UNIVERSITY OF BIRMINGHAM

Optimising Swinging Motion of a NAO Aldebaran Robot on a Hinged Swing

E.Tregoning, H.Birks, H.Bithray, M.Elliott, B.Adams, T.Batchelor,
R.Bouab, R.Dubois, K.Giddha, J.Goldberg, C. Hulme,
A. Mazendame, A.Thanki, A.Thorton

March 23, 2018

Abstract

The aim of this investigation was to optimise the swinging motion of a NAO robot in a realistic swing environment. The swing used was a hinged rod swing intended to model a common rope swing. The mass distribution of the robot was examined and used to improve human modelling of the motion by redistributing the human mass to within 5% of the robot's. It was found that this had little effect on the overall motion as more effort was put in by the human to compensate for the additional mass. In addition, theoretical models were created to investigate optimal swinging motion and were simulated in a C++ program. It was concluded from these simulations that the robot's motion should be applied rapidly at the extrema of the swing's motion with as large a range of motion as possible. A realistic model of the robot and swing was ran, which reached a an amplitude of 34.2°. Four new encoders were built in order to measure the angles of each hinge relative to the primary swing rod axis. This was done in order to measure and record the effects of the hinges. In addition, research was conducted into the use of the robot's inertial sensors. The focus was primarily given to the accelerometer and torso angle data; two filtering methods were applied to improve the raw data. Amplitudes of $(37.0 \pm 0.3)^\circ$ were achieved from pre-determined motions using an initial angle of 8°. Starting from rest an amplitude of $(16.0 \pm 0.3)^\circ$ was achieved in Webots and $(14.8 \pm 0.2)^\circ$ with the real robot. Data from the human modelling was implemented and an amplitude of $(13.0 \pm 0.2)^\circ$ was reached on the real robot. Self-starts using a box yielded initial amplitudes of $(6.9 \pm 0.3)^\circ$ in Webots and $(4.2 \pm 0.2)^\circ$ with the real robot. Another approach towards autonomy was by investigating machine learning. Q-Learning was identified as the best method to approach this and was implemented on the double pendulum model. This was done in both Python and C both demonstrating a distinct difference between learnt actions and random actions.

KEYWORDS: *Robotics, NAO, Pendulum, Webots, Kinovea*

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

Contents

1 Introduction **4**
1.1 Background . . . 4
1.2 Motivation and Report Outline . . . 5
1.3 Background Theory . . . 6

2 Numerical Modelling **7**
2.1 Introduction . . . 7
2.2 Lagrangian Mechanics . . . 7
2.3 Pendulum Motion . . . 8
2.3.1 Simple Plane Pendulum . . . 8
2.3.2 Damped Simple Pendulum . . . 9
2.3.3 Flywheel Pendulum . . . 10
2.4 Analytical Models . . . 11
2.4.1 Double Pendulum . . . 11
2.4.2 Single Flail . . . 12
2.4.3 Triple Pendulum . . . 13
2.4.4 Hinged Single Flail . . . 15
2.5 Numerical Methods . . . 17
2.5.1 Runge–Kutta fourth-order method . . . 17
2.5.2 Decoupling Ordinary Differential Equations . . . 17
2.5.3 Overview of the Simulation Program . . . 18
2.6 Calculation of the Damping Coefficient . . . 19
2.7 Simulation Analysis . . . 21
2.7.1 Normal Modes . . . 21
2.7.2 Resonance . . . 22
2.7.3 Maintaining Amplitude and Parametric Resonance . . . 22
2.7.4 Investigating Phase Relationships . . . 25
2.7.5 Investigating Driving Function Amplitude . . . 27
2.7.6 Creating a Ramp Driving Function . . . 28
2.7.7 Investigating the Periodic Ramp Function . . . 29
2.7.8 Idealised Step Function . . . 31
2.7.9 Investigating the Effect of a Hinge . . . 32
2.8 Realistic Modelling . . . 34
2.8.1 Robot and Swing Properties . . . 34
2.8.2 Simulating a Realistic Model . . . 36
2.9 Conclusions . . . 37
2.10 Extensions . . . 38
2.10.1 Complex Hinged Flail . . . 38
2.10.2 Adding the Arms . . . 39

3 Human Motion **41**
3.1 Introduction . . . 41
3.1.1 Swinging from a Standing Position . . . 41
3.1.2 Swinging from a Seated Position . . . 42
3.1.3 Standing vs. Sitting . . . 42
3.2 Replicating Mass Distribution . . . 43
3.3 Fieldwork and Data Collection . . . 44
3.4 Verifying Motion . . . 46
3.5 Influence of Redistributing Mass . . . 47
3.6 Implementation to Robot . . . 49
3.7 Conclusion . . . 51
3.8 Evaluation . . . 52

4 Encoders **53**
4.1 Justification . . . 53
4.2 Encoder Types . . . 54
4.3 Encoder Build Plan . . . 55
4.4 Build Log . . . 55
4.5 Hinge Encoder API . . . 58
4.6 Results . . . 58
4.7 Data From The Secondary Encoders . . . 59
4.8 Potential Uses of Secondary Encoders . . . 60
4.9 Sources of Error In Encoders . . . 61

5 Inertial Sensors **62**
5.1 Introduction . . . 62
5.2 Gyro-meter . . . 63
5.3 Accelerometer . . . 64
5.3.1 Filtering the data . . . 65
5.3.2 Conclusions on the Accelerometer filtering . . . 65
5.4 Torso Angle Algorithm . . . 66
5.5 The Complementary Filter . . . 67
5.5.1 Conclusions on the Complementary Filter . . . 68
5.6 Evaluation . . . 68

6 Using Webots **69**
6.1 Introduction . . . 69
6.1.1 Installing the software . . . 69
6.1.2 Connecting to Choregraphe . . . 69
6.1.3 Using Choregraphe . . . 70
6.2 Modelling in Webots . . . 70
6.2.1 Introduction . . . 70
6.2.2 The swing model . . . 71
6.2.3 Connecting the virtual robot to the swing . . . 73
6.3 Possible improvements . . . 74

7 Controlling the robot **75**
7.1 Working With Choregraphe . . . 75
7.2 SwingAPI . . . 75
7.3 Full Body Motion . . . 76
7.4 Compatibility Issues . . . 76
7.5 Dynamic Position Feedback . . . 77
7.6 Results with NAO . . . 78
7.7 Asymmetry of Swing Angle Data . . . 78
7.7.1 Initial Conditions . . . 79
7.8 Swinging from Position Feedback using Human Motion . . . 81

8 Machine Learning **83**
8.1 An Introduction to the Machine Learning Group . . . 83
8.2 Theory . . . 83
8.2.1 Applications of Machine Learning . . . 83
8.2.2 Types of Learning . . . 84
8.2.3 Types of Reinforcement Learning . . . 85
8.2.4 Q-Learning . . . 86
8.3 Python . . . 87
8.3.1 Environments . . . 87
8.3.2 Structure . . . 88
8.3.3 Original Model Results . . . 92

8.3.4 New Model Results . . . 96
8.3.5 Conclusion . . . 98
8.4 C . . . 99
8.4.1 Implementation . . . 99
8.4.2 Conclusion . . . 103

9 Self-Start **104**
9.1 Introduction . . . 104
9.2 Investigating the Use of a Block . . . 104
9.3 Constructing the block . . . 105
9.4 Simulating self-start in Webots . . . 106
9.5 Implementing self-start on NAO . . . 107
9.6 Analysis . . . 107

10 Conclusions **108**
10.1 Project Evaluation and Outlook . . . 109

11 Acknowledgements **109**

A Appendix **113**
A.1 Complex Hinged Flail Derivation . . . 113
A.2 Deriving the 4 Bar model . . . 115
A.3 Angular Frequency Change, ω, for Multiple Fits . . . 117
A.4 Q-Learning UML Diagram (Python) . . . 118

1 Introduction

1.1 Background

Benjamin Adams

"If every tool, when ordered, or even of its own accord, could do the work that befits it... then there would be no need either of apprentices for the master workers or of slaves for the lords."

Aristotle, 320 BC

This quote by Aristotle is arguably the genesis of the development of robotics. Although the word 'robot' wouldn't be used for another two millennia, it is remarkable that its conceptual origins trace back so far. In the last few centuries, the field has undergone rapid growth starting in Germany during the industrial revolution. Initially, robots were just mechanical arms, but soon significant progress was made leading to the first humanoid robot, the Automaton Trumpet Player,[1] designed by Friedrich Kaufmann in Dresden in 1810. This robot's only purpose was to produce a trumpet sound and was not intended to be developed further. The robot still exists today and is housed in the Deutsches Museum, Munich. However, despite its retirement, its legacy lives on in the form of modern, humanoid robots such as the Aldebaran NAO and the Erica robot (currently the most advanced humanoid robot). Erica can perform a wide variety of tasks but is set apart from other robots by her artificial intelligence (AI). She can independently move her facial features, neck, shoulders and waist enabling her to interact with humans with incredible autonomy.[2] The development of robots such as Erica has allowed the optimisation of simple human tasks; a principle underpinning this project. Outside of science, throughout the 20^{th} century, robots have been highly prominent in a wide variety of areas, from popular culture such as film and literature; to domestically, with desktop computers and home appliances.

But why are robots so interesting?

The theory of natural selection describes how traits advantageous to a species are inherited by future generations; as a result, driving evolution forward. To date, intellect has proved to be one of these characteristics key to humanities development. To quote Sir Francis Bacon, 'Knowledge is Power,' and so naturally, it is something that humans want more of. Biologically, human brains are restricted by how much knowledge they can hold. However, what if there was a machine that could acquire data, draw conclusions and then develop new technology just like a human, but with infinite capacity? This is somewhat of a reality, with robots already designed with greater mental acuity than that of the brightest humans. An example of this is the recent developments in the field of AI, with computers learning complex board games and competing against humans. In May 2017, Google's AlphaGo AI system dominated the world's top-ranked Go Player in three matches.[3] The game Go has a huge state space and so requires careful examination to select the best move. Therefore, victory was a triumph for AI beating a human at a game that relies strongly on intuition.

Like knowledge, another trait key to human survival has been physical prowess. Even in a world with many complex economic structures, at a basic level society is still driven on a transactional basis, where people perform labour and are paid for their work. The problem with this is that even the most physically fit humans have limited stamina and strength. However, what if a machine could manufacture items just like a human? This would mean infinite, consistent and ultimately cheap labour; all without guilt. This was the motivation behind the automation of simple manufacturing tasks during the industrial revolution and is evident in society today; from car assembly to the currently under development self-piloted planes. The pattern throughout humanity's history has been to constantly strive to be better, even if this means exceeding physical and mental restrictions. As it has been in the past, the path to continuing this in the future will likely be through robotics.

1.2 Motivation and Report Outline

Eleanor Tregoning

Robots are continually being designed to optimise human tasks. There are two main ways this can be done: by optimising the robot's behaviour and by making the robot's environment more realistic. The aim of this project was to get the NAO robot to swing on a hinged swing, which would replicate a human on a rope swing as closely as possible. In choosing to change the swing from previous years to that of a near rope swing, the robot's task was made more realistic. In addition, by adding extra hinges to the swing, the theoretical models for the swing became slightly different and more interesting. In theory, a rope swing should reach higher amplitudes (below an amplitude of 90°) for less driving energy than a rod swing. Another motivation was to find a way to increase the robot's autonomy, again creating a more realistic environment to that of a human on a swing. Overall, the aims were not only to optimise the swinging motion of a robot, but to do so for a more realistic swing model.

In order to investigate this, three areas of interest were identified. Firstly, it was necessary to develop theoretical models for the swing, which began with a double pendulum and evolved to a hinged single flail model These were modelled numerically and analysed to extract idealised driving functions, which were then adapted to fit the limitations of the robot. One can easily picture human motion as an idealised way for the robot to pump the swing, therefore an investigation into human motion was conducted. Several changes have been made to previous years' work which included making changes to the human mass to ensure that the mass distribution was within 5% of the robot's. Following this, the robot and its environment needed to be modelled and simulated in Webots. These simulations were then used and the scripts applied to the real robot. The models were used to inform the best ways to pump the swing and in turn, the limitations of the robot's motion were used to inform any changes that could be made to the models. An investigation was also made into the use of both the external angle encoder sensors and the internal sensors of the robot. It was necessary to build four new encoders to take data from the new hinges on the swing. The internal sensors were investigated and data analysed to assess the potential for their use as a tool for position feedback which would further improve the autonomy of the robot. Lastly, other methods of improving autonomy were investigated in parallel with the rest of the project, in particular, machine learning and implementing self-starts on the robot. This was done using a Q-learning algorithm on a double pendulum model to make it learn when to drive the swing to increase the amplitude. Self-starts were investigated in two ways; kicking off a box and starting from rest.

1.3 Background Theory

Christopher Hulme

At a fundamental level, optimising swinging motion can be described best as increasing the energy and therefore the amplitude of the swing system with the greatest efficiency. There are a number of different mechanisms for which this is achievable. The simplest is to drive the swing periodically with an external force, practically this corresponds to pushing the swing. However, from a physical point of view, this is the most trivial as there is no involvement of the user. The methods of most interest will be those involving internal pumping, this will mean the NAO robot will be able to swing autonomously, ideally from a self-start.

In principle, the core concepts involved in pumping a swing are fairly simple and come naturally to most. The user has the ability to reposition their centre of mass and rotate parts of their body at specific points in the swinging motion. When the user is at the peak of the back-swing, their head and body go backwards, whilst their legs go forward. As for the front-swing, the reverse is the case for the legs, head and body. These actions produce a torque, which is applied to the user's body.

There are two forces acting on the user: The gravitational force, which only acts on the centre of mass of the system; and the tension in the chain/rope. When the swing is stationary, the tension is through the centre of mass, with gravity pulling straight down on the centre of mass. The forces are all along the same axis and balance each other; as a result, there will be no net torque.

When in motion, the only way a torque can be created is through the tension in the chain. The tension cannot pull through the user's centre of mass with the exception of when the swing reaches the bottom of the arc. To keep the tension from passing through the centre of mass at all points of the arc's trajectory, a downward force to a point on the chain to form a pivot can be applied by the user. The line of pull provided by the tension will always end up below the centre of mass.

Looking from the right-side of the user's body, see figure 1.1. As the swing reaches the apex of the back-swing, the user throws their head and body back, legs forwards (i.e the whole body rotates about the centre of mass counter-clockwise) increasing their angular momentum. In doing this, the user also uses their hands to push against the rope, applying a torque and thus forming a pivot. Using simple trigonometry, it can be seen that this raises the centre of mass a small amount to counter the rise in angular momentum such that angular momentum is conserved. In raising the centre of mass, the user is raising the potential energy, which is then consequently converted into kinetic energy. This cycle of countering the increase in angular momentum, and hence increasing potential energy builds and builds such that the amplitude of the swing increases each cycle. This describes, in general, the method and theory of pumping a swing to build amplitude.

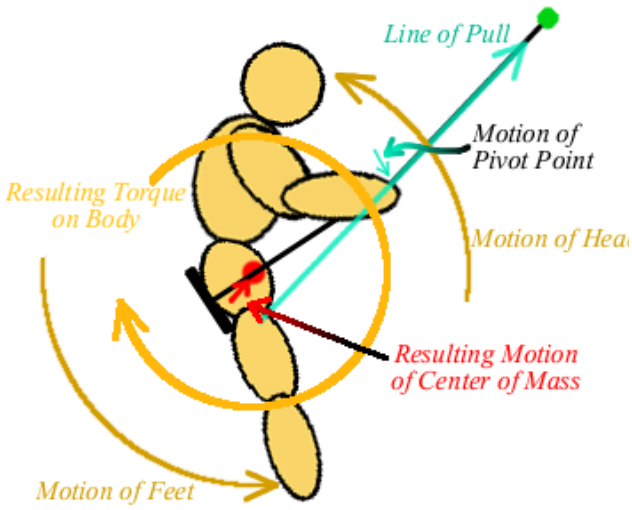


Figure 1.1: Pumping a swing[4]

2 Numerical Modelling

2.1 Introduction

The motivation behind using numerical modelling in this project was to investigate what the ideal swinging motion would be without needing access to the robot. This was achieved by deriving the theory behind swing motion and designing models to represent the robot and swing system. These models would then be simulated using a custom-made computer program, where different parameters would be varied and their effect on swinging motion analysed. Following analysis, the most accurate model would be simulated with realistic parameters (see section 2.8.1) and with optimal motion for comparison with the real robot.

2.2 Lagrangian Mechanics

Josh Goldberg

Lagrangian mechanics was used when developing the models since no conservative forces were considered in the raw theoretical models. It is a convenient alternative to classical Newtonian mechanics, as it only requires a single function to extract all equations of motion of a system.

This function is called the Lagrangian and is defined as

$$\mathcal{L} = T - V \quad , \tag{2.1}$$

where T is defined as the kinetic energy of the system and V is defined as the potential energy of the system. Once the Lagrangian for a system is derived, it can be used to obtain all of the differential equations of motion for a system, with no need for any consideration of forces.

This is achieved by applying the Euler-Lagrange equations, defined as

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) = \frac{\partial \mathcal{L}}{\partial q} \quad . \tag{2.2}$$

As the models increased in complexity, computing Lagrange's equations by hand became cumbersome and so code was obtained to confirm them.[5]

2.3 Pendulum Motion

2.3.1 Simple Plane Pendulum

Benjamin Adams

To demonstrate the use of Lagrangian mechanics, it is applied to the case of the simple pendulum as shown in figure 2.1.

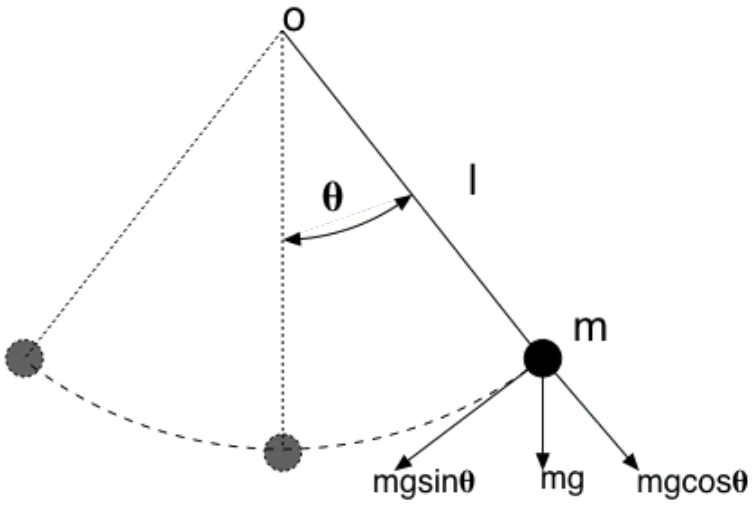


Figure 2.1: Diagram of a Simple Pendulum

The simple pendulum was the starting point for numerical modelling with all subsequent models being extensions of it. In the Cartesian coordinate system, the kinetic energy (T) is as shown in equation 2.3 where m is the mass and $\dot{x}$ and $\dot{y}$ are the velocities in the x and y directions respectively.

$$T = \frac{1}{2}m\left(\dot{x}^2 + \dot{y}^2\right) \tag{2.3}$$

For reference throughout this report the dot notation is used to denote differentiation of a variable with respect to time. The systems considered in this project involve rotation and so it is natural to change variables and use polar coordinates. Using the standard conversions $x = lcos(\theta)$ and $y = lsin(\theta)$ and trigonometric identities the kinetic energy can be written as in equation 2.4

$$T = \frac{1}{2}ml\dot{\theta}^2 \tag{2.4}$$

The only contribution to the potential energy of this system is the gravitational force acting on the mass. The pivot is taken to have a potential of $V = 0$ and so the potential energy of the system is given by equation 2.5.

$$V = -mgy = -mglcos(\theta) \tag{2.5}$$

The Lagrangian for the system is therefore

$$\mathcal{L} = \frac{1}{2}ml\dot{\theta}^2 + mglcos(\theta) \tag{2.6}$$

Note: for these models, gravity is defined as acting downwards. To calculate the acceleration of each variable, the Euler-Lagrange equation is applied. For a simple pendulum of fixed length, only its angle with respect to the vertical changes and so Lagrange's equations need only be applied to θ as $\dot{l} = 0$.

$$\frac{d}{dt}\left(\frac{\partial\mathcal{L}}{\partial\dot{\theta}}\right) = \frac{\partial\mathcal{L}}{\partial\theta} \tag{2.7}$$

$\mathcal{L}$ is the Lagrangian and θ is the generalised coordinate in this case. Applying the Euler-Langrange equation to equation 2.6 and rearranging for $\ddot{\theta}$ the equation of the motion of the simple pendulum is derived.

$$ml^2\ddot{\theta} = -mglsin(\theta)$$
$$\ddot{\theta} = -\frac{g}{l}sin(\theta) \tag{2.8}$$

For small oscillations, equation 2.8 reduces to

$$\ddot{\theta} = -\frac{g}{l}\theta = -\omega_0^2\theta \tag{2.9}$$

Where $\omega_0 = \sqrt{\frac{g}{l}}$ and is the natural oscillation frequency of the pendulum.

The motivation behind the subsequent models is to model the swinging motion of the robot more realistically. This was achieved by considering the addition of a hinge, the mass distribution of the system and also finding the optimum way of driving the robot's limbs to maximise amplitude.

2.3.2 Damped Simple Pendulum

The solution to equation 2.8 for the equation of motion of the simple pendulum results in continuous oscillations. This is very unrealistic, as real world systems are subject to air resistance and friction. These resistive forces cause the system to dissipate energy, which results in the swing amplitude decaying over time. To consider the effect of this a damping term was added into the equations of motion for the main pivot angle ($\ddot{\alpha}$ see section 2.4 for first usage). However, damping was neglected for the other hinges due to the fact that the effect of damping; while present, is fairly negligible. Adding the term to the equation of motion of the simple pendulum leads to equation 2.10.

$$\ddot{\theta} = -\frac{g}{l}sin(\theta) - b\omega \tag{2.10}$$

The coefficient b is a property of the system known as the damping constant (see section 2.6 for its calculation). Making the small angle approximation that $sin(\theta) \approx \theta$ and by relating ω to θ via $\omega = \dot{\theta}$, equation 2.10 can be re-written as equation 2.11.

$$\ddot{\theta} + b\dot{\theta} + \frac{g}{l}sin(\theta) = 0 \tag{2.11}$$

The general solution to this 2^{nd} order differential equation is given in equation 2.12.

$$Ae^{-\frac{b}{2m}t}cos(\omega' t + \phi) \tag{2.12}$$

Where the angular frequency ω' is given by

$$\omega' = \sqrt{\frac{k}{m} - \frac{b^2}{4m^2}} = \sqrt{\frac{g}{l} - \frac{b^2}{4m^2}} = \sqrt{\omega_0^2 - \gamma^2} \tag{2.13}$$

ω_0 is the natural frequency of the undamped simple pendulum and $\gamma = \frac{b}{2m}$ is the damping coefficient (not to be confused with the damping constant b). Note, that if damping is small; such that $\gamma << \omega_0$, then $\omega' \approx \omega_0$ and damping has negligible effect on the oscillation period. Although, as can be seen from equation 2.13 for damped systems, the time period will be greater for an equivalent undamped ones.

As can be seen from equation 2.12, as energy is dissipated, the amplitude of a damped simple pendulum decreases exponentially with time. Naturally, a function of this form was fit to the experimental data to calculate the damping constant.

2.3.3 Flywheel Pendulum

Christopher Hulme

Conservation of angular momentum is a fundamental principle governing how objects rotate. Consider the flywheel pendulum in figure 2.2 shown below.

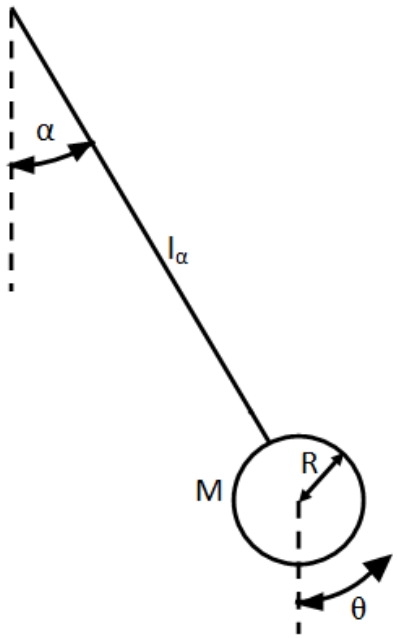


Figure 2.2: Flywheel Pendulum

Defining the origin of the system to be at the pivot, the total angular momentum is the sum of the orbital angular momentum of the flywheel about the pendulum's pivot ($\vec{L}_{Orb}$) and the angular momentum resulting from the spin of the flywheel about its own rotational axis ($\vec{L}_{Spin}$).

$$\vec{L}_{Tot} = \vec{L}_{Orb} + \vec{L}_{Spin} \tag{2.14}$$

Equation 2.14 can be written as follows:

$$L_{Tot} = Ml_{\alpha}^2\dot{\alpha} + \frac{1}{2}MR^2\dot{\theta} \tag{2.15}$$

Taking the system to be initially at rest, in a gravity-free environment with l_α held constant, the flywheel is set into rotation. Equating the initial state and final state provides an expression describing the angular velocity of the flywheel about the pivot in terms of the flywheel's change of spin.

$$\Delta\dot{\alpha} = -\Delta\dot{\theta}\frac{R^2}{2l_{\alpha}^2} \tag{2.16}$$

This demonstrates that there is a linear relationship between the change in the spin of the flywheel and the orbital velocity. If the flywheel's spin velocity is changed, the change in orbital angular velocity of the flywheel about the pivot will oppose this in order to conserve the system's total angular momentum. This change in orbital velocity induces a torque on the system

$$\tau = I\ddot{\alpha} = I\ddot{\theta}\frac{l_{\alpha}^2}{2R^2} \tag{2.17}$$

where I is the moment of inertia of the system.

2.4 Analytical Models

2.4.1 Double Pendulum

Benjamin Adams

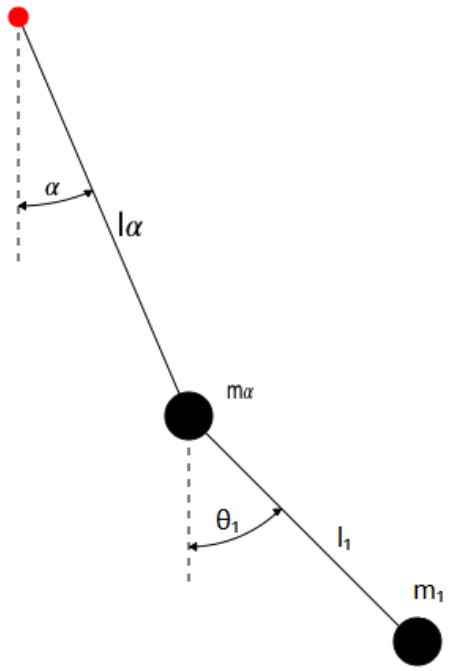


Figure 2.3: Double Pendulum

The system was modelled as a double pendulum (as shown in figure 2.3), where m_α is mass of the robot's torso and the seat of the swing and m_1 is the mass of the robot's legs, which are free to move about the seat. l_α and l_1 are the lengths of the swing and the robot's lower legs respectively. The Lagrangian of the system is expressed in terms of generalised coordinates α and θ_1, which represents the angle with respect to the vertical of the swing and legs respectively. This is shown in equation 2.18.

$$\mathcal{L} = \frac{1}{2}m_\alpha l_\alpha^2\dot{\alpha}^2 + m_\alpha g l_\alpha cos(\alpha) + \frac{1}{2}m_1\big(l_\alpha^2\dot{\alpha}^2 + l_1^2\dot{\theta}_1^2 + 2l_\alpha l_1\dot{\alpha}\dot{\theta}_1 cos(\alpha-\theta_1)\big) + m_1 g\big(l_\alpha cos(\alpha) + l_1 cos(\theta_1)\big) \quad (2.18)$$

Applying Euler-Lagrange equations yields the equations of motion of the system.

$$(m_\alpha + m_1)(l_\alpha\ddot{\alpha} + gsin(\alpha)) + m_1 l_1(\ddot{\theta}_1 cos(\alpha-\theta_1) + \dot{\theta}_1^2 sin(\alpha-\theta_1)) = 0 \quad (2.19)$$

$$l_1\ddot{\theta}_1 + gsin(\theta_1) + l_\alpha\ddot{\alpha}cos(\alpha-\theta_1) - l_\alpha\dot{\alpha}^2 sin(\alpha-\theta_1) = 0 \quad (2.20)$$

Equation 2.19 was re-arranged for $\ddot{\alpha}$ with θ_1 being driven.

$$\ddot{\alpha} = -\frac{\big((m_\alpha + m_1)gsin(\alpha) + m_1 l_1(\ddot{\theta}_1 cos(\alpha-\theta_1) + \dot{\theta_1}^2 sin(\alpha-\theta_1))\big)}{(m_\alpha + m_1)l_\alpha} \quad (2.21)$$

2.4.2 Single Flail

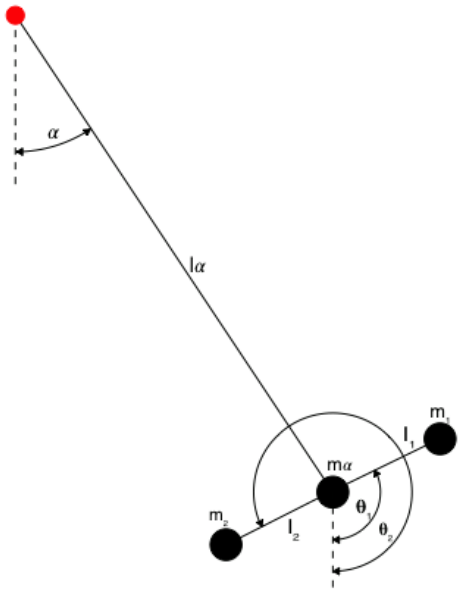


Figure 2.4: Single Flail

A natural extension to the double pendulum system was to consider the single flail system shown in figure 2.4. This model would not only consider driving the robots legs but also its upper body much like how a human swings and also better represent the robot's mass distribution. For this system all variables present in the double pendulum system have the same meaning (see subsection 2.4.1). The additional variables m_2 and l_2 represent the mass and length of the robot's legs respectively and θ_2 is their angle with respect to the vertical.

The Lagrangian of the single flail system is described in terms of generalised coordinates α, θ_1 and θ_2 and is shown in equation 2.22.

$$\begin{aligned}\mathcal{L} = \frac{1}{2}m_\alpha l_\alpha^2\dot{\alpha}^2 + \frac{1}{2}m_1(l_\alpha^2\dot{\alpha}^2 + l_1^2\dot{\theta}_1^2 + 2l_\alpha l_1\dot{\alpha}\dot{\theta_1}cos(\alpha-\theta_1)) + \frac{1}{2}m_2(l_\alpha^2\dot{\alpha}^2 + l_2^2\dot{\theta}_2^2 + 2l_\alpha l_2\dot{\alpha}\dot{\theta_2}cos(\alpha-\theta_2)) \\ +m_\alpha l_\alpha gcos(\alpha) + m_1 g(l_\alpha cos(\alpha) + l_1 cos(\theta_1)) + m_2 g(l_\alpha cos(\alpha) + l_2 cos(\theta_2))\end{aligned} \tag{2.22}$$

Applying Euler-Lagrange equations yields the equation of motion for $\ddot{\alpha}$.

$$\begin{aligned}(m_\alpha + m_1 + m_2)(l_\alpha\ddot{\alpha} + gsin(\alpha)) + m_1 l_1(\ddot{\theta}_1 cos(\alpha-\theta_1) + \dot{\theta}_1^2 sin(\alpha-\theta_1)) \\ +m_2 l_2(\ddot{\theta}_2 cos(\alpha-\theta_2) + \dot{\theta}_2^2 sin(\alpha-\theta_2)) = 0\end{aligned} \tag{2.23}$$

Re-arranging equation 2.23 for $\ddot{\alpha}$.

$$\ddot{\alpha} = \frac{-1}{M_T l_\alpha}\begin{pmatrix} M_T gsin(\alpha) + m_1 l_1(\ddot{\theta_1}cos(\alpha-\theta_1) + \dot{\theta_1}^2 sin(\alpha-\theta_1)) \\ +m_2 l_2(\ddot{\theta_2}cos(\alpha-\theta_2) + \dot{\theta_2}^2 sin(\alpha-\theta_2)) \end{pmatrix} \tag{2.24}$$

Note that if both arms of the flail are held stationary such that $\ddot{\theta_1} = \ddot{\theta_2} = \dot{\theta_1} = \dot{\theta_2} = 0$ then equation 2.24 reduces to the equation of motion of a simple pendulum of length l_α (equation 2.8).

2.4.3 Triple Pendulum

Christopher Hulme

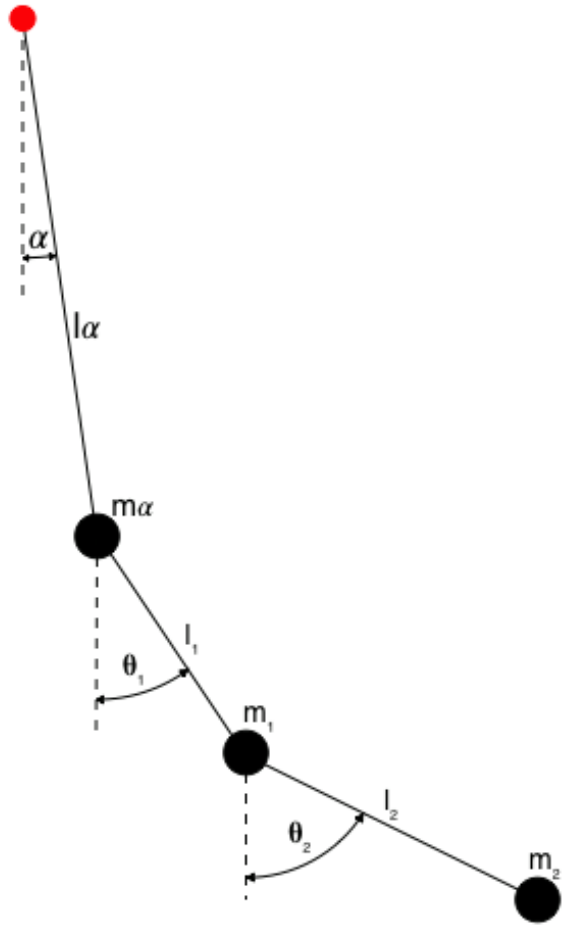


Figure 2.5: Triple Pendulum

To extend the models further one can take the double pendulum and add another mass to form a triple pendulum as shown in figure 2.5. The aim of this model is to represent the robot sitting on the swing using only its legs to drive as in the case with the double pendulum. However, this model incorporates a hinge to simulate a rope swing. From figure 2.5, m_2 is the mass of the robot's freely swinging legs, m_1 is the mass of the seat and the robot's body which is fixed into an upright position. In our case, m_α represents the hinge of the swing. Although the hinges are considered to be massless, it has been included in the equations of motion, but it can be removed by setting its value to be zero. l_α represents the upper portion of the swing's rod, l_1 is the length of the lower portion of the swing below the hinge and l_2 is the length of the robot's legs to its centre of mass. The Lagrangian of the triple pendulum system is shown in equation 2.25.

$$\begin{aligned}\mathcal{L} = \frac{1}{2}m_\alpha l_\alpha^2\dot{\alpha}^2 + \frac{1}{2}m_1(l_\alpha^2\dot{\alpha}^2 + l_1^2\dot{\theta_1}^2 + 2l_\alpha l_1\dot{\alpha}\dot{\theta_1}cos(\alpha-\theta_1)) + \frac{1}{2}m_2(l_\alpha^2\dot{\alpha}^2 + l_1^2\dot{\theta_1}^2 + l_2^2\dot{\theta_2}^2 \\ +2l_\alpha l_1\dot{\alpha}\dot{\theta_1}cos(\alpha-\theta_1) + 2l_\alpha l_2\dot{\alpha}\dot{\theta_2}cos(\alpha-\theta_2) + 2l_1 l_2\dot{\theta_1}\dot{\theta_2}cos(\theta_1-\theta_2)) \\ m_\alpha l_\alpha g cos(\alpha) + m_1 g(l_\alpha cos(\alpha) + l_1 cos(\theta_1)) + m_2 g(l_\alpha cos(\alpha) + l_1 cos(\theta_1) + l_2 cos(\theta_2))\end{aligned} \tag{2.25}$$

Applying Euler Lagrange Equations yields the equations of motion of the triple pendulum.

$$\begin{aligned} M_T(l_\alpha\ddot{\alpha} + gsin(\alpha)) + (m_1+m_2)l_1(\ddot{\theta}_1 cos(\alpha-\theta_1) + \dot{\theta}_1^2 sin(\alpha-\theta_1)) \\ +m_2 l_2(\ddot{\theta}_2 cos(\alpha-\theta_2) + \dot{\theta}_2^2 sin(\alpha-\theta_2)) = 0 \end{aligned} \tag{2.26}$$

$$\begin{aligned} (m_1+m_2)(gsin(\theta_1) + l_1\ddot{\theta}_1 + l_\alpha\ddot{\alpha}cos(\alpha-\theta_1) - l_\alpha\dot{\alpha}^2 sin(\alpha-\theta_1)) \\ +m_2 l_2(\ddot{\theta}_2 cos(\theta_1-\theta_2) + \dot{\theta}_2^2 sin(\theta_1-\theta_2)) = 0 \end{aligned} \tag{2.27}$$

Note that if both the extensions l_1 and l_2 are held stationary ($\dot{\theta_1} = \dot{\theta_2} = \ddot{\theta_1} = \ddot{\theta_2} = 0$) relative to l_α, then the motion returns to that of a simple pendulum.

Now that a hinge has been introduced, the equations of motion need to be decoupled. Once the equations of motion have been extracted, they can be manipulated to take the form $\ddot{\alpha} = -(B\ddot{\theta_1} + C)$ and similar for $\ddot{\theta_1}$. In order for these equations to be useful, they had to be adjusted such that $\ddot{\alpha}$ was not in terms of itself or $\ddot{\theta_1}$ This is known as decoupling 2^{nd} order ODEs and involves rearranging the equations such that they can be processed by the Runge Kutta algorithm. This process will be applied to all models with hinges. Below is the process applied to the triple pendulum.

Equation 2.28 is a rearrangement of equation 2.26 showing $\ddot{\alpha}$ in its original form pre-decoupling.

$$\ddot{\alpha} = -(B\ddot{\theta_1} + C) \tag{2.28}$$

$$B = \frac{(m_1+m_2)l_1 cos(\alpha-\theta 1)}{M_T l_\alpha} \tag{2.29}$$

$$C = \frac{M_T gsin(\alpha) + (m_1+m_2)l_1(\dot{\theta_1}^2 sin(\alpha-\theta_1)) + m_2 l_2(\ddot{\theta_2}cos(\alpha-\theta_2) + \dot{\theta_2}^2 sin(\alpha-\theta_2))}{M_T l_\alpha} \tag{2.30}$$

Equation 2.31 is a rearrangement of equation 2.27 which is of the same form as equation 2.28

$$\ddot{\alpha} = -(D\ddot{\theta_1} + E) \tag{2.31}$$

$$D = \frac{(m_1+m_2)l_1}{l_\alpha cos(\alpha-\theta 1)} \tag{2.32}$$

$$E = \frac{(m_1+m_2)(gsin(\theta_1) - l_\alpha\dot{\alpha}^2 sin(\alpha-\theta_1)) + m_2 l_2(\ddot{\theta_2}cos(\theta_1-\theta_2) + \dot{\theta_2}^2 sin(\theta_1-\theta_2))}{l_\alpha cos(\alpha-\theta 1)} \tag{2.33}$$

Now equations 2.28 and 2.31 are able substituted into one another and decouple them producing a form which can be processed by the Runge Kutta algorithm.

$$\ddot{\alpha} = \frac{BE - CD}{D - B} \tag{2.34}$$

$$\ddot{\theta_1} = \frac{C - E}{D - B} \tag{2.35}$$

When a person swings, they apply a torque at the pivot they form (in this case at the hinge) to raise their centre of mass and thus increase their potential energy in order to pump the swing. In order to simulate driving a swing as realistically as possible, a periodically dependant driving function should be applied to the hinge such that it replicates the raising of centre of mass as the swing reaches its maximum amplitudes. This should be applied to all models that include a hinge that replicates forming a pivot on a rope swing.

2.4.4 Hinged Single Flail

Benjamin Adams and Christopher Hulme

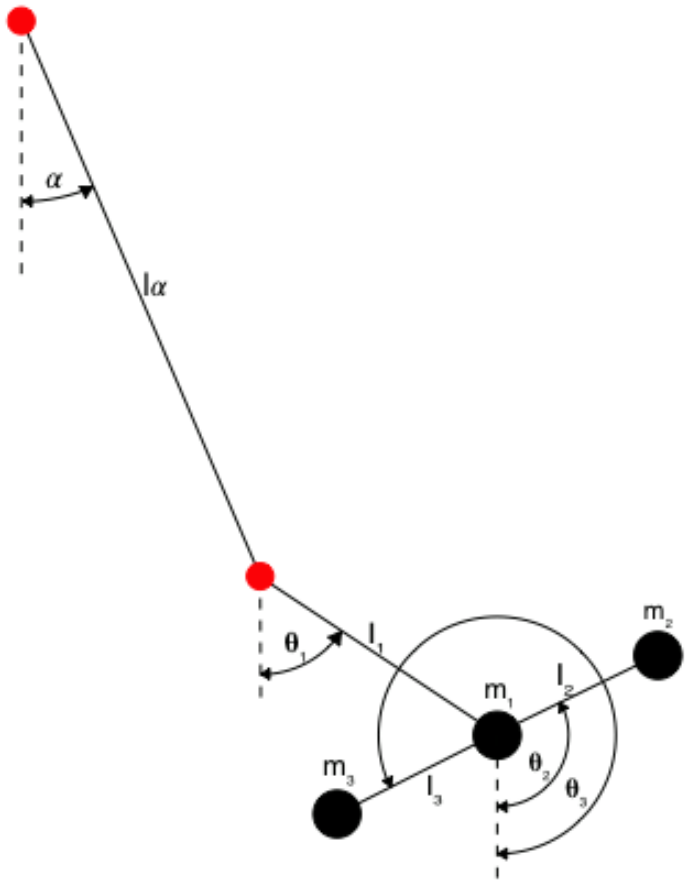


Figure 2.6: Hinged Flail Model

In the 2016 report[6] the hinged flail model (shown in figure 2.6) was considered the best representation of the robot-swing system. Mass m_1 represents the mass of the swing and the rigid upper leg part of the robot. This model is similar to the single flail model, where masses m_2 and m_3 represent the robot's upper body and legs respectively both of which can rotate about the seat (M_T is the total mass of $m_{1,2,3}$). Similar to the case of the triple pendulum model, the pivot point represents a hinge by treating the aforementioned m_α as zero. l_α represents the distance from the main pivot to the hinge, l_1 the distance from the hinge to the seat and l_2 and l_3 are the distances from the seat to the to upper body and legs respectively. An issue with this model is the fact that m_1 is not where the centre of mass of the swing should be. This can be addressed by adding more masses to more realistically distribute the mass of system as seen in more complex models.

The Lagrangian of the system is described by generalized coordinates α, θ_1, θ_2 and θ_3. These correspond to the angles relative to the vertical of the swing below the main pivot, the swing below the hinge and the two parts at end of the flail. The Lagrangian of the system is given by equation 2.36 and equals the sum of the Lagrangians of the seat mass, leg and torso masses represented by a double pendulum and two triple pendulums respectively. This relied on the calculation of the Lagrangian of the triple pendulum and so built on the previous model (see 2.4.3 for more details)

$$\mathcal{L}_{Total} = \mathcal{L}_{Double_{Seat}} + \mathcal{L}_{Triple_{Torso}} + \mathcal{L}_{Triple_{Legs}}$$
$$\mathcal{L} = \frac{1}{2}m_1(l_\alpha^2\dot{\alpha}^2 + l_1^2\dot{\theta}_1^2 + 2l_\alpha l_1\dot{\alpha}\dot{\theta_1}cos(\alpha-\theta_1)) + \frac{1}{2}m_2(l_\alpha^2\dot{\alpha}^2 + l_1^2\dot{\theta_1}^2 + l_2^2\dot{\theta_2}^2$$
$$+2l_\alpha l_1\dot{\alpha}\dot{\theta_1}cos(\alpha-\theta_1) + 2l_\alpha l_2\dot{\alpha}\dot{\theta_2}cos(\alpha-\theta_2) + 2l_1l_2\dot{\theta_1}\dot{\theta_2}cos(\theta_1-\theta_2))$$
$$+\frac{1}{2}m_3(l_\alpha^2\dot{\alpha}^2 + l_1^2\dot{\theta_1}^2 + l_3^2\dot{\theta_3}^2 + 2l_\alpha l_1\dot{\alpha}\dot{\theta_1}cos(\alpha-\theta_1) + 2l_\alpha l_3\dot{\alpha}\dot{\theta_3}cos(\alpha-\theta_3) + 2l_1l_3\dot{\theta_1}\dot{\theta_3}cos(\theta_1-\theta_3))$$
$$m_1g(l_\alpha cos(\alpha) + l_1cos(\theta_1)) + m_2g(l_\alpha cos(\alpha) + l_1cos(\theta_1) + l_2cos(\theta_2)) + m_3g(l_\alpha cos(\alpha) + l_1cos(\theta_1) + l_3cos(\theta_3)) \tag{2.36}$$

Applying Euler Lagrange Equations yields the equations of motion of the hinged flail model. Lagrange's equations associated with angles θ_2 and θ_3 are not included as these angles are pre-determined by driving functions. The rearranged equations of motion for the hinged single flail model are shown below.

$$\ddot{\alpha} = -\frac{1}{M_T l_\alpha}\begin{pmatrix} M_T(gsin(\alpha) + l_1\dot{\theta_1}^2sin(\alpha-\theta_1) + l_1\ddot{\theta_1}cos(\alpha-\theta_1)) \\ +m_2l_2(\ddot{\theta_2}cos(\alpha-\theta_2) + \dot{\theta_2}^2sin(\alpha-\theta_2)) \\ +m_3l_3(\ddot{\theta_3}cos(\alpha-\theta_3) + \dot{\theta_3}^2sin(\alpha-\theta_3)) \end{pmatrix} \tag{2.37}$$

$$\ddot{\theta_1} = -\frac{1}{M_T l_1}\begin{pmatrix} M_T(gsin(\theta_1) + l_\alpha\ddot{\alpha}cos(\alpha-\theta_1) - l_\alpha\dot{\alpha}^2sin(\alpha-\theta_1)) \\ Enco + m_2l_2(\ddot{\theta_2}cos(\theta_1-\theta_2) + \dot{\theta_2}^2sin(\theta_1-\theta_2)) \\ +m_3l_3(\ddot{\theta_3}cos(\theta_1-\theta_3) + \dot{\theta_3}^2sin(\theta_1-\theta_3)) \end{pmatrix} \tag{2.38}$$

To analyse more complex models such as this, the need to decouple the 2nd Order ordinary differential equations (ODEs) is necessary. To achieve this equation 2.37 was separated into two parts of the form shown in equation 2.39.

$$\ddot{\alpha} = -(B\ddot{\theta_1} + C) \tag{2.39}$$

$$B = \frac{l_1cos(\alpha-\theta_1)}{l_\alpha} \tag{2.40}$$

$$C = \frac{1}{M_T l_\alpha}\begin{pmatrix} M_T(gsin(\alpha) + l_1\dot{\theta_1}^2sin(\alpha-\theta_1)) \\ +m_2l_2(\ddot{\theta_2}cos(\alpha-\theta_2) + \dot{\theta_2}^2sin(\alpha-\theta_2)) \\ +m_3l_3(\ddot{\theta_3}cos(\alpha-\theta_3) + \dot{\theta_3}^2sin(\alpha-\theta_3)) \end{pmatrix} \tag{2.41}$$

Equation 2.38 was also written in a form similar to that in equation 2.39.

$$\ddot{\theta_1} = -(D\ddot{\alpha} + E) \tag{2.42}$$

$$D = \frac{l_\alpha cos(\alpha-\theta_1)}{l_1} \tag{2.43}$$

$$E = \frac{1}{M_T l_1}\begin{pmatrix} M_T(gsin(\theta_1) - l_\alpha\dot{\alpha}^2sin(\alpha-\theta_1)) \\ +m_2l_2(\ddot{\theta_2}cos(\theta_1-\theta_2) + \dot{\theta_2}^2sin(\theta_1-\theta_2)) \\ +m_3l_3(\ddot{\theta_3}cos(\theta_1-\theta_3) + \dot{\theta_3}^2sin(\theta_1-\theta_3)) \end{pmatrix} \tag{2.44}$$

Equating equations 2.39 and 2.42 and re-arranging for $\ddot{\alpha}$.

$$\ddot{\alpha} = \frac{BE - C}{1 - BD} \tag{2.45}$$

$\ddot{\alpha}$ has been re-arranged such that it is not a function of $\ddot{\theta_1}$ and so can be solved using the Runge-Kutta algorithm. Thus $\ddot{\alpha}$ can now be taken and inserted into equation 2.42 to obtain $\ddot{\theta_1}$.

$$\ddot{\theta_1} = \frac{CD - E}{1 - BD} \tag{2.46}$$

2.5 Numerical Methods

Max Elliott

2.5.1 Runge–Kutta fourth-order method

For each of the pendulum models developed to simulate swinging motion, the equation of motion for the main swing pivot, α, could be derived as a 2^{nd} order ordinary differential equation. To investigate how changing different parameters in the models changed $\alpha(t)$, these equations would have to be solved. However, these differential equations were too complicated to solve analytically and therefore numerical methods had to be implemented.

The particular method chosen was the fourth-order Runge-Kutta method (RK4), which comes from a family of numerical methods that approximate the solution function to first order ODEs using an iterative technique.[7] It does this by starting at some initial value on the function, given by initial conditions, and then advancing the function along a step by approximating the gradient of the function at that point. By performing many steps, an approximation of the function is built up over a wide range of values.

Consider a first order ODE of the form

$$\frac{d}{dt}y(t) = f(t, y) \tag{2.47}$$

where f is a linear function dependent on t and y, which has initial conditions $y_0 = y(t_0 = 0)$ and t is some independent variable (time in the case of the models). $y(t)$ can be approximated in steps: for a step size h, $y_1 = y(t_0 + h)$. It finds y_1 by approximating the gradient of the function four times over the step. In the form of an iterative equation $y_{n+1} = y(t_0 + nh)$ is found by

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.48}$$

where

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \tfrac{h}{2}, y_n + \tfrac{k_1}{2}) \\ k_3 &= hf(t_n + \tfrac{h}{2}, y_n + \tfrac{k_2}{2}) \\ k_4 &= hf(t_n + h, y_n + k_3). \end{aligned}$$

Approximate values of $y(t)$ can be found over many steps to build up an approximation of the overall function.

RK4 was chosen due to it having a good balance between accuracy and computational efficiency. It has a global accumulated error of $\mathcal{O}(h^4)$ and so, assuming the step size is sufficiently small, this method will produce good approximate solutions to the ODEs produced by the pendulum models. Using methods of lower order such as Euler's Method would improve computational efficiency at the expense of accuracy whereas using higher order methods would have had the converse effect.

2.5.2 Decoupling Ordinary Differential Equations

While the equation of motion produced by each model is a second order ODE, RK4 can only be used on first order ODEs. To work around this, each equation of motion must be decoupled, a process which converts one second order ODE into two first order ODEs. Take as an example a second order ODE of the form

$$\frac{d^2}{dt^2}y(t) = f(t, y). \tag{2.49}$$

By simply setting

$$\frac{d}{dt}y(t) = p(t) \tag{2.50}$$

the ODE becomes

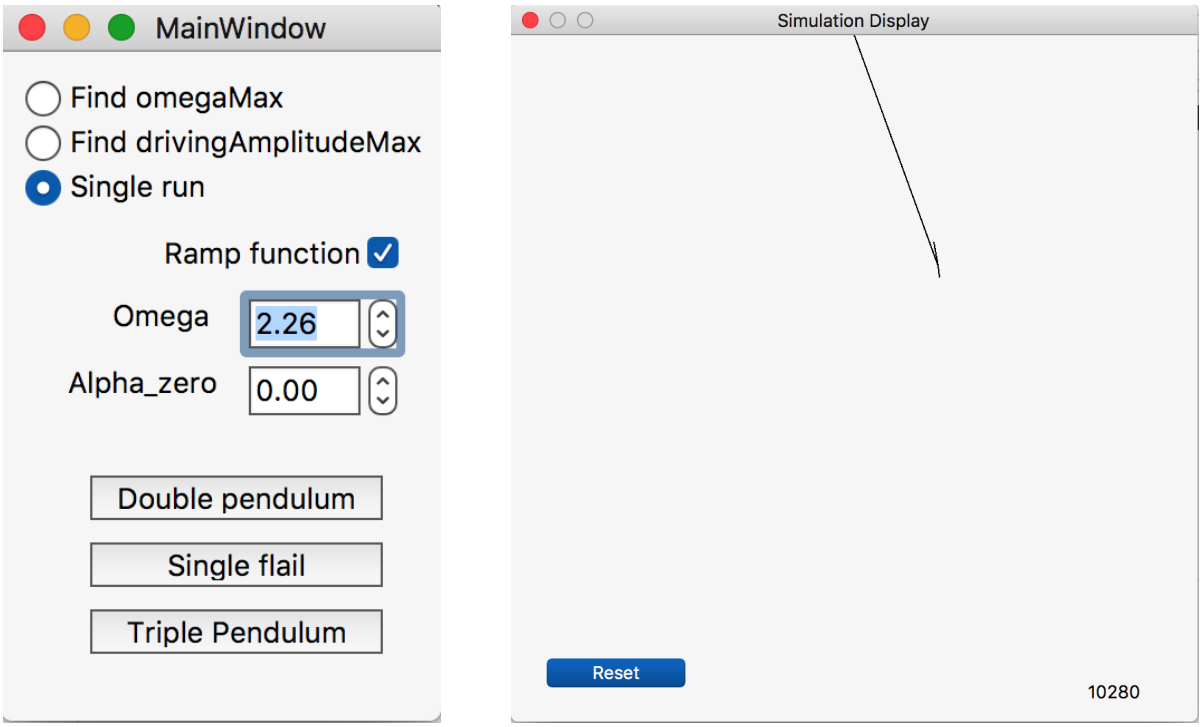
$$\frac{d}{dt}p(t) = f(t, y). \tag{2.51}$$

Equations 2.50 and 2.51 are two ODEs whose approximate solutions can now be found by performing RK4 on them simultaneously. In the case of this report, this meant that both α and $\dot{\alpha}$ could be found simultaneously for each simulation, which would prove useful in later analysis.

2.5.3 Overview of the Simulation Program

To run simulations of the models a C++ program was written where the various parameters of the models could be specified and the initial conditions of the environment set via a GUI (see figure 2.7a). The program can perform single runs of the double pendulum, single flail, and triple pendulum models as well as calculate the optimal driving frequency and driving amplitude for a selected model. The program solves the equations of motion for each model using the fourth order Runge-Kutta method using a step size of 0.001 over a time specified by the user. The iterative solution data is outputted to a .txt for analysis in MATLAB. It also has a visual display that plays the simulation in real time, to easily check if the model is running as expected (shown in figure 2.7b).

The program also was originally designed to run simulations of the hinged flail model, but due to complications in getting the model running in time, it was replaced by the triple pendulum, as all the required analysis could be performed on this model instead.



(a) Program GUI

(b) Simulation Display Window

Figure 2.7: Display Windows of the C++ Program

2.6 Calculation of the Damping Coefficient

Max Elliott

To make the models more akin to the physical swing environment, it was decided that a damping term should be added to the equations of motion for each, corresponding to damping around the main pivot of the swing. Adding a damping term to the equations of motion for the hinges in the hinged models was considered, but it was determined that it would have a negligible effect on the overall behaviour of the system and so was not included.

To determine the damping coefficient of the real swing the robot was allowed to swing freely for three minutes. The angle of the swing over time was tracked and recorded using the main pivot encoder; a plot of the data can be seen in figure 2.8.

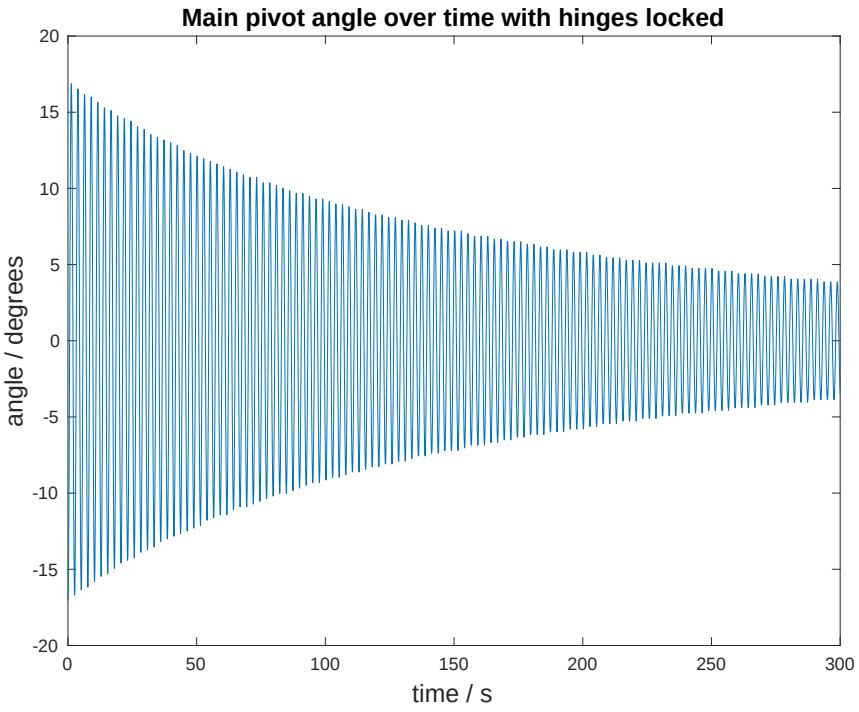


Figure 2.8: Swing angle over time when left to swing freely

This plot shows that the swing amplitude over time follows the behaviour of a damped harmonic oscillator. Therefore, the motion of the swing can be described as a damped simple pendulum (see section 2.3.2) with motion governed by equation 2.10.

The solution to equation of motion of a damped simple pendulum is an exponentially decaying sinusoid (see equation 2.12). The $e^{-\frac{b}{2m}t}$ term is the decay term due to damping and is responsible for the decreasing envelope of the peaks. By taking the peaks and fitting a function of the form $Ae^{-\gamma t}$, where A and γ are constants, the damping constant can be found as $b = 2m\gamma$. MATLAB *Curve Fitting Tool* was used to produce the fit shown in figure 2.9. From the fit, it was found that $\gamma = 0.00533 \pm 0.00008$ and so the damping constant was calculated to be $b = 0.0552 \pm 0.0009 kgs^{-1}$ (taking the mass of the robot as 5.1825 kg from the Aldebaran specifications).

To incorporate damping into the simulated models, a simple damping term can be added to each model's equation of motion. The added term is proportional to the angular velocity of the swing with constant of proportionality b as in equation 2.11.

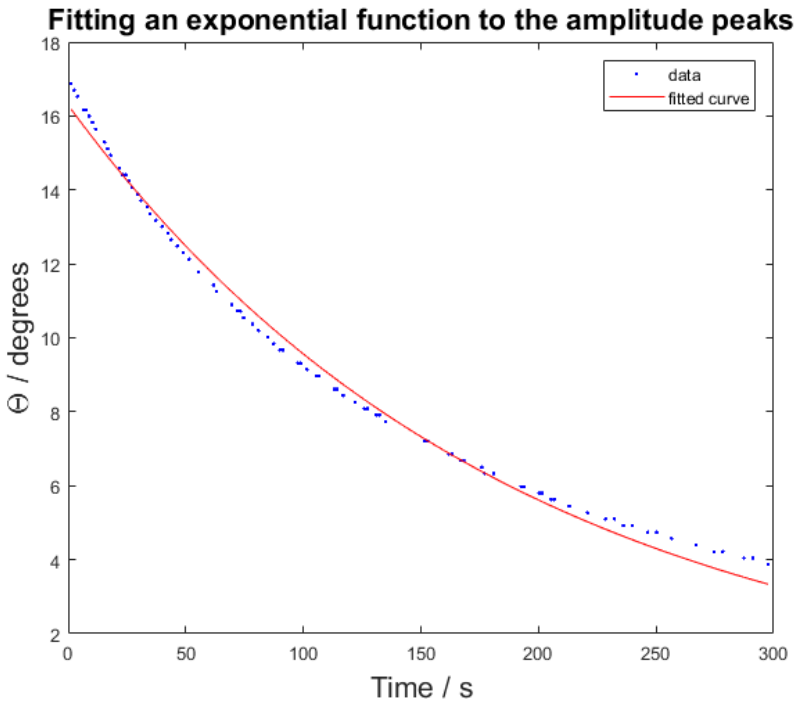


Figure 2.9: Graph showing the fitted function $Ae^{-\gamma t}$ to the peak amplitude data. The function had fitted parameters $A = 16.29$ and $\gamma = 0.005327$.

2.7 Simulation Analysis

2.7.1 Normal Modes

Benjamin Adams

To obtain the normal mode frequencies of the systems the standard normal mode analysis method was applied first approximating the Lagrangians and then solving matrix equations. For a double pendulum the frequencies are given by the solutions to equation 2.52.[8]

$$m_\alpha l_\alpha l_1 \omega^4 - (m_\alpha + m_1)(l_\alpha + l_1)g\omega^2 + (m_\alpha + m_1)g^2 = 0 \tag{2.52}$$

Using the simulation parameter values $l_\alpha = 1.79$ m, $l_1 = 0.2$ m and $m_\alpha = m_1 = 2$ kg the normal mode frequencies for the double pendulum system are $\omega_0 = 2.275\, rad\, s^{-1}$ and $\omega_1 = 10.192\, rad\, s^{-1}$. The normal modes of the other models were obtained, ω_0 is of main interest as it is the frequency of the mode associated with regular swinging motion about the main pivot. The value obtained for these models was the same as the double pendulum ω_0 to within the implied precision indicating that for an ideal system without damping that this would be the ideal theoretical driving frequency.

The natural frequency of the robot-swing system was calculated so the theoretical ω_0 values could be compared with it. This was accomplished by using the data obtained for the damping coefficient analysis (see section 2.6) and extracting ω_0 from the angle against time fitting as shown in figure 2.10.

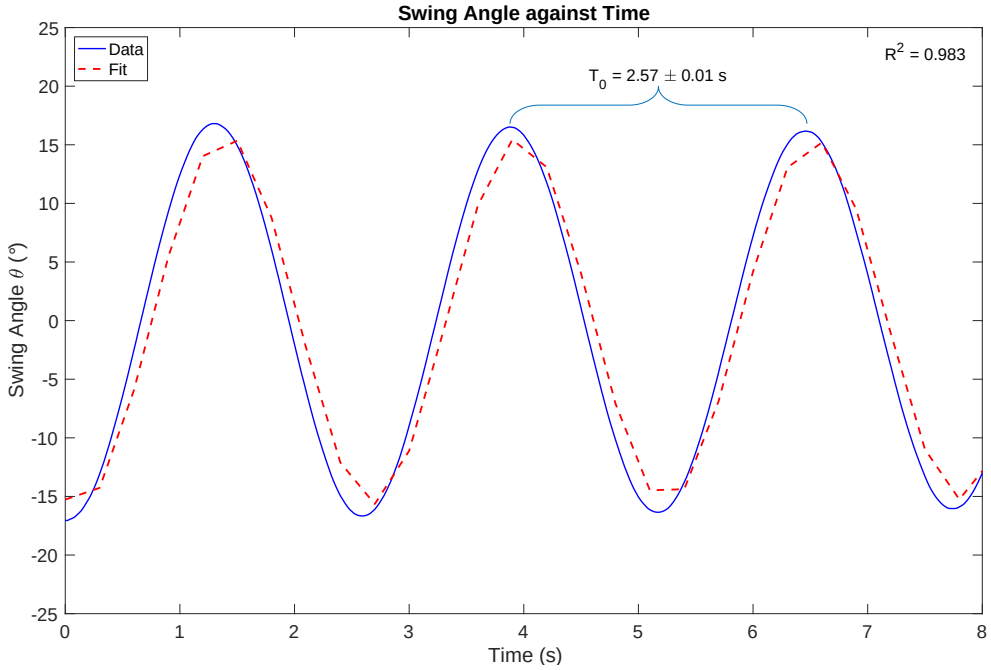


Figure 2.10: Extracting the natural frequency of the NAO Robot on the Swing

From the analysis ω_0 was found to be $2.45\, rad\, s^{-1}$. This value agreed strongly with that obtained in the previous years study[9] and was used in the Webots simulations (see section 6.2). This value is $\approx 8\%$ larger than the value obtained from the normal mode analyses and while this is not a huge disparity it is worthy of note. This would seem to suggest that damping is unlikely to have a significant impact on the resonant frequency. Typically a damped system will have a lower resonant frequency than an equivalent undamped system. The fact that the actual damped system has a higher resonant frequency than the undamped models would indicate that there is a more dominant factor than damping in determining ω_0.

2.7.2 Resonance

Benjamin Adams

Resonance is the phenomena whereby a system oscillates at greater amplitudes when driven at certain frequencies. When a system is driven the driving term may be complicated consisting of many frequencies and so initially on application the resulting motion may be complex. However, over time the nature of resonance dictates that the vibrating object will respond greatest at specific frequency components; these are referred to as its resonant frequencies. In terms of optimum swing motion these frequencies are of interest as they will determine the ideal driving frequencies.[10] As discussed in section 2.7.1 ω_0 corresponds to the natural oscillation frequency of the double pendulum about the main pivot. To find this ω was increased from 0 to 4 rad s^{-1} in 0.01 steps. Each run lasted 100s and the maximum amplitude was recorded and plotted against ω in terms of ω_0 (obtained from the normal mode analysis). The sharp maximum in figure 2.11 indicates that the system undergoes a significant increase in amplitude when driven at ω_0. This would support the conclusion that to maximise amplitude the system should be driven at its natural frequency.

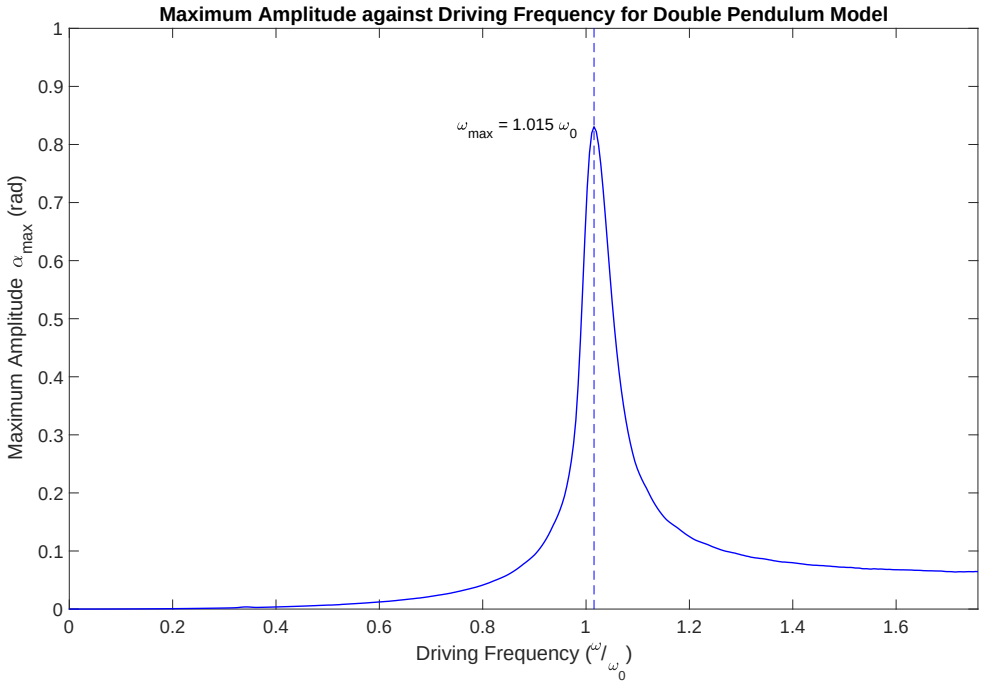


Figure 2.11: Resonance for Double Pendulum

2.7.3 Maintaining Amplitude and Parametric Resonance

Driving the system with a constant periodic driving function at the resonant frequency of the system causes an issue known as parametric resonance to arise. As the driving force goes in and out of phase with the motion of the system the amplitude of the swinging will oscillate between a maximum and zero as can be seen in energy plots in figures 2.12 and 2.13. The collapse is due the fact that driving the system causes its oscillatory time period to change. Practically this can be explained by the motion of the robot's limbs altering the vertical position of its centre of mass changing the "effective" length of the swing and thus varying its oscillation frequency. It is worth noting that the sharpness of the resonance reflects just how sensitive the system's energy is to changes in frequency. This can be seen by comparing the y axis scales on the energy plots, figure 2.12 shows the energy of the system driven near to the resonance and figure 2.13 the energy driven at resonance.

There are two main ways of preventing the decline in amplitude. The first is to only start driving the motion when the system has reached a particular amplitude relative to its previous one. Alternatively, as the driving force effectively changes the natural frequency of the system, the frequency of the driving force could be recalculated constantly by averaging the periods of previous cycles. This would result in the driving function being redetermined each time the system passes through equilibrium. Driven motion would only commence after a certain proportion of the time period had elapsed. Practically the method of dynamically recalculating the driving frequency would be preferable due to the system not being completely idealised and affected by factors such as damping. The first method has the problem that the system will not necessarily reach a specified amplitude where the driven motion is set to start. However, the frequency of the oscillations is unaffected by their amplitude and so in real life constantly re-calculating the time period of the oscillation is therefore a better method of maintaining amplitude. Overall, the issue of parametric resonance highlights that a pre-determined driving function would not be the most effective method of programming swinging motion with the NAO.

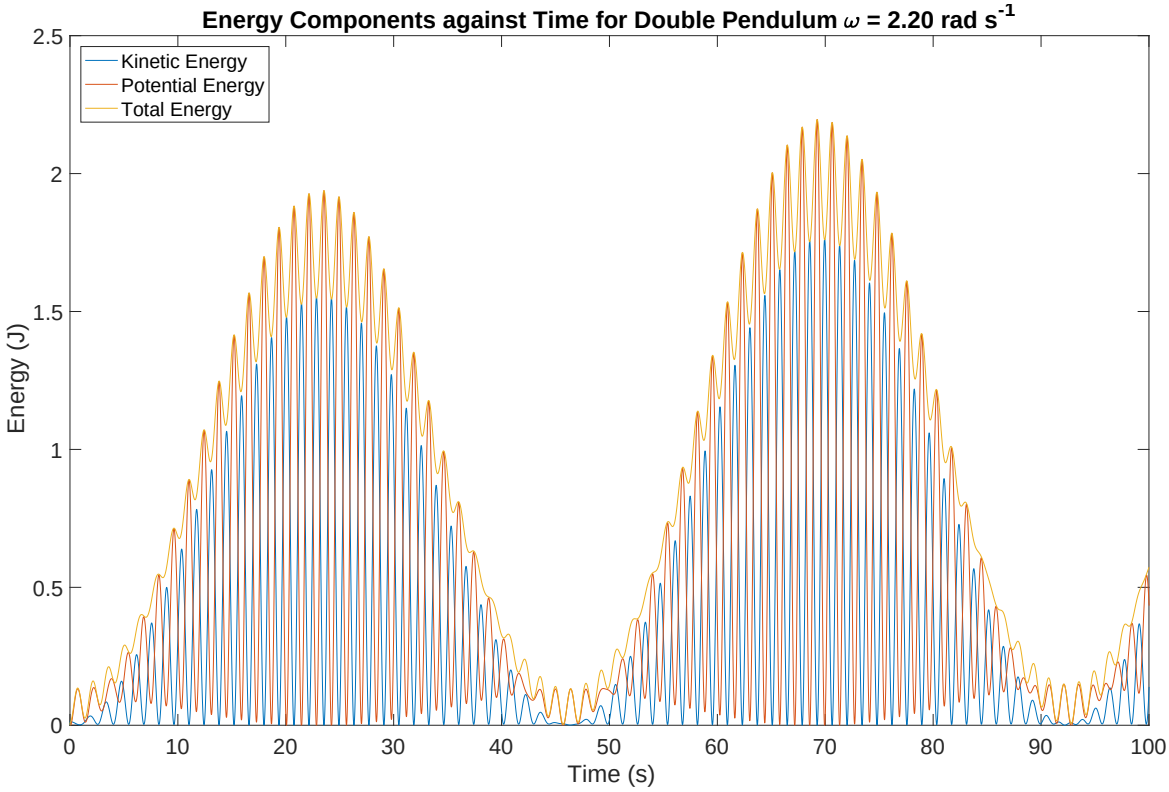


Figure 2.12: Energy Components for Double Pendulum $\omega = 2.20\ rad\ s^{-1}$

To investigate how the robot pumping its legs injects energy into the system the peaks in the energy plot at resonance were extracted (figure 2.13) and fitted using linear regression; the R^2 parameter for the fit was good suggesting that the linear model was suitable. By comparing peaks in the total energy envelope the growth of energy of the system can roughly be obtained.

From the extracted coefficients the gradient of the line A would be indicative of the average energy gained by the system every second due to the applied driving force. Therefore, the fit would suggest that the system gains $\approx 0.1\ J\ s^{-1}$ when driven at resonance. Naturally, one would expect this to saturate at some point as the system cannot continue to gain energy indefinitely and so it would be interesting for future studies to investigate this further.

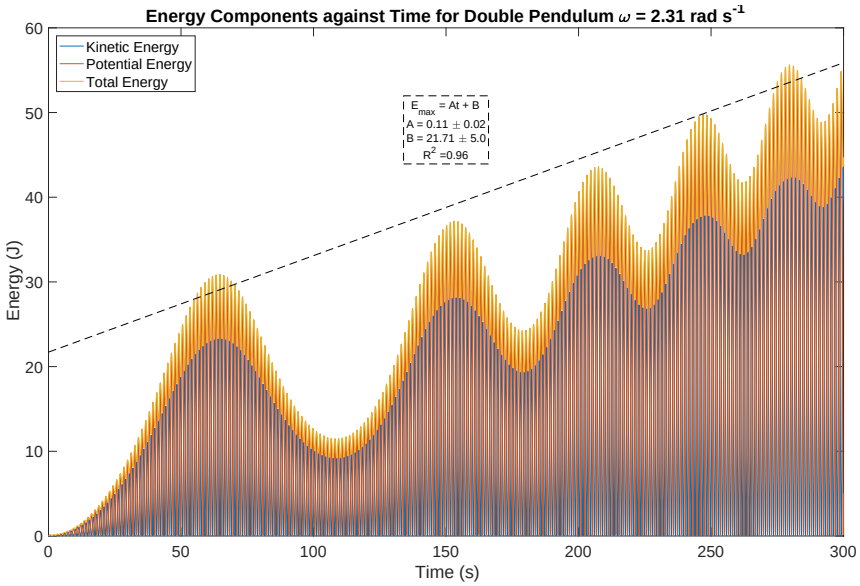


Figure 2.13: Energy Components for Double Pendulum at Resonance $\omega = 2.31\ rad\ s^{-1}$

To compare the growth in energy of the system at different frequencies a similar fitting process was applied to the data in figure 2.12. The result of fitting to the peaks is shown in figure 2.14. As expected the double pendulum not only has less energy at $\omega = 2.2\ rad\ s^{-1}$ but it also gains energy more slowly as it is not the resonant frequency. Comparing the plots at $\omega = 2.2\ rad\ s^{-1}$ and $\omega = 2.31\ rad\ s^{-1}$ the rough energy scale after 300s is approximately $\frac{50}{4} = 12.5$ times larger at $\omega = 2.31\ rad\ s^{-1}$. This is roughly equivalent to the ratio of the rate at which the system gains energy at $\omega = 2.31\ rad\ s^{-1}$ and $\omega = 2.2\ rad\ s^{-1}$ i.e $\frac{0.11}{0.009} \approx 12.2$. This seems to indicate that the energy scale of the system determines the rate of increase of energy of the system. More simply put a system with twice the energy scale of another will gain energy roughly twice as fast when driven. The exact nature of this relationship could be investigated by future groups.

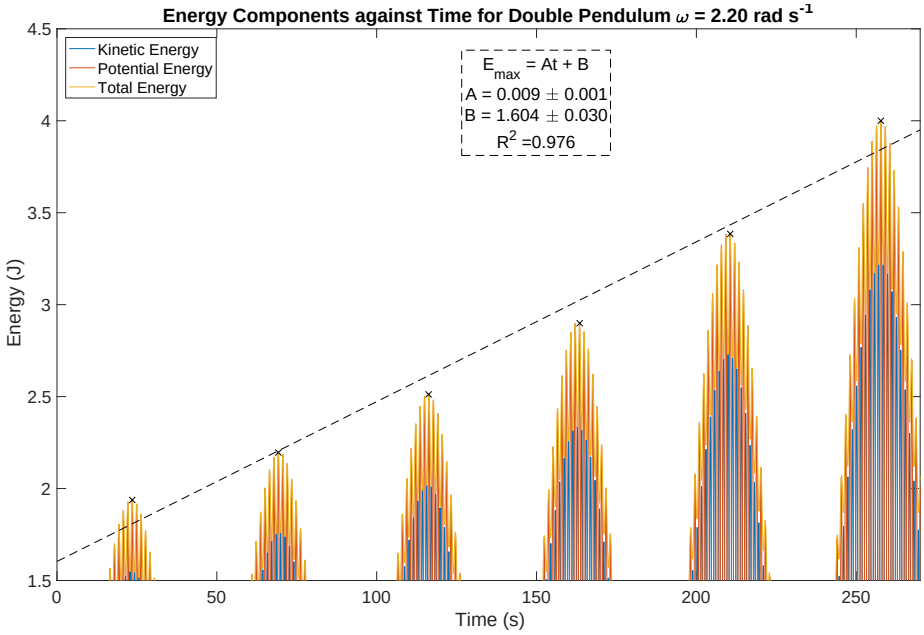


Figure 2.14: Linear Fit to Energy Peaks at $\omega = 2.20\ rad\ s^{-1}$

2.7.4 Investigating Phase Relationships

Max Elliott

The first feature of swinging motion to be investigated using the single flail model was the phase relationship between the driving motion of the legs/torso and the swing pivot angle. The masses were set to $m_\alpha = 0$, $m_1 = 2$, $m_2 = 2$ and the lengths to $l_\alpha = 1.79$, $l_1 = 0.2$, $l_2 = 0.2$. The driving functions applied to θ_1 and θ_2 were set equal but with a difference of π. Doing this produced a dumbbell pendulum model.

The swing angle α was initially set to -45°, which meant that if the model was left to oscillate freely, α would vary as

$$\alpha(t) = -\frac{\pi}{4}\cos(\omega_0 t) \tag{2.53}$$

where ω_0 is the natural frequency of the model. The leg and torso angles were driven by functions

$$\theta_1(t) = -\frac{\pi}{12}\cos(\omega_0 t + \delta) \tag{2.54}$$

and

$$\theta_2(t) = \pi - \frac{\pi}{12}\cos(\omega_0 t + \delta) \tag{2.55}$$

where δ is a constant phase term. Using this term, the phase difference between the driving functions and the swing angle can be varied, and the growth of α over time can be measured for each phase difference.

ω_0 was found using the same process as in section 2.7.2. The model was then ran for 20 seconds, starting at $\alpha_0 = -45°$, using a range of phase term values. For each run, the amplitude of each peak in the swing cycle was extracted to investigate the growth in peak amplitude over time. This growth for every run can be seen in figure 2.15, given in terms of fractional growth from the original 45°amplitude. As seen in the plot, the phase that gives the optimum amplitude growth is when the driving function is $\frac{\pi}{2}$ ahead of the main pivot motion. There is positive growth for phases up to $\delta = 0$ and after that there is an overall decline in peak amplitude, with the steepest reduction occurring when the driving function lags behind α by $\frac{\pi}{2}$.

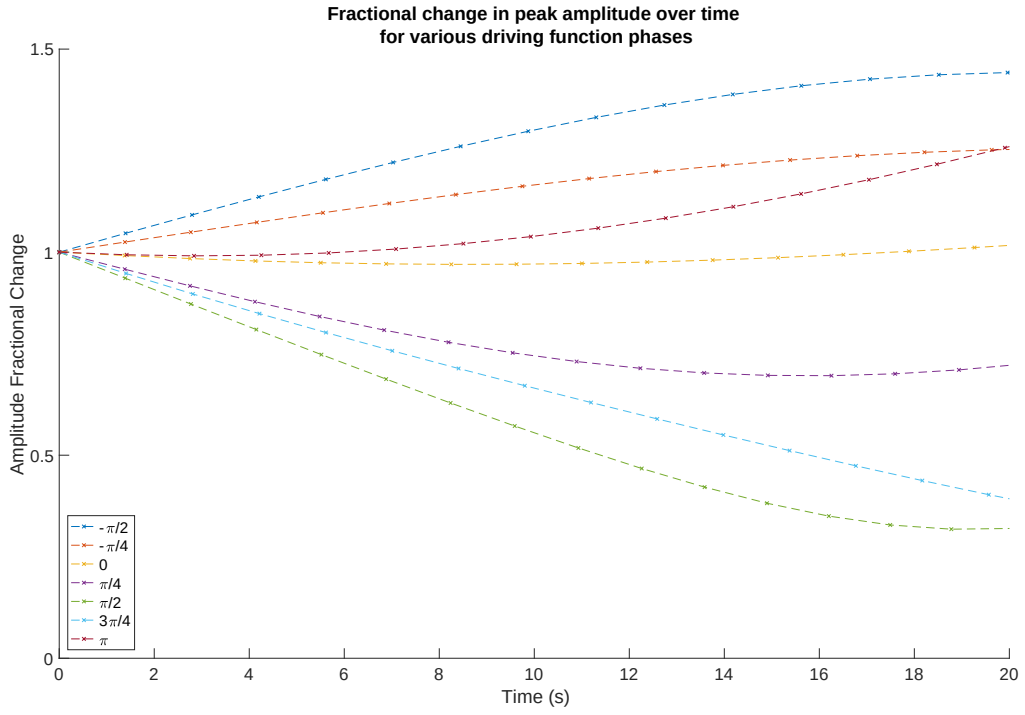
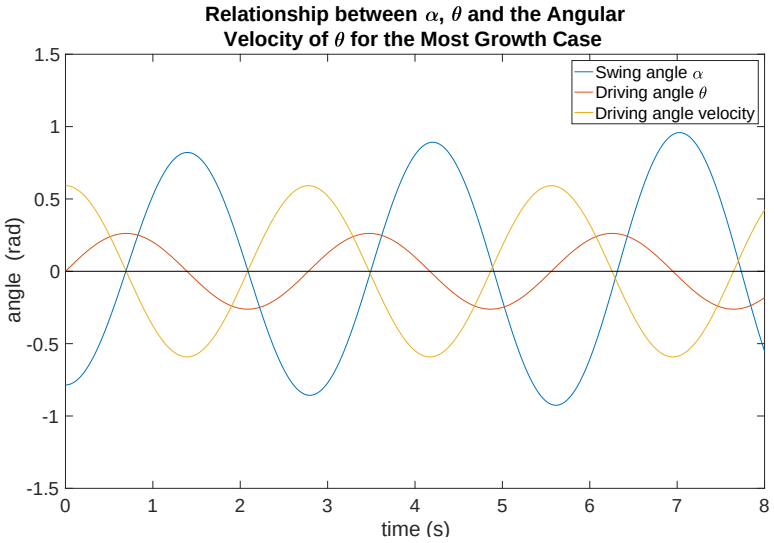
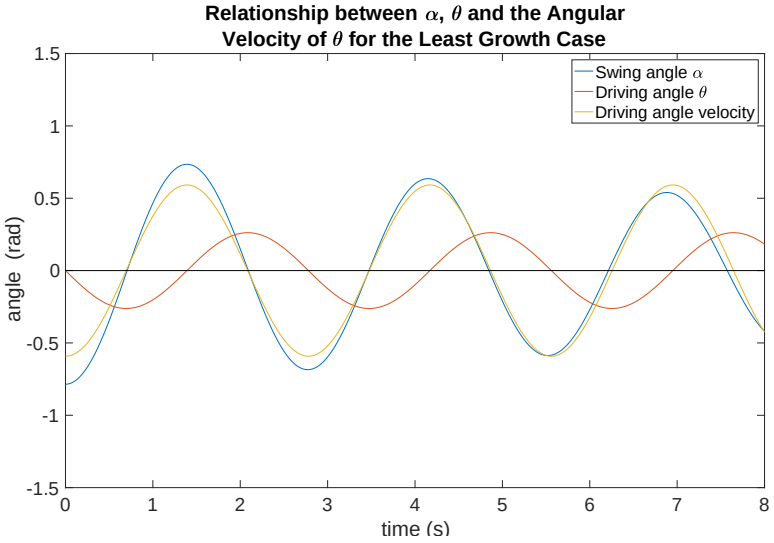


Figure 2.15: Amplitude growth over 20 seconds for different phases between swing angle and driving function

Looking at sections of the data taken from the highest and lowest growth runs can help explain the relationship between phase and amplitude growth. In figure 2.16 plots for α, driving function angle, and driving function angular velocity over roughly two periods of swinging can be seen. In the plot for when the driving function is $\frac{\pi}{2}$ ahead of the swing angle, the angular velocity of the driving function is always negative when the swing angle is increasing, and positive when the swing angle is decreasing. A positive angular velocity about the swing seat due to leg and torso movements causes a reactionary and negative angular velocity about the main swing pivot due to conservation of angular momentum, and vice versa. When the driving function is $\frac{\pi}{2}$ ahead of the swing, the angular velocity induced about the main swing pivot is always in the direction the swing is moving already, and therefore the peak amplitude of the swing angle increases over time. Conversely, when the driving function is $\frac{\pi}{2}$ behind the main swing angle, the opposite effect occurs, meaning the driving motion opposes the swings motion, decreasing its overall energy and hence its peak amplitude. It can be concluded from this analysis that for optimal swinging the robot should move its limbs periodically with a phase that is $\frac{\pi}{2}$ ahead of the main swing angle.



(a) Driving motion leading α by $\frac{\pi}{2}$



(b) Driving motion lagging α by $\frac{\pi}{2}$

Figure 2.16: Relationships between swing angle, driving function angle, and driving function velocity for the maximum and minimum growth cases

2.7.5 Investigating Driving Function Amplitude

Max Elliott

An investigation into how the amplitude of the driving function affects the swing motion was done using the single flail model. The model was set up as a dumbbell pendulum as in section 2.7.4, and the swing angle α initially set to zero. the leg and torso masses were driven sinusoidally as

$$\theta(t) = C + A\sin(\omega t) \tag{2.56}$$

where $C = 0$ for the legs and $C = \pi$ for the torso, to offset them. A is the amplitude of the driving function. This driving amplitude was varied between 0 and 1.2 over multiple 100 second runs, and the maximum swing amplitude achieved by each run was plotted in figure 2.17. This process was repeated for three different driving frequencies.

The plots show a clear increase in maximum swing amplitude as the driving amplitude increases, with the greatest increase occurring when the angular frequency is close to the natural frequency of the model. The relationship between the two amplitudes can be explained simply as higher driving amplitudes adding more energy into the system and hence giving the swing more energy to reach higher amplitudes itself. Because the motion is only beneficial when done at the correct phase with respect to the swing, the maximum swing amplitude attained for the same driving amplitude will vary depending on the driving frequency. When the driving frequency is off-resonance the driving motion will eventually oppose the swing motion, resulting in a reduction of the total energy of the system and a lower maximum amplitude.

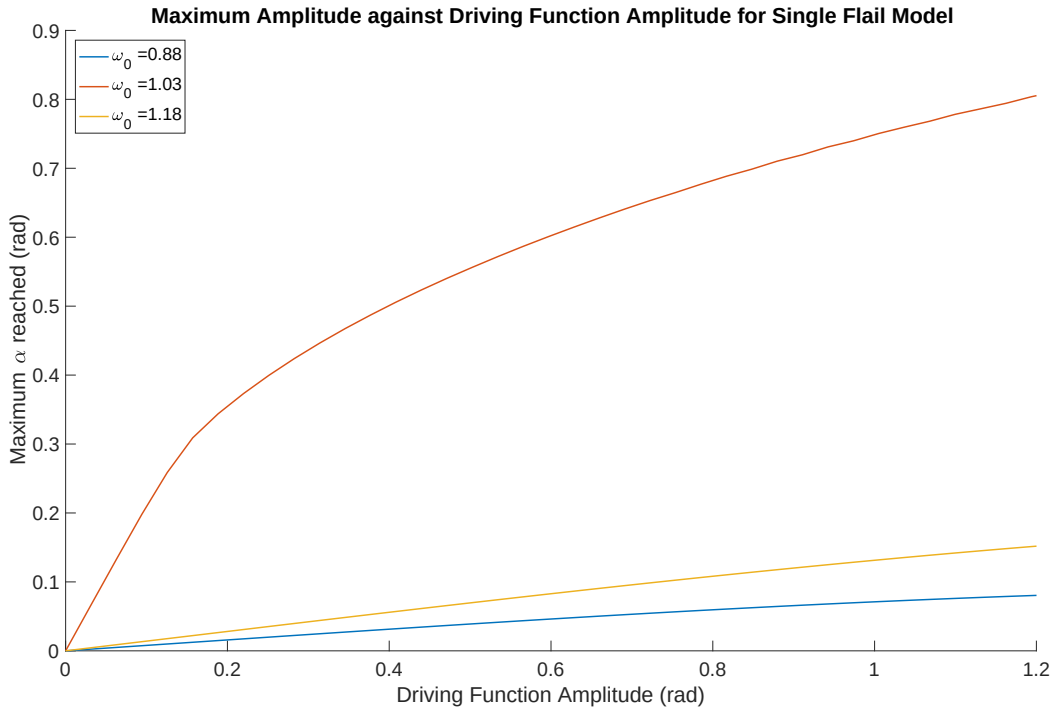


Figure 2.17: Maximum Amplitude reached over 100s against Driving Function Amplitude for Single Flail

2.7.6 Creating a Ramp Driving Function

Max Elliott

In the previous sections of this report, the driving functions used to move the leg and torso masses have been sinusoidal functions. However, to increase the relevance of the modelling to the real robot, and to investigate alternate driving functions, a new function was designed. Instead of a constantly accelerating motion, like the sinusoidal motion, the robot's movement is generally done at a constant velocity, with an initial acceleration and final deceleration when moving between positions.

Looking at different functions, a ramp function would produce this sort of behaviour, except that the function became discontinuous when the gradient changed. This is undesirable as the models' equations of motion require the first and second derivatives of the driving function, and is also unrealistic to the robot as it cannot apply infinite acceleration. Therefore a rounded ramp approximation[11] was used, which is described by

$$f(x) = m\frac{\log(1+e^{ax})}{a}. \tag{2.57}$$

Using this, a periodic ramp function could be created piecewise by using four of these approximation functions. This produced a fully differentiable function that mimicked the robots motion well. It had two parameters: m to define the gradient (or angular velocity) of the movement, and a to define the steepness of change (or angular acceleration) of the function. Figure 2.18 shows examples of different ramp driving functions with different values for m and a. The period and amplitude of the function can also easily be changed in the C++ program.

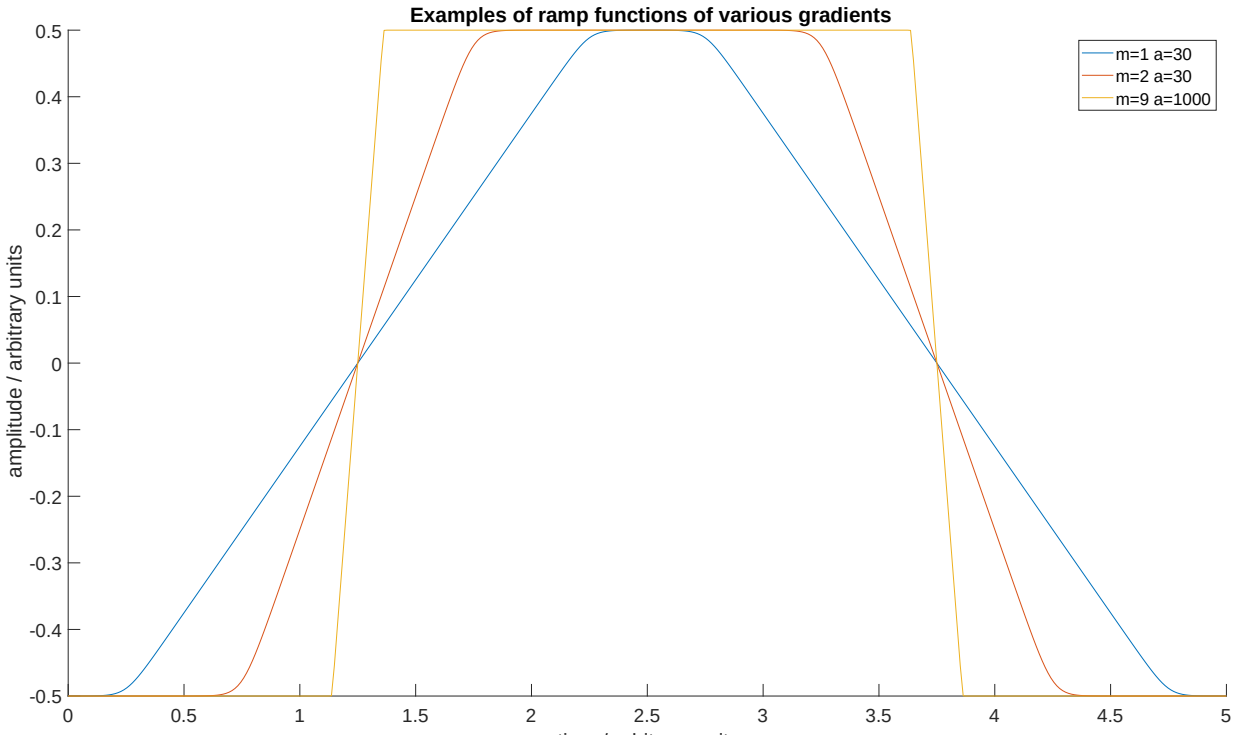


Figure 2.18: Different ramp functions of an arbitrary amplitude and period to show how m and a can vary the function

2.7.7 Investigating the Periodic Ramp Function

Max Elliott

With the single flail model set up as before, it was driven with the periodic ramp function, with amplitude $\frac{\pi}{12}$ and angular frequency $\omega = 2.26$, the natural frequency of the model. The model was simulated for 60 seconds multiple times using ramp functions of various gradients. The peak amplitude growth over time was found for each and plotted in figure 2.19.

It can be seen clearly that the ramp functions perform better than sinusoidal motion, reaching a higher maximum amplitude and also growing quicker. Increasing the steepness of the ramp function generally improves the growth, but begins to generate diminishing returns as the function approaches a step-like function (see figure 2.21).

Max Elliott

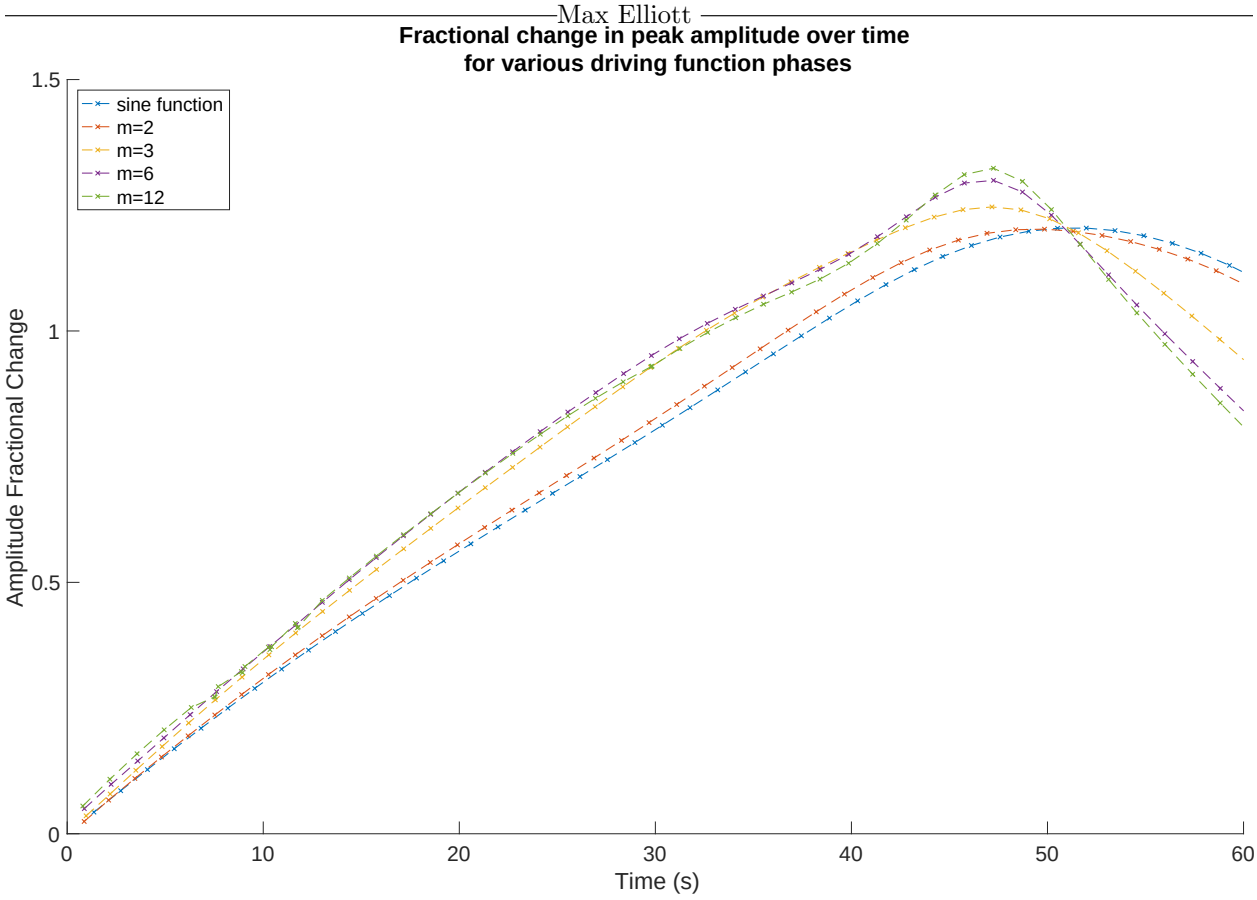


Figure 2.19: Swing peak amplitude growth over 60 seconds for different steepness ramp driving functions. Data from a sinusoidal driving function of equal amplitude is also plotted for comparison.

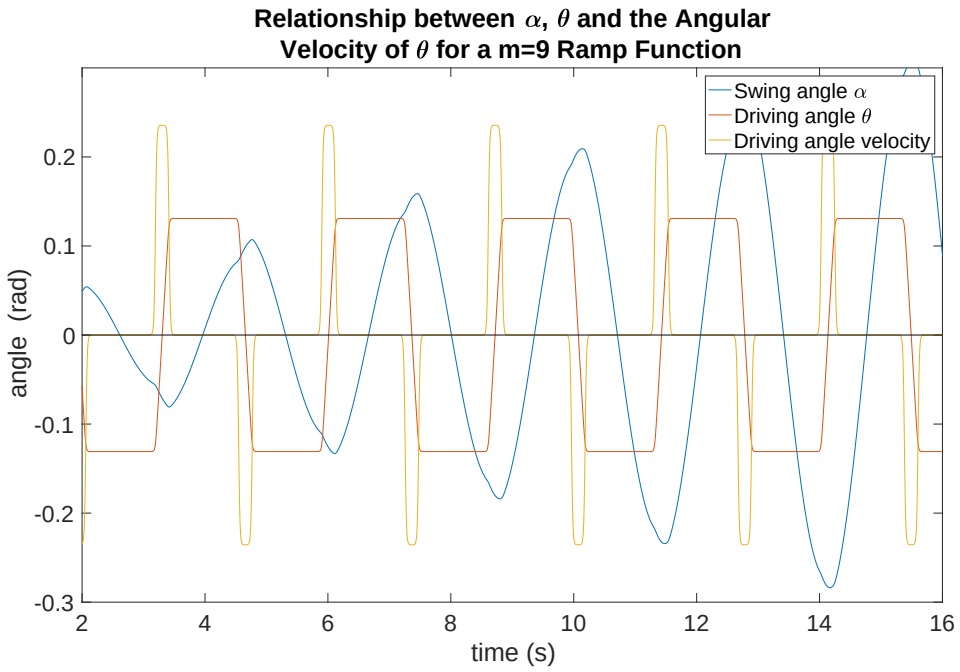


Figure 2.20: Relationship between the swing angle, the driving function angle, and the driving function angular velocity for a periodic ramp function during optimal swinging motion. The angular velocity has been scaled down by a factor of 5 for added clarity.

Figure 2.20 shows the relationship between the swing angle and the driving function during optimal motion. The phase relationship is the same as with the sinusoidal driving function, but now all the motion occurs at the peak amplitudes of the swing angle. Therefore, the driving function velocity is now applied as sharp impulses that start just before and finish just after each peak. It is this concentration of driving movement at the swing extrema that produces quicker amplitude growth when compared to the sinusoidal motion. It was shown in section 2.3.3 that a change in angle about the swing seat will produce an opposing change in angle about the main pivot. Therefore by rapidly changing the driving angle at the peak in a way that causes the swing angle to move further up, the energy in creating this movement has been converted in potential energy by increasing the height of the centre of mass of the swing. This increase in energy is maximised when the driving movement occurs at the swing maxima, as this will cause the most vertical change in the swing; doing the movement at a lower point in the swing cycle will reduce the amount of potential energy added, resulting in less efficient movement and slower amplitude growth. It is interesting to note that the effect of the driving motion can be observed as a slight bump at each swing angle maxima in figure 2.20. It is clear that the driving function movement is causing the swing angle to increase at these points, adding potential energy to the system. Consequently, it can be concluded from this analysis that for optimum amplitude growth the robot should perform its pumping motions as rapidly as possible at the maxima of the swing amplitude.

2.7.8 Idealised Step Function

Benjamin Adams

The ideal way of driving the motion of the swing is with a step function. To consider what driving the motion with this would be like the maximum swing amplitudes for different gradient ramp functions (m) were plotted. The data appeared to be tending to a constant value for increasing m as so it was fitted to a function of the form $\alpha_{max} = A(1 - e^{Bm}) + C$, this proved to be a suitable fit based on the value obtained for the goodness of fit parameter and is plotted alongside the data in figure 2.21. Aside from the offset C the form of the fitted function is the same as how charge varies with time when charging a capacitor. This highlights how one obtains diminished amplitude gains on increasing how step-like the driving function is. Extrapolating the data by taking the limit m tends to infinity corresponds to considering driving the system with a step function. Given the obtained fitting parameters, the analysis suggests that if the simulation had been carried out with a step function then the maximum amplitude obtained would be approximately A+C = 1.32 rad = 76°. Note that the robot cannot move its limbs instantaneously so in some sense this analysis is not completely applicable to the robot. However, it does provide a rough upper limit on the maximum amplitude attainable.

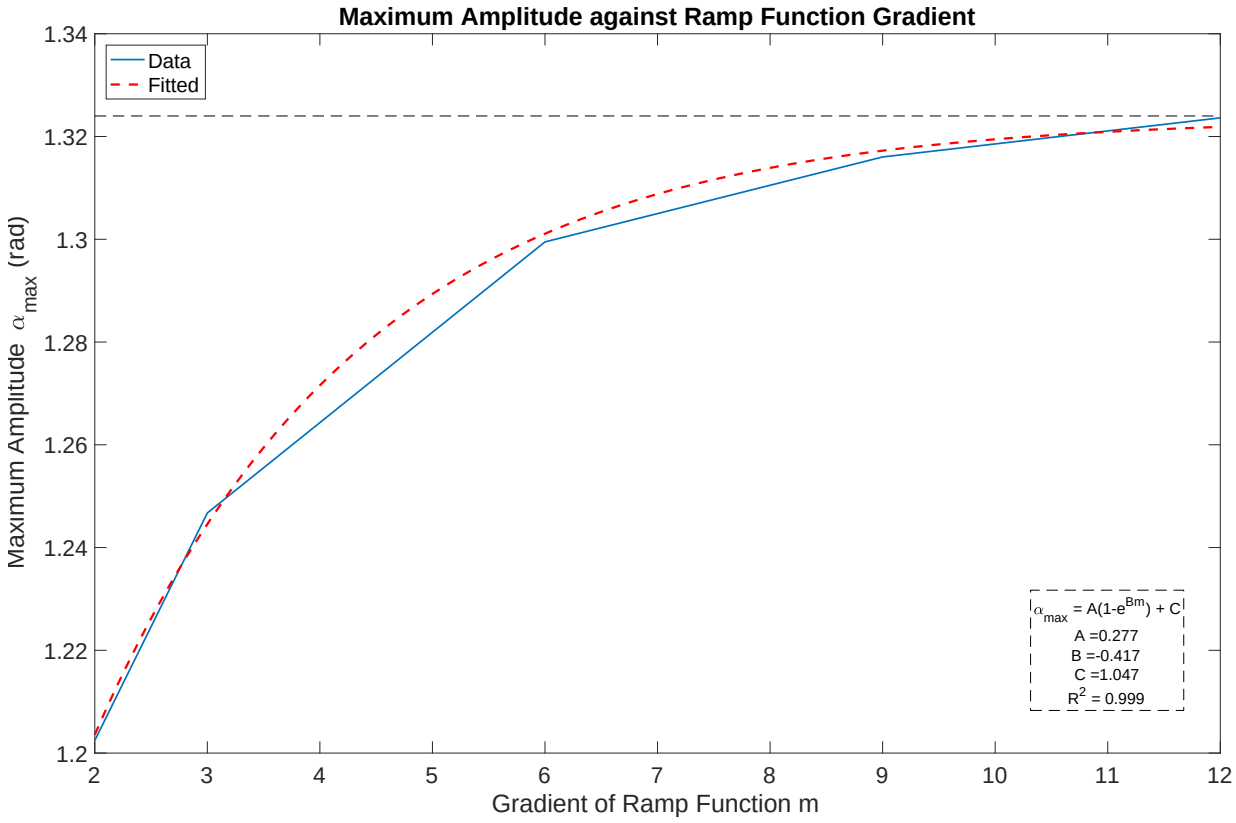


Figure 2.21: Maximum Amplitude reached over 100s against Ramp Function Gradient for Single Flail

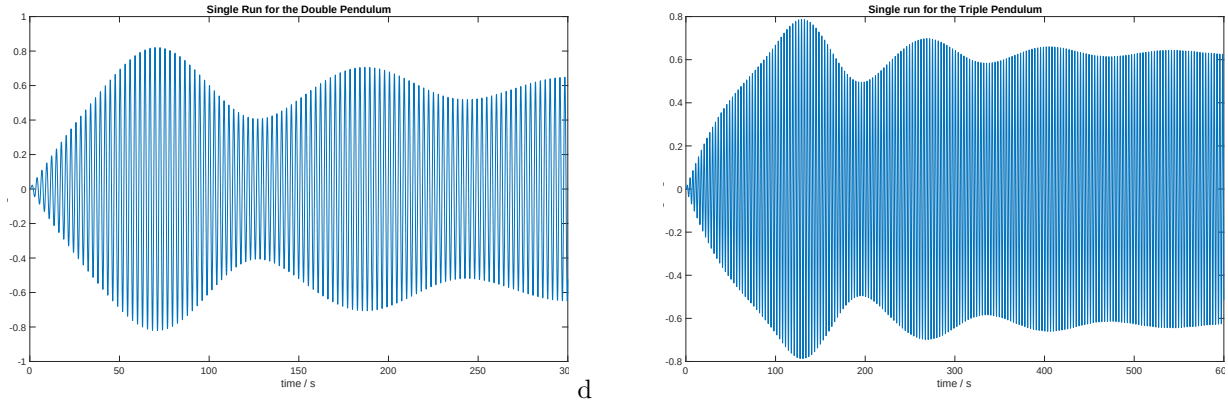
2.7.9 Investigating the Effect of a Hinge

Max Elliott

As the aim of this project was to improve the robot's swinging motion by incorporating the hinge on the swing, the hinged flail model was developed. The initial aim was to compare simulations of the single flail with the hinged flail to see how the added hinge affected to overall swinging motion. However, due to complications in the simulation program, the hinged flail model could not be run. Instead, the same comparison was carried out between the the double pendulum model and triple pendulum model, as these could both be successfully ran in the program. While neither have the torso mass, the conclusions drawn from the comparison will still give some relevant insights into the effects of the hinge on the swing's motion.

Both the double pendulum and triple pendulum were set up so that they were the same model except the triple pendulum had a hinge: for the double pendulum the masses/lengths used were $m_\alpha = m_1 = 2, l_\alpha = 1.79, l_1 = 0.2$ and for triple pendulum they were $m_\alpha = 0, m_1 = m_2 = 2, l_\alpha = 1.565, l_1 = 0.225, l_2 = 0.2$. Both were driven with a sinusoidal driving function of amplitude $\frac{\pi}{12}$ driven at their respective optimal driving frequency, found using the same process as in section 2.7.2. The double pendulum had an optimal driving frequency of 2.31, whereas the triple pendulum had a lower frequency of 2.01.

The swing angle over time for a simulation of both models is shown in figure 2.22. The double pendulum actually reaches a higher maximum amplitude quicker, reaching 0.8211 rad in 69.8s compared to 0.7876 rad in 129.0 s for the triple pendulum. This is contradictory to the expected result that creating a hinge allows you to move your centre of mass higher at the swing peaks, adding more energy into the system and hence reaching a higher amplitude.



(a) Single run for the double pendulum

(b) Single run for the triple pendulum

Figure 2.22: The main swing angle over time for the double pendulum and triple pendulum. Both models have the exact same masses, lengths, and driving functions, the only difference being the hinge on the triple pendulum.

Looking at the hinge angle itself over time could explain this discrepancy between theory and results. Figure 2.23 shows the hinge angle over a few periods of movement, with the swing angle included for reference. It clearly shows a much faster oscillation, implying that some higher order mode of the model is being excited. This higher mode will be of a higher frequency and produce overall a smaller swing amplitude. To further analyse these modes, a Fourier transform of the hinge angle data was performed in MATLAB, and plotted in figure 2.24. The frequency components have been put in units of angular frequency. There are four distinct peaks, the two biggest at 2 rads^{-1} and just below 10 rads^{-1}. The normal modes of this specific triple pendulum model were calculated using the same technique as in section 2.7.1, giving values of 2.275 rads^{-1} and 10.191 rads^{-1}. This similarity in frequencies confirms that some higher order excitation is occurring. It becomes clear from the Fourier transform that some of the spectral power of the system is being wasted in this higher frequency mode. To improve the function of the hinge, work must be done to stabilise the hinge

movement to minimise the amount of energy being put into the higher mode. Unfortunately, due to time constraints, methods for achieving this could not be investigated, and is therefore proposed to subsequent groups as a further extension on this report.

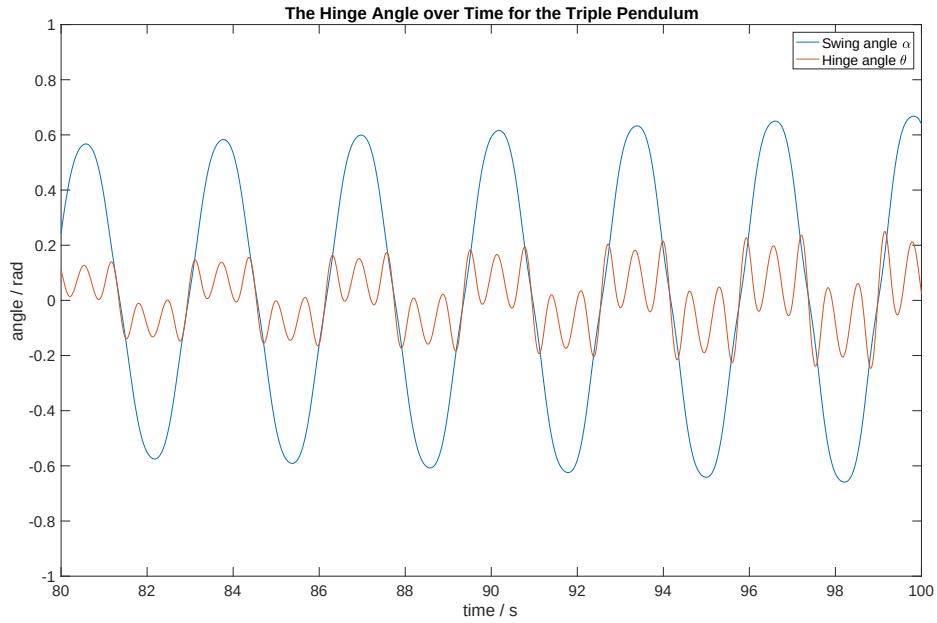


Figure 2.23: The hinge angle over a few periods of the triple pendulum's motion, showing more rapidly oscillating motion than the main swing angle

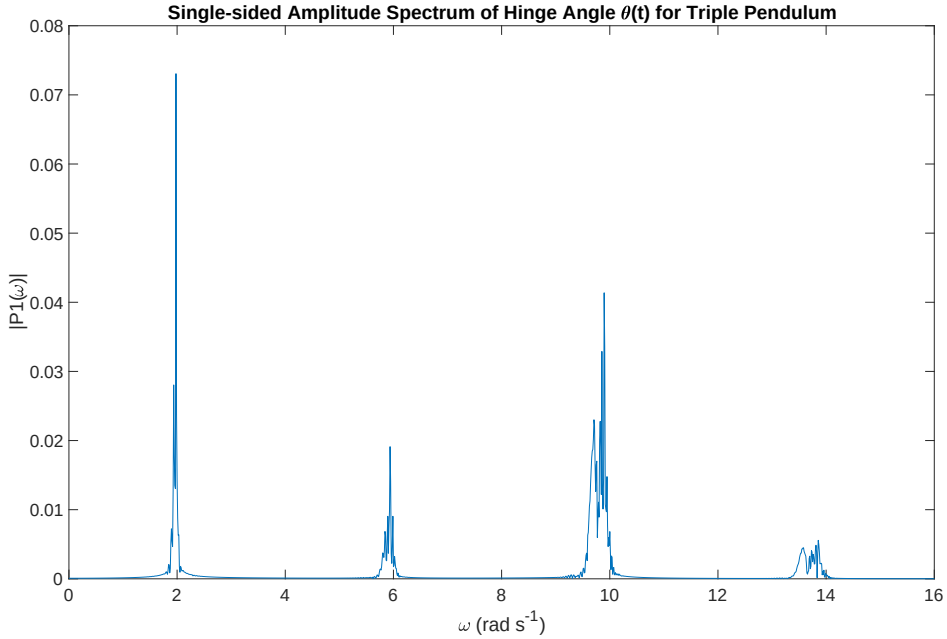


Figure 2.24: The Fourier Transform of the Hinge Angle Data. Four distinct peaks are visible, at roughly 2 rads^{-1}, 6 rads^{-1}, 10 rads^{-1}, and 14 rads^{-1}.

2.8 Realistic Modelling

2.8.1 Robot and Swing Properties

Josh Goldberg

In order for the pendulum models to be applicable to the robot, they were required to have accurate values for their masses, lengths, angle ranges and maximum angular velocities and accelerations. As mentioned earlier, the masses in the pendulum models represent different parts of the robot/swing. For example, in the double pendulum model in figure 2.3, m_α represents the mass of the swing and the robot's head, body and upper legs, while m_1 represents the lower legs and feet, thus modelling the robot only swinging its legs. As the lower legs and feet move together, this can be modelled as one mass with a distance l_1 from the knee to its centre of mass. Using the NAO robot specifications[12][13] , the mass and centre of mass position of the lower legs, ankles and feet were found and used to calculate m_1 and l_1. The same was also done for the torso, arms, neck and head to find m_1 and l_1 in figure 2.4.

Body Part	Mass (kg)	Distance from centre of mass to Knee (cm)
Tibiae	0.602	4.94
Ankles	0.268	9.60
Feet	0.344	13.53

Table 2.1: The properties of the robot below the knee

Body Part	Mass (kg)	Distance from centre of mass to Hip (cm)
Head	0.605	26.41
Neck	0.078	18.41
Arms	1.157	18.7
Torso	1.050	12.84
Pelvis	0.140	0.027
Hips	0.281	-0.52

Table 2.2: The properties of the robot above the hip

Using the values from tables 2.1 and 2.2, the distance to the centre of mass of both the lower legs and the upper body can be calculated using

$$COM = \frac{\sum_i m_i d_i}{\sum_i m_i} \quad , \tag{2.58}$$

where m_i and d_i are the masses and distances respectively for each component of the body. Finally, the mass of the upper legs is 0.603 kg, while the mass of the seat is 1.065 kg.[6] Using equation 2.58, the masses and lengths were calculated for the double pendulum model in figure 2.3 and the single flail model in figure 2.4, as shown in tables 2.3 and 2.4.

m_α	m_1
5.16 kg	1.21 kg
l_α	l_1
179 cm	8.40 cm

Table 2.3: Parameters for the Double Pendulum Model in figure 2.3

m_α	m_1	m_2
1.84 kg	1.21 kg	3.31
l_α	l_1	l_2
179 cm	8.90 cm	16.51 cm

Table 2.4: Parameters for the Single Flail model in figure 2.4

When using the hinged single flail in figure 2.6, the masses and lengths of the robot are the same, but the lengths of the swing to the hinge and the hinge to the seat are 156.5 cm and 22.5 cm respectively.

Following from this, the angles that both the lower legs and the upper body were able to move between were required. Table 2.5 shows the robot's available angles on the swing, whether they're limited by the robots joints or the swing itself. When driving the pendulum masses, the angles were limited to these values to accurately replicate the robot.

Joint	Minimum angle (°)	Maximum angle (°)
Knee	-5.9	79.8
Hip	23.0	60.6

Table 2.5: The range of angles available for the knee and hip joints[14]

Finally, the maximum angular velocities and accelerations of the joints were needed, so that the pendulum masses moved between these angles in the same way that the robot would. From the NAO specifications,[15] the motors in both the knee and hip have a 'no load speed'of 8300±830 rpm (rotations per minute), with speed reduction ratios of 130.85 and 201.3 respectively. The 'no load speed' of a motor is the speed the motor's shaft will rotate at when no load is attached to it, whereas the speed reduction ratio represents what ratio this speed is reduced by due to the load attached, giving the rated speed. The angular acceleration can be calculated from the torque of the motors. The resultant torque is given by the nominal torque, multiplied by the speed reduction ratio.[9] The maximum angular acceleration is then given by

$$\alpha = \frac{\tau}{mr^2}, \tag{2.59}$$

where α is the angular acceleration, τ is the resultant torque, m is the mass of the rotating component and r is the distance between the hinge point and the centre of mass. Table 2.6 shows the calculated maximum angular velocity and acceleration of both the lower legs and upper body.

Lower Legs		Upper Body	
Angular Velocity (rad s^{-1})	Angular Acceleration (rad s^{-2})	Angular Velocity (rad s^{-1})	Angular Acceleration (rad s^{-2})
6.6	335	4.3	22

Table 2.6: The range of angles available for the hip and knee joints

By using the parameters calculated in this section on the pendulum models, their motion will be representing the robot as accurately as possible. The models will however have some inaccuracies with the robot. Firstly, in the models the hinge point between the seat/upper legs and the lower legs and upper body is at the same point as its centre of mass. However, in reality, the hinge points are at the knees and hips, whereas the centre of mass will be somewhere in between these two points. Furthermore, in the case of the real robot, the angular velocity and acceleration values will be lower than those stated. This is because over time, the efficiency of the motors in the robot decreases, thus reducing the maximum torque they can produce. For example, last year's report[9] showed that the torque at each joint on the robot was approximately half of that stated in the specifications. Therefore, it would never be possible for the pendulum models to fully replicate the robot, but by being as accurate as possible, they should give valuable information on the optimal swinging motion required.

2.8.2 Simulating a Realistic Model

Max Elliott

To obtain a rough idea of how well the robot would perform on the swing, the single flail model was simulated with the parameters shown in table 2.4, with the legs and torso moving between the angles shown in table 2.5. The driving functions were chosen to perform optimally within the robot's limits, meaning the limbs were driven by a ramp function of largest feasible amplitude. m and a in the ramp function were set such that the angular velocity and acceleration of the driving functions did not exceed the values given in table 2.6, to produce motion within the limitations of the robot.

The optimal driving frequency was calculated and then used to drive the motion for 600s. Figure 2.25 shows the swing angle over time for the run. The swing reaches a maximum amplitude of 34.2°after 81.8s. While larger than any value reached by the real robot, this simulation represents an ideal model of the real system, and would therefore be expected to perform better. The fact that this value is higher but not excessively higher than what the robot can actually achieve shows that the models do provide an ideal but realistic simulation of real-life.

The graph shows a clear beating frequency in the peak amplitude of the swing over time. This is due to the driving motion frequency not being exactly the natural frequency of the swing, and therefore the motion eventually goes out of phase and becomes destructive. This is a clear indicator that a predetermined motion is a poor choice for driving the real robot, and instead it should be driven by a more dynamic algorithm.

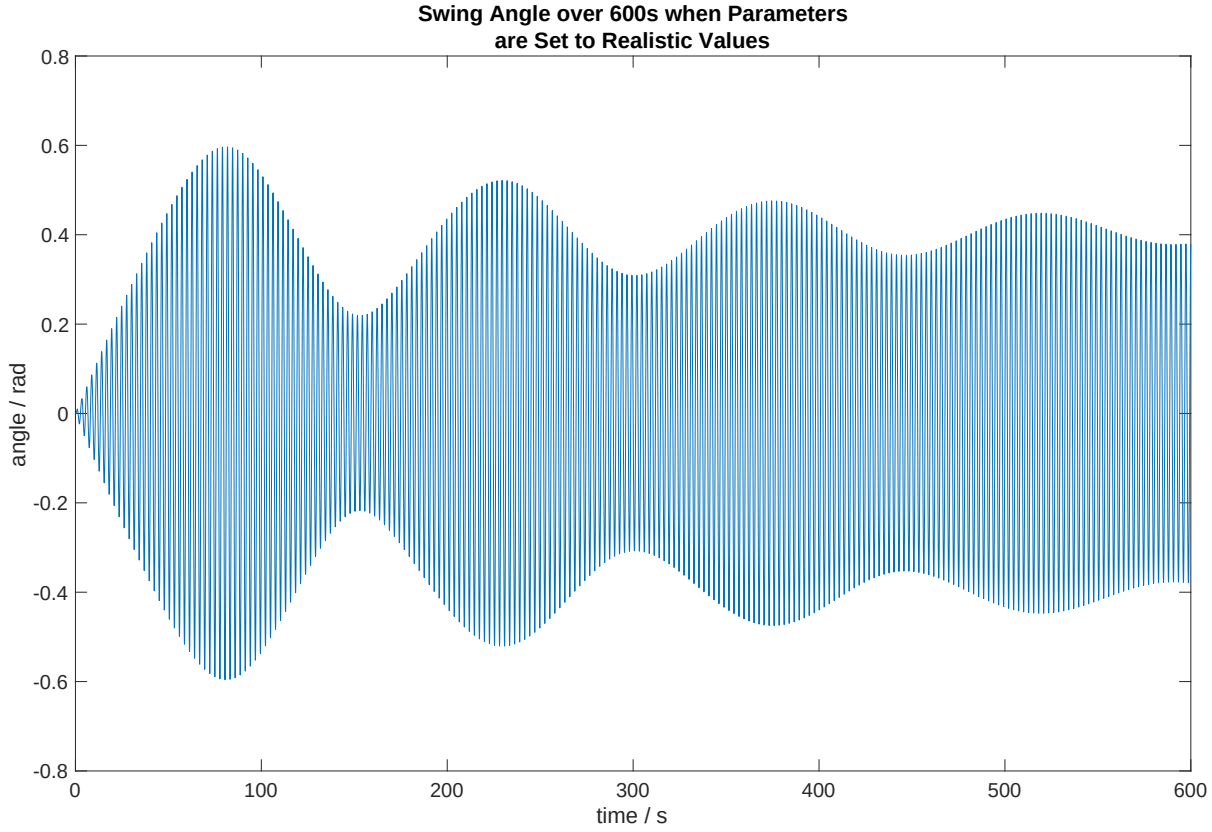


Figure 2.25: Single Flail Simulation ran over 600s with masses, lengths, and movements realistic to the NAO robot

2.9 Conclusions

The numerical modelling section aimed to investigate the ideal swinging motion for a robot on a swing. Using models designed to best describe the robot riding a swing, simulations were generated and data was taken. From section 2.7.2, it was discovered the ideal frequency to drive the system was at its natural resonant frequency. Over a range of frequencies, the natural frequency of the swing will provide the maximum amplitudes, while driving the swing at higher or lower frequencies will result in smaller amplitudes. After investigating the most appropriate method for driving the robot, it was concluded that the *ideal* driving function would be a step function. However, the robot was not an ideal case and so, with this under consideration, a new function was developed. The outcome was a near-step function called a rounded ramp function. The larger the rate of change in amplitude, the larger the torque on the robot and hence, the closer the rounded ramp function will be to the idealised case of the step function. When driving the swing, the driving function requires an amplitude to induce an amplitude on the whole system. The investigation led to a positive relationship between attaining a maximum swing angle and the amplitude of the driving function. This is due to the momentum change being bigger with larger driving amplitudes. An investigation into the optimum driving phase was undertaken and it was discovered that the quintessential phase for applying the driving motion was $\frac{\pi}{2}$ ahead of the swing. The reason being, this would occur when the velocity was in the same direction; hence, the amplitude was able to increase over time. Finally, the effect of adding a hinge to a model was investigated, by running a double pendulum and triple pendulum model with the same parameters, the hinge being the only difference between the two. The hinge actually caused the triple pendulum to perform worse than the double pendulum, due to the unwanted excitation of higher frequency normal modes. This was analysed and confirmed using Fourier transforms, but no solution was proposed. This could be a good starting point of a further investigation into minimising these modes in subsequent years.

Throughout the investigation, the swing rods were considered to be light in all of the models. However, in reality, this is not the case. Perhaps, in future investigations, the mass of l_α and l_0 could be included into the models and used within the simulations. Similarly, the system is in 3-dimensions and as a result of this, if there are any anti-symmetric properties within the system; such as the left side of the robot being heavier than the right or differences in lengths, then this will affect the periodic motion of the left and right hand side. This could cause torsional forces around the z-axis, and hence, could cause energy loss from the swing.

2.10 Extensions

2.10.1 Complex Hinged Flail

Christopher Hulme

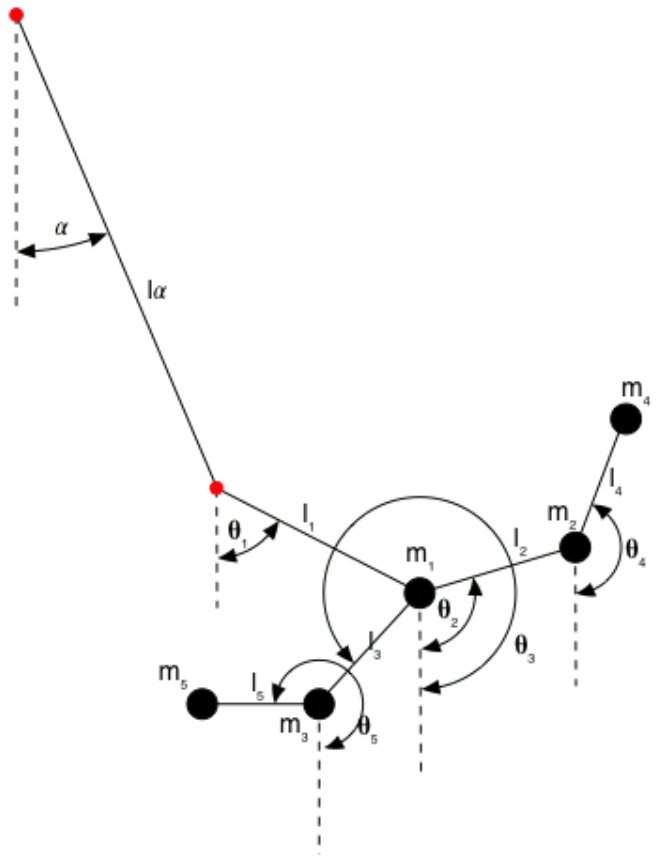


Figure 2.26: Complex Hinged Flail Model

To extend the hinged flail model further more masses were added to better represent to mass distribution of the robot. The complex hinged flail model shown in figure 2.26 takes the hinged flail and incorporates head and feet into the model. It was decided this was important, given the head weighed ∼60% of the robot's torso and both feet together weighed ∼25% of the upper legs' and ankles' masses combined, which are a good fraction of the robot's masses.

To achieve this it was necessary to compute the new Lagrangian of the system (see Appendix A.1). The masses m_1, m_2 and m_3 and their respective angles and lengths have similar representation here to in the hinged single flail model. However, m_4 and m_5 are added to the model to represent the head and feet of the robot respectively. Naturally, l_4 and l_5 represent the lengths of these parts of the robot and the angles θ_4 and θ_5 correspond to their respective angles relative to the vertical axis.

Unfortunately, no further investigation into this model was undertaken. This was due to problems with the fourth order Runge Kutta algorithm accepting the equations of motion for this model. If a working algorithm could be found, this model may allow for an interesting investigation to see if adding the motion of the feet and head make a difference.

2.10.2 Adding the Arms

Christopher Hulme

An attempt was made to try and develop a more realistic model which would resemble the robot more accurately. To make the model more realistic, the hinged flail model (see section (2.4.4)) was taken and the robot's arms were incorporated. This can be seen in figure 2.27 where the torso (m_1) connects from the seat at P_3 to the shoulder at P_4, the upper arm from the shoulder to the elbow at P_5 and the lower arm extends from the elbow to the hinge at P_2. The robot's hands and wrists are limited in movement to rotating about the axis of the lower arm. Due to this limitation, the hands attach to a bracket which extends between the two swing rods, the bracket's position along the swing rod is between two hinges.

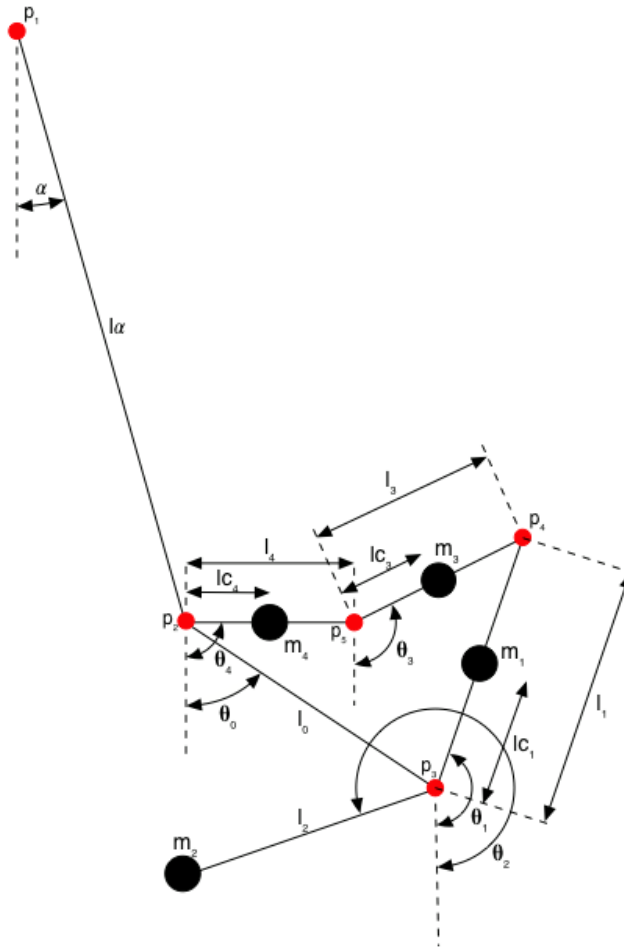


Figure 2.27: Advanced model demonstrating robot holding swing with arms

Figure 2.28 demonstrates the hinge in figure 2.5, figure 2.6 and P_2 from figure 2.27 in more detail for the robot's actual swing. It can be see how having a short rod between two hinges for the lower arm to connect to can be assumed to be equivalent to a single pivot which the lower arms, upper portion of the swing rod and lower portion of the swing rod all connect to. This will reduce the number of degrees of freedom simplifying the mathematics a large amount. It should also be noted that having two hinges with a short bar between for the lower arm to connect to is in fact a more realistic model to human motion. When a person grasps the rope they are in fact holding a small cylindrical segment of rope which bends at the top and at the bottom of their hands.

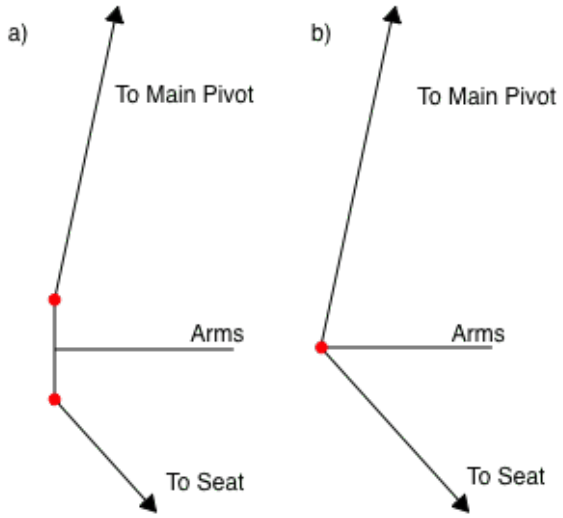


Figure 2.28: a) represents arms holding a rod between the two hinges and can become equivalent to b) where arms are holding one hinge at the hinge. This can be assumed when the length of the rod between the two hinges is small in comparison to the length of the rod to main pivot, and that of the length to the seat.

Although this model is theoretically aimed to be more accurate to the real robot, verification of this was not obtained. An attempt at formulating the Lagrangian was made; however, the system was too complicated for it too be derived. This was due to the loop within the pendulum, which is very clear in figure 2.27. This loop by itself is known as a 4-bar loop and to derive its Lagrangian requires a different approach (see Appendix A.2). Despite calculating the Lagrangian for the 4-bar model (equation A.30), it was never able to be linked to one of our pendulum's Lagrangian in order to find the Lagrangian of the whole model. The issue here was to do with the location of the 4-bar system. The origins of the pendulum and the 4-bar were not located in the same place. In order to align there origins one would need to add the length l_α to the 4-bar, but doing this would mean the Lagrangian of the 4-bar would change and the method for calculating the 4-bar Lagrangian will not work having the length l_α incorporated. Therefore, continuation of deriving the system's Lagrangian was terminated, since it became overly time consuming and with time constraints on the group no more time could be wasted on the model. Given more time, and less concern on designing the previous models, perhaps this model can be completed and then tested to see if it does resemble realistic motion of a human/robot on a swing.

3 Human Motion

3.1 Introduction

Alkesh Thanki

In order to induce a driving motion of the NAO robot on the swing system used in experimentation, both numerical models and human motion modelling were examined to assess their respective capabilities. Due to the possibility of a chaotic influence on the numerical models, it was decided that human modelling could provide a more realistic profile of the angle evolutions that could subsequently be used in driving a swinging motion of the NAO robot. The point at which the human applies torque from their torso and legs movement will be calculated as a function of the local time period in each of the oscillation. This data will subsequently be used to determine when to instruct the robot in each of its swing cycles. In addition to the direct implementation of human motion modelling, the results and deductions made by numerical modelling will be directly compared to further verify our understanding of swinging motion.

The induction of parametric resonance in swinging motion can be achieved in both a seated and standing position. An accurate description of both models must therefore be examined to not only define the optimal driving motion of the NAO robot on a swing but to investigate the theory behind human swinging motion.

3.1.1 Swinging from a Standing Position

In order to induce a periodic swinging motion from standstill, the rider must rotate its torso back and forth to oscillate its centre of mass about the pivotal axis; inducing simple harmonic motion of the swing. This can be modelled as a double pendulum system and is further illustrated in Section 2.4.1. In subsequent oscillations of the system, the rider is constrained to radial displacements of its centre of mass and therefore the impact of standing & squatting is translated to reducing & increasing the length of the pendulum system respectively.

The pumping of the swing is optimal when energy is injected into the system periodically as the rider performs maximal work against resistive forces. The rider should consequently raise its centre of mass by standing at the system's equilibrium; performing work against both the gravitational force due to its weight and the centrifugal force due its motion. At the extrema the reverse motion is made, from standing to squatting. This motion will reduce the gravitational potential energy but not as much as the energy injected into the system from standing at the equilibrium. Therefore the system gains energy. Since the frequency of movement is twice the natural frequency of the system, optimal parametric resonance is induced and the amplitude of the periodic motion is amplified.

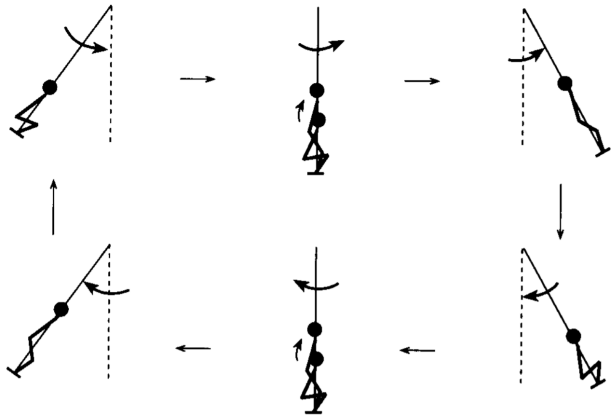


Figure 3.1: Depiction of optimal swinging motion from a standing position[16]

Figure 3.1 above depicts the ideal swinging motion of a rider in a standing position. In order to optimise pumping, the transitions between standing and squatting should be performed in the smallest time possible to enhance the rate of change of angular momentum and induce a greater torque at the critical points of the system.

3.1.2 Swinging from a Seated Position

As illustrated in Section 1.3, in order to induce a periodic motion of swinging from standstill, the rider must rotate its torso and legs about its centre of mass. This is conducted by the user throwing their head and torso in an opposing motion to their legs. At the peak of the arc going back, the user throws their head and torso back and their legs forward. This rotational motion about the centre of mass induces a positive (counter-clockwise) torque to the rider's head and feet. The torso and the swing itself experiences a reactionary negative (clockwise) torque. Since the rider positions their hand on the swing, a pivot is formed at this point. The negative torque induced by the torso is balanced with a positive torque induced around the pivotal point; slightly raising the centre of mass of the rider at the apex. This motion is reversed on the forward swing and therefore raising the centre of mass of the rider at both extrema.

The slight vertical increase of the centre of mass increases the maximum potential energy of the system; this results in a subsequent amplitude increase as potential energy is converted back into kinetic energy at the equilibrium point.

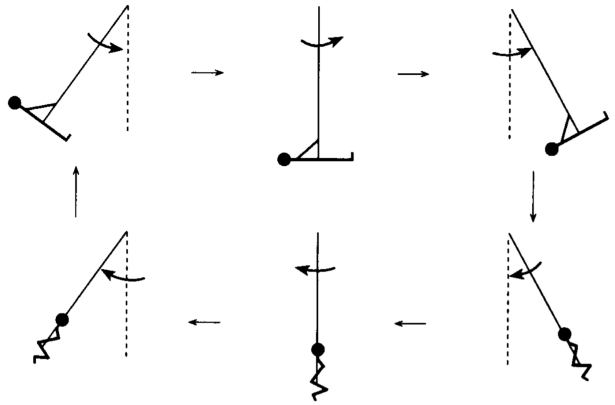


Figure 3.2: Depiction of optimal swinging motion from a seated position[16]

Figure. 3.2 above depicts the rotation of the torso and extension of the legs at the critical points of the motion. As illustrated in the standing scenario, the transfer of angular momentum should occur in an appropriate timescale where gravity does not significantly move the swing down i.e. perform the movements in the smallest time possible.

3.1.3 Standing vs. Sitting

As described in Section 3.1.1, the rider can induce a driving motion from a standstill in the standing position. However, this can only be induced if the rider rotates their centre of mass about the pivotal point in the resonant frequency range. When the rider is in the seated position described in Section 3.1.2, this motion is exaggerated and therefore efficiently induces a swinging motion from a self-start. Swinging from a standing position does, however, pump the swing more efficiently at larger amplitudes[16] and therefore the ideal swinging motion would be a culmination of both positions i.e. initially swinging in a seated position in order to self-start and then from a standing position to pump the swing at larger amplitudes. In the application of human motion, inducing a swinging motion from a standing position is more difficult to execute especially if the user has an altered mass distribution due to the physical instability of the system.

Due to the aforementioned reasons, it was decided to investigate the pumping motion of a human swing user in a seated position and assess its applicability to the NAO robot swinging from a self-start.

3.2 Replicating Mass Distribution

After reviewing the efforts of previous investigations, it was suggested that the difference in the mass distribution of the robot to the swing user could have caused the discrepancy between their resulting motions.[9] Therefore, this was investigated by adding weights to the swinger to not only match the respective mass distributions but also restrict an overextension of the knee joint. The proportion of the total mass of each major contributing body part of the NAO robot was prescribed in the NAO handbook,[12] and similarly the mass distribution of the swing user was found in the human body dynamics textbook.[17] Instead of accurately measuring the mass distribution of the swing user, it was decided that an average mass distribution would suffice due to the similarity between the user and an average human male. After investigating the mass distribution of the NAO robot, it was discovered that a considerable amount of its total mass was located in its legs, and since the centre of mass of an average human is located in the torso; most of the additional mass was added to the lower legs of the human swing user.

Table 3.1: Percentage difference between NAO Robot and Human Mass Distributions

	Human Mass[17] %	NAO Robot Mass[12] %	Weight Additions, kg	Percentage difference
Head	7.30	13.22	0.00	6.82
Upper arm	5.40	11.73	1.20	4.05
Forearm	3.20	2.91	0.00	0.10
Hand	1.32	6.80	0.60	4.99
Trunk	5.08	26.44	0.00	-18.10
Thigh	19.76	14.35	0.00	-2.97
Lower leg	9.30	11.33	2.00	-0.07
Foot	2.90	13.22	4.00	4.18

Table 3.1 above illustrates the altered mass distribution of the human swing user. As shown above, most of the contributing body parts were altered in order to reduce the percentage discrepancy to within 5%. Due to the physical incapabilities of the swing user, additional mass could not be added and therefore the percentage discrepancy in the trunk could not be further reduced. No additional mass was added to the head of the human due to the fragility of their neck and therefore the percentage difference similarly could not be reduced. It was initially decided to use measured sand bags to spread the additional weight across the swing user however, due to the relative accessibility of physical weights, the mid points of each body part were accurately measured and these were used instead. In future practice, extended and malleable weight additions could further refine the replication of the mass distributions and therefore precisely account for the discrepancy. The direct influence of this attempted replication of mass distributions to the driving motion of the human will be illustrated in Section 3.5.

3.3 Fieldwork and Data Collection

To accurately implement the model produced by human motion on the NAO robot, the swing system selected should be similar to that used by the robot. Consequently, it was in our interest to use a swing that is similar to the double-hinged swing used in the robotics laboratory. A comparison between these two swings is shown below in Figure 3.3; highlighting the position of the joints.

(a) Swing used by Robot

(b) Swing used by Human

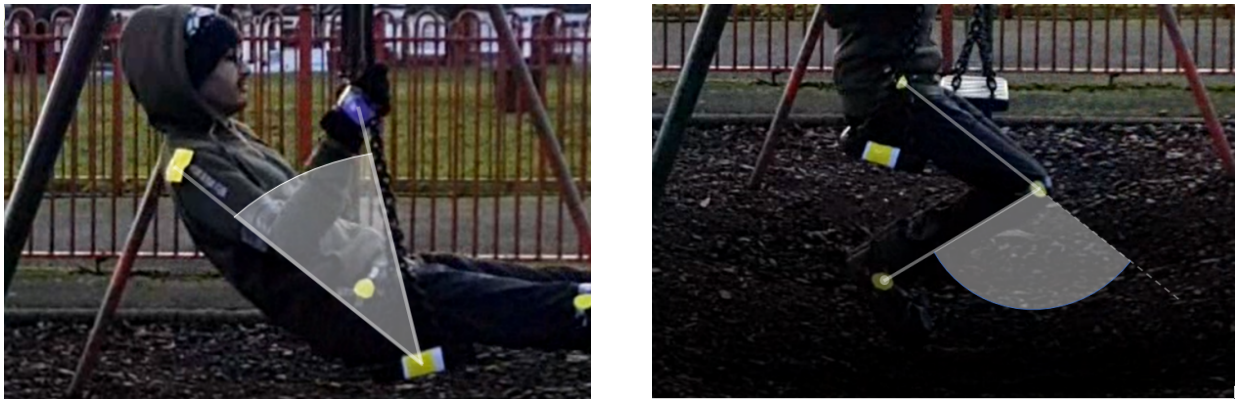
Figure 3.3: Comparison of Human & Robot Swing

As shown above in Figure 3.3, the swing system used in the robotics laboratory is similar, but not identical, to the swing used by the human. Other than its relative similarity to the NAO robot's swing it was chosen due to the convenience of its location (Selly Oak Park). Even though the swing used by the NAO robot is made entirely of rigid components it was assumed that if the swing user did not create a pivot on the chain using their hand placement, the lower section would have a negligible impact on the forthcoming motion. The swing used by the robot is also a double-hinged system whereas the swing used by the human is only single-hinged. This was an unavoidable discrepancy between the systems due to the difficulty in sourcing a double-hinged swing.

A high resolution camera was required to emphasise the contrast of the markers used for tracking. The Samsung Galaxy S8's camera was therefore used due its capability to record in 1080p resolution at 60 frames per second (fps). The camera was attached to a clamp stand and positioned (3.97 ± 0.10)m away from the swing system, at a height of (1.09 ± 0.10)m. The camera was positioned as far as possible away from the swing system in order to reduce the effect of horizontal parallax. Similarly, the camera was raised to a height which was approximately half the height of the swing system to minimise the effect of vertical parallax.

The major components of an induction of pumping motion on a swing from a seated position are the rotation of both the torso and legs. It was therefore imperative to track the evolution of respective angles enclosed

by these motions as the human drives the swing. The torso angle was defined to be between the axis of the torso and the axis normal to the seat. The knee angle was defined as the inverted angle between the thigh and lower leg axis. Markers were therefore positioned on; the hip, knee & ankle in order to track the knee angle and the seat, shoulder & hand to track the torso angle over time. The hand was positioned on the hinge of the swing and therefore was used to define the axis normal to the seat. Figure 3.4 below depicts the defined angles and also illustrates the position of the assigned markers.



(a) Torso Angle (b) Knee Angle

Figure 3.4: Illustration of Torso & Knee Angle

In addition to recording footage of the human driving a swing from rest with the altered mass distribution, motion without the added weight was captured to analyse its influence on parametric resonance. Driving motion isolating the torso was also filmed since it was noticed that the NAO robot's torso movement was limited due to the relatively short length of its arms, and therefore restricting the torso angle may represent the robot's swinging motion more accurately. The video footage was subsequently processed and uploaded to a computer. Due to intermittent clouding during the day of filming, the brightness and contrast of the markers in the footage was diminished. In order to account for this, *movie edit touch*,[18] a video editing software, was used to maximise the brightness and raise the contrast to +50. This would permit the use of the autotracking feature in the video tracking software, discussed later in the section.

In order to extract information of the torso and knee influence on the driving motion from a user on a swing, the angles of these movements must be tracked. The motion capture software used was *Kinovea*,[19] due to its ability to auto-track multiple objects simultaneously. Unfortunately, in practice, this feature was inoperative and as a result of this, the markers were manually tracked frame by frame; consuming time and accuracy. The software also failed to track the angle of the knee and torso directly therefore the position of the markers were tracked relative to a moving origin and simple vector calculation computed the desired angles, shown below in equation 3.1.

$$\theta = \cos^{-1}\left[\frac{d_1^2 + d_2^2 - d_3^3}{2d_1 d_2}\right] \tag{3.1}$$

where, for the torso angle, d_1, d_2 & d_3 are the hand-seat, shoulder-seat and shoulder-hand distances respectively. The knee angle is defined to be π minus equation 3.1, and therefore d_1, d_2& d_3 are the hip-knee, ankle-knee and hip-ankle distances respectively. The tracking frames needed to be calibrated in order to output the data in the relevant format therefore a measurement of the knee to ankle distance was taken and used as the standard calibration length. Due to the relatively short distance between the position of the camera and the swing system, the impact of parallax was assessed. The maximum angle as seen from the recording position was $(1.12 \pm 0.04)^c$; comparing this to actual angular displacement of $(1.14 \pm 0.08)^c$ rendered a parallax error of just 1.57%. Due to this minimal discrepancy, the effect of parallax was neglected in data analysis.

3.4 Verifying Motion

Karandeep Giddha

The angular data that was obtained using *Kinovea* was processed in *MATLAB*. The following graphs were created to verify that the angular data matched what would be expected at different points of the user's traversed motion. The points of interest include (i) the motion of the torso and the knees at extrema, (ii) the motion at equilibrium, and (iii) the angular velocity and acceleration throughout the motion respectively. This is important because of the laborious nature of tracking and how the intricacies of using *Kinovea* can make one susceptible to errors such as misidentifying tracking markers.

Figures 3.5 and 3.6 show the motions of the torso and the knees. With the aid of the swing angle, one can trace the (local) extrema of the swing oscillations to the corresponding angles for the torso and the knees. At the extrema, the rate of change of the torso and knee angles are roughly at a maximum. This is to be expected as one would expect a user on a swing to be attempting to change its position relative to the seat of the swing at the maxima and minima of the swing angle; justified in Section 3.1.2.

Such a justification is also compounded by viewing the motion of the user frame-by-frame using *Kinovea*. With point (i) verified, point (ii) is subtle but almost unnecessary. When the swing goes through the equilibrium position, the rates of change of the torso and knees are almost zero. Given human motion relies on the user to intricately follow a set motion, it is understandable to expect the user to attempt and fail to apply the movements precisely. Even with imprecise applications of movement, one would still expect a very defined oscillatory motion, even if the motion of the user is not as responsive as a robot. Therefore, the motion that is exhibited by the swing user might not be ideal, but it should still be able to be implemented on the robot.

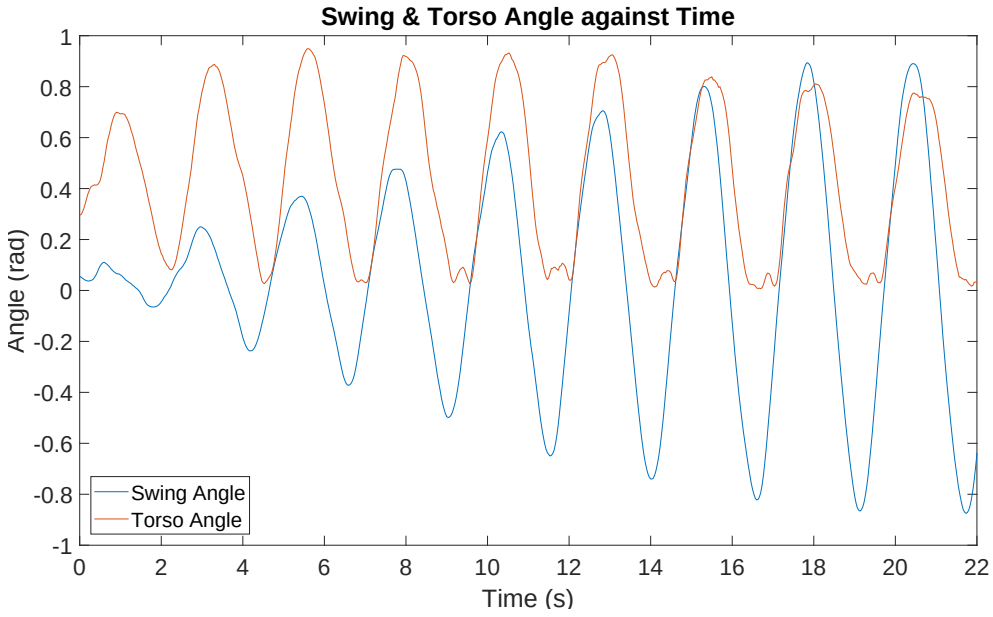


Figure 3.5: Torso angle as a function of time. The main swing angle has been depicted to help track the motion of the swing.

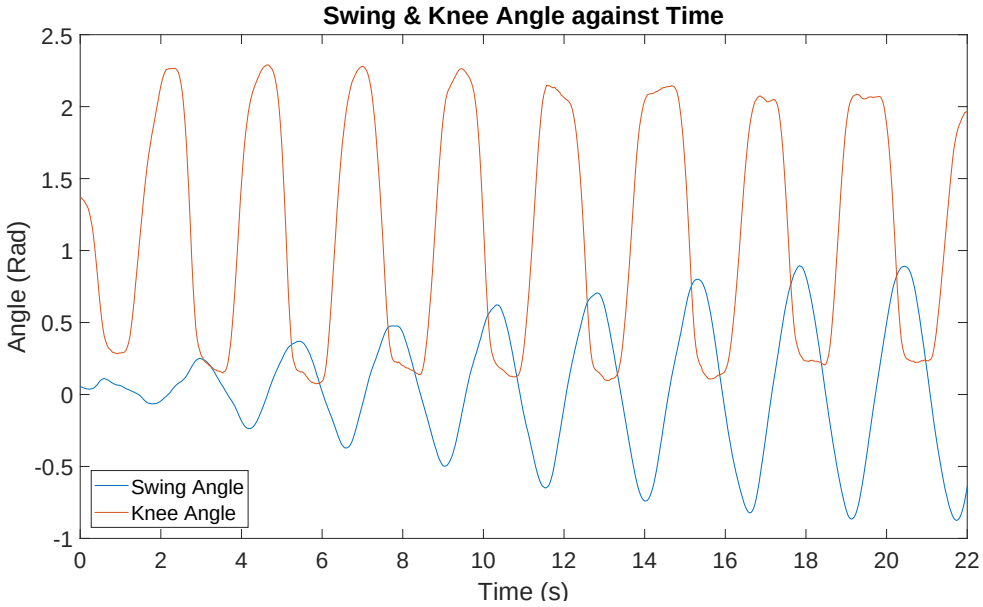


Figure 3.6: Knee angle as a function of time

3.5 Influence of Redistributing Mass

The evolution of the swing angle was compared for altered and unaltered mass distributions. This is shown in Figure 3.7. The distributions almost match. Only through a magnified portion of the illustration in Figure 3.7 can you see minute deviations in the distributions. In reference to the description presented in Section 3.2, one may expect that changing the mass distribution so that the centre-of-mass is lowered would change the torque applied to the system. Hence, a different motion may be expected. For instance, one could redistribute the mass to an extreme - add mass in such a way that the body essentially reduces to the legs only. Therefore, one would expect that the growth in the amplitude to be far steeper than with the torso. This was verified from the simulations that were numerically modelled. Such a model is akin to the double pendulum. The double pendulum distribution in Figure 3.8 has a maximum amplitude that is always greater than the single flail model. At resonance, the difference is maximal and the difference is approximately a factor of 0.6. However, adding mass would only require the user to apply more effort to drive the pumping of the swing. Hence the required pumping actions may take an unideal amount of time to be applied. Overall, although the lowering of the centre of mass is expected to increase the efficiency of reaching the maximum angle, the effect can be considerably reduced due to the effort that is required to make pumping movements by the addition of mass to a human. For this reason, you would expect the motions to be identical, at least to within the tested 5% redistribution accuracy. Indeed, matching the mass distributions to within 5% requires one to work with mass scales that would not be large enough to reduce the system to that of a double pendulum. Hence, the motion would not be expected to be changed to that in Figure 3.8.

It was concluded that redistributing mass does not significantly change the user's motion on a swing. As a result, from here onwards only the redistributed mass dataset was considered. Any further analysis was based on this dataset and it was also chosen to be implemented on to the robot.

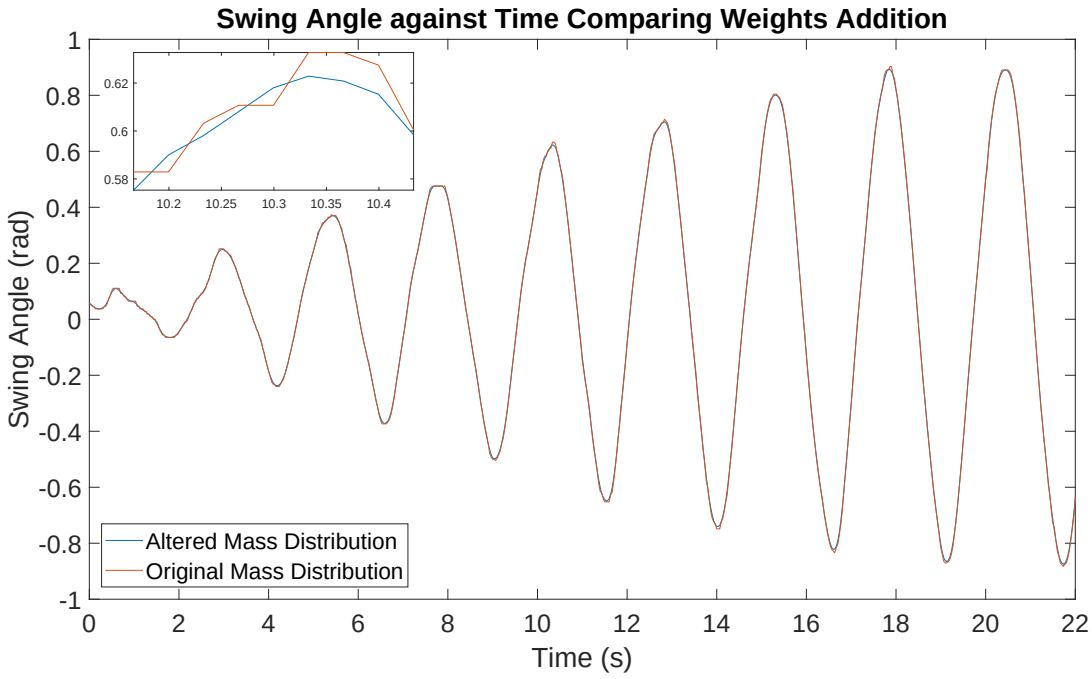


Figure 3.7: Effect of redistributing mass on growth of swing angle. A magnified portion of the graph is shown, in the time interval, t 10.15 $<$ t $<$ 10.45.

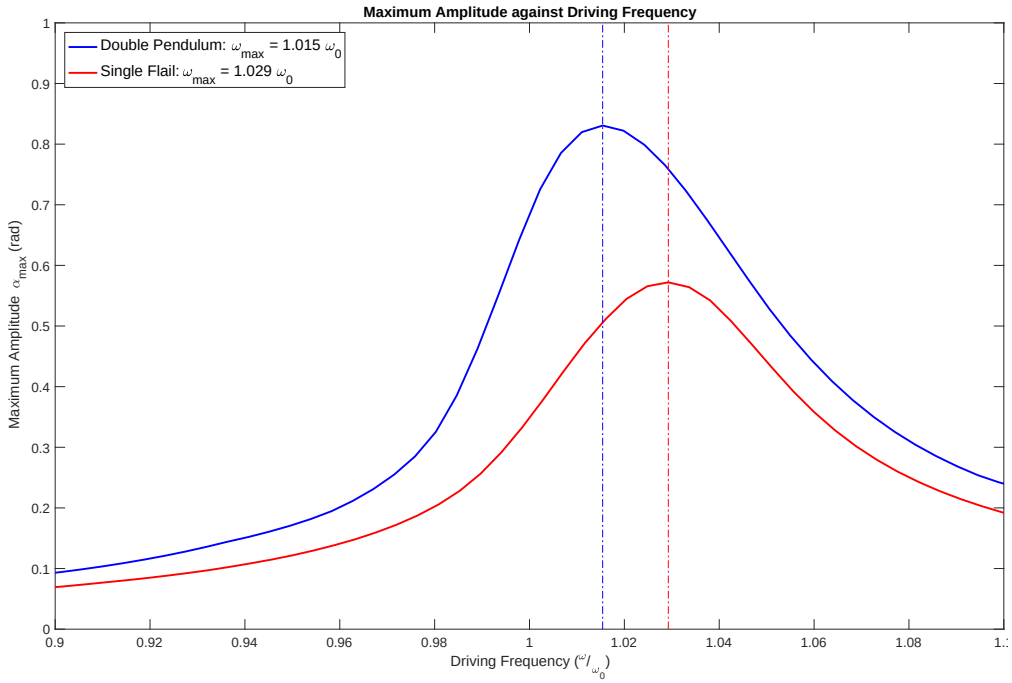


Figure 3.8: Maximum amplitude for the Double Pendulum reached 0.83 rad, whilst the Single Flail simulation reached only 0.55 rad

3.6 Implementation to Robot

By fitting oscillatory functions to the torso, knee and swing datasets, one could establish parameters such as the offset and the angular frequency. Applying fits allows one to implement the human motion data to both the physical robot and to the virtual robot in the *Webots* simulation. Using such fits, two steps were undertaken; (i) the local time period was identified, (ii) an angular acceleration for each of the datasets was computed.

In Figure 3.9, two fits have been applied (to the swing distribution) to better reflect the whole dataset. The R^2 values for each of the separate fits indicated a much better representation of the data than for a single fit. Given each of the fits have different angular frequency parameters, one would conclude that the time period is not constant. Of course, the time period for the torso and knees should also follow suit. The angular datasets for these were also fitted. Indeed, fitting in limited regions provided a better set of R^2 values than a single fit could provide. Hence, to assume that the period is not a constant is consistently justified by applying more than one fit to different regions of the datasets. Although these fits allow one to immediately extract the value of the period in different regions, it was decided that the period of each oscillation should be assumed to be different. Physically this can be explained by factors such as variable air-resistance and chaotic movements which could alter the oscillation. In light of the variable time period, rather than fitting functions to each oscillation it was decided that inspecting the swing dataset itself would be the simplest method in defining time periods for each oscillation. Besides, one would be required to judge the points which correspond to the starts and ends of oscillations to then be able to apply fits, regardless of whether one aims to fit to each oscillation or not.

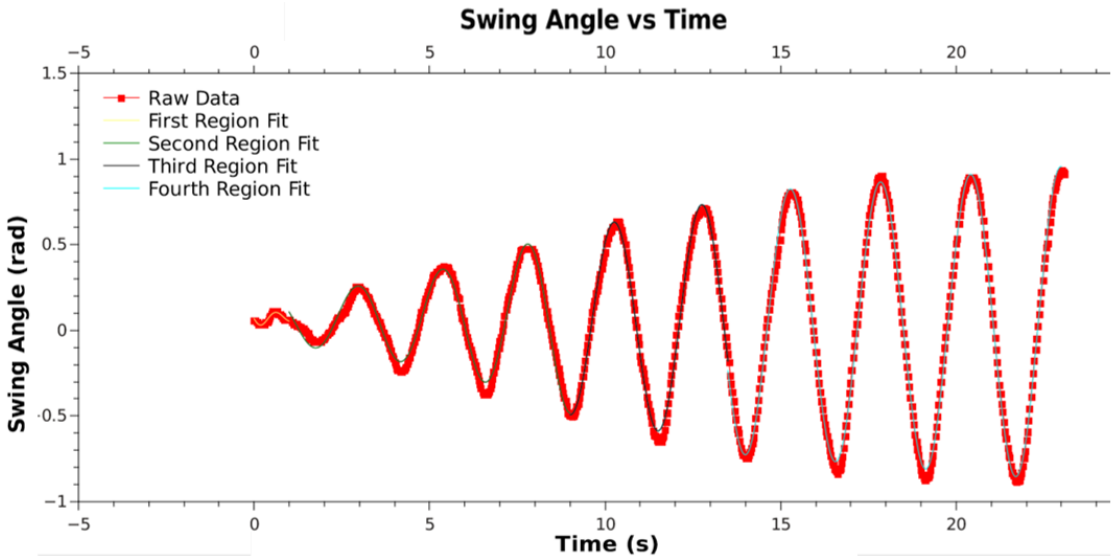


Figure 3.9: Swing angle as a function of time. The fit parameters are shown in Table 3.2.

Table 3.2: Fit parameters for the swing angle, illustrated in Figure 3.9. The number of fits that have been applied are shown, whilst the angular frequencies, ω, have been shown for each of the fits. Values in () represent the R^2 value which corresponds to the parameter of each fit.

Number of Fit Functions (Swing)	Angular Frequency, ω (rad)		
	ω_1	ω_2	ω_3
1	2.485 ± 0.002 (0.942)		
2	2.547 ± 0.002 (0.954)	2.502 ± 0.002 (0.961)	
3	2.599 ± 0.005 (0.981)	2.516 ± 0.005 (0.994)	2.446 ± 0.001 (0.996)

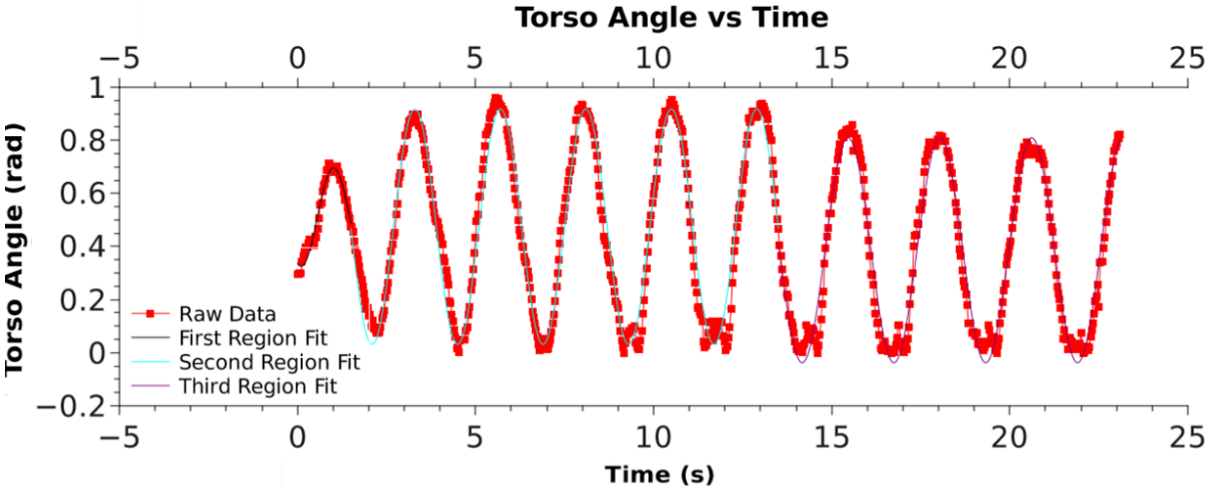


Figure 3.10: Applying multiple fits to torso angle over time

The fit parameters in Table 3.2 suggest the angular frequency is not a constant. Making this assumption, it was important to identify the points at which a single oscillation starts and ends. The fits also provided an offset parameter which gave rise to a *"zero-point"*. This point allowed the local time period of each oscillation to be calculated. The equivalent of Table 3.2 for ω can be located in Appendix A.3

To apply this to the robot, it is only necessary to compute the fractional time period of the swing angle. By finding the fractional time period of the local oscillations for the swing dataset, the points at which the rate of change in angular acceleration (torso and knees) is maximal are the key points at which movements are applied to the robot.

Torso and knee movements are represented as fractions of the time period in an oscillation. These fractions are illustrated in Figure 3.11. The times at which these occur should be noted and the corresponding fractions of the time period for the swing should be referred to. Since these values describe the points at which a movement should occur in a single oscillation, one could theoretically apply these to the robot separately to each oscillation. In consultations with the *Webots* team, it was concluded that only a single value could be applied to the robot. An alternative method was explored. An attempt was made at tracking each oscillation and providing the point at which movements occur. However, programming issues led to only applying a single value. As a result, an average of the points at which movements occur was computed. It is necessary to note that the distribution in Figure 3.11 is of a random nature. Therefore the residuals do not suggest that an underlying function should be fitted. For this reason, an average of the fractional periods is not a detrimental consequence of the (limiting) programming issues. There were two considerations that had to be examined to average over the different movement points. One was to consider that from the data it appears torso and knee movements are applied at different times. The second is that two movements are made in a single oscillation. The *Python* script that instructs the time at which movements are applied is activated every time the swing passes the equilibrium point. To account for this, an average over both the torso and knee times was taken, whilst an average over the two times relative to the equilibrium point was taken for both the torso and knees. This time was calculated to be (0.15 ± 0.02) s, where the quoted number is the average fraction of a time period. The resulting motion of the robot was observed to reach $(13.01 \pm 0.20)^\circ$. The implementation of this has been discussed in Section 7.8 and the angular acceleration of the resulting motion has been noted to help elaborate on differences in the motion.

A future consideration would be to estimate the position of the centre-of-mass and see how the distance from the swing pivot varies. This can be done by looking into the peaks of the angular distribution and using the data to backtrack to the length of the pendulum (which would locate the centre-of-mass). This would help to understand the movement of the centre of mass, which is the main component that drives the motion. Of course, this analysis can then be propagated onto the movement of the robot, and a comparison can be made to indicate how the motion reflects upon human motion more intricately.

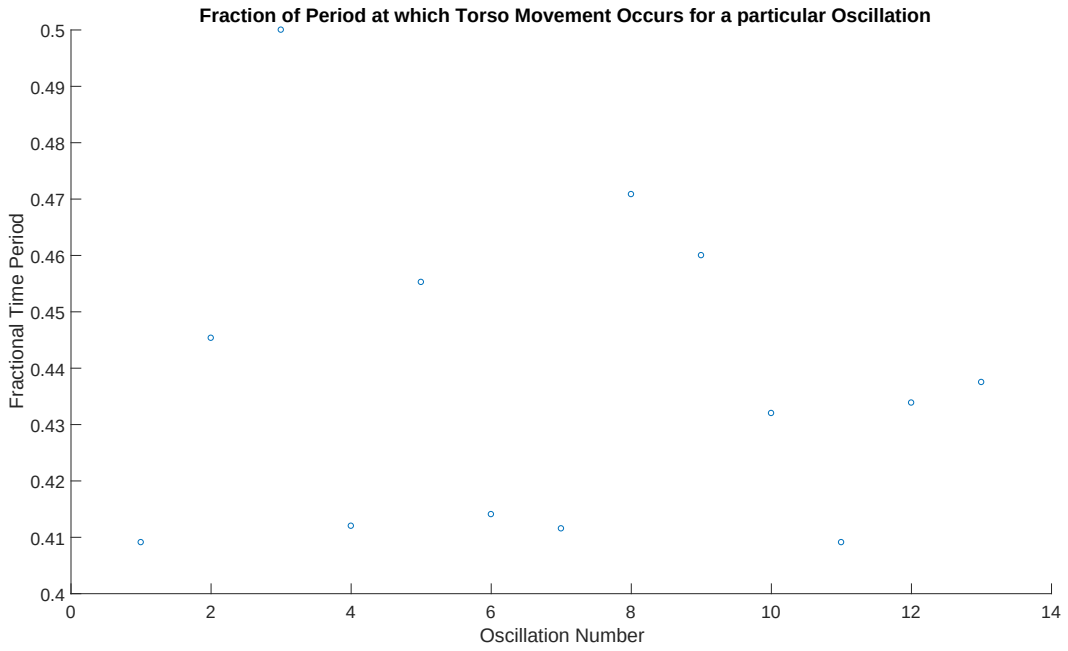


Figure 3.11: The fraction at which a torso movement is applied for a given oscillation is shown.

3.7 Conclusion

Alkesh Thanki

In conclusion, the influence of a altered mass distribution was investigated by attempting to replicate the centre of mass position of the NAO robot. It was initially hypothesised that the difference would have an effect on the resulting motion due to the addition of mass increasing the magnitude of torque applied at the knee joint. However, the converse was found in that the mass distribution had a very negligible impact on the evolution of the swing angle. A possible explanation for this is that the swing user exerts more effort/torque but over a larger period of time and therefore having the same effect on the resulting swing angle. The extreme case of applying a significant amount of mass to the legs was modelled as a double pendulum and found to have a significant effect on the maximum amplitude. However, since most but not all of the robot's mass is located in its legs, its motion can't be modelled as a double pendulum system. Therefore, an exaggerated addition of mass would alter the swing's dynamical motion but would not replicate the mass distribution of the robot and therefore can't be implemented. After extracting the evolutions of the torso and knee angles for swinging motion from standstill, the points of the applied torque were analysed. It was deduced that the torque was applied close to when the swing approaches the apex of swing angle. This is concurrent with the investigations of numerical modelling described in section 2.7.4 and therefore verifies the theory described in Section 3.1.2. In order to implement human motion modelling on the NAO Robot, the angles of the knee and torso were found as a function of the local period. In practice, the implementation of a varying time period proved troublesome and therefore a constant time period was approximated instead. The fraction of the swing angle for which the torque were applied was subsequently applied to the robot; successfully achieving a swing amplitude of $(13.01 \pm 0.20)^{\circ}$ over a period of 4.06 minutes. This initial pumping of the swing verifies the application of self-start from human motion modelling and therefore the methods described in section 3.6 can be replicated in future investigations of robotic swinging motion.

3.8 Evaluation

In retrospect, various areas of human motion modelling could have been refined further to produce more accurate and conclusive outcomes. The mass distribution of an average male was used instead of measuring the mass of each contributing component of the swing user. An accurate measurement could be used to precisely locate the centre of mass of the user and therefore replicate the mass distributions to a more significant degree of accuracy. The swing used for human motion differed from that used by the NAO robot. The addition of an extra hinge in the swing system would provide an additional avenue of energy dissipation; altering the pumping of the system. Therefore, a consistency in the swings used would provide a more applicable profile of angle evolutions; improving the maximum amplitude achieved by the NAO robot. The presence of torsional rotation was noticed in both robotic and human motion. An investigation to the effects of this motion could be examined and accounted for to transfer the dissipated energy into energy that can be used in the pumping of the swing system. Lastly, the time period of each oscillation of human swinging was approximated to be constant even though a variation was noticed. The application of a dynamic fractional period could be used to retain the phase relationship between the applied torque and the swing angle; improving the overall pumping of the swing used by the robot. Ultimately, the aforementioned modifications should be adopted in future investigations of applying human motion to the NAO robot to not only improve on the results achieved in this study but to advance our notions of achieving parametric resonance on a swing.

4 Encoders

4.1 Justification

Henry Birks

The simplest model of a swing has rigid rods for the arms. However, in reality most swings use a chain or rope for the arms of the swing. To bring the model of the swing closer to reality it has been decided that the two joints on the swing arms will be opened, this mimics the two hinges created above and below your hands when you grip a rope swing. The position of the new encoders is shown in figure 4.1. In previous years there has been no way to measure the angles created by these joints. Being able to know these angles in real time will allow more complex models to be created involving feedback mechanisms which will improve the efficiency of the robot's swing. The minimum number of encoders required is two, one for each joint on one arm of the swing. A further two were added onto the other arm so the difference in the angles can be measured. It is expected that there will be torsional forces on the swing which can be seen in the difference of angles on the swing.

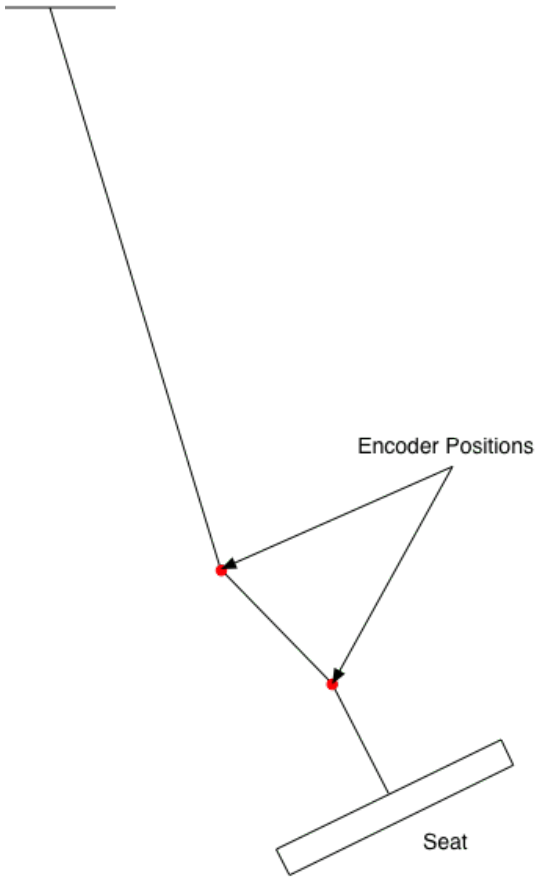


Figure 4.1: Diagram of Swing showing Encoder Positions

4.2 Encoder Types

The angle encoder currently being used at the top of the swing is an optical encoder. The other two encoder types are absolute encoders and incremental encoders. Optical encoders make use of 'Reflected Binary Code' or 'Gray Code'. This is an ordering of the binary numeral system which ensures successive values only differ by one bit. A visual depiction of this can be seen in figure 4.2. Gray code helps to reduce spurious readout errors. With an optical encoder, the wheel with the binary code on can land between two positions, resulting in the encoder reading out a mix of both binary strings. With normal binary this could give a very different value to the actual value due to many of the bits changing between two successive binary values. With Gray code however, only 1-bit changes between successive values which prevents these spurious outputs. Optical encoders are relatively limited by how many individual positions are possible.

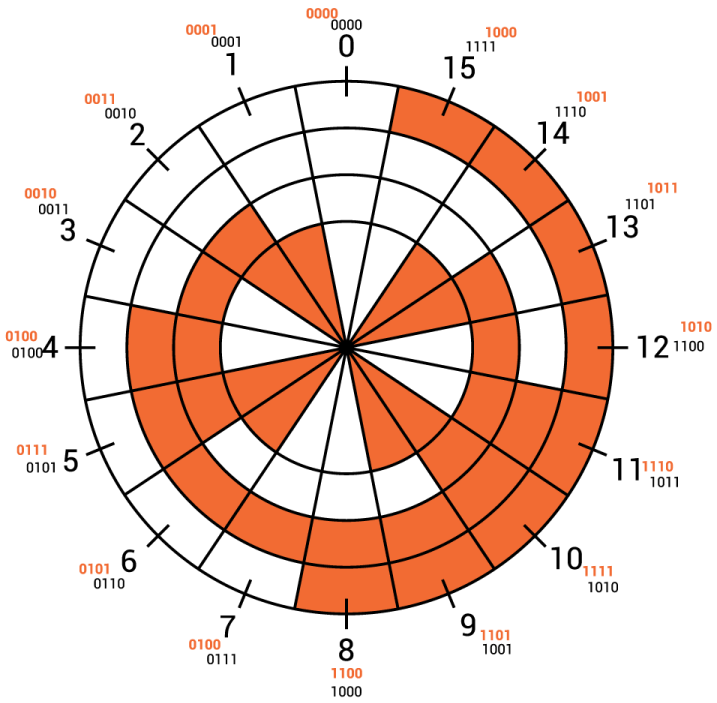
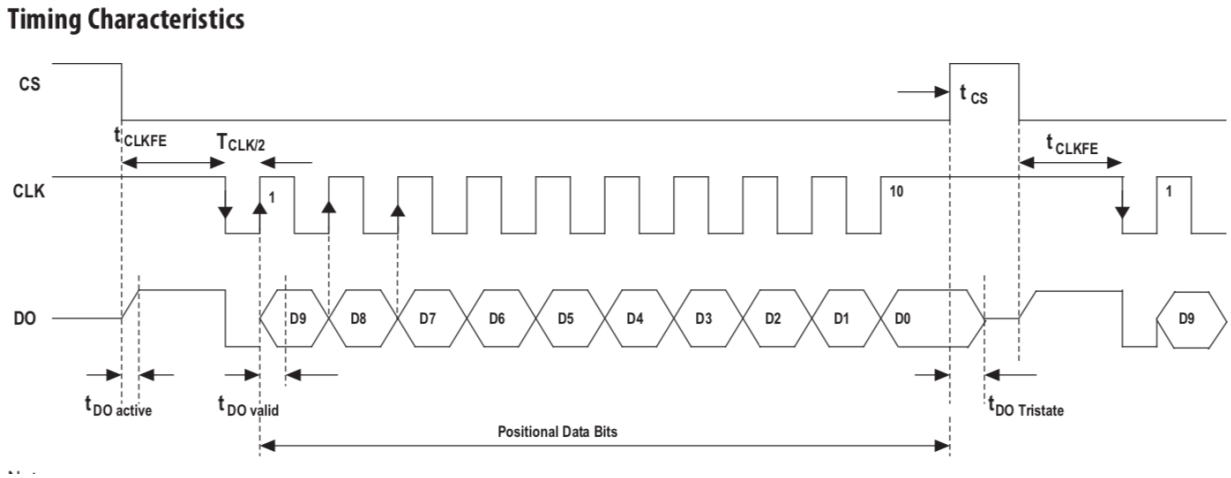


Figure 4.2: A diagram of a typical Gray code optical encoder wheel[20]

Incremental encoders also suffer from the same issue. They tend to have much fewer positions than absolute encoders and often use mechanical switches. Incremental encoders have many uses as dials, such as a volume knob, where a few discrete positions are required. An absolute encoder can have a much higher resolution than an optical encoder. Absolute encoders measure a change in a magnetic field to calculate the angle. Due to the encoder being magnetic, the design is non-contact which ensures reliable operation and a longer lifetime than encoders which utilise contact points. Absolute encoders can also offer a much higher precision than optical or incremental encoders. For these reasons, an absolute encoder was chosen rather than an optical or incremental encoder. The encoder chosen was the AEAT-6012, manufactured by Broadcom Limited.[21] This is a 12-bit magnetic encoder with a resolution of 0.08°.

4.3 Encoder Build Plan

Once the encoder had been selected, a circuit had to be designed. As there were 4 encoders, a way to connect them together and read all 4 encoders at once to a PC was required. A chip would be required to manage the 4 encoders. The chip would need a USB connection so that the data can be read by a PC and used in the code to control the robot as a feedback mechanism. The output from the encoders is a 16-bit string which contains the 12-bit positional data. This can be seen in figure 4.3. Once the chip select pin has been activated it takes 500ns for the first bit of data to be outputted. After this each bit of data will be outputted on the leading edge of the clock signal. The chip selected would have to be able to read this. Also as 4 encoders are being used, the functionality of selecting which encoder to read from is required along with at least 4 I/O connections, one for each encoder. The encoders run on 5 volts, so the chip selected must also have the ability to run on 5 volts. The MCP2210 chip was selected as it met all of the criteria.[22] There are 9 GP (General Purpose) input/output ports, the ability to run on 3.3 volts or 5 volts and a USB 2.0 connection.

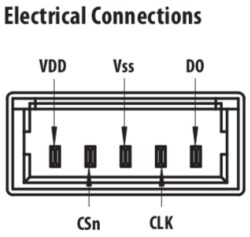


Note: This is a diagram for the 10 bit version of the encoder, the 12-bit version being used in this project has 2 additional positional data bits.

Figure 4.3: Output from the angle encoder[21]

4.4 Build Log

All parts for the encoders were ordered from Farnell.[23] The process of making the electrical connections for the encoders began as soon as the parts had arrived. Initially a breadboard was used as a 'test rig.' One encoder was mounted on the board and a magnet was mounted to a dial that could be turned by hand to test the angle read out. A breadboard was used as it allowed easy modifications to the wiring and did not require any soldering. The angle encoders use a 5-pin MOLEX connector. Figure 4.4 shows the electrical connections of the AEAT-6012 encoder. As expected there is a pin for a 5V supply and ground. The other 3 pins are; chip select, serial clock and serial data.



Pin	Symbol	Description
1	VDD	5V Supply Voltage
2	CSn	Chip Select – Input (See Figure 2)
3	VSS	Supply Ground
4	CLK	Serial Clock - Input (See Figure 2)
5	DO	Serial Data - Output. (See Figure 2)

Figure 4.4: The electrical connections of the AEAT-6012 encoder[21]

The MCP2210 chip board has voltage select pins. The chip can either output 3.3V or 5V. By default the chip runs off 3.3V, however the angle encoders require 5V. To change the voltage of the chip, the connection between the 3.3V pin and the left most voltage select pin must be broken. The connection between these two pins is on the back of the circuit board. In order to sever the connection a small incision between the pins will suffice. Once this connection is broken, a jumper between the middle and right most pin must be added to select 5 volts. If the connection on the back of the board is not broken and a jumper is put into place, the PC will not recognise the board.

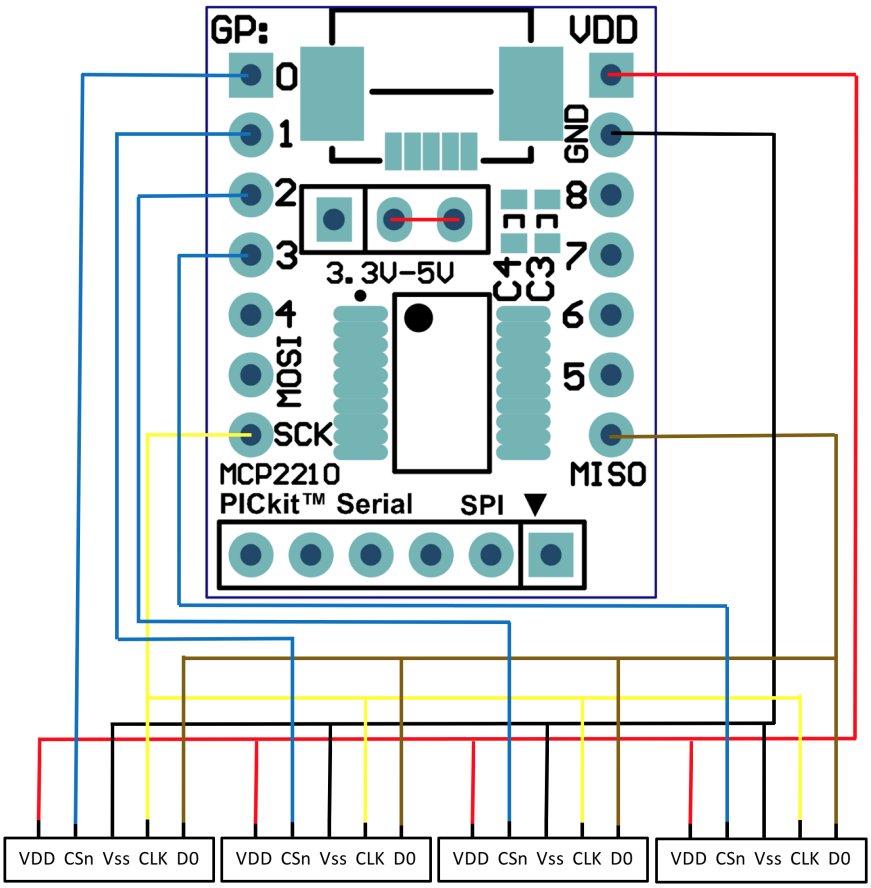


Figure 4.5: Wiring diagram of the MCP2210 chip[22] and four AEAT-6012 encoders

Now that the board was working off 5 volts, connections to the encoders could start being made. A circuit diagram can be seen in figure 4.5. For all four encoders, the VDD pin on the encoders must be connected to the VDD pin on the board, which provides the 5 volt power supply to the encoders. The Vss pin on

the encoders must be connected to the ground pin on the board. All four encoders must have their CLK pins (Serial Clock - Input) connected to the SCK pin on the board, this provides all four encoders with the same clock signal. The DO pins (Serial Data – Output) of the encoders must be connected to the MISO (Master-In Slave-Out) pin on the board. The data bits from the encoders will be sent to this pin. Finally, each individual encoder must have its CSn (Chip Select - Input) connected to a different GP I/O (General Purpose Input/Output) pin on the board. Only 1 encoder can be active at a time as they are all outputting their angle to the same pin on the board. If more than 1 encoder is active at the same time, the data sent to the MISO pin will be incomprehensible. This means care must be taken that any code written only activates one chip select at a time.

Mounts for the encoders had to be made. The encoders measure the relative angle between the two arms connected to one joint. This means that they must be fixed to one arm and allow free rotation about the hinge. A shaft of 6mm must be machined as the magnets slide onto a 6mm shaft. This shaft is fixed to the arm that the encoder is not fixed to. The workshop created a 3D model in Solid Works from sketches, this can be seen in figure 4.6. The workshop then machined the parts from aluminium blocks. The build time of the workshop is roughly 2 weeks, so once the parts had been finished they were attached to the swing. As the new parts are integrated into the original swing design, the swing had to be taken apart and rebuilt with the new encoder mounts. As the swing was already being taken apart, the opportunity was taken to move the centre of mass of the robot back. This was achieved by shifting the seat back by drilling new holes and moving the handles back. Before these modifications were made, the centre of mass of the robot was not in line with the swing arms, causing it to be tilted forwards when the hinges were open. This has now been remedied.

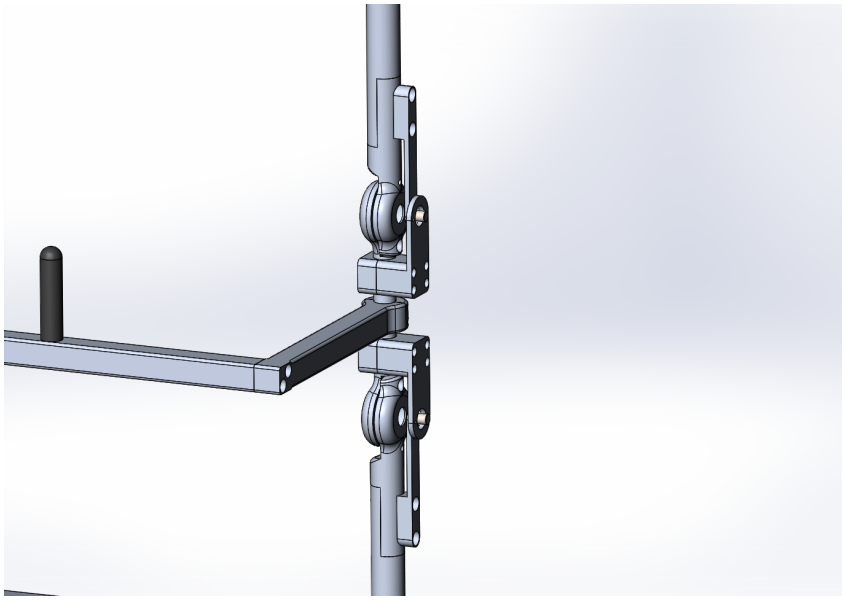


Figure 4.6: Solid Works design of the Hinge Encoder Mounts

The next step was moving the encoders from the breadboard to a PCB (Printed Circuit Board). It was decided that the PCB would be mounted on the bottom of the swing as the longer the wires from the PCB to the encoders the higher the capacitance of the wires, which can cause signal errors. The signal from the USB 2.0 port on the PCB can travel through a wire of up to 5 metres without any issues. The USB wire runs from the PC, down the swing arm to the PCB on the bottom of the swing. Wiring harnesses for each encoder were created. They were designed to be long enough to reach the encoders with enough slack to allow full movement of the swing but not so much slack that it gets in the way or can be snagged easily. All five wires were wrapped together with heat shrink to add to their durability and further reduce the chance of one wire snagging. They were also colour coded with a red wire on one side to signify the VDD connection.

4.5 Hinge Encoder API

Max Elliott

The Python script *HingeEncoders.py* was written as an API for the hinge encoders, with the help of Mark Colclough for interfacing with the encoders. It follows a similar approach to the main angle encoder's API script, containing functions for calibration and for retrieving the angles. Calling the functions from within the script, it takes approximately 8ms to retrieve a single angle from an encoder, which is easily fast enough to use as a feedback mechanism by the robot.

The script function documentation is as follows:

- getBytesX() : returns the absolute value of encoder X (0 to 3) in bytes
- getAngleX() : returns the calibrated value of encoder X (0 to 3) in degrees
- getAngleLeft() : returns the full calibrated angle in degrees for the left hinge (as viewed by the robot)
- getAngleRight() : returns the full calibrated angle in degrees for the right hinge (as viewed by the robot)
- calibrate() : sets current position of all encoders to be the zero point

4.6 Results

Henry Birks

Figure 4.7 shows a plot of the total hinge angle against time. The total angle is the angle of both encoders on one arm combined. This data was taken by simply pulling the swing back and releasing it with no driving force. Due to the chaotic nature of the triple pendulum some of the points are slightly spurious. It can be seen that the amplitude of the swing is decaying with time, which is what would be expected. The motion is also periodic. This is what would be expected, so it shows that the hinge encoders are functioning as designed. The polling rate of the encoder seems to be limited by the python code that is getting the angles rather than the speed at which the encoders can output their angle. The polling rate is fast enough with the current python script, however it is something to be mindful of when writing other code that involves the 'getAngle' functions for the hinge encoders. A work around could be to use a C++ code instead or use threading to run two python loops simultaneously.

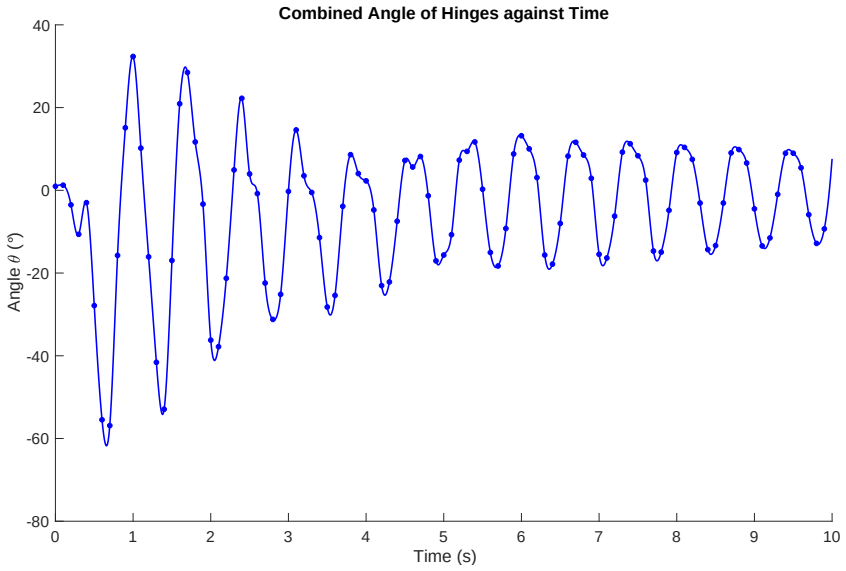
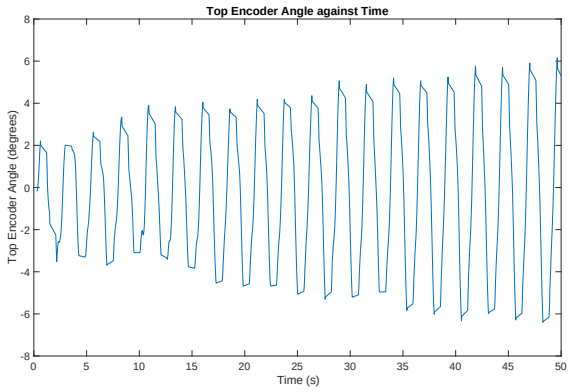


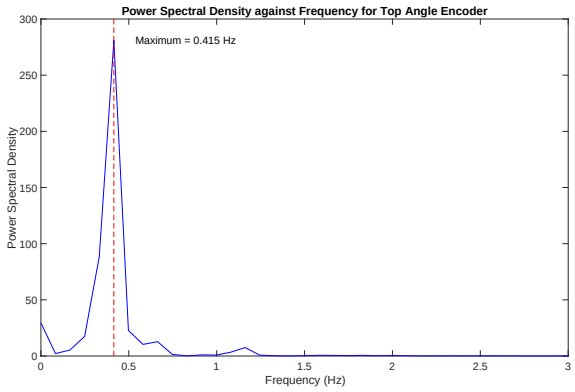
Figure 4.7: Total Hinge Angle against Time

4.7 Data From The Secondary Encoders

Arthur Mazendame

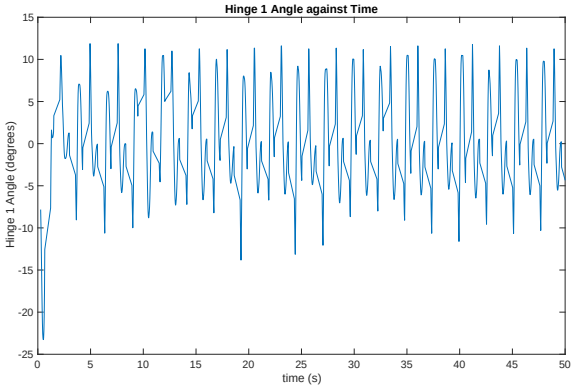


(a) Angle against Time

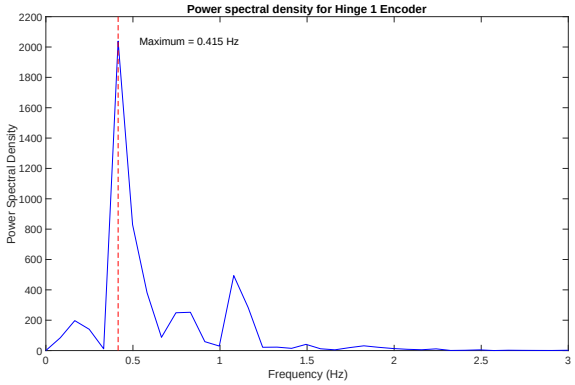


(b) Fourier Transform

Figure 4.8: Top Angle Encoder Angle θ and its Fourier Transform

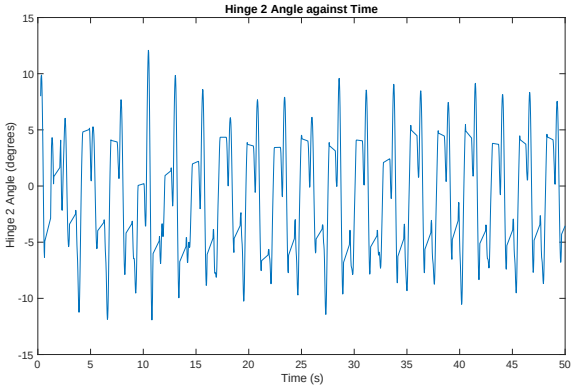


(a) Angle against Time

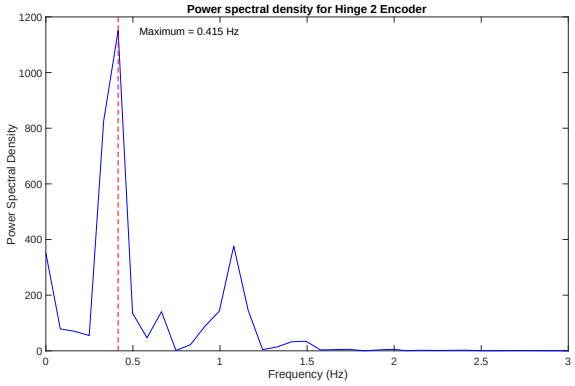


(b) Fourier Transform

Figure 4.9: First Hinge Encoder Angle θ and its Fourier Transform



(a) Angle against Time



(b) Fourier Transform

Figure 4.10: Second Hinge Encoder Angle θ and its Fourier Transform

The four additional encoders added to the swing can provide insights into what is happening during the chaotic motion of the swing, when the lower joints are free. The data for the top encoder and hinge encoders 1 and 2 time can be seen in figures 7.1(a), 7.1(a) and 7.3(a) respectively. The top encoder data in figure X shows a periodic function whose amplitude is increasing, as the swing gains height. The Fourier transforms of this data are provided in figures 7.1(b), 7.1(b) and 7.3(b) respectively. The frequency in each Fourier Transform with highest weight appears to 0.415Hz, corresponding to a period of 2.41s. This comes as no surprise for the top encoder as 2.41 is close to the calculated period of the physical swing 2.56s (within 5.9%). The noise in the data may be as a result of the polling rate of the program being significantly lower than that of the encoders. This can be attributed to complexity of the program and its inability to utilise the encoders at their maximal polling rate. This can be tested by taking data from the encoders with a program designed solely to do so. The data from the top encoder is asymmetric about the x axis. This may be due to a difference in the applied torque at the maxima of the swing, as the motions probably don't produce a torque of equal magnitude. The data also shows quite sharp peaks. A smooth and continuous change is expected. The discontinuity potentially has its cause in the impulse caused by the acceleration of the robot's joints when it moves from one position to the other. With an average value of -3.6±0.2°, a small bias toward negative amplitude may be inferred from the data. This average is taken from data from an integer number of complete periods. In the equilibrium position, the swing's seat leans back, when the legs and arms are extended and when they are not. The net effect this natural position may have on the swinging motion is that it may damp the motion caused by a torque in the forward direction. This may be alleviated by adjusting the position of the seat such that its forward lean in the tucked position is the same as the backward lean in the extended position, with a flat seat when the robot is in between its two extreme positions. It is quite clear that the encoders work effectively and can be used for further investigation. The potential reasons for the data to have this form are discussed later.

The secondary encoders, in principle, can be used to measure torsional forces on the swing and therefore can be used to calculate the amount of energy being wasted in lateral oscillations of the swing. This may be achieved by plotting corresponding left and right-hand encoders and looking for a discrepancy in the angles read by each. The data from 4.8 clearly shows larger amplitude gain from the motion in the negative.

4.8 Potential Uses of Secondary Encoders

As mentioned before, the dominant frequency in all of the Fourier transforms of the encoder data is the same, 0.415Hz. There are also secondary peaks in each plot that may be of interest. For the top encoder, the secondary peaks are at 1.162Hz with an amplitude 2.68% the size of the main peak. The first hinge's largest secondary peak occurs at 1.079Hz, with an amplitude 24% the size of the main peak. Hinge 2 has a secondary peak also at 1.079Hz, with a relative amplitude 32% that of the main peak. When observing the robot in motion, chaotic motion is observed. The secondary joints oscillate after the robot performs actions. If the motion was described entirely by the normal modes of the system, then these additional peaks would not exist. The presence of these higher frequencies, which is apparent when watching the robot swing, causes a loss on the system. When executing timeswing.py in Webots, live digital angular encoder data can be seen. Unfortunately, as mentioned in the section pertaining to compatibility issues, this data could not be exported, and subsequently analysed. It is, however, clear to see that the motion of the robot causes fewer losses in Webots, owing to the fact that there is minimal amplitude in the higher frequency oscillations of the joints in the swing. The lower joint's oscillations appear to have the same frequency as those of the top joint, with a slightly lower amplitude. The middle joint appears to oscillate at a higher frequency, but with considerably lower amplitude. When the amplitude of these oscillations increases, the motion becomes more chaotic. The amplitude of the swing plateaus and decreases slightly, until these small oscillations reduce in size. When the amplitude of the central joint oscillations begins to lower, the amplitude of the robot continues to increase. The excitation of these higher order modes is likely the result of the sharp corners on the ramp function controlling the robot. The sinusoidal function fed into Webots applies torque more smoothly, because the angular acceleration is smoother. Using a rounded ramp may also result in a smoother motion. This could be verified by using the numerical models to compare different periodic functions. A method of optimising the periodic function fed into the robot may come in the form of an algorithm that controls the amplitude of a function. The amplitude of the oscillations in the lower joints can be used as the

input variable for a negative feedback algorithm. The algorithm would work to reduce this amplitude, and therefore optimise the motion of the robot.

4.9 Sources of Error In Encoders

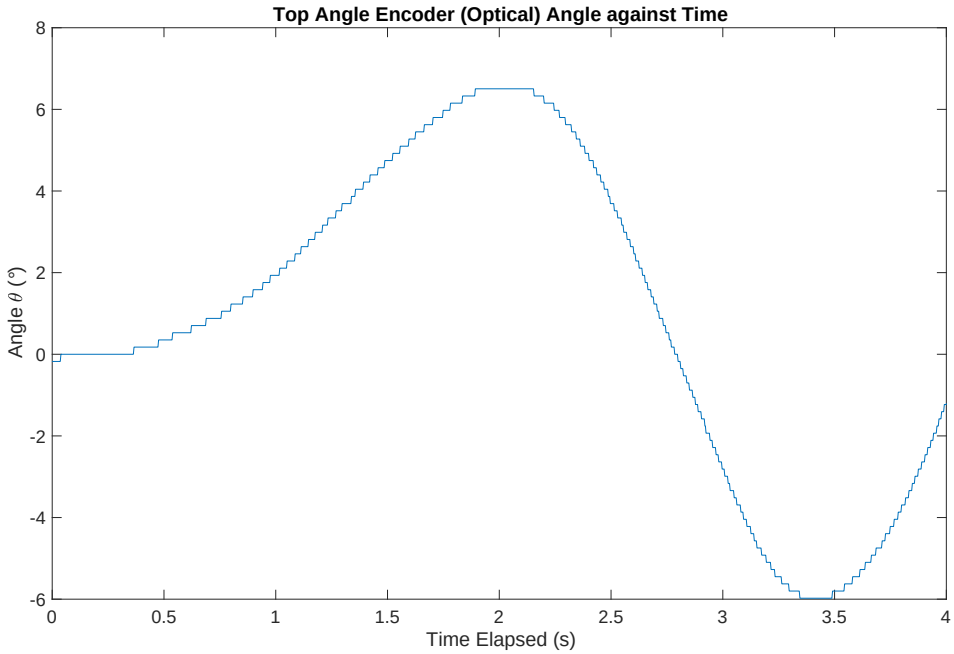


Figure 4.11: Graph showing the Stepwise Nature of the Top Encoder Data

The polling rate of the encoder is a good place to start when attempting to determine the error on the encoder. Figure 4.11 shows data from the optical angular encoder. As laboured previously, the optical encoder converts the angle into a gray code, with discrete increments. These increments are quite clear when plotting data over a short time. The data is encoded as a series of steps. These discrete steps have dimensions that relate to the error. The maximum height of the steps may be taken as the error, as this is the minimum uncertainty in the angle. This error arises due to the discrete nature by which the encoder converts what is an analogue signal into a digital signal, as detailed previously. The maximum height, as well as the average height, of the steps is 0.18°. This may lead to the assumption that the steps are all the same size. Half of this value can be taken as the error on the top encoder. This error can be applied to the top encoder angles obtained during this investigation. The script used to take this data only included encoder data. This should mean that the script is taking data from the encoder as close to its polling rate as possible. Assuming that the data is taken at the encoders polling rate, the polling rate can be calculated by taking the smallest width in the stepping function. The inverse of this will give the polling rate as a frequency. That inverse is 110Hz. This polling rate may be affected by the speed at which the scripts can run. Python is written at a higher level of abstraction, as it is a high level programming language. C is a lower level language, and therefore runs more quickly, as it interacts with the hardware more directly. If the programming language were the limiting factor, using C may help minimise the effects of a lowered polling rate. This has ramifications elsewhere, as the scripts running on the robot are more complicated, and require more functions. Comparing the data collected from the absolute encoders in section 4, and data from the same encoders during swinging motion, a less noisy dataset can be seen. Unless this noise is attributed to the motion of the robot, the reduction in noise may be a result of this reduced polling rate, caused by the fact that the scripts don't run as fast as they could in a lower level language. An error of 0.18° agrees within 5% of the error from the 2017 paper, which was related to input voltage.[9] Because the error may be affected by the script, 0.2 is accepted as the error.

5 Inertial Sensors

5.1 Introduction

Eleanor Tregoning

The development of micro-electro-mechanical system technology[24] [MEMS] has resulted in the ability to produce cheap inertial measurement units [IMU], making them available to be used in a wide variety of modern electronics; from smartphones to complex navigation systems and robotics. IMU's are most commonly made up of an accelerometer, gyro-meter and sometimes magnetometer. They can measure the acceleration, angular velocity and magnetic field surrounding a body and combine this information to feed back the location and orientation of the body. The NAO robot has an IMU which includes a 3-axis accelerometer and 2-axis gyro-meter. In addition, it also has an inbuilt algorithm *(Torso Angle)* for calculating the torso angle. The three axes of measurement for the accelerometer are shown in figure 5.1. The two axes of rotation from which the angular velocity is measured is shown in figure 5.2.

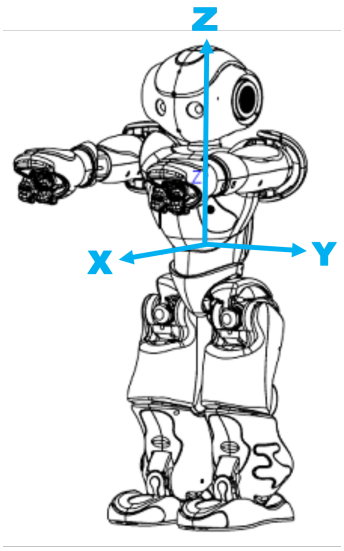


Figure 5.1: The axes of the robot for the IMU

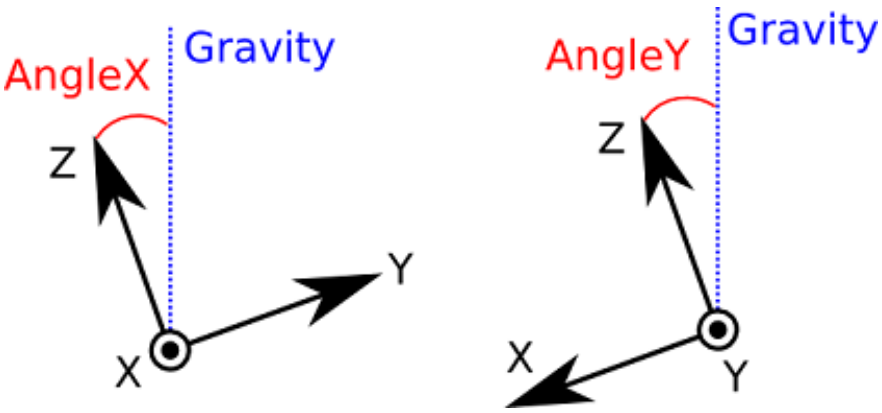


Figure 5.2: The definition of the angles subtended from the X and Y axes respectively where AngleX is the *roll* and AngleY is the *pitch*

Previous years investigations into the inertial sensors concluded that that gyro-meter results were the most promising to use as feedback and that it is very difficult to extract the swing motion from data produced with added torso motion from the hip. The fact that upper body movement and a hinged swing have been added

this year increases the number of complications with using the gyro-meter results as a form of feedback. However, the *Torso Angle* and accelerometer data has been left largely uninvestigated, therefore this year the investigation into the use of the inertial sensors has been focused around these two sets of data. To begin with, the accelerometer data was analysed and filtered to produce results that were easier to analyse and distinguish from noise. Next, in order to understand the features of the *Torso Angle* data the gyro-meter data and accelerometer data was combined using a complementary filter in order to compare the calculated torso angle with the *Torso Angle* data.

5.2 Gyro-meter

The gyro-meter is a 2 axis gyro-meter with 5% precision with an angular speed of $500°s^{-1}$.[25] It consists of a microstructure which uses a combination of one actuator and one accelerometer. A sensing element with a single driving mass is kept in a continuous oscillating movement; it is then able to react to changes in angular velocity based on the Coriolis principle. The angular rate is output as an analogue voltage. The data values are accessible from the memory key:

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

By using:

```
Device/SubDeviceList/InertialSensor/GyroscopeY/Sensor/Value
```

the extracted values will already be calibrated to $rads^{-1}$. Hence the resulting plot of the gyro-meter data was easy to extract and is shown in figure 5.3.

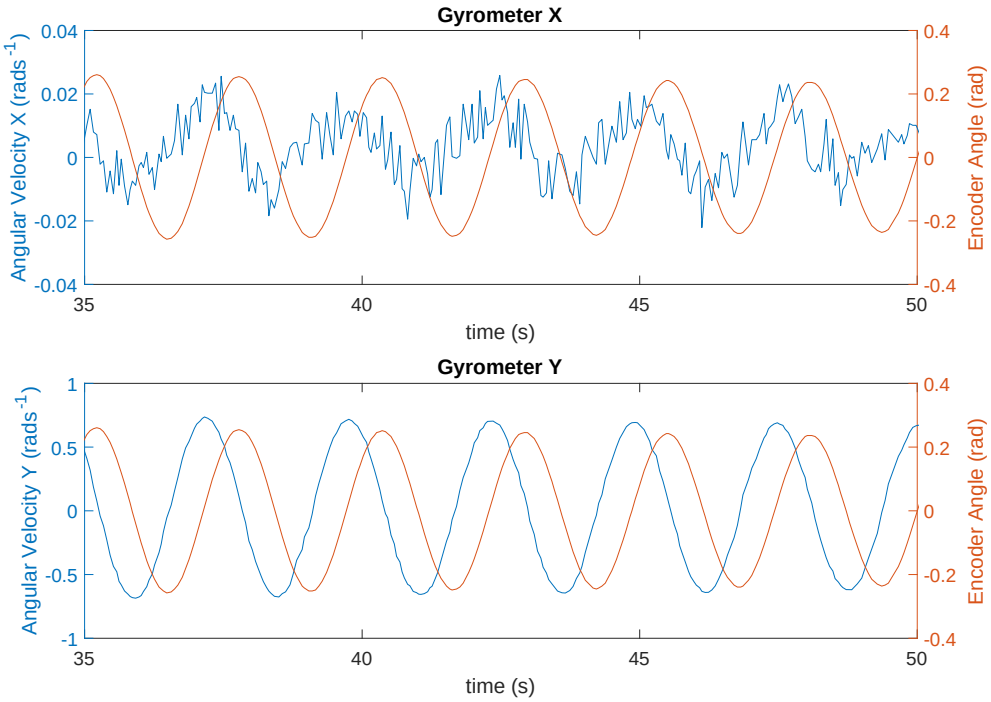


Figure 5.3: Gyro-meter data for swinging motion with no torso movement or driving motion

Figure 5.3 shows that there is some periodicity present in the gyro-meter X data, suggesting there is some sideways motion of the swing that is periodic and contributing to noise. The gyro-meter data for the y angle

is very clear over the 5 oscillations shown with very little noise present. It is $\frac{\pi}{2}$ out of phase with the angle encoder as one would expect.

5.3 Accelerometer

The accelerometer is a 3-axis accelerometer with a 1% precision at 2g.[26] Accelerometers are a great example of MEMS technology. They can measure both dynamic and stationary acceleration due to gravity. Most commonly, accelerometers use capacitance plates to measure acceleration. These plates are attached to small springs so are allowed to move. When undergoing any form of motion, the capacitance between the plates will change. This can be converted and then output as a voltage. Other forms of accelerometers use piezoelectric crystals which give out a voltage when placed under mechanical stress. Accelerometers with a digital interface use either SPI of I^2C communication protocols. The advantage of this over analogue accelerometers is that digital accelerometers are less susceptible to noise.[27]

The accelerometer data shown in figure 5.4 is taken with the robot in a fixed position with an initial angle of about 20° and with no driving motion. It shows the acceleration of the torso with respect to the perpendicular, in this case the torso is roughly in line with the swing frame.

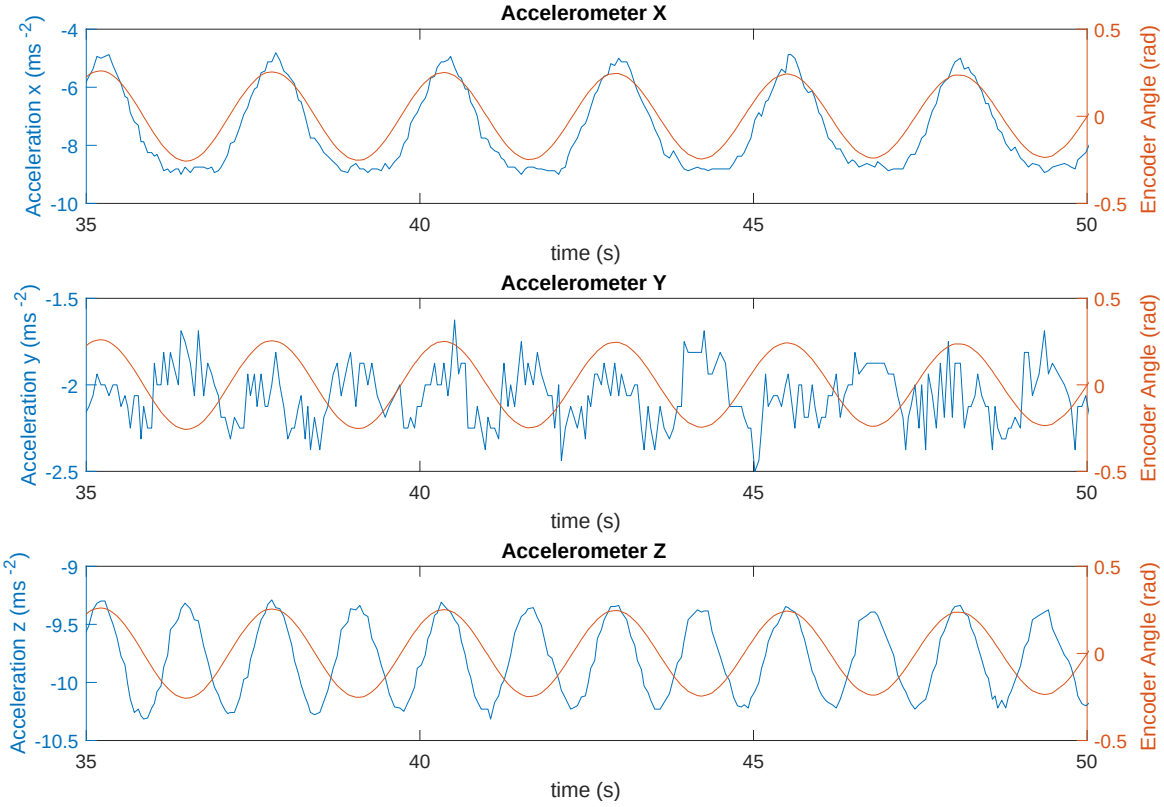


Figure 5.4: The acceleration in the x, y and z directions (raw calibrated data)

As one can see there is significantly more noise present in this data compared to the gyro-meter data. The accelerometer z data seems to have half the period of the angle encoders. This is also present in acceleration x when the torso is not in line with the swing. It is possible to see that, in this case, the torso is not perfectly in line with the swing from the small gaps between peaks where smaller secondary peaks are present underneath the noise.

5.3.1 Filtering the data

In order to use this data and be able to see it clearly, it is necessary to use some form of filtering. A Fourier filter is a filtering function which focuses and manipulates certain frequency components of a chosen signal. It takes the Fourier transform of the signal - converting it into frequency space. In this space certain frequencies (such as those due to noise effects) can be reduced. An inverse Fourier transform is then used to transform the result. One can use a cut-off to cut out frequencies above or below a certain limit, thus creating a low or high pass filter. In this case the MATLAB function FouFilter.m[28] was used. It is a very flexible filter that can be used as a low-pass, high-pass, band-pass or band-reject filter with variable cut off-rate.[29] In order to filter the accelerometer data, a band-pass filter was used with a frequency width of 100Hz and centre frequency of 10Hz. The results are shown in figure 5.5.

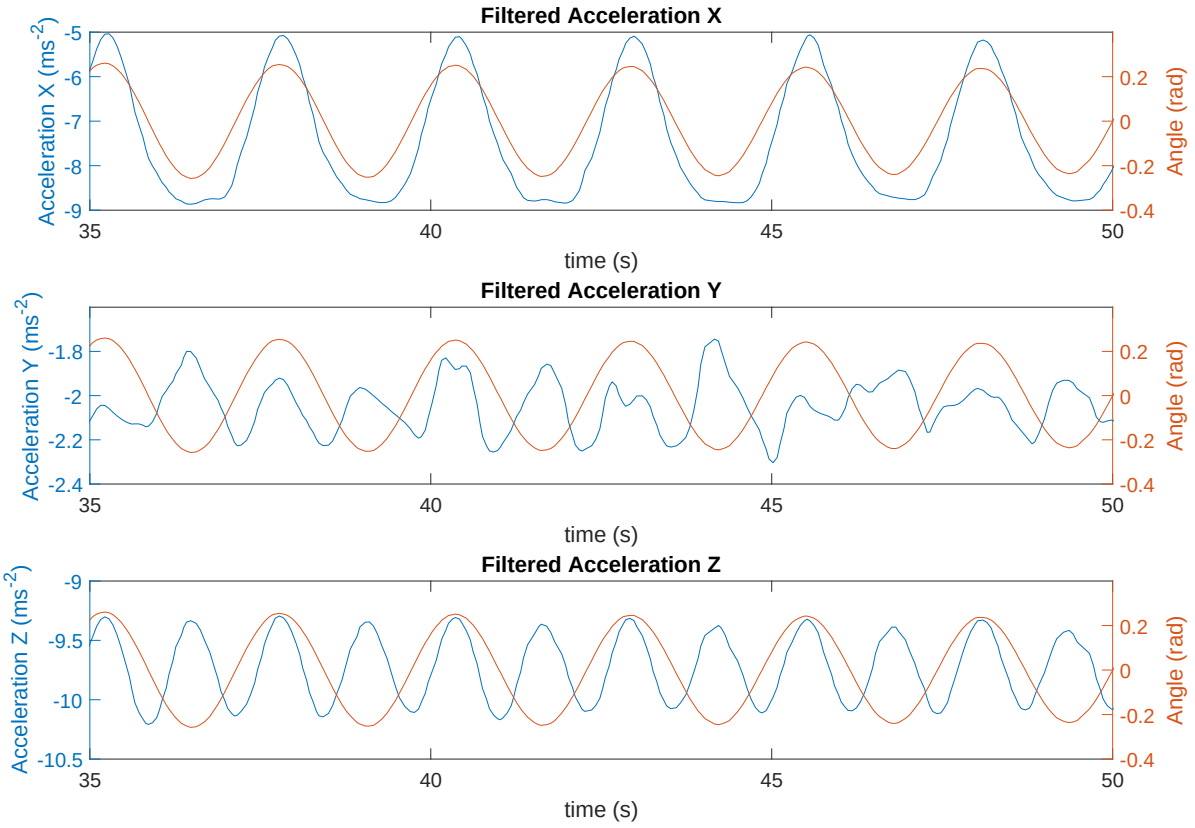


Figure 5.5: The filtered accelerometer data

5.3.2 Conclusions on the Accelerometer filtering

Without the noise it is easier to see the small secondary peaks where the torso angle forms a small angle with the swing rod frame. The accelerometer y data looks almost as messy as before which is what would be expected. However, the periodic nature of the data becomes more apparent with the filter applied. This is in keeping with what can be seen in the gyro-meter x data, therefore it can be said that the movement in the y direction, although small, is periodic with rest of the data. This motion could be due to an anti-symmetric distribution of mass about the y axis. The robot itself is slightly heavier on one side than the other and there may be small discrepancies in the mass on either side of the swing due to its design. Minimizing this motion would reduce the amount of energy being wasted in the sideways direction and by removing it could lead to a more efficient swing. This is something that could be explored further in the future. Although in the past it was noticed that the gyro-meter data would be the best candidate for position feedback,

filtering the accelerometer data vastly improves its usefulness and potential for position feedback. The one downfall in using a filter is that it takes more time to process so may be inaccurate when taking real time measurements.

5.4 Torso Angle Algorithm

As mentioned in the introduction to this section the NAO robot has an inbuilt *Torso Angle* algorithm. According to the Aldebaran documentation this computes the angle of the torso from the accelerometer and gyro-meter data. However, there are sharp peaks occurring periodically in the otherwise sinusoidal *Torso Angle* Y data as can be seen in figure 5.6. There is also a slight lag between the *Torso Angle* and encoder angle, possibly due to the processing time of the algorithm. In order to investigate the source of these peaks it is necessary to look at the methods used to combine the data. Since the algorithm is owned by Aldebaran Robotics, it is not possible to access the details of the code. However, instead it is possible to look at techniques for combining the inertial sensor data using a different method and compare the results to that of the *Torso Angle*.

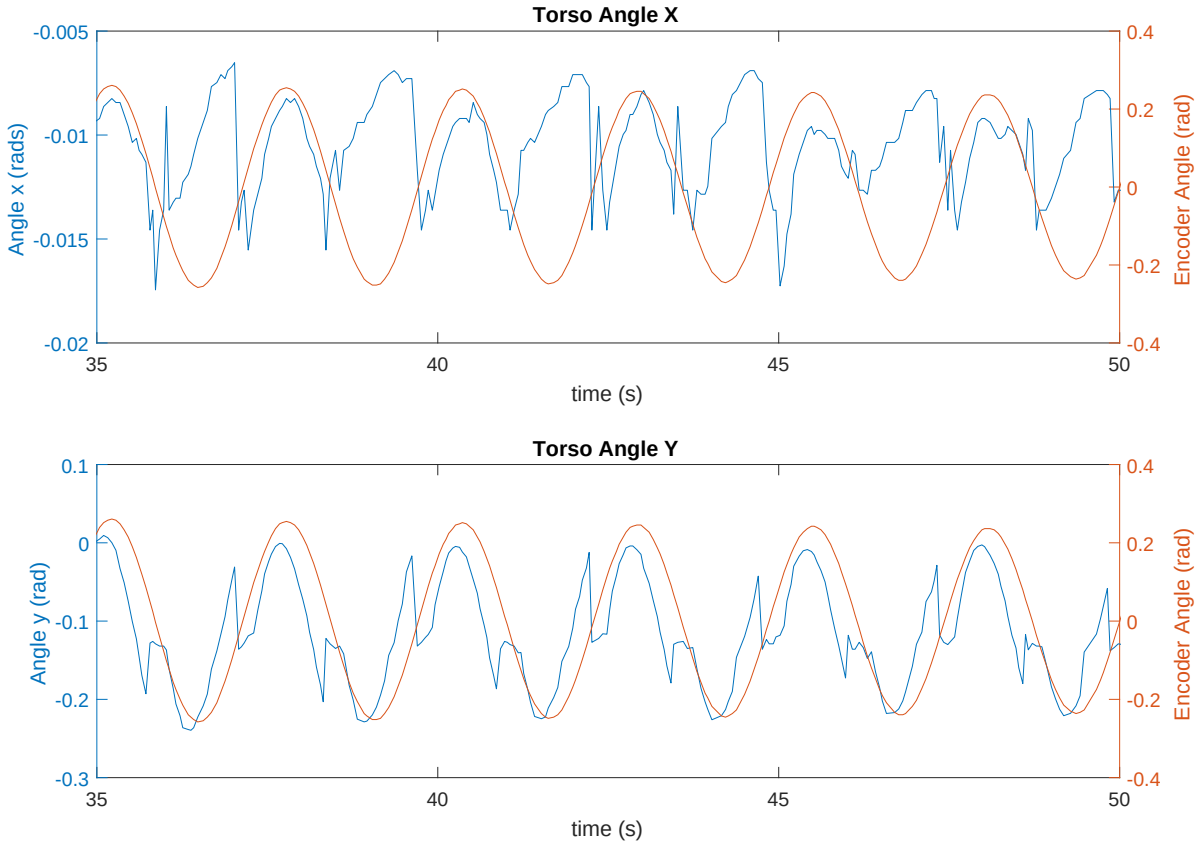


Figure 5.6: The *Torso Angle* for AngleX and AngleY defined in figure 5.2

5.5 The Complementary Filter

There are two main ways to combine gyro-meter and accelerometer data; the Kalman filter and the complementary filter. The complementary filter is much easier to use and understand and it was decided, therefore, to use this filter on the sensor data. Firstly it was necessary to calibrate the data and calculate the angle. The angle can be calculated from the acceleration simply using equations 5.1 and 5.2.

$$Pitch = AngleY = \arctan\left(\frac{A_x}{\sqrt{(A_y)^2 + (A_z)^2}}\right) \tag{5.1}$$

$$Roll = AngleX = \arctan\left(\frac{A_y}{\sqrt{(A_x)^2 + (A_z)^2}}\right) \tag{5.2}$$

Similarly, the angle can be calculated from the angular velocity simply by integration. However, there are disadvantages to each approach. Firstly, the accelerometer is much better at taking accurate measurements when stationary and hence, is very susceptible to noise during motion. Secondly, simply integrating the gyro-meter data is not ideal as the integration is done over time causing the measurement to drift away from the zero equilibrium point.[30] The combination of these two advantages is achieved by using this filter. Since the gyro-meter data is reliable in the first few seconds and not susceptible to external forces or noise it is used in the short term. On the long term the accelerometer data as used as there is no drift. Essentially the complementary filter algorithm applies a high-pass filter to the gyro-meter data and a low-pass filter to the accelerometer data then combines the two as shown in 5.7.

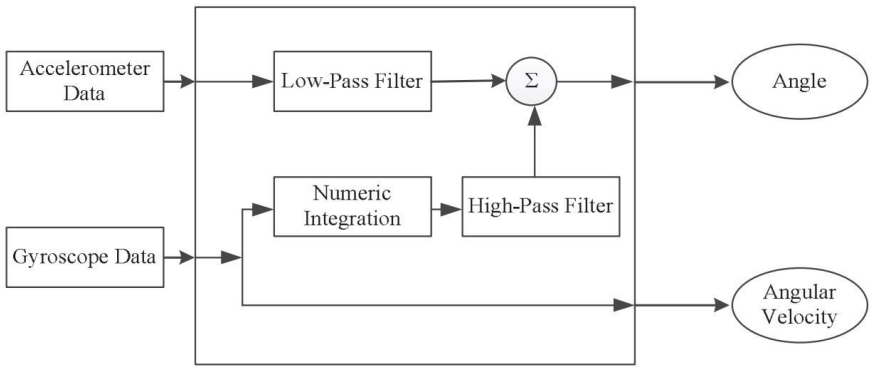


Figure 5.7: The Complementary Filter Algorithm[31]

The following equation is the main complementary filter equation where `rgy` is the angle from the integrated gyro-meter data, `AY` is the angle from the accelerometer data and `k` is a constant parameter between 0 and 1.

```
angle=k*(angle+(rgy))+((1-k)*AY);
```

the constant k can be changed to prioritize either the accelerometer angle data or the gyro-meter angle data.

Figure 5.8 shows the angle as calculated by the accelerometer, gyro-meter and angle encoder. The angle as calculated using the complementary filter is shown in blue. Figure 5.8 shows a significant amount of drift in the gyro-meter data and there is significantly more noise in the accelerometer data. The parameters on this filter are set to prioritize the accelerometer data, consequently, there is still noise present in the filtered data. However, the filtered data does show a better fit to the encoder data.

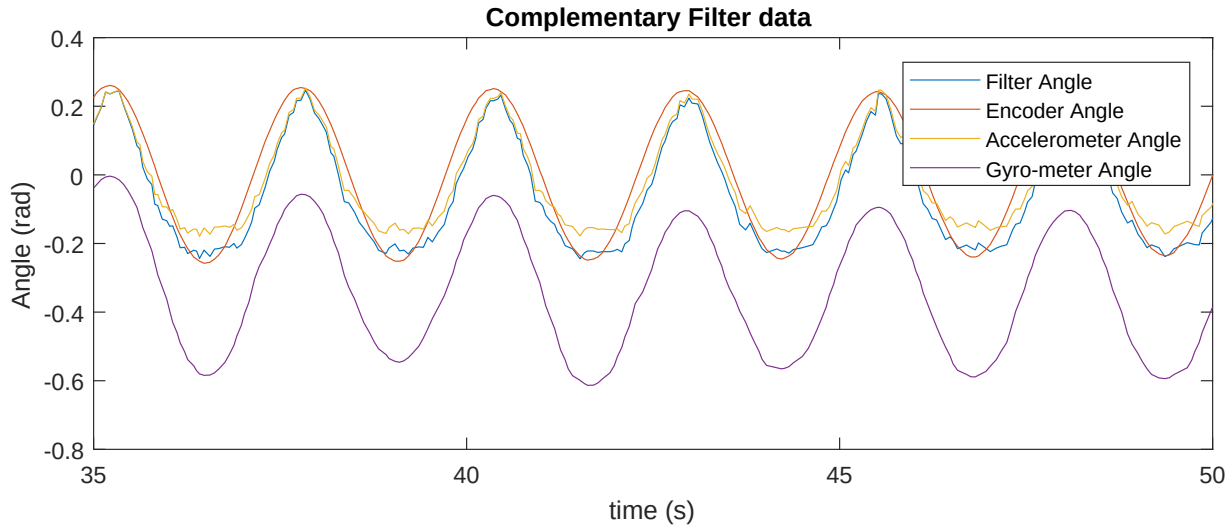


Figure 5.8: Complementary filter angle compared to the angle calculated from the accelerometer, gyro-meter and angle encoder

5.5.1 Conclusions on the Complementary Filter

The most important feature shown in figure 5.8 is the lack of the peaks seen in the *Torso Angle* data. It demonstrates that the source of these peaks is not due to any feature present in the accelerometer or gyro-meter data. This supports the theory made in previous years that the peaks are due to the algorithm swapping between two methods of calculating the angle. It is clear that the algorithm was not designed for use during a swinging motion and it can therefore be concluded that the *Torso Angle* is not useful in providing and accurate measurement of the angle. The complementary filter presents a much better measurement of angle despite the slight noise and drift still present in the filtered data.

5.6 Evaluation

Overall it was found that much of the data can be made clearer by the use of filters and that the *Torso Angle* algorithm is not useful for determining the angle. This filtered data could prove to be more useful in terms of feedback. By using the acceleration data instead of the gyro-meter data for feedback it may be easier to separate the torso motion form the swing motion. In addition the filters could be applied to the data representing the sideways motion of the swing in order to look at where energy is lost to this motion. This could also tie in with looking at the new encoder data. The encoders on each of the hinges could be used to find discrepancies in the angles recorded on each side of the swing. In total the use of these sensors is now in a better position to be applied to increasing the efficiency of the swing through analysis and minimization of the sideways motion.

6 Using Webots

6.1 Introduction

Ronan Dubois

Throughout this project, a considerable number of software-related issues were encountered and it was realised that many of these specific problems could have been avoided and that a considerable amount of time could have been saved had the previous reports given a more comprehensive and explicit description of the software. Therefore, this section will aim to present a detailed yet concise guide of modelling in Webots.

6.1.1 Installing the software

First and foremost, it should be noted that compatibility between the software is an important concern. In this project, the versions used were an older version of Webots (8.5.3) and the newest version of Choregraphe (2.1.4.). It is also possible to use the latest version of Webots, R2018a with Choregraphe 2.1.4. Currently, the versions of Webots (8.0) and Choregraphe (1.14) installed on the university computers are not mutually compatible and do not allow Webots to be connected to Choregraphe. It is therefore advised that the user install the software on their own computer to circumvent these issues. Furthermore, to connect the real NAO to Choregraphe, version 1.14 is required but fortunately it is possible to install more than one version of Choregraphe on a single computer.

- For Choregraphe, the user should follow this link: *https://community.ald.softbankrobotics.com/en/--resources/software/language/en-gb* and download the appropriate version of Choregraphe (in section *2-Choregraphe Suite*). The user will first need to create a free account using their university email.
- For Webots, the user should go to *https://www.cyberbotics.com/download* to install the latest version (R2018a), or use the archive at *https://www.cyberbotics.com/archive/index.php* to install an older version, such as Webots 8.5.3.

Although a university-owned licence is available for Choregraphe, the Webots licence is no longer useable and one must therefore obtain a free trial version from Cyberbotics. However, it should be kept in mind that the trial licence only lasts for 30 days, and that the licence is linked to the computer's IP address which prevents the use of multiple accounts on the same computer. During the project, several computers had to be used successively which proved to be a major issue.

6.1.2 Connecting to Choregraphe

Connecting Webots's virtual NAO to Choregraphe is a good way to both familiarise oneself with both software and to easily change the position of the Webots robot.

In order to connect a virtual robot to Choregraphe, a Webots (.wbt) world containing a virtual NAO was first launched. Any of the swing models created in previous projects could be used as well as pre-built Webots worlds, such as the nao.wbt world (*Webots/projects/robots/aldebaran/world/nao.wbt*) which consists of a virtual NAO in an empty environment. To connect to Choregraphe, it was necessary to use *naoqism* as the virtual NAO's controller program. Once the world file was open, Choregraphe was launched and the *Connect to...* button in the top left corner of the screen was selected. A connection window with a virtual robot then appeared, displaying the robot's corresponding port (9559) and virtual IP address (127.0.0.1). Note that the port and IP address are different than those used when connecting to the real robot. One can check that the port is correct by checking the value of the virtual robot's *controllerArgs* field in Webots. If the connection to Choregraphe failed, both the Webots world and Choregraphe were reloaded and the above steps were repeated. In some cases, restarting the computer was necessary, as processes running in the background sometimes seemed to prevent a connection.

6.1.3 Using Choregraphe

Once the virtual robot was connected to Choregraphe, a 'clone' of the Webots robot appeared at the bottom of the screen in Choregraphe. The *Robot View* tab was selected, as opposed to *Robot Applications*, to visualise the virtual NAO. The latter could now be controlled. An easy way to make sure that the robot was connected was to check its colour, which would change from orange to red when the connection was successful. Note that connecting to Choregraphe was done using Windows operating systems. Using a Mac computer, a connection to Choregraphe could be established but for an unknown reason the robot would not respond to any commands, although this particular issue was not encountered by last year's group.

Once a connection was established, some experimentation was carried out, both using Choregraphe's pre-defined motions and animations and changing the robot's joint angles manually. In order to move the NAO's joints, the body part to be moved was selected (the moveable parts are the head and limbs) and the *Stiffen chain on/off* box was toggled. In addition, the *mirroring* box was ticked if a symmetric movement of the limbs (e.g. raising both arms at the same time) was required.

If the Webots simulation was running, the virtual robot would then replicate the corresponding motion exactly. If the robot did not appear to move, it was verified whether the *Wake Up* 'sun' symbol in the top right corner was highlighted, and whether the *Do not update robot position* button above the robot view was not selected.

6.2 Modelling in Webots

6.2.1 Introduction

Webots is a simulation software developed by Cyberbotics. It allows the user to build a virtual environment using a wide variety of tools and contains a large collection of virtual robots, including Aldebaran's NAO, which can be made to interact with their virtual environments. Furthermore, it allows the user to run scripts on the virtual robot both from outside of Webots and inside Webots using controller programs in one of the supported computing languages, which include Python and C++.[32]

Simulating swinging on a virtual robot is advantageous for several reasons. It enables the user to test a large variety of models and scripts instantaneously, allowing quick identification of the scripts which are worth testing on the real robot. Furthermore, it bypasses several pragmatic considerations such as possible damage to the robot, overheating of the motors and using up of useful battery life.

When Webots was first launched, an error message appeared warning that the graphics were below the minimal system requirements, but this did not limit our ability to run simulations. However, a few tricks had to be used to increase the speed of simulations, which was found to be too low. The frame rate of the simulation was changed from its default value of 20 to a lower value (by changing the *FPS* field in the *WorldInfo* node) to smoothen the motion. The simulation was also be sped up by disabling the virtual robot's camera, by adding the *-nocam* argument in front of the robot's port.

Note that in instances when a world file created using an older version of Webots was loaded, the first line of the code had to altered. This was done simply by opening the world file using the Notepad and modifying the top line (e.g. *#VRML_SIM V8.5.3 utf8* for Webots 8.5.3) to that displayed in the error message of the Webots console.

In Webots, one works in a virtual environment called a world (.wbt file). In this project, the world consisted of a virtual 3D environment, a virtual NAO and a swing which it could interact with.

A Webots virtual environment is organised in a tree-like structure of nodes which define the properties of the virtual elements within it. In the scene tree side bar of the swing model, the following nodes may be found:

- *WorldInfo*: contains general information about the simulated world

- *ViewPoint*: contains a set of coordinates defining the orientation of the user's point of view
- *TexturedBackground*, *TexturedBackgroundLight*, *PointLight* and *Floor*: define the appearance and aesthetics of the virtual environment
- *NAO*: the virtual robot node, which contains the port and controller fields, as well as various slots to attach connector nodes
- *Swing*: the swing model, which is actually a *Robot* node. This is required to enable the use of a controller script throughout simulations, as was done in previous years.[9]

However, only the last two nodes were of interest in this project. Each object in Webots is formed of parent and children nodes. The parent node characterises are the fundamental properties of the object, whilst its secondary attributes are defined by its children. A simple object might thus be created by defining a *Solid* node (which can be translated and rotated) with a *Shape* child node, which is used to define the material, appearance and geometry of the object. It is also important to assign a *boundingObject* node corresponding to the geometry of the object, as well as *Physics* node, to each *Solid* so that it can be interacted with.

As should become apparent in the next section, the swing model is more complex than this, due to the modelling of joints and other components which all have to move in a cohesive way.

6.2.2 The swing model

The new swing model was motivated by both the flaws in previous models and by the need to connect the virtual robot's hands to the swing handle to enable upper body motion. Last year's group created a double-hinged swing model (Chainbarswing.wbt) in case chain swing motion was investigated.[9] However, it quickly became apparent in this project that the latter model would have to be modified because of several major errors.

First of all, connectors were used to attach different parts of the swing together. This was thought to be necessary because a node cannot have more than one parent and thus the handle arm needed to be fixed to the other side of the swing.[9] However, connectors should only be used to connect the NAO to the swing. This bad use of connectors prevented *boundingObject* and *Physics* nodes from being added to the handle, thus rendering any interaction with the robot impossible and hindering the realism of the model by negating the influence of the handle's mass.

In addition, the robot's initial position on the swing could not be changed because of a solid top bar which was added to secure the robot onto the seat. Furthermore, removing the top bar would cause the swing to break along its joints and therefore the model had to be recreated entirely. Leaving this non-negligible mass on the seat would also certainly hinder the accuracy of the model by altering the period of the swing.

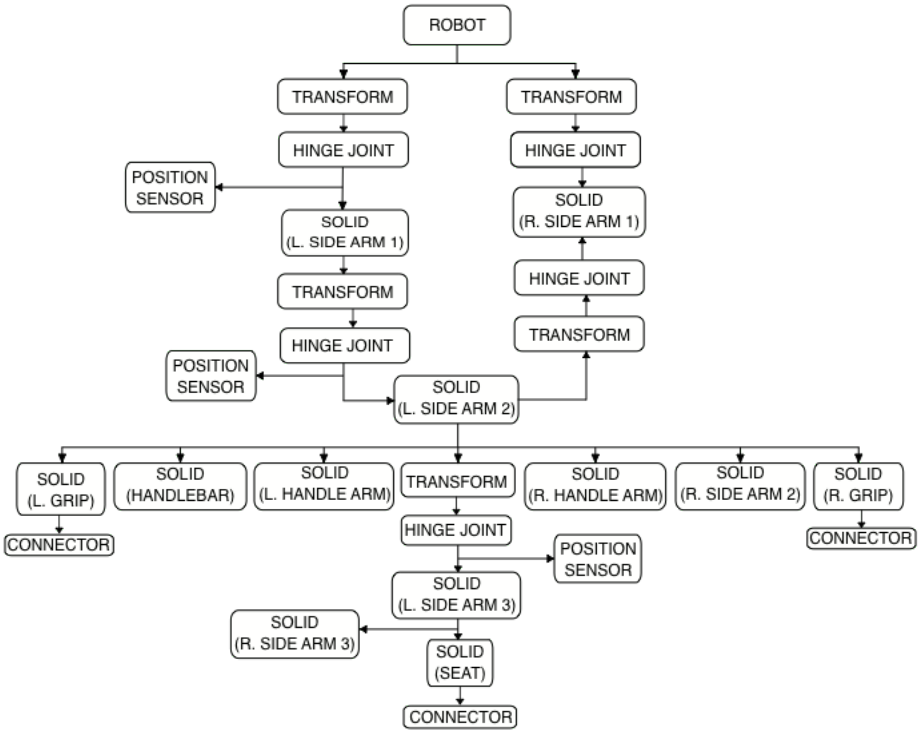


Figure 6.1: Simplified scene tree of the new swing model

The swing model consists of two arms. A top bar shape was actually used in previous models as the *boundingObject* for the swing but removing this was found not to affect the model in any way. The right and left arms of the swing stem from *Transform* nodes. A *Transform* allows the user to define a system of coordinates which is relative to the coordinate system of the parent.[33] This simplifies the coordinates of children nodes which lie along an axis of symmetry of the parent, such as hinge joints. The left arm *Transform* node was arbitrarily chosen as the parent for most of the swing elements, as can be seen in the scene tree diagram (Figure 6.1). It connects via a regular hinge joint to the left side arm 1 - the main arm of the swing - which itself connects to the left side arm 2. The same damping constant (0.1 Nm^{-1}) as calculated last year was included in the hinge joint parameters.[9] Position sensors were also added to all hinges to mimic the real angle encoders.

At this point, the scene tree branches off into multiple children. These include two *Transform* nodes and six *Solid* nodes: the handle bar, the handle side arms, the hands grips and the right side arm 2. Note that each of these have respective *Shape* nodes, omitted from the diagram for the sake of clarity. Together, the *Solid* nodes form a mechanical loop corresponding to the handle. There was no need to connect them as children nodes of a common parent will automatically connect. However, the mechanical loop had to be closed by connecting the right side arm 2 to the right side arm 1. This was done through one of the *Transform* nodes, using a hinge joint with a *SolidReference* end point, allowing the fixation of the handle to the right side arm 1, defined earlier in the scene tree. The other *Transform* node was used to connect the left side arms 2 and 3 with a regular hinge joint (the rotation axis direction of the hinge joint had to be changed from the default to prevent a stiffness of the hinge).

The left side arm 3 is a parent of both the seat and the right side arm 3, forming a second mechanical loop. Another hinge joint with a *SolidReference* end point was initially used to close this loop but this made the whole swing rigid and the issue was eventually resolved by removing the hinge joint altogether. This was likely due to the fact that Webots automatically merged those elements with the right side arm 2. The completed model, shown in Figure 6.2, was used in all following Webots simulations (see Section 7).

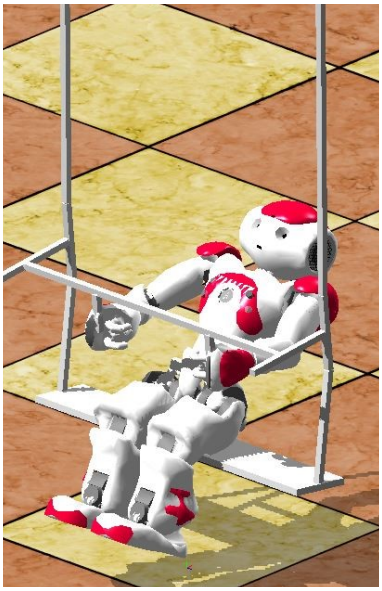


Figure 6.2: New Webots swing model with seated NAO

6.2.3 Connecting the virtual robot to the swing

The robot needed to be placed on the swing in a realistic seated position in order to make the simulation as accurate as possible. Table 6.1 below shows the values of the joint angles which were used in Choregraphe to seat the virtual NAO on the swing. These angles were found by placing the real NAO on the swing, connecting it to Choregraphe via Ethernet and reading off the joint angles directly.

Table 6.1: Arm and leg joint angles used to seat the virtual NAO on the swing

Joint angle	Value (°)
Ankle Pitch	-3.8
Ankle Roll	-0.7
Knee Pitch	69.7
Hip Pitch	-41.0
Hip Yaw Pitch	0.0
Hip Roll	2.3
Elbow Roll	58.6
Elbow Yaw	58.2
Shoulder Pitch	65.5
Shoulder Roll	-17.3
Hand	0.42
Wrist Yaw	41.7

To change the position of the virtual robot, the Webots simulation was paused and the robot moved off the swing and placed on the floor. Controlling the robot was thus facilitated as the latter would not be disturbed by the swing's motion. Once the robot was placed in the desired position, it was connected to the swing using connector nodes. A pair of connectors was required to attach one part of the robot to a part of the swing.[34] In this case, connectors were assigned to each of the handle grips and peer connectors were added to the NAO's left and right *handSlot* nodes.

The *distanceTolerance* field of the connectors, which determines the maximum distance between the connectors until they become separated, was increased to 0.1 m to prevent them from disconnecting. The other connector fields which were modified were *autoLock*, which automatically creates a locked connection if a peer connector is detected, and *isLocked*, which defines the locking state of the connectors. The *snap* field, which enables connectors to align/adjust during the simulation was set to FALSE, as some erratic behaviour was observed in the opposite case.

At first, it was observed that a connection could only be established between the right hand and grip, whilst the left hand disconnected from its grip throughout swinging simulations. This was found to be due to the orientation of the connectors. Indeed, the connector axes should face in opposite directions for the connectors to lock together.[34] The issue was therefore solved by rotating the left grip connector by 180° along the y axis. The robot was also connected to the seat, using its *bodySlot* node, to prevent it from moving or sliding on the seat. Again, the orientation of the connectors was changed to make locking possible.

6.3 Possible improvements

Although the overall structure of the model has been greatly improved and has shown to yield impressive results, some minor amendments could still be made to improve the accuracy of the model:

- For simplicity, the seat was modelled as a single solid, as was done in all previous models. However, in reality, it consists of two parts - a solid metal bar and a plastic seat. The location of the centre of mass of the swing could be made more accurate by modelling the seat more realistically

- The mass of the handle was not measured, so the masses of the handle elements had to be estimated by assuming the same density as the side arms (whilst the density of the hand grips was taken to be that of rubber). Measuring these masses could be done to increase the model's accuracy

- The friction coefficients of the hinge joints could also be measured and included in the model.

- The real side arms of the swing are cylindrical bars, but were modelled as rectangular bars of equal width for simplicity. While this is not expected to make a significant difference to the model, it could be tested if time permits.

7 Controlling the robot

7.1 Working With Choregraphe

Arthur Mazendame

Choregraphe is a piece of software developed by Aldebaran allowing control and communication with their products. It allows investigators to send basic commands to the robot and learn about some of its limitations. These limitations include the ranges of each degree of freedom the robot's joints have. This information is useful when programming movement for the robot as it allows investigators to operate within safe parameters. Choregraphe 1.14.5 is required to control the real robot (NAO V4). The program allows users to execute predetermined actions called behaviours. These behaviours can be performed by a virtual robot in Choregraphe, or by the physical robot connected to Choregraphe. The timeline function in Choregraphe allows for the saving of a total body position. This position is set for a given frame in the timeline and other positions can be added for different points in the timeline. This function gives users information about the joint angles of the physical robot (that has been manually put in position). The timeline, when executed, moves the robot between these saved positions at a constant velocity, which is determined by how close they are to each other in the timeline. This was the first method, by which the robot was controlled, and a primitive one at that. These timelines can be exported as Python scripts, allowing for the joint angles to be recorded and used in subsequent attempts to simulate swinging. Importing a Python script (such as an edited version of the ones from the timelines), is one way of tuning the script to give the robot's motion the same period as the swing. Choregraphe had its limitations. These included an inability for any feedback mechanism to be employed, because there was no way of using the angular encoders to give the robot information about its angular position in Choregraphe.

To overcome these limitations both in Webots and reality the robot is communicated with directly via Python scripts. This can only be achieved with the use of Python 2.7 SDK 1.14.5(for the real robot) and SDK 2.1.4 for the robot in Webots 8.

7.2 SwingAPI

The first port of call, when communicating with the real robot, was building on the work of previous years. The Swing API script, developed in 2017, includes positions that allow investigators to define motions. It only includes hip (torso) and knee (lower leg) motion, which did not utilise the forces, and therefore additional torque, that may be produced by using extensors and flexors of the robot's arm, which would be attached by the hands to a handle on the swing. Because the aim, this year, was to more closely mimic the motion associated with a rope swing, using the arms is quite necessary. A full treatment of the upper body motion was obtained by using Choregraphe to obtain the joint angles associated with both extreme positions taken by the robot during swinging motion. These angles were mirrored in Choregraphe to ensure lateral movement of the swing was minimised and that the swinging motion was as efficient as possible. Altering the swing API involved adding additional joints from the upper body, along with changing values for the lower body. The script also includes terms for each angle which define their velocities such that the positions are reached simultaneously, again maximising the efficiency of the motion.

The functions in this swingAPI.py script are called upon in the scripts that communicate with the robot. One such script, swingLegs.py, is a script that makes the robot change between extreme positions of the robot periodically. The robot's joints move with constant velocities, so the periodic functions controlling them can be thought of as ramp functions. They all have the same periodicity, but different velocities, as the joint ranges are not all the same.

7.3 Full Body Motion

Another script that was written to control the robot using predetermined periodic functions was timeswing. This varies the angle of the joints with a sine function. The amplitudes of each sine for each joint are defined as half the range, with an offset that means the function oscillates between minimum and maximum angles for given joint. The frequency of all sine waves is the same. The timeswing.py script produced an amplitude of as much as $37^\circ \pm 0.3^\circ$ using a swing with both of the lower joints free. In Webots, the script allows the robot to swing smoothly and, while the robots motion is in phase with the swing's motion, gain a substantial amplitude. This phase, however, is a limiting factor in the robot's ability to continue gaining amplitude. It is apparent that the period of the swing isn't quite constant, because the phase of the robot's actions and the swing's oscillations changes over time. The robot's actions begin to have a detrimental effect on the amplitude, as this phase changes. This is because the torque is being applied at sub optimal times during the swinging motion. Eventually, when the phase is π radians, the robot's actions cause a torque opposing the motion of the swing.

In reality, timeswing.py didn't run smoothly on the robot. The motion of the robot was incredibly convulsive and would cause the motors to heat up incredibly fast. This jerky motion may be a result of the robot's inability to respond to the sinusoidal input continuously, instead taking joint angles at whatever the polling rate of the robot's processor is. This would result in a jerky motion, between each angle polled by the robot, instead of a smooth motion, as seen previously with positionswing.py. This is a problem not limited to sinusoidal inputs. Any complicated periodic input defined this way (as a function in the script), is likely to cause jerky motion and potentially damage the robot.

7.4 Compatibility Issues

Ryan Bouab

This project was plagued with compatibility issues that were a major hindrance to the progression of the subgroup for about four weeks; hence the need to address these issues so that future groups do not encounter the same problems and can focus on making new contributions to the field of robotics. The main issues arose from the use of the Python SDK NAOqi and the different CPU versions of Python 2.7.13 on Windows. The SDK allows one to create Python modules that can run remotely or on the robot NAO.[35] The SDK utilises the NAOqi framework to run scripts and control the robot. The module used to control the robot's movements and position is the ALMotion module containing set functions that control the robot's joints to move it accordingly. This module is controlled by a proxy called ALProxy which contains ALMotion's available functions. The following Python command was used in every script to connect to the robot NAO:

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

The robot's IP address for the real robot NAO is 192.163.1.3, the simulated one is 127.0.0.1 and the port number is 9559 for both real and simulated robots.

- In order to run scripts with a Windows computer on the virtual robot, one needs to download the Python 2.7 SDK 2.1.4 Win 32 Setup from *https://community.ald.softbankrobotics.com/en/resources/—-software/language/en-gb* and follow the proper installation procedure outlined on the Aldebaran documentation website *http://doc.aldebaran.com/2-1/dev/python/install_guide.html*
- One should also download the required Python 2.7.13 32-bit from *https://www.python.org/downloads/—-release/python-2713/* (MSI installer).

Installing these two software would allow one to connect to the virtual robot in Webots. However, Webots is an environment that operates a 64-bit Python 2.7; and since the only SDK available for Windows is a 32-bit version, one requires Python 2.7 32-bit to connect to the robot. This compatibility issue renders one unable to connect to the Webots environment whilst simultaneously running a simulation on the robot. This issue considerably limits the scope of simulations on Webots using NAO. This problem is seemingly solved by using an Apple Macintosh computer. Indeed Mac platforms boast a 64-bit Python NAOqi SDK allowing one to simultaneously connect to the virtual robot and control its environment. In fact Harry Withers from the 2017

group[9] used a Mac which explains why he only encountered minor compatibility issues. Unfortunately, this was found at an advanced stage in the project hence why no simulation acting on the Webots environment was conducted. These compatibility issues on Windows were confirmed by the Cyberbotics team making it clear that Aldebaran's Python NAOqi SDK was not being developed anymore on Windows and therefore the team was considering removing it from Webots. Another compatibility issue was connecting to the real robot using a personal laptop. This was not possible using Choregraphe 2.1.4 and Python NAOqi SDK 2.1.4 since the robot's internal software is Choregraphe 1.14.5. Instead of having to uninstall software on one's personal computer, the robotics lab already possesses Linux computers with pre-installed Choregraphe 1.14.5 and its corresponding Python SDK 1.14.5. The conclusion of this compatibility investigation is that to simulate swinging motion on the virtual robot, connect to it and use its environment to drive its motion one should use a Mac for reasons outlined earlier. To connect to the real robot and use the angle encoders, using the Linux computers will be sufficient.

7.5 Dynamic Position Feedback

Given the success of the dynamic period model coded by Harry Withers last year[9] to drive the robot, it was clear that it had to be modified and improved to drive the robot's motion on the double-hinged swing. The script is able to measure the current angle of the swing using either the real or virtual swing angle encoder. However, only the real encoder could be used since the virtual encoder required to run a controller script on Webots which is incompatible with the 32-bit Python 2.7.13 used to run the SDK NAOqi for aforementioned reasons. The crucial feature of this script is its ability to determine when the robot is at the bottom of the swing, this point defined the zero angle. However, the encoder did not read out an angle of 0° at this point, thus an initial angle was read at the start of the script in order to calibrate the subsequent angles read. At this point a while loop was started in which the current angle was measured. The script then determined the position of the robot as a function of the swing angle by comparing the previous angle (initially set to 0°) and the current angle. By calculating the product of these two angles, one can see that if it is equal or smaller than zero then it means the robot has just passed the bottom of the swinging motion. When this condition was met, a timer was started allowing the robot to then calculate the current period of the swing. The timer was started with a delay of 0.1 seconds[9] corresponding to the delay between the robot processing tasks sent to it and performing said tasks. In order to change positions at prescribed intervals, the period needed to be known accurately for every oscillation. To do so an initial period of the system had to be fed to the program, this period was provided by the numerical modelling subgroup as $T_0 = 2.5661$ seconds. In order to compute the period for every oscillation a moving average method was calculated using equation 7.1.

$$\sum_{n=1}^{N}(t + (n-1)\frac{T_{n-1}}{n}) \tag{7.1}$$

Where N is the total count number up to the end of the program, n is the integer count number initially set equal to 1, T_{n-1} is the period calculated in the previous iteration and t is the time elapsed between the current total time and time at which the robot passed the bottom of the swing. This method is used to accurately determine the optimal position changes to apply during the motion. The optimal positions at which the robot would change its position configuration were also provided by the numerical modelling group. One can see that this is ideal at the extreme points of the motion; this is done so that the change in position causes a change in angular momentum of the robot resulting in a torque being applied on the pivot point that the hands of the robot create, that is between the two free hinges, reducing the effective length of the swing and thus raising the centre of mass of the robot which increases the gravitational potential energy at maxima. This maximises the transfer of kinetic energy to the system resulting in larger amplitudes. This point was found to occur when the driving force of the robot and the period of the swing were out-of-phase by $\frac{\pi}{2}$ corresponding to when the displacement from equilibrium is maximum. In terms of period, this is the same as acting the change at $\frac{1}{4}T_n$ as measured from the equilibrium position. The sign of the angle at this point determined which position the robot would be changed to. The two positions used were stored in the improved SwingAPI script previously outlined in section 7.2. If the sign was positive the robot was set to position2Dynamic (extended), on the contrary if the sign was negative the robot was set to position1Dynamic

(seated). The Python file PositionDynamicFeedback.py contains the code for this approach.

7.6 Results with NAO

Using PositionDynamicFeedback.py amplitudes up to $(14.8 \pm 0.2)^\circ$ were obtained using a self-start function outlined in Section 9.5 over a total time elapsed of 300 seconds on the double-hinged swing. This amplitude is smaller than the one obtained last year of $(15.8 \pm 0.2)^\circ$ over a total time elapsed of 7.5 minutes on a solid rod swing. An explanation as to why NAO wasn't able to drive its motion as efficiently as last year can be explained by the chaotic nature of the motion of the double-hinged swing which caused the joints to randomly oscillate out-of-phase with the motion of the swing system, drawing energy from the system and thus resulting in a decrease in amplitude. Furthermore, adding the upper body movement to the robot caused the robot's joints to heat up faster than last year; therefore the simulation could not be run for more than 300 seconds after which NAO would repeatedly emit warning messages that its motors are too hot. At this point, the simulation was stopped to avoid inflicting damage to the robot NAO.

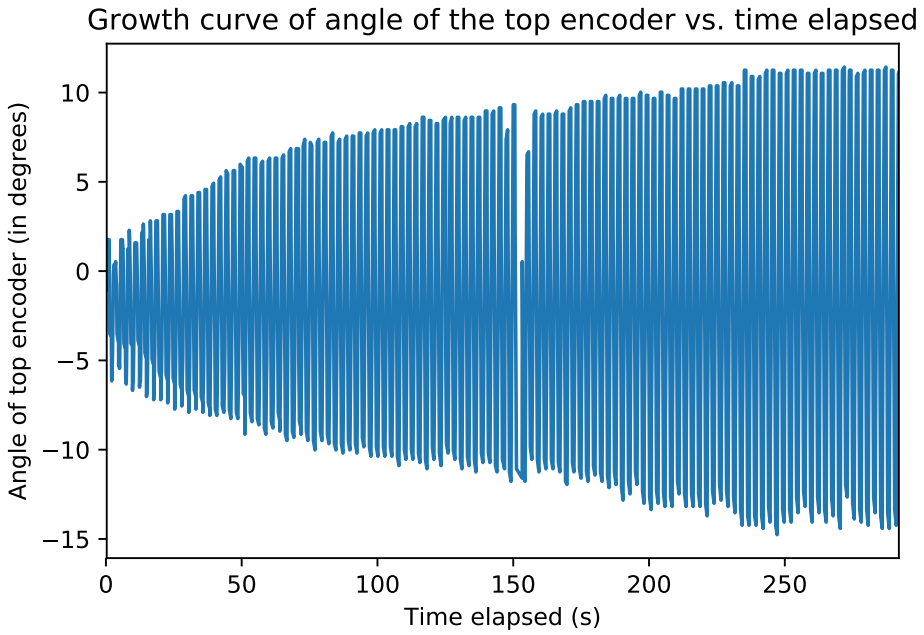


Figure 7.1: Graph showing the amplitude of the swing angle as a function of time

7.7 Asymmetry of Swing Angle Data

Figure 7.1 represents the growth of the swing angle driven by the robot's dynamic motion. The abnormal drop of amplitude growth at 150 seconds arose from sudden random out of phase oscillations of the hinges which drew energy from the system. One can clearly see that the envelope obtained is asymmetric which means that there was more amplitude gain in the negative direction when the robot was shifting from position1Dynamic (seated position) to position2Dynamic (extended position) than in the opposite direction. This could be the result of a larger torque applied periodically to the swing by the robot during this specific position change. However since the swing system can also be approximated as a double pendulum, it should be very sensitive to initial conditions[36] which could offset the data from the start. Both sources of asymmetry were investigated to determine which one was the more dominant.

7.7.1 Initial Conditions

Looking at amplitudes obtained when the robot isn't moving might explain the reasons behind the asymmetry of figure 7.1. This was done by first looking at the motion of the swing without the robot, motion of the swing with the robot in the two different prescribed positions (seated and extended) and by looking at the effect of the harness and its tether on the motion of the robot.

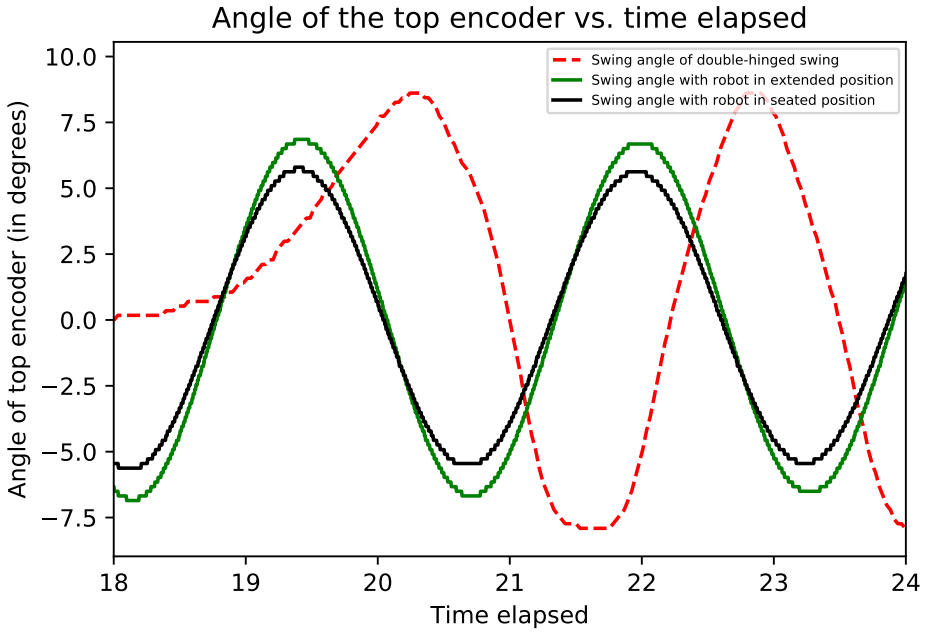


Figure 7.2: This figure displays the swing angle as a function of time of the swing with no robot and that of the robot tethered to the system in both seated and extended positions.

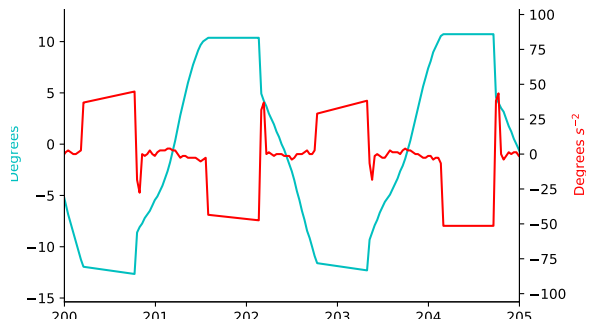
The natural angular displacement of the swing (no robot) was symmetric about zero and the increase in amplitude corresponds to the point at which an initial amplitude was given to the swing and therefore does not seem to contribute to the overall asymmetry of the motion. The seated position seemed to also be symmetric about zero which was expected since the robot was stationary and the tether (should it have an effect) was only taut when the robot was in its extended position. The extended position swing angle decay was also symmetric and therefore was not a contributing factor to the overall asymmetry of the motion. The effect of the tether was also investigated but it was found that there were no dominant discrepancies arising from use of the tether on the robot and thus did not explain the asymmetry of figure 7.1. It was also found that if the angle encoder was zeroed at any other position other than equilibrium, the swing angle read would display a net bias in the direction of the initial angle. Therefore one should always make sure that the angle encoder is zeroed at the equilibrium position to avoid systematic biases appearing in the encoder data. Initial conditions investigated arising from the swing design and the robot's position did not have a dominant effect on the amplitude of the swing angle and hence the reasons for the asymmetry for figure 7.1 must lie elsewhere.

Another area that one could explore is to search for a periodic discrepancy in the torques applied to the swing by the robot that could explain the asymmetry of the amplitude of the swing angle. Since torque (τ) is related to the angular acceleration (α) by equation 7.2.

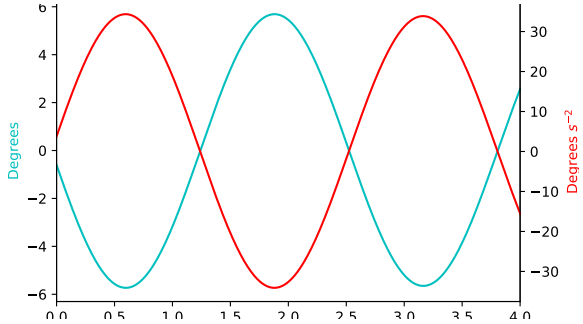
$$\tau = I\alpha, \tag{7.2}$$

where I is the moment of inertia of the swing system, which was constant for both changes in position, one can deduce that a large angular acceleration applied by the robot would yield a large applied torque to the swing. In order to investigate this matter, one can produce a graph of the angular acceleration of the swing

angle against time which can then be used to quantitatively compare the angular acceleration applied by the robot to the swing at the different position changes (extreme points).



(a) Angular acceleration and displacement of the robot driven swing against Time



(b) Damped Oscillation of stationary robot in extended position against Time

Figure 7.3: These figures display the angular acceleration and displacement of the swing as a function time showing the restoring nature of the angular acceleration being always directed towards equilibrium.

Figure 7.3 displays the amplitude of the swing angle and its corresponding angular acceleration for a specific time interval of the swing angle curve. By comparing figures 7.3 (a) and (b), one can observe the periodic occurrence of abnormal spikes in both the positive and negative directions of figure 7.3 (a) which seem to drive the motion just before a change of swing direction. Since these spikes are not present in the naturally damped oscillations of the robot, one can infer that they must be a result of the torque applied by the robot at the extreme points (the origin of this torque is discussed in Section 1.3).

Another important feature of Figure 7.3 (a) is the direction of the applied torque which generates a spike in the angular acceleration of the swing in the opposite direction of the swing's motion. This was expected since the applied torque was responsible for lifting the robot's centre of mass and therefore must be directed away from equilibrium so that this supplementary height gained can result in an increase in gravitational potential energy of the system and thereby increasing the kinetic energy transferred to system which results in an increase in the swing angle (for more details see Section 1.3). One can see that the positive angular acceleration spikes are larger than that in the negative direction and this is true for the remainder of the recorded amplitudes; this means that a larger torque is applied to the swing when shifting from the seated to the extended position than in the opposite order. This resulted in larger negative swing amplitudes. One can then quantify this difference in torques applied to the swing by the robot by reading off angular acceleration and displacement values at the spikes on figure 7.3 (a).

The uncertainty in reading data point values on the graph was taken to be half of the smallest scale division after zooming in on figure 7.3 (a), that is 0.05°. This uncertainty was added using standard statistical error propagation methods to the initial instrumental uncertainty of 0.2° which was determined in Section 4.9, yielding a 0.21° error on the data points. The maximum angular acceleration applied to the swing was (37.05 $\pm$ 0.21)°s^{-2} in the positive direction which was more than twice the value in the negative direction of (-18.45 $\pm$ 0.21)°s^{-2} yielding amplitudes of (-12.35 $\pm$ 0.21)° and (10.70 $\pm$ 0.21)° respectively. The torque applied when the robot shifted from the seated to the extended position was more than twice the torque applied in the opposite position change. This lead to the asymmetry exhibited by Figure 7.1.

Based on this investigation one can conclude that this periodic difference in torque is the dominant contributor to the asymmetry of the motion. However, had this project not been delayed by compatibility issues, more reasons for this asymmetry could have been discussed providing a more substantial insight into the complexities of the motion of the robot on a double-hinged swing. This should definitely be investigated in future years, in order to better understand these discrepancies and achieve a more symmetric motion pumping the swing angle equally at the extreme points.

7.8 Swinging from Position Feedback using Human Motion

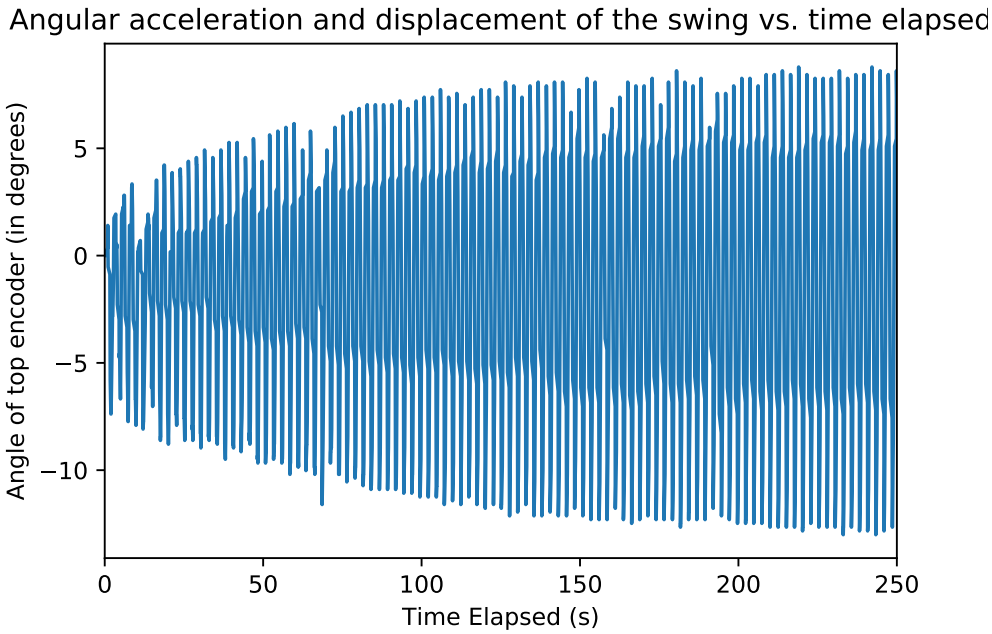


Figure 7.4: Graph showing the driven amplitude of the swing angle as a function of time using human motion data

Following Karandeep and Alkesh's investigation into tracking and analysing the motion of a human on a swing, it was then decided that in order to see whether said motion was applicable to the robot, their results had to be simulated on the real robot NAO. Karandeep and Alkesh were able to determine the points at which the angular acceleration of the torso and knee angles are at a maximum. The occurrence of these extreme points was expressed as fraction of the period as measured from the equilibrium position; these periods were then averaged to obtain a single mean fractional period. This value replaced the numerically determined fractional period used in PositionFeedbackDynamic.py to drive the robot's swinging motion. The average fractional period prescribed was now $(0.15 \pm 0.02)\ T_n$. This change resulted in maximum amplitudes up to $(13.01 \pm 0.20)^\circ$ which were obtained over a total time elapsed of 250 seconds using the self-start function.

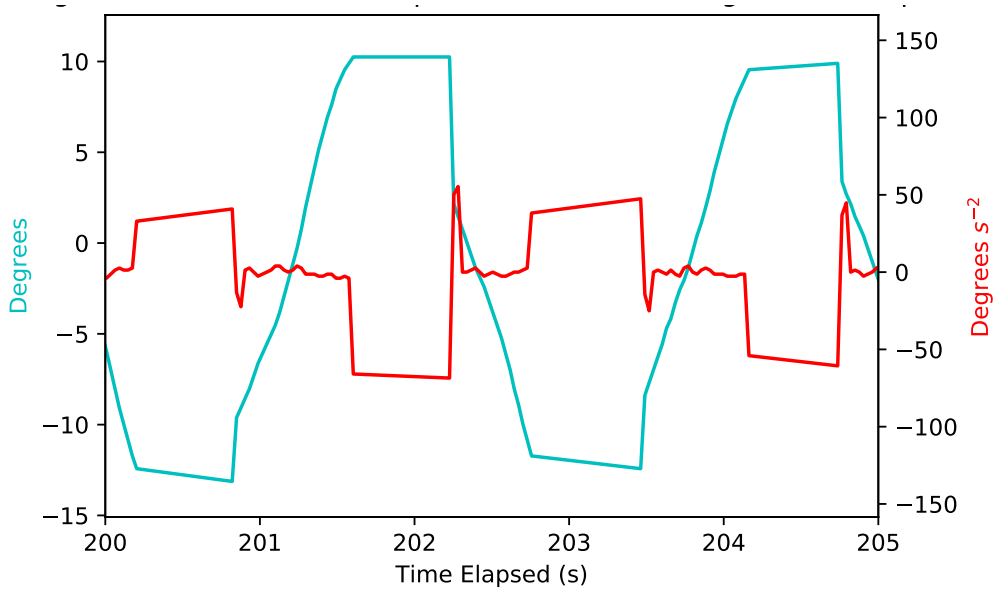


Figure 7.5: Graph showing the driven amplitude and the angular acceleration of the swing angle as a function of time using human motion data

A striking feature of Figure 7.4 is again the asymmetry of the curve produced with a net gain in swing angle being greater in the negative direction than in the positive direction. By looking at the periodic angular acceleration spikes in Figure 7.5, one can again see that an asymmetry in torques applied to the swing by the robot at the extreme points exists. Furthermore, by comparing human motion data and the robot's position one can infer whether the excess of torque produced in one direction is a result of a more human-like position of the robot rendering it more effective at pumping energy into the swing system and hence explaining the asymmetry of the envelope.

The human modelling subgroup tracked the angles of the knee joint relative to the axis of the thigh and the axis of the lower leg, and the torso angle relative to the axis of the torso and the normal of the seat, this angle is zeroed when the torso axis and the normal to the seat coincided. Choregraphe defines the knee and hip joint angles similarly except that the hip joint angle is zeroed when the axis of the torso and the axis of the thigh coincide.

Table 7.1: Table of the Maximum and Minimum knee and torso/hip angles of the human swinger and the robot NAO

	Seated Position Knee Angle (°)	Extended Position Knee Angle (°)	Seated Position Torso/Hip Angle (°)	Extended Position Torso/Hip Angle (°)
Choregraphe Knee Angle	85.26 ± 0.05	-5.15 ± 0.05	57.3 ± 0.05	34.37 ± 0.05
Human Modelling	131.50 ± 0.11	3.64 ± 0.12	0 ± 0.001	54.93 ± 0.55

Given the difference in torques is connected to the asymmetry in the angular distribution; the data from human motion was inspected. It was found that no asymmetry was exhibited. The maximum knee angle was identified and the difference between this and the robot's maximum is (46.24 ± 0.16)°. A full account of the maximum and minimum joint angles is shown in Table 7.1. It was found that the limit on the knee angle in the seated position is a decisive factor in the torque applied to the swing.

Future investigations could measure the angular acceleration applied to swing by the robot at varying knee pitch angles to determine a relationship between applied angular acceleration and the amplitude of the swing obtained as a function of the knee pitch angle. This would drastically increase the pertinence and credibility of the asymmetry argument outlined above.

8 Machine Learning

8.1 An Introduction to the Machine Learning Group

Harvey Bithray

One of the main aims in robotics is to create as fully autonomous a machine as possible; one of the best ways to achieve this is machine learning. Allowing a robot to use its past experiences to inform its future actions means less human supervision is required and a machine can pick up positive behaviours for a wide set of situations, quickly and effectively. The machine learning subgroup was formed with the aim to see how it could be applied in the context of the swinging robot.

Previous years of the project had performed extensive work on machine learning, specifically 2017's report[9] who's 'no stone unturned' approach was hugely informative in defining the route this subgroup would take through our own studies. It was decided early on that rather than repeating a wide coverage of different methods of machine learning as in 2017, it would be better to develop on the previous years research and hone in on one of the more effective methods studied, and apply this to a model more relevant to our new swing - specifically a double pendulum model. In previous years a dumbbell model[9] or an inverted pendulum[6] were used. This would involve applying the physics of a more complex model, allowing one to see how the efficacy of the machine learning algorithm would be affected by this. Moreover, focusing more specifically on one particular model allowed time for a more in depth statistical analysis of data, which brought up some interesting aspects of machine learning which will be elaborated on later in this section.

8.2 Theory

Tom Batchelor

Learning is a process that all humans partake in. They perform actions and whether good or bad, they learn from them. This can be by ensuring that they do not make the same mistakes again, or repeat positive behaviours for which they were rewarded. Machine learning is a branch of Artificial Intelligence (AI) with the aim of enabling systems to learn in a similar way to humans. The computers are programmed to observe how an action they perform impacts their environment. They then take this data and learn how to perform a better action in the future. The algorithms are designed in a way to make the computer self-sufficient in order to minimise the amount of human input required. The fundamental goal is for the computer to generalise beyond its training method and positively interpret data it has never come across before.[37–39]

8.2.1 Applications of Machine Learning

There are numerous ways in which machine learning is applied outside of robotics, it is finding its way into almost every sector of life. This is mainly due to the extent to which computers excel over humans in data processing and analysis. It is down to this excellence that machine learning is becoming progressively more widespread.

One such way in which machine learning has been implemented is in Computer-aided Diagnosis (CAD). CAD works using Artificial Neural Networks (ANN) to go through several steps including image processing, image feature analysis and data classification. It has been used in radiology to detect early-stage breast cancers that would have otherwise been missed. A study investigated the use of CAD on scans which had been diagnosed as negative, when in fact a year later the women who had the scans developed breast cancer. The CAD workstation identified 52% of these missed cancers. Radiologists now use CAD as a second opinion in all routine work with the detection of breast cancer.[40,41] Another way is the use of cognitive computing with genomic tumour sequencing. This enables laboratory sequencing and analysis of a tumour's genetic makeup to help reveal any associated targeted therapies and clinical trials.[42]

An application of machine learning that is rife with anticipation and excitement within the technological industry today is the idea of autonomous vehicles (AVs) – vehicles that can operate without human input.

They rely heavily on deep-learning algorithms to integrate data points from several different sensors (cameras, radars etc.) to become more intelligent as time goes on, leading to safer driving. The idea of AVs is not only a huge technological advancement, but one that has potentially life-saving consequences. A report has shown that the infiltration of AVs into daily life could reduce accidents by up to 90% in the US, leading to roughly 190 billion saved in the economy.[43] Despite this, AVs are not perfect. According to the State of California Department of Motor Vehicles, there have been 34 reported incidents from 2014-2017 involving self-driving cars in California. Interestingly, just 4 of these incidents were the fault of the self-driven car, meaning 30 were the fault of a human-driven car; a remarkably stark difference.[44] AVs are only going to improve but already they are becoming a thing of the present, rather than a thing of the future.

8.2.2 Types of Learning

To fully understand why autonomy is important in regard to robots, let's begin with the definition of a robot - 'A machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer.'[45] Thus, enabling a robot to act of its own accord is essential to it being a robot. The application of machine learning allows the robot to learn from its own actions, closer replicating autonomous behaviour.

Machine learning can be split up into three main types of learning: supervised, unsupervised, and reinforcement learning. The type of learning chosen depends completely on the problem. If one has a target, a value or a class to predict, then one will need to give the model historical data and train it to use that data to predict what will occur in the future – the model therefore knows what it needs to learn, so is classified as supervised learning. If the data which one has is unspecified and there is a need to group this data by observing patterns, then unsupervised learning should be used. If one has an objective one needs to complete, then one should use reinforcement learning so that the model can be repeated iteratively in order to find the best way to complete said objective.[46]

Supervised learning is where one knows an input and its associated output, and then teaches the algorithm the function that maps the input to the output. The name supervised comes from the fact the algorithm is trained using known data in order to predict new data – the known data can be thought as supervising the algorithms learning process. The algorithm will make adjustments according to how its prediction compares to the known data. The learning will halt when the algorithm provides a satisfactory level of performance.[47]

Unsupervised learning is where one only has input data and must use an algorithm in order to model how this input data is distributed. The algorithm could discover intrinsic groups within the data or discover rules that describe parts of your data. There are no correct answers and the algorithm is designed so that it finds and presents patterns or groupings in the data.[47]

Reinforcement learning is a reward-based learning. The reward is wholly dependent on the objective of the algorithm. With this objective in mind, the agent performs a certain action in its environment. If the action acts in such a way that it complies with the objective, then the reward for that action will be good, and vice versa. It is trained via trial-and-error, so when an action is performed that gives a bad reward, the agent learns never to do that action again. The ultimate goal is to maximise the long-term reward.[48]

The main aim of this project is to optimise the swinging motion of the NAO robot. In order for this to be achieved, the robot must effectively increase its swinging amplitude. Therefore, there is an objective in mind, and so reinforcement learning would be the most suitable type of learning for this project. There are several different types of reinforcement learning and in order to determine which one shall be used, research into the different types was made.

8.2.3 Types of Reinforcement Learning

Alex Thornton

A review of previous years reports and literature was done to determine which reinforcement algorithms were best suited to pursue in this project. The main criteria was the scope for progress compared to other years work.

Actor critic

The actor has a policy in a particular state s. This policy is a weighting of all the possible actions. The actor acts out the chosen action, chosen randomly according to policy weightings, and obtains a reward for being in the state $s+1$. The critic estimates the total reward if the general policy was followed. Then if the reward it receives is better than if it followed policy, the policy is altered to favour that action. The advantages of this are quick convergence to optimal policy as long as random noise doesn't affect the system early on. A disadvantage is the system is susceptible to early bias. Bias could easily occur due to prior actions affecting ill-defined qualities in the system. The algorithm also struggles with large state action spaces due to memory and computational issues.[49]

SARSA

SARSA is an on policy reinforcement learning algorithm. Analogous to Q-learning it uses a similar algorithm but predicts rewards obtainable in the next state. Therefore it allows back propagation to occur, the decision to take a certain action in a state is determined by the weighting of the Q-values, semi analogous to the policy weightings in the actor critic algorithm.[49]

Q-Learning

Q-learning is different as it uses the maximum reward for the whole system as a motivator, making it final destination orientated rather than the path orientated SARSA. Q-learning requires less computational time as the simulation doesn't have to continuously find the highest Q-values in the resulting states.[49]

Deep Q-Learning

Deep Q-Learning works by using a set of artificial neurons which form a neural network. The neurons are used to determine weightings for different Q-Values. This is good for finding generalised solutions to high dimensional environments with low constraints on behaviour. However this is difficult to implement and has been done extensively in last year's report[9] so the scope for improvement is low given limited human resources. Furthermore, deep neural networks can require large amounts of memory, so memory management can prove an unnecessary distraction from the main aim of the project.[49] Explicitly within the category of Deep Learning, Deep-Deterministic policy gradients would prove a good avenue for investigation, however multiple different actions are needed to be used in sequence to produce a desired result. Also bad actions can perturb the system in a way which makes good actions bad. It does this by taking the differential of the reward from a policy with respect to different parameter to find which parameters produce the reward.[49]

To summarise, the reinforcement learning algorithm chosen was Q-learning. This algorithm was picked as it is computationally efficient, allowing versions of the simulations to be rerun quicker in comparison to other algorithms. Given that the problem was to be given reasonable constraints, more involved deep neural networks with deep deterministic policy gradients were decided against due to their difficult implementation and the previous year's team extensive work into them. With Q-learning there was plenty of scope for improvements on previous years work, whereby other algorithms had been exhausted in there extent, especially given the small group tasked with machine learning in this project.

8.2.4 Q-Learning

Harvey Bithray

Q-Learning is a temporal difference algorithm, which means it aims to create an agent which uses its past experiences to inform its future actions. It is does so by splitting an environment into discrete states. Within these states, an agent is allowed to perform discrete, predefined actions. Each of these State-Action pairs will have a measurable effect on the system, to which a reward can be attributed. This is commonly done by applying an equation in a similar form to the equation in figure 8.1. Here $Q(s_t, a_t)$ represents the Q-value in a particular state, for a particular action,

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

Figure 8.1: General form of the Q-Value Equation

It can be seen that the equation can be repeated iteratively, taking into account the previous value added to the difference between the learned value, and the initial Q-value. The learned value takes into account a number of constants. The learning rate, α, multiplies the entire new term that is added to the original Q-value. This has the effect of determining to what degree new information is valued in comparison to the old information. A value of $\alpha = 0$ would mean no learning takes place, whereas $\alpha = 1$ would impose that only new information is used. Practically a constant value is most commonly used that accounts for both these factors, however in some cases this constant can be more dynamic and change as the epoch number increases, reflecting the more filled Q-Space.

The discount factor, γ, determines how much a future reward is strived for. With knowledge of the best possible value, an agent will strive to achieve a better long term reward, at the price of short term rewards. $\gamma = 0$ means that the agent is myopic and only considers the immediate reward, whereas a γ closer to 1 means that the agent looks for a longer term reward.

A Q-Value can then be stored and updated iteratively in an array for each action, in each state. The agent will then search through this array to find and perform the best action possible over a swing cycle. How this is applied will be explained more specifically in section 8.3.2.

This method of machine learning was applied in two different coding languages, C and Python. Python is a more common language, and it made logical sense to write in python due to it being what has been done in previous years, and it being the language that can be interpreted most easily by the robot. C was also used partly because of it being easier to access with a windows PC, however it also has some practical benefits for computation. Python, unlike C, is an interpreted language, meaning before code is ran through the CPU it must be interpreted, by an interpreter. This increases computation time, meaning any code will take longer to run. This means, if the machine learning code got to a position where it could be applied to the robot, which is possible as the NAO can interpret interpret C, it could give quicker instructions, and as such more accurately timed actions. Although this was not achieved within the scope of this years project, there is room for investigation in future years.

8.3 Python

8.3.1 Environments

Tom Batchelor

A Q-Learning algorithm requires an environment in which to run. So, an investigation into different ways an environment could be set up was undertaken. One such toolkit was OpenAI Gym – an open source toolkit used for developing reinforcement learning algorithms.[50] This was seen to potentially be an ideal way of testing any Q-learning algorithms that were developed. Installation of all but one of the dependencies went without disruption, but due to the fact MuJoCo is proprietary software meant that it had to be installed separately after obtaining a free student licence. MuJoCo had an inverted double pendulum environment and so was appealing due to the double pendulum environment that was planned. After attempting the installation of MuJoCo, and it being unsuccessful, the environments that depended on MuJoCo were tested and would still not work. Plenty of time was dedicated to this installation, however, it was decided that time would be better spent exploring other avenues within OpenAI Gym.

One environment that was investigated fairly extensively was the Cart Pole environment within the Classic control dependency, this can be seen in figure 8.2. The idea of this environment was to maintain the poles upright position for as long as possible, and when the pole has declined beyond a certain angle, the environment will reset and restart the balancing process. Now without a learning algorithm in place, the agent will perform random actions indefinitely. A learning algorithm was found online that applied Q-Learning to this environment, which can be found at.[51] Investigating this algorithm proved useful in understanding the basics of how Q-Learning worked, and developing our own Q-learning code. Utilising existing environments within OpenAI Gym was a possibility – but given the amount of changes that would have had to have been made, time was better spent developing a new, simpler, bespoke environment.

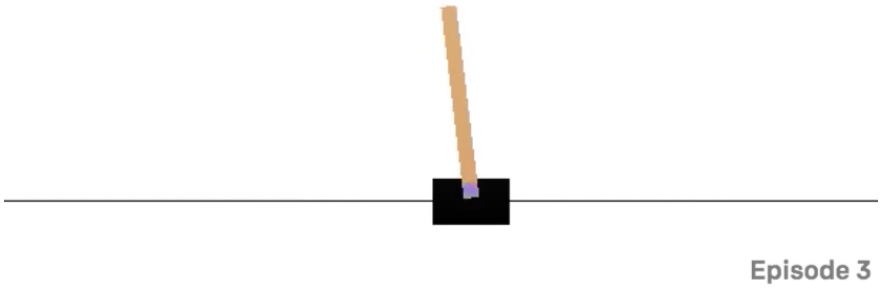


Figure 8.2: Screenshot of the CartPole-v0 environment within the Classic Control Dependency of OpenAI Gym

There is a module in python called `tkinter` which could do precisely what was required. A simple Graphical User Interface (GUI) that could be updated to show the movement of the different parts of the pendulum. All that had to be done was to initialise two sections of a double pendulum with a variable position. The lines for each pendulum can be formed simply by putting in two sets of x and y coordinates. These pendulums were then updated in the `numod` class (covered in detail in the beginning of Section 8.3.2) to get the associated angles, and then there were functions within the GUI code to convert these angles to each respective x and

y position. This was updated iteratively over a loop in order to show the motion of the double pendulum. The updating of the UI will be explained in more detail in Section 8.3.2. The GUI can be seen in figure 8.3.

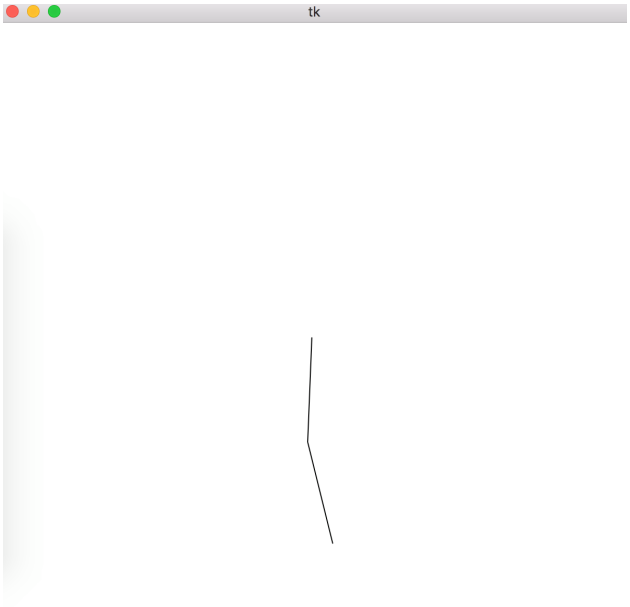


Figure 8.3: Double Pendulum Model using the `tkinter` module in Python

8.3.2 Structure

Harvey Bithray

The Program is split into 3 classes, `QUI`, `Numod`, and `QMain`. A UML of these classes is shown in A.2

`Numod` deals with all the physics of the simulation engine, as well as solving the equations of motion using the Fourth Order Runge-Kutta method, as outlined in section 2.5.1.

`Numod` contains functions for the position of the tip of both the first and second pendulum which are `x0pos`, `y0pos` for the first pendulum, and `x1pos`, `y1pos` for the second. These multiply the length of the pendulum by the appropriate trigonometric function to determine the position, with `x1pos` and `y1pos` doing the same but also taking into consideration the position $(x0, y0)$. $(x0, y0)$ represents the end of the first pendulum, so considers the main swinging angle α, whereas $(x1, y1)$ represent the end of the second pendulum, so concerns the second angle θ. `Numod` also contains functions which relate the action performed by the robot to the resulting change in θ. This begins with the method `theta_dot_dot` which simply returns the action it takes as a variable. The conversion between torque provided by the robot, and the angular acceleration $\ddot{\theta}$ is handled in the `QUI` class. The `theta_dot` function then multiplies the value returned by `theta_dot_dot` to a small time step, hard coded as 0.1s throughout the code, as well as adding the initial value of $\dot{\theta}$. This follows a simple integration of the angular acceleration with respect to time. Similarly the `theta` function returns θ by adding its initial value added to the change in $\dot{\theta}$ over a small time-step.

Next is the function `function1`. This simple function returns the value that it takes as a variable, utilised in the Runge-Kutta method. `functiondoublependulum` returns the decoupled equation of motion for a double pendulum as outlined in equation 2.21 using the functions involving θ aforementioned.

`rk_alpha` and `rk_alphadot` use Runge-Kutta methods to numerically solve the equations of motion, returning α and $\dot{\alpha}$ respectively.

Tom Batchelor

The QUI includes the definition of one class, called DPendulum. This includes all of the methods used in this code. The first method is the __init__ method, which takes eight variables: self, m0, m1, l0, l1, num_pos, alpha and gamma. The last seven are the variables which will get given a value in the QMAIN. The __init__ method in Python is a constructor for the class. This is simply used as a function which uses self to create an instance of the object DPendulum, to which the rest of the code is applied. In Python, creating and mentioning self is explicit, unlike in C++ where self is not parametrised and it is just assumed.

The num_pos variable is defined as the number of states in the state space, which is defined by the possible positions of the robot. The learning rate (self.a = alpha) and the discount rate (self.g = gamma) are used as part of the Q-value calculation. There are two instances relating to the actions of the robot: num_actions and actions where num_actions is simply the number of possible actions (i.e. three in this case), and actions is the value of each action. So, actions is an array in which there is an action corresponding to a negative acceleration due to the torque, no action and a positive acceleration due to the torque of the robots legs.

Within __init__, there is a need to define the boundaries in which the pendulum may exist, and so max_alpha and min_alpha are set to π and -π respectively. The difference between two consecutive states in radians is defined as one_alpha and is [(max_alpha – min_alpha) / num_pos]. It can also be thought as the value of one index. Next, the q_values array was initialised as a numpy.zeros array and will be updated in perform_action. The UI had to be designed in this method due to the need of updating its position in other functions. The centre of the canvas was defined using ui_cent. This was useful in order to keep the pendulum designs consistent with one another and the size of the canvas itself. Each pendulums tips were defined as functions tip1 and tip2, which both made use of l0 and l1 and the ability to change their respective lengths, using ui_cent as a reference to the middle of the canvas. Each pendulum, defined as ui_pend1 and ui_pend2, were then plotted and initialised.

The next function is the reward function, which quite unambiguously returns the reward for our system. The objective of the system is to increase the amplitude. The greater the increase in alpha, the greater the reward should be. So, it is simply defined as the (final_alpha – initial_alpha)*100. The factor of 100 simply scales the reward. The next function is index_of_state which returns index, which is simply the index of the state which the pendulum is currently in. It also includes a loop in which if the state is out of bounds, it raises an exception. So the if loop is if alpha > self.max_alpha or alpha < self.min_alpha, then the function returns IndexError.

There are a few methods involved with updating the pendulums position. Firstly, functions to define the x and y positions throughout the simulation are needed. This is because when α and θ change, the position of each pendulum needs to show that change. This was done by converting the alpha and theta variables from numod into positions on the canvas. The two methods for the first pendulums position, ui_x1_pos and ui_y1_pos, and the two methods for the second pendulums position, ui_x2_pos and ui_y2_pos are very similar to that as explained in numod - the only difference is the use of ui_cent in the first pendulums methods in order to ensure that the pendulum is in the centre of the canvas. There was also a scaling factor of 100 in all of them which helps in the visualisation of the pendulum. There was a method needed to update the second pendulums position and this was called ui_move_pendulum2. In it were new updated definitions of the tips of each pendulum. The definitions used the four preceding methods in order to define the new position of each tip using α and θ. A method that was added later on after the second pendulum was found to not be updating was change_theta. This simply calls the function theta_dot from numod and sets a new variable theta_dot equal to it. It then returns theta from the numod code using this new theta_dot variable as a parameter.

Within the method perform_action, which will be described in more detail later, there is an if loop which updates the UI. Within the loop, is another update of the tip of pendulum one, again using ui_x1_pos and ui_y1_pos, with its variable alpha set to current_alpha. This new x and y position is then used in defining the new position of pendulum one. In this loop, the function ui_move_pendulum2 is called in order to update the position of the second pendulum, with its variables alpha and theta set to current_alpha and current_theta. Then self.ui.update() updates the UI.

Harvey Bithray

All other methods in `QUI` regard the updating of the Q-space. The Q-space is defined as a three dimensional array consisting of three arrays of size `num.positions` by `num.positions`. The third dimension, or rather the fact there are three arrays, represent the three performable actions, a negative torque, a positive torque, and no action, whilst the `num.positions` by `num.positions` arrays represent the state space. This means the state space is two dimensional. One dimension takes into account all of the states in the system, defined as all the states that exist, between $-\pi$ and π. The second array represents all states which are accessible for the agent to perform an action in, defined as the states between a current state, and its mirror. This second array is referenced throughout the code as the `action_alpha_index` array. Defining the state space in this way means that the system will not fail to perform an action by waiting to receive a reward that it will never be able to access, given its current position. Higher positions inevitably relate to higher rewards, so this new array allows a more relative swing cycle. There are other ways of doing this, such as having a velocity matrix (as in section 8.4.1). With this, the system would know to perform an action when it is slowing down or changing direction. Having this second position array defined has uses in other functions of the code, such as searching the `action_alpha_index` array for the best action to perform, rather than the whole array. A definition at this point means this array would not have to be redefined in multiple methods. This state space is depicted in figure 8.4, where the grey circle represents the first array, being made up of all states. For a particular swing angle, the states enclosed in the black sector represent the `action_alpha_index` array.

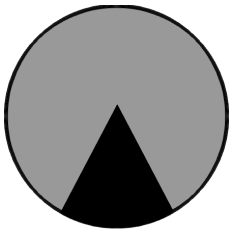


Figure 8.4: Illustration of State Space

With all the practicalities taken care of one can begin to fill the Q-space with values. The first method that relates to this is `max_q_value`. This method searches all the Q-values in the `action_alpha_index` array for that particular swinging angle, and returns it. As explained in section 8.2.4, this is used to determine how the algorithm prioritises new information in the Q-Learning equation.

The main part of the Q-Learning algorithm takes place in `perform_action`. This begins by updating the UI with the iteration number of the swing, and marking out the height of the initial state. Then if requested, random actions will be performed at random positions. This process allows the Q-space to be filled with values so a base set of actions can be learned. This was investigated for different amounts of learning epochs, as will be touched on in section 8.3.3. Then for the duration of the swing, the positions and velocities of the motion are updated and visualised using the functions in the `numod` class. The action is set to zero, unless the agent is about to change direction. At this point, if the state corresponds to the state in the `action_alpha_index` array that provides the best reward, it will perform an action. Imposing this limit seems excessive, but it ensures an action is only performed once per half-cycle of swinging, in the fashion you would expect to see. Without this limit, the action will apply a torque so the agent moves quickly up to the maximum swinging angle by applying a torque in the same direction at every time step. This is unrealistic as the robot in real life, as for any humanoid swinger, can only apply a torque once per half swing. Its 'legs' must return to the neutral position to be able to apply a torque again, and this is enforced by this rule. Moreover, this doesn't force the agent to perform an action at its maximal points, it only allows the action to have an effect on the system at these points. This ensures that the agent is still operating on learnt behaviours. This action is then applied to the swinging motion, updating the variables and UI. The final points of interest in this method is the large negative reward applied if the main swinging angle exceeds the maximum α. Finally, the Q-value array is updated based on these new variables.

The method `best_action` searches all the possible actions, at each position in the `action_alpha_index` array, returning the best position to perform an action, and its associated action. Finally `give_q` simply prints the current q value to a file.

Tom Batchelor

`QMAIN` includes one method: `main()`. This involves two main loops. Firstly, there is a loop which performs random learning. This is where the agent will perform random actions throughout each epoch. This is essential because the agent needs some time to explore as many states as possible. Once these states have been explored then the agent learns from them by updating its Q-space with the associated Q-value for that state-action pair. The loop is `for num1 in range(0, x):` where `x` is the number of epochs for which the random learning will take place. Within the loop, `DPendulum` is called from `QUI`. This is where the variables initialised in the `__init__` method can be given a value. The values were chosen in order to replicate the parameters of the robot. `m0` and `m1` were set to the joint mass of the torso, head and swing and the mass of the legs respectively (in kilograms). Then `l0` and `l1` were set to the length of the swing to the centre of mass of `m0` and the length of the legs. The number of positions (`num_pos`) was set so that there were 501 states in the state space. 501 was deemed a reasonable value for the number of states, so the state corresponding to an `alpha` of 0 has 250 other states either side of it. As aforementioned, the learning rate and discount rate were set to 0.5 and 0.9 respectively.

The simulation needs to know at which value of α it needs to begin at. So this was done using the `random` module in order for the simulation to begin at a random alpha each time. The function `random.random()` gets a random number between 0 and 1 and then this is scaled using `num_positions`, `one_alpha` and `max_alpha`. Next there is a `while` loop which sets a limit on how small of a value of alpha the simulation can start off at. This was done because the algorithm finds it difficult to learn if it starts too close to equilibrium.

Within this loop there is a `for` loop which tells the simulation what to do within each epoch. So it is `for num2 in range(0, x):` where x is the number of swings within each epoch. Now `perform_action` is called in order to update `alpha` and set the variables to required values. Firstly, α and θ are set to the variables `alpha` and `theta`, as these will be changing throughout the loop. `dt` is set to $0.1s$, `action_alpha_index` and `action` are both set to -1 in order to trigger the random action loop in `perform_action`. In this loop θ is also updated using the `change_theta` function from `QUI`. There is also a loop which resets α if it becomes greater than `max_alpha` or less than `min_alpha`.

After the random learning was complete, now the implementation of the learnt Q-values took place. This was accomplished in another loop which follows an almost identical structure to the random loop. The differences come in the second `for` loop. Within this loop, instead of performing a random action, `best_action` is used. This means the agent will perform the best action it currently knows how to perform from that specific state. The variable `action` was simply set to `action` due to the fact this will vary throughout the loop. To show the iteration number on the UI, `iteration` was set to `num1` and `ui` was set to 1 in order for the UI to render. When taking data, the `ui` was set to 0 in order to reduce computation time.

8.3.3 Original Model Results

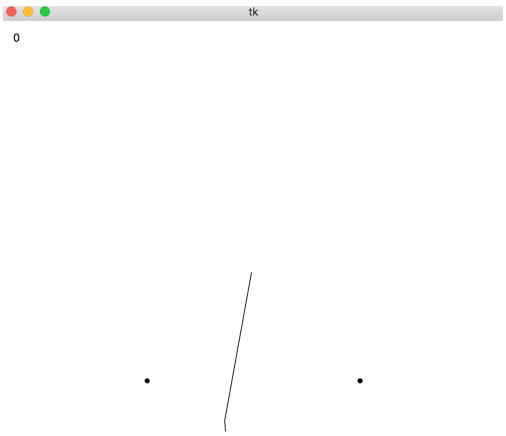


Figure 8.5: Final GUI of the double pendulum using the `tkinter` module

The final GUI is shown above in figure 8.5. The respective lengths of the swing and the leg can be clearly seen and these could be edited easily. The dots represent the amplitude of the current swing and are updated if any amplitude change occurs. The iteration number is displayed in the top left hand corner and increments every epoch, when the environment is reset. Initially, the second pendulum was not moving as it should be, after debugging the code it was realised that θ was not updating correctly and consequently the second pendulum was remaining in its initial position. This prompted the addition of the `change_theta` function in order to successfully update theta and update the second pendulum position.

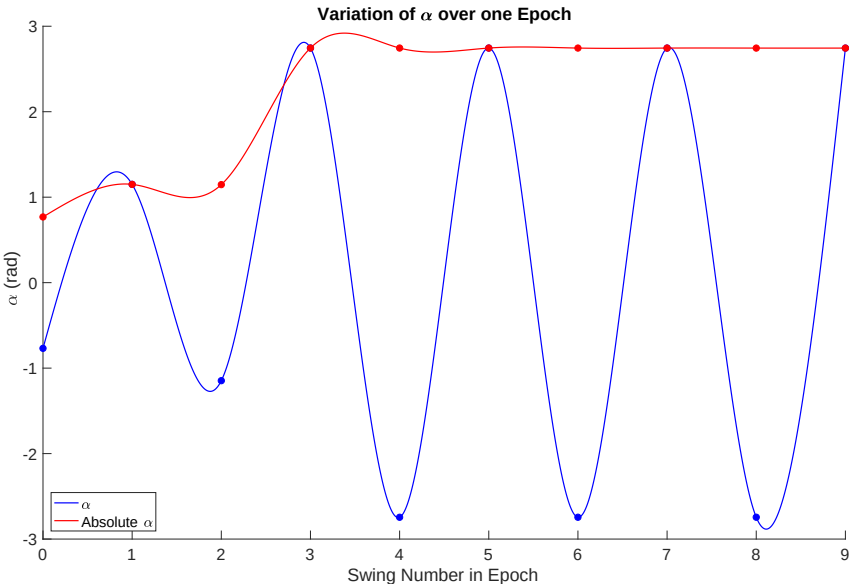


Figure 8.6: Graph showing one epoch of 10 swings in which the agent performed actions leading to increased amplitude.

Figure 8.6 is a graph showing how the agent acts in one of the learnt epochs. This shows that our agent

works in the way it is supposed to, as one can clearly see the agent is performing the desired actions and therefore increasing its amplitude. This epoch occurs in the latter stages of the implemented Q-learning but is not representative of every epoch that takes place. As seen in figure 8.7, there are still epochs where the agent does not perform well.

Harvey Bithray

Following this the simulation was run for differing amounts of random actions, the results of one of these is shown in figure 8.7.

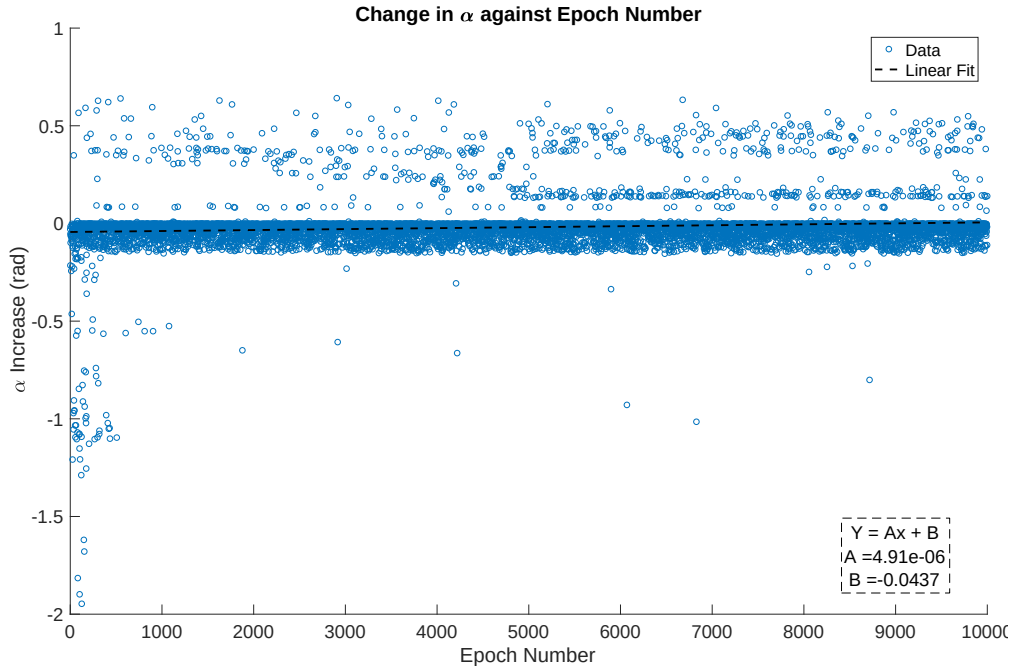


Figure 8.7: Change in alpha for 1000 learning epochs, 10000 learnt epochs

Here one can see the change in alpha over 10,000 epochs, after 1000 learning (random) epochs. Initially, around the first 1,000 implemented epochs, the agent performs poorly, getting very large negative changes of up to 2 radians, though over time these stop occurring. There is a densely populated band, around 0, where presumably no action is taken and the swing cycle just decays. From this graph it is clear to see that with time, more positive actions are taken as the Q-Space is filled and more positive actions are learnt. This is accentuated with comparison to figure 8.8, which shows purely random actions. Here again there is a dense band around 0 where no action is performed, and although large positive changes can occur, there are much more epochs with a large loss in alpha. This is reflected in how the gradient, from a simple linear fit, is an order of magnitude less than that of the learnt actions.

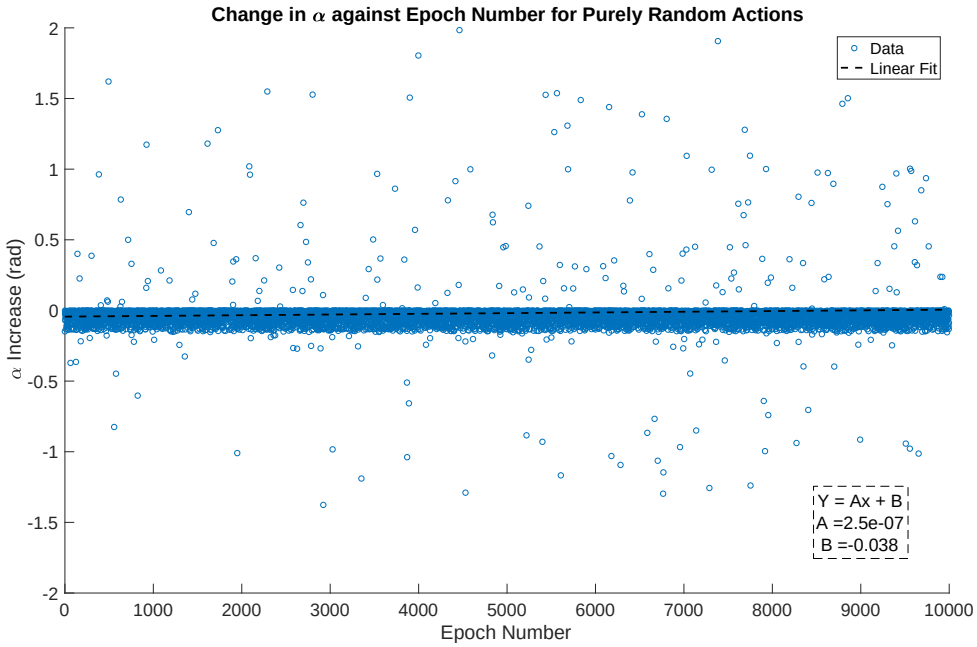


Figure 8.8: Change in α for Purely random actions

The next logical step was to increase the learning time. Before doing so it was presumed that an increased learning time would correlate to a more filled Q-space, and better actions. Surprisingly, as shown in figure 8.9 this was not the case.

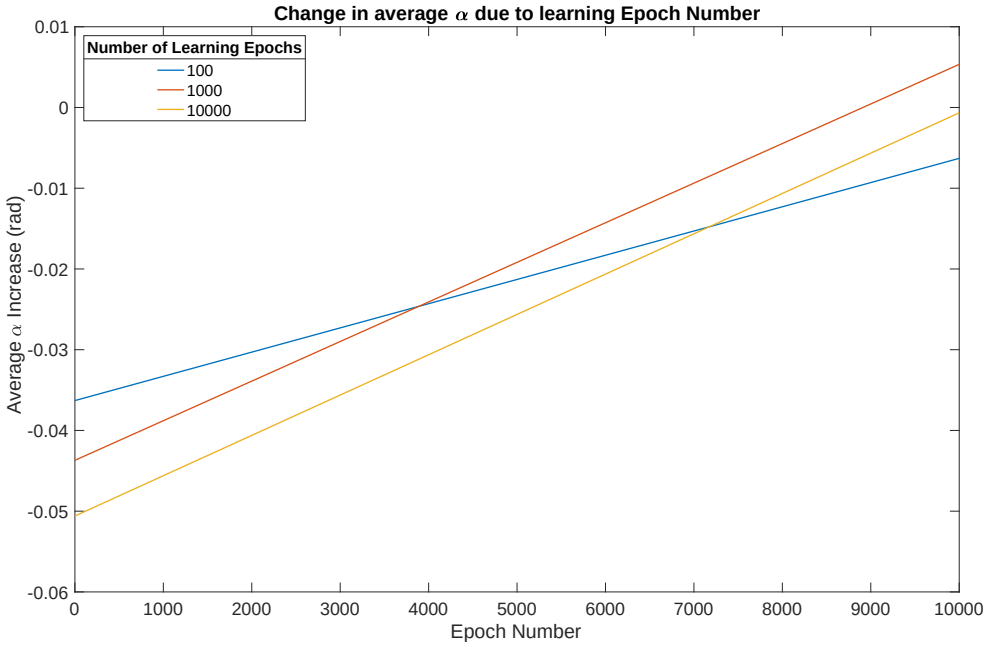


Figure 8.9: Change in α for Differing Learning Time

Here the linear equations fitted to the alpha sets, as shown in figure 8.7, were plotted. One can notice that surprisingly 1,000 learning epochs performed better 10,000 learning epochs. Initially this was thought to be due to the stochastic nature of the generation of the position of the starting angle, however when

repeated this was found to consistently be the case. There are a few possible explanations for this result. One explanation is overfitting. This occurs when a machine learning algorithm learns for too long in an environment and begins to pick up 'noise' from the system and learns negative behaviours as a result. This is often characteristic of low variance, high bias systems such as this. Another potential explanation could be saturation of the maximum Q-value. As seen in figure 8.1, if the Q-max term gets too high, as it will as time goes on, then it is clear that this could certainly become a leading order term in the equation. This is a concept that has been investigated in the C side of the code, where no Q-max term is used. Both these characteristics are factors intrinsic to machine learning that cannot be changed, so other ways of improving the performance of the agent were investigated.

Tom Batchelor

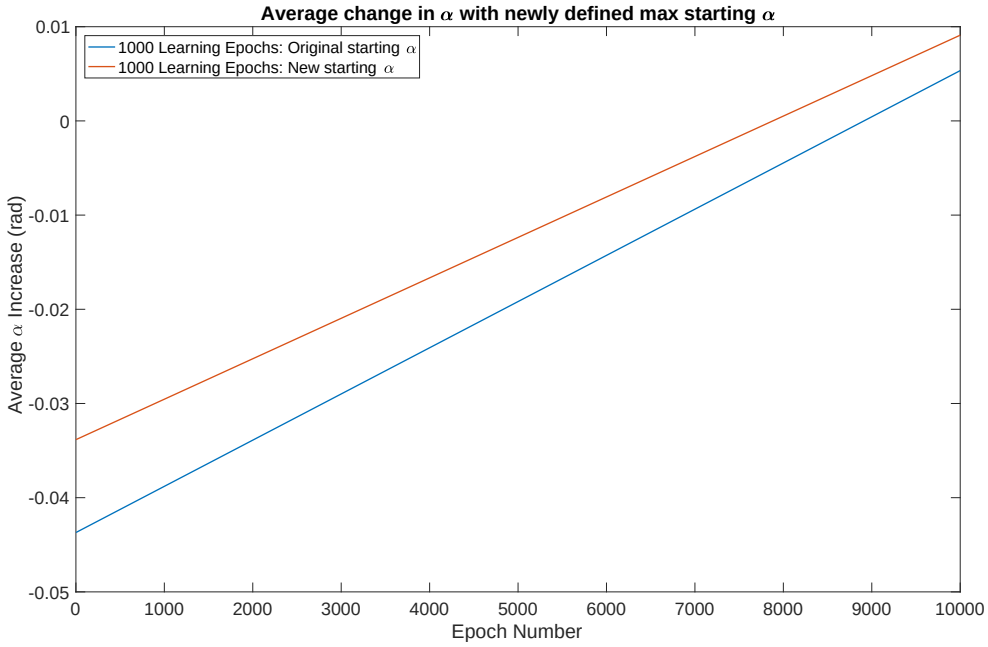


Figure 8.10: A comparison between a simulation ran with the old limits on the intial value of alpha and a simulation ran with the new limits

One improvement considered was to define a new range of starting alphas. One theory was that due to the agent being able to start close to the maximum and minimum of alpha, then the agent would perform no action in case it goes out of bounds. This could explain the dense band around 0 in the previous figures 8.7 and 8.8. This meant that there was reason to limit the maximum starting alpha to a value that would perhaps enable the agent to learn at its highest starting point. The limit was changed so that the range of starting alphas was $-3\pi/4$ to $3\pi/4$ instead of $-\pi$ to π. The figure showing this implementation versus the old limits can be seen in figure 8.10. As seen in this figure, the new limits made the learning slightly better than previously seen, although not hugely different. A dense 0 band is still found though. Due to the logical reasoning behind this change, this new limit was kept for future simulations.

The torque that comes from the robots legs is fairly small at about 8.66Nm. This means that the algorithm was probably doing better that it shows, as the limits of this small torque prevented any significant growth. This is because swinging from the legs alone is an inefficient way of pumping a swing. The torque was increased to an arbitrary 20Nm in order to see if the algorithm was performing as expected. This increase in torque meant that the change in alpha should be better than when the torque was 8.66Nm. This can be confirmed in figure 8.11. Whilst this step served little purpose in use for the future simulations, it did provide confirmation of a successful Q-learning algorithm.

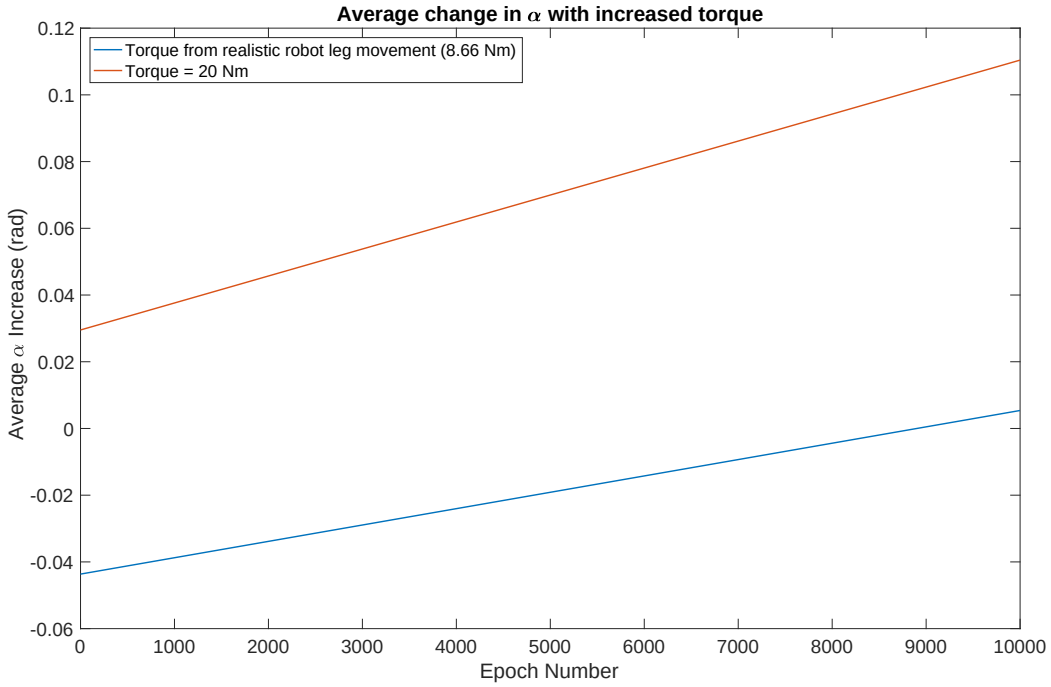


Figure 8.11: Graph showing how the increased torque lead to a general increase in amplitude over time

8.3.4 New Model Results

Following the seminar, it was suggested that a good metric to assess the efficacy of the algorithm would be to start the agent from the same point and see how its swinging improves, or converges to a solution. In doing this it was observed that to achieve the best angle the agent would perform unusual actions that should not be allowed in the system. This involved θ fully rotating as actions were applied that were unrealistic. To counteract this, it was decided that a limit should be placed on the maximum and minimum values of θ. This prevented any un-physical actions from being applied. This small change had a huge effect on how well the system performed, as shown in figure 8.12. In comparison to figure 8.7, one can note how the average increase in α is significantly higher than before, the shape being similar but the magnitude of the positive gains being much higher.

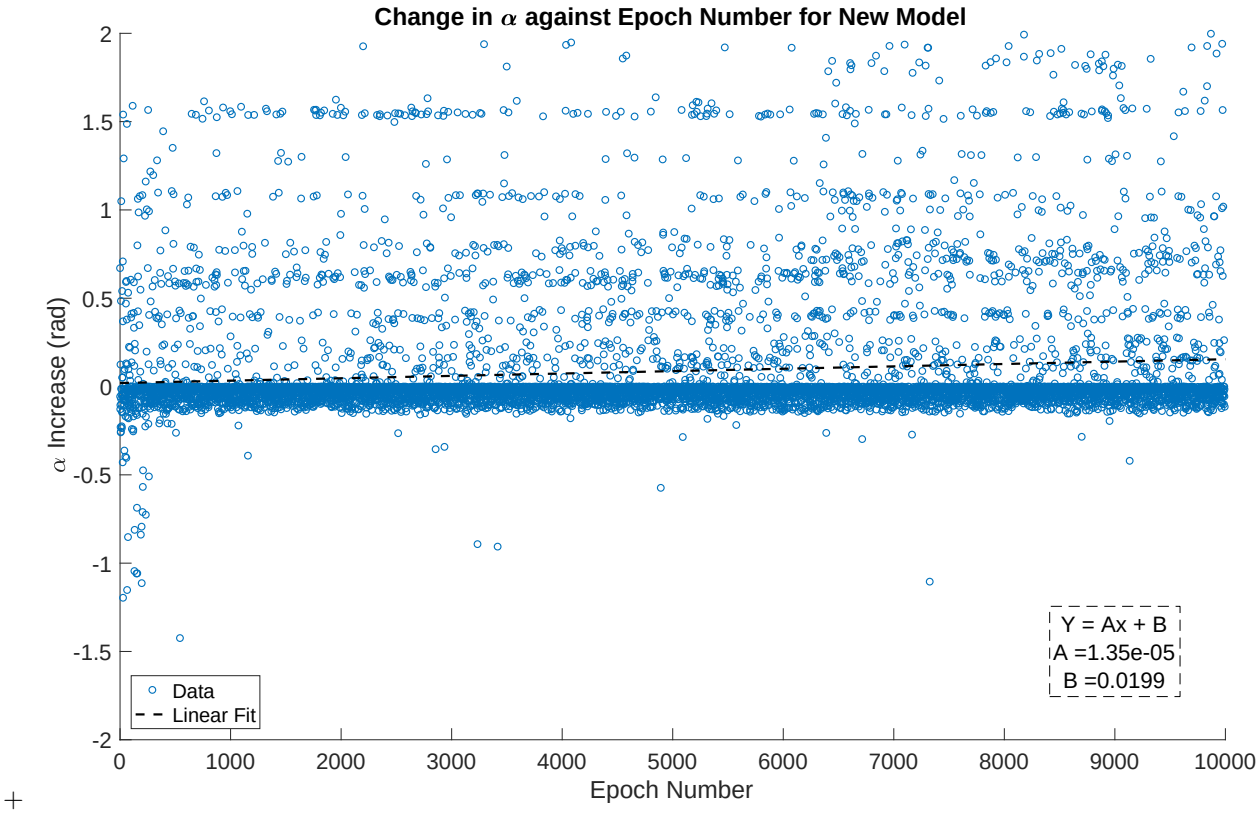


Figure 8.12: New Model

This increase in efficacy may ironically be due to the actions now having a reduced effect on the swinging angle. Now the α gain from one torque is more limited, the agent can perform more actions per cycle, with less fear of pushing over the vertical position and receiving a large negative reward, meaning a bigger total swinging angle can be achieved. There are some more negative actions late in the cycle which is disconcerting, but this is not so much of an issue given the more common positive actions, reflected in how one might note that the gradient of this graph is an order of magnitude higher than figure 8.7, and 2 orders higher than figure 8.8. This could be made more accurate by imposing the limits on θ to be the same as the NAO.

Finally for closure, the case where the swing performs normal random learning, then starts from the same swinging angle is shown in figure 8.13. Here you can see the agent quickly converges on a 'correct' solution, where it makes a significant enough a change in alpha to not want to explore any more. This aspect of Q-Learning could be investigated more by trying to change the q value equation to make the agent less risk averse, so it would aim to reach a higher angle rather than settling for a moderate reward.

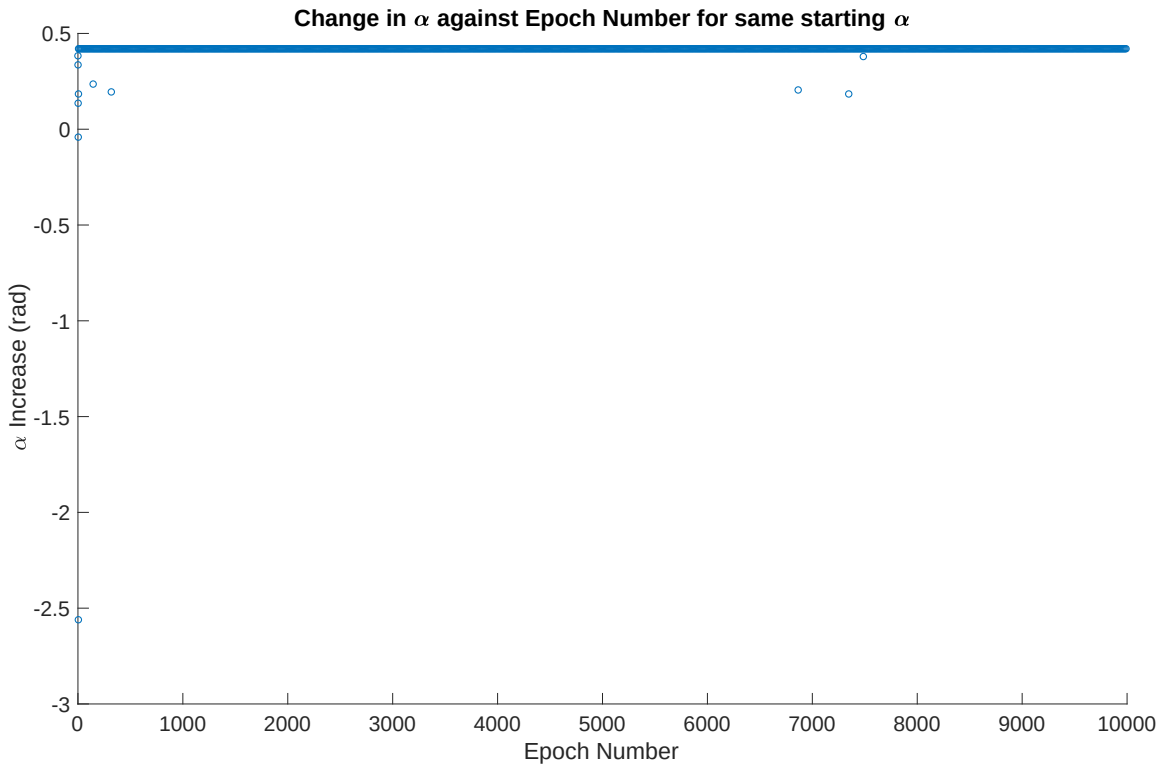


Figure 8.13: New model with the same starting angle after each epoch

8.3.5 Conclusion

To summarise, an effective machine learning algorithm was created that aimed to maximise the swinging angle of a double pendulum in python. An adaptable code was written capable of showing a UI implementation of the model, as well as printing key values to files for analysis. Multiple aspects of this code were changed to improve the efficacy of the Q-Learning algorithm, ending with a model that showed significant improvements from the original, and large differences when compared to purely random actions.

8.4 C

8.4.1 Implementation

Alex Thornton

The lab system was the first to be modelled using the double pendulum environment. The lab swings weights and dimensions were used and the small pendulum was used to drive the larger one. A small State-Action space was used to reduce computation time. As the system that was being modelled was a chaotic one, a modification to the typical Q-learning algorithm was trialled. This was done by omitting terms Q-max and old value. Doing this made it so that old information was treated the same as new information. The downside to this method is that the agent converges to a solution slower, however it is much less likely to get stuck in bad action pathways. These pathways are due to an action having different results on the same state which is useful for chaotic problems, as the state space can never model the system precisely. This became more apparent in the environment which was significantly more chaotic. With this algorithm the Q-space can be thought of as a quasi-probability density space. When performing with an algorithm that treats all information the same, it is important to keep the scope of the reward within the range of the values of the Q-Space.

The State-Action Space used for the lab double pendulum environment had dimensions of, the large Pendulum angle, Velocity direction, and the Action of the small pendulum. Velocity Direction was included to keep the results consistent with an environment which didn't have left-right symmetry such as a robot being on a swing. Initially the Agent didn't have full autonomy over its actions within the state, with actions only being enacted once the velocity of the main swing had changed direction. The main swing angle was split into 1000 discrete states. Velocity direction was a unit vector of 1 or -1 and Action being +1, -1 or 0. The small pendulum was restricted from going into un-physical states so it could only be within the range of $-\pi$ to π so it could never lap itself. The reward system was taken to be the (new large pendulum amplitude - old pendulum amplitude)÷(old pendulum amplitude). The denominator was included in this expression as for this environment, it was believed that shifting the centre of mass was the main driver of amplitude growth and that this shift is known to grow in its effect linearly with increased amplitude.[52] So to normalise the rewards given for good actions the reward would decrease with increased starting amplitude.

The simulation was run first by allowing the agent to preform 6000 random actions. Then the agent could perform another 6000 actions based on what it had learnt from its prior experience. The state was reset after 20 actions so effectively 300 epochs were undertaken. Each time the agent started out at a small, random, angle with negligible velocity. The system was run with different driving amplitudes to probe how this effected the absolute value of swinging achieved and also the efficiency of the algorithm. The results are displayed below.

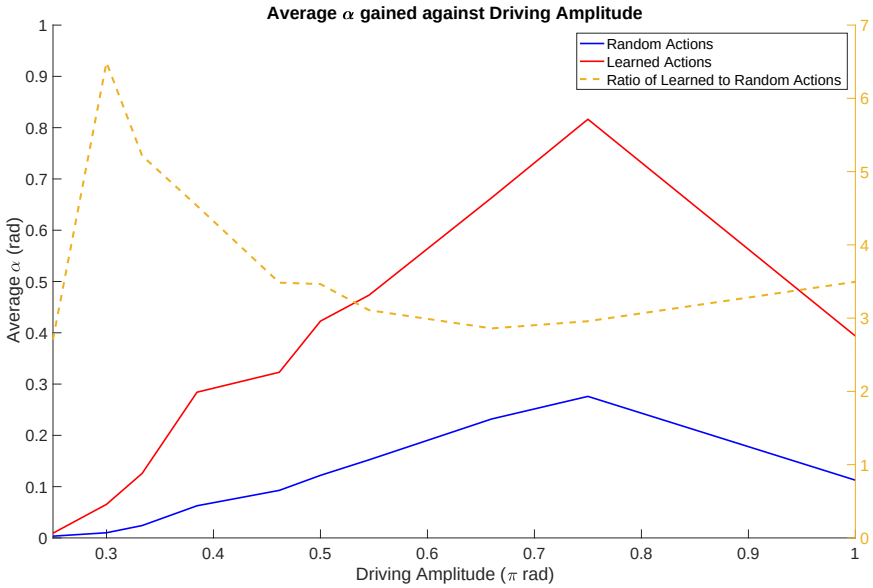


Figure 8.14: Driving Amplitude against Average Angle Gained for Double Pendulum Lab System

As can be seen above the algorithm peaks in efficiency at 0.3π radians, this is believed to be because at lower amplitudes the agent has little influence on the overall state so the difference between good and bad actions is reduced. And at higher amplitudes the system becomes more chaotic so the same action on the same state can produce differing outcomes. Furthermore at consistently higher amplitudes, the agent has much less scope for improvement compared to lower amplitudes. The algorithm has proved to be effective for a relatively stable system such as the lab system. To test how the algorithm deals with more chaos, the lengths of the pendulums were made equal. The results are shown below.

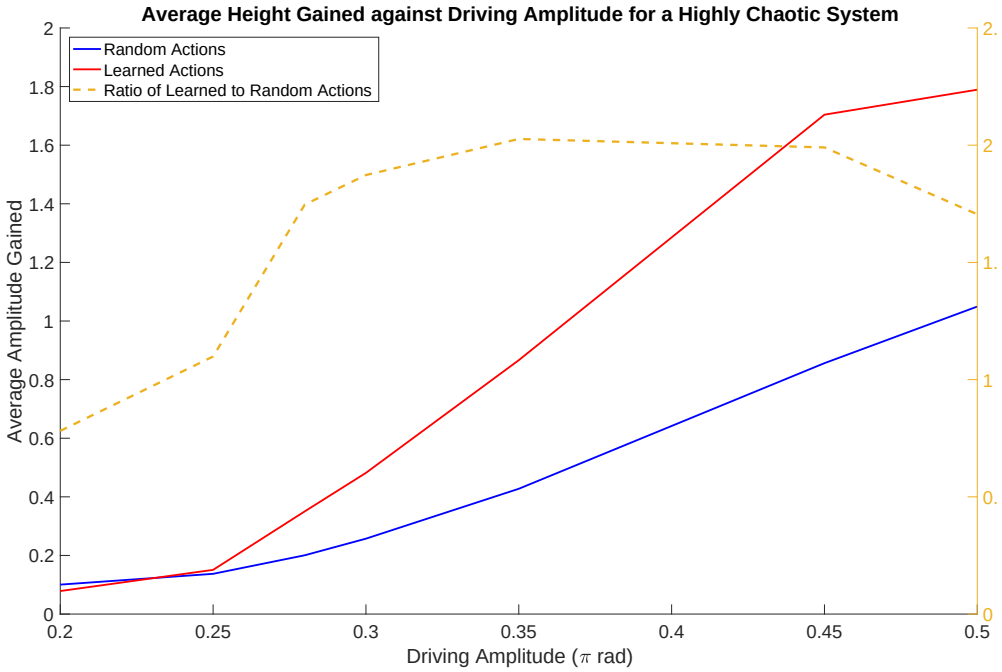


Figure 8.15: Driving Amplitude against Average Angle Gained for Double Pendulum, Equal-Length System

The algorithm is effective at increasing the height obtained by the agent apart from at a discrepancy at 0.2π radians, whereby the agent preforms worse by enacting its learnt actions. Perhaps this is because the amplitude reached is consistently low, so the system is effectively mapped by a small States space, effectively reducing its precision. For larger amplitudes the efficiency of the algorithm remains roughly constant which is a slightly surprising result. For a more qualitative understanding of what is happening within an epoch (20 actions). The graph below shows how the swinging motion is affected by learnt and random actions. The epochs were selected such that there starting alpha was similar and their average swing height achieved was representative of their group as a whole.

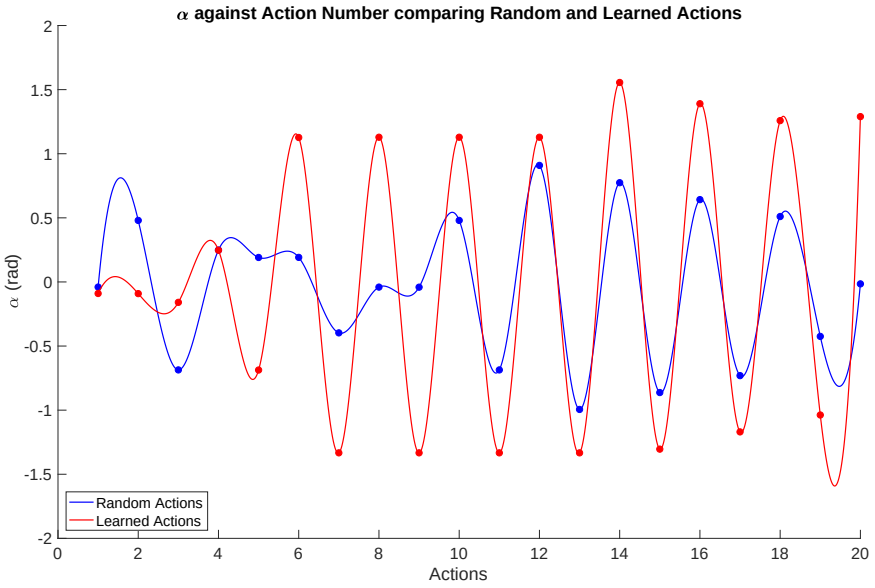


Figure 8.16: Representation of Swinging Motion Within a Epoch for Random and Learnt Actions

As can be seen the random actions make a system that starts stable into something chaotic, then returns to stability with decaying amplitude, typical of what you would expect for random actions. The learned actions make the system stable with a large amplitude.

With the algorithms preforming well given the agents restricted ability on when to perform its actions. It was useful to next explore whether the algorithm could still preform when giving the agent more freedom on when to perform its actions. Hence an extra dimension for the action space was added, a phase space. Allowing the agent to perform an action at any one of four discrete points in its swing cycle. This would again greatly increase the chaos with system.

So to test the increased freedom in action space the more stable lab dumbbell configuration was used. The algorithm performed well, consistently out preforming random actions in a significant way. The same number of actions were used to teach the agent before. Given that the action space is significantly larger it is impressive the agent learned with such efficiency.

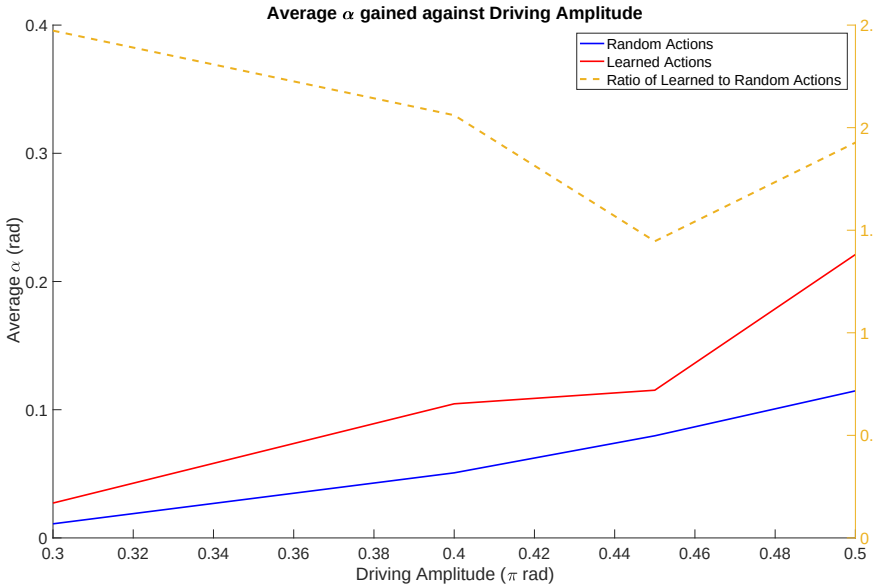


Figure 8.17: Driving Amplitude against Average Angle Gained for Double Pendulum Lab System with Multi-Dimensional Action Space

The Multi-Phase space model was then tried with equal length pendulums. The algorithm struggled to learn at small oscillations. This is probably due to the state being so sensitive, the success of actions as a whole are predicated on the actions that were undertaken before. Without the agent encoding knowledge of previous actions into its Q-Space, it struggled to distinguish between good and bad actions. For this specific case a generalised policy algorithm which is analysed at the end of each epoch would probably prove superior. As then bad actions do not perturb the effectiveness of good actions following them. At larger amplitudes significant learning was observed, this is probably due to the increased stability of the system which would be less affected by bad actions.

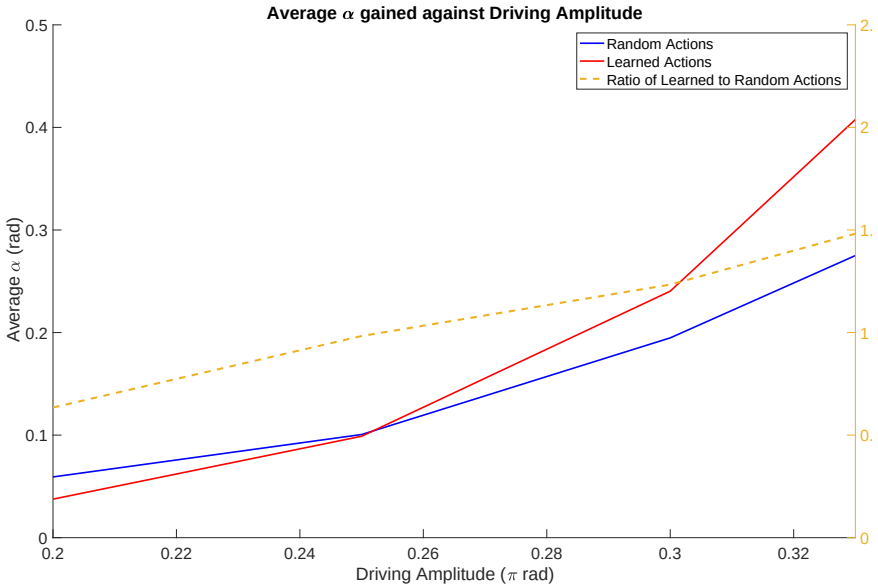


Figure 8.18: Driving Amplitude against Average Angle Gained for Double Pendulum Equal-Length System with Multi-Dimensional Action Space

8.4.2 Conclusion

For a more chaotic system with a lot of degrees of freedom, an evaluating generalised policy algorithm may have worked best. This is due to prior actions making the agents state ill definable and thus making theoretically good actions less effective. For constrained problems with a definable state that behaves consistently with applied action, Q-learning has displayed impressive efficacy. The Q-Learning algorithm produced average swinging amplitudes up to seven times better compared to random motion for the lab-system double pendulum. When the system was made more chaotic, the algorithm still proved effective in creating, stable high amplitude oscillations where the random actions could not. When the constraints on the action space were lifted, allowing the agent to perform an action at four discrete points in the cycle, the learnt actions proved to be significantly better than random actions for the lab dumbbell system. However the algorithm was not as efficient as before for obvious reasons, due to the Action space being larger and the problem being more dimensional. When the multi-dimensional action space was used in conjunction with the equal length double pendulum environment, the agent struggled significantly at low amplitude oscillations. As mentioned before probably due to prior bad actions perturbing the system to make theoretically good actions less effective. However at larger amplitudes, the agent showed signs of learning, preforming better than random actions. This is probably approaching the limits of this algorithms effectiveness with less constrained problems becoming difficult to learn. Q-learning given that it is computationally less intense has advantages of being able to test many iterations in a relatively short time whilst still producing effective learning outcomes.

9 Self-Start

Josh Goldberg

9.1 Introduction

Another way in which the swinging motion of the robot was looked to be improved and more 'human-like' was by allowing the robot to begin its swinging motion autonomously and through no human input. The way humans do this is by pushing themselves back from the ground and then lifting their legs to start swinging. Applying this to the robot was investigated in last year's report with the idea of a raised platform, but was found to be almost impossible to replicate. This is because the robot has very short legs, so is unable to get a significant push when straightening them, and because the robot's available leg movement while seated is very limited compared to a human.[9]

It was decided that there are two feasible ways for the robot to self-start. The first, is for the robot to move back and fourth from rest, gradually building up its amplitude and initiating its swinging motion. The second, is to have a block placed by the robot's feet, which it can kick off. The block needs to be designed so that it then falls over, allowing the robot to continue its swinging motion. The advantage of having the robot build up amplitude from rest is that it means the robot is completely autonomous, and requires no human input. However, when using a block, it needs to be made and put in the correct position before each swing. There are also several advantages of self-starting with a block, due to the fact that it allows the robot to instantly reach a significant initial amplitude. Firstly, as the robot's motors can overheat after a few minutes of continuous use, kicking off the block prevents this from happening before a maximum amplitude has been reached. Furthermore, the use of the block means that more tests can be done on the robot, as less time and battery is wasted waiting for the robot to build up to a significant amplitude. Finally, it is arguably more 'human-like' for the robot to use a block. When a human begins a swing, they kick off from the ground, which, as mentioned earlier, is not possible for the robot. Therefore, the block acts as a substitute for the ground, allowing the robot to kick off and instantly begin swinging, as a human would. Therefore, building up amplitude from rest would be the best way for the robot to swing autonomously, whereas self-starting through the use of a block may allow the robot to reach its greatest amplitude, whilst also being more 'human-like'.

9.2 Investigating the Use of a Block

Firstly, it was necessary to investigate what kind of amplitudes the robot would be able to reach from kicking off the block. It was clear that the optimum starting position of the block and robot would be for the legs to be bent fully back underneath the robot, with the feet pointing down and the block in front of the feet, as close as possible. It was initially calculated that if the block stayed upright, and the robot straightened its legs and rested on the block, the swing would reach an amplitude of approximately 9°. In reality, due to the block needing to fall over before the legs have straightened, the block will lean away and the legs won't be straight at the last point of contact, meaning the robot will not reach 9°. It was difficult to calculate exactly what initial amplitude would be achieved, but a prediction that it should be able to reach approximately half of 9° seemed reasonable, which would give an amplitude that can't be reached quickly from rest. The motion was also simulated in Webots, which showed that the robot could successfully kick the block over and reach a decent amplitude, thus making the method worth pursuing. This is later explained in detail in section 9.4.

The next step was to design a block that would maximise the initial amplitude. By looking at the block size required for the force produced by the legs to only just knock it over, the rough dimensions could be found. For a block of mass m, height h, width w, depth d (from the perspective of the robot sitting behind) and density ρ, with its centre of mass in the centre and a force F at the top of the block, it will fall when the destabilising moment, $F\frac{h}{2}$, is greater than the stabilising moment, $mg\frac{d}{2}$, where g is the acceleration of gravity on Earth. By substituting in $m = hwd\rho$, it was found that the block will fall over if $d^2w < \frac{F}{\rho g}$. In last year's report it was measured that the torque, τ, at each knee was approximately 4 Nm, while the

distance, d, from the knee to the foot (where it will kick from) is 10cm. Using $F = \frac{\tau}{d}$, the maximum force from both knees is approximately 80N. Setting the width to 25cm (so both feet impact on the block) and the density to 500 kgm^{-3} (that of wood), this gives a depth of 26cm. It was also measured that the height of the block needs to be 50cm, so that the feet are pushing on the top of the block. This was important in order to know the rough dimensions of the block, but it would still need to be tested on the robot and adjusted.

9.3 Constructing the block

It was decided that it would be advantageous to use a cardboard box with masses inside, rather than a block of wood or similar material. One reason for this was that looking at the rough dimensions required, a very large amount of material would have been needed, which would be difficult to obtain. However, the main advantage was that it meant that it would be easy to choose and adjust the dimensions of the block, its total mass and most importantly, the position of the centre of mass. Therefore, after testing the block on the robot, adjustments could be made to make the block easier or more difficult to push over.

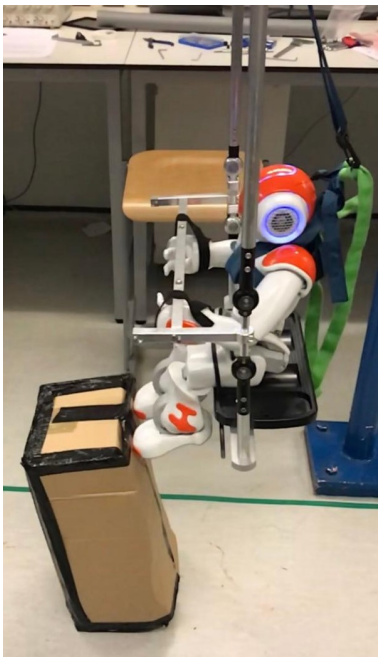


Figure 9.1: Image of real robot self-starting using the block

Using scissors and duct tape, a large cardboard box was reduced in size to dimensions of 50x25x20 cm, before smaller boxes were placed inside with dense weights attached to the top. By using dense weights, the centre of mass of the block was approximately at the centre of mass of the weights, so could therefore be chosen. When testing the block on the robot, some problems were initially encountered. Firstly, although the block was the correct height in earlier tests, once many adjustments had been made it was slightly too small to reach the robots feet. This is believed to be due to the fact that tape kept needing to be put on and taken off to close and open the box, which, over time, forced the front, top edge to be pulled down. To resolve this, a brick from the lab was placed underneath the block to give it the correct height. On top of this, when initially testing the block after increasing the mass, it would begin to tip away from the robot, before falling back the opposite direction under the seat. After opening up the block, it was found that this was due to the extra mass causing the cardboard box inside to fold and lean to the other side. This meant that the centre of mass was in fact closer to the robot's feet than the centre, causing it to fall in the opposite direction to that required. To avoid this, a stronger cardboard box was used on the inside, that could withstand the mass.

The final block that was used on the robot had (5.8 $\pm$ 0.2) kg inside, with the centre of mass positioned (5 $\pm$ 2) cm from the top and back of the block, and in the centre of the width. Therefore, the relatively heavy mass meant that the block was difficult to kick over, and stayed upright for the majority of the kick, whereas the position of the centre of mass meant that once it began to tip, it would easily fall over. This maximised the initial amplitude that the robot would reach, because the more upright the block was at the last point of contact with the feet, the further the robot would be pushed back.

9.4 Simulating self-start in Webots

Ronan Dubois

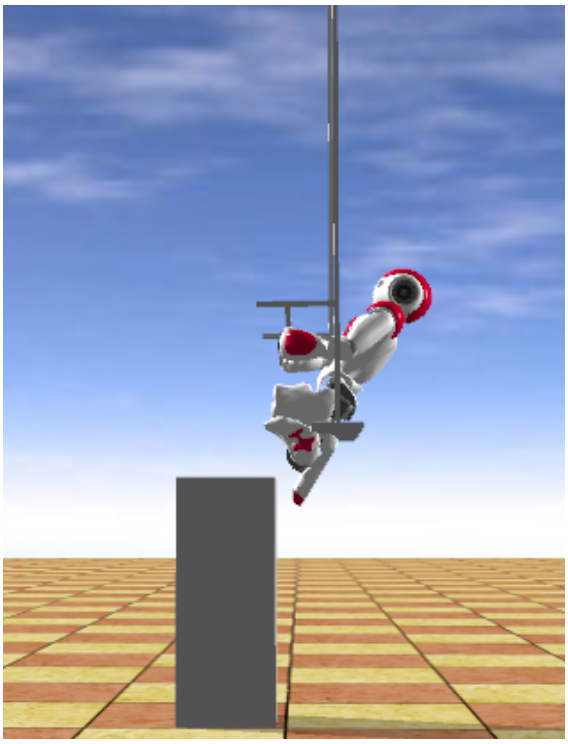


Figure 9.2: Side view of the initial self-start configuration in Webots

The self-start situation was modelled in Webots in order to put its effectiveness to the test, using first the solid rod model and, later on, the new double-hinged swing model (Figure 9.2). A simple block was created in Webots using a *Solid* node with a *Box* geometry. The coordinates of the block's centre of mass were defined in its *Physics* node parameters and densities ranging from 300 kgm^{-3} to 3000 kgm^{-3} were tested to simulate different possible block materials. The robot was placed on the swing in a seated position and its legs and feet were brought back by angles of 95° and 53° respectively, corresponding to the maximum realistic angles. The simulation was then paused whilst the block was moved in front of the robot's feet. Finally, the legs were extended through the maximum angle in Choregraphe and the simulation was resumed, allowing the robot to topple the block, which resulted in a satisfactory initial swinging amplitude.

Later on in the project, amplitudes of up to (37.0 ± 0.3)° were obtained in Webots when the robot started off with a small initial amplitude, which further justified the use of a block to self-start. A virtual block of dimensions (20 x 25 x 50) cm was used in Webots, replicating the box devised by Josh. A mass of 5.8 kg was used and the centre of mass was raised to 5 cm from the top of the block and moved to the outer edge of the block by 5 cm, allowing the virtual robot to topple the block. A maximum initial amplitude of (6.9 ± 0.3)° was obtained with this configuration.

9.5 Implementing self-start on NAO

Ryan Bouab

After successfully simulating the self-start in Webots, the final stage of its implementation was to apply it to the real robot NAO on the double-hinged swing using PositionDynamicScript.py. To do so, a self-start function was created by modifying the SwingAPI to also include a treatment of the AnklePitch angle (in addition to hips, head and knees angles). This was done on Choregraphe to measure the maximum AnklePitch value for which the feet of the robot are pointed outwards. This process was done to ensure that the robot's feet could reach the box and kick off it. This API was called in the dynamic period script before the while loop was started thus allowing the robot to kick off the block before starting swinging from position feedback. Amplitudes of up to $(4.2 \pm 0.2)^\circ$ were obtained using the self-start function on the real robot. This initial amplitude was heavily dependent on how close the robot's feet were to the block and in which position the robot was self-starting. Indeed the robot had to be positioned in the seated position with its knees bent and its torso upright and as close as possible to the handle bar. This was done so that when the script was started the robot would move its torso and head back and extend its arms and legs generating a stronger torque. Future work should investigate the most efficient way to position the block and the robot in order to maximise the initial amplitude produced by this function.

9.6 Analysis

Josh Goldberg

The initial amplitude reached in Webots was encouraging, and not much lower than the calculated maximum angle of 9° that could be reached from resting on an upright block with straight legs. The slightly reduced initial amplitude reached in real life was expected, due to inherent problems that didn't occur in the ideal simulation of Webots. One possible reason for this is that, as mentioned in section 2.9.1, the torque produced by the physical robot is less than that of the robot simulated in Webots. Therefore, the robot feels less of a push off the block and does not reach as great an amplitude. Furthermore, if the kick is not perfectly symmetrical, the real swing can wobble about the vertical axis. This wastes energy on unwanted movement, thus pushing the robot less far back. On top of this, there is also friction in the real swing, as well as a harness that can sometimes tug on the robot and stop it reaching its maximum amplitude. Despite this, the robot did successfully self-start using the block, giving it a significant initial amplitude which it could then spend the rest of the swing working to increase and maximise.

In future projects, a new block could be devised using similar parameters and materials, due to the current block not being tall enough to stand alone and becoming quite worn after multiple adjustments were made. Also, the use of the ankles could be used to rotate the feet during the kick, which would apply an additional torque on the block. This could be investigated in order to find out whether it has a significant effect on the initial amplitude reached. Finally, it could be investigated whether any changes could be made to the seat, in order to allow the robot to have its legs further back below the seat. This would not only allow the block to be placed further back, causing the self-start amplitude to increase, but also the legs would rotate more for the entirety of the swing, thus increasing the torque applied by each rotation.

10 Conclusions

Eleanor Tregoning

The project utilised multiple methods to achieve its initial aims. Firstly, two different means were identified to model the swing in order to provide insights into optimum swinging motion. These two methods were theoretical modelling and human modelling and ran concurrently. Models were designed and identified to best describe a robot on a swing. The numerical modelling concluded that the best way to gain the highest amplitude was for the driving function to be $\frac{\pi}{2}$ ahead of the swing motion, for the motion to be as fast as possible and over the largest possible range of motion. The ideal driving functions were adapted to ramp functions to fit the limitations of the robot. To improve the models, the masses of the rods could be added as they were treated as light. The human modelling was conducted to find the optimal motion for a human using a swing. The mass of the human was altered to be within 5% of the robot mass and the effect of this was compared with the motion where no mass was added. It was found that there was little difference between the two, it was concluded that in order to compensate for the added mass the swing user exerts more effort, but over a larger period of time resulting in no net effect on swinging motion. The angles of the knee and torso were found as a function of the local time period and the fraction of the swing angle for which the torque should be applied was implemented on the robot. This resulted in a swing amplitude of $(13.0 \pm 0.2)^\circ$ using the block to self-start.

Four new external sensors were built to add to the new hinges on the swing. Despite setbacks meaning the encoders took longer to build than anticipated, the extracted data confirmed that they were calibrated and functioned correctly. Data was also taken whilst the robot was on the swing but without a driving function using the internal sensors. The aim of this was to improve the raw data and to assess its usefulness as a form of feedback. A focus of this project was on filtering techniques which vastly improved the viability of the acceleration data in terms of feedback. An investigation was conducted to combine data from the gyroscope and accelerometer to provide an alternative to using raw data or the *Torso Angle* algorithm data.

As in previous years, this project encountered compatibility issues between versions of Webots, Python SDK and Choregraphe. In addition, only having one full Webots licence meant resources were limited. Efforts were simultaneously focused into solving these issues and on improving the hinged swing environment in Webots. This resulted in a better environment with an improved use of connectors providing a swing more accurate to the real one and also enabling the motion of the robot's upper body. In this environment, scripts were run simulating the system with predetermined motion. An amplitude of $(37 \pm 0.3)^\circ$ was reached with an initial angle of $(8.0 \pm 0.3)^\circ$. Following this, scripts using a dynamic period were implemented starting from rest and achieving an amplitude of $(16.0 \pm 0.3)^\circ$. Work was also carried out investigating ways of implementing self-starts. This was done by starting from rest and kicking off a box. This box was added into the environment in order to simulate the self-start mechanism and achieved an amplitude of $(6.9 \pm 0.3)^\circ$. All of the aforementioned scripts were run on the robot. In each case the results were slightly lower, this was expected due to the added complexities of the real model such as non-symmetric motion or mass distribution. Starting from rest yielded a result of $(14.8 \pm 0.2)^\circ$ whilst an initial amplitude of $(4.2 \pm 0.2)^\circ$ was achieved from kicking off a box.

Machine learning was investigated as a means of improving autonomy. Learning from previous years conclusions regarding different machine learning algorithms, Q-learning was selected as the best algorithm to use and was applied to a double pendulum model. This was done in both C and Python using different q-spaces for each. Overall, a clear difference between random and learnt actions in a double pendulum environment was demonstrated in both languages using the Q-Learning methods. Different characteristics of the models were investigated in depth to achieve a more consistent, higher swinging angle.

10.1 Project Evaluation and Outlook

Many of the problems faced in this project were similar to previous years. Only having one permanent Webots licence meant limited access to simulations meaning optimisation could not be completed to its full potential with many tasks taking longer than planned. However, compatibility issues were the main source of set-backs with scripts not being run successfully on the robot until week 5. To help future groups section 6 offers a useful guide to help with software installation processes. Due to limited access to the robot it was difficult to assign and find useful tasks at the beginning of the project. The first few weeks could have been structured better and dedicated to a more thorough research of previous project work to help identify areas of investigation; this would have provided more time to build the new encoders. Despite these initial setbacks once the necessary areas of investigations had been decided the project went smoothly; deadlines and assigned timescales were all adhered to. Excellent communication by all project members ensured possible problems were avoided and that the schedule was edited accordingly to provide enough time to solve problems.

One major difference between all of our models and the real swing was the amount of sideways motion present in the real swing. It was evident simply from a visual assessment of the swing and could be measured quantitatively using the inertial sensors and new encoders. This sideways motion was identified in both section 3.8 and 5.6.

as an area of improvement. The sideways motion results in reduced amplitude gain as energy which could be used in pumping the swing is wasted. Numerical models could be extended to the four bar model or even to three dimensions in order to investigate the theoretical effects of these energy losses. Both the external and internal sensors could be used to measure the extent of this motion and practical changes could be made to the swing or swinging method in order to reduce this motion.

11 Acknowledgements

We would like to express our sincere gratitude to Dr. Miguel Navarro-Cía and Ross Griffin for their support supervising the project. Their insights were invaluable in helping develop our understanding of the project. Thanks should also go to Dr. Mark Colclough for his assistance coding the encoders, the workshop staff and to Dr. Dimitri Gangardt for responding to our enquiry in relation to solving the 4 Bar Model.

Special thanks should go also to Dr Wolfgang Theis for his overseeing of the laboratory.

References

[1] G. Dalakov. (1999) Trumpet Player of Friedrich Kaufmann. [Online]. Available: https://history-computer.com/Dreamers/Kaufmann.html

[2] B. Specktor. (2018) Meet Erica, Japan's Next Robot News Anchor. [Online]. Available: https://www.livescience.com/61575-erica-robot-replace-japanese-news-anchor.html

[3] M. Irving. (2017) Google's AI beats world's top-ranking Go player. [Online]. Available: https://newatlas.com/alphago-defeats-ke-jie-go-victory/49675/

[4] S. Lawrence, "Pumping swing image," 2005. [Online]. Available: http://www.physicsinsights.org/images/swing-leanback-2.png

[5] Ivanovich, "Lagrange's equations," 2009. [Online]. Available: https://uk.mathworks.com/matlabcentral/fileexchange/23037-lagrange-s-equations?requestedDomain=true

[6] D. B. et al., "Teaching an Aldebaran NAO Robot to Swing," University of Birmingham, Tech. Rep., 2016.

[7] E. W. Weisstein. "runge-kutta method." from mathworld–a wolfram web resource. [Online]. Available: http://mathworld.wolfram.com/Runge-KuttaMethod.html

[8] Math24. (2018) Differential Equations Systems of Equations: Double Pendulum. [Online]. Available: https://www.math24.net/double-pendulum/

[9] M. K. et al., "Optimising Swinging Motion with a NAO Robot," University of Birmingham, Tech. Rep., 2017.

[10] R. Nave. (1998) Resonance. [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/reson.html

[11] R. O. Lagerlöf, "Interpolation with Rounded Ramp Functions," Chalmers University of Technology, year = 1974,, Tech. Rep.

[12] A. Robotics. Masses. [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/masses_robot.html

[13] A. Robotics. Links. [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/links_robot.html

[14] A. Robotics. Joints. [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/joints_robot.html

[15] A. Robotics. Motors. [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/motors_robot.html

[16] S. Wirkus, R. Rand, and A. Ruina, "How to pump a swing," *The College Mathematics Journal*, vol. 29, no. 4, p. 266, 1998.

[17] A. Tözeren, *Human body dynamics: classical mechanics and human movement.* Springer Science & Business Media, 1999.

[18] MAGIX. (2018) MAGIX - Movie Edit Touch Windows Application . [Online]. Available: https://www.microsoft.com/en-gb/store/p/movie-edit-touch

[19] K. Ltd. (2018) Kinovea: A microscope for your videos. [Online]. Available: http://www.kinovea.org

[20] M. Christiano. (2016) Gray Code Basics. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/gray-code-basics/

[21] A. Technologies. (2011) AEAT-6010/6012 Magnetic Encoder Data Sheet. [Online]. Available: http://www.farnell.com/datasheets/1884437.pdf?_ga=2.195742564.958583698.1517927087-339883265.1516977895&_gac=1.121415162.1517495544.CjwKCAiAksvTBRBFEiwADSBZfGIMrNr3FMZcM8FeCDY4YCLJIaGL2DMB2zo-HApy9yO0CENtx05L7BoCPQkQAvD_BwE

[22] M. T. Inc. (2012) MCP2210 Evaluation Kit User's Guide. [Online]. Available: http://www.farnell.com/datasheets/1681461.pdf?_ga=2.161664884.958583698.1517927087-339883265.1516977895&_gac=1.114868725.1517495544.CjwKCAiAksvTBRBFEiwADSBZfGIMrNr3FMZcM8FeCDY4YCLJIaGL2DMB2zo-HApy9yO0CENtx05L7BoCPQkQAvD_BwE

[23] (2018). [Online]. Available: http://uk.farnell.com

[24] I. Skog and P. Händel, "Calibration of a mems inertial measurement unit," in *in Proc. XVII IMEKO WORLD CONGRESS, (Rio de Janeiro)*, 2006.

[25] A. Robotics. (2017) Inertial Unit. [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/inertial_robot.html

[26] A. Robotics. (2013) Inertial Unit. [Online]. Available: http://doc.aldebaran.com/1-14/family/robots/inertial_robot.html#robot-inertial

[27] Learn.sparkfun.com. (2017) Accelerometer Basics - learn.sparkfun.com. [Online]. Available: https://learn.sparkfun.com/tutorials/accelerometer-basics

[28] T. O'Haver, "Foufilter.m," 2007. [Online]. Available: https://terpconnect.umd.edu/~toh/spectrum/FourierFilter.html#example

[29] T. O'Haver. (2018) A Pragmatic Introduction to Signal Processing with applications in scientific measurement . [Online]. Available: https://terpconnect.umd.edu/~toh/spectrum/IntroToSignalProcessing.pdf

[30] P.-J. V. de Maele. (2013) Reading a IMU Without Kalman: The Complementary Filter. [Online]. Available: http://www.pieter-jan.com/node/11

[31] P. Gui, L. Tang, and S. Mukhopadhyay, "Mems based imu for tilting measurement: Comparison of complementary and kalman filter based data fusion," vol. 26, pp. 2004–2009, 06 2015.

[32] Cyberbotics. (2018) Webots User Guide R2018a - Introduction. [Online]. Available: https://www.cyberbotics.com/doc/guide/introduction

[33] Cyberbotics. (2018) Webots Reference Manual R2018a - Transform. [Online]. Available: https://www.cyberbotics.com/doc/guide/introduction

[34] Cyberbotics. (2018) Webots Reference Manual R2018a - Connector. [Online]. Available: https://www.cyberbotics.com/doc/guide/introduction

[35] Aldebaran. (2013) Key concepts. [Online]. Available: http://doc.aldebaran.com/2-1/dev/naoqi/index.html#naoqi-framework-overview

[36] R. Levien and S. Tan, "Double pendulum: An experiment in chaos," *American Journal of Physics*, vol. 61, no. 11, pp. 1038–1044, 1993.

[37] S. I. inc. (2018) Machine Learning: What it is and why it matters . [Online]. Available: https://www.sas.com/en_gb/insights/analytics/machine-learning.html

[38] E. S. S.p.A. (2018) What is Machine Learning? A definition . [Online]. Available: http://www.expertsystem.com/machine-learning-definition/

[39] D. Faggella. (2017) What is Machine Learning? . [Online]. Available: https://www.techemergence.com/what-is-machine-learning/

[40] UBM. (2000) Computer Technology Helps Radiologists Spot Overlooked Small Breast Cancers . [Online]. Available: http://www.cancernetwork.com/articles/computer-technology-helps-radiologists-spot-overlooked-small-breast-cancers

[41] S. J. L. Q. A. D. D. K, "Computer-aided diagnosis and artificial intelligence in clinical imaging," 11 2011.

[42] M. S. K. C. Center. (2018) IBM Watson and Quest Diagnostics Launch Genomic Sequencing Service Using Data from MSK. [Online]. Available: https://www.mskcc.org/ibm-watson-and-quest-diagnostics-launch-genomic-sequencing-service-using-data-msk

[43] M. B. Dominik Wee. (2015) Ten ways autonomous driving could redefine the automotive world. [Online]. Available: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/ten-ways-autonomous-driving-could-redefine-the-automotive-world

[44] K. Kokalitcheva. (2017) Humans cause most self-driving car accidents . [Online]. Available: https://www.axios.com/humans-cause-most-self-driving-car-accidents-1513304490-02cdaf3d-551f-46e6-ad98-637e6ef2c0b9.html

[45] O. U. Press. (2018). [Online]. Available: https://en.oxforddictionaries.com/definition/robot

[46] E. D. Science. (2017) Machine Learning Explained: supervised learning, unsupervised learning, and reinforcement learning. [Online]. Available: http://enhancedatascience.com/2017/07/19/machine-learning-explained-supervised-learning-unsupervised-learning-and-reinforcement-learning/

[47] J. Brownlee. (2016) Supervised and Unsupervised Machine Learning Algorithms. [Online]. Available: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms//

[48] V. Maini. (2017) Machine Learning for Humans, Part 5: Reinforcement Learning. [Online]. Available: https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265

[49] R. S. Sutton, "Reinforcement learning: An introduction," 2016. [Online]. Available: ⟨https://webdocs.cs.ualberta.ca/~sutton/book/bookdraft2016sep.pdf⟩

[50] (2018) OpenAI Gym. [Online]. Available: https://github.com/openai/gym

[51] Y. Lu. (2017) Solving OpenAI Gym Cartpole with Q-learning. [Online]. Available: http://178.79.149.207/posts/cartpole-qlearning.html

[52] S. W. et al., "How to Pump a Swing," Cornell University, Tech. Rep., 1998.

[53] H. ÜLKER. (2010) DYNAMIC ANALYSIS OF FLEXIBLE MECHANISMS BY FINITE ELEMENT METHOD. [Online]. Available: http://library.iyte.edu.tr/tezler/master/makinamuh/T000879.pdf

[54] C. P. Tang. (2010) Lagrangian dynamic formulation of a four-bar mechanism with minimal coordinates. [Online]. Available: http://www.usp.br/ldsv/wp-content/uploads/2014/10/Fourbar_Lagrange_10.pdf

A Appendix

A.1 Complex Hinged Flail Derivation

Christopher Hulme

The Lagrangian of the system can be described by generalized coordinates α, θ_1, θ_2, θ_3, θ_4 and θ_5 and their respective angular velocities as is given by equation A.1.

$$\mathcal{L} = \frac{1}{2}\begin{pmatrix} m_1(l_\alpha cos(\alpha)\dot{\alpha} + l_1 cos(\theta_1)\dot{\theta_1})^2 + m_1(l_\alpha sin(\alpha)\dot{\alpha} + l_1 sin(\theta_1)\dot{\theta_1})^2 \\ +m_2(l_\alpha cos(\alpha)\dot{\alpha} + l_1 cos(\theta_1)\dot{\theta_1} + l_2 cos(\theta_2)\dot{\theta_2})^2 + m_2(l_\alpha sin(\alpha)\dot{\alpha} + l_1 sin(\theta_1)\dot{\theta_1} + l_2 sin(\theta_2)\dot{\theta_2})^2 \\ +m_3(l_\alpha cos(\alpha)\dot{\alpha} + l_1 cos(\theta_1)\dot{\theta_1} + l_3 cos(\theta_3)\dot{\theta_3})^2 + m_3(l_\alpha sin(\alpha)\dot{\alpha} + l_1 sin(\theta_1)\dot{\theta_1} + l_3 sin(\theta_3)\dot{\theta_3})^2 \\ +m_4(l_\alpha cos(\alpha)\dot{\alpha} + l_1 cos(\theta_1)\dot{\theta_1} + l_2 cos(\theta_2)\dot{\theta_2} + l_4 cos(\theta_4)\dot{\theta_4})^2 \\ +m_4(l_\alpha sin(\alpha)\dot{\alpha} + l_1 sin(\theta_1)\dot{\theta_1} + l_2 sin(\theta_2)\dot{\theta_2} + l_4 sin(\theta_4)\dot{\theta_4})^2 \\ +m_5(l_\alpha cos(\alpha)\dot{\alpha} + l_1 cos(\theta_1)\dot{\theta_1} + l_3 cos(\theta_3)\dot{\theta_3} + l_5 cos(\theta_5)\dot{\theta_5})^2 \\ +m_5(l_\alpha sin(\alpha)\dot{\alpha} + l_1 sin(\theta_1)\dot{\theta_1} + l_3 sin(\theta_3)\dot{\theta_3} + l_5 sin(\theta_5)\dot{\theta_5})^2 \end{pmatrix}$$
$$+gm_1(l_\alpha cos(\alpha) + l_1 cos(\theta_1)) + gm_2(l_\alpha cos(\alpha) + l_1 cos(\theta 1) + l_2 cos(\theta_2)) + gm_3(l_\alpha cos(\alpha) + l_1 cos(\theta_1) + l_3 cos(\theta_3))$$
$$+gm_4(l_\alpha cos(\alpha) + l_1 cos(\theta_1) + l_2 cos(\theta_2) + l_4 cos(\theta_4)) + gm_5(l_\alpha cos(\alpha) + l_1 cos(\theta_1) + l_3 cos(\theta_3) + l_5 cos(\theta_5)) \tag{A.1}$$

Equations A.2 and A.3 show the solutions to the equations of motion for the pivot and hinge respectively

$$\ddot{\alpha} = \frac{-1}{M_T l_\alpha}\begin{pmatrix} \dot{\theta_1}^2 M_T l_1 sin(\alpha - \theta_1) + \dot{\theta_2}^2 (m_2 + m_4) l_2 sin(\alpha - \theta_2) + \dot{\theta_3}^2 (m_3 + m_5) l_3 sin(\alpha - \theta_3) \\ +\dot{\theta_4}^2 m_4 l_4 sin(\alpha - \theta_4) + \dot{\theta_5}^2 m_5 l_5 sin(\alpha - \theta_5) + M_T g sin(\alpha) \\ +\ddot{\theta_1} M_T l_1 cos(\alpha - \theta_1) + \ddot{\theta_2}(m_2 + m_4) l_2 cos(\alpha - \theta_2) \\ +\ddot{\theta_3}(m_3 + m_5) l_3 cos(\alpha - \theta_3) + \ddot{\theta_4} m_4 l_4 cos(\alpha - \theta_4) + \ddot{\theta_5} m_5 l_5 cos(\alpha - \theta_5) \end{pmatrix} \tag{A.2}$$

$$\ddot{\theta_1} = \frac{-1}{M_T l_1}\begin{pmatrix} -\dot{\alpha}^2 M_T l_\alpha sin(\alpha - \theta_1) + \dot{\theta_2}^2 (m_2 + m_4) l_2 sin(\theta_1 - \theta_2) + \dot{\theta_3}^2 (m_3 + m_5) l_3 sin(\theta_1 - \theta_3) \\ +\dot{\theta_4}^2 m_4 l_4 sin(\theta_1 - \theta_4) + \dot{\theta_5}^2 m_5 l_5 sin(\theta_1 - \theta_5) + M_T g sin(\theta_1) \\ +\ddot{\alpha} M_T l_\alpha cos(\alpha - \theta_1) + \ddot{\theta_2}(m_2 + m_4) l_2 cos(\theta_1 - \theta_2) \\ +\ddot{\theta_3}(m_3 + m_5) l_3 cos(\theta_1 - \theta_3) + \ddot{\theta_4} m_4 l_4 cos(\theta_1 - \theta_4) + \ddot{\theta_5} m_5 l_5 cos(\theta_1 - \theta_5) \end{pmatrix} \tag{A.3}$$

Here M_T is the total mass of $m_{1,2,3,4,5}$ as described in previous models. When one decouples the differential equations, such that $\ddot{\alpha}$ is independent of $\ddot{\theta_1}$, one will obtain the following equations of motion for $\ddot{\alpha}$ and $\ddot{\theta_1}$:

$$\ddot{\alpha} = \frac{BE - C}{1 - BD} \tag{A.4}$$

$$\ddot{\theta_1} = \frac{DC - E}{1 - BD} \tag{A.5}$$

Where B, C, D and E are described below

$$B = \frac{1}{l_\alpha}\left(l_1 cos(\alpha - \theta_1)\right) \tag{A.6}$$

$$C = \frac{1}{M_T l_\alpha}\begin{pmatrix} \dot{\theta_1}^2 M_T l_1 sin(\alpha - \theta_1) + \dot{\theta_2}^2 (m_2 + m_4) l_2 sin(\alpha - \theta_2) + \dot{\theta_3}^2 (m_3 + m_5) l_3 sin(\alpha - \theta_3) \\ +\dot{\theta_4}^2 m_4 l_4 sin(\alpha - \theta_4) + \dot{\theta_5}^2 m_5 l_5 sin(\alpha - \theta_5) + M_T g sin(\alpha) + \ddot{\theta_2}(m_2 + m_4) l_2 cos(\alpha - \theta_2) \\ +\ddot{\theta_3}(m_3 + m_5) l_3 cos(\alpha - \theta_3) + \ddot{\theta_4} m_4 l_4 cos(\alpha - \theta_4) + \ddot{\theta_5} m_5 l_5 cos(\alpha - \theta_5) \end{pmatrix} \tag{A.7}$$

$$D = \frac{1}{l_1}\left(l_\alpha cos(\alpha - \theta_1)\right) \tag{A.8}$$

$$E = \frac{1}{M_T l_1} \begin{pmatrix} -\dot{\alpha}^2 M_T l_1 sin(\alpha - \theta_1) + \dot{\theta_2}^2 (m_2 + m_4) l_2 sin(\theta_1 - \theta_2) + \dot{\theta_3}^2 (m_3 + m_5) l_3 sin(\theta_1 - \theta_3) \\ + \dot{\theta_4}^2 m_4 l_4 sin(\theta_1 - \theta_4) + \dot{\theta_5}^2 m_5 l_5 sin(\theta_1 - \theta_5) + M_T g sin(\theta_1) + \ddot{\theta_2}(m_2 + m_4) l_2 cos(\theta_1 - \theta_2) \\ + \ddot{\theta_3}(m_3 + m_5) l_3 cos(\theta_1 - \theta_3) + \ddot{\theta_4} m_4 l_4 cos(\theta_1 - \theta_4) + \ddot{\theta_5} m_5 l_5 cos(\theta_1 - \theta_5) \end{pmatrix} \tag{A.9}$$

A.2 Deriving the 4 Bar model

Christopher Hulme

This section will briefly discuss how to derive the 4-bar model's Lagrangian.

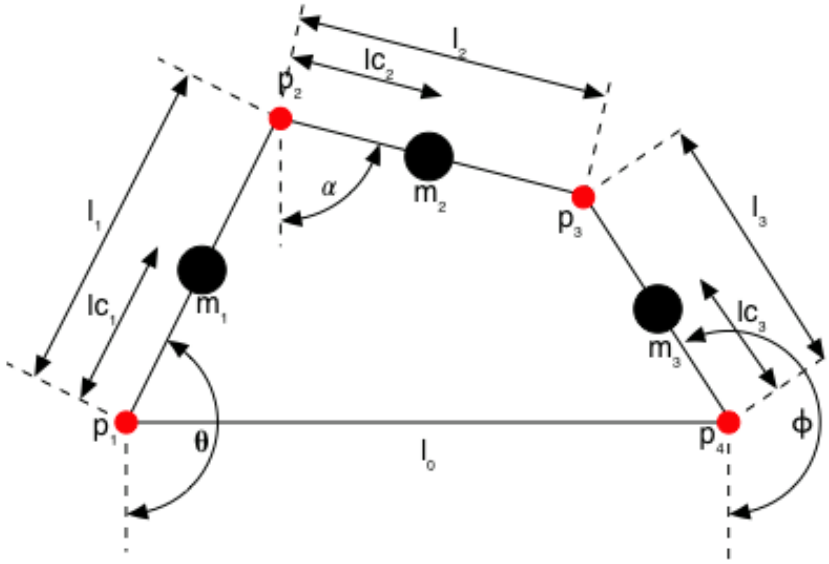


Figure A.1: 4 Bar Model

It is worth noting that the lower pivot points (red dots) in the diagram above are fixed in place, meaning the rod l_0 is unable to move. For the diagram above the closed loop equations below can be obtained, where equation (A.10) and (A.11) are the loop closure constraints in the polar coordinates.[53,54]

$$-l_1 cos(\theta) - l_2 cos(\alpha) + l_0 + l_3 cos(\phi) = 0 \tag{A.10}$$

$$-l_1 sin(\theta) - l_2 sin(\alpha) + l_3 sin(\phi) = 0 \tag{A.11}$$

rearrange these such that α and ϕ are expressed in terms of θ .

$$l_2 cos(\alpha) = l_0 - l_1 cos(\theta) + l_3 cos(\phi) \tag{A.12}$$

$$l_2 sin(\alpha) = -l_1 sin(\theta) + l_3 sin(\phi) \tag{A.13}$$

Taking the sum of the squares of equations (A.12) and (A.13) will now give:

$$k_1(\theta) sin(\phi) + k_2(\theta) cos(\phi) + k_3(\theta) = 0 \tag{A.14}$$

Where k_i (i= 1, 2, 3) are functions of θ as follows:

$$k_1(\theta) = -2 l_1 l_3 sin(\theta)$$
$$k_2(\theta) = 2 l_3 (l_0 - l_1 cos(\theta))$$
$$k_3(\theta) = l_0^2 + l_1^2 - l_2^2 + l_3^2 - 2 l_0 l_1 cos(\theta)$$

Equation (A.14) is the Freudenstein Equation, and can be solved easily as follows:

$$t = tan(\phi/2) \tag{A.15}$$

$$sin(\phi) = \frac{2t}{1+t^2} \tag{A.16}$$

$$cos(\phi) = \frac{1-t^2}{1+t^2} \tag{A.17}$$

substituting these into (A.14) gives:

$$(k_3 - k_2)t^2 + (2k_1)t + (k_3 + k_2) = 0 \tag{A.18}$$

thus giving:

$$t = \frac{-k_1 \pm \sqrt{k_1^2 + k_2^2 - k_3^2}}{k_3 - k_2} \tag{A.19}$$

By substituting (A.15) into (A.19) yields:

$$\phi(\theta) = 2tan^{-1}\left(\left[\frac{-k_1 \pm \sqrt{k_1^2+k_2^2-k_3^2}}{k_3-k_2}\right]\right) \tag{A.20}$$

Now divide (A.13) by (A.12) and obtain:

$$\alpha(\theta, \phi) = tan^{-1}(2[-l_1 sin(\theta) + l_3 sin(\phi), l_0 - l_1 cos(\theta) + l_3 cos(\phi)]) \tag{A.21}$$

Analysing the velocity. Differentiating the loop closure constraints in equations (A.10) and (A.11) with respect to time and expressing them in matrix form:

$$\mathbf{A} = \begin{pmatrix} l_1 sin(\theta) & l_2 sin(\alpha) & -l_3 sin(\phi) \\ -l_1 cos(\theta) & -l_2 cos(\alpha) & l_3 cos(\phi) \end{pmatrix} \tag{A.22}$$

$$\mathbf{A}\begin{pmatrix} \dot{\theta} \\ \dot{\alpha} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{A.23}$$

Choose θ to be the independent variable:

$$\begin{pmatrix} l_2 sin(\alpha) & -l_3 sin(\phi) \\ -l_2 cos(\alpha) & -l_3 cos(\phi) \end{pmatrix}\begin{pmatrix} \dot{\alpha} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} -l_1 sin(\theta) \\ l_1 cos(\theta) \end{pmatrix}\dot{\theta} \tag{A.24}$$

Thus obtaining :

$$\mathbf{A}\begin{pmatrix} \dot{\alpha} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} S_1(\theta\ \alpha\ \phi) \\ S_2(\theta\ \alpha\ \phi) \end{pmatrix}\dot{\theta} \tag{A.25}$$

The matrix containing S_1 and S_2 wil be called, $\mathbf{S}$, which is the null-space of matrix $\mathbf{A}$.

$$S_1(\theta\ \alpha\ \phi) = \frac{\partial \alpha}{\partial \theta} = \frac{l_1 sin(\phi - \theta)}{l_2 sin(\alpha - \phi)} \tag{A.26}$$

$$S_2(\theta\ \alpha\ \phi) = \frac{\partial \alpha}{\partial \theta} = \frac{l_1 sin(\phi - \theta)}{l_2 sin(\alpha - \phi)} \tag{A.27}$$

Now equation (A.25) can be used to determine the velocities $\dot{\alpha}$ and $\dot{\phi}$ in terms of $\dot{\theta}$.
Given the velocities have been found, the Lagrangian can be formulated.

$$T = \frac{1}{2}[m_1 l_{c1}^2 \dot{\theta}^2 + I_1 \dot{\theta}^2] + \frac{1}{2}[m_2 l_1^2 \dot{\theta}^2 + l_{c2}^2 \dot{\theta}^2 + 2 l_1 l_{c2} cos(\theta - \alpha)\dot{\theta}\dot{\alpha} + I_2 \dot{\alpha}] + \frac{1}{2}[m_3 l_{c3}^2 \dot{\phi}^2 + I_3 \dot{\phi}^2] \tag{A.28}$$

$$V = m_1 g l_{c1} sin(\theta) + m_2 g(l_1 sin(\theta) + l_{c2} sin(\alpha)) + m_3 g l_{c3} sin(\phi) \tag{A.29}$$

Subtracting equation (A.29) from (A.28) and thus ending up with the Lagrangian for the system:

$$\begin{aligned}\mathcal{L} = (&\frac{1}{2}(m_1 l_{c1}^2 + I_1 + m_2 l_1^2) + \frac{1}{2}(m_2 l_{c2}^2 + I_2) S_1^2(\theta\,\alpha\,\phi) + \frac{1}{2}(m_3 l_{c3} r + I_3) S_2^2(\theta\,\alpha\,\phi) + m_2 l_1 l_{c2} cos(\theta - \alpha) \\ &S_1(\theta\,\alpha\,\phi))\dot{\theta}^2 + ((-m_1 g l_{c1} - m_2 g l_1) sin(\theta) - m_2 g l_{c2} sin(\alpha) - m_3 g l_{c3} sin(\phi))\end{aligned} \tag{A.30}$$

A.3 Angular Frequency Change, ω, for Multiple Fits

Table A.1: Fit parameters for the swing angle, illustrated in Figure 3.9. The number of fits that have been applied are shown, whilst the angular frequencies, ω, have been shown for each of the fits. Values in () represent the R^2 value which corresponds to the parameter of each fit.

Number of Fit Functions (Swing)	Angular Frequency, ω (rad)		
	ω_1	ω_2	ω_3
1	2.545 ± 0.002 (0.846)		
2	2.455 ± 0.113 (0.897)	2.502 ± 0.002 (0.912)	
3	2.455 ± 0.113 (0.959)	2.618 ± 0.003 (0.961)	2.428 ± 0.005 (0.942)

A.4 Q-Learning UML Diagram (Python)

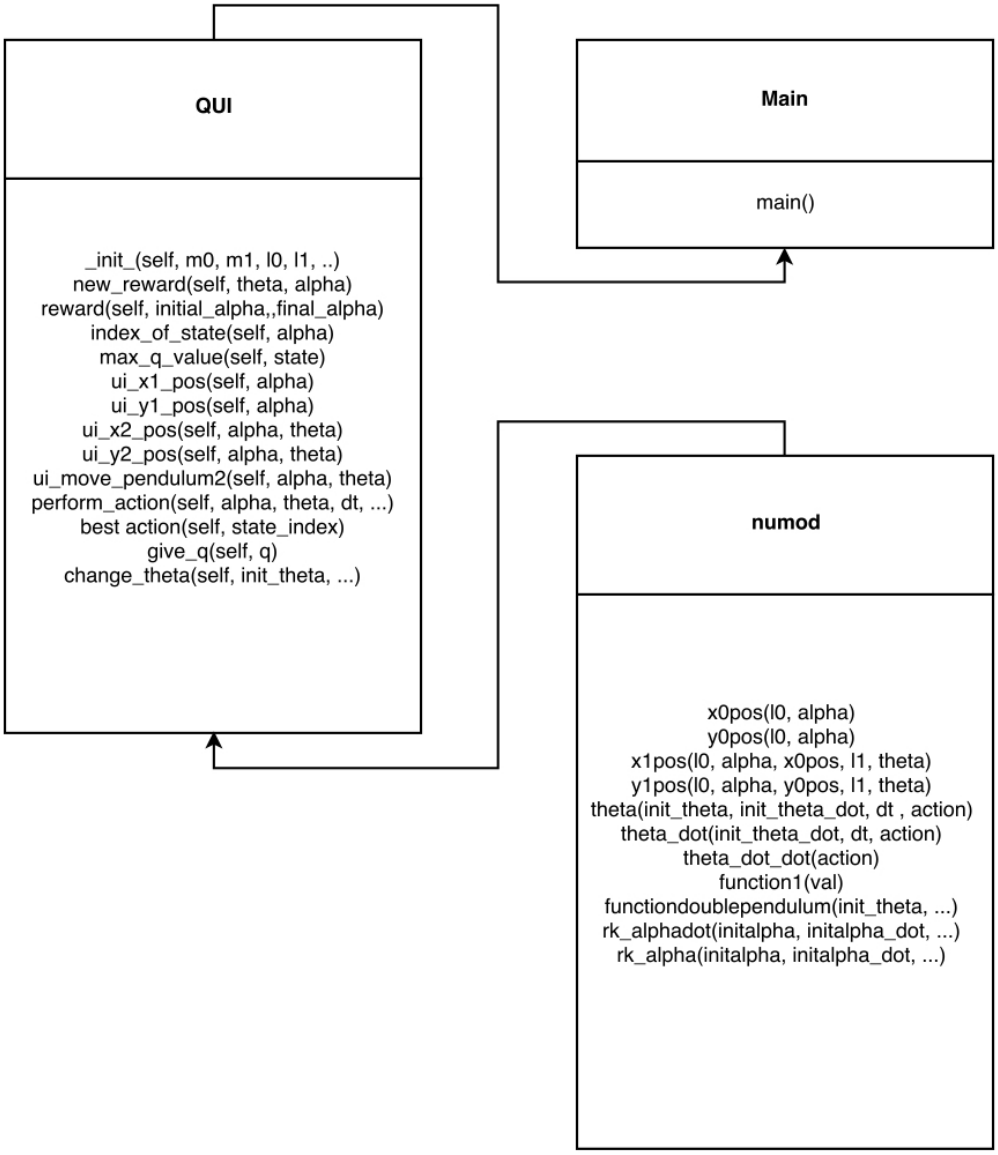


Figure A.2: Q-Learning UML Diagram (Python)