Robotics Group Studies

Teaching an Aldebaran NAO Robot to Swing

D. Butters, O. Diba, J. Forster, G. Hazar, Z. Hodgins,
C. Hogg, E. Humphreys, H. Jacobs, P. Jones, M. Lim,
S. Rowlinson, G. Sly, B. Stokes, J. Sweeney, M. Toon

School of Physics and Astronomy
University of Birmingham

Spring 2016

Abstract

The purpose of this report is to show the findings of investigations into the NAO robot and having it complete a swinging motion. Initial research was done into the NAO's movement, strength and internal sensor limitations. This resulted in a seated swing design, with pivot points in the supporting rods and a front cross bar for the robot to hold. Two methods of swinging were attempted: swinging from predetermined functions, and using sensor feedback to calculate when to move to maximise the motions amplitude. The angular acceleration of the swing was found using Lagrangian modelling, but upon simulation this was found to be an ineffective method of swinging. This was due to the idealised nature of the system. Human swinging motion was tracked and converted into corresponding movements on the robot. This showed some very promising results with simulations reaching an amplitude of approximately 11° independent of starting position, but when applied to the robot the amplitude only reached a maximum of 3.2±0.1°. For swinging from sensor feedback, all methods required a small initial push it was unable to start from rest. Initially the robot used an angle encoder at the swing pivot to move at the correct time in the cycle, which produced an amplitude of 7.2±0.1°. Vision tracking software was used determine the robot's direction of travel and from this the robot calculated when to move. This resulted in the highest swinging amplitude of the project, 8.6±0.2°. Finally, the internal gyrometer was used and it reached a steady state of amplitude 6.9±0.1°. A final approach was to have the robot learn itself when best to swing without predetermined functions or movements, via a machine learning library. The library could not be tested on the robot as it would have taken around eight hours to learn. Therefore, it was simulated using an inverted pendulum which consisted of a motor learning how much torque to apply to get the pendulum inverted. It could be seen that the system was learning, but the final goal was never reached.

Contents

Nomenclature

The following was contributed by: P. Jones

Aldeberan NAO: NAO is the name of the robot that was used throughout this project, and is produced by Aldeberan robotics. It runs a custom operating system, NAOqi OS, which contains multiple libraries for interfacing with the NAO.

Choregraphe: Choregraphe is both a node-based program editor for the NAO robot, and a GUI for many of the functions provided by the NAO API.

Nodes: A node in the context of Choregraphe is effectively a graphical representation of a function, with a number of inputs and outputs. Multiple nodes can be connected together to create a program within Choregraphe.

API: The API, or Application Programming Interface, is the set of libraries and functions provided on the NAO that allow control over various parts of the robot, for example movement of the limbs and vision tracking.

Angle encoder: The encoder is a small device at the swing's pivot that reads the angle of the swing. The angle can then be sent to a computer or the NAO robot via a USB cable.

Landmark Detection: The NAO documentation provides a number of images called landmarks, which the NAO robot is capable of identifying. Upon seeing one of these landmarks, the NAO stores information about its position and size.

Webots: Webots is a program that can simulate the NAO robot, as well as others. It allows the creation of custom environments and behaviours for the robot, which were used to simulate the robot on a variety of swings.

Kinovea: Kinovea is a piece of video analysis software, that allows tracking of positions and angles of objects in a video.

Stepper motor: A stepper motor is a motor that can be accurately moved to set angles, rather than simply rotated one way or the other for a period of time.

Phidget: A Phidget is a small controller circuit board, which was used to control the stepper motor in this project.

Runge-Kutta: Runge-Kutta methods are commonly used methods of numerical integration, that are more efficient than simpler methods at minimising errors.

GNU Scientific Library, GSL: GSL is an open-source library containing a multitude of useful numerical methods. In this project, the Ordinary Differential Equation (ODE) solvers from the library were used.

gnuplot: gnuplot is a third party graphing software that can also provide data fits with errors.

Smoothing: In the context of this project, smoothing data means in some way accounting for multiple previous results in the calculation of the next step. For example, smoothing position may mean taking the running average of three position readings, to reduce the effect of noisy results.

Torso frame: The robot can make measurements in three reference frames: the torso frame, the world frame, or the robot frame. The torso frame is the one used throughout this project, and is a frame of reference fixed in the robot's torso, with the z-axis pointing towards the robot's head. This means that if the robot for example bends forwards, the torso frame rotates with it.

Module: A module is type of programming class that can be loaded by the NAO on startup, or loaded manually. It allows functions in the class to be called by other programs, without having to include the function code in them.

Priority queue: The `PriorityQueue` is a data structure used to store data with associated "keys" in an efficiently ordered configuration, allowing retrieval of maximum/minimum priority elements in constant time. This structure was primarily used for quick retrieval of actions in the machine learning algorithms of the project.

Statespace: The state space is an abstract vector space whose dimensions are the unique parameters of the system and whose domain is the set of all possible states that an agent can be in.

Q-value: The Q-value, or utility, is the metric used by the machine learning program to assign priorities to its actions. It is effectively the modified reward for a state action pair.

Chapter 1

Introduction

1.1 Drive

———————The following was contributed by: S. Rowlinson ———————

The underlying idea behind this project was to successfully get the NAO robot swinging with a motion similar to humans via a variety of different techniques. Whilst this is the ultimate goal of the project, the question of why we are doing this and what the fundamental *drive* behind the project was must be answered. There are several factors outlining the answers to this question, each of which involves a different aspect of this collaboration as a whole.

Humans have a habit of personifying inanimate objects, whether it is naming their car, or putting eyes on a robot. For this project, we wanted to take this a step further and see if the robot can be made to move like a human does. When swinging, your brain makes many calculations in a split second; how fast you're moving, when to lean back, how much force to pull back with. Is it possible to replicate this in a machine?

One of these major influencing factors behind making this a reality, was the idea of machine learning and delving into the principles behind how a robot may learn about a system (in this case, the robot-swing system), the issues that arise during the learning process as well as the possible extensions and applications to other areas for motion-based machine learning. Our work within this field comes at a time of rapid advances in machine learning and artificial intelligence techniques given Google DeepMind's recent breakthrough in developing an intelligence algorithm to defeat the world champion in the ancient board-game of Go [1]. The inherently complicated dynamics behind developing an algorithm to deal with such large state spaces (as is the case in Go) was one of the main points of inspiration as far as the machine learning related work in this project was concerned, especially given that one could potentially discretise the robot-swing system to a larger number of states than any board games. For further details on the progress of machine learning in this project, refer to Chapter 6 of this report.

Moving away from machine learning, the other key influences behind the motivation of this research were improving our understanding of applying Lagrange's equations to complicated physical systems' overcoming the challenges posed by attempting to accurately model human swinging motion on a robot, and finally learning to utilise sensor feedback.

1.2 Literature Review

One of the main sources of information for driving the outcomes of this project was the 2015 report on the progress made on teaching the Aldebaran NAO robot how to use a swing [2]. In this report, the method of swinging for the robot that was chosen was a standing position using pumping of the arms to drive the motion of the swing given a periodic driving force in the movement code. Upon reviewing this method, it was decided for our project to focus on a more typical swinging motion for humans by instead having the robot seated on the swing and using a combination of driving forces from the torso and legs to perform the swinging cycle; further details on human swinging motion can be found in Section 3.2 of this report.

Additionally, the previous report [2] outlined the topic of machine learning and how this rapidly advancing field could potentially be used to allow the robot to use the swing using minimal initial information about the system - this was also chosen to be another focus of this project in order to properly address the objective of the problem at hand which was to, implicitly, *teach* the robot to use the swing. Thus achieving a machine learning algorithm which could handle this system as generally as possible was targeted as one of the major goals of this project. Details on this can be found in Chapter 6 of this report.

In our reviews of similar work in the field of robot based machine learning, it was found that the topic of learning from demonstrations is a common technique used for teaching robots to emulate human motion [3, 4]. This area of research involves giving human inputs to a robot and analysing these inputs such that the robot can mimic them and perform such tasks by itself - an area of machine learning known as *Supervised Learning*. Whilst this technique was initially considered for this project, it was decided that the preferred route was *Reinforcement Learning* [5] due to the more natural approach of allowing the robot to learn by itself over many trials rather than employing a difficult, and more forced, approach of showing the robot the correct motion. For our approach to the machine learning aspect of this project, the book "Reinforcement Learning: An Introduction" [6] was one of the more invaluable sources of information in providing insights into how reinforcement learning could be applied to a system such as the robot-swing system. Further details on the choices made with regard to the machine learning techniques used in this project can be found in Section 6.1.2 of this report.

The main reference for our review into sensor feedback was the NAO documentation [7, 8] which detailed the exact specifications of the sensors on our robot model and the capabilities of these sensors for different movement types; details on our progress on this section are in Chapter 5. For our research into human motion replication we reviewed the previous report [2] and looked into the process of recording our own human swinging motion in order to achieve more accurate modelling of motion on the robot - see Section 3.2 for details on our progress in this field.

1.3 Background Theory

The following was contributed by: B. Stokes

The problem of driving a swing is fundamentally one of how to increase the energy of the system as efficiently as possible and thus create motion. There are several physical mechanisms that allow a person to increase the energy of the swing. Trivially, an external agent can provide the swing with energy by pushing it periodically but this is the least physically interesting method and does not involve input on the part of the person on the swing.

Another method involves standing on the swing and performing rhythmic crouching and standing. This works by manipulating the application of angular momentum. Whenever the angle of the swing is displaced from the central position, gravity exerts a torque on the swing that accelerates the motion in the direction of the centre point. The torque applied is given by $\tau = mr \times g$ where r is the distance to the system's centre of mass, g is the acceleration due to gravity and m is the relevant mass. The greater the displacement of the swing, the greater the torque applied by gravity. If a person on the swing crouches from standing they increase the radial distance of the centre of mass which directly increases the torque exerted on the swing and drives the motion. The best place to crouch is at the maximum displacement of the motion as this is where the torque due to gravity is greatest. Once crouched the person must stand again and the best place to do this is when the swing is directed straight downwards. Here gravity exerts no torque on the swing due to the cross product so standing up will not decelerate the swing at all (assuming the person were to stand instantaneously). The conservation of angular momentum will also cause an increase in the magnitude of the swing's velocity when standing at this point. Note that this method of swinging cannot start the swing from stationary as crouching and standing can only increase a torque that is already present.

The other way to drive a swing is to assume a seated position and alternate between the two actions of swinging the legs back and torso forwards and swinging the legs forwards and torso backwards. There are several components to this motion. Firstly, when the swinging person performs the motion they move their centre of mass further from the pivot. This can add energy to the system in the same way as for standing motion. Secondly, the rotation of the two masses of the upper body and lower legs about

the swing seat exerts a separate torque on the central pivot of the swing each time they are accelerated. Lastly, a minor contribution to the energy is also made by the gain in gravitational potential energy achieved when the centre of mass is shifted upwards somewhat at the height of each swing. Since the rotation of the limbs around the swing seat can exert torque on the main pivot from any initial angular position, this method of swinging can start the swinging motion from a stationary position by rapidly performing the actions until some small amplitude is acquired and then settling into a periodic pattern of movement. Once in this periodic regime the best way to perform the motion is to change from one position to the other at the points of greatest displacement of the swing. This is for the same reason as in the standing motion that at these points the increasing of the centre of mass to pivot distance has the greatest effect.

In last year's report [2] the team focused exclusively on the first swinging method described. They were able to get the robot to swing successfully from an elevated starting position by instructing the robot to stand at maximum displacement and crouch at zero displacement. Whilst it did get the robot swinging successfully, this method is neither very theoretically sound nor representative of the way a human uses a swing. It was decided that this project would focus on seated swinging for the reasons given in Section 1.2.

1.4 Report Outline

———————————The following was contributed by: Z. Hodgins ———————————

The aim of making the NAO robot swing was a very broad one, meaning that the research could be taken in many directions. As mentioned, after a review of literature and background theory, it was decided that the group's focus would be on optimising the swinging motion when the robot is sitting down. Four main research sections were investigated, which will each be presented in this report.

Firstly, a swing had to be designed. The robots range of movements, strength and sensing systems were all investigated in order to design a swing seat that was as nonrestrictive as possible. Secondly, the group looked into making the robot swing from predetermined functions. This involved using theoretical swing motions found by modelling the swinging system using Lagrangian methods. Human swinging motion was replicated by tracking the motion and fitting the functions to obtain equations of motion.

Moving forwards, instead of using predetermined functions, the robot was made to read various sensors to calculate when to swing. By individually looking into an angle encoder at the top of the swing, and the gyroscopes, accelerometers, gyrometer and visions trackers built into the robot, the NAO was made to swing by determining when the maximum amplitude of motion was reached and then making the relevant movement. Finally, to improve the swinging even further, a machine learning library was written specifically for the robot so it could learn how and when to move to maximise its amplitude. This report will go through the research areas and successes of each of the methods, summarising which were found to be the most effective.

Chapter 2

Robot Properties and Swing Design

2.1 Introduction

———————————The following was contributed by: M. Toon ———————————

For this project the NAOv4 robot created by Aldebaran was used to swing on a specially designed swing. It uses the NAOqi v1.14 Framework to allow the robot to be programmed and perform specific actions. The NAO robot has humanoid movements but is much more restricted due to the limitations of manufacture and technology. A human uses their many flexible joints and muscles to finely control themselves on a swing, whereas the NAO robot is unable to achieve this level of motion. Instead of the robot adapting to suit the shape of the swing, the swing must then be adapted to work with the range of motions and strengths of the robot. Other factors including the safety of the robot must be addressed as there is a high risk of damaging the expensive equipment. A swing was provided from a previous year's project but was unsuited for allowing the robot to swing in the seated position [2]. The aim is to modify this swing to suit the needs of our seated robot.

Strength tests were carried out to assess the robot's ability to move itself on the swing. If the robot could not move itself in a certain way such as lifting its own weight, then the swing design would have to be compromised. The robot was moved repeatedly and the temperature of the joints were measured to ensure that there were no overheating issues. If a joint stays too hot for too long then it will lose the functionality of said joint which would hinder progress in the project.

Each of the various sensors of the robot including its vision, gyrometer and accelerometer needed to be tested to assess how useful they would be when providing feedback for the motion of the robot. An encoder provided from previous year's project is an external sensor which measures the angle of the swing and was also tested as a more reliable source of feedback and may be used to compare the robot's internal sensors.

———————————The following was contributed by: J. Sweeney ———————————

To determine the parameters of the swing, the limitations of robot's range of motion needed to be determined. In order to achieve this, the program Choregraphe was used. Choregraphe is a bespoke graphical user interface, produced by Aldebaran, for use with the NAO. Through the use of prepackaged modules, the NAO can be programmed to move either to predetermined positions or to specified joint angles. This was of particular advantage as, unique to this program, multiple joint angle values could be viewed in real time via the Memory watcher panel[9].

To enable concurrent work during the assembly of the swing, the design had to be implemented in the Cyberbotics Ltd. robot simulation software: Webots. Webots is a commercial program designed primarily for use in the research and development of robots. Through the use of the Open Dynamics Engine (ODE), Webots is able to simulate a variety of objects and environments reasonably faithfully via rigid body dynamics[10],[11]. Of particular advantage is the presence of the Aldebaran Robotics NAO robot within the program. This provided an easily implementable, and physically accurate model of the robot with which to perform tests and as such was the main factor in selecting this software.

2.2 Robot Dimensions

2.2.1 Angular Movement

The following was contributed by: E. Humphreys

Upon receiving the robot the range of motion of the limbs was measured. We needed to ensure that the robot was not damaged in any way from previous operations that could have affected the range of motion of the limbs. These measurements would lead to a swing design that didn't restrict the robot's movement and theoretical models of the system being made.

The robots range of motion is taken from the NAO documentation [8]. The measured range of motion is provided in Table 2.1. The experimental values were measured using Choregraphe's memory watcher, watching the appropriate values for the positions of the motors. Maximum and minimum values were obtained by using Choregraphe (the program provided by Alderbaran that allows for the robot to be programmed and controlled) to move the joints to extremes of motion.

Table 2.1: Range of Angular Movement from measured values.

Joint Name	Min ± 0.1 (°)	Max angle ± 0.1 (°)	% Deviation under theoretical
Head(yaw)	-38.5	29.0	1%
Shoulder(roll)	-18	71.5	5%
Shoulder(pitch)	117.9	-118.0	1%
Elbow(yaw)	117.9	-117.5	2%
Elbow(roll)	2.7	86.7	3%
Wrist(yaw)	-104.4	104.4	0%
Hip(roll)	-45.3	12.7	13%
Hip(pitch)	-88	9.2	16%
Knee(pitch)	-5.3	121.1	0%
Ankle(roll)	25.0	17.6	0%
Ankle(pitch)	52.9	-67.9	0%

As can be seen in Table 2.1, some of the joints are limited in their range of motion by a few degrees less than the documentation says the robot should be able to move. Despite this, the difference is small enough that it should not interfere with the swinging motion of the robot. The errors in Table 2.1 are the resolution to which the Choregraphe memory watcher observes the joint angle.

The only joints which show a significant difference between the actual range of motion and the theoretical are the Hip joints, which are 13% and 16% (for the roll and pitch respectively) less than the theoretical. While this is a large deviation on the robot that would affect the robot's ability to lean backwards, it is possible the design of the seat would limit it more.

A preliminary examination of how the preexists seat restricts motion was performed. Since it had been discussed that the seat would be altered but not drastically changed it was decided that testing the range of motion on the preexisting seat would produce useful results. This consisted of placing the robot on the swings seat and moving both the hip joints and knee joints until the motion of these joints stalled. The robot was firmly placed on the seat using zip-ties and then programmed to move using Choregraphe. Table 2.2 confirms that the hip is limited by the seat more than the angular limit measured.

Table 2.2: Range of Knee and Hip on seat.

Joint Name	Max angle (°)	Min angle (°)
Knee(pitch)	-5.3 ± 0.1	91.1 ± 0.1
Hip(pitch)	-88 ± 0.1	6.9 ± 0.1

2.2.2 Limb Lengths

The following was contributed by: M. Toon

As the robot was to be fixed to the swing at both its hands and hips, the length of each limb determined the range of motion that the robot was capable of. For the NAO robot to freely move itself in a seated position without obstruction, the range of motion for the each limb needed to be assessed. The robot was initially examined in a loose state to see where each limb could be flexed to its extreme positions. Due to the rounded shape of the limbs and limp nature of the robot, physically measuring the distances on the robot proved difficult to obtain accurate measurements.

The upper leg was fixed rigidly to the seat such that the main body torso could lean back and forth on its hip joints. Depending on where the hands and legs were fixed determines the range in which the robot can lean backward or come forward. Ideally this would be as large as possible.

The robot was sat with its legs outstretched in front of itself. A tape measure was used to measure the joint to joint length of the upper arm, forearm and back height. Other important geometries such as fully outstretched arm length and minimum arm length where the elbow is fully bent were measured. The arm can bend in two main orientations with the first being with elbows sticking out and the second with the elbows coming down and the arm rotating about the shoulder axis. The important length from the arms is the distance between the shoulder and hand and are shown in Table 2.3. The minimum arm length was smaller with the elbows in the outward position which gave a larger range of motion to the robot.

Table 2.3: The robots documented and measured limb lengths.

Robot Limb	Measured Length (±15mm)	Documented Length (mm)
Back (Hip - Shoulder)	200	185
Upper Arm	125	105
Forearm (Elbow - Hand)	125	113.7
Fully Stretched Arm	250	218.7
Bent Arm (Elbows Out)	150	152.7
Bent Arm (Elbows Down)	155	-
Shoulder Width	-	275

These values were used to create a simplified two dimensional model of the robot's back and arms to better visualise the range of motion available from the robot. The model's limbs were moved to the minimum and maximum positions and fixed in place at each. Ideally the centre of mass of the robot would move over the largest range of vertical height possible. It was decided to restrain the motion of the robot should be upright on the back swing and lean as far backward as was allowed on the forward swing more similar to how a human would swing.

The exact specifications for the robots were found online from the manufacturers website [7]. The discrepancy between the measured and documented lengths is due to the measured values including the thickness of the shell of the robot where as the documented values are strictly joint to joint lengths with zero tolerance. It was noted that one must account for the thickness of the limbs when considering clearance for the range of motion on the swing.

The shoulder to hand length was calculated from the documented values using the cosine rule with the minimum angle for the elbow, 88.5°, also taken from the manufacturers website [8]. The distance that the elbow protrudes sideways due to the bending of the arm was calculated to be 76.2mm. This is then combined with the shoulder width to give a minimum swing width of 427.4±0.5mm.

2.3 Strength and Temperature Tests

The following was contributed by: G. Hazar

2.3.1 Strength Test

The strength tests were part of a group of investigations which aimed to discover any possible limitations of the robot which would hinder our ability to meet the aims of the project. The results would provide insight into how the robot's motors have deteriorated overtime as the robot is over two years old. The results were compared to the values given by the Aldebaran data sheet [12] to provide an indication into any discrepancies of the data sheet for the robot in use. Finally the tests aimed to give the analytical group information on the capabilities of the robot to perform certain motions (i.e lift its self) and inform the swing design team about any considerations which need to be taken into account when designing the seat.

Strength tests were also performed by the previous year's report [2], which provided a base for us to improve upon. In their tests the Newton metre was attached to the fingers of the robot. The fingers would flex, leading to a reduced extension of the Newton metre resulting in readings below the actual value. We improved upon their method by using a zip tie to attach the Newton metre to the wrist instead of the fingers and securing the robot to table during the experiment.

The strength tests were performed on the critical motors used by the robot to execute a swinging motion. These were determined to be the shoulder, elbow and knee motor. Each motor was tested separately to observe how much force it could exert before stalling.

2.3.1.1 Theoretical Values

The theoretical value for the force exerted is given by Equation 2.1 [13],

$$F = \frac{\tau r}{l} \tag{2.1}$$

where F is the force exerted on the Newton metre, τ is the motor torque, r is the motor speed reduction ratio and l is the distance between the motor and where the force is applied. The error calculation can be found in Appendix A1.

The values used to calculate the theoretical force values are shown in table 2.4.

Table 2.4: Table of the data for the theoretical force calculations [12].

Limb	Length ($\pm 0.01mm$)	Torque (Nm)	Speed Reduction Ratio
Lower arm	55.95	14.3±1.1	173.22
Upper arm	105.00	14.3±1.1	173.22
Lower leg	100.00	68.0±5.4	130.85

2.3.1.2 Method

The robot was strapped into the swing seat (using zip ties) which were clamped on to a table as shown in Figure 2.1. A zip tie was attached to one of the robot's wrists and a Newton metre was attached to the wrist using a zip tie. The other end of the Newton meter was then attached to a bar, which was securely fastened on to two clamp stands. The height of the Newton metre was altered to ensure it was in line with the arm at the start of the experiment. The lower arm (elbow motor) was then made to move from its initial position to minimum extension using Choregraphe. If the robot could complete the motion then the experiment was repeated with the addition of 5±2N of preload (obtained by stretching the Newton meter). The process was repeated until the robot could no longer reach minimum extension. The same method was used to determine the strength of the lower legs (knee motor). The Newton metre was attached to ankles of the robot during the leg tests.

(a) " Slack" position. (b) " Tau" position.

Figure 2.1: Positions used in the strength test.

Table 2.5: Results for the Strength Test.

Limb	Theoretical Force (Nm)	Experimental Results (±2Nm)	
		Right	Left
Lower Arm	44 ± 4	28	24
Upper Arm	20 ± 2	18	16
Lower Leg	89 ± 8	26	30

2.3.1.3 Results

As shown in Table 2.5 the experimental and theoretical values are within the same order of magnitude, but the difference between the two values is significantly greater than the error for most of the limbs. There are a number of factors which may have contributed to the disparity between the two values. The experiment required repeated attempts of the same motion in quick succession of each other, which may have led to the motors becoming too hot. When a motor of the robot surpasses a certain temperature the robot will limit the output torque of the motor as a safety precaution [14]. This explanation for the disparity between the two values would also explain why all the experimental measurements are below the theoretical value.

Another contributing factor is the fact that the robot would move other limbs that were not being tested in order to try to complete the motion. This made it difficult to read the value on the newton meter, as there is only a short period between when it exerted the maximum force and when the robot starts to use other limbs.

Finally the lower arms and the lower legs preformed an arc motion during the tests as shown in Table 2.5. This resulted in a smaller reading being measured then the actual force as the direction of the force was not parallel to the newton metre. This explains why the upper arm values are within two standard deviations of the theoretical value, whereas the other two limbs are not.

2.3.2 Grip and Pull Up Test

2.3.2.1 Method

This experiment aimed to test the strength of the robots grip and also provide a lower limit for the robots arm strength, as the strength test results proved to be inconclusive. The robot was strapped on to a table as described in the strength test. The rod was clamped on to two clamp stands fastened on to stools as shown in Figure 2.2. Choregraphe was used to make the robot's hand grip onto the rod. The robot was extended as far back as possible after which it attempted to pull its self back up. The robot was successful in executing this motion and the motors in the hands did not stall.

After the grip test the robot performed a pull up to provide a lower limit on the robots strength. Choregraphe was used to set an initial and final position for the pull up. A rod attached to two clamp stands was placed above the robot. The robot was made to grip the rod and attempt to move from position one to position two in Figure 2.3, emulating a human doing a pull up. The robot was successful in doing so which places a lower limit of $51.99 \pm 0.001N$ on the combined strength of the arms (obtained from the force required to lift the mass of the robot).

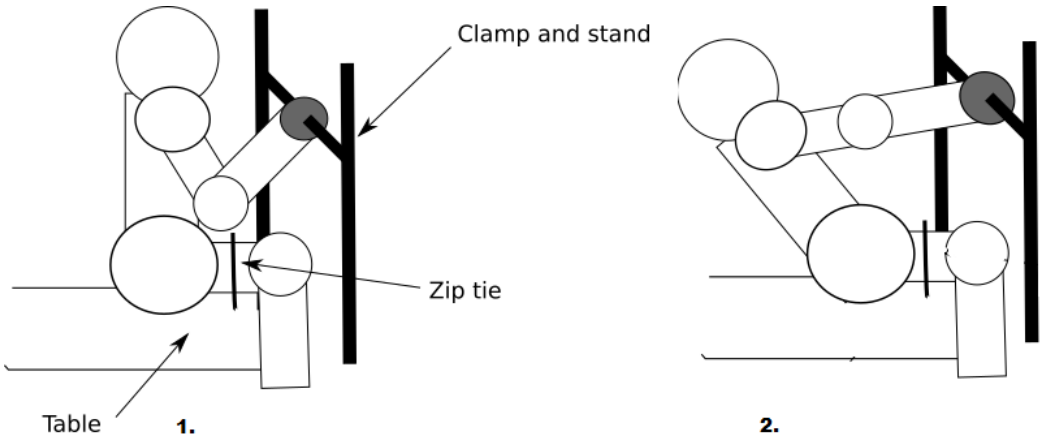


Figure 2.2: Illustration of the robot set up for the grip test.

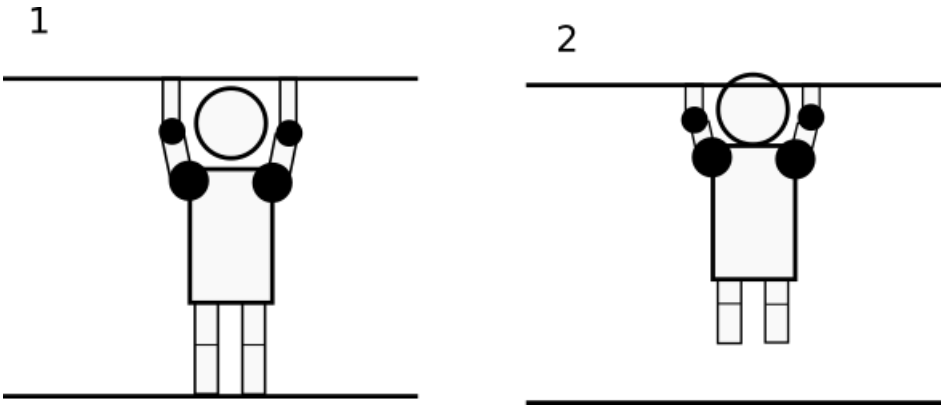


Figure 2.3: Illustration of the robot set up for the pull up test.

2.3.2.2 Results and Conclusion

The results of the strength test were significantly lower than theoretical values for reasons mentioned in the previous sections. The aims of the experiment were still achieved as the grip and pull up test showed that the strength of the robot would not be a limiting factor for a swinging motion. The robot was able to perform a pull up and lift itself from an extended seated position shown in Figure 2.2 which showed that it was strong enough to perform the necessary motions for swinging and the arms have a lower strength limit of 51.99 ± 0.001N.

If the test were to be repeated it would be recommended a Newton metre is not used to measure the strength because of the difficulties explained earlier and the inherently large errors that Newton metres have. Instead it would be recommended that tests such as the pull up test be carried out to find lower and upper limits for the strength as these provide a more accurate indication of the robots strength.

2.3.3 Overheating Test

Tests were performed to ensure the motors would not overheat when the robot is swinging. This experiment was deemed necessary because in the previous year's report [2] it was mentioned that the hands overheated when preforming strength tests [13]. The experiment was carried out by making the robot simulate the motion it would perform on a swing. Measurements of the motors' temperature were taken periodically while the robot repeatedly performed a swing motion.

2.3.3.1 Method

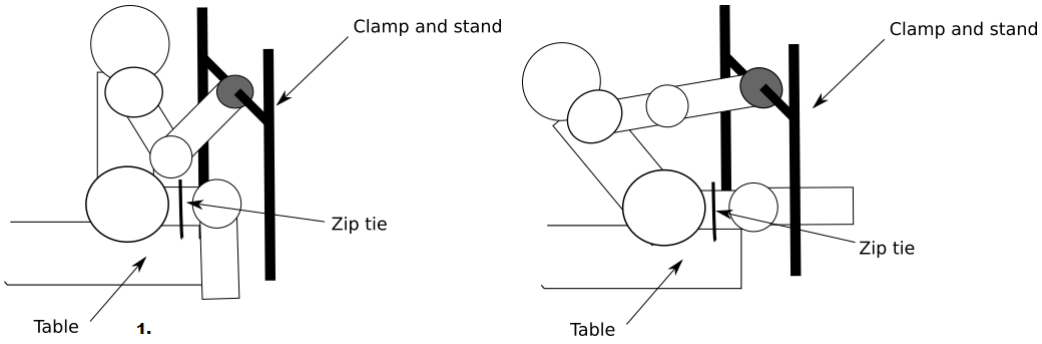


Figure 2.4: An Illustration of the two positions used in the heat tests.

The robot was secured firmly into the seat which was clamped onto a table. The robots hand were tied onto a bar which was fixed between two clamp stands, as shown in Figure 2.4. The desired motions were programmed into Choregraphe. This was done by setting two positions for the robot, one upright and the other extended backwards. A loop was then created to command the robot to move between the two positions. The rod was used to replicate the one included in the swing design. The robot was programmed to move between between the upright and extended position for ten minutes and readings were taken every fifteen seconds from the robot sensor memory using Choregraphe.

2.3.3.2 Results

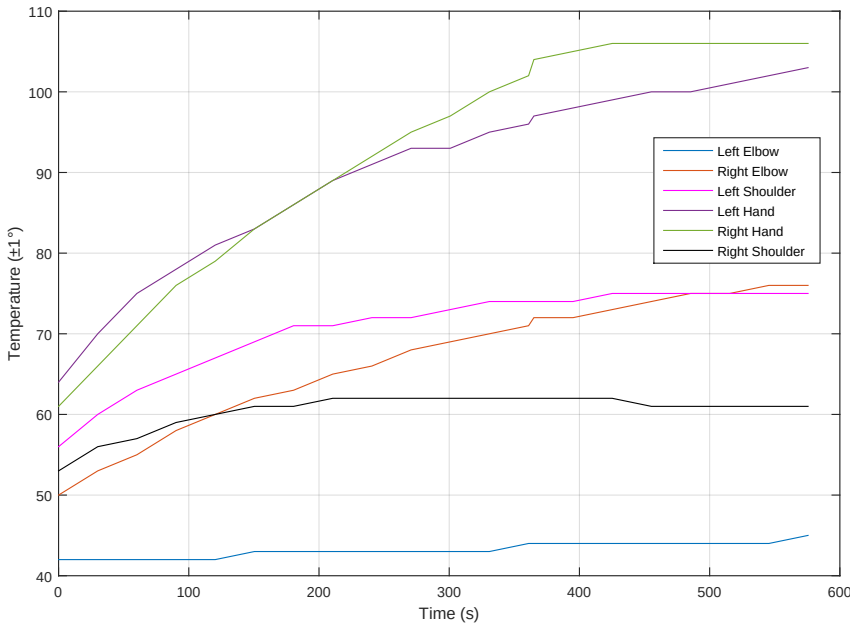


Figure 2.5: Temperature of the arm motors over time as the robot performs a swing motion.

From Figure 2.5 it can be seen that the temperature increase in the different motors varied greatly. The temperature of the right shoulder motor only increased by 8 ± 1^oC whereas the right hand increased by 45 ± 1^oC. There are a number of reasons why this may have occurred. One of the possible reasons

may be due to fact Choregraphe had to be used to carry out the motion of the robot. Choregraphe requires that the robot be physically placed into the two positions. This introduced uncertainties in the positioning of the arms and, in turn, the amount of work done by each motor. Furthermore this method fails to ensure that both the right and left arm used the same motors to move between positions. Evidence supporting this can be seen in Figure 2.5 where it is clear that all the right hand motors experienced a larger increase in temperature than the left side equivalent. This implies that a majority of the work done to lift the robot from position 2 to position 1 was done by the right arm.

Evidence of the non-symmetric method used by the robot to lift itself can be seen by further comparing the right and left motors. For instance the right hand and elbow motor showed the largest increase in temperature on the right side, as its temperature increased by 45 ± 1^oC and 26 ± 1^oC respectively. Whereas on the left side the hand and shoulder motors had the greatest increase in temperature, as their temperature increased by 39 ± 1^oC and 19 ± 1^oC. This implies that the right arm mainly used the elbow motor to complete the motion whereas the left side used the shoulder motor. This difference in temperature could be explained by taking into consideration the fact that the robot is two years old. Over time some of the motors on one side may have been put under more strain then their counterparts on the opposite side, leading to asymmetric deterioration of the motors.

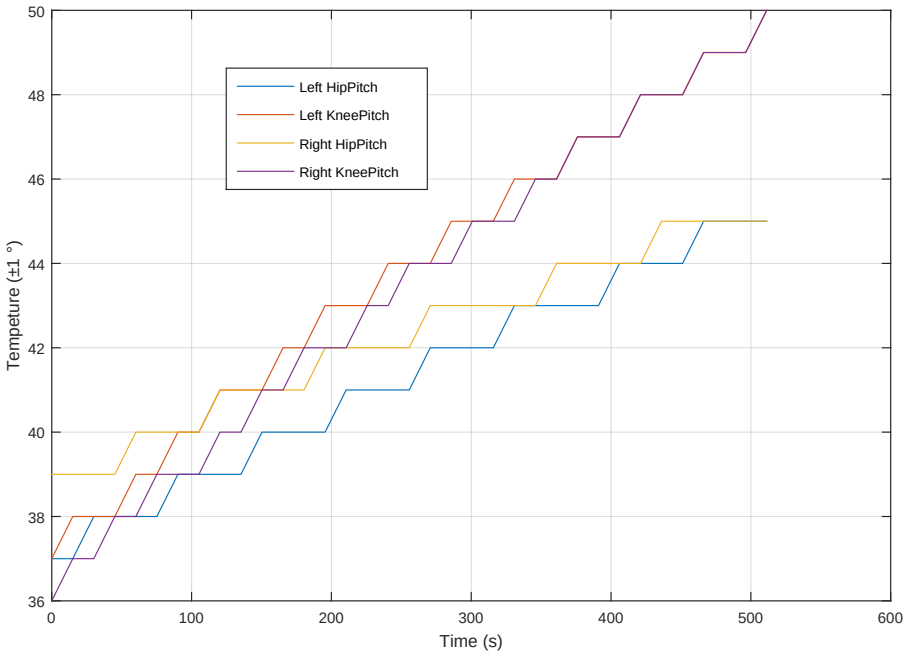


Figure 2.6: Temperature of the leg motors over time as the robot performs a swing motion.

As Figure 2.6 shows, there is a much smaller spread in the change in temperature in the legs compared to the arm motors. Furthermore it can be seen that leg motors on the right and left side reached the same temperature unlike the arm motors. It can also be noted that the legs did not reach as high temperatures as the arms. The maximum temperature of the legs was 50 ± 1^oC compared to 106 ± 1^oC for the arms. This is because arms did significantly more work than the legs, as the arms pulled up the upper body of the robot during the motion.

2.3.3.3 Conclusion

In conclusion, the arms experienced a greater increase in temperature then the legs, because the arms were pulling up the torso as well as their own weight. The robot is programed to provide warnings when its motors are in danger of overheating. It did not do so during the experiment therefore it can be concluded that none of the motor reached dangerously high temperatures. Hence no special consideration is needed in the swing design to take into account heating issues. However, it was agreed that hands of the robot would not be used to grip the handle of swing and instead Velcro would be used to fasten the hands

in place. This decision was based on the fact that the hand reaching significantly higher temperatures then the rest of the motors and they had experienced overheating issues in the past. Using Velcro would act as a safety precaution against further damage.

2.4 Sensors

2.4.1 Introduction

The following was contributed by: H. Jacobs

A key aim of this project was for the robot to be able to swing without any external input. To accomplish this, the robot had to be able to use its own internal sensors to be able to know its position in the swing and then execute movements accordingly.

Thorough investigation into its internal sensors was therefore necessary with two main focuses: Vision and Inertial Sensors. The former would use the landmark detection software built into the robot using its internal camera whilst the latter would focus on using the motion of the robot to calculate the position during swinging. Multiple approaches were taken because, when a human swings, they have more than one sensory input that would be analogous to the robot's sensors. The cameras will replicate the eyes of a person and the inertial sensors will simulate a human's ability to be able to tell their position and acceleration. A combination of both of these methods would most closely replicate that of a human on a swing and so both sensors were investigated.

2.4.2 Inertial Sensors

The following was contributed by: H. Jacobs

The robot has two main inertial sensors types which it uses to calculate its location, velocity and acceleration. These were investigated to see which could be used allow the robot to know its position in the swing. One of the sensor types is the accelerometer which measures the acceleration in the robot's three axes (see Figure 2.7), with the Z accelerometer measuring angular acceleration about the axis. The other sensors are gyrometers which measure the angular rotation around the X and Y axis.

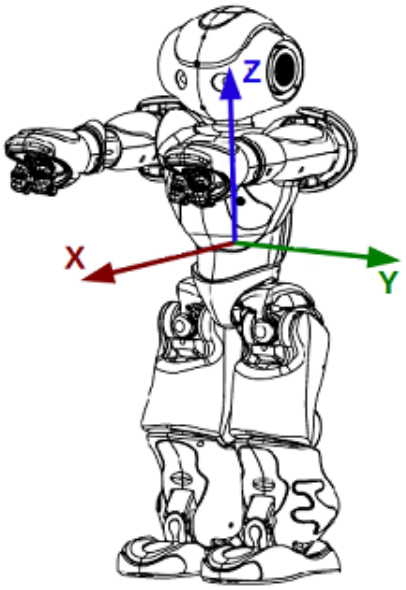


Figure 2.7: NAO internal Axises [7].

The accelerometer has a precision of 1% at an acceleration of $\approx 2g$ and the gyrometer has a precision

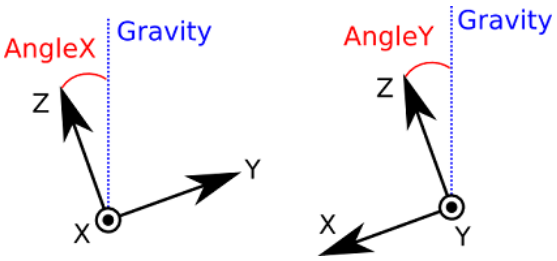


Figure 2.8: NAO Angle Function [15].

of 5% at $\approx 500°s^{-1}$ [15]. Table 2.6 shows the positions of the sensors relative to the robot's torso frame (a frame of reference with the origin at a fixed point in the centre of the torso) as defined by the specifications. The sensors are slightly offset from the centre of the robot, however, any effects from this will be negligible.

Table 2.6: Inertial sensor locations [15].

Device	X (m)	Y (m)	Z (m)
Accelerometer	-0.008	0.00606	0.027
Gyrometer	-0.008	0.006	0.029

2.4.3 Inertial Sensor Tests

A series of swing tests were performed to get base readings that were used to decide which of the sensors could be used to determine the location of the robot in the arc of the pendulum. These tests were qualitative and were done to examine the usability of the sensors for calculating when the robot should perform a movement on the swing.

The robot was attached to the swing in an upright, seated position and the swing was released from an angle of $\approx$ 20-25°. The swing was left to make a few oscillations whilst the sensor values were transmitted over WiFi to a computer (see Appendix D for more details). The NAOqi software development kit contains built in functions for transmitting this data over the local network by creating a *proxy* to the required module on the robot. In this case it was to `ALMemory` which has a function to return specific data stored in the robot's memory. This computer was attached to the angle encoder and would measure the current angle of the swing. This had to be calibrated by recording the value of the swing in the equilibrium position and subtracting this value from all the other data points.

2.4.3.1 Angle Function

The NAO software has an inbuilt function to get the angle of the robot with respect to gravity. The function can measure two angles as seen in Figure 2.8, which shows the X and Y angles relative to gravity and there are two ways it calculates the angle. If the robot is stationary it will only use the accelerometers and, if not, it will use a combination of the gyrometers and accelerometers [15].

The two graphs in Figure 2.9 show the results of the test and show that the angle to the X axis is unreliable when comparing it to the encoder angle. The Y angle however is more periodic and predictable despite it not being perfectly sinusoidal. There is a sharp change, depending on the direction of the swing, that occurs when it passes through the zero angle. It is unclear why this happens as the code used to calculate this value is not available for inspection, however, it is likely to do with the two ways it calculates the angle with the sharp gradients due to it swapping between the two methods. The robot is likely to be confused as the the code was not written with a swinging motion in mind. Furthermore the smooth peaks are almost exactly in phase with the angle peaks. The slight offset could be because of the delay in the transfer of data from the robot over the network.

The Y angle function could be used to determine the swing location, however, specialised code must be written to interpret the values because of its non-sinusoidal nature.

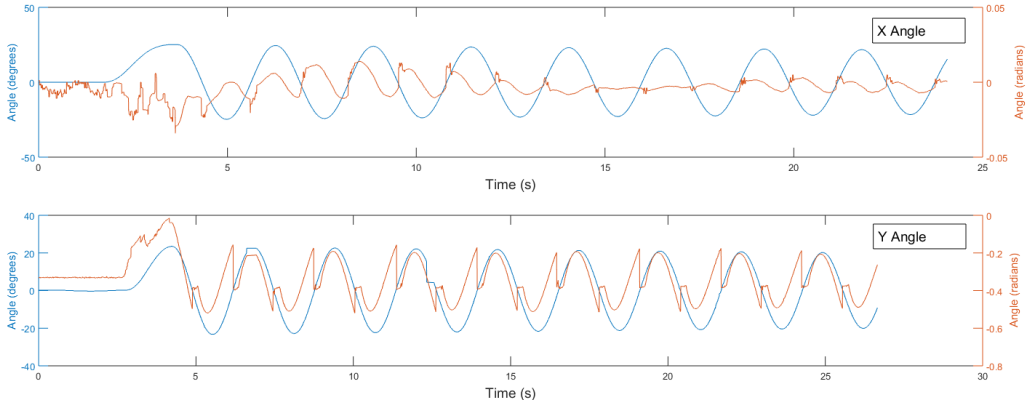


Figure 2.9: Base Readings from NAO angle function.

2.4.3.2 Gyrometers

Figure 2.10 shows the X and Y gyrometer values recorded with the angle of the pendulum. The X values show the natural side-to-side perturbations of the swing, as is expected. The Y values show the rotation of the robot with respect to its Y axis and it can be seen that these are sinusoidal with the swing but a factor of $\frac{\pi}{2}$ out of phase. This is expected as the robot will experience the greatest amplitude of angular rotation when it is travelling fastest. This would be at a swing angle of zero.

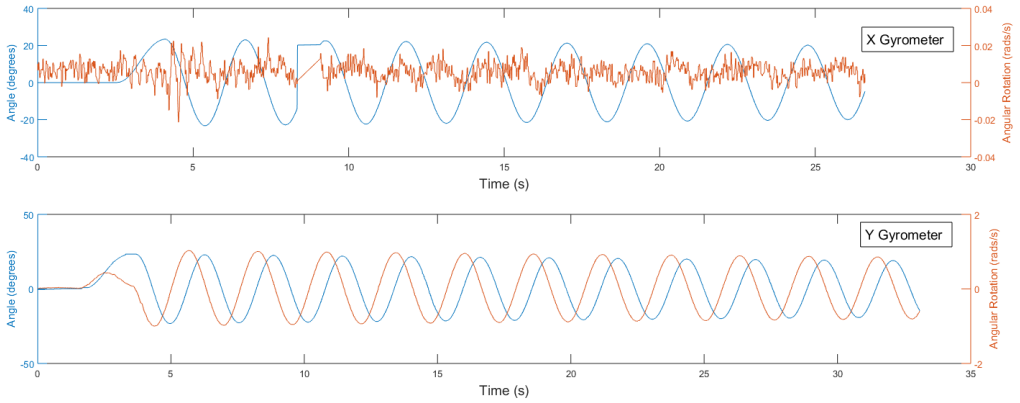


Figure 2.10: Base readings from the Gyrometers.

Figure 2.10 shows that the Y gyrometer is the only capable of being used to compute the current swing angle or position through the swing. This is because of its sinusoidal nature with a period equivalent to the swing. The X gyrometer is measuring the side-to-side perturbations of the swing and therefore is not useful.

2.4.3.3 Accelerometers

The three accelerometers on the robot were tested and the results are shown in Figure 2.11. All three show predictable, periodic behaviour in the simple pendulum. The X axis acceleration graph shows a secondary peak which is due to the presence of gravity as the robot is measuring both its acceleration in the swing and towards the ground. The X accelerometer gives a noisy curve which will means an extra, noise cancelling step will be needed if it is to be used.

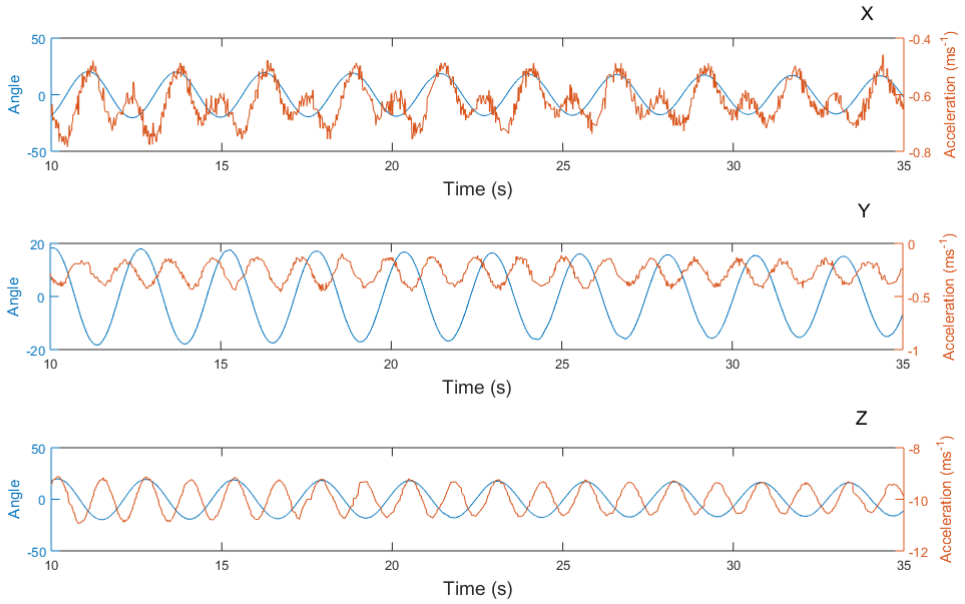


Figure 2.11: Base readings from the Acceleromters.

The Y accelerometer has a smoother curve with a more sinusoidal nature than the X axis sensor. The Y accelerometer varies with a period twice that of the pendulum which means that it could be used in a programme to calculate when the robot should move.

2.4.3.4 Findings

From the tests it is clear that the most useful sensor for computing the position in the swing is the Y gyrometer which has a shape and period that matches the sinusoidal nature of the swing motion with a phase difference of $\frac{\pi}{2}$. Its predictability would allow the robot to know when it will reach the peak of the swing and, therefore, when it should start its movements.

The angle function, whilst not perfectly sinusoidal, is predictable based on swing angle. It can therefore be used to compute the swing angle, however, it would be more difficult than the gyrometer. The difference in the peaks and troughs can be used to calculated a period for the swing and the robot could therefore simply time when it should perform its next swing. The same can be said of the accelerometers. The Y and Z accelerometers have a period half that of the swing and with very similar code would be able to output the same period as the angle function.

2.4.4 Vision Sensors

2.4.4.1 Camera Systems

The following was contributed by: E. Humphreys

The robot is equipped with two cameras fixed on the head as shown in Figure 2.12. Both of these cameras can output a resolution of 640x480 (pixels) at 30 frames per second (fps) [16].

The camera outputs at any speed from 5fps to 30fps, and can downgrade its outputted resolution to 160 pixels wide by 120 pixels high resolution (QQVGA) from its original 640 pixels wide by 480 pixels high resolution (VGA) according to the documentation [16]. Each camera has a field of view of 47.64°, with a central one facing 1.2° below the horizontal plane (with the neck motor set at 0°) and a lower camera facing 39.7° below the horizontal plane, intended for allowing the robot to observe obstacles near its feet. This lower camera is not as useful due to its positioning (the body and swing blocks the view) so it was decided to not be used.

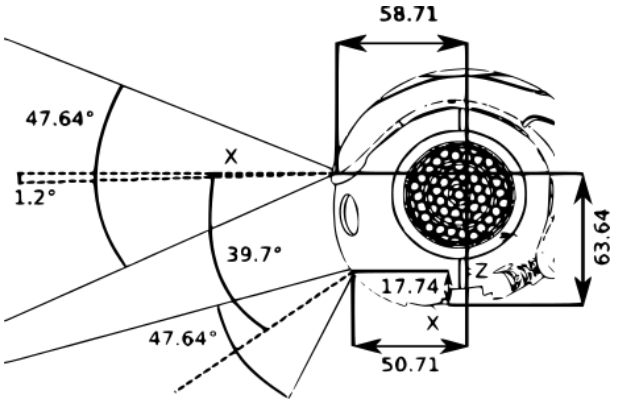


Figure 2.12: Range of Camera Vision [16].

2.4.4.2 Vision API

The robot's API (Application Program Interface) has a built in range of modules that allow the vision data to be processed in many ways. Some of these methods are useful for designing feedback systems required for making the robot swing. Out of the many available [17], two are of particular interest for use in tracking the motion of the robot.

`ALRedBallDetection:` ALRedBallDetection is a module that allows NAO to detect in its vision a red circular blob of pixels, hence allowing the detection of a red ball. This returns an event"redBallDetected" which allows for the position of the red ball (in the field of vision), the scale of the ball (in field of vision) to be measured. The documentation for this module was sparse, however, and there was no indication that more than one ball could be tracked.

`ALLandMarkDetection:` ALLandMarkDetection is a module that allows NAO to track the positions of various "landmarks", circular patterns with specific identifying numbers. When the vision software detects a landmark, it returns a"LandmarkDetected" event with information about the landmark number, position, and scale. The documentation for this was more detailed, and an example piece of code allowed for us to construct our own program that allows for the tracking of the landmarks, and according to the documentation more than one landmark could be tracked at once. Our tests showed that the analysis of position and size could be done in ≈ 0.05s for two landmarks. This was determined by checking at what rate the LandmarkDetected event updates the memory (and as such how fast it can be accessed).

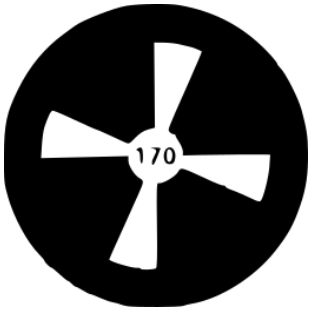


Figure 2.13: Example Landmark provided in the NAO documentation [17].

We decided that if any tracking was to be done with the vision software, the `ALLandMarkDetection` module would be preferable to the `RedBallDetection`. Preliminary tests were performed to determine the maximum range at which the landmarks could be detected. While occasionally they could be detected at a larger range the maximum distance continual detection would be observed at was $\approx 2.5 \pm 0.1$m. To ensure the robot would always have the landmarks in view it was positioned approximately 1m away

from the landmarks whenever tested.

2.4.4.3 Measurements of Tracking Ability

Both the size and position tracking of the `ALLandMarkDetection` were tested. Before this was done, tests were performed to determine whether fps and resolution would have an impact on the robots ability to track the landmarks. To perform this test we took measurements of a landmark in front of the robot as let the robot swing freely (no joint movement). Four measurements were taken, so that all variations of the robot's camera were taken into account (i.e high fps/low fps, high Resolution/low Resolution). Preliminary readings showed the camera would only "update" every 0.2s. This was too long for this method to be useful in analysing the landmarks and reacting. However, it was discovered that this was due to the speed at which NAO reads the "LandmarkDetected" event (every 0.2s), and was altered to measure this reading every 0.05s, this resulted in Figures 2.14 and 2.15.

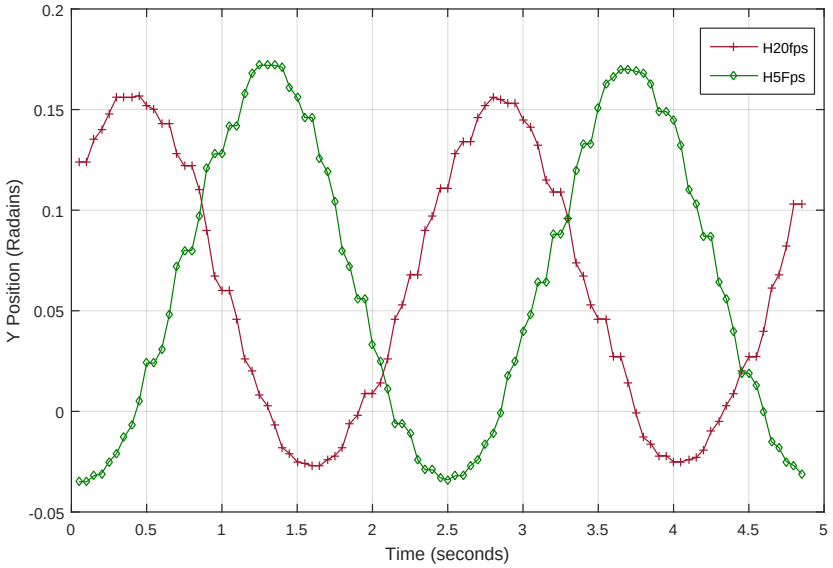


Figure 2.14: Graph of camera tracking at high resolution and at varying fps.

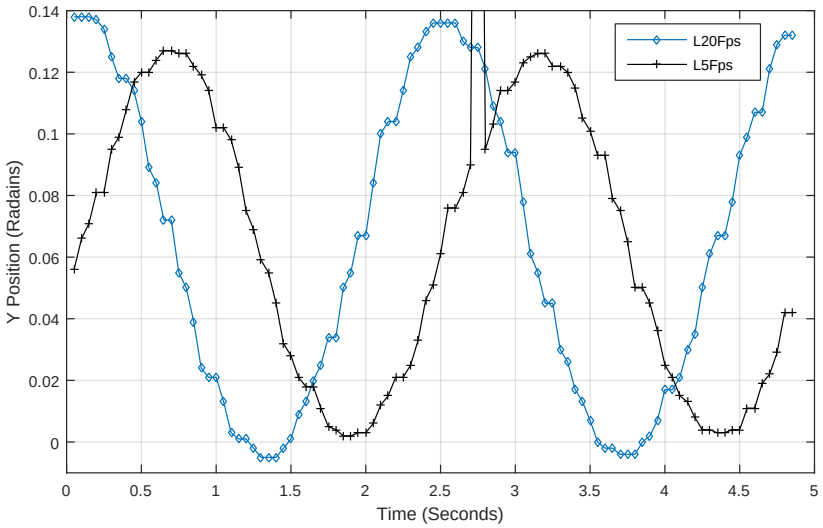


Figure 2.15: Graph of camera tracking at low resolution and at varying fps.

As can be seen, there appears to be no difference between the different resolutions or the different frame-per-second limits. According to the graphs, the landmarks are tracked every 0.05s, with the occasional marker being tracked twice, which indicates that the vision input module updates the footage slightly slower than the memory is accessed. It also shows that the resolution and fps are downgraded after analysis. This is to allow the robot to transmit images as close to real time as possible over WiFi, as the higher the image quality and fps, the larger the files and the slower it would take to transfer.

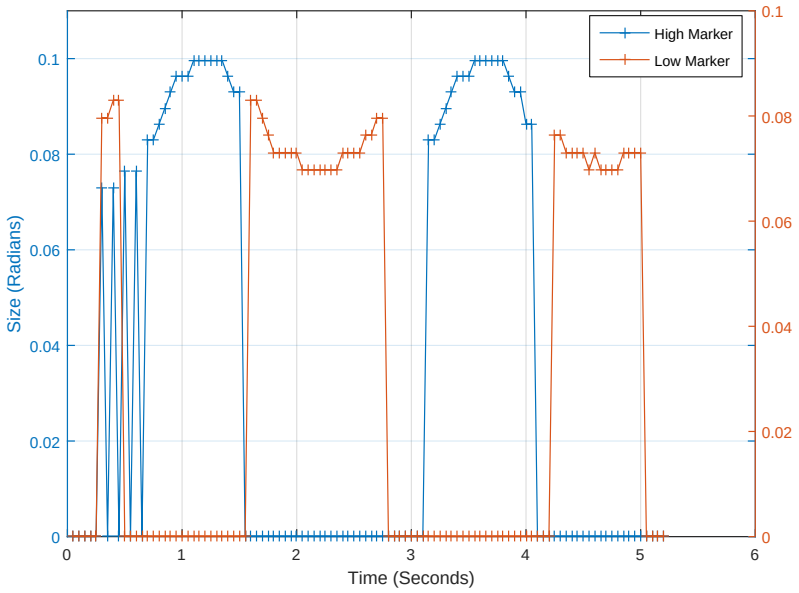


Figure 2.16: Graph of tracking ability of two landmarks when moving.

Figure 2.16 is the results of test of how the robot can track the size of two markers, each positioned at 112±1cm away and at 22±1cm and 111±1cm above ground, as it swings. The errors on these values are not that important as long as the total distance of the landmark to the robot does not exceed 2.5m the robot can still track it. The goal of this test was to see primarily if the robot is capable of tracking the extremes of its motion (the furthest and closests points to the landmarks) with the secondary goal of examining how well the robot tracks the size of landmarks. As can be seen in Figure 2.16, the robot is capable of identifying the furthest and closest positions to the marks (the troughs and peaks of the graph, respectively). Any measurement taken at 0 radians is a result of the camera not detecting the specific landmark. There is a strange quantisation when using the size detection used in `ALLandMarkDetection` that was not present in the angular position detection. While it can still be used to determine where the robot is in the swing motion, it does have the flaw in not being as accurate as the position detection. As such, any feedback systems using the ALLandMarkDetection should use the position tracking as it produces a more accurate result.

2.4.4.4 Findings

After examining the vision hardware/software of the robot it was decided that this was a prospective avenue to investigate in relation to a feedback system. The `ALLandMarkDetection` module was found to be sufficiently capable of tracking landmarks and would be able to be used in a feedback system without much work.

2.5 Final Swing Design

The following was contributed by: M. Toon

2.5.1 Physical Swing Design

The various limitations of the robot were accounted for when generating ideas for the design needed. Primarily the range of motion tests were used as these were the most limiting factors for the robot. Its strength proved adequate for any motion required and the sensors used would not be obstructed. The design had to be decided with all measurements swiftly as the swing would take some time to be fabricated. Modifying the existing design reduced the time to wait for the swing to be complete.

The final design chosen for the swing was to add a crossbar in front of the robot at shoulder height to attach the hands. A person swinging on a swing holds the chains in line with their shoulders when in a vertical sitting position. As this is impossible to achieve due to the restricted movement shown in Section 2.2.1, the bar was attached to the existing swing at shoulder height and extended forwards by 150mm such that the robot was upright with minimum arm extension. This value was 2.7mm shorter than the actual value of 152.7mm to account for the thickness of the material on the robot's hands. The components for the swing were machined from aluminium with very high precision and so any uncertainty in length was assumed to be negligible.

The double joints on the length of the swing were added to simulate the distortion from the force of the hands pulling on the chain. For the force to be applied in the correct position, the crossbar was attached in the centre of the two joints. This is not an ideal solution to represent a human pulling on a swing chain as the force is not translated exactly, instead a torque is created at the fixing between the swing and the crossbar.

The hands of the robot were attached to the swing using hook and loop fasteners wrapped around the the back of the hand and onto the bar. The rubber grips were set at shoulder width 275mm and extend vertically from the cross bar. The grips rotate on the vertical axis to compensate for the limited wrist movement of the robot. This allows the robot to retract its arms freely and push its elbows out sideways without needing to flex the wrist joints.

The previous swing was designed to have the robot in a standing position and so the distance between the double joints and seat was too large. This was corrected by recreating the lower aluminium poles with a shorter length to raise the seat. This also increased the natural frequency of the swing due to the reduced total length.

Improving the seat: The previous seat was milled from a solid block of high density plastic which fixed to the bottom of the lower poles with two bolts. Two cavities were milled out for the legs to rest in such that the robot does not slide on the seat. To further restrict the movement of the legs, a 5.0mm hole was drilled 5.0±0.5mm in from the edge of the leg cavities so that cable ties may be looped round the legs to hold the robot down to the seat. The holes were placed at the width of the leg such that the cable ties pull perpendicular to the surface of the seat so that maximum force was applied to hold the robot down in place. This was important because the legs were used as an anchor point for the movement of the back and prevented robot falling when it moved. Another cavity was created behind the leg cavity to allow the hip joint to have free movement when the robot leaned backwards, otherwise the hip would have collided with the surface of the seat and restricted the movement.

Mass Reduction: Extra cavities were milled out to remove excess mass from the seat. Before the extra mass is removed, the seat weighed 1356.0±0.1g and was reduced to 1065.2±0.1g. This was done to reduce the overall mass of the swing to make it easier for the robot to start the swinging motion. A larger force would be required to accelerate a more massive swing, so a lighter seat is beneficial. It was decided that reducing the mass of the long aluminium poles by replacing them with hollow tubes would require too much time for a small reduction in the weight of the swing.

Increasing the width: To increase the width of the swing a wider aluminium bar was substituted in the original position of the seat on which the seat may then be mounted on top as shown in Figure 2.17. The new length of the shorter aluminium poles accounts for the difference in height to the hands when

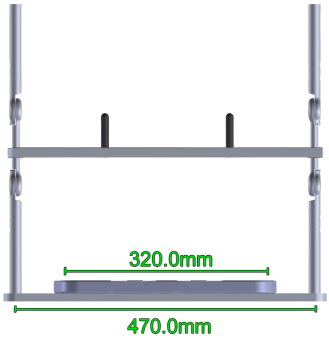


Figure 2.17: Front view of the lower part of the swing design with the seat mounted on the aluminium bar displaying the difference in width before and after.

the seat is mounted in the new position. A bracket was also fabricated to extend the width of the swing at the top pivot shown in Figure 2.18.

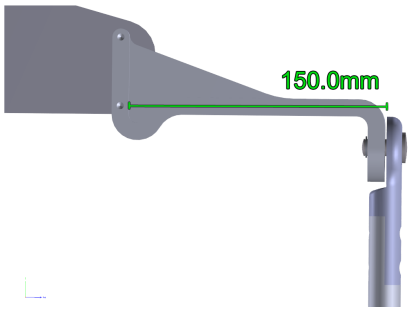


Figure 2.18: Replacement bracket used to extend the width of the swing at the pivot end by 150.0mm.

2.5.2 Modelling the Swing in Webots

The following was contributed by: J. Sweeney

The creation of a custom object within Webots requires the use of various, in-built classes, referred to as nodes. These are placed in a hierarchical list of objects known as the Scene Tree (see Figure 2.19 below). To construct a complex object, it must first be decomposed into constituent parts and then built up using the appropriate nodes, with subsidiary nodes placed within the *children* field of their parent.

To replicate a swing faithfully, six primary nodes were used: *Solid*, *Shape*, *HingeJoint*, *Transform*, *Robot* and *Connector*. *Solid* nodes, are the base class for which collision detection occurs. As such, they posses fields for position and rotation, as well as additional fields: *Physics*, *boundingObject* and *contactMaterial*. The *Physics* field allows for the specification of mass; centre of mass; and mass distribution – thereby enabling the computation of forces on an object. *contactMaterial* allows for the specification of material type for the object, allowing for the specification of Coulomb friction and coefficient of restitution between objects, whilst *boundingObject* determines the boundary of the solid and is typically inherited from a *Shape* node. The *Shape* node determines how an object is rendered within Webots using a bank of typical three dimensional shapes (e.g. "box"/cuboid, cylinder, sphere, etc.) In the instance of curved geometries, objects are reduced to convex polyhedra. To increase the accuracy of this approximation, the number of polygons comprising the curved face can be specified using the *subdivision* field.

The *HingeJoint* class replicates pivot points. It simulates joints with a single degree of rotational freedom. In addition to joint axis and position: static friction; damping constant; and spring constant parameters may be applied.

To position objects with respect to their parent node, *Transform* nodes were introduced. These allow

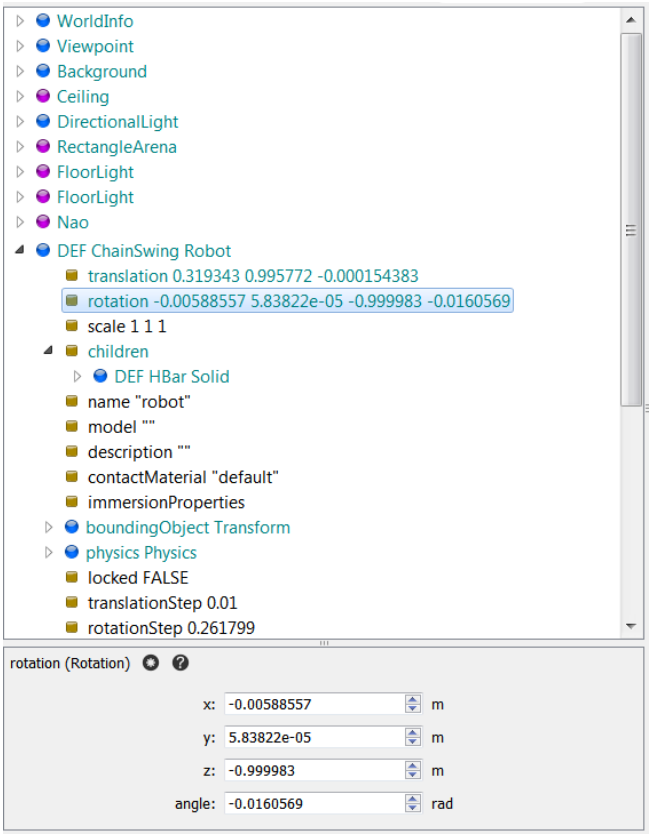


Figure 2.19: Example Scene Tree depicting multiple objects.

for the specification of a local coordinate system relative to that of the parent node.

The *Robot* node is a direct subclass of the *Solid* node. As such, it exhibits all of the properties of the superclass whilst additionally having exclusive access to a class of nodes: *Device*. Members of the *Device* class perform unique functions within Webots, with the *Connector* node of primary concern (descriptions of additional *Device* members can be found in Section 4.3.1. In order to implement these nodes, the *Robot* node must be the topmost node within an object tree.

The *Connector* node is a computationally efficient way to simulate the docking of two robots. By adjusting the tensile and shear strengths of two linked nodes, connectors can be adapted to permanently affix limbs of an object to one another.

Following the above, the swing was implemented using the node structure as can bee seen in Figure 2.20 below.

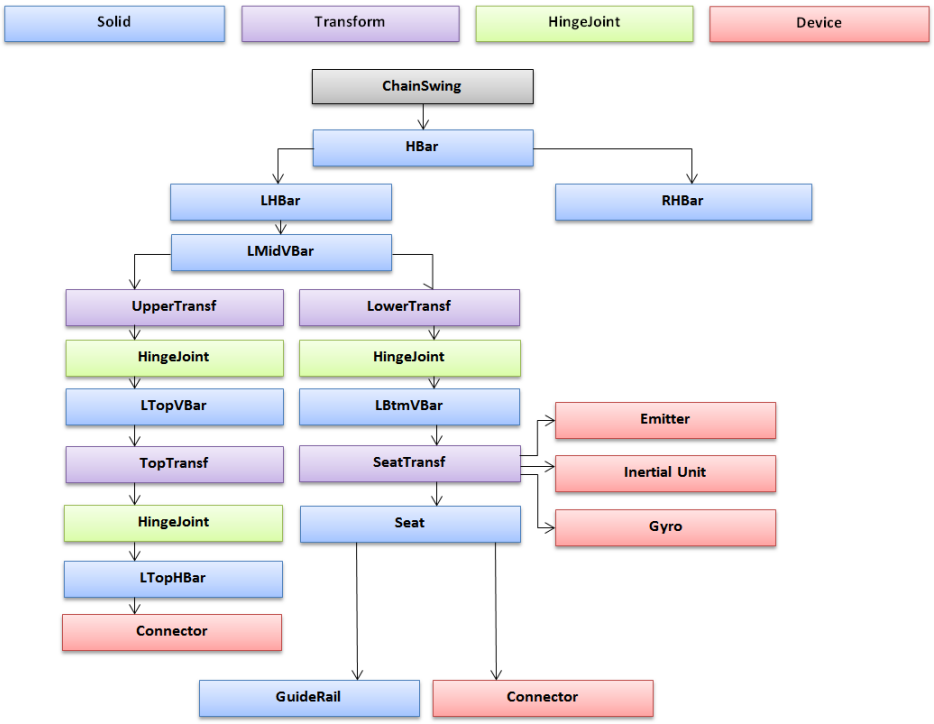


Figure 2.20: Hinged swing node diagram with *ChainSwing Robot* node denoted in grey. Arrows depict direct children. *Shape* nodes and the right hand swing arm have been omitted for clarity. Examples of additional devices used in Section 4.3 are shown to the right of the *SeatTransf* node.

Each **Bar* node consisted of a cylindrical *Shape* node from which it inherited *boundingObject* parameters. These were approximated using the default polygon subdivision number: twelve. This was chosen as improving the objects resolution was found to dramatically increase runtime. As the Scene Tree follows a top-down structure, the right hand side of the swing was able to inherit its *boundingObject* properties from the left hand arm and thus reduce computational intensity.

Uniquely, the nodes *LTopHBar* and *RTopHBar* (not displayed) had their physics fields disabled. This prevented their motion about their corresponding *HingeJoint*s. This, however, was not sufficient to counter the effects of gravity, as their positions were defined relative to parent physics enabled objects – due to the *Transform* nodes above them in the object tree. As a result, a further, physics disabled, robot consisting of two cylinders had to be connected to each arm, to suspend the swing above the ground. This is shown in Figure 2.21.

Due to the rigid nature of the metal bars, the *HingeJoint* nodes were assumed to have a null spring constant. The damping coefficient was chosen over static friction to reproduce the decay effects of swing oscillation. This was due to its velocity dependence, thereby making it second order accurate[18]. This is opposed to friction, which is modelled linearly in ODE[19]. This was found experimentally to be $0.00679 \pm 0.00025 Nsm^{-1}$ (see Appendix A.3 for derivation).

To ensure the robot remained in place on the swing, the relative Coulomb friction between it and the seat, and it and the horizontal bar were set to be infinite, so as to prevent any sliding. Similarly, the coefficient of restitution –referred to as *bounce* – was set to zero within the simulation. This thereby removed the effects of momentum normally incident to the seat. The addition of the *Solid* node, *GuideRail*, was included to prevent the raising of the robot's legs from the seat during its motion. This was necessary as the robot does not have a dedicated motor to articulate its torso, but instead was required to alter its hip angle. Ordinarily, this would result in the raising of upper portion of its legs, however to prevent this experimentally, the NAO was secured using cable ties to the seat. These physical restraints, moreover, provided adequate justification for the aforementioned simulation constraints.

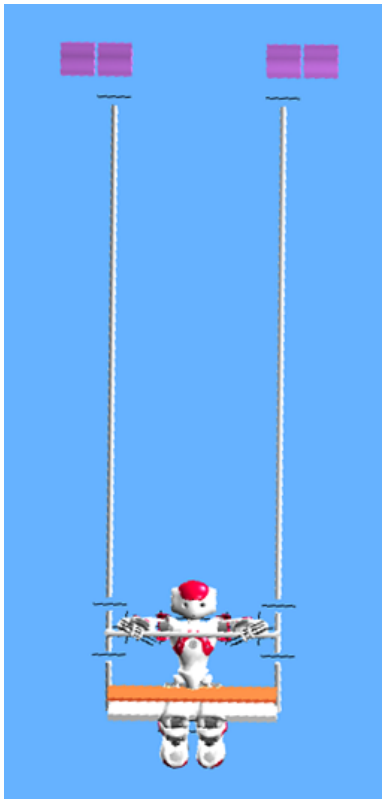


Figure 2.21: Webots simulation of swing. Physics disabled objects are coloured purple; joints are represented by black lines; *GuideRail* is coloured orange.

2.6 Chapter Conclusion

The following was contributed by: E. Humphreys

The initial tests and analysis of the robot were very useful in providing data for the theoretical model and swing design group. The majority of the limbs were found to be able to move to a near complete range of motion, and those that could not, namely the hip joints, were restricted by the seat of the swing more than the degradation of the joints. The limb lengths and how they affect the range of motion was also analysed. Table 2.3 was produced and used to create a simple two dimensional model of the robot which was then moved to determine what movements would be required to cause a maximum and minimum extension. The minimum swing width of 427.0±0.5mm was also determined from the results.

Strength tests were carried out to measure how well the robot could move itself on the swing with its own motors. These tests were only measured on the motors critical to the movement of the robot on the swing in a seated position. The torques of the vital motors were measured by ordering the arm to move with a preloaded Newton metre attached. Our actual values for the torques of these motors was much less than we expected, however we explained this deviation as firstly the metres were not as accurate as we would have liked, and secondly the robot would move other limbs in response to the force of the Newton metre which would then also start to overheat the limbs, which would cause the motors to loosen and stop applying force (as a safety feature of the robot). We did however determine the robot can lift itself with its own hands from a seating position, which was useful to know when designing the swing and the model.

Temperature tests were performed to analyse the overheating of the limbs as the robot moved. A sit up test was performed on the robot repeatedly and the temperature of the robot's limbs over time were measured. This was important as the robot's hands were said to overheat by the previous users of the robot, something we verified. We also found that for most of the limbs, the robot reaches a certain point where dissipation of heat is approximately equal to that gained by friction from moving, due to

the increased "looseness" of the joint due to the aforementioned safety feature. From the results, it was determined the robot would have to be tethered to the swing by the hands as relying on the grip was unsafe, as the robot could loose grip and damage itself mid swing.

Tests of the gyrometer, accelerometers, and NAO's built in "angle" function were done. The angle function seemed useful for determining the angle to the Y axis for the robot, while the X axis angle was unreliable, and did not have much documentation on the methods of how it calculates these angles. As such specialised code would have to be written for this to be of any use for feedback or machine learning systems. The gyrometer tests showed the most promise of the results, especially the Y gyrometer, which produced a very similar plot to the encoder angle, despite it being out by a factor of $\frac{\pi}{2}$ out of phase. The accelerometers also produced results which indicated they may be useful in determining the period of the motion of the swing.

The camera tracking investigations produced results which indicated the vision hardware and software in the robot is capable of tracking the position of the robot in the swing very well through the use of `ALLandMarkDetection` and various Landmark markers. It was found to be useful up to a range of 2.5m and allowed the robot to detect the movement of a landmark in its field of vision which produces a plot proportional to the position of the robot in the swing. Out of the four values `ALLandMarkDetection` measures per landmark (two size and two position) the position detection was found to give a reading every 0.05s which allows for an accurate detection of the robot's position.

The swing design was altered in the following ways compared to the one provided;

- A crossbar was added across the swing in front of the robot.
- Double joints were added around the crossbar to simulated distortion from tension on an actual swing from the torque from the robot's limbs.
- The hands were attached to the swing so that the hand could still move freely, but did not rely on the hands gripping the swing.
- The seat was raised to decrease the distance between the double joints and the seat (since the robot is now sitting, not standing).

Other alterations included milling the seat to allow the robot's legs to be secured in the seated position, reducing the mass of the seat by drilling cavities in it (to reduce the force needed to move the swing), and increasing the width of the swing by adding a bracket and adding a wider bar on the bottom of the swing to allow the robot's arms to move freely with its motions.

Lastly the swing was modelled in Webots. The program was used with the provided NAO model in the program was used and a swing was designed to best approximate the actual swing. Some approximations included applying an infinite friction between the robot and the swing to simulate the robot being tethered to the seat, and locking the legs in place with a rail to allow the hip motors to move the torso instead of the legs (which is how the robot would move when attached to the seat). Despite these approximations a model that effectively represented the NAO seated on the swing was produced and could function similarly to the real life robot.

Chapter 3

Theory Behind Swinging Motion

3.1 Analytical Models

The following was contributed by: Z. Hodgins

Lagrangian mechanics is an alternative to classical Newtonian mechanics to make a system easier to mathematically model by making it easier to calculate the equation of motion. For the duration of this report, all systems will be analytically modelled using Lagrangian methods. The Lagrangian is defined as,

$$\mathcal{L} = T - V \tag{3.1}$$

where $\mathcal{L}$ is the Lagrangian, T is the total kinetic energy of the system and V is the total potential energy [20]. $\mathcal{L}$ is a new quantity that makes manipulation of the equations of motion easier in the future. To use Equation 3.1 all the contributions of the kinetic and potential energy of the system must be found, in terms of the varying coordinates.

For a trivial example, consider the mass at the end of a pendulum. The kinetic energy, T, is known to be $\frac{1}{2}m(\dot{x}^2 + \dot{y}^2)$, where m is the mass of the bob and $\dot{x}$ and $\dot{y}$ are the velocities at which it moves in the x, y co-ordinate plane. For all future notations, a dot above a variable represents this differentiated with respect to time, t. For rotating systems, such as a pendulum, polar coordinates are far more useful. Using the conversions $x = lsin\theta$ and $y = lcos\theta$, the kinetic energy becomes,

$$T = \frac{1}{2}ml^2\dot{\theta}^2 \tag{3.2}$$

The only contribution of the potential energy to this system is the gravitational force acting on the mass. It can be clearly seen that this is $V = mgh = -mglcos\theta$. Putting these together gives a final Lagrangian of,

$$\mathcal{L} = \frac{1}{2}ml^2\dot{\alpha}^2 + mglcos(\alpha) \tag{3.3}$$

It can be noted that for this Lagrangian, as well as for all the following ones in this report, gravity is taken to be acting downwards. To calculate the acceleration of each variable, the Euler-Lagrange equation is used,

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) = \frac{\partial \mathcal{L}}{\partial q} \tag{3.4}$$

Where $\mathcal{L}$ is the Lagrangian and q is the generalised coordinate you want the equation of motion of, and $\dot{q}$ is the differential with respect to time. This relationship will be used in all analytical sections. Applying it to the pendulum above gives:

$$\ddot{\alpha} = -\frac{g}{l}\sin\alpha \tag{3.5}$$

The aim of the following examples is to find analytical models for systems as close to the swinging robot as possible. Four systems were modelled, each building on the previous one to refine the likeness of the models.

3.1.1 Triple Pendulum

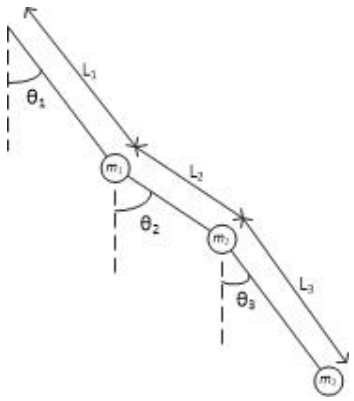


Figure 3.1: Diagram of the triple pendulum system.

The first non-trivial model used is the triple pendulum. This links to the robot by considering m_3 as the centre of mass of the robot and m_2 representing the seat; so that the robot's centre of mass can move around the seat. This is slightly unrealistic as the robot cannot shift its whole centre of mass upside down in the sitting position. The top mass, m_1 corresponds to the hinge in the swings rods, which is equivalent to where the a human would hold the swing, adding an extra pivot point.

Let (x_i, y_i) be the position of the ith mass. The kinetic energy of the system is given by,

$$T = \frac{1}{2}m_1\left(\dot{x}_1^2 + \dot{y}_1^2\right) + \frac{1}{2}m_2\left(\dot{x}_2^2 + \dot{y}_2^2\right) + \frac{1}{2}m_3\left(\dot{x}_3^2 + \dot{y}_3^2\right) \tag{3.6}$$

We now want to write the kinetic energy in terms of the generalised coordinates θ_1, θ_2 and θ_3. The positions of the three masses in terms of the generalised coordinates are as follows,

$$\begin{aligned} x_1 &= l_1 sin\theta_1 & y_1 &= -l_1 cos\theta_1 \\ x_2 &= l_1 sin\theta_1 + l_2 sin\theta_2 & y_2 &= -(l_1 cos\theta_1 + l_2 cos\theta_2) \\ x_3 &= l_1 sin\theta_1 + l_2 sin\theta_2 + l_3 sin\theta_3 & y_3 &= -(l_1 cos\theta_1 + l_2 cos\theta_2 + l_3 cos\theta_3) \end{aligned} \tag{3.7}$$

Taking the time derivative of these coordinates gives,

$$\begin{aligned} \dot{x_1} &= l_1\dot{\theta_1} cos\theta_1 & \dot{y_1} &= l_1\dot{\theta_1} sin\theta_1 \\ \dot{x_2} &= l_1\dot{\theta_1} cos\theta_1 + l_2\dot{\theta_2} cos\theta_2 & \dot{y_2} &= l_1\dot{\theta_1} sin\theta_1 + l_2\dot{\theta_2} sin\theta_2 \\ \dot{x_3} &= l_1\dot{\theta_1} cos\theta_1 + l_2\dot{\theta_2} cos\theta_2 + l_3\dot{\theta_3} cos\theta_3 & \dot{y_3} &= l_1\dot{\theta_1} sin\theta_1 + l_2\dot{\theta_2} sin\theta_2 + l_3\dot{\theta_3} sin\theta_3 \end{aligned} \tag{3.8}$$

By using Equation 3.6 the final expression for the kinetic energy can be found to be,

$$\begin{aligned} T = &\frac{m_1}{2} l_1^2\dot{\theta_1^2} \\ &+ \frac{m_2}{2}(l_1^2\dot{\theta_1^2} + l_2^2\dot{\theta_2^2} + l_1 l_2\dot{\theta_1}\dot{\theta_2} cos(\theta_1 - \theta_2)) \\ &+ \frac{m_3}{2}((l_1^2\dot{\theta_1^2} + l_2^2\dot{\theta_2^2} + l_3^2\dot{\theta_3^2} + l_1 l_2\dot{\theta_1}\dot{\theta_2} cos(\theta_1 - \theta_2) \\ &+ l_1 l_3\dot{\theta_1}\dot{\theta_3} cos(\theta_1 - \theta_3) + l_2 l_3\dot{\theta_2}\dot{\theta_3} cos(\theta_2 - \theta_3)) \end{aligned} \tag{3.9}$$

The potential energy is equal to the sum of the contributions of the three masses, with the y coordinates being as stated in Equation 3.7. Therefore the potential energy of the system is equal to,

$$\begin{aligned} V = &- m_1 l_1 g cos\theta_1 \\ &- m_2 l_2 g(cos\theta_1 + cos\theta_2) \\ &- m_3 l_3 g(cos\theta_1 + cos\theta_2 + cos\theta_3) \end{aligned} \tag{3.10}$$

Using Equation 3.1, the Lagrangian can be found to be,

$$\begin{aligned}\mathcal{L} =&\frac{m_1}{2}l_1^2\dot{\theta_1^2} + \frac{m_2}{2}(l_1^2\dot{\theta_1^2} + l_2^2\dot{\theta_2^2} + l_1l_2\dot{\theta_1}\dot{\theta_2}cos(\theta_1-\theta_2)) + \frac{m_3}{2}(l_1^2\dot{\theta_1^2} + l_2^2\dot{\theta_2^2} + l_3^2\dot{\theta_3^2} + \\ & l_1l_2\dot{\theta_1}\dot{\theta_2}cos(\theta_1-\theta_2) + l_1l_3\dot{\theta_1}\dot{\theta_3}cos(\theta_1-\theta_3) + l_2l_3\dot{\theta_2}\dot{\theta_3}cos(\theta_2-\theta_3)) + m_1l_1gcos\theta_1 + \\ & m_2l_2g(cos\theta_1 + cos\theta_2) + m_3l_3g(cos\theta_1 + cos\theta_2 + cos\theta_3)\end{aligned} \tag{3.11}$$

where all variables are defined as in Figure 3.1. For the swinging robot the aim is to maximise the amplitude of motion. This means maximising $\dot{\theta_1}$, so the acceleration equation for the pendulum is needed. This was found by applying Equation 3.4 to the Lagrangian for the system to give,

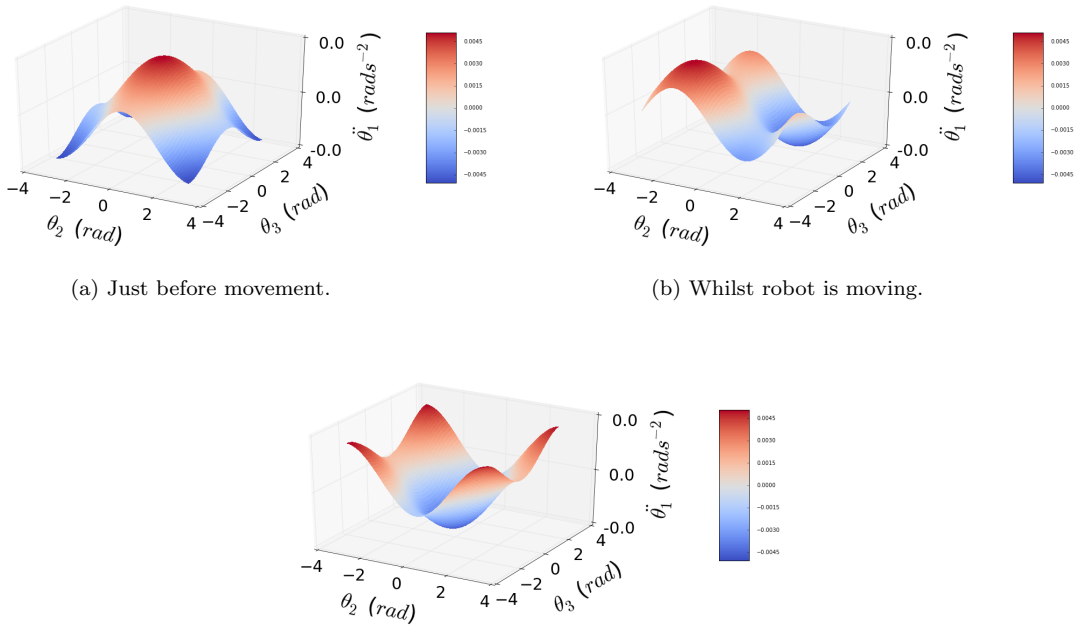
$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\dot{\theta_1}} =& l_1^2\dot{\theta_1}(m_1+m_2+m_3) + \frac{1}{2}m_2l_1l_2\dot{\theta_2}cos(\theta_1-\theta_2) \\ &+ \frac{1}{2}m_3l_1l_2\dot{\theta_2}cos(\theta_1-\theta_2) + \frac{1}{2}m_3l_1l_2\dot{\theta_3}cos(\theta_1-\theta_3) \\ \implies \frac{\mathrm{d}}{\mathrm{d}t}(\frac{\partial\mathcal{L}}{\partial\dot{\theta_1}}) =& l_1^2\ddot{\theta_1}(m_1+m_2+m_3) + \frac{1}{2}l_1l_2\ddot{\theta_2}(m_2+m_3)cos(\theta_1-\theta_2) \\ &+ \frac{1}{2}l_1l_2(\dot{\theta_2}^2 - \dot{\theta_1}\dot{\theta_2})(m_2+m_3)sin(\theta_1-\theta_2) \\ &+ \frac{1}{2}m_3l_1l_3(\ddot{\theta_3}cos(\theta_1-\theta_3) + (\dot{\theta_3}^2 - \dot{\theta_1}\dot{\theta_3})sin(\theta_1-\theta_3)) \\ \frac{\partial\mathcal{L}}{\partial\theta_1} =& -\frac{1}{2}l_1l_2\dot{\theta_1}\dot{\theta_2}(m_2+m_3)sin(\theta_1-\theta_2) - \frac{1}{2}m_3l_1l_3\dot{\theta_1}\dot{\theta_3}sin(\theta_1-\theta_3) \\ &- gsin\theta_1(m_1l_1+m_2l_2+m_3l_3)\end{aligned} \tag{3.12}$$

Which finally gives the equation of motion of the triple pendulum as,

$$\begin{aligned}&-\frac{1}{2}l_1l_2\dot{\theta_1}\dot{\theta_2}(m_2+m_3)sin(\theta_1-\theta_2) - \frac{1}{2}m_3l_1l_3\dot{\theta_1}\dot{\theta_3}sin(\theta_1-\theta_3) \\ &- gsin\theta_1(m_1l_1+m_2l_2+m_3l_3) + \frac{1}{2}l_1l_2\ddot{\theta_2}(m_2+m_3)cos(\theta_1-\theta_2) \\ &+ \frac{1}{2}l_1l_2(\dot{\theta_2}^2 - \dot{\theta_1}\dot{\theta_2})(m_2+m_3)sin(\theta_1-\theta_2) \\ &+ \frac{1}{2}m_3l_1l_3(\ddot{\theta_3}cos(\theta_1-\theta_3) + (\dot{\theta_3}^2 - \dot{\theta_1}\dot{\theta_3})sin(\theta_1-\theta_3)) \\ \therefore \ddot{\theta_1} =& \frac{}{l_1^2(m_1+m_2+m_3)}\end{aligned} \tag{3.13}$$

If the extra two extensions, l_2 and l_3 are held stationary, $\dot{\theta_2} = \dot{\theta_3} = \theta_2 = \theta_3 = 0$, then the equation of motion returns to that of the simple pendulum with unevenly distributed mass.

This motion was modelled to see the effect varying values of θ_2, θ_3 and their time derivative have on the motion of the pendulum. For the modelling, the masses and lengths were set to be values in proportion to if they represented the robot's centre of mass, the swing seat and the hinge in the swing.



(a) Just before movement.

(b) Whilst robot is moving.

(c) Just after movement.

Figure 3.2: Graphs to show the variation in the pendulums acceleration before, during and just after the robot moves.

Figure 3.2 shows how the angular acceleration of the pendulum changes with varying θ_2 and θ_3, which range approximately by $\pm 180°$. Before the movement, the angular velocity of each mass is set to zero, whilst the acceleration is positive. Figure 3.2a shows that the closer θ_2 and θ_3 are to $0°$, the higher the angular acceleration of the pendulum. Figure 3.2b corresponds to when the instantaneous acceleration is zero, but the velocity is constant; this produces Figure 3.2b. Finally, Figure 3.2c shows that variations when the acceleration is zero and the velocity is the negative of that in Figure 3.2a.

This shows some clear results about the best motion for a triple pendulum. However, the robot would not realistically be able to move it's centre of mass in the motions required to make these maximum accelerations of the pendulum.

It can be noted that the physical swing can be modelled as a triple pendulum with m_1 and m_2 being the two hinges in the sides and m_3 being the seat. The Lagrangian for the triple pendulum is used in the derivation of the equations of motion for the more complex systems covered later in this Chapter. This is why establishing the equation of motion for the triple pendulum is vital, even if it seems like this system is very different to the swinging one.

3.1.2 Dumbbell Model

The following was contributed by: O. Diba

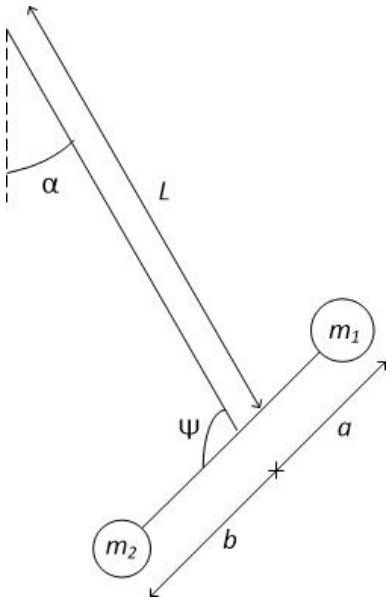


Figure 3.3: Diagram of the dumbbell system.

The next model investigated features a pendulum arm with a hinge at its end, about which a light, rigid rod pivots (see Figure 3.3). The rigid rod has two masses at either end, hence the name "dumbbell model" often used in the literature. The dumbbell is driven at its pivot which we expect to drive the pendulum itself. The motivation behind exploring such a model is that it loosely approximates the real system of a person in the seated position, the masses representing the legs and torso, and the oscillation about the pivot the flicking of a person's legs. The Lagrangian for this system can be written in terms of the generalised coordinates α and φ. The full derivation can be seen in appendix 3.15.

$$\begin{aligned}\mathcal{L} =&\frac{m_1}{2}\left[\dot{\alpha}^2 l^2 + a^2\left(\dot{\alpha}+\dot{\varphi}\right)^2 - 2\dot{\alpha}la\left(\dot{\alpha}+\dot{\varphi}\right)\cos\varphi\right]\\ &+\frac{m_2}{2}\left[\dot{\alpha}^2 l^2 + b^2\left(\dot{\alpha}+\dot{\varphi}\right)^2 + 2\dot{\alpha}lb\left(\dot{\alpha}+\dot{\varphi}\right)\cos\varphi\right]\\ &+gl(m_1+m_2)(\cos\alpha-1)+g(m_2b-m_1a)\cos(\alpha+\varphi)\end{aligned} \tag{3.14}$$

Applying the Euler-Lagrange equation, gives the following one dimensional equation of motion,

$$\begin{aligned}0 = (m_1+m_2)l^2\ddot{\alpha} + (m_1a^2+m_2b^2)\left(\ddot{\alpha}+\ddot{\varphi}\right) + l(m_2b-m_1a)\left(2\ddot{\alpha}+\ddot{\varphi}\right)\cos\varphi\\ + gl(m_1+m_2)\sin\alpha + g(m_2b-m_1a)\sin(\alpha+\varphi)\end{aligned} \tag{3.15}$$

By making the simplifications $m_1 = m_2$ and $a = b$, the equation of motion is drastically simplified,

$$I_1\ddot{\alpha} + I_2\ddot{\phi} + glm\sin\alpha = 0 \tag{3.16}$$

$$I_1 = 2m(a^2+l^2) \tag{3.17}$$

$$I_2 = 2ma^2 \tag{3.18}$$

where I_1 is the time-averaged moment of inertia about the pendulum pivot, and I_2 is the moment of inertia about the dumbbell pivot. In this form, the equation of motion elucidates how motion of the pendulum is generated by the driving of the dumbbell. The term $I_1\ddot{\alpha}$, is related to the torque on the pendulum pivot, and the term $I_2\ddot{\phi}$ is equal to the torque of the dumbbell about its own pivot. The

torque from the driving of the dumbbell causes a reaction torque on the pivot causing it to oscillate from rest. To make the system more realistic, a linear damping term was added to the equations of motion.

$$I_1\ddot{\alpha} + I_2\ddot{\phi} + mlg\sin\alpha = -\beta\dot{\alpha} \tag{3.19}$$

Simplifying,

$$\ddot{\alpha} + \ddot{\phi}\frac{a^2}{l^2+a^2} + w_0^2\sin\alpha + \frac{\beta}{2m\left(a^2+l^2\right)}\dot{\alpha} = 0 \tag{3.20}$$

$$\omega_0^2 = \frac{gl}{l^2+a^2} \tag{3.21}$$

$$\tag{3.22}$$

where ω_0 has been used in anticipation that this will be the system's natural frequency. The substitution $\beta/2m\left(a^2+l^2\right) = 2\zeta w_0$ was made to aid numerical analysis of the system,

$$\ddot{\alpha} + \ddot{\phi}\frac{a^2}{l^2+a^2} + w_0^2\sin\alpha + 2\zeta\omega_0\dot{\alpha} = 0 \tag{3.23}$$

$$\zeta = \frac{\beta}{4m}\sqrt{\frac{1}{gl\left(a^2+l^2\right)}} \tag{3.24}$$

When $\zeta = 1$ the system is critically damped, below this the system is under-damped. Next, the equations were converted to a two dimensional system of first order ordinary differential equations (ODEs) by making the substitution, $\dot{\alpha} = \Omega$,

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\alpha\\ \Omega\end{bmatrix} = -\begin{bmatrix}-\Omega\\ \ddot{\phi}\frac{a^2}{l^2+a^2} + w_0^2\sin\alpha + 2\zeta w_0\Omega\end{bmatrix} \tag{3.25}$$

A Python script was written that numerically solved the system in Equation 3.25 using Scipy's `integrate.odeint` package. The program script may be found in `dumbbell.py` (see Appendix E). The constants used in the program were $g = 9.81\,\mathrm{Nkg}^{-1}$, $a = 0.3\,\mathrm{m}$, $l = 1.0\,\mathrm{m}$, $m = 3.0\,\mathrm{kg}$. The driving function for $\varphi(t)$ was set with the form $\varphi(t) = A\sin(\omega t + \phi) + \frac{\pi}{2}$, where A is the amplitude of motion, and ϕ is a phase factor. This ensured motion was symmetric and oscillatory about the pendulum arm. Figure 3.4 shows that the pendulum responds to the driving by oscillating at the driving frequency, with an amplitude growing from zero, eventually coming to a steady state after a few periods. It can be seen that the pendulum is $\frac{\pi}{2}$ out of phase with the driving oscillations when $\omega = \omega_0$. Figure 3.5 shows a phase plot for the system, it can be seen that, the pumping pushes the pendulum to increasingly higher energy orbits. Initially, the amplitude grows very fast with each swing, but slowly tails off and sequential orbits become closer and closer together, until the system reaches its steady state.

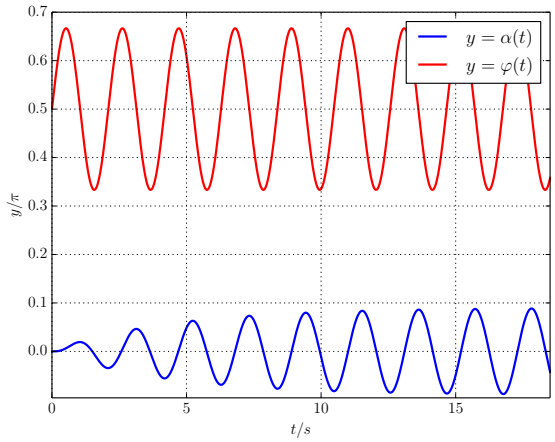


Figure 3.4: Time solution of the Equation 3.25, with initial conditions $\left(\alpha(t=0) = 0, \Omega(t=0) = 0\right)$, $\varphi(t) = \frac{\pi}{6}\sin(\omega t) + \frac{\pi}{2}$. The driving frequency is set to the natural frequency.

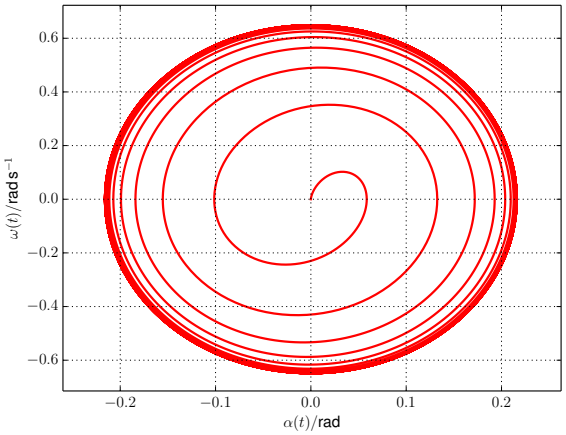


Figure 3.5: Phase plot of the system with $\zeta = 1$ driven at its natural frequency after 20 natural periods.

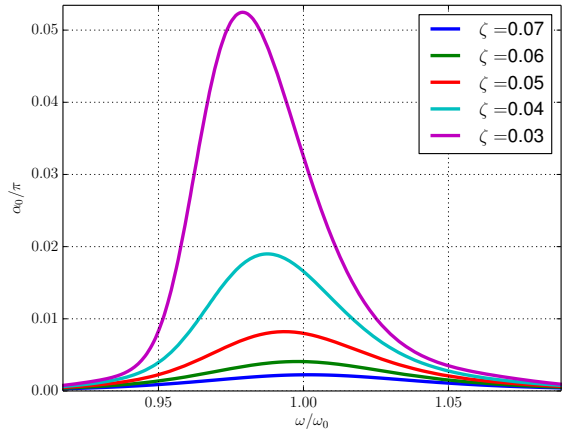


Figure 3.6: Steady state response of the dumbbell system for numerous low damping coefficients. α_0 denotes the root mean square amplitude of the pendulum angle during steady state in radians.

An extension was made to the program to produce resonance responses of the system. It does this by numerically solving the ordinary differential equations (ODEs) for some driving frequency, and then finding the root mean square value for the response in the steady state. It only averages the root mean square value for the response after many natural periods to ensure the system has come to a steady state. Figure 3.6 shows that the resonance is pushed away from the natural frequency as ζ decreases. Presumably, this is because lower ζ result in greater steady state amplitudes which causes the equation of motion to become non-linear (when amplitudes are small the $\sin\alpha$ term can be well approximated by α. Even so, the resonant frequencies are very close to the natural frequencies, and they give similar amplitudes. Analysis of this model suggests that a human can pump a swing by swinging their legs back and forth about the seat of the swing, and that for optimal swinging height, they should perform this motion close to the natural frequency of the combined person-swing system.

3.1.3 Flail

The following was contributed by: B. Stokes

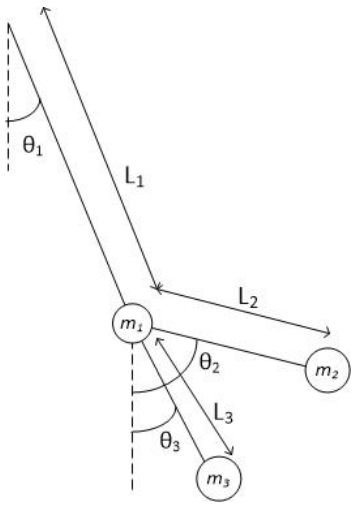


Figure 3.7: Diagram of the flail system.

The next system that was investigated is the one pictured in Figure 3.7. This "flail" system somewhat mimics the mass distribution of a person on a swing. The components of the system Lagrangian can be determined from those of the triple pendulum if the terms for the third mass are excluded and another set of terms of the same form as those for the second mass are added. Lagrangian of the system is quoted below, and the full derivation can be found in Appendix B.2. Note that the angles are all taken from the vertical and $M = m_1 + m_2 + m_3$.

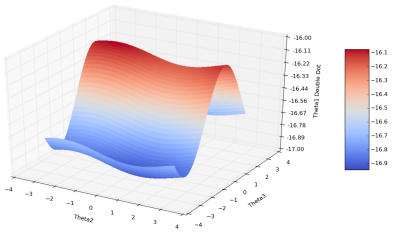
$$\begin{aligned}\mathcal{L} =& \frac{M}{2} l_1^2 \dot{\theta_1}^2 + \frac{1}{2} m_2 (l_2^2 \dot{\theta_2}^2 + 2 l_1 l_2 \dot{\theta_1} \dot{\theta_2} \cos(\theta_1 - \theta_2)) + \frac{1}{2} m_3 (l_3^2 \dot{\theta_3}^2 + 2 l_1 l_3 \dot{\theta_1} \dot{\theta_3} \cos(\theta_1 - \theta_3)) \\ &+ M g l_1 \cos\theta_1 + m_2 g l_2 \cos\theta_2 + m_3 g l_3 \cos\theta_3.\end{aligned} \tag{3.26}$$

The Euler-Lagrange equation for θ_1 was then applied. The result of the calculation is shown below:

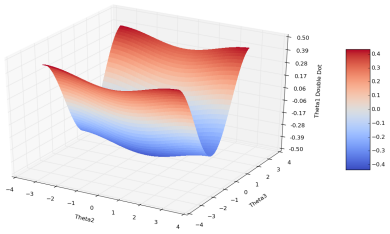
$$\ddot{\theta_1} = \frac{\begin{array}{c} m_2 l_2 \dot{\theta_2} \sin(\theta_1 - \theta_2)(\dot{\theta_1} - \dot{\theta_2}) - m_2 l_2 \ddot{\theta_2} \cos(\theta_1 - \theta_2) + m_3 l_3 \dot{\theta_3} \sin(\theta_1 - \theta_3)(\dot{\theta_1} - \dot{\theta_3}) \\ - m_3 l_3 \ddot{\theta_3} \cos(\theta_1 - \theta_3) - M g \sin(\theta_1) - m_2 l_2 \dot{\theta_1} \dot{\theta_2} \sin(\theta_1 - \theta_2) - m_3 l_3 \dot{\theta_1} \dot{\theta_3} \sin(\theta_1 - \theta_3) \end{array}}{M l_1} \tag{3.27}$$

It is interesting to note that if the two arms of the flail are held stationary ($\dot{\theta_1} = \dot{\theta_2} = \ddot{\theta_1} = \ddot{\theta_2} = 0$) then Equation 3.27 becomes the equation of motion for a simple pendulum of length l_1, from Equation 3.5.

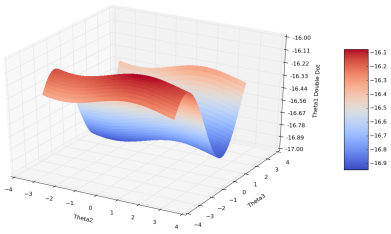
The equation of motion for $\ddot{\theta_1}$ can be graphed with several of the variables set to be constants to find the position of the arms that give the maximum acceleration to θ_1 in various circumstances. This is shown below.



(a) Just before movement from a peak.



(b) Motion at the bottom of the swing.



(c) Just before movement from the other peak.

Figure 3.8: Diagrams showing $\ddot{\theta_1}$ with constant accelerations and velocities at different parts of the motion.

3.1.4 Hinged Flail

The following was contributed by: C. Hogg

To create a more accurate model of the swing, the pivot points on the swing had to be taken into consideration. By treating the two pivots about the hand location as a single pivot, as the length between the two of them was much shorter than the other lengths, we produced a more detailed model. This model was similar to the flail model but with a pivot point between the top of the swing and the seat mass.

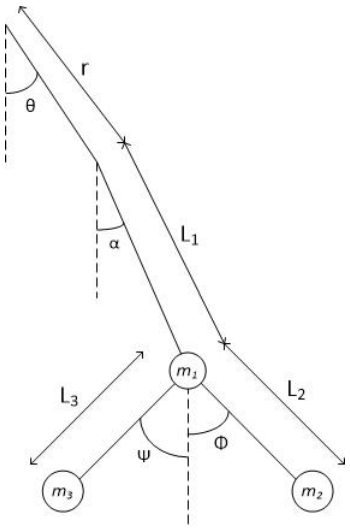


Figure 3.9: Diagram of the hinged flail system.

Producing the Lagrangian of this complex model was made easier by linearly summing the Lagrangian

of the individual masses. In this case the Lagrangian is equal to the sum of the Lagrangian of the seat mass, and the leg and torso masses, which are a double pendulum and two triple pendulums respectively.

$$\mathcal{L}_{Total} = \mathcal{L}_{Double_{Seat}} + \mathcal{L}_{Triple_{Torso}} + \mathcal{L}_{Triple_{Legs}} \tag{3.28}$$

All of these could be derived from the triple pendulum Equation 3.13. Where the $\mathcal{L}_{Double_{Seat}}$ takes the m_2 terms of the triple pendulum equation replacing m_2 with m_{Seat}, whilst $\mathcal{L}_{Triple_{Torso}}$ and $\mathcal{L}_{Triple_{Legs}}$ take the m_3 terms of the triple pendulum equation replacing m_3 with m_{Torso} and m_{Legs} respectively.

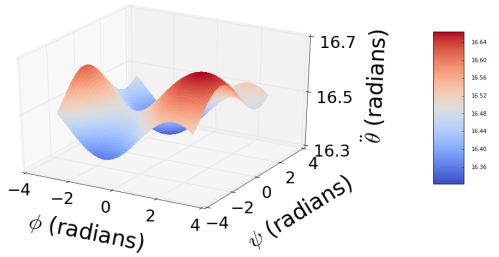
This gives us a final Lagrangian of,

$$\begin{aligned}\mathcal{L} =& \frac{1}{2}m_1(r^2\dot{\theta}^2 + l_1^2\dot{\alpha}^2 + 2rl_1\dot{\theta}\dot{\alpha}\cos(\theta-\alpha)) + m_2(\frac{1}{2}\dot{\phi}^2 l_2^2 + \frac{1}{2}\dot{\alpha}^2 l_1^2 + \frac{1}{2}\dot{\theta}^2 r^2 + \frac{1}{2}\dot{\theta}\dot{\alpha}rl_1\cos(\theta-\alpha) \\ &+ \frac{1}{2}\dot{\theta}\dot{\phi}rl_2\cos(\theta-\phi) + \frac{1}{2}\dot{\alpha}\dot{\phi}l_1l_2\cos(\alpha-\phi)) + m_3(\frac{1}{2}\dot{\psi}^2 l_3^2 + \frac{1}{2}\dot{\alpha}^2 l_1^2 \\ &+ \frac{1}{2}\dot{\theta}^2 r^2 + \frac{1}{2}\dot{\theta}\dot{\alpha}rl_1\cos(\theta-\alpha) + \frac{1}{2}\dot{\theta}\dot{\psi}rl_3\cos(\theta-\psi) + \frac{1}{2}\dot{\alpha}\dot{\psi}l_1l_3\cos(\alpha-\psi))\end{aligned} \tag{3.29}$$

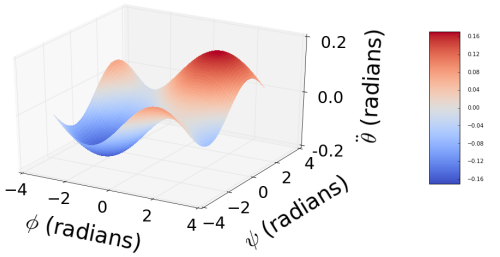
where m_1 is the mass of the seat and thighs of the robot, m_2 is the mass of the robots torso, head and arms and m_3 is the mass of the robots lower legs and feet. The length r is the distance between the swings pivot and the pivot point on the swing, l_1 is the distance between the pivot point of the swing to the seat, l_2 is the distance to the centre of mass of the robots upper body from the seat and l_3 is the distance to the centre of mass of the robots lower body from the seat. The full derivation for all the equations can be found in Appendix B.3.

Applying the Euler-Lagrange equation, we rearranged for the acceleration of the swing, so that we had an equation of motion to simulate.

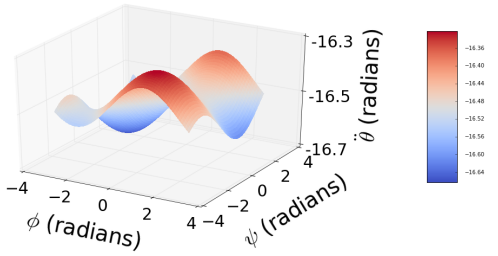
$$\begin{aligned}\ddot{\theta} = \frac{\begin{aligned}&m_1l_1(\dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha}) - \ddot{\alpha}\cos(\theta-\alpha)) + l_1m_2(\dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha}) - \ddot{\alpha}\cos(\theta-\alpha)) + \\ &l_2m_2(\dot{\phi}\sin(\theta-\phi)(\dot{\theta}-\dot{\phi}) - \ddot{\phi}\cos(\theta-\phi)) + l_1m_3(\dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha}) - \ddot{\alpha}\cos(\theta-\alpha)) + \\ &l_3m_3(\dot{\psi}\sin(\theta-\psi)(\dot{\theta}-\dot{\psi}) - \ddot{\psi}\cos(\theta-\psi)) + (m_1+m_2+m_3)l_1(\dot{\theta}\dot{\alpha}\sin(\theta-\alpha)) - \\ &\dot{\theta}\dot{\phi}l_2m_2\sin(\theta-\phi) - \dot{\theta}\dot{\psi}l_3m_3\sin(\theta-\psi) - (m_1+m_2+m_3)g\sin(\theta)\end{aligned}}{(m_1+m_2+m_3)r}\end{aligned} \tag{3.30}$$



(a) Just before movement from a peak.



(b) Motion at the bottom of the swing.



(c) Just before movement from the other peak.

Figure 3.10: Graphs to show the variation in the swings acceleration in different states.

These graphs were generated by this equation using the limbs travelling in the same direction. By comparing them to robot limb limitations, the optimal angles for the acceleration we want (positive in the first two graphs, negative in the third) associate with the swinging positions we expected, lying back whilst travelling towards positive theta whilst sitting up whilst travelling towards negative theta.

This final model enabled us to simulate the swing using different motions of the robot in an attempt to investigate the best possible motions. It also allowed us to investigate the swinging motion of the robot through simulation using the equation as a basis by attempting to optimise the appropriate acceleration during a swinging motion. We used this method to investigate how we could use the limb gyros to get the robot to swing using its own inputs and to see if there was a more efficient action other than the expected method of swinging. It also made if possible to investigate the importance of of different variables within the swinging action. For example the weight of the seat had to be lowered as much as possible, due to its beneficial impact on the acceleration of the swing was dependent on the smaller angle changes, whilst its hindering effect was linear.

3.2 Human Swinging

The following was contributed by: G. Sly

3.2.1 Theory

Human motion on a swing contains a combination of horizontal and vertical applications of force and torque to generate the swinging motion. The aims of this section are to understand if human motion is optimal and, if so, how to apply this to the NAO robot. An additional goal is to investigate if a better method of motion for swinging can be achieved by the NAO robot.

The human swinging motion discussed in this section has come from regarding the swing as a mass on the end of a light chain with a length that varies periodically [21]. Figure 3.11 shows a simulation of a pendulum attached to a drum, such that its length changes as the drum rotates. This simulation does not exhibit simple pendulum motion as the varying distance between the mass and the pivot produce

changes in amplitude and frequency. However, the simulation would more accurately represent the swinging motion of a human if the drum was further down the pendulum and acted more like a flywheel. This would help in representing how humans change the swing's length, as their hands are used low down on the swing rather than at the top near the pivot.

Alternatively, this could be treated as a 'slightly elaborate simple pendulum' as in [22]. These cases are very similar, taking theses cases into account on the swing proves that although it looks like a simple pendulum, it is varying in length once it reaches a certain height, and when humans pull on the chains it causes the amplitude and frequency to change, affecting the motion.

The next step is to consider the same pendulum but being driven by an external force from a human swinging. This is the case when the swing is being swung at twice the natural frequency of swing to generate greater amplitude much faster [23]. This is also demonstrated and discussed by [22], [21] and [24] saying the swinging motion should be treated as a combination of 'driven oscillation and parametric pumping'.

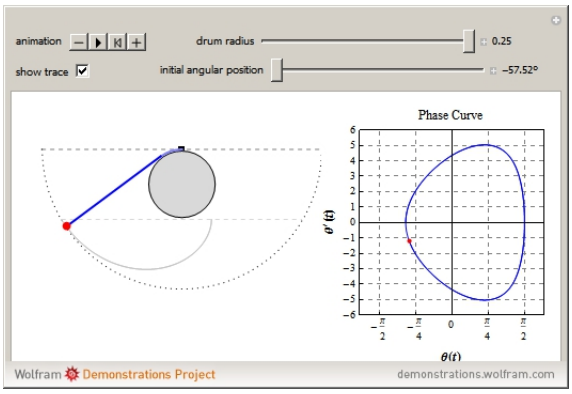


Figure 3.11: A Wolfram Alpha [25] example of a pendulum of varying length.

To swing, humans add energy to the system by leaning forwards and backwards to provide torque to the swing. There is a shifting of centre of mass that contributes to this motion by adding stability and gravitational potential energy. The swing already has kinetic and potential energy when starting from displacement. The theory suggests that in order to add energy to the system, a human must apply torque to the swing. They shift their body above and below the seat to generate kinetic energy, generating force that is greater than the drag opposing it. [26] states that the chain provides the reverse torque to the human through the line of impulse, due to the pivot a human creates with their hands. This pulls you towards the bar or support of the swing, resulting in little to no rotational or sideways movement being applied. Most of the energy is applied directly to the motion of the pendulum. Therefore, the swing will be treated as a pendulum of varying length, which is caused by a human pulling on the chain.

This theory also suggests that the motion is simultaneous and immediate when changing positions in order to create the swinging motion and torque at the extremes of the pendulum. This theory will be investigated, to see if both the legs and upper body generate torque, but with the mass of the legs being considerably less, it needs to be proven that the legs can generate torque. Considering these aspects of the movement means motion capture was chosen to be the best way to provide evidence of which theories are correct, and then use this data to generate movement functions to apply to the robot.

A second theory that will be tested is one of the motion being two separate movements that change almost immediately to shift centre of mass, rather than to generate torque [27]. This theory implies the changing of position is purely to stabilise the system. Filming the human motion and analysing the data through motion capture will be used to determine which of these two theories is more accurate, and decide what is more influential in swinging, torque or shifting the centre of mass. This will further our understanding of human motion and help our knowledge of how best to apply this to the robot.

This section will go on to cover the best way to test these theories via motion capture. By the end of these sections a greater understanding of what it takes to swing and what the optimum motion should be for seated swinging motion will have been found.

3.2.2 Method

The following was contributed by: J. Forster

The approach previously used [2] was to record a human performing the swing motion and process the video files through a motion tracking software. This year an alternative software was used to reduce the limitations; such as the maximum of three tracking points and the difficulty in distinguishing different colours (especially important when trying to track markers).

The motion tracking software used in this study was Kinovea [28], a free open source software which was chosen for its intuitive user interface and simple but powerful tracking capabilities. Improvements on the Tracker software used from last years project are that you can track many points simultaneously, compare two videos side by side, measure angles and most importantly export all the data to various differing file types.

Other improvements that were implemented with respect to last years project was to film in an open space free from clutter so that the software had an easier time of tracking the motion and less opportunity to be confused by changes in the colour tone of the backdrop. The various swinging motions were filmed multiple times to acquire averages and reduce error, multiple people were filmed to see how the human swing motion changes from person to person.

As previously mentioned the role of human motion in last years report was a low priority, this year however the project has progressed considerably, and now it seems that studying the human motion has given useful data and findings that were used in Section 4.4 to the end goal of having the robot emulate a human swinging.

Kinovea is the free open source video tracking software used to capture and analyse human motion on a swing during the group project. The interface is intuitive and user friendly and the software is of the highest standard available for free in the current market. Kinovea tracks points selected via the contrast in the colouring and as such it was important that our markers be clear and readable, physical white markers were placed on the shoulder, elbow, knee and foot with the swing seat also being marked for tracking.

Figure 3.12: Screenshot of Kinovea tracking a video file.

When filming, the camera was mounted and kept in the same position for each recording ensuring that the full range of motion was in view and that the recordings were comparable with each other.The optimum distance from the swing for the camera location was 360±10cm perpendicular to the swing. This distance produced the best compromise on the reduction of tracking error and minimising parallax error, while still capturing the whole range of motion. The camera used for filming was a Sony Xperia Z2 camera phone due to the fact that it provided the best frame rate and largest field of view of the cameras available to use (see Appendix C.1 for more details of the camera).

Parallax was an issue to be considered, the camera was placed parallel to the motion and central, in an ideal world multiple cameras as close as possible, which would have been used from various angles in order to minimise any parallax errors but access to capable cameras was restricted and the processing

Figure 3.13: Demonstration of swinging from a starting displacement.

(a) Demonstration of legs only swinging motion.

(b) Demonstration of upper body only swinging motion.

Figure 3.14: Two swinging types that were filmed.

of the data from multiple camera angles would have been too time-consuming for the respective gains in accuracy.

The video files recorded were imported and then the markers were assigned a tracking point. The software tracks the motion automatically but should the computer be unable to keep up you can go frame by frame and correct any irregularities in the tracking line manually. At the peak of the swing or when a marker passed too close to another marker the tracking point could sometimes follow the wrong path and being able to move the tracker point at each frame is a big advantage to the Kinovea software. The length of the swing seat was used to calibrate the software so that useful X and Y position data could be exported for later analysis. Kinovea exports to many different file types but it was decided that an excel spreadsheet should be used such that velocities and accelerations could be calculated using the position data with ease.

Another useful feature of Kinovea is the ability to measure angles, unfortunately the software is unable to track these angles as they change and as such the angle readings had to be taken from the footage frame by frame. Whilst this process may be slightly time consuming at least the capability is there.

———————————The following was contributed by: G. Sly ———————————

One of the movements filmed was normal swinging motion from a starting displacement of 95±1cm in X and 25±1cm in Y from equilibrium as shown in Figure 3.13. These values were calculated in Kinovea after the calibration data had been input. The data was generated by drawing lines from equilibrium during motion. The starting position was kept consistent by placing a pole in line with the first starting position. This start position was chosen because it allowed for the motion to be seen easily. Motion with an initial displacement was also filmed with purely leg motion and filmed where the legs were kept stationary and the body and arms were moved. Both of these motions are shown in Figure 3.14.

The leg only motion and torso only motion were chosen because the legs and the shoulders/arms are

theorised as being used to generate torque for the body from below and above the centre of mass and are used to initiate the swing movement via torque [26]. In order to verify this theory, the two motions were filmed in isolation to see how much they were contributing to the motion and how much of the motion was purely about shifting centre of mass and stability.

Another motion filmed was from rest position, to see how the human motion changes when starting from rest compared to from displacement. Examining starting with no initial displacement is useful when testing human motion because it tests whether human motion is capable of inputting significant energy to the system, as well as maintaining the swinging motion.

Filming multiple people enabled us to identify reasons for people having different swinging variations and how this changes the system and the motion, and which variation is best. Alongside this, these data sets can back up the findings from the initial persons motion.

Kinovea was able to successfully obtain the X and Y coordinates of each of the tracked points every 0.03±0.01s. Kinovea's high sample rate resulted in a lot of noise. To counteract this, MATLAB was used to smooth the data, which meant the sample rate changed from 0.03s to 0.1s which resulted in much more easily interpretable graphs such as Figure 3.15.

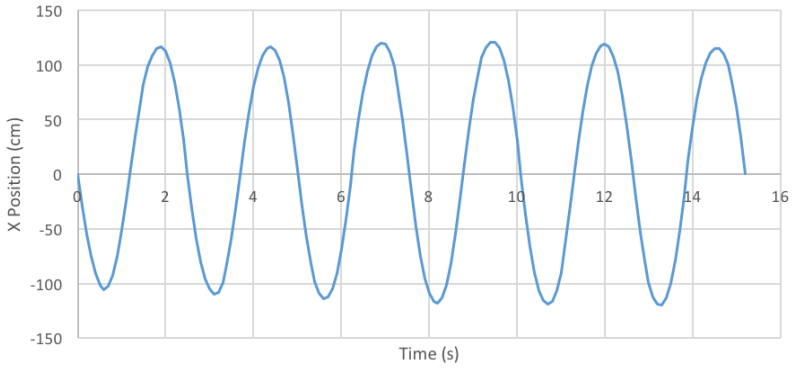


Figure 3.15: Graph of normal seated motion of X position against time.

The data was used was to obtain graphs for X and Y components of position, velocity and acceleration for the swing, shoulder and ankle. From this the graphs were compared and contrasted to allow for conclusions to be drawn about human motion and further the understanding of human swinging motion.

This overall method, and the process, was the same for both other types of motion investigated in Section 3.3 and Section 3.4.

3.2.3 Results

The main observation from filming was that for the normal swinging motion the legs seem to slightly precede the extremes of the swing, and the rest of the body changes positions at the extremes. This is due to legs having to move further distance than the torso, and the lack of mass compared to the torso causes the leg movement to seem to be quicker than torso.

Another observation was that moving only the legs was not able to sustain amplitude, this is likely due to friction and drag being greater than force and torque produced by the legs. This is hypothesised to be due to the legs having much less mass than that of the torso meaning it can generate muchf less force and torque. This suggests that legs motion is used to allow the body to lower its centre of mass for stability, rather than for torque generation. This was backed up by the body only motion which seemed to almost achieve the same amplitude as the normal swing, suggesting the force and torque generation is brought mostly by the upper body. However, the stability was reduced, as there was more sideways motion and the swing tilted more, thus suggesting the legs are aiding in stability, and shifting the centre of mass to do so. From rest it seemed that the humans rocked to generate some motion into the system horizontally, but also used an exaggerated motion of the displacement motion. This was then used to apply vertical motion via the pumping of the swing from our extension and flexion of the body. Once a sufficient height was reached movement seemed to be identical to from displacement motion.

There was a slight difference in movements between people, one person seemed to focus on a gradual approach to switching positions, this appeared to generate slightly less amplitude and distance compared to the initial motions we had filmed. This suggests that the changes need to be as immediate and simultaneous as possible. Another person seemed to not extend as far with their torso when lowering but seemed to have a closer to vertical position, when pulling up. This appeared to result in an even lower amplitude achieved. This suggests that human motion is generated mostly by the extension of flattening the torso to as low a position as possible. The change from sat upright to almost lying flat is clearly vital to generating the maximum amplitude as fast as possible. This could however also be due to the length of the legs and torso being longer for the initial person being filmed. Allowing for more torque to be generated rather than ability to shift centre of mass.

These observations were then compared with the actual results to deem whether the observations are true or not and conclude what human motion is and why. Alongside this, graphs from the collected data were compared between each of the motions filmed in Section 3.2.2. As this allowed for conclusions to be drawn from these comparisons on how human seated motion works and why humans have chosen it.

The first comparison of results was between normal motion from displacement and legs only motion from displacement. What was found was that legs only motion, decayed in amplitude for each of the graphs for position, velocity and acceleration, over time. Never returning to its initial displacement, as Figure 3.16 shows an example of. Whereas normal motion actually increased in amplitude in each of the graphs. This suggests that the legs are not capable of contributing any significant amount of torque to aid in the motion of swinging the human body. This is most likely due to the lack of mass in the legs compared to the rest of the body, as discussed earlier. Therefore, the legs are believed to be vital in the motion for lowering centre of mass of the body to help apply greater potential energy and also to stabilise the system so that all the energy is being applied directly to the forwards and backwards motions and minimising any sideways motion.

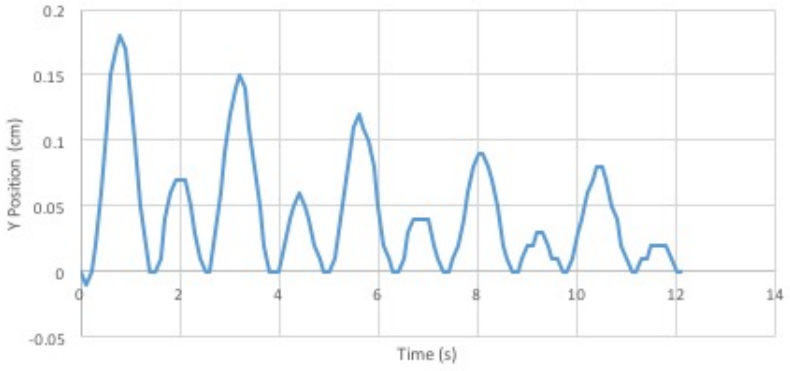


Figure 3.16: Example of legs only motion decay in Y position against time.

However, when comparing normal motion from displacement and upper body only from displacement, contrasting evidence was shown to legs only. The observations for upper body only suggested this motion managed to achieve a similar amplitude if not slightly higher than normal seated motion. Yet when this motion is compared via graphs with the normal motion, it is shown that whilst the observation is correct for heading forwards (from left to right whilst extending the body), on the way back there is a significant difference in amplitude between the two motions. As seen when comparing the troughs in Figures 3.15 and 3.17. This suggests a contrary theory to legs only motion, in that although the legs are vital for shifting centre of mass and stability, they also contribute torque significantly when the body is upright. This is due to the quick nature of the movement of the legs returning from extension to flexion with the aid of gravity. Therefore, it can be concluded that the upper body is used to generate the majority of the torque on the way forwards, and the legs are extended to add slight torque but moreover to allow for an even greater shifting of centre of mass. Whereas the roles are reversed for the return section, with the exception that the body is contributing more torque than the legs do on the way forwards. In combination with this, gravitational potential energy is more vital at the extreme where backwards motion starts then at the other for aiding in applying energy to the swing.

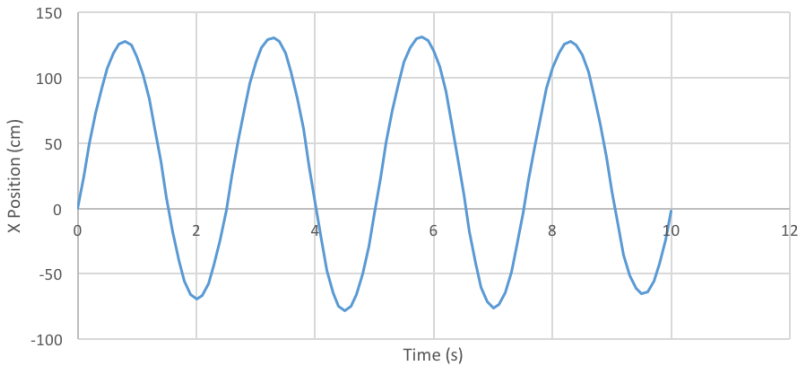


Figure 3.17: Graph of upper body only X position against time.

The observations from rest seated motion seem to be correct, as the patterns of the graphs were almost identical between rest and from displacement. The only difference was that rest grew more in amplitude relative to starting amplitude, but took longer to reach the same amplitudes that displacement motion did, as expected. So what we can conclude from this is that both from rest and from displacement follow the same movement pattern. The main difference is that from rest, the motion has to be more exaggerated, (more extension and flexion of the body and legs) as when at rest there is the minimal amount of energy possible in the system, as potential energy is at it's minimum and there is no kinetic energy present. So the more energy that can be put into the horizontal motion which then in turn can be translated into vertical motion, will result in the start of the periodic motion seen with human motion from displacement. Once there is enough energy produced by the exaggerated motion to achieve enough displacement, the exaggerated motion will no longer be required and disappears, we hypothesise for energy efficiency. So overall there is not any difference in the pattern of the motion just the extent that is required of the motion from rest differs to displacement, as expected.

3.2.4 Discussion

These comparisons when combined evidence that upper body is mainly responsible for generating the forwards motion, and the legs are mainly responsible for generating the backwards motion along with maximising the potential energy at the forward extreme. Alongside this, centre of mass shifting seems to play a larger part than torque in driving the motion hence the change of positions needs to be as instantaneous and simultaneous between the body parts as possible. This has furthered our knowledge of why and how humans swing when seated, the next task is to fit this to a function and apply to the robot to investigate whether it can emulate this motion.

3.2.5 Data Analysis

The best data set for each motion was selected and fitted to obtain functions to input to the robot. A function was also needed for simulation in Webots as otherwise there would be no indication that the function would work correctly on the NAO robot.

This process was started by fitting the X and Y positions with respect to time. These functions are shown by equation 3.31, and Figures 3.18 and 3.19,

$$f(t) = A\sin(\omega t + \phi_1) + B\sin(\frac{1}{2}\omega t + \phi_2) + \gamma \tag{3.31}$$

Where γ is an offset, A, B, ϕ_1, ϕ_2 and ω are constants to be found from the fitting.

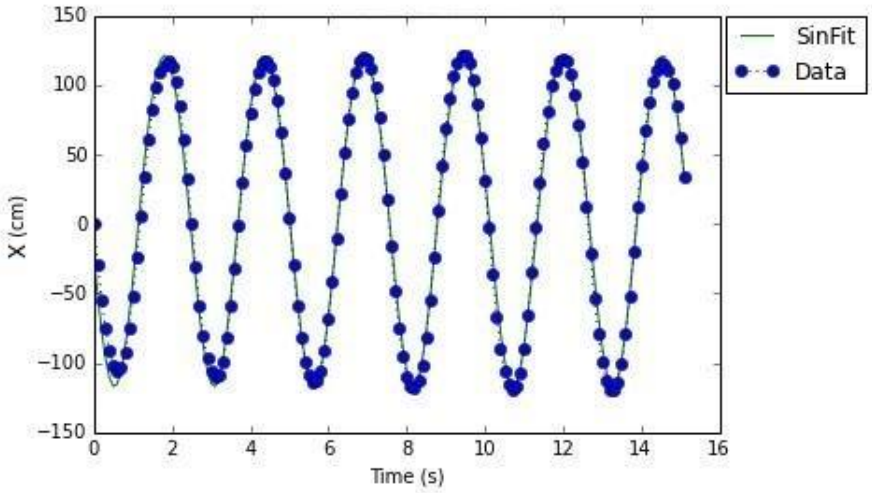


Figure 3.18: Plot of fitted X position function against time.

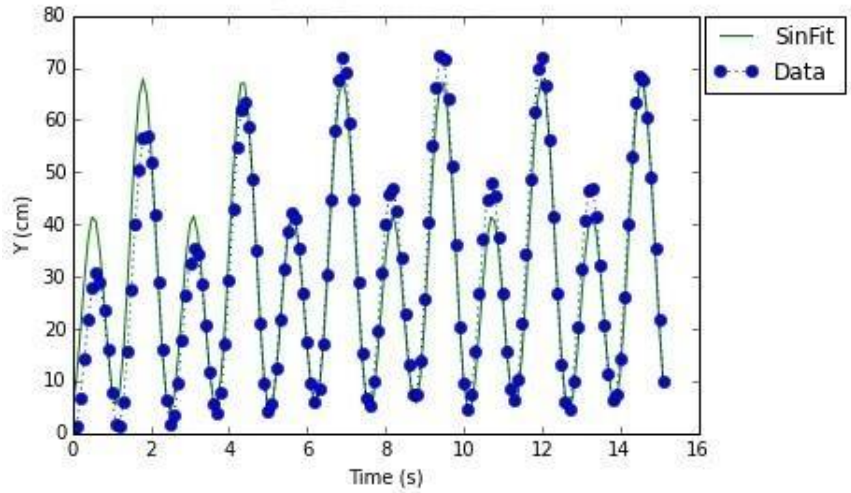


Figure 3.19: Plot of fitted Y position function against time.

These are the functions for human motion with regards to X and Y position with respect to time, which were applied to each of the tracked points. The reason these functions were chosen is it allowed us to have multiple approaches to translating this to the NAO robot. The most successful method was to use these very same functions to obtain a function for the angle of the torso relative to the swing and the same for the knee angle. This was done by determining the position of the shoulder, swing and the lower leg from theses functions and applying it every 0.01s. From this the angle of the swing from the pivot was calculated by using the equilibrium point of the swing. Then using the position we had calculated from the function and applying the difference the angle was determined with some simple trigonometry. From here the distance was found from the swing to the shoulder, and swing to lower leg at every 0.01s using some simple trigonometry rules. Alongside this, the thigh and ankle were assumed to be the same length, due to some rough measurements with a ruler. The knee angle relative to the swing and torso angle relative to the swing were then calculated. This then meant we now had data for the angles of the knee and shoulder each relative to the angle of the swing to the pivot which then meant a function could be fitted that would remove the factor of time. This is demonstrated below by Figure 3.20. This allows for a function to be obtained, for where the knee and shoulder angles should be, as a percentage of the

angle of the swing relative to the pivot. This function can be seen in Equation 3.32,

$$\alpha = \frac{A\sin(\Omega t + \psi_1) + B\sin(\frac{1}{2}\Omega t + \psi_2) + \delta_1}{C\sin(\Omega t + \psi_3) + D\sin(\frac{1}{2}\Omega t + \psi_4) + \delta_2}\phi \tag{3.32}$$

where α is the knee angle, ϕ is the pivot angle, δ_1, δ_2, A, B , C, D, ψ_1, ψ_2 ψ_3, ψ_4 and Ω are constants to be found from the fitting.

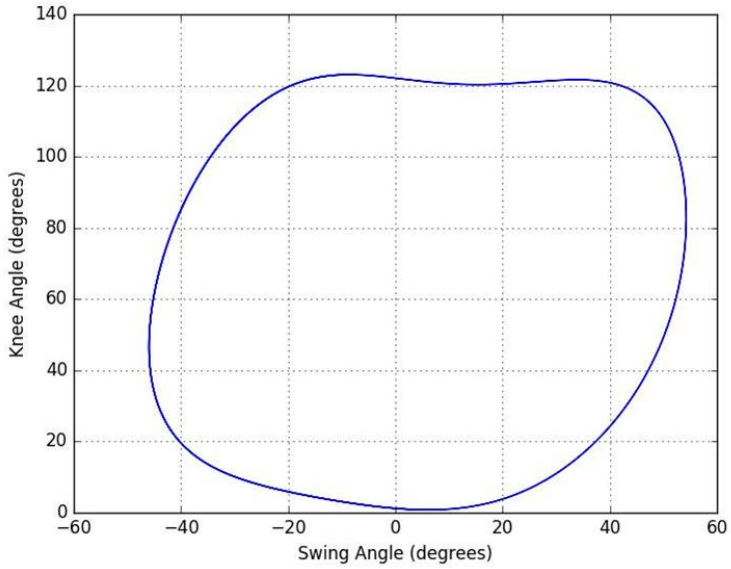


Figure 3.20: Knee angle Generated from function against pivot angle.

This function can now be applicable to any swing, as it is a percentage and not a value, so does not matter what the period of the swing is or what amplitude the swing is achieving. Therefore, this function is the ideal and purest function we could obtain for human motion with the data, and means it could be used for any robot on any swing. This function was then ran in Webots and tested on the robot which will be discussed in more detail in Sections 4.4.1 and 4.4.2 respectively.

3.3 Swinging Motion with Added Weight

3.3.1 Theory and Method

The theory of the added weights case is the same as in Section 3.2 as all adding the weights does is alter the mass distribution of the body, this shouldn't alter the movement pattern. The methods used were the same as normal seated swinging motion, just with an added weight on the legs.

The motions filmed were normal swing motion from displacement, and from rest with added weight to the legs. Alongside this the same setup was used for legs only with the extra weight, which was 8.5±0.5kg due to the accuracy of the scale we measured it on. This weight was chosen due to estimations from [29] and [30] about the legs approximate weight. It was decided to land in the middle at around fifteen percent of body weight. It was chosen that the average weight of a human was to be 62kg due to recommendations of [31]. So applying this weight doubled the weight on the legs roughly, so from this it can be seen to be an appropriate weight, as can then see if it doubled the amplitude etc, and if it changed the motion pattern. From this it can then be seen if the normal human motion pattern and the torque generation is purely down to mass distribution, or that the legs are part of the motion purely for just stabilising the system. Also this should indicate how vital controlling the centre of mass is for the motion, if the centre of mass does indeed shift. This might also help explain the slight variations in different peoples swinging techniques due to different mass distribution and having to shift their centre of mass in a different way to others.

3.3.2 Results

When the weights were added it seemed to increase the amplitude of the motion. However, due to footage being taken over a range of days, it was not always possible to compare the maximum amplitudes. This is due to the limitations mentioned in Section 3.2.2. This was found to be same for rest, however one extra difference noticed was that the motion was not as exaggerated as normal motion was. The major difference seemed to be in legs only motion, where the amplitude was sustained if not slightly improved. So this does indeed suggest there is a significant increase in ability to create force and torque on the torso and swing from the legs when having more weight on your legs, as expected.

The normal motion from displacement comparison of results of with and without weight found that there was no difference in amplitude achieved in X or Y when moving forwards. Which was not expected as the legs being able to generate more torque was expected to amplify the motion. The real difference was noticed when the swing was moving backwards, (where body had moved from extended to flexed) the peak in Y managed to achieve a greater height than that of normal motion. This can be seen when comparing Figure 3.21 and 3.22. As the motion went on, this backwards peak reached roughly the same height as that of the forward peak. These results are completely different to that of normal motion with no extra weight, where there is a clear difference in amplitude of the two peaks. Suggesting that for a more even pendulum motion where the two extremes are similar amplitude more weight is required on the legs. Indicating that the legs are the vital component in the backwards motion of the swing, especially to torque generation.

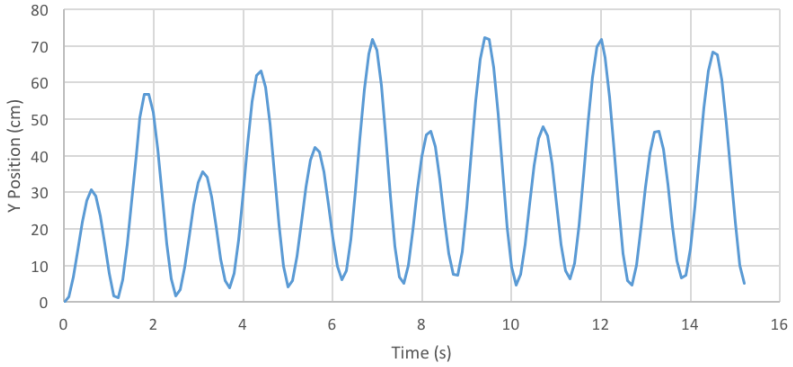


Figure 3.21: Graph of normal seated motion Y position against time.

Results from rest demonstrate another unexpected finding. When weights were added to the legs it took the system a lot longer to reach the same amplitudes then without weights did. This is against what was expected, as the added weight should lead to greater generation of torque and therefore more motion input into the system. However, this was not the case suggesting torque may not be that influential in human seated motion. Shifting centre of mass is more vital, as the less exaggerated motion of with weights from rest resulted in less amplitude then that of no weights. Proving shifting of centre of mass is the most vital aspect to human motion. However, these results may be down to the human not having enough power to move the legs as easily and quickly due to the extra weight. Hence the torque generation is not as great, but overall the extra mass should at least generate enough torque to result in a similar motion to normal without weight, which was not found, hence these conclusions above have been drawn.

The last comparison was that of legs only, once again similar results were found. The legs only motion decayed forwards in a similar fashion to that of legs only with no weight. Yet, the backwards motion for added weights lost much less amplitude in its extreme. Although the motion decayed, it decayed much less and was higher than the amplitude of that achieved on the forward motion in both X and Y by the end of the data set. Once again showing that legs are vital in the backwards motion but not so much in the forwards motion.

3.3.3 Discussion

So overall these three comparisons have suggested that legs are vital to the backwards motion but have little to no effect on the forwards motion. The shifting of centre of mass seems to be more vital then torque generation, as if torque was more vital, the legs would have effected forwards motion also but there was little to no effect found through the three comparisons. Hence it is believed that centre of mass shifting is crucial to the human seated motion more so than ability to generate torque.

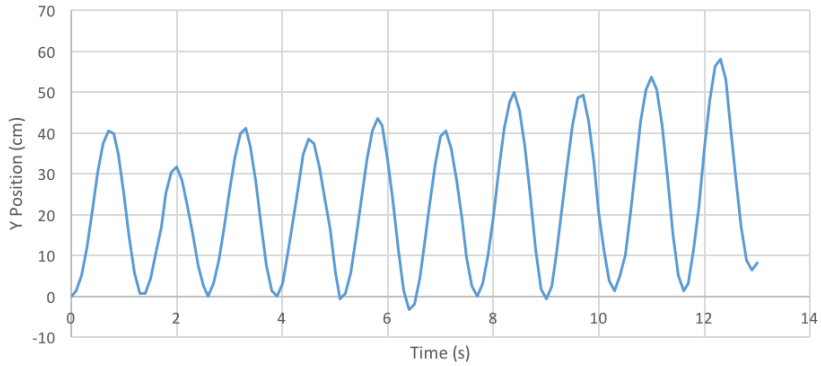


Figure 3.22: Graph of normal motion with weights added Y position against time.

3.4 Parametric Resonance Standing Motion

3.4.1 Theory and Method

One final theory to test is parametric resonance [32] and [23], and is based fundamentally on the principles discussed previously. This theory suggests that the optimum motion for swinging in the quickest time is to swing at double the natural frequency of the swing. For a human this is achieved by swinging from seated with the usual motion. However, if human standing motion is considered, this motion achieves parametric resonance by squatting at both extreme positions and standing at equilibrium position as shown by Figure 3.23. This would be useful to test to see if this theory in practice would result in a better motion and can then see which motion is the optimum. The standing motion is suggested to be parametric resonance by articles such as [33]. So if we were to find standing is better, this would lead to the question of why humans haven't adopted this motion of standing instead than sitting. This would help further understanding of human motion and also whether the robot could possibly emulate parametric resonance. From this, evidence will be provided which will help in choosing which motion for the NAO robot to use, and whether this would this be better for it than normal human seated motion.

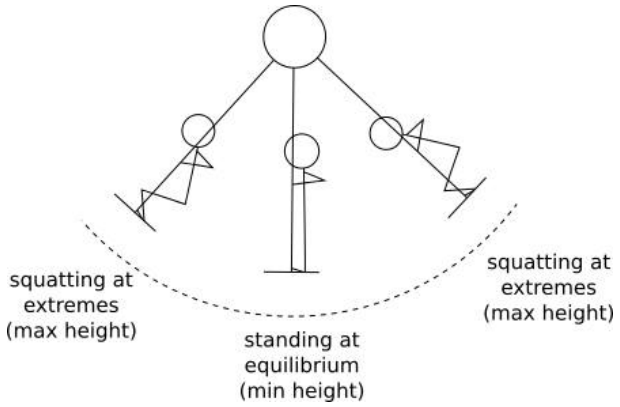


Figure 3.23: Demonstration of generating parametric resonance whilst standing on a swing.

Once again the method for this motion was very similar to that of normal seated swinging motion as mentioned in Section 3.2. The motion filmed was parametric resonance from standing motion as demonstrated by Figure 3.23.

3.4.2 Results

Parametric resonance standing motion seemed to generate a greater amount of amplitude than seated motion very quickly, as at one point, the swing managed to achieve a height greater than the pivot of the swing. The heights achieved were much quicker than that of seated motion also. So if these observations are correct this certainly backs up the theory that standing motion is more effective than seated motion and should be considered for the robot. However, this needs to be verified via the motion capture data.

When comparing the graphs from standing motion and seated motion there was an interesting discovery made. The standing motion was found to generate a much larger amplitude than that of normal seated swinging motion, as theorised and observed. However, what was not expected was how much greater amplitude it would obtain and how quickly these amplitudes would be achieved. Starting from around the same displacement as normal seated motion the double pump managed to achieve magnitude in X of around 1.5 times that of normal seated motion in the same time frame. In Y it managed to achieve practically double the amplitude of the seated motion, such as Figure 3.24 shows. This was the case until a point where the swing went over double the amplitude and actually managed to reach a height that was greater than the pivot of the swing, at this point the motion was stopped due to becoming dangerous. From these findings it can clearly be seen that parametric resonance from standing is a much quicker and better motion for generating the most amplitude, which means this may be more suitable for the robot to use than seated human motion, due to the ability to generate motion much quicker. Alongside this it should be much easier for the robot to control as all it need to do is flex and extend its legs resulting in a much easier to maintain motion.

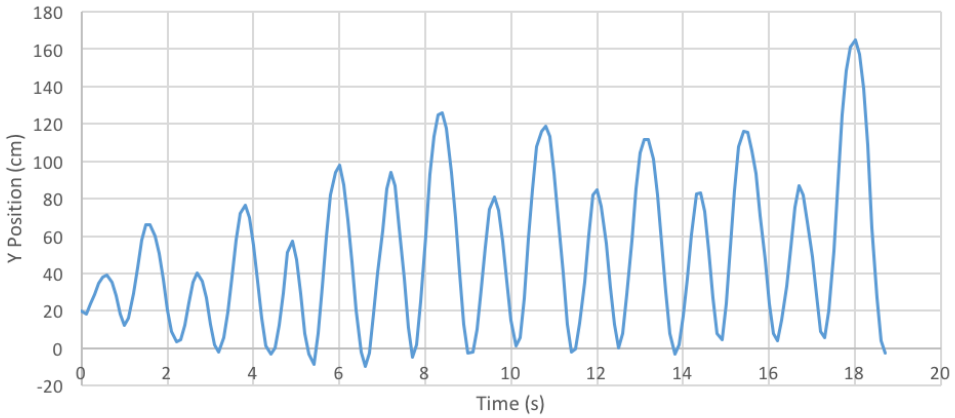


Figure 3.24: Graph of parametric resonance motion Y position against time.

3.4.3 Discussion

Despite these findings suggesting standing parametric motion is the better motion, it would be helpful to know why it is that humans have not chosen this motion. One hypothesis is that the motion is too energy demanding and unstable, due to the fact that when swinging with parametric resonance motion the tester felt their centre of mass was never truly stable. This was due to the quick nature of the motion, causing a lot more effort to keep the body stable within the motion. Alongside this observation the tester felt they had no choice but to continue the motion as it was taking so much effort to keep themselves stable that they had no way of easily exiting with this fast motion. So alongside it taking a lot of energy the motion feels very unsafe to the person swinging. This provides the hypothesis that standing parametric resonance is just too energy taxing and unsafe, that humans have felt the rewards of the motion were not worth the risks and effort required to perform it. Instead humans opt for a motion that still generates good amplitude, it just takes a little more time to reach these same amplitudes that

standing parametric resonance does. However, the robot should not have as many limitations, as it will not be aware of the danger of the motion and the stabilisation should not be too difficult for it, due to its lower mass and dimensions. The only real limitation that can be seen would be how to secure the robot so that it can perform this motion, as the motion requires a very strong grip on the ropes to perform the motion but also to allow for the centre of mass to be as stable as possible. Yet the NAO robot has been proven to not have a strong grip at all and has to be strapped to the swing for both its arms and feet, as it is incapable of holding its weight for long periods of time. This may cause a difficulty in getting it to emulate parametric resonance, but if this part was solved then it would certainly be a recommended motion for the robot to use to achieve the greatest amplitude in the shortest time.

3.5 Human Motion Conclusions

3.5.1 Conclusions from Movement Comparisons

From all these comparisons we can now more conclusively say why humans swing the way they do. Human motion is a combination of manipulating the shift of centre of mass and generation of torque and use of the chain providing reverse torque to help stabilise the system. Each part of the motion is responsible for vital generation of torque at different places within the motion of the swing, but even more vital is allowing the body to shift the centre of mass and maximising potential energy. This balance is struck by splitting the motion into the two positions where on the way through the body is extended to generate as much torque relative to the swing as possible. But on the way back, the body cannot generate that much torque in line with the motion, (other than in the legs) and due to low mass, this can't generate much torque. The goal instead is to raise the centre of mass as much as possible by sitting upright, which in turn maximises the potential energy at the extreme which can then be translated to kinetic energy to provide the greatest amount of motion throughout the period possible. Proof of this, and the legs torque contribution was emphasised by the findings from the added weights tests where the only improvements were in the backwards motion.

Humans have chosen this motion as it has optimised the amount of torque produced whilst shifting the centre of mass of the body as much as possible so the potential energy can be maximised and manipulated. This demonstrates that shifting centre of mass is a very key aspect as this is essentially what is done in standing parametric resonance, which was shown to generate a much greater motion. These findings suggest that flexibility may be more vital than mass distribution and torque generation in achieving a greater swinging motion, as flexibility allows for a greater potential shifting of centre of mass by increasing the possible range of motion of humans.

To conclude, human motion managed to achieve both of the goals set out at the start as there is now a hypothesis on why human motion is the way it is, based on the evidence from motion capture, and what motion is best to apply to the NAO robot. Alongside this a function has been obtained that can be applied to the NAO robot that was successful in simulations. This function can be used alongside the encoder to get the robot to attempt a human swinging motion, and can be used by any robot on any swing, which was the main goal of the human motion aspect of the project.

3.5.2 Outlook

Certain areas of human motion could be improved to further the approach and provide even more conclusive results that hopefully will follow along the lines of what was found. Certain areas that would be possible to look into would firstly be effort functionals, as this would help provide evidence to back up the hypothesis that human seated motion is maximising energy efficiency and therefore minimising effort. A second area to look into would be a greater more in depth look into parametric resonance. Investigating into generating a function for parametric resonance motion to simulate in Webots. Eventually applying this to the robot in the same nature as the seated swing was by not only tracking the swing but the legs, knee, hip and shoulder to see how the positions and angles change so they can generate a function without time present. Finally, the use of 3D motion tracking would allow for an even greater accuracy of tracking of each point and therefore better data and functions. This also would allow for a much more obtainable elbow angle which can then be applied to the seated motion also, making it a true representation of seated motion which could then be applied to the robot for optimum effect with the seated motion.

3.6 Chapter Conclusion

The following was contributed by: B. Stokes

This section has used various methods to investigate the mechanics of a person driving a swing. Several progressively more complex analytical models were used in the analysis. Firstly, the triple pendulum model was used to establish the nature of the terms used in the Lagrangian equations. From this it was easy to derive the terms of future models. Next, a dumbbell model was analysed to understand how the rotation of a person's limbs can provide torque to the swing. The slightly more advanced flail model was then analysed as the limbs of the flail provide a more accurate approximation to the swing scenario than the dumbbell. Finally, the previous models were used as stepping stones stones to the last, most complex model intended to approximate a person on a swing. This "hinged flail" model seems to be a good approximation of the mass distribution and degrees of freedom of a swinging human. Graphing this model produced the expected results for the positions of the limbs to give maximum acceleration to the swing on each side (ie. positions analogous to the ones a human would take at these times).

The software Kinovea [28] was used to analyse the movements of a person using a real swing. Numerous observations were made about the timing and purpose of these motions. It was found that a sitting human cannot drive a swing using their legs alone, presumably as a result of friction in the system combined with the comparatively low mass of a humans legs. This means that the torso movement is by far the greatest contributor to increasing the swing amplitude in this type of motion. It was proposed that the movement of a persons legs whilst swinging is mostly to stabilise the motion and reduce lateral movement and this was backed up by the motion observed when using only the upper body to swing. We also experimented with adding additional weight to the legs and this increased the persons ability to add amplitude to the swing. It was also observed that the motions performed on a swing must be as immediate and simultaneous as possible to generate a large amplitude quickly.

Chapter 4

Swinging from Predetermined Functions

4.1 Introduction

The following was contributed by: D. Butters

A simple method to program the NAO robot to swing was to define a set of motions that can be carried out in a predetermined way. Prior knowledge of the system, such as the natural frequency of the swing, dimensions of the robot etc. can be used to develop a set of actions that will make up a swinging motion. A predetermined model may use inputs from the system to trigger the robot to move, however, the conditions for motion and motions themselves are all determined based on systems examined previously. A predetermined model does not have the potential to modify its behaviour based on the conditions of the system as the predetermined model is hard-coded into the robot.

Three predetermined models are discussed in this chapter: the Driven model, the Analytical model and the Human Motion model. The Driven model uses knowledge of the swing apparatus to derive its motion. The Analytical model uses mathematical analysis of the system modelled as various pendulums to derive an equation of motion for the system. The Human Motion model replicates recorded motions of a human on a rope swing on the robot.

4.2 The Driven Model

4.2.1 Set Up

The following was contributed by: M. Toon

To simulate the motion of the robot swinging using its legs, a 500g mass was attached to an aluminium rod and swung below the swing over a range of 90° using a stepper motor as shown in Figure 4.1. The mass of the lower leg of the robot was calculated to be 588g from the values provided from the manufacturers website [34]. The distance of the centre of mass of the lower leg from the knee joint was calculated using the exact values from the documentation to be 86.8mm and so was the distance set between the axis of the motor and the centre of the 500g mass on the aluminium rod.

The `3308_0 bipolar` stepper motor used had a rated current of 1.7A, step angle of 0.9° and holding torque of 9kgcm equivalent to 0.8826Nm [35]. At 90° the torque caused by the mass on the rod was 0.43±0.08Nm which meant that the motor could easily move the mass.

To control the stepper motor a 1067_0−`PhidigetStepper bipolar HC` controller was used to interface with the stepper motor and a computer as shown in Figure 4.2 [36]. The angle of the swing was recorded using an encoder which converts the physical angle at the pivot of the swing to a digital signal that can be interpreted by the computer [2]. When connected, the stepper motor was fixed centrally beneath the swing using a clamp to minimise any twisting motion caused by asymmetry when applying driving force.

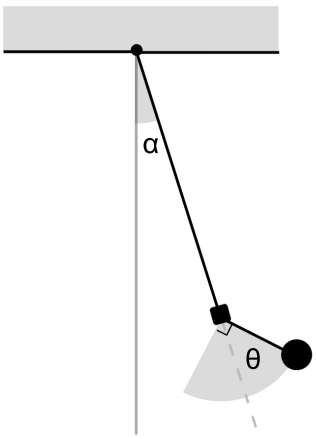


Figure 4.1: Diagram showing how the motor and mass were attached to the swing and how the swing angle α and motor position θ were defined.

The stepper motor was connected to the Phidget board with four wires shown in Figure 4.2 according to the manufacturers guide [37]. The motor drew its power from the Phidget board which was in turn connected to a power supply set at 11.7±0.1V and current limit of 2.00±0.01A. The board drew 0.95±0.01A when moving the motor and nothing when idle. The encoder was also connected to a separate power supply set at 6.60±0.01V and 0.43±0.01A. The encoder would normally require 5V but the potential drop from the resistances due to the long length of the wire required a higher voltage supply.

4.2.2 Finding Optimal Driving Period

4.2.2.1 Introduction

The periodically driven swing is a damped driven oscillator and so the optimal driving frequency should lie close to the natural frequency. To find the optimal driving frequency the swing was driven at a set number of frequencies either side of the natural frequency. When a driving force is not exactly equal but close to the natural frequency of the harmonic system then beating in the amplitude of the swing should occur [38]. The periodic driving motion would only be used to achieve some initial amplitude such that the swing direction could be determined by the feedback source, and so the maximum amplitude achieved is extracted from the recorded oscillation.

4.2.2.2 Method

Initially the natural period of the swing was determined by recording the time for ten freely swinging oscillations. A natural period of 2.542±0.001s was calculated and set as the mid point for setting the other values of period.

The code, `Stepper-simple.py` [39], was modified to suit the purpose of this experiment to produce `singleRunPeriod.py` which may be found in the Appendix E. The Python language was chosen due to its high versatility and simplicity. Both the encoder and Phidget board can use the language as well as the many libraries which are useful to plot and analyse data which made it the optimal choice. This code works with the encoder and stepper motor to automatically drive the mass for a defined time at a set number of periods whilst also recording all of the angle data to a text file.

The swing was allowed to be driven with a period of 2.54s for a total time of 250s to see if the swing would reach a steady state amplitude. It was found that the amplitude increased to a maximum and then proceeded to beat as shown in Figure 4.3.

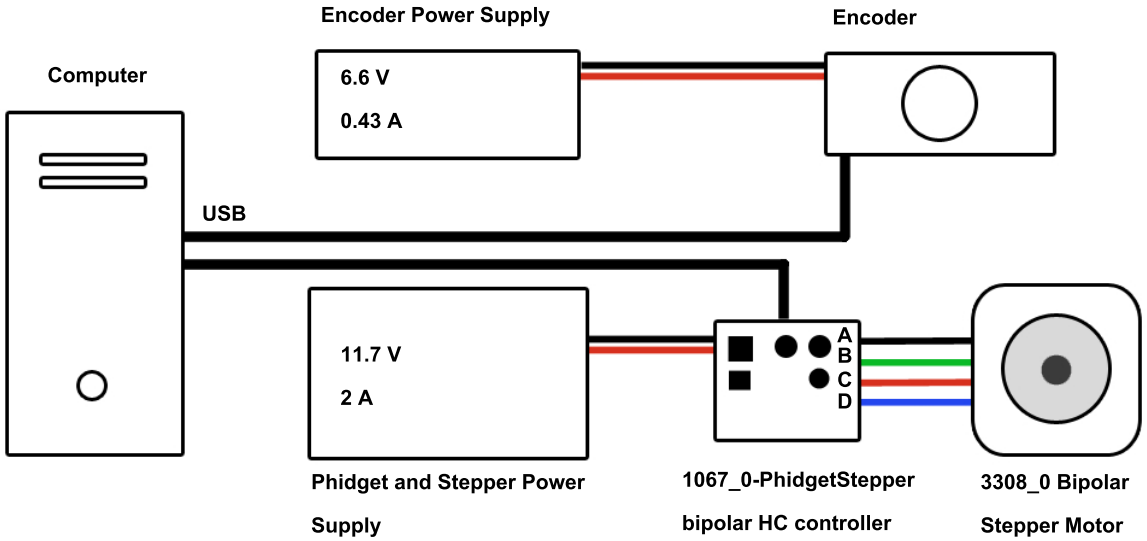


Figure 4.2: Schematic diagram of the set up used for the predetermined swinging experiment [37].

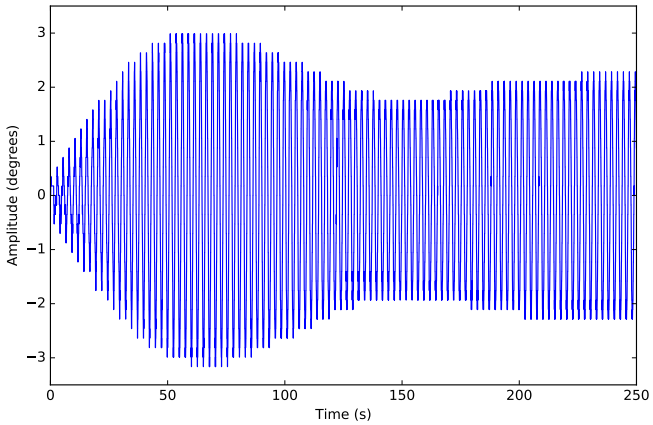


Figure 4.3: Plot displaying the how the amplitude varies with time for the initial run of the swing driven with a period of 2.54s. A steady state is not achieved and beating is observed.

The swing was made stationary before powering the stepper motor with the power supply. The python code `singleRunPeriod.py` was ran from the computer to automatically collect the data for the first period of 2.522s. When prompted by the program, the motor disengages and the swing was made stationary once more. The program then switched the driving period to increase by 0.002s before starting the next data collection cycle when given an input. This was repeated twenty times until a full data set was collected.

It was found that the stepper motor occasionally did not return to the equilibrium position and so drove the swing asymmetrically. This systematic error was overcome by retaking erroneous data points and averaging over multiple data sets.

4.2.2.3 Analysis

The swing was driven for the same amount of time for each period and so the optimal driving period would attain the largest amplitude. The python code `resonanaceplotter.py` was written to loop through all of the text files which stored the time and angular position of both the swing and stepper

motor. It then extracted the largest value of angle from each text file and stores it into a new text file. This last file containing the driving period and respective max angle achieved was then put into the `resonancefittingprogram.py` to both plot the resonance curve and find a fit.

The best data set was analysed using these programs and produced the resonance curve shown in Figure 4.4. A systematic error of $\pm 0.5°$ was given to each data point due to the issues with the stepper motor previously mentioned. The data point for the period at 2.540s was outside the error margin and similar results were found with other data sets. Once the driving program reached this period it would cause the stepper motor to malfunction and so the data point had to be taken again.

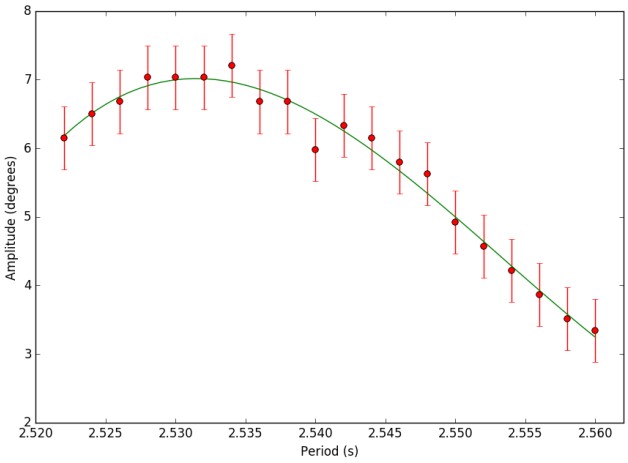


Figure 4.4: Resonance curve produced from plotting maximum angle achieved against the driving period with fit to find optimal period.

4.2.2.4 Results and Conclusion

As shown in Figure 4.4, the largest amplitude achieved was $7.0 \pm 0.5°$ corresponding to a period of 2.531s. As expected, it was close to the natural period 2.542 ± 0.001s but not exactly the same due to the friction from the air and swing joints. The large amplitude created would easily be enough for most feedback sources to determine direction. However it would not be ideal to fully drive the swing as the driving force might fall out of phase with the swing (as displayed in Figure 4.3) and thus lose amplitude.

The resonance peak has a width of 0.030 ± 0.004s and it is clear that a small deviation of 0.03s from the maximum value reduces the amplitude attained by the swing by approximately 4°. Adding more damping would widen the resonance peak and make it easier to reach the largest amplitude, however more energy would be lost from the system and so the maximum amplitude would be lower. So using a periodic driving force is very powerful method of driving the swing if the optimum driving frequency is used, but finding and sustaining this frequency would prove impractical with the robot. The best method would be to ensure that the robot swings at the optimal period up to the point at which it begins to go out of phase and then switch to a feedback driving motion.

4.3 Analytical Motion

4.3.1 Webots Controllers

The following was contributed by: D. Butters

In order to test models that cannot be easily implemented on the swing apparatus using the robot, the Webots robot simulation software that was mentioned in Section 2.5.2 was used. This software allows for the creation of physical structures and robots and simulates them using the Open Dynamics Engine [10]. The software runs with a minimum time step of 8ms, whereupon the entire world is updated according to the physics engine and displayed to the user.

Webots has the NAO robot as a pre-existing object that can be added to the virtual environment. The simulated robot controls all of it's motors, sensors etc. from a single controller file written in *C*. This controller file is used to implement all motions on the robot. For both the analytical motion and the human motion simulations, the robot runs a loop which checks for certain conditions for movement (such as the swing reaching a certain angle) and then, if the condition is matched, calls functions to change the angles of certain joints to make the robot move. In all cases, the only joints that are told to move are the left and right hip joints (to move the torso, as the upper leg is attached to the seat), and the left and right knee joints. The arms are kept stationary as they do not contribute significantly to the motion of the robot.

The simulated robot has the same dimensions and distribution of mass as the real robot, however there are some key differences between the simulated robot and the real one. The sensors in the simulation, including the gyroscope and inertial unit, are idealised. This means that they output with the maximum accuracy allowed by the physics engine. Inertial Unit nodes in Webots output the absolute angle of the node with respect to the world in three dimensions (pitch, yaw and roll). Gyroscopes output the angular velocity of the node in three dimensions. Sensors can be placed wherever they are needed throughout the program, for example, rather than using an encoder system to measure the angle of the swing, a gyroscope can be placed within the seat of the swing, with the data sent directly to the robot.

When receiving data from sensors, the controller file of the robot only has access to sensors that are part of the tree of objects that makes up the robot. So, when data needs transferring between two systems e.g. between the gyroscope on the swing seat and the NAO robot, an Emitter - Receiver system must be used. An Emitter node can be placed on the swing which can emit data with an (idealised) infinite range and takes one time step to transmit. This node must send data in packets of a predefined size. A Receiver node can be placed on the robot. This node stores a queue of packets which it has received from a certain channel. It is simple to write a function in the controller code of the robot which automatically returns the most recent packet to be sent to the Receiver, allowing for quick transfer of information between the swing and the robot.

4.3.2 Simulation in Webots

4.3.2.1 Flail Simulation

Webots was used to model the motion of the robot controlled using an analytical equation of motion. The simple swing was constructed in Webots, as described in Section 2.5.2. This swing had a single hinge joint on each arm of the swing, and a solid seat for the robot to sit on. The seat and the hinge joint were connected by two non-deformable cylinders. A guide rail was placed on the seat so that the robot was physically prevented from slipping off the seat. The set up of this swing is shown in Figure 4.5.

The analytical motion described in Section 3.1.3 uses angles defined in Figure 4.5. Angle 1 is defined as the swing angle, θ_1, with 0 defined as the swing hanging vertically down. Angle 2 is defined as the angle of the body, θ_2, of the robot from vertical. Angle 3 is defined as the angle of the legs, θ_3, of the robot from vertical. These are the same angles used in Equation 3.27. At each time step, the controller file calculated the acceleration of the swing using this formula. The controller would test the acceleration of the swing for different angles of the torso and knee. The test would include all combinations of the torso being at it's current angle, tilted slightly forward or backward, and the knees being at their current angle, tilted slightly forward or back. The combination of positions that generated the best acceleration would then be used to move the robot to the position that would generate the biggest acceleration in that time step.

The performance of this movement model was not completely successful. Figure 4.6 shows that as the robot carries out the analytical motion as described, the swing quickly loses amplitude. The swings amplitude then stays relatively constant at ≈ 0.4 radians. This demonstrates that the model implemented in its current form is not suitable for transfer to the robot.

Adjustments were made to the boundary conditions of the model. At a given moment, the best angle for the torso may be an angle that the NAO robot cannot achieve, so the algorithm would instruct the robot to stay at the limit of its range of movement. An adjustment was made so that if the robot was at it's maximum or minimum angle, it would test both extremes for their accelerations. Then the robot would change its position to whichever position had the better acceleration, ignoring other commands until it reached that position. This would avoid the robot getting stuck in a local maximum which would

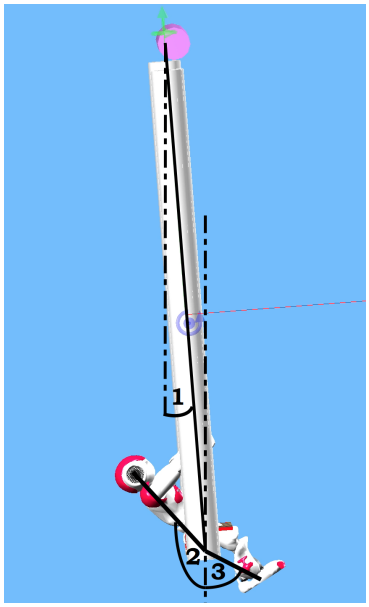


Figure 4.5: Angles defined in the analytical simple pendulum simulation on Webots.

not be the true maximum. This adjustment helped, but even after these adjustments the robot did not achieve a significant amplitude on the swing.

A problem with this model of movement is that it tries to maximise its acceleration at every time step. However, this does not account for the cyclical nature of a swinging motion, where the acceleration cannot and should not always be maximised. This is because at the end of the swing, it may be better for the motion of the swing if the robot stays still or prepares to swing backwards. However, this algorithm would still be attempting to keep the swing moving forwards. This behaviour is a greedy algorithm, where the motion that gives best short term result is always chosen, regardless of how this action affects success in the long term. This demonstrated a flaw in the model, where it is too short sighted to provide a true optimum swinging motion as it is always attempting to accelerate when, inevitably, the swing will change direction eventually. This flaw could be mitigated by the application of machine learning techniques on the robot, where the cyclical nature of the swinging motion is taken more fully into account.

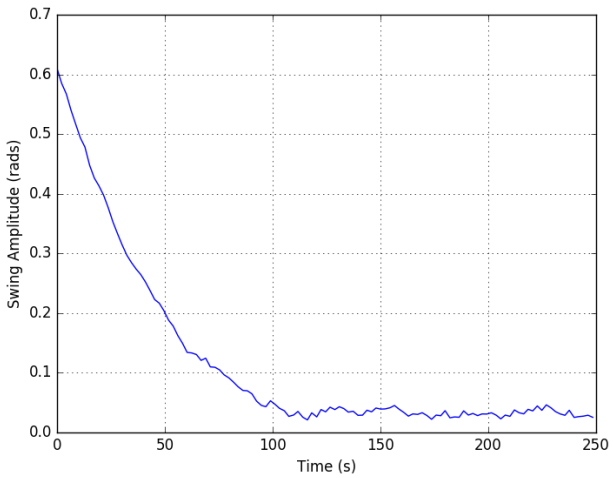


Figure 4.6: Swing amplitude vs time for Analytical flail motion in Webots.

4.3.2.2 Hinged Flail Simulation

The following was contributed by: J. Sweeney

To simulate the motion expected with the swing design outlined in Section 2.5.1, two methods were considered.

First, a numerical solution to the analytical equations of motion for the hinged flail for $\ddot{\theta}$ and $\ddot{\alpha}$, from Equation 3.30, was determined for the case of generalised sinusoidal driving functions of Φ andΨ. To achieve this, the equations for $\ddot{\theta}$ and $\ddot{\alpha}$ were decoupled and reduced to a set of first order differential equations. To accurately model the experiment, damping terms proportional to angular velocity were introduced (see Appendix A.3 for values and derivations). The two masses shown in Figure 3.9 were constrained to oscillate according to the equations,

$$\Phi = \alpha + \frac{\pi}{2} + \Phi_r cos(\omega_1 t + \phi_1), \tag{4.1}$$

$$\Psi = \alpha + \Psi_r cos(\omega_2 t + \phi_2). \tag{4.2}$$

Here Φ and Ψ were assumed to be the angle between the robot's upper and lower body with regards to vertical, respectively, whilst Φ_r, Ψ_r are their corresponding angular ranges of motion (see Section 2.1); $\omega_{1,2}$ their frequency of oscillation; and $\phi_{1,2}$ their associated phase differences.

The decoupled equations were then solved via a Runge-Kutta, 8th order accurate, adaptive step size routine. This is a standard iterative algorithm used for solving ordinary differential equations. This was implemented using a `C` numerical library: the `GNU Scientific Library (GSL) version 1.13`. This was initially performed for a range of $\omega_{1,2}$ from zero to three Hertz as it was assumed that the robot could not sustain continuous motion at frequencies above this rate. This method was then repeated about the observed peak value to obtain a more accurate estimate, as shown in Figure 4.8 below.

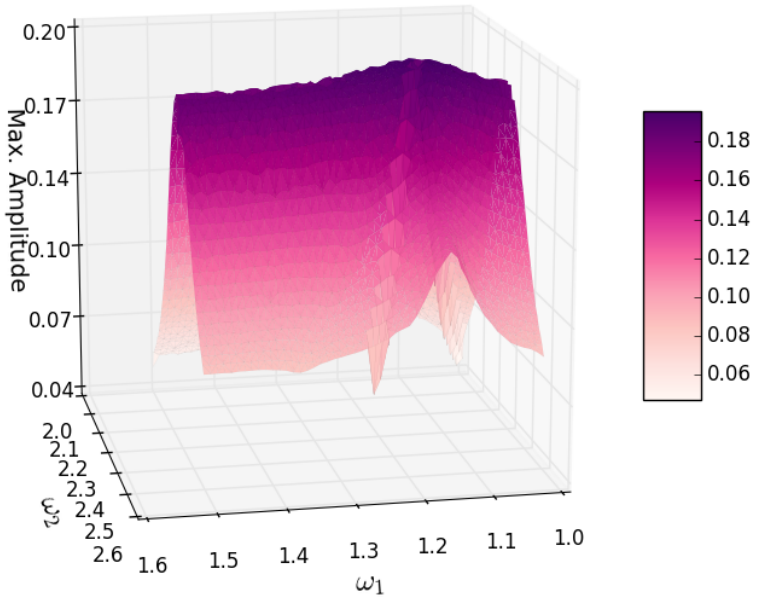


Figure 4.7: Surface plot of peak amplitude (radians) for varing frequencies of upper (ω_1) and lower (ω_2) body motion.

From this, it was determined that the peak achievable amplitude was 0.196 ± 0.001 radians, or $\sim 11.5°$ and corresponded to the frequencies $\omega_1 = 1.130 \pm 0.005$ Hz, $\omega_2 = 2.280 \pm 0.005$ Hz. Although difficult to verify analytically, if it is presumed that the upper mass acts as a driven oscillator – i.e. with greatest

amplitude at the natural frequency – it can be reasonably assumed that the lower mass acts akin to a parametric oscillator with resonant behaviour at approximately twice this frequency. This is supported further by previous work performed by A. Post et al.[40].

Nevertheless, it can be seen in Figure 4.8 that values close to this peak can be seen to drop sharply in amplitude, with almost a twenty percent decrease in amplitude in th case $\omega_2 = 2.27$ Hz instead. This is evidence of an unstable resonance, typical of a chaotic system. As such, it was determined that values of ω_1 away from this peak may be more suitable due to their greater stability despite a reduced peak amplitude.

To further investigate the behaviour exhibited using the hinged flail model, an identical approach was performed as in Section 4.3.2, with additional gyrometers and inertial units placed within the vertical bar connected to the hand rail. From Figure 4.8, it can be seen that this approach fared even worse when implemented on the more complex swing.

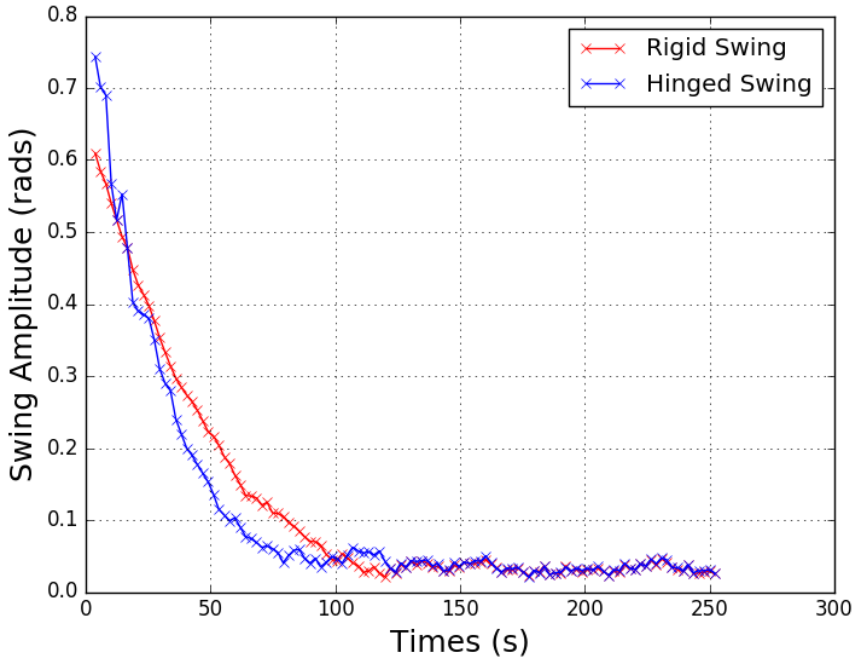


Figure 4.8: Swing Amplitude vs time for analytical motion using both rigid arm and hinged swings.

This was presumably due to the larger degrees of freedom present within the hinged swing when compared to the rigid model. As such, it exhibits greater chaotic behaviour and therefore has a higher sensitivity to current conditions and is more difficult to keep at resonance, as evident from the previous results. Consequently, it can be seen to decay to the same amplitude level as the rigid swing, but in a shorter time frame.

From this, it can been seen that a "greedy" algorithmic approach was not suited to achieving a large amplitude for either the rigid or hinged swing systems. Via the numerical solution, however, an estimate of the magnitude of peak amplitude could be taken to be 0.196 radians.

4.3.3 Implementing to the Robot

The following was contributed by: P. Jones

The next task was to try and get the robot to swing from the analytical motion for the flail described in Section 3.1.3. It was decided that this would be achieved by the robot periodically finding its acceleration from Equation 3.27, and subsequently determining the best course of action to take in order to maximise this, similar to the Webots simulation. Initially, this was to be done simply by using the equation to determine whether moving to a forwards or backwards position would lead to the greatest

increase in acceleration. However, a number of difficulties were encountered during the implementation of this.

The first problem was that of measuring the angles used in Equation 3.27. This requires the angles to the centres of mass of both the upper and lower body of the robot relative to the swing. The NAO API provides a method for finding the centre of mass (COM) of a specific body part or chain, `ALMotionProxy::getCOM()`. This gives the COM of a body part in the reference frame of either the robot's torso, a fixed initial position, or the average of the robot's feet. As a frame of reference that moved with the swing was required, the torso frame was chosen. The positions of the upper and lower body COMs then had to be transformed from the torso frame, into the seat's frame, as these were the values required. This proved relatively simple for the upper body, which only had to be offset by the height of the torso's frame above the seat, then rotated by the angle of the robot's hips. This method did not work perfectly for the lower body, however a more significant problem arose before a workaround could be found.

The major problem with using Equation 3.27 for the motion of the robot arose from the need for values of the angular velocities of the upper and lower body centres of mass about the seat. Calculating the position of the centre of mass of each body part was a slow process on the robot. As calculating the upper and lower COMs' angular velocities required the use of at least two COM readings, the value would lag far behind the actual motion of the robot, leading to inefficient movement. The same problems also apply to the hinged flail model. A possible workaround for this would be to use Equation 3.27 or 3.30, along with measurements of the robot's movement speed, to pre-calculate the best motion for the robot at various points throughout the swing. This is a source for possible future improvement.

4.4 Human Motion

4.4.1 Simulating in Webots

The following was contributed by: D. Butters

In order to implement human motion on Webots, data from the Human motion tests was analysed in order to produce a graph of torso and knee angle with respect to swing angle.

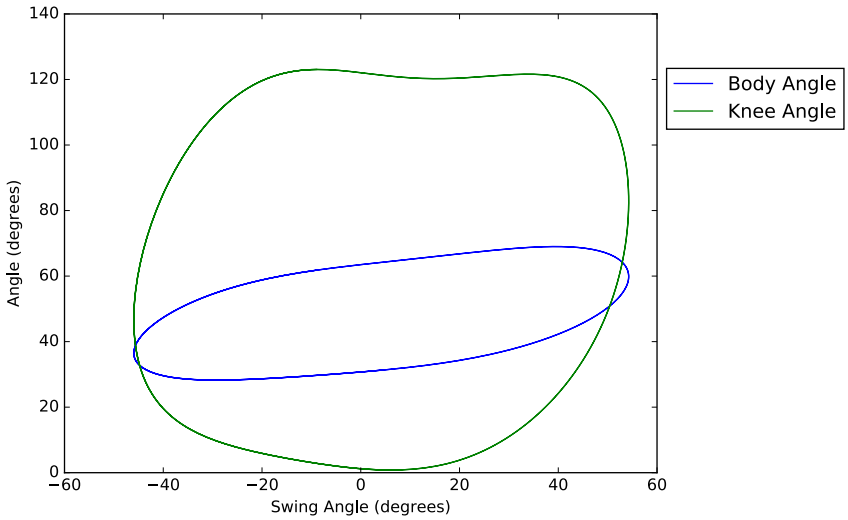


Figure 4.9: Torso and knee angles vs swing angle from human motion.

Figure 4.9 shows the angle of the knee and torso for a human swinging on a classic rope swing. This graph can be normalised so that the motion is not defined by swing angle, but by the portion of the way through the swing. For example, if the minimum amplitude is -20° and the maximum amplitude is +20 degrees, the robot will match the motion defined at $\frac{3}{4}$ of the way through the swing when it reaches +10°.

As the robot swings, the maximum and minimum amplitude of the previous swing is used to calibrate the motion of the robot in the current swing, normalising the curves shown in Figure 4.9. As the swing travels forward, the robot is instructed to match the forward curve with its torso and knee angles. Then, as the swing travels backwards, the robot can do the same thing with the backwards angles. In this way the robot mimics human motion. An adjustment can be made so that the robot anticipates the position its torso and knees should be in, beginning the motion before the target angle is reached. This adjustment improves the swinging because the robot is more closely matching the true motion of a human. This happens because it takes some time for the robot to change position, so beginning the movement earlier by anticipating the movement will provide a more responsive movement than retrospectively copying the movement once the target angle is reached.

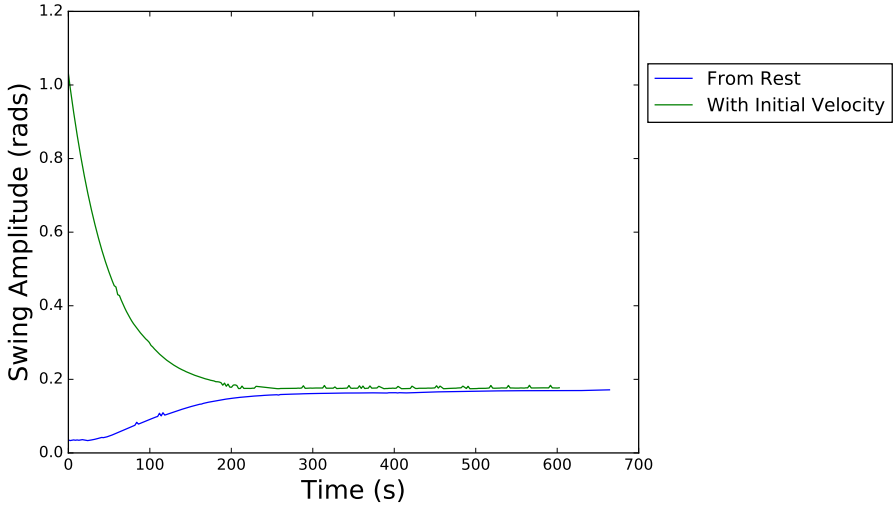


Figure 4.10: Swing amplitude vs time for Human motion in Webots.

The simulated performance of this movement model showed some promise. Figure 4.10 shows that when the robot begins this motion with a large initial velocity, the swing amplitude does decrease in an exponential decrease. However, the decrease gradually slows until the amplitude of the swing tends to ≈ 0.2 radians. This shows that the motion can carry on swinging at a significant amplitude, overcoming the frictional forces of the system when initial energy is given to the system. The fact that the swing initially decreases in amplitude from the large initial value is expected, as the system takes some time to reach equilibrium. After a long time the system is expected to reach an equilibrium point where the energy input to the system by the swinging robot is equal to the energy lost from the system through damping.

Another interesting feature of Figure 4.10 is that it shows that when the robot is only given a small initial push, the human swinging motion actually introduces energy into the system, increasing the amplitude of the swing until it again tends to ≈ 0.2 radians. This graph again shows that after a significant time (≈ 600s), the system approaches equilibrium. The results from this simulation lead to the conclusion that the human motion model is suitable for implementation on the robot in the lab.

4.4.2 Implementing to the Robot

The following was contributed by: P. Jones

In order to replicate human motion on the robot, the human motion data was normalised to represent the fraction of the swing complete, rather than the actual swing angle, as described in Section 3.2.5. 4^{th} order polynomials were then fitted, with `gnuplot`, to each of the robot's knee angles and hip angles, on both the forwards and backwards swing. 4^{th} order polynomials were chosen as they gave a good trade-off between speed and accuracy - fitting a 5^{th} order polynomial provided very little accuracy increase for the extra performance cost. An example fit is shown in Figure 4.11.

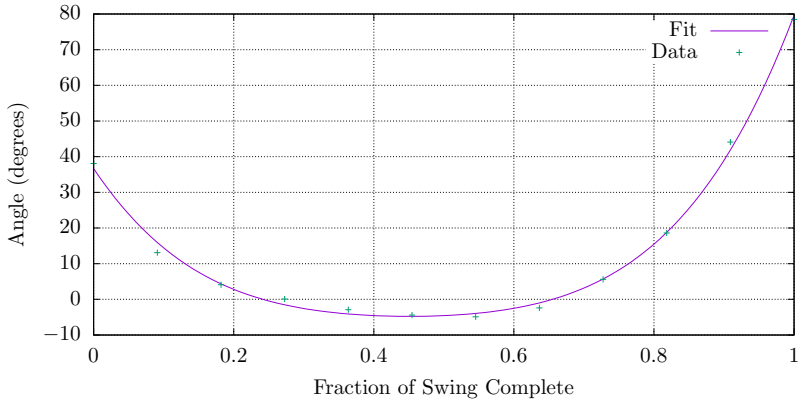


Figure 4.11: The human motion model of the robot's knee angle during the forwards swing, with a 4^{th} order polynomial fitted. The maximum deviation of the fit from the data is $\approx 3°$.

To determine how to move, the robot would read the swing's angle from the encoder, and recorded the maximum and minimum angles achieved so far. It would also use the previous two readings from the encoder to determine whether it was swinging forwards or backwards. The robot would then measure the swing angle every ≈ 60ms, and converted this into a fraction of the way between the current maximum and minimum angles. This value was then inserted into the relevant fitted equations of motion, depending on whether the robot was moving forwards or backwards to find the correct angle to move both its knees and hips to. Unfortunately, at this point in the project, the robot's right arm was not working, and so only the legs could be moved safely, and the upper body was kept still. The results of this are shown in Figure 4.12.

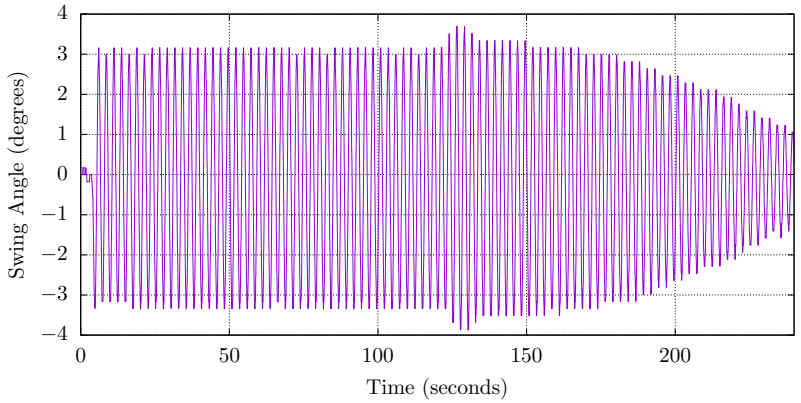


Figure 4.12: The results of implementing human-like motion on the robot, using only its legs. The decreasing amplitude after about three minutes was cause by the knee joints beginning to overheat.

As can be seen from Figure 4.12, the robot was able to swing successfully by emulating a human's motion with just its legs, with a steady amplitude of $3.2 \pm 0.1°$ being achieved for about three minutes. However, after this time, the knee joints on the robot began to overheat, leading to a rapid decrease in amplitude. The time taken to overheat was much shorter than observed when testing other swinging methods, and is likely due to the fact that human motion involves constant movement throughout a swing, rather than a single, quick movement at each end of the swing, as used with many of the other swinging methods investigated. It would still be desirable to test the human motion with the torso as well as the legs, as the torso was shown to be the major contributing factor to the swinging motion in Section 3.2.3, and the Webots simulation was able to reach a much greater steady amplitude of $\approx 11°$ with the torso.

4.5 Chapter Conclusion

In this section, various methods of using predetermined motion were investigated. Initially, a stepper motor was controlled by a Phidget controller to periodically drive a mass, representing the robot's legs, with different fixed periods. A resonance was seen at a period close to the natural period of the swing of 2.542 ± 0.001s, resulting in a swinging amplitude of $7.0 \pm 0.5°$. Beating was also observed, as the motion moved in and out of phase with the swing. It was therefore decided that the best method for getting the robot to swing from rest would be to use this fixed period motion until maximum amplitude was reached, and then switch to a more advanced method, such as swinging from visual or gyrometer feedback. This was because many of the more advanced swinging methods require some initial movement, which was otherwise provided by a person pushing the swing.

The analytical motion for the flail and hinged flail described in Sections 3.1.3 and 3.1.4 were used to try and allow the robot to determine its next best movement, based on its current state. These was initially simulated in Webots, and both resulted in a very small swinging amplitude of $\approx 2°$. It was decided that this was most likely a result of the robot trying to maximise its acceleration at every time step, which is not the optimal solution for swinging on a swing. A possible improvement would be to take future movements into account.

An attempt was then made to implement the analytical motion on the robot. However, numerous difficulties were encountered in this, such as the need to find the angles to and angular velocities of the robot's upper and lower body relative to the swing. In addition, the analytical model performed poorly in the Webots simulation, as seen in Figure 4.6. As a result it was decided that time should be used to improve other areas of the project, and the analytical motion was not implemented on the robot.

Finally, the way in which a human moves on a swing was investigated, and was replicated in both Webots and on the robot. It was found that humans do not swing in the optimal way to maximise swinging amplitude on a swing, via parametric motion. Instead, the seated motion used was thought to be a result of trying to maintain stability, and minimise the effort required to swing effectively. The appropriate angles for both the knees and hips were found throughout the swing, and used to replicate human motion.

The motion was first replicated in a Webots simulation. It was found that, from both an initial velocity and from rest, the robot approached a steady state amplitude of 0.2 radians, $\approx 11°$. This was a promising result, and so the motion was replicated again on the robot. However, this was not as successful. The robot's right arm was broken at this point, so only the legs could be used safely. As the majority of the torque was found to come from the torso, this greatly reduced the steady state amplitude of the swinging to $3.2 \pm 0.1°$, roughly one quarter of that achieved in Webots. Additionally, after about three minutes of swinging, the robot's knee joints began to overheat from the constant motion, resulting in a rapid decrease in swinging amplitude. It would still be desirable to test the motion with torso movement, as the $\approx 11°$ amplitude observed in Webots is promising, however the overheating is likely to be the limiting factor, as the time to overheat was much shorter than the required time for the simulation to reach its steady state.

Chapter 5

Feedback Swinging

5.1 Introduction

The following was contributed by: M. Lim

The aim of these experiments was to illustrate the method of using feedback to accurately control the motion of the swing. Two fundamental requirements of using feedback to control the driving motion of the swing are: define driving force, this is the movement of the driving mechanism from one position to another; and use a detection method to find the peaks of the swinging motion in order to trigger the driving force.

In Chapter 4 we demonstrated the efficacy of using a predetermined driving force. In this Chapter, we explored the advantages and limitations of using the encoder in the top of the swing, and built in sensors on the robot, including the gyrometer and the cameras. The main challenge was to accurately interpret the information from the sensors to identify when to execute the driving motion.

The accuracy of the encoder was investigated. This was done firstly by driving the swing with the Phidget and stepper motor. Next, the encoder was connected to the robot, and the encoder data was read by the robot to find the peak of the swing. Then, the cameras built into the robot's head were used to track landmarks to identify the swing motion. And finally, the gyrometer was used to model each swing oscillation as a sinusoidal function. A comparison could then be made between the different methods to identify which was most effective.

5.2 Encoder Feedback

5.2.1 Phidget Double Pendulum with Encoder Feedback

The first instance of using feedback to control swinging motion was performed using the Phidget-double pendulum setup described in Section 4.2.1. As illustrated, using a periodic driving force has its own limitations. It requires a lot of experimentation to determine the best period to use, and it would all have to be repeated if another swing or robot was to be used. As an alternative method, feedback regarding the motion and position of the swing may be used to determine the motion of the robot or the pendulum. The encoder attached to the top of the swing may be used as one example of feedback into the system.

In order to control the Phidget, the position of the swing was taken from the encoder, and two successive values were compared in order to identify when the direction of motion of the swing changed. This is the basic principle of using the encoder to identify the peak of the swing motion, Figure 5.4 is an example of Python code using this method of peak detection on the robot. When encoder values begin decreasing, one position is chosen, when encoder values begin increasing then the other position is chosen. This principle was demonstrated with the stepper motor on the swing, and a similar system was used on the robot.

The Phidget and stepper motor were attached to the swing, as in the periodic motion tests shown in Figure 4.1. Each test began with the swing at rest. A program called `phidgetstepperfeedback.py` was written in Python. This program began moving the mass on the stepper motor in a periodic fashion at

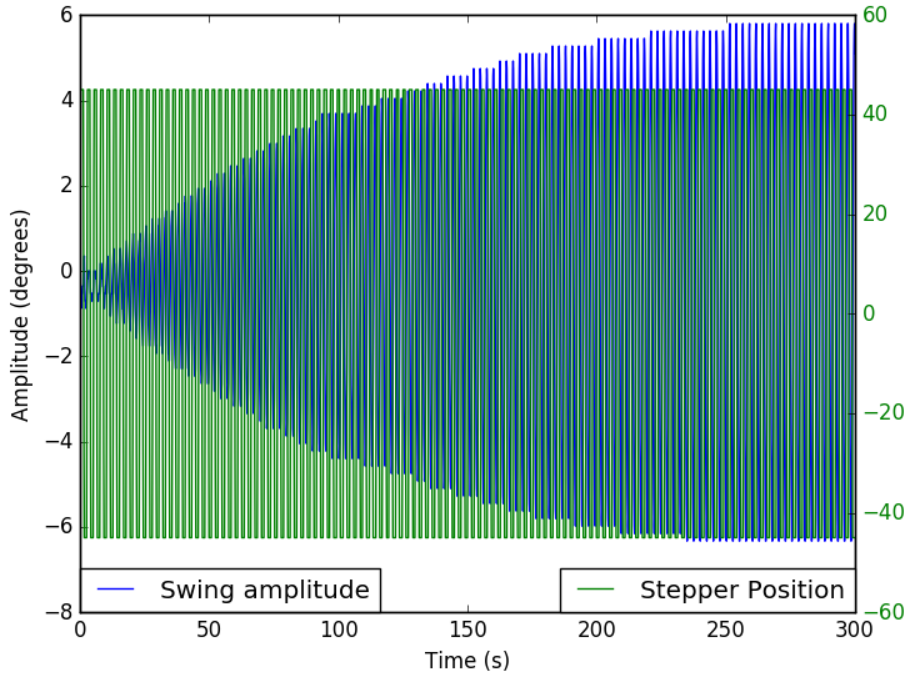


Figure 5.1: The amplitude of swing driven by the encoder and stepper motor. Stepper motor changes between 45° to -45°.

the natural frequency in order to start the swing from rest and to build up the amplitude of the swing. Once the swinging achieved an amplitude specified in the program, the Phidget controller then stopped driving the swing periodically and began using the encoder feedback to drive the swing.

We took data for the swing where the switch from periodic to encoder feedback occurred at 0.5°, 1.0°, 2.0°, and 3.0° (all with an error of ±0.1°). The error on the switchover angle was taken to be equal to the precision of the encoder. Data was recorded for the position of the swing and of the stepper motor. We chose these values to allow us to directly compare the characteristics of the encoder method and the periodic method when building up amplitude of the swing oscillations.

Figure 5.1 shows the swing oscillations when the switchover occurred at 3±0.1°. By comparison with Figure 4.3, we can see that the motion does not have the same pattern of beats occurring and the swing has a far more consistent amplitude. The maximum amplitude achieved by the swing when driven by a predetermined driving force was found to be 7.0±0.5°, as previously stated. However, the encoder method achieved a maximum amplitude of 6.3±0.5° over a similar time period, the error on this maximum amplitude value was found in the same manner as in Section 4.2. This indicates that the predetermined driving force is more efficient than using the encoder.

Figure 5.2 shows how the amplitude of the swing changed with time. It can clearly be seen that the curve increases in discrete steps. This is due to the limited precision of the encoder. This is a very important feature in the efficacy of using the encoder as a method of feedback.

It can also be seen that when the change from a periodic to the encoder feedback occurred at 3±0.1°, then the largest final amplitude was achieved, 6.3±0.5°. Conversely, when the change over occurred at 0.5±0.1° of amplitude, then a much smaller final amplitude was achieved of 4.0±0.5°. This illustrates that the predetermined periodic driving force is far more effective at small amplitudes and when building up the amplitude of the swing than the encoder feedback method. This is a result of the far larger error in determining the peak of a single oscillation when using the encoder. At small amplitudes, the error as a fraction of the amplitude of oscillation is very large, for example, at 0.5°, and since the encoder has a precision of 0.1°, then the error is one fifths of the amplitude of oscillation.

Hence, an encoder with greater precision would more accurately determine the peak of the oscillations and so would achieve larger amplitudes. In this instance, we found that driving the swing at a

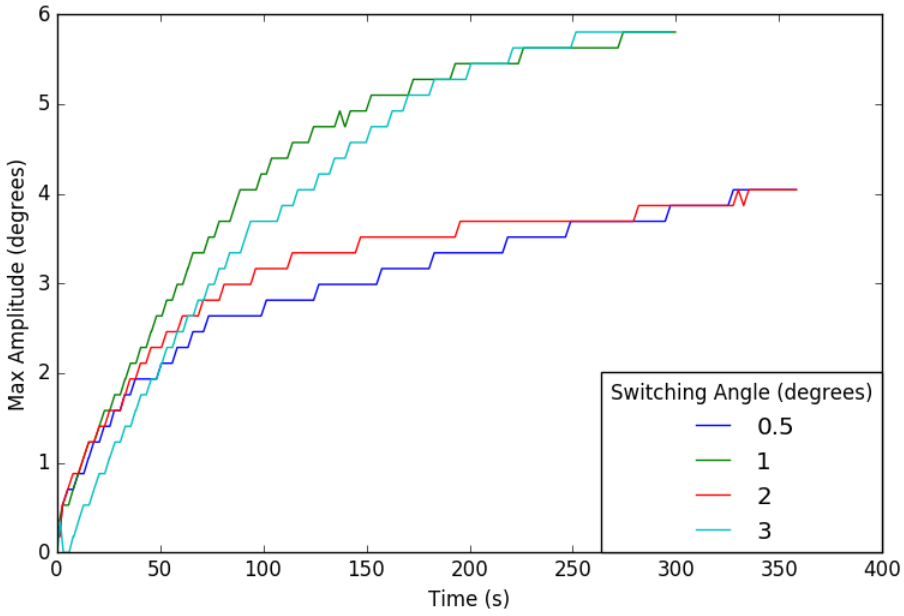


Figure 5.2: The amplitude of oscillations for various switching times. The later the switch from predetermined to using encoder feedback, the larger the final amplitude.

predetermined rate is by far more effective than using encoder feedback, especially at small amplitudes and when building up the amplitude of oscillations from rest. Then encoder feedback does, however, produce oscillations with a more consistent amplitude, without beat patterns occurring.

5.2.2 NAO Robot with Encoder Feedback

The following was contributed by: H. Jacobs

The first attempt to get the robot to swing based on feedback was from sending the robot commands based on the measured angle of the swing. This was done via the encoder device [2] and a simple Python script. The script would continually check the angle and, if it started to change direction, issue commands to the robot to either move to its maximum or minimum extension.

This script was run on a computer, connected to the same Wifi network as the robot, that was additionally connected to the encoder. The computer would then send the commands to the robot to move over the network (see Appendix D for more details).

With the robot starting at rest, the code, Figure 5.4, would move the robot every half a period and would continue until the swing reached a predefined amplitude of 6.5°. From then on, the robot would only move when the angle changed direction. This relatively simple code produced decent swing amplitude, which can be seen in Figure 5.3. This was able to reach an amplitude of $7.2° \pm 0.1$ after 3mins. The swap between the two sections of the code can be seen at just before 150s in Figure 5.3.

For such a simple attempt, the amplitude reached on the swing was very promising. Both methods in the Python code: using an approximation of the period of the swing; and using the change in direction of the angle, were able to increase the amplitude of the swing.

As this work was done before the robot's right arm was damaged, these tests were conducted with moving both the legs and upper body. This, together with the fact that the priority of the project was to enable the robot swing without external data, meant that this line of experimentation was not pursued any further.

In future, further work could be done with the encoder as it can be used to produce an excellent swing motion. This is because the robot would be able to know its location precisely. One possible area

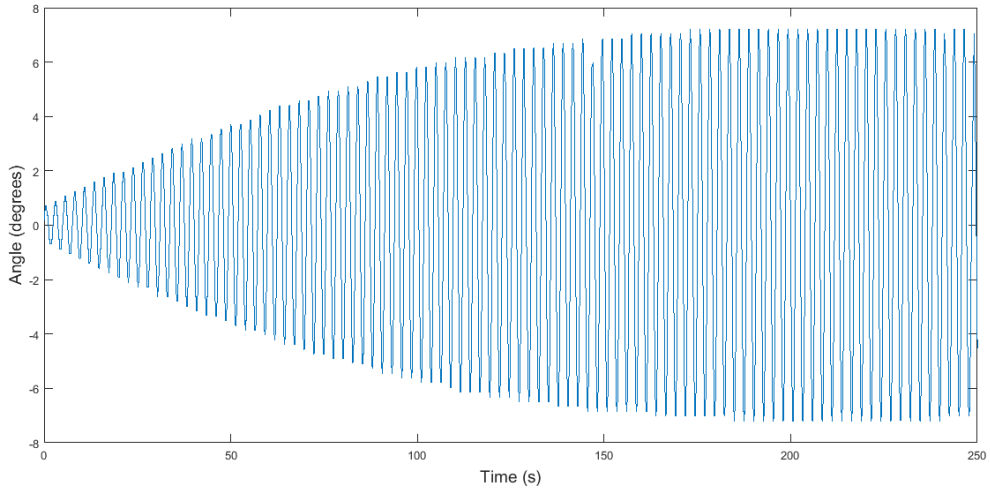


Figure 5.3: Encoder feedback robot swing.

```
        motionProxy.setAngles(min_joints,  min_angles,  fractionMaxSpeed)
        time.sleep(1.3)
        motionProxy.setAngles(max_joints,  max_angles,  fractionMaxSpeed)
        time.sleep(1.3)

    while True:
        if a > b: # Compare consecutive angles to see which way the swing is moving
            motionProxy.setAngles(min_joints,  min_angles,  fractionMaxSpeed)
            time.sleep(0.3)

        if a < b:
            motionProxy.setAngles(max_joints,  max_angles,  fractionMaxSpeed)
            time.sleep(0.3)

        b = a
        a = encoder.getAngle()
        pass
```

Figure 5.4: Angle Feedback Python Script

would be determining the absolute maximum amplitude the robot could reach which could then be used to compare the success of other swing tests.

5.3 Vision Feedback

The following was contributed by: E. Humphreys

Since it was shown that the vision software built into the robot was capable of tracking the circular landmarks, it was decided to utilise this software in a feedback system. There were two ways considered to use the vision for feedback. The first would be to determine an equation linking the vertical position of a landmark in its field of vision with the angle of the swing. This method was decided to not be followed as it would have to take into account the positioning of the landmark, and the equation would need to be altered for different landmark positions. It would be possible to take these positions into account when determining the equation, however, time could have been better spent investigating the second method.

The second method for vision feedback compares the (angular) vertical position of one or more landmarks with their previous position. If the position change was positive (i.e. landmark(s) moving up in field of vision) it would be determined the robot was moving backwards, and the robot would

be ordered to move to a minimal extension position. If the change is negative (i.e landmark(s) moving down in field of vision), then the robot would be moving forward and would attempt to reach a maximal extension position. Since this only relies on the robot having a landmark in its field of vision that it can reliably track we decided to focus entirely on this method since it could quickly be tested.

When doing preliminary testing a smoothing function had to be added to the script so that any sudden movements by the robot would not affect the robots perception of its movement. This, rather than comparing the position with the previous position, compared it to the average of a number of previous positions. While this helped avoid any irregularities in motion caused by change in the robots position, it did occasionally cause the robot to not change positions at peaks of motion at higher smoothing. When taking data, however, due to performance issues with the robot's arms being damaged and the hip joints overheating when being used to move the entire torso, only the legs could be used reliably. As such, the smoothing sample could be set as low as the previous two positions and the robot would still move effectively.

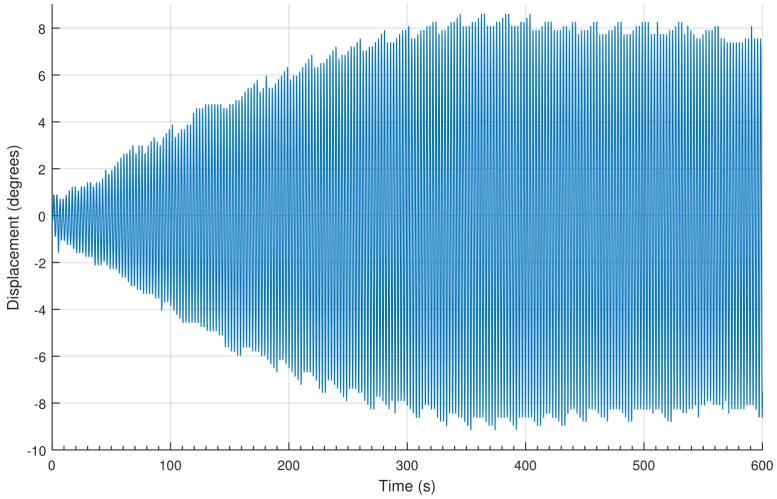


Figure 5.5: Performance of vision feedback after a near minimal push off.

Figure 5.5 shows a test of the vision tracker for feedback. The method of testing this was as follows. The robot was set up with two landmarks held 1.12±0.01m away and at a height of 0.22 and 1.11 (±0.01)m respectively. The encoder was then set to track the angle (with a resolution of 0.2°) every 0.02s for 10mins. When the encoder began to track the swing was completely stationary with the robot in a seating position. After the zero position of the system was recorded by the robot it was given a gentle push to about 1° and then left to swing using just the feedback system, which was started just after the robot was given a push. If the robot had no initial movement then the robot would not be able to tell it was moving and would remain at its current pose. The robot was then left to swing under just the vision feedback system. As can be seen, it takes around 5 minutes for the robot under vision feedback alone to reach a peak amplitude of 8.6±0.2°, after which it slowly beings to decrease due to overheating in the joints slowly decreasing the efficiency with which the robot can move. The robot would sometimes miss a peak and not switch positions which can be seen on the graph as a small drop in amplitude. Despite this, this graph as well as our other tests show that the vision tracker is a viable if sometimes sporadic method of providing feedback based movement, that would work well if coupled with another kind, such as gyrometer feedback.

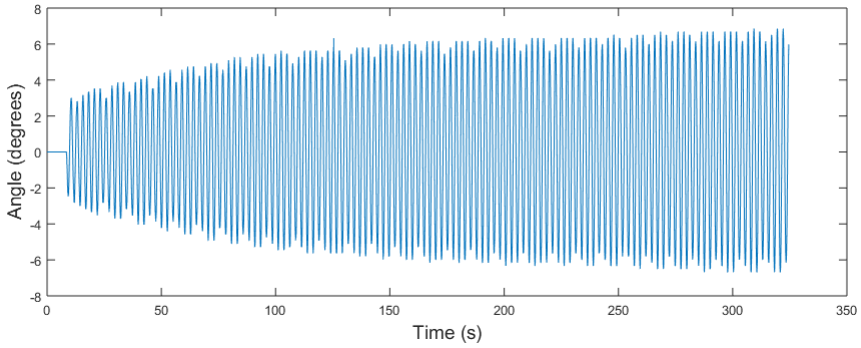


Figure 5.6: Gyrometer feedback swing from a small push.

5.4 Gyrometer Feedback

5.4.1 Method

The following was contributed by: H. Jacobs

The tests in Section 2.4 showed that the Y gyrometer was the best inertial sensor to use for feedback swinging. The base gyrometer readings gave a periodic sinusoidal measurement that lead the encoder angle by $\frac{\pi}{2}$. Using this information, code was designed to interpret the gyrometer measurements so the robot would be able to know when to swing.

The code recorded the maximum and minimum values of the gyrometer during a period and then work out the average amplitude. On each loop, the position of the swing through its period is calculated using the following inverse trigonometric function: arcsin(gyrometer average/gyrometer amplitude). The function models each swing as a sinusoidal function and allows the robot to work out how far through the swing it is using its current moving gyrometer average. For the greatest accuracy it was decided that a moving average of the gyrometer data would be used rather than the raw value which would be updated every 20ms. This would prevent small perturbations in the swing movement from accidentally triggering a swing motion by smoothing out the data. This would require the use of multi-threading so the gyroscope data was updated as quickly as possible whilst another thread would then ask the robot to move. Ideally the gyrometer data would update quicker, so that the data is smoother, however this slowed down the robot to the point where other functions would not execute, including simple terminal commands over WiFi.

Two modules on the robot were created, one which would handle the gyrometer data and another that would request the movement. A module is a C++ class compiled as a library that can be automatically loaded with the NAOqi processes on the robot [41]. The first module, `GyroEvent`, is designed to read the gyrometer value and then based on the sinusoidal nature of the graph in Figure 2.10 inform the second process if it should move. The second `SubscribeGyro` would execute the movement function when asked to by `GyroEvent`. This process was done using the NAOqi API which allows a module to use the `SubscribeToEvent` function and assigns a callback to the event. The event is defined in GyroEvent module and called through the *raiseEvent* function passing a value. When called, the NAOqi process starts all callback functions that are currently subscribed to the event, passing through the value.

Both modules would record the last time that the event was called or a movement was requested. This was so that duplicate movements requested from similar values would not be performed.

5.4.2 Results

The robot was start with a small push which would be equivilant to a human having a slight momentum as they begin their swing. This can be seen in Figure 5.6. From this starting point, the robot was able to increase its amplitude significantly to around $6.9 \pm 0.1°$. The robot was unable to start from rest and merely shifted from one small extreme to another without any net momentum gain.

Figure 5.7 shows that the robot was able to sustain its amplitude if started from the maximum found from 5.6 as it had reached a steady state. This state was also reached if the swing was released from

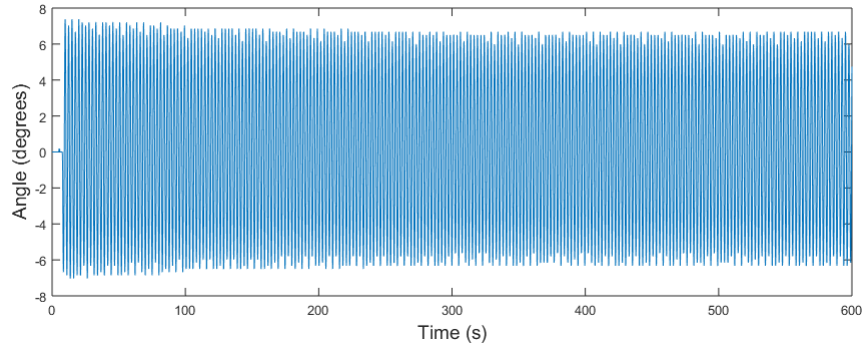


Figure 5.7: Gyrometer feedback steady state.

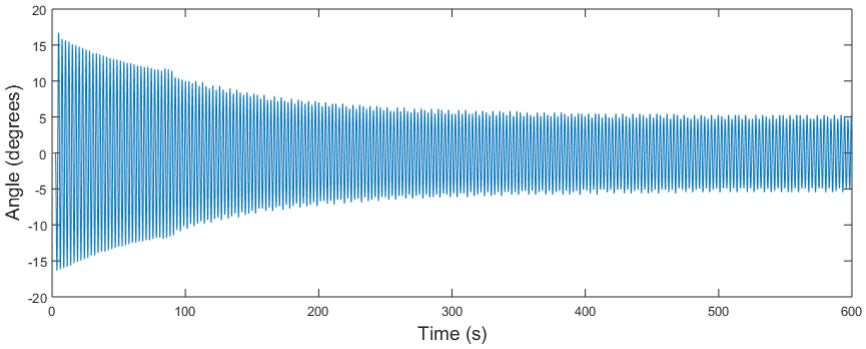


Figure 5.8: Gyrometer feedback decay from large push.

a large amplitude as is shown in Figure 5.8. This shows that the highest amplitude possible from this method using only the legs is just under $\approx 7°$.

5.4.3 Conclusion

It can be seen from these results that the gyrometers are effective in being able to produce a good swinging motion from near rest. There are, however, a few things to note, the first being the effect of movement of the robot itself. As the robot's right arm was damaged, the robot's motion was limited to the legs with the torso remaining stationary in the robot's reference frame. This means that the motion of the robot did not affect the gyrometer values. Future work would require that a test be applied on the gyrometer data for a fully functioning robot whereby excessively large values from the gyrometer are ignored. Instead, the periodic nature of the swing should be used whilst the gyrometer data is unusable. This is necessary as Figure 5.9 shows the high spike on upper body movement during a test run of the maximum and minimum extension before the robot was damaged.

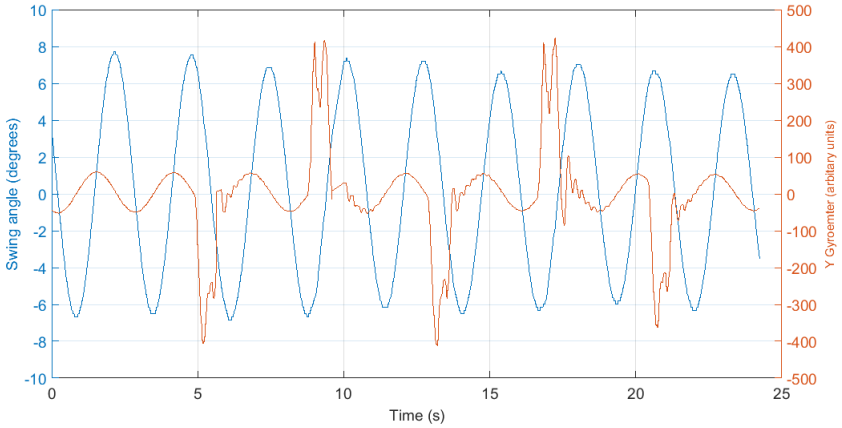


Figure 5.9: Extreme Gyrometer spikes on upperbody movement.

During the swing, the robot would undergo a cycle where its movements would gradually go out of phase with the swing until it would move when it was very close to the zero angle. At this point the robot would recalibrate and start the cycle again with movements in phase with the swing. Future work should be done with the code and the gyrometer data analysed so that in future the robot will stay in phase. This will also increase the maximum amplitude of the swing even with only the leg movement.

5.5 Chapter Conclusion

The following was contributed by: G. Hazar

Overall three different methods of using feedback were used successfully to make NAO swing. The first experiment carried out with the Phidget found that using a periodic driving force is more effective than the encoder feedback at smaller angles and when trying to increasing the amplitude from rest. This can be seen in Figures 5.2 and 4.3 where the maximum amplitude reached using the encoder was $6.3 \pm 0.2°$ compared to $7.0 \pm 0.5°$ for the predetermined driving force. This is mainly due to the significantly large relative error of the encoder at small angles which make it difficult to determine the peaks of the motion accurately. Evidence supporting this can be seen in Figure 5.2 where is it clear that the data has quantisation issues due to the low precision of the encoder. One advantage of using the encoder which can also be seen in Figures 5.2 and 4.3 is that it was able to produce oscillations that lacked beats and also had a more consistant amplitude. Furthermore using the periodic driving force required a lot of time consuming experimentation to determine the optimal driving period, whereas the encoder did not.

The second feedback method used was the `ALLandMarkDetection` vision module, which worked by determining the whether the robot was moving forwards or backwards from the relative position of landmarks in its field of view. Unlike the encoder feedback the vision was able to start swinging virtually from rest, a small displacement of $1 \pm 0.2°$ was required, which is equivalent to a human providing themselves with some momentum before they start swing. As shown in Figure 5.5 the method produced a rapid increase in amplitude; reaching over 75% of the maximum amplitude in under 2 minutes. Furthermore the vision module feedback was able to produce a larger maximum amplitude of $8.6 \pm 0.2°$. One of the drawbacks of the method was that it provided little information on the position of the robot during swinging, as it could only inform the user on whether the robot was moving forwards or backwards. This limited our capabilities to test the accuracy of the method quantitatively as we unable to determine how close to the true peaks and troughs NAO was preforming the motions. Another limiting factor of `ALLandMarkDetection's` usefulness was its inability to identify land markers when the torso and head was moving which lead to gaps in the data collected. However this limitation did not affect the robot ability to swing as it only needed continuous data at two points during the motion (the minimum and maximum displacement).

The final feedback method tested was the gryometer. One of the advantages of the geometer over the vision is that it provided data on the position of the robot during the swinging. This was done by recording the maximum and minimum values of the gyrometer during each swing and then using

trigonometry to convert this along with its current reading into a position value. Like the vision feedback, the gyrometer was able to start virtually from rest and produce a significant increase in amplitude to 6.9 ± 0.1. Again like the vision, the gyrometer data output is effected by the movement of the robots torso as shown in Figure 5.9. The movement of the torso would lead to sudden spikes in the gyrometer values that would need to be accounted for with a fully functioning robot.

Finally the accuracy of the position data produced was limited by the rate at which the gyroscope updated. Faster updates would have smoother results however it not possible as this would have slowed down NAO's other primary functions.

Each method tested provided a unique approach to make the robot and also provided insight into different aspects of the swinging motion. The encoder was by far the most reliable method for determining the position of the robot. However its relatively low precision meant it was an ineffective method used from rest. The vision feedback proved to be the best use from rest, because it was not affected by the movement of the torso to the same extent as the gyrometer at small angles. Furthermore the simplicity of the vision feedback method also made it the most reliable method and less prone to code crashes. The gyrometer feedback was shown to be the middle ground between the vision and encoder, as it could provide data on the robots position on the swing and like the vision it was also capable of starting from rest.

To obtain the optimal swing motion a combination of all three methods must be used. The vision feedback is capable of providing a rapid increase in amplitude from rest. Once a constant amplitude was reached switching to the gyrometer feedback would provide a further increase in amplitude. The encoder should be used in parallel with the gyrometer to provide confirmation of position values, making the system as a whole more reliable.

Chapter 6

Machine Learning

6.1 Introduction

6.1.1 Theory Behind Machine Learning

———————————The following was contributed by: B. Stokes ———————————

Machine learning is a broad field that encompasses all algorithms and methods that allow a program to solve a problem that it has not been programed to solve explicitly. Essentially, any problem can be thought of as a (typically very large or even infinite) solution space encompassing every set of actions that form a solution that an agent can perform. Within this space lie one or more optimal solutions that solve the problem the most successfully. Machine learning starts from a point of knowing little or nothing about the solution space and aims to place an agent somewhere in this space and program it such that it can iteratively move towards an optimal solution. This generally requires the program to perform informed guesses as to the solution of the problem and then use some kind of reward metric provided by the programmer to evaluate how successful it has been. The success (or not) of previous guesses is then used to inform subsequent ones.

The field of machine learning can be roughly divided into the following three categories:

Supervised Learning: In this category the program is provided with a set of accurate inputs and their corresponding outputs and attempts to map future inputs onto outputs consistent with its example ones. The algorithms are used to search the space of functions that map inputs onto outputs. The reward of these algorithms is based on how correctly the programs predictions match with the results of future inputs.

Unsupervised Learning: In this branch of machine learning no labelling is attached to the data provided and the program is tasked with finding patterns in the data. No reward function is used in this area of machine learning so the program must progress with no error feedback and very limited knowledge of ow successful it has been.

Reinforcement Learning: Reinforcement learning problems place an agent capable of performing a set of actions "A" in an environment whose state is parametrised by the variables "S" and task the agent with achieving some objective within the environment. Generally, the agent uses the state transitions that certain A cause from certain S, along with a reward function that gives the reward associated with each S, to navigate the environment and collect as much reward as possible. This branch of machine learning is the one that is the most applicable to the robot on the swing.

6.1.2 Why Reinforcement Learning

The following was contributed by: S. Rowlinson

Upon considering the different types of machine learning techniques outlined in Section 6.1.1, it was decided that the most suitable approach for the robot-swing system was to use a temporal difference based reinforcement learning algorithm (often referred to as Q-Learning or model-free learning). One of the primary reasons behind this choice was the benefit of not requiring additional information about the system in the form of transition probabilities from state to state as is required for model-based reinforcement learning [42]. It should also be noted that both supervised and unsupervised learning algorithms were rejected relatively early due to the unsuitability of these areas for our system. In the case of the former, this technique requires inputs to the system from an external agent (a "teacher") in order to make decisions on the optimal actions to perform - this would add extra complexity to the machine learning process as any input action to the robot-swing system would need to be parsed, interpreted and analysed. Whilst the latter case is generally more concerned with using algorithms to find patterns and structure in a set of existing data.

The principle advantages behind not requiring transition probabilities are that these values can be difficult to estimate, which is pertinent to the robot-swing system as there is no simple approach to determining what the probabilities of moving between given states may be. Additionally, these transition probabilities require a larger quantity of trials to learn in the algorithm employed. Assuming a system with n states and m actions per state, a total of n^2m values would be required for the learning process in *model-based* algorithms (due to the requirement to store probabilities in each state as well as the m actions) whereas for *model-free* algorithms only nm values are required for the state-action utility pairs as these transition probabilities are not required - i.e. model-free reinforcement learning reduces the required "state-space" by a factor of the number of states from model-based reinforcement learning; a significant reduction for the robot-swing system with small discretisation steps.

The core routine behind the Q-learning algorithm (used in our program) is shown in the utility iteration equation below [6],

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha \left[R(s') + \gamma \max_a Q_i(s',a) - Q_i(s,a) \right], \tag{6.1}$$

where $Q(s,a)$ represents the utility (or Q-value) of a state-action pair, α is the learning rate, γ is the discount factor and $R(s')$ is the reward of a next possible state s'. The learning rate ($\alpha \in [0,1]$) determines the relevancy of new information received over old information experienced, lower values indicate previous information is preferred whilst higher values give higher priority to more recent data acquired. To regulate the importance of future states the discount factor ($\gamma \in [0,1]$) is used, lower values of this coefficient result in "short-sighted" behaviour (the robot would perform actions with preference to immediate results) whilst larger values allow the agent to "plan for the future" such that actions performed take longer time frames into consideration. The choices of these parameters will be discussed in Section 6.2.1 along with the importance of the action-max term in Equation 6.2.

6.2 Machine Learning Library

6.2.1 Main Algorithm

The following was contributed by: C. Hogg

Q-learning requires a data set containing all the possible states of the agent in its environment where each state contains an agent's experiences. Each experience stores an action the agent performed and what utility the agent had gained from doing that action. With these in place the agent follows an algorithm until it has gained enough experiences to determine the best actions to take in each state.

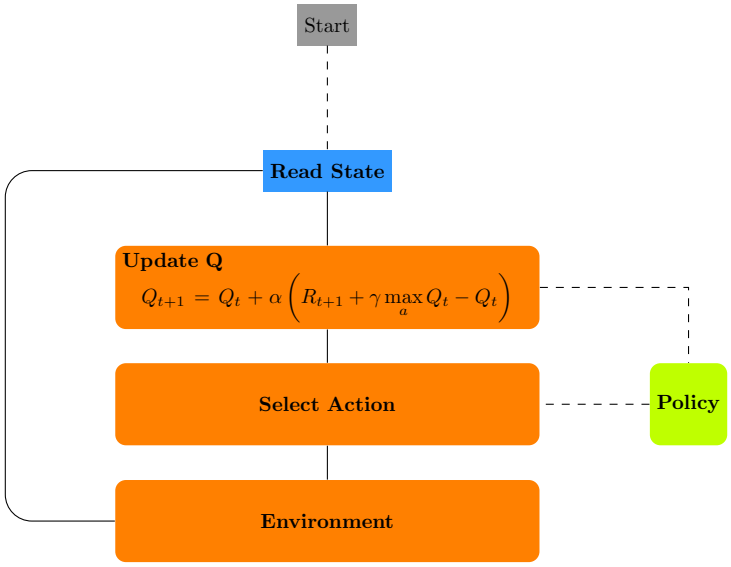


Figure 6.1: Flowchart of Q learning process.

Figure 6.1 shows the process that q learning follows. First, the agent does a random action from an initial state, this is due to the algorithm requiring knowledge of both the agents starting state and the state it has ended up in. After this, using the sensors it has, the agent records what state it is in after the action. Using information from the experiences of the two different states it then updates the experience of doing the action by changing the utility based on the update Q equation. After the new experience for the old state is calculated it then chooses a new action based on the experiences of the utilities of the new state by enacting a policy which will be explained further in Section 6.2.4. The agent then does the chosen action and the process repeats.

The update Q equation is the method by which the utility of a particular experience is calculated. In its most general form it comes out to be,

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha\left[R(s') + \gamma \max_a Q_i(s',a) - Q_i(s,a)\right]. \tag{6.2}$$

where $Q(s,a)$ is the utility of a particular state s and a particular action a. $R(s)$ is a reward function based on the state s, α is called the learning rate and γ is the discount factor. The equation calculates the updated utility of the old state after doing action a by taking the old utility of the old state after doing action a and adding the value of the reward function of the new state the agent had reached and the highest utility within the experiences of the new state whilst taking away the utility of the old state after doing action a.

The entirety of the equation after the initial utility of the old state and after doing action a is weighted by the learning rate. The learning rate determines how much the new information will override the old information. This can hold any value between zero and one. Zero corresponds to the agent not learning whilst one corresponds to the agent always prioritising the new information. In the case that a particular action from one state results in the same state, a learning factor of one is optimal, however in the case of a more random final state it is better to have a lower learning rate to effectively average out the differences.

The highest utility within the experiences of the new state is weighted by the discount factor, this determines the importance of future utilities. Again the discount factor can be any value between zero and one. In the case the discount factor equals zero the agent would show short sightedness as it would not take into consideration the utility it could gain from producing an action in the new state, whilst a discount factor of one means that the future action takes an extremely high precedence. Having a discount factor too low could make the agent avoid doing an action because it is not instantaneously beneficial whilst it could be more beneficial in the long run, however having it too high could cause the results to diverge due to the future states taking a higher precedence than the reward.

6.2.2 Priority Queue

The following was contributed by: S. Rowlinson

One of the pivotal data structures of the Q-learning algorithm was the custom coded `PriorityQueue` template class; written in `C++`. This class was implemented using both Minimum and Maximum Binary Heaps (MBH) - structures which store data items in a binary tree configuration whereby the "key" (i.e. priority) of each node in the tree is less than all child node keys in a minimum heap and greater than all child node keys in a maximum heap; this is demonstrated by Figure 6.2. The reason for implementing the priority queue through a heap structure is purely due to efficiency benefits when enqueueing objects into the queue, dequeueing items from the root of the queue and performing other necessary operations such as altering the priority of a specified object in the queue instance. For example, the worst-case time complexity of inserting an item into the queue is $O(\log_2 n)$ where n is the number of entries in the queue when using a binary heap as the underlying structure. This is much more efficient than the $O(n)$ time complexity for insertion operations if an ordered linked list was used instead; especially for queues containing large numbers of elements. Most importantly, the maximum/minimum priority data element in the queue is stored at the root of the binary heap (maximum/minimum heaps, respectively) - thus obtaining the value of the item stored at this node is evaluated in constant, $O(1)$, time; i.e. the time complexity of this operation is not dependent upon the size of the queue giving a highly significant efficiency saving for large priority queues.

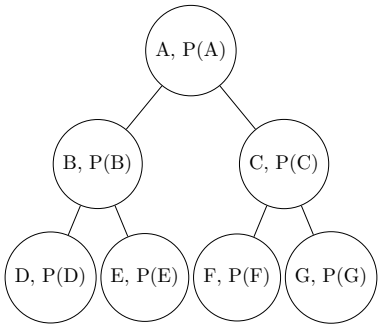


Figure 6.2: Schematic of a binary heap, the underlying structure used for implementing the priority queue - each node stores a data item, X, and its associated priority, $P(X)$. In a Maximum Binary Heap the priorities, $P(X)$, of any given node are greater than the priorities of its child nodes in the heap; and vice-versa for Minimum Binary Heaps.

The main reason for custom coding this priority queue class rather than using the Standard Template Library (STL) implementation was the lack of flexibility and functionality of the latter - several operations such as searching for item-priority pairs or altering the priority of items in the queue (methods used in the machine learning algorithm as is outlined in Section 6.2.4) are not available to use with the STL version. Therefore a more general priority queue was written from scratch in order to obtain a data structure suitable for use with the Q-learning algorithm used. Additionally, the STL priority queue can only store a single type of data which prevents one from being able to store actions and their corresponding utility values in the queue. Therefore using the custom coded `PriorityQueue` allowed the storage of any data item along with a priority value associated with the data item thereby granting the ability to store action-utility pairs in the queue, which was vital for our machine learning algorithm.

The `PriorityQueue` was used extensively in the final machine learning algorithm in order to efficiently store and perform operations on action objects. These queue instances stored actions along with their corresponding priorities (given by the utility value) in a maximum binary heap configuration in order to continuously find the most suitable action to perform after altering the priorities of the possible actions based upon the current state of the robot. Using the priority queue structure in this case significantly decreased the overall time complexity of the whole algorithm due to the operation times considered above. Additionally, the use of this structure allows for further flexibility of the code by expanding the number of possible actions that the robot could take in any given state without compromising the run-time efficiency of the solution.

6.2.3 Space Parametrisation

The following was contributed by: B. Stokes

In any machine learning problem it is important to correctly parametrise the space in which the agent is acting. In q-learning in particular the agent has to be able to store memories of the results of its actions in previous states. For this, a set of variables must be chosen such that every point in the state space can be uniquely identified. In the case of the robot on the swing it was decided that the state of the robot was described at any given time by its angular position, angular velocity and the positions of its arms and legs. To simplify the problem the robot was limited to the two states of torso backwards, legs forwards and torso forwards, legs backwards with the two actions the robot could take being changing its current position to the opposite one and remaining in its current position. This greatly reduced the number of states that the robot would have to search to find a reasonable solution. It was hoped that this model would provide a relatively simple initial machine learning problem which, if successful, could be made more complex.

To physically store the memories in the program, an object was created to conceptually embody the state space. This state space object stored a pair of two dimensional `std::vectors` representing the two dimensions of angle and velocity in each of the two robot positions. Due to the discrete nature of storage in a program, the dimensions had to be limited and partitioned in bins. To achieve this the object was programed with several parameters that allowed specification of the maximum absolute value of a dimension and the number of bins used to represent the dimension when the object was instantiated. At each of the "points" in the space a priority queue (see Section 6.2.2 for details) was stored. There was one entry in the queue for each action that could be performed in a state and the priority of each action corresponded directly to the utility that the robot had gained from performing that action in that state previously. Because of the way vectors are stored in memory, once this object was setup it was very quick to access the priority queue of any state that the robot found itself in. To allow a zero initialisation of the state space's utility values, an additional parameter was added to its constructor to allow an initial queue to be passed that would be copied to every point in the space during construction. Finally, the subscript operator ("[ ]") was overloaded for the state space class. This allowed access to the class's priority queues by specifying the three space indices where the desired priority queue resided as if the class were a 3 dimensional array. The operators were overloaded such that continuous values could be used to index the state space. If such values were used the operators would use a simple transformation to find the bins corresponding to the values the subscript operators were called with. Figure 6.3 below shows the structure of the state space class.

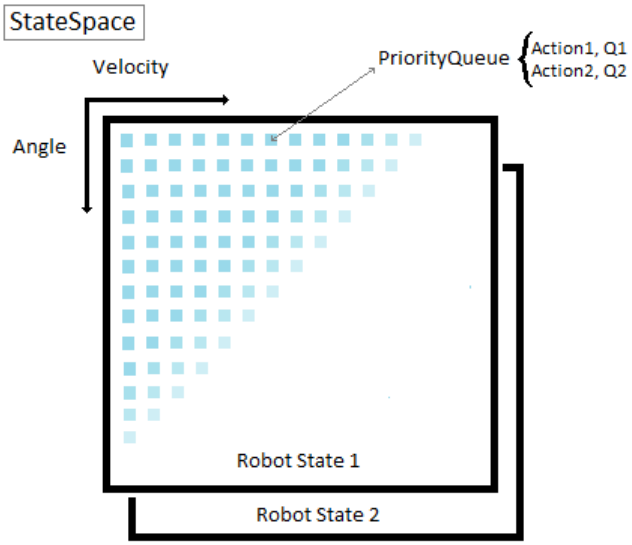


Figure 6.3: Conceptual layout of the state space object.

An object was also created to represent the states of the robot. This was a simple struct with member

variables storing the angle, velocity and limb-position of the robot. A reward function was created for this object to give the reward associated with the stored state as a function of the state variables. An additional subscript operator was added to the state space object so that the space could also be indexed with a state object instead of just raw numbers.

With these objects, the program had a basis upon which to store and represent the agent's state and to store the "memories" created by applying the q-learning algorithm.

6.2.4 Action Selection

The following was contributed by: O. Diba

At each state the agent arrives at it must pick a new action to perform, and it needs a procedure that utilises its past experiences so that it can progress in the learning task. The most obvious choice is to select the action corresponding to the highest utility, or if we denote this optimal action by a^*, it will be the action satisfying $Q(s, a^*) = \max_a Q(s, a)$. This approach is known as *greedy* selection since the agent tries to maximise its short term reward using this method. When an agent follows greedy selection the actions it chooses are called greedy actions, and it is said that it is *exploiting* its current knowledge [6]. If the agent has the correct values for the utility of every state-action pair, then this method must be the best way for the agent to maximise its reward or to reach its goal state. But generally, the agent has incomplete, or even unreliable knowledge of the utilities. Using greedy selection in this case would only favour the agent on the short term. It is necessary for it to choose non-optimal actions occasionally, so that it can better its estimations of these actions' utilities, and ultimately, find the best solution in the long term. When the agent chooses any non-greedy action, it is said that the agent is *exploring*. Thus, it is important that the agent strikes a careful balance between exploitation and exploration to efficiently solve a learning task.

There are two standard ways of programming the agent to incorporate some exploration into its behaviour. The first method, ϵ-greedy, works by ensuring that there is a small probability ϵ that the agent will pick an action at random, and then the rest of the time the agent uses greedy selection.

$$\text{selected action} = \begin{cases} \text{random action} \quad \text{in} \quad \{a_i\}, & \text{if} \quad \eta < \epsilon \\ a^* = \arg\max_a Q(s, a), & \text{otherwise} \end{cases} \tag{6.3}$$

where $0 \leq \eta \leq 1$ is a random number. This method is often sufficient, but there may be scenarios where the low utility actions should really be avoided, and a method is needed that discriminates between utilities more. Such a method is called softmax selection [6]. It works by selecting actions with probabilities that increase with utility. A particular type of softmax selection, and perhaps the most commonly used, is Boltzmann exploration, where actions are selected according to the Boltzmann distribution. The probability that any action is chosen is given by [6],

$$P(a_i|s) = \frac{e^{Q(s,a_i)/T}}{\sum_k e^{Q(s,a_k)/T}}, \qquad i = 1, 2, \ldots, n. \tag{6.4}$$

The parameter T is called the temperature, in reference to the absolute temperature of classical thermodynamics, and it decides the shape of the distribution. As $T \to \infty$, actions are selected equiprobably, with no dependence on their utility, conversely, as $T \to 0$ only the greedy action is selected. The downfall of this method is that it requires more computation and is more subtle than the previous; it is quite hard to set the temperature to an appropriate level.

Functions were written for the Q-learning code to select actions, implementing both the ϵ-greedy and Boltzmann exploration methods. The ϵ-greedy function simply takes a `PriorityQueue` of actions, and returns the selected action. At the start of the function, a random number in the interval $[0, 1]$ is generated (seeded at the beginning of the program), then an if statement checks if this random number is between 0 and ϵ, which is defined in `main`. If it is, then another random number is generated to randomly pick an index in the `PriorityQueue` of actions, and the corresponding action is returned. Otherwise, the function returns the first item in the queue - the action with the highest utility. The Boltzmann exploration function, calculates the normalised Boltzmann factors for each action in the `PriorityQueue` of actions, according to Equation 6.4. It then calculates the cumulative probability of this list of action-probabilities, this effectively gives each action a bin with a width equal to its probability. A random

number is generated and the function iterates through the vector of pairs, until it finds an action with a cumulative probability greater than itself. This action is then returned. This method ensures that each action is picked with the correct probability.

The action-probabilities are stored in a `vector` containing elements of type `pair`, the `pair` object itself holding the action-probabilities. Both of these containers are types provided by C++'s standard library. They were chosen for the efficient accessibility of their elements. Originally, the values were put into a `PriorityQueue`, but it was found that the queue would try and reorder itself before the cumulative probabilities had been calculated, which interfered with the calculations.

It was decided that temperature should initially be set high to encourage exploration, and be steadily lowered as the robot converges towards a solution. To achieve this the temperature was set by a Gaussian function,

$$T_i = T_0 \, \mathrm{e}^{-i^2/2\sigma^2} + \delta \tag{6.5}$$

Where i is an integer, initially set to zero, which is incremented by 1 upon each iteration of the Q-learning algorithm. T_i denotes the temperature after the ith action is performed and T_0 is the initial temperature, which is set high. The standard deviation, σ, determines the exploration 'time' - for how many iterations exploration dominates over exploitation. A small number δ is added to the Gaussian to prevent the temperature from getting to close to zero, which would cause computational errors in the calculation of the Boltzmann factors. Unfortunately the efficacy of this procedure is not known as it was not properly tested.

6.3 Inverted Pendulum Simulation

6.3.1 Method

———————The following was contributed by: Z. Hodgins ———————

The aim of the inverted pendulum test was to investigate the capabilities of the machine learning library before it was put onto the robot. The inverted pendulum is one of the traditional machine learning problems that has been very thoroughly investigated in the past. It was decided to computationally model a single pendulum with a motor as the pivot; the "machine" must learn how much torque to apply to balance the pendulum vertically. The simple system can be seen in Figure 6.4. This investigation was done to test the library whilst minimising the differences between the program used on the robot.

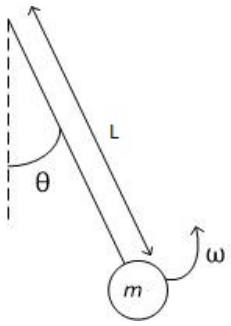


Figure 6.4: The system used for the inverted pendulum simulation.

The biggest difference was the change made to the action vectors. The library was written for a swinging robot which can only perform two possible actions: move forwards or more backwards. Whereas, the motor can apply a large number of different torques over a given range. These torques were discretised into distinct actions that the motor could perform. By the nature of the inverted pendulum problem, the motor must not have a torque high enough to lift the pendulum vertical in one movement, so a maximum torque was chosen which was smaller than the holding torque required to hold the pendulum horizontal. In this model, the pendulum was taken to be 8cm long and the end mass was 0.5 kg. There are no errors on these values, however, there are errors in computing accuracy and in the numerical methods used to solve the equations of motion, even if there aren't errors in the theoretical set-up. These values were

chosen so that if the computer simulation of the pendulum worked, the code could be applied to the Phidget (see Section 4.2 for more details). By approximating gravity as $10\,\mathrm{kgms}^{-2}$ the maximum torque of the pendulum was found to be 4 Nm. This can be applied in both directions, so for simplicity, the possible actions were split into nine movements, applying torque between -4 Nm and 4 Nm increasing in integer steps.

The second main change that was made to the library was the use of trials. The robot could continue learning as it would always be in the swinging cycle. If the motor applied large amounts of torque in quick succession, the pendulum would rotate in a chaotic fashion. The inverted pendulum was also a two part problem, the pendulum must learn how to swing up from its initial position, and it must be able to apply small corrections to keep the pendulum at the top of the swing. Without trials, the pendulum may swing up and stay up and so will not continue exploring the best method of swinging up. To stop this happening, trials are introduced. Each trial lasted for ten seconds, it was stopped before that time if the angular velocity became greater than $2\pi\,\mathrm{rad\,s}^{-1}$. The results of that trial were recorded and a new trial starts with the pendulum reset to its initial position, $\theta = 0, \omega \equiv \dot{\theta} = 0$.

Other small changes that were made to the library included changing the State and `StateSpace` classes to involve the extra variable, torque. Finally, a new class was made called `Environment.h`. The `selectaction` function chooses a torque from the Q-table, this value is input to the `propagate` function in the `Environment` class. The `propagate` function performs a Runge-Kutta integration on the equations of motion of the pendulum, which we solve for θ and $\omega \equiv \dot{\theta}$ by applying the given torque over a time $\mathrm{d}t$. The equation of motion for this pendulum is the standard equation for a driven oscillator (see Appendix E). These values are updated at the current state and the process is repeated. The class also contains the function `resetPendulum` which initialises the variables after a trial has finished. The values for each of the variables were output into a text file and then animated in MATLAB using the script `inverted_pendulum_animation.m`, this script also saves the animation as a `.avi` video file and it can be found in Appendix E.

6.3.2 Results

———————The following was contributed by: B. Stokes ———————

The pendulum simulation was run a great number of times and, during this process, a great many utilities were added to the program to allow the monitoring and altering of it's variables. The initial runs of the simulation showed some promise - if the pendulum ever reached the top at a relatively low speed it would make a visible attempt to arrest its motion. However, there was a greater tendency for the pendulum to decide that spinning round continuously and in one direction as fast as possible was the best course of action. The reason can be seen from the first reward function used of $-\cos\theta$. Not only is this function quite flat at the top of the loop, giving a similar reward for all positions near the top, but it places no constraints on the angular velocity of the agent. Thus the agent eventually works out that if it swings past the top the fastest way to get back to the top is to continue swinging round as fast as possible. It then misses again and enters a cycle of continuously torquing the pendulum in one direction and is not penalised for doing so. Whilst this result was not ideal, it did at least show that the program was successfully building up a picture of it's state and navigating it to get the most reward.

The task then became a problem of selecting the best action function to cause the agent to find the strategy of greatest reward to be balancing the pendulum in the upwards position. Numerous functions were tried. Two of the most successful functions were one that was dependant on the energy and another that consisted of a single spike of reward for being at low velocity and $\theta \approx \pi$. Functions dependant on only high θ and/or low ω tended to do poorly. In the former case the agent would simply torque the pendulum permanently in one direction in an attempt to gain height and end up oscillation near the bottom. In the latter case the agent would never acquire enough velocity to get itself to the upright position. Unfortunately even the best reward functions could not get the pendulum to balance.

The simulation could be improved by more finely partitioning the space so that the agent can gather more information about how to get to the top or by increasing the number of available torques to give the agent more control. Finally, if the library were ever upgraded to use continuous rather than discrete states this would undoubtedly give better results.

6.4 Outlook

The following was contributed by: C. Hogg

Due to time pressures and the complexity of the task, we had to simplify much of the program in order to produce any results. This left us with limitations within the program. One of such was the discretisation of the states. In both the case inverted pendulum and the robot, all components of the states were separated into bins. This meant that each action in a particular state would not always result in a same final state. This randomness in the final state meant that the learning rate had to be decreased, increasing the time required for the machine learning algorithm to produce enough data to work out the best motions. This could be improved by either increasing the size of the arrays to decrease the randomness or by investigating a way for the states to be continuous.

The discretised action values also meant that we had to predetermine what actions the agents could take, this in itself caused some problems. In the case of the inverted pendulum the discretised torque meant that the pendulum could not fine tune its position with fractional torques, meaning we had to choose torque values that made it possible for the pendulum to balance without consistently over shooting. Meanwhile in the robots case the actions were predetermined to be moving to a lying back position or moving to a sitting up position. These initial parameters meant that the robot was choosing between actions that were known to be the most optimal. Increasing the amount of torques the motor could produce in the inverted pendulum model, it is likely that the pendulum would be able to balance in less tests and be able to remain upright for a longer duration, whilst in the case of the robot, increasing the amount of actions would allow the robot to investigate new forms of swinging and perhaps find an unexpected solution which could be investigated further to improve our understanding of swinging.

The reward function is a pivotal part of the update q function and determines how the agent learns. However in both the inverted pendulum and on the robot, we could not be sure that our chosen reward functions were the best choice for the agent. The only way we could check them is by repeating the tests and seeing how well the agent learnt given a particular reward function. In the case of the robot, repeating multiple tests was not possible for us due to the demand of the use of the robot, however working on the inverse pendulum has allowed us to refine our reward function however it is still not perfected.

The values of the learning rate and the discount factor were obstacles for our machine learning. These values had a large impact on the rate and accuracy of the learning. However we were unable to determine the particular values to optimise the learning, thus we were left to trial and error, this paired up with the time taken for each individual run and the uncertainty of the reward function, meant we could not repeat the test enough times to settle on a value. This could only be improved by a deeper understanding of the origins of these values or more time for trial and error.

6.5 Chapter Conclusion

The following was contributed by: S. Rowlinson

Unfortunately due to high demand for the robot as well as the issue of overheating of the joints mentioned in Section 2.3.3, there was not enough time to perform a thorough execution of the machine learning algorithm we built up detailed in this chapter. However, there were other means for testing the machine learning library through the use of numerical modelling in order to simulate an inverted pendulum (detailed in Section 6.3) as well as a basic model of the robot using a simple pendulum system with the Q-Learning algorithm applied to it. This latter case was achieved via the help of the GNU Scientific Library (GSL) which provided the necessary Ordinary Differential Equation (ODE) solving routines for implementing the physical characteristics of the system. Figure 6.5 demonstrates a single execution of this simple pendulum system with our machine learning procedure applied to it, over a time frame of 900 s (i.e. 15mins) which was incidentally the time taken for the robot to significantly overheat shown in Section 2.3.3.2.

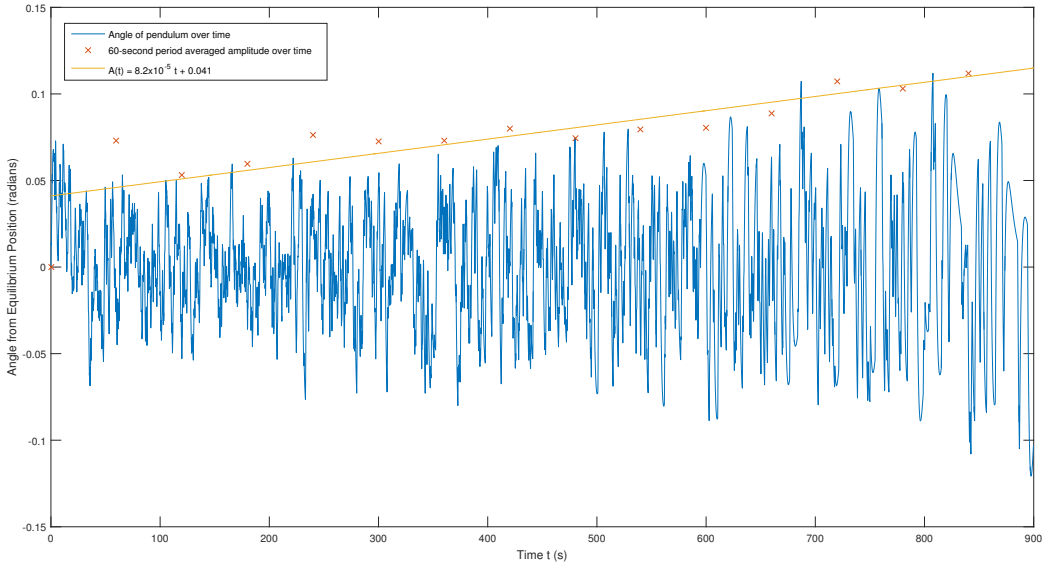


Figure 6.5: Plot of machine-learning simple pendulum displacement over time with minute-long averaged amplitudes showing a steady linear increase in the pendulum amplitude over the time frame of the simulation.

The machine learning algorithm used for this pendulum simulation was identical to the procedures we applied to the physical robot itself insofar as the basic Q-Learning features are concerned - the same structures such as the `PriorityQueue`, `StateSpace` and `State` classes (see Sections 6.2.2 and 6.2.3, respectively) were used as well as the epsilon greedy action selection procedure detailed in Section 6.2.4; additionally the reward function applied to the system was the principle of energy maximisation used for the robot shown in Section 6.2.1. The only difference was the, aforementioned, use of GSL to simulate the physical system and thus provide a way to access angles and angular velocities of the pendulum during run-time whereas the encoder data was used for the physical robot machine learning program.

There are several interesting features to note from Figure 6.5, the first of which is the sharp increase to the displacement angle of the pendulum immediately after $t = 0$ (where the pendulum was started at equilibrium with zero velocity) - this shows that the learning algorithm decides that the best initial action is to apply a large force consistently in a single direction in order to gain a significant initial amplitude. Another feature of this plot is the erratic behaviour of the pendulum for $t \lesssim 600\,\mathrm{s}$ indicating that the early stages of the learning process result in a large amount of exploration of the state space (i.e. a higher degree of stochasticity) which is to be expected from this program as the epsilon factor for selecting actions was initialised to a value of zero and then increased linearly to a final value of 0.9 by the end of the simulation in order to encourage a higher exploration factor initially followed by more exploitation (i.e. exploiting knowledge gained about the system through the Q-Learning algorithm) later on. Further evidence of this can be seen for $t \gtrsim 600\,\mathrm{s}$ on Figure 6.5 where the motion of the pendulum is smoother - undergoing full oscillations through higher amplitudes.

A final interesting aspect of Figure 6.5 is the equation describing the increase in amplitudes (averaged over minute-long periods) over the time of the simulation, given by,

$$A(t) = 8.2 \times 10^{-5} t + 0.041, \tag{6.6}$$

where $A(t)$ is the amplitude as a function of time, in radians, and t is the time. One can clearly observe from Figure 6.5 that the amplitudes steadily increase over the whole simulation run-time however the small gradient in Equation (6.6) shows that the rate of increase of the amplitude is relatively slow. This slow rate of increase could be attributed to a number of factors in this system such as the fact that the parameter epsilon was initially low giving a higher degree of randomness to the pendulum motion for small time scales, however one of the major contributing factors is likely due to the issue of relatively

slow-convergence of the Q-Learning algorithm itself to an acceptable "solution" [6].

In conclusion, it has been shown that the machine learning algorithms designed for this robot-swing system were successful insofar that the programs coded were able to make progress with learning about their respective systems (such as the inverted pendulum, Section 6.3, and the simple pendulum detailed above). As far as the machine-learning algorithm directly applied to the robot was concerned, it is clear that several issues arose such as the overheating of joints preventing long execution-times for the Q-Learning algorithm (as was required for reaching a suitable solution) as well as high demand for the robot for other tasks further inhibiting the ability to perform relatively long running machine learning tasks on the robot. There is further scope for future projects to easily build upon the work achieved during this section using the extensively documented code-repository created for these machine learning tasks [43, 44].

Chapter 7

Discussions and Conclusions

7.1 Discussions

7.1.1 Summary of Findings

The following was contributed by: Z. Hodgins

This project contained many different ways of getting the robot to swing, some being more successful than others. This process started by modifying the swing provided for the robot to use in a seated position. In order to do this, the properties of the robot were investigated. It was found that the robots range of movements and limb lengths meant that it could not hold the swing at the sides, so instead a front bar was added. After completing strength and motor overheating tests it was found that neither of these would limit the robots swinging motion; however the grip and pull tests found that the hands struggled to grip a bar. As a result, the robots hands were strapped to rotating grips on the swings cross bar. Preliminary tests were done on the robots internal sensors to assess their usefulness. It was found that the vision tracking software and the gyrometer gave the most helpful feedback, which could result in the robot swinging from their feedback.

The method of swinging was split into three sections: swinging from inputting predetermined functions onto the robot, swinging using sensor feedback and finally using machine learning to learn how to swing from no prior knowledge. Three main areas were investigated in order to create functions for the robot to swing from. This started by using a driven model to simulate the robots legs moving to calculate the highest amplitude this could reach. By setting the period to different values and fitting the data, the maximum swinging amplitude was found to be $7.0 \pm 0.5^\circ$. This was a very high amplitude considering the driven model only represented the movement of one of the robots legs. Following that, mathematical models of the system were made. By modelling the system as a flail and a hinged flail, the equations of motion for the swing were found using Lagrangian methods. These analytical models were put into the simulation software, Webots, and both models resulted in a maximum amplitude of around 2°, even when started with a push. This was a very disappointing result, and because of this, time was not spent putting the functions onto the robot.

The way a human swings and why was thoroughly investigated. The motion was tracked using a software called Kinovea [28] for various swinging methods and the movement functions were calculated. When input into the Webots simulation, a swinging amplitude of approximately 11° occurred, independent of starting displacement. However, when these functions were coded into the robot, only an amplitude of 3.2 ± 0.1 occurred. This was most likely to be due to the idealised nature of the simulation and the lack of similarity between the robot and humans dimensions.

In order to swing from sensor feedback, the robot had to be able to read the sensor and know from this value when in the cycle to move. This method was initially tested using the driven model coupled with the reading from the angle encoder. This produced a motion with amplitude $6.3 \pm 0.2^\circ$, which was successful considering the driven model only represents one of the robots legs swinging. By then using the built in vision tracking software, LandmarkDetection, the robot was able to calculate whether it was

moving forwards or backwards. From this, the robot knew when to move, and this resulted in a motion with amplitude 8.6±0.2°.

The gyrometers in the torso of the robot were used to predict the peak of the motion and subsequently move at the relevant time. A drawback of this method was that only the robots legs could be used as the torso had to remain stationary, which is why the highest amplitude reached was lower, at 6.9±0.1°. Finally the angle encoder readings were sent to the robot and it swinging from this feedback resulted in an amplitude of 7.2±0.1°, which is very close to the one created when the driven model used the encoder feedback. A problem with all of the feedback methods was that the robot had to be started with a push.

Finally, a machine learning library was written specifically for the robot. By using Q-learning, the robot could calculate which movement would fit best to improve its swinging. Due to the nature of machine learning, this was a very complex task, and when the library was run on the robot, it would take many hours to learn. A simulation was attempted using an inverted pendulum system; a motor attempting to learn how much torque to apply to balance the pendulum upright. Unfortunately this never succeeded, but very good progress was made with this, which can hopefully be continued in future research.

7.1.2 Unexpected Results

The most surprising result was from the robot swinging using the feedback from the internal gyrometers. This was because a relatively high amplitude was reached considering this swinging was only from the leg movement. The human motion research found that swinging from just leg movements was not enough to sustain motion and it would eventually decrease to stationary. When using the gyrometers, after being given a push, the robot reached a steady state. This result shows that the optimum method of swinging for the robot is very different to that of a human.

To further support this are the results of the human motion on the robot. Equations of motion were found by tracking a humans swinging motion. When these functions were put onto the robot, the amplitude was more than 60% less than any of the other swinging methods. This is likely to be because of the highly different specifications of the robot compared to a person. For example, the NAO robot contains a much higher percentage of its body weight in its torso compared to a human and it's legs are much shorter. This means that the torque created when moving the legs and torso is significantly different for the robot compared to a human; this explains why the human motion was so unsuccessful when on the robot.

Another surprising result was when the equations of motion from analytically modelling the systems were simulated. The resultant amplitudes were the smallest from the whole project, even when the swing was started from a displacement. This is likely to be because the models were done in an idealised system with no frictional forces or movement limitations. The functions are constantly trying to maximise the acceleration of the swing, but this is not always wanted from a swinging motion. It is believed that the models are correct, but they need to be coupled with a second function that takes into account that at higher amplitudes that the acceleration doesn't want to be maximised.

7.1.3 Outlook

The biggest limitation during this project was the resources. The NAO robot was in very high demand and as a result not all the experiments could be completed within the time frame. Webots only had one license with the swing and robot, so only one simulation could be tested at a time. This meant that there was not much of a chance to optimise these motions. The robots capabilities limited the research, especially once the arm broke.

The vision trackers were limited by the frame rate of the built in camera. The readings had to be smoothed over many values in order to get a reliable result. Occasionally this smoothing would result in a "false" peak which meant the robot would move when it should not. The gyrometer gave good readings but it could only be used with the torso stationary. For future research, it is highly recommend that these two methods are coupled to find an optimum swinging motion. If the robot starts from stationary using a predetermined function, once a sufficient amplitude is reached, it can then switch to using the combined feedback of the vision sensors and gyrometer.

The machine learning section was very promising and it can be seen that in the inverted pendulum simulation, it is trying to learn. The biggest limitation of the machine learning was the time it takes to learn movements. In order to run the library on the robot it would require approximately eight hours of learning which is not possible as the robot tires after just thirty minutes of movements. However, if the library was optimised more, for example by changing the reward function and learning factors, the learning time could be decreased. Therefore, for future research this is thoroughly recommended.

7.2 Report Conclusion

This project was successful at exploring many different approaches for making the robot swing in a seated position. After preliminary tests into the suitability of the robots internal sensors, it was decided the cameras and gyrometer would be used. The robots properties were investigated and as a result the swing was modified to suit the needs to the robot swinging in a seated position.

The driven model using predetermined periods showed high amplitudes of $7.0\pm0.5^\circ$, but this could not be replicated using the other functions. The most successful method in simulation was the human motion functions (producing an amplitude of approximately 11°), but when implemented on the robot produced $54\pm1\%$ lower amplitude than the driven model. Likewise, swinging from the predetermined analytical functions also proved unsuccessful due to the idealised nature of the models.

Comparatively, swinging using the sensor feedback was very successful. The driven model showed that using the encoder feedback could be effective, and this resulted in the robot using this to swing to an amplitude of $7.2\pm0.1^\circ$. The vision trackers showed the best results after the data was smoothed or $8.6\pm0.2^\circ$, but the gyrometer in the torso also showed very successful results, reaching a steady state at amplitude $6.9\pm0.1^\circ$ when only using the legs to swing.

The machine learning research made great progress. A complete library was written and when tested it showed promise. Simulated systems did attempt to learn but never succeeded. However with small manipulations to the library, this could be successful in the future.

For future research it is recommended that the vision trackers and gyrometer are coupled together in order to optimise the motion. Also, the machine learning library needs more testing as many parameters can only be optimised from a trial and error basis. This project acts as a good basis for showing which approaches were most effective for making the Aldebaran NAO robot complete a swinging motion.

References

[1] David Silver and Aja Huang *et. al.* "Mastering the game of Go with deep neural networks and tree search". In: 529.484 (2016). DOI: 10.1038/nature16961.

[2] Joe Allen *et. al.* "*Year Three Group Studies: Teaching the Aldebaran Nao Robot to Use a Swing*". 2015.

[3] Elizabeth Cha, Klas Kronander, and Aude Billard. "Combined Kinesthetic and Simulated Interface for Teaching Robot Motion Models". In: (2015).

[4] B. D. Argall *et. al.* "A Survey of Robot Learning from Demonstration". In: 57.5 (2009), pp. 469–483.

[5] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement Learning in Robotics: A Survey". In: (2013).

[6] Richard S. Sutton and Andrew Barto. "Reinforcement Learning: An Introduction". In: MIT Press, 1998. Chap. 6.5: Temporal-Difference Learning. URL: https://webdocs.cs.ualberta.ca/~sutton/book/ebook/node65.html. (Accessed 2016-2-27).

[7] Doc.aldebaran.com. *H25 - Links NAO Software 1.14.5 documentation.* URL: http://doc.aldebaran.com/1-14/family/nao_h25/links_h25.html. (Accessed 2016-1-22).

[8] Doc.aldebaran.com. *H25 - Joints NAO Software 1.14.5 documentation.* URL: http://doc.aldebaran.com/1-14/family/nao_h25/joints_h25.html. (Accessed 2016-2-24).

[9] Doc.aldebaran.com. *Memory watcher panel — NAO Software 1.14.5 documentation.* URL: http://doc.aldebaran.com/1-14/software/choregraphe/panels/memory_watcher_panel.html. (Accessed 2016-1-28).

[10] Russell Smith. *Open Dynamics Engine Homepage.* URL: http://www.ode.org. (Accessed 2016-3-18).

[11] Cyberbotics. *Webots: reference manual - ode-open-dynamics-engine.* URL: https://www.cyberbotics.com/reference/ode-open-dynamics-engine.php. (Accessed 2016-1-28).

[12] Doc.aldebaran.com. *H25 - Motor Specification NAO Software 1.14.5 documentation*. URL: http://www.istec.org/wp-content/uploads/2013/11/Datasheet_H25_NAO_Next_Gen_EN.pdf. (Accessed 2016-2-24).

[13] J.Allen C.birkinshaw et al. "Teaching the Aldebaran Nao Robot to Use a Swing". In: 1 (2015), p. 70.

[14] Doc.aldebaran.com. *H25 - NAO Messages NAO Software 1.14.5 documentation*. URL: http://doc.aldebaran.com/1-14/nao/led_messages.html. (Accessed 2016-2-28).

[15] NAO Software 1.14.5 documentation. *Inertial unit — NAO Software 1.14.5 documentation*. URL: http://doc.aldebaran.com/1-14/family/robots/inertial_robot.html. (Accessed 2016-2-23).

[16] NAO Software 1.14.5 documentation. *Video Camera - NAO Software 1.14.5 documentation*. URL: http://doc.aldebaran.com/1-14/family/robots/video_robot.html#robot-video. (Accessed 2016-3-01).

[17] NAO Software 1.14.5 documentation. *NAOqi Vision - NAO Software 1.14.5 documentation*. URL: http://doc.aldebaran.com/1-14/naoqi/vision/index.html#naoqi-vision. (Accessed 2016-3-01).

[18] Cyberbotics. *Webots: reference manual - ode-open-dynamics-engine*. URL: https://www.cyberbotics.com/reference/jointparameters.php#joint_parameters_node. (Accessed 2016-1-28).

[19] Russell Smith. *Open Dynamics Engine v0.5 User Guide: Concepts: Physics model: Friction Approximation*. URL: http://www.ode.org/ode-latest-userguide.html#sec_3_11_1. (Accessed 2016-3-18).

[20] Louis N Hand and Janet D Finch. *Analytical mechanics*. Cambridge University Press, 1998, p. 19.

[21] Anton O. Belyakov, Alexander P. Seyranian, and Angelo Luongo. "Dynamics of the pendulum with periodically varying length". In: *Physica D: Nonlinear Phenomena* 238.16 (2009), pp. 1589–1597. DOI: 10.1016/j.physd.2009.04.015.

[22] Stephen Wirkus, Richard Rand, and Andy Ruina. "How to Pump a Swing". In: *The College Mathematics Journal* 29.4 (1998), p. 266. DOI: 10.2307/2687680.

[23] Eugene I Butikov. "Parametric resonance in a linear oscillator at square-wave modulation". In: *European Journal of Physics* 26.1 (2004), pp. 157–174. DOI: 10.1088/0143-0807/26/1/016.

[24] Andreas Daffertshofer Auke A. Post Gert de Groot and Peter J. Beek. *Pumping a Playground Swing*. URL: https://www.researchgate.net/profile/Peter_Beek2/publication/6262462_Pumping_a_Playground_Swing/links/0046352a181cb88d13000000.pdf. (Accessed 2016-1-28).

[25] Wolfram Demonstrations Project. *Motion of Pendulum With Varying Length*. URL: http://demonstrations.wolfram.com/MotionOfPendulumWithVaryingLength. (Accessed 2016-2-14).

[26] Stephen Lawrence. *How do you go up in a swing?* URL: http://www.physicsinsights.org/up_in_a_swing.html. (Accessed 2016-1-27).

[27] K.J. Astrom and K. Furuta. "Swinging up a pendulum by energy control". In: *Automatica* 36.2 (2000), pp. 287–295. DOI: 10.1016/s0005-1098(99)00140-5.

[28] Kinovea Ltd. *Kinovea: A microscope for your videos*. URL: http://www.kinovea.org. (Accessed 2016-2-10).

[29] Evans F.G. Plagenhoef S. and T. Abdelnour. *Body Segement Data*. URL: http://www.exrx.net/Kinesiology/Segments.html. (Accessed 2016-3-3).

[30] David Wynick. *What is the weight of a human leg?* URL: http://www.askabiologist.org.uk/answers/viewtopic.php?id=1477. (Accessed 2016-3-4).

[31] Conrad Quilty-Harper. *The world's fattest countries: how do you compare?* URL: http://www.telegraph.co.uk/news/earth/earthnews/9345086/The-worlds-fattest-countries-how-do-you-compare.html. (Accessed 2016-3-4).

[32] A.P. Seyranian. "The swing: Parametric resonance". In: *Journal of Applied Mathematics and Mechanics* 68.5 (2004), pp. 757–764. DOI: 10.1016/j.jappmathmech.2004.09.011. (Accessed 2016-1-27).

[33] Anonymous. *Is using a swing an example of normal or of parametric resonance?* URL: http://physics.stackexchange.com/questions/98448/is-using-a-swing-an-example-of-normal-or-of-parametric-resonance. (Accessed 2016-3-2).

[34] Doc.aldebaran.com. *H25 - Masses NAO Software 1.14.5 documentation*. URL: http://doc.aldebaran.com/2-1/family/robots/masses_robot.html. (Accessed 2016-1-28).

[35] Wantai. *Mini Stepper Product Specifications*. URL: http://www.phidgets.com/documentation/Phidgets/3308_0_Datasheet.pdf. (Accessed 2016-1-28).

[36] Phidgets Inc. *1067_0 - PhidgetStepper Bipolar HC*. URL: http://www.phidgets.com/products.php?product_id=1067. (Accessed 2016-1-28).

[37] Phidgets Inc. *1067 User Guide*. URL: http://www.phidgets.com/docs/1067_User_Guide. (Accessed 2016-1-28).

[38] B. Lee Roberts. *Notes on Linear and Nonlinear Oscillators, and Periodic Waves*. Jan. 28, 2016. URL: http://physics.bu.edu/py231/osc-nl-fourier.pdf. (Accessed 2016-1-28).

[39] Adam Stelmack. *Stepper-simple.py.* URL: http://www.phidgets.com/downloads/examples/Python_2.1.8.20160222.zip. (Accessed 2016-1-28).

[40] A. A. Post et al. "Pumping a Playground Swing". In: *Motor Control* 11 (2007), pp. 136–150.

[41] NAO Framework 1.14.5 documentation. *NAOqi Framework - NAO Software 1.14.5 documentation.* URL: http://doc.aldebaran.com/1-14/dev/naoqi/index.html. (Accessed 2016-2-22).

[42] Richard S. Sutton. "Learning to Predict by the Methods of Temporal Differences". In: (1988), pp. 11–12.

[43] Group Studies Robotics 2016 Team. *Year 3 UOB Physics Robotics Group Studies Code Repository.* URL: https://github.com/philj56/robot-swing. (Accessed 2016-2-15).

[44] Samuel J. Rowlinson. *Robotics Group Studies 2016 Machine Learning Library Documentation.* URL: http://robotics2016.azurewebsites.net. (Accessed 2016-3-15).

[45] hyperphysics.com. *Equation for under damped oscillator.* URL: http://hyperphysics.phy-astr.gsu.edu/hbase/oscda.html. (Accessed 2016-3-20).

[46] Sony. *Xperia Z2 Features.* URL: http://www.sonymobile.com/gb/products/phones/xperia-z2/features/#camera. (Accessed 2016-02-05).

Appendices

A Robot Properties and Swing Design

A.1 Documented values for robot limb angular movement

The following was contributed by: E. Humphreys

Table A.1 shows there documented values of the angular movements of the robots limbs. These values were used to calculate the percentage decrease between these and the measured values.

Table A.1: Range of Angular Movement from documentation.

Joint Name	Min angle (°)	Max angle (°)
Head(yaw)	-38.5	29.5
Shoulder(roll)	-18	76
Shoulder(pitch)	119.5	-119.5
Elbow(yaw)	119.5	-119.5
Elbow(roll)	2	88.5
Wrist(yaw)	-104.5	104.5
Hip(roll)	-45.29	21.74
Hip(pitch)	-88	27.73
Knee(pitch)	-5.29	121.04
Ankle(roll)	25.0	17.6
Ankle(pitch)	52.9	-67.9

A.2 Error on the force exerted during the strength tests

The following was contributed by: G. Hazar

The error for the force is given by the following equation,

$$\sigma F = \sqrt{\left(\frac{r\sigma\tau}{l}\right)^2 + \left(\frac{\tau\sigma r}{l}\right)^2 + \left(\frac{\tau r\sigma l}{l^2}\right)^2} \tag{A.1}$$

No error value is given in the NAO documentation for the speed reduction ratio therefore σr is set to zero in the calculations.

A.3 Approximating the swings damping coefficient

To determine the damping coefficient of the swing, the lower joints were locked such that the swing acted like a simple, rigid pendulum. The swing was then set in to motion from a displacement of $28.8 \pm 0.2°$ and left to move freely until the amplitude of the oscillations decrease zero. During the motion of the swing, readings were taken from the encoder every 0.2s. The data was then fitted to the equation for a damped oscillator using the `gnuplot` program, the results are shown in Figure A.1. The following equation was used [45],

$$\Phi(t) = A \exp^{-\gamma t} \cos(\omega t - \alpha)$$

where $\Phi(t)$ is the angular displacement, A is the initial amplitude, γ is damping coefficient, ω is the sinusoid frequency and α is a constant.

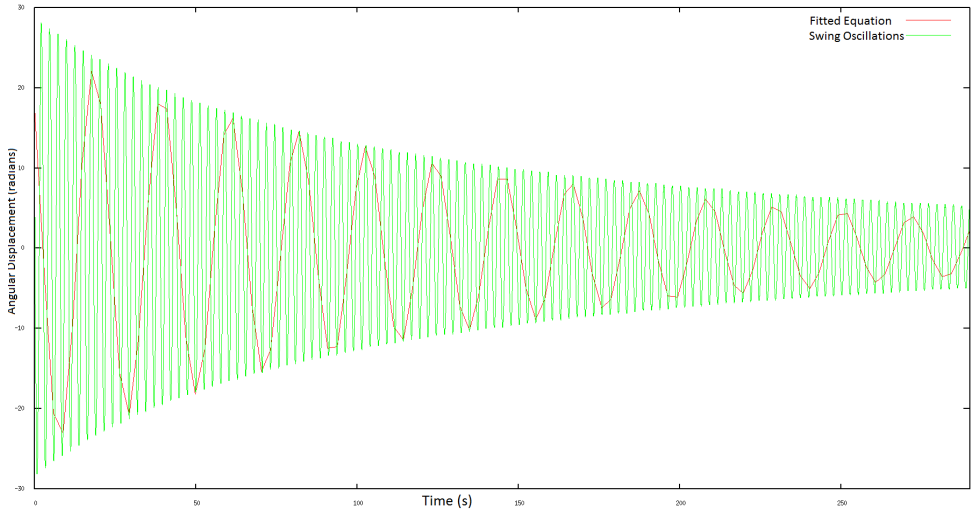


Figure A.1: Fit of the swing oscillations to the equation for a damped oscillator.

Although the fit does not follow the general oscillations of the swing well it does fit the decay of oscillations to an appropriate degree of accuracy for use in Webots swing simulations. The fitted values for the constants can be found in Table A.2. The errors were provided by `gnuplot`. The value for the damping coefficient of the top joint, obtained here, was assumed the same for all other joints,due to their identical construction.

Table A.2: The result of the damped oscillation equation fit for the oscillations of the swing.

Constant	Value
A	$25.578 \pm 0.009°$
γ	$0.00679 \pm 0.00025\ s^{-1}$
ω	$5.965 \pm 0.001\ s^{-1}$
α	-0.670 ± 0.003

B Theory Behind Swinging Motion

B.1 Derivation of the equation of motion for the dumbbell model

The following was contributed by: O. Diba

Let (x_1, y_1) be the position of mass 1, and (x_2, y_2) be the position of mass 2. Then the kinetic energy of the system is given by,

$$T = \frac{1}{2}m_1\left(\dot{x_1}^2 + \dot{y_1}^2\right) + \frac{1}{2}m_2\left(\dot{x_2}^2 + \dot{y_2}^2\right) \tag{B.1}$$

We now want to write the kinetic energy in terms of the generalised coordinates φ and α. We start by finding the position of the pivot P ,in terms of the generalised coordinates. By simple geometry,

$$x_p = l\sin\alpha \tag{B.2}$$
$$y_p = \cos\alpha \tag{B.3}$$

The position of mass 1 and mass 2 can be written relative to the position of the pivot P. From figure, it can be seen that mass 1 and mass 2, make an angle $\theta = \pi/2 - \alpha - \varphi$ to the horizontal. Now denoting the positions relative to P by x' and y', the coordinates are,

$$\begin{aligned} x_1' &= -a\cos\theta \quad & x_2' &= b\cos\theta \\ y_1' &= -a\sin\theta \quad & y_2' &= b\sin\theta \end{aligned} \tag{B.4}$$

where $\theta = \frac{\pi}{2} - \alpha - \varphi$. Using the double angle formula this can be simplified to,

$$\begin{aligned} x_1' &= -a\sin(\alpha+\varphi) \quad & x_2' &= b\sin(\alpha+\varphi) \\ y_1' &= -a\cos(\alpha+\varphi) \quad & y_2' &= b\cos(\alpha+\varphi) \end{aligned} \tag{B.5}$$

Thus, is the lab frame the masses have the coordinates,

$$\begin{aligned} x_1 &= l\sin\alpha - a\sin(\alpha+\varphi) \quad & x_2 &= l\sin\alpha + b\sin(\alpha+\varphi) \\ y_1 &= l\cos\alpha - a\cos(\alpha+\varphi) \quad & y_2 &= l\cos\alpha + b\cos(\alpha+\varphi) \end{aligned} \tag{B.6}$$

Taking the time derivative of the coordinates,

$$\begin{aligned} \dot{x_1} &= l\dot{\alpha}\cos\alpha - a\left(\dot{\alpha}+\dot{\varphi}\right)\cos(\alpha+\varphi) \quad & \dot{x_2} &= l\dot{\alpha}\cos\alpha + b\left(\dot{\alpha}+\dot{\varphi}\right)\cos(\alpha+\varphi) \\ \dot{y_1} &= -l\dot{\alpha}\sin\alpha + a\left(\dot{\alpha}+\dot{\varphi}\right)\sin(\alpha+\varphi) \quad & \dot{y_2} &= -l\dot{\alpha}\sin\alpha - b\left(\dot{\alpha}+\dot{\varphi}\right)\sin(\alpha+\varphi) \end{aligned} \tag{B.7}$$

Now we square the coordinates and add, as they appear in the kinetic energy B.1, using the

Pythagorean identity to simplify terms,

$$\dot{x}_1^2 + \dot{y}_1^2 = \dot{\alpha}^2 l^2 + a^2 \left(\dot{\alpha} + \dot{\varphi}\right)^2 - 2\dot{\alpha} l a \left(\dot{\alpha} + \dot{\varphi}\right) \cos\varphi \tag{B.8}$$

$$\dot{x}_2^2 + \dot{y}_2^2 = \dot{\alpha}^2 l^2 + b^2 \left(\dot{\alpha} + \dot{\varphi}\right)^2 + 2\dot{\alpha} l b \left(\dot{\alpha} + \dot{\varphi}\right) \cos\varphi \tag{B.9}$$

So the kinetic energy, in terms of the generalised coordinates is given by,

$$T = \frac{m_1}{2}\left[\dot{\alpha}^2 l^2 + a^2 \left(\dot{\alpha} + \dot{\varphi}\right)^2 - 2\dot{\alpha} l a \left(\dot{\alpha} + \dot{\varphi}\right) \cos\varphi\right] + \frac{m_2}{2}\left[\dot{\alpha}^2 l^2 + b^2 \left(\dot{\alpha} + \dot{\varphi}\right)^2 + 2\dot{\alpha} l b \left(\dot{\alpha} + \dot{\varphi}\right) \cos\varphi\right] \tag{B.10}$$

Now we must find the potential term of the Lagrangian. First we define the zero potential of the system to be when the pendulum arm is hanging parallel to the vertical, with the dumbbell perpendicular to the arm. Then the potential is,

$$V = m_1 g \Delta y_1 + m_2 g \Delta y_2 \tag{B.11}$$

where Δy refers to the change in height of a mass above the zero position. As was done with the kinetic term, we first find the change in height of the pivot point from the zero position. This is just,

$$\Delta y_p = l\left(1 - \cos\alpha\right) \tag{B.12}$$

The change in height of the masses relative to the pivot, has already been found in equation B.6, but we must be careful with the minus signs, since the mass gains potential energy as its y-coordinate becomes more negative.

$$\begin{aligned} \Delta y_1' &= a\cos(\alpha + \varphi) \\ \Delta y_2' &= -b\cos(\alpha + \varphi) \end{aligned} \tag{B.13}$$

So the potential is,

$$\begin{aligned} V &= m_1 g\left(l\left(1 - \cos\alpha\right) + a\cos(\alpha + \varphi)\right) + m_2 g\left(l\left(1 - \cos\alpha\right) - b\cos(\alpha + \varphi)\right) \\ &= gl(m_1 + m_2)(1 - \cos\alpha) + g(m_1 a - m_2 b)\cos(\alpha + \varphi) \end{aligned} \tag{B.14}$$

Finally, the Lagrangian is,

$$\begin{aligned} \mathcal{L} =& \frac{m_1}{2}\left[\dot{\alpha}^2 l^2 + a^2 \left(\dot{\alpha} + \dot{\varphi}\right)^2 - 2\dot{\alpha} l a \left(\dot{\alpha} + \dot{\varphi}\right) \cos\varphi\right] \\ &+ \frac{m_2}{2}\left[\dot{\alpha}^2 l^2 + b^2 \left(\dot{\alpha} + \dot{\varphi}\right)^2 + 2\dot{\alpha} l b \left(\dot{\alpha} + \dot{\varphi}\right) \cos\varphi\right] \\ &+ gl(m_1 + m_2)(\cos\alpha - 1) + g(m_2 b - m_1 a)\cos(\alpha + \varphi) \end{aligned} \tag{B.15}$$

Applying the Euler-Lagrange equation for α,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \dot{\alpha}} =& (m_1 + m_2) l^2 \dot{\alpha} + \left(m_1 a^2 + m_2 b^2\right)\left(\dot{\alpha} + \dot{\varphi}\right) + l(m_2 b - m_1 a)\left(2\dot{\alpha} + \dot{\varphi}\right)\cos\varphi \\ \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial \mathcal{L}}{\partial \dot{\alpha}} =& (m_1 + m_2) l^2 \ddot{\alpha} + \left(m_1 a^2 + m_2 b^2\right)\left(\ddot{\alpha} + \ddot{\varphi}\right) + l(m_2 b - m_1 a)\left(2\ddot{\alpha} + \ddot{\varphi}\right)\cos\varphi \\ \frac{\partial \mathcal{L}}{\partial \alpha} =& -gl(m_1 + m_2)\sin\alpha - g(m_2 b - m_1 a)\sin(\alpha + \varphi) \end{aligned} \tag{B.16}$$

This gives the following equation of motion,

$$\begin{aligned} 0 =&(m_1+m_2)l^2\ddot{\alpha} + (m_1a^2+m_2b^2)\left(\ddot{\alpha}+\ddot{\varphi}\right) + l(m_2b-m_1a)\left(2\ddot{\alpha}+\ddot{\varphi}\right)\cos\varphi \\ &+ gl(m_1+m_2)\sin\alpha + g(m_2b-m_1a)\sin(\alpha+\varphi) \end{aligned} \tag{B.17}$$

B.2 Derivation of the equation of motion for flail model

The following was contributed by: B. Stokes

The kinetic and potential energies of the flail system are,

$$\begin{aligned} T =&\frac{1}{2}m_1l_1^2\dot{\theta_1}^2 \\ &+\frac{1}{2}m_2(l_1^2\dot{\theta_1}^2 + l_2^2\dot{\theta_2}^2 + 2l_1l_2\dot{\theta_1}\dot{\theta_2}\cos(\theta_1-\theta_2)) \\ &+\frac{1}{2}m_3(l_1^2\dot{\theta_1}^2 + l_3^2\dot{\theta_3}^2 + 2l_1l_3\dot{\theta_1}\dot{\theta_3}\cos(\theta_1-\theta_3)). \end{aligned} \tag{B.18}$$

$$\begin{aligned} V =& - m_1gl_1\cos\theta_1 \\ &- m_2g(l_1\cos(\theta_1)+l_2\cos\theta_2) \\ &- m_3g(l_1\cos(\theta_1)+l_3\cos\theta_3). \end{aligned}$$

This gives a Lagrangian of,

$$\begin{aligned} \mathcal{L} = \frac{M}{2}l_1^2\dot{\theta_1}^2 + \frac{1}{2}m_2(l_2^2\dot{\theta_2}^2 + 2l_1l_2\dot{\theta_1}\dot{\theta_2}\cos(\theta_1-\theta_2)) + \frac{1}{2}m_3(l_3^2\dot{\theta_3}^2 + 2l_1l_3\dot{\theta_1}\dot{\theta_3}\cos(\theta_1-\theta_3)) \\ +Mgl_1\cos\theta_1 + m_2gl_2\cos\theta_2 + m_3gl_3\cos\theta_3; \end{aligned}$$

the Euler-Lagrange (Equation 3.4) was then applied. This yields,

$$\begin{aligned} \tfrac{\partial\mathcal{L}}{\partial\dot{\theta_1}} =& Ml_1^2\dot{\theta_1} + m_2l_1l_2\dot{\theta_2}\cos(\theta_1-\theta_2) + m_3l_1l_3\dot{\theta_3}\cos(\theta_1-\theta_3), \\ \implies \tfrac{\mathrm{d}}{\mathrm{d}t}(\tfrac{\partial\mathcal{L}}{\partial\dot{\theta_1}}) =& Ml_1^2\ddot{\theta_1} + m_2l_1l_2\ddot{\theta_2}\cos(\theta_1-\theta_2) - m_2l_1l_2\dot{\theta_2}\sin(\theta_1-\theta_2)(\dot{\theta_1}-\dot{\theta_2}) \\ &+ m_3l_1l_3\ddot{\theta_3}\cos(\theta_1-\theta_3) - m_3l_1l_3\dot{\theta_3}\sin(\theta_1-\theta_3)(\dot{\theta_1}-\dot{\theta_3}); \\ \tfrac{\partial\mathcal{L}}{\partial\dot{\theta_1}} =& - Mgl_1\sin(\theta_1) - m_2l_1l_2\dot{\theta_1}\dot{\theta_2}\sin(\theta_1-\theta_2) - m_3l_1l_3\dot{\theta_1}\dot{\theta_3}\sin(\theta_1-\theta_3); \\ \therefore Ml_1^2\ddot{\theta_1} &+ m_2l_1l_2\ddot{\theta_2}\cos(\theta_1-\theta_2) - m_2l_1l_2\dot{\theta_2}\sin(\theta_1-\theta_2)(\dot{\theta_1}-\dot{\theta_2}) \\ &+m_3l_1l_3\ddot{\theta_3}\cos(\theta_1-\theta_3) - m_3l_1l_3\dot{\theta_3}\sin(\theta_1-\theta_3)(\dot{\theta_1}-\dot{\theta_3}) \\ &= - Mgl_1\sin(\theta_1) - m_2l_1l_2\dot{\theta_1}\dot{\theta_2}\sin(\theta_1-\theta_2)m_3l_1l_3\dot{\theta_1}\dot{\theta_3}\sin(\theta_1-\theta_3). \end{aligned} \tag{B.19}$$

This last equation can be rearranged to give Equation 3.27.

B.3 Derivation of the equation of motion for the hinged flail model

The following was contributed by: C. Hogg

The Lagrangian of the system is made up of the three contributions,

$$\mathcal{L} = \mathcal{L}_{Double_{Seat}} + \mathcal{L}_{Triple_{Torso}} + \mathcal{L}_{Triple_{Legs}} \tag{B.20}$$

$$\begin{aligned}
\implies \mathcal{L} = \; & \frac{\frac{1}{2}l_1 r\dot{\theta}\dot{\alpha}m_1\sin(\theta-\alpha) - g\sin\theta m_1 l_1 + \frac{1}{2}l_1 r\ddot{\alpha}m_1\cos(\theta-\alpha) + \dot{\theta}\dot{\alpha}m_1\sin(\theta-\alpha)}{r^2 m_1} \\
& + \frac{\left(\begin{array}{r}\frac{1}{2}l_1 r\dot{\theta}\dot{\alpha}m_2\sin(\theta-\alpha) - \frac{1}{2}m_2 l_2 r\dot{\theta}\dot{\phi}\sin(\theta-\phi) - m_2 l_2 g\sin\theta + \frac{1}{2}l_1 r\ddot{\alpha}m_2\cos(\theta-\alpha) + \\ \frac{1}{2}l_1 r\dot{\alpha}^2 - \dot{\theta}\dot{\alpha}m_2\sin(\theta-\alpha) + \frac{1}{2}m_2 l_2 r(\ddot{\phi}\cos(\theta-\phi) + \dot{\phi}^2 - \dot{\theta}\dot{\phi}\sin(\theta-\phi))\end{array}\right)}{r^2 m_2} \\
& + \frac{\left(\begin{array}{r}\frac{1}{2}l_1 r\dot{\theta}\dot{\alpha}m_3\sin(\theta-\alpha) - \frac{1}{2}m_3 l_3 r\dot{\theta}\dot{\psi}\sin(\theta-\psi) - m_3 l_3 g\sin\theta + \frac{1}{2}l_1 r\ddot{\alpha}m_3\cos(\theta-\alpha) + \\ \frac{1}{2}l_1 r\dot{\alpha}^2 - \dot{\theta}\dot{\alpha}m_3\sin(\theta-\alpha) + \frac{1}{2}m_3 l_3 r(\ddot{\psi}\cos(\theta-\psi) + \dot{\psi}^2 - \dot{\theta}\dot{\psi}\sin(\theta-\psi))\end{array}\right)}{r^2 m_3}
\end{aligned} \tag{B.21}$$

This rearranges into Equation 3.29. The Euler-Lagrange (Equation 3.4) for θ was applied, producing,

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\dot{\theta}} =& m_1 r^2\dot{\theta} + m_1 l_1(\dot{\alpha}\cos(\theta-\alpha) + \dot{\theta}r^2 m_2 + r\dot{\alpha}l_1 m_2\cos(\theta-\alpha) + rl_2 m_2\dot{\phi}\cos(\theta-\phi) \\
& + \dot{\theta}r^2 m_3 + \mathring{\alpha}l_1 m_3\cos(\theta-\alpha) + rl_3 m_3\dot{\psi}\cos(\theta-\psi)
\end{aligned}$$

$$\begin{aligned}
\implies \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial\mathcal{L}}{\partial\dot{\theta}} =& Mr^2\ddot{\theta} + m_1 l_1(\ddot{\alpha}\cos(\theta-\alpha) - \dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha})) + rl_1 m_2(\ddot{\alpha}\cos(\theta-\alpha) - \dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha})) \\
& + rl_1 m_3(\ddot{\alpha}\cos(\theta-\alpha) - \dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha})) + rl_2 m_2(\ddot{\phi}\cos(\theta-\phi) - \dot{\phi}\sin(\theta-\phi)(\dot{\theta}-\dot{\phi})) \\
& + rl_3 m_3(\ddot{\psi}\cos(\theta-\psi) - \dot{\psi}\sin(\theta-\psi)(\dot{\theta}-\dot{\psi}))
\end{aligned}$$

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\theta} =& -(m_1 rl_1\dot{\theta}\dot{\alpha}\sin(\theta-\alpha) + \dot{\theta}r\dot{\alpha}l_1 m_2\sin(\theta-\alpha) + \dot{\theta}r\dot{\phi}l_2 m_2\sin(\theta-\phi) \\
& + \dot{\theta}r\dot{\alpha}l_1 m_3\sin(\theta-\alpha) + \dot{\theta}r\dot{\psi}l_3 m_3\sin(\theta-\psi) + Mgr\sin(\theta))
\end{aligned} \tag{B.22}$$

$$\begin{aligned}
\therefore & Mr^2\ddot{\theta} + m_1 l_1(\ddot{\alpha}\cos(\theta-\alpha) - \dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha})) + rl_1 m_2(\ddot{\alpha}\cos(\theta-\alpha) - \dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha})) \\
& + rl_1 m_3(\ddot{\alpha}\cos(\theta-\alpha) - \dot{\alpha}\sin(\theta-\alpha)(\dot{\theta}-\dot{\alpha})) + rl_2 m_2(\ddot{\phi}\cos(\theta-\phi) - \dot{\phi}\sin(\theta-\phi)(\dot{\theta}-\dot{\phi})) \\
& + rl_3 m_3(\ddot{\psi}\cos(\theta-\psi) - \dot{\psi}\sin(\theta-\psi)(\dot{\theta}-\dot{\psi})) = -(m_1 rl_1\dot{\theta}\dot{\alpha}\sin(\theta-\alpha) + \dot{\theta}r\dot{\alpha}l_1 m_2\sin(\theta-\alpha) \\
& + \dot{\theta}r\dot{\phi}l_2 m_2\sin(\theta-\phi) + \dot{\theta}r\dot{\alpha}l_1 m_3\sin(\theta-\alpha) + \dot{\theta}r\dot{\psi}l_3 m_3\sin(\theta-\psi) + Mgr\sin(\theta))
\end{aligned} \tag{B.23}$$

Rearranging this gives the equation of motion of the swing, Equation 3.30.

C Swinging from Predetermined Functions

C.1 Sony Xperia Z2 Phone specifications

The following was contributed by: Z. Hodgins

As mentioned in Section 3.2.2, the Sony Xperia Z2 Phone was used for the filming the swinging motions. This was used because it was the best camera that was available. It had 4K video capture with Full HD 1080p image quality, with a large 1/2.3" 20.7 MP Exmor R mobile image sensor. It supported 4K (3840 x 2160) resolution and playback on a 4K TV or projector using the latest MHL 3.0 connector [46].

D Cross-Compilation and C++

The following was contributed by: P. Jones

There were a number of difficulties encountered regarding the use of C++ on the robot. Firstly, while the NAOqi documentation claimed that the C++ framework is cross platform, the cross-compilation toolchain that allowed programs to be run locally on the robot was only available for Linux. In addition, great difficulty was met when trying to install the framework on both Windows and OSX. In comparison, the Linux installation process was much simpler and quicker. It is therefore recommended that future projects use Linux from the start if deciding to program in C++, as this will save time, especially if cross-compilation is required later in the project.

It should be noted that while version 1.14 of the SDK only officially supports Ubuntu 10.04 - 12.04, Ubuntu 14.04 was used throughout the project without issue.

Another difficulty that arose during the course of the project was in transferring cross-compiled programs and libraries to the robot. During the project, the robot was unable to connect to either the *UOBwifi* or *eduroam* networks available throughout the university, as these required a user name and password to access the network, rather than just a password. Instead, a separate router had to be used, which was not connected to the Internet. Paired with the lack of a laptop with the cross-compilation toolchain installed, this caused a significant slowdown at various points throughout the project. In order to build a program and run it on the robot, a laptop had to access a remote server that could build the program, copy the program to the laptop, switch WiFi networks to be able to communicate with the robot, copy the program onto the robot, and finally access the robot and run the program. While not such a problem when making large changes to a program, this quickly became a source of much annoyance when changing a single line or so of code. For this reason, it is strongly recommended that if C++ is used, any future projects either:

- find a way of connecting the robot to the Internet, so that the laptop "middleman" can be skipped, or
- set up a local computer or laptop with Linux, the SDK, and the cross-compilation toolchain, so that a remote computer is not needed.

Both of these would have the effect of reducing the number of steps required to transfer a program to the robot, however the first is more desirable if at all possible. This is because it would allow the set up of single remote build server, which many people could access at once, including with the lab computers which do not have WiFi. Otherwise, each group would have to set up the SDK and toolchains on individual laptops, which they may not own.

Another advantage to using a remote server, as was used in this project, was that everyone had access to the same build environment. This meant that changes did not need to be made on multiple computers. For example, in order to build the encoder library, 32bit versions of some libraries had to be installed. The use of a shared remote server meant that this only had to be done once, then everyone could use them.

E USB

———————————The following was contributed by: Z. Hodgins ———————————

All of the data, code and video files that were used within this report can be found in the memory stick that was submitted alongside this report.

Index of Authors