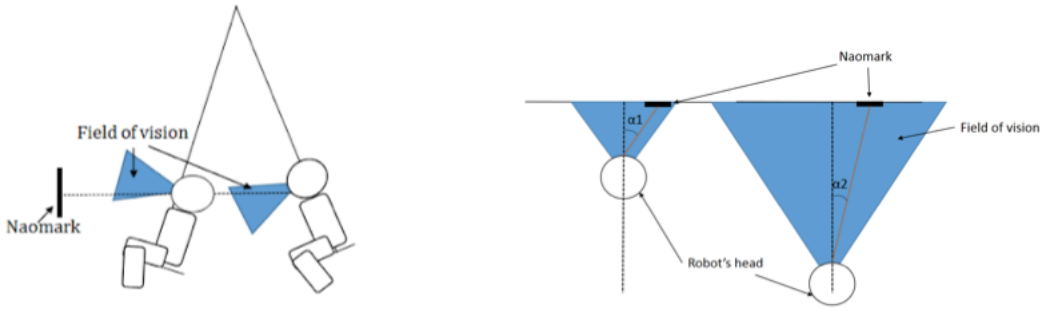


Figure 7.4: Diagram showing how the landmark beta position alters in the field of vision of the camera at the two maxima of swing movement, and diagram of system as seen from above, to show the origin of the variation in the alpha position of the Naomark at the two maxima of swing movement

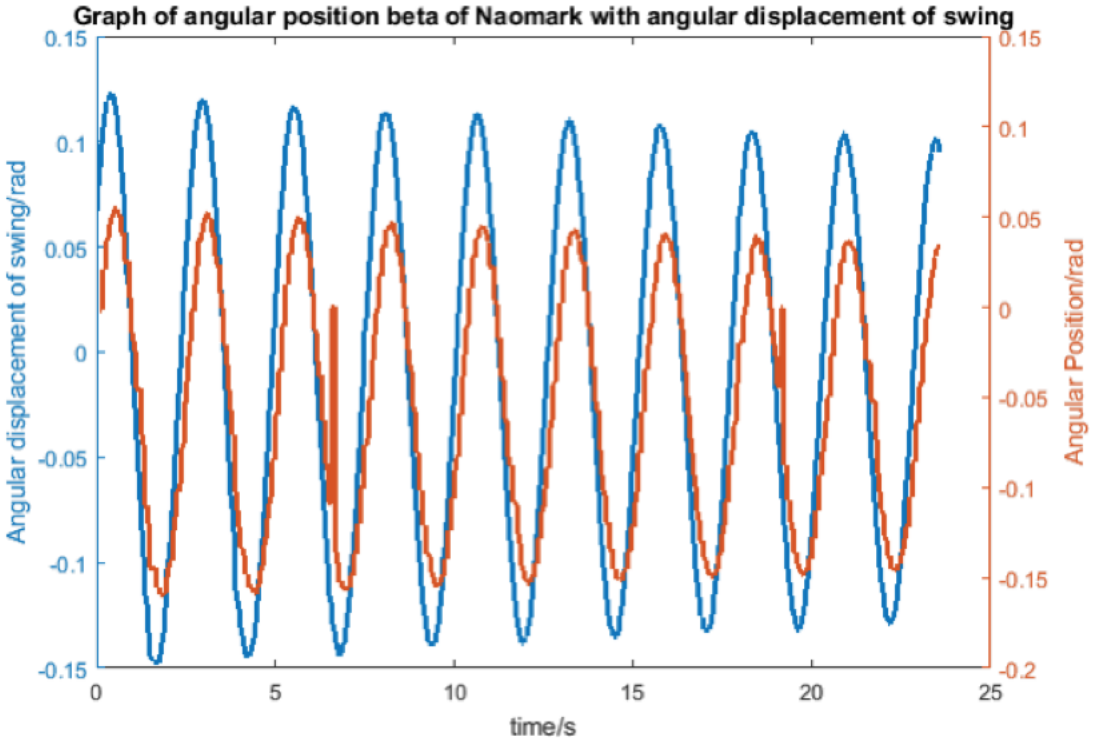


Figure 7.5: Angular position of Naomark compared with the angular displacement of the swing. The swing angles in this figure have been multiplied by -1 for clarity in comparing the times where each signal peaks, as in reality the signals peak in antiphase. It is clear from this figure that there is a slight delay between the two signals.

an accurate depiction of landmark detection is not produced, as the robot may be in the wrong positions at the wrong times. To gain a clearer picture of when the robot and swing were in phase, its hip pitch memory was recorded alongside the angle of the swing and the beta positions of the two Naomarks.

As this is a very limiting requirement, other options were explored. An attempt was made to counteract clockwise hip pitch movement with anticlockwise head pitch movement of the same angle, so the head could remain parallel to some axis at all times. In this case, the only change in Naomark position would be due to

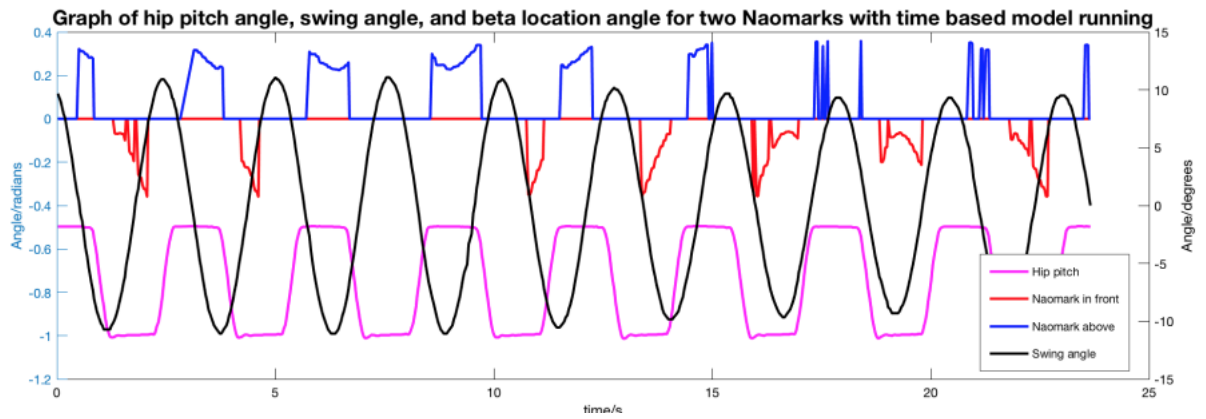


Figure 7.6: Graph showing the hip pitch angle, swing angle, and the angular beta positions of the Naomarks in front and above the robot. This graph illustrates how the motion became out of phase with the angle of the swing and the effect that has on the detection of Naomarks.

the change in height due to rotating torso, and the motion of the swing. As hip pitch angle memory values can be accessed in any script, it could be possible to geometrically subtract the robot's own torso motion from his vision response, thus extracting the swing position. This method proved problematic, as the head will only rotate forward 29.5 in from the neutral position. [13] This would therefore limit the extended torso position to the same angle, which is less than ideal especially when combined with the anticlockwise head rotation causing an unwanted torque in the wrong direction. One must also note that the motor speeds between two joints do not necessarily match, so a position based subtraction may not be entirely effective.

7.2.3 Implementation

7.2.3.1 Fixed Interval Method

The aim of the code written to implement the landmark detection feedback was to recreate the successful motion produced by the angle encoder position based feedback. This method involved moving between the two positions at a set delay time before the maximum amplitude was reached. This was achieved using dynamic calculation of the time period at every swing by comparing the time between two consecutive zero angle points.

This method of calculating the time period was altered, as the initial tests showed that the simplest way to recognise a particular point in the swing oscillation was to recognise the turning points of the beta position. Code was written to calculate the gaps in time between consecutive peaks in the position of the front Naomark. These peaks coincided with the back maxima on the swing, and thus the code was written to move the robot to its extended position after a time period minus the image processing delay found earlier in addition to a small delay so that the movement occurs before the swing reaches the maximum. The robot then waits another half period before returning to its upright position. Another piece of code was written where the robot begins in the backwards position and detects the Naomark's change in direction at the front-most maximum.

Checks were necessary in the code to ensure the period was not calculated to be a very small value due to small vibrations. Another was also introduced to enable the period to be updated only if the robot was in its upright position, thus avoiding any false detection of troughs due to the motion of the robot. These checks successfully filtered detections so that only a relatively accurate period could be used to time the robot's movements between the two positions.

This code produced a motion in which the robot swung for two periods, and then had to wait for another period before it began the motion again. This was due to the robot still being in motion when the desired data

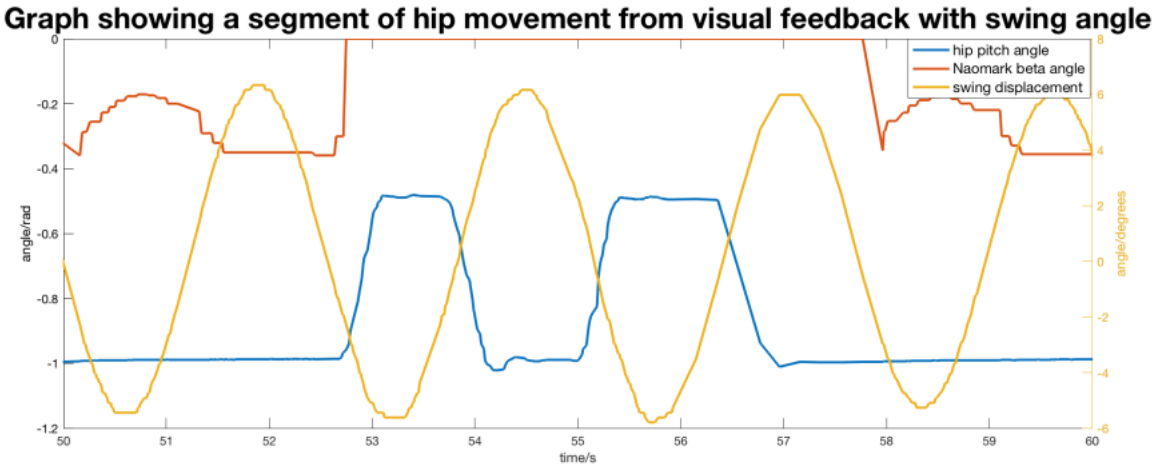


Figure 7.7: Graph showing the hip pitch angle response to the maximum in Naomark beta angle in relation to the swing displacement.

collection point was reached which, as previously discussed, does not allow the robot to locate the maxima in the Naomark position. However, although the movement was not continuous over every period, the amplitude was able to maintain a steady state of (4.6±0.2)°when released from a higher initial displacement. This method of swinging is therefore named the fixed interval method, as the motion is skipped at fixed intervals.

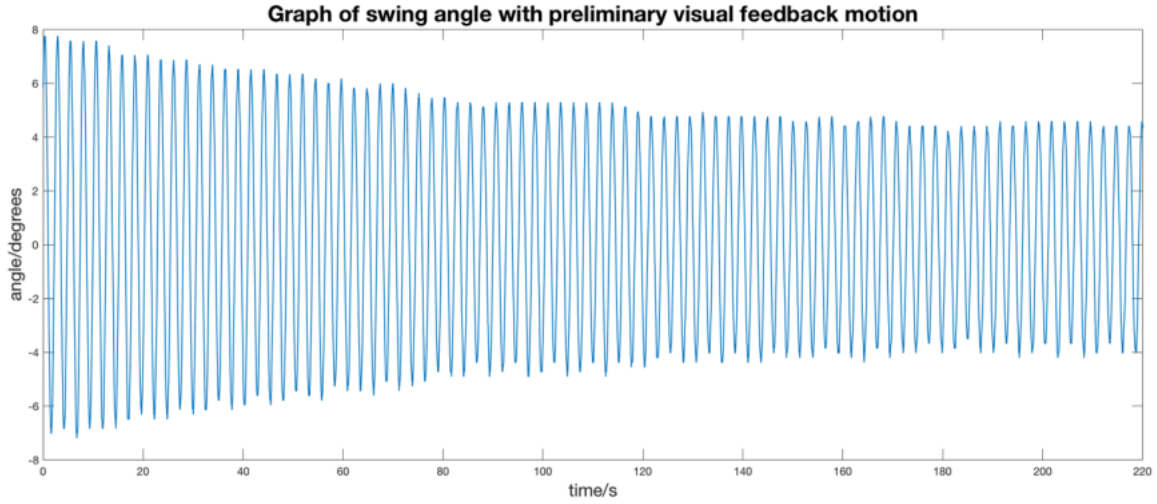


Figure 7.8: Graph of swing angle with the preliminary visual feedback motion, showing a decay to a fairly stable steady state of (4.6±0.2)°

7.2.3.2 Increasing Interval Method

Further investigation was implemented to try to eliminate the regular wait between subsequent calculations of the period. By running the same code with different delay values, it became clear that the robot's ability to detect the landmarks at the relevant times was fairly sensitive to the delay. This was because a delay too large or too small would result in the robot still being in motion at the maxima of the swing's displacement. It was then concluded that periods of stationary robot position were unfortunately necessary to calculate the period dynamically.

It was, however, possible to increase the intervals between the stationary periods. This was done by implementing a check in the code so that if the new calculated period is very similar to the previous one, the repetitions of the swinging motion are performed an increasing number of times. This is increased until the calculated period varies from the previous values, if, for example, the robot became out of phase, at which point the motion resets to one repetition.

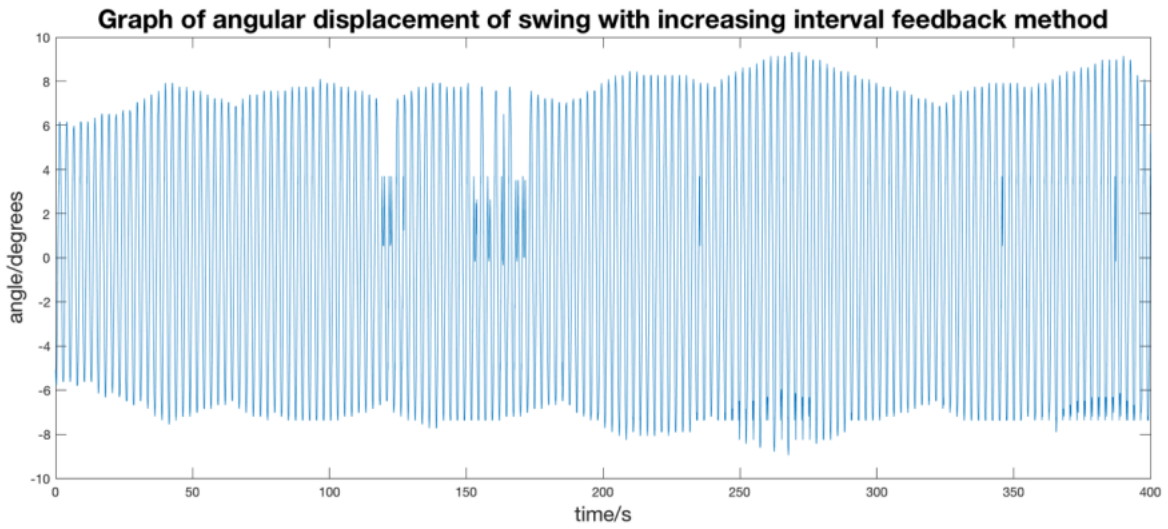


Figure 7.9: Graph of swing angle employing the increasing interval feedback method. This method was much less stable, due to small miscalculations in time period being carried out over large intervals without re-checking the time period. The maximum angle reached is (9.1±0.2)°

This was found to produce a higher amplitude, with a maximum displacement of (9.1±0.2) °, as displayed in Figure 7.9, however after a particular number of cycles, small differences in the period became more significant when the time based action was repeated many times, so the margin of error on the phase difference between the robot's movement and the swing's movement increases as the interval between period calculations increases.

7.2.3.3 Further Models

An attempt to resolve the phase problems described in the increasing interval method involved limiting the maximum interval to 10 time periods. An alternative alteration was made to force the robot to average over many time periods, however by the time these changes were made, the battery in the robot was unable to sustain two minutes of swinging before running out, so conclusions on these improvements cannot be made.

At this stage in the testing, the robot also frequently reported issues of overheating joints, occasionally despite being allowed to cool for days. It is suspected that the temperature sensors may be damaged, and thus the current to the joints may have been limited unnecessarily. Figure 7.10 shows hip pitch data taken while implementing a timed periodic motion for the purpose of visual sensor tests. In this case, the Choregraphe memory watcher reported temperatures of above 60°in both of the hip joints. It is clear that the torso on the third movement is slower to return to its upright position. This issue has an effect especially on the visual feedback methods, as a longer time to reach a particular position may result in missing a maximum, or mistaking the position of a maximum due to its own change of position relative to the Naomark.

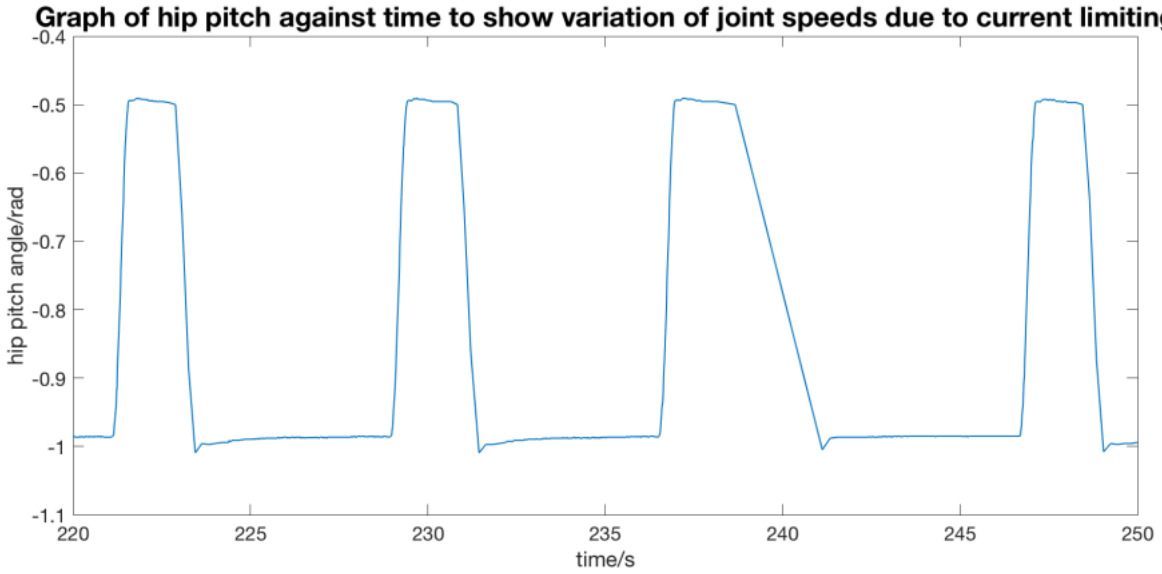


Figure 7.10: Graph to show the variation of motor speed of the hip pitch when current limiting is occurring due to high reported temperatures.

7.3 Gyrometer Feedback

Katherine Evans

7.3.1 Further Testing

From initial sensor tests it was concluded that the use of gyrometer y, with the removal of signal due to the angular velocity of hip motion, showed potential to replace the angle encoder in determining the position of the robot for feedback swinging methods.

It was realised that since the 1.14 versions of NAOqi a new ALMemory key was available which returned a calibrated gyrometer value in rad/s rather than the raw data used in preliminary investigations. Instead, *"Device/SubDeviceList/InertialSensor/GyroscopeY/Sensor/Value"* should be used. Values returned using this memory key, when simply converted to degrees per second, closely follow the calculated swing velocity with the robot stationary on the swing thus eliminating the efforts previously needed to scale and offset the raw data.

To progress towards a method of determining swing position from the gyrometer readings, it was necessary to monitor the gyrometer readings when the robot was executing what was expected to be the most successful motion on the swing. This would give clear indication of the feasibility and reliability of using the gyrometer to replace external inputs in feedback swinging. The code written for angle encoder feedback, which recalculates the period of the swing dynamically and instructs the robot when to swing based on this provided the template for the motion. The limb motions and speeds were given by `position1Dynamic` and `position2Dynamic` taken directly from SwingAPI.py (see section 3.2.1) but used in a time based model using the natural period of the swing, rather than the angle encoder feedback model they were used in previously. The robot was simply instructed to switch between these two positions every half a period for these further tests of gyrometer readings. This allowed all the relevant data to be recorded whilst both the swing and the robot were moving and as in initial tests, the angle of the swing, hip pitch, gyrometer y and time were all recorded.

Results had previously been taken to confirm the relationship between the gyrometer reading and swing

motion with the robot stationary on the swing. Further tests needed to be taken with the robot executing this motion on the desk. These were needed to check the relationship and scaling for the hip velocity effect with the calibrated gyrometer data before it could be removed from readings taken with the robot moving whilst swinging. Plotting the gyrometer and calculated hip velocity for motion executed on a stationary seat showed no significant need for scaling. Relative peak heights varied, with the gyrometer peaks usually appearing at a greater magnitude in the backwards motion which was proposed to arise from the speed of this motion being slightly faster that of the forwards movement due to working with or against gravity. However, the ratio of gyrometer peak height to hip velocity peak height only varied between about -0.9 and -1.2, so it was decided that scaling the data was unnecessary as any leftover signal after addition did not appear significantly higher than other noise in the gyrometer reading.

With this simple method of addition of calculated hip velocity and gyrometer reading roughly cancelling the robot's own torso motion it was possible to begin efforts to extract the swing velocity from the gyrometer reading whilst executing this swinging motion on the swing. The swing was released from some amplitude and the robot instructed to execute the motion every half period so that its change occurred at the extremes of the swing amplitude. The results of this are shown in Figure 7.11 which illustrates the total gyrometer reading alongside the swing angle.

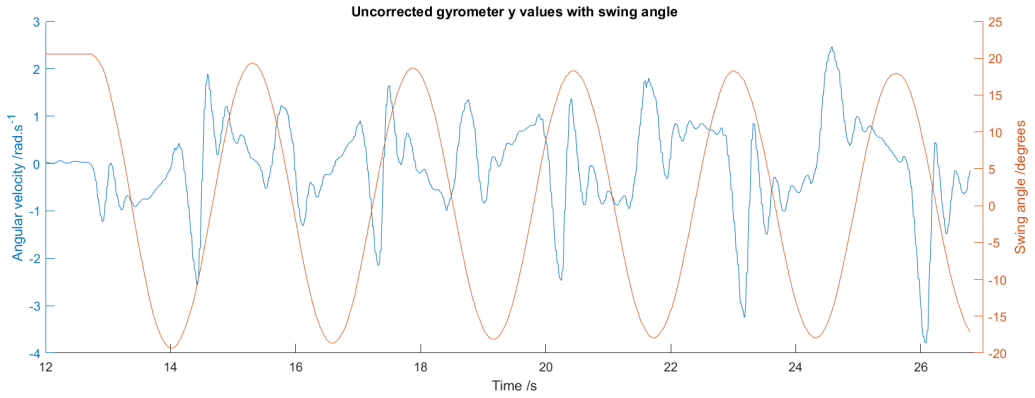


Figure 7.11: Uncorrected, smoothed gyrometer y values plotted alongside the swing angle. Motion executed on moving swing at motor speed=0.75

7.3.2 Analysis

The resultant gyrometer value smoothed over 20 points after removal of the robot's own torso motion via the hip velocity is shown alongside the calculated swing velocity in Figure 7.12. The periodicity of the swinging motion is visible from the gyrometer data but the additional spikes in the curve required attention as they spikes do not consistently occur at the maxima of the swing velocity and therefore any method of implementation in feedback swinging based on timing between subsequent maxima could give unreliable results.

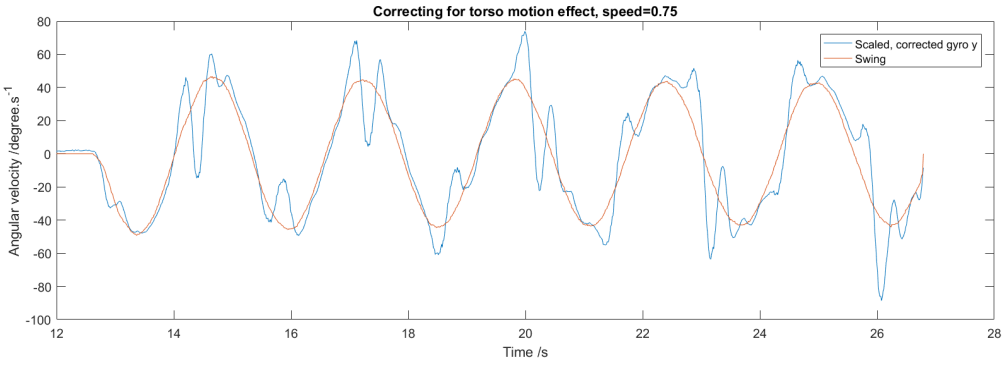


Figure 7.12: Plot of corrected gyrometer value alongside calculated swing velocity. Motion executed on moving swing at motor speed=0.75

Efforts were made to establish where these arise from and after watching the robot execute the motion at this speed. It became clear that the momentum it has when moving back and then the sudden stop of motion caused collisions with the seat and resultant small but rapid vibrations of the torso forwards and backwards. This is shown in the plot of gyrometer readings and inverted hip velocity for the robot on a stationary seat in figure 7.13 where the small oscillations on either side of the gyrometer peaks are clearly not due to any variation in hip pitch. It may be possible to limit the extent of these oscillations in gyrometer value by better securing the robot to the seat so that its torso is only minimally effected by these vibrations.

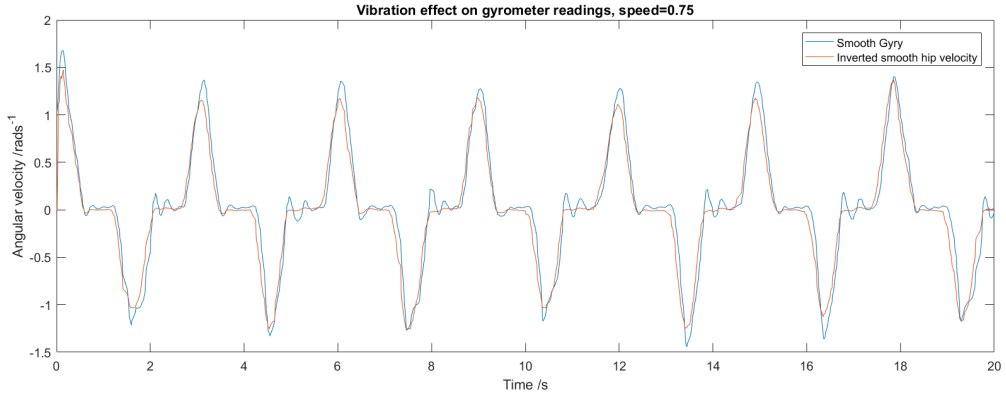


Figure 7.13: Vibrational effects seen clearly with motion off the swing at a motor speed of 0.75

Another method to reduce the extent of these spikes in the corrected gyrometer value was to reduce the motor speed at which the motion is executed. This meant that the stopping of the motion was less violent and the robot does not end up colliding with the seat. Retaking the motion tests on a stationary seat with the motor speed set to 0.50 instead of the maximum speed used of 0.75 resulted in the corresponding plot of gyrometer readings and inverted hip velocity shown in figure 7.14. Evidently the gyrometer readings for this case follow the hip velocity values much more closely with minimal variation at the edges of the motions and so it can be concluded that this method is more reliable with lower motor speeds where torso motion is affected solely by the hip pitch variation. Consequently, the recordings of motion on a moving swing were repeated with this lower motor speed. The results of the gyrometer corrections with a motor speed of 0.50 are shown in figure 7.15 and the deviations from the swing velocity behaviour are visibly much less severe. There are still some differences in the values of the maxima of each plot but for the majority they occur at the correct, predicted location which would be the necessary feature for implementing in feedback models. Where there is some offset between the gyrometer maximum and swing velocity maximum it is suspected that this may again arise from loose fixing of the robot to the swing such that it does not move in total synchronisation with the same motion as the swing.

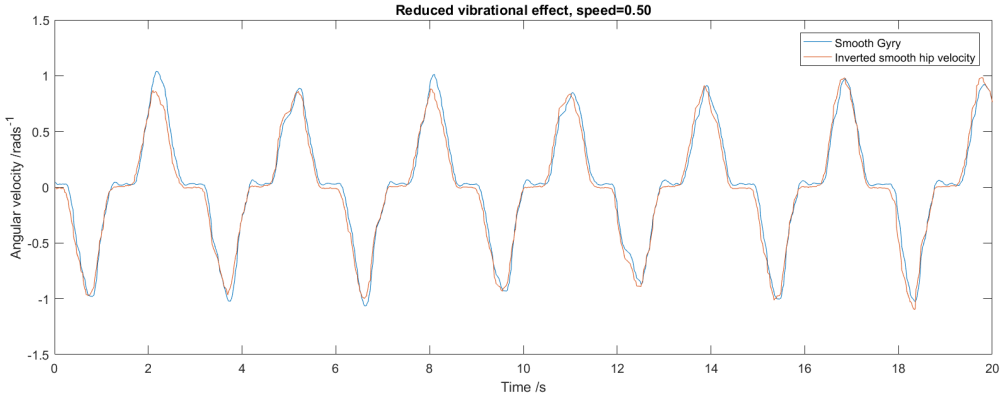


Figure 7.14: No significant vibrational effects in gyrometer reading with motion off the swing at a reduced motor speed of 0.5

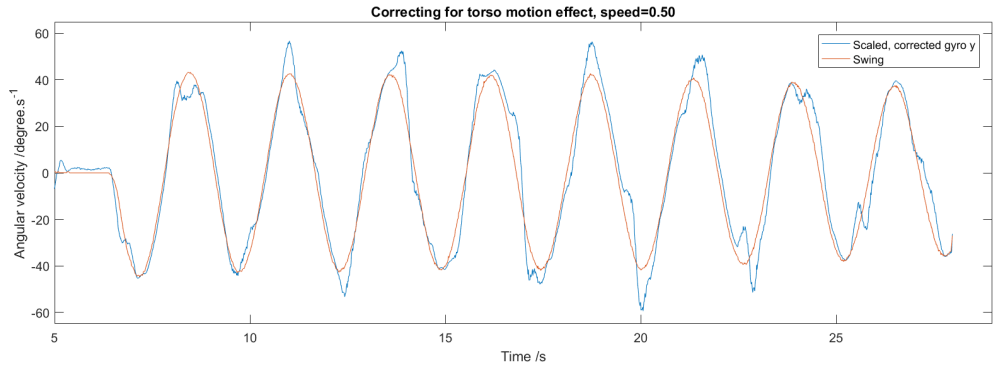


Figure 7.15: Plot of corrected gyrometer value alongside calculated swing velocity. Motion executed on moving swing at motor speed=0.50

7.3.3 Feasibility of Implementation

Gyrometer feedback was ultimately never trialled within this investigation. However, it is hoped that if the project is repeated this further investigation including the effects of torso motion will provide a starting point for the testing process. Figure 7.15 provides evidence that, with torso motion, a corrected gyrometer reading is obtainable and that it fairly reliably follows the behaviour of the swing velocity thus making it possible to determine swing position from the adjusted data, which could then be used in a position based swinging model. The barrier to having this implemented is the number of steps needed before a corrected gyrometer reading is obtained which would require a more extended piece of code that was never written in this investigation. Both the hip pitch and gyrometer y values must be monitored and the hip velocity calculated continuously so that its affect can be removed from the gyrometer readings. The resultant, corrected gyrometer value must also be smoothed to prevent the robot reacting to a sudden spike unrelated to the swing motion. Once this corrected gyrometer value stage has been reached, similar methods to those described in the previous year's report [7] which utilised the gyrometer feedback but without torso motion could be trialled. Alternatively, seeing as the gyrometer reading is just $\frac{\pi}{2}$ out of phase with the swing position, an adaptation to the method of period calculation in the angle encoder feedback example from this investigation could be trialled to instruct the robot when to move. This would provide a conclusive answer as to whether the gyrometer y could reliably replace external inputs.

7.4 Chapter Conclusion

Katherine Evans

Of the three sensors proposed for use in feedback swinging: visual, gyrometer and the external angle encoder, only methods utilising the angle encoder and vision feedback were written for the robot and tested. The angle encoder was known from previous investigations to be a reliable method of obtaining swing position. Implementing the numerical model suggesting the robot should change position over the extremes of the swing motion, with dynamic calculation of period and choice of position based on direction of motion via angle encoder feedback, resulted in swing amplitudes of $15 \pm 0.2^\circ$ which was the maximum amplitude achieved on the NAO in this investigation. The difference between this maximum amplitude, and that achieved in the Webots simulation is attributed to swing flex and the robot rising up off the seat during motion resulting in energy loss. Both of these effects could be investigated and modifications to the seat could be made such that the robot is better secured.

Considering one of the aims of the investigation was to eliminate the need for external inputs, the swinging motion using the visual sensors, reaching $9.1 \pm 0.2^\circ$is a successful demonstration of this despite the difficulties in implementation. The difficulties experienced with the visual sensors arose from the challenge of finding a way for the robot to see a reliable maximum in the Naomark position with torso motion. The requirement that the robot is not moving at the point where the desired data is to be taken placed limitations on the continuity of motion and methods to increase the number of motions executed before the robot stays stationary to recalculate the period, though successful in building to a greater amplitude, suffered eventually where the phase difference between the robot and swing motions becomes significant. Further tests of adaptations to the visual feedback to overcome this issue limiting the amplitude of the swing were unfortunately not possible, as described in the section, due to issues with the NAO's battery. These suggestions should provide a starting point for further investigations to build on the success of this method.

The testing within this section extends the work done in previous years by working towards internal sensor feedback methods that remain reliable even with the robot varying its torso position. As well as demonstrating successful swinging from visual sensors with torso motion, the efforts within this section to provide a method to remove the torso motion effect from gyroscope readings shows promise for being successfully implemented in future investigations. With a reduced motor speed the resulting gyrometer reading of the swing velocity seems to be a reliable method to use in feedback models based on velocity or position and it is hoped that if investigations are continued this could replace the angle encoder in the most successful motion. In conclusion, despite it still being the external angle encoder sensor which provided the method achieving the most successful swinging motion, the vision sensor feedback demonstration is a good indication that moving away from external sensors should eventually result in similar amplitudes if the refinements to the method are continued. Both this and the work done on the gyroscope readings should provide a starting point for eliminating the use of external sensors in later years so that the robot can swing independently, mimicking how a human would rely on their own vision and balance on a swing.

8 Machine Learning

8.1 Machine Learning Overview

8.1.1 Introduction

Henry Gaskin

Machine learning is an area of artificial intelligence research. Artificial intelligence attempts to build an intelligent agent. Popularised forms of artificial intelligence are often concerned with making an intelligent agent appear human. The goal of machine learning is to make an agent interact with its surroundings in a rational

way. These surroundings are often called observations, where each possible arrangement of these observations is called a *state*. If an agent makes a choice in a state, that choice is called an *action*. This action will lead an agent to a new state, from which it takes a new action, and so on.

Different forms of machine learning are defined by the type of feedback they give the agent when it makes a choice. The form used in this investigation is reinforcement learning. In reinforcement learning, clear feedback is given to a given to an explorative agent for most, if not all, of its actions. The agent is either rewarded or punished for an action it performs in a state. A general machine learning agent will then learn from its mistakes and make extrapolations towards the best actions to maximise its reward. A machine learning algorithm does not give its agent examples of what it should do. The agent begins a problem with no knowledge of a system. The agent must build its perception of the system based only on the rewards and punishments given to it by the algorithm.

8.1.2 Learning Algorithms

Mike Knee

Temporal difference learning is a technique used in artificial intelligence based around the idea of using past experiences to predict the outcome of future events. This type of learning is useful because it does not require any previous assumptions about the system. The agent is not required to reach a final goal state in order to start learning. In other words, it bootstraps. Some temporal difference algorithms store a policy, which specifies the likelihood of taking a specific action in a given state.

Q-learning is a temporal difference learning algorithm. Rather than having the robot learn a policy, a Q-function is learned, which outputs the utility of taking an action in a specific state (called the Q-value). This Q-function is updated at each iteration, i.e. after each action is taken in a state, and will converge to give correct actions in each state. Often the Q-function is implemented as an array, storing the Q-value for each action in each state in memory. Usually Q-learning is applied to discrete state and action space problems. It will therefore be necessary to either discretize the state-action space of the robot or use a modified Q-learning algorithm to handle continuous data.

Sam Bull

Another form of machine learning is actor-critic learning, which is similar to Q-learning but separates the policy from the value function. The policy is known as the actor and executes actions, and the function which finds the estimated value of a state is known as the critic. The critic must learn the policy being used by the actor and critique it by returning a values called the temporal difference error. This is the sole output of the critic and drives both the learning of the critic and the actor.

Jay Morris

Yet another machine learning algorithm is SARSA (standing for State, Action, Reward, State, Action), which is similar to Q-learning, but uses a policy based on its Q-values. The Q-values themselves are based on not only the current state-action pair and that state's reward, but also the Q-value of the next state-action pair that will be encountered as a result of the action taken. The policy is then calculated to represent the value of taking particular actions in particular states.

8.1.3 Review of relevant work

Sam Bull

[28] discusses the theory of machine learning and its application to the real-world task of solving the inverted pendulum problem. This paper provides some useful information in the application of Q-learning to optimise movement. They also note the importance of machine learning in these situations as "It is impossible to

mathematically compute general solutions of the nonlinear equations describing the physics of the pole cart system". Machine learning is therefore vital to achieve the project goal of discovering the optimal movements of the Webots robot on the swing.

[29] provides an overview of reinforcement learning, containing pseudocode implementations of Q-learning, SARSA and actor-critic methods. It contains a particular case study where an "acrobot" robot swings in a gymnast-like fashion using reinforcement learning. The acrobot itself was a two-joint pendulum, with the first joint corresponding to the gymnast's hands on the bar and unable to exert torque. The second joint corresponded to the gymnast bending at the waist and was able to exert torque. This is a similar system to the Webots robot, with the goal being to maximise the amplitude of the swing.

Mike Knee

[30] details how Q-learning algorithms, which are generally applied only to problems parametrised by discrete state and action spaces, can be transferred to problems with continuous spaces. They describe a number of key requirements for setting up a Q-learning algorithm in continuous space. The algorithm implemented as part of this paper uses an artificial neural network (an arrangement of connected artificial neurons) to generate Q-values, and an interpolation technique to decrease computation time. A technique similar to this was used to create a continuous state-space Q-learning algorithm in section 9.3.

Chris Patmore

[31] details creating a deep Q-learning network for a robot with continuous input and output, as opposed to creating discrete output states. It contains great detail about the thought process of determining the used algorithm and how it was implemented, including discussion about online vs. offline learning and on or off policy learning, which is extremely helpful for understanding the subject. The thesis also looks into many of the issues with using a neural network as a function approximator and applying Q-learning to a continuous state space. The final implementation is a deep Q-learning network with an interpolation web for the continuity problem.

8.1.4 Review of Machine learning libraries

Chris Patmore

Many machine learning code libraries are available, which have the potential to massively reduce the work needed to build a learning system. For example, they avoid the need to program an artificial neural network down to the individual neuron logic.

TensorFlow is a library for numerical computation that can be used to create neural networks for machine learning tasks. The internal operations of the neural network can be treated as a black box. It has been used in the past to let a neural network learn to play the video game 'pong', where each player has to deflect a ball across the screen with a translatable 'paddle' [32]. It can also make use of GPUs to substantially increase learning performance.

Theano is another library, which like TensorFlow can be used to create neural networks for machine learning tasks, in general it operates faster than TensorFlow [33], and can also make use of GPUs to increase processing speed.

Both Theano and Tensoflow offer very fine control over the architecture of the system and therefore how it learns. However, the cost of such control is the increased difficulty in operating them, the expertise required was not attainable in the limited time available and as a result a wrapper was used. Tflearn is a wrapper for TensorFlow while Keras is a wrapper for TensorFlow and Theano. Both wrappers make their neural network libraries (backends) far simpler to use. Keras is used later for the implementation of DQN see 9.3 and Tflearn for DDPG see 9.2.

Henry Gaskin

PyBrain is a lightweight reinforcement learning library with neural networking. This library was created by Dalle Molle Institute for Artificial Intelligence Research [34]. TensorFlow and Theano are used for general neural network applications and are not as streamlined or focused as the PyBrain project. This was probably the reason PyBrain was the machine learning library of choice used by the group studies students two years previous [19]. Unfortunately this library has since been abandoned and the documentation has been left incomplete.

8.1.5 Conclusions of Research

Henry Gaskin

The research into reinforcement learning has provided insight into the possible implementations for the problem. However, it is not clear which approach will be most successful for the application of a robot on a swing. For this reason, it was decided to implement five different algorithms and compare their performance. The five algorithms chosen were State-Action-Reward-State-Action (SARSA), Q-learning, actor-critic learning, deep Q-networks (DQN) and deep deterministic policy gradients (DDPG). DQN and DDPG use neural networks to provide a continuous state space. The implementation of DQN uses Theano with the Keras API for ease of use and computational speed. The DDPG algorithm uses TensorFlow with the TfLearn API. DQN and DDPG are discussed further in Section 9. The other algorithms perform a discrete approximation and are discussed in this section.

8.2 Learning Environments

Henry Gaskin

8.2.1 Introduction

To prove that a reinforcement learning algorithm is working, simple benchmark tests are run. These tests are performed in environments. An environment provides an agent with all of the information it needs to perform and succeed. An agent and its environment make up a closed system, the agent can only interact with the environment and the environment can only can only communicate with the agent. An environment supplies an agent with state information and the reward value of the last action. An environment may also notify the agent if it reaches a terminal state, such as a win or a loss. An agent processes this information and then makes its next action to receive new information. Whether an agent performs a random explorative action or an action influenced by their received information depends on the specifics of the algorithm used. Each action performed, and the associated receipt of observations, is called a step. Once an agent has reached a terminal state, the agent requests that the environment be reset so it can perform another run. Each run of an environment is called an epoch or episode. For environments without terminal states, an epoch is usually defined as a set number of steps, after which the environment is reset. Even in environments with terminal states, there is usually a maximum number of steps before an epoch is terminated as a failure and the environment resets.

8.2.2 Implementation Strategy

Machine learning algorithms often take a long number of epochs to optimise a task. In tasks that are trivial to humans, such as traversing a maze, this can appear strange. In more complex problems, the computer's ability to perform a large number of actions in a small amount of time is more beneficial. Even then, as complexity increases, the number of epochs or steps required also increases. In a real-time system, a step can take much longer than a step of the same system simulated on a computer. For these reasons, before our algorithms were considered for use on the NAO robot, they were first implemented on environments of increasing similarity to the NAO swing problem.

Figure 8.1: The environment is completed when the agent traverses the lake from the start, 'S', to the goal 'G'.

OpenAI is a non-profit research company with the goal of making advances in artificial intelligence that benefit the human experience [35]. They have released a reinforcement learning platform called OpenAI Gym [36]. This platform provides standardised environments that can be used to test an agent's performance and inter-disciplinary flexibility. These environments also provide pre-made graphics to visualise the environments. The frozen lake and inverted pendulum environments have been used from this library. Further experimentation was performed on a dumbbell pendulum model provided by the Numerical Modelling subgroup. Finally, the reinforcement learning algorithms were tested on an environment working with a Webots simulation of a NAO robot on a swing.

8.2.3 OpenAI Gym: Frozen Lake

In the frozen lake environment, an agent moves around a grid and attempts to reach a goal state [37]. There are both terminal win and loss states for this environment. This type of environment is often called a grid world. The agent only has knowledge of this grid, so from their perspective, the grid is their universe, hence 'grid world'. The frozen lake narrative provides a context to the rules of the environment. Each epoch ends with the agent either reaching their desired location on the lake, or falling through one of the holes into the icy waters. Additionally, there is a stochastic element to the agent's motions. Due to the slippery ice, there is a chance that the agent will not be transferred to the state it expects to get to after an action. Thus, an agent must perform multiple epochs to see these effects and counter them. A learned agent can be seen giving holes a wide berth for fear of 'slipping'. At the end of an epoch, the agent is given a reward of 1 if it reaches a goal and no reward in any other state. Grid world based environments, like the frozen lake, are often used for debugging purposes to see if an algorithm is working as expected. As such it is not included in the discussion of all following algorithm implementations, as there are other, less trivial results to analyse. Figure 8.1 shows the graphical output of the environment. The tiles are lettered as follows, 'S' is where the agent starts an epoch, 'G' is the goal and 'H' are the holes in the frozen lake 'F'. The location of the agent on the grid is given by a pink highlight.

8.2.4 OpenAI Gym: Inverted Pendulum

Inverting a pendulum is a classical feedback control problem that has been adapted for use in reinforcement learning [38, Chapter 5.21.6.2]. The OpenAI Gym platform provides a widely documented environment for this problem [39]. The objective of this environment is to keep a pendulum balanced upright by applying clockwise or counter-clockwise torques. Both actions and state variables for this system are continuous. Algorithms that can't accommodate continuous values must make approximations for their discrete variables. There are no terminal states in this environment so the length of an epoch is set by the algorithm. The environment supplies an agent with a reward for each action it performs. The reward supplied is always negative. This encourages an agent to perform more explorative actions. The reward is proportional to the angle of the pendulum. The more inverted the pendulum, the better the reward. The agent attempts to maximise its reward by maximising the amount of time it spends in a region of low negative reward. When the reset function is called, the environment resets the pendulum to a random position and velocity. The ability to run this simulation faster than real time

Figure 8.2: Two examples of the inverted pendulum environment are shown. To swing the pendulum up, the agent must apply large torques. To keep the pendulum balanced, smaller, alternating torques are applied.

allows for large amounts of training that would not be feasible in a physical inverted pendulum implementation. Figure 8.2 shows the the graphical output of the OpenAI Gym pendulum environment.

8.2.5 Dumbbell Pendulum

A numerical model of a dumbbell on the end of a pendulum was provided by the Numerical Modelling subgroup. This model was formatted into an environment similar to those from the OpenAI Gym platform. This similarity allows for swift transition of agents to the new environment. The pendulum swings freely and action is applied to the rotation of the dumbbell. The reward function of this model is based on the simulated energy of the pendulum. The kinetic energy and potential energy of the pendulum are compared to a target potential energy at a target angle. This target was placed at 45°. As the total energy of the pendulum increases, towards the target energy, the reward increases. The reward then decreases as the energy of the pendulum increases beyond the target energy. A reward function shapes the objective of a environment in the eyes of an agent. Thus, the objective of this environment is to maintain a swing of 45° from any provided starting angle and velocity. Resetting the environment sets the pendulum to a random position. Unlike the inverted pendulum environment, the pendulum in the dumbbell environment isn't damped. This means that the agent needs to learn how to increase and decrease the energy of the pendulum if it wants to maximise its reward. The action supplied to the environment applies a positive or negative rotational force to the dumbbell. The magnitude of this force is limited to prevent the agent from being able to swing the pendulum to the target angle and sustain the position with constant force on the dumbbell.
The dumbbell model was closer to a real swinging exercise than an inverted pendulum, but still contained discrepancies. To accommodate these differences, an additional environment could have been created. Instead, to save time, the dumbbell model was extended. A version of the dumbbell environment was created in which a force cannot be reapplied until a torque has been applied in the opposite direction. This simulates the limited motion of the legs of a humanoid.
A graphical output was created for this environment and is shown in Figure 8.3. The black line represents the pendulum and the red lines describe the target swing. The dumbbell itself is not visualised as the numerical model used only involves the forces applied to the dumbbell so specifics of the dynamics are unavailable.

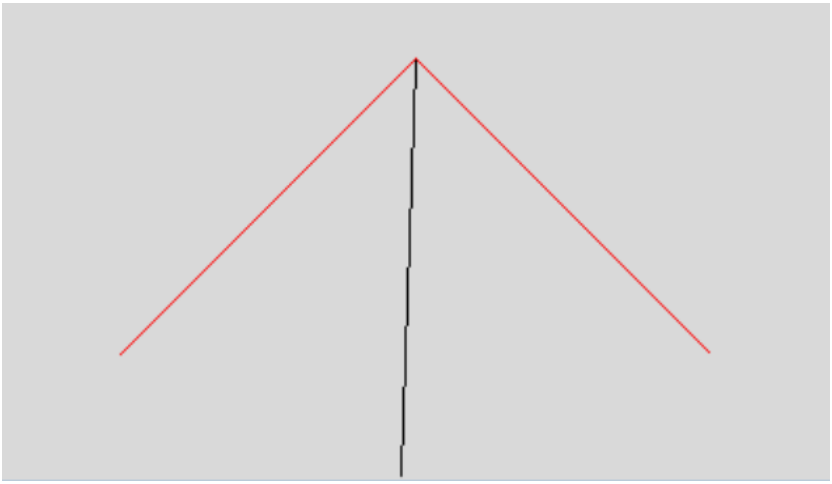


Figure 8.3: The black pendulum begins at a random position beneath the red target angles. The agent then attempts to maximise its reward by increasing the total energy of the pendulum until it matches that of a stationary pendulum at the red target angle.

8.2.6 Webots

There were many iterative attempts to create an environment fit for a reinforcement learning algorithm. The dynamics of the NAO robot are position based. All agents up to this point had only used torque based environments for dynamical systems. An environment was constructed in which the use of torque on the robot was emulated using small discrete angle movements of custom speeds. There is a limit on the number of actions that can be sent to the NAO robot before the connection is unexpectedly severed. This issue can only be resolved by reverting the world environment from inside the Webots software and then terminating the NAOqi process in a task manager program. The torque emulation version of the environment sends 100 commands to the robot each second. The limit on the number of commands that can be sent is around 12000. As such this version was abandoned as it can only run for two minutes.
A new version of the environment was created where the robot alternated between the two positions. These positions are the limits of the range of motion of the physical robot, shown in Section 3.4.2. The robot can only be sent actions when it has finished its last motion. This increases the time the simulation can be run to around 30 minutes. A reset function was built for this environment but implemented too late to be used in any tests. Once called by the agent, the function revertd the Webots world and reconnected the robot. Before the creation of this function, this process was done manually. Most algorithms were able to save their progress between executions. If the algorithm were to crash, a previous state could be loaded. The position of the swing was provided by the Webots program. The velocity was calculated within the Python environment. Unfortunately, the Webots program would often slow down. The velocity calculations assume the dynamics of the system are simulated in real-time, but this is not always the case. As such, the state variables provided to the agent cannot be guaranteed to be accurate.

8.2.7 Method Evaluation

Using the OpenAI Gym environments provided reinforcement learning environments that had already been tested by researchers from other institutions. The API is very usable and would be recommended to any future researchers. The environments produced ‘in-house’ sought to emulate this usability with similar function names and program structure. This method increased productivity and environment turnover. There were still issues integrating the machine learning algorithms with the newer environments that could have been avoided by directly inheriting from the OpenAI Gym API when creating new environments.
The only numerical model converted into a reinforcement learning environment was the dumbbell pendulum.

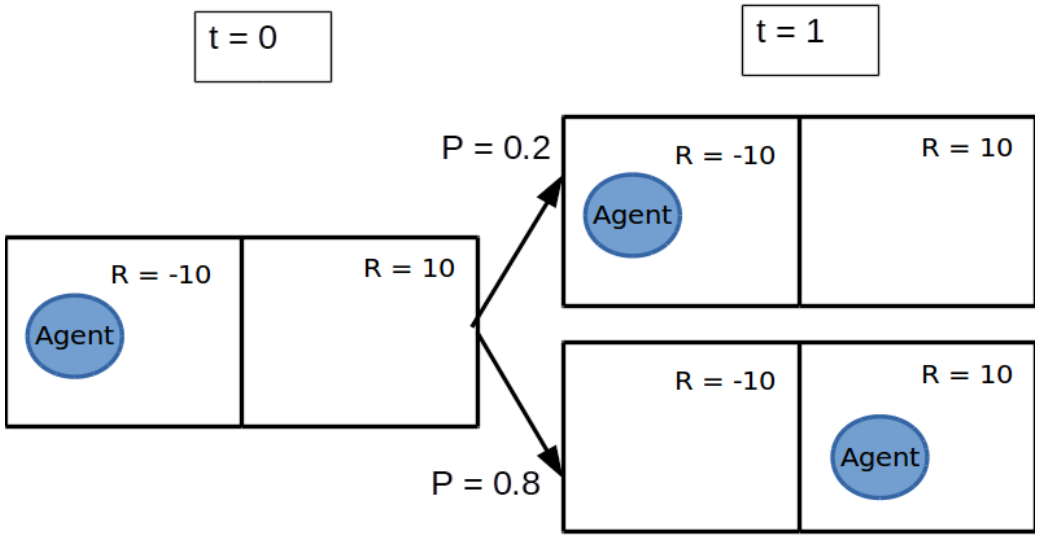


Figure 8.4: An environment with two states. It ends after t=1. An agent starts in the left state. There is a probability of 0.2 that the agent chooses to remain in the left state and of 0.8 it moves to the right state.

This model was produced relatively early in the project. More advanced and applicable models were later created by the Numerical Modelling subgroup. Translating these numerical models into learning environments could have proved more beneficial than getting the algorithms to run on the Webots simulation. The Webots simulation with the NAO robot does not produce an optimal environment for machine learning. The limited number of commands that could be sent to the robot, and the inability to easily reset the simulation, impair the learning capabilities of the agent. An agent is much more likely to learn in an environment that can be iterated over multiple times with minimal delay between each iteration. Using additional numerical environments built by peers would have provided additional support unavailable through propitiatory software with limited access to licenses and documentation.

8.3 Implementation of Actor-Critic Learning

Harry Shaw

8.3.1 Theory of TD(0) Actor-Critic

An Actor-Critic system is formed of one actor and one critic. The actor stores the policy

$$\pi(a|s) \tag{8.1}$$

which specifies the probability that an action a is taken in a given state s. The optimal policy maximises the reward the agent receives. The 'critic' in an Actor-Critic system modifies the policy to make it optimal.

To optimise the policy, the critic must be able to calculate the value of a state. The value of a state in Actor-Critic learning is defined as the expected total future reward if the policy is followed from that state.

Formally, if at $t = 0$ the agent is in some state s in the state space S, the value of that state is given by

$$V(s) = E\left[\sum_{k=0}^{\infty} \gamma^k R_{t=k+1} \mid S_{t=0} = s\right] \tag{8.2}$$

where R_t is the reward of a state the agent is in at time t and $0 < \gamma < 1$ is the discount factor. When $\gamma \approx 0$, only the most immediate rewards contribute to the value calculation (the agent is 'short sighted'), while when $\gamma \approx 1$, rewards further away in time contribute more (the agent takes a longer term view). The possible states at time t depend on the actions taken, the probabilities of which are determined by the policy in the actor. For example, consider a two state environment as in fig. 8.4. In this example, t does not advance past 1, which is to say the system 'ends' after $t = 1$. There is a probability of 0.2 that the agent chooses to remain in the left state, where it will receive a reward of -10. There is a probability of 0.8 that the agent chooses to move to the right state, where it will receive a reward of 10. Therefore, according to eq. 8.2, the value of the left state is 6.

In temporal difference learning, the values of all states are calculated iteratively using the *temporal difference error* (TD error). The critic used in this section uses the TD(0) algorithm. The first step of this algorithm is to calculate the temporal difference error:

1. The agent starts in some state, called s_t.
2. The agent carries out a specific action with a probability determined by the policy. The agent is now in the resulting state s_{t+1}.
3. The reward of this action is measured along with the value of the resulting state s_{t+1}.
4. This is compared to the current estimate of the value of state s_t, which is equal to the expected future reward of following the current policy.

Formally, the TD error σ_s, due to performing an action at time t in state s_t which leads to state s_{t+1}, is given by

$$\sigma_s = \underbrace{R_{t+1} + \gamma V(s_{t+1})}_{\text{Observed value}} - \underbrace{V(s_t)}_{\text{Predicted value}} . \tag{8.3}$$

Since this is equal the difference between the observed value of the state and the current estimate, adding this to the current estimate of the value will improve that estimate. TD(0) specifies that the new value estimate $V_{\text{new}}(s)$ is given by

$$V_{\text{new}}(s) = V(s) + \alpha\sigma_s \,, \tag{8.4}$$

where $0 < \alpha < 1$ is the *learning rate*, and affects how fast the algorithm will converge to the value function. The optimum α depends on the environment.

The TD error is also used by the actor to improve the policy:

- Eq. 8.2 defines the value of a state as the total expected future reward if the policy is followed from that state.
- The TD error is the difference between the observed value of a state and the current estimate for the value of that state.
- Therefore, if the TD error is positive, then the action taken will lead to greater rewards than the expected reward from following this policy.
- Therefore, the policy should be modified so that action is more likely to be taken in that state. Conversely, if the TD error is negative, this action should happen less often in this state [29].

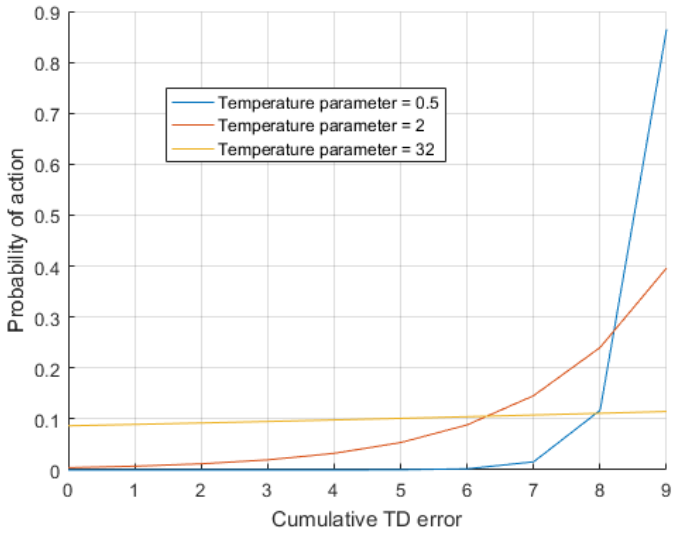


Figure 8.5: The effect of three different temperature parameters τ on the probability of different actions with a given cumulative TD error. A larger temperature parameter leads to more equal probabilities despite different cumulative TD errors.

There are many ways of calculating the probabilities using the TD error. In the method used in this section, the cumulative TD error

$$\epsilon_{a,s} = \sum_t \sigma_{a,s}\,, \tag{8.5}$$

calculated when a is performed in s multiple times, is stored for each pair of a and s. Using this, the probability that action a is taken in state s can be determined by a Boltzmann distribution

$$\pi(a|s) = \frac{\exp(\epsilon_{a,s}/\tau)}{\sum_a \exp(\epsilon_{a,s}/\tau)}\,, \tag{8.6}$$

where the denominator is a sum over all actions in that state and τ is the temperature parameter, named for its similarity to the kT term in the Boltzmann distribution in thermodynamics. If $\tau >> \epsilon_{a,s}$ then all actions are equally likely, regardless of the cumulative TD error. If $\tau \approx 0$, then the action with the most positive cumulative TD error will always be taken in a state. This is illustrated by fig. 8.5.

8.3.2 Implementation

Actor-Critic learning was implemented in Python in `ActorCritic.py`, `_Actor.py` and `_Critic.py`. The `ActorCritic` object contains an `Actor` object and a `Critic` object.

The `Actor` contains the cumulative TD error for each action in each state, and the policy. The `Critic` contains the current estimate for the value of each state. The following process is used to learn:

1. The agent is in some state
2. The the `Actor` method `getNextAction` returns the next action the agent will take using the policy.
3. The agent performs this action. This depends on the environment.
4. The `Critic` method `critique` updates the current value estimates using eq. 8.2 and updates the policy using the process in sec. 8.3

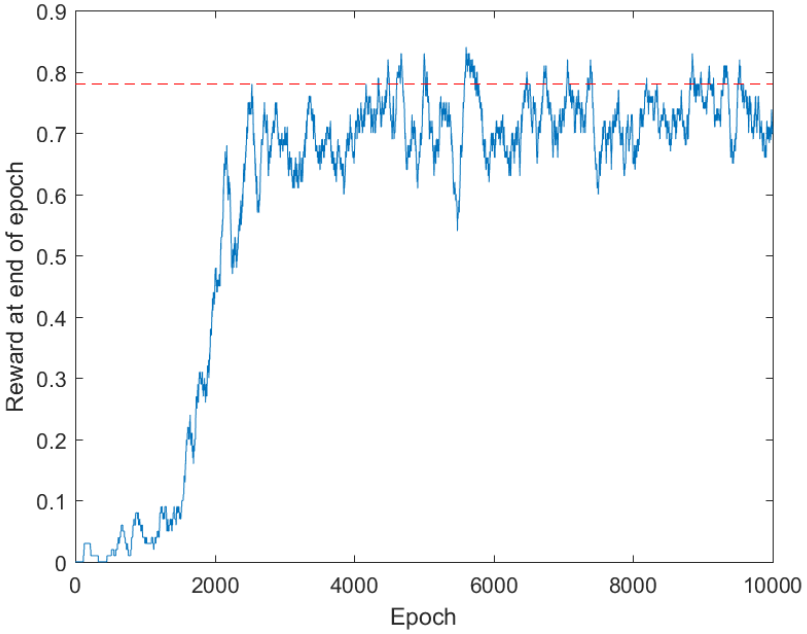


Figure 8.6: 100-point moving average of the reward after an epoch on the grid world environment. The threshold for "solving" [37] is shown.

8.3.3 Evaluation of Implementation on a Grid World

Fig. 8.6 shows the performance of the implementation with $\gamma = 0.9$, $\alpha = 0.5$ and $\tau = 2$. The grid world is defined as solved when the 100-point moving average of the reward of an epoch exceeds 0.78 [37], which it first does in the interval centred on epoch 2548 (which took approximately 3 seconds). The moving average then oscillates about 0.7, crossing 0.78 multiple times again. This is expected if the environment has been successfully solved because the environment is partly stochastic (for instance, if the agent attempts to move down, it may move in a different direction). Therefore, the implementation has successfully solved the environment.

8.3.4 Evaluation of Implementation on an Inverted Pendulum

Since the inverted pendulum has a continuous state space, the angles and velocities are rounded to a discrete set of values. The inverted pendulum also has a continuous action space, but the implementation will only output discrete values of torque. This is implemented in `_ActorCritcGymTest.py`. Due to its similarity to the problem of swinging the robot, a deeper investigation was carried out.

Fig. 8.7 shows the cumulative reward received during each epoch with different discount factors with $\tau = 10$ and $\alpha = 0.5$. It can be seen that if the discount factor $\gamma \geq 0.9$, the 50-point moving average cumulative reward approaches -300, which corresponds to as rapid inversion of the pendulum as possible. Therefore, the implementation learnt the inverted pendulum system successfully.

For discount factor γ less than 0, the algorithm doesn't consider states far enough into the future. If the pendulum is close to the lower vertical and moving slowly, the algorithm was not be able to "see" that if it oscillated enough, it would have reached a higher reward state.

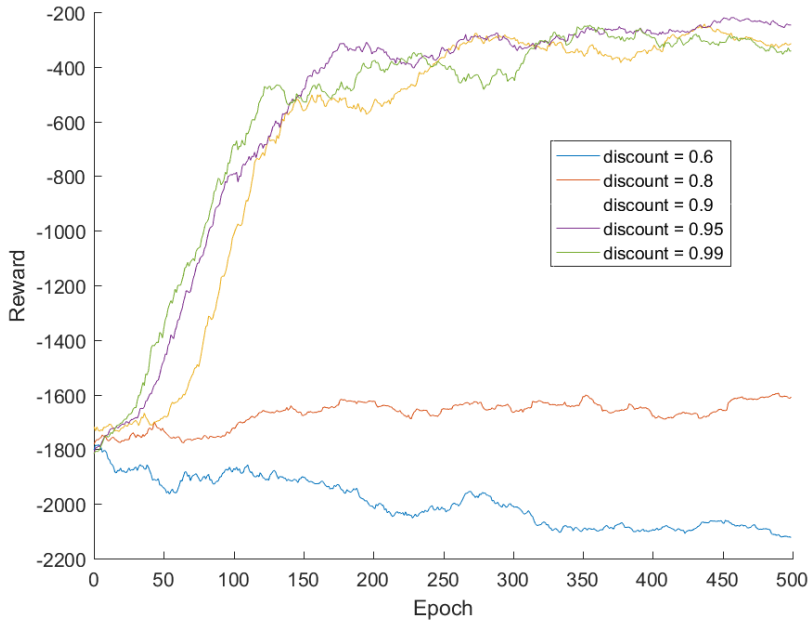


Figure 8.7: 50-point moving average of reward gained during a given epoch in the inverted pendulum environment.

8.3.5 Evaluation of Implementation on dumbbell pendulum

As with the inverted pendulum, the angles, velocities and actions are continuous in the environment but have to be discretised for the implementation. This is implemented in `_ActorCriticDumbbellTest.py`.

The system did not learn the dumbbell environment with any combination of discount factor γ, learning rate α and temperature parameter τ. The 100 point moving average of the cumulative reward during each epoch oscillated between -3000 and -6000. This is shown in fig. 8.8. To make the algorithm converge, the learning rate α was made to linearly decrease from 1 in the first epoch to 0 in the final epoch. This made the algorithm converge to an average reward of -3000 per epoch, which corresponded to an oscillation with an amplitude of approximately two degrees above the target amplitude. This is also shown in fig. 8.8. One possibility is that the system "over learned", which is to say that there are many optimal policies, and the algorithm oscillated between them. If the learning rate decreased with time, the algorithm should have began to converge on one. However, for an unknown reason, it converged to a flawed policy.

8.3.6 Application to the Swinging of the Robot

The implementation stores discrete values (as defined in eq. 8.2) and a policy with probabilities for each action in each state. For an environment with a continuous state space with many dimensions and a continuous action space with many dimensions, there are two major limiting factors. The first is that storing the policy and the value estimates requires too much memory. The second is that it may take too long to fill the table of values and update all the policy values to the optimum policy. When performing a swinging motion, the NAO robot has many degrees of freedom. Each of these can be actuated continuously, so each contribute a dimension to both the action space and the state space. Therefore, it is not feasible to use this implementation on the NAO robot.

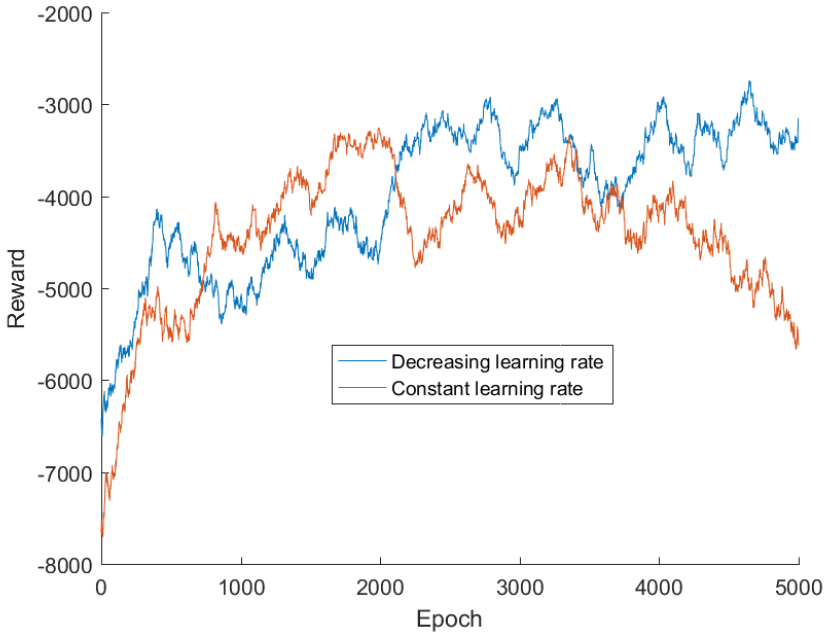


Figure 8.8: 200-point moving average of the reward gained during a given epoch in the dumbbell pendulum environment, with a constant learning rate and a linearly decreasing learning rate.

8.4 Implementation of SARSA

Jay Morris

SARSA is an on-policy reinforcement learning algorithm that is generally less used and less well documented than Q-Learning and Actor-Critic algorithms. However, it was an interesting point of comparison with the other algorithms and thus implementing it was a crucial part of the "no stone unturned" approach intended to lay a foundation for future years.

8.4.1 Q-Values

Many Machine Learning algorithms make use of Q-Values. Q-Values are most simply defined as the total expected future reward of any state-action pair given that the current policy is followed. More formally, if the agent is in some state s of state space S, and takes some action a from potential actions A, at $t = 0$, the Q-Value for that state-action pair is given by

$$Q(s,a) = E\left[\sum_{k=0}^{\infty} \gamma^k R_{t=k+1} \mid S_{t=0} = s, A_{t=0} = a\right] \tag{8.7}$$

Where R_t is the reward of the state the agent is in at time t, and γ is the discount factor, which can take some value $0 < \gamma < 1$. This factor governs the weight placed on future rewards as opposed to immediate ones. For example, when $\gamma = 0$, the agent will only consider the reward for the current step, not taking into account the future at all.Higher values increase the agent's ability to consider the rewards that potentially result from the choice.

Considering the simple two-state environment shown in fig. 8.9, which differs from fig. 8.4 by allowing the

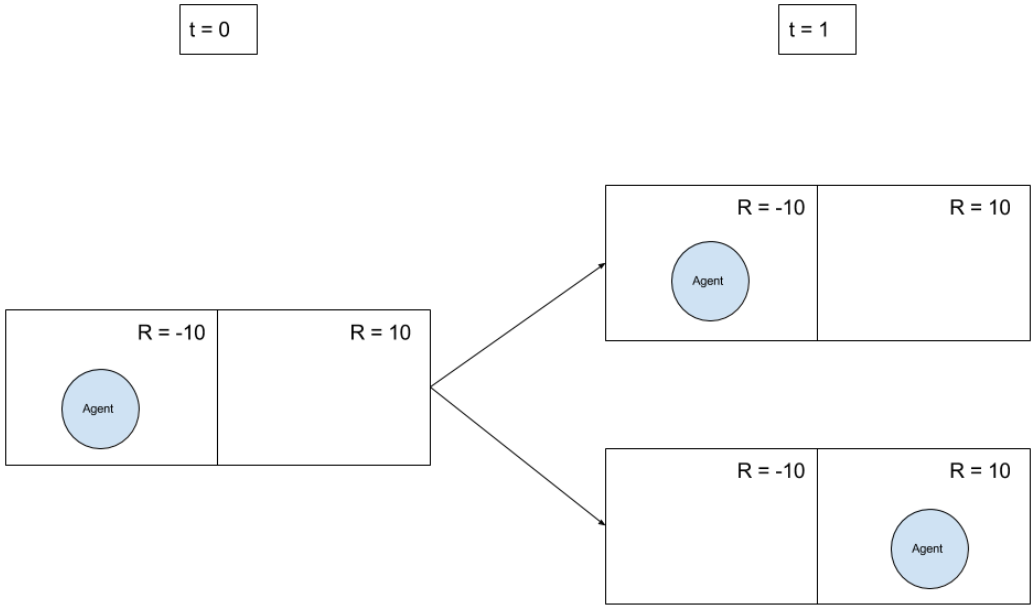


Figure 8.9: A simple environment with two states. The agent begins in the left state at $t = 0$ and will terminate after $t = 1$. It may take an action to move to either of the states.

agent a choice of actions rather than having them be probabilistically determined, as an example. The agent begins in the left state, and may choose to either remain there, or move to the right state. The Q-Value of the left state in conjunction remaining there is thus given as -20 (the sum of the left state's intrinsic reward of -10, and the expected reward of the action, which in this simple system is clearly guaranteed to be -10). Likewise, the Q-Value of moving to the right state from the left state is 0, reasoned similarly.

8.4.2 Theory of SARSA

SARSA stands for State, Action, Reward, State, Action, after the arguments that the algorithm takes.The formal definition of the algorithm used to assign the Q-Values is

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \tag{8.8}$$

where α is the learning rate and other terms are defined similarly to the general Q-Value equation. The learning rate determines the rate at which the algorithm adjusts its Q-Values (i.e. 'learns'). When $\alpha = 0$ the Q-Values will not be updated, and the algorithm will not improve its performance at all.

From Equation 8.8 it is possible to see that the SARSA algorithm takes into account two consecutive state-action pairs when assigning a Q-Value. This allows for a degree of back-propagation of Q-Values, meaning that state-action pairs that lead towards high reward states will gain higher values.These Q-Values are then used to build up a stochastic model that will be used as the policy. This is done simply by weighting a probability distribution for a state's actions based on the relative size of their Q-Values. This means that initially for every state, taking each action is equally likely, but as rewards are discovered the algorithm will prefer the actions that are expected to either immediately or eventually provide higher rewards. The decision does remain random, so even with very high or low values of α and γ there will not be a choice that is made with absolute certainty, ensuring a balance between exploration (taking choices that may not be known to yield rewards, but may do so in the future) and exploitation (taking choices that are known to be good ones based on rewards already).

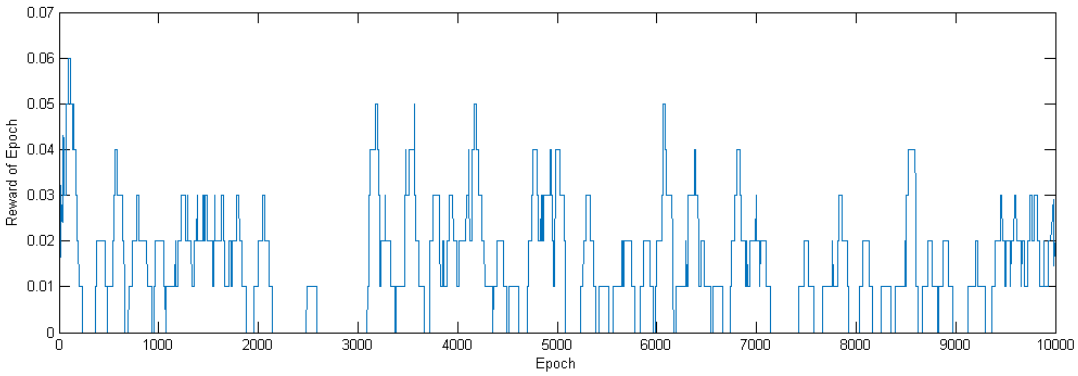


Figure 8.10: 100-point moving average of the reward after an epoch on the grid world environment.

8.4.3 Implementation

The SARSA algorithm was created in a single python class. The class contains objects that store the available actions in any state, the dimensions of the state, an array of Q-Values, a corresponding array that forms the policy. It also has values for the discount factor, the learning rate, and a temperature parameter.

The class contains a method `update_policy`, which calculates the Q-Value of the state-action pair corresponding to the input and then recalculates the policy of the corresponding state. The `get_next_action` method can then be used to randomly determine the action to be followed for a given state based on the policy.

8.4.4 Evaluation of Implementation on a Grid World

SARSA was only implemented on the AI Gym Frozen Lake Grid World environment, in the class `SARSAGridTest.py` As it failed to solve this problem, and therefore attempting to implement it to a more complicated environment would not be sensible. Fig. 8.10 shows the performance of the algorithm on the Frozen Lake using $\gamma = 0.9$, $\alpha = 0.7$, and $\tau = 2$. The moving average peaks at 0.06, but is often 0. The algorithm can clearly be seen to be achieving the goal state randomly, rather than as a result of learning.

The Q-Values that can be observed through troubleshooting appear to converge several times throughout the 10000 epochs the system is run for, meaning that instead of converging slowly on a singular optimal policy, the algorithm shifts between several different policies, without any converging to find a solution that will reliably solve the environment.

The cause of this inability to learn is unknown, as all troubleshooting by the group has found no error in either the code itself or the logic of the algorithm used. One potential flaw identified was the possibility of the algorithm 'over learning', meaning it would adjust its Q-Values from finding a reward, then adjust them again in a manner that alters the Q-Values completely upon finding another. An identified solution to this was to change the learning rate from a static variable to one that would slowly decrement over the epochs. This would mean the values would be adjusted significantly at first and then smaller changes would be made, fine-tuning the solution rather than changing it completely. Sadly this did not noticeably improve the algorithms success in the environment.

The SARSA algorithm was not intended to be applicable from the robot, instead simply being used as a comparison point for the other manual methods. Even as a failed method, it succeeds in this aspect.

8.5 Implementation of Q-Learning

Sam Bull

8.5.1 Theory of Q-Learning

To use basic Q-Learning with the robot, the continuous position of the swing must be split into discrete states such that there is a finite number of states. This can then be used to create a policy so that the robot can be in any state and know the best action to take in that state. In Q-Learning, the "best" action is determined by its Q-value, with the action having the highest Q-value being the best.

The equation to update the Q-value of a state-action pair is given by the following equation: 8.9:

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha(R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a)) \tag{8.9}$$

where $Q_i(s,a)$ is the Q-value for taking an action a in a state s, s' is the resulting state, α is the learning rate, $R(s)$ is the reward given by state s, γ is the discount factor and $max_{a'}Q_i(s',a')$ is the maximum Q-value for any action a' in state s'.

As the new Q-value of the state-action pair depends on both the previous Q-value of that state-action pair and the highest Q-value of the final state, this calculation must be repeated iteratively for all states and actions until each Q-value converges.

For all of the simulations discussed in this section, the learning rate $\alpha = 0.5$ and the discount factor $\gamma = 0.9$. These values were chosen so that the robot would learn at an intermediate rate and consider its future actions.

8.5.2 Basic Model

To begin, a basic one-dimensional simulation was created to acclimatise to the Python language and to Q-Learning. A state space of 6 states, numbered 0 to 5, was created in which the agent could select one of two actions: execute a finite force or do nothing. The goal was for the agent to be in state 5. Executing the force would move the agent towards the goal by 1 or 2 states, while doing nothing would either cause the agent to remain in the current sate or move back 1 state.

The reward function for this basic model was very simple; the reward of any of the states 0 to 4 was -1, and the reward of state 5, the goal state, was 10. This encouraged the agent to reach the goal as quickly as possible.

Once the simulation was compiling and running without errors and updating the Q-values it was deemed to be working, as the physics behind the movement was very basic and so could not be "reality-checked".

8.5.3 Numerical Model

The first accurate model of the robot was created using the Numerical Modelling subgroup's model of a dumbbell pendulum, written in C++. In order to achieve the required functionality, a specific set of functions was requested that would take arguments of the state of the robot and an action to perform and return the resulting state of the robot. One function was needed for each dimension of state space, and in this first model, the state space was defined by the robot's position and velocity. As the velocity was a vector term it included in it the direction of the motion of the robot, so therefore every possible state the robot could be in while swinging was covered.

At this stage the actions were defined within a range of torques, with 5 in total. There were 2 positive torques, 2 negative torques and a torques of zero, for no action performed. These finite torques were an

accurate representation of actions that the robot could take, as they would correspond to the robot making discrete movements with its legs and/or torso.

The reward function for this model was fairly simple; as the position of the pendulum was defined to be the angle between it and the vertical, and a position of zero was the lowest amplitude possible for the robot, the reward for a given state was defined as the absolute value of that state's position. Thus, positions further away from rest and therefore higher would have a greater reward.

The function to execute the Q-Learning algorithm was `perform_action`. It took arguments of the robot's position and velocity and a torque to exert on the robot, calculated the Q-value for that state-action pair and returned the new position the robot would be in after performing that action in that state. This enabled the function to be used iteratively from a set starting state through all subsequent states.

A graphical interface was developed so that the actions taken by a the learned robot could be easily visualised and evaluated. The graphical display was simple and at this stage showed only the pendulum and an arrow to show the timing and direction of the torque being applied with relation to the moving pendulum.

This script did undergo some learning; the Q-values were being updated and the behaviour of the simulation looked reasonable. However, because the state space was defined as the position and velocity and there were no restrictions on the actions, the agent was performing actions not common to or impossible in swinging. These actions included performing multiple actions in one swing and consecutively performing the same action, which would not be possible as there is a limit to how far the robot can move in one direction before it needs to move in the opposite direction.

Further problems were also discovered with the implementation of the reward function; the robot would accrue negative reward as it transitioned from maximum to maximum through intermediate states, leading to attempts by the simulation to levitate by quickly applying a lot of torque in one direction, which would not be a physically accurate simulation of the swinging robot.

Attempts were made to restrict the motion of the robot so that it could only apply non-zero force if the direction of that force was opposite to the direction of the last non-zero force applied. However, these changes were only made to the final motion of the robot and did not restrict the way the robot learnt; this required a new simulation to be devised.

8.5.4 Improved Model

In the second simulation, the state space was defined by the position of the robot only, specifically that at the start of a swing. The action space was two-dimensional, being defined by the value of the torque as well as the position to apply it. This ensured that the robot would be limited to one action per swing and that the reward would only be calculated based on the change in total amplitude of each swing rather than any intermediate amplitudes achieved.

This was achieved by altering the function `perform_action` such that it executed a 'for' loop to continue the pendulum's movement until it finished one swing. It did this by first checking whether the pendulum had passed the centre of its swing (where the position equalled zero) and then that the pendulum had changed the direction of its velocity.

In this improved model, to convert between the continuous state of the robot and the discrete state space used by Q-Learning, the function `index_of_state` was needed to return the index of a value of the continuous state. The position of the robot was defined as being the angle between the arms of the swing (referred to in the model as the pendulum) and the vertical, and had a range between $-\pi$ and π radians. 0 was defined to be the position of the pendulum at rest. This range was then split into a number of discrete states set by the variable `num_positions`.

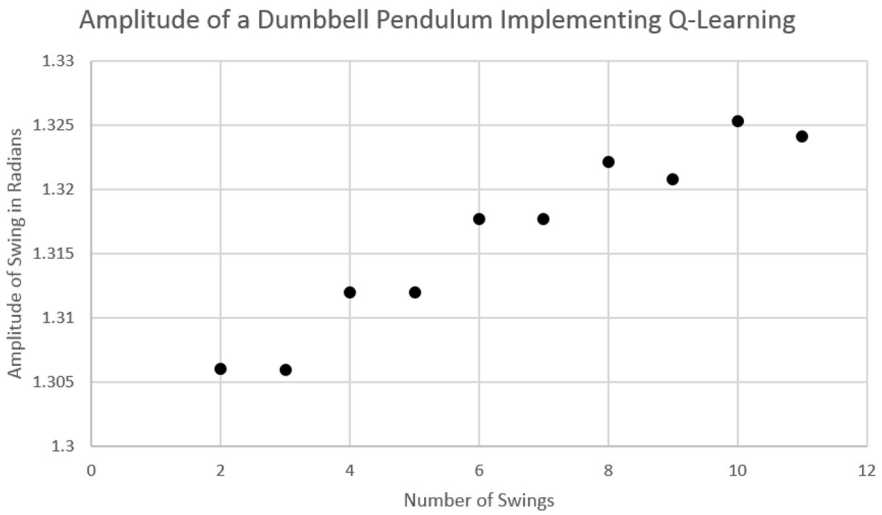


Figure 8.11: Graph to show the change in the amplitude of consecutive swings of the dumbbell pendulum simulation after undergoing 10,000 iterations of Q-Learning.

The other functions created in this final version of the simulation were `max_q_value` and `best_action`. The former takes arguments of the position of the robot, i.e. its state, and returns the value of the highest Q-value for some action that could be taken in that state. The action was not needed, as this function was used in the calculation of the new Q-value in `perform_action` only. The latter function again took the argument of the position of the robot, although in this case in the form of the state's index, and returns a tuple of the torque and position of the action with the highest Q-value for that state. This allows for the behaviour of the learnt robot to be demonstrated such that for each iteration of `perform_action`, the action with the highest Q-value is the one to be executed.

Following the completion of the Robot subgroup's measurements, this model made use of real values from the robot. The range of actions the robot could perform was defined to be moving the legs forwards, backwards or not at all. This meant that there were 3 possible values for the torque; the value measured by the Robot subgroup for the torque exerted by moving the legs in both directions and zero. These actions could be performed at any position in the swing, which is another continuous value but can use the same conversion function between state and index. This model also used the Robot subgroup's measured values for the mass of the robot and swing system and the length of the swing arms.

As the matrix holding the Q-values had a set size and shape, there will always be some Q-values in it that will not be updated. This is because not all states can apply a torque at all positions, so the Q-values where the position the torque is applied is outside of the range of positions that will be reached from the starting position are never updated.

This model has 2 versions. One is `QLearning.py`, which has all of the features mentioned but no graphical display, and the other is `QLearningUI`, which incorporates the graphical display. The latter was the one most used in developing and testing this simulation, and the former was created in preparation for transferring the simulation into Webots, as Webots would have no use for or way to interpret the graphical display.

8.5.5 Results and Evaluation

The final version of the simulation was successful, and the robot learnt a motion that consistently increased the amplitude of its swing. This can be seen clearly in Figure 8.11. The size of the state space used in this simulation was 500.

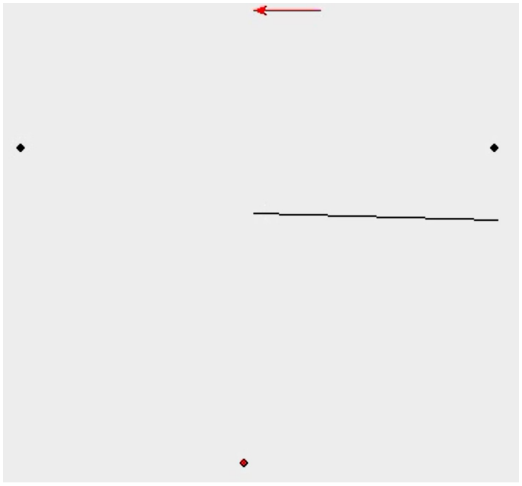


Figure 8.12: Graphical representation of the Q-Learning simulation showing the pendulum (moving from right to left) but not the dumbbell. The black dots show the initial amplitude of the swing and the red dot shows the position of the torque was applied at. The red arrow shows the direction of the applied torque.

Overall, the robot gains amplitude the longer it swings. For some unknown reason the robot only applies force in one direction, however it has learnt when and in which direction of swing to apply that force. Theoretically the robot's motion should be symmetrical, which suggests an undiscovered bug in the code making it asymmetric.

This simulation was unsuitable to be tested with the AI Gym inverted pendulum as the restrictions placed on its movements (namely, only allowing one motion per swing and not allowing the robot to swing over the fulcrum) meant that it would be unable to complete the task of inverting a pendulum.

As shown in Figure 8.12 the robot learns to apply the force at the midpoint of the swing, when the pendulum's angular velocity is greatest. This is opposite to normal human swinging where the torque is applied when the swing is at its highest point.

An attempt was made to transfer this simulation into Webots ready for use on the NAO robot, however lack of time and problems using the Webots software meant that this goal could not be realised.

Given more time, it would obviously be useful to have this simulation working in Webots and on the robot itself. It also needs to be investigated why the robot only applies torque in one direction. It would be useful to develop automated tests for this simulation and a way of measuring the rate and effectiveness of the learning; this was considered during this project but due to the time limit and complexity of the model was never considered a high enough priority.

8.6 Conclusions of manual approaches

Jay Morris

The results of the three manual approaches taken are not entirely comparable, as different models were used for Q-Learning. However, there are enough similarities that it is reasonable to talk about comparative success.Python was used for all three algorithms, and AI Gym was used for Actor-Critic and SARSA, while Q-Learning used entirely bespoke models.

The basic one-dimensional model implemented for Q-Learning is very similar to AI gym's Frozen Lake grid world environment. Despite the name, grid world's are represented as one-dimensional spaces for the agent to traverse. The added complexity therefore comes in the stochastic nature, the possibility of failure, and having

four potential actions rather than one. As this was simply used as an initial functionality test for Q-Learning, it was not investigated in depth, but the algorithm did prove able to learn in the system. In the Frozen Lake, Actor-Critic was able to solve the system, as shown in fig. 8.6, whereas SARSA could not.

Actor-Critic was tested on AI Gym's inverted pendulum, successfully solving it, as shown in fig. 8.7. This is a standard test for machine learning algorithms, so the ability to solve it is a good indicator.

Both Actor-Critic and Q-Learning were trialled on the dumbbell pendulum model devised by the Numerical Modelling subgroup. Actor-Critic did not successfully solve the environment, though in its final implementation did successfully converge to an incorrect solution. Q-Learning also partially succeeded, finding an optimal solution, though only doing so through undertaking physically impossible actions.

A more advanced model was then devised for the Q-Learning algorithm, forcing it to learn more realistic swinging motions. The algorithm succeeded in doing so, as shown in fig. 8.11. However, there is evidence of at least some amount of error in the code due to the agent only applying force in one direction, rather than symmetrically.

None of these models were applied to the robot, or even a Webots simulation within the scope of the project. Of the three, only Q-Learning would be suitable to do so, due to its different handling of state-space. The main factors preventing this direct application to a simulation of the NAO robot were a lack of time, and the extensive difficulties experienced when attempting to use the Webots software.

9 Machine Learning with Neural Networks

9.1 Background

9.1.1 Neural Networks

Harry Shaw

An artificial neural network attempts to solve machine learning problems that by creating a system similar in principle to the human brain. The network is composed of artificial neurons that send signals to each other. A neural network can be used in a Q-learning system to calculate Q values for a given state. Artificial neural networks have been trained to, for example, play video games with only pixel values and the score as inputs [32].

An artificial neural network is composed of artificial neurons, such as that in fig. 9.1. An artificial neuron receives signals, from which a weighted sum is produced. A function is then used to calculate the the output for a given input. Each state variable will have a neuron that takes that variable as an input. These will form the input layer. Each output variable will have a neuron, which will form the output layer. There maybe many layers between the output and input layers, forming a 'deep' neural network.

The weightings of the input signals for each neuron can be changed to help a network learn. First a certain set of signals is passed to the input layer. The resulting output signals are then compared to the desired output signals for that input using a loss or cost function. The difference between the desired output and the actual output is then passed back through the layers of the network, a process called Backpropagation, which gives the amount each neuron contributes to the error. The weightings for the inputs on each neurons are then changed to minimise this error. This is how the network 'learns', and is an iterative process.

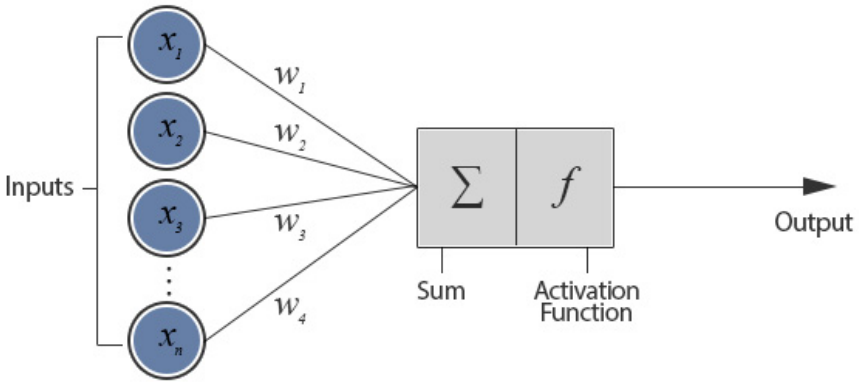


Figure 9.1: An artificial neuron. It takes a weighted sum of all input signals, from which an activation function determines the output [40].

9.1.2 Convergence Problems with Neural Networks

Mike Knee, Chris Patmore

Trying to directly adapt a classical algorithm is not without problems. Naively using a neural network to approximate a Q-value function or an actor and critic in continuous space will not produce a usable and stable algorithm. Two important changes to the learning process are needed to cause convergence and stability.

The first problem is with updating the networks based on experiences that are connected in time (i.e. updating directly as the environment progresses). Doing this causes small errors in the networks function to be compounded, and prevents convergence [41]. To avoid this problem a "replay buffer" (also called "memory buffer") is used to store past experiences. The buffer is sampled in batches, and these batches are used to update the networks rather than using the on-line state of the environment. This lets the networks learn using temporally disconnected situations, which prevents error propagation [42] [43] [44].

The next problem with using neural networks is that learning directly from results produced by a neural network that is on-line can cause the network not to converge. This is because an update that increases the value of some action in some state often also increases the the value of the next state for all actions and hence can cause poor actions to be assigned a higher Q-value. This can be avoided by using a "target" network, which calculates the target values or next state values and thus disconnects the targets from the updates, reducing the chances of oscillations or divergence. The target networks follow the actual on-line networks, but are constrained to update more slowly [45] [41].

Incorporating these two techniques into the DQN and DDPG algorithms allows the learning process to converge correctly to an optimal policy.

9.2 Continuous Control with Neural Networks

Mike Knee

It is classically[1] difficult to use reinforcement learning to solve problems with continuous state and action spaces; into which category almost all robotics problems (including a robot on a swing) fall. It is possible to discretise the state and action space, and then solve the problem using traditional reinforcement learning algorithms such as Q-learning. However, for high dimension state spaces discretising is often impractical. If the

[1] in this section the word "classically" refers to reinforcement learning and other algorithms not using neural networks

state or action space has a high number of degrees of freedom then even coarse discretisation produces a space with a very high dimensionality [42].

For example, if an action space of human legs is modelled, one leg has three joints; hip, knee and ankle. The set of total available actions is labelled as $S = \{A_1, A_2, A_3 ... A_n\}$, and one action possibility A contains the actions of each joint $A = \{a_1, a_2 ... a_m\}$, where m is the number of joints. If the available actions for each joint are discretised into five possibilities (strong forward/backward motion, weak forward/backward motion and no motion, i.e. $a = \{-K, -k, 0, k, K\}$) then the total action space size for one leg is $5^3 = 125$, and for both legs the action space is $5^6 = 15625$. As the dimensionality of the space increases the size of the action space the algorithm must process grows exponentially

$$S_n = (N_a)^D \tag{9.1}$$

where N_a is the size of one discrete action space, D is the dimensionality of the system and S_n is the size of the complete space (i.e. the number of members of S, the complete action space). In a classical Q-learning algorithm each action possibility has a Q-value associated with it when it is in each distinct state. The total number of Q-values required for a classical Q-learning algorithm to solve a high-dimension action-space problem quickly becomes very large. This impacts the speed at which these algorithms can converge on a solution, as a very large number of states and actions must be explored multiple times.

The continuous state space problem can be overcome using deep Q-networks, as discussed in 9.3. In order to solve the continuous action space problem a different algorithm using neural networks was developed by the Google Deepmind team, and is laid out in detail in [42]. This Deep Deterministic Policy Gradient (DDPG) algorithm is related to the actor-critic algorithm, and uses some similar techniques to the DQN algorithm. The difference between this algorithm and the DQN algorithm is the ability to handle continuous action and state spaces.

It is important to note that while continuous state and action spaces can be incorporated, it is not currently possible to incorporate continuous time into the algorithm. This is because the environment cannot be sampled continuously by a computer; it requires discrete time periods to take environment data in and process it before the next set of environment data is received. This limitation applies to all reinforcement learning techniques and, until processors are able to perform calculations continuously, is likely to remain a restriction.

9.2.1 Policy Gradients

One issue with many reinforcement learning problems is that of credit assignment. It is difficult to know whether an earned reward is due to the previous action taken, or due to an action taken in the past which is only now affecting the system. It is possible that the important reward-giving action took place some time ago, and every action since has had no affect. To avoid this problem policy gradient techniques can be used. Aside from this policy gradient algorithms can be applied to continuous and discrete action spaces.

The policy P outputs which action to take in some specific state, and so P can be thought of as some function, taking in the state-space as a parameter and outputting the best action

$$a = P(s) \tag{9.2}$$

in general the policy P could be stochastic, with a probability of each action in any given state. In the case discussed in this section (i.e. the DDPG algorithm) the policy is deterministic.

The policy depends on some set of parameters (given by a vector Θ) which determine how specific states produce specific actions. After a reward is given back to the agent the policy can be updated by modifying the parameters Θ to either favour the actions taken producing a positive reward or discourage actions producing

a negative reward [46]. This "backwards pass" of the earned rewards is the main feature of policy gradient techniques. In order to improve the policy, a number of actions are taken and the parameters of the policy Θ are updated towards giving greater rewards [47]. To do this we look at the gradient (i.e. differential) of the reward earned by the policy with respect to the parameter vector Θ and increase Θ in the direction of increasing reward. This is summarised by

$$\Theta_{n+1} = \Theta_n + \alpha \nabla_\Theta R(\Theta_n) \tag{9.3}$$

where α is some learning rate, that influences how strongly the gradient affects the policy, and R is the performance metric, or reward function ([29] page 265).

If the reward is only given after a large number of actions it is difficult to ascertain which actions caused the reward, and therefore which actions to encourage or discourage in the policy [41]. The policy-gradient solution to this problem is to update all the actions taken using the reward, which will mean after a large number of trials eventually the correct actions will be encouraged, because on average they are more likely to result in positive rewards. In the robot swinging system it is possible to give a reward to the agent based on the state it has currently achieved, and so in general this "credit assignment" problem is not relevant to the problem being assessed in this report.

Policy gradient techniques also work well with continuous state and action spaces, which is highly relevant to many robotics problems. There are some disadvantages to using pure policy gradient methods, notably that most policy gradient algorithms are on-policy and very susceptible to bias [48].

A drawback of classic policy gradient techniques is that they are inherently on-policy. This means information is needed about the state space of the problem, which in many cases is too complex or expansive to store in memory. The DDPG algorithm does not implement a policy gradient method in the usual sense, instead computing the gradient of increasing reward with respect to the network weights. This keeps the DDPG algorithm off-policy.

9.2.2 Actor-Critic with Neural Networks

The DDPG algorithm is closely related to actor-critic algorithms and uses two separate neural networks, to model an actor predicting optimal actions and a critic to evaluate the quality of those actions in specific states.

This is very similar to the usual actor critic algorithms, outlined in section 8.3. The critic provides information used to update and improve both the actor and the critic networks. In the DDPG algorithm the network modelling the actor takes in information about the state, and results in an action prediction. The critic takes in the same state, along with the predicted best action and outputs a Q-value for that state-action pair.

Using a neural network to simulate the actor and critic in this algorithm allows the complicated and high dimensional functions needed to correctly output a continuous action estimate and Q-value from complicated state information to be approximated and continually improved upon. The use of a neural network means we can approximate a continuous state-action function that classically could not be reproduced at all in an ordinary algorithm [42].

9.2.3 Implementing DDPG

These techniques were brought together alongside a suitable neural network implementation from [49]. The training and test algorithms are contained in the accompanying code, in the file `ActorCriticTrain.py`, in the `MachineLearning/DDPG/` folder. The `train()` method from this file is the core training loop, and is designed to be run on almost any environment provided it has a method to influence the environment via a method (usually

called `step`) which takes in an action or set of actions. This method should then return the parametrised state of the environment and a reward or measure of the usefulness/desirability of the state.

This algorithm is based on those outlined in [50], [49] and [41]; it contains modifications that allow the algorithm to run on a variety of different environments.

On each step through the train method the algorithm chooses an action by inputting the state information into the actor network. The output of the actor network will be the action for the agent to take. This action is then performed in the environment, and the new state is recorded along with the reward earned for reaching this new state. The old state, the action chosen, the reward earned and the new state are all then stored in the memory buffer of the algorithm. Once the memory buffer contains enough data sets to start learning from a sample of a specified size from the memory buffer is chosen at random. The Q-values for this sample state are then output by the critic network, and from these and the reward earned the new Q-values are found from the equation

$$Q_{n+1} = r + \psi Q_n \tag{9.4}$$

where r is the earned reward and ψ is the specified learning rate. These new Q-values are back-propagated through the critic network, so that the states and actions input now produce the updated Q-values. Next the actions that the actor would have taken in these states are evaluated, and the gradient of the critic network with respect to taking these actions in the input states is found. This gradient is then used to update the actor network. Finally the two target networks are updated, and the state is moved to the new state.

The training algorithm uses an exploration technique to allow the agent (actor and critic networks) to see a large portion of the environment state space. This increases the performance of the algorithm, and prevents it becoming stuck in low-return loops. For more details on the exploration versus exploitation problem see [29].

In the algorithm in `ActorCriticTrain.py`, exploration is implemented as a randomly generated "noise" factor added to the action chosen by the agent on each run. This decreases exponentially until it reaches a specified minimum. In [42] a different, more sophisticated, exploration process is implemented; called the Ornstein-Uhlenbeck [51] process. In the training loop the variable `epsilon` (labelled ϵ for the rest of this section) controls the range of noise added to the decided action. It is decreased iteratively in the loop by

$$\epsilon_{n+1} = \gamma \epsilon_n \tag{9.5}$$

where $0 < \gamma < 1$ is the decay factor, which can be specified to control the rate of noise decay. The amount of exploration performed by the agent can have a dramatic effect on the ability of the agent to learn, and so specifying an appropriate γ is important.

9.2.4 DDPG Learning Performance

To evaluate the performance of the algorithm it was applied to the inverted pendulum environment from OpenAI gym, outlined in section 8.2.4; and two simple models of a dumbbell, based on models developed by the Numerical Modelling subgroup. All of these have fully continuous state and action spaces. The inverted pendulum environment is a standard test of artificial intelligence performance, though it is not very similar to the task of the swinging robot. The dumbbell models are closer to the swinging robot, and are both based on a model developed by the Numerical Modelling sub-group. In all of these environments the agent has no final goal state, and so is given a set amount of time or number of actions in which to earn the highest possible reward.

9.2.4.1 Inverted Pendulum Environment

The `main()` function in `MachineLearning/DDPG/ActorCriticTrain.py` in the accompanying code files applies the agent to the inverted pendulum environment. The reward earned over one epoch was used as a measurement of the performance of the agent. For each epoch the agent was allowed 200 actions, and each full run consisted of 300 epochs. The performance of the agent in two separate runs is shown in Figure 9.2. At the start of each run the agent is created from scratch, and has no previous knowledge of the environment. In both runs the initial value of ϵ (the amount of noise) is set to the full range of allowed actions. In the case of the inverted pendulum environment this is the range $-2 \leq a \leq 2$ where a is the torque the agent can apply to the pendulum. At the start of the algorithm ϵ corresponds to the full range of actions, meaning the action is effectively chosen at random.

As ϵ decays the amount of noise added to the agent's chosen actions decreases, and the agent's action choices are made more precise. In run 1 the decay factor $\gamma = 0.95$, and in run 2 the decay factor $\gamma = 0.98$. This means run 1 spends less time exploring the environment, and more time choosing actions that the actor believes to be the most highly rewarding. ϵ decay for both runs is shown in Figure 9.3.

The plots in Figure 9.2 show that in both runs the agent has successfully learned how to perform the task. In both cases the reward earned converged to a high value, meaning that the agent had learned to quickly swing the pendulum directly up (where the reward earned is greatest) and balance the pendulum using appropriate torque applications. In both runs the algorithm has converged to a high reward by epoch 150, and both remain stable with no epochs earning a considerably worse reward.

Run 1 converged to an average earned reward of around -250, whereas run 2 converged to a slightly higher average value of around -175. Run 2 also converged before run 1, at around epoch 125 with run 1 converging at around epoch 150. This shows that the amount of exploration performed by the agent can have an effect on not only the speed at which the agent converges to a solution, but also the quality of the agents performance at the end of the training period. Having a more extended exploration period means that the agent is more likely to see actions and states that can result in higher overall reward earned. Without the exploration time the agent may not see these higher utility states if it only chooses actions based on what it believes to be the best. This is evident in Figure 9.2, as the run with a slower ϵ decay (and hence more exploration) reaches a higher average reward.

In summary the performance of the DDPG algorithm on the inverted pendulum environment was good. The agent reaches a high earned reward, and by the end of the training run was observed[2] to perform the task well. The importance of choosing an appropriate level of exploration has been demonstrated. However, there is no simple way of determining a suitable value of γ. Trial and error had to be used to settle on a value. Though this environment does not share many similarities with the robot on a swing, it nevertheless allows the performance of the DDPG algorithm to be observed.

9.2.4.2 Dumbbell Environment

The dumbbell environment consists of simulated a rigid bar with a flywheel at the end. The agent is able to apply torque to the flywheel in the range $-5 \leq a \leq 5$ where a is the torque. In this way the agent is able to influence the environment. The goal for the agent in this environment is to swing the bar at a constant specified amplitude. This is very similar to the swinging robot or a person on a swing. As before in the inverted pendulum environment the agent was given 300 epochs, each consisting of 200 actions. At the start of each run the agent is created from scratch, as previously explained. The algorithm was applied to this environment in the script `MachineLearning/DDPG/ActorCriticDumbell.py` also in the accompanying code.

The results of two separate runs in the dumbbell environment are shown in Figure 9.4. As in the inverted pendulum environment run 1 had decay factor $\gamma = 0.95$ and run 2 had $\gamma = 0.98$; the epsilon decay for the two

[2] Using the `test()` function also in `ActorCriticTrain.py`.

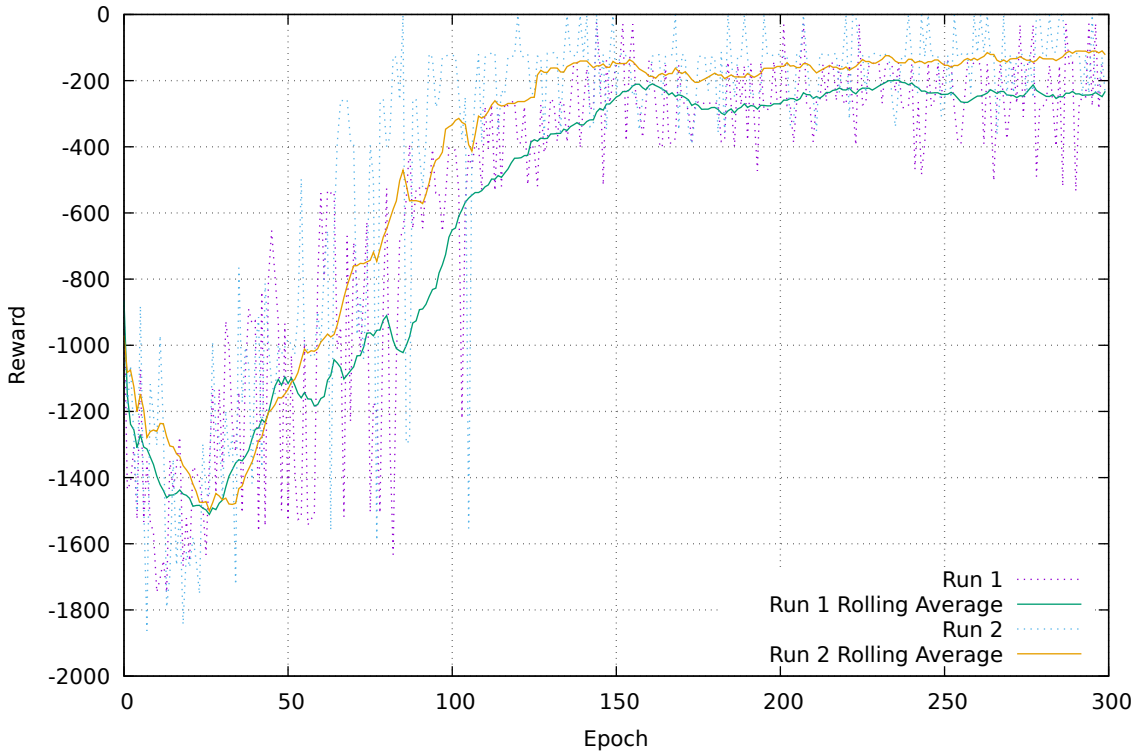


Figure 9.2: Reward and rolling average reward for two separate runs in the inverted pendulum environment. The exploration decay factor ϵ varies between the two runs, as shown in the plot below.

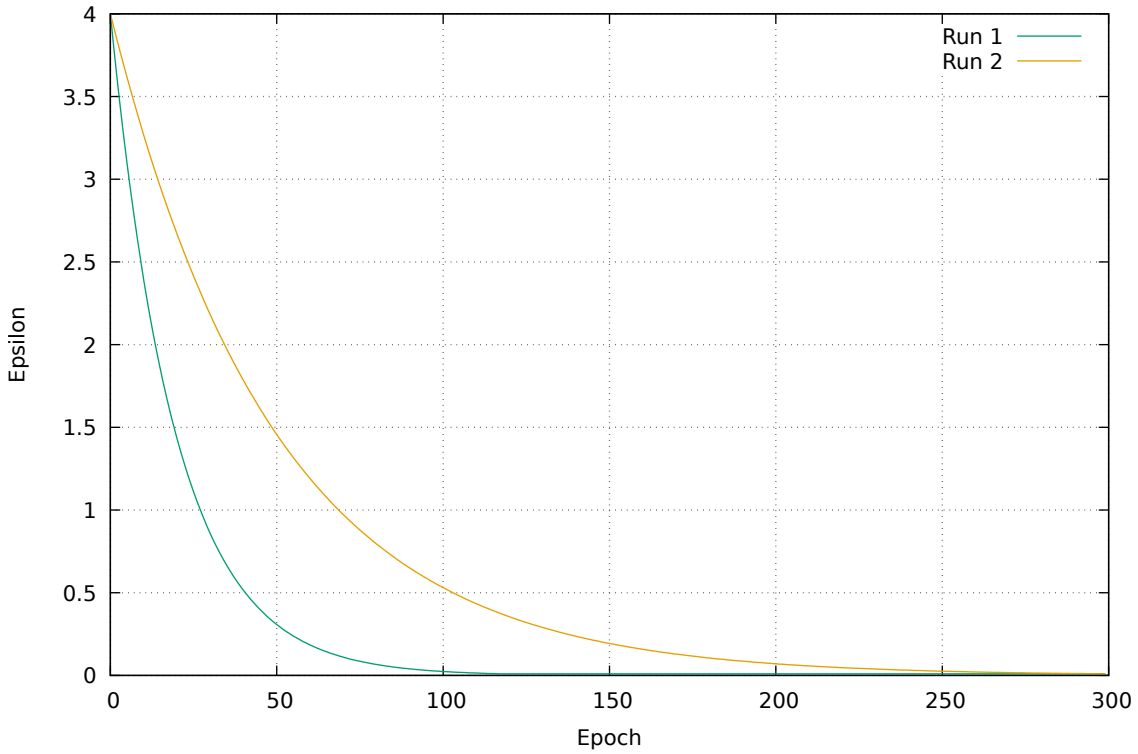


Figure 9.3: Epsilon (i.e. amount of exploration) decay for runs 1 & 2, as shown in the plot above.

Figure 9.4: Reward and rolling average reward for two separate runs in the dumbbell environment. The exploration decay factor ϵ varies between the two runs, as shown in the plot below.

runs is identical to the decay shown in Figure 9.3.

In contrast to Figure 9.2 in this environment the agent in run 1 (with the lower exploration time) did not learn to achieve high reward in the environment. The reward for this agent did not increase and converge to an acceptable solution. However, run 2 did converge to a high average reward, and produced an appropriate solution to this environment. Without being given enough time to properly explore the state and action space of the environment, the agent in run 1 did not get a chance to observe how to achieve high rewards. This means that as the agent began to pick its own actions without the noise exploration it continued to learn from poor states and actions, causing feedback and preventing the agent from learning further.

These plots dramatically display the importance of the amount of exploration performed by the agent. An appropriate amount of time must be allowed for the agent to see different aspects of the environment in order to perform properly. This exploration time must be taken into consideration when attempting to apply these learning algorithms to the NAO robot as there are certain limitations of the robot. In particular, the amount of time the robot can run for is limited, whereas an agent in a simulated environment can be allowed to run for an arbitrary amount of time.

9.2.4.3 Non-Repeating Dumbbell Environment

The non-repeating dumbbell environment is a modification of the dumbbell environment discussed in the previous section. Here the agent was restricted to only apply a fixed amount of torque simultaneously. For example, if the agent chose to apply a torque of +5, no further torque could be applied in the positive direction until some negative torque had been applied. This increased the similarity between the environment being tested and the robot swinging, as the robot is also limited to applying up to a maximum amount of torque simultaneously. However, the robot is limited by how far it can move its legs rather than how much torque can be applied. So

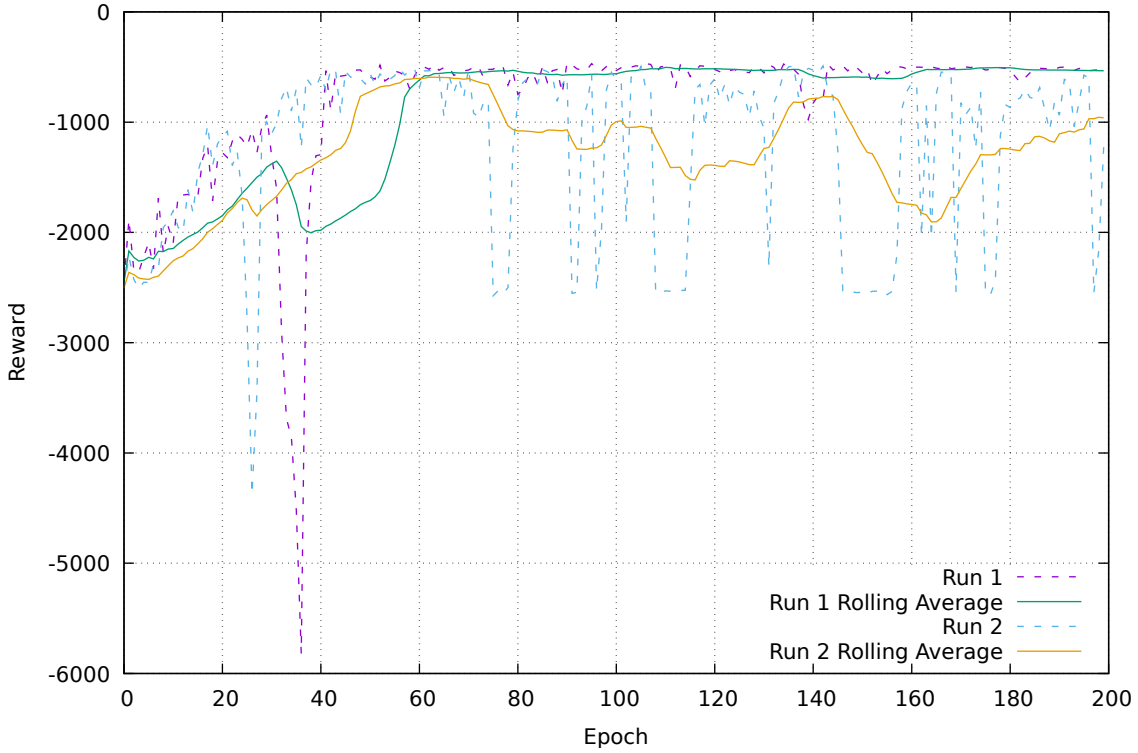


Figure 9.5: Reward and rolling average reward for two separate runs in the non-repeating dumbbell environment. The memory buffer size varies between the two runs; run 1 has memory buffer size 10000 and run 2 has memory buffer size 1000.

the environment here is not an exact parallel of the robot's environment. The algorithm was applied to this environment in the script `MachineLearning/DDPG/ActorCriticDumbellNR.py`.

For these runs it was necessary to increase the number of actions allowed in an epoch, as using 200 actions caused the agent to be unable to learn. This was because the environment was reset before the agent was able to increase its swing amplitude enough to see high reward states. The action number for an epoch was therefore increased to 500, and the number of epochs performed decreased to 200. As the affect of choosing an appropriate γ factor had already been investigated, in this environment two runs with different memory buffer sizes were performed (see section 9.1.2). The results of these two runs are shown in Figure 9.5. Run 1 in this case had a memory buffer of 10000, with initial ϵ set to the full action range ($-5 \leq a \leq 5$) as before and $\gamma = 0.98$. Run 2 had a memory buffer of 1000, and all other parameters were the same as run 1.

Figure 9.5 shows that run 2, using the lower memory buffer, performed poorly compared to run 1. Run 1 converged to a higher average reward, and stayed more stable compared to run 2. This shows that a higher memory buffer improved the ability of the agent to learn how to perform well in the environment. It also prevented the agent from forgetting how to perform well on some runs; the main cause of the large fluctuations in run 2. This has demonstrated that a larger memory buffer size can have a large effect on the ability of an agent to learn.

Run 1 was able to learn the environment well, and earned a high average reward from around epoch 50. This shows that the DDPG algorithm is able to perform well in a restricted environment, such as the one that the robot resides in. The agent was not stable in run 2, using a lower memory buffer; this meant that setting a large memory buffer is important for the learning algorithm to stabilise.

9.2.4.4 Conclusion

By applying the DDPG algorithm to these three simple environments the high performance of the algorithm was demonstrated in fully continuous environments. The agent is able to quickly learn how to maximise reward in all situations, provided it has been allowed sufficient time to explore and had a memory buffer size set appropriately. The effect of choosing appropriate exploration decay functions has also been demonstrated.

The inverted pendulum environment demonstrated the basic effectiveness of the algorithm. The agent was able to perform well in this environment even with a lower γ factor causing quick exploration decay. The dumbbell environment showed that the agent could still perform well in slightly more complex environments, but that the effect of γ was more pronounced in certain situations. Finally the non-repeating dumbbell environment has shown that the agent can perform well in limited action complex environments, providing a suitable γ is chosen and the memory buffer is set to an appropriate size.

None of these environments had the same level of complexity as the robot, which has a larger number of degrees of freedom (and therefore action space dimensions) due to the number of different joints and actuators. In the next section the appropriateness of this algorithm for the robot swinging problem will be discussed, along with necessary techniques to apply the algorithm to such an environment.

9.2.5 Application to Webots

Henry Gaskin

DDPG was the first reinforcement learning algorithm to be applied to the Webots environment. DDPG was chosen because it is the only algorithm with the ability to provide continuous actions. This would provide the robot with smooth motion similar to a human. The limitations on the Webots simulation, as described in Section 8.2.6, prevented learning from being observed. The next iteration of the Webots environment didn't use continuous actions. For this reason, integrating the DDPG algorithm was abandoned in favour of DQN.

9.3 Discrete Control with Neural Networks

Chris Patmore

Whilst the possible movements of the robot were continuous; in reality, when a person swings, they essentially switch between two different positions with very little variation. Either they switch from standing to squatting or, when seated, they change from leaning forward with their legs tucked back to leaning back with their legs outstretched. This creates a discrete action space of only three actions, either change to position one, to position two or do nothing. With a discrete set of actions it became possible to use Q-learning with a neural network to attempt to learn to swing.

Unlike in DDPG discussed previously, learning can occur using only one network. Typically however, to increase stability and aid convergence, a target network is used in much the same way as they are used in DDPG. A notable use of Q-learning based neural networks is the Deep Q-network (DQN), an algorithm developed by Google's DeepMind. DQN has been used to learn to play a vast number of video games by simply taking the pixels from the screen as the input. The algorithm is considered deep due to the number of hidden layers in the Q-network. A deep network allows for more complex functions to be learned and is often better at generalizing a solution rather than memorising the actions for every state, as with a wide network. In the case of DQN for solving arcade video games, the network is designed to emphasise splitting the view of the screen into sections, using so called convolutional layers.

Figure 9.6 shows the use of convolutional layers in the network to emphasise splitting the viewed pixels into more manageable chunks, and learning using the new abstracted view of the system. This is much like how a person recognises an entity in the game, rather than playing by analysing every individual pixel. DQN

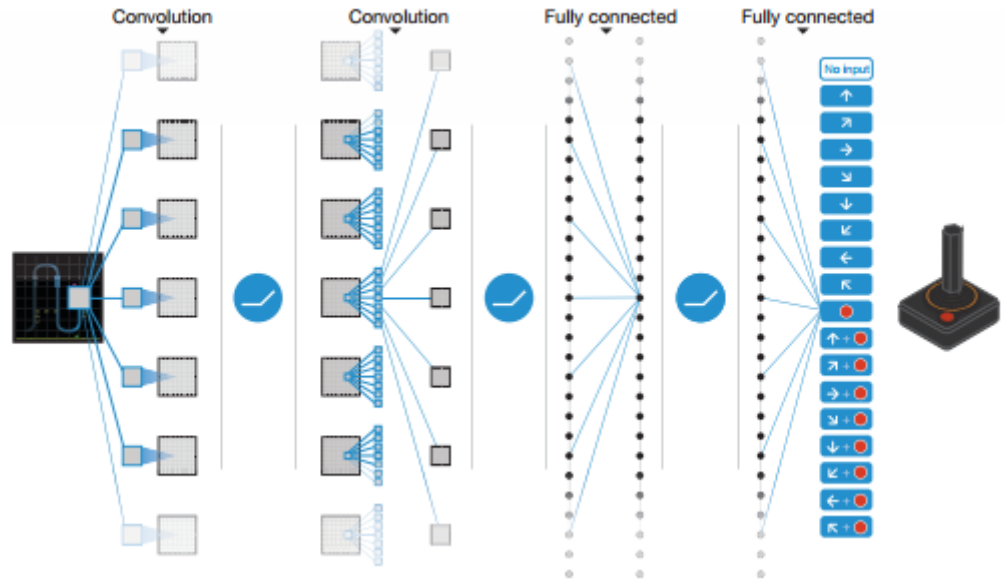


Figure 9.6: Visual representation of the neural network used by Mnih et al in DQN.

is implemented as described by Mnih et al.[45], and its performance with video games also analysed. The implementation features a customised loss function which compares the output of the network for a state, to the value of the desired output for that state.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \underset{a'}{max} Q(s', a', \theta_i^-) - Q(s, a, \theta_i)\right)^2\right] \tag{9.6}$$

In this equation θ and θ^- represent the weights of the Q-network and target network respectively, whilst $U(D)$ represents a data set of (s, a, r, s') from the replay memory being used for training.

The technique of Backpropagation, which assigns an error value to each neuron in the network from the loss computed, is used with the results from this loss function. Once these values have been assigned, the gradient of the loss function can be found and Stochastic Gradient Descent (SGD) performed on the weights and biases in the network to minimise the loss with respect to the events it was last trained on, see 9.3.1 for more detail. The events trained are a random sample from a replay memory of previously experienced events to aid convergence, see 9.1.2. Finally a target network is used to calculate the new states Q-values, its weights are updated after so many steps to the values of the Q-network such that the target network follows the Q-network, again with the intent to aid convergence.

9.3.1 Gradient Descent

If the weights and biases in a network are thought of as being variables in the loss function, to minimise the loss function and therefore find the optimal solution, it would be ideal to find various derivatives of the loss function with respect to these variables and hence analytically determine the minimum. However, this is extremely difficult when there are a large number of variables, which is the case for a neural network. This is where Gradient Descent comes in, Gradient Descent is based upon the idea of a ball rolling down a hill from the perspective of the ball. The ball will logically find the minimum of the hill, but the ball does not see the hill's shape, it simply makes a move down the hill in that instant.

To perform Gradient Descent however, it is still necessary to know the first derivative of the loss function in the area around the ball. Luckily, Backpropagation solves this problem. As previously stated, it assigns

an error to each of the neurons in the network and can therefore be used to represent the gradient of the loss function in relation to said neurons.

To perform gradient descent, the weights and biases of the neurons are changed to move down the slope of the gradient calculated using Backpropagation. Only the gradient within the immediate vicinity of the ball is known, so it cannot move directly to the minimum, its movement must be constrained to the immediate area. To this end, the learning rate η is used.

$$\Delta \mathbf{v} = -\eta \nabla C \tag{9.7}$$

Here, C is the loss function. This equation calculates the change in $\mathbf{v}$, i.e. the changes in the neurons to make (the movement of the ball) which ensure the loss function decreases with the change. To prove this, all that is required is that the change in the loss function is given by its gradient multiplied by the change in the neurons.

$$\begin{aligned} \Delta C &= \nabla C \cdot \Delta \mathbf{v} \\ \Delta C &= \nabla C \cdot -\eta \nabla C \\ \Delta C &= -\eta ||\nabla C||^2 \end{aligned}$$

Therefore, changing $\mathbf{v}$ as prescribed by Equation 9.7, guarantees reducing the loss function provided η is sufficiently small. Subsequently, if the gradient of the network is found and Gradient Descent applied repeatedly, the loss function should reach its minimum. Ideally this would be performed using all the training data available, however, this is computationally unreasonable once the number of training examples has grown sufficiently large. Thankfully SGD solves this problem, it uses a random sample of the data of a manageable size to find an approximate gradient of the loss function. This feature ties in well with replay memory and the two work extremely well together providing convergence for a neural network. This is quite a broad overview of Backpropagation and Gradient Descent, a more in depth discussion can be found in the online book Neural Networks and Deep Learning [52].

9.3.2 Implementation of DQN

This DQN agent was created featuring everything discussed previously as closely as possible, whilst adhering to the time constraints of the project. The code took a large amount of inspiration from from the work of Matthias Plappert [53] and can be found on the Git Repository Appendix **??**. In the *QNetworkMain* class; the architecture of the network, the parameters for learning and the Keras optimizer were decided from which a DQN was created. In contrast to the convolutional layers used to solve video games, only fully connected layers were used as there was little benefit to abstracting the view of the system, as the nature of the input was relatively simple when compared with the large input space of a game. Keras optimizers vary the way the loss function is optimised, for example SGD is a Keras optimizer. Here 'Adam' was used as it provides better performance than standard SGD, but is still based upon the fundamental ideas of Gradient Descent. The loss function used was the mean squared error (MSE), Equation 9.8, though it is not identical to the DQN loss function it was an extremely close representation.

$$L_i(\theta_i) = \frac{1}{n} \sum_{i=1}^{n} \Big(r + \gamma \underset{a'}{max} Q(s', a', \theta_i^-) - Q(s, a, \theta_i) \Big)^2 \tag{9.8}$$

The actual loss function from DQN was not implemented due to complexity, it however could be investigated for future projects.

The selection policy for actions and the memory for replay functionality must be initialised and passed into the DQN agent. The selection policy used here was a Boltzmann distribution based policy. An action was selected randomly but each action's chance was weighted by following a Boltzmann distribution, such that the optimal choice was most likely. The next best actions are less likely but more so than the next, and so on. The shape of the distribution was set by the variable *tau* so that the best action can either be by far the most likely or almost as likely as the worst choice, this constant should decay over training to train on states only accessible after good performance.

The replay memory used was a simple class, that stores data sets of the initial state, action, reward and next state, this meant that the data sets could be used to update the network at any time (offline). The memory was created with a maximum buffer size, beyond which old memories were overridden to ensure the program was realistically runnable. During training, a set of memories were pulled from the class, the size of which corresponded to the size of the training batch selected for the specific DQN agent.

The *QNet* class is the implementation of the DQN agent. Its main features are firstly, the *train()* method, which runs training on an environment for a given number of episodes (epochs) of a set number of steps. At each step, an action was selected following the policy and performed in the environment. The experience was then added to the memory and a batch retrieved from memory to train the Q-network. After every episode, the target network's weights were updated to those of the Q-network. The second main feature is the *test()* method, when the network was being tested it was not updated and τ was set to 1. The action selection policy therefore allowed non-optimal actions to occur, however, this was a very small chance. Correcting these random actions further displayed that the skill had been learnt properly, rather than just memorising a string of actions.

9.3.3 Performance of the DQN implementation

To allow an in depth evaluation of the performance of the DQN agent and a comparison to the performance of the DDPG agent, the agent was also applied to the inverted pendulum and the simple model of a dumbbell. It was also applied to a modified version of the dumbbell where an action could only be performed once in succession. Whilst these environments have continuous action spaces, the actions could be discretised to function with DQN. The modified dumbbell model is a closer approximation to the swinging robot than the standard version. The variation to the model meant that the agent could perform an action, say -5, and would then not be able to perform the action again till $+5$ had been applied, any attempt to do so would result in a zero action. This is similar to the robot changing between two positions. Incorporating the change meant the state information returned by the environment also had to include the current "position" of the dumbbell, or more specifically, the last action that was performed.

9.3.3.1 Inverted Pendulum

The inverted pendulum environment was the first tested using DQN, the results shown here were produced using 200 Episodes of 500 actions each. The learning rate was set to be constant at $\gamma = 0.99$ and the exploration factor of the action selection policy τ, was also constant with $\tau = 1$. The neural network architecture was an input layer of the size of the state space for the environment (three for the inverted pendulum), followed by three 16 neuron wide hidden layers, and finally, an output action space of five actions. The output actions were discrete states of the continuous actions possible. Thus given that the continuous action space was $-2 \leq a \leq 2$, equally spaced actions of -2,-1,0,1,2 were used as the possible actions for the agent.

From Figure 9.7 the algorithm is seen to learn the environment very quickly, converging to an average reward of ≈ 250 after only 20 episodes. As a measure of performance, the average reward over 100 test episodes of 200 actions was found after training had finished. For this set up the test reward was -153 ± 90, the large variation was caused by the random initial conditions for each episode.

Knowing the algorithm worked and could learn the environment, even if the environment was not particularly applicable to the robot, allowed specific features of the DQN implementation to be tested. Firstly the impact on performance when using a larger action state space and a decaying tau was investigated, see Figure 9.8. This plot shows the rolling reward when the algorithm was run with an action state space of 11 equally spaced actions, and when the exploration factor τ was set to decay from 100 to 1 by a factor of 1.2 every episode. From the figure it can be seen that having a decaying τ meant the algorithm took longer to converge, but eventually lead to an almost identical performance to that of the constant τ run, with an average test performance of -155 ± 90. This was expected, as towards the start of the run, where τ is larger, there will be far more random actions, leading to a generally worse performance until τ decreases sufficiently. When using a larger action space

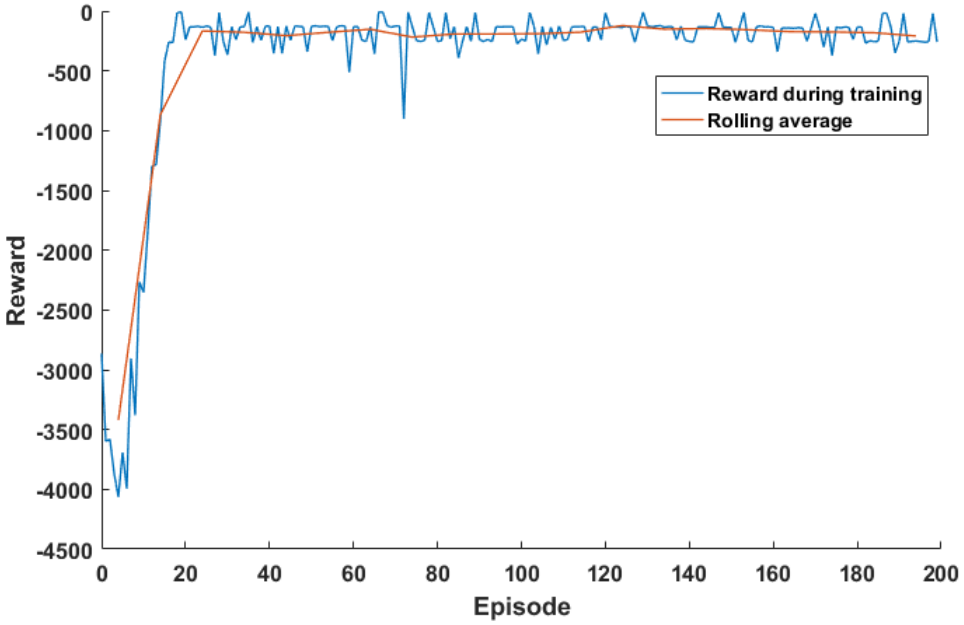


Figure 9.7: Reward and rolling average (over 10 episodes) reward for the inverted pendulum environment, using all techniques to improve performance. Each episode has 500 steps and tau is constant and unitary.

the algorithm took longer to converge, this was expected as there were more actions to explore in the visited states. The algorithm still converged, but to a slightly lower point with average test reward -190 ± 110, with increased variation in performance.

The subsequent tests were on DQN specific features added with the express intention of aiding convergence, see Figure 9.9. Firstly, the algorithm was tested without using the target network, this led to a dramatically reduced learning rate. Though the algorithm still converged to approximately the same level, with a test performance of -218 ± 144, it took over 100 more episodes to do so. This indicated that the act of disconnecting the calculation of the Q-values for the next state from the Q-network was vital for effective learning on the robot. The algorithm was then tested with the target network but without a replay memory, the effect of this change was catastrophic, learning completely stopped and the reward was essentially random. After 200 episodes there was no indication of this changing, therefore a replay memory was required if the algorithm was to be used to learn any environment.

In conclusion, this environment has demonstrated that the DQN algorithm is very capable of learning quickly and to a high performance level, and was therefore used on further environments with the aim of running it on the Webots simulation. This environment was incredibly useful for testing parameters for learning as there was no calculable optimal set up. It was a matter of trial and error to determine effective parameters. The environment was also incredibly useful for visualising the impact of adding the two convergence aiding techniques, replay memory and target networks.

9.3.3.2 Dumbbell Environment

When using the dumbbell environment with the DQN agent, the action space was limited to a torque of zero and ± 5 on the dumbbell (or analogously flywheel). The agent was rewarded most highly for achieving a swing amplitude of 45^o, with the reward varying with the energy of the centre of mass of the dumbbell at the end of the pendulum. This environment lends itself more naturally towards the swinging behaviour of the robot.

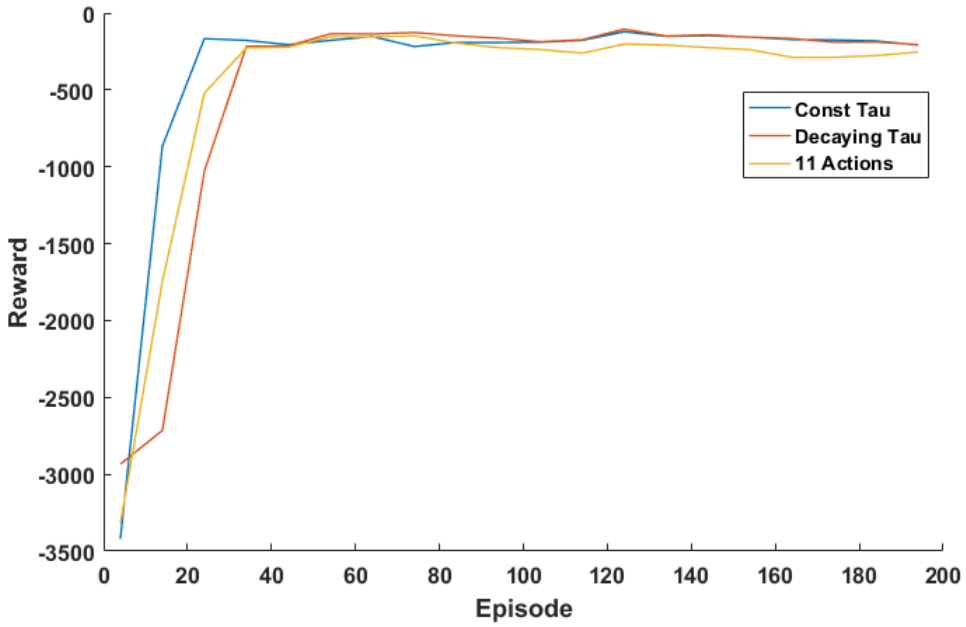


Figure 9.8: Comparison of rolling average reward for various small changes to the DQN agents main set up. The constant τ plot has $\tau = 1$, the Decaying τ plot has an initial τ of 100 decaying by a factor of 1.2 every episode till $\tau = 1$. The 11 actions plot simply has 11 possible actions.

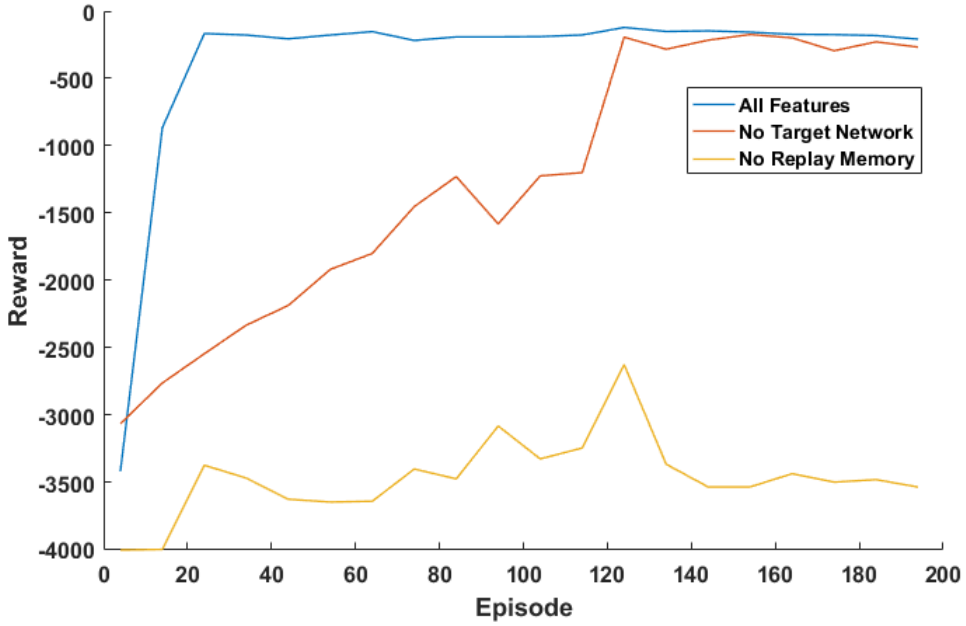


Figure 9.9: Comparison of adding various convergence aiding techniques. Shown are the learning rolling average rewards when all features are used, when there is no target network and when there is no replay memory.

For this environment 200 episodes of 500 steps were used, and for every episode, the pendulum started at some random angle below the goal angle. As with the inverted pendulum the decay factor γ was set to 0.99, however, in an attempt to increase the memory of the network, the hidden layers were increased in size to a width of 32 neurons. The average test performance was taken over 500 steps to ensure the agent had time to reach the optimal height.

Figure 9.10 shows the results for two different set ups for training the algorithm. In one, the exploration factor τ for the Boltzmann action selection policy was kept constant at 1, as in the inverted pendulum best case. In the other, τ was decayed from 100 to 1 exponentially, by a factor of 1.2 every episode. From the figure, a significant difference is seen in the performance of the two set ups, for the constant τ, learning was erratic and did not converge, it also tended to perform especially poorly at low starting angles with an average test performance of -6909 ± 5610. The decaying τ set up however, performed well, quickly converging to a good reward and performing from both high and low starting angles with a performance of -713 ± 516. This difference suggested that to learn the energy based reward function and its relation to actions and state information, more exploration was required early on in the learning process, so that the agent could discover how to improve its reward.

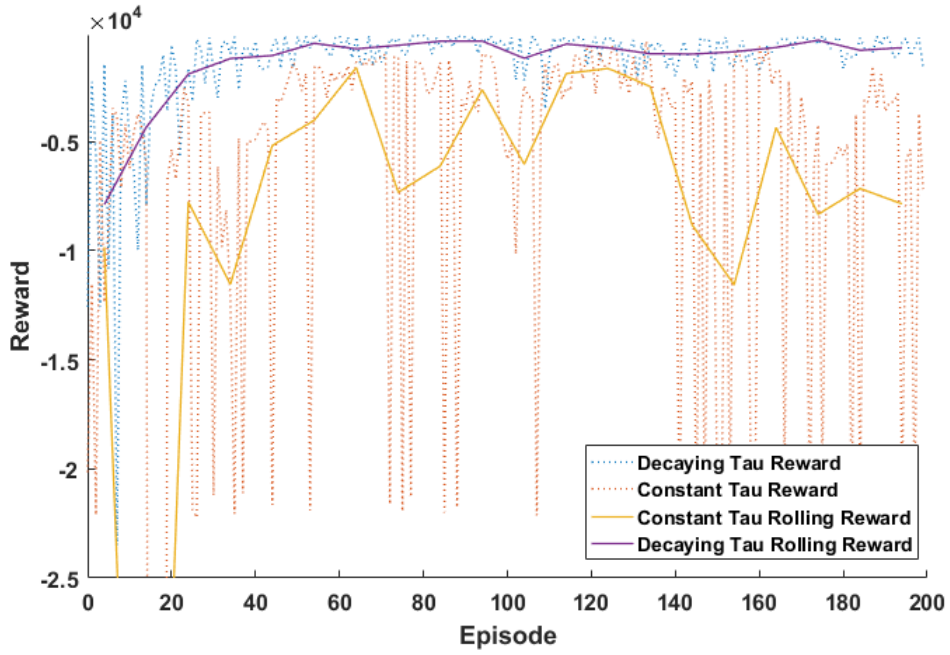


Figure 9.10: Reward and rolling average reward for two separate runs of the standard dumbbell environment. Tau varies between the two, displaying the importance of having more exploration in this environment.

Running the algorithm on this environment was especially important in terms of progressing towards getting the robot or simulation to learn to swing, as it was a much closer approximation to that environment than the inverted pendulum. It was also the first test of the reward function which would be used on Webots and the robot. Testing on this environment made it apparent that having a decreasing exploration will be instrumental in the learning process.

9.3.3.3 Non-repeating Dumbbell Environment

The non-repeating environment was created and tested with the express intention of making an even closer representation of the robot actually swinging. The exploration change that was found to aid learning in the

normal dumbbell environment, was kept for the non-repeating dumbbell, except the decay rate was lowered to 1.05 (from 1.2), as the changes made the environment much harder to learn. The changes meant the agent could perform two actions ± 5, whichever was performed last could not be performed again until the other had been called, attempting to do so resulted in a 0 action. For this environment 300 episodes of 500 steps were used with $\gamma = 0.99$ and 32 neuron wide hidden layers. Furthermore, the last action performed was included in the state space information, this meant the agent could "see" the difference between performing the same action for the second time and beyond, and the first execution of the action.

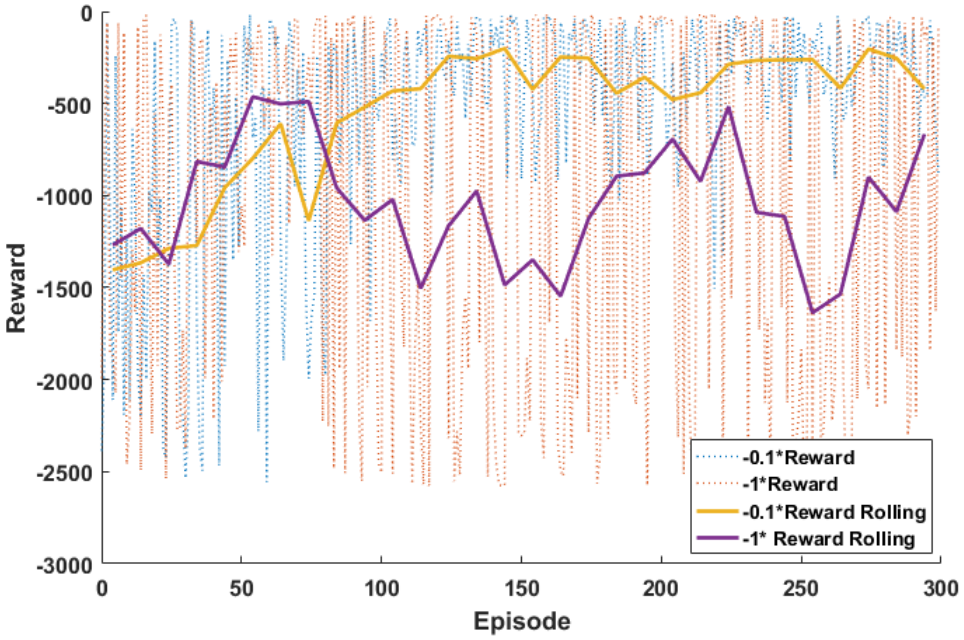


Figure 9.11: Reward and rolling average for two runs of the non-repeating dumbbell environment. This figure shows that to learn in this case the reward function had to be changed else the network would forget how to act at low angles. The non reduced reward function has been scaled for comparison.

Figure 9.11 shows the results from training on this environment, an interesting behaviour can be seen. When the algorithm was trained with the same reward function as for the normal dumbbell environment, initially it appeared to learn then completely forgot how to behave at low angles, leading to a huge range of performance for most of the training episodes and a test performance of -8191 ± 7596. This meant the algorithm did not converge and essentially produced random rewards dependent upon the starting position of the pendulum. The reward function was then changed to be reduced by a factor of 10, and the algorithm was able to learn properly, converging to around -250 and performing better for all initial angles. This lead to an average test performance of -317 ± 298 (remember reward is factor of 10 lower), far better than before. After investigation, it was discovered that this convergence problem was caused by the Q-values for the low states becoming too negative, of the order ≈ -6000 as the pendulum was spending longer in those areas than with the normal environment. This meant that the selection policy could no longer properly differentiate between the two actions, and was instead acting randomly rather than optimally.

This was an extremely important discovery as the robot will also take longer to swing up to a good reward and will therefore require the reduced reward. The issue with further reduction is, whilst it makes the worst states better and therefore worth performing well in, it makes the best solution far less obvious, meaning there is a trade off between final performance and learning rate.

9.3.3.4 Conclusion of Environments

The DQN agent performed well in all the environments outlined above achieving convergence to high rewards after a reasonable length of time. The environments have also been invaluable in outlining the importance of exploration and a good reward function for successful learning.

Furthermore, their use allowed the demonstration of the necessity of various features which aid the convergence of neural network based learning algorithms. All of this creates an extremely solid base from which to progress with network based machine learning in this group project. The environments used were simple in comparison to the robot swinging, but over the course of this project many more complex models were created by the Numerical Modelling subgroup (2), which would be ideal to further test the applicability of the DQN agent to the problem. In the following section, the applicability of DQN to the robot and Webots simulation will be discussed in greater detail.

9.3.4 Application to the NAO

Application to the Webots simulation and ultimately to the robot is tough. In terms of application to the actual robot, the largest difficulties were mainly technical. By their nature, network based algorithms are slow to run and computationally intensive, they were therefore not ideal to run on the computers which have access to the angle encoder data for the swing. This makes it extremely unlikely that this algorithm will be applied to the robot this year.

However, it was not impossible to run the code on Webots, though there were still numerous difficulties, foremost of which was the time required. It took around five minutes to run one episode of the algorithm with sufficient steps to possibly allow the simulation to build a decent swing. Meaning it would possibly take hours to see any significant progress on the simulation. Furthermore, there was no guarantee that the parameters used on the first run were conducive to learning, making the debugging of problems for the algorithm an extremely slow process. This was further compounded by the tendency of Webots to crash after approximately half an hour. This meant there was no way the simulation could be learned from scratch in any reasonable time frame.

In an attempt to overcome some of these problems, an attempt was made to use a degree of supervised learning to train the agent. Firstly, the weights of a network trained on the non-repeating dumbbell environment were saved, these weights were from an agent trained to swing in the environment without the aid of random start positions, its performance can be seen in Figure 9.12. Whilst it can be seen that the agent had learned the environment, it took a large number of episodes, further confirming that learning the Webots simulation from scratch was unlikely to yield results. Unfortunately, whilst the non-repeating environment is a reasonably close approximation of the physical system of swinging, it is a poor approximation of reality. In the simulation the swing is constantly moving and does not progress in well defined steps, moreover, there was damping on the swing and not the dumbbell environment. This meant that even after loading the weights before retraining on the simulation, the agent still failed to learn to swing in Webots within a reasonable time.

A second attempt to supervise the learning was made by filling the replay memory before training started, with experiences saved from using a numerical modelling solution on the simulation. Again the agent failed to learn, this was possibly due to reasons which will be discussed in the comparison subsection, (9.4).

9.3.5 Conclusion of DQN

DQN has been shown to learn a number of simple environments, to a high reward and in a reasonable time frame. The algorithm has also been used to demonstrate the necessity of the convergence techniques outlined in 9.1.2, from experimentation with the inverted pendulum model. The use of the dumbbell and non repeating dumbbell model highlighted the need for a well created reward function and appropriate exploration to secure

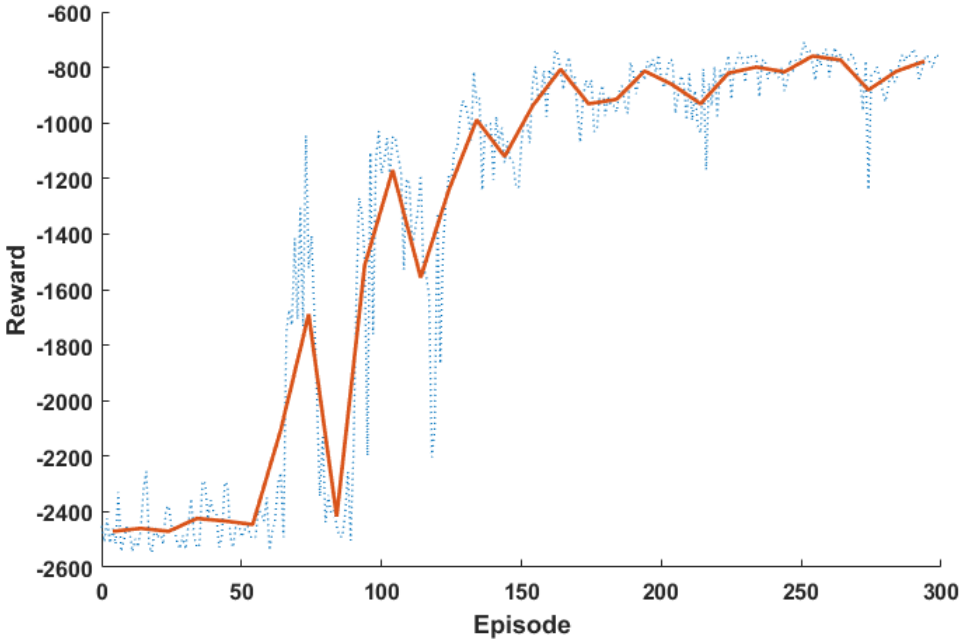


Figure 9.12: Closest representation of the non-repeating dumbbell environment to the robot as the environment would always start with the pendulum at rest at the bottom.

convergence to a high reward.

Progression onto using Webots was slow and hindered largely by the program itself, due to the programs run speed and tendency to crash. However, there were also issues with the large complexity jump between environments, from the non-repeating dumbbell to Webots. For these reasons no progress was made with learning on Webots, even after some simple attempts at supervision of the learning were made, using a previously trained network and pre loading the replay memory. Running the agent on the actual robot was not attempted due to the failure on Webots and the requirement for the code to be run on outdated computers.

With more time, the agent would have been applied to the most recent numerical models to test learning. If the agent could succeed in these environments, it would strongly suggest that with sufficient time and supervision it would be capable of learning in Webots, and on the robot. This supervision could either be in terms of using the weights for the network from one of these very close models, or by correctly pre loading experiences taken from running a simple driving code, which functions as discussed in Section 9.4.

With these two supervision techniques and optimising the reward function, there is no reason why DQN would be unable to learn to swing, therefore, the only requirement would be more time.

9.4 Comparison

Henry Gaskin, Mike Knee, Chris Patmore

Both the network algorithms worked well in the tested environments. They were able to perform the tasks in the inverted pendulum and dumbbell pendulum environments and earned a high reward after a similar amount of learning time.

The actions chosen by both of these agents were investigated, in order to observe the differences between

the actions chosen by the machine learning agents and those found analytically by the numerical models. First the agents were trained on the non-repeating dumbbell environments until they had converged on a solution. Then the action chosen by the agents were output during a test run, along with the current swing angles and current angular velocities. The results of these tests are shown in Figures 9.13 and 9.14.

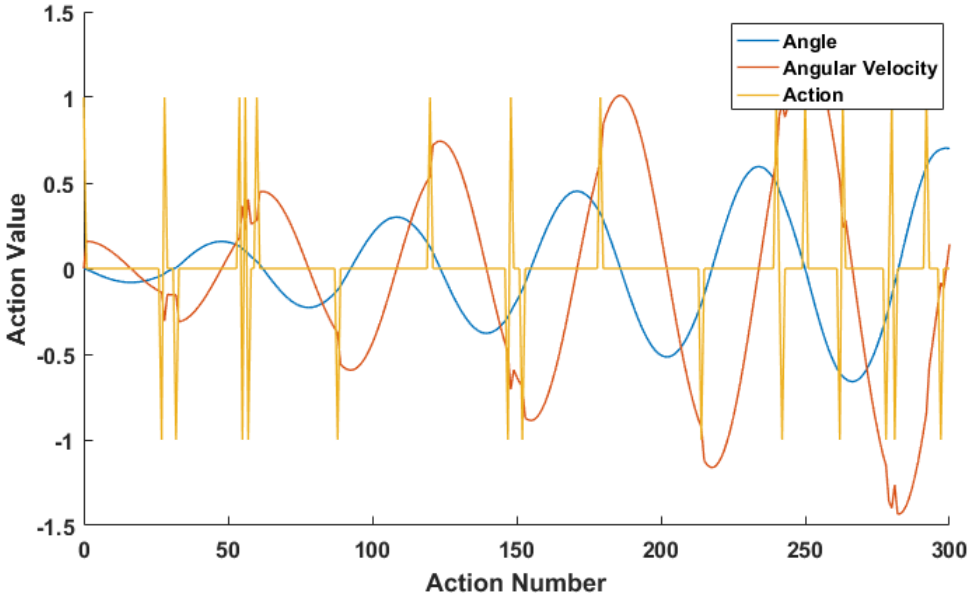


Figure 9.13: Action decisions made by the DQN agent in the non-repeating dumbbell environment.

The graphs in Figures 9.13 and 9.14 show that the agents were applying the maximum torque at the minimum of the swing. This is in contradiction to the results of the Numerical Modelling group, who found that the optimal pumping motion was at the maximum. A comparison of these two types of motion is shown in Figure 9.15. This graph shows that pumping at the bottom of the swing motion allows the robot to reach its maximum amplitude quicker. Pumping at the top of the swing motion, however, allows the robot to reach a higher maximum amplitude.

In the dumbbell environments the DQN and DDPG agents were tested on, the agent had a goal swinging amplitude, rather than the goal of swinging as high as possible. This means the agent's actions were optimised to reach the goal amplitude as quickly as possible and stay there for the rest of the epoch. The optimal motion to achieve this was to pump at the minimum of the swing motion.

The reason for the difference in the performance of the two driving strategies can be explained using the fact that driving at the swing minimum adds more energy per cycle, due to the torque being applied when the speed is at its greatest, thus building up a swinging motion quicker. However, as the speed is at a maximum, it becomes increasingly likely that the driving force will occur late; which can introduce error into the driving motion. This error could account for the maximum amplitude achieved being lower than that achieved from driving at the maximum of the swing, where it is easier to drive at the correct time.

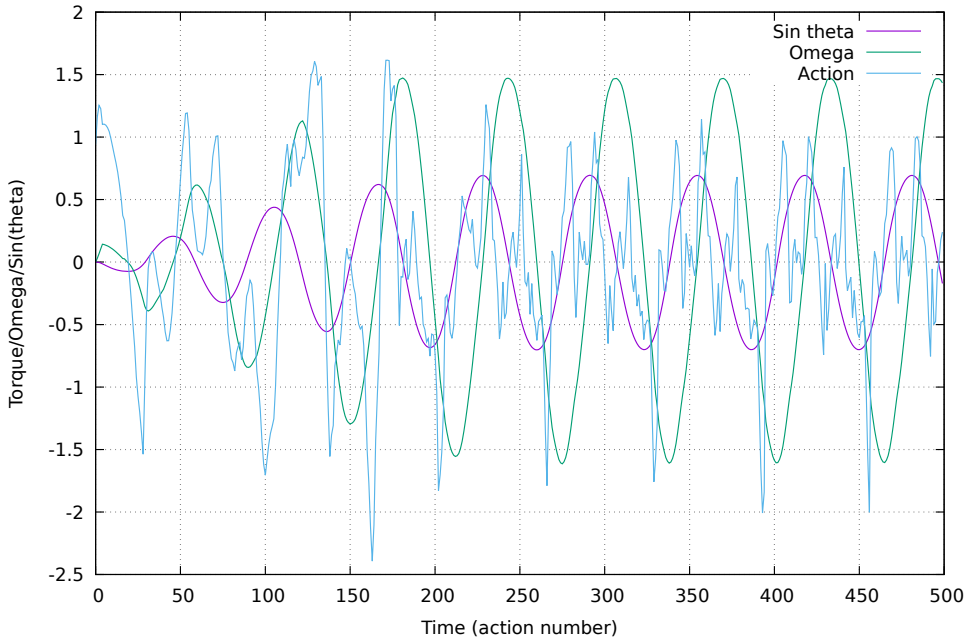


Figure 9.14: Action decisions made by the DDPG agent in the non-repeating dumbbell environment.

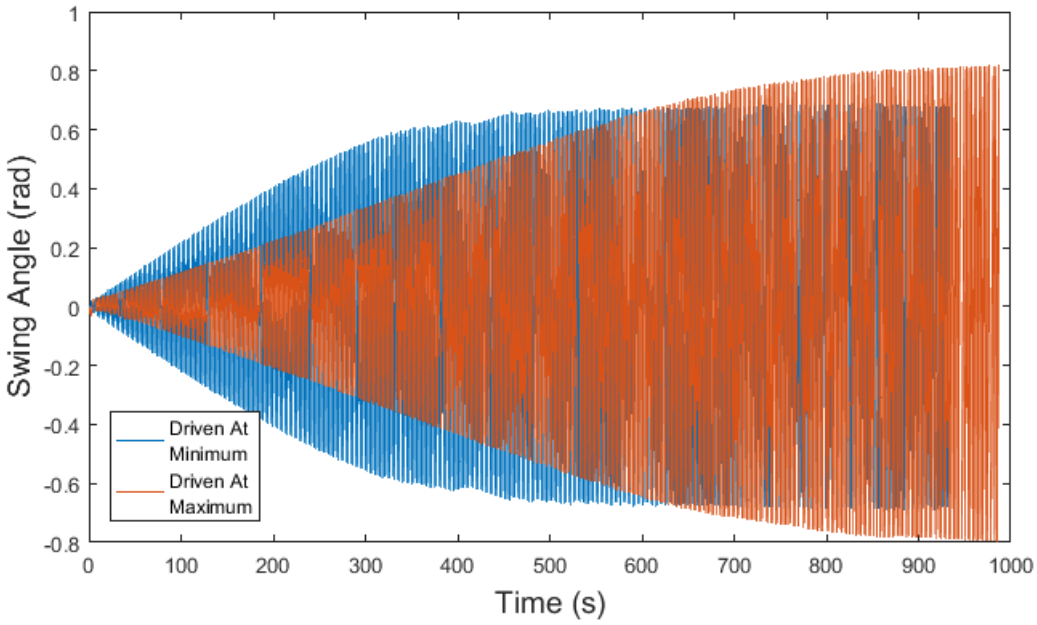


Figure 9.15: Comparison of pumping at the maximum and minimum of motion in the Webots environment.

10 Conclusions

Harry Shaw

Many successful simulations that could be used to model the behaviour of the robot using Runge-Kutta methods were developed. A flexible GUI was developed that could display the output from all models. A damped simple pendulum was used to find a realistic damping force on the swing for a simulation of the robot. For an angular velocity of $\omega\,\mathrm{rad.s}^{-1}$, the drag torque was $-0.00886M\omega\,\mathrm{Nm}$. The effect of changing the length of a pendulum with a dumbbell attached was investigated. It was found that changing the angular momentum of the pendulum with the same period of the swing resulted in the largest amplitude swinging. It was also found that changing the length of the pendulum with double the frequency of the swing resulted in the largest amplitude swinging. A flail model and then a double flair model was used as closer approximation to the robot. It was found that the optimal movement was for the robot to begin rotating just before the swing reached a peak, and finish rotating as soon as possible afterwards. A limited investigation into a model of the robot on a swing with a hinge was also carried out. This could be investigated further. In addition, the effect of specifying the speeds of the movements of individual joints could be evaluated, as this would be an even more accurate model of the robot.

Various investigations into the NAO robot were carried out. Strength and range of motion tests were carried out on the joints of the robot. The swing with the robot was found to have a natural time period of (2.561 ± 0.001)s and a damping decay constant of $(4.43 \pm 0.02) \times 10^{-3}\mathrm{s}^{-1}$. The robot was made independent of external sensors by using its stereoscopic vision. The possibility of having the robot self-start was investigated, either by simply rotating itself or by kicking off another object. This could be implemented in the future.

A virtual robot was build in Webots for testing swinging motions investigated using numerical models, and machine learning systems. A periodic motion was tested in Webots, and on the NAO robot using the swing angle encoder. The maximum amplitude generated was found to be 4.5° in Webots and the amplitude oscillated due to the anharmonic nature of the swing. A motion that depended on the detected position of the swing was also investigated, which resulted in a stable larger amplitude oscillation of 47° in webots and $15.8 \pm 0.1^\circ$ on the NAO. This was repeated using the NAO's stereoscopic vision but resulted in smaller less stable oscillations. The use of the NAO's inertial sensors with torso motion were investigated, but was not successfully implemented. This could be attempted in the future.

The optimal human swinging motion was analysed but was found to be not applicable to the NAO robot. This was because a far greater proportion of the NAO robot's mass was in its legs than is in a human.

Three classic machine learning algorithms were tested on three environments, consisting of simulations of a grid world, an inverted pendulum and a dumbbell pendulum, with mixed results. They were all found to be unsuitable for use with systems as complex as Webots or the physical NAO robot. Two neural network based systems were also tested: DDPG could choose an action from a continuous range while DQN specified discrete actions. Both learned the inverted pendulum and the dumbbell pendulum systems, with both learning the dumbbell pendulum much faster than any of the manual approaches. It was found that Webots was not conducive to machine learning due to problems issuing commands to the system on each time step, which was the main obstacle in applying the two neural network based methods. It is therefore recommended that the more complex numerical models, such as the double flail model, should be adapted for use with machine learning. This would allow further investigation into the effectiveness of DDPG and DQN.

Appendices

A Equation of Motion Derivations

Daniel Corless

A.1 Double Flail

Let x_i and y_i be the horizontal and vertical positions of mass m_i relative to the main pivot point of the swing. The kinetic energy of the system is therefore

$$T = \sum_{i=1}^{5} \frac{m_i}{2}(\dot{x}_i^2 + \dot{y}_i^2). \tag{A.1}$$

Using simple geometry, these positions can be rewritten in terms of generalised coordinates θ_1, θ_2, θ_3, θ_4 and θ_5.

$$\begin{aligned}
x_1 &= l_1 \sin(\theta_1) & y_1 &= l_1 \cos(\theta_1) \\
x_2 &= r_1 \sin(\theta_1) + l_2 \sin(\theta_2) & y_2 &= r_1 \cos(\theta_1) + l_2 \cos(\theta_2) \\
x_3 &= r_1 \sin(\theta_1) + l_3 \sin(\theta_3) & y_3 &= r_1 \cos(\theta_1) + l_3 \cos(\theta_3) \\
x_4 &= r_1 \sin(\theta_1) + r_2 \sin(\theta_2) + l_4 \sin(\theta_4) & y_4 &= r_1 \cos(\theta_1) + r_2 \cos(\theta_2) + l_4 \cos(\theta_4) \\
x_5 &= r_1 \sin(\theta_1) + r_3 \sin(\theta_3) + l_5 \sin(\theta_5) & y_5 &= r_1 \cos(\theta_1) + r_3 \cos(\theta_3) + l_5 \cos(\theta_5)
\end{aligned} \tag{A.2}$$

The time derivatives of these positions are therefore

$$\begin{aligned}
\dot{x}_1 &= l_1\dot{\theta}_1 \cos(\theta_1), & \dot{y}_1 &= -l_1\dot{\theta}_1 \sin(\theta_1), \\
\dot{x}_2 &= r_1\dot{\theta}_1 \cos(\theta_1) + l_2\dot{\theta}_2 \cos(\theta_2), & \dot{y}_2 &= -r_1\dot{\theta}_1 \sin(\theta_1) - l_2\dot{\theta}_2 \sin(\theta_2), \\
\dot{x}_3 &= r_1\dot{\theta}_1 \cos(\theta_1) + l_3\dot{\theta}_3 \cos(\theta_3), & \dot{y}_3 &= -r_1\dot{\theta}_1 \sin(\theta_1) - l_3\dot{\theta}_3 \sin(\theta_3), \\
\dot{x}_4 &= r_1\dot{\theta}_1 \cos(\theta_1) + r_2\dot{\theta}_2 \cos(\theta_2) + l_4\dot{\theta}_4 \cos(\theta_4), & \dot{y}_4 &= -r_1\dot{\theta}_1 \sin(\theta_1) - r_2\dot{\theta}_2 \sin(\theta_2) - l_4\dot{\theta}_4 \sin(\theta_4),
\end{aligned} \tag{A.3}$$

$$\dot{x}_5 = r_1\dot{\theta}_1\cos(\theta_1) + r_3\dot{\theta}_3\cos(\theta_3) + l_5\dot{\theta}_5\cos(\theta_5), \qquad \dot{y}_5 = -r_1\dot{\theta}_1\sin(\theta_1) - r_3\dot{\theta}_3\sin(\theta_3) - l_5\dot{\theta}_5\sin(\theta_5).$$

Squaring these coordinates and applying the trigonometric identities $\sin^2(A)+\cos^2(A) = 1$ and $\cos(A)\cos(B)+\sin(A)\sin(B) = \cos(A-B)$ results in the following expression for the kinetic energy of the system.

$$\begin{aligned} T = \frac{m_1}{2}l_1^2\dot{\theta}_1^2 + \frac{(m_2+m_3+m_4+m_5)}{2}r_1^2\dot{\theta}_1^2 + \frac{m_2}{2}\left[l_2^2\dot{\theta}_2^2 + 2r_1l_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\right] \\ +\frac{m_3}{2}\left[l_3^2\dot{\theta}_3^2 + 2r_1l_3\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)\right] + \frac{m_4}{2}\left[r_2^2\dot{\theta}_2^2 + l_4^2\dot{\theta}_4^2 + 2r_1r_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\right. \\ \left.+2r_1l_4\dot{\theta}_1\dot{\theta}_4\cos(\theta_1-\theta_4) + 2r_2l_4\dot{\theta}_2\dot{\theta}_4\cos(\theta_2-\theta_4)\right] + \frac{m_5}{2}\left[r_3^2\dot{\theta}_3^2 + l_5^2\dot{\theta}_5^2\right. \\ \left.+2r_1r_3\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3) + 2r_1l_5\dot{\theta}_1\dot{\theta}_5\cos(\theta_1-\theta_5) + 2r_3l_5\dot{\theta}_3\dot{\theta}_5\cos(\theta_3-\theta_5)\right] \end{aligned} \tag{A.4}$$

The zero point of potential will now be defined as being at $y = 0$. Therefore, the potential of the system is given by

$$\begin{aligned} V = -\sum_{i=1}^{5} m_i g y_i = -m_1gl_1\cos(\theta_1) - (m_2+m_3+m_4+m_5)gr_1\cos(\theta_1) - (m_2l_2 \\ +m_4r_2)g\cos(\theta_2) - (m_3l_3+m_5r_3)g\cos(\theta_3) - m_4gl_4\cos(\theta_4) - m_5gl_5\cos(\theta_5). \end{aligned} \tag{A.5}$$

This results in the Lagrangian

$$\begin{aligned} \mathcal{L} = \frac{m_1}{2}l_1^2\dot{\theta}_1^2 + \frac{(m_2+m_3+m_4+m_5)}{2}r_1^2\dot{\theta}_1^2 + \frac{m_2}{2}\left[l_2^2\dot{\theta}_2^2 + 2r_1l_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\right] \\ +\frac{m_3}{2}\left[l_3^2\dot{\theta}_3^2 + 2r_1l_3\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)\right] + \frac{m_4}{2}\left[r_2^2\dot{\theta}_2^2 + l_4^2\dot{\theta}_4^2 + 2r_1r_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\right. \\ \left.+2r_1l_4\dot{\theta}_1\dot{\theta}_4\cos(\theta_1-\theta_4) + 2r_2l_4\dot{\theta}_2\dot{\theta}_4\cos(\theta_2-\theta_4)\right] + \frac{m_5}{2}\left[r_3^2\dot{\theta}_3^2 + l_5^2\dot{\theta}_5^2\right. \\ \left.+2r_1r_3\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3) + 2r_1l_5\dot{\theta}_1\dot{\theta}_5\cos(\theta_1-\theta_5) + 2r_3l_5\dot{\theta}_3\dot{\theta}_5\cos(\theta_3-\theta_5)\right] \\ +m_1gl_1\cos(\theta_1) + (m_2+m_3+m_4+m_5)gr_1\cos(\theta_1) + (m_2l_2+m_4r_2)g\cos(\theta_2) \\ +(m_3l_3+m_5r_3)g\cos(\theta_3) + m_4gl_4\cos(\theta_4) + m_5gl_5\cos(\theta_5). \end{aligned} \tag{A.6}$$

Applying the Euler-Lagrange equation for θ_1 gives

$$\begin{aligned} \frac{\partial\mathcal{L}}{\partial\dot{\theta}_1} = \left[m_1l_1^2 + (m_2+m_3+m_4+m_5)r_1^2\right]\dot{\theta}_1 + m_2r_1l_2\dot{\theta}_2\cos(\theta_1-\theta_2) \\ +m_3r_1l_3\dot{\theta}_3\cos(\theta_1-\theta_3) + m_4\left[r_1r_2\dot{\theta}_2\cos(\theta_1-\theta_2) + r_1l_4\dot{\theta}_4\cos(\theta_1-\theta_4)\right] \\ +m_5\left[r_1r_3\dot{\theta}_3\cos(\theta_1-\theta_3) + r_1l_5\dot{\theta}_5\cos(\theta_1-\theta_5)\right], \end{aligned} \tag{A.7}$$

$$\begin{aligned}\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1} = &\big[m_1 l_1^2 + (m_2+m_3+m_4+m_5)r_1^2\big]\ddot{\theta}_1 + m_2 r_1 l_2\big[\ddot{\theta}_2\cos(\theta_1-\theta_2)\\ &-\dot{\theta}_2(\dot{\theta}_1-\dot{\theta}_2)\sin(\theta_1-\theta_2)\big] + m_3 r_1 l_3\big[\ddot{\theta}_3\cos(\theta_1-\theta_3) - \dot{\theta}_3(\dot{\theta}_1-\dot{\theta}_3)\sin(\theta_1-\theta_3)\big]\\ &+m_4 r_1\big[r_2\ddot{\theta}_2\cos(\theta_1-\theta_2) - r_2\dot{\theta}_2(\dot{\theta}_1-\dot{\theta}_2)\sin(\theta_1-\theta_2) + l_4\ddot{\theta}_4\cos(\theta_1-\theta_4)\\ &-l_4\dot{\theta}_4(\dot{\theta}_1-\dot{\theta}_4)\sin(\theta_1-\theta_4)\big] + m_5 r_1\big[r_3\ddot{\theta}_3\cos(\theta_1-\theta_3)\\ &-r_3\dot{\theta}_3(\dot{\theta}_1-\dot{\theta}_3)\sin(\theta_1-\theta_3) + l_5\ddot{\theta}_5\cos(\theta_1-\theta_5) - l_5\dot{\theta}_5(\dot{\theta}_1-\dot{\theta}_5)\sin(\theta_1-\theta_5)\big],\end{aligned} \tag{A.8}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta_1} = &-m_2 r_1 l_2\dot{\theta}_1\dot{\theta}_2\sin(\theta_1-\theta_2) - m_3 r_1 l_3\dot{\theta}_1\dot{\theta}_3\sin(\theta_1-\theta_3)\\ &-m_4 r_1\big[r_2\dot{\theta}_1\dot{\theta}_2\sin(\theta_1-\theta_2) + l_4\dot{\theta}_1\dot{\theta}_4\sin(\theta_1-\theta_4)\big] - m_5 r_1\big[r_3\dot{\theta}_1\dot{\theta}_3\sin(\theta_1-\theta_3)\\ &+l_5\dot{\theta}_1\dot{\theta}_5\sin(\theta_1-\theta_5)\big] - \big[m_1 g l_1 + (m_2+m_3+m_4+m_5)g r_1\big]\sin(\theta_1),\end{aligned} \tag{A.9}$$

$$\begin{aligned}\therefore &\big[m_1 l_1^2 + (m_2+m_3+m_4+m_5)r_1^2\big]\ddot{\theta}_1 + m_2 r_1 l_2\big[\ddot{\theta}_2\cos(\theta_1-\theta_2)\\ &-\dot{\theta}_2(\dot{\theta}_1-\dot{\theta}_2)\sin(\theta_1-\theta_2)\big] + m_3 r_1 l_3\big[\ddot{\theta}_3\cos(\theta_1-\theta_3) - \dot{\theta}_3(\dot{\theta}_1-\dot{\theta}_3)\sin(\theta_1-\theta_3)\big]\\ &+m_4 r_1\big[r_2\ddot{\theta}_2\cos(\theta_1-\theta_2) - r_2\dot{\theta}_2(\dot{\theta}_1-\dot{\theta}_2)\sin(\theta_1-\theta_2) + l_4\ddot{\theta}_4\cos(\theta_1-\theta_4)\\ &-l_4\dot{\theta}_4(\dot{\theta}_1-\dot{\theta}_4)\sin(\theta_1-\theta_4)\big] + m_5 r_1\big[r_3\ddot{\theta}_3\cos(\theta_1-\theta_3) - r_3\dot{\theta}_3(\dot{\theta}_1-\dot{\theta}_3)\sin(\theta_1-\theta_3)\\ &+l_5\ddot{\theta}_5\cos(\theta_1-\theta_5) - l_5\dot{\theta}_5(\dot{\theta}_1-\dot{\theta}_5)\sin(\theta_1-\theta_5)\big] = -m_2 r_1 l_2\dot{\theta}_1\dot{\theta}_2\sin(\theta_1-\theta_2)\\ &-m_3 r_1 l_3\dot{\theta}_1\dot{\theta}_3\sin(\theta_1-\theta_3) - m_4 r_1\big[r_2\dot{\theta}_1\dot{\theta}_2\sin(\theta_1-\theta_2) + l_4\dot{\theta}_1\dot{\theta}_4\sin(\theta_1-\theta_4)\big]\\ &-m_5 r_1\big[r_3\dot{\theta}_1\dot{\theta}_3\sin(\theta_1-\theta_3) + l_5\dot{\theta}_1\dot{\theta}_5\sin(\theta_1-\theta_5)\big]\\ &-\big[m_1 g l_1 + (m_2+m_3+m_4+m_5)g r_1\big]\sin(\theta_1).\end{aligned} \tag{A.10}$$

This can be simplified and rearranged to give Equation 2.35, the equation of motion of the system for θ_1.

A.2 Double-Hinged Double Flail

Let x_i and y_i be the horizontal and vertical positions of mass m_i relative to the main pivot point of the swing. The kinetic energy of the system is therefore

$$T = \sum_{i=1}^{5}\frac{m_i}{2}\big(\dot{x}_i^2 + \dot{y}_i^2\big). \tag{A.11}$$

Using simple geometry, these positions can be rewritten in terms of generalised coordinates α_1, α_2, θ_1, θ_2, θ_3, θ_4 and θ_5.

$$\begin{aligned}x_1 &= d_1\sin(\alpha_1) + d_2\sin(\alpha_2) + l_1\sin(\theta_1)\\ y_1 &= d_1\cos(\alpha_1) + d_2\cos(\alpha_2) + l_1\cos(\theta_1)\end{aligned}$$

$$\begin{aligned}x_2 &= d_1\sin(\alpha_1) + d_2\sin(\alpha_2) + l_1\sin(\theta_1) + l_2\sin(\theta_2)\\ y_2 &= d_1\cos(\alpha_1) + d_2\cos(\alpha_2) + l_1\cos(\theta_1) + l_2\cos(\theta_2)\end{aligned}$$

$$\begin{aligned} x_3 &= d_1 \sin(\alpha_1) + d_2 \sin(\alpha_2) + l_1 \sin(\theta_1) + l_3 \sin(\theta_3) \\ y_3 &= d_1 \cos(\alpha_1) + d_2 \cos(\alpha_2) + l_1 \cos(\theta_1) + l_3 \cos(\theta_3) \end{aligned} \tag{A.12}$$

$$\begin{aligned} x_4 &= d_1 \sin(\alpha_1) + d_2 \sin(\alpha_2) + l_1 \sin(\theta_1) + r_2 \sin(\theta_2) + l_4 \sin(\theta_4) \\ y_4 &= d_1 \cos(\alpha_1) + d_2 \cos(\alpha_2) + l_1 \cos(\theta_1) + r_2 \cos(\theta_2) + l_4 \cos(\theta_4) \end{aligned}$$

$$\begin{aligned} x_5 &= d_1 \sin(\alpha_1) + d_2 \sin(\alpha_2) + l_1 \sin(\theta_1) + r_3 \sin(\theta_3) + l_5 \sin(\theta_5) \\ y_5 &= d_1 \cos(\alpha_1) + d_2 \cos(\alpha_2) + l_1 \cos(\theta_1) + r_3 \cos(\theta_3) + l_5 \cos(\theta_5) \end{aligned}$$

$$\begin{aligned} x_6 &= d_1 \sin(\alpha_1) \\ y_6 &= d_1 \cos(\alpha_1) \end{aligned}$$

$$\begin{aligned} x_7 &= d_1 \sin(\alpha_1) + d_2 \sin(\alpha_2) \\ y_7 &= d_1 \cos(\alpha_1) + d_2 \cos(\alpha_2) \end{aligned}$$

The time derivatives of these positions are therefore

$$\begin{aligned} \dot{x}_1 &= d_1\dot{\alpha}_1 \cos(\alpha_1) + d_2\dot{\alpha}_2 \cos(\alpha_2) + l_1\dot{\theta}_1 \cos(\theta_1), \\ \dot{y}_1 &= -d_1\dot{\alpha}_1 \sin(\alpha_1) - d_2\dot{\alpha}_2 \sin(\alpha_2) - l_1\dot{\theta}_1 \sin(\theta_1), \end{aligned}$$

$$\begin{aligned} \dot{x}_2 &= d_1\dot{\alpha}_1 \cos(\alpha_1) + d_2\dot{\alpha}_2 \cos(\alpha_2) + l_1\dot{\theta}_1 \cos(\theta_1) + l_2\dot{\theta}_2 \cos(\theta_2), \\ \dot{y}_2 &= -d_1\dot{\alpha}_1 \sin(\alpha_1) - d_2\dot{\alpha}_2 \sin(\alpha_2) - l_1\dot{\theta}_1 \sin(\theta_1) - l_2\dot{\theta}_2 \sin(\theta_2), \end{aligned}$$

$$\begin{aligned} \dot{x}_3 &= d_1\dot{\alpha}_1 \cos(\alpha_1) + d_2\dot{\alpha}_2 \cos(\alpha_2) + l_1\dot{\theta}_1 \cos(\theta_1) + l_3\dot{\theta}_3 \cos(\theta_3), \\ \dot{y}_3 &= -d_1\dot{\alpha}_1 \sin(\alpha_1) - d_2\dot{\alpha}_2 \sin(\alpha_2) - l_1\dot{\theta}_1 \sin(\theta_1) - l_3\dot{\theta}_3 \sin(\theta_3), \end{aligned} \tag{A.13}$$

$$\begin{aligned} \dot{x}_4 &= d_1\dot{\alpha}_1 \cos(\alpha_1) + d_2\dot{\alpha}_2 \cos(\alpha_2) + l_1\dot{\theta}_1 \cos(\theta_1) + r_2\dot{\theta}_2 \cos(\theta_2) + l_4\dot{\theta}_4 \cos(\theta_4), \\ \dot{y}_4 &= -d_1\dot{\alpha}_1 \sin(\alpha_1) - d_2\dot{\alpha}_2 \sin(\alpha_2) - l_1\dot{\theta}_1 \sin(\theta_1) - r_2\dot{\theta}_2 \sin(\theta_2) - l_4\dot{\theta}_4 \sin(\theta_4), \end{aligned}$$

$$\begin{aligned} \dot{x}_5 &= d_1\dot{\alpha}_1 \cos(\alpha_1) + d_2\dot{\alpha}_2 \cos(\alpha_2) + l_1\dot{\theta}_1 \cos(\theta_1) + r_3\dot{\theta}_3 \cos(\theta_3) + l_5\dot{\theta}_5 \cos(\theta_5), \\ \dot{y}_5 &= -d_1\dot{\alpha}_1 \sin(\alpha_1) - d_2\dot{\alpha}_2 \sin(\alpha_2) - l_1\dot{\theta}_1 \sin(\theta_1) - r_3\dot{\theta}_3 \sin(\theta_3) - l_5\dot{\theta}_5 \sin(\theta_5), \end{aligned}$$

$$\begin{aligned} \dot{x}_6 &= d_1\dot{\alpha}_1 \cos(\alpha_1), \\ \dot{y}_6 &= -d_1\dot{\alpha}_1 \sin(\alpha_1), \end{aligned}$$

$$\dot{x}_7 = d_1\dot{\alpha}_1\cos(\alpha_1) + d_2\dot{\alpha}_2\cos(\alpha_2),$$
$$\dot{y}_7 = -d_1\dot{\alpha}_1\sin(\alpha_1) - d_2\dot{\alpha}_2\sin(\alpha_2).$$

Squaring these coordinates and applying the trigonometric identities $\sin^2(A)+\cos^2(A) = 1$ and $\cos(A)\cos(B)+\sin(A)\sin(B) = \cos(A-B)$ results in the following expression for the kinetic energy of the system.

$$\begin{aligned} T = \frac{M}{2}\big[d_1^2\dot{\alpha}_1^2 + d_2^2\dot{\alpha}_2^2 + l_1^2\dot{\theta}_1^2 + 2d_1d_2\dot{\alpha}_1\dot{\alpha}_2\cos(\alpha_1-\alpha_2) + 2d_1l_1\dot{\alpha}_1\dot{\theta}_1\cos(\alpha_1-\theta_1) \\ +2d_2l_1\dot{\alpha}_2\dot{\theta}_1\cos(\alpha_2-\theta_1)\big] + \frac{m_2}{2}l_2^2\dot{\theta}_2^2 + \frac{m_3}{2}l_3^2\dot{\theta}_3^2 + \frac{m_4}{2}\big(r_2^2\dot{\theta}_2^2 + l_4^2\dot{\theta}_4^2\big) \\ +\frac{m_5}{2}\big(r_3^2\dot{\theta}_3^2 + l_5^2\dot{\theta}_5^2\big) + (m_2l_2 + m_4r_2)\big[d_1\dot{\alpha}_1\dot{\theta}_2\cos(\alpha_1-\theta_2) + d_2\dot{\alpha}_2\dot{\theta}_2\cos(\alpha_2-\theta_2) \\ +l_1\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\big] + (m_3l_3 + m_5r_3)\big[d_1\dot{\alpha}_1\dot{\theta}_3\cos(\alpha_1-\theta_3) + d_2\dot{\alpha}_2\dot{\theta}_3\cos(\alpha_2-\theta_3) \\ +l_1\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)\big] + m_4l_4\big[d_1\dot{\alpha}_1\dot{\theta}_4\cos(\alpha_1-\theta_4) + d_2\dot{\alpha}_2\dot{\theta}_4\cos(\alpha_2-\theta_4) \\ +l_1\dot{\theta}_1\dot{\theta}_4\cos(\theta_1-\theta_4) + r_2\dot{\theta}_2\dot{\theta}_4\cos(\theta_2-\theta_4)\big] + m_5l_5\big[d_1\dot{\alpha}_1\dot{\theta}_5\cos(\alpha_1-\theta_5) \\ +d_2\dot{\alpha}_2\dot{\theta}_5\cos(\alpha_2-\theta_5) + l_1\dot{\theta}_1\dot{\theta}_5\cos(\theta_1-\theta_5) + r_3\dot{\theta}_3\dot{\theta}_5\cos(\theta_3-\theta_5)\big] \\ +\frac{m_7}{2}\big[d_1^2\dot{\alpha}_1^2 + d_2^2\dot{\alpha}_2^2 + 2d_1d_2\dot{\alpha}_1\dot{\alpha}_2\cos(\alpha_1-\alpha_2)\big] \end{aligned} \tag{A.14}$$

The zero point of potential will now be defined as being at $y = 0$. Therefore, the potential of the system is given by

$$\begin{aligned} V = \sum_{i=1}^{5} m_i g y_i = -Mg\big[d_1\cos(\alpha_1) + d_2\cos(\alpha_2) + l_1\cos(\theta_1)\big] - (m_2l_2 + m_4r_2)g\cos(\theta_2) - (m_3l_3 \\ +m_5r_3)g\cos(\theta_3) - m_4l_4g\cos(\theta_4) - m_5l_5g\cos(\theta_5) - m_6gd_1\cos(\alpha_1) - m_7g\big[d_1\cos(\alpha_1) + d_2\cos(\alpha_2)\big]. \end{aligned} \tag{A.15}$$

This gives a Lagrangian of

$$\begin{aligned} \mathcal{L} = \frac{M}{2}\big[d_1^2\dot{\alpha}_1^2 + d_2^2\dot{\alpha}_2^2 + l_1^2\dot{\theta}_1^2 + 2d_1d_2\dot{\alpha}_1\dot{\alpha}_2\cos(\alpha_1-\alpha_2) + 2d_1l_1\dot{\alpha}_1\dot{\theta}_1\cos(\alpha_1-\theta_1) \\ +2d_2l_1\dot{\alpha}_2\dot{\theta}_1\cos(\alpha_2-\theta_1)\big] + \frac{m_2}{2}l_2^2\dot{\theta}_2^2 + \frac{m_3}{2}l_3^2\dot{\theta}_3^2 + \frac{m_4}{2}\big(r_2^2\dot{\theta}_2^2 + l_4^2\dot{\theta}_4^2\big) \\ +\frac{m_5}{2}\big(r_3^2\dot{\theta}_3^2 + l_5^2\dot{\theta}_5^2\big) + (m_2l_2 + m_4r_2)\big[d_1\dot{\alpha}_1\dot{\theta}_2\cos(\alpha_1-\theta_2) + d_2\dot{\alpha}_2\dot{\theta}_2\cos(\alpha_2-\theta_2) \\ +l_1\dot{\theta}_1\dot{\theta}_2\cos(\theta_1-\theta_2)\big] + (m_3l_3 + m_5r_3)\big[d_1\dot{\alpha}_1\dot{\theta}_3\cos(\alpha_1-\theta_3) + d_2\dot{\alpha}_2\dot{\theta}_3\cos(\alpha_2-\theta_3) \\ +l_1\dot{\theta}_1\dot{\theta}_3\cos(\theta_1-\theta_3)\big] + m_4l_4\big[d_1\dot{\alpha}_1\dot{\theta}_4\cos(\alpha_1-\theta_4) + d_2\dot{\alpha}_2\dot{\theta}_4\cos(\alpha_2-\theta_4) \\ +l_1\dot{\theta}_1\dot{\theta}_4\cos(\theta_1-\theta_4) + r_2\dot{\theta}_2\dot{\theta}_4\cos(\theta_2-\theta_4)\big] + m_5l_5\big[d_1\dot{\alpha}_1\dot{\theta}_5\cos(\alpha_1-\theta_5) \\ +d_2\dot{\alpha}_2\dot{\theta}_5\cos(\alpha_2-\theta_5) + l_1\dot{\theta}_1\dot{\theta}_5\cos(\theta_1-\theta_5) + r_3\dot{\theta}_3\dot{\theta}_5\cos(\theta_3-\theta_5)\big] + Mg\big[d_1\cos(\alpha_1) \\ +d_2\cos(\alpha_2) + l_1\cos(\theta_1)\big] + (m_2l_2 + m_4r_2)g\cos(\theta_2) + (m_3l_3 + m_5r_3)g\cos(\theta_3) \\ +m_4l_4g\cos(\theta_4) + m_5l_5g\cos(\theta_5) + \frac{m_6}{2}d_1^2\dot{\alpha}_1^2 + \frac{m_7}{2}\big[d_1^2\dot{\alpha}_1^2 + d_2^2\dot{\alpha}_2^2 \\ +2d_1d_2\dot{\alpha}_1\dot{\alpha}_2\cos(\alpha_1-\alpha_2)\big] + m_6gd_1\cos(\alpha_1) + m_7g\big[d_1\cos(\alpha_1) + d_2\cos(\alpha_2)\big]. \end{aligned} \tag{A.16}$$

The Euler-Lagrange equation can now be applied to α_1, giving

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1} &= M\big[d_1^2\dot{\alpha}_1 + d_1 d_2\dot{\alpha}_2\cos(\alpha_1-\alpha_2) + d_1 l_1\dot{\theta}_1\cos(\alpha_1-\theta_1)\big] \\ &+(m_2 l_2 + m_4 r_2)d_1\dot{\theta}_2\cos(\alpha_1-\theta_2) + (m_3 l_3 + m_5 r_3)d_1\dot{\theta}_3\cos(\alpha_1-\theta_3) \\ &+m_4 l_4 d_1\dot{\theta}_4\cos(\alpha_1-\theta_4) + m_5 l_5 d_1\dot{\theta}_5\cos(\alpha_1-\theta_5) \\ &+(m_6+m_7)d_1^2\dot{\alpha}_1 + m_7 d_1 d_2\dot{\alpha}_2\cos(\alpha_1-\alpha_2),\end{aligned} \tag{A.17}$$

$$\begin{aligned}\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1} &= Md_1\big[d_1\ddot{\alpha}_1 + d_2\ddot{\alpha}_2\cos(\alpha_1-\alpha_2) - d_2\dot{\alpha}_2(\dot{\alpha}_1-\dot{\alpha}_2)\sin(\alpha_1-\alpha_2) \\ &+l_1\ddot{\theta}_1\cos(\alpha_1-\theta_1) - l_1\dot{\theta}_1(\dot{\alpha}_1-\dot{\theta}_1)\sin(\alpha_1-\theta_1)\big] + (m_2 l_2 + m_4 r_2)\big[d_1\ddot{\theta}_2\cos(\alpha_1-\theta_2) \\ &-d_1\dot{\theta}_2(\dot{\alpha}_1-\dot{\theta}_2)\sin(\alpha_1-\theta_2)\big] + (m_3 l_3 + m_5 r_3)\big[d_1\ddot{\theta}_3\cos(\alpha_1-\theta_3) \\ &-d_1\dot{\theta}_3(\dot{\alpha}_1-\dot{\theta}_3)\sin(\alpha_1-\theta_3)\big] + m_4 l_4 d_1\big[\ddot{\theta}_4\cos(\alpha_1-\theta_4) - \dot{\theta}_4(\dot{\alpha}_1-\dot{\theta}_4)\sin(\alpha_1-\theta_4)\big] \\ &+m_5 l_5 d_1\big[\ddot{\theta}_5\cos(\alpha_1-\theta_5) - \dot{\theta}_5(\dot{\alpha}_1-\dot{\theta}_5)\sin(\alpha_1-\theta_5)\big] + (m_6+m_7)d_1^2\ddot{\alpha}_1 \\ &+m_7 d_1 d_2\big[\ddot{\alpha}_2\cos(\alpha_1-\alpha_2) - \dot{\alpha}_2(\dot{\alpha}_1-\dot{\alpha}_2)\sin(\alpha_1-\alpha_2)\big],\end{aligned} \tag{A.18}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \theta_1} &= -Md_1\big[d_2\dot{\alpha}_1\dot{\alpha}_2\sin(\alpha_1-\alpha_2) + l_1\dot{\alpha}_1\dot{\theta}_1\sin(\alpha_1-\theta_1)\big] - (m_2 l_2 \\ &+m_4 r_2)d_1\dot{\alpha}_1\dot{\theta}_2\sin(\alpha_1-\theta_2) - (m_3 l_3 + m_5 r_3)d_1\dot{\alpha}_1\dot{\theta}_3\sin(\alpha_1-\theta_3) \\ &-m_4 l_4 d_1\dot{\alpha}_1\dot{\theta}_4\sin(\alpha_1-\theta_4) - m_5 l_5 d_1\dot{\alpha}_1\dot{\theta}_5\sin(\alpha_1-\theta_5) - Mgd_1\sin(\alpha_1) \\ &-m_7 d_1 d_2\dot{\alpha}_1\dot{\alpha}_2\sin(\alpha_1-\alpha_2) - (m_6+m_7)gd_1\sin(\alpha_1),\end{aligned} \tag{A.19}$$

$$\begin{aligned}&\therefore Md_1\big[d_1\ddot{\alpha}_1 + d_2\ddot{\alpha}_2\cos(\alpha_1-\alpha_2) - d_2\dot{\alpha}_2(\dot{\alpha}_1-\dot{\alpha}_2)\sin(\alpha_1-\alpha_2) + l_1\ddot{\theta}_1\cos(\alpha_1-\theta_1) \\ &-l_1\dot{\theta}_1(\dot{\alpha}_1-\dot{\theta}_1)\sin(\alpha_1-\theta_1)\big] + (m_2 l_2 + m_4 r_2)\big[d_1\ddot{\theta}_2\cos(\alpha_1-\theta_2) \\ &-d_1\dot{\theta}_2(\dot{\alpha}_1-\dot{\theta}_2)\sin(\alpha_1-\theta_2)\big] + (m_3 l_3 + m_5 r_3)\big[d_1\ddot{\theta}_3\cos(\alpha_1-\theta_3) \\ &-d_1\dot{\theta}_3(\dot{\alpha}_1-\dot{\theta}_3)\sin(\alpha_1-\theta_3)\big] + m_4 l_4 d_1\big[\ddot{\theta}_4\cos(\alpha_1-\theta_4) - \dot{\theta}_4(\dot{\alpha}_1-\dot{\theta}_4)\sin(\alpha_1-\theta_4)\big] \\ &+m_5 l_5 d_1\big[\ddot{\theta}_5\cos(\alpha_1-\theta_5) - \dot{\theta}_5(\dot{\alpha}_1-\dot{\theta}_5)\sin(\alpha_1-\theta_5)\big] + (m_6+m_7)d_1^2\ddot{\alpha}_1 \\ &+m_7 d_1 d_2\big[\ddot{\alpha}_2\cos(\alpha_1-\alpha_2) - \dot{\alpha}_2(\dot{\alpha}_1-\dot{\alpha}_2)\sin(\alpha_1-\alpha_2)\big] = -Md_1\big[d_2\dot{\alpha}_1\dot{\alpha}_2\sin(\alpha_1-\alpha_2) \\ &+l_1\dot{\alpha}_1\dot{\theta}_1\sin(\alpha_1-\theta_1)\big] - (m_2 l_2 + m_4 r_2)d_1\dot{\alpha}_1\dot{\theta}_2\sin(\alpha_1-\theta_2) - (m_3 l_3 \\ &+m_5 r_3)d_1\dot{\alpha}_1\dot{\theta}_3\sin(\alpha_1-\theta_3) - m_4 l_4 d_1\dot{\alpha}_1\dot{\theta}_4\sin(\alpha_1-\theta_4) - m_5 l_5 d_1\dot{\alpha}_1\dot{\theta}_5\sin(\alpha_1-\theta_5) \\ &-Mgd_1\sin(\alpha_1) - (m_6+m_7)gd_1\sin(\alpha_1) - m_7 d_1 d_2\dot{\alpha}_1\dot{\alpha}_2\sin(\alpha_1-\alpha_2).\end{aligned} \tag{A.20}$$

This can be simplified and rearranged to get Equation 2.37, the equation of motion of the system for α_1. Using the same method, the equations of motion for α_2 and θ_1 can be found, which give equations 2.38 and 2.39 respectively.

References

1. Anorak. *1810: Friedrich Kauffman of Dresden's First Robot In History* <`http://flashbak.com/1810-friedrich-kauffman-of-dresdens-first-robot-in-history-5614`> (2014).

2. Robots, T. O. *Elmer Elsie Robots* <http://www.theoldrobots.com/ElmerElsie.html> (2010).
3. Borowiec, S. *AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol* <https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol> (2016).
4. Johnson, P. *World's First Playground Swing* <http://www.play-scapes.com/play-history/pre-1900/worlds-first-playground-swing-says-the-daily-mail-not-again> (2013).
5. kiiking.com. *About Kiiking* <http://kiiking.com/en/about-kiiking> (2014).
6. Hand, L. N. & Finch, J. D. *Analytical Mechanics* 1 (Cambridge University Press, 1998).
7. Teaching an Aldebaran NAO Robot to Swing (2016).
8. Aldebaran. *NAO H25 Masses* <http://doc.aldebaran.com/2-1/family/nao_h25/masses_h25.html> (2013).
9. Bae, S. & Kang, Y.-H. Optimal pumping in a model of a swing. *European Journal of Physics* **27** (2005).
10. *NAOqi Framework* 2015. <http://doc.aldebaran.com/1-14/dev/naoqi/index.html> (visited on 21/03/2017).
11. Aldebaran. *NAO H25 Motors* <http://doc.aldebaran.com/2-1/family/nao_h25/motors_h25.html> (2013).
12. Aldebaran. *NAO H25 Links* <http://doc.aldebaran.com/2-1/family/nao_h25/links_h25.html> (2013).
13. Aldebaran. *NAO H25 Joints* <http://doc.aldebaran.com/1-14/family/nao_h25/joints_h25.html> (2013).
14. NASA. *The Drag Equation* <https://www.grc.nasa.gov/www/k-12/airplane/drageq.html> (2017).
15. Toolbox, T. E. *U.S Standard Atmosphere* <http://www.engineeringtoolbox.com/standard-atmosphere-d_604.html> (2017).
16. Alderaban. *NAO - Construction*
17. Aldebaran. *Architecture of the pref file Device.xml / of key/values in ALMemory* <http://doc.aldebaran.com/1-14/naoqi/sensors/dcm/pref_file_architecture.html#dcm-accelerometer> (2013).
18. Aldebaran. *NAO Inertial Unit* <http://doc.aldebaran.com/1-14/family/robots/inertial_robot.html> (2013).
19. Teaching the Aldebaran NAO Robot to Use a Swing (2015).
20. Aldebaran. *NAO - Video Camera Documentation* <http://doc.aldebaran.com/2-1/family/robots/video_robot.html#robot-video> (2013).
21. Aldebaran. *ALLandMarkDetection* <http://doc.aldebaran.com/2-1/naoqi/vision/allandmarkdetection.html#allandmarkdetection> (2013).
22. Tözeren, A. *Human Body Dynamics - Classical Mechanics and Human Movement* (Springer, 1996).
23. Brown, D. *Tracker Video Analysis and Modelling Tool* <http://physlets.org/tracker/> (2017).
24. *Webots, Overview* 2017. <https://www.cyberbotics.com/overview> (visited on 21/03/2017).
25. *Controller Programming* 2017. <https://www.cyberbotics.com/doc/guide/controller-programming> (visited on 21/03/2017).
26. *Using the NAO robot* 2017. <https://www.cyberbotics.com/doc/guide/using-the-nao-robot> (visited on 21/03/2017).
27. *Supervisor Programming* 2017. <https://www.cyberbotics.com/doc/guide/supervisor-programming> (visited on 21/03/2017).
28. Scherffig, L. *Reinforcement Learning in Motor Control* BSc. (University of Osnabrbruck, 2002).
29. Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* <https://webdocs.cs.ualberta.ca/~sutton/book/bookdraft2016sep.pdf> (visited on 16/02/2017) (2016).

30. Gaskett, C., Wettergreen, D. & Zelinsky, A. in *Advanced Topics in Artificial Intelligence: 12th Australian Joint Conference on Artificial Intelligence, AI'99 Sydney, Australia, December 6–10, 1999 Proceedings* (ed Foo, N.) 417–428 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1999). ISBN: 978-3-540-46695-6. doi:10.1007/3-540-46695-9_35. <http://dx.doi.org/10.1007/3-540-46695-9_35>.
31. Gaskett, C. *Q-Learning for Robot Control* PhD. (The Australian National University, 2002).
32. *DeepLearningVideoGames Readme* <https://github.com/asrivat1/DeepLearningVideoGames/blob/master/README.md> (2016).
33. Emirik, T. *Torch vs TensorFlow vs Theano* <http://www.ccri.com/2016/12/09/torch-vs-tensorflow-vs-theano/> (2016).
34. PyBrain. *Journal of Machine Learning Research,* 743–746 (2010).
35. *Introducing OpenAI* <https://openai.com/blog/introducing-openai> (2015).
36. *OpenAI Gym* 2016. eprint: arXiv:1606.01540.
37. *FrozenLake-v0* 2016. <https://gym.openai.com/envs/FrozenLake-v0> (visited on 12/03/2017).
38. Russell, S. J. & Norvig, P. *Artificial intelligence: a modern approach (3rd edition)* (Prentice Hall, 2009).
39. *Pendulum-v0* 2016. <https://gym.openai.com/envs/Pendulum-v0> (visited on 19/03/2017).
40. *Introduction to Artificial Neural Networks - Part 1* The Project Spot. <http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7> (2013).
41. Emami, P. *Deep Deterministic Policy Gradients in TensorFlow* <http://pemami4911.github.io/blog/2016/08/21/ddpg-rl.html> (2016).
42. Lillicrap, T. P. *et al.* Continuous control with deep reinforcement learning. *CoRR* **abs/1509.02971.** <http://arxiv.org/abs/1509.02971> (2015).
43. Mnih, V. *et al.* Playing Atari with Deep Reinforcement Learning. *CoRR* **abs/1312.5602.** <http://arxiv.org/abs/1312.5602> (2013).
44. Heess, N., Hunt, J. J., Lillicrap, T. P. & Silver, D. *Memory-based control with recurrent neural networks* <http://rll.berkeley.edu/deeprlworkshop/papers/rdpg.pdf> (2012).
45. Human-level control through deep reinforcement learning. *Nature* **518,** 529–533 (2015).
46. Karpathy, A. *Deep Reinforcement Learning: Pong from Pixels* <http://karpathy.github.io/2016/05/31/rl/> (2016).
47. Lever, G. Deterministic policy gradient algorithms (2014).
48. Peters, J. Policy gradient methods. *Scholarpedia* **5.** revision 137199, 3698 (2010).
49. Keulen, B. *Algorithm on Pendulum v0* <https://gym.openai.com/evaluations/eval_n7JgacQRiK3MMrWFnaz6g> (2017).
50. Retkowski, G. *Actor Critic with OpenAI Gym* <http://www.rage.net/~greg/2016-07-05-ActorCritic-with-OpenAI-Gym.html> (2016).
51. Cheng, S. *Ornstein-Uhlenbeck process* <http://planetmath.org/ornsteinuhlenbeckprocess> (2007).
52. Nielsen, M. *Neural networks and deep learning* http://neuralnetworksanddeeplearning.com/index.html (online, 2017).
53. Plappert, M. *keras-rl* <https://github.com/matthiasplappert/keras-rl> (2017).